Unstructured Data Analysis: Final Coursework
Grayscale Image Denoising Methods: a Comparison

Student	Fortunato Nucera
CID	02165798
Link to GitHub Repository	`https://github.com/tinosai/UDA_CW4_NUCERA`
Statement	**I declare that I have worked on this assessment independently.**

1 Imaging and Noise Fundamentals

Photographers all over the world, during their first experiences, learn to deal with exposure, blur, and light reflection, so that they can make the most out of their surroundings given the available hardware - be it a Digital Single-Lens Reflex Camera (DSLR), a mirrorless camera, or even the camera of a smartphone.

Most open-source software readily available on such devices allows for contrast and exposure adjustments, making it possible for the inexperienced user to take stunning pictures with minimum or no manual setup. Although this, in theory, is sufficient for most, problems arise when taking pictures in a dark environment, such as an indoor setting in the absence of a flash device or an outdoor setting at night.

Cameras' exposure can be controlled via 3 distinct parameters: shutter speed, aperture, and ISO.

- **Shutter Speed** is related to the amount of time the camera sensor is exposed to light. The shutter speed is inversely proportional to the exposure: the higher the speed, the lower the amount of light the sensor is exposed to and the darker the output image.
- **Aperture** is the diameter of the opening through which light passes when reaching the sensor. In digital imaging, the aperture is usually expressed in terms of the f-stop, where $f\text{-stop} = \phi/D$ in which ϕ represents the distance (in millimeters) between the center of the lens and the focal plane (the camera sensor) and D is the diameter of the opening. Higher values of the f-stop indicate a smaller aperture, which translates into a smaller amount of light reaching the sensor, but also in a higher depth of field in the output image.
- The **ISO** indicates the sensitivity of a camera to light. Lower values of ISO correspond to low sensitivity, whereas larger values of ISO correspond to high sensitivity. In most recent DSLRs (and even more recent mirrorless cameras) this value ranges from as low as 100 to as high as 4 million.

The three parameters above all contribute to determining the final image exposure, and the result can be made more or less dramatic depending on the used parameters (for example, it is customary to use a very low f-stop number in portraits, in order for the viewer to focus on the subject rather than on the background, which is then blurred). This *balance* between the parameters can be represented via an *exposure triangle* (included in Figure 1). There are cases where the use of tripods is possible; therefore, the increased exposure can be achieved by a lower

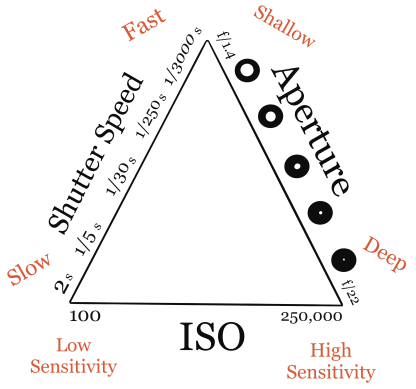

Figure 1: The exposure triangle: the construction of a similar triangle starting from the base one allows to infer the needed adjustments to keep the exposure constant, i.e. keeping the aperture and exposure constant, a higher shutter speed (needed to avoid hand motion blur when capturing the image) must be balanced by a higher ISO.

shutter speed without compromising the results. In other cases, even if a tripod is available, the shutter speed should be kept high in order to avoid light trails (astrophotography is an example, where the Earth's rotation would cause

the light emitted by the stars to move across the sky, leaving a trail behind and compromising the final results). In these circumstances, the use of high ISO is necessary. As previously explained, ISO increases the sensitivity of the sensor to light, at the cost of image quality: the images become much noisier, as shown in Figure 2.

Figure 2: A comparison of the same image taken at two different ISO values: 100 on the left and 65000 on the right. Note how some spots appear on the lantern pole on the right picture, along with blurred edges. Image by the author.

2 The Denoising Problem

The denoising problem is therefore formulated in the following way: let $\tilde{F} \in \mathbb{R}^{m \times n}$ be a noisy picture, i.e. the realization of a noiseless ground truth picture $F \in \mathbb{R}^{m \times n}$. We wish to find a denoised image $\hat{F}$ that approximates the ground truth F well. It is important to state that, in general, the noise distribution in $\tilde{F}$ is not known to us, therefore we cannot make any distributional assumptions regarding it, as explained in [1]. In particular, even though the Gaussian Noise hypothesis is often taken for granted, this assumption may result in the underestimation of the noise intensity if the noise is, for example, t-distributed. We will refer to the denoising model as $\mathcal{M}$, such that:

$$F \approx \hat{F} = \mathcal{M}(\tilde{F})$$

The topic of the current project is the analysis of the performances of different models $\mathcal{M}$ which are most suitable for the chosen data set.

3 The Data Set

The assignment sheet requested a data set weighting less than 100MB. We could not find readily available data sets for denoising fulfilling this constraint. As a result, we modified an available data set so as to fit the given size. We obtained the data set from [2]. The author of this report has asked and obtained the written permission from the data set creator to use and share the preprocessed data set for academic purposes.

The data set contains about 500 pictures of 120 different scenes. The pictures are captured in low-light conditions with 3 different devices: a smartphone (Xiaomi Mi3), a compact camera (Canon S90) and a DSLR (Canon 600D). Each of the three devices features a different sensor size and a different sensitivity to light at high ISO.

The number of pictures and the size of each (the largest images would measure 3461×5198 pixels) would make processing quite challenging. On the one hand, training a neural network from scratch on 120 images would most likely result in overfitting. On the other, processing large images before the input into a neural network would require the application of an interpolation, therefore the denoising model would be trained on the interpolated image rather than the original one. The interpolation itself would result in the mixture of unknown noise distributions, making the entire reasoning shaky at best.

Given the points detailed above, we proceeded with data pre-processing in the following way:

- we downloaded the original data set (approx. 10GB).
- from each of the batch subfolders, we selected *one* input (noisy) image and the corresponding target (noise-free) image.
- we then converted the images from RGB to grayscale color space using `opencv`.

- we split each of the images into 128×128 pixel non-overlapping patches.
- we kept only the 4050 patches featuring the highest intensity standard deviation: this allowed to exclude monochromatic images whose denoising would be not as interesting.
- we flattened each pair of images (input / target) into a long column vector v structured as

$$v^t = [\text{camera index}, \tilde{p}_1, \cdots, \tilde{p}_{128^2}, p_1, \cdots, p_{128^2}]$$

  That is, a vector of length $1 + 128^2 + 128^2 = 32769$, where $\tilde{p}_1, \cdots, \tilde{p}_{128^2}$ indicate the pixels belonging to the noisy image and $p_1, \cdots, p_{128^2}$ indicate the pixels belonging to the target image.
- these long vectors are stacked horizontally and saved as a `parquet` data frame, which is easily loadable via the Python library `Pandas`. This resulted in a data set of size 99MB.

Given the granularity of the pre-processing involved, we could not find any results concerning the described approach in the literature. A sample of the pre-processed data set can be found in Figure 3.

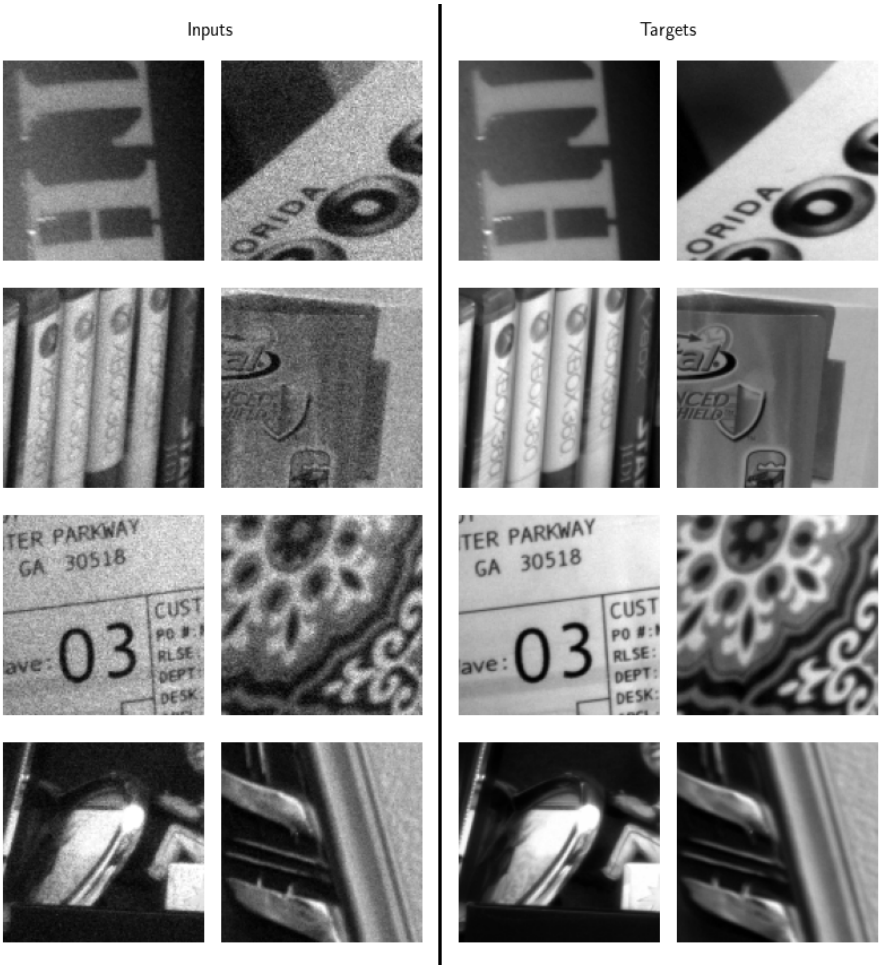

Figure 3: A sample of the data set on which we will train the models. The image group on the left contains the noisy input image. The group on the right contains the corresponding targets.

4 Investigated Techniques

In this report, we want to provide a brief yet comprehensive view of the approaches for image denoising. We will investigate 4 families of denoising techniques:

- **Spatial Noise filters**: Mean/Median/Maximum/Minimum value filters.
- **Non-Local Means Denoising**.
- **Dimensionality Reduction methods**: principal component analysis (PCA) and non-negative matrix factorization (NMF).

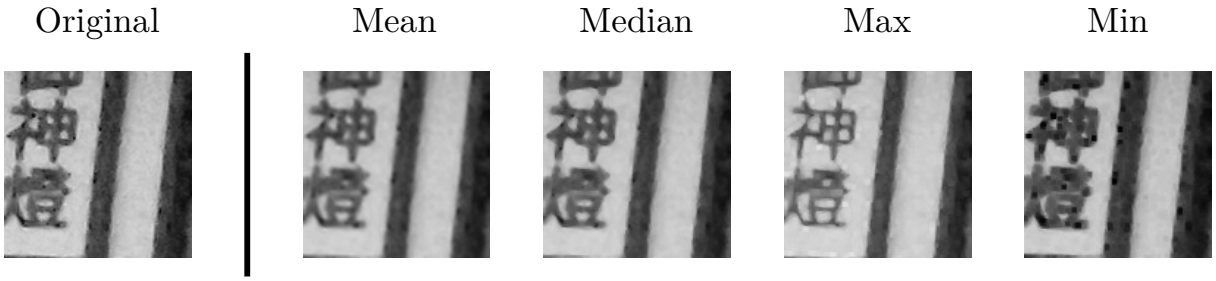

Figure 4: A comparison between Mean/Median/Maximum/Minimum value filters for a window size of 3. The image is a 128×128 patch from Figure 2.

- **Deep Learning methods**: U-net.

In the following sections, we will explain the rationale behind each of the mentioned denoising techniques in a concise format.

4.1 Spatial Noise Filters:

Given a mask W of size $M \times N$ and an image $\tilde{F}(x, y)$ of size $m \times n$, a spatial noise filter assigns a function over the mask W centered at (x, y) to the pixel (x, y) in the denoised image $\hat{F}(x, y)$. Depending on the used function, we may have different filters:

- *Mean Value Filter*: $\hat{F}(x, y) = \frac{1}{MN} \sum_{a=1}^{m} \sum_{b=1}^{n} \mathbb{1}_W(a, b) f(a, b)$, where $\mathbb{1}_W$ is the indicator function on W.
- *Median Value Filter*: $\hat{F}(x, y) = \text{median}(\mathbb{1}_W \tilde{F})$
- *Maximum Value Filter*: $\hat{F}(x, y) = \max(\mathbb{1}_W \tilde{F})$
- *Minimum Value Filter*: $\hat{F}(x, y) = \min(\mathbb{1}_W \tilde{F})$

The problem is then reduced to establishing the optimal value of the mask sizes, M and N, on a given image via a predefined loss. Since we have a training set, we may consider using that to establish which filter size brings, in general, good results on the data set, via the definition of a loss function. We will here impose two constraints: W must be a square window ($M = N$) and the search space for the window sizes will be limited to the set of odd integers between 1 and 11 ($S = \{1, 3, 5, 7, 9, 11\}$). For these filters, we used the optimized implementation in the `scikit-image` package. An application of such filters can be found in Figure 4. Note how Maximum and Minimum filter tend to, respectively, erode and dilate edges.

4.2 Non-Local Means Denoising (NLM)

The traditional mean value filtering algorithm discussed in Section 4.1 generates a square patch W of specified dimension M and assigns the mean over W to the center pixel (x, y) in the denoised image $\hat{F}(x, y)$. Such approach has an evident limitation regarding the locality of the information: even if the image were to exhibit periodic patterns outside the window size, simple spatial noise filters would be unable to exploit this information. The ability to include distant patterns in the denoising algorithm is the fundamental idea behind NLM, introduced in [3]. We provide a summary of the algorithm, adapted for grayscale images.

Given a noisy image $\tilde{F}$, we indicate the pixel with a single index p. This is always allowed, as the subscripts for a matrix (x, y) can always be converted into linear indices p.

For each pixel p in $\tilde{F}$:

1. Obtain $B(p, r)$, a search window centered at p of size $(2r + 1) \times (2r + 1)$
2. Calculate the average squared Euclidean distance between a patch $B(p, f)$ and a patch $B(q, f)$ where $(2f + 1) \times (2f + 1)$ is the size of the image patch, $q \in B(p, r)$.

$$d^2(B(p, f), B(q, f)) = \frac{1}{(2f + 1)^2} \sum_{i=1}^{(2f+1)^2} (B(p, f)_i - B(q, f)_i)^2$$

where $B(p, f)_i$ indicates the linear index i in the $B(p, f)$ patch. In order to simplify the notation, we will just refer to $d^2(B(p, f), B(q, f))$ as d^2_{pq}.

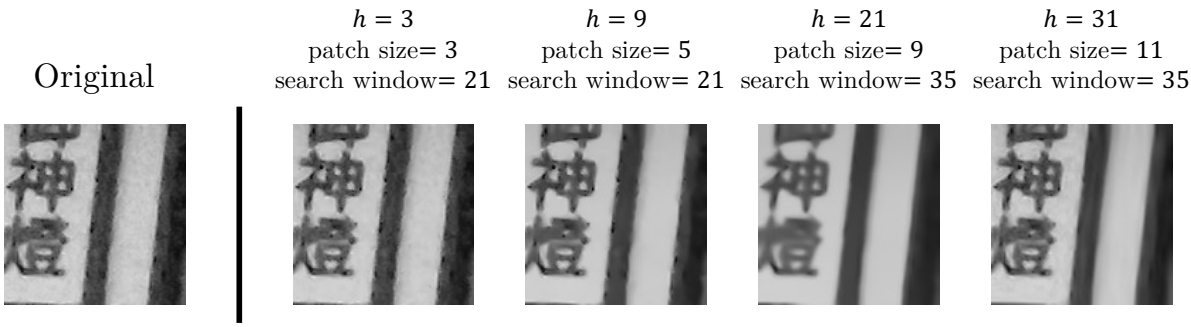

Figure 5: A comparison of the application of Non-Local Means for different hyper-parameters. The numbers above the images indicate, in order, the scaling constant h, the patch size, and the search window size. The image is a 128×128 patch from Figure 2.

3. Use a Gaussian kernel to obtain weights from the distances d_{pq}^2:

$$w(p,q) = \exp\left\{-\frac{\max(d_{pq}^2 - 2\sigma^2, 0.0)}{h^2}\right\}$$

where σ is the magnitude of the noise in the image and h is a filtering parameter (these two parameters are not independent). In particular, the original paper suggests that $w(p,p)$ should be not be set to 1, but to $w(p,p) = \max_q w(p,q)$ in order to avoid giving too much importance to the patch center.

4. Finally, we assemble the previous 3 steps to calculate the pixel value in the denoised image:

$$\hat{F}(p) = \frac{\sum_{q \in B(p,r)} \tilde{F}(q) w(p,q)}{\sum_{q \in B(p,r)} w(p,q)}$$

This algorithm is, in general, quite expensive, as the research window can be rather large, but its results are pleasant to the human eye. We use the highly optimized `C++` implementation offered in `openCV` (`fastNlMeansDenoising`) where the tunable quantities are the noise scale parameter h, the search window size $(2r+1)$ and the patch size $(2f+1)$. An example of the application of NLM for different parameters setup can be found in Figure 5.

4.3 Dimensionality reduction methods

Both the treated dimensionality reduction methods focus on decomposing a single matrix into the product of several matrices. The usual way this is done is by flattening the images in a data set and decomposing the *complete* data set matrix. This is of course not possible in our case for two reasons: 1) we have no guarantee that, during the deployment, we will exclusively deal with multiple-image inputs and 2) at runtime all the information we can use is the one included in the input image, and no other sources of information are available. As a result, the input image, treated as a matrix with rows and columns, will be decomposed via the methods explained below.

4.3.1 Principal Component Analysis (PCA)

The purpose of PCA is finding a linear subspace which maximizes the preserved variance or minimizes the reconstruction error. For the purpose of this report, we implemented PCA from scratch via Singular Value Decomposition (SVD). Let $\tilde{F} \in \mathbb{R}^{m \times n}$, the matrix $\tilde{F}$ can be decomposed into the product of 3 matrices:

$$\tilde{F} = UDV^T$$

where $U \in \mathbb{R}^{m \times n}$, $D \in \mathbb{R}^{n \times n}$ and $V \in \mathbb{R}^{n \times n}$ (if $m \geq n$, but there exist multiple conventions for the sizes). The matrix U contains the left singular vectors, the matrix D contains the singular values (in descending order) on the diagonal and the matrix V contains the right singular vectors. U and V are orthogonal matrices $U^{-1} = U^T, V^{-1} = V^T$. From this it is easy to see that: $\tilde{F}\tilde{F}^T = UDV^TVD^TU^T = UD^2U^T$, that is, U contains the eigenvectors of the Gram matrix. On the other hand, $\tilde{F}^T\tilde{F} = VD^TU^TUDV^T = VD^2V^T$, where $\tilde{F}^T\tilde{F}$ is called *scatter matrix*, equal to the covariance matrix up to a multiplicative constant $(n-1)^{-1}$, as long as the columns of $\tilde{F}$ are centered.

PCA could be a suitable candidate for noise removal as it automatically detects the *modes* in the image, where each mode is associated with different levels of "spatial vibration". In general, high-order modes, represented by small singular values on the diagonal of D are associated with the noise in the image. Removing them could therefore lead to a denoised image. The general algorithm for denoising via PCA then becomes, given an input image $\tilde{F}$:

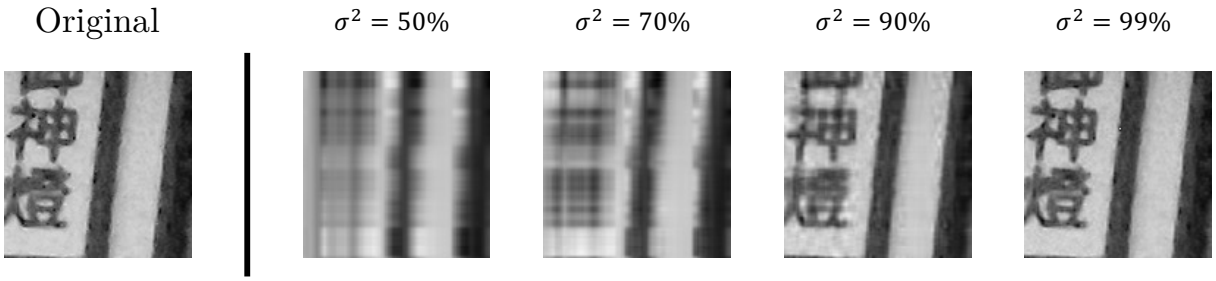

Figure 6: Image quality and PCA retained variance. Higher values of retained variance -included at the top of each image- are necessary not to discard too much information. The image is a 128×128 patch from Figure 2.

1. center and scale the columns of $\tilde{F}$, resulting in $\tilde{Z}$.
2. calculate the SVD on $\tilde{Z}$: $\tilde{Z} = UDV^T$.
3. depending on the desired rank r for denoising, pick only the first r columns of U, the top-left square matrix of D of size $r \times r$, and the first r rows of V^T (or the first r columns of V), these result in three submatrices U_r, D_r and V_r^T.
4. multiply the matrices $U_r D_r V_r^T$, rescale the columns and add the column means subtracted in 1. This results in $\hat{F}$

The singular values s_i on the diagonal of D control how much variance the reconstruction retains. Discarding variance (and noise) can be controlled by setting the rank r of the submatrices in step 3. In particular, the total variance can be calculated as $\mathrm{Var}(\tilde{F}) \propto \sum_{i=1}^{n} s_i^2$ and therefore the reconstruction for rank r of $\tilde{F}$ would have a residual variance as in $\mathrm{Var}(\hat{F}) = \mathrm{Var}(\tilde{F}_r) \propto \sum_{i=1}^{r} s_i^2$. The proportion of retained variance is then calculated as $\mathrm{Var}(\hat{F})/\mathrm{Var}(\tilde{F})$. In our experiment, for each image, we will calculate for what percentage of retained variance the loss is minimized, generate a distribution over the entire training set, and then define the threshold of retained variance to apply over the whole set, for comparison with other methods. The loss over the total dataset for the fixed threshold will allow to draw conclusions on the efficacy of PCA for noise removal. An example of how the image quality changes depending on the retained variance can be found in Figure 6.

4.3.2 Non-Negative Matrix Factorization (NMF)

NMF [4], is a dimensionality reduction technique to be applied exclusively to non-negative matrices. As $\tilde{F}$ is an image matrix in `uint8` format, we can rescale this to be a floating point matrix in the $[0, 1]$ interval which still satisfies the requirements of NMF. The basic idea behind NMF is to decompose the image into the product of two matrices (possibly rectangular) with low rank. Let $\tilde{F} \in \mathbb{R}^{m \times n}$, we want to find two matrices W and H such that the reconstruction of $\tilde{F}$, i.e $\hat{F}$, is defined by their product $\hat{F} = WH$. In particular, the rank of W and H, r, can be selected depending on the amount of information to be retained, such that $W \in \mathbb{R}^{m \times r}$ and $H \in \mathbb{R}^{r \times n}$. We wrote the code for NMF from scratch, starting from the iteration described in [4], which we will include below for completeness. Given a rank r, a noisy image $\tilde{F}$ of size $m \times n$, and an iteration index t:

1. for $t = 0$ (initialization) sample $W_{(0)} \in \mathbb{R}^{m \times r}$ and $H_{(0)} \in \mathbb{R}^{r \times n}$ from the standard uniform distribution.
2. for the following iteration steps, update W and H in the following way:

$$H_{(t+1)} = H_{(t)} \otimes \left[\left(W_{(t)}^T \tilde{F} \right) \oslash \left(W_{(t)}^T W_{(t)} H_{(t)} \right) \right]$$

$$W_{(t+1)} = W_{(t)} \otimes \left[\left(\tilde{F} H_{(t+1)}^T \right) \oslash \left(W_{(t)} H_{(t+1)} H_{(t+1)}^T \right) \right]$$

where $\otimes$ and $\oslash$ represent, respectively, the Hadamard product and Hadamard division operations.

The stop criterion for Step 2 is arbitrary, but since image pixels are on a 0-255 scale and we turn the images into floating point matrices, a tolerance on the maximum absolute difference between the reconstruction and the original image of $1/255 \approx 0.004$ (maximum error of 1 intensity level) would be a sensible choice. During the experimental phase, we will select a rank which minimizes the loss over the entire training set. An example of the output of NMF for different rank values r is included in Figure 7.

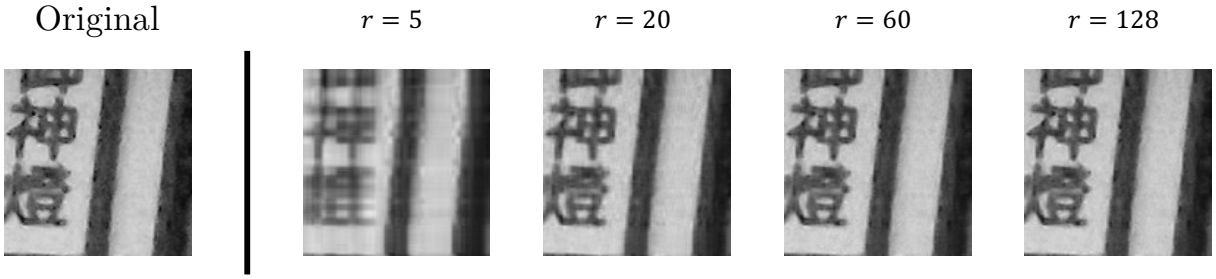

Figure 7: Image quality and submatrices rank r for NMF. Note how large amounts of image features are lost for a low rank r. $r = 60$ seems to remove some of the noise in the original image without discarding too much information. The image is a 128×128 patch from Figure 2.

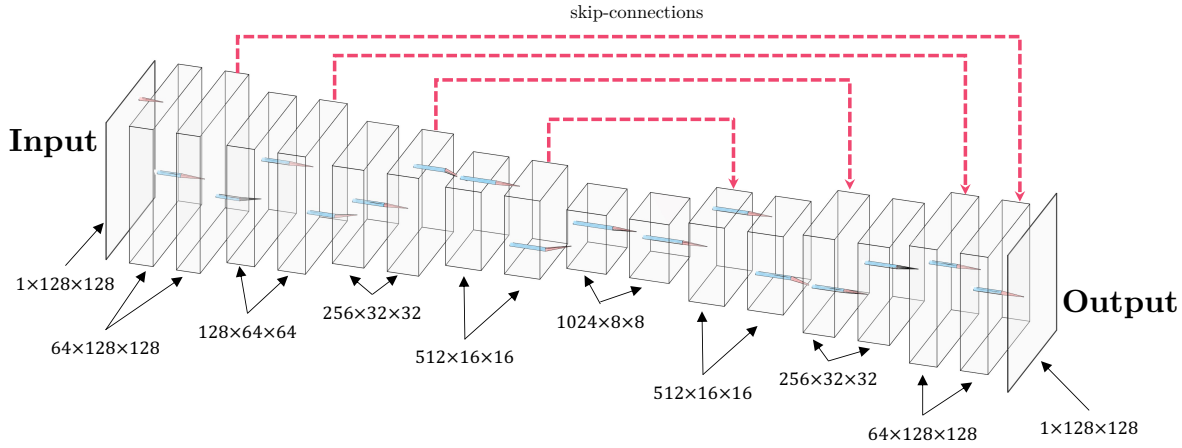

Figure 8: The U-Net structure.

4.4 Deep Learning methods

There are multiple ways to attack the denoising problem using Deep Learning, such as Dense Neural Networks (DNN) and Convolutional Neural Networks (CNN). The problems with using DNNs for computer vision are related to their focus on the global structure of the patterns rather than the local one and to the massive amount of memory required to keep track of their fully connected structure. As the relationship between pixels is inherently local, a CNN, which uses convolutions, is able to exploit the features of an image at a fraction of the computational cost of a DNN. Within the CNN category, there are a multitude of suitable architectures for denoising. One of the most popular ones is the U-Net architecture introduced in [5]. U-Nets found their first applications in the field of semantic segmentation for MRIs. However, more generally, these models are suitable in any context requiring an image-to-image mapping, such as denoising. We drew the 3D structure of U-Net and included it in Figure 8. In U-Net, there are pairs of blocks close to each other. These pairs are a sequence 7 operations: Convolutional Layer (with kernel of size 3×3), 2D Batch Normalization, Activation Function (we used the Gaussian Error Linear Unit - GELU - instead of the common RELU as the former has lead to a lower loss value on the training set), a second Convolutional Layer, a second 2D Batch Normalization, a second Activation Function, and 2×2 max pooling, where the width and height of the output are shrunk by half. The skip connections, included at the top of the network graph, allow the information to flow to the deepest layers of the network. To be more precise, the skip-connections allow the mitigation of the vanishing gradient and of the saturation problems one would encounter when training deep neural networks. Overall, the model includes approximately 31 million parameters.

We built the U-Net architecture from scratch in `PyTorch`. As the network tends to require a lot of memory, we limited the batch size to 128 images (thus requiring 29 iterations to complete one epoch over the entire training set). We trained the model for 3000 epochs. The outcome of U-Net denoising on a single image can be found in Figure 9.

5 Performance Assessment

In this section we will discuss the performance assessment methodology for the chosen models.

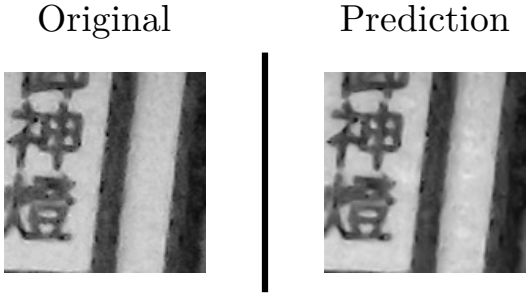

Figure 9: The U-net prediction output. The image is a 128×128 patch from Figure 2.

5.1 Data Split

In order to avoid any data leakage, we split the data set into training set and test set at the very beginning of the data processing. 90% of the images (3645 patches of size 128×128) will be assigned to the *training set*. The remaining 10% of the data set (405 images), will be split between a *machine test set* (360 images) and a *human perception set* (45 images). The details and uses of each subset are described below.

- the *training set* is exclusively used for parameter tuning, training, and preliminary performance evaluation. For each of the families of techniques described in Section 4, we will train/tune the parameters to minimize the loss on the training set for all the given models, and then pick one model for each family with the lowest overall training loss. This is done for two reasons: to decrease the burden in the output analysis and to avoid accessing the test set before the final judgement, as this could implicitly lead to bias in the final decision and require corrections for multiple testing.

- the *test set* is used for the final machine-driven performance evaluation. All the models are frozen after training them on the training set, and the inference on the test set would then simulate the efficacy during deployment. The model with the lowest loss on the test set will be judged as the best model.

- we came up with the original idea of a *human perception set*. Normally, we let computers judge the quality of a prediction, but this could be misleading in the case of computer vision: how do we know whether a low loss necessarily implies high denoising capabilities? Do the denoised pictures look natural/similar to the target image *in a human sense*? In order to answer these questions, we picked 45 pictures from the data set (not included in either the training set or the test set), processed them with the 4 selected algorithms resulting from the choice of one algorithm from each family in Section 4. For each of the 45 images, we created a page in PDF format, with the ground truth at the top of the page and the 4 different model outputs at the bottom. We then shuffled the denoised images, so that the location of the output of each model would change across the pages, and asked 2 surveyees to answer the 45-page questionnaire. In case of match with the test set results, we will accept that the loss is indeed explanatory and useful enough to lead to unsupervised conclusions on the model quality. If the human perception test were not to match the results of the test set, further considerations and/or an improved loss would have to be considered.

5.2 Loss

In literature, multiple definitions for the loss have been explored. Among them, $\mathcal{L}_1$ and $\mathcal{L}_2$ losses are the most common for denoising problems. The debate is open on which loss leads to the best results: $\mathcal{L}_2$ is sensitive to outliers, whereas $\mathcal{L}_1$ makes minimizing small differences between the prediction image and the ground truth difficult. Given an image data set of size Z indexed by z, the ground truth for an image $F^{(z)}(x,y) \in \mathbb{R}^{m \times n}$ and a prediction $\hat{F}^{(z)}(x,y) \in \mathbb{R}^{m \times n}$, for (x,y) indicating a pair of coordinates in the image, the $\mathcal{L}_1$ and $\mathcal{L}_2$ losses are defined as below:

- $\mathcal{L}_1^{(z)} = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \left| \hat{F}^{(z)}(i,j) - F^{(z)}(i,j) \right|$

- $\mathcal{L}_2^{(z)} = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(\hat{F}^{(z)}(i,j) - F^{(z)}(i,j) \right)^2$

The average of the loss across the training set is used as an indicator for the model quality. For example $\mathcal{L}_1^{(Z)} = \frac{1}{Z} \sum_{z=1}^{Z} \mathcal{L}_1^{(z)}$ for $\mathcal{L}_1$ loss or $\mathcal{L}_2^{(Z)} = \frac{1}{Z} \sum_{z=1}^{Z} \mathcal{L}_2^{(z)}$ for the $\mathcal{L}_2$ loss.

We do not want to constrain ourselves to one loss or the other, but rather merge these into a loss enjoying the benefits of both: this is exactly what the Huber Loss was meant for!

The (pixel-wise) Huber Loss is defined as:

$$\mathcal{L}_{\text{Huber}}(i,j) = \begin{cases} 0.5\left(\hat{F}^{(z)}(i,j) - F^{(z)}(i,j)\right)^2 & \text{if } \left|\hat{F}^{(z)}(i,j) - F^{(z)}(i,j)\right| < \epsilon \\ \epsilon\left(\left|\hat{F}^{(z)}(i,j) - F^{(z)}(i,j)\right| - 0.5\epsilon\right) & \text{otherwise} \end{cases} \tag{1}$$

For some small threshold value ϵ. Equation 1 reveals that both $\mathcal{L}_1$ and $\mathcal{L}_2$ losses are used depending on whether the difference between the ground truth and the prediction is large or small, respectively. In literature, it is common to use a value $\epsilon = 0.2$ when the ground truth and the output are in the range $[0, 1]$. We will adopt this value for our calculation.

6 Hardware Setup

We used different machines for different tasks.

- **a MacBook Pro 16-inches, 2021, with CPU Apple M1 Max 10 cores, 64GB of RAM** for the tasks of: preliminary analysis, mean value filtering, median value filtering, maximum value filtering, minimum value filtering, NLM, PCA, NMF and all the related graphic tasks, as well the survey generation and the results processing.
- **a workstation with 30 cores, 200 GB of RAM and 1 nVidia A100 GPU with 40GB of dedicated memory** for the task of U-Net training. The training of U-Net for 3000 epochs requires approximately 6s/epoch on the nVidia A100, for a total training time of 5 hours. The same training on the MacBook Pro would have required 10 times that amount (on average, the GPU of the M1 Max would process 1 epoch in 61 seconds) thus resulting in 50 hours of training. For this reason, we have decided to include the weights of the trained U-Net model in the submitted folder (with the filename `model_03000_huberloss.pt`).

7 Results

7.1 Training

7.1.1 Spatial Noise Filters

We applied Mean Value Filter, Median Value Filter, Maximum Value Filter and Minimum Value Filter for window sizes $S = \{1, 3, 5, 7, 9, 11\}$ on the entire training set. For each image, we recorded the image size which minimizes the Huber loss with respect to the target. This allowed to obtain 4 distributions (one for each of the filter types). The distributions are included in Figure 10. Minimum and Maximum Value filters have, as best window size, 1, which means that unfiltered images would perform better than the filtered ones, on the given Huber Loss. For both Mean and Median Value filters, the mode of optimal window size across the training set is equal to 3. For each of the filters, after observing the distributions of optimal window size, we filtered *all* the images in the training set for the best window size and calculated the Huber loss, whose lowest value is achieved for the Median Filter, which will then be chosen as the best method in the Spatial Noise Filter family. Some clarifications must be made though: maximum and minimum value filters work best on pepper and salt noise, respectively, but this kind of noise is not common in this data set. As for the superiority of the median filter in comparison with the mean filter, we notice that - as shown in Figure 3 - there is a certain amount of impulse noise, which the median filter effectively suppresses. However, the very small difference between the two losses for mean ($\mathcal{L}^{\text{Mean}}_{\text{train}} = 0.00030$) and median ($\mathcal{L}^{\text{Median}}_{\text{train}} = 0.00028$) filter may indicate that the latter's superiority is only due to chance.

7.1.2 Non-Local Means Denoising (NLM)

For each of the images in the training set, we applied NLM with all the combinations of the following parameters.

- scaling parameter $h \in \{4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 88, 92, 96, 100\}$
- search window size $(2r + 1) \in \{21, 35\}$
- patch size $(2f + 1) \in \{3, 5, 7, 11\}$

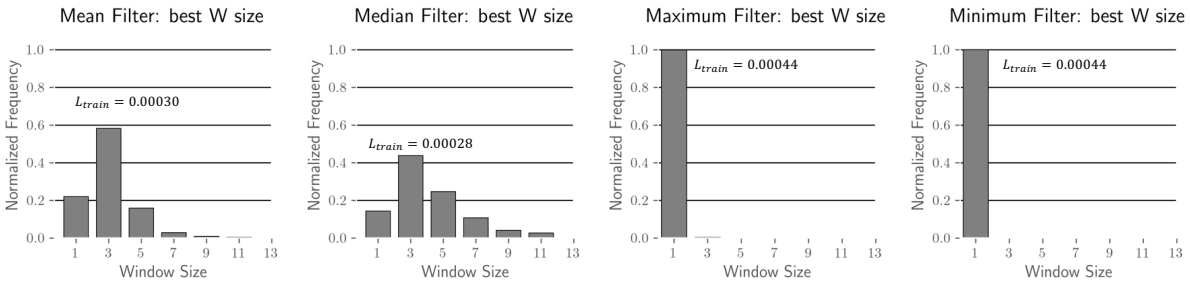

Figure 10: The distributions for the best window size for Mean/Median/Maximum/Minimum value filters. The losses on the training set are, respectively, $\mathcal{L}_{\text{train}}^{\text{Mean}} = 0.00030$, $\mathcal{L}_{\text{train}}^{\text{Median}} = 0.00028$, $\mathcal{L}_{\text{train}}^{\text{Max}} = 0.00044$, $\mathcal{L}_{\text{train}}^{\text{Min}} = 0.00044$.

From the combinations, we picked the one guaranteeing the lowest loss over the entire training set. This resulted in opting for $h = 12.0$, patch size $= 3$, and search window size $= 21$, with a total loss of $\mathcal{L}_{\text{train}}^{\text{NLM}} = 0.00022$. Non-local means is a very powerful algorithm that is sometimes employed in commercial photo retouch software on mobile devices. It comes as no surprise that this achieves such a low loss.

7.1.3 Dimensionality reduction methods

We then focused on PCA and NMF. In particular, for each of the images:

- for **PCA**: we obtained the SVD of the image and calculated the amount of retained variance guaranteeing the lowest image-wise Huber Loss. This resulted in a distribution over the optimal retained amount of variance for each image. As the distribution is highly skewed (Figure 11.a), we took the median of the values, resulting in an optimal retained variance of 97.92%.
- for **NMF**: we looped over a finite set of ranks $r = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120\}$ and recorded - for each image - the loss associated with each rank r. Then, for each rank, we averaged the loss across the entire training set and chose the rank $\hat{r}$ guaranteeing the lowest average loss. This resulted in opting for $\hat{r} = 50$ (although we admit that the difference with $\hat{r} = 40$ is minimal), as can be seen in Figure 11.b .

PCA and NMF achieve, in their respective optimal setup described above, on the training set, a loss of $\mathcal{L}_{\text{train}}^{\text{PCA}} = 0.00079$ and $\mathcal{L}_{\text{train}}^{\text{NMF}} = 0.00034$. PCA loss is higher than NMF and is therefore discarded from further consideration. A question lingers though: why does PCA exhibit such a poor performance on the data set? In order to answer this question, we should go back to the description of the inner workings of PCA in Section 4.3.1. As PCA finds the singular values of a matrix and sorts them by magnitude, depending on the chosen amount of preserved variance, smaller singular values (and high order modes - a concept associated with the spatial frequency) are discarded. These smaller values are associated with smaller artifacts in the image that are therefore treated as noise. However, noise does not necessarily appear as a high-frequency phenomenon, and losing high-frequency components may in fact have a negative effect on the loss, which results in PCA under-performing even in comparison with the unretouched noisy images. NMF, instead, does not treat the noise as a high-frequency phenomenon, which explains its effectiveness for image denoising.

7.2 U-Net

The results for U-Net on the training set are quite good. As optimizer, we employed `AdamW`. `Adam` has been the default amateur's choice for training neural networks since its original paper in 2014 [6]. Most of the largest state-of-the-art networks available are instead training using Stochastic Gradient Descent (`SGD`). The reason for this lies in the fact that SGD, although more time consuming, leads to models that tend to generalize better on unseen data. Both `Adam` and `SGD` employ $\mathcal{L}_2$-regularization on their weights (i.e. tend to favor models with smaller weights) however, the implementation of $\mathcal{L}_2$-regularization for the traditional `Adam` algorithm has been shown to be ineffective. More recent literature [7] has pointed out this issue, and has come up with a newer approach, `AdamW`, where the weight decay is applied post-parameter-wise check rather than prior to it.

The loss tends to be quite noisy for the first 1000 epochs, but it keeps decreasing, on average, until approximately 2500 epochs. It then reaches a plateau, and we stopped the training at 3000 epochs. The graph of the loss can be found in Figure 12. At the 3000th epoch the reached training loss was $\mathcal{L}_{\text{train}}^{\text{U-Net}} = 0.00003$, the lowest among the trained models.

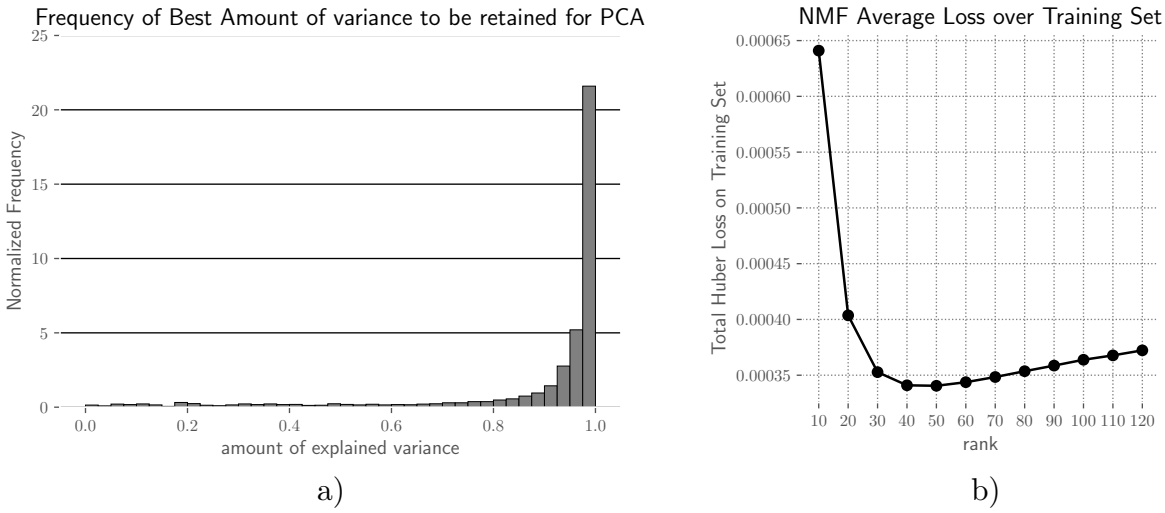

Figure 11: a) the distribution of optimal preserved variance for PCA over the training set with median 97.92%; b) rank-wise NMF average loss over training set. Minimum achieved for rank $\hat{r} = 50$.

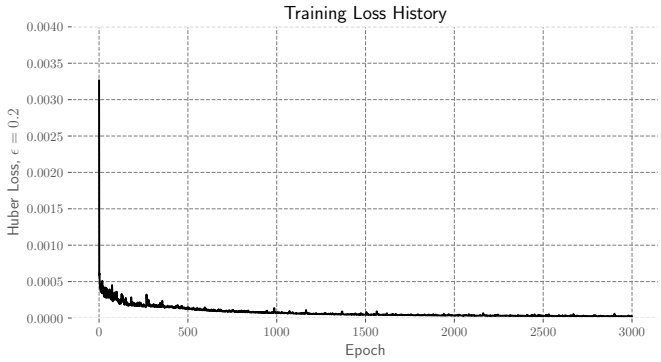

Figure 12: The training loss history for U-Net across the 3000 epochs it has been trained for.

7.3 Test Set

For the reasons above, the final comparison over the machine test set will involve only Median Value Filter, NLM, NMF and the U-Net model. On the 360-image machine test set, the losses for each of the models are $\mathcal{L}_{\text{test}}^{\text{Median}} = 0.00029$, $\mathcal{L}_{\text{test}}^{\text{NLM}} = 0.00024$, $\mathcal{L}_{\text{test}}^{\text{NMF}} = 0.00034$, $\mathcal{L}_{\text{test}}^{\text{U-Net}} = 0.00016$. For comparison, on the training set, the models achieved the following performances: $\mathcal{L}_{\text{train}}^{\text{Median}} = 0.00028$, $\mathcal{L}_{\text{train}}^{\text{NLM}} = 0.00022$, $\mathcal{L}_{\text{train}}^{\text{NMF}} = 0.00034$, $\mathcal{L}_{\text{train}}^{\text{U-Net}} = 0.00003$. These results on the test set allow us to conclude that:

- the ranking of model performance does not change between training set and test set. The ranking, sorted by decreasing performance, is: 1) U-Net; 2) NLM; 3) Median Value Filter; 4) NMF.
- while most models' performances do not vary between training set and test set, the gap for the U-Net loss is quite large (the loss on the test set is 5 times that on the training set). This could be a red flag for overfitting on the training set, and further evaluations could be done to improve the model performance by increasing the probability of dropout in the dropout layers, for example, which would add stochasticity and robustness to the model.

Even though the U-Net is the best performing model for the given Huber loss, we now need to answer the question as to whether this lower loss is perceived by the human eye. In order to do this, we will resort to a survey administered to 2 people. The results for the survey (comprising 45 images) are included below.

7.4 Human Survey

We administered the 45-picture survey to 2 surveyees, thus leading to 90 answers in total. The results are included in Table 1. We notice that, in the vast majority of the pictures, the comparison via human eye chose the U-Net results over NLM, therefore, even though the difference between NLM and U-Net might appear small, it is non-null. Also,

Table 1: Results for the Human Perception data set.

Ranking	Model Name	Selected as Best on
1	U-Net	56 images
2	NLM	28 images
3	Median	4 images
4	NMF	2 images

there is a stark contrast between the performance of U-Net and that of Median value filter/NMF, which reveals these last two methods should not be applied to the current data set. A question comes natural though: what is the average absolute value (and the error) we would expect for the performance improvement of U-Net over NLM, irrespective of the current realization of the 45-image the data set?

We can bootstrap over this data set 100000 times, create the difference distribution for the means, and calculate the estimates for both mean and standard deviation for such distribution. The results show that the difference is a normal distribution centered at 27.99 images with standard deviation of 8.66 images. Which means that in 99.91% of the scenarios from the bootstrapped set, U-Net outperforms NLM on average. Therefore we do accept that U-Net is in fact superior to NLM.

8 Conclusion

Our analysis has brought to light the superiority of a Deep Learning (DL) model (U-Net) as compared to more traditional denoising algorithms. This result is both meaningful and useful, because it motivates us to keep researching and building even better models in the future. Two points need to be stressed though:

- DL models are more data-reliant than non-DL models. As a result, larger models require, in general, more data in order to be trained to a satisfactory level of accuracy. If the available data set is limited in size, DL models may not be the most suitable tool and NLM may be the winner.
- DL models of this size can be trained in a reasonable amount of time only if a GPU is available. Although GPUs are becoming cheaper and more widespread, the GPU we used - nVidia A100 - features a MSRP of $10000, which is neither affordable nor reasonable for a purchase from a private user. The usage of this graphic card in the cloud allowed us to greatly reduce the cost. However, high-end GPUs are power hungry, an issue which demands ethical and ecological considerations of the real benefits of using a DL model over a non-DL one.

In conclusion, we acknowledge the power of U-Net for denoising, and recommend its usage, but we also believe something more should be done to further improve NLM.

References

[1] T. Plötz and S. Roth, "Benchmarking denoising algorithms with real photographs," *CoRR*, vol. abs/1707.01313, 2017. [Online]. Available: http://arxiv.org/abs/1707.01313

[2] A. Barbu, "Renoir data set," http://adrianbarburesearch.blogspot.com/p/renoir-dataset.html.

[3] A. Buades, B. Coll, and J.-M. Morel, "Non-local means denoising," *Image Processing On Line*, vol. 1, pp. 208–212, 2011.

[4] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999. [Online]. Available: https://doi.org/10.1038/44565

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

[6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *3rd International Conference for Learning Representations*, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[7] I. Loshchilov and F. Hutter, "Fixing weight decay regularization in adam," *CoRR*, vol. abs/1711.05101, 2017. [Online]. Available: http://arxiv.org/abs/1711.05101