Reconstructing images with convolutional denoising auto-encoder

V Desgrange[1], M Theodosiou[2] and M Tienstra[3]

[1] University of Amsterdam, Faculty of Science
[2] Utrecht university
[3] Leiden University

E-mail: viviane.desgrange@student.uva.nl

E-mail: m.e.theodosiou@students.uu.nl

E-mail: m.s.z.tienstra@umail.leidenuniv.nl

Abstract. Image denoising and image deblurring are classical ill-pose inverse problems. A unique solution exists but this solution is often not continuously dependant on the noise level. we can resolve this ill-posedness by introducing regularization of the reconstructions. We know that common methods are able to produce stable reconstruction but are interested in performance of a convolutional denoising auto-encoder in such tasks, as we have seen other deep learning methods outperform the more regular methods. In particular we would like to explore if convolutional denoising auto-encoder give stable reconstructions, as this is not a guarantee. We review and test common algorithms in as denoising/deblurring algorithms on the grey scaled MNIST data set, and will compare their reconstruction to the convolutional denoising auto-encoder using the peak signal-to-noise ratio,PSNR, and structural similarity index measure, SSIM, as measures of how good the reconstructions are.

Introduction

During the course "Inverse problems in imaging" we have worked with variational methods that are used for image denoising or image deblurring problems. However, they are not adaptive to data, in contrast to deep learning methods e.g. denoising auto-encoders. Neural networks have proven to give very promising results and in our report we will discuss both methods and compare their results in detail. Various methods have been used in []

Auto-encoders:

Auto-encoders are used widely in dimension reduction and image denoising applications. Briefly, the auto-encoder is trying to reconstruct the input data, i.e. image, in the most accurate way possible. In order not to let the auto-encoder learn the identity function to map the input to the output data, we build a "bottleneck", called the latent representation. This makes sure that the bottleneck will learn a well-compressed description of the input data. In reality, an auto-encoder first "encodes", i.e. compresses, the image into a lower-dimension space and then "decodes", i.e. decompresses, this information to the other half of the network. Inevitably this leads to information loss of some extent. No matter, the decoder utilises what it has learnt in training about the dataset to reconstruct the input data in the best way possible.

There are many variations on the basic idea of an auto-encoder network. Auto-encoders are inclined to fail if the input is quite different than the training dataset. In our project we will be using a denoising auto-encoder with convolutional layers. During training, we add a layer of noise before putting the image in the network. The loss function is based on the differences

between the reconstructed image by the auto-encoder and the original image. In this way, our trained auto-encoder is ready to denoise a noisy image. When we train the auto-encoder we train it to minimize the loss. The denoising auto-encoder is fed with noisy images in order to reconstruct a clean version of the image by learning certain features i.e. weights. As soon as the loss is computed, the weights are updated using back propagation. The combination of forward pass, i.e. an epoch, and back propagation are repeated to train the model, i.e. update the weights, until the asked number of epochs are completed. One of the best methods to help modelling image data is a convolutional auto-encoder. Normally a neural network would start slicing the image row-by-row and stack the data. Alternatively, a convolutional auto-encoder would preserve the spatial information of the image, and keep some part of the data in the convolution layer.

In the first section of the report we will focus on the theory that was mainly provided from the course. We will discuss the ill-posedness of the inverse problem, define the forward model, showcase the variational problem and display a possible strategy, i.e. use the total variation (TV) of u, to solve it.

In the second section we will discuss the numerical implementations to solve the variational problem making use of tools given from the course. These are the following: the Perona-Malik method, the Rudin-Osher-Fatemi PDE minimisation via gradient flow, the proximal gradient descent and the alternating direction method of multipliers, i.e. the ADMM.

In the third section, we will test these four methods and compare their results on denoising and deblurring with those from the auto-encoder. For the comparison we will make use of different noise and blurring levels. Additionally, we will introduce two error metrics, the peak to signal-to-noise ratio, i.e. the PSNR, and the structural similarity index measure, i.e. the SSIM, to evaluate our reconstruction methods on the MNIST dataset.

1. Theory

1.1. Background and Ill-posedness

Suppose we have some function $f(x)$ where $f : \Omega \to \mathbb{R}$, and $\Omega \subset \mathbb{R}^d$. Set $d = 2$ for simplicity. Assume we observe a noisy version of f, denoted as f^δ, and that we want to compute the derivative of f^δ. Further assume that this noisy version f^δ is f with additive noise. That is,

$$f^\delta(x) = f(x) + n^\delta(x), \qquad x \in [0, 1] \tag{1}$$

and boundary conditions $f^\delta(0) = f(0) = 0$, $f^\delta(1) = f(1) = 0$, and $n^\delta(x)$ is the data noise. Suppose further that the noise at each point x is normally distributed with mean zero and variance $\delta > 0$. Additionally, we assume independence for different measurements, e.g. of points x_1 and x_2. Then, by the Law of Large numbers we have that

$$\int_0^1 |n^\delta(x)|^2 dx \approx \delta^2 \tag{2}$$

Even in the case where we know exactly the noise δ and is arbitrarily small we cannot estimate df/dx. In addition, n^δ may not be differentiable. Let us assume for the moment that n^δ is differentiable. The other problem is that the noise can be arbitrarily large. So we see that numerical differentiation of noisy functions is an ill-posed problem.

1.2. Forward model

In image denoising the forward model is defined by $Ku = f$ where K is a linear operator, u is the image and f are the measurements of this image. We corrupt this image by adding noise giving us $f^\delta = Ku + n^\delta$, where n^δ is some random noise function dependent on the noise level δ. In the simple denoising case, where we are only adding noise, K is the identity matrix. Therefore the inverse problem we try to solve is the following:

$$u^\delta(x) = u(x) + n^\delta(x), \qquad x \in \Omega. \tag{3}$$

In image denoising our goal is to find a good approximation of the original image u based on the corrupted image u^δ without oversmoothing. This involves preserving edges which are related to the derivatives of u hence we are in the case of differentiating a noisy function u^δ, which would lead to an ill-posed problem

We mentioned above the case where we had additive noise. However, in the case of image blurring the image u is first convolved with a type of linear kernel and the model is now

$$f^\delta(x) = K(y) * u(x) = \int_\Omega k(x-y)u(y)dy + n^\delta, \qquad x \in \Omega. \tag{4}$$

We again aim to find a good approximation of the original image based on f^δ and known convolutional kernel but with the added problem that the convolution cause a lost of information. To see that this problem is ill-posed we can apply the convolution theorem to $f(x) = K(y) * u(x) = \int_\Omega k(x-y)u(y)dy$ and get

$$f = 2\pi F^{-1}(F(u)F(k)) \tag{5}$$

For a Hilbert space the inverse Fourier transformation is unique and so we can solve for u

$$u = 2\pi F^{-1}\left(\frac{F(u)}{F(k)}\right) \tag{6}$$

Even though we have a unique solution when we add n^δ, we have

$$2\pi|u - u^\delta| = \left|F^{-1}\left(\frac{F(f-f^\delta)}{F(k)}\right)\right| \tag{7}$$

$$= \left|F^{-1}\left(\frac{F(n^\delta)}{F(k)}\right)\right| \tag{8}$$

for large frequency $F(k) \to 0$, while $F(n^\delta) \nrightarrow 0$ for high noise. So the error does not depend continuously on the noise and therefor we have an non-stable solution. This implies that the problem is ill-posed.

1.3. Variational problem

In order to solve the ill-posed inverse problem $f^\delta = Ku + n^\delta$, we introduce a regularizer. The problem to solve is then given by a variational method

$$\min_u ||Ku - f||^2 + \alpha R(u) \tag{9}$$

where $D(u, f) = ||Ku - f||^2$ is the data fidelity term and $R(u)$ is a type of regularizer of u and α the smoothing parameter (ie the amount of regularization added). We are particularly

interested in the variational problem where the regularizer $R(u) = TV(u)$, i.e. is equal to the total variation of u. Then, the minimization problem becomes the following:

$$\min_u ||Ku - f||^2 + \alpha TV(u) = \min_u ||Ku - f||^2 + \alpha \sup_{\substack{\phi \in C^\infty(\Omega, \mathbb{R}^d) \\ ||\phi||_\infty \leq 1}} \int_\Omega u \cdot \nabla \phi \tag{10}$$

This type of regularization will guarantee the stability of the reconstructions.

1.4. Solution Strategy

In order to solve the variational minimization problem in a practical setting, we need to estimate the total variation of u. If we assume that $u \in W^{1,1}$, then

$$\begin{aligned} TV(u) &= ||\nabla u||_{L_1} && (11)\\ &= \int_\Omega |\nabla u|_{\ell_1} d\tilde{x} && (12)\\ &= \int_\Omega |\nabla u| d\tilde{x} && (13)\\ &= \int_\Omega |u_x| + |u_y| d\tilde{x} && (14) \end{aligned}$$

where u_x and u_y denote the partial derivative in the first and second direction respectively. Since we do not know the space where u lies, we can still use (11) as a good approximation of u. This kind of regularization is then called *lasso*/sparse regularization of the gradient of u or antitropic TV.

2. Implementation

Recall that we wish to solve

$$\min_u ||Ku - f||^2 + \alpha \int_\Omega |\nabla u| \tag{15}$$

where $\int_\Omega |\nabla u|$ is an approximation of $TV = (u)$. There are several different numerical implementations to solve the minimization problem above. For example, the Perona-Malik method, and the Rudin–Osher–Fatemi PDE minimize via gradient flow. While other methods are using use a generalized form of gradient descent such as proximal methods. Proximal gradient descent and alternating direction method of multipliers (ADMM) are two well known examples. However we are interested in seeing the effects when we replace these methods with a different kind of denoiser, mainly a convolutional denoising auto-encoder with convolutional layers.

Methods such as, Perona-Malik, Rudin–Osher–Fatemi PDE, and proximal methods with the correct parameters converge and give stable reconstruction. It has also been shown [insert ref] the auto-encoders do not give stable reconstructions. However, auto-encoders are adaptive to data, relatively easily to train and implement, and do give "good" reconstruction. Good in the sense that the reconstruction have similar or higher peak signal to noise ratio (PSNR) error when compared to the above mentioned methods. We tested all four of these methods and will compare the reconstruction with those from the auto-encoder.

2.1. Numerical Implementation

As a benchmark we implemented more rudimentary methods of denoising. We compared the reconstruction of the convolutional auto-encoder with the reconstructions from the Perona-Malik method, Rudin–Osher–Fatemi PDE tv denoising, the Chambolle-Pock proximal method, and the ADMM.

2.2. Diffusion Algorithms

The Perona Malik method of regularization is an anitsotropic diffusion that removes noise via filter with a diffusion coefficient. Let $r(u)$ denote the regularizer then

$$r(u) = \log(1 + ||u||_2^2/\epsilon^2) \tag{16}$$

which by using the Lagrange Theorem gives us the Persona-Malik evolution equations:

$$\frac{\partial u}{\partial t} + u - \alpha\nabla \cdot \left(\frac{\nabla u}{1 + \epsilon^{-2}||\nabla u||_2^2}\right) = f^\delta. \tag{17}$$

Alternatively, using the ROF-method let

$$r(u) = \int_{\mathbb{R}} |\nabla u|_{euclidean} \tag{18}$$

then the Euler Lagrange equation are given by:

$$\begin{cases} u(x) - \alpha\nabla\left(\frac{\nabla u}{\sqrt{|\nabla u|^2}}\right) = f^\delta & u \in \Omega \\ \frac{\partial u}{\partial n} = 0 & u \in \partial\Omega \end{cases} \tag{19}$$

which then gives the evolution equation

$$\frac{\partial u}{\partial t} + u - \alpha\nabla\left(\frac{\nabla u}{\sqrt{|\nabla u|^2}}\right) = f^\delta \tag{20}$$

The difference between the two equation can be seen as the difference in the coefficient

$$\frac{1}{1 + \epsilon^{-2}||\nabla u||_2^2} \text{ vs } \frac{1}{\epsilon + \sqrt{|\nabla u|^2}}. \tag{21}$$

Having constant coefficients the two evolution equations are reduced to the heat equation. This is equivalent to Gaussian blurring. For this reason we see blurring in the reconstruction.

2.3. ADMM algorithm

Instead of using the heat equation to denoise, we test methods such as the Chambolle proximal method and ADMM. These methods use projections to a convex set to solve non-differentiable convex optimization problems. Recall that the regularizer was the l_1 regularization of the ∇u, where the l_1 norm is convex but not differentiable. First we describe ADMM method descent.

For ADMM, Let the functional be of the following form: $J(u) = ||Ku - f||^2 + \alpha R(u)$ and suppose that we can split $J(u) = D(u) + R(Au)$, where $D(u)$ is smooth, $R(Au)$ is a regularizer and A is a linear operator. Here A is an addition to the model we saw before in the proximal gradient descent case. Suppose we want to solve,

$$\min_{u\in\mathbb{R}^n} D(u) + R(Au) \tag{22}$$

with D smooth and $\mu-$strongly convex, $R(\cdot)$ convex and $A \in R^{m\times n}$ a linear map. It is possible that $R(\cdot)$ admits an efficient proximal operator but $R(A\cdot)$ will not. To shift the operator A to the other part of the objective $J(u)$, we introduce an auxiliary variable ν and rewrite (22) as

$$\min_{u\in\mathbb{R}^n} D(u) + R(\nu), \tag{23}$$

where $Au = \nu$. This gives us a constrained optimization problem which can be solved with the use of Lagrange multipliers. We can now define the Lagrangian,

$$\Lambda(u, \nu, v) = D(u) + R(\nu) + \langle v, Au - \nu\rangle \tag{24}$$

where $v \in \mathbb{R}^n$ are the Lagrange multipliers. The solution to Λ is a saddle point and can be found by

$$\min_{u,\nu} \max_{v} \Lambda(u, \nu, v) \tag{25}$$

the minimization maximization in (25) is equivalent to the minimization in (23) by the saddle point theorem from the lecture notes. The dual problem to (25) is given by

$$\max_{v} \min_{u,\nu} \Lambda(u, \nu, v) \tag{26}$$

where for convex problems the primal and dual are equivalent. Supposing that are problem is convex we can solve 26 directly by the alternating directions of multiplies methods. Now we call the Lagrangian from (24 we then add the quadratic term $\frac{\rho}{2}||Au-\nu||_2^2$ and then get the following iterations:

$$u_{k+1} = \operatorname*{argmin}_{u} \Lambda_\rho(u, \nu_k, v_k) \tag{27}$$

$$\nu_{k+1} = \operatorname*{argmin}_{u} \Lambda_\rho(u_{k+1}, \nu, v_k) \tag{28}$$

$$v_{k+1} = v_k + \rho(Au_{k+1} - \nu_{k+1}) \tag{29}$$

where $\Lambda_\rho(u, \nu, v) = D(u) + R(\nu) + \langle v, Au - \nu\rangle + \frac{\rho}{2}||Au - \nu||_2^2$. Then, let's consider our functional $J(u)$ to be $K(u)$, which can be seen in the graph as concave and smooth. Also, take A to be the ∇, i.e. nabla, operator which is a linear operator, then for the regularization term $R(Au) = A(\nabla u)$ we get the following:

$$u_{k+1} = (I + A^*A)^{-1}(f^\delta + A^*(\rho\nu_k - v_k)) \tag{30}$$

$$\nu_{k+1} = \operatorname{prox}_{(\lambda/\rho)||\cdot||_1}(Au_{k+1} + \rho^{-1}v_k) \tag{31}$$

$$v_{k+1} = v_k + \rho(Au_{k+1} - \nu_{k+1}) \tag{32}$$

2.4. Chambolle-Pock algorithm

Another algorithm that was used in our implementation was the Chambolle-Pock algorithm. In fact, it is a primal-dual method for non-smooth optimisation problems.

$$\max_{y\in Y} \min_{x\in X} (\langle Ax, y\rangle_Y + g(x) - f^\star(y)) \tag{33}$$

in which X and Y are known Hilbert spaces equiped with the inner product $\langle \cdot, \cdot \rangle$ and a norm $|| \cdot ||_2 = \langle \cdot, \cdot \rangle^{1/2}$, A is a linear operator $A : X \to Y$ which is also continuous, $g : X \to [0, +\infty]$ and $f : Y \to [0, +\infty]$ are known to be functionals which are convex, l.s.c. and proper. Lastly, $f^\star$ is known as the Fenchel conjugate of f, i.e. the functional for which the following is true,

$$f^\star(x^\star) = \sup_x \left(\langle x, x^\star \rangle - f(x) \right) \tag{34}$$

We can hence get the saddle-point problem. In fact, this would be a primal dual conceptualisation for the following primal problem

$$\min_{x \in X} \left(g(x) + f(Ax) \right) \tag{35}$$

Accordingly, we can find the dual problem

$$\max_{y \in Y} \Big(g^\star(-A^\star x)) - f^{\star(y)} \Big) \tag{36}$$

where $A^\star$ denotes the adjoint of A.

The Champolle-Pock algorithm is comprised by alterations from a gradient ascent regime for dual variable y to a gradient descent regime for primal variable x.

At the beginning of the algorithm, we choose $\tau > 0$, $\sigma > 0$, $\theta \in [0, 1]$, $x_0 \in X$, $y_0 \in Y$, $\bar{x}_0 = x_0$. Therefore, for $n > 0$, we update iteratively x_n, y_n and $\bar{x}_n$ in the following manner:

$$y_{n+1} = \text{prox}_{\sigma f^\star}(y_n + \sigma A \bar{x}_n), \tag{37}$$

$$x_{n+1} = \text{prox}_{\tau g}(x_n - \tau A^\star y_{n+1}), \tag{38}$$

$$\bar{x}_{n+1} = x_{n+1} + \theta(x_{n+1} - x_n) \tag{39}$$

where prox is the proximal operator. In terms of step sizes, we could take $\tau = \sigma < \frac{1}{||A||}$. By taking $\sigma\tau||A||^2 < 1$ we guarantee the convergence of the above algorithm. Of course, it is not necessary that this choice is optimal

2.5. Convolutional denoising auto-encoder

We will now describe a one layer simple auto-encoder. Denote $\mathbb{X} = \mathbb{R}^n$ as the input, $\mathbb{F} \subset \mathbb{R}^m$ the feature space, define the encoder by $e : \mathbb{X} \to \mathbb{F}$ and the decoder by $d : \mathbb{F} \to \mathbb{X}$ such that $e, d = \min_{e,d} ||X - (d \circ e)X||^2$. Suppose further that the auto-encoder has one input layer, one hidden layer ($\mathbf{h}$), and one output layer. Then, the auto-encoder works by taking $\mathbf{x} \in \mathcal{X}$ and maps it to the hidden layer $\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$ where σ is the activation function, $\mathbf{W}$ is the weight matrix and $\mathbf{b}$ is the bias vector. We then update the weights and bias using backpropagation. Next, the hidden layer $\mathbf{h}$ is mapped to the reconstructed image in the decoder stage by $\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{h} + \mathbf{b}')$. Here σ' is the activation function, $\mathbf{W}'$ is the weight matrix and $\mathbf{b}'$ is the bias vector for the decoder and could be different than those of the encoder. Subsequently, the minimization problem becomes the following

$$\min_{\mathbf{x}' \in \mathcal{X}} \mathcal{L}(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||^2 \tag{40}$$

$$= ||\mathbf{x} - \sigma'(\mathbf{W}'(\sigma'\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')||^2 \tag{41}$$

When we train the auto-encoder we train it to minimize equation 17. The denoising auto-encoder is given noisy images and its goal is to reconstruct a clean version of the image by learning certain features. The input $\mathbf{x} \in \mathcal{X}$ is first mapped to corrupted version of itself $\tilde{\mathbf{x}}$. Then $\tilde{\mathbf{x}}$ is mapped to $\mathbf{h} = f_\theta(\tilde{\mathbf{x}}) = \sigma(\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b})$. The model's parameter θ is trained to minimize on the mean reconstruction error.

We implemented a convolutional denoising auto-encoder that consisted of convolutional and pooling layers. The input grey scaled images that are sized 28×28 pixels and computed the loss using the binary cross entropy loss function to train weights. Denote this image as U such that U is a matrix of dimension 28×28. We number the entries of U by $U_{i,j}$ for $i, j = 1, ..., 28$ which each $U_{i,j}$ represents a pixel and let the intensity of that pixel $U_{i,j}(x) \in [0, 1]$. Now denote a filter as f_n such that f_n is of dimension $n \times n$ Then when this image is passed to the convolutional layer the filter acts on the image, by $U * f_n(x)$ where $*$ denotes convolution, resulting in feature extract by taking a weighted sum of the pixel contained in the f_n. Finally the result of the convolution is acted on by an activation function and passes to the next layer, max pooling. We can see a connection between the gradient flow methods, and the convolution layers of the auto-encoder. Recall that in the Perona-Malik and ROF-TV denoising algorithms, that these methods used diffusion to denoise the image, which was essential a convolution with a 2-D Guassian kernel with a certain step size in between the pixels and diffusion rate given by the diffusion coefficients. In the max pooling layer the we look a some nn window where $n < 28$, and take the maximum pixel value in that window. We slide this window over the whole image and in the end have compressed version of U. The encoder consists of both kinds of layers mentioned above until the image is compressed to some smaller dimension. The decoder then use up sampling to output, a reconstruction that is of the original dimension. The loss is then computed and the weights are updated using back propagation. The combination of forward pass (epoch) and back propagation are repeated to train the model (ie update the weights) until the asked number of epochs are completed.

The implementation for the ROF-TV is a SciKit learn method [6], and the Perona-Malik one is the one we saw in the course. Additionally we implemented a method for proximal graident decent and ADMM in the 2D case, but these algorithms were very slow to run so we tested wtih a more optimal algorithm from ProxImaL library [4]. The code given in a separate file.

3. Results

Data-set The experiments of this paper are performed on the MNIST dataset of handwritten digits, one of the principal benchmark dataset used in visual recognition. Composed of a training set of $60,000$ and a validation set of $10,000$ normalized and anti-aliased 28-by-28 pixels grey-scale images [1], the small dimension of the data provides a simple way to quickly train the convolutional denoising auto-encoders and analyse the efficiency of this method (See Section 1).

Measurements In order to analyse the quality of the reconstruction of corrupted images, we make usage of different measurements. First, the *Peak signal-to-noise ratio* (see Eq. 42), a ratio between maximum possible power of a signal and the power of corrupting noise, based on *Mean Square Error*. Given that measurement based on MSE does not consider other images criteria such as texture, pattern, etc; we also use the *Structural Similarity Index Measure* (see Eq. 43) which consider image corruption as change in structural information, therefore it gives

[1] MNIST dataset

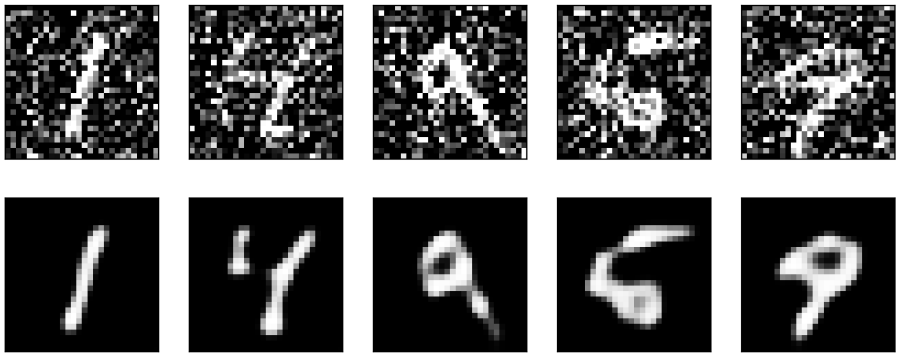
the similarity between reconstruction and ground-truth image. Additionally, the *binary cross-entropy loss function* is used during training and validation steps of the auto-encoders.

$$PSNR = 10 \cdot log_{10}\Big(\frac{MAX_I^2}{MSE}\Big) \; with \; MAX_I = 1 \tag{42}$$

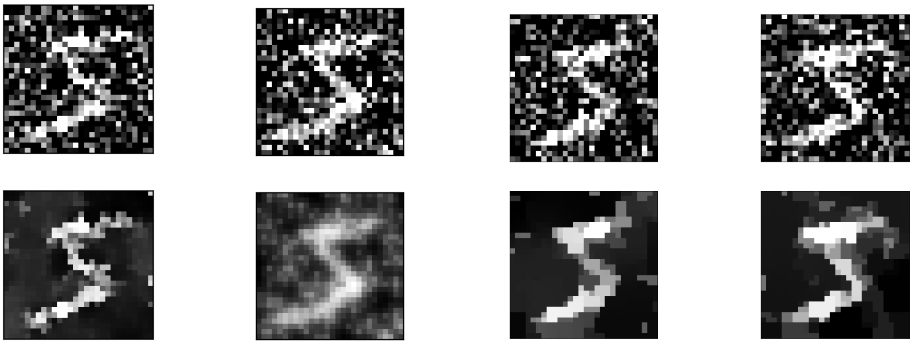
$$SSIM = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \; with \; c_1, c_2 \; for \; division \; by \; 0 \tag{43}$$

3.1. Evaluation of de-noising

Experiments The first analysis consists in corrupting the MNIST dataset images with different levels of additive Gaussian noise and try to reconstruct them with each methods previously studied (See Section 1). Considering their different nature, one of the difficulty is to perform a fair comparison between each regularization method. Therefore, it should be highlighted that the results corresponding to usual regularization methods (See Figure 1 with and Appendix) were obtained with manual adjustment of the parameters and considered as the best results obtained for the specified level of Gaussian noise.



(a) Convolutional auto-encoders (epochs = 100)



(b) Perona-Malik ($\alpha = 10$, $it = 1000$)

(c) ROF TV ($\alpha = 5$, $it = 1000$)

(d) ADMM ($\delta = .4$, $it = 1000$)

(e) Chambolle-Pock ($\delta = .4$, $it = 1000$)

Figure 1: Denoising of MNIST dataset images corrupted with additive Gaussian noise ($\mu = 0, \sigma = 0.5$). Auto-encoders performs efficient and resilient reconstruction works in comparison to other regularization methods. Additional examples with different noise distribution available in the Appendix.

As visible in the Figure 1, while the number of epochs used for training remains small (100 epochs), the convolutional de-noising auto-encoders appears to make an effective and resilient

reconstruction of the corrupted images; this is especially visible with high level of noise (See Appendix A.1). This is confirmed by the PSNR and SSIM measurements (See Table 1), which shows that both in term of pixel-wise quality and structure similarity auto-encoders provides equivalent, if not better, reconstruction than usual regularization methods.

This is especially visible if we consider the results from *Rudin-Osher-Fatemi Total Variation* or *Perona-Malik* regularization methods for de-noising tasks (See Table 1 and Appendix Appendix A.2) which results in greater loss in the structural information of the image.

	Perona-M.		**TV**		**ADMM**		**Chambolle**		**auto-encoders**		
Noise	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	Train Loss
0.1	9.973	0.25	10.85	0.020	18.73	0.59	18.69	0.59	22.67	0.68	0.104
0.2	9.972	0.25	10.78	0.023	16.69	0.53	16.71	0.53	16.75	0.6	0.097
0.3	9.972	0.25	10.58	0.025	14.98	0.47	14.97	0.47	13.33	0.51	0.099
0.4	9.972	0.25	10.29	0.026	13.50	0.40	13.48	0.40	11.0	0.43	0.106
0.5	9.972	0.25	9.94	0.025	12.30	0.35	12.28	0.35	9.38	0.36	0.118
0.6	9.971	0.25	9.57	0.023	11.27	0.30	11.38	0.30	8.24	0.3	0.130
0.7	9.971	0.25	9.26	0.021	10.56	0.26	10.56	0.26	7.43	0.26	0.144
0.8	9.971	0.25	8.98	0.020	9.99	0.23	10.02	0.23	6.82	0.23	0.156
0.9	9.971	0.25	8.72	0.018	9.56	0.21	9.52	0.21	6.35	0.2	0.168
1	9.971	0.25	8.52	0.016	9.11	0.19	9.17	0.19	5.99	0.18	0.171

Table 1: Evaluation of reconstruction methods on MNIST data-set images subjects to additive Gaussian noise with PSNR and SSIM. Over 100 simulations for Perona, TV, ADMM and Chambolle; 5000 samples for auto-encoders.

3.2. Evaluation of deblurring

Measured images can be subject to other types of corruption than additive Gaussian noise. Among them, we decided to consider the case of Gaussian blurring as a way to experiment the efficiency of auto-encoders on non-smooth optimization problem. Such task would usually be handled by *Alternating Direction Method of Multipliers* or *Chambolle-Pock* method.

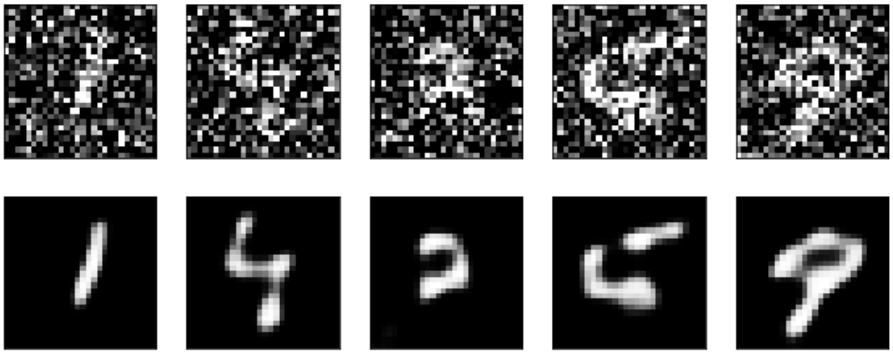
	ADMM		**Chambolle**		**auto-encoders**		
Noise	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	Train Loss
0.2	15.92	0.49	15.84	0.48	14.94	0.51	0.0975
0.5	12.01	0.31	12.00	0.33	8.8	0.28	0.1385
0.9	9.49	0.19	9.48	0.19	6.1	0.15	0.1983

Table 2: Evaluation of reconstruction methods on MNIST data-set images subjects to Gaussian blur and additive Gaussian noise with PSNR and SSIM. Run over 100 simulations for ADMM, Chambolle. 5000 sampels for auto-encoders.

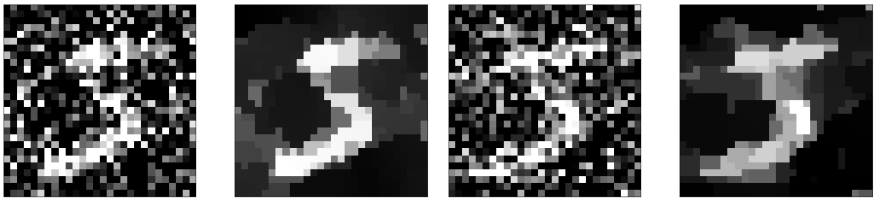
The figure 2 shows a sample of reconstruction obtained with auto-encoders in comparison to these two methods. We can observe that auto-encoders performs also well on such case (while the results are not as stable than in a pure denoising case, if we compare to Figure 1), with more consistent results that the *ADMM* and *Chambolle-Pock* methods.

4. Conclusion

Overall, one of the main drawback appears to be the computing resources required to train the auto-encoders, while the results are promising for high-level of corruption, it would be interesting



(a) Convolutional auto-encoders (epochs = 100)



(b) ADMM ($\delta = .4$, $it = 1000$)

(c) Chambolle-Pock ($\delta = .4$, $it = 1000$)

Figure 2: Reconstruction of MNIST dataset images corrupted with additive Gaussian noise and Gaussian blur ($\mu = 0, \sigma = 0.5$ in both cases).

to analyse the fair balance between usage of usual regularization method which will requires less computing and auto-encoders based on the quantity of noise. Additionally, an interesting idea would be to analyse the capacity of auto-encoders to train on a data-set subjects to a larger diversity of noise, which so far was quite limited in our experiments.

5. References

[1] Cremers D, Hazirbas C, Meinhardt T, and Moeller M 2017 Learning proximal operators: using denoising networks for regularizing inverse imagining problems (*Preprint* arXiv: 1704.03488)

[2] Elad M, Milanfar P and Romano Y 2017 The little engine that could regularization by denoising (RED) (*Preprint* ArXiv: 1611.02862)

[3] Genzel M, Macdonald J and März M 2020 Solving inverse problems with deep neural networks - robustness included? (*Preprint* ArXiv: 2011.04268)

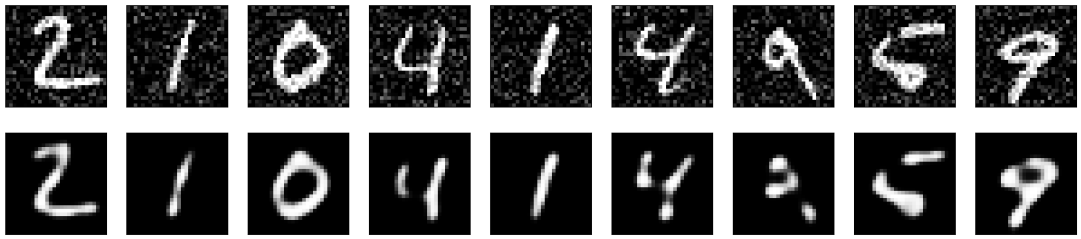
[4] Heide H, Diamond S, Nießner M, Ragan-Kelly, Heidrich W and Wetzstein G ProxImaL: efficient image optimization using proximal algorithms *ACM Trans. Graph* **35**

[5] Huttinga N, 2017 Insights into deep learning methods with application to cancer imaging (*Master Thesis*)

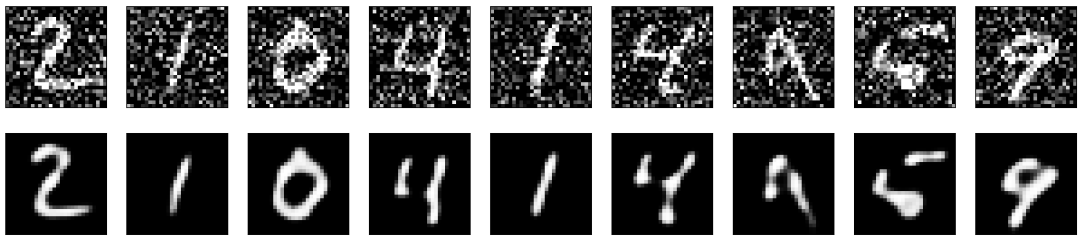
[6] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Brucher M, et al 2011 Scikit-learn: machine learning in python *JMLR.* **12** 2825-30,

Appendix A. Convolutional denoising

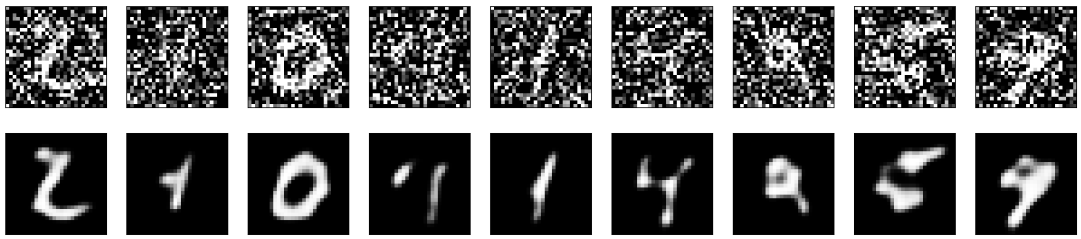
Appendix A.1. Example of denoising from auto-encoders



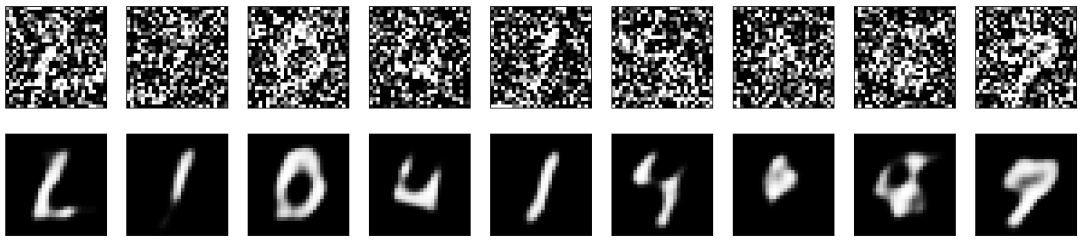
(a) epochs $= 100$, $\delta = 0.2$



(b) epochs $= 100$, $\delta = 0.4$



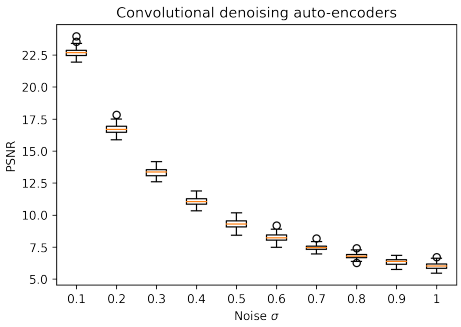
(c) epochs $= 100$, $\delta = 0.7$



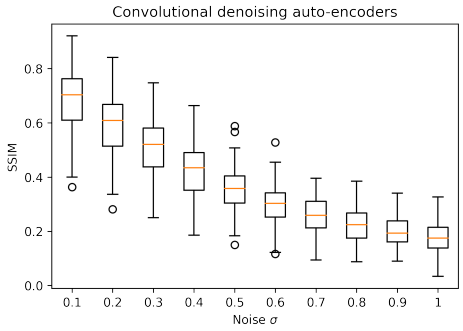
(d) epochs $= 100$, $\delta = 0.9$

Figure A1: Reconstruction of MNIST dataset images corrupted by additive Gaussian noise with usage of convolutional denoising auto-encoders. Reconstruction is efficient even for high level of noise.

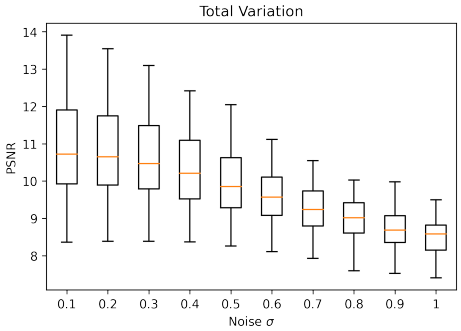
Appendix A.2. Distribution of PSNR and SSIM



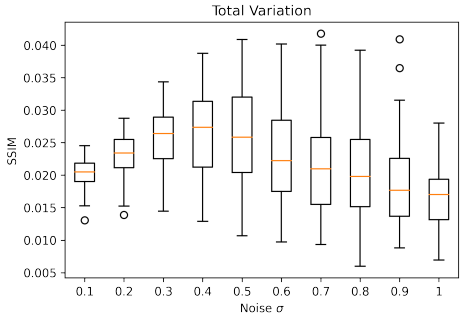
(a) Distribution of PSNR for convolutional denoising auto-encoders (100 epochs)



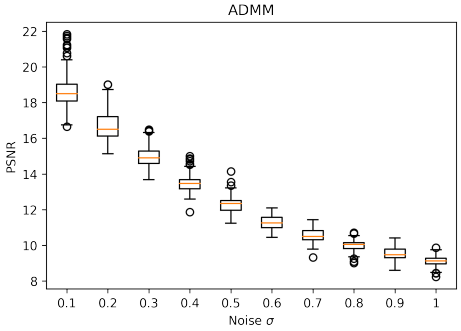
(b) Distribution of SSIM for convolutional denoising auto-encoders (100 epochs)



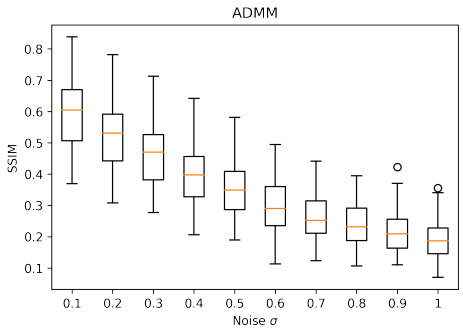
(c) Distribution of PSNR for ROF TV



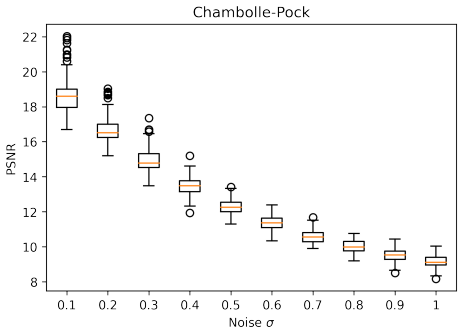
(d) Distribution of SSIM for ROF TV



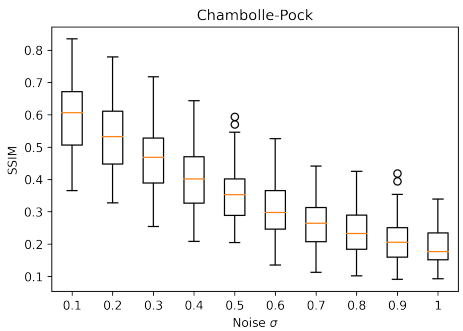
(e) Distribution of PSNR for ADMM



(f) Distribution of SSIM for ADMM



(g) Distribution of PSNR for Pock-Chambolle



(h) Distribution of SSIM for Pock-Chambolle

Figure A2: Distribution of PSNR and SSIM per level of gaussian noise for different regularization methods for a set of 100 images.