Convolutional Proximal Neural Networks and Plug-and-Play Algorithms

Johannes Hertrich | TU Berlin

j.hertrich@math.tu-berlin.de
joint work with
S. Neumayer, G. Steidl
Code available at `https://github.com/johertrich/Proximal_Neural_Networks`

Contents

Contents

Averaged Operators

Goal: Build averaged Neural Networks.

Definition

An operator $T\colon \mathbb{R}^d \to \mathbb{R}^d$ is ***t*-averaged** wrt. $\|\cdot\|$ if

$$T = tR + (1-t)I_d, \quad \text{where } R\colon \mathbb{R}^d \to \mathbb{R}^d \text{ is 1-Lipschitz continuous wrt. } \|\cdot\|.$$

Properties of averaged operators:

- 1-Lipschitz continuous.
- If $T_1, \dots, T_K$ are averaged wrt. $\|\cdot\|$, then also $T_K \circ \cdots \circ T_1$ is averaged wrt. $\|\cdot\|$.
- If $T\colon \mathbb{R}^d \to \mathbb{R}^d$ is averaged with non-empty fix-point set, then the sequence $x^{(r+1)} = Tx^{(r)}$ converges for any initialization $x^{(0)}$ to a fixed point of T.

Special case: Proximity Operators

Definition

For a proper, convex and lower semi-continuous function $f \in \Gamma_0(\mathbb{R}^d)$ and $T \in \mathbb{R}^{n \times d}$ we define the **proximity operator** by

$$\operatorname{prox}_{f,T}(x) = \operatorname*{argmin}_{y \in \mathbb{R}^d} \left\{ \tfrac{1}{2} \|x - y\|_T^2 + f(y) \right\},$$

where

$$\|x\|_T^2 := \frac{\|Tx\|_2^2}{\|T\|^2} + \|P_{\mathcal{N}(T)} x\|_2^2.$$

- proximity operators are $\frac{1}{2}$**-averaged** wrt. $\|\cdot\|_T$.
- **Obersevation:** $T^{\mathrm{T}} T = I$ or $T T^{\mathrm{T}} = I$ implies $\|\cdot\|_T = \|\cdot\|_{I_d} = \|\cdot\|_2$.
  $\rightarrow$ write prox_f for $\operatorname{prox}_{f,I_d}$

Interplay between Proximity and Linear Operators

Theorem (Hasannasab, H., Neumayer, Plonka, Setzer, Steidl, 2020)

Let $T \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^n$ and $f \in \Gamma_0(\mathbb{R}^d)$. Then,

$$T^\dagger \operatorname{prox}_{f, I_n}(T \cdot + b) = \operatorname{prox}_{g, T}, \quad \text{for some } g \in \Gamma_0(\mathbb{R}^d).$$

- Proof is based on a characterization of Moreau.
- Result holds true in Hilbert spaces.

Corollary

Let $T \in \mathbb{R}^{n \times d}$ with $T^{\mathrm{T}} T = I$ or $T T^{\mathrm{T}} = I$, $b \in \mathbb{R}^n$ and $f \in \Gamma_0(\mathbb{R}^d)$. Then,

$$T^{\mathrm{T}} \operatorname{prox}_f(T \cdot + b) = \operatorname{prox}_g, \quad \text{for some } g \in \Gamma_0(\mathbb{R}^d).$$

Proximal NN Layers

An activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is called **stable** if it is monotone increasing, 1-Lipschitz continuous and satisfies $\sigma(0) = 0$.

$\to$ Includes ReLU, ELU and the sigmoid function.

Lemma (Combettes, Pesquet, 2018)

A function $\sigma : \mathbb{R} \to \mathbb{R}$ is a stable activation function if and only if there exists $g \in \Gamma_0(\mathbb{R})$ having 0 as a minimizer such that $\sigma = \operatorname{prox}_g$.

Proximal NN layers

Let $T^{\mathrm{T}} T = I$ or $T T^{\mathrm{T}} = I$ and let σ be a stable activation function, then **NN layers of the form**

$$T^{\mathrm{T}} \sigma(T \cdot + b)$$

are **proximity operators** wrt. $\| \cdot \|_2$.

Proximal Neural Networks

Assumption: σ is a stable activation function.

Definition

Define a **Proximal Neural Network (PNN)** as

$$\Phi(\cdot; u) = L_K(\cdot; u) \circ \cdots \circ L_1(\cdot; u), \qquad L_k(x; u) = T_k^{\mathrm{T}} \sigma(T_k x + b_k)$$

with parameters $u = ((T_k)_{k=1}^K, (b_k)_{k=1}^K)$, where $T_k^{\mathrm{T}} T_k = I_d$ or $T_k T_k^{\mathrm{T}} = I_{n_k}$ and $b_k \in \mathbb{R}^{n_k}$.

- $L_k(x, u) = T_k^{\mathrm{T}} \sigma(T_k x + b) = \mathrm{prox}_g(x)$ for some $g \in \Gamma_0(\mathbb{R}^d)$.

  $\rightarrow$ Each layer is $\frac{1}{2}$-averaged wrt. $\|\cdot\|_2$.
- Averaging constant $t = \frac{K}{K+1}$.

PNNs are averaged operators!

Parameter Manifold

We have to ensure that $T^{\mathrm{T}} T = I$ or $T T^{\mathrm{T}} = I$.

Definition

For $d \leq n$ the (compact) Stiefel manifold $\mathrm{St}(d, n)$ is given by

$$\mathrm{St}(d, n) := \left\{ T \in \mathbb{R}^{n \times d} : T^{\mathrm{T}} T = I_d \right\}.$$

- The above requirement can be rewritten as
  - $T \in \mathrm{St}(d, n)$ if $n \geq d$ and $T^{\mathrm{T}} \in \mathrm{St}(n, d)$ if $d < n$.
- Orthogonal projection by the U-factor of the polar decomposition $T = US$, $U \in \mathrm{St}(d, n)$, $S \in \mathrm{SPD}(d)$.
  $\rightarrow$ Computation by the fixed point equation $U_{n+1} = 2U_n \left(I + U_n^{\mathrm{T}} U_n\right)^{-1}$, $U_0 = T$.
- The parameters $u = \left((T_k)_{k=1}^K, (b_k)_{k=1}^K \right)$ of a PNN live on the manifold

$$\mathcal{M} = \bigtimes_{k=1}^{K} \mathcal{M}_k, \quad \mathcal{M}_k := \mathrm{St}(d, n_k) \times \mathbb{R}^{n_k}.$$

Training methods

$$\text{Minimize } \mathcal{J}(u) := \sum_{i=1}^{N} \ell(\Phi(x_i; u), y_i) \text{ s.t. } u \in \mathcal{M}.$$

Possible training methods:

- SGD on $\mathcal{M}$ using the Riemannian geometry.
- Inertial stochastic PALM[1]:

$$\tilde{u}^{(r+1)} = u^{(r)} + \alpha(u^{(r)} - u^{(r-1)})$$
$$u^{(r+1)} = P_{\mathcal{M}}(\tilde{u}^{(r+1)} - \tilde{\nabla}\mathcal{J}(\tilde{u}^{(r+1)}))$$

  with stochastic gradient estimator $\tilde{\nabla}$ and $\alpha \in [0, 1)$.
- Minimize $\mathcal{J}(P_{\mathcal{M}}(u))$ with $u \in \times_{k=1}^{K}(\mathbb{R}^{n_k \times d} \times \mathbb{R}^{n_k})$ using the Adam optimizer.

[1] Ref.: H., Steidl. Inertial Stochastic PALM and Applications in Machine Learning, 2022.

Contents

Convolutional Proximal Neural Networks (cPNNs)

- Replace matrix vector multiplications Tx by convolutions, i.e.

$$T = \begin{pmatrix} \mathrm{Circ}_m(a^{(1,1)}) & \cdots & \mathrm{Circ}_m(a^{(1,m_2)}) \\ \vdots & & \vdots \\ \mathrm{Circ}_m(a^{(m_1,1)}) & \cdots & \mathrm{Circ}_m(a^{(m_1,m_2)}) \end{pmatrix}, \quad \mathrm{Circ}_m((a_1,\ldots,a_m)^{\mathrm{T}}) = \begin{pmatrix} a_1 & a_2 & \cdots & a_m \\ a_m & a_1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_2 \\ a_2 & \cdots & a_m & a_1 \end{pmatrix}$$

- We say the corresponding PNN is a **convolutional PNN (cPNN)** if $T_k \in \mathrm{bCirc}(m, m_1, m_2)$ for all $k = 1, \ldots, K$.

Lemma (H., Neumayer, Steidl, 2021)

The intersection $\mathrm{bCirc}(m, m_1, m_2) \cap \mathrm{St}(mm_1, mm_2)$ *defines a submanifold of* $\mathrm{St}(mm_1, mm_2)$. *In particular, the parameter space of cPNNs is again a manifold.*

Train cPNNs as in the non-convolutional case!

Convolutional PNNs with limited filter length

Usually, in CNNs the filters of the convolutions are sparse!

- Let $u = \left((T_k)_{k=1}^K, (b_k)_{k=1}^K\right)$ be the parameters of a PNN.
- We say the corresponding PNN is a **convolutional PNN (cPNN) with filter length** $2l+1$ if there exist $a^{(e,f)} = (a_0^{(e,f)}, \ldots, a_l^{(e,f)}, 0, \ldots, 0, a_{-l}^{(e,f)}, \ldots, a_{-1}^{(e,f)})$ such that

$$T_k = \begin{pmatrix} \mathrm{Circ}_m(a^{(1,1)}) & \cdots & \mathrm{Circ}_m(a^{(1,m_2)}) \\ \vdots & & \vdots \\ \mathrm{Circ}_m(a^{(m_1,1)}) & \cdots & \mathrm{Circ}_m(a^{(m_1,m_2)}) \end{pmatrix} \in \mathrm{bCirc}(l, m, m_1, m_2).$$

Unfortunately, $\mathrm{bCirc}(l, m, m_1, m_2) \cap \mathrm{St}(mm_1, mm_2)$ **is not longer a manifold!**

Orthogonal projection

Task: Compute orthogonal projection of $T \in \mathbb{R}^{n,d}$ onto $\mathrm{bCirc}(I, m, m_1, m_2) \cap \mathrm{St}(mm_1, mm_2)$.

- The fixed point iteration $U_{n+1} = 2U_n(I + U_n^{\mathrm{T}} U_n)^{-1}$, $U_0 = T$ increases the filter length in each step.
  $\rightarrow$ previous procedure not applicable
- Compute instead

$$\hat{T} \in \operatorname*{argmin}_S \|T - S\|_F^2 + \lambda \|S^{\mathrm{T}} S - I\|_F^2, \quad \lambda \gg 0$$

  $\rightarrow$ Minimizers converge to the orthogonal projection for $\lambda \to \infty$.

Computationally costly!

Training of cPNNs with limited filter length

Given: Samples (x_i, y_i), $i = 1, \ldots, N$

Let $\mathcal{J}(u) = \sum_{i=1}^{N} \ell(\Phi(x_i; u), y_i)$.

Goal: Solve

$$(\hat{T}, \hat{b}) \in \operatorname*{argmin}_{T_k \in \mathrm{bCirc}(I,m,m_1,m_2) \cap \mathrm{St}(mm_1,mm_2), b_k \in \mathbb{R}^{mm_1}} \mathcal{J}(u).$$

Approach: Solve the relaxed problem

$$(\tilde{T}, \hat{b}) \in \operatorname*{argmin}_{T_k \in \mathrm{bCirc}(I,m,m_1,m_2), b_k \in \mathbb{R}^{mm_1}} \left\{ \mathcal{J}(u) + \lambda \sum_{k=1}^{K} \| T_k^{\mathrm{T}} T_k - I \|_F^2 \right\}$$

for some large $\lambda > 0$. Here we use the Adam optimizer.

Finally, compute the orthogonal projection $\hat{T}$ of $\tilde{T}$ onto $\mathrm{bCirc}(I, m, m_1, m_2) \cap \mathrm{St}(mm_1, mm_2)$.

Contents

Network Architecture

Ingredients for a powerful cPNN denoiser:

- Learn the noise instead of the noise-free images ("Residual Learning")[2]
- Learn cPNNs with one additional fixed scaling parameter $\gamma \geq 1$.
  $\rightarrow$ Upper bound for the Lipschitz constant of the cPNN
- Start with m_2 copies of the input images.

Summarized, the denoiser has the form

$$\mathcal{D}(x; u) = x - \gamma A^{\mathrm{T}} \Phi(Ax; u), \quad A = \frac{1}{\sqrt{m_2}} \begin{pmatrix} I_m \\ \vdots \\ I_m \end{pmatrix},$$

where Φ is a cPNN.

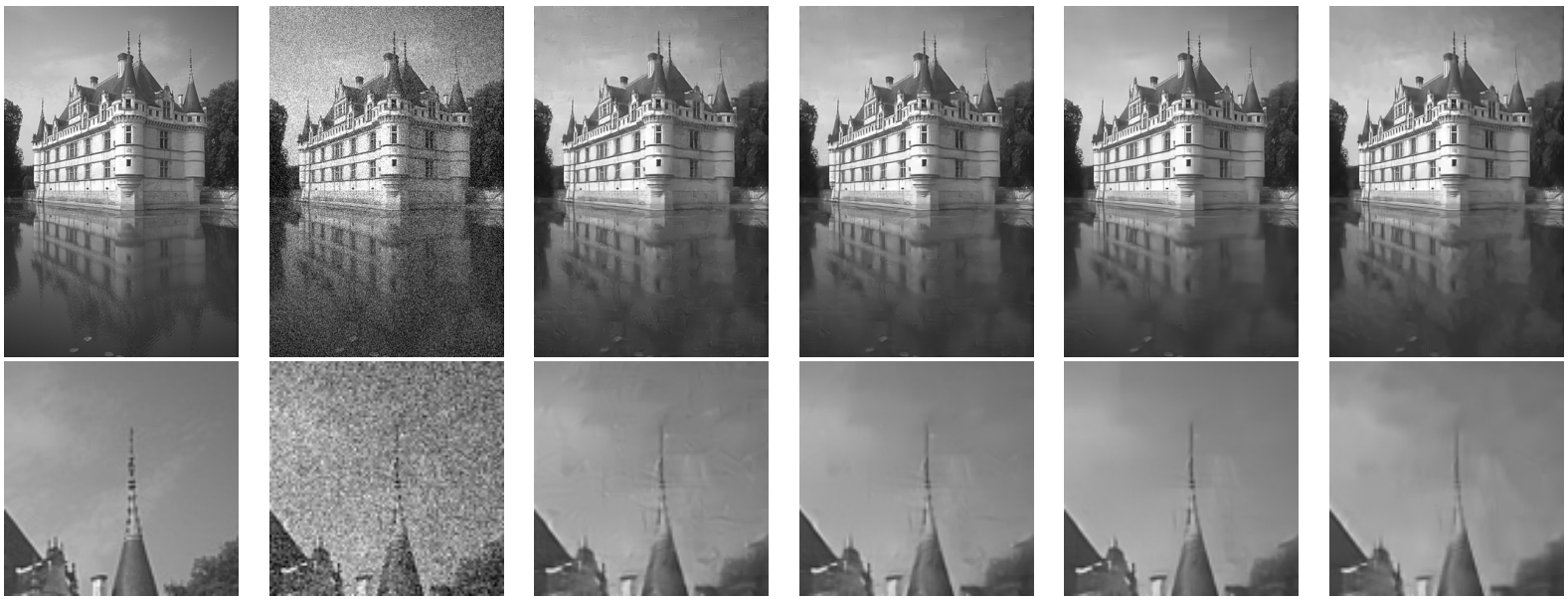
Parameters: 8 layers, $m_1 = 64$, $m_2 = 128$.

[2] Ref.: Zhang et al., 2016

Numerical Results

We train the networks on the 400 training images of the BSDS500 data set and test it on the BSD68 data set. For the corrupted images, we add Gaussian noise of noise level $25/255 \approx 0.098$.

Method	BM3D	$\gamma = 1$	$\gamma = 1.99$	$\gamma = 5$	$\gamma = 10$	unconst. CNN	DnCNN[3]
PSNR	28.59	28.48	28.81	29.02	29.08	29.11	29.23



Original | Noisy PSNR 20.17 | cPNN, $\gamma = 1.99$ PSNR 29.68 | cPNN, $\gamma = 5$ PSNR 29.93 | unconst. CNN PSNR 29.86 | BM3D PSNR 29.56

[3] Ref.: Zhang et al., 2016

Contents

Plug-and-Play[4]

Algorithm 1 FBS and FBS-PnP

Initialization: $y^{(0)} \in \mathbb{R}^m$, $\eta \in (0, \frac{2}{L})$

Iterations: For $r = 0, 1, \ldots$

$$y^{(r+1)} = x^{(r)} - \eta \nabla f(x^{(r)})$$
$$x^{(r+1)} = \operatorname{prox}_{\eta g}(y^{(r+1)})$$

PnP Step: $x^{(r+1)} = \mathcal{D}(y^{(r+1)})$

- Replace the proximity operator wrt the regularizer by a more general denoiser.
- The same can be done with other algorithms from convex analysis, e.g. ADMM.

Lemma

Let $f\colon \mathbb{R}^m \to \mathbb{R}$ be convex and differentiable with L-Lipschitz gradient and let $\mathcal{D}\colon \mathbb{R}^m \to \mathbb{R}^m$ be averaged. Then for $0 < \eta < \frac{2}{L}$, FBS-PnP converges.

[4] Venkatakrishnan et al., 2013

Averaged Denoiser with cPNNs

Recall that we used for denoising the mapping $I_m - \gamma A^{\mathrm{T}} \Phi(A\cdot; u)$.
$\rightarrow$ $\Psi = A^{\mathrm{T}} \Phi(A\cdot; u)$ is averaged.

Lemma (H., Neumayer, Steidl, 2021)

Let $x^* \in \mathbb{R}^m$ be fixed. Further, let $\Psi : \mathbb{R}^m \to \mathbb{R}^m$ be an t-averaged operator with $t \in [\frac{1}{2}, 1]$. For a scaling factor $0 < \gamma < 2$, the mapping

$$\mathcal{D}(x) = \left(1 - \tfrac{1}{1-\gamma+2t\gamma}\right) x^* + \tfrac{1}{1-\gamma+2t\gamma}\left(I_m - \gamma \Psi(x)\right)$$

is $\tilde{t}$-averaged with $\tilde{t} = \frac{t\gamma}{1-\gamma+2t\gamma}$.

- For $t = \frac{1}{2}$, it we have in the lemma $\mathcal{D}(x) = I_m - \gamma\Psi(x)$.
- If $t > \frac{1}{2}$, we need some oracle.

Numerical Avaraging Parameter

It is important that the avaraging parameter t of $\Psi = A^{\mathrm{T}}\Phi(A\cdot; u)$ is as close to $\frac{1}{2}$ as possible.

- By theory we obtain that $t = \frac{K}{K+1}$.
- In practice, we observe numerically that t is often smaller than it can be shown by theory.

Approximate $t^* := \min\{t \in [\frac{1}{2}, 1] : \Psi \text{ is } t\text{-averaged}\}$ numerically as follows:

- Start with $t = \frac{1}{2}$.
- Check if $R := \frac{1}{t}\Psi - \frac{1-t}{t} I$ is 1-Lipschitz.
- If yes: set $t^* = t$, otherwise: increase t by 0.05 and repeat this procedure.

A necessary criterion for R being 1-Lipschitz:

- generate x_i, $i = 1, \ldots, N$ uniformly on $[0, 1]^m$.
- Check if $\|JR(x_i)\|_2 \leq 1$ (can be implemented matrix-free via the power-method).

Numerical Results: PnP-Denoising

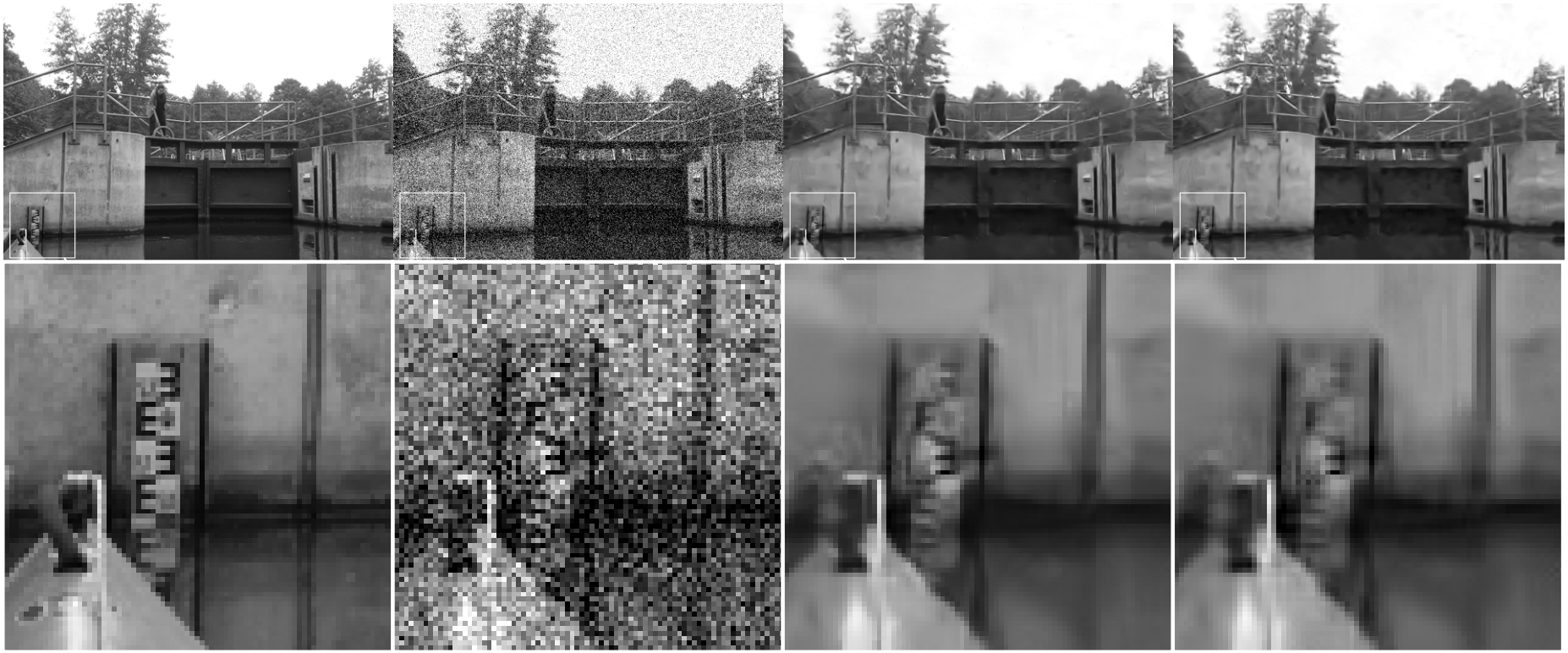
- cPNNs are trained on BSD500 for noise level of $25/255 \approx 0.098$.
- The numerical averaging parameter is $t = 0.6$.
- The test images are corrupted by Gaussian noise with standard deviation σ.
- Use BM3D as oracle.

Method	$\sigma = 0.075$,	$\sigma = 0.1$,	$\sigma = 0.125$,	$\sigma = 0.15$
Noisy images	22.50	20.00	18.06	16.48
FBS-PnP with cPNN	30.12	28.80	27.82	27.06
Variational network[5]	30.05	28.72	27.72	26.95
BM3D[6]	29.88	28.50	27.50	26.73

[5] Ref.: Effland, Kobler, Kunisch, Pock. Variational networks: an optimal control approach to early stopping variational methods for image restoration, 2020.

[6] Ref.: Dabov, Foi, Katkovnik, Egiazarian. Image denoising by sparse 3D transform-domain collaborative filtering, 2007.

Numerical Results: PnP-Denoising



Original	Noisy PSNR 16.44	FBS-PnP with cPNN PSNR 27.00	BM3D PSNR 26.62

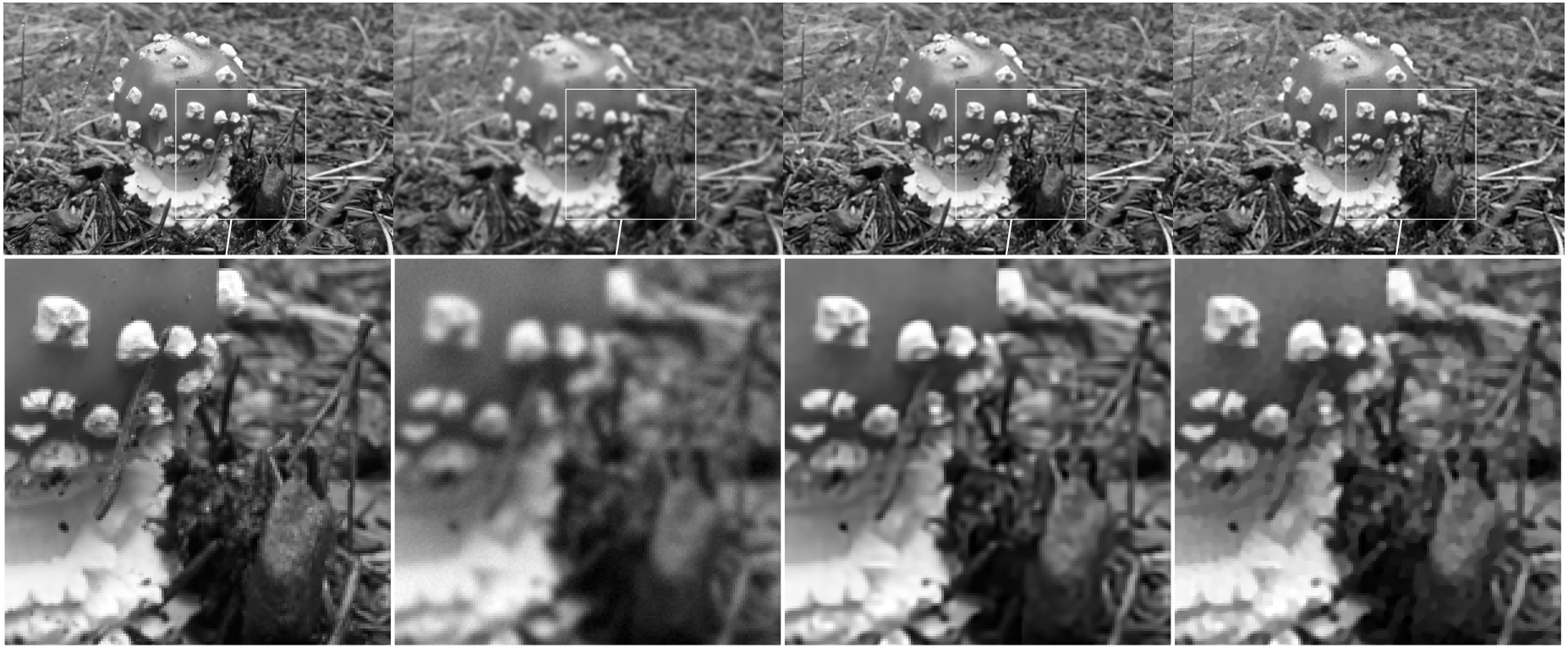
Numerical Results: PnP-Deblurring

- cPNNs are trained on BSD500. The test set is BSD68.
- The test images are corrupted by a Gaussian blur kernel of width τ and Gaussian noise with standard deviation 0.01.
- The cPNN-denoiser is trained for the noise level of 0.005.
- The numerical averaging parameter is $t = 0.5$. No oracle is needed.

Method	$\tau = 1.25$,	$\tau = 1.5$,	$\tau = 1.75$,	$\tau = 2.0$
Blurred images	26.46	25.60	24.98	24.53
FBS-PnP with cPNN	29.78	28.62	27.70	26.98
Variational network[7]	29.95	28.76	27.87	27.13
L_2-TV, $\lambda = 0.001$	29.14	28.08	27.22	26.53

[7] Ref.: Effland, Kobler, Kunisch, Pock. Variational networks: an optimal control approach to early stopping variational methods for image restoration, 2020.

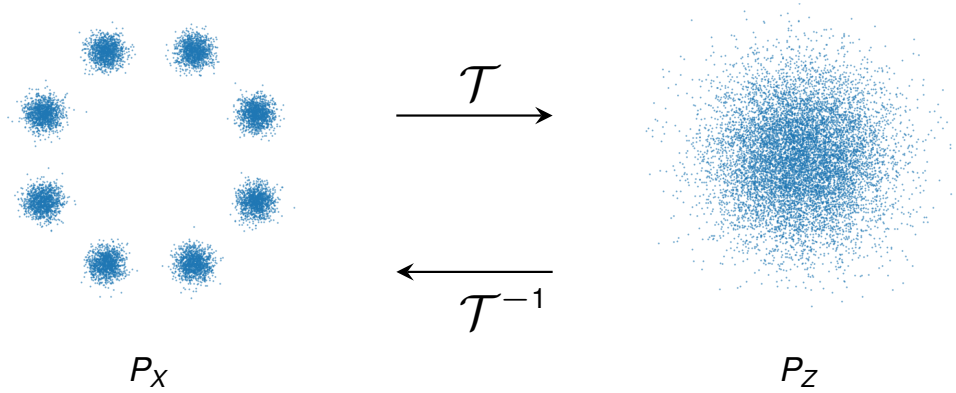
Numerical Results: PnP-Deblurring



Original	Blurred PSNR 25.21	FBS-PnP with cPNN PSNR 30.51	L_2-TV PSNR 29.77

Contents

Normalizing Flows



- Goal: Approximate a complicated probability distribution P_X, which is given by samples $x_1, \ldots, x_N$.
- Learn diffeomorphism $\mathcal{T}$ such that $\mathcal{T}_{\#}P_X \approx P_Z$, i.e., $P_X \approx \mathcal{T}_{\#}^{-1}P_Z$.
- Sampling by drawing a sample z from P_Z and computing $\mathcal{T}^{-1}(z)$.
- Density evaluation by the change-of-variables formula

$$p_{\mathcal{T}_{\#}^{-1}P_Z}(x) = p_Z(\mathcal{T}(x))|\det(\nabla\mathcal{T}(x))|.$$

- Maximum likelihood loss: Maximize

$$\mathcal{J}(\theta) = \sum_{i=1}^{N} \log(p_{\mathcal{T}_{\#}^{-1}P_Z}(x_i)) = \sum_{i=1}^{N} \log(p_Z(\mathcal{T}(x_i))) + \log(\det|\nabla\mathcal{T}(x_i)|).$$

Residual Flows

- Proposed by Behrmann et al., 2019.
- Concatenate layers of the form $L(x) = x + g(x)$, where g is a c-Lipschitz neural network with $c < 1$.

Theorem

Let g be c-Lipschitz continuous for some $c < 1$. Then L is invertible and the inverse $L^{-1}(y)$ is given by the limit of the sequence

$$x^{(r+1)} = y - g(x^{(r)}).$$

- Lipschitz constraints imposed by spectral normalization.

Can be improved by the use PNNs!

Ref.: Behrmann et. al. Invertible Residual Networks, Chen et. al. Residual Flows for Invertible Generative Modeling

Proximal Residual Flows

Use layers of the form $L(x) = x + \gamma\Phi(x)$, where Φ is a PNN.

Theorem (H., 2022)

Let Φ be a t-averaged operator with $\frac{1}{2} < t \leq 1$ and let $0 < \gamma < \frac{1}{2t-1}$. Then, L is invertible and the inverse of $L^{-1}(y)$ is given by the limit of the sequence

$$x^{(r+1)} = \frac{1}{1+\gamma-\gamma t} y - \frac{\gamma t}{1+\gamma-\gamma t} R(x^{(r)}), \quad \text{where} \quad R(x) = \frac{1}{t}\Phi(x) - \frac{1-t}{t}x.$$

Additionally, if Φ is t-averaged with $0 \leq t \leq \frac{1}{2}$, the above statement is true for arbitrary $\gamma > 0$.

- $\gamma\Phi$ might have a Lipschitz constant larger than 1.
  $\rightarrow \gamma < \frac{K+1}{K-1}$ for a K-layer PNN.
- Averaged constraint imposed directly by the definition of the PNN.

Toy Examples

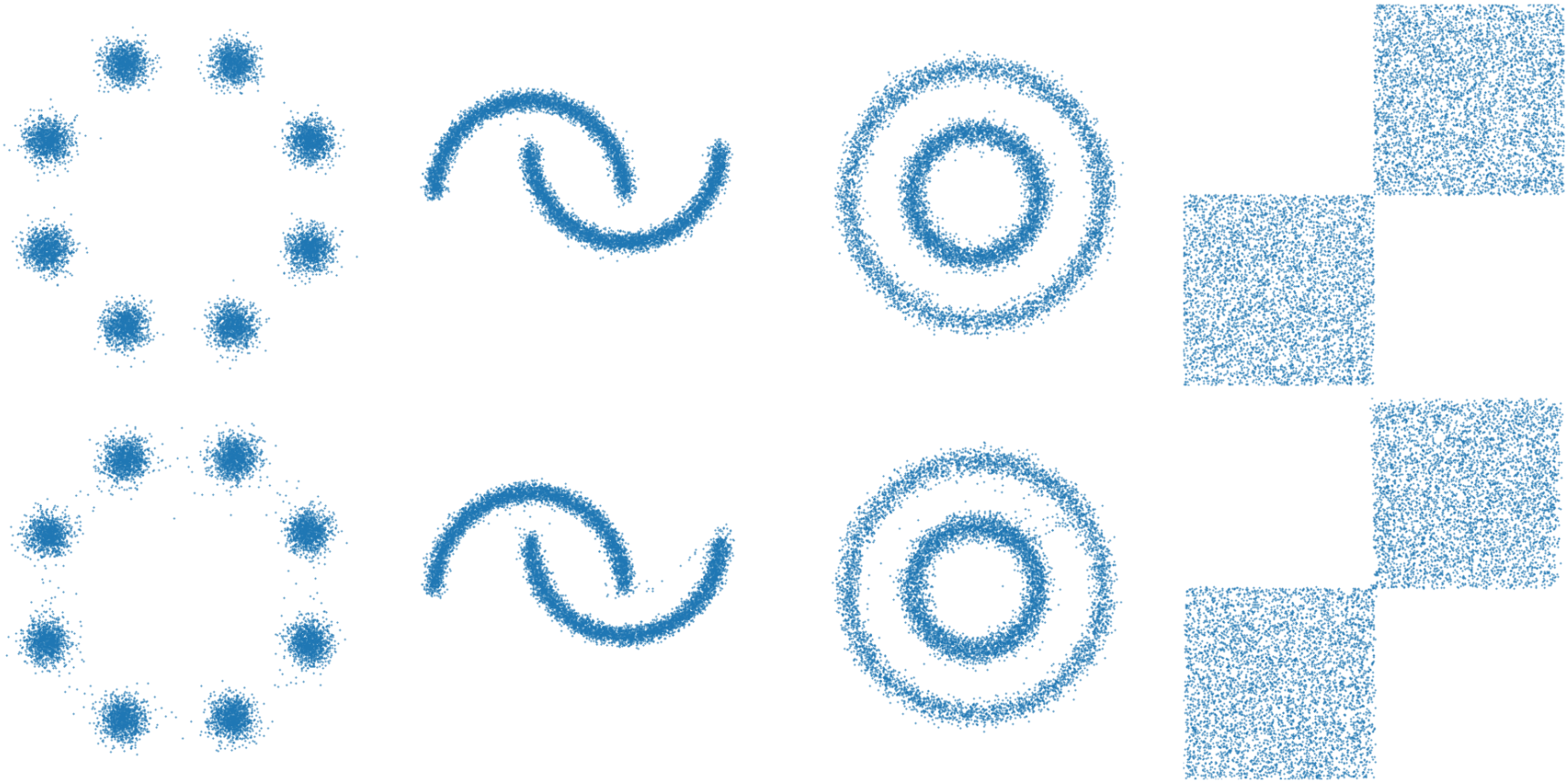


Figure: Reconstruction of toy densities. Top: Ground truth, Bottom: Reconstructions with proximal residual flows.

Conditional Proximal Residual Flows

Consider the Bayesian inverse problem

$$Y = F(X) + \Xi, \quad X \sim P_X, \quad \Xi \sim \mathcal{N}(0, \sigma^2).$$

Goal: $P_{X|Y=y} \approx \mathcal{T}(y;\cdot)_{\#} P_Z$ for all y.

$\rightarrow$ $\mathcal{T}(y;\cdot)$ has to be invertible for any y.

Conditional Proximal Residual Flows

Use layers of the form:

$$L(y;x) = x + \gamma \Phi_2(y,x), \quad \text{where } \Phi = (\Phi_1, \Phi_2) \colon \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^n \times \mathbb{R}^d \text{ is a PNN.}$$

$\rightarrow$ If Φ is averaged, then also Φ_2 is averaged.

Maximize:

$$\mathcal{J}(\theta) = \sum_{i=1}^{N} \log(p_{\mathcal{T}^{-1}(y_i;\cdot)_{\#} P_Z}(x_i)) = \sum_{i=1}^{N} \log(p_Z(\mathcal{T}(y_i;x_i))) + \log(|\nabla \mathcal{T}(y_i;x_i)|).$$

Approximation of the Posterior for Gaussian Mixtures

Consider the Bayesian inverse problem

$$Y = AX + \Xi, \quad A \in \mathbb{R}^{d \times d}, \quad \Xi \sim N(0, b^2 I)$$

where

- The prior distribution P_X is a GMM in $\mathbb{R}^{50}$ with 5 mixture components.
- A diagonal matrix with diagonal entries $0.1/k^2$ for $k = 1, \ldots, 50$.
- $\Xi \sim \mathcal{N}(0, 0.05I)$.
- Posterior can be derived analytically.

Approximation error (sample-based Wasserstein distance):

Method	Real NVP	Residual Flow	Proximal Residual Flow
Error	2.122 ± 1.007	1.374 ± 0.060	1.028 ± 0.079

Conclusions

- We constructed (convolutional) PNNs, which are averaged (C)NNs.
- cPNNs can be used for constructing denoisers with upper bound on the Lipschitz constant.
  $\rightarrow$ Competitive performance.
- cPNNs can be used for convergent Plug-and-Play iterations.
- PNNs can be used for proximal residual flows, a new architecture of normalizing flows.

Future work:

- Which class of functions can be approximated by PNNs?
- Efficient projection onto the orthogonal convolutions with limited filter length.
- Overcome the numerical averaging parameter.

Technische Universität Berlin

Thank you for your attention!

M. Hasannasab, J. Hertrich, S.Neumayer, G. Plonka, S. Setzer and G. Steidl (2020).
Parseval Proximal Neural Networks.
Journal of Fourier Analysis and Applications, vol. 26, no. 59.
Code available at `https://github.com/johertrich/Proximal_Neural_Networks`.

J. Hertrich, S. Neumayer and G. Steidl (2021).
Convolutional Proximal Neural Networks and Plug-and-Play Algorithms.
Linear Algebra and its Applications, vol 631, pp. 203-234.
Code available at `https://github.com/johertrich/Proximal_Neural_Networks`.

J. Hertrich and G. Steidl (2022).
Inertial Stochastic PALM and Applications in Machine Learning.
Sampling Theory, Signal Processing, and Data Analysis, vol. 20, no. 4.
Code available at `https://github.com/johertrich/Inertial-Stochastic-PALM`.

J. Hertrich (2022).
Proximal Residual Flows for Bayesian Inverse Problems.
ArXiv Preprint 2211.17158.
Code available at `https://github.com/johertrich/Proximal_Residual_Flows`.