Image denoising with deep learning

Course Maths and Data sciences

Master mathématiques appliquées, statistique - 2ème année

Parcours Data science.

Frédéric Richard

AMU, 2025

Outline

Image Denoising Problem

- Goal: recover a clean image x from a noisy observation y.
- For instance, recover x when corrupted by an additive noise n:

$$y = x + n.$$

- Bayesian viewpoint:
  - **Degradation model** $p(x|y)$. For instance, if $n \simeq \mathcal{N}(0, \sigma^2 \mathrm{Id})$ and is independent of x, then

$$p(x|y) \propto \exp\left(-\frac{1}{2\sigma^2}|x - y|^2\right).$$

  - **Prior distribution** $p(x)$ of clean images. For instance, assuming some regularity of image, we can use a Tikhonov regularisation:

$$p(x) \propto \exp(-|Dx|^2)$$

    on derivatives Dx of x.
  - Using Bayes'rule, we obtain the **posterior distribution**:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}.$$

Denoising

- The denoiser can be found as the Maximum A Posteriori (MAP) estimator:

$$\begin{aligned}\hat{x} &= \arg\max_x p(x|y) = \arg\max_x p(y|x)p(x) \\ &= \arg\min_x \left(-\log(p(y|x)) - \log p(x)\right).\end{aligned}$$

- Alternately, the denoiser can be searched by minimizing the mean-square error (MSE)

$$\mathbb{E}(|y-x|^2),$$

leading to the Mimimum MSE (MMSE):

$$\tilde{x} = \mathbb{E}(x|y) = \int x p(x|y) dx.$$

- Both, MAP and MMSE are reliant on the posterior distribution.

Evolution of research

- A key feature is the modeling of $p(x)$, which has driven the literature of image denoising.
- Classical era (around 1970-2010).
    - Design of $p(x)$, mostly in the form of a Gibbs distribution
      $$p(x) \propto \exp(-\rho(x)),$$
      where ρ is an energy function.
    - Design and study appropriate algorithms to find the MAP depending on the chosen distribution.
- AI revolution (starting around 2012 to nowadays).
- Another approach:
    - Gather a large dataset of clean images $(x_k)_k$ and create noisy image (y_k) from these image,
    - Design a neural network taking y as input and giving an estimate of x as output.
    - Train the neural network by minimising a loss defining the distance between the clean image x and its estimate $\hat{x}$.

Further reading: A survey by M. Elad et al.

Architecture of a sequential neural network

- Series of U layers : for $u = 1, \cdots, U$,

$$f^{(u)} = L^{(u)}(f^{(u-1)}; w^{(u)}),$$

  with
  - $L^{(u)}$: the layer at level u.
  - $f^{(u-1)}$: input signal at level u,
  - $f^{(u)}$: output signal at level u,
  - $w^{(u)}$: parameters at level u.
- Classical components of a layers:
  - Dense layer $g^{(u)} = W^{(u)} f^{(u)}$: linear link between input and output signals.
  - Convolutional layer $g^{(u)} = w^{(u)} * f^{(u)}$: spatially-invariant linear transform.
  - Activation function (*tanh*, sigmoïd, ReLU, etc.): non-linear representation.
  - Max or Average pooling: local summary of spatial information.
  - Down- (resp. up-) sampling: reduction (resp. increase) the signal size.

Early CNNs: LeNet (1990s)

One of the first CNNs, applied to handwritten digit recognition.

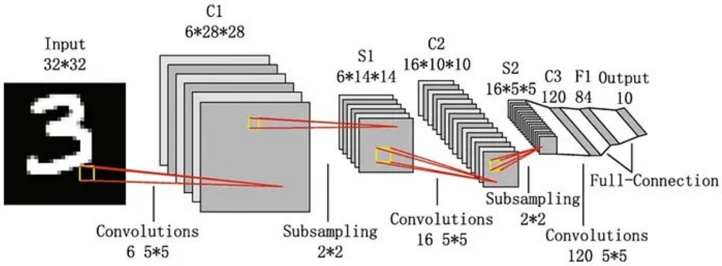


Figure: Image from Medium paper of Abishek Jain.

Typical block of a convolutional layer :

- Convolutions with several kernels: $g^{(u)} = S^{(u)}(f^{(u-1)}; w^{(u)})$
- Activation with an non-linear function φ: $h^{(u)} = \varphi(g^{(u)})$
- Pooling (with down-sampling): $f^{(u)} = P^{(u)}(h^{(u)})$.

Learning a neural net.

- Learning set: $(x_i, y_i)_{i=1}^n$.
- Predict y_i from x_i with the output of the last layer $\hat{y}_i(w)$.
- $\hat{y}_i(w)$ is defined by recursion:
  $f_i^{(0)} = x_i$ and for all $u = 1, \cdots, U$, $f_i^{(u)} = L^{(u)}(f_i^{(u-1)}, w^{(u)})$.
- $w = (w^{(u)}, u = 1, \cdots, U)$ is the set of model parameters.
- Minimisation of the empirical risk

$$E_n(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \hat{y}_i(w)),$$

defined for some loss function ℓ (mean square error, binary-cross entropy, etc.)

Optimization

- Minimization by stochastic gradient algorithm and variants.
- involve computing gradients of the loss function.
- Back-propagation: recursive expression of gradients throught layers:
- Typical example:

$$E_n(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_i^{(U)}) \text{ with } \left\{ \begin{array}{lcl} g_i^{(u)} & = & S^{(u)}(f_i^{(u-1)}; w^{(u)}), \\ f_i^{(u)} & = & \varphi^{(u)}(g_i^{(u)}). \end{array} \right.$$

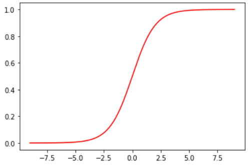
$$\frac{dE_n}{dw^{(v)}}(w) = \frac{1}{n}\sum_{i=1}^{n} \frac{d\ell}{dz}\left(y_i, \frac{df_i^{(L)}}{dw^{(v)}}\right)$$

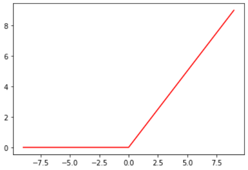
$$\frac{df_i^{(u)}}{dw^{(v)}} = d\varphi^{(u)}(g_i^{(u)}) \; \frac{dg_i^{(u)}}{dw^{(v)}}.$$

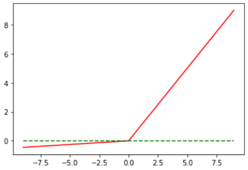
Vanishing gradient and activation functions

$$u = 1, \cdots, U, \quad \left|\frac{df_i^{(u)}}{dw^{(v)}}\right| \leq \left|d\varphi^{(u)}(g_i^{(u)})\right| \quad \left|\frac{dg_i^{(u)}}{dw^{(v)}}\right|.$$

- Differential norm is not of the same order throught the layers,
- Activation functions $\varphi^{(u)}$ may saturate and make the differential vanish ($|d\varphi^{(u)}(g_i^{(u)})| \simeq 0$).
- Consequence: slow convergence on last layers.


Logistic
$\varphi(t) = \frac{1}{1+e^{-t}}$


ReLU
$\varphi(t) = \max(0, t)$


leaky ReLU(α)
$\varphi(t) = \max(\alpha t, t)$

Internal covariate shift and batch normalization

- Issue: statistics of layer output vary drastically from a layer to another, resulting in unstabilities.
- Solution (Ioffe et Szegedy, 2015) : centering and normalizing the output with batch normalization:
- Example:
  - Output of the layer u :

$$g_i^{(u)} = S^{(u)}(f_i^{(u-1)}; w^{(u)}).$$

  - Estimation of the mean and variance of $g^{(u)}$ from a batch B:

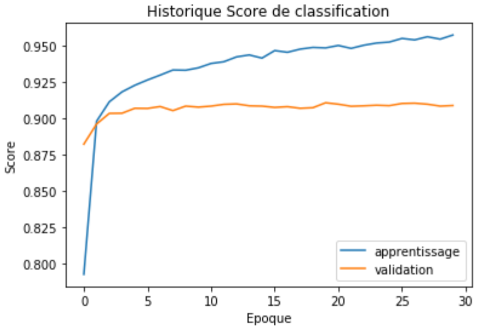
$$\hat{\mu}_B^{(u)} = \frac{1}{|B|} \sum_{i \in B} g_i^{(u)} \quad \text{et} \quad \hat{v}_B^{(u)} = \frac{1}{|B|} \sum_{i \in B} (g_i^{(u)} - \hat{\mu}_B^{(u)})^2.$$

  - Centering and normalizing the output.

$$\tilde{g}_i^{(u)} = \frac{1}{\sqrt{\hat{v}_B^{(u)}) + \varepsilon}} (g_i^{(u)} - \hat{\mu}_B^{(u)}).$$

Overfitting

- Issue: Although the model performs well on the learning set, it gives poor results on test data (not seen during learning).
- Factors:
  - Model is too complex.
  - Learning set is too small or not enough representative of all data.
- Early diagnostic during the training step with a validation dataset.



Some solutions to over-fitting.

- Penalization of the optimisation problem, implicitely reducing model complexity.
- **Dropout** (Hinton, 2012 et Srivastava et al. 2014):
  - Cancel randomly and temporarily the evolution of weights during the learning,
  - Attenuate specialization of weights on learning data.
- **Data augmentation**: increase the learning set using data transforms (translation, rotation, contrast changes, etc.).
- Fine tuning: use an already learned neural network and modify only a few of its parameters to adapt it to another context.u classifieur.

From LeNet to AlexNet

- **LeNet**: one of the first CNNs, applied to handwritten digit recognition.
- Limitations:
  - Vanishing gradients in deep networks.
  - Sigmoid / tanh activations $\Rightarrow$ slow convergence, computational cost.
  - Hardware at the time could not support large CNNs.
- AlexNet: Winner of ImageNet Large Scale Visual Recognition Challenge (ILSVRC, 2012).
- Key innovations:
  - Large set of data for training **ImageNet**.
  - **GPU** acceleration for training (Cuda programming, Nvidia).
  - **ReLU** activations instead of sigmoid and tanh $\Rightarrow$ faster training.
  - Local response normalization $\Rightarrow$ stability.
  - **Dropout** layers to attenuate over-fitting

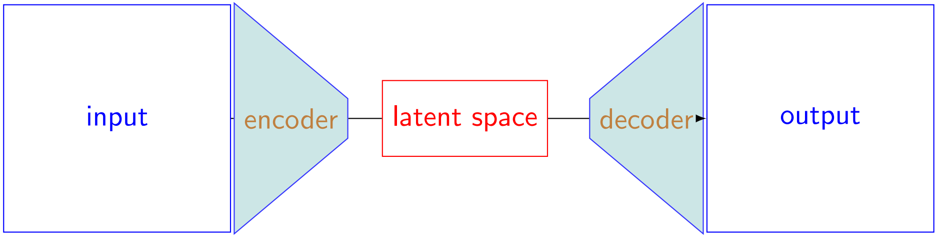
CNN of ILSVRC

- **AlexNet** - milestone in deep learning.
  - Limitation: very large kernels $\Rightarrow$ high memory usage.
- **VGG** (Winner of ILSVRC 2014)
  - Used many **stacked small** 3×3 **filters** $\Rightarrow$ parameter reduction.
  - Deeper architecture improved accuracy.
  - Drawback: very deep $\Rightarrow$ risk of vanishing gradients.
- **GoogLeNet** (2014)
  - **Inception module**.
  - Replaced large filters with multiple smaller ones to reduce parameters.
  - Increased network **width** instead of only depth.
  - Drawback: wide networks may suffer from overfitting.
- **ResNet** (Winner of ILSVRC 2015).
  - Introduced **residual connections**: $y = F(x) + x$.
  - Allowed training of very deep networks without vanishing gradients.
  - Foundation for many modern architectures.

Further reading: Medium paper on AlexNet, VGGNet, ResNet, and Inception.

Autoencoder Basics

- Encoder compresses the input image into a low-dimensional **latent representation**.
- Decoder reconstructs the image from this latent representation.
- For denoising: train the autoencoder to reconstruct the clean image from a noisy input.



Convolutional auto-encoder for image denoising:

- **Encoder:** convolution + pooling + downsampling layers.
- **Latent space:** compressed representation of images.
- **Decoder:** upsampling + transposed convolution layers.
- **Output:** denoised image $\hat{y}$.

Auto-encoder for image denoising

- Training Objective:
  - Dataset: pairs of noisy images x_i and clean images y_i.
  - Loss function: Mean Squared Error (MSE)

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \| f_\theta(y_i) - x_i \|^2$$

  - Alternative: Mean Absolute Error (L1), SSIM-based loss, perceptual loss.
- Advantages:
  - Simple architecture.
  - Forms the basis for more advanced denoising models (DnCNN, U-Net, VAEs, diffusion).
- Difficulty to recover the image details.

U-Net Architecture

- Originally proposed by Ronneberger et al. (2017) for biomedical segmentation.
- Encoder–decoder architecture with **skip connections**.
- Captures both global context and fine local details.

- **Contracting part:**
  convolution + downsampling (captures context).
- **Bottleneck:**
  deepest layer with high-level features.
- **Expansive part:**
  upsampling + convolution (restores resolution).
- **Skip connections:**
  transfer fine details.

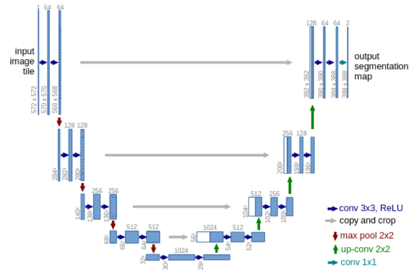
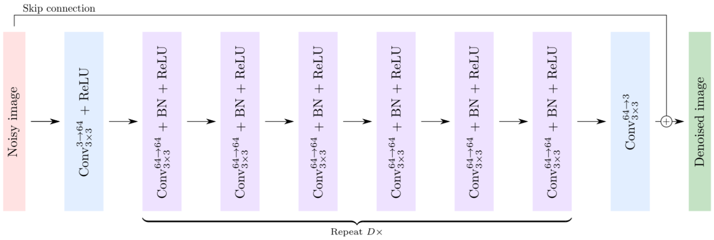


Figure: source: (Ronneberger et al., 2015)

DnCNN Overview

- Overview:
  - Proposed by Zhang et al. (2017).
  - Deep CNN specifically designed for image denoising.
  - Uses **residual learning**: instead of directly predicting x, the network predicts the noise $\hat{n}$.
  - Output: $\hat{x} = y - \hat{n}$.
- Architecture:
  - Input: noisy image $x \in \mathbb{R}^{H \times W \times C}$.
  - First layer: Conv + ReLU.
  - Middle layers: several Conv + BatchNorm + ReLU blocks.
  - Last layer: Conv (predicts residual noise).


Skip connection
Noisy image
$\mathrm{Conv}_{3\times3}^{3\to64}$ + ReLU
$\mathrm{Conv}_{3\times3}^{64\to64}$ + BN + ReLU
$\mathrm{Conv}_{3\times3}^{64\to64}$ + BN + ReLU
$\mathrm{Conv}_{3\times3}^{64\to64}$ + BN + ReLU
$\mathrm{Conv}_{3\times3}^{64\to64}$ + BN + ReLU
$\mathrm{Conv}_{3\times3}^{64\to64}$ + BN + ReLU
$\mathrm{Conv}_{3\times3}^{64\to64}$ + BN + ReLU
$\mathrm{Conv}_{3\times3}^{64\to3}$
Denoised image
Repeat $D\times$

Training Objective

- Training objective
  - Training data: pairs (x, y) of noisy/clean images.
  - Loss function: Mean Squared Error (MSE) between predicted noise $\hat{n}$ and true noise n:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \|\hat{n}_i - (y_i - x_i)\|^2$$

  - Equivalent to a skip connection with a classical MSE.
- Advantages:
  - Residual learning simplifies optimization.
  - Batch Normalization stabilizes training.
  - Deeper CNN captures spatial context for stronger denoising.
  - Can generalize to different noise levels (blind denoising).