Generative Models

"Image Classification: Modern Approaches"

Giacomo Boracchi, Alessandro Giusti
DEIB, Politecnico di Milano

February, 23rd , 2018

giacomo.boracchi@polimi.it
home.deib.polimi.it/boracchi/

Generative Adversarial Networks

A very effective generative model for images

Generative Models

Goal:

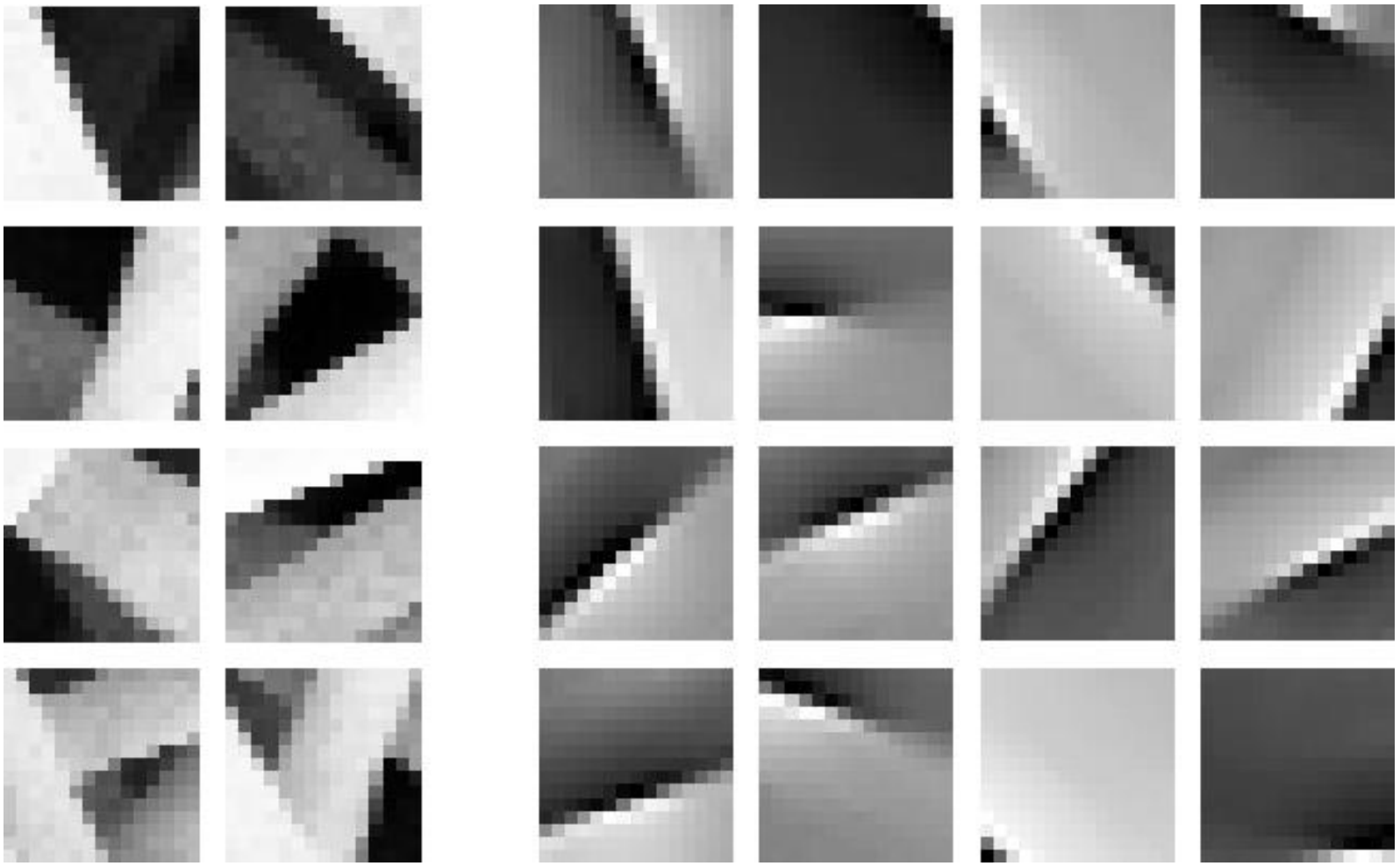
generative models generate, given a training set of images (data) S, other images (data) that are similar to those in S

Remark:

Synthesis models like dictionaries yielding sparse rerpresentations are not able to generate realistic data

$$\boldsymbol{s} = \sum_i x_i \, \boldsymbol{d_i}$$

Example of Dictionary Learned from Nanofibers

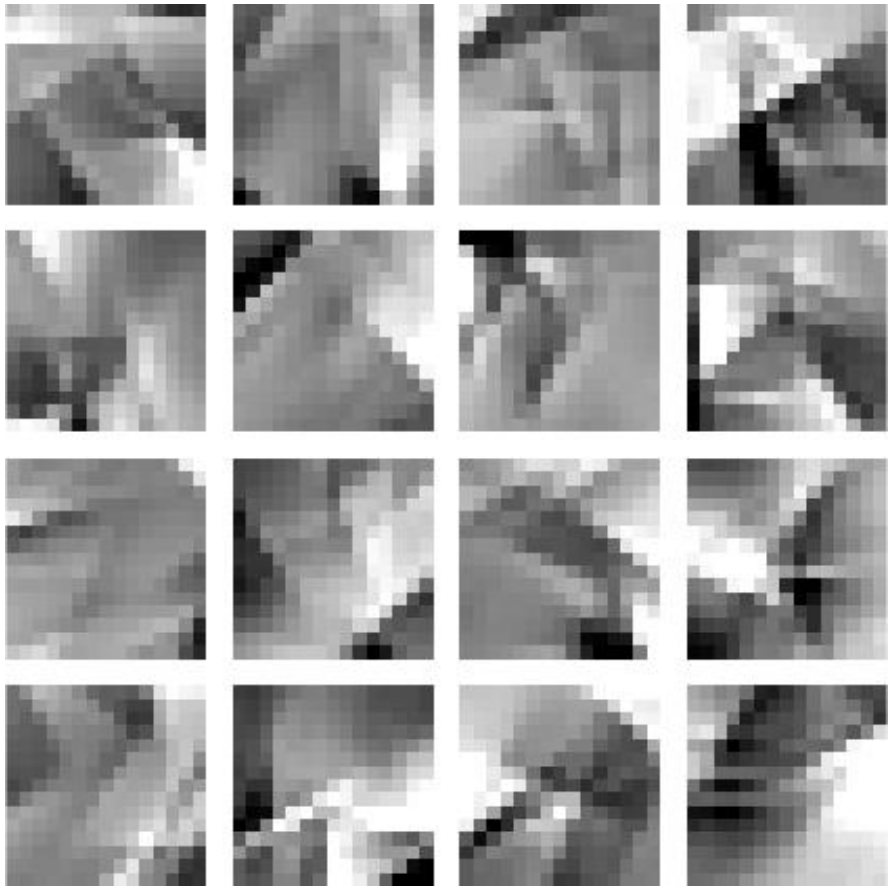


Patches generated using sparse representations

We randomly generate $\boldsymbol{x}$ as

$$x_i = \begin{cases} 0 & with\ probability\ p \\ N(0, \sigma^2) & with\ propability\ 1-p \end{cases}$$

Examples of $\boldsymbol{s} = D\boldsymbol{x}$, where D is the dictionary learned from the nanofiber patches



Difficult to find $\phi_{\boldsymbol{x}}$ such that $\boldsymbol{x} \sim \phi_{\boldsymbol{x}}$ makes $D\boldsymbol{x}$ a realistic image patch.

Convolutional Sparsity

Convolutional sparse models are a recent development of sparse representations

$$\boldsymbol{s} \approx \sum_{i=1}^{n} \boldsymbol{d_i} \circledast \boldsymbol{\alpha_i}\,, \qquad \text{s.t. } \boldsymbol{\alpha_i} \text{ is sparse}$$

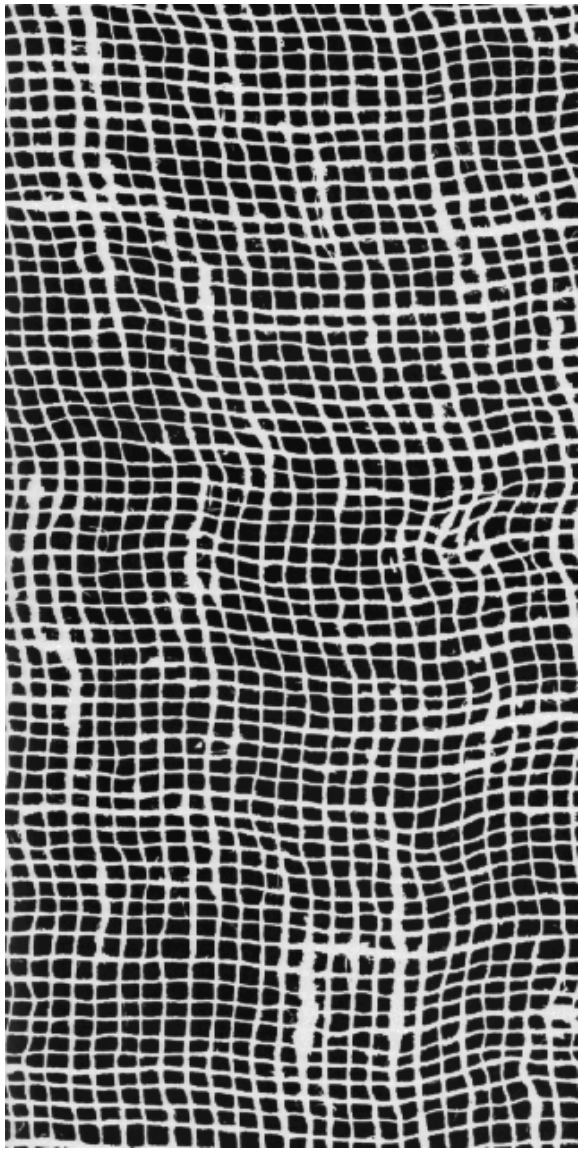
where a signal $\boldsymbol{s}$ is **entirely encoded** as the sum of n convolutions between a filter $\boldsymbol{d_i}$ and a coefficient map $\boldsymbol{\alpha_i}$

Pros:

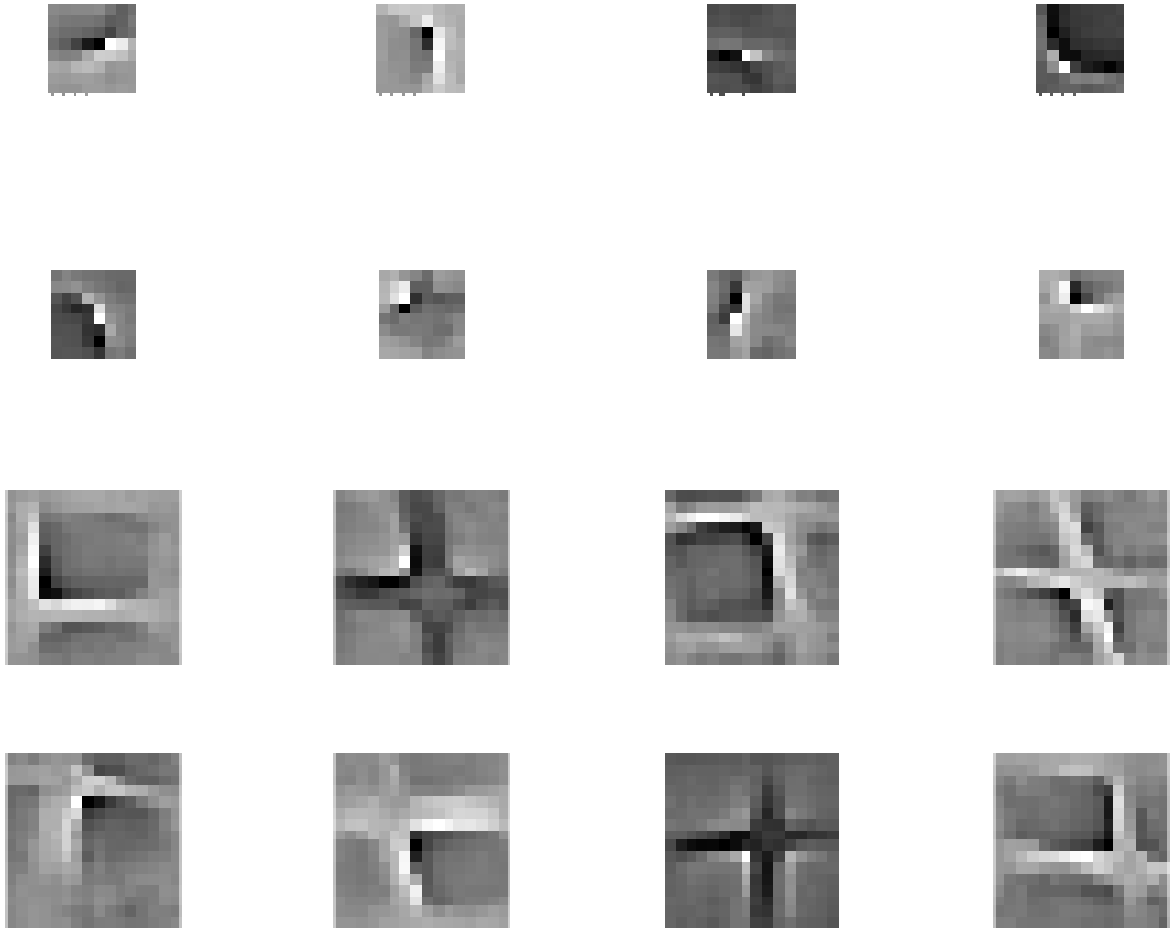
- Translation invariant representation
- Few small filters are typically required
- Filters exhibit very specific image structures
- Easy to use filters having different size

Example of Learned Filters

Training Image



Learned Filters



Convolutional Sparsity

Convolutional sparse models are a recent development of sparse representations

$$\boldsymbol{s} \approx \sum_{i=1}^{n} \boldsymbol{d_i} \circledast \boldsymbol{\alpha_i}\,, \qquad \text{s.t. } \boldsymbol{\alpha_i} \text{ is sparse}$$

where a signal $\boldsymbol{s}$ is **entirely encoded** as the sum of n convolutions between a filter $\boldsymbol{d_i}$ and a coefficient map $\boldsymbol{\alpha_i}$

These however do not yield a generative model as the same issue seen for patch-based sparsity apply

Generative Adversarial Networks (GAN)

Models able to generate samples that looks like training samples $S \subset \mathbb{R}^n$

The core idea: Train a pair of models with different tasks that compete in a sort of game.

These models are:

- Generator $\mathcal{G}$ that produces realistic samples e.g. taking as input some random noise
- Discriminator $\mathcal{D}$ that takes as input an image and assess whether it is real or generated by $\mathcal{G}$

Train the two and at the end, keep only $\mathcal{G}$

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, ... & Bengio, Y. Generative adversarial nets NIPS 2014

GAN

Both $\mathcal{D}$ and $\mathcal{G}$ are conveniently chosen as MLP

Setting up the stage

Our networks are

- $\mathcal{D} = \mathcal{D}(\boldsymbol{s}, \theta_d)$
- $\mathcal{G} = \mathcal{G}(\boldsymbol{z}, \theta_g)$

θ_g and θ_d are network parameters, $\boldsymbol{s} \in \mathbb{R}^n$ is an input image and $\boldsymbol{z} \in \mathbb{R}^d$ is some random noise to be fed to the generator.

$$\mathcal{D}(\cdot, \theta_d): \mathbb{R}^n \to [0,1]$$

indicates the posteriori for an input to be a true image

$$\mathcal{G}(\cdot, \theta_d): \mathbb{R}^d \to \mathbb{R}^n$$

Is the generated image

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, ... & Bengio, Y. Generative adversarial nets NIPS 2014

GAN Training

A good discriminator is such:

- $\mathcal{D}(\boldsymbol{s}, \theta_d)$ is maximum when $s \in S$
- $1 - \mathcal{D}(\boldsymbol{s}, \theta_d)$ is maximum when s was generated from $\mathcal{G}$
- $1 - \mathcal{D}\big(\mathcal{G}\big(\boldsymbol{z}, \theta_g\big), \theta_d\big)$ is maximum when $\boldsymbol{z} \sim \phi_Z$

Training $\mathcal{D}$ consists in maximizing the binary cross-entroy

$$\max_{\theta_d}\big(\mathrm{E}_{s\sim\phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)] + \mathrm{E}_{z\sim\phi_Z}\big[\log(1 - \mathcal{D}\big(\mathcal{G}\big(\boldsymbol{z}, \theta_g\big), \theta_d\big))\big]\big)$$

A good generator $\mathcal{G}$ is the one which makes $\mathcal{D}$ to fail

$$\min_{\theta_g}\max_{\theta_d}\big(\mathrm{E}_{s\sim\phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)] + \mathrm{E}_{z\sim\phi_Z}\big[\log(1 - \mathcal{D}\big(\mathcal{G}\big(\boldsymbol{z}, \theta_g\big), \theta_d\big))\big]\big)$$

GAN Training

Solve by an iterative numerical approach

$$\min_{\theta_g} \max_{\theta_d} \left(\mathrm{E}_{s\sim\phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)] + \mathrm{E}_{z\sim\phi_Z}\left[\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d))\right]\right)$$

Alternate:

k-steps of Stochastic Gradient Ascent w.r.t. θ_d to solve

$$\max_{\theta_d} \left(\mathrm{E}_{s\sim\phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)] + \mathrm{E}_{z\sim\phi_Z}\left[\log\left(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d)\right)\right]\right)$$

1-step of Stochastic Grandient Descent w.r.t. θ_g to solve

$$\min_{\theta_g} \left(\cancel{\mathrm{E}_{s\sim\phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)]} + \mathrm{E}_{z\sim\phi_Z}\left[\log\left(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d)\right)\right]\right)$$

i.e.,

$$\min_{\theta_g} \left(\mathrm{E}_{z\sim\phi_Z}\left[\log\left(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d)\right)\right]\right)$$

GAN Training

for $i = 1 \ldots$

for k −times

- Draw a minibatch $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m\}$ of noise realization
- Sample a minibatch of images $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m\}$
- Update $\mathcal{D}$ by stochastic gradient ascend:

$$\nabla_{\theta_d} \left[\sum_i \log \mathcal{D}(\boldsymbol{s_i}, \theta_d) + \log \left(1 - \mathcal{D}\left(\mathcal{G}\left(\boldsymbol{z}_i, \theta_g\right), \theta_d\right)\right)\right]$$

Draw a minibatch $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m\}$ of noise realization

Update $\mathcal{G}$ by stochastic gradient descent:

$$\nabla_{\theta_g} \left[\sum_i \log \left(1 - \mathcal{D}\left(\mathcal{G}\left(\boldsymbol{z}_i, \theta_g\right), \theta_d\right)\right)\right]$$

At the end of the day…

The discriminator $\mathcal{D}$ is discarded

The generator $\mathcal{G}$ and ϕ_Z are preserved as generative model

Remarks:

- There is no explicit expression for the generator, it is provided in an implicit form -> you cannot compute the likelihood of a sample w.r.t. the learned GAN
- The training is rather instable, need to carefully synchronize the two steps (many later works in this direction, e.g. Wasserstein GAN)
- Training by standard tools: backpropagation and dropout
- Theoretical results provided
- Generator does not use S directly during training
- Performance is difficult to assess quantitatively