Deep Avancé : Generative Models

Part II: texture synthesis and style transfert with deep neural-network based methods

julien.Rabin (at) ensicaen.fr

Version : 7 décembre 2023

In previous episodes ...

Pros & Cons of standard image editing techniques

- **Fast** and **simple**, requires **only one example image**
- **Restrictions** (e.g. stationary hypothesis, scaling issues with NN) which generally **prevent from using large datasets**
- **Does not generalise** well (for synthesis either copy real data or gives unrealistic smooth or blurry images)

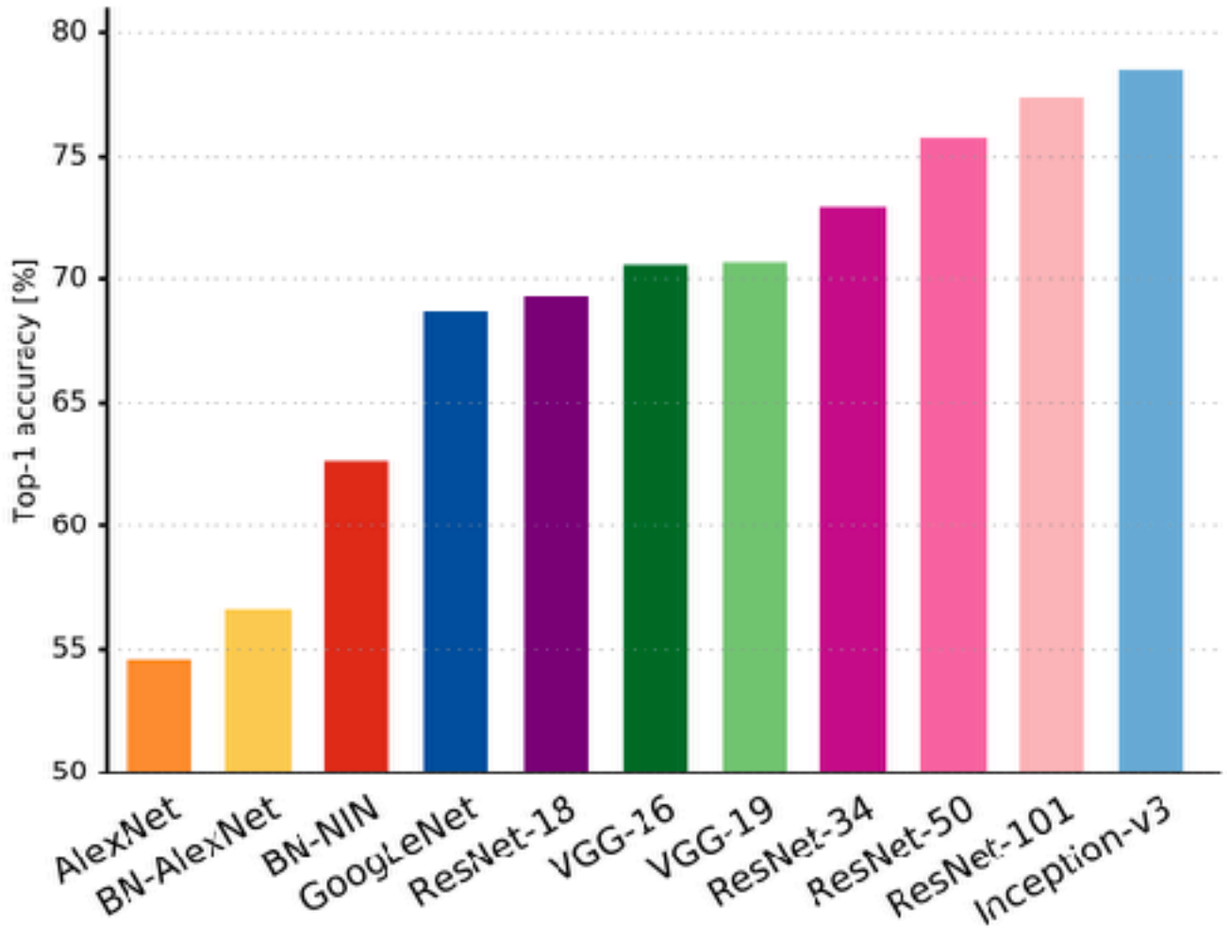
Machine Learning and **Convolutional Neural Network**

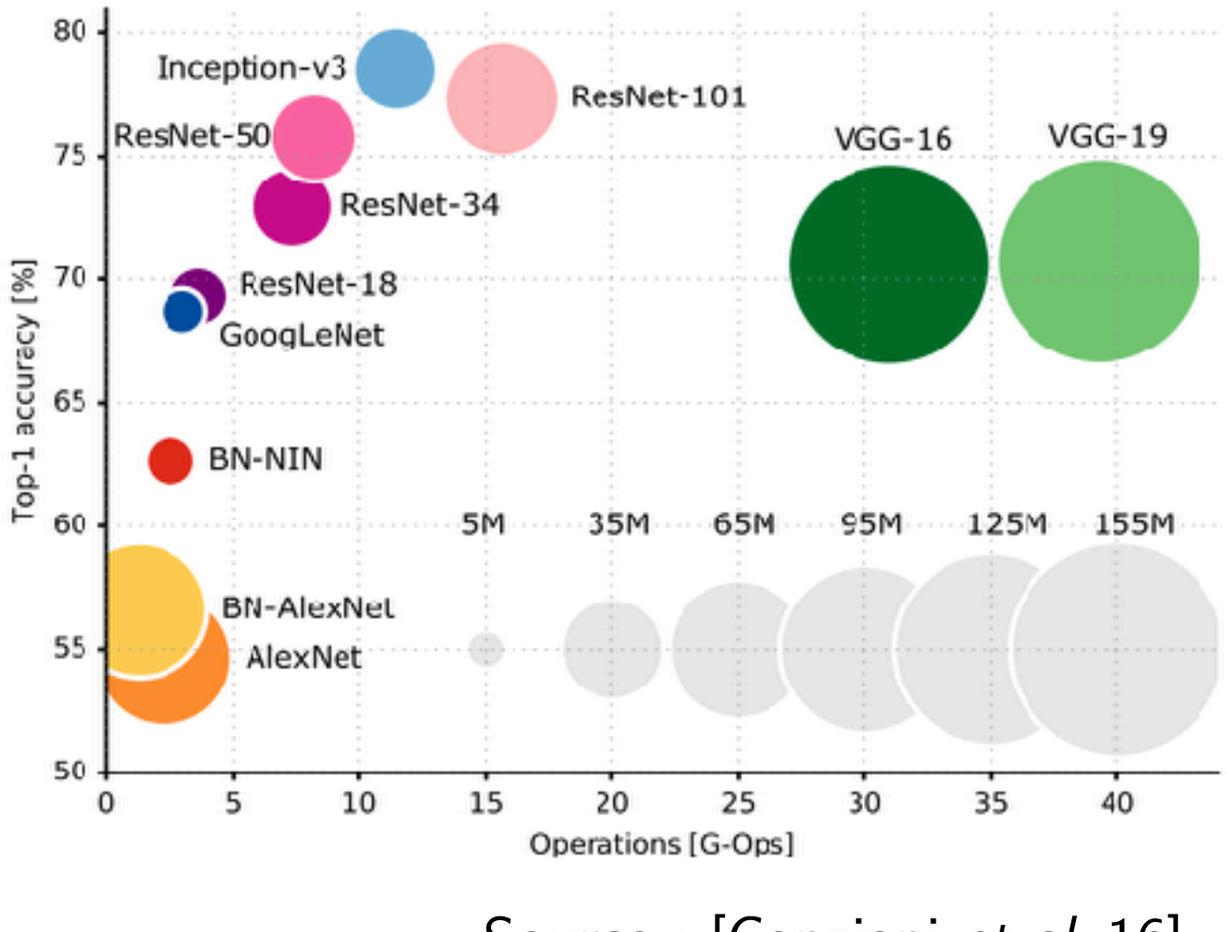
- **Classification:** AlexNet [Krizhevsky *et al.*'12], VGG-19 [Simonyan and Zisserman'14], ResNet [He at al.'15], Inception Network [Szegedy *et al.*'16], ...

- In this course, application to other computer vision problems:
  - **Filtering** (denoising, artifact removing), super-resolution, inpainting: deep image prior [Ulyanov *et al.*'17]
  - **Texture synthesis** and **style transfer:** [Gatys *et al.*'15]
  - **Image generation** with deep generative models

Deep classifiers

- Quick overview of several deep architectures in classification





Source : [Canziani *et al.*,16]

Deep CNNs rely too much on patterns !

- **A telling experiment from [Geirhos'18]**

(a) Texture image

81.4%	**Indian elephant**
10.3%	indri
8.2%	black swan

(b) Content image

71.1%	**tabby cat**
17.3%	grey fox
3.3%	Siamese cat

(c) Texture-shape cue conflict

63.9%	**Indian elephant**
26.4%	indri
9.6%	black swan

Deep CNNs rely too much on patterns !

- **Early** Neural Net classification seems better to define "perceptual" descriptors that are good for pattern synthesis (see e.g. **DeepDream**)

- **A telling experiment from [Geirhos'18]**

(a) Texture image

81.4% **Indian elephant**
10.3% indri
8.2% black swan

(b) Content image

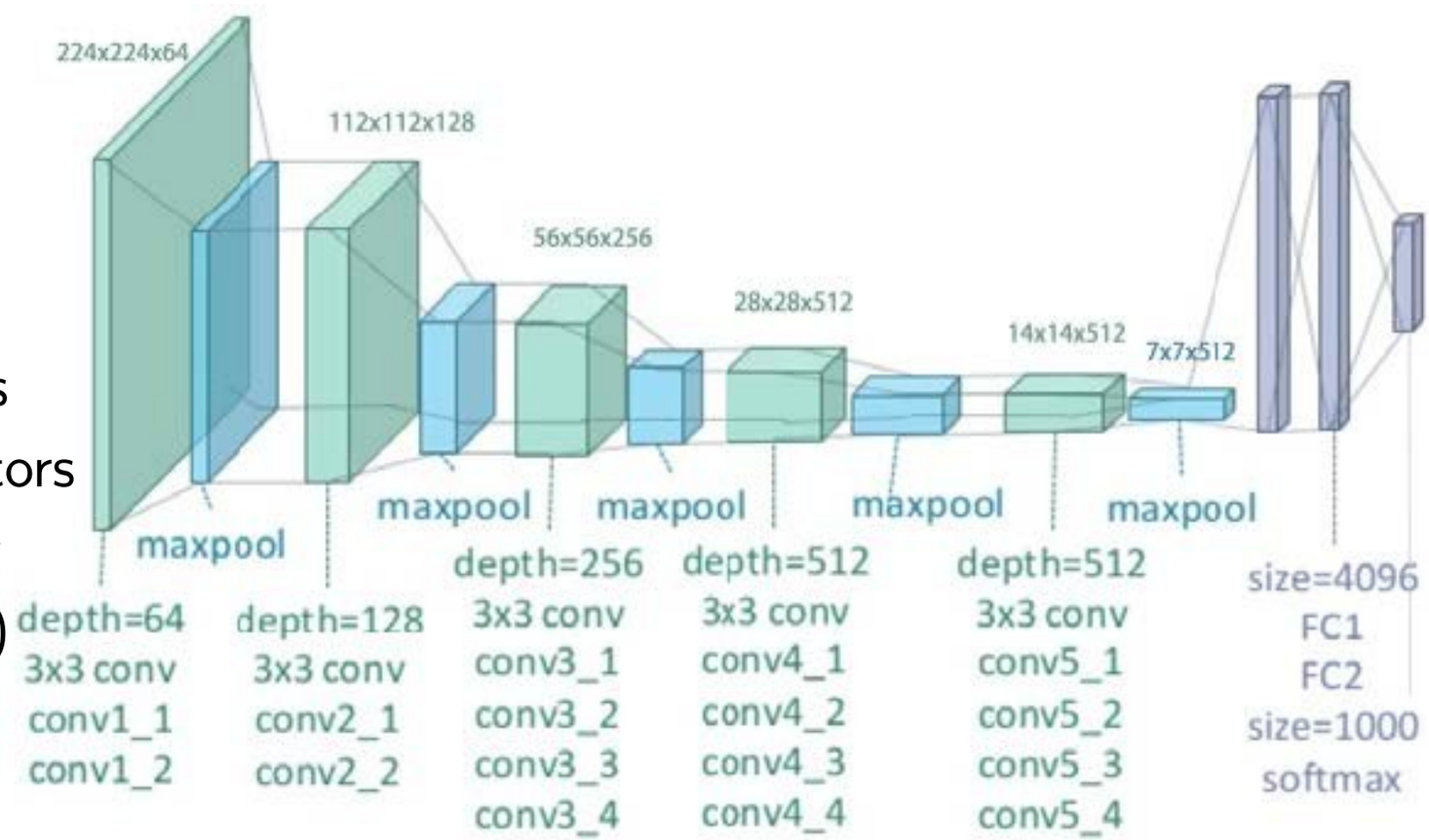
71.1% **tabby cat**
17.3% grey fox
3.3% Siamese cat

(c) Texture-shape cue conflict

63.9% **Indian elephant**
26.4% indri
9.6% black swan

Closer look at VGG-19

- **[Simonyan and Zisserman'14]** from Visual Geometry Group
- 7.5% classification error on ImageNet '14 (training with ~500k images with 200 categories)
- **19 layers** composed of
  - Convolutions (3x3),
  - ReLU,
  - Max Pooling
  - 3 final Fully-Connected layers
- Final Softmax for probability vectors
- Includes Drop-out during training
- No normalisation (batch/layer ...)
- 144 M Weights ! (~500MB)



Deep CNNs are good at learning patterns !

Remark : not true anymore for other architectures, such as ResNet

- Source: [Mordvintsev et al.'17 & 18]
- https://distill.pub/2017/feature-visualization/
- https://distill.pub/2018/differentiable-parameterizations/

Today's Menu

Optimization-based approaches **using deep features**

- **Image filtering** [Ulyanov et al.'17]: training a network from a fixed single image

- **Texture synthesis & style transfert**: [Gatys *et al.*'15]: optimizing an image pixel-wise with a fixed (pre-trained) network

Forward methods (with off-line training / optimization)

- Feed-forward Networks: Texture Network [Ulyanov *et al.*'16],

- Multi-scale Patch-based Optimal Transport [Galerne *et al.*'18]

Image Filtering with CNN

Deep Image Prior [Ulyanov et al.'18]

- **Observation**: deep convolutional generative networks synthesize regular image (even without training ...)

- **Principle**: train a generative network g_θ to generate an example image c, from a **fixed** random input n. The synthesized image is $u{=}g_\theta(n)$

$$\min_\theta \|c - g_\theta(n)\|$$

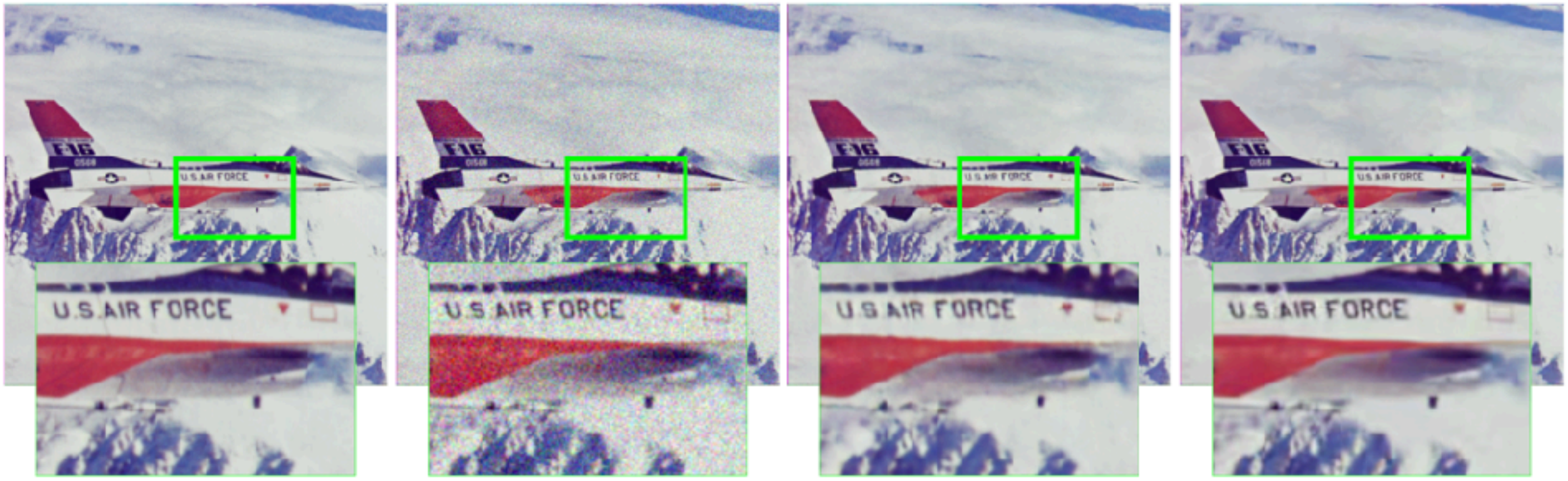
- **Optimization**: Adam (with squared norm), stop before overfitting

- Results depend *strongly* on the chosen architecture

- Better results with a batch of random samples from a law with low variance

$$\min_\theta \mathbb{E}_{n\sim\mathcal{N}} \|c - g_\theta(n)\|$$

Results for denoising



(a) GT (b) Input (c) Ours (d) CBM3D

(patch-based denoising)

Results from: https://dmitryulyanov.github.io/deep_image_prior

Pytorch Source: https://github.com/DmitryUlyanov/deep-image-prior

Results for denoising

(patch-based denoising)

Results from: https://dmitryulyanov.github.io/deep_image_prior

Pytorch Source: https://github.com/DmitryUlyanov/deep-image-prior

Demo: denoising

- **Pytorch implementation** : deep_image_prior.ipynb

Deep Prior

with MLP + Multi-scale CNN (bilinear upsample)

Patch filtering

GT

512x512

Noisy Data

Demo: denoising

- **Pytorch implementation** : deep_image_prior.ipynb (**lab session #3**)
- **Training** by minimising torch.nn.MSELoss() with noisy image v and input constant n

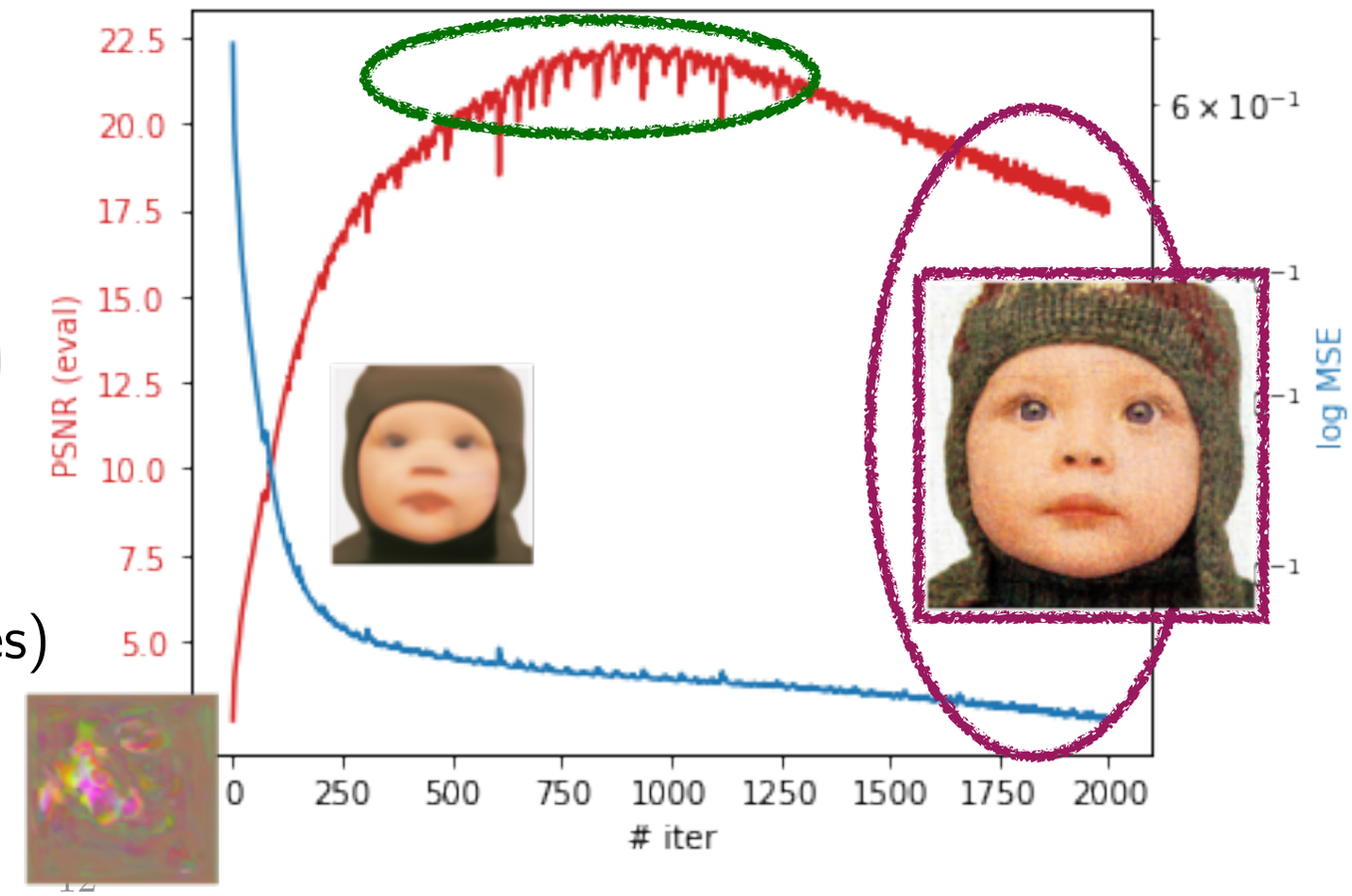
$$\min_{\theta} \|v - g_\theta(n)\|$$

- takes a few minutes on a GPU with a deep network (16 layers)
- **Visualisation :** beware of **overfitting** !
  - for **training** log MSE
  - for **evaluation** PSNR :

$$\mathrm{PSNR}(u, v) = 10 \log_{10} \left(\frac{N \ \mathrm{range}\ (v)}{\|u - v\|^2} \right)$$

with N = number of pixels

range v = max - min (255 for 8-bit images)



Demo: denoising

- what happen when changing the (random) input vector ?

Train with fixed input vector z $\min_{\theta} \|v - g_\theta(z)\|$

$z \mapsto u = g_\theta(z)$

$z' \mapsto u' = g_\theta(z')$

Results for super-resolution

Results: https://dmitryulyanov.github.io/deep_image_prior

Pytorch Source: https://github.com/DmitryUlyanov/deep-image-prior

Results for super-resolution

GT Bicubic Deep Prior with subsampling

Results: https://dmitryulyanov.github.io/deep_image_prior

Pytorch Source: https://github.com/DmitryUlyanov/deep-image-prior

Demo: SR x 4

- **Pytorch implementation** : deep_image_prior.ipynb
- **Training** by minimising torch.nn.MSELoss() **combined with x4 downsampling**

$$\min_{\theta} \|v - \phi(g_\theta(n))\|$$

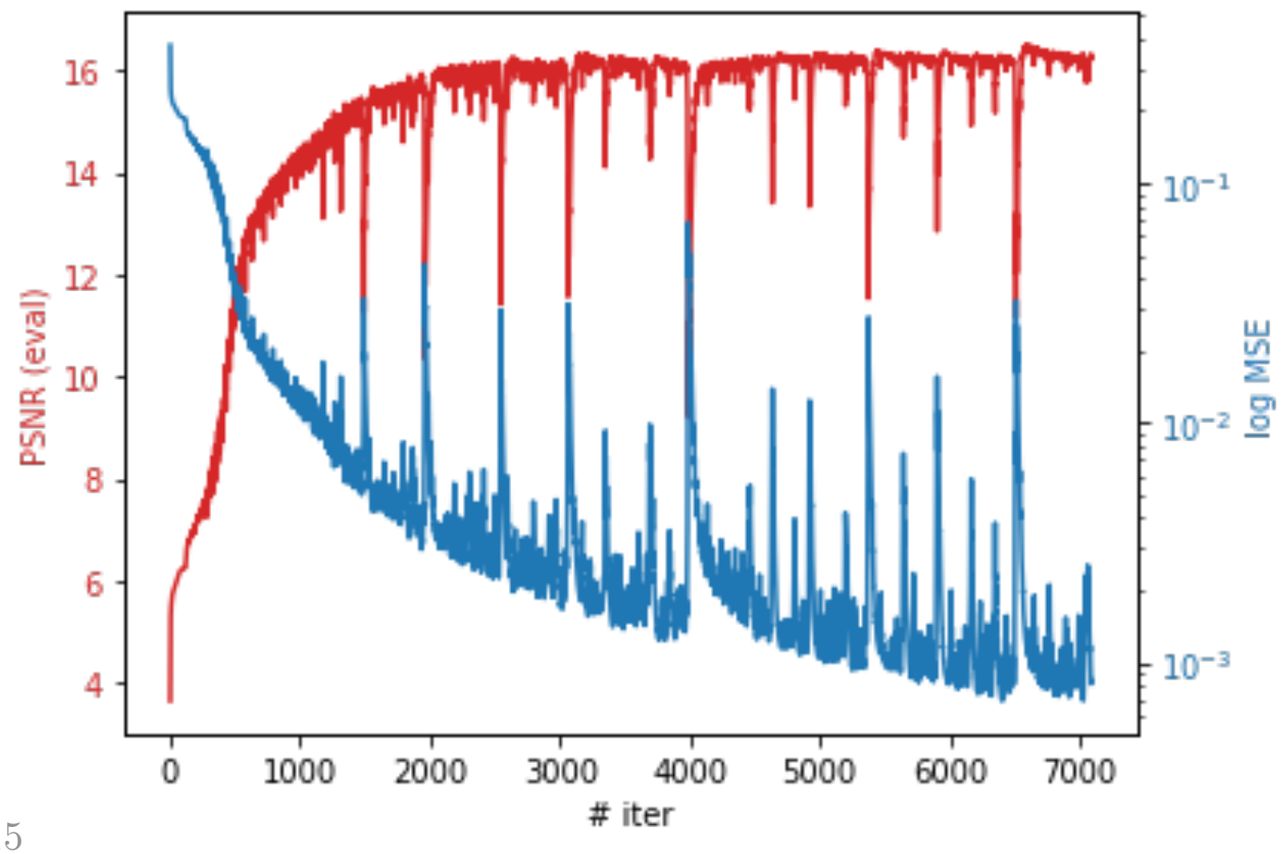
with ϕ **the degrading operator** (here downsampling for SR, e.g. AvgPool2d)

- **Visualisation :** beware of overfitting !
  - for **training** log MSE
  - for **evaluation** PSNR :

$$\mathrm{PSNR}(u, v) = 10 \log_{10} \left(\frac{N \ \mathrm{range}\ (v)}{\|u - v\|^2} \right)$$

with N = number of pixels

range v = max - min (255 for 8-bit images)



Demo: SR x 4

- **Pytorch implementation** : deep_image_prior.ipynb

Deep Prior

with MLP + Multi-scale CNN (bilinear upsample)

Bicubic interpolation

effet de bords
(ici zero-padding)

GT
512x512

Data
128x128

Demo: SR x 4

- **Pytorch implementation** : deep_image_prior.ipynb
- **Architecture tuning** : allows to visualise the prior used when designing a generative network
- **Exemple** : comparison of upsampling in the CNN architecture

bilinear

prior = smooth images

nearest

piecewise-constant

ConvTranspose

allows high-frequencies,
not filtered here due to phi operator !

Demo: SR x 4

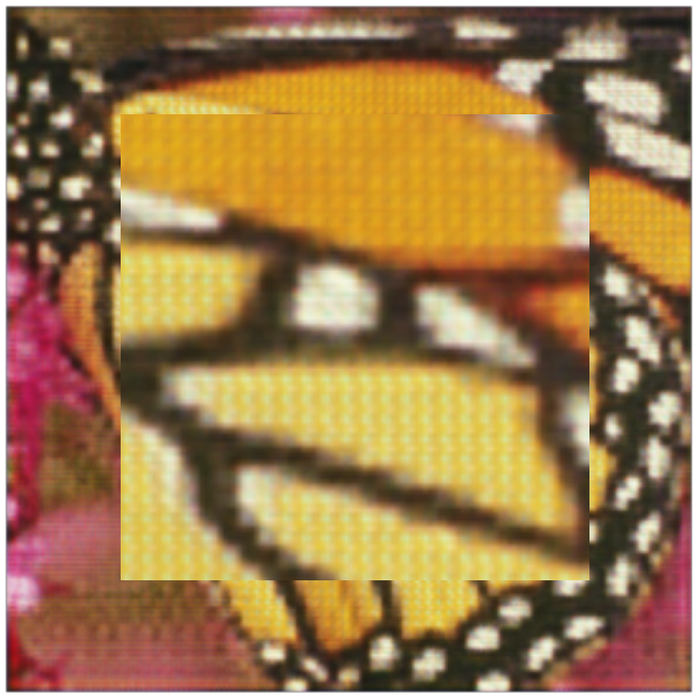
- **Pytorch implementation** : deep_image_prior.ipynb
- **Architecture tuning** : allows to visualise the prior used when designing a generative network
- **Exemple** : comparison of upsampling in the CNN architecture

bilinear

prior = smooth images

nearest

piecewise-constant



ConvTranspose

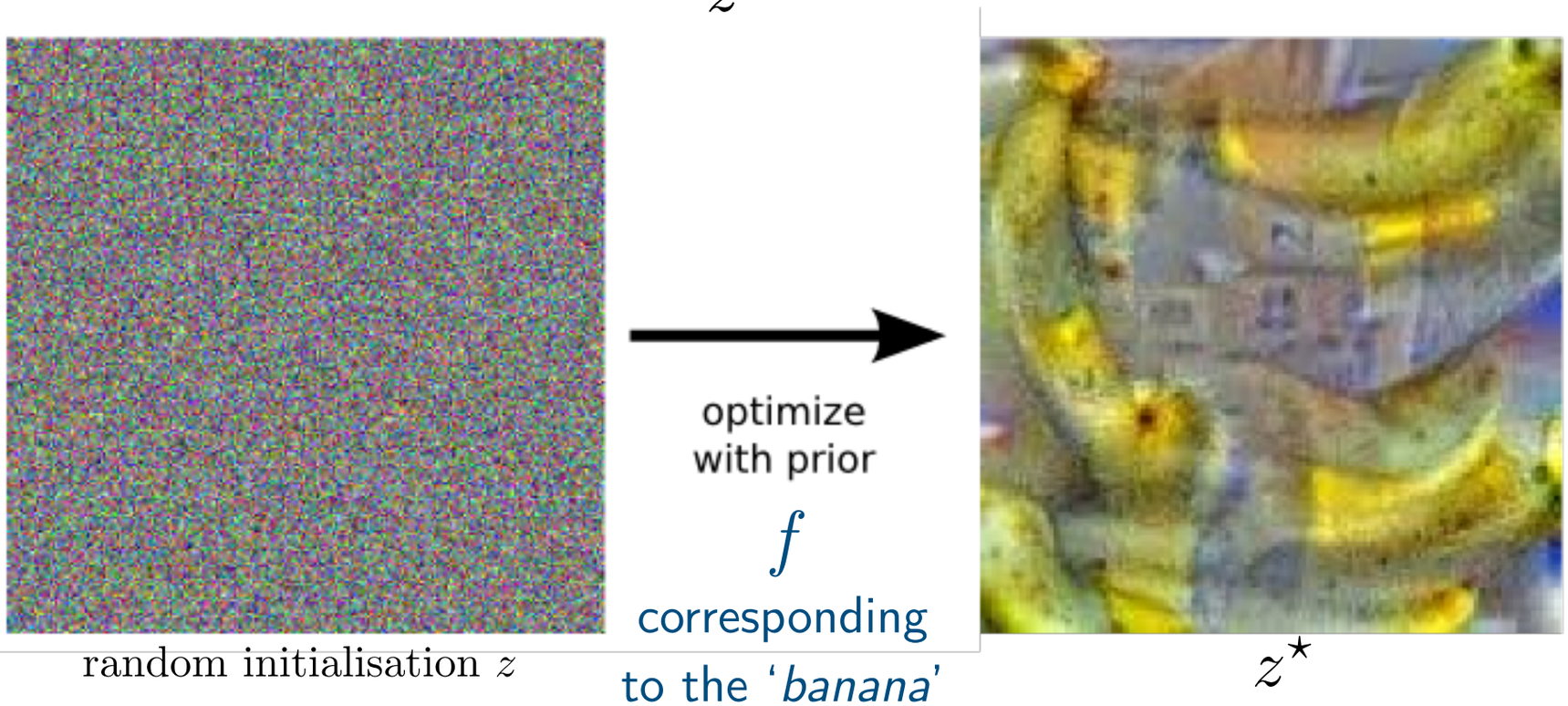
allows high-frequencies,
not filtered here due to phi operator !

Texture Synthesis and style transfer with pixel optimization

Google “Deep Dream” *

- **Principle :** optimize an input image z using some **learnt prior** ϕ_θ (here Inception Network with **fixed** parameters θ) and a given feature representation f

$$z^\star \in \arg\min_z \|f - \phi_\theta(z)\|$$



- * Source :

https://blog.research.google/2015/07/deepdream-code-example-for-visualizing.html

https://blog.research.google/2015/06/inceptionism-going-deeper-into-neural.html

Google "Deep Dream" *

- **Examples :**

$$z^\star \in \arg\min_z \|f - \phi(z)\|$$

Initialisation of z

corresponding label f

Horizon

Trees

Leaves

$z^\star$

label f

Towers & Pagodas

Buildings

Birds & Insects

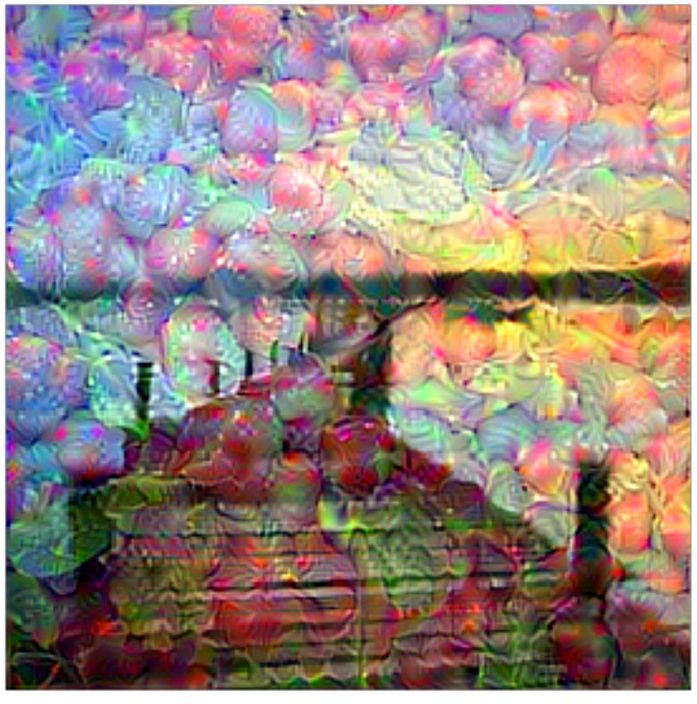
Variant with VGG prior

- **Pytorch implementation** : neural_texture.ipynb (**lab session #4**)
- **Optimisation** by minimising torch.nn.MSELoss() between **truncated VGG** features (using the **last convolutional layer**)

$$u^\star \in \arg\min_z \|\phi(c) - \phi(u)\|$$

where s is some example image (*style*), and u is initialised from another image c (*content*)

content c



"deep dream" image u

style c

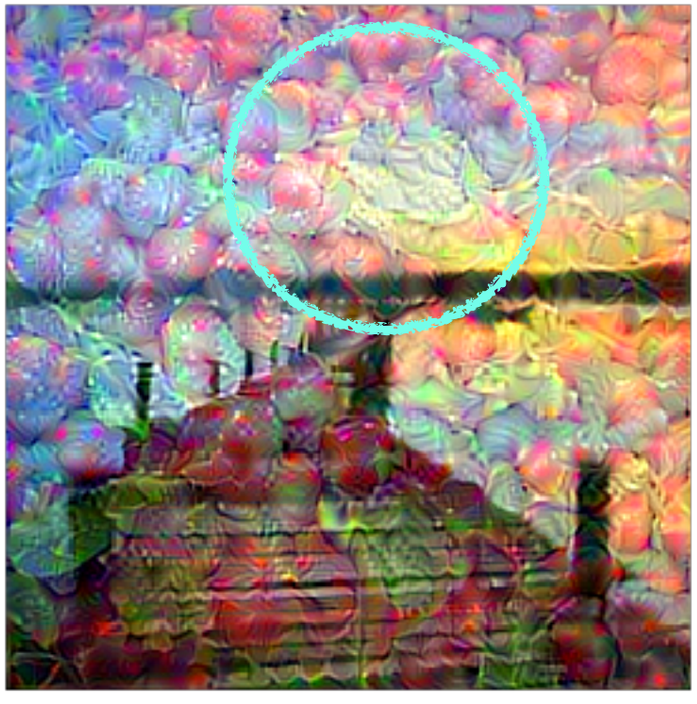
Variant with VGG prior

- **Pytorch implementation** : neural_texture.ipynb (**lab session #4**)
- **Optimisation** by minimising torch.nn.MSELoss() between **truncated VGG** features (using the **last convolutional layer**)

$$u^\star \in \arg\min_z \|\phi(c) - \phi(u)\|$$

where s is some example image (*style*), and u is initialised from another image c (*content*)

content c



"deep dream" image u

style c

- **Important remark** : **spatial information** is preserved here !

Texture optimization with deep features

- **Context:** same optimization framework than [Kwatra *et al.*'05], *i.e.* optimize the pixels of an image to match statistics from an exemplar image

- **Idea:** Instead of using a patch transform, use a **pretrained network** on a large database

- **In practice:** comparison of VGG-19 features at several layers (but other deep networks *may* work as well...) **without spatial information (stationarity)**

Texture Synthesis [Gatys *et al.*'15]

- The texture example is s and u is the resulting image

$$\arg\min_u \|\phi(u) - \phi(s)\|$$

- Features extracted from layer i (C_i channels, N spatial coordinates)

$$\phi(u) = (G_i(u))_{i \in \mathcal{I}} \quad \textbf{Gram matrices} \text{ at layer i} \quad G_i(u) \in \mathbb{R}^{C_i \times C_i}$$

$$G_i(u) = \frac{1}{NC_i} F_i^T(u) \times F_i(u) \quad \text{Feature matrice at layer i} \quad F_i(u) \in \mathbb{R}^{N \times C_i}$$

dot product = averaging of spatial features

Note the normalisation by feature dimensions : C_i

$$\|G_i\|^2 = \sum_{1 \leq i,j \leq C_i} G_i(i,j)^2 \quad \text{Frobenius norm}$$

- Layers: relu1_1, relu2_1, relu3_1, relu4_1 and relu5_1

Focus on Feature Normalization

- **Features Normalisation** : **[Gatys'16]** Gram matrices have different dimensions (number of channels C_i) and should be normalized accordingly :

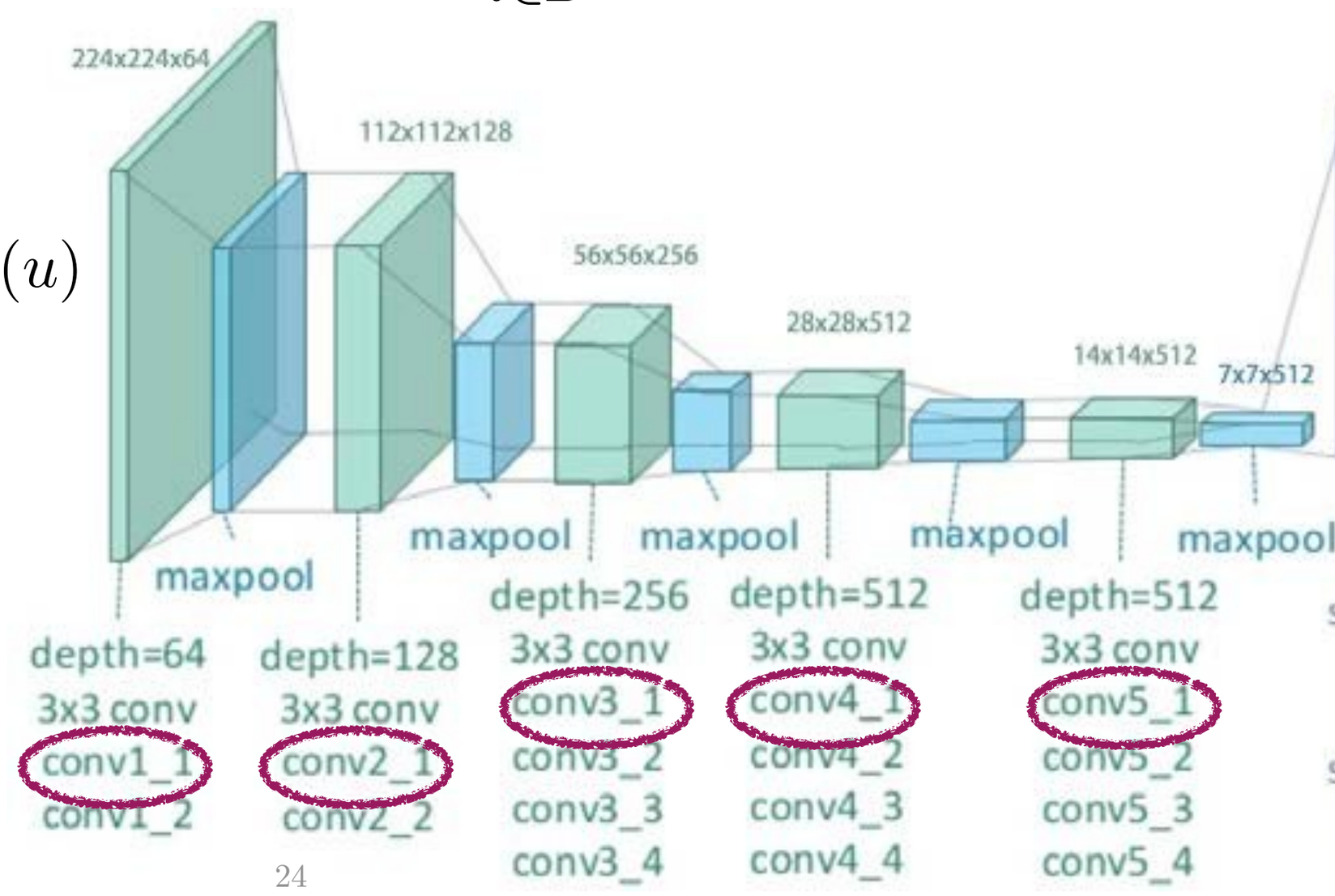
$$\mathcal{L}_{\text{style } s}(u) = \|\omega \odot \phi(u) - \phi(s)\|^2 = \sum_{i \in \mathcal{I}} \omega_i^2 \|G_i(u) - G_i(s)\|^2$$

with standard Gram matrix :

$$G_i(u) = \frac{1}{N} F_i^T(u) \times F_i(u)$$

weights in **[Gatys'16]**:

$$\omega_i = \frac{1}{C_i}$$



Texture Synthesis [Gatys *et al.*'15]

- **Initialization**: white noise sample (or previous scale if required)
- **Optimization**: L-BFGS (does not work with usual gradient descent)

- **Pros:** state of the art result, innovative (but copies can still happen with padding)

- **Cons:** optimisation take minutes, no control, non-convex optim. with a trivial solution GPU with large memory required, high resolution is difficult, small visible artefacts

Style Transfer [Gatys *et al.*'16]

- The style example is s and c **is the content image** to stylized

$$\arg\min_u \|\phi(u) - \phi(s)\|^2 + \lambda \|\psi(u) - \psi(c)\|^2$$

- **Features** : same as before for texture, but **without** spatial averaging for *content* features

$$\psi(u) = (F_j(u))_{j \in \mathcal{J}}$$

- Content parameter: $\lambda \approx 10^3$

- **Layers** for content feature ψ : only relu4_2

- **Layers** for style feature φ : relu1_1, relu2_1, relu3_1, relu4_1 and relu5_1

Style transfer results

- source: [Gatys *et al.*'17]

Demo

- Pytorch implementation :
  style_transfert.ipynb

- **Lab session #4 : Neural Texture Synthesis**

s

c

u

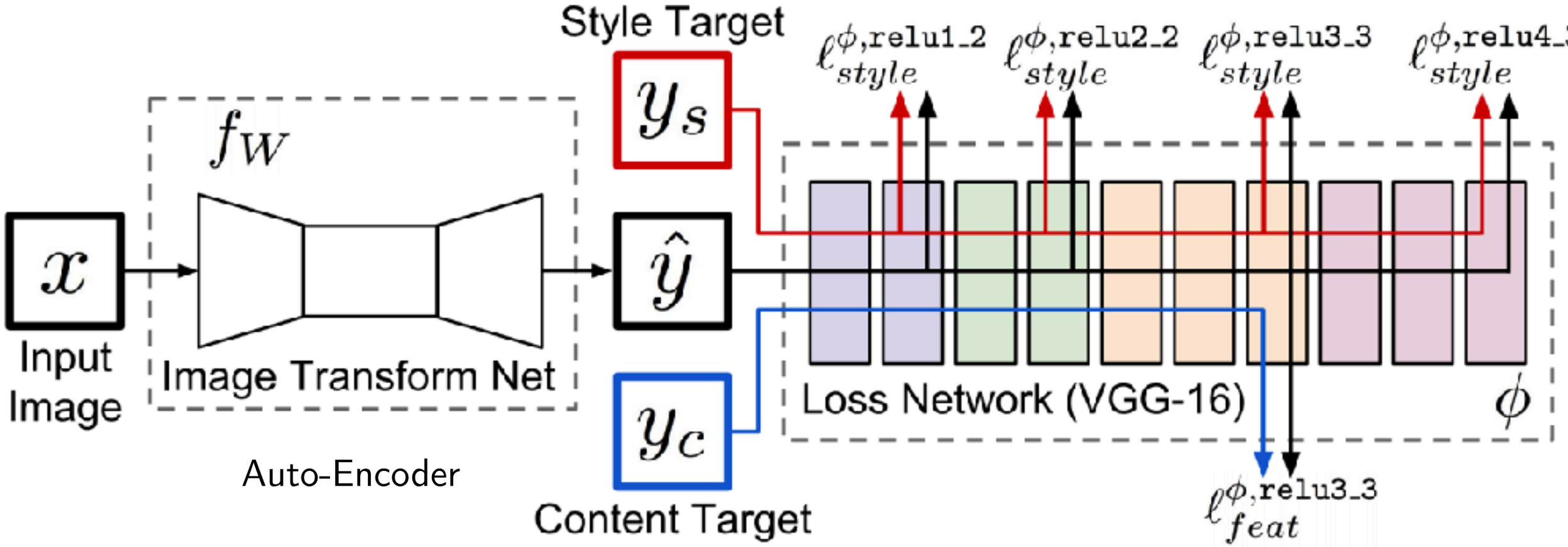
Texture Synthesis and style transfer with generative networks

Training a generator

- **Principle :** can we achieve the same results by combining the 2 previous techniques :
  - **training** a feed-forward generative neural network $z \sim p(z) \mapsto u = g_\theta(z) \sim P_\theta(u)$
  - using statistics phi from features representation of a **fixed pre-trained network (perceptual evaluation)** $\mathcal{L}(u, s) = \|\phi(u) - \phi(s)\|^2$
- Some successful approaches:
  - Perceptual Losses [Johnson *et al.*'16] : style transfer, super resolution
  - Texture Networks [Ulyanov *et al.*'16]
  - 3D Texture Networks for solid texture [Gutiérrez *et al.*'19]
  - Universal Style Transfer via Feature Transforms [Li *et al.*'17]

Perceptual Losses for real-time style transfer [Johnson et al.'16]

- **Training** on 80k images using a « perceptual loss » similar to [Gatys et al.'16]



- **Synthesis**: 40ms for a full HD image (up to x1000 speed-up)

Perceptual Losses for real-time style transfer [Johnson et al.'16]

- Results and comparison with [Gatys et al.'16]

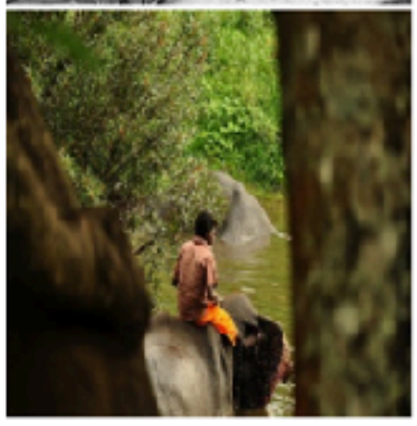
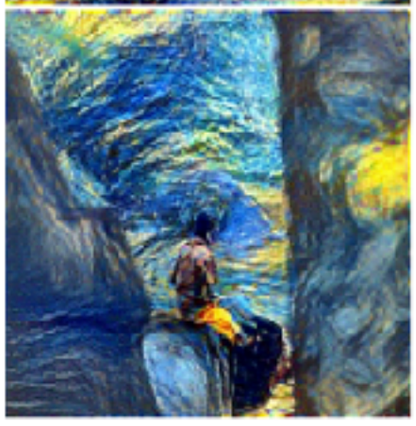
Style
The Starry Night,
Vincent van Gogh,
1889

Style
The Muse,
Pablo Picasso,
1935

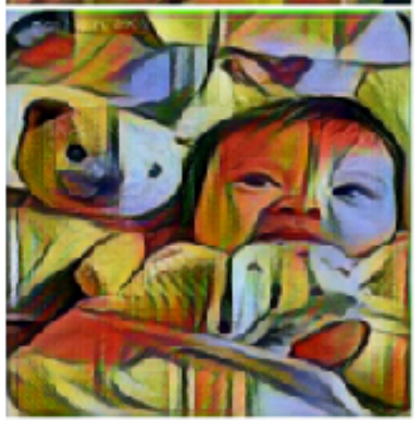








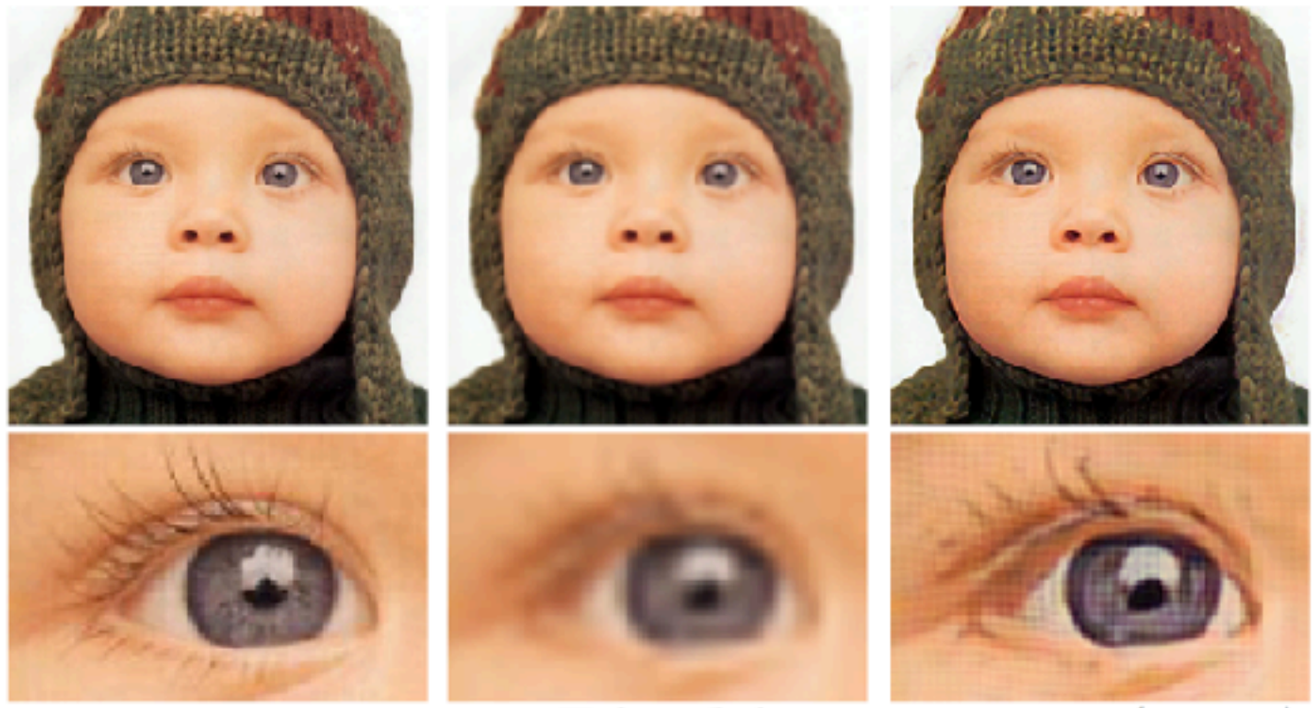


Results [Johnson et al.'16]

- Results for x4 super-resolution :

- **Pros:** very fast, more applications

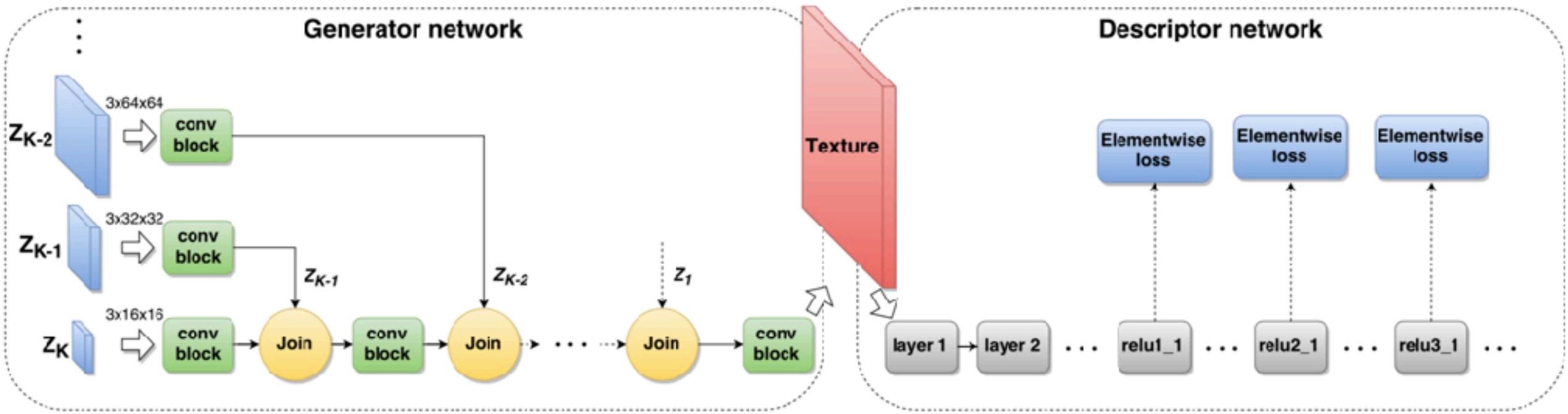
- **Cons:** not as good as [Gatys et al.'16] with artefacts (checker-board) and periodic patterns
  One network for each style (requiring several hours of training)



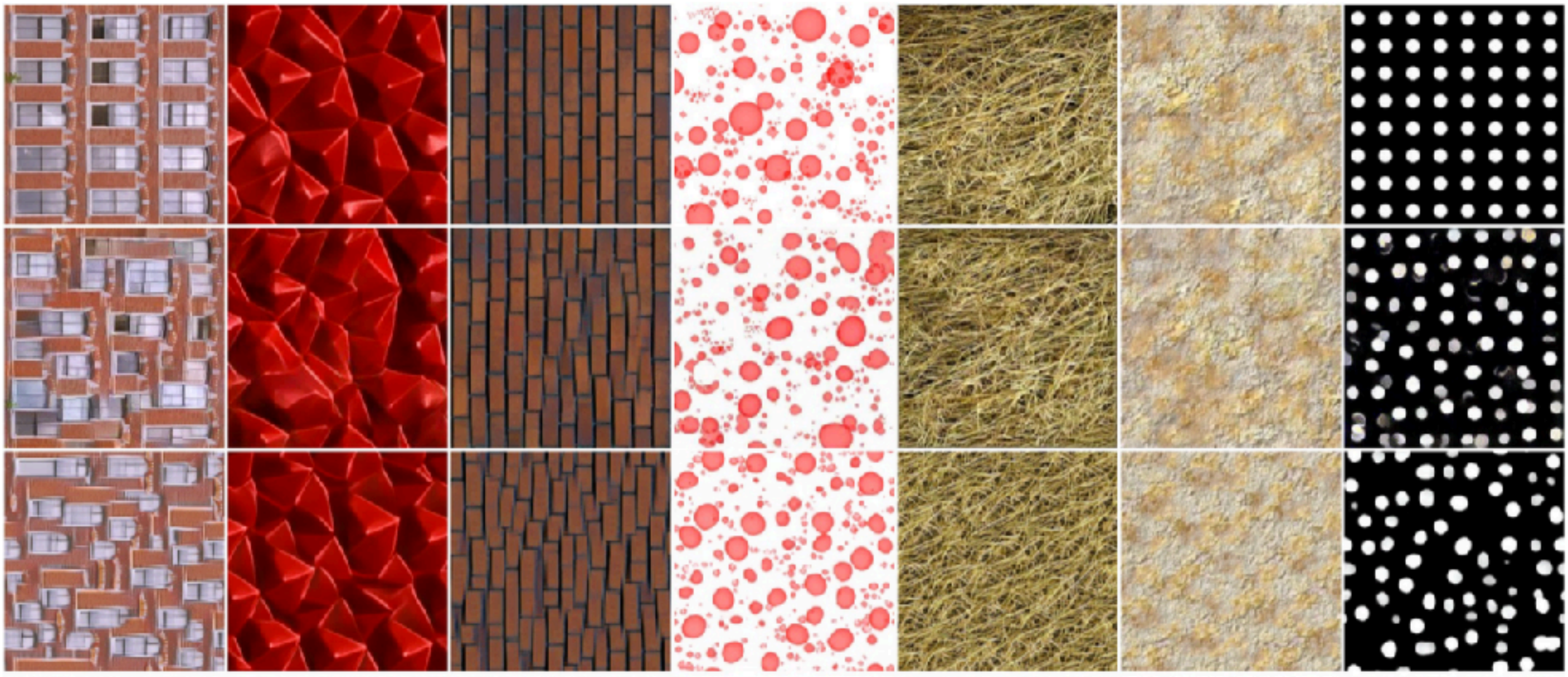
Ground Truth - bicubic interp. - generation

Texture Networks [Ulyanov *et al.*'16]

- Very similar approach to [Johnson et al.'16] with 2 main differences:
  - Investigate texture synthesis with **training from a single image** (still using VGG « perceptual » features)
  - **fully convolutional** & **compact** generative network with multi-scale noise input (~65K parameters)

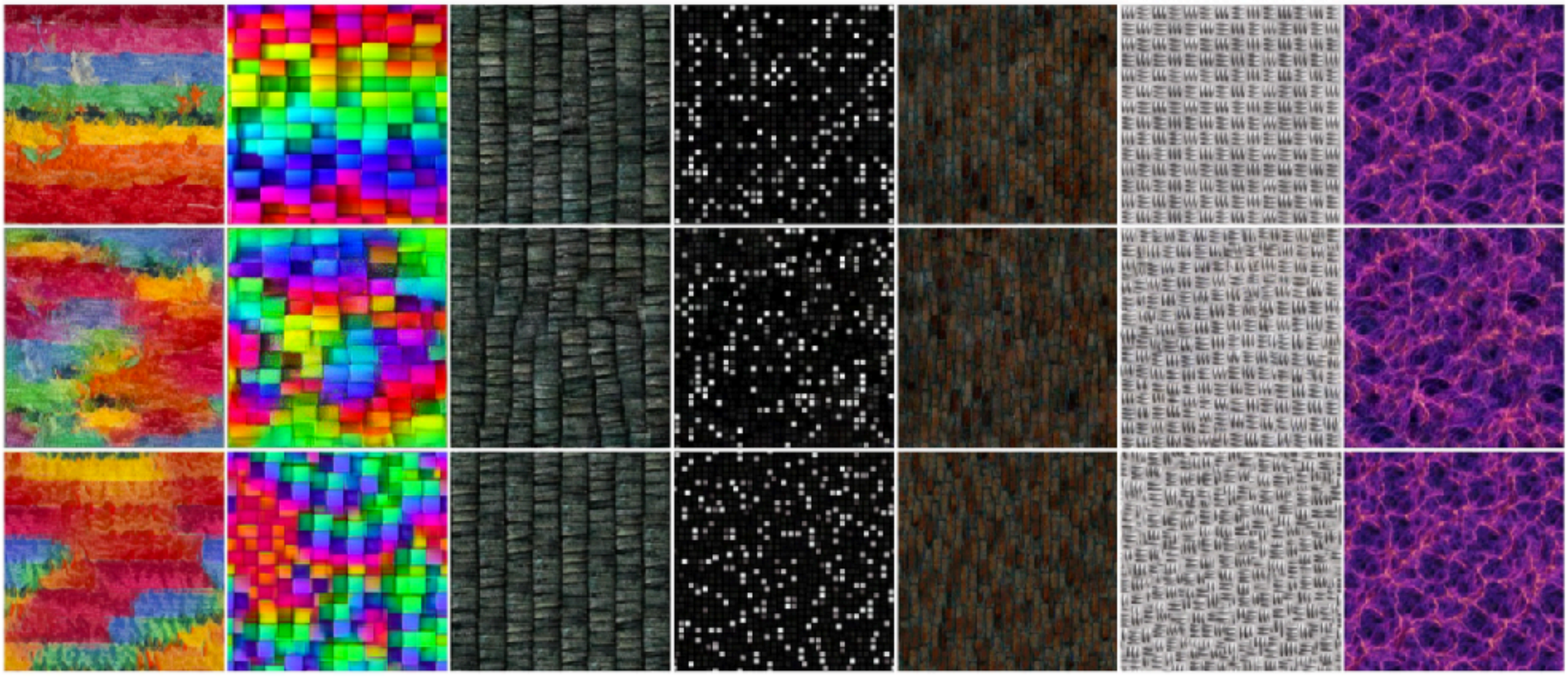


Results



- Pytorch implementation by Jorge Gutiérrez: https://github.com/JorgeGtz/TextureNets_implementation

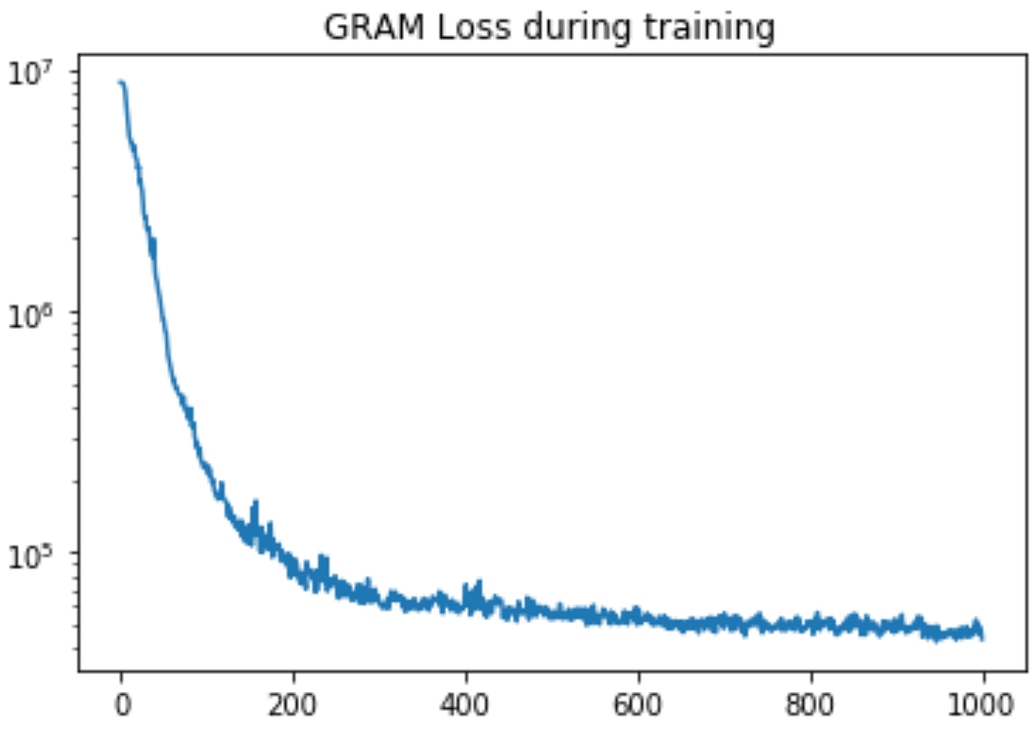
Results



- Pytorch implementation by Jorge Gutiérrez: https://github.com/JorgeGtz/TextureNets_implementation

Demo

- Pytorch implementation : texture_networks.ipynb
- **Lab session #5**



3D results

- This approach can be extended for 3D data called « solid textures »: 3D Texture Networks [Gutiérrez *et al.*'19]

3D results

- This approach can be extended for 3D data called « solid textures »: 3D Texture Networks [Gutiérrez *et al.*'19]

Universal Style transfer

- **Question**: why not training once for all a generator for several style ?

- **Various approaches**:

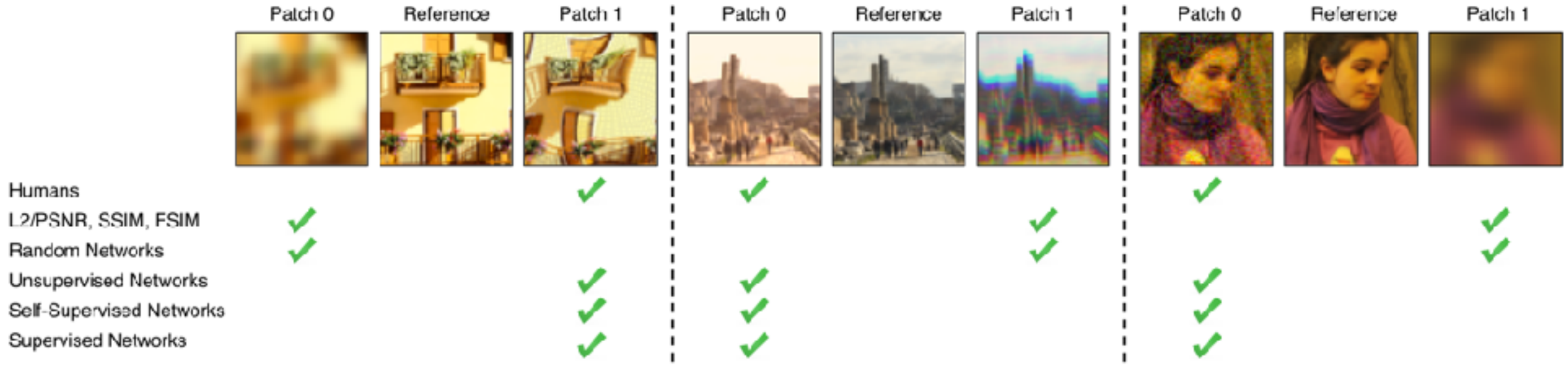
- Add information to the input to control the style: [Zang & Dana'17] (Multi-style Generative Network for Real-time Transfer), [Dumoulin *et al.*'17] (A learned representation for artistic style)

- Universal Style Transfer via Feature Transforms [Li et al.'17] : similar to Variationnal Auto-Encoder (VAE) method where the Encoder is defined from VGG and the Decoder is trained on a dataset; a feature transform (moment matching) is used in the **latent space**

Perceptual comparison of Images

Deep Features for perceptual comparison of **images** : LPIPS

- [Zhang'18] "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric" :



LPIPS

- [Zhang'18] "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric" for image patch comparison

- Learned Perceptual Image Patch Similarity (**LPIPS**) metric :

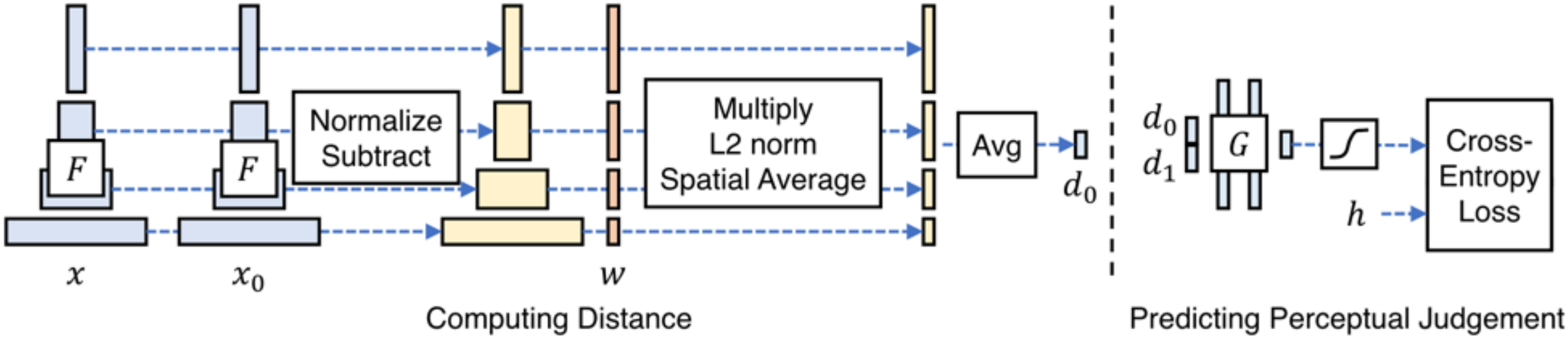
$$d(x,y) = \sum_{\ell} \frac{1}{H_\ell W_\ell} \sum_{(h,w)\in[H_\ell][W_\ell]} \|\omega_\ell \odot \left(\phi_\ell(x)|_{h,w} - \phi_\ell(y)|_{h,w} \right)\|^2$$

  - where ϕ_ℓ is the layer ℓ of a trained neural network (**SqueezeNet**, **AlexNet**, or **VGG**, as before)

  - and **hidden variables vectors ω_ℓ are learnt for "perceptual calibration"** (rather than manually fixed in previous works)

LPIPS

- Training weights ω_ℓ requires to compute a **score on the distance** $d(x, y)$ between two image patchs
- A **dataset** of **human** perceptual similarity judgments has been collected, between triplet (x, x_0, x_1) with a **query** patch x and **two reference** patchs x_0 and x_1: $\mathrm{h}(x, x_0, x_1) = 1\{$ x closer to x_0 or x_1 $\}$?
- An additional classification network G is train to provide a decision based on $d_0 = d(x, x_0)$ **and** $d_1 = d(x, x_1)$

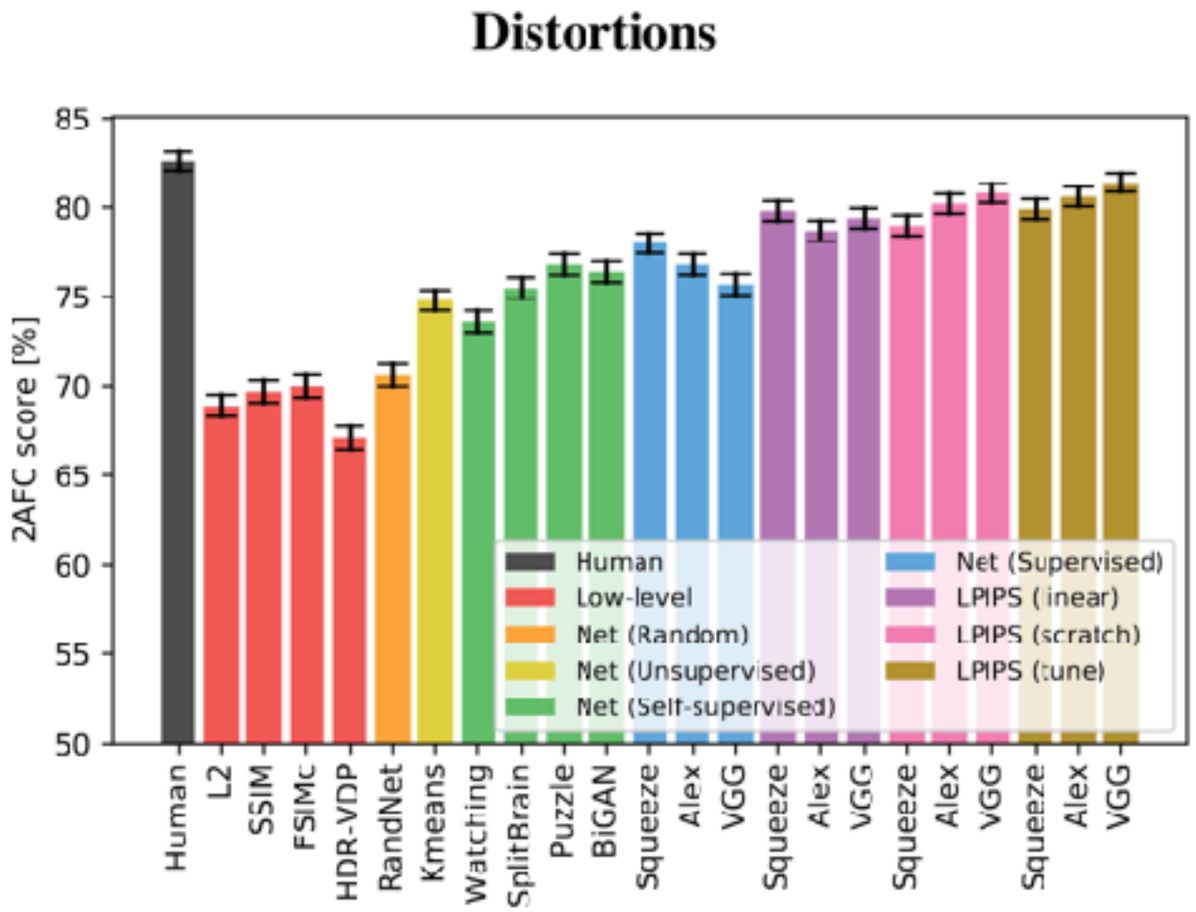


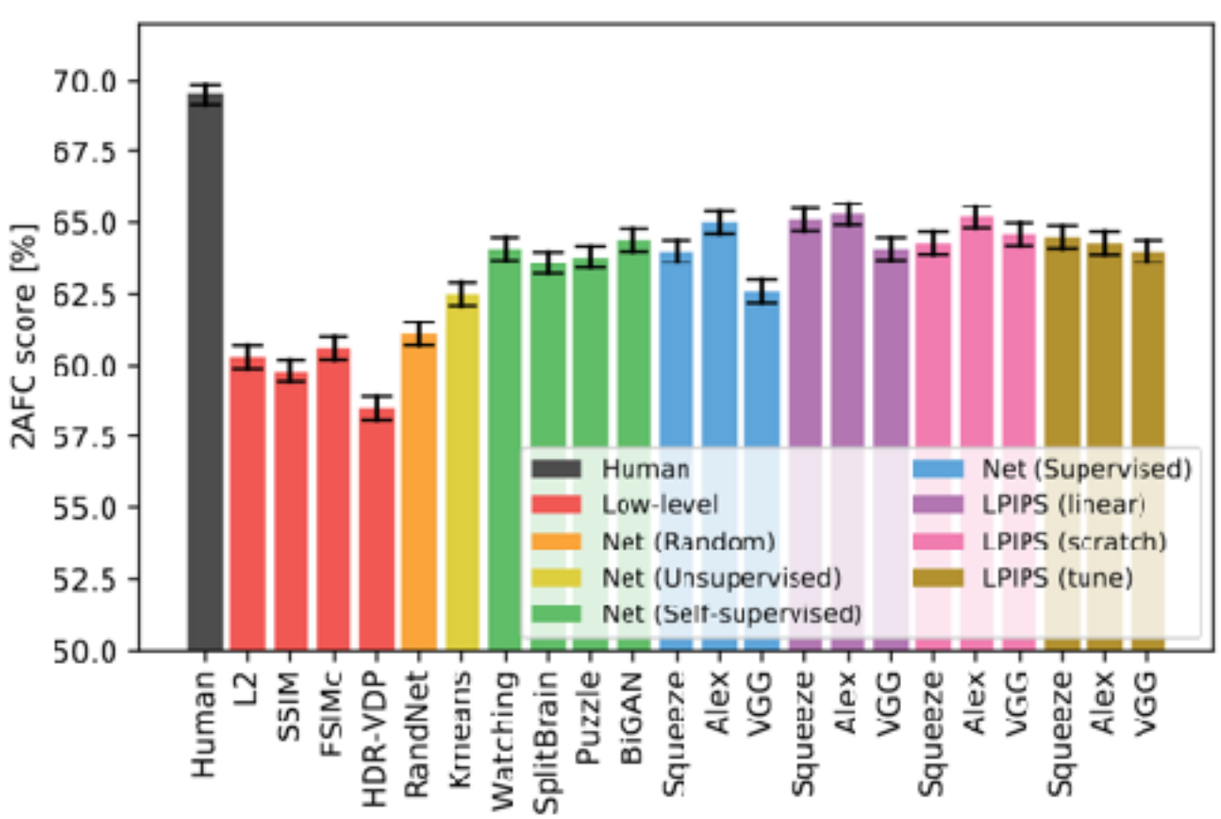
$$d(x, y) = \sum_\ell \frac{1}{H_\ell W_\ell} \sum_{(h,w)\in[H_\ell][W_\ell]} \|\omega_\ell \odot \left(\phi_\ell(x)|_{h,w} - \phi_\ell(y)|_{h,w}\right)\|^2$$

Training with Binary Cross-Entropy using human judgement h

LPIPS

- Results on artificial patch distorsions (blur, noise, warping,) and image tasks (super-resolution, deblurring, frame interpolation, colorisation):
  - training improve consistency over supervised representation in both case





2AFC = two alternative forced choice

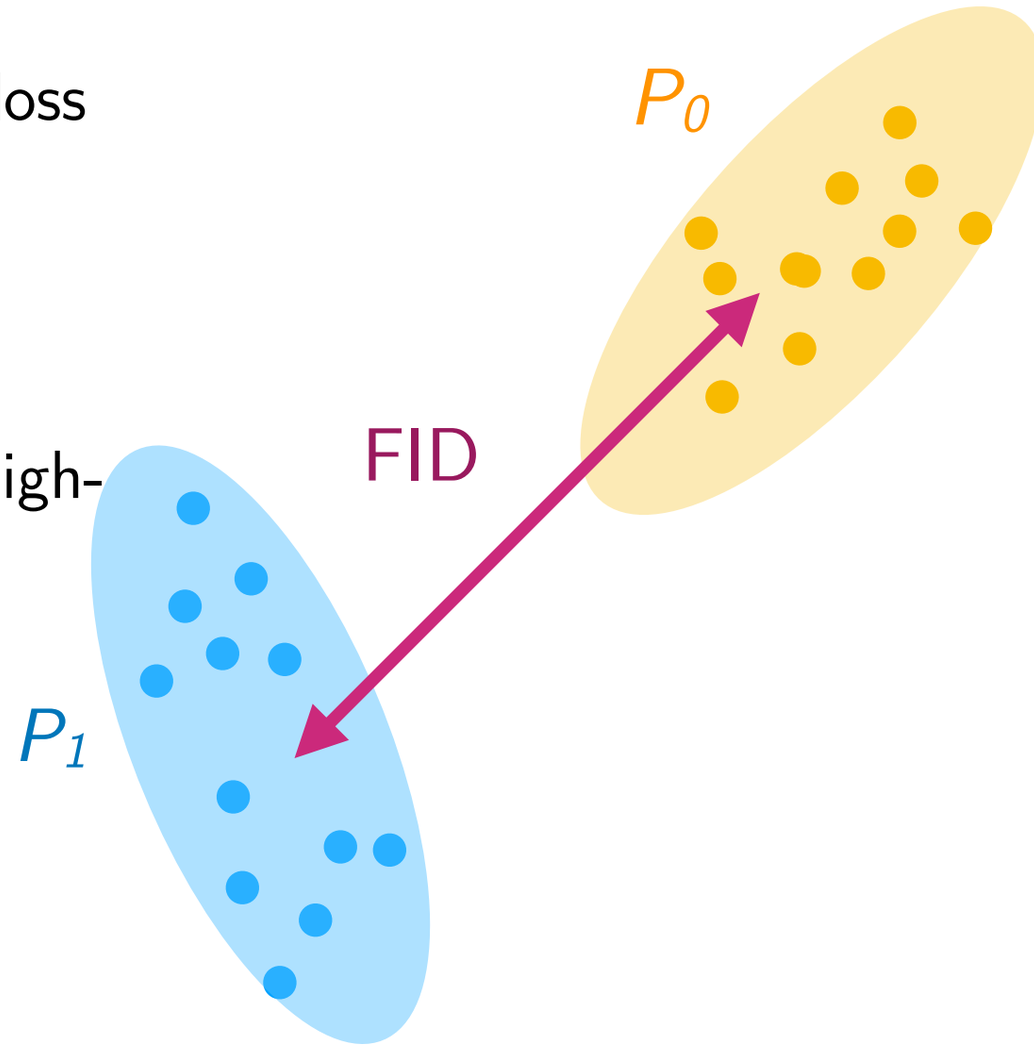
Note : Human ≠ 100% because of inconsistency across different users

Deep Features for perceptual comparison of **image distributions** : **FID**

- [Heusel'18] introduces the "Fréchet Inception Distance" (FID) as a metric between collections of images

$$\mathrm{FID}(P_0, P_1)^2 = \|\mu_0 - \mu_1\|_2^2 + \mathrm{tr}\left(\Sigma_0 + \Sigma_1 - 2\left(\Sigma_0^{\frac{1}{2}} \cdot \Sigma_1 \cdot \Sigma_0^{\frac{1}{2}}\right)^{\frac{1}{2}}\right)$$

- **Remark:** Using VGG would be unfair when evaluating a model trained with the perceptual loss based on VGG
- Inception Network is used as an alternative (yet another ImageNet Classifier)
  - multi-variate gaussian modelling $\mathcal{N}(\mu_i, \Sigma_i)$ (high-dimensional feature) for both training and generated samples
  - closed-form formula
- Other variations (SiFID, R-FID, S-FID ...)



Conclusion

- Deep learning approaches overview for image editing / synthesis:
  - **Pros:** feed-forward generative network are very fast, by far state-of-the-art performance
  - **Cons:** not simple, no control nor flexible, need GPUs with large memoryEvaluation
- What's next : image generation (part III)