Transfert de style & Generative Adversarial Networks

Tout un art au service de l'art !

FEV
1

de 19h à 22h

Meetup TDS

chez HARRY COW

INSTITUT Mines-Télécom

TDS NoBlaBla : Style Transfer and Generative Adversarial Networks

Julien Guillaumin
julien.guillaumin@imt-atlantique.net
IMT Atlantique (ex Télécom Bretagne)

- Student at **IMT Atlantique**, Brest.
  - Engineering degree - Computer Vision

- Deep Learning Engineer at **Lighton.io**
  - Optical co-processor for Machine Learning

  - **SummerIA.fr**, Deep Learning Tutor
  - **Continental**, Deep Learning for ADAS (intern)
  - **Thales Services**, Large-scale Image Processing with Spark (intern)

- Talks and MeetUps
  - **PyCon France 2016**
  - **Toulouse Data Science, Paris AI, Data Science Brest**, **Machine Learning Aix-Marseille**
  - **Airbus Defence and Space, Quantmetry, Ercom**

Outline

- **Introduction to Deep Learning**
  - Basics of Machine Learning
  - Machine Learning vs Deep Learning
  - Convolutional Neural Networks - CNNs
- **Style Transfer with Neural Networks**
  - Perceptual loss : content loss + style loss
  - Optimization-based method: first steps in neural style transfer
  - Train CNNs to perform neural style transfer
- **Generative Adversarial Networks - GANs**
  - How to generate realistic images ?
  - Two networks, two players: Hi Game Theory !
  - GANs for semi-supervised learning and domain adaptation
  - Can we understand the latent space ?
  - Applications: Super-resolution

Introduction to Deep Learning

Basics of Machine Learning

- **Field of Artificial Intelligence**
- **Develop learnable algorithms**
  - Learn from data
  - To resolve complex tasks
- **Many tasks :**
  - Natural Language Processing
  - Image classification
  - Object segmentation

Two major phases :

- **Training**
  - From training data
  - Adjust internal parameters
  - Goal : find generalization !
- **Inference**
  - New data (not seen)
  - Evaluation / Production

Basics of Machine Learning

Case of Image Classification

$$\theta \rightarrow f_\theta(x^{(i)})$$

$$y^{(i)} \rightarrow bird$$

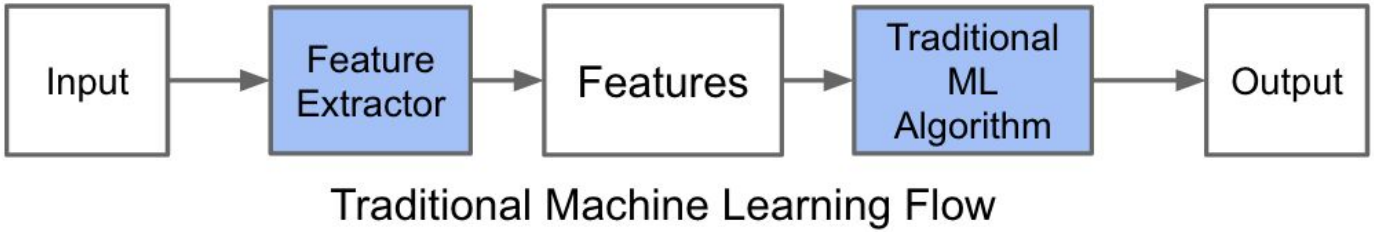
$$\begin{array}{l} \mathbb{P}(\ Y\ =\ truck \mid x^{(i)}\ , \theta) \\ \mathbb{P}(\ Y\ =\ bird \mid x^{(i)}\ , \theta) \\ \vdots \\ \mathbb{P}(\ Y\ =\ plane \mid x^{(i)}\ , \theta) \end{array}$$

0.12
0.81
.
.
.
.
0.05

$$E = -log\ (P(\ Y = y^{(i)} \mid x^{(i)}\ , \theta))$$

negative log-likelihood
E : error for this example

Machine Learning vs. Deep Learning

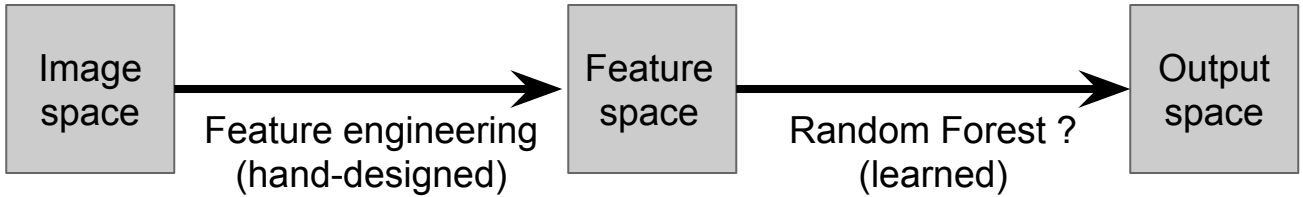


Traditional Machine Learning Flow

Deep Learning Flow

Machine Learning vs Deep Learning

Machine Learning approach

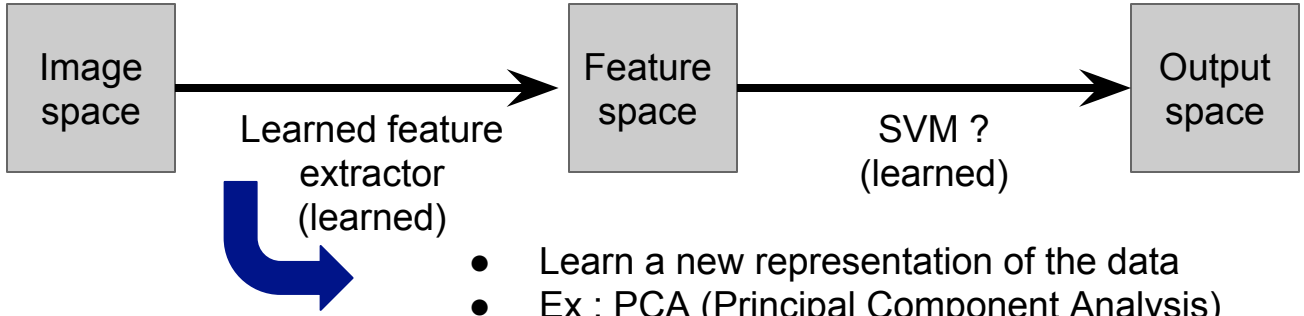


- Domain dependence
- Need expert domain to tune
- Hard to extract complex patterns

For images : HOG features, SIFT methods, Histograms, LBP features, ….

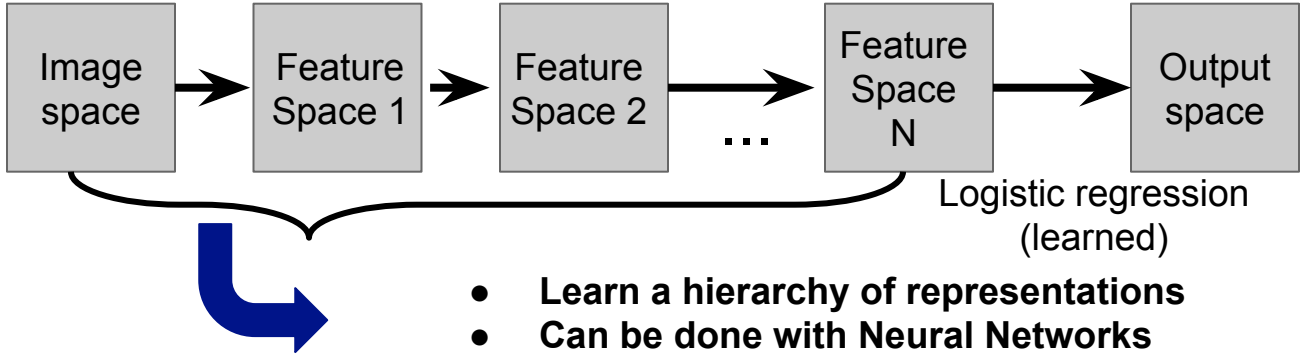
Machine Learning vs. Deep Learning

Representation Learning approach



- Learn a new representation of the data
- Ex : PCA (Principal Component Analysis)

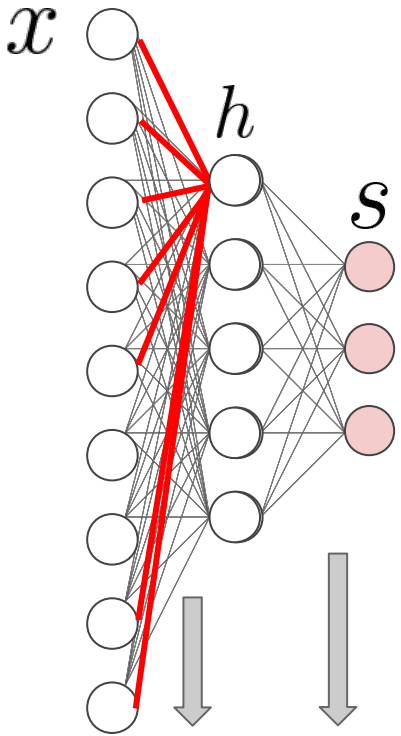
Deep Representation Learning approach



- **Learn a hierarchy of representations**
- **Can be done with Neural Networks**

Deep Neural Networks - DNNs

- Bilogically inspired
- Representation as vectors
- Learn to perform vector transformations
- Weighted sum + biases
- Activation function



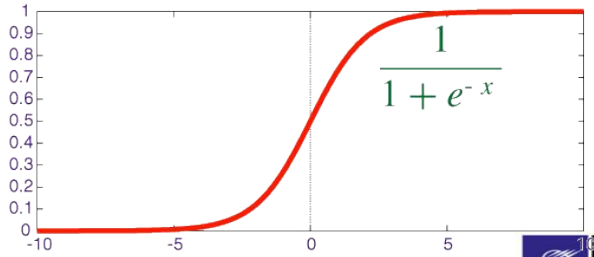
$$\theta = \{W_1, b_1, W_2, b_2\}$$

1. Weighted sum + biases
2. Activation function

Weights and biases are learned !

$$h = act(W_1 x + b_1)$$
$$s = softmax(W_2 x + b_2)$$

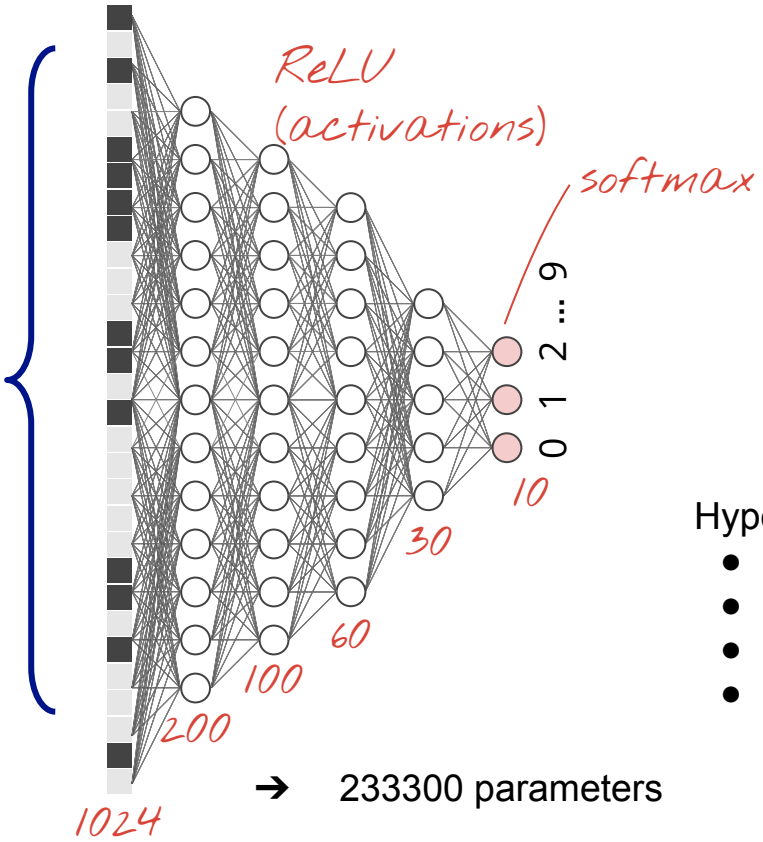
Activation function :
- Sigmoid
- Tanh
- ReLU !



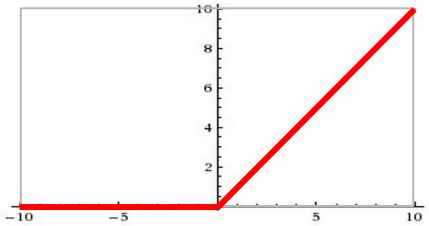
Deep Neural Networks - DNNs

32

32



ReLU function



Hyperparameters to tune:

- How many hidden layers ?
- How many neurons per layer ?
- Which activation ?
- Regularization ?

➔ 233300 parameters

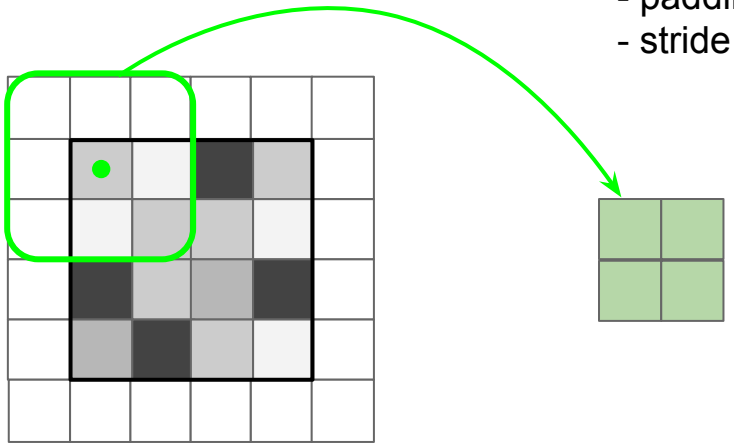
Convolutional Neural Networks - CNNs

Intuition for CNNs :

- Keep 2D representation
- High correlation between adjacent pixels
- Weight sharing

- Many hyper-parameters :
  - kernel size, padding, stride, with bias ?

x : [4,4] + zero padding



- kernel 3x3
- padding = ‘same’
- stride = 2

To learn $\left\{ \begin{array}{|c|c|c|} \hline w_1 & w_3 & w_3 \\ \hline w_4 & w_5 & w_6 \\ \hline w_7 & w_8 & w_9 \\ \hline \end{array} \right. \quad b$

Convolutional Neural Networks - CNNs

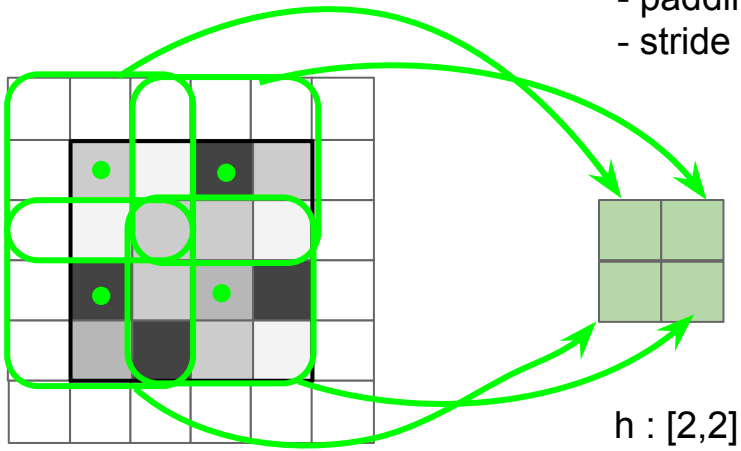
Intuition for CNNs :

- Keep 2D representation
- High correlation between adjacent pixels
- Weight sharing

- Many hyper-parameters :
  - kernel size, padding, stride, with bias ?

x : [4,4] + zero padding

- kernel 3x3
- padding = ‘same’
- stride = 2



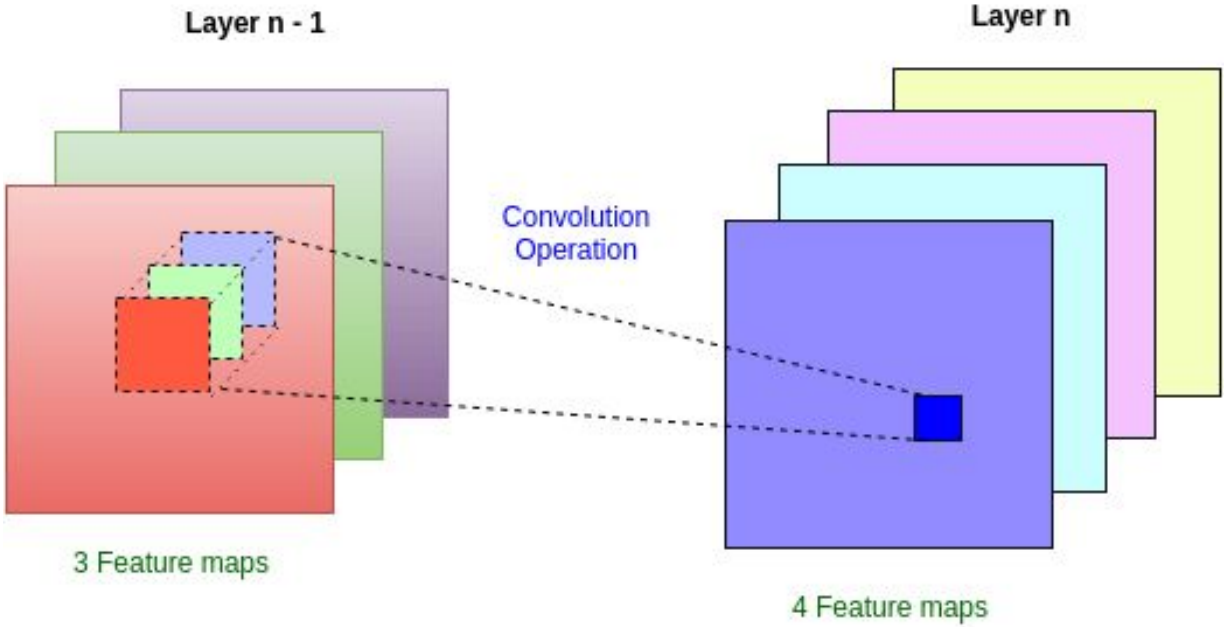
h : [2,2]

To learn $\left\{ \begin{array}{|c|c|c|} \hline w_1 & w_3 & w_3 \\ \hline w_4 & w_5 & w_6 \\ \hline w_7 & w_8 & w_9 \\ \hline \end{array} \right. \quad b$

$$h = act(x * W + b)$$

Convolutional Neural Networks - CNNs

New representation is composed of **“Feature Maps”**

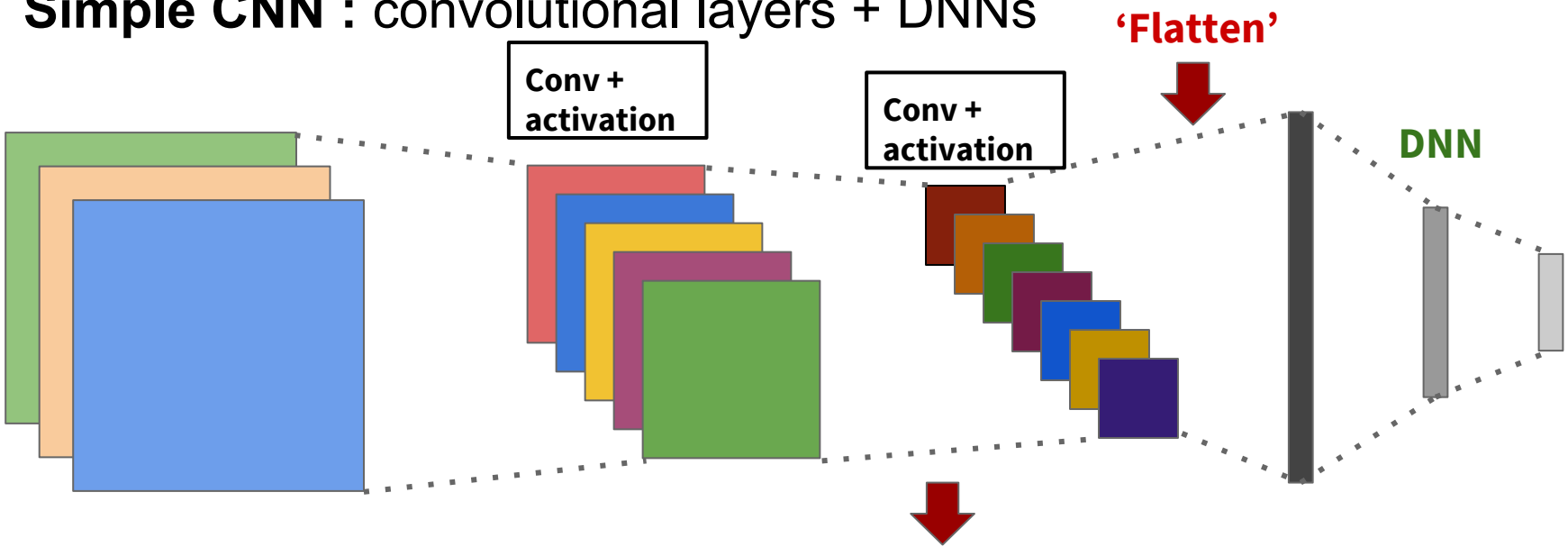


Here :

- ➔ 4 kernels to create 4 feature maps
- ➔ from 3 feature maps (RBG images, for example)
- ➔ (3x3x3 + 1)x4 : 112 parameters !

Convolutional Neural Networks - CNNs

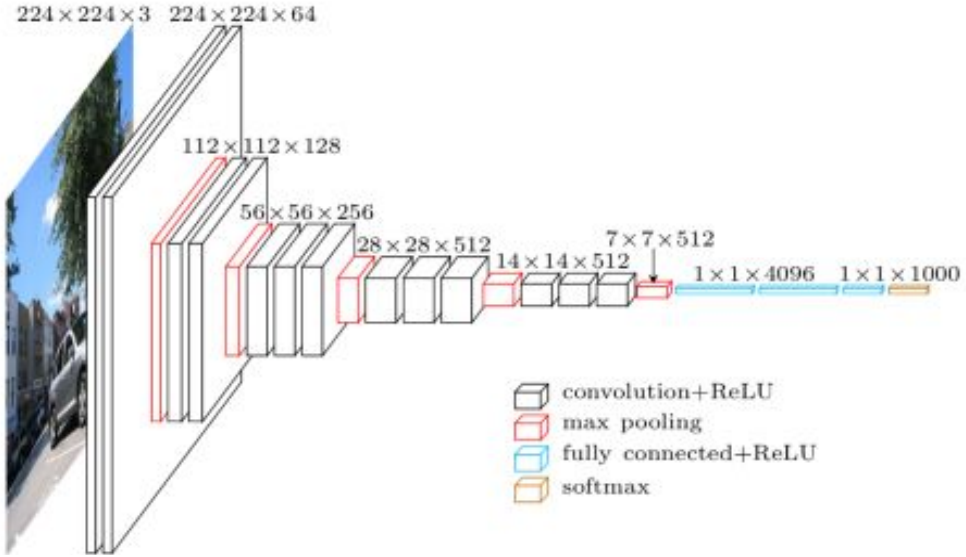
Simple CNN : convolutional layers + DNNs



Add **pooling operations** (Average, Max), to reduce the feature maps !

Deep Network : VGG-16 [1]

- Simple : no **inception modules[2]** or **residual connections[3]**
- Trained for **image classification** on ImageNet[4] (1000 classes)
- State of the art in 2014 (92.7% top-5 test accuracy)
- 138,357,544 parameters (10% conv weights, 90% FC layers)

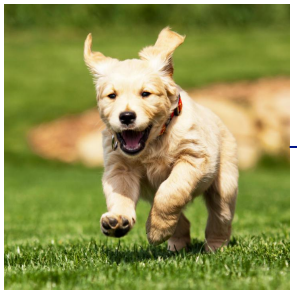


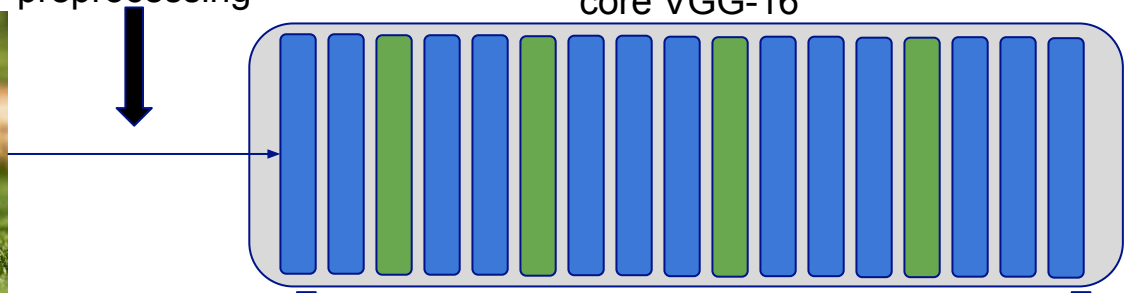
Neural Style Transfer

Neural Style Transfer: Motivations

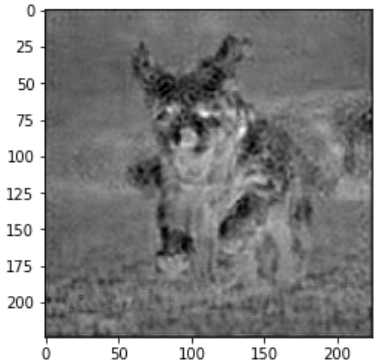
- **Generative task**
  - From an image, generate a new one
- **Introduction to more complex tasks**
  - Super-resolution and colorisation
- **CNNs understanding is required**
  - Hierarchy of representations
  - Feature spaces

CNN visualization



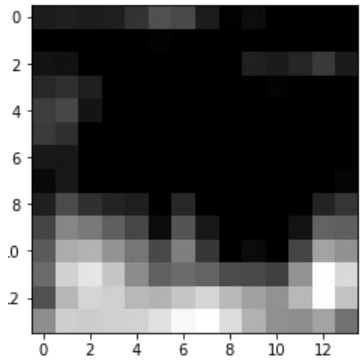


conv1_1 (224x224x64)



From low-level to high-level feature spaces

conv5_3 (14x14x512)

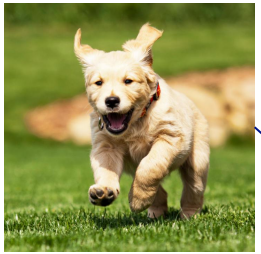


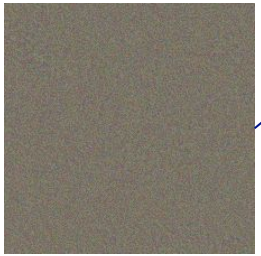
Additional visualization methods :

- Deep Dream approach [5]
  - Optimization-based
- Zeiler & Fergus [6]
  - Transposed convolutions and unpooling operations

Content Representation/Reconstruction

x_c



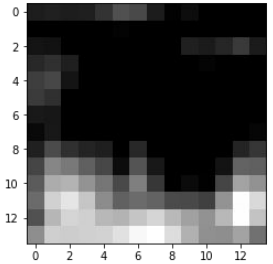


x

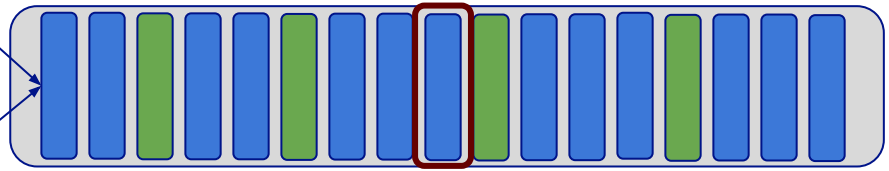
$\Phi_j(x)$: activations of the *j*th layer

Eg :

$\Phi_j(x)[40]$



conv3_3 : 56x56x256



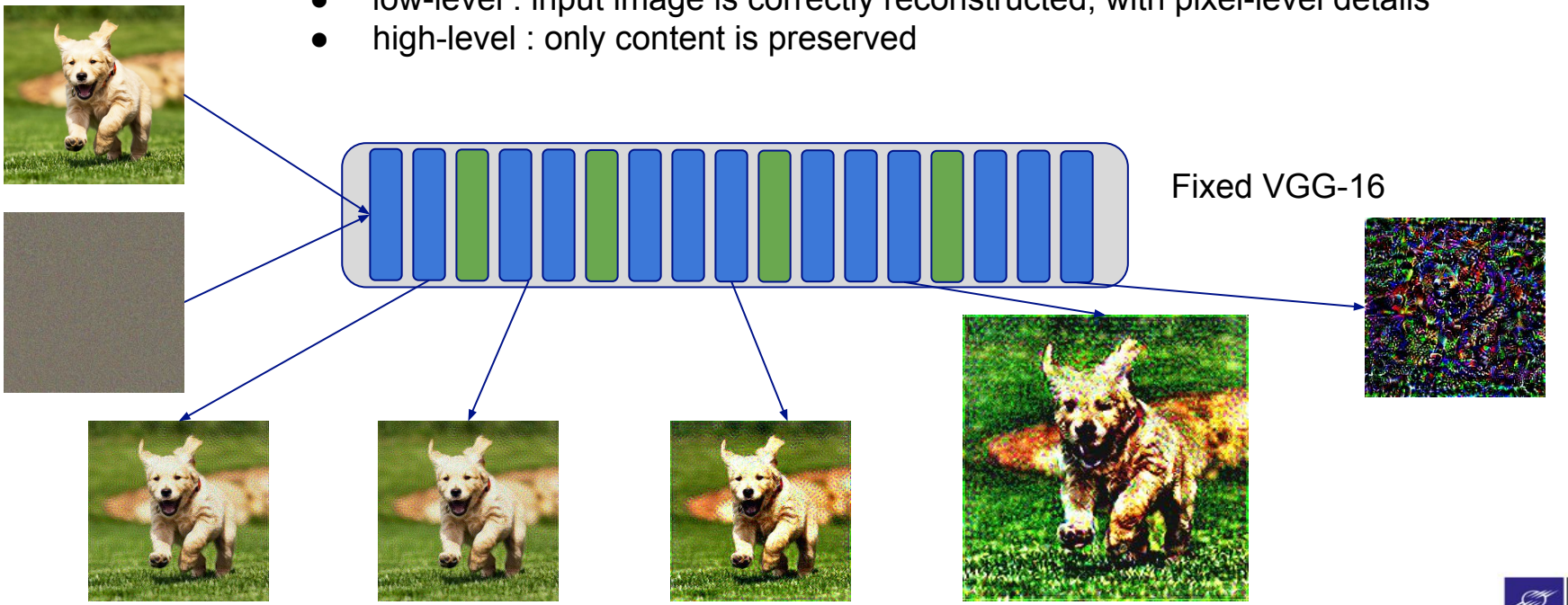
Fixed VGG-16

$$\hat{x} = \underset{x}{\operatorname{argmin}} \; ||\Phi_j(x) - \Phi_j(x_c)||_2^2$$

- Goal : find an image with the same activations at a given layer (all feature maps)
- Optimization problem, start from a random image

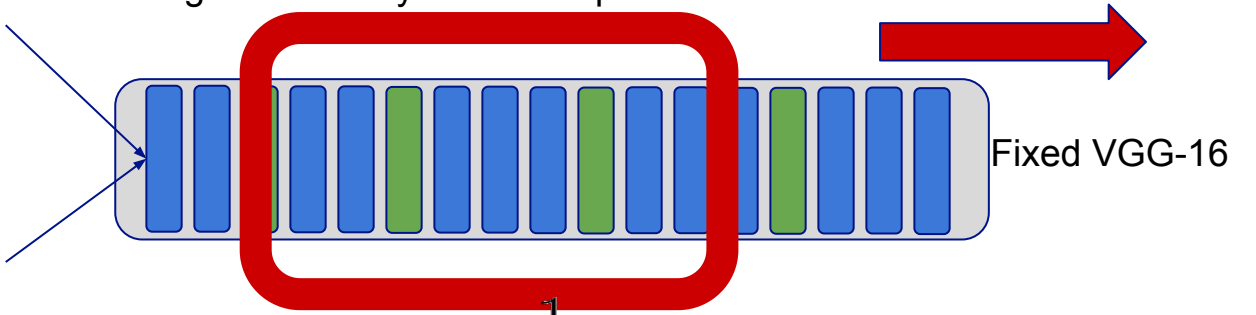
Content Representation/Reconstruction

- gradient descent optimization on input image, network does not change
- loss = MSE on feature maps, 1000 iterations, Adam (lr=2.0)
- low-level : input image is correctly reconstructed, with pixel-level details
- high-level : only content is preserved



Content Representation/Reconstruction

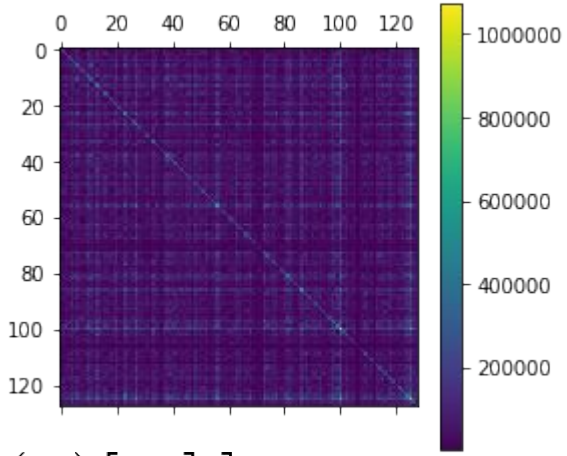
- From a random image, reconstruct the feature maps obtained with a normal image, on a specific layer
- Gradient descent optimization on image input, network does not change
- Loss = MSE on feature maps, 1000 iterations, Adam (lr=2.0)
- Low-level : input image is correctly reconstructed, with pixel-level details
- High-level : only content is preserved



Content only

$$L_c(x, x_c) = \frac{1}{C_j H_j W_j} ||\phi_j(x_c) - \phi_j(x)||_2^2$$

Style Representation/Reconstruction



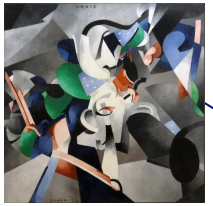
- Needs more complex statistics on feature maps : **Gram matrix**
  - Second-order statistics
  - Can capture texture information, no spatial information
- For a given layer j with C_j feature maps of size (W_j, H_j)
- The Gram matrix is a (C_j, C_j) matrix :

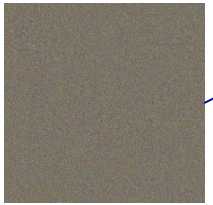
$$G_j(x)_{c_1,c_2} = \mathbb{E}[\ \Phi_j(x)[c_1] * \Phi_j(x)[c_2]\ ]$$

- Where $*$ is an element-wise operation between 2 feature maps (Hadamard product)
- Contains the correlation between every pair of feature maps

Style Representation/Reconstruction

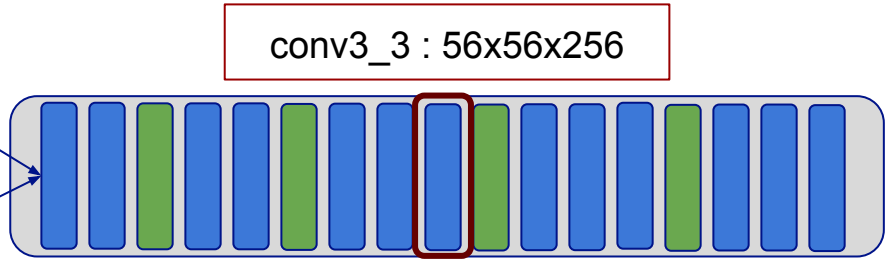
x_s





x

$G_j(x)$ Gram matrix of the *j*th layer (256 x 256)



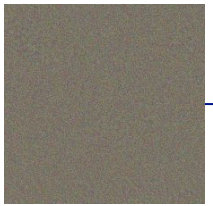
$$\hat{x} = \underset{x}{\operatorname{argmin}} \; ||G_j(x) - G_j(x_s)||_2^2$$

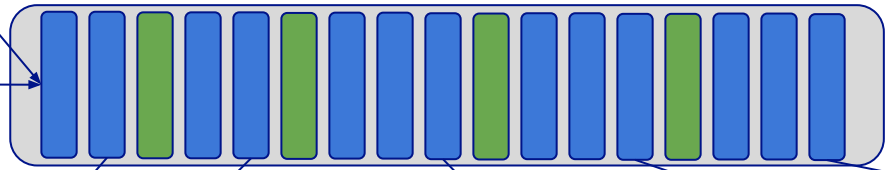
- Goal : To find an image with the same Gram matrix for a given layer
- Optimization problem: Start from a random image

Style Representation/Reconstruction

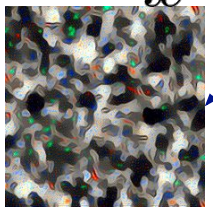
- Gradient descent optimization on image input, network is freezed
- Loss = MSE on feature maps, 1000 iterations, Adam (lr=2.0)
- Low-level : Small and simple patterns
- High-level : More complex patterns

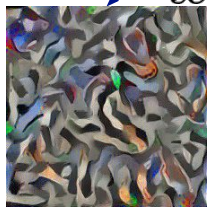
$$L_s(x, x_s) = \sum_j \frac{\lambda_j}{C_j^2} ||G_j(x_s) - G_j(x)||_2^2$$

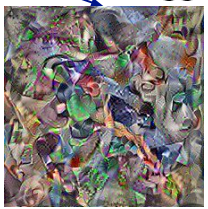




Fixed VGG-16



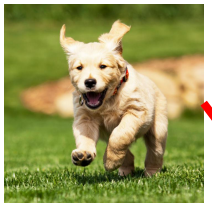




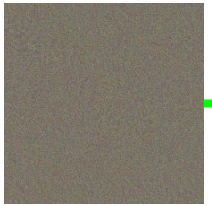
Content & Style Representations

- Content is preserved in high level features
- Style is present in second-order statistics in low and medium levels
- **Content and Style are separable**
- ***content_loss* and a *style_loss* are defined**
- Combine style and content from different images is possible, via feature extraction learned within a VGG network, trained on a generic image classification task !

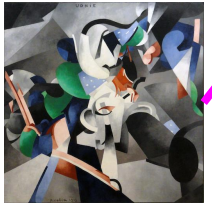
Mix content & style via specific losses



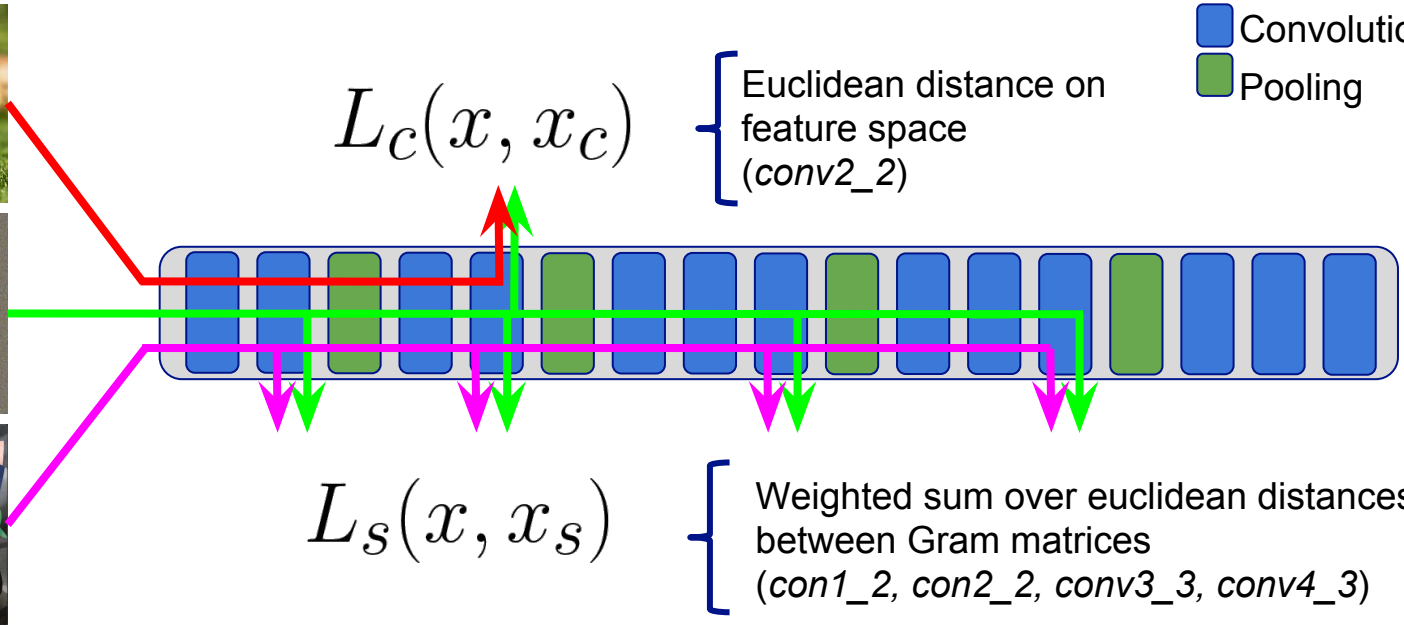
x_c



x



x_s



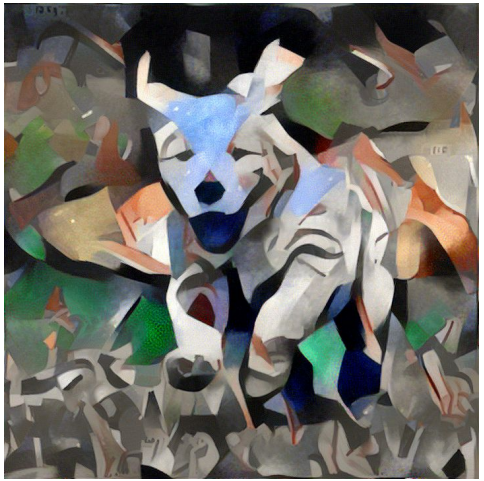
$$L(x, x_s, x_c) = \alpha L_s(x, x_s) + \beta L_c(x, x_c)$$

Perceptual loss and method defined in [7]

Optimization process

- Compute *content_target* (feature maps) with *content_image*
- Compute *style_target* (Gram matrices) with *style_image*
- Start from a random image (*input_image*)
- Optimization process :
  - Compute *content_loss* and *style_loss* with targets + *input_image*
  - Minimize this loss by modifying *input_image*
  - Possible thanks to gradient-descent method (like Adam)

TensorFlow implementation



it = [100, 400, 800, 3000]

Results

- **Produce high-quality images**
- **Easy to tune effects (more content ? more style ?)**
- **Any input/output size**

- **Running time (1000 #iter)**
  - GPU (GTX 1070) : ~ 5 min (1920 CUDA cores)
  - CPU (i7-7700K) : ~ 150 min (4 cores x 2 threads)

- **Avoid any real-time applications**
- **But perceptual loss (content+style) is defined**

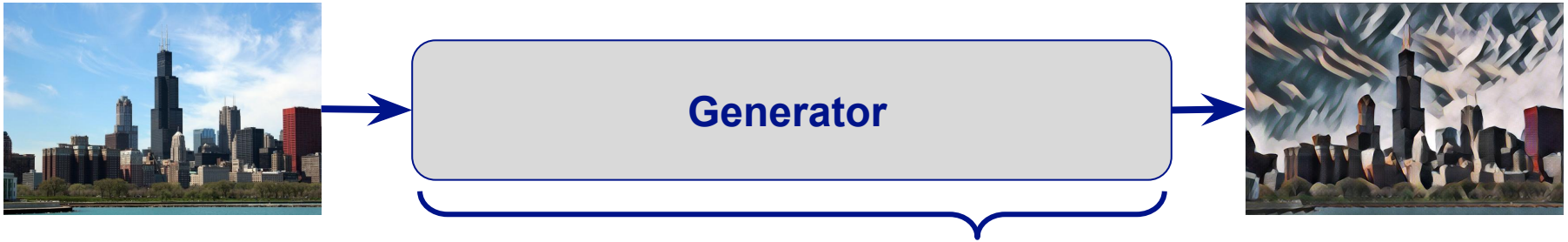
Improvements

- Time dependency for video transformation (see [8])
- Change optimizer : L-BFGS !
- Tune weights between style and content loss
- Start from : content image? style image? noisy image? or a mix?
- Color constraint : preserve color from content image ! (see [9])

from : github.com/tensorflow/magenta

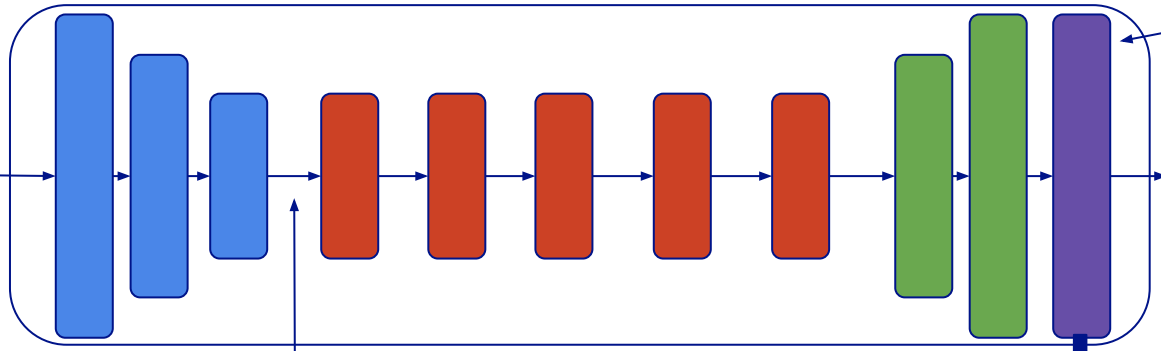
Feed-forward method [10, 11]

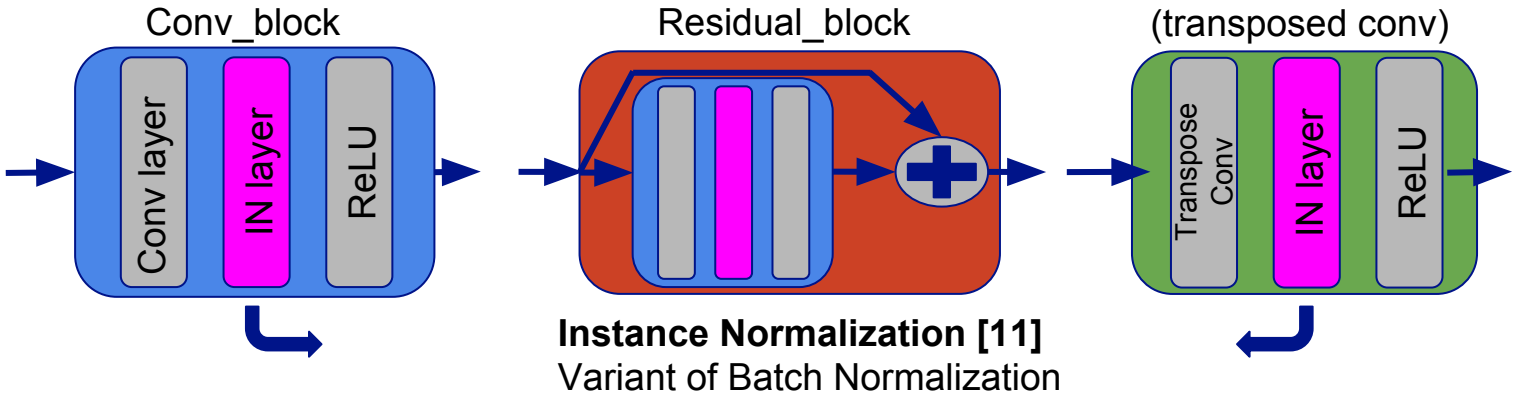
- Train a network to obtain a stylized image in one pass as an output
- Used for one specific style (fixed)



- Trained to add this style
- With a dataset of content images
- Same input/output size

Architecture of the generator ?





Instance Normalization [11]
Variant of Batch Normalization

How to train a generator ?

content image x_c

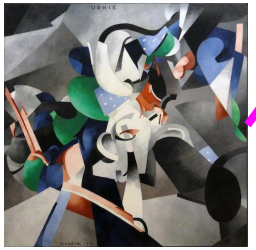
Generator θ

$G_\theta(x_c)$

pre-trained VGG-16

$L_c(x_c, \theta)$

$L_s(x_s, \theta)$



style image x_s

- Train with batch of content images
- Minimize total loss w.r.t. theta

Need a dataset of content images

- COCO dataset[12], about 80k images
- Only 1 style image

Training process (loop) :

- Take a batch of samples from COCO
- Pass this batch through the generator to get generated images
- Compute *style_loss* between the generated images and the style image
- Compute *content_loss* between the generated images and the original ones
- Minimize the *total_loss* by updating the weights from the generator

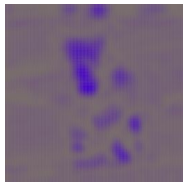
Training information :

- Adam optimizer (*lr=0.05*)
- Only 20k iterations (with *batch_size=4*)
- For 512x512x3:
  - **Training time (on GTX 1070) : 10 hours**
  - **Inference time : 330 ms (GTX 1070)**

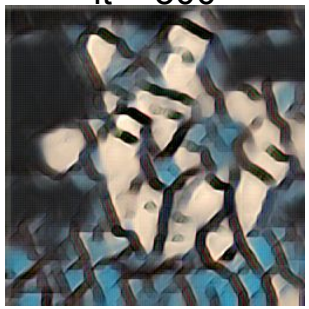
Results and improvements

- With a new content image :

it = 1

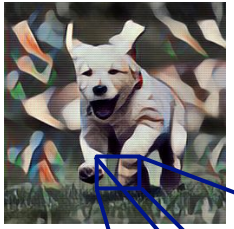


it = 500



it = 2000

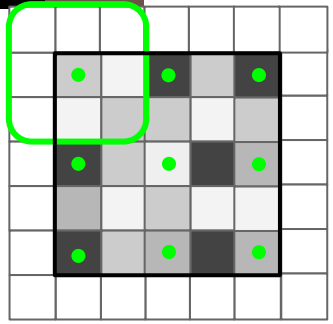
it = 12000



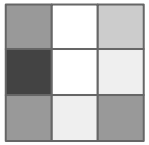
- Learn to apply only one style (with fixed style/content levels!!)
- In [13] (ICLR 2017) :
  - Add '**Conditional Instance Normalization**'
  - Learn to apply a fixed set of styles (until 64)
  - Can learn quickly a new style (incremental learning)

- Use **resized convolutions**[14] instead of transposed convolutions : Improves quality
- Add **variational loss** to encourage spatial smoothness
- Now : Universal/Arbitrary Style Transfer ! [15, 16]

Conv vs. Transposed Conv



x : [5, 5] , ẍ : [25,]



y : [3,3], ÿ : [9,]

Conv2d, kernel size : 3x3, stride=2, padding='same'

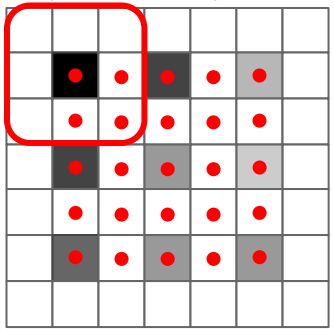
$$w * x = y$$

$$M\ddot{x} = \ddot{y}$$

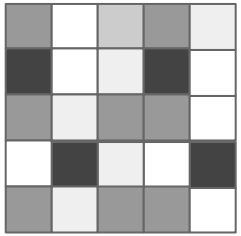
M : [9, 25]

$$\begin{bmatrix} w_1 & w_3 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix}$$

y : [3, 3] , ŷ : [9,]



x : [5, 5] , x : [25,]



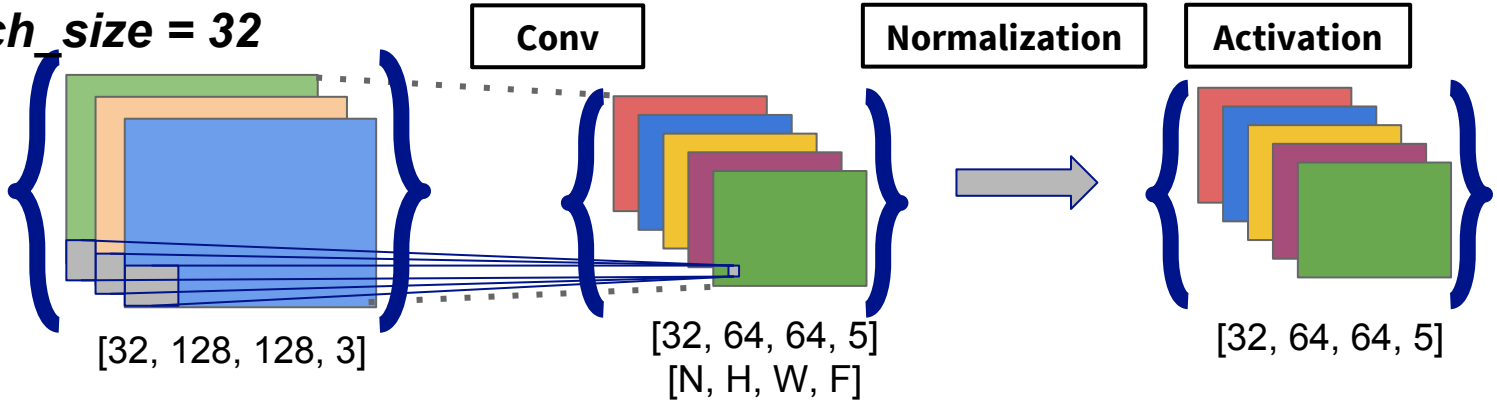
TransposeConv2d, kernel size 3x3, stride=2, padding="same "

Conv2d, kernel size 3x3, stride=1, "internal zero padding", padding="valid",

$$\mathbf{M}^\top \hat{y} = \hat{x}$$

More info about resized conv and transposed conv : https://distill.pub/2016/deconv-checkerboard/

BatchNorm vs. InstanceNorm



$$x[\ldots, f] \leftarrow \alpha_f \frac{x[\ldots, f] - \mathbb{E}[x[\ldots, f]]}{\sigma(x[\ldots, f])} + \beta_f$$

BatchNorm :
channel-wise
⇒ Discriminative tasks !

$$x[n, \ldots, f] \leftarrow \alpha_f \frac{x[n, \ldots, f] - \mathbb{E}[x[n, \ldots, f]]}{\sigma(x[n, \ldots, f])} + \beta_f$$

InstanceNorm :
(sample,channel)-wise
⇒ Generative tasks !

Conditional Instance Normalization : Add meta-data to your CNNs !

Add conditions on $\{\alpha, \beta\}$ within an Instance Normalization layer :

- Traffic Sign classification : $\{\alpha, \beta\}_{night}$ vs $\{\alpha, \beta\}_{day}$
- SAR images : $\{\alpha, \beta\}(inc\_angle)$

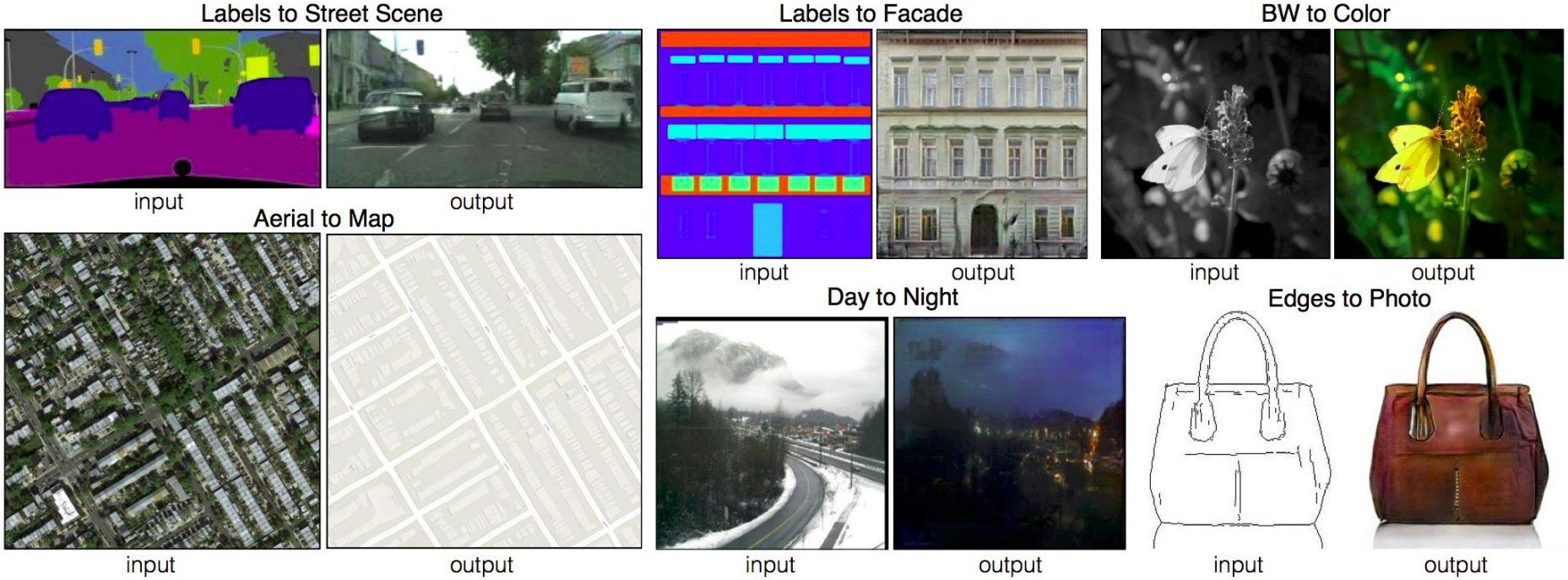
[13] : **Conditional Instance Normalization applied to Style Transfer**

- **64 styles** with 1 generator and 64 sets of normalization parameters
- **direct interpolation** with the learned normalization parameters to create new styles

Some results

Generative Adversarial Networks- GANs

Some “generative” tasks



Source :https://phillipi.github.io/pix2pix/
Paper [17]

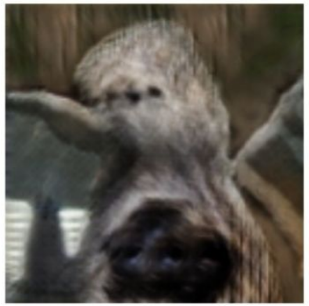
How to generate realistic images ?

Task: given a dataset, generate samples following a distribution similar to the dataset p_{data}

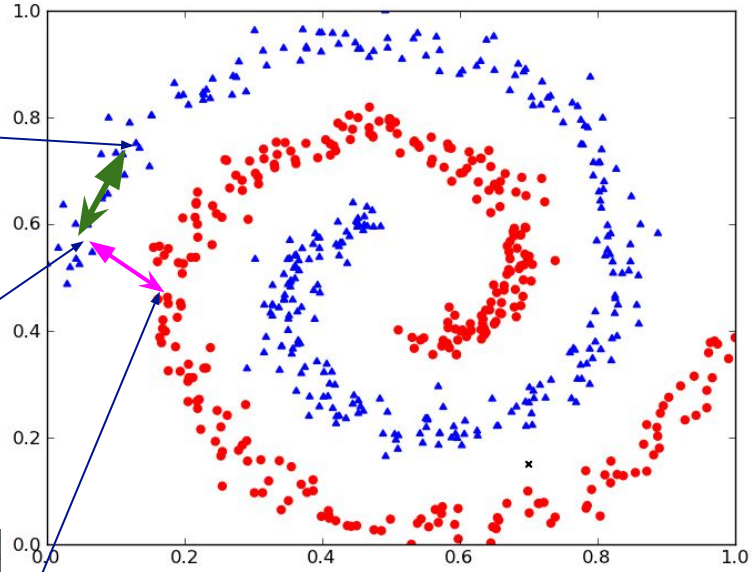
Which loss to use ?

- **MSE** (Mean Squared Error) on image space
- **Total Variation Loss** (impose smoothness)
- **Feature Matching** (MSE on feature maps)
  - **Perceptual loss** (cf Style Transfer)

➡ Blurred images, **non-realistic images**



Find the Manifolds of ‘realistic images’ ?



Ships vs Planes manifolds !

Main issue in Machine Learning :

- How to define a good loss for a given task ??

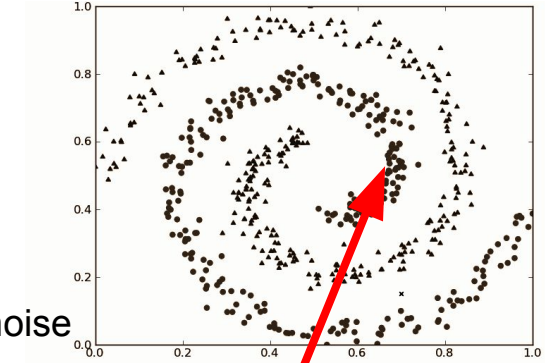
MSE for image generation ?

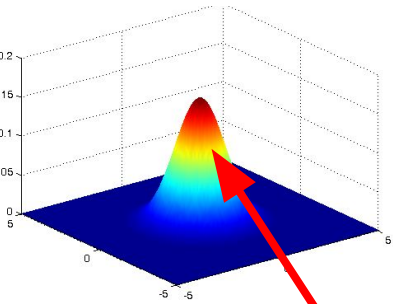
- Does not capture the concepts
- Distance on low-level representation (pixel-level) !

Hard to define a loss that **measure the photorealism** ?

LEARN THIS LOSS WITH NEURAL NETS

How to generate cats : Meow generator !

p_z (distributions)  p_{data}



- ➔ start from a random noise
- ➔ to a realistic image in the manifold of cats !
- ➔ with a ‘mapping’ function

z [100,] (samples) 

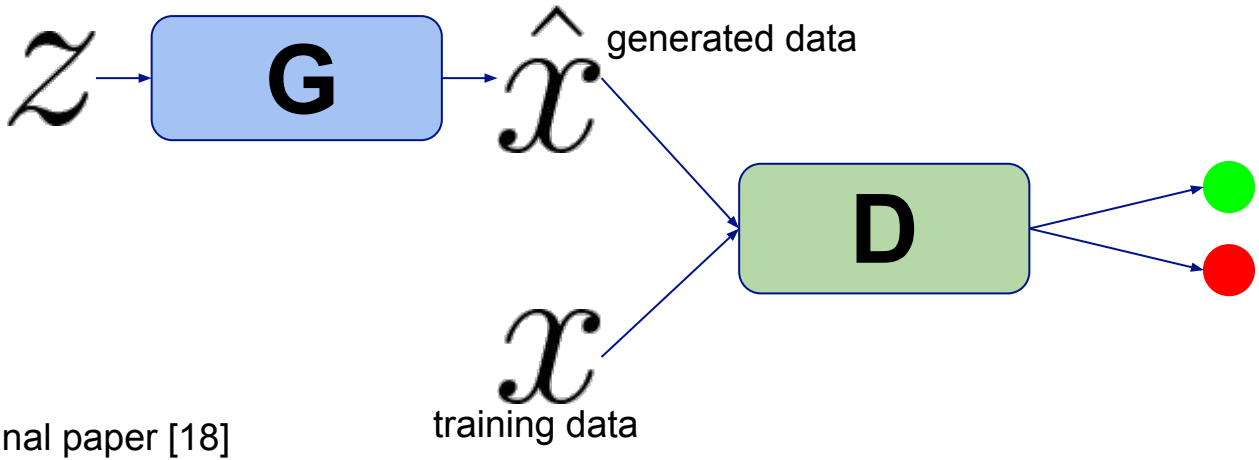
$\hat{x} = G_\theta(z)$

[256, 256, 3]

Generative Adversarial Networks (GANs)

General framework : Generator(G) + Discriminator(D)

- **G : generates data from a latent space (noise)**
- **D : is trained to classify real vs fake data**
- **G : is trained to fool D**



Original paper [18]

GANs in equations

$$\min_G \max_D \left[\mathbb{E}_{x \sim p_{data}}[log(D(x))] + \mathbb{E}_{z \sim p_z}[log(1 - D(G(z)))] \right]$$

min/max game : **Game Theory** - 2 agents : 2 neural networks

- equivalent to minimize the Jensen-Shannon divergence between $(p_{data} || p_\theta)$
- **Nash equilibrium** : $D(x) = \frac{1}{2} \quad \text{and} \quad p_\theta = p_{data}$

- Learn an **implicit distribution**, throught the generator :

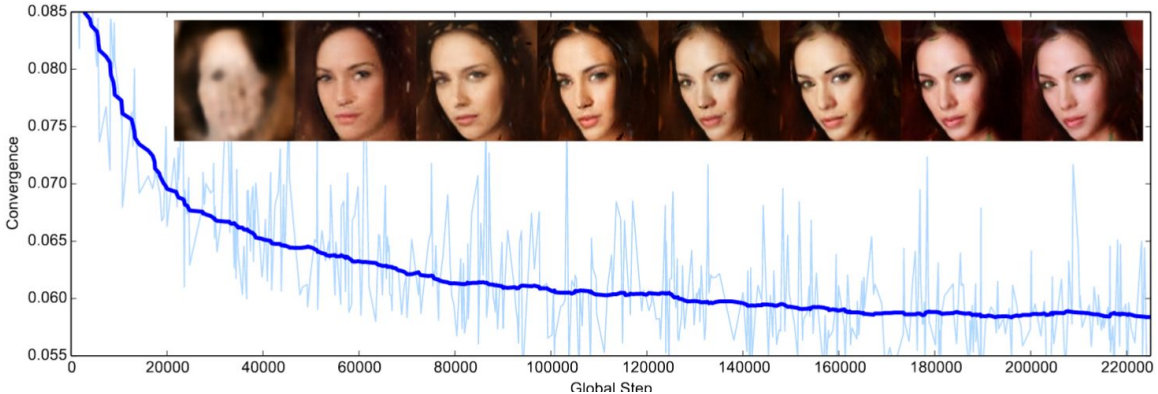
$$z \sim p_z$$
$$x = G_\theta(z) \sim p_\theta$$

In practice : how to train GANs ?

$$L_D = -\mathbb{E}_{x \sim p_{data}}[log(D(x))] - \mathbb{E}_{z \sim p_z}[log(1 - D(G(z)))]$$

$$L_G = -\mathbb{E}_{z \sim p_z}[log(D(G(z)))]$$

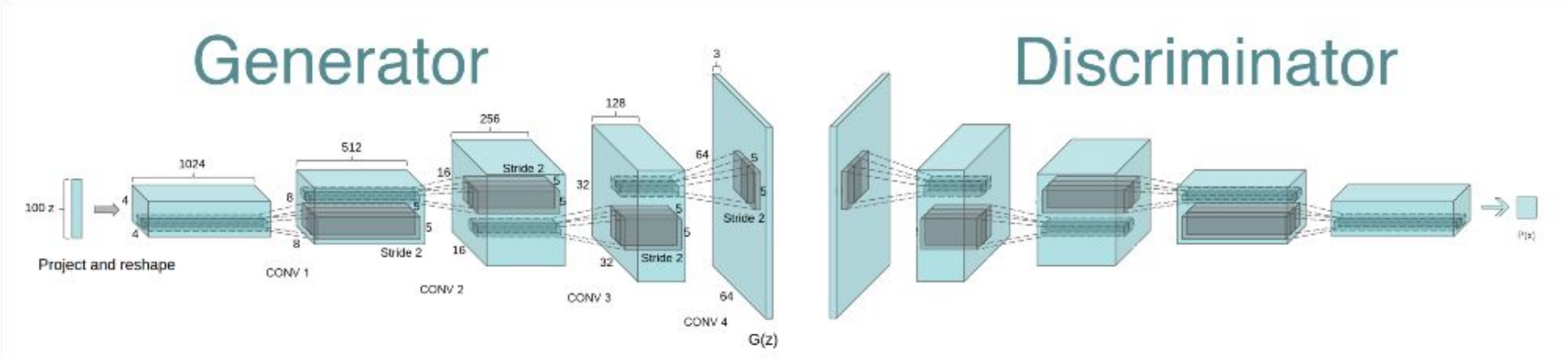
Simultaneously training for D and G:
- train G to fool D with a batch of z
- train D to detect samples from G or from the dataset



Many other to train G and D :
- f-divergence, Wasserstein loss, feature matching, ... see [19, 20](Jan 2018)

Which architectures for G and D ?

Ex : Deep Convolutional GAN -DCGAN[21]

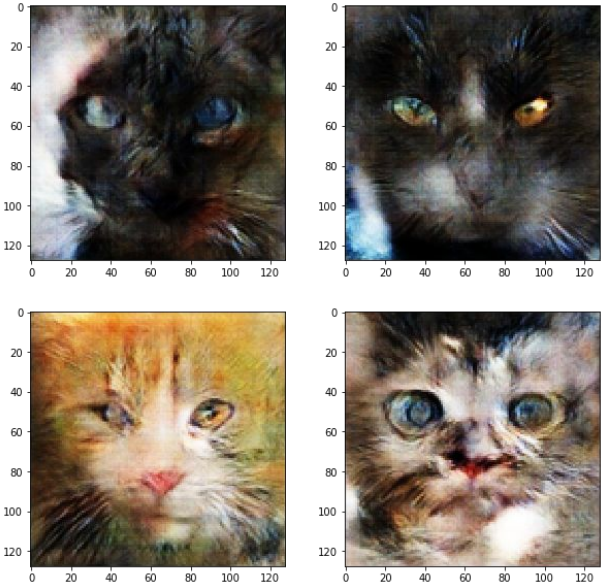


Same improvements as in Style Transfer:

- resized conv > transposed conv
- residual blocks
- several discriminators with random projections [22]

Some results

GAN, LapGAN, DCGAN, BeGAN, BiGAN, DiscoGAN, LSGAN, WGAN, f-GAN, Fisher-GAN, AE-GAN, APE-GAN, Gang of GANs, InfoGAN, CycleGAN, StackedGAN, DualGAN, DeliGAN, …..



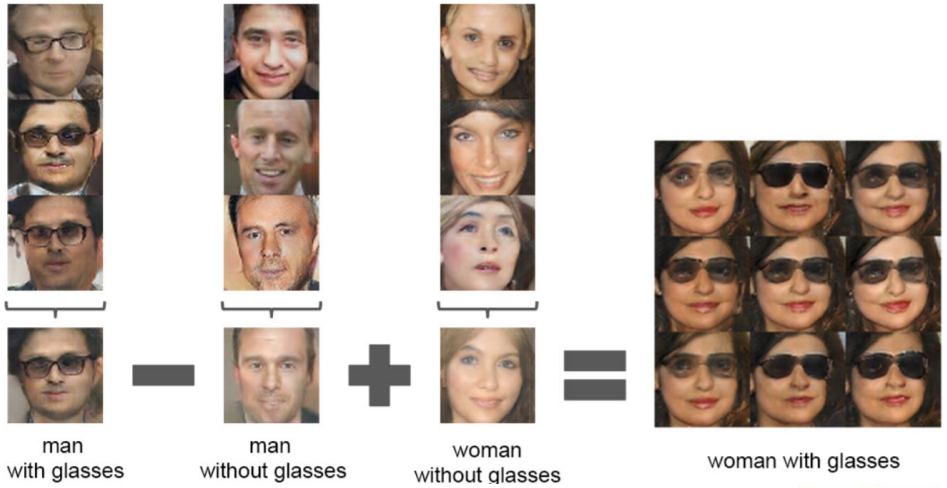
-> Meow generator

Here, results with a DCGAN, trained with `feature matching` loss !

Latent Space understanding (z)

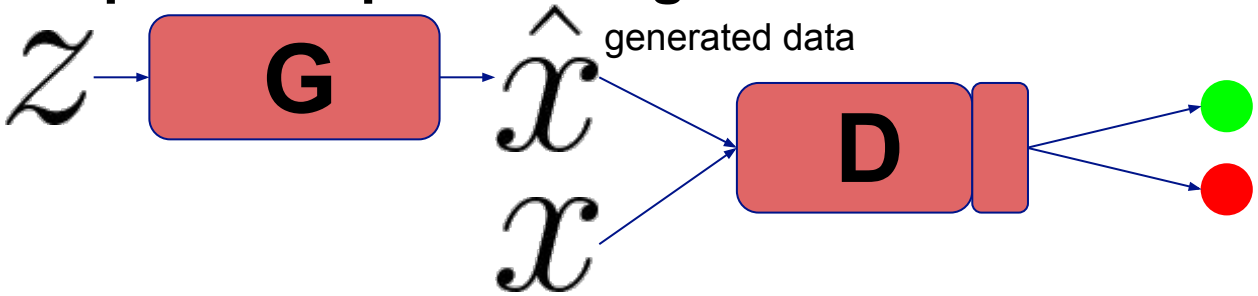
How to get z from a photo :
- **recover z by optimization**
- **learn an encoder z=E(x)** when training D and G
  - BiGAN : GAN + auto-encoder [23]

Arithmetic operation in the latent space :

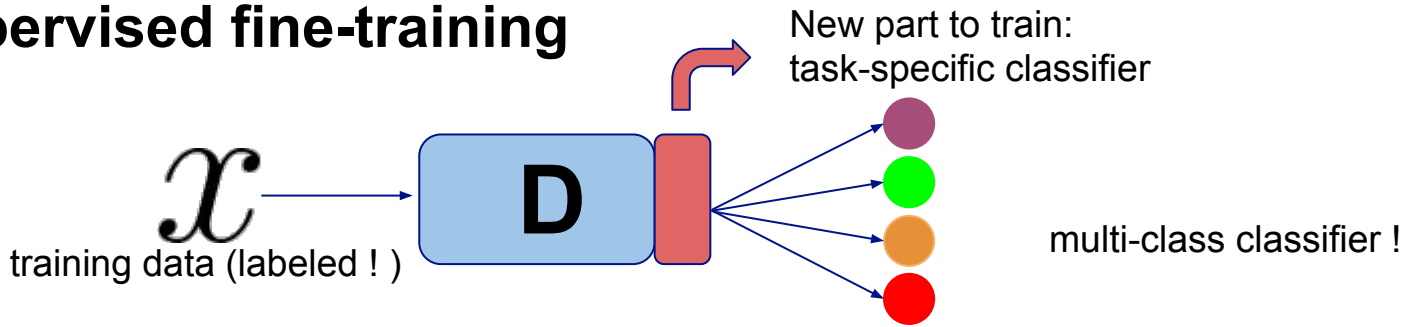


GANs for semi-supervised learning

Unsupervised pre-training



Supervised fine-training



Adversarial Domain Adaptation (1/3)

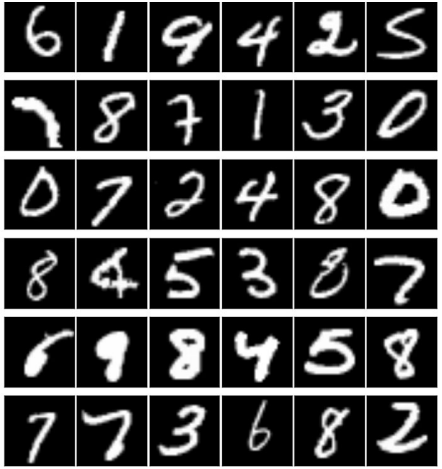
Source domain : SVHN

- with labels

~ 150k samples
10 classes, 32x32 pixels

Target domain : MNIST

- without labels



60k + 10k samples
10 classes, 28x28 pixels

Similar concepts, not the same data source (ex : optical vs SAR images)

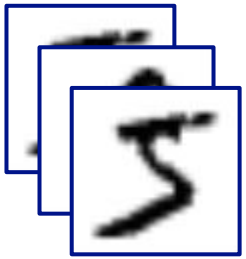
Adversarial Domain Adaptation (2/3)

SVHN CNN

Classifier

Pre-training
- supervised learning
- on source domain
- train ‘SVHN CNN’ + ‘Classifier’

SVHN CNN



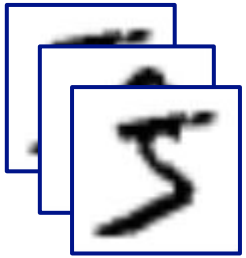
MNIST CNN

Discriminator

Task : binary classification
- features ‘SVHN CNN’
- or from ‘MNIST CNN’ ?

Adversarial Adaptation:
- learn a target encoder CNN (Generator)
- features from ‘MNIST CNN’ will follow the same distribution as the features from ‘SVHN CNN’
- **without labels** from both domains !

Adversarial Domain Adaptation (3/3)



MNIST CNN

Classifier

Testing

- **'Classifier' can understand features from 'MNIST CNN'**
- and make classification

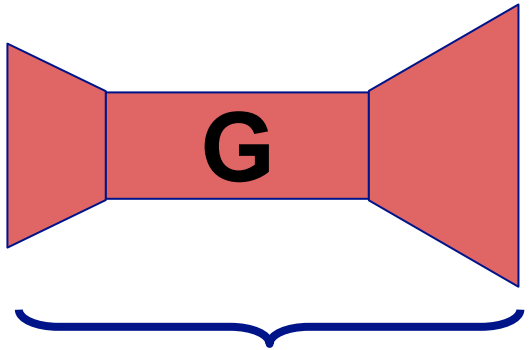
Results :

Method	MNIST → USPS	USPS → MNIST	SVHN → MNIST
Source only	0.752 ± 0.016	0.571 ± 0.017	0.601 ± 0.011
Gradient reversal	0.771 ± 0.018	0.730 ± 0.020	0.739 [16]
Domain confusion	0.791 ± 0.005	0.665 ± 0.033	0.681 ± 0.003
CoGAN	0.912 ± 0.008	0.891 ± 0.008	did not converge
ADDA (Ours)	0.894 ± 0.002	0.901 ± 0.008	0.760 ± 0.018

[24] : Adversarial Discriminative Domain Adaptation, E. Tzeng et al, Feb 2017

Enhance Super-Resolution with GANs (1/3)

LR : [64, 64, 3]



Generator LR->SH
based on residual blocks

SR : [256, 256, 3]
(prediction)

$$l_{MSE}$$

HR : [256, 256, 3]
(groundtruth)

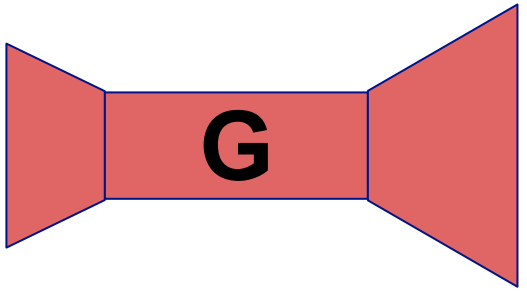
$$l_{MSE} = \frac{1}{WH} \sum_{x=1}^{W} \sum_{y=1}^{H} (G_\theta(I^{LR})_{x,y} - I^{HR}_{x,y})$$

Intuitive loss : Mean Squared Error (MSE)

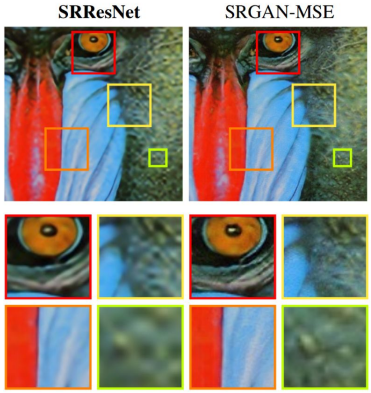
- Blurry images !

Enhance Super-Resolution with GANs (2/3)

LR : [64, 64, 3]



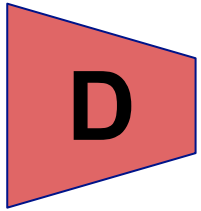
Generator LR->SH
based on residual blocks



SR : [256, 256, 3]
(prediction)

l_{MSE}

HR : [256, 256, 3]
(groundtruth)

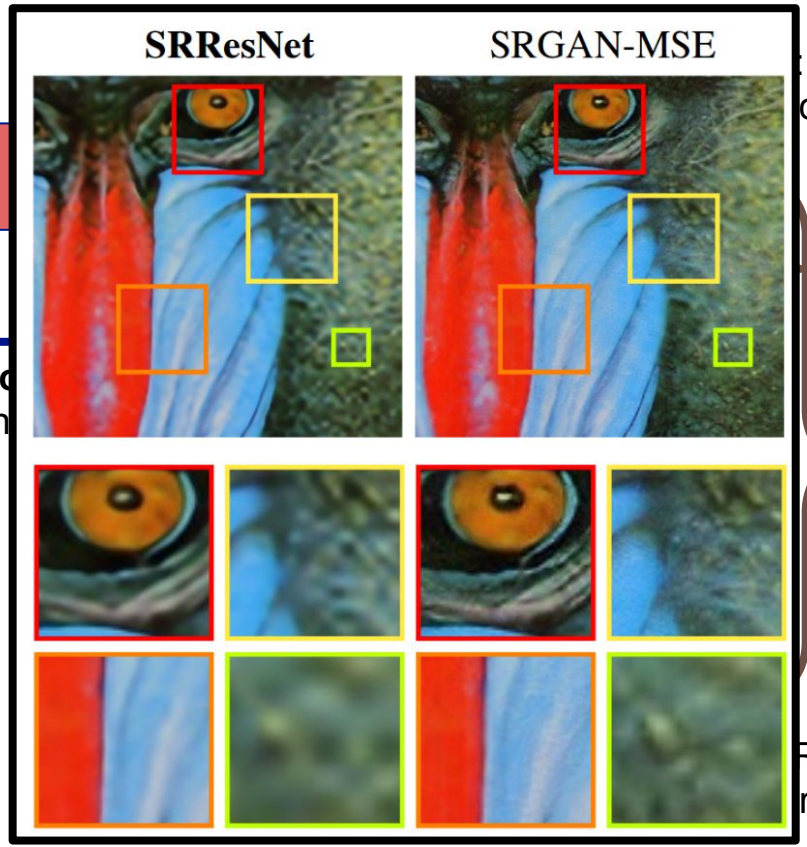


real or fake ?
(HR vs SR)

Enhance Super-Resolution with GANs (3/3)

LR : [64, 64, 3]

Generato
based on



[256, 256, 3]
diction)

l_{MSE}

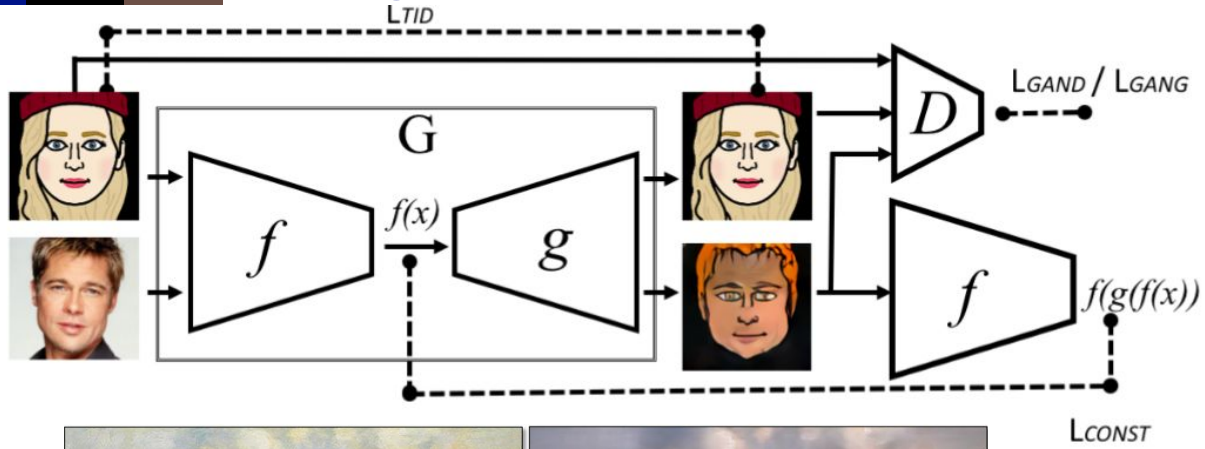
D

real or fake ?
(HR vs SR)

R : [256, 256, 3]
roundtruth)

approach from [25]

Many applications of GANs …



Cross-domain image generation [26] (FAIR)

Reverse style transfer with CycleGAN [27]

paper [28] : “High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs”, Nvidia, Dec 2017
demo :https://www.youtube.com/watch?v=3AIpPlzM_qs

Thanks !

References (1/2)

[1] : K. Simonyan, A. Zisserman : “Very Deep Convolutional Networks for Large-Scale Image Recognition”, 2014, arXiv:1409.1556

[2] : C Szegedy et al. : “Going Deeper with Convolutions”, 2014, arxiv:1409.4842

[3] : K He et al : “Deep Residual Learning for Image Recognition”, 2015, arxiv.org:1512.03385

[4] : ImageNet dataset : http://www.image-net.org/

[5] : About Deep Dream visualization technique : “Inceptionism: Going Deeper into Neural Networks”

[6] : M. Zeiler, R. Fergus: Visualizing and Understanding Convolutional Networks, 2013 arXiv:1311.2901

[7] : L. Gatys, A. Ecker, M. Bethge : A neural algorithm of artistic style, 2015, arXiv:1508.06576

[8] : M. Ruder, A. Dosovitskiy, T. Brox : Artistic style transfer for video, 2016, arXiv:1604.08610

[9] : Gatys et al : Preserving color in Neural Artistic Style Transfer, 2016, arXiv:1606.05897

[10] : J. Johnson et al : “Perceptual losses for real-time style transfer and super-resolution”, 2016, arXiv:1603.08155

[11] : D. Ulyanov et al : “Instance Normalization: The Missing Ingredient for Fast Stylization”, 2016, arXiv:1607.08022

[12] : MS-COCO dataset : http://cocodataset.org/#home

[13] : V. Dumoulin et al : “A learned representation for artistic style”, 2017, arXiv:1610.07629

[14] : A Aitken et al : “Checkerboard artifact free sub-pixel convolution”, 2017, arxiv.org:1707.02937

[15] : X. Huang and S. Belongie : “Arbitrary Style Transfer in real-time with AdaIN”, 2017, arXiv:1703.06868

[16] : Y Li et al : “Universal Style Transfer via Feature Transforms”, 2017, arxiv:1705.08086

References (2/2)

[17] : P Isola et al : “Image-to-Image Translation with Conditional Adversarial Networks”, 2016, arxiv:1611.07004

[18] : I Goodfellow et al : “Generative Adversarial Networks”, 2014, arxiv:1406.2661

[19] : Y Hong et al : “How GANs and its variants work : an overview of GAN”, 2017, arxiv:1711.05914v6

[20] : S Hitawala : “Comparative Study on GANs”, 2018, arxiv:1801.04271v1

[21] : A Radford et al : “Unsupervised Representation Learning with Deep Convolutional GANs”, 2015, arxiv:511.06434

[22] : B Neyshabur et al : “Stabilizing GAN Training with Multiple Random Projections”, 2017, arxiv:1707.02937

[23] : J Donahue et al : “Adversarial Feature Learning” , 2016, arxiv:1605.09782

[24] : Eric Tzeng et al : “Adversarial Discriminative Domain Adaptation”, 2017, arxiv:1702.05464

[25] : C Ledig et al : “Photo-realistic Single Image Super-Resolution using GANs” 2016, arxiv:1609.04802

[26] : Y Taigman et al : “Unsupervised Cross-Domain Image Generation”, 2016, arxiv:1611.02200

[27] : J-Y Zhu et al : “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks”, 2017, arxiv:1703.10593

[28] : T-C Wang et al (NVIDIA) : “High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs”, Dec 2017, arxiv:1711.11585