ECE 285

Machine Learning for Image Processing

Chapter VI – Generation, super-resolution and style transfer

Charles Deledalle

November 16, 2019

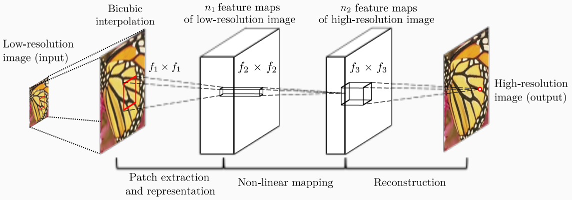
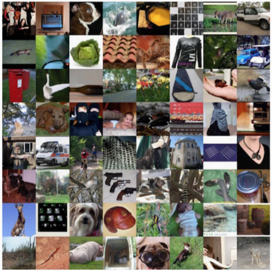


Image generation

Motivations – Image generation

- **Goal:** Generate images that look like the ones of your training set.



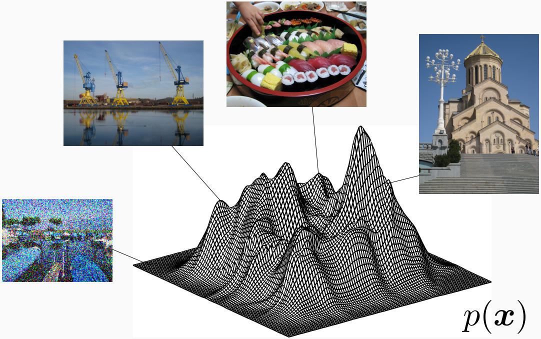
Real images (ImageNet)
(Training set)

Generated images
(Results)

- **What?** Unsupervised learning.
- **Why?** Different reasons and applications:
  - Can be used for simulation, *e.g.*, to generate labeled datasets,
  - Must capture all subtle patterns $\rightarrow$ provide good features,
  - Can be used for other tasks: super-resolution, style transfer, ...

Image generation – Explicit density

1. Learn the distribution of images $p(\boldsymbol{x})$ on a training set.



2. Generate samples from this distribution.

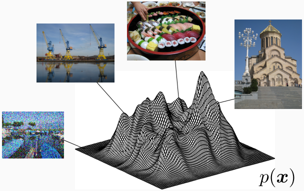
Image generation – Gaussian model

- Consider a Gaussian model for the distribution of images $\boldsymbol{x}$ with n pixels:

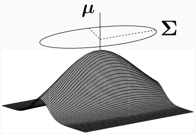
$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

$$p(\boldsymbol{x}) = \frac{1}{\sqrt{2\pi}^n |\boldsymbol{\Sigma}|^{1/2}} \exp\left[(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\right]$$

- $\boldsymbol{\mu}$: mean image,
- $\boldsymbol{\Sigma}$: covariance matrix of images.



$\approx$



Gaussian prior $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$

Image generation – Gaussian model

- Take a training dataset $\mathcal{T}$ of images:

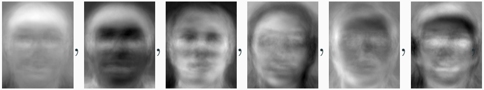
$$\mathcal{T} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$$

$= \Big\{$  $_{\times N} \; \ldots \Big\}$

- Estimate the mean

$$\hat{\boldsymbol{\mu}} = \frac{1}{N} \sum_i \boldsymbol{x}_i =$$

- Estimate the covariance matrix: $\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_i (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})^T = \hat{\boldsymbol{E}} \hat{\boldsymbol{\Lambda}} \hat{\boldsymbol{E}}^T$

$\hat{\boldsymbol{E}} = \Big\{$  $_{\times N} \; \ldots \Big\}$

eigenvectors of $\hat{\boldsymbol{\Sigma}}$, *i.e.*, main variation axis

Image generation – Gaussian model

You now have learned a generative model:

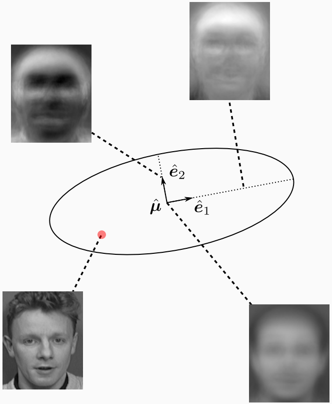


Image generation – Gaussian model

How to generate samples from $\mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})$?

$$\begin{cases} \boldsymbol{z} & \sim \mathcal{N}(0, \mathrm{Id}_n) \\ \boldsymbol{x} & = \hat{\boldsymbol{\mu}} + \hat{\boldsymbol{E}}\hat{\boldsymbol{\Lambda}}^{1/2}\boldsymbol{z} \end{cases}$$ ← Generate random latent variable

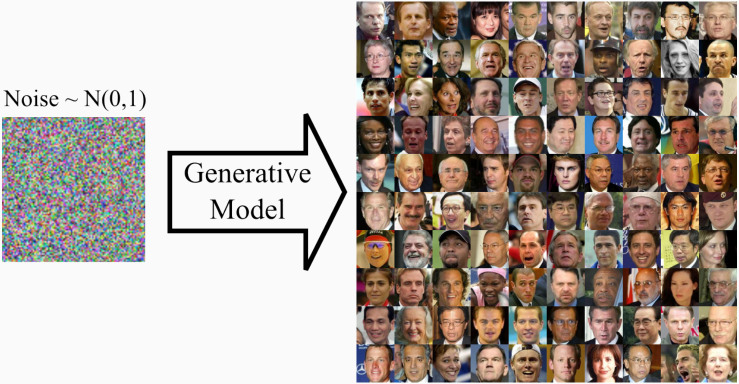
The model does not generate realistic faces.

The Gaussian distribution assumption is too simplistic.

Each generated image is just a linear random combination of the eigenvectors.

The generator corresponds to a linear neural network (without non-linearities).

Image generation – Beyond Gaussian models

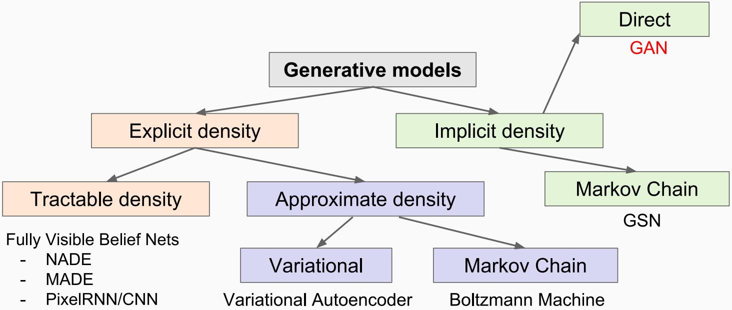


But the concept is interesting: can we find a transformation such that each random code can be mapped to a photo-realistic image?

We need to find a way to assess if an image is photo-realistic.

Image generation – Beyond Gaussian models

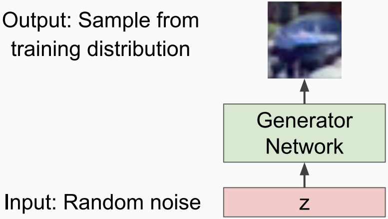
Taxonomy of Generative Models



Generative Adversarial Networks

(Goodfellow et al., NIPS 2014)

- **Goal:** design a complex model with high capacity able to map latent random noise vectors z to a realistic image x.
- **Idea:** Take a deep neural network

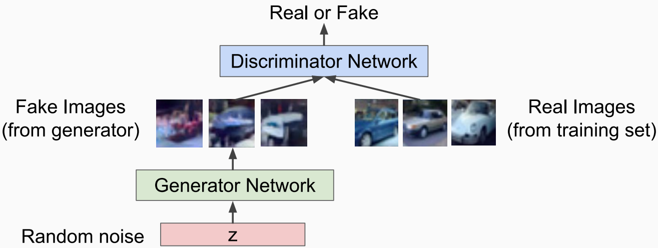


- **What about the loss?** Measure if the generated image is photo-realistic.

Generative Adversarial Networks

(Goodfellow et al., NIPS 2014)

Define a loss measuring how much you can fool a classifier that has learned to distinguish between real and fake images.



- **Discriminator network:** try to distinguish between real and fake images.
- **Generator network:** fool the discriminator by generating realistic images.

Generative Adversarial Networks

(Goodfellow et al., NIPS 2014)

- **Discriminator network:** Consider two sets
  - $\mathcal{T}_{\text{real}}$: a dataset of n real images,
  - $\mathcal{T}_{\text{fake}}$: a dataset of m fake images.

- **Goal:** find the parameters θ_d of a binary classification network $\boldsymbol{x} \mapsto D_{\theta_d}(\boldsymbol{x})$ meant to classify real and fake images.
  Minimize the cross entropy, or maximize its negation

$$\max_{\theta_d} \underbrace{\frac{1}{n} \sum_{\boldsymbol{x} \in \mathcal{T}_{\text{real}}} \log D_{\theta_d}(\boldsymbol{x})}_{\substack{\text{force predicted labels to be 1} \\ \text{for real images}}} + \underbrace{\frac{1}{m} \sum_{\boldsymbol{x} \in \mathcal{T}_{\text{fake}}} \log(1 - D_{\theta_d}(\boldsymbol{x}))}_{\substack{\text{force predicted labels to be 0} \\ \text{for fake images}}}$$

- **How:** use gradient ascent with backprop (+SGD, batch-normalization. . .).

Generative Adversarial Networks

(Goodfellow et al., NIPS 2014)

- **Generator network:** Consider a given discriminative model $\boldsymbol{x} \mapsto D_{\theta_d}(\boldsymbol{x})$ and consider $\mathcal{T}_{\text{rand}}$ a set of m random latent vectors.
- **Goal:** find the parameters θ_g of a network $\boldsymbol{x} \mapsto G_{\theta_g}(\boldsymbol{z})$ generating images from random vectors $\boldsymbol{z}$ such that it fools the discriminator

$$\min_{\theta_g} \quad \underbrace{\frac{1}{m} \sum_{\boldsymbol{z} \in \mathcal{T}_{\text{rand}}} \log(1 - D_{\theta_d}(G_{\theta_g}(\boldsymbol{z})))}_{\substack{\text{force the discriminator to think that} \\ \text{our generated fake images are not fake (away from 0)}}} \tag{1}$$

or alternatively (works better in practice)

$$\max_{\theta_g} \quad \underbrace{\frac{1}{m} \sum_{\boldsymbol{z} \in \mathcal{T}_{\text{rand}}} \log D_{\theta_d}(G_{\theta_g}(\boldsymbol{z})))}_{\substack{\text{force the discriminator to think that} \\ \text{our generated fake images are real (close to 1)}}} \tag{2}$$

- **How:** gradient descent for (1) or gradient ascent for (2) with backprop...

Generative Adversarial Networks

(Goodfellow et al., NIPS 2014)

- **Train both networks jointly**.
- Minimax loss in a two player game (each player is a network):

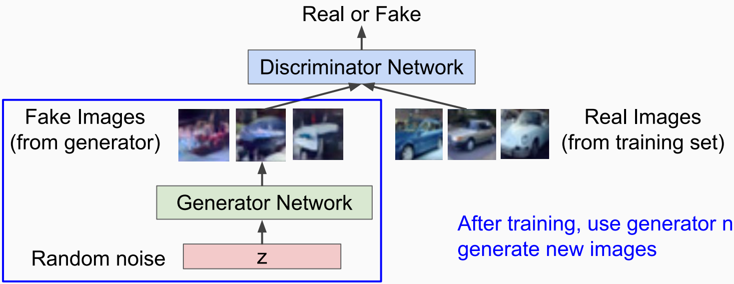
$$\min_{\theta_g} \max_{\theta_d} \frac{1}{n} \sum_{\boldsymbol{x} \in \mathcal{T}_{\text{real}}} \log D_{\theta_d}(\boldsymbol{x}) + \frac{1}{m} \sum_{\boldsymbol{z} \in \mathcal{T}_{\text{rand}}} \log(1 - D_{\theta_d}(\underbrace{G_{\theta_g}(\boldsymbol{z})}_{\text{fake}}))$$

- **Algorithm:** repeat until convergence
  1. Fix θ_g, update θ_d with one step of gradient ascent,
  2. Fix θ_d, update θ_g with one step of gradient descent for (1),

     (or one step of gradient ascent for (2).)

Generative Adversarial Networks

(Goodfellow et al., NIPS 2014)



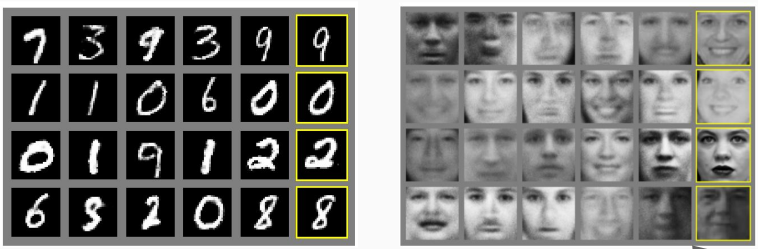
After training, use generator network to generate new images

Image generation

Generative Adversarial Networks

(Goodfellow et al., NIPS 2014)

Generated samples

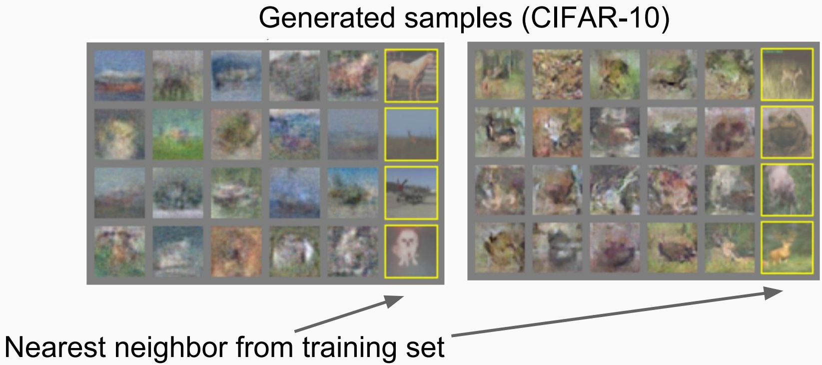


Nearest neighbor from training set

Image generation

Generative Adversarial Networks

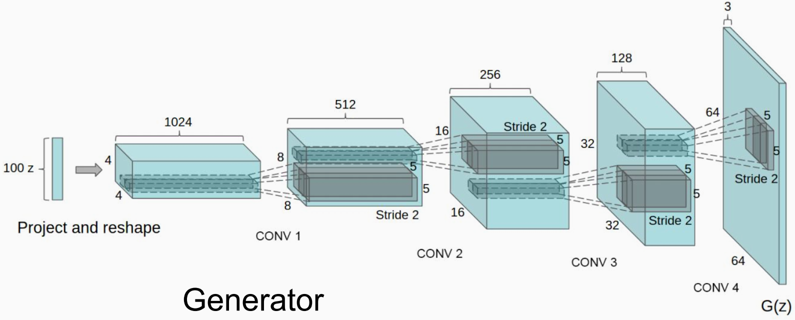
(Goodfellow et al., NIPS 2014)



Convolutional GAN

(Radford et al., 2016)

- **Generator:** upsampling network with fractionally strided convolutions,
- **Discriminator:** convolutional network with strided convolutions.



Convolutional GAN

(Radford et al., 2016)

Generations of realistic bedrooms pictures, from randomly generated latent variables.

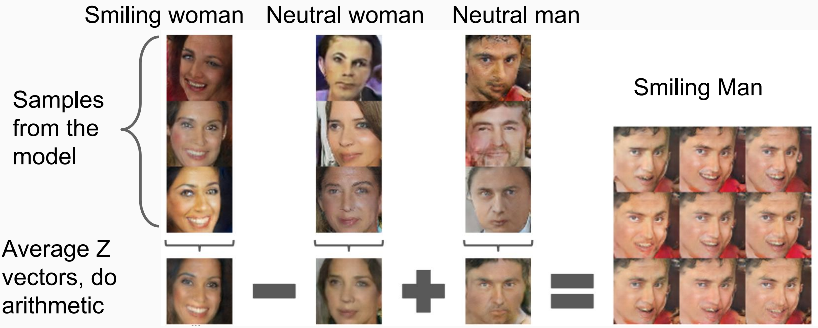
Convolutional GAN

(Radford et al., 2016)

Interpolation in between points in latent space.

Convolutional GAN – Arithmetic

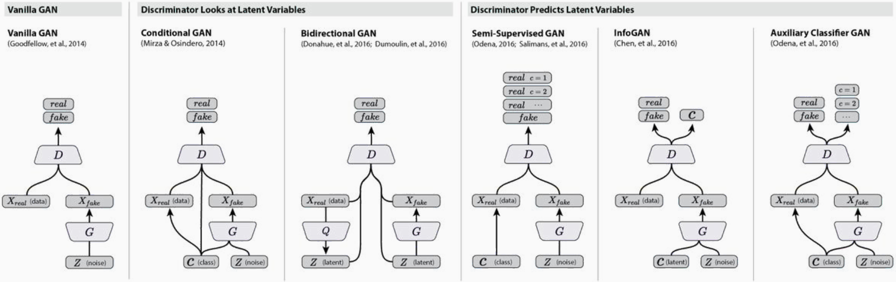
(Radford et al., 2016)



Convolutional GAN – Arithmetic

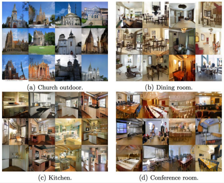
(Radford et al., 2016)

GAN Sub-classification



2017: Year of the GAN

Better training and generation



LSGAN. Mao et al. 2017.

BEGAN. Bertholet et al. 2017.

Source->Target domain transfer

CycleGAN. Zhu et al. 2017.

Text -> Image Synthesis

Reed et al. 2017.

Many GAN applications

Pix2pix. Isola 2017. Many examples at https://phillipi.github.io/pix2pix/

Super-resolution

Super-resolution

Super-resolution

Low-resolution (LR) image

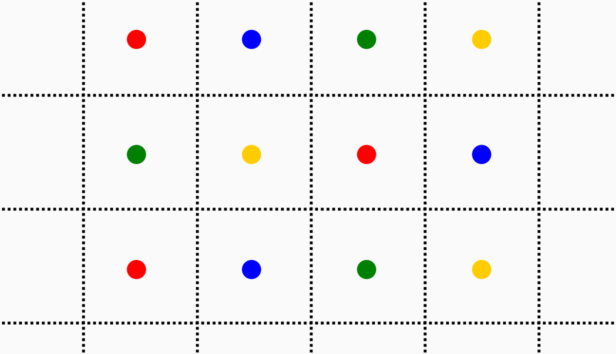
$\xrightarrow{\text{Super-resolution (SR)}}$

High-resolution (HR) image

Goal: Create a high-resolution (HR) image from a low-resolution (LR) image.

Super-resolution – Interpolation

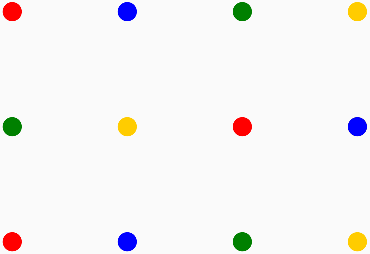
Approach 1: Consider the SR problem as a 2d interpolation problem.



Look at the pixel values of the original LR image on its LR grid.

Super-resolution – Interpolation

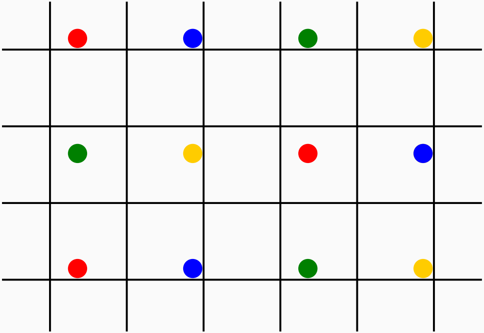
Approach 1: Consider the SR problem as a 2d interpolation problem.



Forget about the LR grid and consider each pixel as a 2d point.

Super-resolution – Interpolation

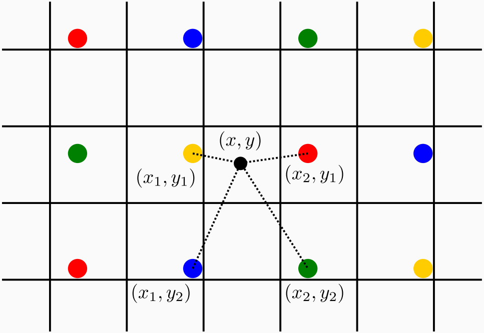
Approach 1: Consider the SR problem as a 2d interpolation problem.



Inject the targeted HR grid.

Super-resolution – Interpolation

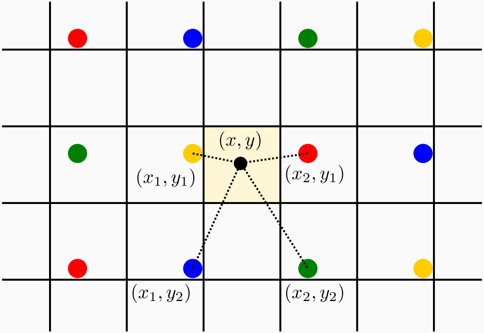
Approach 1: Consider the SR problem as a 2d interpolation problem.



Deduce HR pixel values based on LR points.

Super-resolution – Nearest neighbor interpolation

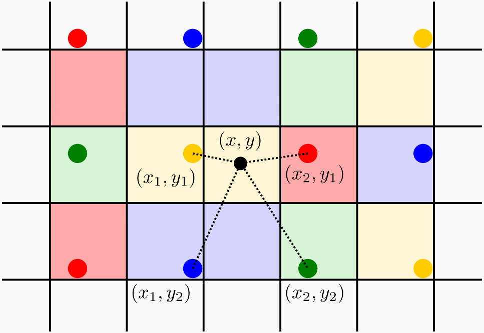
Nearest neighbor: affect the pixel value of the closest LR point.



$$I^{\mathsf{SR}}(x, y) = I^{\mathsf{LR}}(x_{k^\star}, y_{l^\star}) \quad \text{where} \quad (k^\star, l^\star) = \underset{(k,l)}{\operatorname{argmin}} \, (x_k - x)^2 + (y_l - y)^2$$

Super-resolution – Nearest neighbor interpolation

Nearest neighbor: affect the pixel value of the closest LR point.

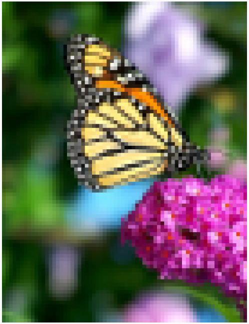


$$I^{\mathsf{SR}}(x, y) = I^{\mathsf{LR}}(x_{k^\star}, y_{l^\star}) \quad \text{where} \quad (k^\star, l^\star) = \underset{(k,l)}{\operatorname{argmin}}\ (x_k - x)^2 + (y_l - y)^2$$

Super-resolution – Nearest neighbor interpolation

LR image

→



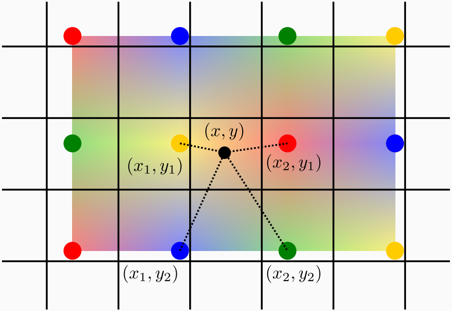
Obtained HR image

v.s.

Targeted HR image

Problem: pixels are independently copied into a juxtaposition of large rectangular block of pixels.

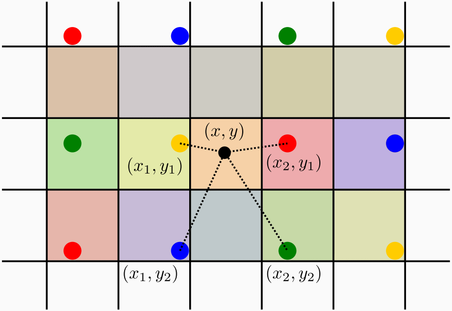
Super-resolution – Bi-linear interpolation



Bi-linear: combine pixel values of LR points with respect to their distance to the HR point.

$$I^{\mathsf{SR}}(x,y) = \frac{y_2 - y}{y_2 - y_1} \underbrace{\left(\frac{x_2 - x}{x_2 - x_1} I^{\mathsf{LR}}(x_1, y_1) + \frac{x - x_1}{x_2 - x_1} I^{\mathsf{LR}}(x_2, y_1) \right)}_{\text{linear interp. for } x \text{ with } y = y_1}$$
$$+ \frac{y - y_1}{y_2 - y_1} \underbrace{\left(\frac{x_2 - x}{x_2 - x_1} I^{\mathsf{LR}}(x_1, y_2) + \frac{x - x_1}{x_2 - x_1} I^{\mathsf{LR}}(x_2, y_2) \right)}_{\text{linear interp. for } x \text{ with } y = y_2}$$

Super-resolution – Bi-linear interpolation



Bi-linear: combine pixel values of LR points with respect to their distance to the HR point.

$$I^{\mathsf{SR}}(x,y) = \frac{y_2 - y}{y_2 - y_1} \underbrace{\left(\frac{x_2 - x}{x_2 - x_1} I^{\mathsf{LR}}(x_1, y_1) + \frac{x - x_1}{x_2 - x_1} I^{\mathsf{LR}}(x_2, y_1) \right)}_{\text{linear interp. for } x \text{ with } y = y_1}$$

$$+ \frac{y - y_1}{y_2 - y_1} \underbrace{\left(\frac{x_2 - x}{x_2 - x_1} I^{\mathsf{LR}}(x_1, y_2) + \frac{x - x_1}{x_2 - x_1} I^{\mathsf{LR}}(x_2, y_2) \right)}_{\text{linear interp. for } x \text{ with } y = y_2}$$

Super-resolution – Bi-linear / Bi-cubic interpolation

Bi-linear interpolation

- Interpolate the 4 points by finding the coefficients a, b, c and d of

$$f(x, y) = ax + by + cxy + d$$

- 4 unknowns and 4 (independent) equations $\Rightarrow$ unique solution.

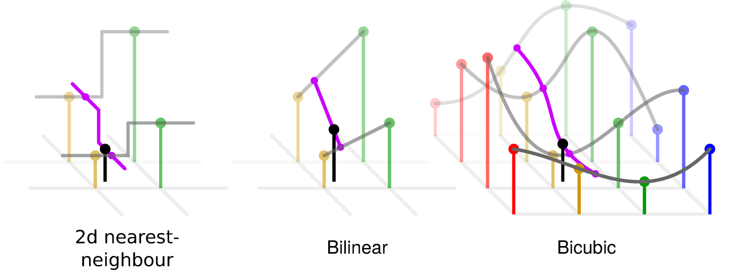
Bi-cubic interpolation

- Same but with 16 coefficients a_{ij}, $0 \leqslant i, j \leqslant 3$, of

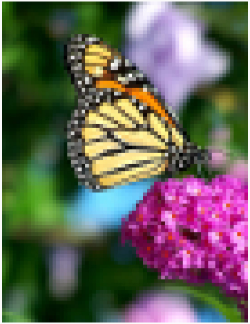
$$f(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j$$

- 16 unknowns and 4 equations $\rightarrow$ infinite number of solutions,
- Interpolate the derivatives: 4 in x + 4 in y + 4 in xy $\rightarrow$ 16 equations.
- Closed-form solution obtained by inverting a 16×16 matrix.

Super-resolution – Bi-linear / Bi-cubic interpolation



Super-resolution – Bi-linear / Bi-cubic interpolation



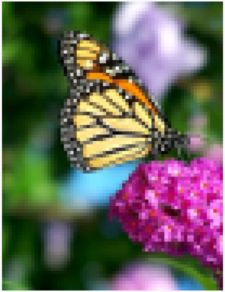
Nearest neighbor

Bi-linear

Bi-cubic

Problem: bi-cubic interpolation is a bit better, but the image still appears blurry/blobby, it is missing sharp content.

Super-resolution – Interpolation + Sharpening

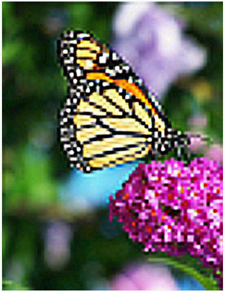


Bi-cubic

$+ \alpha$

Details

$=$



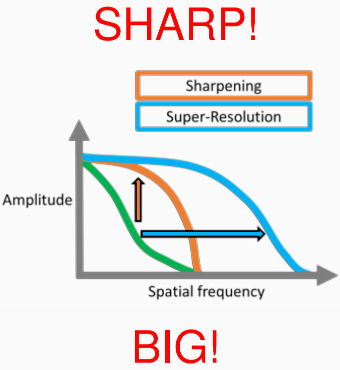
Sharpening

Naive sharpening:

- extract details (high-pass filter),
- amplify them by $\alpha > 0$,
- add them to the original image.

More contrast but still blocky artifacts and lacking fine details.

Super-resolution – Image sharpening



- **Sharpening**: amplifies existing frequencies (visible details).
- **Super-resolution**: retrieves missing high-frequencies (lost details).

(Source: Taegyun Jeon)

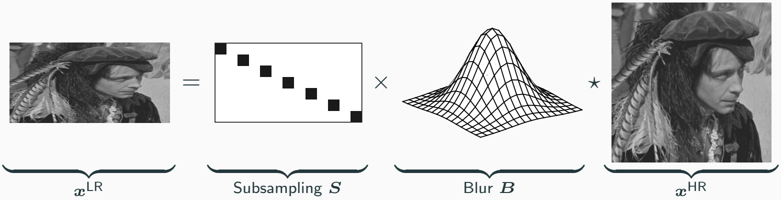
Super-resolution – Linear inverse problem

Approach 2: Model SR as a linear inverse problem

- Linear model based on the characteristics of the camera

$$\boldsymbol{x}^{\text{LR}} = \underbrace{\boldsymbol{S}}_{\text{sub-sampling}} \underbrace{\boldsymbol{B}}_{\text{blur}} \boldsymbol{x}^{\text{HR}} = \underbrace{\boldsymbol{H}}_{\text{both}} \boldsymbol{x}^{\text{HR}}, \quad \boldsymbol{H} = \boldsymbol{S}\boldsymbol{B}$$

- **Blur:** convolution with the point spread function of your digital camera,
- **Sub-sampling:** depends on the targeted HR and the intrinsic resolution of your digital camera (number of photo receptors, cutting frequency, . . .)



Super-resolution – Linear inverse problem

- SR is then a problem of solving a linear system of equations

$$\boldsymbol{x}^{\text{LR}} = \boldsymbol{H}\boldsymbol{x}^{\text{HR}} \quad \Leftrightarrow \quad \begin{cases} h_{11}x_1^{\text{HR}} + h_{12}x_2^{\text{HR}} + \ldots + h_{1n}x_n^{\text{HR}} & = x_1^{\text{LR}} \\ h_{21}x_1^{\text{HR}} + h_{22}x_2^{\text{HR}} + \ldots + h_{2n}x_n^{\text{HR}} & = x_2^{\text{LR}} \\ \vdots & \\ h_{n1}x_1^{\text{HR}} + h_{n2}x_2^{\text{HR}} + \ldots + h_{nn}x_n^{\text{HR}} & = x_n^{\text{LR}} \end{cases}$$

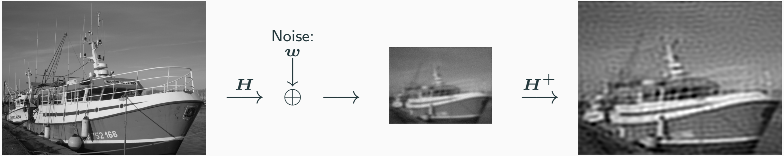
- Retrieving $\boldsymbol{x}^{\text{HR}} \Rightarrow$ Inverting $\boldsymbol{H}$.

- But, $\boldsymbol{H} = \boldsymbol{S}\boldsymbol{B}$ is not invertible $\to$ the problem is said to be ill-posed.
- There are more unknowns (#HR pixels) than equations (#LR pixels),
- Infinite number of solutions satisfying the normal equation:

$$\boldsymbol{x}^{\text{HR}*} \in \underset{\boldsymbol{x}^{\text{HR}}}{\operatorname{argmin}} \ \|\boldsymbol{H}\boldsymbol{x}^{\text{HR}} - \boldsymbol{x}^{\text{LR}}\|_2^2 \quad \Leftrightarrow \quad \boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{x}^{\text{HR}*} - \boldsymbol{x}^{\text{LR}}) = 0$$

Super-resolution – Linear inverse problem

- The solution of minimum norm is given by: $\boldsymbol{x}^{\text{HR}+} = \boldsymbol{H}^{+}\boldsymbol{x}^{\text{LR}}$ where $\boldsymbol{H}^{+}$ is the Moore-Penrose pseudo-inverse.
- But, as the problem is ill-posed:
  - small perturbations in $\boldsymbol{x}^{\text{LR}}$ lead to large errors in $\boldsymbol{x}^{\text{HR}+}$,
  - and unfortunately $\boldsymbol{x}^{\text{LR}}$ is often corrupted by noise,
  - or $\boldsymbol{x}^{\text{LR}}$ is quantized/compressed/encoded with limited precision.



(a) HR image $\boldsymbol{x}^{\text{HR}}$ (b) Noisy LR $\boldsymbol{x}^{\text{HR}}$ (c) Pinv $\boldsymbol{x}^{\text{HT}+}$

The pseudo-inverse solution is similar to image sharpening: (over)amplifies existing frequencies but does not reconstruct missing ones.

Super-resolution – Linear inverse problem

Regularized inverse problems

- **Idea:** look for approximations instead of interpolations by penalizing irregular solutions:

$$\boldsymbol{x}^{\text{HR-R}} \in \underset{\boldsymbol{x}^{\text{HR}}}{\operatorname{argmin}} \ \|\boldsymbol{H}\boldsymbol{x}^{\text{HR}} - \boldsymbol{x}^{\text{LR}}\|_2^2 + \tau R(\boldsymbol{x}^{\text{HR}})$$

- $R(\boldsymbol{x}^{\text{HR}})$ regularization term penalizing large oscillations:
  - Tikhonov regularization: $R(\boldsymbol{x}) = \|\nabla \boldsymbol{x}\|_2^2$ (1943)
    $\rightarrow$ convex optimization problem: closed-form expression.
    $\rightarrow$ remove unwanted oscillations, but blurry (similar to bi-cubic).
  - Total-Variation: $R(\boldsymbol{x}) = \|\nabla \boldsymbol{x}\|_1$ (Rudin *et al.*, 1992)
    $\rightarrow$ convex optimization problem: gradient descent like techniques.
    $\rightarrow$ smooth with sharp edges (recover some high frequencies).
- $\tau > 0$ regularization parameter.

Super-resolution – Linear inverse problem

(a) LR image

(b) Tiny $\tau \sim$ pinv (c) Small τ (d) Good τ (e) High τ (f) Huge τ

Tikhonov regularization for $\times 4$ upsampling (16 times more pixels)

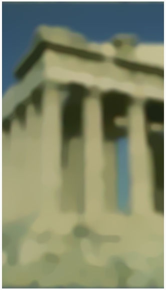
Super-resolution – Linear inverse problem

(a) LR image

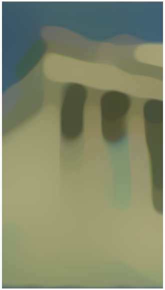
(b) Tiny $\tau \sim$ pinv

(c) Small τ

(d) Good τ



(e) High τ



(f) Huge τ

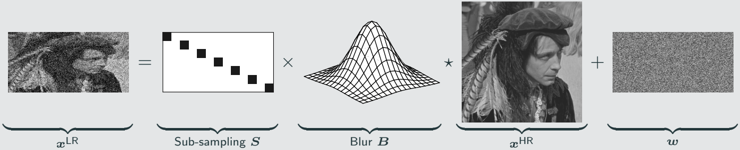
Total-Variation regularization for $\times 4$ upsampling (16 times more pixels)

Super-resolution – Linear inverse problem

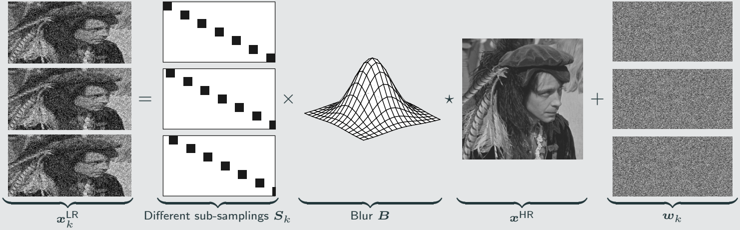
- Standard approach since 1949 (Wiener deconvolution) until 2015.
- Pros:
  - Based on strong mathematical theory,
  - Properties of solutions have been well studied,
  - Convex optimization.
- Cons:
  - Based on hand-crafted regularization priors,
  - Unable to model correctly the subtle patterns of natural images,
  - Results are often blobby and not photo-realistic,
  - Require to model correctly the subsampling $\boldsymbol{S}$ and blur $\boldsymbol{B}$,
  - Slow and not available in standard image processing toolboxes.
- Still relevant for multi-frame super-resolution
  (since with multiple frames the problem becomes well-posed).

Super-resolution – Single vs Multi-frame

Single-frame super-resolution (sub-sampling + convolution + noise)



Multi-frame super-resolution (different sub-pixel shifts + noise)



With several frames: more equations than unknowns.

Super-Resolution CNN (SRCNN) (Dong *et al.*, 2016)

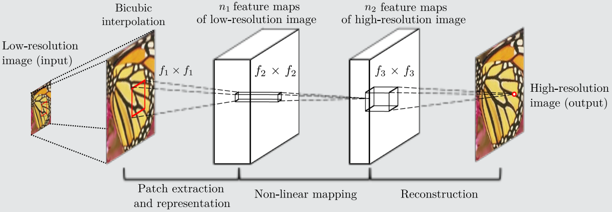
Approach 3: learn to map LR to HR images using a dataset of HR images.

- **Training set:** $\mathcal{T} = \{(\boldsymbol{x}_i^{\text{LR}}, \boldsymbol{x}_i^{\text{HR}})\}_{1=1..N}$

  HR images: $\boldsymbol{x}_i^{\text{HR}}$ (labels)

  LR versions: $\boldsymbol{x}_i^{\text{LR}} = \boldsymbol{H}(\boldsymbol{x}_i^{\text{HR}})$ (inputs generated from labels)

- **Model:**



- **Loss:** $E = \sum_{i=1}^{N} \|\boldsymbol{y}_i - \boldsymbol{x}_i^{\text{HR}}\|_2^2$ where $\boldsymbol{y}_i = f(\boldsymbol{x}_i^{\text{LR}}; \boldsymbol{W})$

Super-Resolution CNN (SRCNN) (Dong *et al.*, 2016)

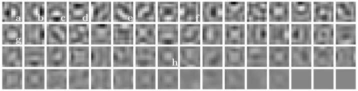
Settings

- Trained on ≈400,000 images from ImageNet.
- LR images obtained by Gaussian blur + subsampling.
- Inputs are Interpolated LR images (ILR) obtained by bi-cubic interpolation.
- 3 convolution layers: $\left\{ \begin{array}{l} \bullet\ f_1 \times f_1 \times n_1 = 9 \times 9 \times 64 \\ \bullet\ f_2 \times f_2 \times n_2 = 1 \times 1 \times 32 \\ \bullet\ f_3 \times f_3 \times n_3 = 5 \times 5 \times 1 \end{array} \right.$
- No pooling layers.
- ReLU for hidden layer, linear for output layers.

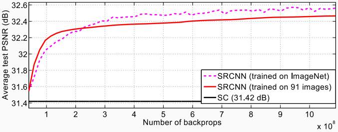
Standard measure of performance in dB (the larger the better):

$$\text{PSNR} = 10 \log_{10} \frac{255^2}{\frac{1}{n}\|\boldsymbol{y} - \boldsymbol{x}^{\text{HR}}\|^2}$$

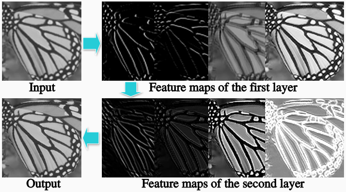
Super-Resolution CNN (SRCNN) (Dong *et al.*, 2016)



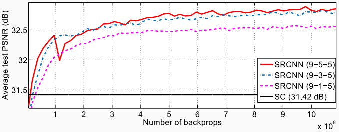
The figure shows the first-layer filters trained on ImageNet with an upscaling factor 3. The filters are organized based on their respective variances.



Training with the much larger ImageNet dataset improves the performance over the use of 91 images.



Example feature maps of different layers.

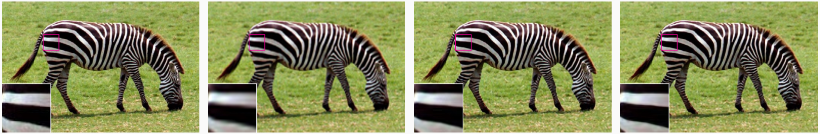


A larger filter size leads to better results.

- Training on RGB channels was better than training on YCbCr color space,
- Deeper did not always lead to better results,
- Though larger filters are better, they chose small filters to remain fast.

Super-Resolution CNN (SRCNN) (Dong *et al.*, 2016)

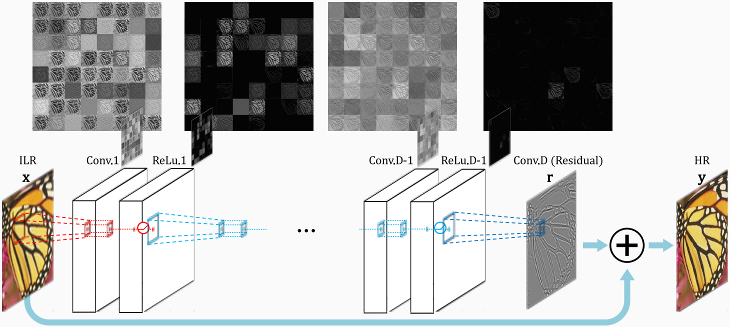
Original / PSNR — Bicubic / 24.04 dB — A+ / 27.24 dB — SRCNN / **27.95** dB



Original / PSNR — Bicubic / 26.63 dB — A+ / 28.98 dB — SRCNN / **29.29** dB

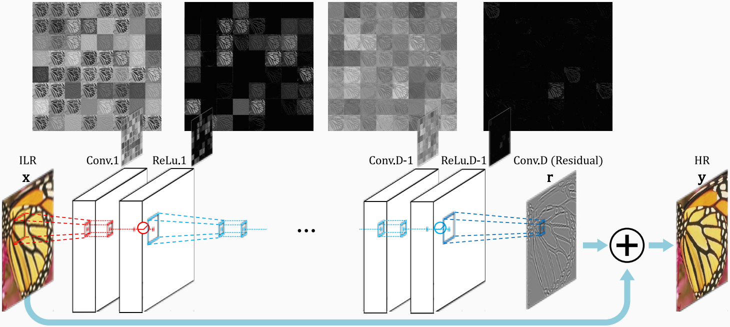
$\times$3 upsampling (9 times more pixels)

Super-resolution – VDSR (Kim *et al.*, 2016)



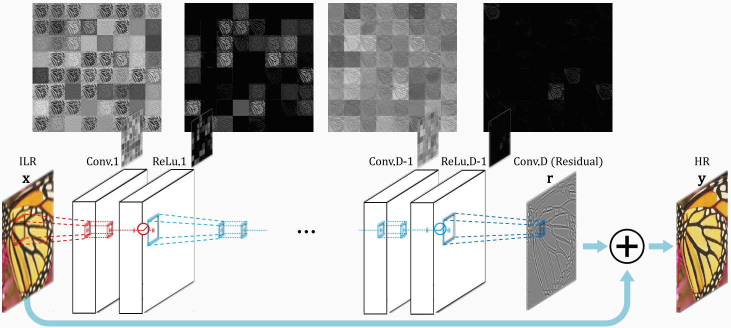
- Inspired from SRCNN:
  - Inputs are Interpolated LR images (ILR),
  - Fully convolutional (no pooling).

Super-resolution – VDSR (Kim *et al.*, 2016)



- Inspired from VGG: deep cascade of 20 small filter banks.
  - First hidden layer: 64 filters of size 3×3,
  - 18 other layers: 64 filters of size $3 \times 3 \times 64$,
  - Output layer: 1 filter of size $3 \times 3 \times 64$,
  - Receptive fields 41×41 (vs 13×13 for SRCNN)

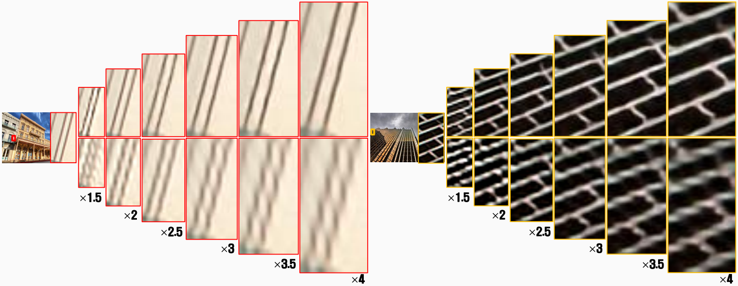
Super-resolution – VDSR (Kim *et al.*, 2016)



- Inspired from ResNet:
  - Learn the difference between HR and ILR images.
  - Inputs and outputs are highly correlated,
  - Just learn the subtle difference (high frequency details),
  - Allows using high learning rates (with gradient clipping).

Super-resolution – VDSR (Kim *et al.*, 2016)

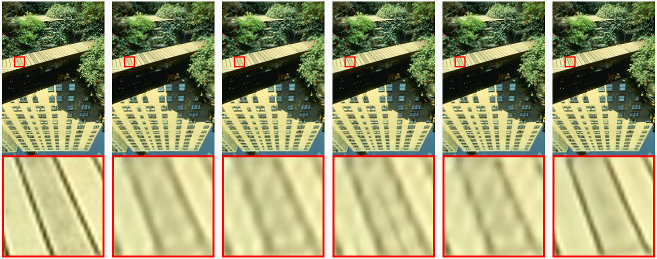
Test / Train	×2	×3	×4	×2,3	×2,4	×3,4	×2,3,4	Bicubic
×2	37.10	30.05	28.13	37.09	37.03	32.43	37.06	33.66
×3	30.42	32.89	30.50	33.22	31.20	33.24	33.27	30.39
×4	28.43	28.73	30.84	28.70	30.86	30.94	30.95	28.42



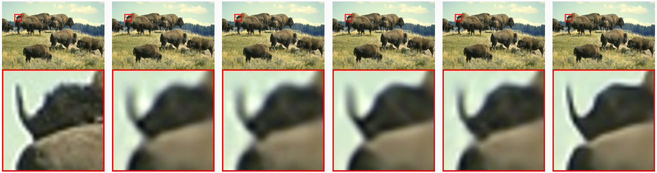
Single model for multiple scales

- Bottom: SRCNN trained for $\times 3$ upscaling,
- Top: VDSR trained for $\times 2, 3$ and 4 upscaling jointly.

Super-resolution – VDSR (Kim *et al.*, 2016)



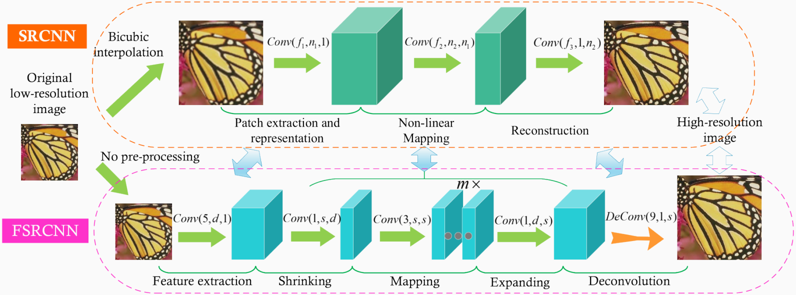
Ground Truth (PSNR, SSIM) | A+ [22] (22.92, 0.7379) | RFL [18] (22.90, 0.7332) | SelfEx [11] (23.00, 0.7439) | SRCNN [5] (23.15, 0.7487) | VDSR (Ours) (23.50, 0.7777)



n Ground Truth (PSNR, SSIM) | A+ [22] (27.08, 0.7514) | RFL [18] (27.08, 0.7508) | SelfEx [11] (27.02, 0.7513) | SRCNN [5] (27.16, 0.7545) | VDSR (Ours) (27.32, 0.7606)

$\times$3 upsampling (9$\times$ more pixels)

Super-resolution – Fast SRCNN (FSRCNN) (Dong *et al.*, 2016)



- Working in HR space is slow,
- Perform instead feature extraction in LR space,
  - $\rightarrow$ shared features regardless of the upscaling factor!
- Use fractionally strided convolutions only at the end to go to HR space.

Super-resolution – Fast SRCNN (FSRCNN) (Dong *et al.*, 2016)

	SRCNN-Ex	Transition State 1	Transition State 2	FSRCNN (56,12,4)
First part	Conv(9,64,1)	Conv(9,64,1)	Conv(9,64,1)	**Conv(5,56,1)**
Mid part	Conv(5,32,64)	Conv(5,32,64)	**Conv(1,12,64)-4Conv(3,12,12)-Conv(1,64,12)**	**Conv(1,12,56)-** 4Conv(3,12,12) **-Conv(1,56,12)**
Last part	Conv(5,1,32)	**DeConv(9,1,32)**	**DeConv(9,1,64)**	**DeConv(9,1,56)**
Input size	S_{HR}	S_{LR}	S_{LR}	S_{LR}
Parameters	57184	58976	17088	12464
Speedup	1×	8.7×	30.1×	41.3×
PSNR (Set5)	32.83 dB	32.95 dB	33.01 dB	33.06 dB

Compared to SRCNN

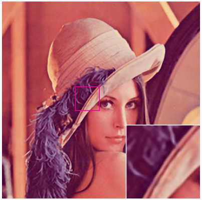
- Deeper: 8 hidden layers (compared to 3 for SRCNN),
- Faster: 40 times faster,
- Even superior restoration quality.

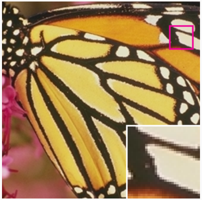
Super-resolution – Fast SRCNN (FSRCNN) (Dong *et al.*, 2016)

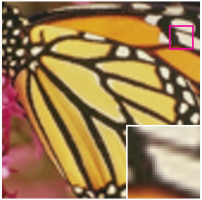

Original / PSNR

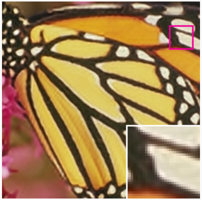

Bicubic / 31.68 dB

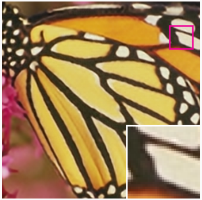

SRCNN / 33.39 dB


FSRCNN / **33.85** dB


Original / PSNR


Bicubic / 24.04 dB


SRCNN / 27.58 dB


FSRCNN / **28.68** dB

$\times$3 upsampling (9$\times$ more pixels)

Super-resolution – SRGAN (Twitter, Ledig *et al.*, CVPR 2017)

For large upsampling factors $\geqslant 4$:

- It becomes unrealistic to expect localizing exactly the edges,
- The MSE highly penalizes misplaced edges (even for a few pixel shift),
- Blurry solutions have lower MSE than sharp ones with misplaced edges,

$\Rightarrow$ The system will never be able to reconstruct high frequency content.

Idea: use a perceptual loss based on

- content loss to force SR images to be perceptually similar to the HR ones,
- adversarial loss to force SR results to be photo-realistic.

Connection with GAN: Learn to fool a discriminator trained to distinguish Super-Resolved images from HR photo-realistic ones.

Super-resolution – SRGAN (Twitter, Ledig *et al.*, CVPR 2017)

Adversarial loss: Same as GAN but replace the latent code by the LR image

$$\min_{\theta_{\text{SR}}} \max_{\theta_{\text{D}}} \sum \log D_{\theta_D}(\boldsymbol{x}^{\text{HR}}) + \log(1 - D_{\theta_D}(\boldsymbol{x}^{\text{SR}})) + \lambda L_{\text{content}}(\boldsymbol{x}^{\text{SR}}, \boldsymbol{x}^{\text{HR}})$$

$$\text{where} \quad \boldsymbol{x}^{\text{SR}} = G_{\theta_{\text{SR}}}(\boldsymbol{x}^{\text{LR}}) \quad \text{and} \quad \boldsymbol{x}^{\text{LR}} = \boldsymbol{H}(\boldsymbol{x}^{\text{HR}})$$

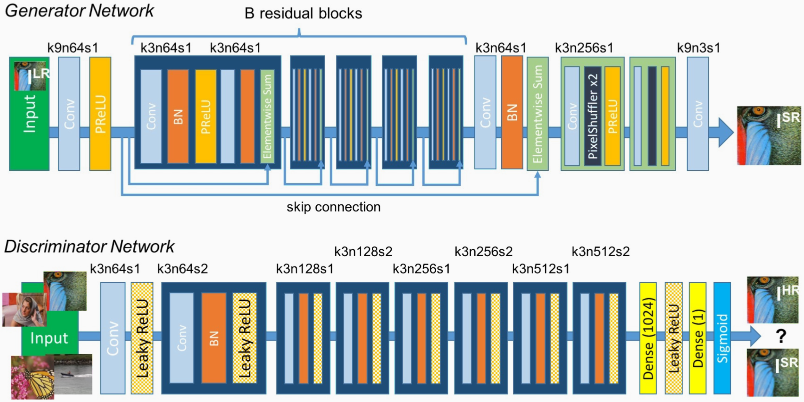
$\rightarrow L_{\text{content}}$ ensures generating SR images corresponding to their HR version.

Content loss: Euclidean distance between the L^{th} feature tensors obtained with VGG for the SR and HR images, respectively:

$$L_{\text{content}}(\boldsymbol{x}^{\text{SR}}, \boldsymbol{x}^{\text{HR}}) = \|\boldsymbol{h}^{\text{SR}} - \boldsymbol{h}^{\text{HR}}\|_2^2 \quad \text{with} \quad \begin{cases} \boldsymbol{h}^{\text{SR}} = \text{VGG}^L(\boldsymbol{x}^{\text{SR}}) \\ \boldsymbol{h}^{\text{HR}} = \text{VGG}^L(\boldsymbol{x}^{\text{HR}}) \\ \boldsymbol{x}^{\text{SR}} = \text{G}_{\theta_{\text{SR}}}(\boldsymbol{x}^{\text{LR}}) \end{cases}$$

- Force images to have similar high level feature tensors.
- Supposed to be closer to perceptual similarity.

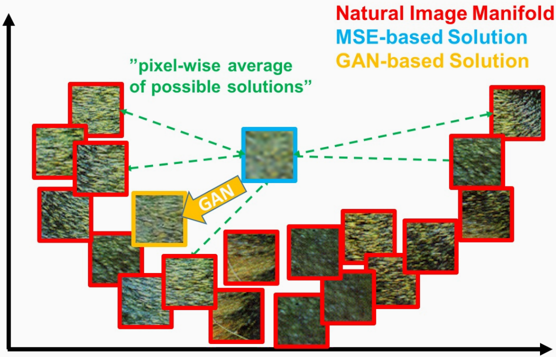
Super-resolution – SRGAN (Twitter, Ledig *et al.*, CVPR 2017)



Architecture of Generator and Discriminator Network with corresponding kernel size (k), number of feature maps (n) and stride (s) indicated for each convolutional layer.

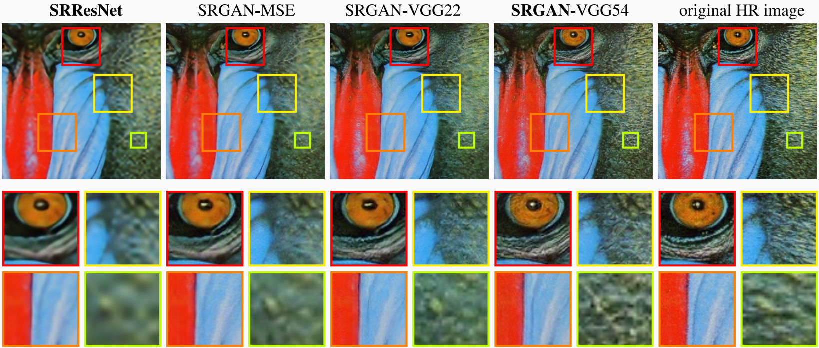
Both networks are trained by alternating their gradient based updates.

Super-resolution – SRGAN (Twitter, Ledig *et al.*, CVPR 2017)



- The SR problem is ill-posed → infinite number of solutions,
- MSE promotes a pixel-wise average of them → over-smooth,
- GAN drives reconstruction towards the "natural image manifold".

Super-resolution – SRGAN

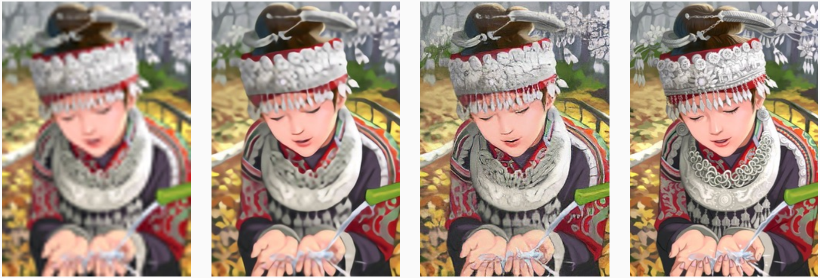


$\times 4$ upsampling ($16\times$ more pixels)

- SRResNet: ResNet SR generator trained with MSE,
- SRGAN-MSE: generator and discriminator with MSE content loss,
- SRGAN-VGG22: generator and discriminator with VGG22 content loss,
- SRGAN-VGG54: generator and discriminator with VGG54 content loss.

Super-resolution – SRGAN

bicubic (21.59dB/0.6423)	SRResNet (23.53dB/0.7832)	SRGAN (21.15dB/0.6868)	original



$\times$4 upsampling (16$\times$ more pixels)

Even though some details are lost, they are replaced by “fake” but photo-realistic objects (instead of blurry ones).

Remark that SRResNet is blurrier but achieves better PSNR.

Style transfer

Style transfer (Gatys, Ecker and Bethge, 2015)

- VGG feature maps are very good to capture relevant image features,
- A photo-realistic image $\boldsymbol{y}$ can be approximated by $\boldsymbol{x}$ minimizing

$$\ell^l_{\text{content}}(\boldsymbol{x}; \boldsymbol{y}) = \|\mathrm{VGG}^l(\boldsymbol{x}) - \mathrm{VGG}^l(\boldsymbol{y})\|_2^2 \qquad (l\text{: a chosen hidden layer})$$

- Non-convex optimization problem: can use GD with Adam, L-BFGS, ...

Style transfer (Gatys, Ecker and Bethge, 2015)

- VGG feature maps are very good to capture relevant image features,
- A photo-realistic image $\boldsymbol{y}$ can be approximated by $\boldsymbol{x}$ minimizing

$$\ell^l_{\text{content}}(\boldsymbol{x}; \boldsymbol{y}) = \|\text{VGG}^l(\boldsymbol{x}) - \text{VGG}^l(\boldsymbol{y})\|_2^2 \qquad (l\text{: a chosen hidden layer})$$

- Non-convex optimization problem: can use GD with Adam, L-BFGS, ...

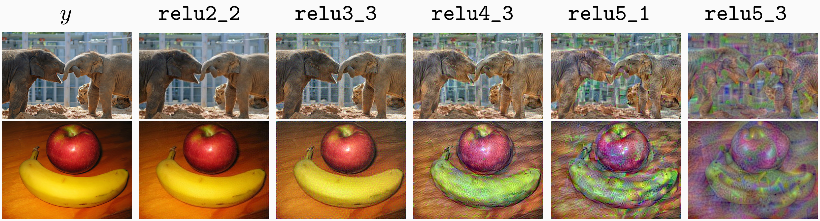
```
x   = torch.rand(y.shape).cuda()
x   = nn.Parameter(x, requires_grad=True)
hy  = VGGfeatures(y)[l]
optimizer = torch.optim.Adam([x], lr=0.01)
for t in range(0, T):
  optimizer.zero_grad()
  hx   = VGGfeatures(x)[l]
  loss = ((hx - hy)**2).mean()
  loss.backward(retain_graph=True)
  optimizer.step()
```

Style transfer (Gatys, Ecker and Bethge, 2015)

- VGG feature maps are very good to capture relevant image features,
- A photo-realistic image $\boldsymbol{y}$ can be approximated by $\boldsymbol{x}$ minimizing

$$\ell^l_{\text{content}}(\boldsymbol{x}; \boldsymbol{y}) = \|\mathrm{VGG}^l(\boldsymbol{x}) - \mathrm{VGG}^l(\boldsymbol{y})\|_2^2 \qquad (l\text{: a chosen hidden layer})$$

- Non-convex optimization problem: can use GD with Adam, L-BFGS, ...

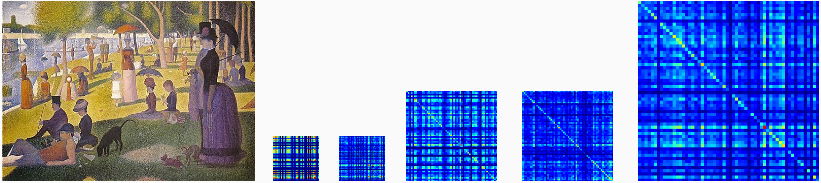


Style transfer (Gatys, Ecker and Bethge, 2015)

- Texture/Style can be captured by looking at the covariances between all VGG feature maps m and n of the same layer l.
- The matrix of all covariances is called Gram matrix:

$$\boldsymbol{G}^l(\boldsymbol{x})_{m,n} = \sum_{i,j} \mathrm{VGG}^l(\boldsymbol{x})_{i,j,m} \mathrm{VGG}^l(\boldsymbol{x})_{i,j,n}$$

where (i, j) are pixel indices.



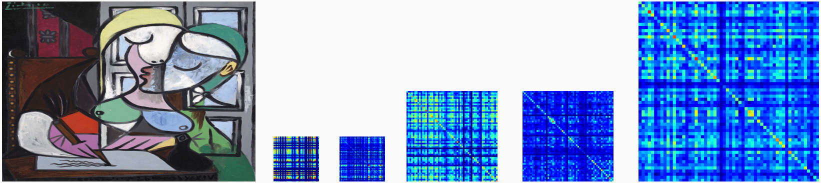
Style transfer

Style transfer (Gatys, Ecker and Bethge, 2015)

- Texture/Style can be captured by looking at the covariances between all VGG feature maps m and n of the same layer l.
- The matrix of all covariances is called Gram matrix:

$$\boldsymbol{G}^l(\boldsymbol{x})_{m,n} = \sum_{i,j} \mathrm{VGG}^l(\boldsymbol{x})_{i,j,m} \mathrm{VGG}^l(\boldsymbol{x})_{i,j,n}$$

where (i, j) are pixel indices.

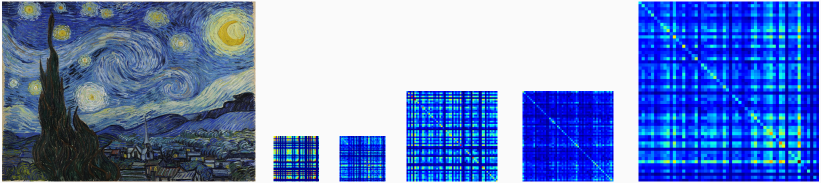


Style transfer (Gatys, Ecker and Bethge, 2015)

- Texture/Style can be captured by looking at the covariances between all VGG feature maps m and n of the same layer l.
- The matrix of all covariances is called Gram matrix:

$$\boldsymbol{G}^l(\boldsymbol{x})_{m,n} = \sum_{i,j} \mathrm{VGG}^l(\boldsymbol{x})_{i,j,m} \mathrm{VGG}^l(\boldsymbol{x})_{i,j,n}$$

where (i, j) are pixel indices.



Style transfer (Gatys, Ecker and Bethge, 2015)

- Textures $\boldsymbol{y}$ can be synthesized by $\boldsymbol{x}$ minimizing

$$\ell^l_{\text{style}}(\boldsymbol{x};\boldsymbol{y}) = \|\boldsymbol{G}^l(\boldsymbol{x}) - \boldsymbol{G}^l(\boldsymbol{y})\|^2_F$$

- Again a non-convex optimization problem.

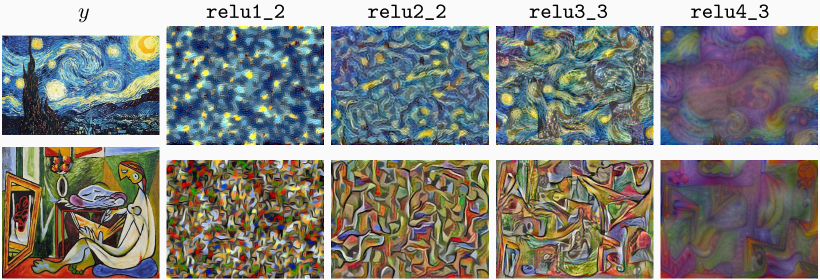
```
x    = torch.rand(y.shape).cuda()
x    = nn.Parameter(x, requires_grad=True)
hy   = VGGfeatures(y)[l].view(C, W * H)
Gy   = torch.mm(hy, hy.t())
for t in range(0, T):
  optimizer.zero_grad()
  hx   = VGGfeatures(x)[l].view(C, W * H)
  Gx   = torch.mm(hx, hx.t())
  loss = ((Gx - Gy) ** 2).sum()
  loss.backward(retain_graph=True)
  optimizer.step()
```

Style transfer (Gatys, Ecker and Bethge, 2015)

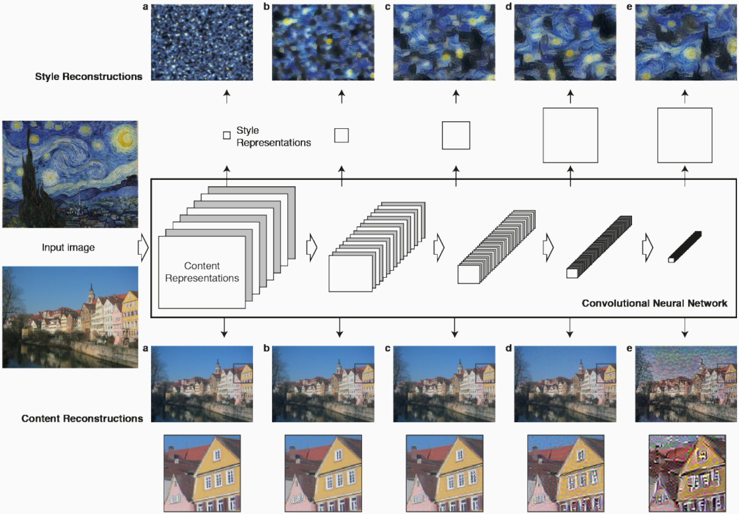
- Textures $\boldsymbol{y}$ can be synthesized by $\boldsymbol{x}$ minimizing

$$\ell^l_{\text{style}}(\boldsymbol{x};\boldsymbol{y}) = \|\boldsymbol{G}^l(\boldsymbol{x}) - \boldsymbol{G}^l(\boldsymbol{y})\|_F^2$$

- Again a non-convex optimization problem.



Style transfer (Gatys, Ecker and Bethge, 2015)



Style transfer (Gatys, Ecker and Bethge, 2015)

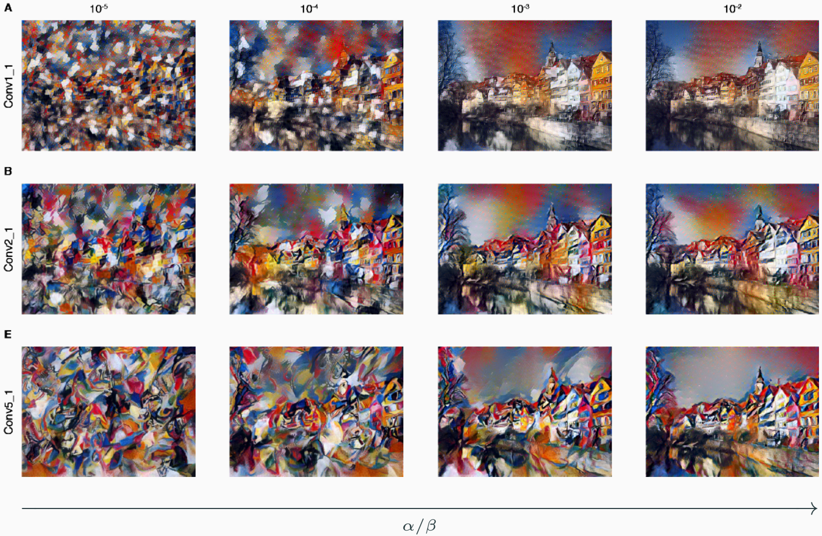
Style transfer: $$\min_{\boldsymbol{x}} \quad \underbrace{\alpha \ell^l_{\text{content}}(\boldsymbol{x}; \boldsymbol{y}_c)}_{\text{match content at depth } l} + \underbrace{\beta \sum_{j=1}^{J} \ell^j_{\text{style}}(\boldsymbol{x}; \boldsymbol{y}_s)}_{\text{match texture from depth } 1 \text{ to } J}$$

- Look for $\boldsymbol{x}$ such that
  - its content corresponds to $\boldsymbol{y}_c$,
    $\rightarrow$ match VGG features at layer l (typically: $l = 2, 3$ or 4)
  - its style corresponds to $\boldsymbol{y}_s$,
    $\rightarrow$ match VGG correlations at layers 1 to J (typically: $J = 4$ or 5)
- Again a non-convex optimization problem.
- Remark: no training data $\Rightarrow$ not a ML algorithm,
  $\rightarrow$ this is just a simple CV technique relying on image features.

Style transfer (Gatys, Ecker and Bethge, 2015)

Style transfer: $$\min_{\boldsymbol{x}} \quad \underbrace{\alpha \ell^l_{\text{content}}(\boldsymbol{x}; \boldsymbol{y}_c)}_{\text{match content at depth } l} + \underbrace{\beta \sum_{j=1}^{J} \ell^j_{\text{style}}(\boldsymbol{x}; \boldsymbol{y}_s)}_{\text{match texture from depth } 1 \text{ to } J}$$

Style transfer (Gatys, Ecker and Bethge, 2015)



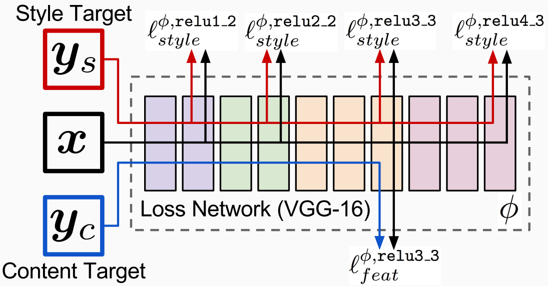
Style transfer (Gatys, Ecker and Bethge, 2015)

Super easy to implement in PyTorch:
just sum all the losses with weights α and β!

But slow, about 15mins on DSMLP with GPU
for a 444×295 image.

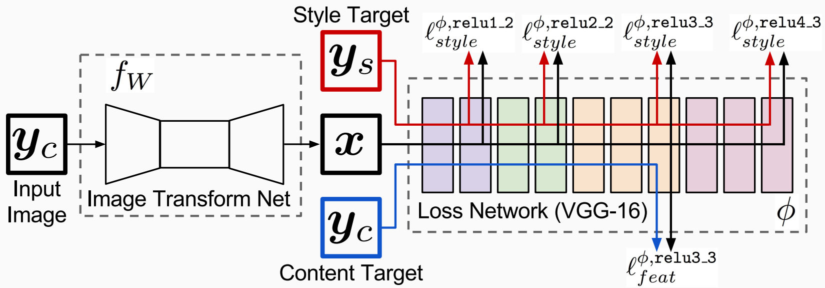
Style transfer (Johnson, Alahi, Fei-Fei, 2016)

Problem: Optimizing the input x of the VGG network is slow
(requires about 500 forward and backward passes through VGG during runtime).



Style transfer (Johnson, Alahi, Fei-Fei, 2016)

Problem: Optimizing the input $\boldsymbol{x}$ of the VGG network is slow (requires about 500 forward and backward passes through VGG during runtime).



Solution: train a network to predict $\boldsymbol{x}$ from $\boldsymbol{y}_c$ using Gatys' loss.

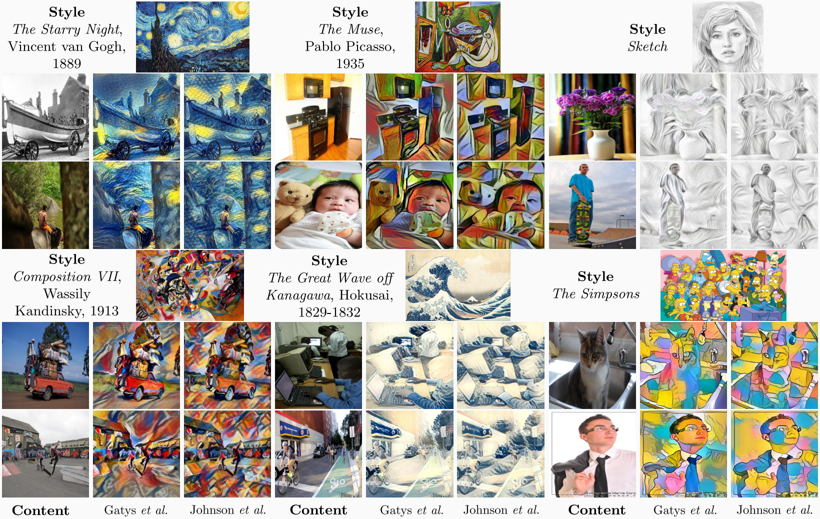
Style transfer (Johnson, Alahi, Fei-Fei, 2016)

$$\min_{\boldsymbol{\theta}_s} \quad \sum_{i=1}^{N} \alpha \ell^{l}_{\text{content}}(\boldsymbol{x}; \boldsymbol{y}^i_c) + \beta \sum_{j=1}^{J} \ell^{j}_{\text{style}}(\boldsymbol{x}; \boldsymbol{y}_s) + \underbrace{\gamma \|\nabla \boldsymbol{x}\|_1}_{\text{Total-Variation}} \quad \text{where} \quad \boldsymbol{x} = f(\boldsymbol{y}^i_c; \boldsymbol{\theta}_s)$$

- Add a Total-Variation term to encourage smoothness,
- Use a residual network with:
  - 2 strided convolution to downsample,
  - Several residual blocks (with shortcut connections),
  - 2 fractionally strided convolutions to upsample.
- Trained on $N = 80,000$ images of size 256×256 from MS COCO,
- One network has to be trained for each single target style $\boldsymbol{y}_s$,
- At test time, requires only one forward pass of this new network,
- Unlike Gatys' method, this one is a ML algorithm.

Style transfer

Style transfer (Johnson, Alahi, Fei-Fei, 2016)



Questions?

That's all folks!

Sources, images courtesy and acknowledgment

A. Horodniceanu
T. Jeon
J. Johnson

F.-F. Li
V. Veerabadran
S. Yeung

Wikipedia
+ all referenced articles