

ECE 285
Machine Learning for Image Processing

Chapter VI – Generation, super-resolution and style transfer

Charles Deledalle

September 23, 2019

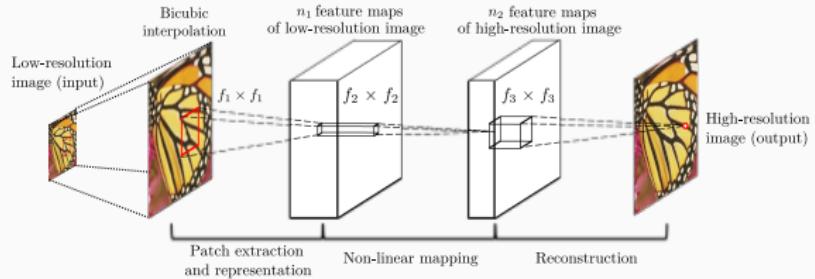
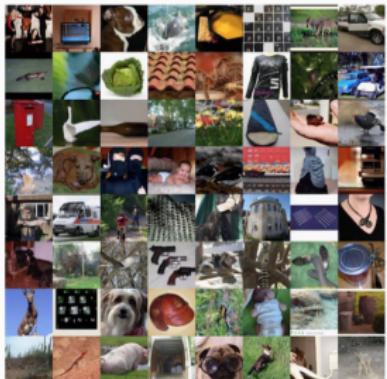


Image generation

Motivations – Image generation

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



Real images (ImageNet)
(Training set)

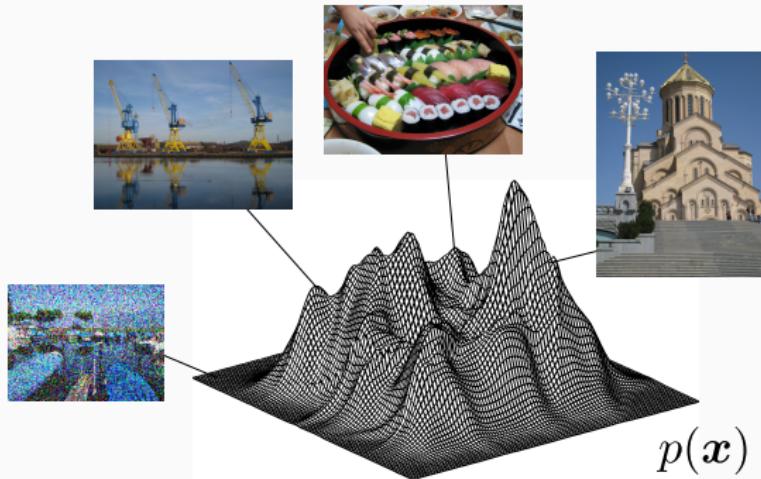


Generated images
(Results)

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

Image generation – Explicit density

- ① Learn the distribution of images $p(x)$ on a training set.



- ② Generate samples from this distribution.

Image generation – Gaussian model

- Consider a Gaussian model for the distribution of images x with n pixels:

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

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

- μ : mean image,
- Σ : covariance matrix of images.

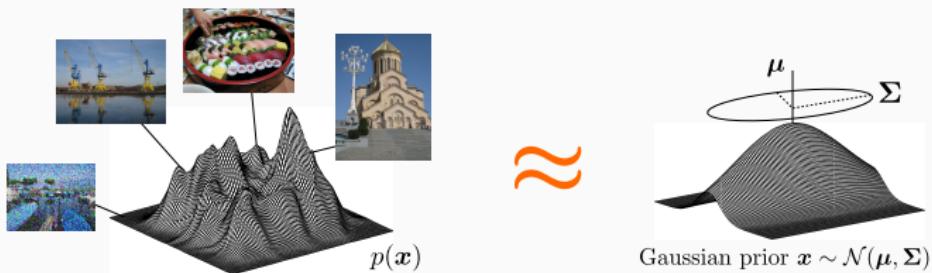


Image generation

Image generation – Gaussian model

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

$$\mathcal{T} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$$

$$= \left\{ \begin{array}{c} \text{[Image of a man]} \\ \text{[Image of a man with glasses]} \\ \text{[Image of a woman with glasses]} \\ \text{[Image of a woman]} \\ \text{[Image of a man]} \\ \text{[Image of a woman]} \\ \vdots \end{array} \times N \right\}$$

- Estimate the mean

$$\hat{\mu} = \frac{1}{N} \sum_i \mathbf{x}_i = \text{[Blurry face image]}$$

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

$$\hat{\mathbf{E}} = \left\{ \begin{array}{c} \text{[Blurry face image]} \\ \vdots \end{array} \times N \right\}$$

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

Image generation – Gaussian model

You now have learned a generative model:

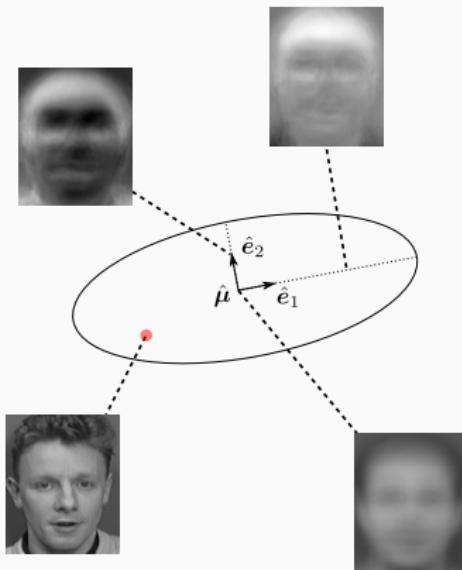


Image generation – Gaussian model

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

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



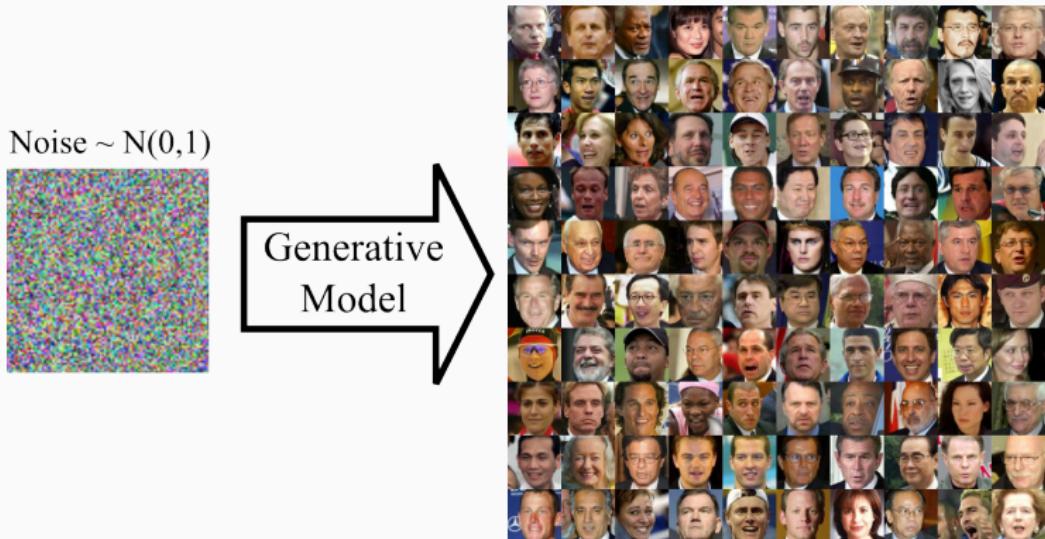
The model does not generate realistic faces.

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

Image generation – Beyond Gaussian models

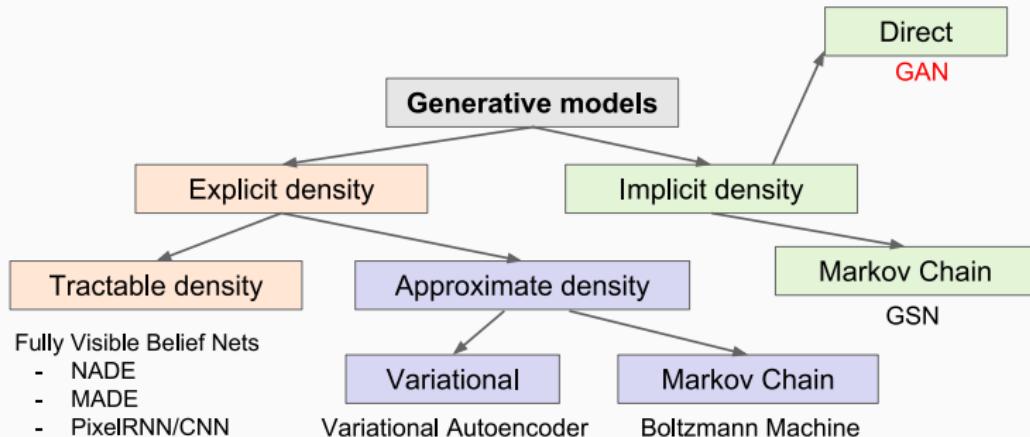


But the concept is interesting: can we find a transformation such that each random code can be mapped to a photo-realistic image?

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

Image generation – Beyond Gaussian models

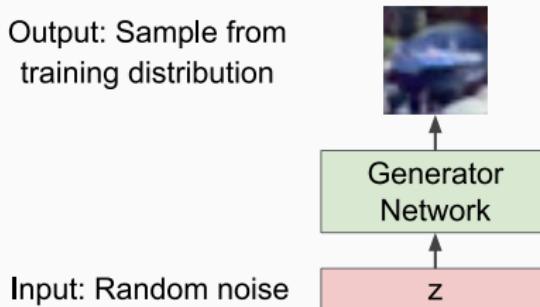
Taxonomy of Generative Models



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**

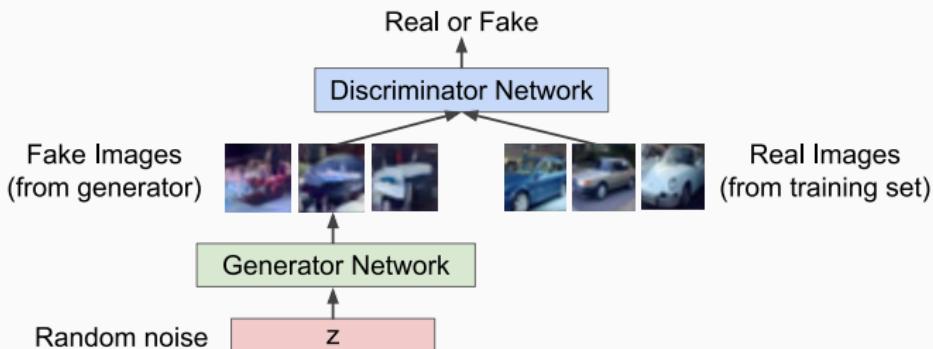


- **What about the loss?** Measure if the generated image is **photo-realistic**.

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 $\mathbf{x} \mapsto D_{\theta_d}(\mathbf{x})$ meant to classify real and fake images.
Minimize the cross entropy, or maximize its negation

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

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

Generative Adversarial Networks

(Goodfellow et al., NIPS 2014)

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

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

or alternatively (works better in practice)

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

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

Generative Adversarial Networks

(Goodfellow et al., NIPS 2014)

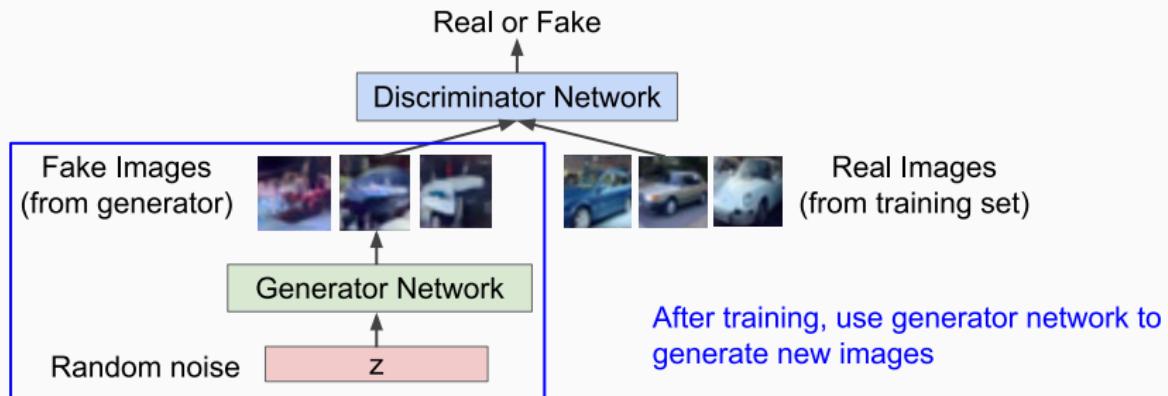
- Train both networks jointly.
- Minimax loss in a two player game (each player is a network):

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

- **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)



Generative Adversarial Networks

(Goodfellow et al., NIPS 2014)

Generated samples

7	3	9	3	9	9
1	1	0	6	0	0
0	1	9	1	2	2
6	3	2	0	8	8



Nearest neighbor from training set

Generative Adversarial Networks

(Goodfellow et al., NIPS 2014)

Generated samples (CIFAR-10)

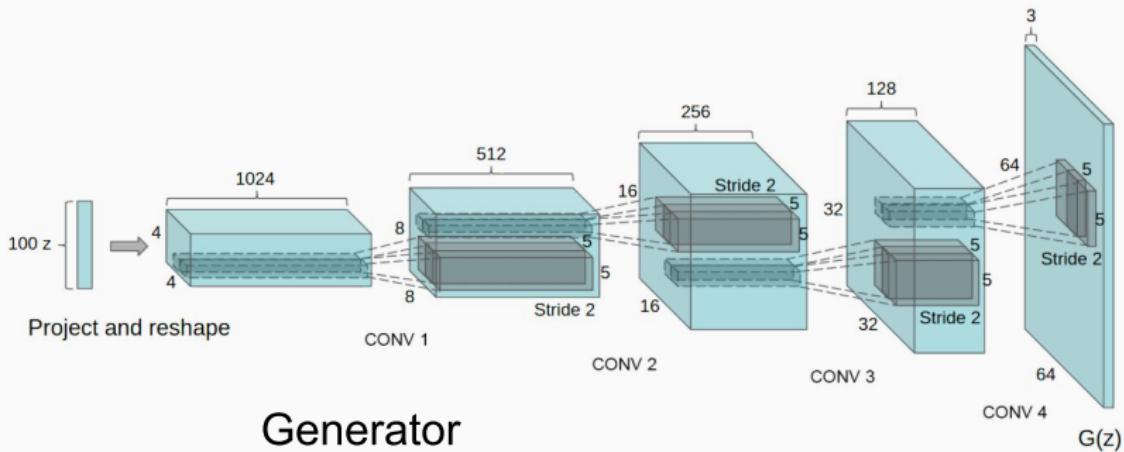


Nearest neighbor from training set

Convolutional GAN

(Radford et al., 2016)

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



Generator

Convolutional GAN

(Radford et al., 2016)



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

Image generation

Convolutional GAN

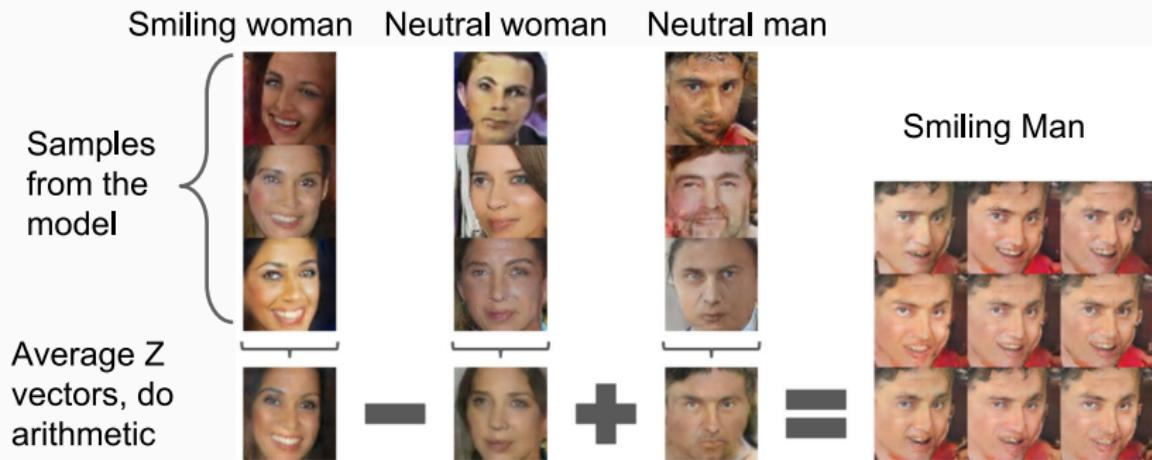
(Radford et al., 2016)



Interpolation in between points in latent space.

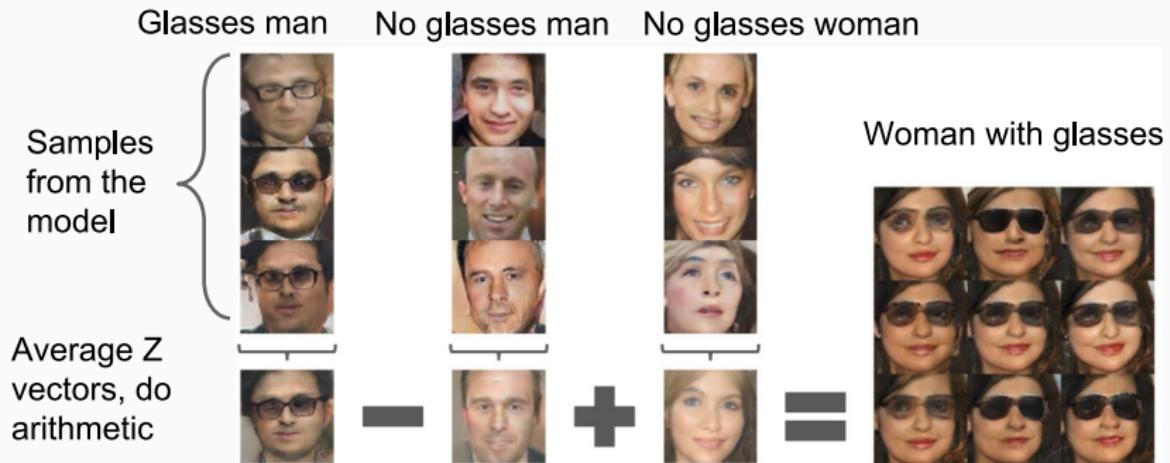
Convolutional GAN – Arithmetic

(Radford et al., 2016)



Convolutional GAN – Arithmetic

(Radford et al., 2016)



GAN Sub-classification

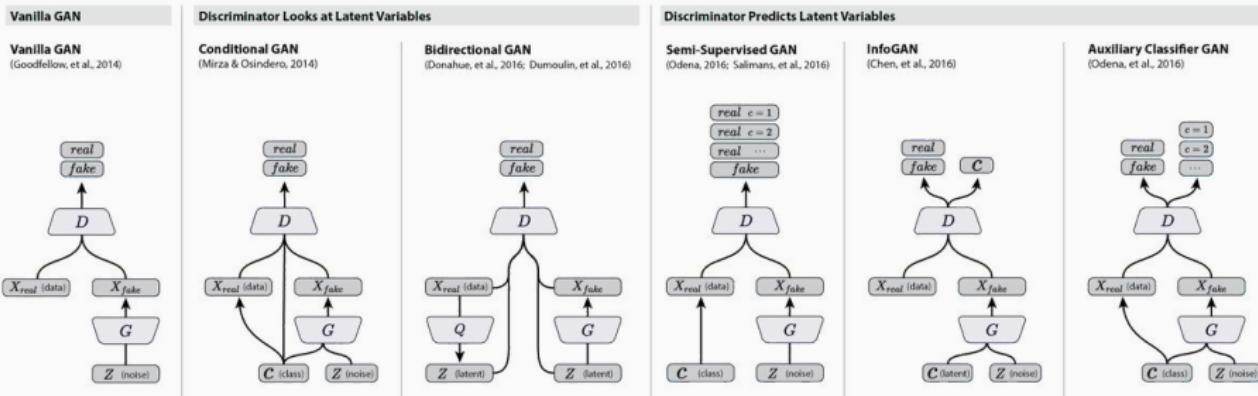


Image generation

2017: Year of the GAN

Better training and generation

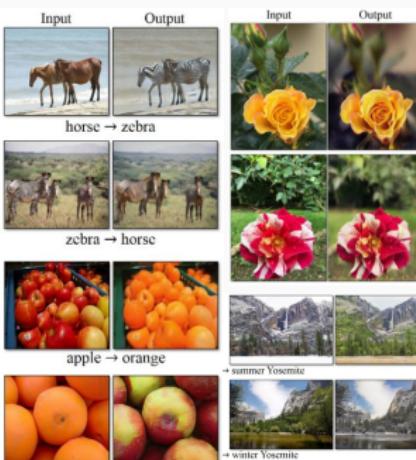


LSGAN. Mao et al. 2017.



BEGAN. Bertholet et al. 2017.

Source->Target domain transfer



CycleGAN. Zhu et al. 2017.

Text -> Image Synthesis

this small bird has a pink breast and crown, and black primaries and secondaries.



Reed et al. 2017.

Many GAN applications

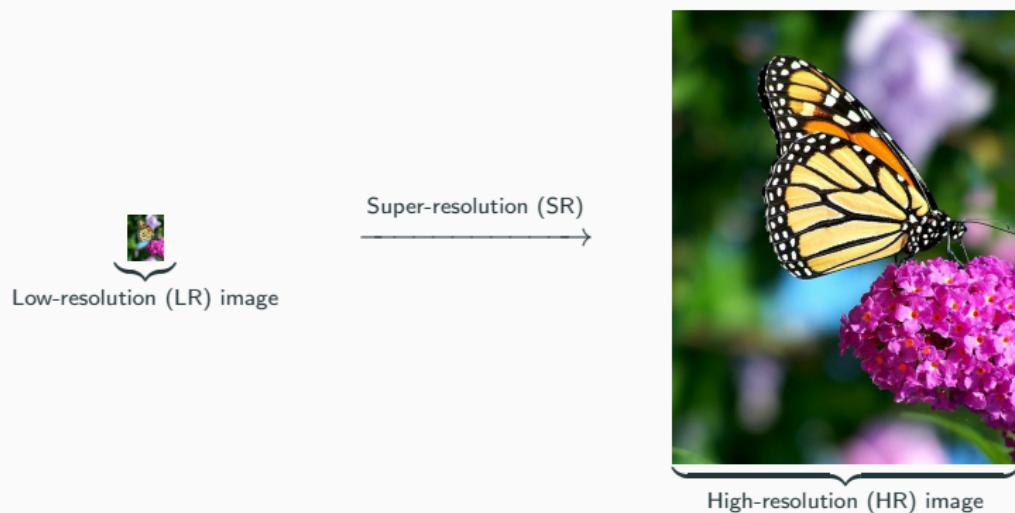


Pix2pix. Isola 2017. Many examples at <https://phillipi.github.io/pix2pix/>

Super-resolution

Super-resolution

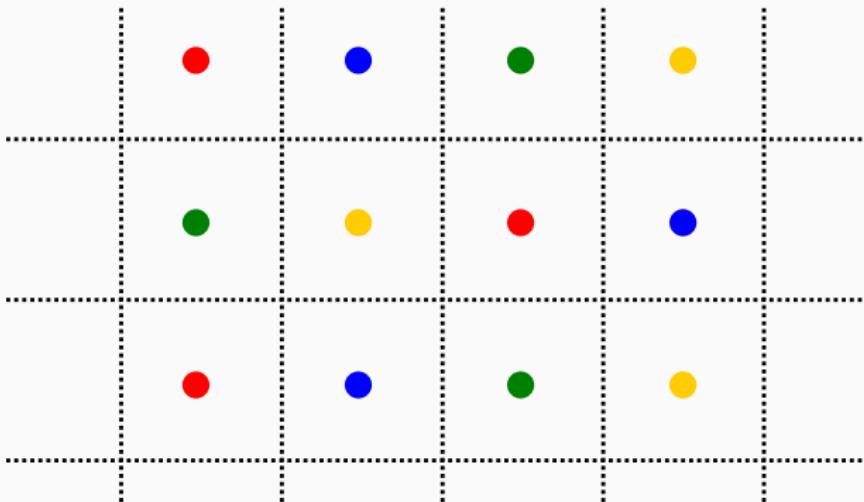
Super-resolution



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

Super-resolution – Interpolation

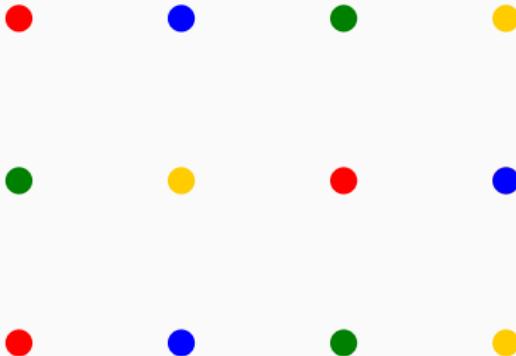
Approach 1: Consider the SR problem as a 2d interpolation problem.



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

Super-resolution – Interpolation

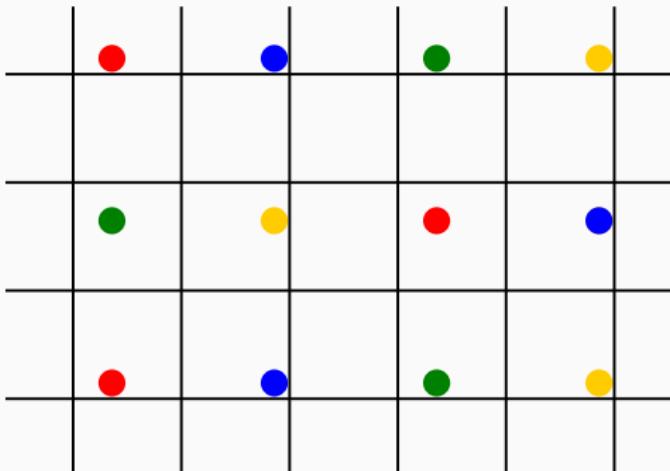
Approach 1: Consider the SR problem as a 2d interpolation problem.



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

Super-resolution – Interpolation

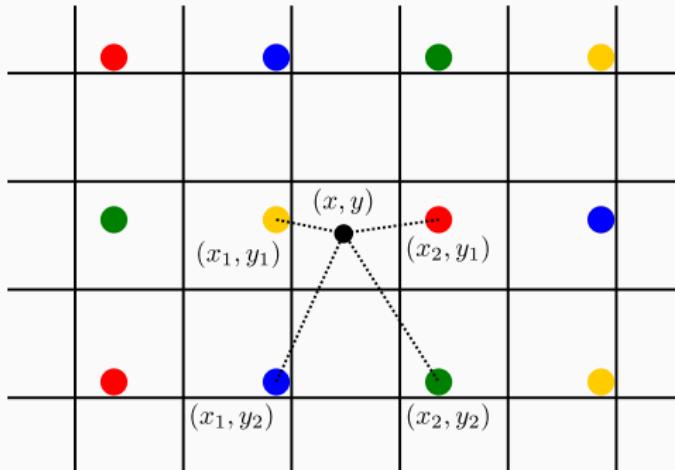
Approach 1: Consider the SR problem as a 2d interpolation problem.



Inject the targeted HR grid.

Super-resolution – Interpolation

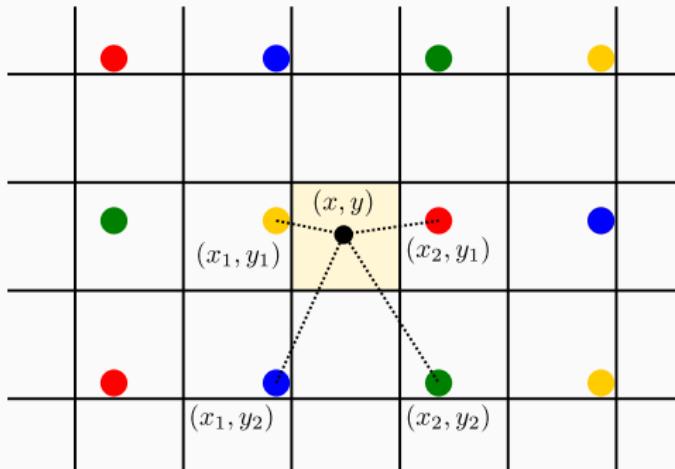
Approach 1: Consider the SR problem as a 2d interpolation problem.



Deduce HR pixel values based on LR points.

Super-resolution – Nearest neighbor interpolation

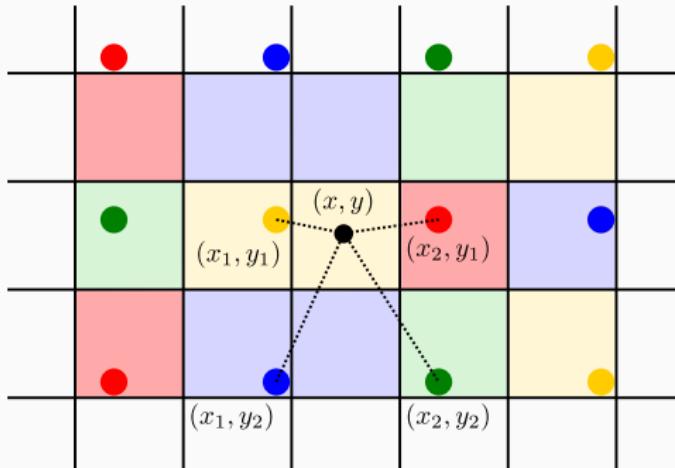
Nearest neighbor: affect the pixel value of the closest LR point.



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

Super-resolution – Nearest neighbor interpolation

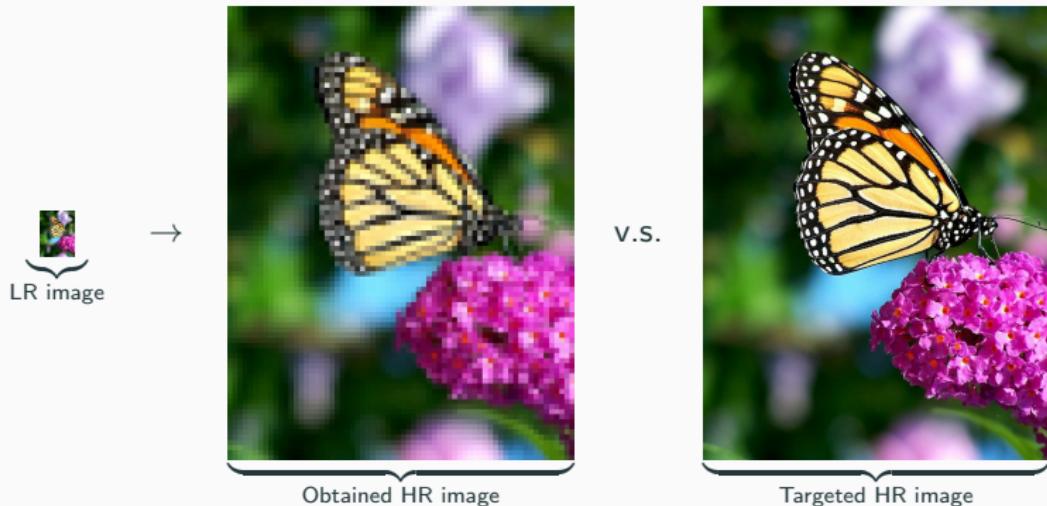
Nearest neighbor: affect the pixel value of the closest LR point.



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

Super-resolution – Approach 1 – Interpolation

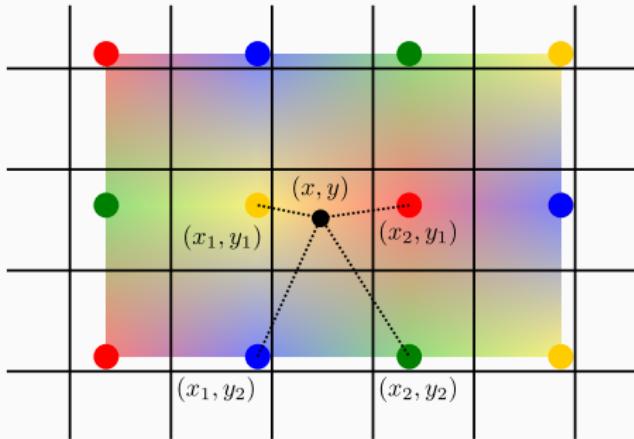
Super-resolution – Nearest neighbor interpolation



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

Super-resolution – Approach 1 – Interpolation

Super-resolution – Bi-linear interpolation

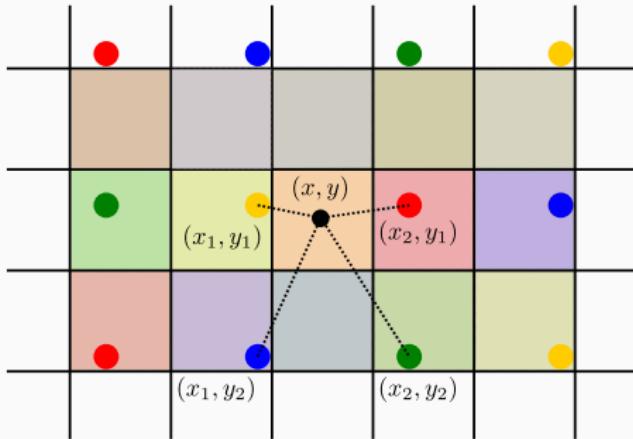


Bi-linear: combine pixel values of LR points with respect to their distance to the HR point.

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

Super-resolution – Approach 1 – Interpolation

Super-resolution – Bi-linear interpolation



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

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

Super-resolution – Approach 1 – Interpolation

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

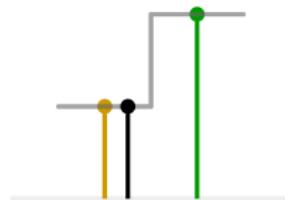
- Same but with 16 coefficients a_{ij} , $0 \leq i, j \leq 3$, of

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

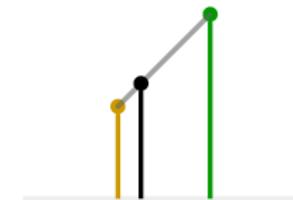
- 16 unknowns and 4 equations \rightarrow infinite number of solutions,
- Interpolate the derivatives: 4 in $x + 4$ in $y + 4$ in $xy \rightarrow 16$ equations.
- Closed-form solution obtained by inverting a 16×16 matrix.

Super-resolution – Approach 1 – Interpolation

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



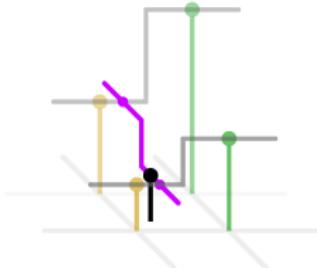
1d nearest-neighbour



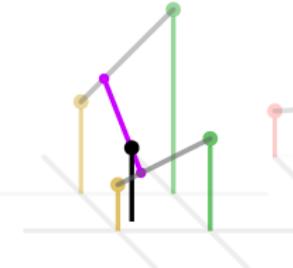
Linear



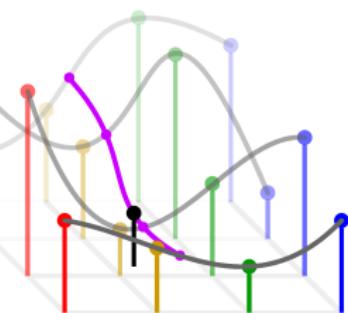
Cubic



2d nearest-neighbour



Bilinear



Bicubic

Super-resolution – Approach 1 – Interpolation

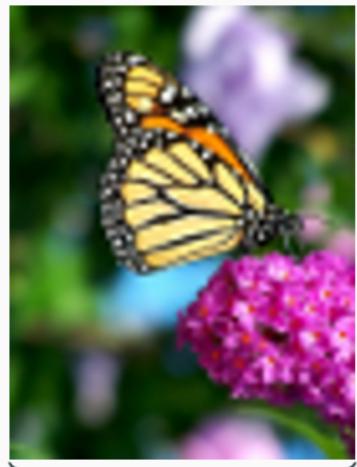
Super-resolution – Bi-linear / Bi-cubic interpolation



Nearest neighbor



Bi-linear

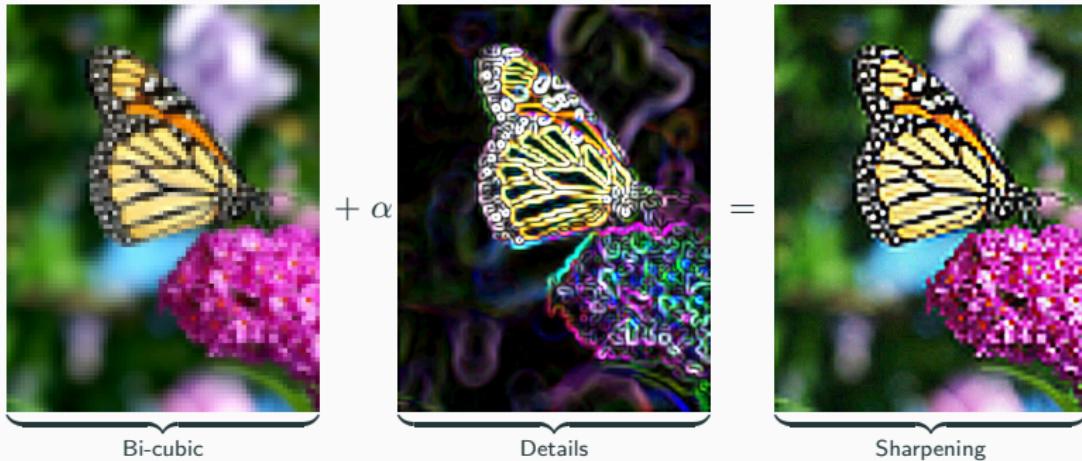


Bi-cubic

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

Super-resolution – Approach 1 – Interpolation

Super-resolution – Interpolation + Sharpening



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

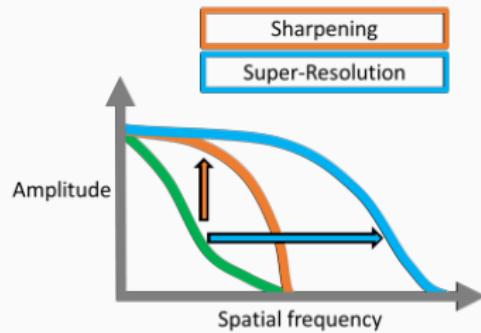
More contrast but still blocky artifacts and lacking fine details.

Super-resolution – Approach 1 – Interpolation

Super-resolution – Image sharpening



SHARP!



BIG!

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

(Source: Taegyun Jeon)

Super-resolution – Approach 2 – Linear inverse problem

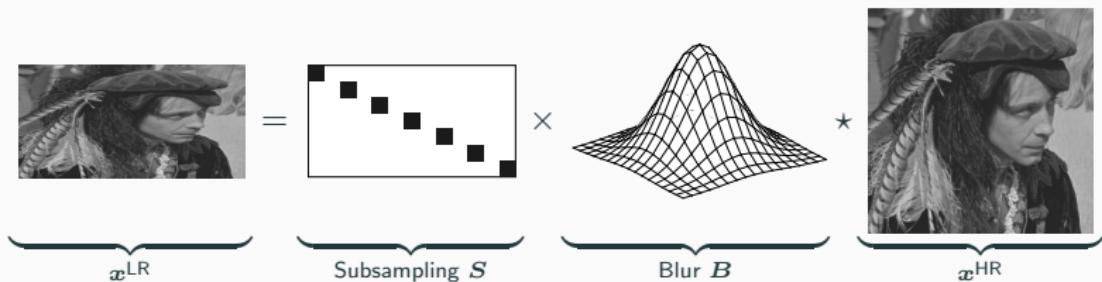
Super-resolution – Linear inverse problem

Approach 2: Model SR as a linear inverse problem

- Linear model based on the characteristics of the camera

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

- **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

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

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

- But, $\mathbf{H} = \mathbf{S}\mathbf{B}$ is not invertible \rightarrow 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:

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

Super-resolution – Approach 2 – Linear inverse problem

Super-resolution – Linear inverse problem

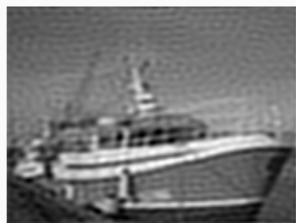
- The solution of minimum norm is given by: $\mathbf{x}^{\text{HR+}} = \mathbf{H}^+ \mathbf{x}^{\text{LR}}$ where \mathbf{H}^+ is the Moore-Penrose pseudo-inverse.
- But, as the problem is ill-posed:
 - small perturbations in \mathbf{x}^{LR} lead to large errors in $\mathbf{x}^{\text{HR+}}$,
 - and unfortunately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



(a) HR image \mathbf{x}^{HR}



(b) Noisy LR \mathbf{x}^{HR}



(c) Pinv $\mathbf{x}^{\text{HT+}}$

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:

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

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

Super-resolution – Approach 2 – Linear inverse problem

Super-resolution – Linear inverse problem



(a) LR image



(b) Tiny $\tau \sim \text{pinv}$



(c) Small τ



(d) Good τ



(e) High τ



(f) Huge τ

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

Super-resolution – Approach 2 – Linear inverse problem

Super-resolution – Linear inverse problem



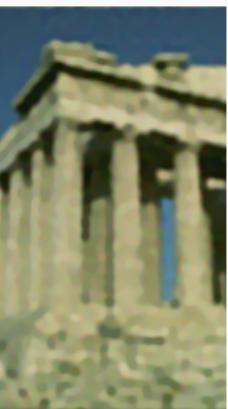
(a) LR image



(b) Tiny $\tau \sim \text{pinv}$



(c) Small τ



(d) Good τ



(e) High τ



(f) Huge τ

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

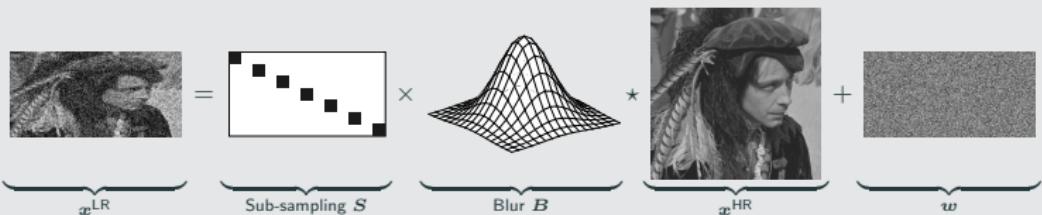
Super-resolution – Linear inverse problem

- 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 S and blur 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).

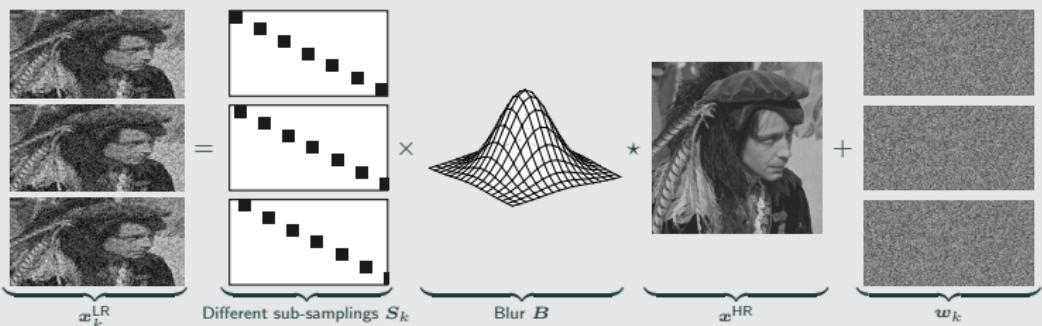
Super-resolution – Approach 2 – Linear inverse problem

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 – Approach 3 – CNN

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

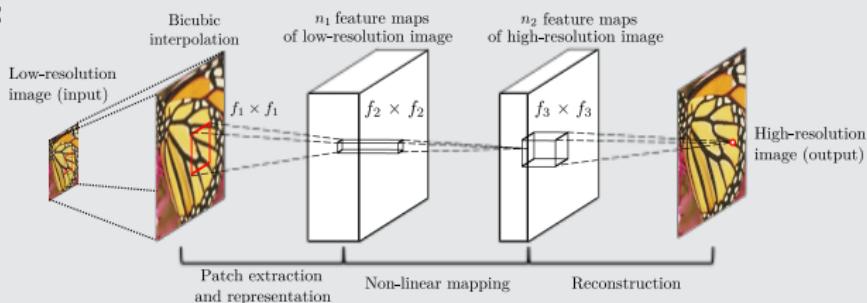
Approach 3: learn to map LR to HR images using a dataset of HR images.

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

HR images: \mathbf{x}_i^{HR} (labels)

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

- **Model:**



- **Loss:**

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

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

Settings

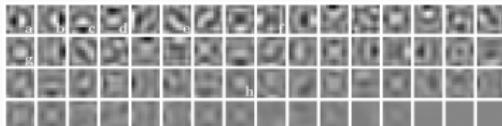
- Trained on $\approx 400,000$ images from ImageNet.
- LR images obtained by Gaussian blur + subsampling.
- Inputs are Interpolated LR images (ILR) obtained by bi-cubic interpolation.
- 3 convolution layers:
$$\left\{ \begin{array}{l} \bullet f_1 \times f_1 \times n_1 = 9 \times 9 \times 64 \\ \bullet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

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

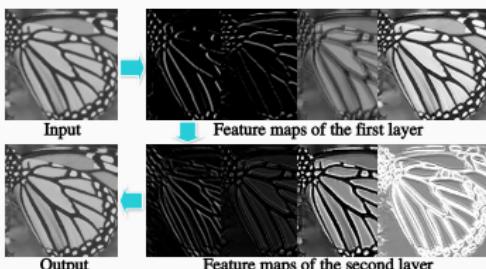
$$\text{PSNR} = 10 \log_{10} \frac{255^2}{\frac{1}{n} \|y - x^{\text{HR}}\|^2}$$

Super-resolution

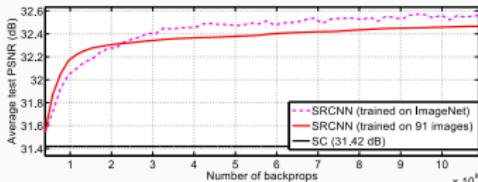
Super-Resolution CNN (SRCNN) (Dong *et al.*, 2016)



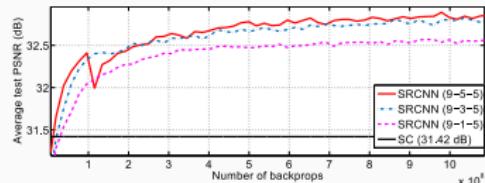
The figure shows the first-layer filters trained on ImageNet with an upscaling factor 3. The filters are organized based on their respective variances.



Example feature maps of different layers.



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

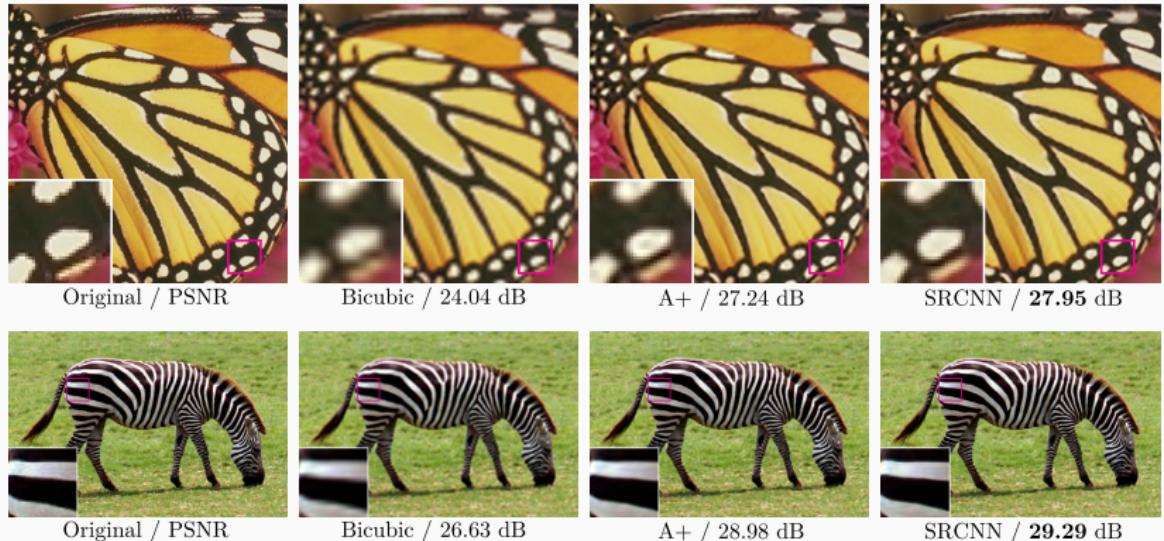


A larger filter size leads to better results.

- 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

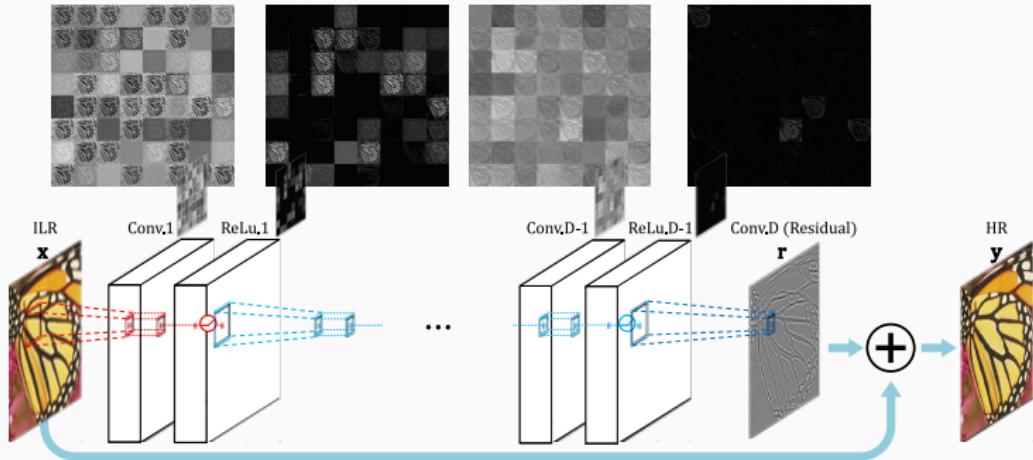
Super-Resolution CNN (SRCNN) (Dong *et al.*, 2016)



×3 upsampling (9 times more pixels)

Super-resolution

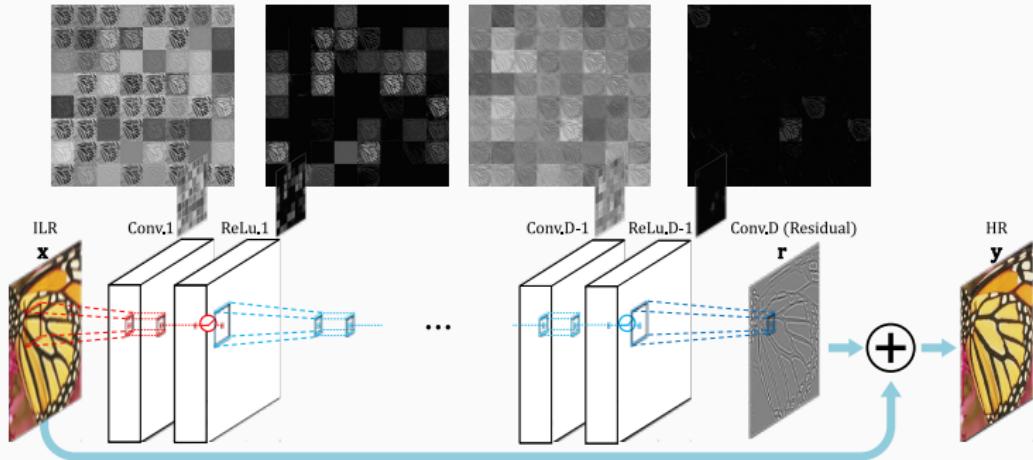
Super-resolution – VDSR (Kim et al., 2016)



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

Super-resolution

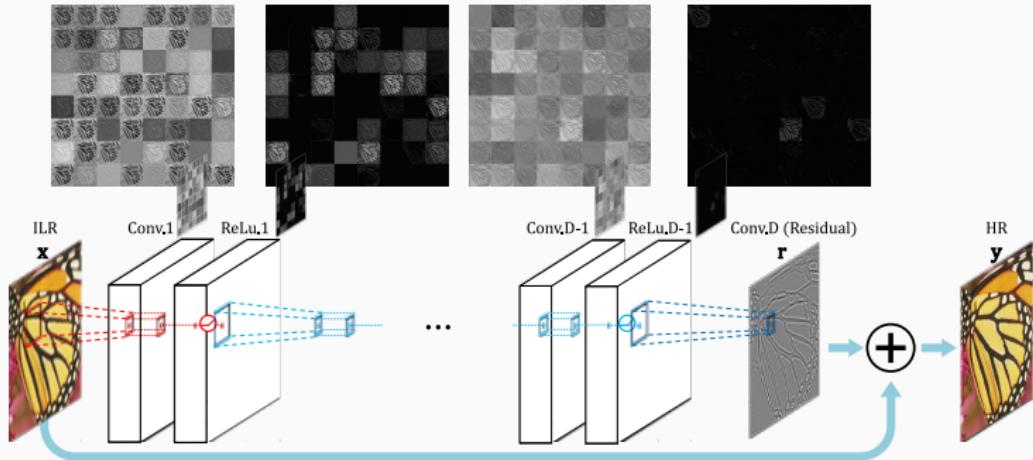
Super-resolution – VDSR (Kim et al., 2016)



- 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

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

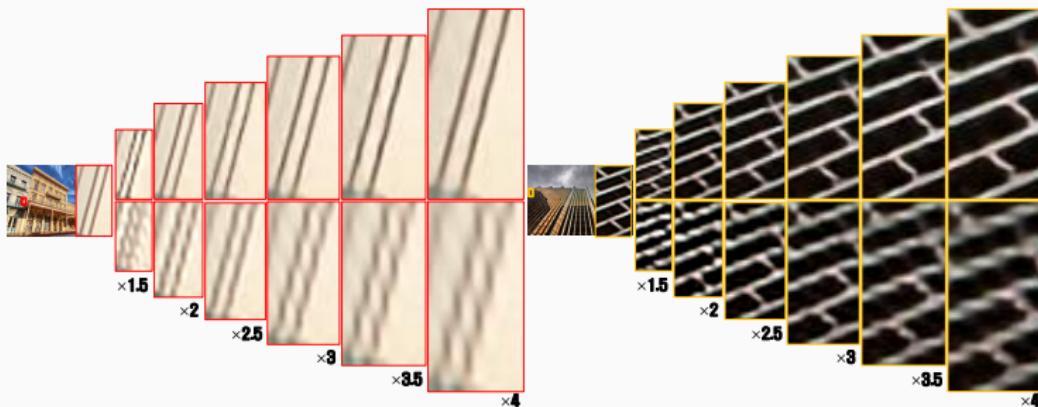


- Inspired from ResNet:
 - Learn the difference between HR and ILR images.
 - Inputs and outputs are highly correlated,
 - Just learn the subtle difference (high frequency details),
 - Allows using high learning rates (with gradient clipping).

Super-resolution

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

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

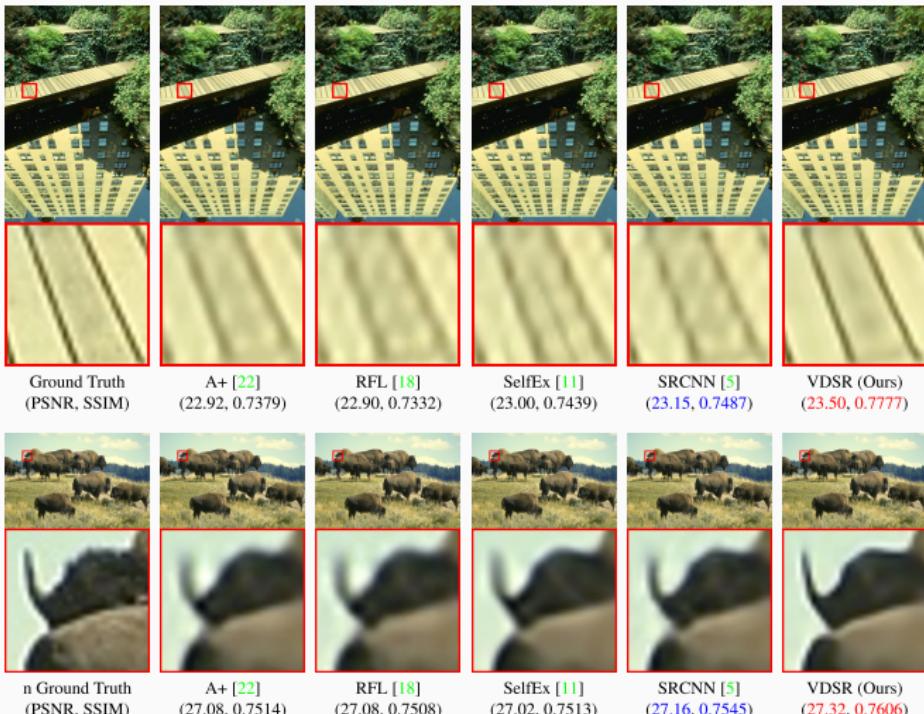


Single model for multiple scales

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

Super-resolution

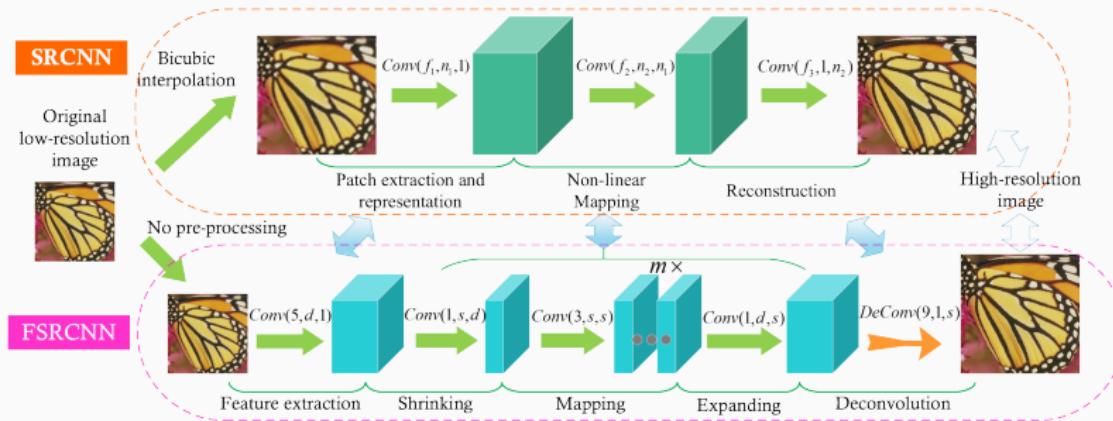
Super-resolution – VDSR (Kim et al., 2016)



×3 upsampling (9× more pixels)

Super-resolution

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



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

Super-resolution

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

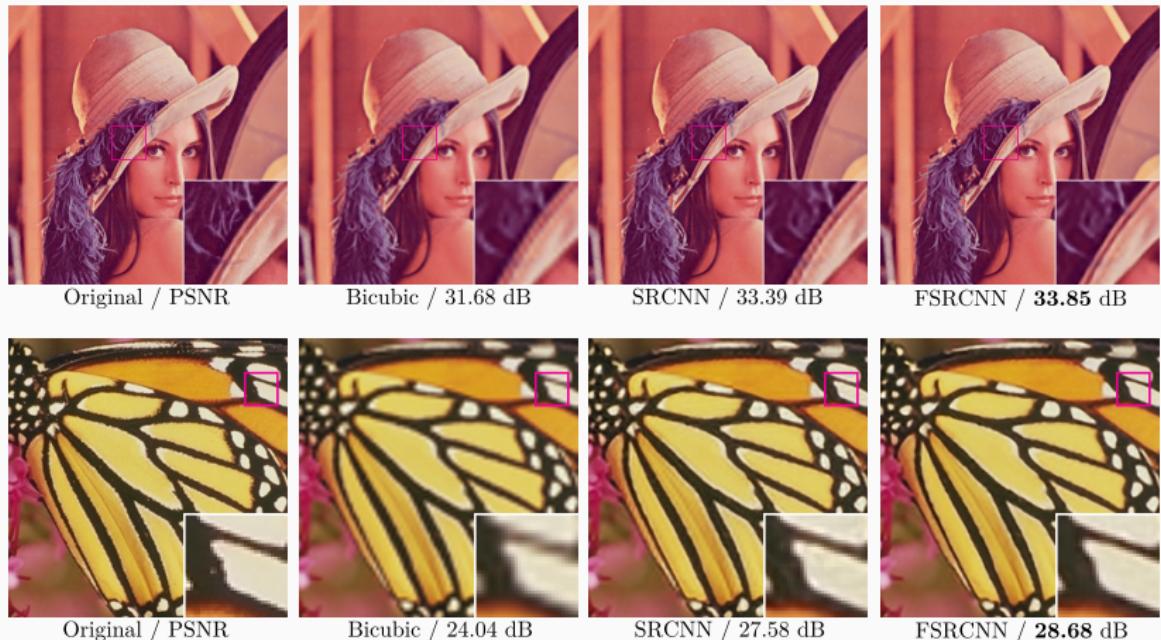
	SRCNN-Ex	Transition State 1	Transition State 2	FSRCNN (56,12,4)
First part	Conv(9,64,1)	Conv(9,64,1)	Conv(9,64,1)	Conv(5,56,1)
Mid part	Conv(5,32,64)	Conv(5,32,64)	Conv(1,12,64)- 4Conv(3,12,12) -Conv(1,64,12)	Conv(1,12,56)- 4Conv(3,12,12) -Conv(1,56,12)
Last part	Conv(5,1,32)	DeConv(9,1,32)	DeConv(9,1,64)	DeConv(9,1,56)
Input size	S_{HR}	S_{LR}	S_{LR}	S_{LR}
Parameters	57184	58976	17088	12464
Speedup	$1 \times$	$8.7 \times$	$30.1 \times$	$41.3 \times$
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

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



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

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

For large upsampling factors ≥ 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,
⇒ 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

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

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

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

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

→ $L_{content}$ 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:

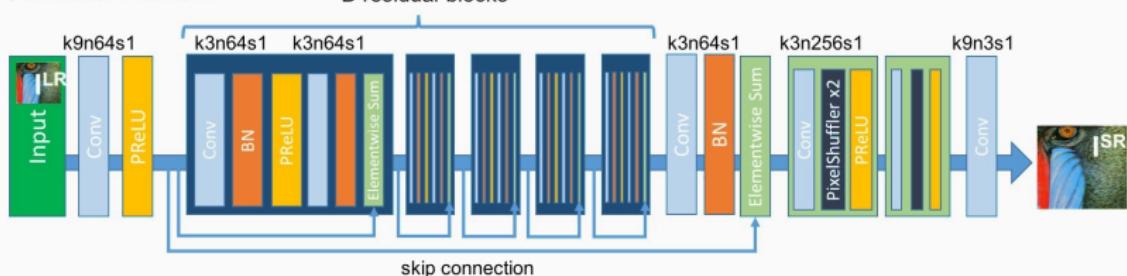
$$L_{content}(\mathbf{x}^{SR}, \mathbf{x}^{HR}) = \|\mathbf{h}^{SR} - \mathbf{h}^{HR}\|_2^2 \quad \text{with} \quad \begin{cases} \mathbf{h}^{SR} = \text{VGG}^L(\mathbf{x}^{SR}) \\ \mathbf{h}^{HR} = \text{VGG}^L(\mathbf{x}^{HR}) \\ \mathbf{x}^{SR} = G_{\theta_{SR}}(\mathbf{x}^{LR}) \end{cases}$$

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

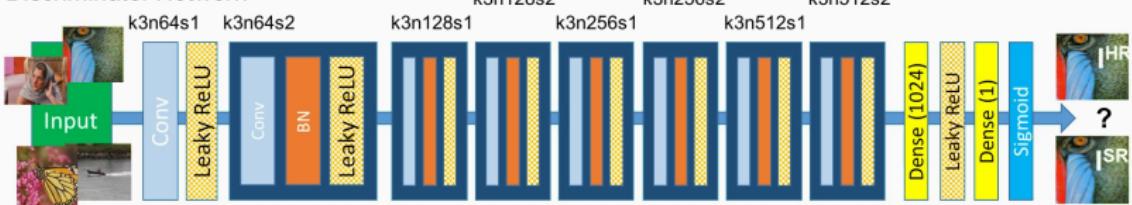
Super-resolution

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

Generator Network



Discriminator Network

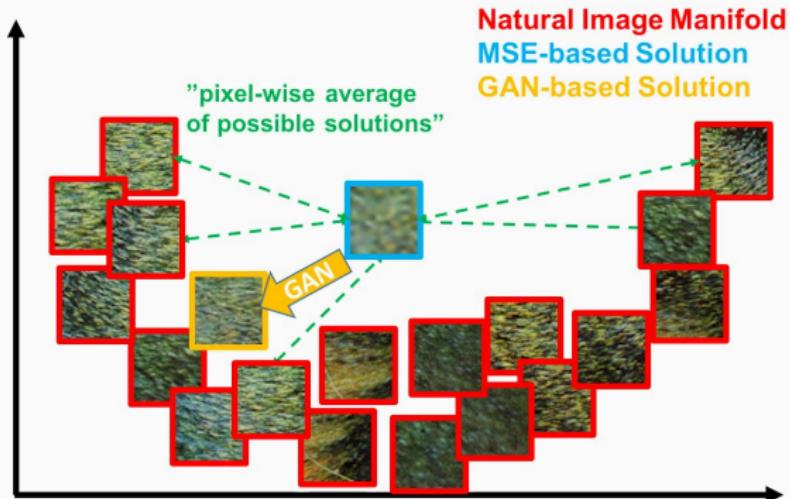


Architecture of Generator and Discriminator Network with corresponding kernel size (k), number of feature maps (n) and stride (s) indicated for each convolutional layer.

Both networks are trained by alternating their gradient based updates.

Super-resolution

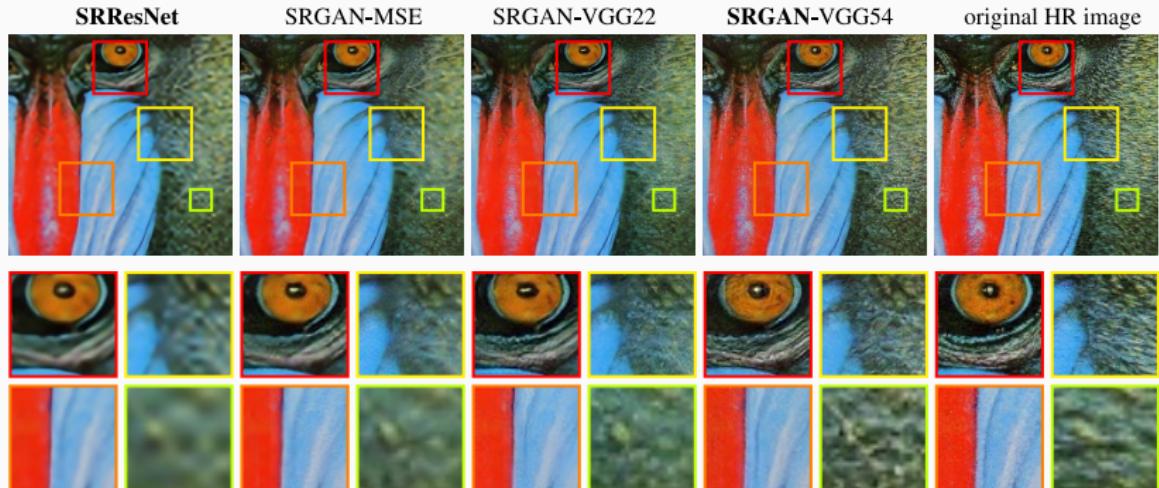
Super-resolution – SRGAN (Twitter, Ledig *et al.*, CVPR 2017)



- 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

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

Super-resolution – SRGAN



$\times 4$ 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

Style transfer (Gatys, Ecker and Bethge, 2015)

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

$$\ell_{\text{content}}^l(\mathbf{x}; \mathbf{y}) = \|\text{VGG}^l(\mathbf{x}) - \text{VGG}^l(\mathbf{y})\|_2^2 \quad (l: \text{a chosen hidden layer})$$

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

Style transfer

Style transfer (Gatys, Ecker and Bethge, 2015)

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

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

- 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)[1]
optimizer = torch.optim.Adam([x], lr=0.01)
for t in range(0, T):
    optimizer.zero_grad()
    hx    = VGGfeatures(x)[1]
    loss = ((hx - hy)**2).mean()
    loss.backward(retain_graph=True)
    optimizer.step()
```

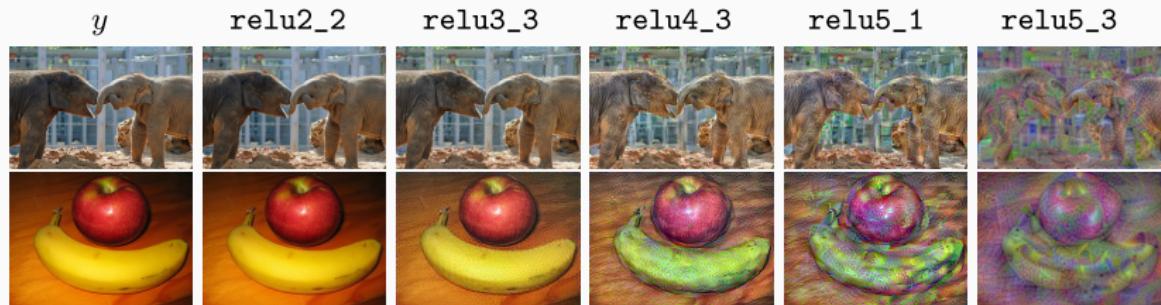
Style transfer

Style transfer (Gatys, Ecker and Bethge, 2015)

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

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

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

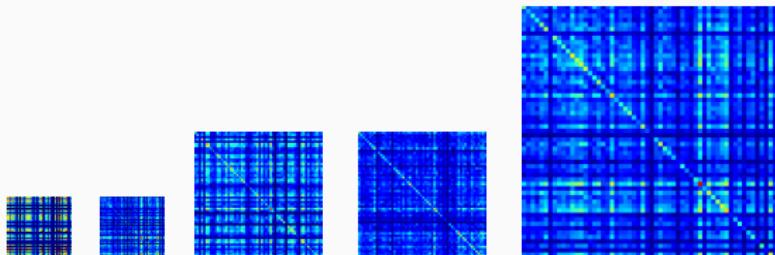


Style transfer (Gatys, Ecker and Bethge, 2015)

- 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:

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

where (i, j) are pixel indices.



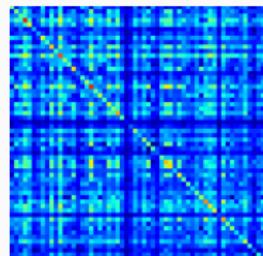
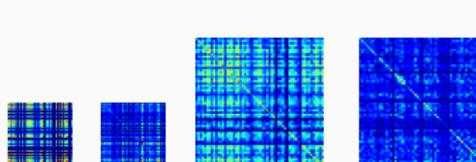
Style transfer

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:

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

where (i, j) are pixel indices.

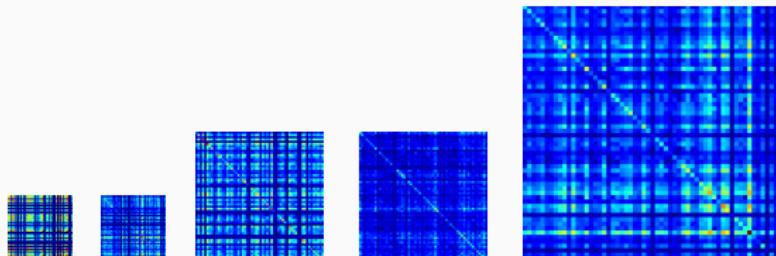


Style transfer (Gatys, Ecker and Bethge, 2015)

- 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:

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

where (i, j) are pixel indices.



Style transfer

Style transfer (Gatys, Ecker and Bethge, 2015)

- Textures y can be synthesized by x minimizing

$$\ell_{\text{style}}^l(x; y) = \|\mathbf{G}^l(x) - \mathbf{G}^l(y)\|_F^2$$

- Again a non-convex optimization problem.

```
x      = torch.rand(y.shape).cuda()
x      = nn.Parameter(x, requires_grad=True)
hy    = VGGfeatures(y)[1].view(C, W * H)
Gy    = torch.mm(hy, hy.t())
for t in range(0, T):
    optimizer.zero_grad()
    hx    = VGGfeatures(x)[1].view(C, W * H)
    Gx    = torch.mm(hx, hx.t())
    loss = ((Gx - Gy) ** 2).sum()
    loss.backward(retain_graph=True)
    optimizer.step()
```

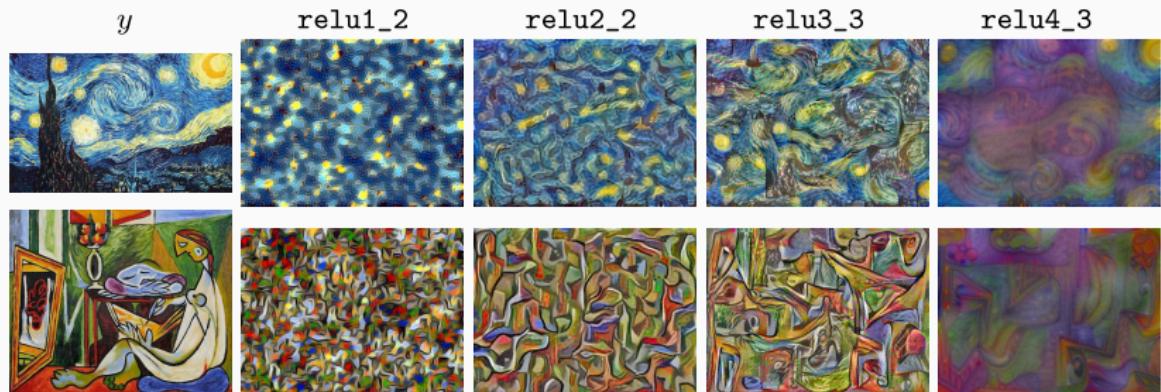
Style transfer

Style transfer (Gatys, Ecker and Bethge, 2015)

- Textures y can be synthesized by x minimizing

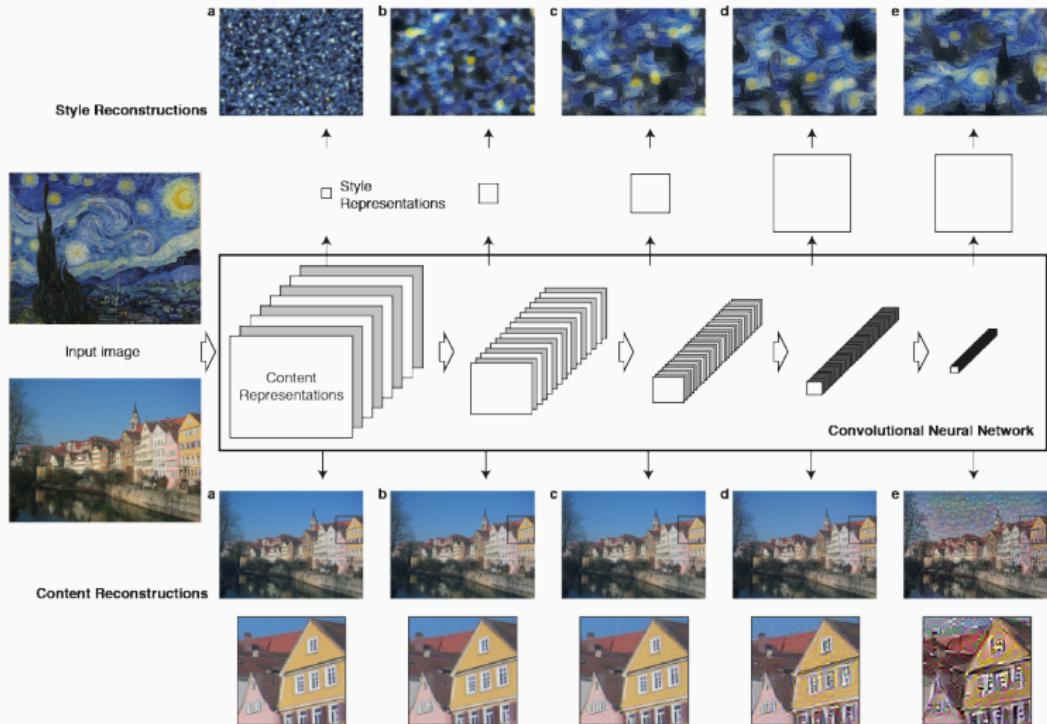
$$\ell_{\text{style}}^l(x; y) = \|\mathbf{G}^l(x) - \mathbf{G}^l(y)\|_F^2$$

- Again a non-convex optimization problem.



Style transfer

Style transfer (Gatys, Ecker and Bethge, 2015)



Style transfer

Style transfer (Gatys, Ecker and Bethge, 2015)

Style transfer:

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

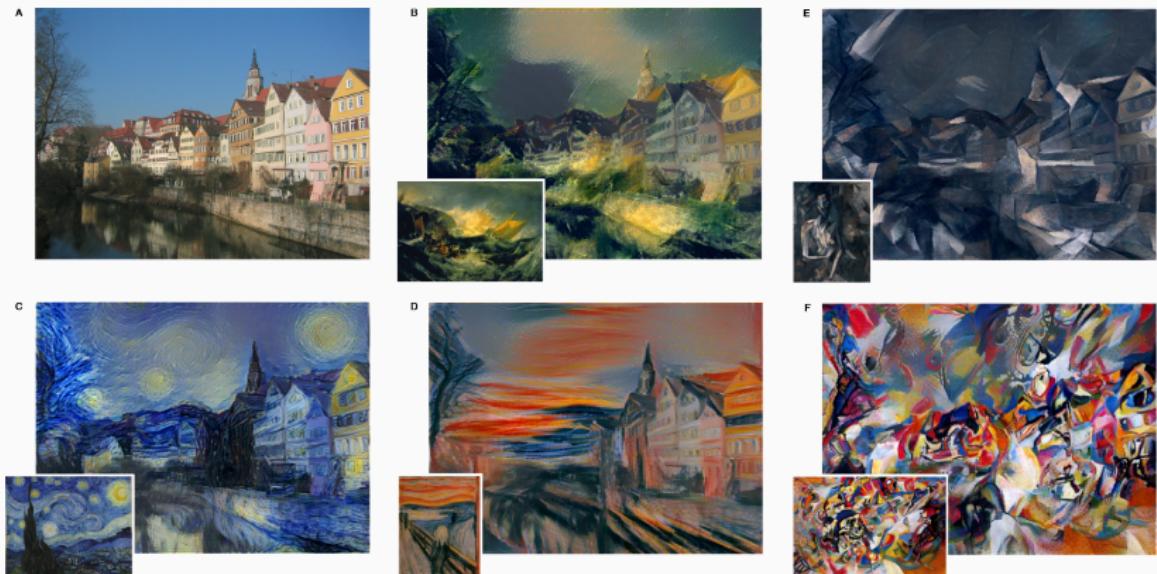
- Look for \boldsymbol{x} such that
 - its content corresponds to \boldsymbol{y}_c ,
→ match VGG features at layer l (typically: $l = 2, 3$ or 4)
 - its style corresponds to \boldsymbol{y}_s ,
→ match VGG correlations at layers 1 to J (typically: $J = 4$ or 5)
- Again a non-convex optimization problem.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

Style transfer

Style transfer (Gatys, Ecker and Bethge, 2015)

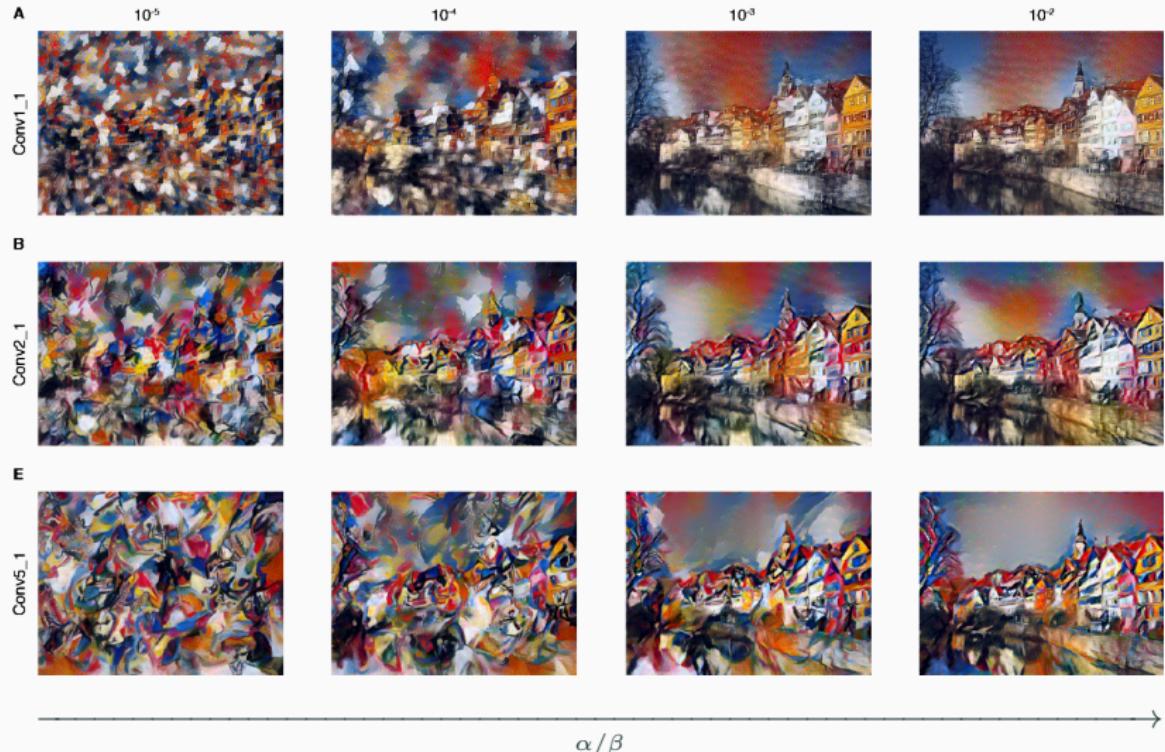
Style transfer:

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



Style transfer

Style transfer (Gatys, Ecker and Bethge, 2015)



Style transfer

Style transfer (Gatys, Ecker and Bethge, 2015)



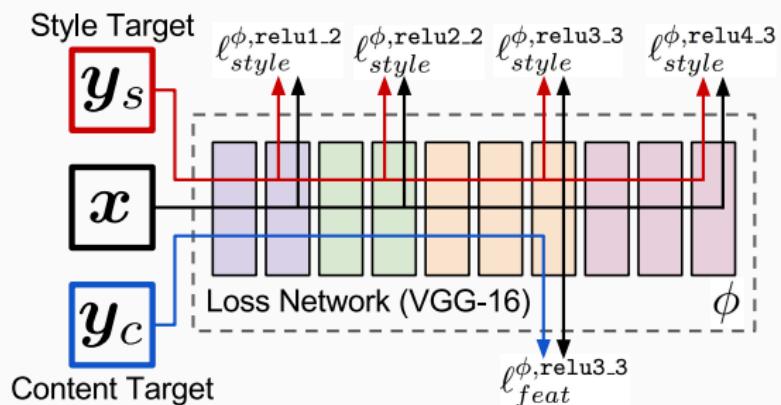
Super easy to implement in PyTorch:
just sum all the losses with weights α and β !

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



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

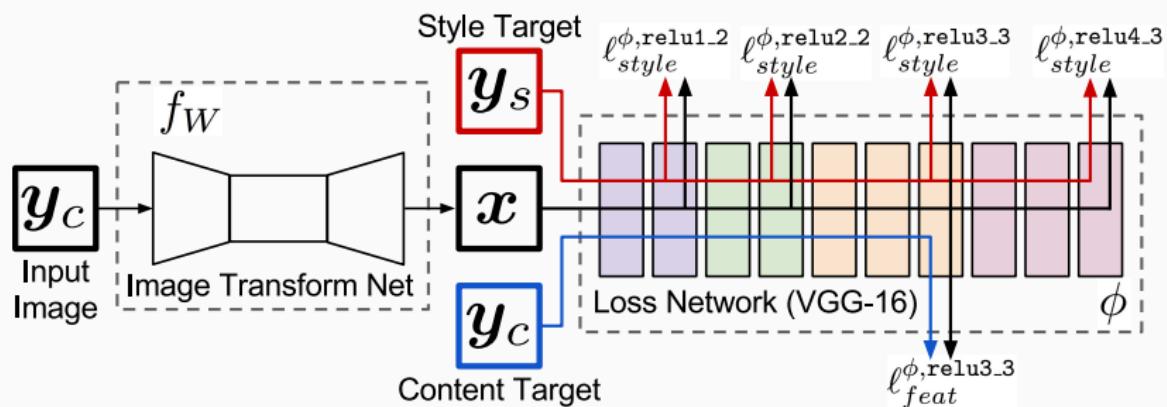
Problem: Optimizing the input x of the VGG network is slow
(requires about 500 forward and backward passes through VGG during runtime).



Style transfer

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).



Solution: train a network to predict x from y_c using Gatys' loss.

Style transfer

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

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

- 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 \mathbf{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