Course Notes: Deep Learning for Visual Computing

Peter Wonka

August 30, 2021

Contents

1 StyleTransfer

1.1 Overview

- Goal: transfer the style of a **style image** $\mathbf{a}$ to a **content image** $\mathbf{p}$
  - result is a **target** (output) image $\mathbf{x}$
- $\mathbf{x}$ should show the content of $\mathbf{p}$ (e.g. KAUST) in the style (colors, brush strokes, etc.) of $\mathbf{a}$
- Example:

content image

style image

target image

- Discussion: difficult to define the problem exactly

1.2 Iterative Approaches

- General idea:
  - Initialize $\mathbf{x}$ with white noise or the content image to improve convergence time
  - Iteratively apply small changes to target image $\mathbf{x}$ using optimization
- **Advantage**
  - No training of a neural network necessary
  - A pre-trained network (e.g. VGG19) is used for feature extraction
- **Disadvantage**
  - Computationally expensive (each style transfer can take minutes)

1.3 Feed Forward Neural Networks

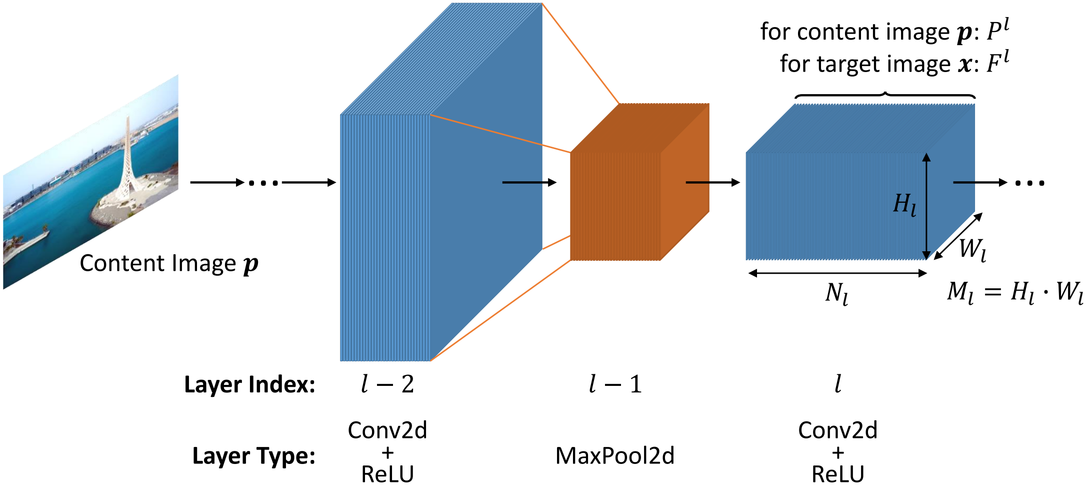
- General idea: Move the computational burden to a learning stage
  - Train a neural network perform style transfer via a single feed forward pass
  - Input is only the content image, output is the stylized image
  - Style images are used during network training
- **Advantage**
  - Fast: Learning has to be done only **once**.
  - Style transfer can then be done within milliseconds
- **Disadvantage**
  - The neural network is trained for only one or a limited number of styles
  - Results are usually not as good as with an iterative approach

1.4 Style Transfer by Gatys et al.

- Literature: Image Style Transfer Using Convolutional Neural Networks (Gatys et al., CVPR 2016)
- Iterative approach.

1.4.1 Notation and Definitions

- N_l: the number of feature maps at layer l
- M_l: the number of scalars in each feature map (channel) at layer l (number of pixels)
- P^l: the feature representations of the content image $\mathbf{p}$ in layer l
- F^l: the feature representations of the target image $\mathbf{x}$ in layer l
  - $\rightarrow M_l =$ height times width of each feature map at layer l
- Notation assumes that each $2d$ feature map (channel) is reshaped into a vector:
  - Rank-3 tensor of layer l is reshaped into matrix $F^l \in \mathbb{R}^{N_l \times M_l}$
  - $\rightarrow F^l_{ij}$: activation value of channel i at pixel position j in layer l


for content image $\boldsymbol{p}$: P^l
for target image $\boldsymbol{x}$: F^l
Content Image $\boldsymbol{p}$
H_l
W_l
N_l
$M_l = H_l \cdot W_l$
Layer Index:
$l-2$
$l-1$
l
Layer Type:
Conv2d + ReLU
MaxPool2d
Conv2d + ReLU

1.4.2 VGG as Feature Extractor

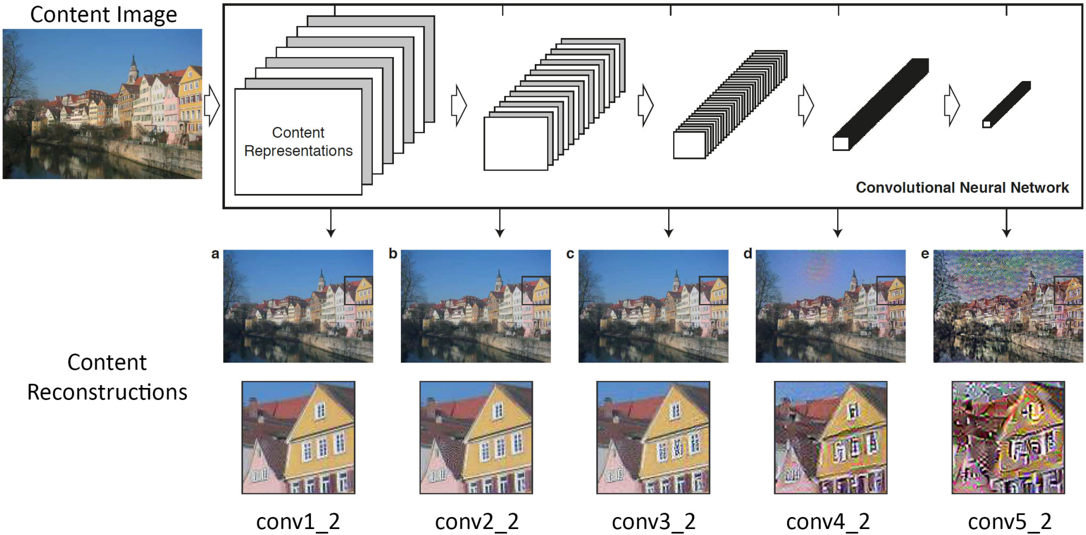
- Authors propose to normalize the VGG network by scaling the weights
  - Mean output of each conv filter over images and positions should be 1
  - Possible for VGG network without changing its output
  - Only ReLU activation functions
  - No normalization or pooling over feature maps
- Authors propose to replace max pooling by average pooling
  - slightly more appealing results?

1.4.3 Content Loss

- We want to keep the content of $\mathbf{p}$ in our target image $\mathbf{x}$
  - minimize the squared difference between their corresponding activations P^l and F^l.
- How to choose a set of layers l_i for feature (tensor) extraction?
  - Perform tests to identify the amount of **content information** in a layer
  - Input to the pre-trained VGG19 network is the content image
  - At a chosen layer l of the network extract the activation tensor P^l
  - Try to reconstruct input image based on the activation tensor P^l

1.4.4 How to Reconstruct Images based on Activation Tensors?

- Initialize using white noise
- Optimize the image (e.g. gradient descent)
  - Loss minimized difference in activation tensor from reference tensor
  - Requires one feed-forward pass through pre-trained network per iteration
- Reconstruction examples:


Content Image
Content
Representations
Convolutional Neural Network
a
b
c
d
e
Content
Reconstructions
conv1_2
conv2_2
conv3_2
conv4_2
conv5_2

1.4.5 Content Loss Details

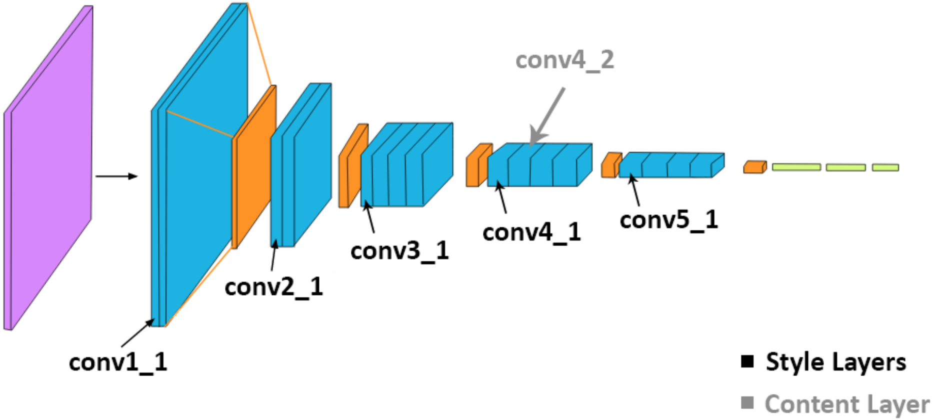
- The **higher layers** of a neural network do not specify detail information
  - Structure (content) is still determined to some degree
  - Details can be specified by style image
  - → choose a higher layer number for **content preservation**
- For example, choose `conv4_2` in VGG19 to define the content loss $\mathcal{L}_{\text{content}}$:

$$\mathcal{L}_{\text{content}}\left(\mathbf{p},\mathbf{x},l\right)=\frac{1}{N_l M_l}\sum_{i,j}\left(F^l_{ij}-P^l_{ij}\right)^2=\operatorname{mean}\left(\left(F^l-P^l\right)\odot\left(F^l-P^l\right)\right) \tag{1.1}$$

- where $\odot$ is the Hadamard product (elementwise) product of two matrices
- Gradient of the content loss w.r.t. the activations in layer l:

$$\frac{\partial\mathcal{L}_{\text{content}}}{\partial F^l_{ij}}=\begin{cases}\frac{2}{N_l M_l}\left(F^l-P^l\right)_{ij} & \text{if } F^l_{ij}>0\\ 0 & \text{if } F^l_{ij}\leq 0\end{cases}$$

- VGG-19 with highlighted layers used for content and style features:



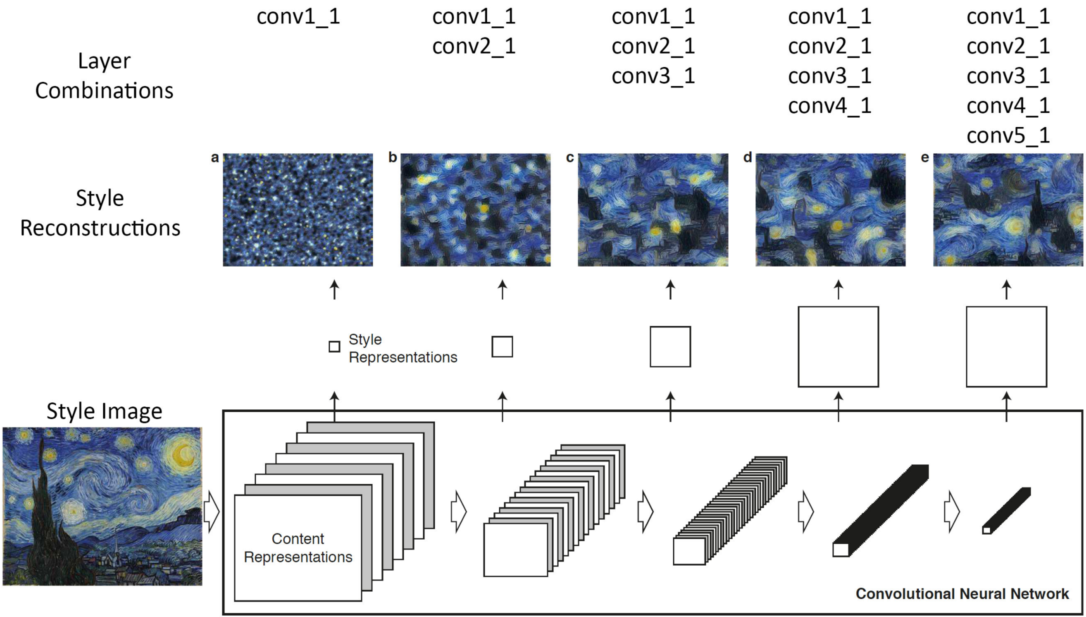
1.4.6 Style Loss

- The goal is to preserve the style of $\mathbf{a}$
  - we **cannot** directly compare the feature maps of $\mathbf{a}$ and $\mathbf{x}$ for this purpose
  - we can compare **feature correlations** which are given by the **Gram matrix** $G^l \in \mathbb{R}^{N_l \times N_l}$:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$

- where G^l_{ij} is the inner product between the vectorised feature maps i and j in layer l
  - vectorisation of the $h \times w$ feature maps in PyTorch by `tensor.view(d, h * w)`
- Again, we can identify the amount of **style information** that is encoded in each layer of the used neural network (VGG19 in our case)
  - Input to the pre-trained VGG19 network is the style image.

- At each layer of the network, we store the responses (activation maps).
- For each layer, we try to reconstruct based on the **Gram matrix** of the stored response.
- Reconstructions (depending on the chosen combination of layers) look like this:



conv1_1
conv1_1
conv2_1
conv1_1
conv2_1
conv3_1
conv1_1
conv2_1
conv3_1
conv4_1
conv1_1
conv2_1
conv3_1
conv4_1
conv5_1
Layer
Combinations
Style
Reconstructions
a
b
c
d
e
Style
Representations
Style Image
Content
Representations
Convolutional Neural Network

1.4.7 Style Loss Details

- the **lower layers** encode **small-scale** style features
- the **higher layers** encode **large-scale** style features
- → by including the feature correlations of **multiple layers**, we obtain a stationary, **multi-scale representation** of the input image, which captures its texture information but not the global arrangement.
- the contribution of layer l to the style loss can then be defined as

$$E_l = \frac{1}{N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l\right)^2 = \frac{1}{M_l^2} \operatorname{mean}\left(\left(G^l - A^l\right) \odot \left(G^l - A^l\right)\right)$$

  - with A^l and G^l being the style representations (given by the corresponding Gram matrix) of the style image $\mathbf{a}$ and the target image $\mathbf{x}$ respectively
- By choosing a suitable weight w_l for each layer l we obtain the style loss $\mathcal{L}_{\text{style}}$ as

$$\mathcal{L}_{\text{style}}(\mathbf{a}, \mathbf{x}) = \sum_{l=0}^{L} w_l E_l$$

- Gradient of E_l w.r.t. the activations in the layer l can be easily derived by

$$\frac{\partial E_l}{\partial F^l_{ij}} = \begin{cases} \frac{2}{N_l^2 M_l^2} \left(\left(F^l \right)^T \left(G^l - A^l \right) \right)_{ji} & \text{if } F^l_{ij} > 0 \\ 0 & \text{if } F^l_{ij} \leq 0 \end{cases}$$

1.4.8 Total Loss

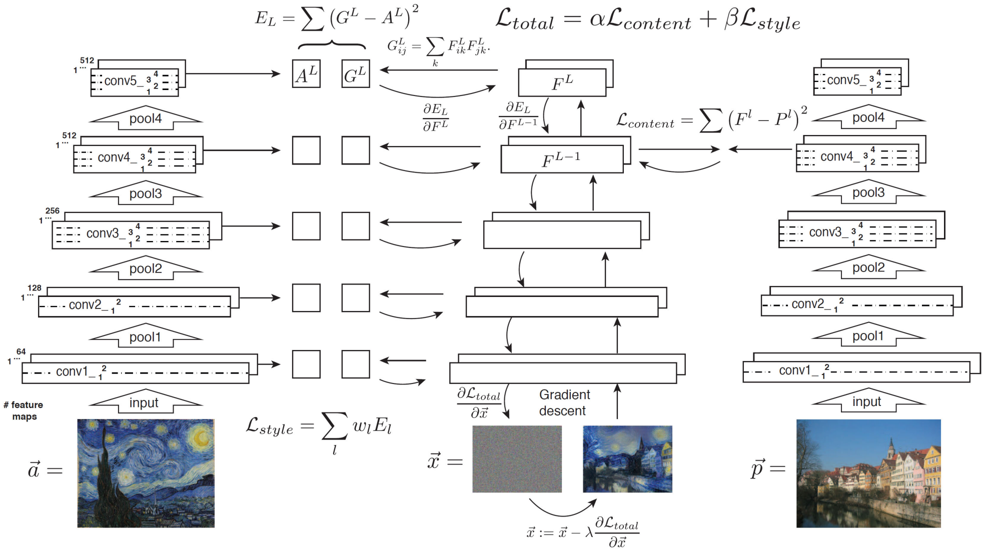
- The total loss is simply a weighted sum of the content loss and the style loss:

$$\mathcal{L}_{\text{total}}\left(\mathbf{p},\mathbf{a},\mathbf{x}\right) = \alpha\mathcal{L}_{\text{content}}\left(\mathbf{p},\mathbf{x}\right) + \beta\mathcal{L}_{\text{style}}\left(\mathbf{a},\mathbf{x}\right)$$

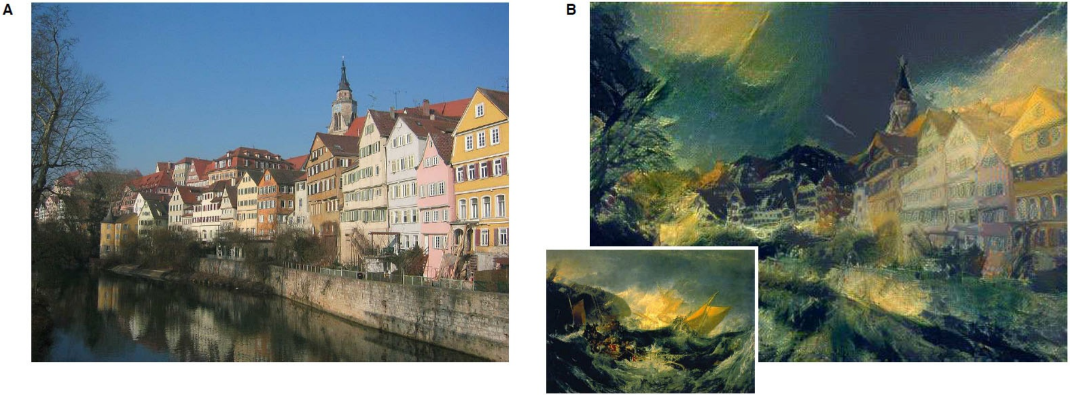
1.4.9 Optimization

- Target image can be initialized with white noise or (to reduce convergence time) the content image.
- Authors suggest:
  - L-BFGS for image optimization
  - a ratio $\frac{\alpha}{\beta}$ of about $1.0e-3$ causing emphasis on the style
  - layer `conv4_2` for content features
  - layers `conv1_1`, `conv2_1`, `conv3_1`, `conv4_1`, and `conv5_1` for style features
  - a style layer weighting of $w_l = 1/5$

1.4.10 Pipeline of the approach by Gatys et al. (CVPR 2016)



1.4.11 Results of the approach by Gatys et al. (CVPR 2016)



C

D

E

F

1.5 Texture Networks

1.5.1 Overview

- Literature: Texture Networks: Feed-forward Synthesis of Textures and Stylized Images (Ulyanov et al., ICML 2016)
- Train a Feed Forward Neural Network that performs the style transfer
  - → this network is referred to as the **generator network**
- A loss is derived from another pre-trained and fixed network
  - → this network is referred to as the **descriptor network** (e.g. VGG-19)
- For each texture or style, a separate generator network must be trained
  - after training, it can synthesize an arbitrary number of images of arbitrary size

1.5.2 Notation and Definitions

- g: generator network (function)
- using g for **texture synthesis**
  - input: noise sample $\mathbf{z}$
  - output: texture $g(\mathbf{z})$
- using g for **style transfer**
  - input: noise sample $\mathbf{z}$ and content image $\mathbf{y}$
  - output: image $g(\mathbf{y}, \mathbf{z})$ where the learned style has been applied to $\mathbf{y}$
- $\mathbf{x_0}$: prototype texture (style image)
- $\mathbf{x}$ is the output image
- $F_i^l(x)$ is the i-th map (feature channel) computed by the l-th convolutional layer of the descriptor CNN (e.g. VGG-19) applied to image x

1.5.3 Texture and Content Loss Functions

- Loss function is derived from the method by Gatys et al.
- A combination of Gram matrices G^l is used as texture descriptor with $l \in L_T$
  - L_T contains selected indices of convolutional layers

$$G^l_{ij}(x) = \langle F^l_i(x), F^l_j(x) \rangle \tag{1.2}$$

- The **texture loss** (= style loss) between images x and x_0 is defined as

$$\mathcal{L}_T(x; x_0) = \sum_{l \in \mathcal{L}_T} \| \boldsymbol{G^l}(\boldsymbol{x}) - \boldsymbol{G^l}(\boldsymbol{x_o}) \|_F^2 \tag{1.3}$$

- The **content loss** is defined as

$$\mathcal{L}_C\left(x; y\right) = \sum_{l \in \mathcal{L}_C} \sum_{i=1}^{N_l} \| \boldsymbol{F^l_i}(\boldsymbol{x}) - \boldsymbol{F^l_i}(\boldsymbol{y}) \|_F^2 \tag{1.4}$$

- where N_l is the number of maps (feature channels) in layer l of the descriptor CNN

1.5.4 Generator Network for Texture Synthesis

- Train generator network g for **texture synthesis**
- Find optimal parameters θ for g given a prototype texture x_0:

$$\theta_{x_0} = \arg\min_{\theta} \mathbb{E}_{z \sim \mathcal{Z}} \left[\mathcal{L}_T \left(g\left(z; \theta\right), x_0 \right) \right] \tag{1.5}$$

- Input: set of K random tensors z_i of different size:

$$z_i \in \mathbb{R}^{\frac{M}{2^i} \times \frac{M}{2^i}}, i = 0, 1, \ldots, K - 1 \tag{1.6}$$

  - Authors use $M = 256$ and $K = 5$
- → **multi-scale architecture**
- Loss: only texture loss, no content loss, no content image

1.5.5 Learning Algorithm for Texture Synthesis

- optimize the objective (1.5) using stochastic gradient descent (SGD).
- At each iteration SGD:
  - Draw a mini-batch of noise vectors $\mathbf{z}_k$; $k = 1, \ldots, B$
  - Forward evaluation of generator network g to obtain images $\mathbf{x_k} = g(\mathbf{z_k}; \theta)$
  - Forward evaluation of descriptor network to obtain Gram matrices $G^l(\mathbf{x_k})$, $l \in L_T$
  - Computation of the loss (1.5).
  - gradient computation using backpropagation
    - of the texture loss w.r.t. the generator network parameters θ
  - update the parameters using the gradient

1.5.6 Generator Network for Style Transfer

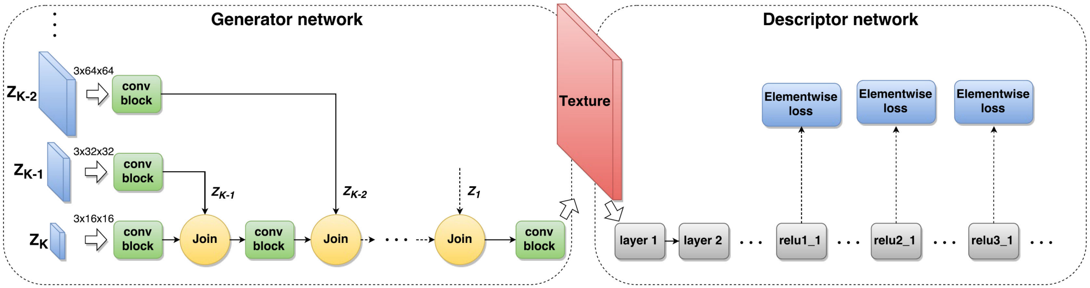
- Similar to texture synthesis with some modifications:
  - Input: noise tensors z_i concatenated with downsampled versions of the content image y
  - Number of random input tensors K is increased to 6
- **Learning** process
  - sample noise vectors $z_i \sim \mathcal{Z}$
  - sample natural images $y_i \sim \mathcal{Y}$
  - compute loss:

$$\theta_{x_0} = \underset{\theta}{\arg\min}\, \mathbb{E}_{z \sim \mathcal{Z}; y \sim \mathcal{Y}} \left[\mathcal{L}_T \left(g \left(y, z; \theta \right), x_0 \right) + \alpha \mathcal{L}_C \left(g \left(y, z; \theta \right), y \right) \right] \tag{1.7}$$

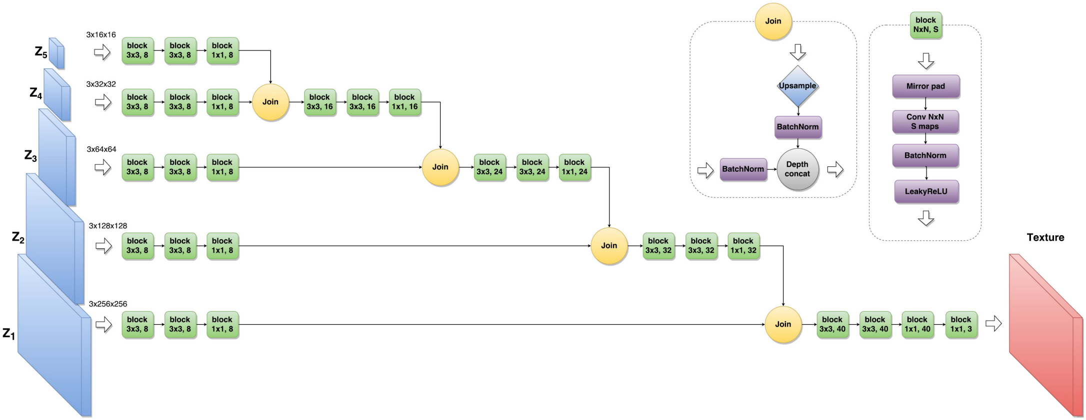
  - update network parameters θ of the generator $g(y_i; z_i; \theta)$ using backpropagation:

1.5.7 Overview of the proposed architecture

- generator network g is fully convolutional
  - $\rightarrow$ independent to input resolution during test time
- for the descriptor network, a pre-trained network (e.g VGG-19) is used



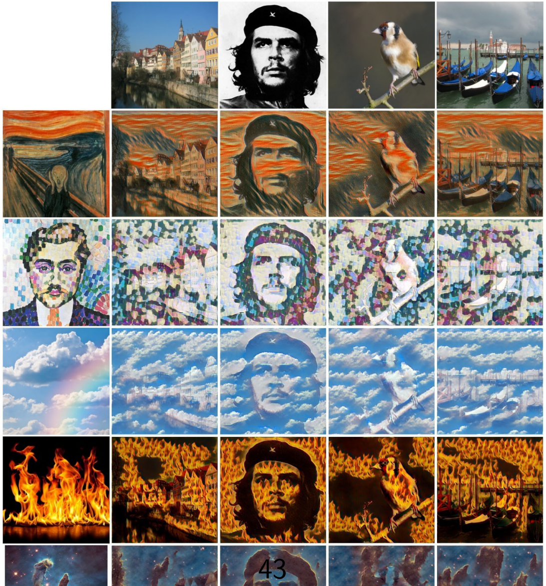
1.5.8 Generator Network in Detail



1.5.9 Technical Details

- Network weights are initialized using Xavier's method
- Training is done using the Adam optimizer for 2000 iterations
- initial learning rate of 0.1 that is reduced by a factor 0.7 at iteration 1000 and then again every 200 iterations
- batch size $= 16$
- choice of layers from VGG-19 for content and style loss similar to Gatys et al.
- Training of the network takes two hours on an NVidia Tesla K40

1.5.10 Results of Ulyanov et al., ICML 2016



1.5.11 Comparison of Ulyanov et al. to Gatys et al.

- While orders of magnitudes faster, the perceptible quality is inferior to output of Gatys et al.


Content Texture nets (ours) Gatys et al. Style

1.6 Perceptual Losses for Real-Time Style Transfer and Super-Resolution

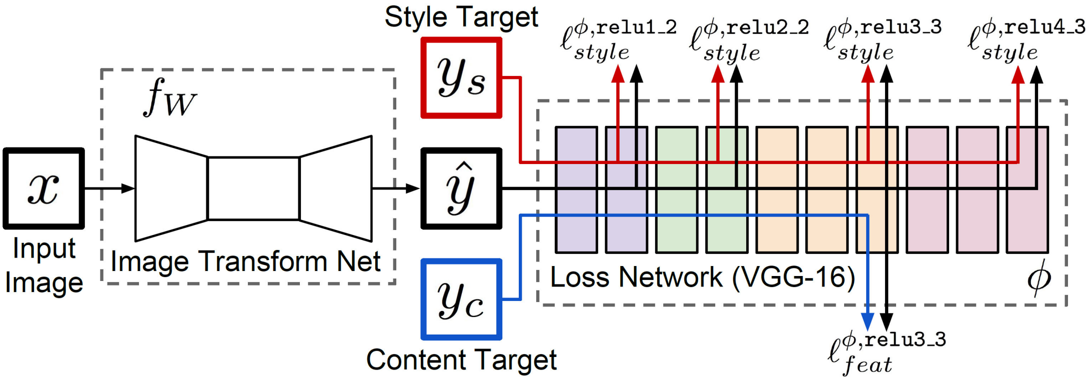
Johnson et al., Perceptual Losses for Real-Time Style Transfer and Super-Resolution, ECCV 2016)

1.6.1 Overview

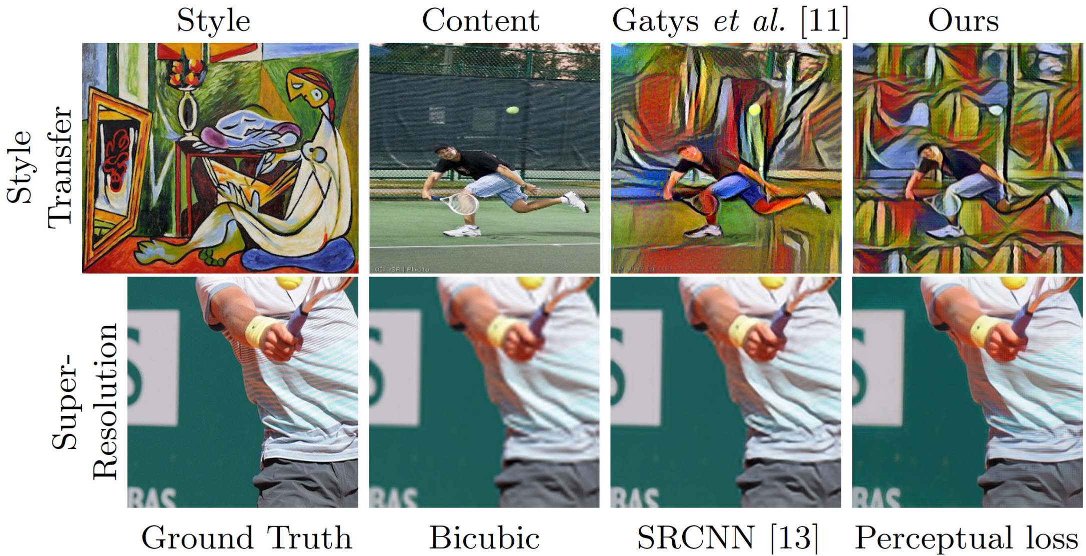
- Very similar to **Texture Networks** of Ulyanov et al.
- Feed-forward network trained either for **style transfer** or **super-resolution**
- Main difference: network architecture
  - network body comprises five residual blocks
  - all non-residual convolutional layers are followed by batch normalization and ReLU nonlinearities
  - output layer uses a scaled $\tanh$ to ensure valid pixel range
  - first and last layers use 9×9 kernels
  - all other convolutional layers use 3×3 kernels

1.6.2 Pipeline

- For **style transfer**
    - the input image x equals the content target y_c
    - the output image $\hat{y}$ should combine the content of $x = y_c$ with the style of y_s
    - one network is trained per style target
- For **super-resolution**
    - the input x is a low-resolution input
    - the content target y_c is the ground-truth high-resolution image
    - the style reconstruction loss is not used
    - one network is trained per super-resolution factor
- the image transform net f_W is the network to be trained
- the loss network a pre-trained network (e.g. VGG-16)


Style Target
y_s
$\ell_{style}^{\phi,\mathtt{relu1\_2}}$
$\ell_{style}^{\phi,\mathtt{relu2\_2}}$
$\ell_{style}^{\phi,\mathtt{relu3\_3}}$
$\ell_{style}^{\phi,\mathtt{relu4\_3}}$
f_W
x
Input Image
Image Transform Net
$\hat{y}$
Loss Network (VGG-16)
ϕ
y_c
Content Target
$\ell_{feat}^{\phi,\mathtt{relu3\_3}}$

1.6.3 Results

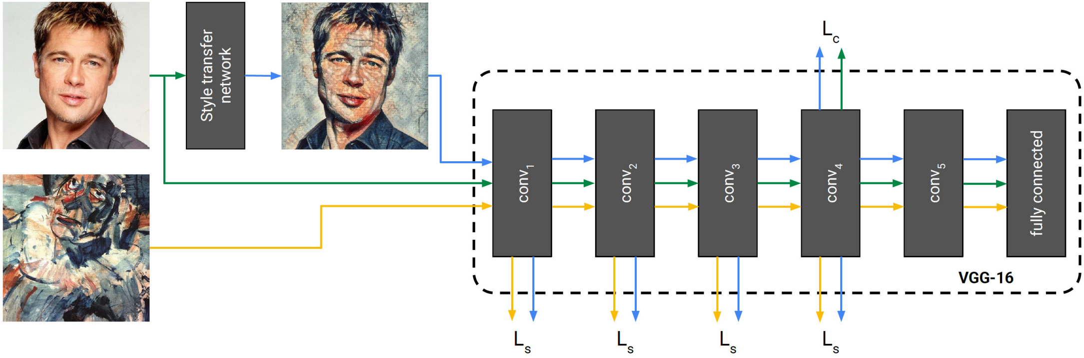

Style
Content
Gatys et al. [11]
Ours
Style Transfer
Super-Resolution
Ground Truth
Bicubic
SRCNN [13]
Perceptual loss

1.7 A Learned Representation For Artistic Style

- Literature: Dumoulin et al., A Learned Representation For Artistic Style

1.7.1 Overview

- also trains a feed-forward neural network for style transfer
- a single network is trained for up to **32 styles**
- key contribution is the introduction of **conditional instance normalization**
- model reduces each style image into a point in an embedding space
- training procedure very similar to Johnson et al. and Ulyanov et al.
  - VGG-16 is used for feature extraction
  - content loss $\mathcal{L}_c$ is computed using response values
  - style loss $\mathcal{L}_s$ is computed using a set of Gram matrices
- very similar network architecture to Johnson et al.


Style transfer network
L_c
conv$_1$
conv$_2$
conv$_3$
conv$_4$
conv$_5$
fully connected
VGG-16
L_s
L_s
L_s
L_s

1.7.2 Notation and Definitions

- the goal is to find a pastiche image p
  - pastiche: an artistic work in a style that imitates that of another work, artist, or period
- the content is given as a content image c
- the style is given as a style image s
- the style transfer network to be trained is referred to as T

1.7.3 Loss Function

- **Style loss**:

$$\mathcal{L}_s(p) = \sum_{i \in \mathcal{S}} \frac{1}{U_i} \|G(\phi_i(p)) - G(\phi_i(s))\|_F^2 \tag{1.8}$$

- **Content loss**:

$$\mathcal{L}_c(p) = \sum_{i \in \mathcal{S}} \frac{1}{U_j} \|G(\phi_j(p)) - G(\phi_j(c))\|_2^2 \tag{1.9}$$

- Total loss using the content image c as input to the transfer network T:

$$\mathcal{L}(s,c) = \lambda_s \mathcal{L}_s(T(c)) + \lambda_c \mathcal{L}_c(T(c)) \tag{1.10}$$

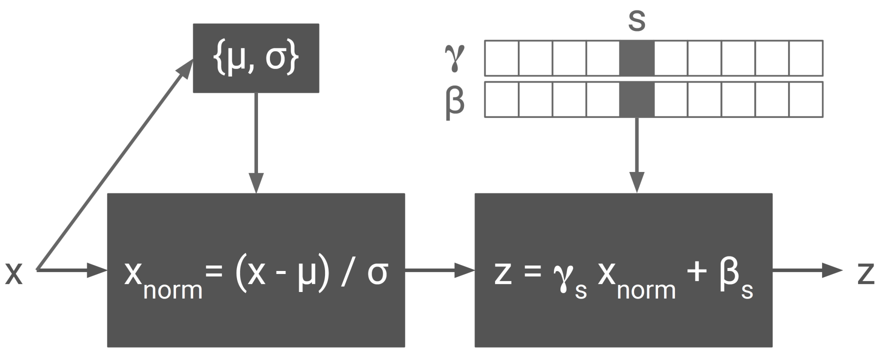
- $\phi_l(x)$ are the classifier activations at layer l
- U_l is the total number of units at layer l
- $G(\phi_l(x))$ is the Gram matrix associated with the layer l activations

1.7.4 Conditional Instance Normalization

- Intuition behind the proposed method:
  - Many styles probably share some degree of computation
  - Wasteful to treat a set of N impressionist paintings as completely separate styles
- **Goal**: transform a layer's activation tensor x into a normalized activation z specific to painting style s
- conditioning on a style s is achieved as follows:

$$z = \gamma_s \frac{x - \mu}{\sigma} + \beta_s$$

  - where μ and σ are x's mean and standard deviation taken across spatial axes
  - γ_s and β_s are obtained by selecting the row corresponding to s in the γ and β matrices:



- for N styles γ and β are $N \times C$ matrices where
  - N is the number of styles being modeled
  - C is the number of output feature maps (channels)
- the input activation x is normalized across both spatial dimensions and subsequently scaled and shifted using style-dependent parameter vectors γ_s, β_s where s indexes

the style label.

- **Benefit of this approach**
  - one can stylize a single image into N painting styles with a single feed forward pass of the network with a batch size of N
  - a single-style network requires N feed forward passes to perform N style transfers
- conditional instance normalization presents the advantage that integrating an $N+1^{th}$ style to the network is cheap because of the very small number of parameters to train ($\sim 3K$ for a typical network setup)

1.7.5 Style Transfer Network Hyperparameters

Operation	Kernel size	Stride	Feature maps	Padding	Nonlinearity
Network – $256 \times 256 \times 3$ input					
Convolution	9	1	32	SAME	ReLU
Convolution	3	2	64	SAME	ReLU
Convolution	3	2	128	SAME	ReLU
Residual block			128		
Residual block			128		
Residual block			128		
Residual block			128		
Residual block			128		
Upsampling			64		
Upsampling			32		
Convolution	9	1	3	SAME	Sigmoid
Residual block – C feature maps					
Convolution	3	1	C	SAME	ReLU
Convolution	3	1	C	SAME	Linear
	Add the input and the output				
Upsampling – C feature maps					
	Nearest-neighbor interpolation, factor 2				
Convolution	3	1	C	SAME	ReLU
Padding mode	REFLECT				
Normalization	Conditional instance normalization after every convolution				
Optimizer	Adam (Kingma & Ba, 2014) ($\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$)				
Parameter updates	40,000				
Batch size	16				
Weight initialization	Isotropic gaussian ($\mu = 0$, $\sigma = 0.01$)				

1.7.6 Results

- N-styles network can arbitrarily combine artistic styles.
- In the example below four styles are combined, shown in the corners.
- Each pastiche corresponds to a different convex combination of the four styles' γ and β values.