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

Author: Justin Johnson, Alexandre Alahi, Li Fei-Fei

References: https://arxiv.org/abs/1603.08155

Overview of Real-Time Style Transfer:

Problem: Image transformation task. Super-resolution and style transfer

Solution 1: Train a supervised feed forward CNN using per-pixel loss

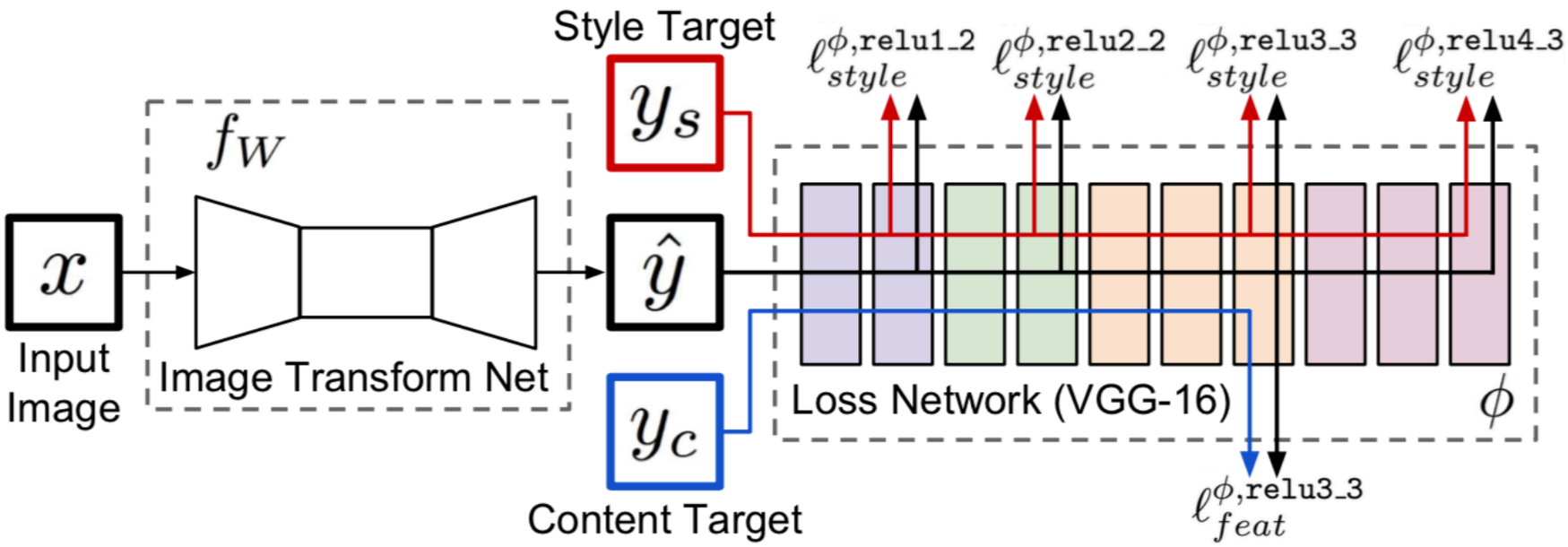
- Pros: efficient at test time
- Cons: per-pixel loss does not capture perceptual difference

Solution 2: Minimize perceptual loss functions

- Pros: generate image based on high-level image feature representations
- Cons: solving optimization problem is slow

Propose the use of perceptual loss function(over per-pixel loss function) for training a feed-toward networks for image transformation tasks.

System Overview:



fw: a deep residual convolutional neural network

3 convolution layer, 5 residual layer, 3 deconvolutional layers

Step1: $\hat{y} = fw(x)$

Step2: two loss function measures the difference between the output image and the target image: $l_i(\hat{y}, y_i)$

Step3: gradient descent to minimize a weighted combination of loss function

$$W* = arg \min_{w} E_{x,\{y_i\}} \left[\sum_{i=1} \lambda_i l_i(fw(x), y_i) \right]$$

A Neural Algorithm of Artistic Style(Last Presentation)

Style Image (S)

Content Image (C)

Generated Image (G)

Loss Function: $L_{total}(S, C, G) = \alpha L_{content}(C, G) + \beta L_{style}(S, G)$

Content Loss: $L_{content}(C, G) = \frac{1}{2} \sum_{ij} (a_{ij}^{l\_C} - a_{ij}^{l\_G})^2$

Style Loss: $L_{style}(S, G) = \sum_{l} w_l E_l$

Feature correlations(Gram Matrix):

$$G_{ij}^{l\_S} = \sum_{k} a_{ik} a_{jk'}$$

$$G_{ij}^{l\_G} = \sum_{k} a_{ik} a_{jk'}$$

The contribution of layer l:

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{ij} (G_{ij}^{l\_S} - G_{ij}^{l\_G})^2$$

Neural Style Transfer Algorithm:

Step 1: Randomly initialize G

Step 2: Minimize $L_{total}(S, C, G)$ through gradient descent.

Perceptual Loss Functions

$$W* = arg \min_{w} E_{x,\{y_i\}} \left[\sum_{i=1} \lambda_i l_i(fw(x), y_i) \right] \qquad l_i(\hat{y}, y_i)$$

Feature Reconstruction Loss:

$$\ell^{\phi,j}_{feat}(\hat{y}, y) = \frac{1}{C_j H_j W_j} \|\phi_j(\hat{y}) - \phi_j(y)\|_2^2$$

Style Reconstruction Loss:

$$G^{\phi}_j(x)_{c,c'} = \frac{1}{C_j H_j W_j} \sum_{h=1}^{H_j} \sum_{w=1}^{W_j} \phi_j(x)_{h,w,c} \phi_j(x)_{h,w,c'}.$$

$$\ell^{\phi,j}_{style}(\hat{y}, y) = \|G^{\phi}_j(\hat{y}) - G^{\phi}_j(y)\|_F^2.$$

Pixel loss: $\ell_{pixel}(\hat{y}, y) = \|\hat{y} - y\|_2^2 / CHW.$

Total Variation Regularization

Real-time style transfer training details and results

- Network trained on COCO dataset
- Resize image to 256x256
- Batch size 4 for 40000 iterations
- Adam with learning rate of 1X10^-3
- No weight decay and dropout: no overfit within two epochs
- Use VGG layer relu2_2 for style reconstruction
- Use VGG layers relu1_2, relu2_2, relu3_3, and relu4_3 for style reconstruction
- Our implementation uses Torch and cuDNN
- Training takes roughly 4 hours on a single GTX Titan X GPU

Image Size	Gatys *et al* [10] 100	300	500	Ours	Speedup 100	300	500
256 × 256	3.17	9.52s	15.86s	**0.015s**	212x	636x	**1060x**
512 × 512	10.97	32.91s	54.85s	**0.05s**	205x	615x	**1026x**
1024 × 1024	42.89	128.66s	214.44s	**0.21s**	208x	625x	**1042x**