

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

Justin Johnson, Alexandre Alahi, Li Fei-Fei

By: Rey Reza Wiyatno

## Key Findings

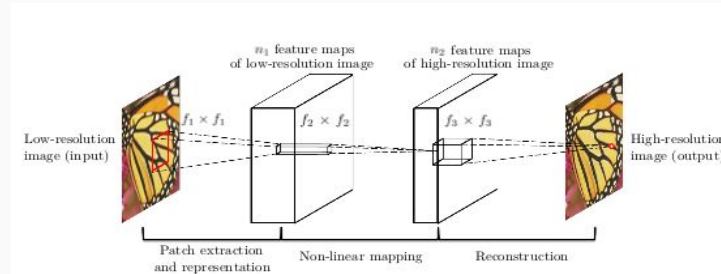
- Perceptual losses vs. per-pixel losses
- Super-resolution using perceptual losses
- Real-time style transfer using perceptual losses

There are 3 key findings in this paper:

1. The authors mentioned how perceptual loss can be advantageous compared to the usual per-pixel loss
2. The authors proposed a generative transformation network to perform super-resolution but using perceptual loss
3. The authors showed how we can create a generative transformation network to perform style transfer while also using the perceptual loss for the content loss instead of per-pixel loss

# Super-Resolution (Background)

## Introduction

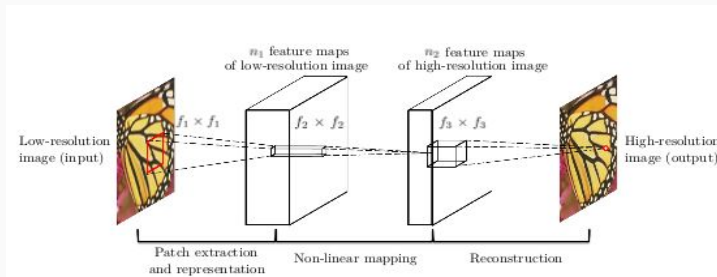


SRCNN (Dong et al., 2015)

This is one of the early works of performing super-resolution using convolutional neural networks. The network is called Super Resolution Convolutional Neural Networks (SRCNN)

# Super-Resolution (Background)

## Per-Pixel Loss & Fully Convolutional Network



SRCNN (Dong et al., 2015)

$$L(\Theta) = \frac{1}{n} \sum_{i=1}^n \|F(Y_i; \Theta) - X_i\|^2$$

$n$  = Number of training samples

$Y_i$  = Low-res image (input)

$F(Y_i; \Theta)$  = High-res image (output)

$X_i$  = Target image (ground truth)

As we see from the equation in this slide, the loss function in SRCNN is a per-pixel loss that tries to minimize  $L_2$  distance between high resolution image and low-resolution image.

# Super-Resolution (Background)

## What Is Wrong with Per-Pixel Losses?

Per-pixel losses do not capture perceptual differences (e.g. images being shifted from each other by 1 or more pixel in one direction can produce high MSE although they are visually indistinguishable)

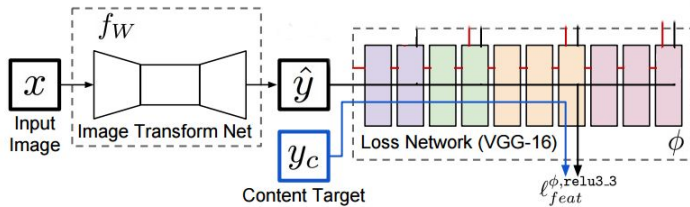
$$\begin{pmatrix} 7 & 2 & 4 & 8 & 9 \\ 6 & 0 & 9 & 2 & 7 \\ 2 & 9 & 2 & 1 & 1 \\ 1 & 8 & 0 & 8 & 6 \end{pmatrix} - \begin{pmatrix} 7 & 7 & 2 & 4 & 8 \\ 6 & 6 & 0 & 9 & 2 \\ 2 & 2 & 9 & 2 & 1 \\ 1 & 1 & 8 & 0 & 8 \end{pmatrix} = \begin{pmatrix} 0 & -5 & 2 & 4 & 1 \\ 0 & -6 & 9 & -7 & 5 \\ 0 & 7 & -7 & -1 & 0 \\ 0 & 7 & -8 & 8 & -2 \end{pmatrix}$$

$$\text{MSE} = 25.85$$

Intuitively, we can tell that  $L_2$  (or mean squared error (MSE)) in pixel space is not a good perceptual metric. One can shift an image by 1 pixel to the left/right, and the MSE can be a large number although the image may not look different at all to a human.

# Super Resolution

## Proposed Idea - Perceptual Losses



Edited figure from Johnson et al. (2016)

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

$\phi_j(y)$  = Output of the network  $\phi$  at layer  $j$ , taking  $y$  as the input to the network

$C_j, H_j, W_j$  = Number of tensor's channel, height, and width at layer  $j$

$\hat{y}$  = High-res image (output of  $f_W$ )

$y_c$  = Target image (ground truth) =  $y$

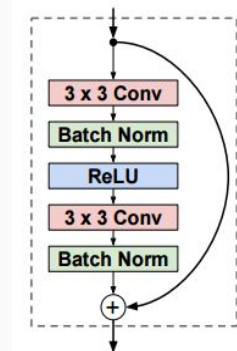
The authors then proposed to measure  $L_2$  distance in latent space instead of pixel space which is very similar to the loss functions used by Gatys et al. (2015). In order to do this, a pre-trained network, in this case VGG16 (Simonyan & Zisserman, 2014), was used just as a feature extractor. In other words, we want to pass an original input and take the output of VGG16 (loss network in the figure) at some intermediate layer (denoted as  $\phi_j(\cdot)$ ). We also want to pass the output of the super-resolution network and pass it to the feature extractor network (loss network in the figure) and take the output at the same intermediate layer. We then compare the 2 outputs and try to minimize the  $L_2$  distance between these 2 outputs. In practice, one may pick the output of more than one intermediate layer of the loss network, compute the loss at each layer separately, and take the average.

# Super-Resolution

Image Transform Net ( $f_w$ )

×4		×8	
Layer	Activation size	Layer	Activation size
Input	$3 \times 72 \times 72$	Input	$3 \times 36 \times 36$
$64 \times 9 \times 9$ conv, stride 1	$64 \times 72 \times 72$	$64 \times 9 \times 9$ conv, stride 1	$64 \times 36 \times 36$
Residual block, 64 filters	$64 \times 72 \times 72$	Residual block, 64 filters	$64 \times 36 \times 36$
Residual block, 64 filters	$64 \times 72 \times 72$	Residual block, 64 filters	$64 \times 36 \times 36$
Residual block, 64 filters	$64 \times 72 \times 72$	Residual block, 64 filters	$64 \times 36 \times 36$
Residual block, 64 filters	$64 \times 72 \times 72$	Residual block, 64 filters	$64 \times 36 \times 36$
$64 \times 3 \times 3$ conv, stride 1/2	$64 \times 144 \times 144$	$64 \times 3 \times 3$ conv, stride 1/2	$64 \times 72 \times 72$
$64 \times 3 \times 3$ conv, stride 1/2	$64 \times 288 \times 288$	$64 \times 3 \times 3$ conv, stride 1/2	$64 \times 144 \times 144$
$3 \times 9 \times 9$ conv, stride 1	$3 \times 288 \times 288$	$64 \times 3 \times 3$ conv, stride 1/2	$64 \times 288 \times 288$
-	-	$3 \times 9 \times 9$ conv, stride 1	$3 \times 288 \times 288$

Network architecture used (Johnson et al., 2016)



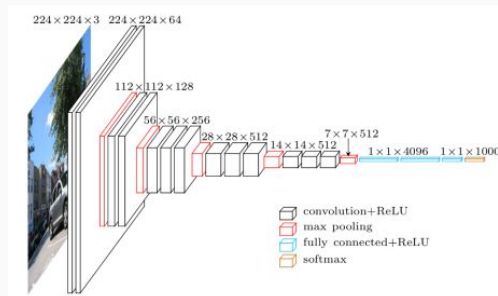
Residual Block

This is just showing the super-resolution network that the authors used, which is based on ResNet (He et al., 2015) architecture.

# Super-Resolution

## Loss Network ( $\phi$ )

The loss network used is the VGG16 (Simonyan & Zisserman) pretrained on ImageNet (it already understands of what to look for in an image)



VGG16 (Simonyan & Zisserman) Network architecture











Source:  
<https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/>

As mentioned, the loss network used in the paper is VGG16 pre-trained on ImageNet. The reason for this is because the pre-trained VGG16 already learned to extract interesting features from an image.








# Super-Resolution

## Results

				
				
<b>Ground Truth</b>	<b>Bicubic</b>	<b>Ours (<math>\ell_{pixel}</math>)</b>	<b>SRCNN [11]</b>	<b>Ours (<math>\ell_{feat}</math>)</b>
This image	31.78 / 0.8577	31.47 / 0.8573	32.99 / 0.8784	29.24 / 0.7841
Set5 mean	28.43 / 0.8114	28.40 / 0.8205	30.48 / 0.8628	27.09 / 0.7680

				
<b>Ground Truth</b>	<b>Bicubic</b>	<b>Ours (<math>\ell_{pixel}</math>)</b>	<b>SRCNN [11]</b>	<b>Ours (<math>\ell_{feat}</math>)</b>
This Image	21.69 / 0.5840	21.66 / 0.5881	22.53 / 0.6524	21.04 / 0.6116
Set14 mean	25.99 / 0.7301	25.75 / 0.6994	27.49 / 0.7503	24.99 / 0.6731
BSD100 mean	25.96 / 0.682	25.91 / 0.6680	26.90 / 0.7101	24.95 / 0.6317

Results for x4 super-resolution showing PSNR/SSIM for each example (Johnson et al., 2016)

This is their results, you can see how their produced images are arguably better perceptually, although the PSNR/SSIM may not reflect that. Remember that perceptual similarity metric such as PSNR and SSIM are not perfect and does not reflect perceptual similarity as the brain understands. It is also important to note that, if we zoomed in to their produced images, we may see a checkboard artifacts. This is due to the usage of deconvolutional layer in the super-resolution network. This is also one of the reason of why the PSNR/SSIM may not reflect the good perceptual appearance.

# Real Time Style Transfer (Background)

## Introduction



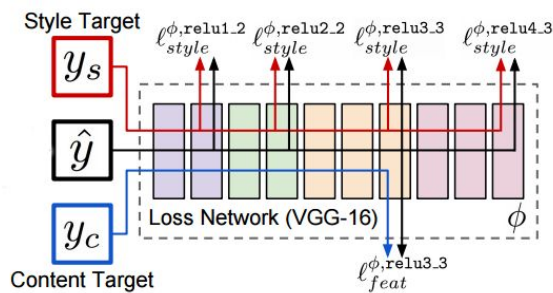
Illustration of neural style transfer (Gatys et al., 2015)

Source: <https://deepart.io/>

Recall on neural style transfer. Style transfer is a technique to “transfer” the style from an image to another. For example, in the image above, we can see the style from the famous painting “The Starry Night” by Vincent van Gogh (middle) was transferred to a photograph (left), which gives us the image on the right.

# Real Time Style Transfer (Background)

A Neural Algorithm of Artistic Style - Gatys et al. (2015)



Johnson et al. (2016, edited)

The loss network is the VGG16 pretrained on ImageNet

$\hat{y}$  = Input image (can be initialized to random image)

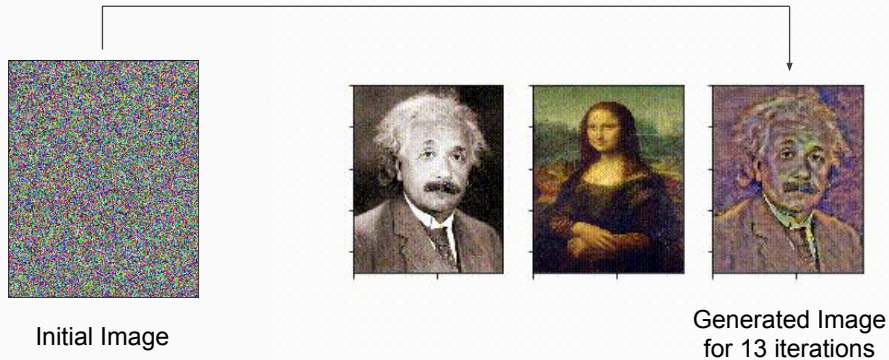
- Interested in the **derivative of the loss functions (style loss + content loss) with respect to the input image**, not weights of the network (so the **input image is getting updated** for each iteration)
- Slow, we need to perform **backprop for each iteration** until we are satisfied

The disadvantage of the original technique proposed by Gatys et al. (2015) is that it is too slow, since for a given content image and style image, the algorithm uses L-BFGS to optimize for the content and style losses. As a result, one needs to solve for the optimization problem everytime she needs to perform style transfer.

The question is now, what if we can create a neural network that is trained to generate an image that has a single specific style (i.e. "The Starry Night" style)? This means, we can just feed a content image to the neural network, and it will give us back the stylized image. This is fast because we just need to perform a single forward-pass!

# Real Time Style Transfer (Background)

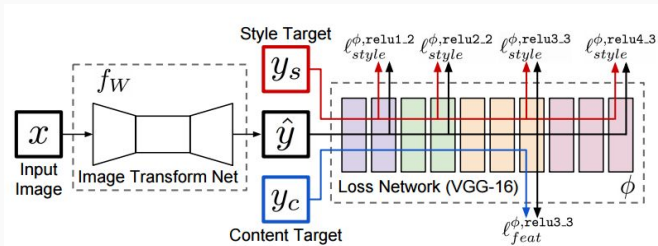
A Neural Algorithm of Artistic Style - Gatys et al. (2015)



This is an example of performing the original neural style transfer algorithm from Gatys et al. (2015). We can see how the image is being updated at every iteration by solving for an optimization problem using L-BFGS.

# Real Time Style Transfer

## Proposed Idea



Johnson et al. (2016)  $x$  &  $y_c$  = Input/content image  
 $\hat{y}$  = Generated/combined image

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

$$\ell_{style}^{\phi,j}(\hat{y}, y) = \|G_j^{\phi}(\hat{y}) - G_j^{\phi}(y)\|_F^2. \quad y = y_s \text{ here}$$

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

$$\text{Or simply, } G_j^{\phi}(x) = \psi \psi^T / C_j H_j W_j$$

Where  $\psi$  is just  $\phi_j(x)$  reshaped into  $C_j \times H_j W_j$  (2D matrix)

This is their proposed method. We can see that for the content loss, Johnson et al. (2016) proposed to use the perceptual loss by passing the generated image  $y_{\hat{}}$ ,  $y_s$ , and  $y_c$  to the VGG16 as we saw earlier in the super resolution application. Note that the loss network is frozen (i.e. the parameters are fixed).

We can also see that the whole architecture are differentiable and can be trained end-to-end via backpropagation. It is important to note that the style target (denoted  $y_s$  in the figure) is fixed. Which means, one image transform network ( $f_w$ ) can only learn one specific style. Note that the content target ( $y_c$ ) is usually the same as the input image ( $x$ ).

# Real Time Style Transfer

Image Transform Net Architecture For Style Transfer

Layer	Activation size
Input	$3 \times 256 \times 256$
$32 \times 9 \times 9$ conv, stride 1	$32 \times 256 \times 256$
$64 \times 3 \times 3$ conv, stride 2	$64 \times 128 \times 128$
$128 \times 3 \times 3$ conv, stride 2	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
$64 \times 3 \times 3$ conv, stride 1/2	$64 \times 128 \times 128$
$32 \times 3 \times 3$ conv, stride 1/2	$32 \times 256 \times 256$
$3 \times 9 \times 9$ conv, stride 1	$3 \times 256 \times 256$

The network used (Johnson et al.,2016)

This is just showing the architecture that Johnson et al. (2016) used in their paper.

# Real Time Style Transfer

## Advantages

- By adding image transform net,  $f_w$ , we can calculate the **derivative of the losses with respect to  $f_w$ 's weights**
- We can train our image transform net by feeding in more content images while keeping the style image constant (one network is only trained for one particular type of style)
- Once the network ( $f_w$ ) is trained, **we can just forward propagate through  $f_w$  once to generate the combined image**
- So it is fast!

# Real Time Style Transfer

## Results - Speed

Note: speed test was done on a GTX Titan X GPU

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

Speed comparison (Johnson et al., 2016)

This is showing that this method is definitely faster during test time since once the network is trained, one only needs to perform single forward-pass through the network rather than solving the optimization problem.



# Real Time Style Transfer

Results - Generated Image

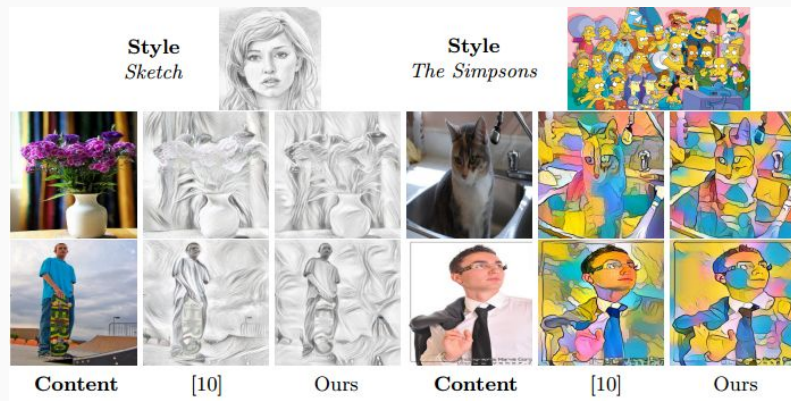


Johnson et al. (2016)

This is their results compared to the one from Gatys et al. (2015), one may argue that the results are comparable.

# Real Time Style Transfer

Results - Generated Image



Johnson et al. (2016)

More results.

# References

Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. CoRR, abs/1501.00092, 2015. URL <http://arxiv.org/abs/1501.00092>.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. CoRR, abs/1508.06576, 2015. URL <http://arxiv.org/abs/1508.06576>.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. CoRR, abs/1512.03385, 2015. URL <http://arxiv.org/abs/1512.03385>.

Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. CoRR, abs/1603.08155, 2016. URL <http://arxiv.org/abs/1603.08155>.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014. URL <http://arxiv.org/abs/1409.1556>.

These are the references. I encourage you to read the papers :)

# Thank You

Email:

[reywiatno@gmail.com](mailto:reywiatno@gmail.com)

Super-Resolution implementation in Keras:

[https://github.com/rrwiatn/super\\_resolution](https://github.com/rrwiatn/super_resolution)

Feel free to contact me if you have any questions, and please feel free to grab my super-resolution implementation and play with it!