Papers :

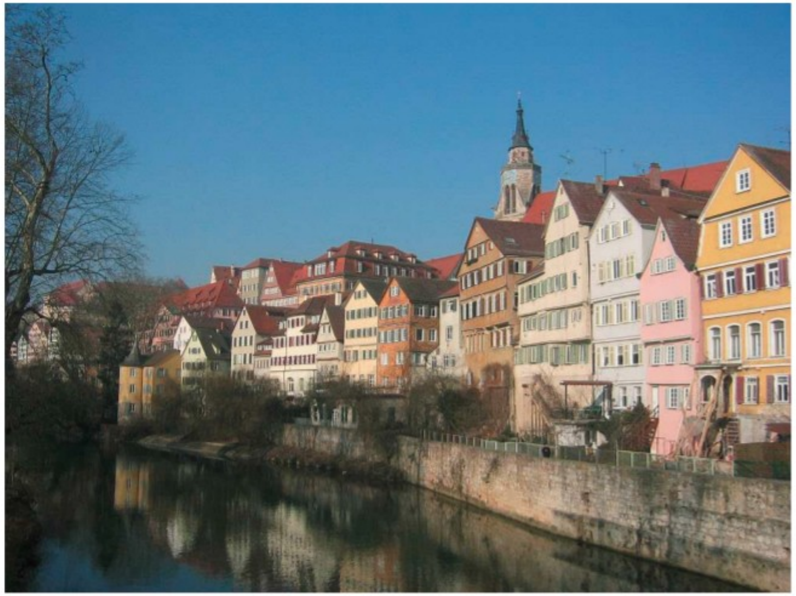
- Image Style Transfer Using Convolutional Neural Networks in CVPR 2016
  Authors: Leon A. Gatys,Alexander S. Ecker,Matthias Bethge

- Texture Synthesis Using Convolutional Neural Networks in NIPS 2015
  Authors: Leon A. Gatys,Alexander S. Ecker,Matthias Bethge

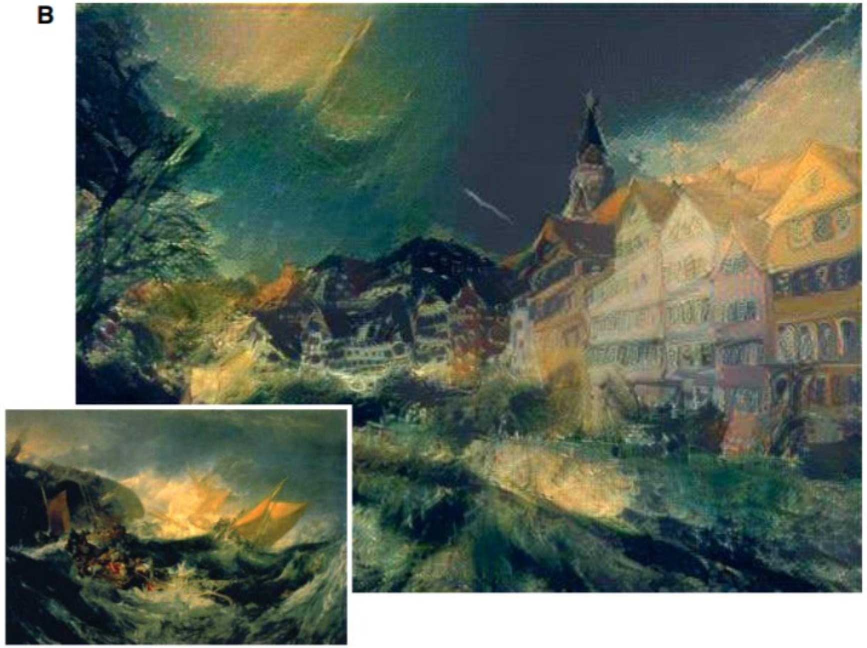
- Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization in ICCV 2017
  Authors: Xun Huang,Serge Belongie

# Image Style Transfer using Convolutional Neural Networks in CVPR 2016

A



B



## Introduction

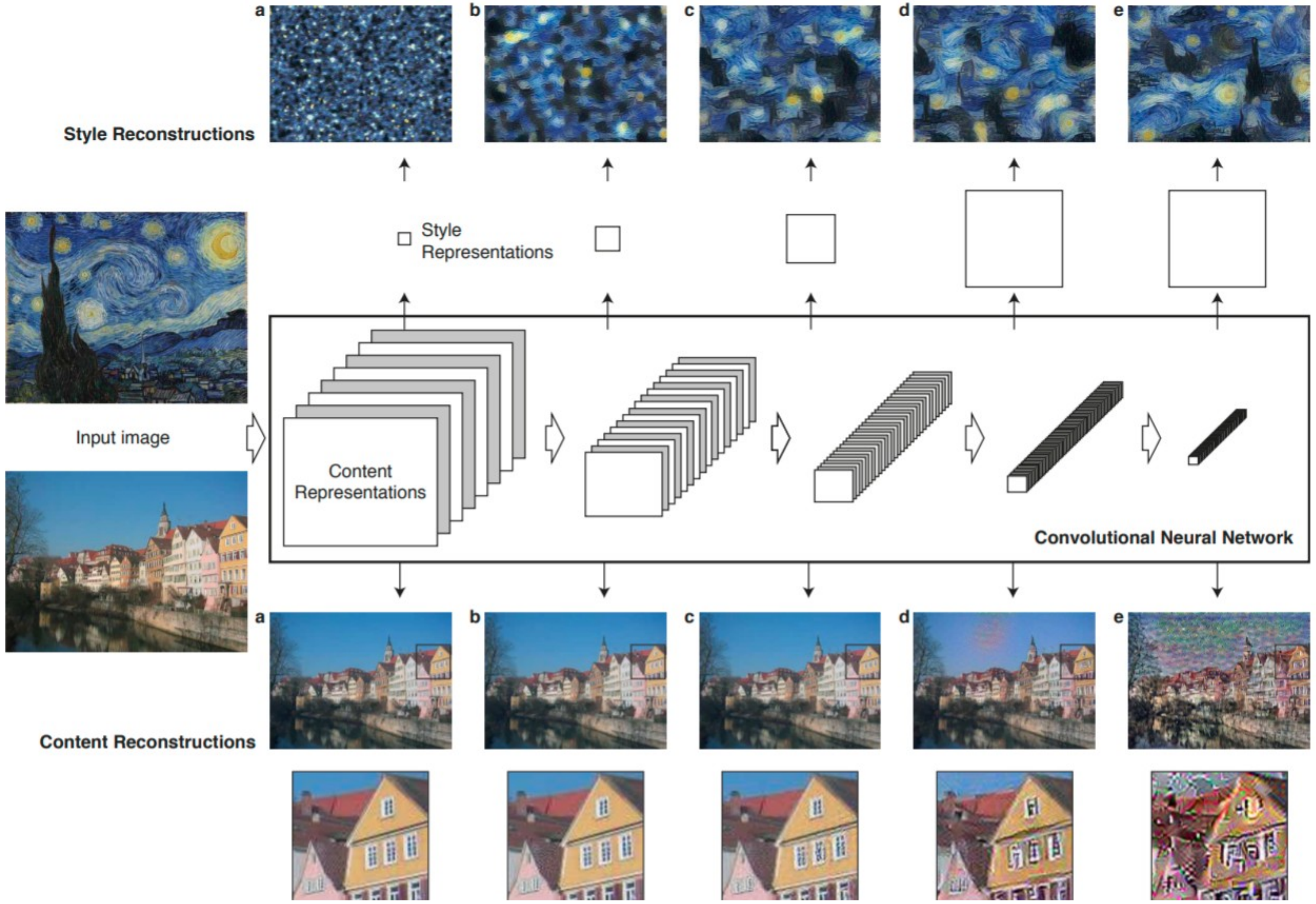
- How we can render semantic of images?
- We need a way to separate content and style.
- Previous approaches lack image representations.

## Solution given in this paper

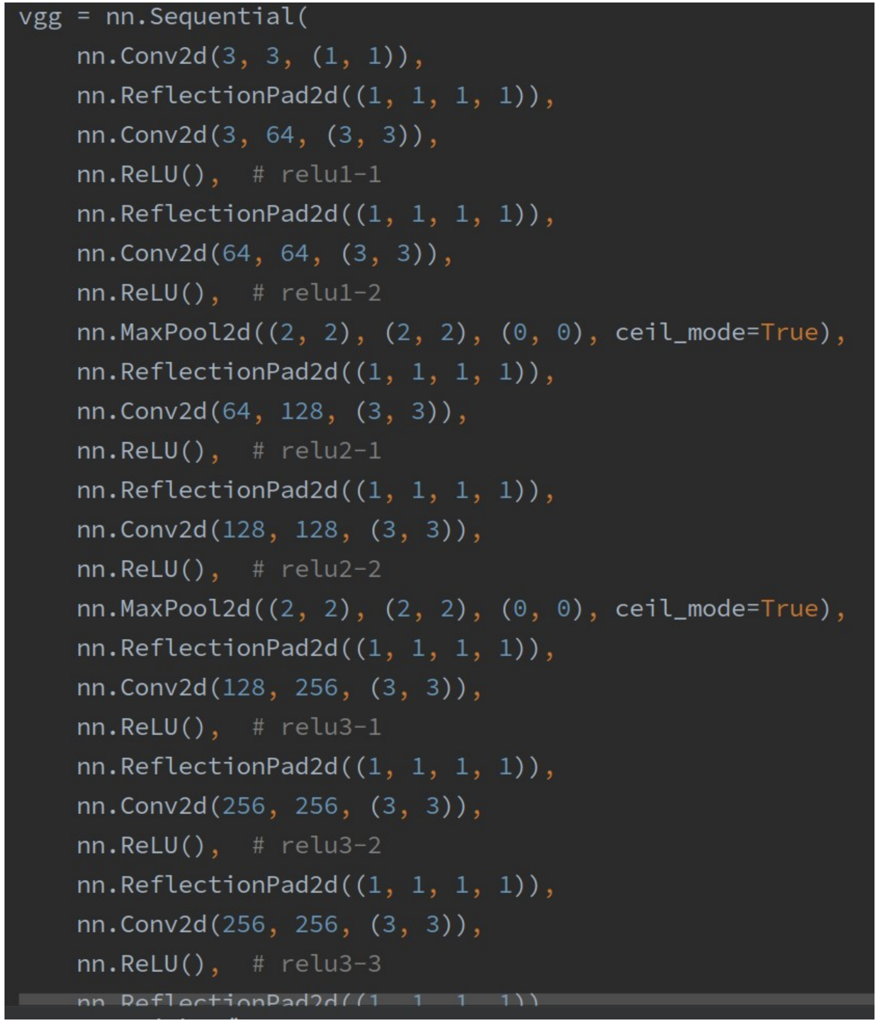
We can use image representations derived from Convolutional Neural Networks optimised for object recognition.

## Dataset

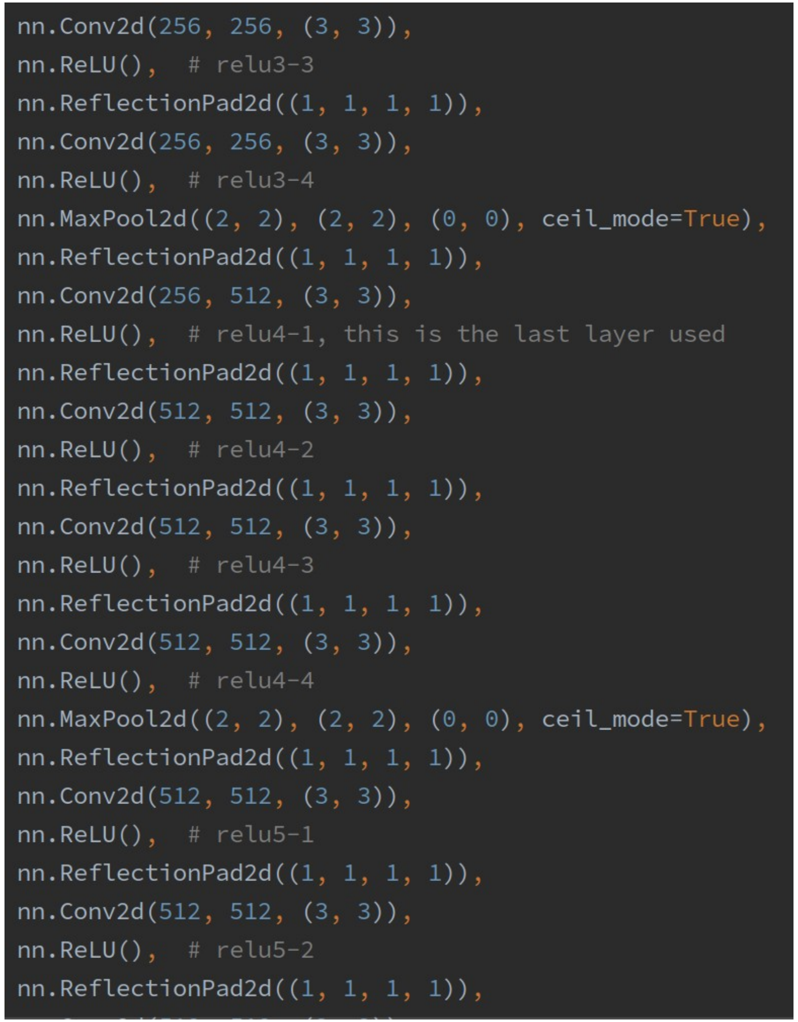
In,this paper there is no specific dataset mentioned as the authors have  proposed an algorithm which runs on test time and does not require pre-training.



Some observations



```
vgg = nn.Sequential(
    nn.Conv2d(3, 3, (1, 1)),
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(3, 64, (3, 3)),
    nn.ReLU(),  # relu1-1
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(64, 64, (3, 3)),
    nn.ReLU(),  # relu1-2
    nn.MaxPool2d((2, 2), (2, 2), (0, 0), ceil_mode=True),
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(64, 128, (3, 3)),
    nn.ReLU(),  # relu2-1
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(128, 128, (3, 3)),
    nn.ReLU(),  # relu2-2
    nn.MaxPool2d((2, 2), (2, 2), (0, 0), ceil_mode=True),
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(128, 256, (3, 3)),
    nn.ReLU(),  # relu3-1
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(256, 256, (3, 3)),
    nn.ReLU(),  # relu3-2
    nn.ReflectionPad2d((1, 1, 1, 1)),
    nn.Conv2d(256, 256, (3, 3)),
    nn.ReLU(),  # relu3-3
    nn.ReflectionPad2d((1, 1, 1, 1))
```



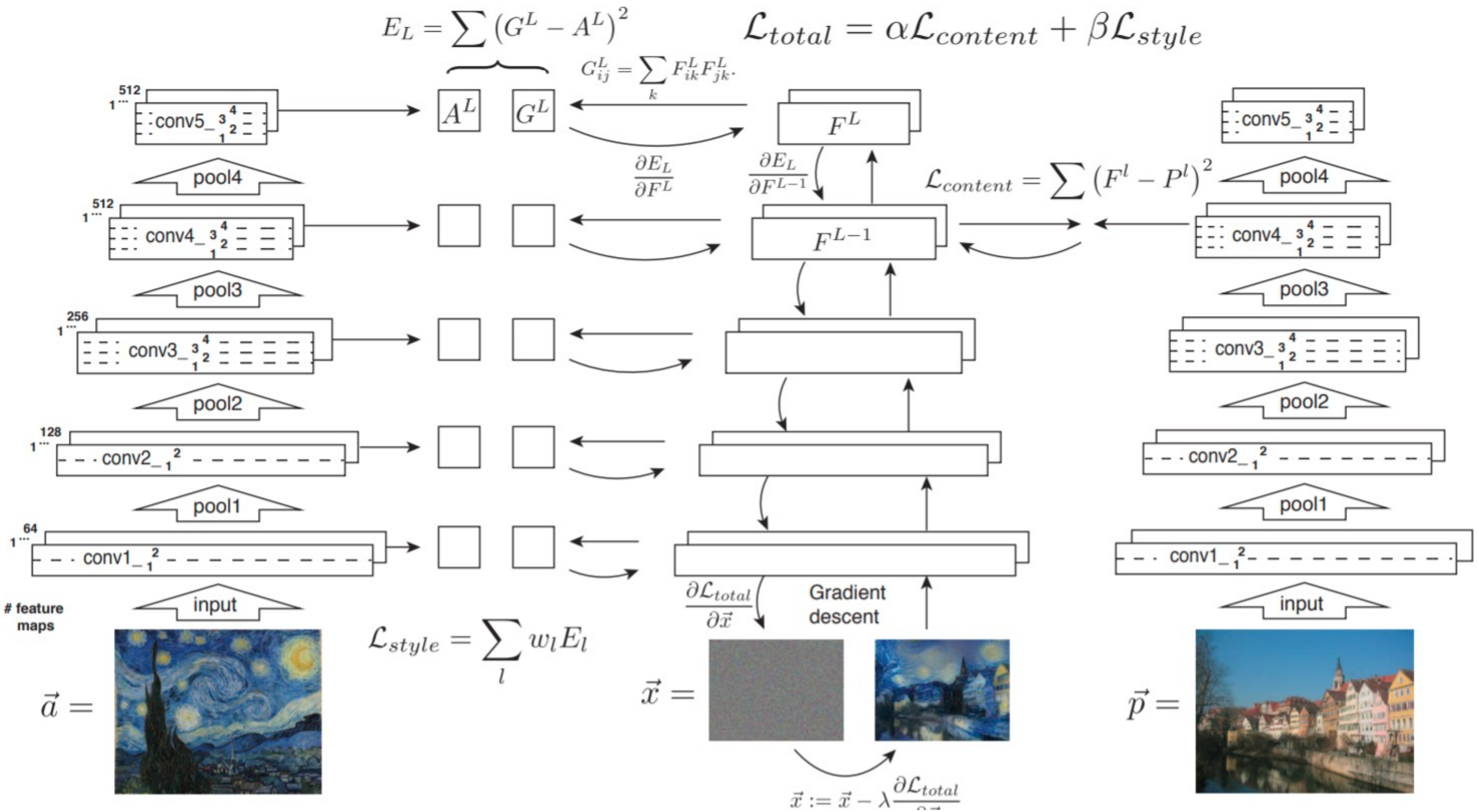
```
nn.Conv2d(256, 256, (3, 3)),
nn.ReLU(),  # relu3-3
nn.ReflectionPad2d((1, 1, 1, 1)),
nn.Conv2d(256, 256, (3, 3)),
nn.ReLU(),  # relu3-4
nn.MaxPool2d((2, 2), (2, 2), (0, 0), ceil_mode=True),
nn.ReflectionPad2d((1, 1, 1, 1)),
nn.Conv2d(256, 512, (3, 3)),
nn.ReLU(),  # relu4-1, this is the last layer used
nn.ReflectionPad2d((1, 1, 1, 1)),
nn.Conv2d(512, 512, (3, 3)),
nn.ReLU(),  # relu4-2
nn.ReflectionPad2d((1, 1, 1, 1)),
nn.Conv2d(512, 512, (3, 3)),
nn.ReLU(),  # relu4-3
nn.ReflectionPad2d((1, 1, 1, 1)),
nn.Conv2d(512, 512, (3, 3)),
nn.ReLU(),  # relu4-4
nn.MaxPool2d((2, 2), (2, 2), (0, 0), ceil_mode=True),
nn.ReflectionPad2d((1, 1, 1, 1)),
nn.Conv2d(512, 512, (3, 3)),
nn.ReLU(),  # relu5-1
nn.ReflectionPad2d((1, 1, 1, 1)),
nn.Conv2d(512, 512, (3, 3)),
nn.ReLU(),  # relu5-2
nn.ReflectionPad2d((1, 1, 1, 1)),
```

## Used modified VGG19 Network

- They normalized the network by scaling the weights such that the mean activation of each convolutional filter over images and positions is equal to one.

Let $\mu_i^l \equiv \mathbb{E}_{X,j} F_{ij}^l = \frac{1}{NM^l} \sum_X \sum_{j=1}^{M^l} F_{ij}^l = \frac{1}{NM^l} \sum_X \sum_{j=1}^{M^l} max(0,\ W_i^l \bullet P_j^{l-1} + b_i^l)$ be the mean activation of the $i$-th filter in layer $l$ over all $N$ images in the dataset $X$ and all $M^l$ positions in the filter's activation map. Note that this is a

- Replacing max-pooling with average pooling improves gradient flow.Images generated are cleaner.


$E_L = \sum (G^L - A^L)^2$
$\mathcal{L}_{total} = \alpha\mathcal{L}_{content} + \beta\mathcal{L}_{style}$
$G^L_{ij} = \sum_k F^L_{ik} F^L_{jk}.$
$A^L$
$G^L$
$F^L$
$\frac{\partial E_L}{\partial F^L}$
$\frac{\partial E_L}{\partial F^{L-1}}$
$F^{L-1}$
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
512
conv5_ 1 2 3 4
pool4
512
conv4_ 1 2 3 4
pool3
256
conv3_ 1 2 3 4
pool2
128
conv2_ 1 2
pool1
64
conv1_ 1 2
input
# feature maps
$\vec{a} =$
$\mathcal{L}_{style} = \sum_l w_l E_l$
$\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
Gradient descent
$\vec{x} =$
$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{}$
$\vec{p} =$

## Losses used in training

1. Content Loss

$$\mathcal{L}_{\text{content}}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^{l} - P_{ij}^{l}\right)^2$$

F,P are the feature maps of images p,x

2. Style Loss of a layer

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^{l} - A_{ij}^{l}\right)^2$$

$$G_{ij}^{l} = \sum_k F_{ik}^{l} F_{jk}^{l}$$

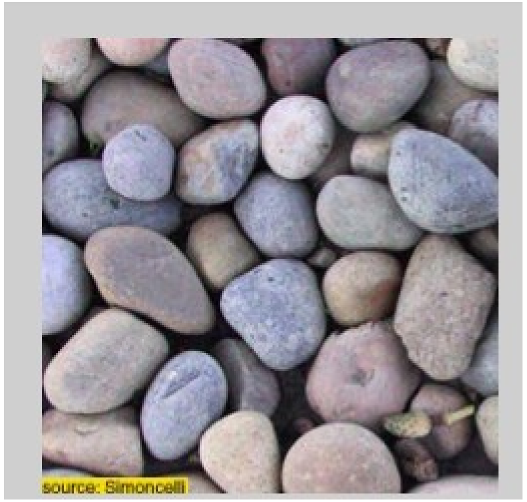
Contribution of layer **l** in style loss

and the total style loss is

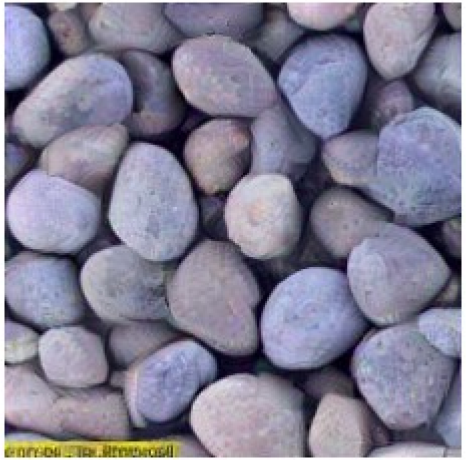
$$\mathcal{L}_{\text{style}}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l,$$

## Texture Synthesis Using Convolutional Neural Networks in NIPS 2015

In,this paper authors are trying to generate a texture from a given example texture.



Original



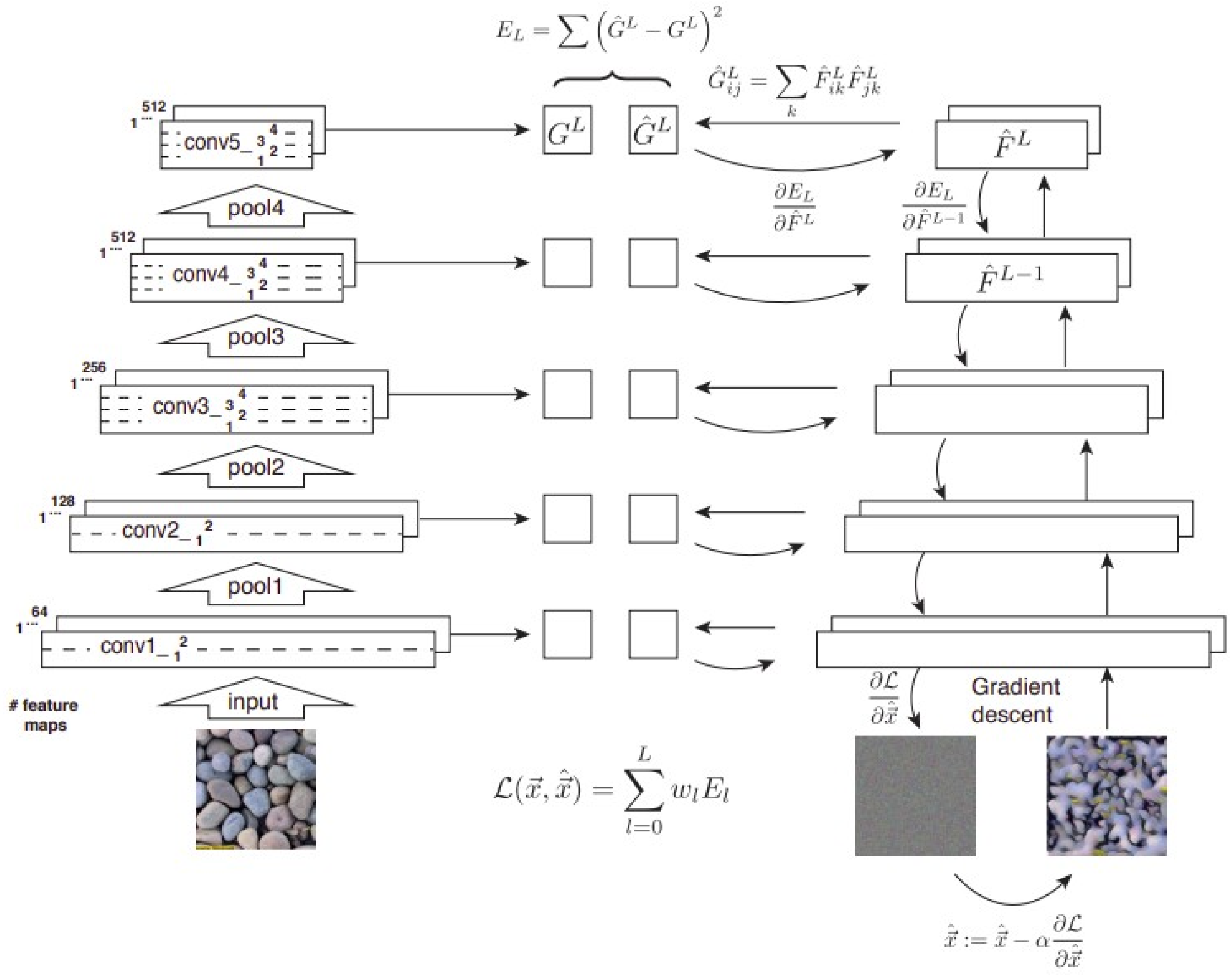
Generated

## Main Approaches for Texture Generation

- Generate new texture by resampling of pixels or whole patches.This is non-parametric approach.Do not define an actual model for generating textures.Use mechanistic procedure without changing its perceptual properties.
- Define a parametric  texture model.This paper used this approach.

## Advantage of using parametric approach of CNN

Since,CNN is spatially invariant due to training on object recognition we get a texture model which is also spatially invariant.



$E_L = \sum \left(\hat{G}^L - G^L\right)^2$
$\hat{G}^L_{ij} = \sum_k \hat{F}^L_{ik}\hat{F}^L_{jk}$
$G^L$
$\hat{G}^L$
$\hat{F}^L$
$\hat{F}^{L-1}$
$\frac{\partial E_L}{\partial \hat{F}^L}$
$\frac{\partial E_L}{\partial \hat{F}^{L-1}}$
conv5_1 2 3 4
conv4_1 2 3 4
conv3_1 2 3 4
conv2_1 2
conv1_1 2
512
512
256
128
64
pool4
pool3
pool2
pool1
input
# feature maps
$\mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l$
$\frac{\partial \mathcal{L}}{\partial \hat{\vec{x}}}$
Gradient descent
$\hat{\vec{x}} := \hat{\vec{x}} - \alpha \frac{\partial \mathcal{L}}{\partial \hat{\vec{x}}}$

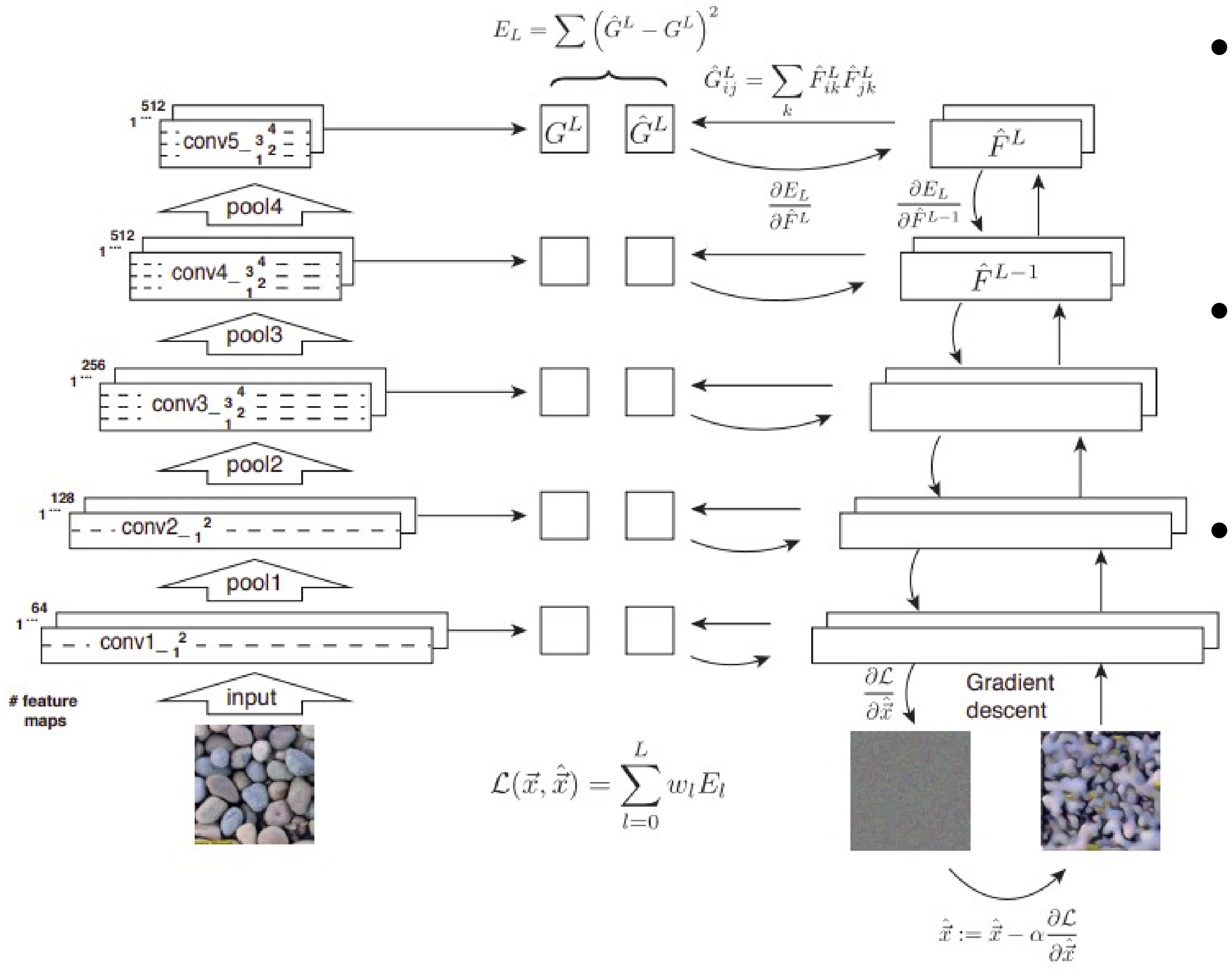
Each Layer of CNN has $N_l$ feature maps of size $M_l$ when vectorized.So,

$$F^l \in \mathcal{R}^{N_l \times M_l}$$

$F^l_{jk}$ is the activation of the jth filter at position k in layer l.

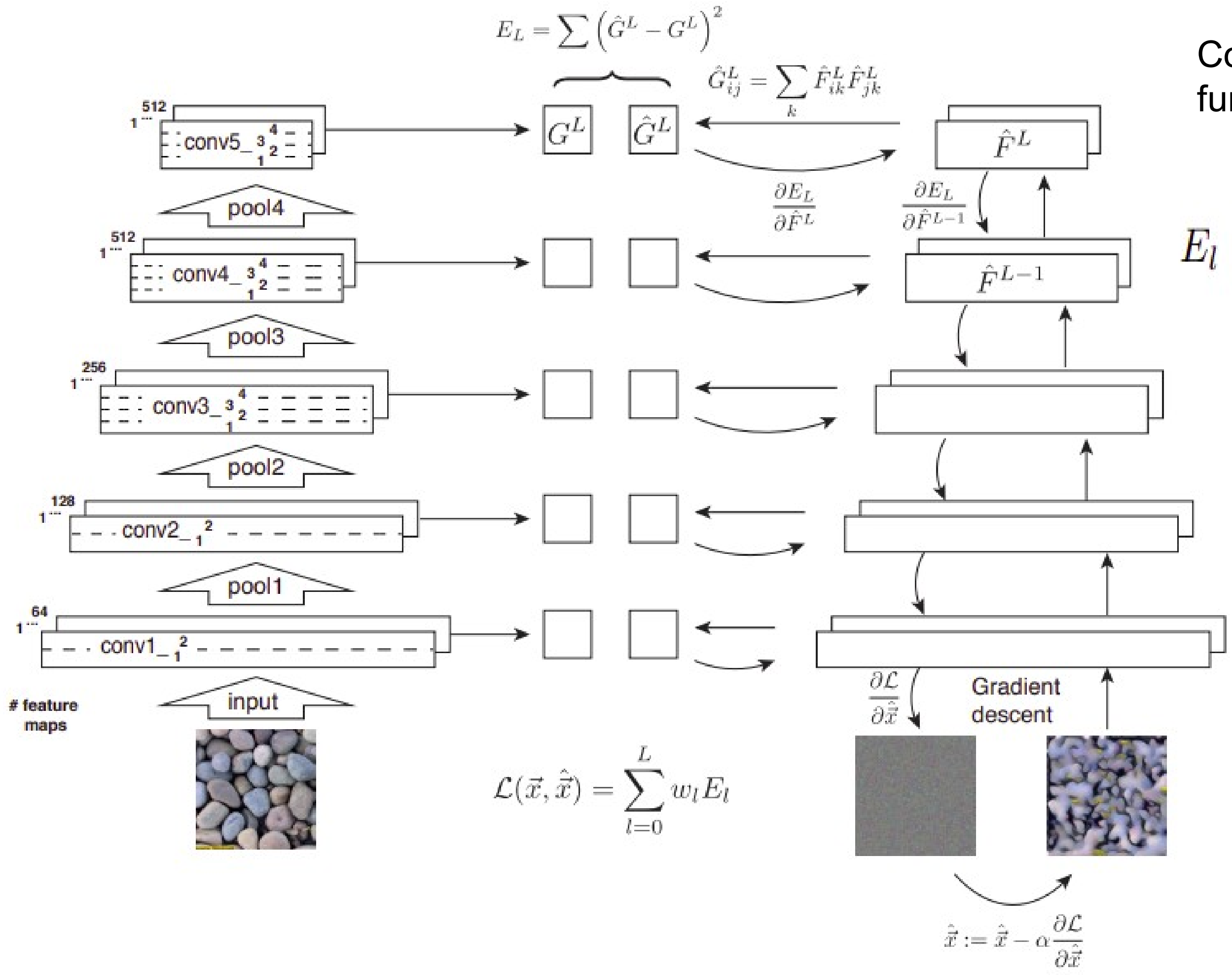
In,this method texture is learnt using feature correlations.It is given by Gram matrix.

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



$E_L = \sum \left(\hat{G}^L - G^L\right)^2$
$\hat{G}^L_{ij} = \sum_k \hat{F}^L_{ik}\hat{F}^L_{jk}$
$G^L$
$\hat{G}^L$
$\hat{F}^L$
$\frac{\partial E_L}{\partial \hat{F}^L}$
$\frac{\partial E_L}{\partial \hat{F}^{L-1}}$
$\hat{F}^{L-1}$
512 conv5_ 1 2 3 4
pool4
512 conv4_ 1 2 3 4
pool3
256 conv3_ 1 2 3 4
pool2
128 conv2_ 1 2
pool1
64 conv1_ 1 2
input
# feature maps
$\frac{\partial \mathcal{L}}{\partial \hat{\vec{x}}}$
Gradient descent
$\mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l$
$\hat{\vec{x}} := \hat{\vec{x}} - \alpha \frac{\partial \mathcal{L}}{\partial \hat{\vec{x}}}$

- Here,Optimization done for Gram matrix by minimizing the mean-squared distance between its entries.
- They consider all the intermediate layers Gram matrix representation for this.
- L-BFGS is used to learn the correct representation for the white noise image.


$E_L = \sum \left(\hat{G}^L - G^L\right)^2$
$\hat{G}^L_{ij} = \sum_k \hat{F}^L_{ik}\hat{F}^L_{jk}$
512
1 ...
conv5_ 1 2 3 4
pool4
512
conv4_ 1 2 3 4
pool3
256
conv3_ 1 2 3 4
pool2
128
conv2_ 1 2
pool1
64
conv1_ 1 2
input
# feature maps
$G^L$
$\hat{G}^L$
$\hat{F}^L$
$\hat{F}^{L-1}$
$\frac{\partial E_L}{\partial \hat{F}^L}$
$\frac{\partial E_L}{\partial \hat{F}^{L-1}}$
$\frac{\partial \mathcal{L}}{\partial \hat{\vec{x}}}$
Gradient descent
$\mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l$
$\hat{\vec{x}} := \hat{\vec{x}} - \alpha \frac{\partial \mathcal{L}}{\partial \hat{\vec{x}}}$

Contribution of a layer in loss function:

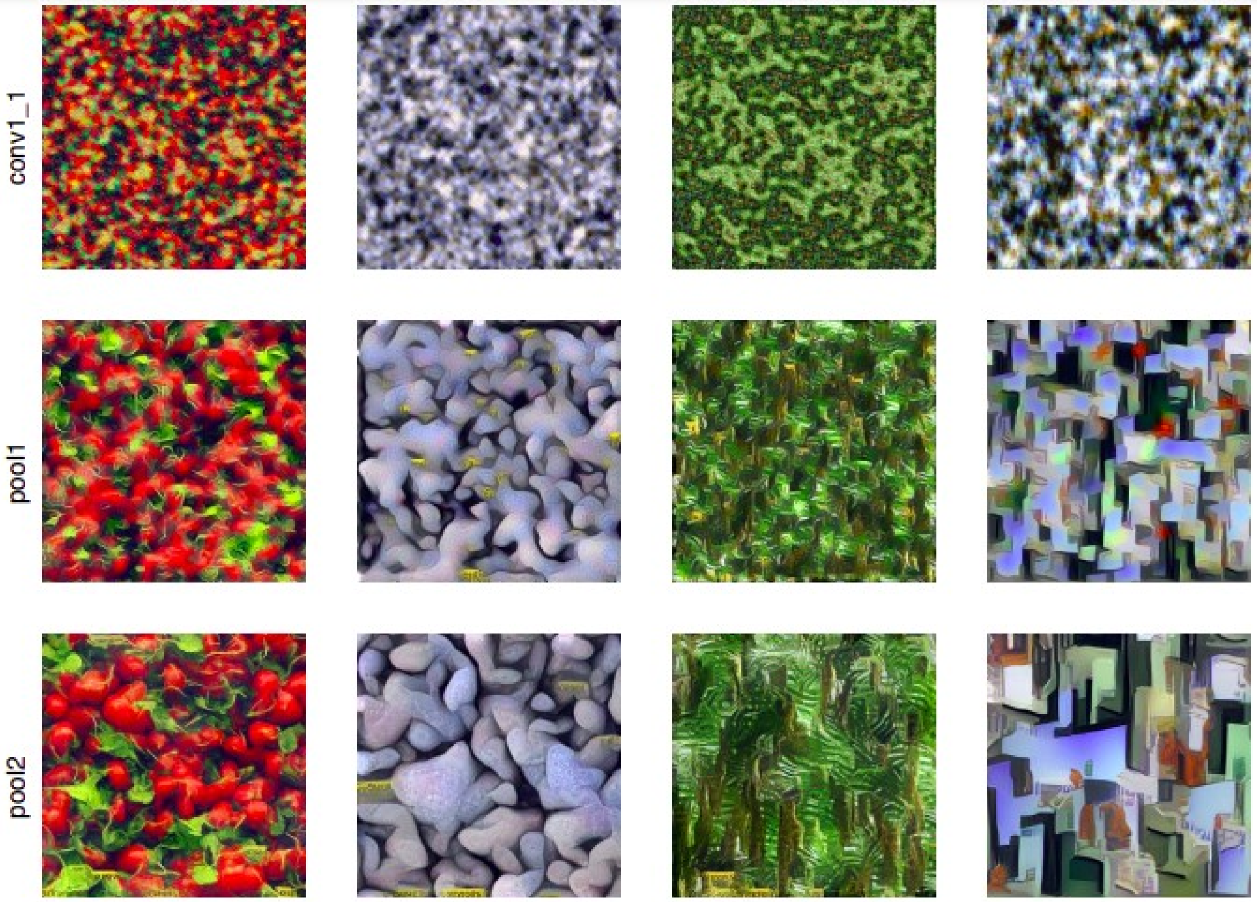
$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G^l_{ij} - \hat{G}^l_{ij}\right)^2$$

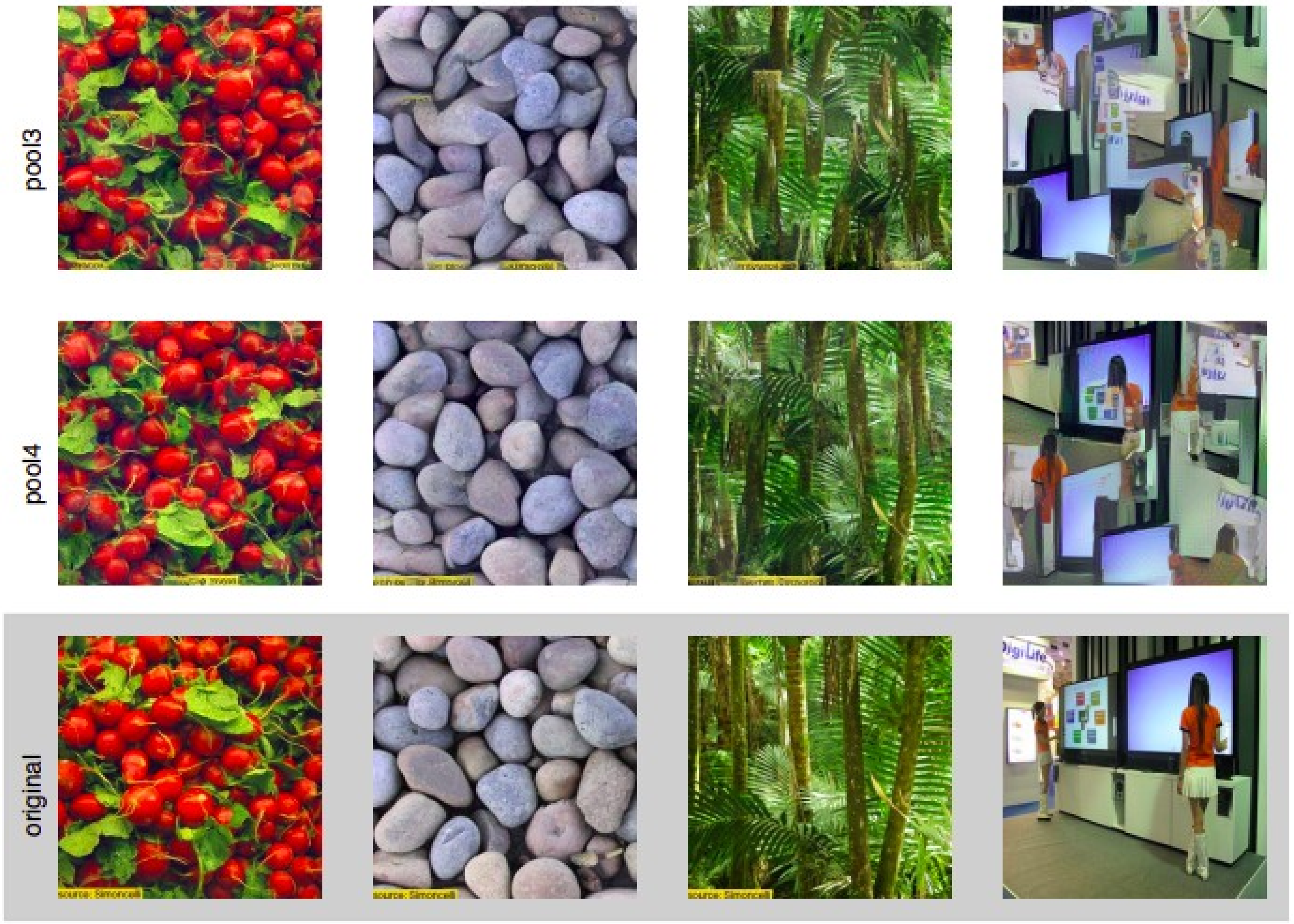
Total loss is:

$$\mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l$$

$w_l$ is hyperparam

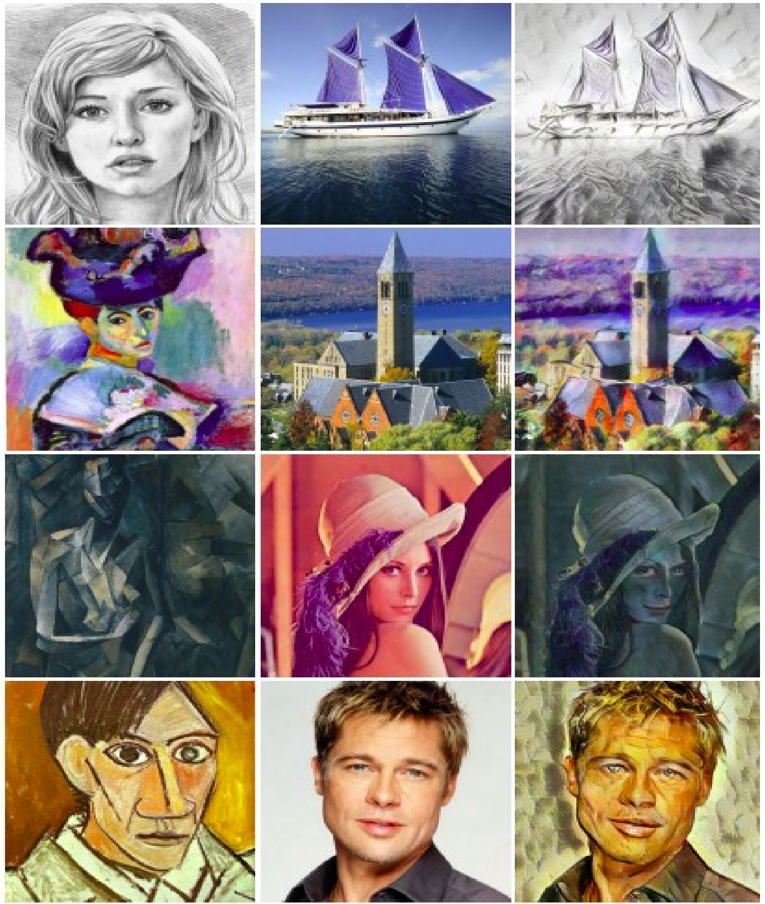
Results




pool3
pool4
original
source: Simoncelli

Portilla & Simoncelli

## Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization in ICCV 2017



## Introduction

- Present a simple effective approach for the first time that enables arbitrary style transfer in real-time.

- Propose a novel Adaptive Instance Normalization(AdaIN) layer that aligns the mean and variance of content features with style features.

- Before this,most feed-forward methods each network is restricted to a single style.

## Instance Normalization(IN)

$$\mathbf{IN}(x) = \gamma\left(\frac{x-\mu(x)}{\sigma(x)}\right)+\beta$$

$\gamma,\beta \in \mathbb{R}^C$ are affine parameters learned from data.

$$\mu_{nc}(x) = \frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nchw}$$

Mean and variance are calculated for each channel and each sample.

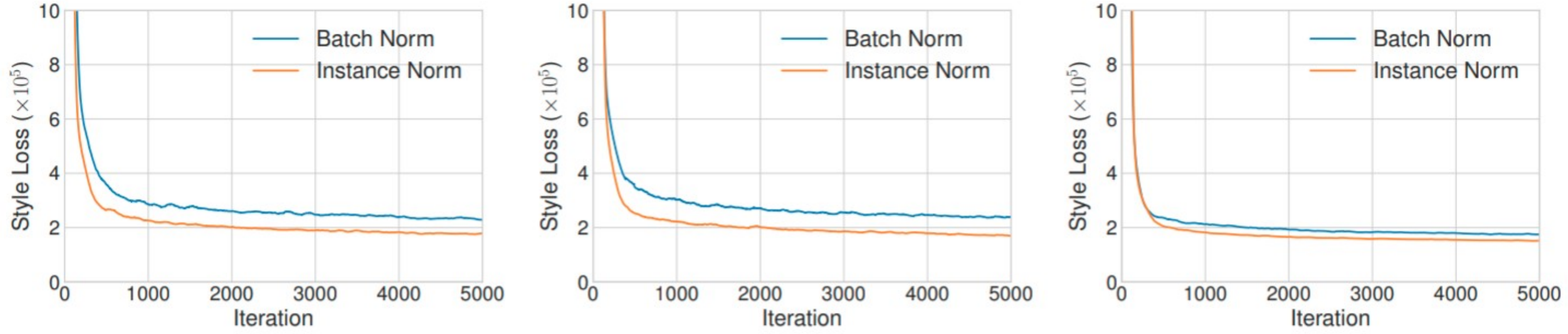
$$\sigma_{nc}(x) = \sqrt{\frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{nchw}-\mu_{nc}(x))^2+\epsilon}$$

Observation 1: Affine parameters γ and β in IN can completely change the style of the output image.

Observation 2: Matching many other statistics,including channel-wise mean and variance are also effective for style transfer

## Argument by the author

Instance normalization performs a form of style normalization by normalizing feature statistics, namely the mean and variance.



(a) Trained with original images. (b) Trained with contrast normalized images. (c) Trained with style normalized images.

Figure 1. To understand the reason for IN's effectiveness in style transfer, we train an IN model and a BN model with (a) original images in MS-COCO [36], (b) contrast normalized images, and (c) style normalized images using a pre-trained style transfer network [24]. The improvement brought by IN remains significant even when all training images are normalized to the same contrast, but are much smaller when all images are (approximately) normalized to the same style. Our results suggest that IN performs a kind of style normalization.

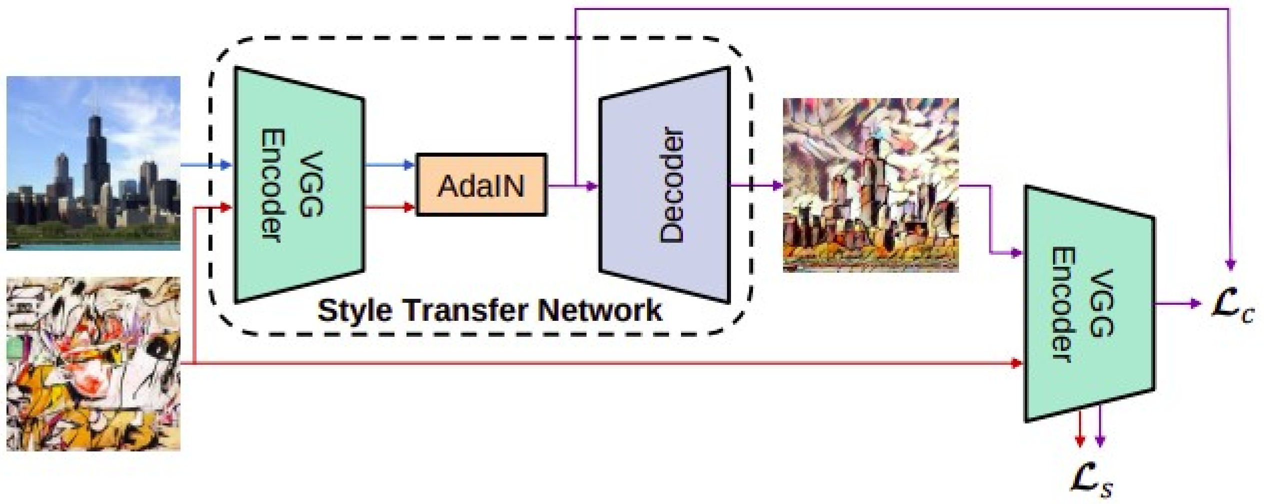
## Adaptive Instance Normalization

Question Asked: If IN normalizes the input to a single style specified by the affine parameters,is it possible to adapt it to arbitrary given styles by using adaptive affine transformation?

Answer is the AdaIN layer.It takes a content input x and style input y and simply aligns the channel-wise mean and variance of x to match those of y.

$$\mathrm{AdaIN}(x, y) = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y)$$

x is content image,y is style image.Mean and variance are calculated same as the IN.



f is encoder,g is decoder

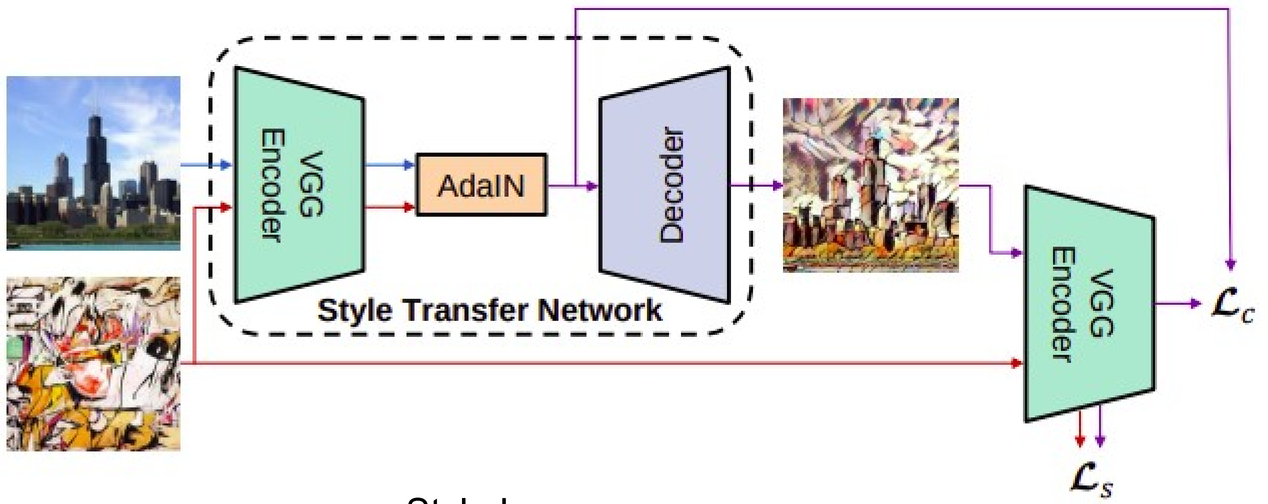
$$t = \text{AdaIN}(f(c), f(s))$$

Content Loss

$$\mathcal{L}_c = \|f(g(t)) - t\|_2$$

Overall Loss

$$\mathcal{L} = \mathcal{L}_c + \lambda\mathcal{L}_s$$



Style Loss

$$\mathcal{L}_s = \sum_{i=1}^{L} \|\mu(\phi_i(g(t))) - \mu(\phi_i(s))\|_2 \quad + \quad \sum_{i=1}^{L} \|\sigma(\phi_i(g(t))) - \sigma(\phi_i(s))\|_2$$

$\Phi_i$ is ith relu layer

## Control of degree of style transfer

$$T(c, s, \alpha) = g((1-\alpha)f(c) + \alpha \mathrm{AdaIN}(f(c), f(s)))$$

Provides content-style tradeoff at test-time by interpolating between feature maps that are fed to the decoder.Equivalently, interpolating between the affine parameters of AdaIN.

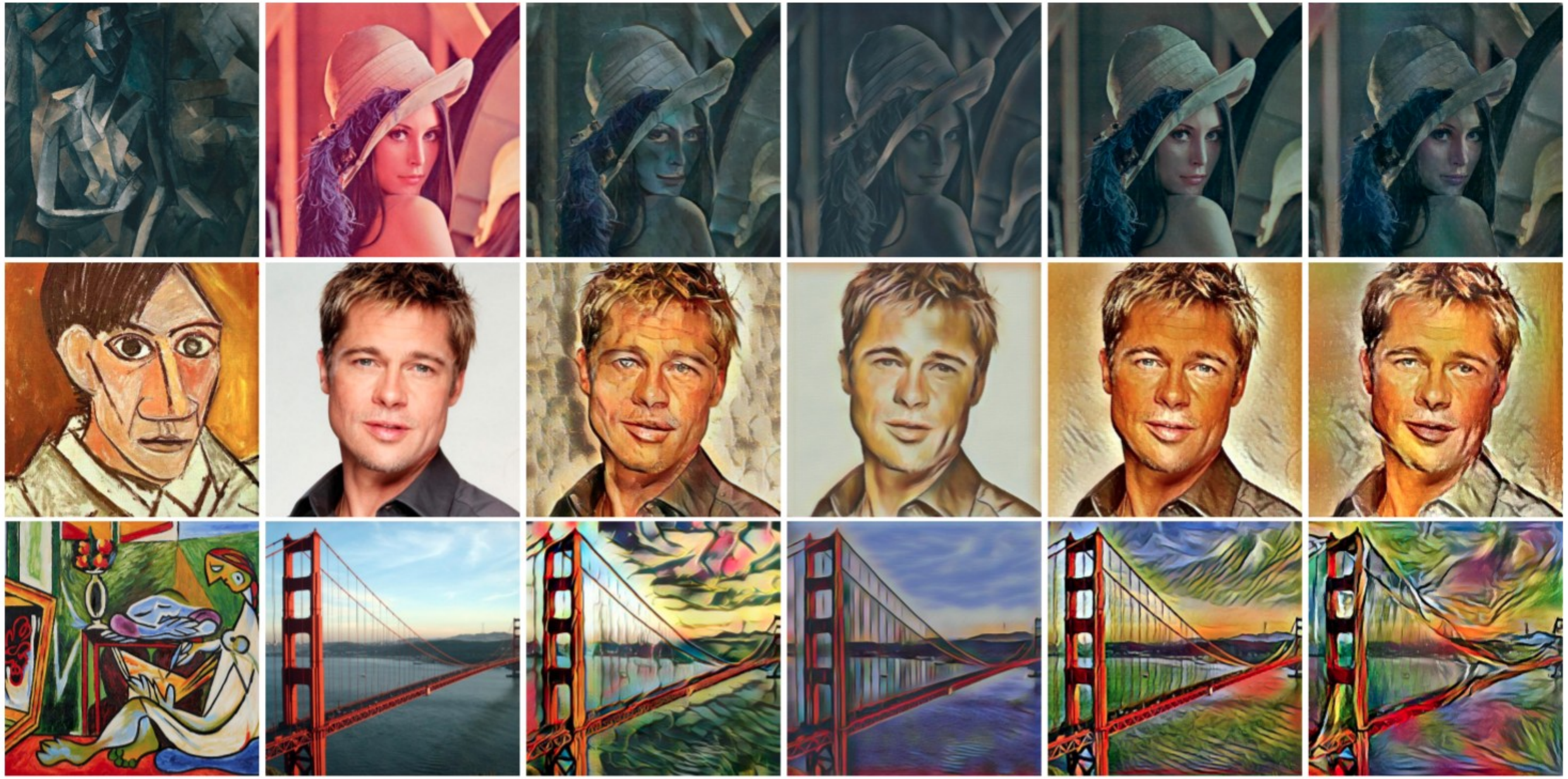
**Style interpolation.** To interpolate between a set of $K$ style images $s_1, s_2, ..., s_K$ with corresponding weights $w_1, w_2, ..., w_K$ such that $\sum_{k=1}^{K} w_k = 1$, we similarly interpolate between feature maps (results shown in Fig. 8):

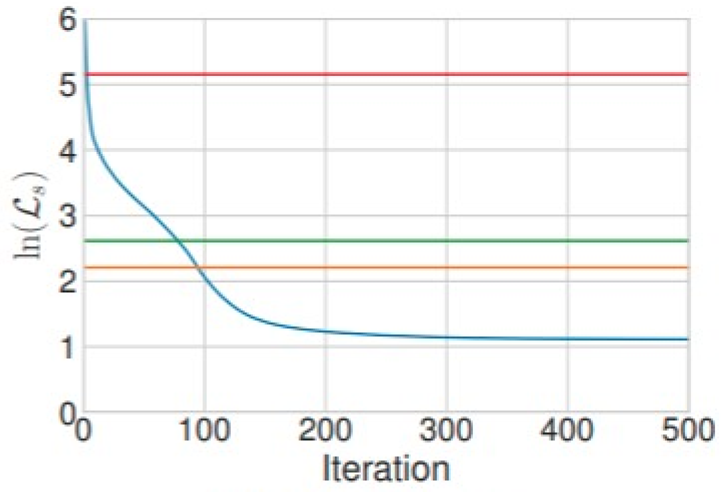
$$T(c, s_{1,2,...K}, w_{1,2,...K}) = g(\sum_{k=1}^{K} w_k \mathrm{AdaIN}(f(c), f(s_k)))$$

## Dataset Used

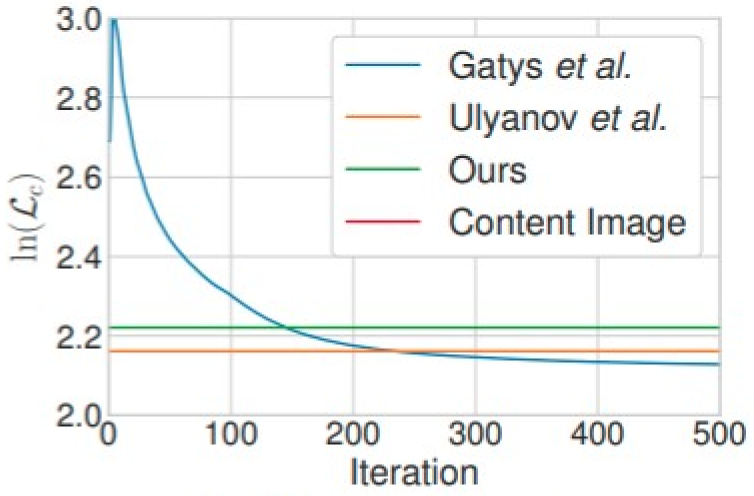
- MS-COCO dataset for content images.
- Wikiart dataset for style images.

Each contains around 80K training images which were used by the authors for there model training.


Style
Content
Ours
Chen and Schmidt
Ulyanov *et al*.
Gatys *et al*.



(a) Style Loss



(b) Content Loss

Figure 3. Quantitative comparison of different methods in terms of style and content loss. Numbers are averaged over 10 style images and 50 content images randomly chosen from our test set.