Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization

Presented by: Karen Lucknvalai and Alexandr Kuznetsov
Example

Style

Content

Result
Motivation

● Transforming content of an image to different styles
  ○ Photos to Paintings
  ○ Horses to Zebras
  ○ Day to Night

● Current Approaches
  ○ Optimization based
    ■ Arbitrary styles
    ■ Slow (takes minutes)
  ○ Feed-forward based
    ■ Limited number of styles
    ■ Fast
Previous works

- **Image Style Transfer Using CNNs** by Gatys et al.
  - Seminal work (2016)
  - Separates style and content
  - Uses VGG
  - Slow optimization process

- **A Learned Representation For Artistic Style** by Dumoulin et al.
  - Uses feed forward network
    - Fast
  - Supports 32 styles
  - Restricted to trained styles
Contributions

- Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization
  - Achieves speed comparable to existing approaches
  - Not limited to predefined set of styles
  - Single feed-forward network
  - Adaptive Instance Normalization
  - Flexible user controls
    - Content-style
    - Style interpolation
    - Color and spatial controls
Background: Batch Normalization (BN)

- Designed to accelerate training

\[ \text{BN}(x) = \gamma \left( \frac{x - \mu(x)}{\sigma(x)} \right) + \beta \]

- \( \gamma, \beta \in \mathbb{R}^C \) are learned parameters during the training
- \( \mu_c, \sigma_c \) are calculated using pixels of the whole mini-batch

\[
\mu_c(x) = \frac{1}{NHW} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw} \\
\sigma_c(x) = \sqrt{\frac{1}{NHW} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_c(x))^2 + \epsilon}
\]

- Replaces them with statistics during the test time
  - Discrepancy between train and test
Background: Instance Normalization (IN)

- Similar to batch normalization
  \[ \text{IN}(x) = \gamma \left( \frac{x - \mu(x)}{\sigma(x)} \right) + \beta \]

- But \( \mu_{nc}, \sigma_{nc} \) are computed only across spatial domains
  - For each channel and each sample
    - Images are independent of the batch
    - While for BN, it’s computed for the whole batch
  - Uses the same image statistics for training and for testing
    \[
    \mu_{nc}(x) = \frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw} \\
    \sigma_{nc}(x) = \sqrt{\frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_{nc}(x))^2 + \epsilon}
    \]

- Ulyanov et al. shows significant improvement in style transfer by replacing BN with IN layers.
Background: Conditional Instance Normalization (CIN)

- Based on Instance Normalization
- Learn different set of parameters $\gamma^s, \beta^s$ for each style $s$

\[
\text{CIN}(x; s) = \gamma^s \left( \frac{x - \mu(x)}{\sigma(x)} \right) + \beta^s
\]

- Supports multiple different styles (32) using a single network
  - Instance Normalization supports only single style
- Requires retraining for new styles
Interpreting Instance Normalization

- Normalization of contrast of the content images doesn’t help
- Instance normalization is style normalization
- Convolutional feature statistics captures style
- Different styles have different statistics
Adaptive Instance Normalization (AdaIN)

- Based on Instance Normalization, but sheared params are calculated, not learned
- Can be used for arbitrary styles without retraining
- Normalize content features (like IN)
- Shear content features by style feature statistics (mean and variance)

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

- Feature statistics captures style
- Spatial structure captures content
- Normalize by style of content, shear by desired style
Proposed Style Transfer Algorithm

- Fixed first few layers of pretrained VGG-net
- Decoder mirrors encoder
  - Nearest upsampling instead of pooling
  - No normalization layers in decoder
- AdaIN layer aligns the channel-wise mean and variance of content image to style image
VGG-19

- Designed for image recognition
- 3x3 convolutional layers
- With max pooling layers in between
- Followed by fully connected layers
- And softmax
- Deeper layers capture complex features

Image from Stanford cs231n
Visualizing deep convolutional neural networks using natural pre-images

by Mahendran et al.
Training

- Trained using
  - MS-COCO for the content images
  - Paintings from WikiArt for the style images
  - Adam optimizer (instead of a stochastic gradient descent procedure to update the network weights in backpropagation)

- During training the images are resized to be all be 256 x 256
  - Resized the smallest dimension to be 512 while preserving the aspect ratio
  - Then randomly cropped to 256 x 256
Training

- Loss function computed from the pre-trained VGG-19
  \[ \mathcal{L} = \mathcal{L}_c + \lambda \mathcal{L}_s \]

- Content loss is the Euclidean distance between the target features and the features of the output from the AdaIN layer
  - Used AdaIN output as content target
  \[ \mathcal{L}_c = \| f(g(t)) - t \|_2 \]

- The style loss is calculated in terms of mean and standard deviation since their AdaIN layer transfers the mean and standard deviation
  \[ \mathcal{L}_s = \sum_{i=1}^{L} \| \mu(\phi_i(g(t))) - \mu(\phi_i(s)) \|_2 + \sum_{i=1}^{L} \| \sigma(\phi_i(g(t))) - \sigma(\phi_i(s)) \|_2 \]
Results

Chen & schmidt - patch based

Ulyanov et al. - fast feed-forward method that is restricted to a single style

Gatys et al. - original optimization based method (flexible but slow)
Loss Comparisons

- **Gatys** method is 3 orders of magnitude slower
- **Ulyanov** method needed to be trained on the test styles
- **AdaIN** method did not see the test styles beforehand, therefore is the best
Comparison with Baselines

- Concatenation method unable to transfer just the style
  - Fails to disentangle the style information from the content of the style image
- BN and IN decoders
  - Unable to preserve the original content
  - Underperformed with the style transfer
- AdaIN more effective at combining the content and style
Training Curves of Style and Content Loss

- Want low style and content loss
- AdaIN does the best job preserving both style and content
Speed Comparison

Achieves speeds similar to the those who train on a limited set of styles while achieving the flexibility of being applicable to arbitrary styles

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (256px)</th>
<th>Time (512px)</th>
<th># Styles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gatys et al.</td>
<td>14.17 (14.19)</td>
<td>46.75 (46.79)</td>
<td>∞</td>
</tr>
<tr>
<td>Chen and Schmidt</td>
<td>0.171 (0.407)</td>
<td>3.214 (4.144)</td>
<td>∞</td>
</tr>
<tr>
<td>Ulyanov et al.</td>
<td>0.011 (N/A)</td>
<td>0.038 (N/A)</td>
<td>1</td>
</tr>
<tr>
<td>Dumoulin et al.</td>
<td>0.011 (N/A)</td>
<td>0.038 (N/A)</td>
<td>32</td>
</tr>
<tr>
<td>Ours</td>
<td><strong>0.018 (0.027)</strong></td>
<td><strong>0.065 (0.098)</strong></td>
<td>∞</td>
</tr>
</tbody>
</table>

* Time in parentheses includes the style encoding procedure
Speed Comparison

Achieves speeds similar to the those who train on a limited set of styles while achieving the flexibility of being applicable to arbitrary styles

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (256px)</th>
<th>Time (512px)</th>
<th># Styles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gatys et al.</td>
<td>14.17 (14.19)</td>
<td>46.75 (46.79)</td>
<td>∞</td>
</tr>
<tr>
<td>Chen and Schmidt</td>
<td>0.171 (0.407)</td>
<td>3.214 (4.144)</td>
<td>∞</td>
</tr>
<tr>
<td>Ulyanov et al.</td>
<td>0.011 (N/A)</td>
<td>0.038 (N/A)</td>
<td>1</td>
</tr>
<tr>
<td>Dumoulin et al.</td>
<td>0.011 (N/A)</td>
<td>0.038 (N/A)</td>
<td>32</td>
</tr>
<tr>
<td>Ours</td>
<td><strong>0.018 (0.027)</strong></td>
<td><strong>0.065 (0.098)</strong></td>
<td>∞</td>
</tr>
</tbody>
</table>

* Time in parentheses includes the style encoding procedure
Speed Comparison

Achieves speeds similar to the those who train on a limited set of styles while achieving the flexibility of being applicable to arbitrary styles.

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (256px)</th>
<th>Time (512px)</th>
<th># Styles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gatys et al.</td>
<td>14.17 (14.19)</td>
<td>46.75 (46.79)</td>
<td>∞</td>
</tr>
<tr>
<td>Chen and Schmidt</td>
<td>0.171 (0.407)</td>
<td>3.214 (4.144)</td>
<td>∞</td>
</tr>
<tr>
<td>Ulyanov et al.</td>
<td>0.011 (N/A)</td>
<td>0.038 (N/A)</td>
<td>1</td>
</tr>
<tr>
<td>Dumoulin et al.</td>
<td>0.011 (N/A)</td>
<td>0.038 (N/A)</td>
<td>32</td>
</tr>
<tr>
<td>Ours</td>
<td>0.018 (0.027)*</td>
<td>0.065 (0.098)</td>
<td>∞</td>
</tr>
</tbody>
</table>

* Time in parentheses includes the style encoding procedure
Runtime Controls

The Huang-Belongie method also allows users to control various properties of the style transfer including:

- degree of stylization
- interpolation between different/multiple styles
- transferring the style while preserving original colors
- using different styles in different spatial regions

All of these controls are applied at runtime using a network that has already been trained.
Content-Style trade off

The degree of style transfer can be controlled in two ways
1. During training by adjusting the style weight $\lambda$ in the loss function
   a. The higher the value of $\lambda$ the more the style will be transferred
   b. The lower the value of $\lambda$ the less the style will be transferred

2. At test time by interpolating between feature maps that are fed to the decoder.
   a. $\alpha = 0$ the network will reconstruct the content image
   b. $\alpha = 1$ the network will reconstruct the most stylized image

$$T(c, s, \alpha) = g((1 - \alpha)f(c) + \alpha\text{AdaIN}(f(c), f(s)))$$
Multiple-Style Interpolation

Interpolate between a set of K style images \( s_i \) with weights \( w_i \) st the sum of the weights is one

\[
T(c, s_{1,2,...,K}, w_{1,2,...,K}) = g\left(\sum_{k=1}^{K} w_k \text{AdaIN}(f(c), f(s_k))\right)
\]
Multiple-Style Interpolation
Multiple-Style Interpolation
Multiple-Style Interpolation
Color Control

- Matches the color distribution of the content image to the style image so the style image
- Now perform style transfer with this color-aligned style image
Spatial Control

To style match different areas of an image to different styles

- Use masks to separate out the desired regions
- Run the AdaIN separately for each region
Prior Work

Gaty’s - 2016
● Showed that DNN encode style and content of images
● Style and content are somewhat separable
● Optimization process manipulated pixel values to match feature statistics

Ulyanov - 2017
● Improved speed with feed-forward neural network
● Trained a network to modify pixel values to indirectly match feature statistics
Presented Work

Huang & Belongie - 2017

- Created single feedforward network with an Adaptive Instance Normalization Layer to transfer feature statistics from the style image to the content image
- Directly aligned image statistics in the feature space in one shot with the AdaIN layer then inverted the features back to pixel space
Future Work

● More advanced network architectures
  ○ Residual architecture
  ○ Architecture with additional skip connections from the encoder

● More complicated training schemes
  ○ Incremental training

● Replace AdaIN with
  ○ Correlation alignment
  ○ Histogram matching
  ○ Using higher-order statistics
Summary

Compared Instance and Batch Normalization
- Showed that it is optimal to use neither in the decoder
- Showed that Instance Normalization performs a form of style normalization by normalizing feature statistics

AdaIN
- Aligns the channel-wise mean & variance
- Allows for style transfer to arbitrary styles in real-time