Convolutional Neural Networks + Neural Style Transfer

Justin Johnson
2/1/2017
Outline

● Convolutional Neural Networks
  ○ Convolution
  ○ Pooling
  ○ Feature Visualization

● Neural Style Transfer
  ○ Feature Inversion
  ○ Texture Synthesis
  ○ Style Transfer
Convolutional Neural Networks: Deep Learning with Images
The Image Classification Challenge:
1,000 object classes
1,431,167 images

Output:
- Scale
- T-shirt
- Steel drum
- Drumstick
- Mud turtle

Output:
- Scale
- T-shirt
- Giant panda
- Drumstick
- Mud turtle

Russakovsky et al. arXiv, 2014
Object Detection = What, and Where

Localization

Where?

Recognition

What?

car: 1.000
person: 0.992
doctor: 0.997
horse: 0.993
person: 0.979

Slide credit: Kaiming He, ICCV 2015
Object segmentation
Pose Estimation

Figure credit: Cao et al, "Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields", arXiv 2016
Image Captioning

Figure credit: Karpathy and Fei-Fei, “Deep Visual-Semantic Alignments for Generating Image Descriptions”, CVPR 2015
Dense Image Captioning

Figure credit: Johnson*, Karpathy*, and Fei-Fei, “DenseCap: Fully Convolutional Localization Networks for Dense Captioning”, CVPR 2016
## Visual Question Answering

<table>
<thead>
<tr>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image 1" /></td>
</tr>
<tr>
<td><img src="image2.png" alt="Image 2" /></td>
</tr>
<tr>
<td><img src="image3.png" alt="Image 3" /></td>
</tr>
<tr>
<td><img src="image4.png" alt="Image 4" /></td>
</tr>
<tr>
<td><img src="image5.png" alt="Image 5" /></td>
</tr>
</tbody>
</table>

### Questions and Answers

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>What color are her eyes?</td>
<td>Catcher, Umpire, Ball girl</td>
</tr>
<tr>
<td>What is the mustache made of?</td>
<td>Catcher, Umpire, Ball girl</td>
</tr>
<tr>
<td>How many slices of pizza are there?</td>
<td>Catcher, Umpire, Ball girl</td>
</tr>
<tr>
<td>Is this a vegetarian pizza?</td>
<td>Catcher, Umpire, Ball girl</td>
</tr>
<tr>
<td>Is this person expecting company?</td>
<td>Catcher, Umpire, Ball girl</td>
</tr>
<tr>
<td>What is just under the tree?</td>
<td>Catcher, Umpire, Ball girl</td>
</tr>
<tr>
<td>Does it appear to be rainy?</td>
<td>Catcher, Umpire, Ball girl</td>
</tr>
<tr>
<td>Does this person have 20/20 vision?</td>
<td>Catcher, Umpire, Ball girl</td>
</tr>
</tbody>
</table>

### Figure Credits
Figure credit: Ledig et al, "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network", arXiv 2016
Generating Art

Figure credit: Mordvintsev, Olah, and Tyka, “Inceptionism: Going Deeper into Neural Networks”,
https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

Figure credit: Gatys, Ecker, and Bethge, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016

Figure credit: Johnson, Alahi, and Fei-Fei: “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016, https://github.com/jcjohnson/fast-neural-style
What is a Convolutional Neural Net?
Fully-Connected Neural Network

\[ x \times W_1 \rightarrow s \rightarrow a \rightarrow \hat{y} \]

- \( x \) of size \( C_1 \)
- \( s \) of size \( C_2 \)
- \( a \) of size \( C_2 \)
- \( \hat{y} \) of size \( C_3 \)

Matrix Multiply
Nonlinearity
Matrix Multiply

\( W_1 \) of size \( C_2 \times C_1 \)
\( W_2 \) of size \( C_3 \times C_2 \)
Convolutional Neural Network

\[ x \in C_1 \times H \times W \]

Convolution

\[ s \in C_2 \times H \times W \]

Nonlinearity

\[ a \in C_2 \times H \times W \]

Pooling

\[ p \in C_2 \times H/2 \times W/2 \]

Fully Connected

\[ \hat{y} \in C_3 \]
Convolution Layer

32x32x3 image

- Width: 32
- Height: 32
- Depth: 3

Slide credit: CS231n Lecture 7
Convolution Layer

32x32x3 image

5x5x3 filter

**Convolve** the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolution Layer

32x32x3 image

5x5x3 filter

Filters always extend the full depth of the input volume

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Slide credit: CS231n Lecture 7
Convolution Layer

32x32x3 image
5x5x3 filter $w$

1 number: the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. $5*5*3 = 75$-dimensional dot product + bias)

$$w^T x + b$$
Convolution Layer

- 32x32x3 image
- 5x5x3 filter
- convolve (slide) over all spatial locations

activation map

Slide credit: CS231n Lecture 7
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

consider a second, green filter

activation maps

Slide credit: CS231n Lecture 7
For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get a “new image” of size 28x28x6!
Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:

Slide credit: CS231n Lecture 7
MAX POOLING

Single depth slice

\[
\begin{array}{cccc}
1 & 1 & 2 & 4 \\
5 & 6 & 7 & 8 \\
3 & 2 & 1 & 0 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

max pool with 2x2 filters and stride 2

\[
\begin{array}{cc}
6 & 8 \\
3 & 4 \\
\end{array}
\]

Slide credit: CS231n Lecture 7
Case Study: LeNet-5

[LeCun et al., 1998]

Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]
Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1 and 2x2 MAX POOL stride 2

11.2% top 5 error in ILSVRC 2013

->

7.3% top 5 error

Slide credit: CS231n Lecture 7
Case Study: GoogLeNet  

Inception module  

ILSVRC 2014 winner (6.7% top 5 error)
Case Study: ResNet

[He et al., 2015]

Slide credit: CS231n Lecture 7

spatial dimension only 56x56!
Case Study: ResNet [He et al., 2015]
ILSVRC 2015 winner (3.6% top 5 error)

2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet!
(even though it has 8x more layers)

Revolution of Depth

- AlexNet, 8 layers (ILSVRC 2012)
- VGG, 19 layers (ILSVRC 2014)
- ResNet, 152 layers (ILSVRC 2015)

(slide from Kaiming He’s ICCV 2015 presentation)
Revolution of Depth

ImageNet Classification top-5 error (%)

ILSVRC'15 ResNet 3.57
ILSVRC'14 GoogleNet 6.7
ILSVRC'14 VGG 7.3
ILSVRC'13 8 layers 11.7
ILSVRC'12 AlexNet 8 layers 16.4
ILSVRC'11 shallow 25.8
ILSVRC'10 shallow 28.2

(slide from Kaiming He’s ICCV 2015 presentation)
Visualizing ConvNet Features
Visualizing CNN features: Look at filters

Slide credit: CS231n Lecture 9
Many networks learn similar filters
Visualizing CNN features: Look at filters

Filters from higher layers don’t make much sense
Visualizing CNN features: (Guided) Backprop

Choose an image

Choose a layer and a neuron in a CNN

Question:
How does the chosen neuron respond to the image?
Visualizing CNN features: (Guided) Backprop

1. Feed image into net

Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

Slide credit: CS231n Lecture 9
Visualizing CNN features: (Guided) Backprop

1. Feed image into net

Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

2. Set gradient of chosen layer to all zero, except 1 for the chosen neuron
Visualizing CNN features: (Guided) Backprop

1. Feed image into net

2. Set gradient of chosen layer to all zero, except 1 for the chosen neuron

3. Backprop to image:

Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014


Slide credit: CS231n Lecture 9
Visualizing CNN features: (Guided) Backprop

1. Feed image into net

2. Set gradient of chosen layer to all zero, except 1 for the chosen neuron

3. Backprop to image:

Guided backpropagation: instead

Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014
Slide credit: CS231n Lecture 9
Visualization of patterns learned by the layer \texttt{conv6} (top) and layer \texttt{conv9} (bottom) of the network trained on ImageNet.

Each row corresponds to one filter.

The visualization using “guided backpropagation” is based on the top 10 image patches activating this filter taken from the ImageNet dataset.

Visualizing CNN features: Gradient Ascent

(Guided) backprop: Find the part of an image that a neuron responds to

Gradient ascent: Generate a synthetic image that maximally activates a neuron

\[ I^* = \arg \max_I f(I) + R(I) \]

Neuron value
Natural image regularizer
Visualizing CNN features: Gradient Ascent

1. Initialize image to zeros

\[
\arg \max_I S_c(I) - \lambda \| I \|_2^2
\]

score for class c (before Softmax)

Repeat:
2. Forward image to compute current scores
3. Set gradient of scores to be 1 for target class, 0 for others
4. Backprop to get gradient on image
5. Make a small update to the image
Visualizing CNN features: Gradient Ascent

Visualizing CNN features: Gradient Ascent

Better image regularizers give prettier results:

- Flamingo
- Pelican
- Hartebeest
- Billiard Table
- Ground Beetle
- Indian Cobra
- Station Wagon
- Black Swan
Visualizing CNN features: Gradient Ascent

Use the same approach to visualize intermediate features
Visualizing CNN features: Gradient Ascent

Use the same approach to visualize intermediate features

Visualizing CNN features: Gradient Ascent

You can add even more tricks to get nicer results:

Nguyen et al, “Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks”, ICML Visualization for Deep Learning Workshop 2016
Visualizing CNN features: Gradient Ascent

GAN image priors give amazing results:

Feature Inversion

Given a feature vector for an image, find a new image such that:
- Its features are similar to the given features
- It “looks natural” (image prior regularization)

Feature Inversion

Given a feature vector for an image, find a new image such that:
- Its features are similar to the given features
- It “looks natural” (image prior regularization)

\[
\mathbf{x}^* = \arg\min_{\mathbf{x} \in \mathbb{R}^{H \times W \times C}} \ell(\Phi(\mathbf{x}), \Phi_0) + \lambda \mathcal{R}(\mathbf{x})
\]

\[
\ell(\Phi(\mathbf{x}), \Phi_0) = \|\Phi(\mathbf{x}) - \Phi_0\|^2
\]

\[
\mathcal{R}_{V^\beta}(\mathbf{x}) = \sum_{i,j} \left( (x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\beta/2}
\]

Feature Inversion

Given a feature vector for an image, find a new image such that:
- Its features are similar to the given features
- It “looks natural” (image prior regularization)

\[
x^* = \arg\min_{x \in \mathbb{R}^{H \times W \times C}} \ell(\Phi(x), \Phi_0) + \lambda \mathcal{R}(x)
\]

\[
\ell(\Phi(x), \Phi_0) = \|\Phi(x) - \Phi_0\|^2
\]

\[
\mathcal{R}_{V^\beta}(x) = \sum_{i,j} \left( (x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\beta}{2}}
\]

Given feature vector

Features of new image

Total Variation regularizer (encourages spatial smoothness)

Feature Inversion

Reconstructions from the 1000 log probabilities for ImageNet (ILSVRC) classes

Feature Inversion

Reconstructions from the representation after last last pooling layer (immediately before the first Fully Connected layer)

Feature Inversion

Reconstructions from intermediate layers

Higher layers are less sensitive to changes in color, texture, and shape

(Neural) Texture Synthesis
Texture Synthesis

Given a sample patch of some texture, can we generate a bigger image of the same texture?

Input  Output
Texture Synthesis


Texture Synthesis


I have a Torch implementation here: https://github.com/jcjohnson/texture-synthesis
Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)

2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$

3. At each layer compute the Gram matrix giving outer product of features:

4. Initialize generated image from random noise

5. Pass generated image through CNN, compute Gram matrix on each layer

6. Compute loss: weighted sum of L2 distance between Gram matrices

7. Backprop to get gradient on image

8. Make gradient step on image

9. GOTO 5
Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer $i$ gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the Gram matrix giving outer product of features:
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5
Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer $i$ gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the Gram matrix giving outer product of features:

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

(shape $C_i \times H_i$)
Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer $i$ gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the Gram matrix giving outer product of features:
   \[
   G_{ij}^l = \sum_k F_{ik}^l \cdot F_{jk}^l \quad \text{(shape } C_i \times H_i)\]
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5
Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer \( i \) gives feature map of shape \( C_i \times H_i \times W_i \)
3. At each layer compute the Gram matrix giving outer product of features:
   \[
   G^L_{ij} = \sum_k F^l_{ik} F^l_{jk} \text{(shape } C_i \times H_i)\]
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5
Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer $i$ gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the Gram matrix giving outer product of features:
   \[
   G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \quad \text{(shape $C_i \times H_i$)}
   \]
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
   \[
   E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left( G_{ij}^l - \hat{G}_{ij}^l \right)^2
   \]
   \[
   \mathcal{L}(\hat{x}, \tilde{x}) = \sum_{l=0}^{L} w_l E_l
   \]
Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer $i$ gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the Gram matrix giving outer product of features:
   \[ G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \text{(shape } C_i \times H_i) \]
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image

\[ E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left( G_{ij}^l - \hat{G}_{ij}^l \right)^2 \]
\[ \mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{l=0}^{L} w_l E_l \]
Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer $i$ gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the Gram matrix giving outer product of features:
   \[
   G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \text{ (shape } C_i \times H_i) \]
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5

---

Neural Texture Synthesis

Reconstructing from higher layers recovers larger features from the input texture

Style Transfer: Feature Inversion + Texture Synthesis
Neural Style Transfer: Feature + Gram reconstruction

Feature reconstruction

Texture synthesis (Gram reconstruction)

Figure credit: Johnson et al, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016
Neural Style Transfer

Given a **content image** and a **style image**, find a new image that
- Matches the CNN features of the content image (feature reconstruction)
- Matches the Gram matrices of the style image (texture synthesis)

Combine feature reconstruction from Mahendran et al with Neural Texture Synthesis from Gatys et al, using the same CNN!
Neural Style Transfer

Given a **content image** and a **style image**, find a new image that
- Matches the CNN features of the content image (feature reconstruction)
- Matches the Gram matrices of the style image (texture synthesis)

Combine feature reconstruction from Mahendran et al with Neural Texture Synthesis from Gatys et al, using the same CNN!
Neural Style Transfer

1. Pretrain CNN
2. Compute features for content image
3. Compute Gram matrices for style image
4. Randomly initialize new image
5. Forward new image through CNN
6. Compute style loss (L2 distance between Gram matrices) and content loss (L2 distance between features)
7. Loss is weighted sum of style and content losses
8. Backprop to image
9. Take a gradient step
10. GOTO 5

Neural Style Transfer

Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Neural Style Transfer

From my implementation on GitHub:
https://github.com/jcjohnson/neural-style

Neural Style Transfer: Style / Content Tradeoff

More weight to content loss

More weight to style loss

Neural Style Transfer: Style Scale

Resizing style image before running style transfer algorithm can transfer different types of features

Larger style image  Smaller style image

Neural Style Transfer: Multiple Style Images

Mix style from multiple images by taking a weighted average of Gram matrices

Neural Style Transfer: Multiple Style Images

More “Scream”  More “Starry Night”

Neural Style Transfer: Preserve colors

Perform style transfer only on the luminance channel (eg Y in YUV colorspace);
Copy colors from content image

- Normal style transfer
- Color-preserving style transfer

http://blog.deepart.io/2016/06/04/color-independent-style-transfer/
Simultaneous DeepDream and Style Transfer!

Jointly minimize feature reconstruction loss, style reconstruction loss, and maximize DeepDream feature amplification loss!

https://github.com/jcjohnson/fast-neural-style/issues/5
Style Transfer on Video

Running style transfer independently on each video frame results in poor per-frame consistency:

![Image of original frames and style image with independent per-frame processing](image-url)
Running style transfer independently on each video frame results in poor per-frame consistency:

Appearance of the rock formation different in each frame!
Style Transfer on Video

Tricks for video style transfer:
- **Initialization**: Initialize frame $t+1$ with a warped version of the stylized result at frame $t$ (using optical flow)
- **Short-term temporal consistency**: warped forward optical flow should be opposite of backward optical flow
- **Long-term temporal consistency**: When a region is occluded then visible again, it should look the same
- **Multipass processing**: Make multiple forward and backward passes over the video with few iterations per pass
Beyond Gram Matrices: CNNMRF

Idea: Use patch matching like classic texture synthesis, but match patches in CNN feature space rather than pixel space!

\[ E_s(\Phi(x), \Phi(x_s)) = \sum_{i=1}^{m} \| \Psi_i(\Phi(x)) - \Psi_{NN(i)}(\Phi(x_s)) \|^2 \]

(2)

For each neural patch in generated image, find nearest-neighbor neural patch in style image; minimize distance between patches

https://github.com/chuanli11/CNNMRF
Beyond Gram Matrices: CNNMRF

https://github.com/chuanli11/CNNMRF
Fast Style Transfer

**Problem:** Style transfer is slow; need hundreds of forward + backward passes of VGG

**Solution:** Train a feedforward network to perform style transfer!
Fast Style Transfer

(1) Train a feedforward network for each style
(2) Use pretrained CNN to compute same losses as before
(3) After training, stylize images using a single forward pass

Works real-time at test-time!

Fast Style Transfer

- Style: The Starry Night, Vincent van Gogh, 1889
- Style: The Muse, Pablo Picasso, 1935
- Style: Composition VII, Wassily Kandinsky, 1913
- Style: The Great Wave off Kanagawa, Hokusai, 1829-1832

Gatys  Ours  Gatys  Ours

Works real-time on video!

https://github.com/jcjohnson/fast-neural-style
Fast Style Transfer: Texture Networks

Concurrent work with mine
with comparable results

Multiscale architecture for generator

https://github.com/DmitryUlyanov/texture_nets
Fast Style Transfer: Instance Normalization

A minor tweak to the architecture of the generator significantly improves results.

Ulyanov et al, Johnson et al

Batch Normalization

Instance Normalization

Fast Style Transfer: Multiple styles with one network

Use the same network for multiple styles using *conditional instance normalization*:
learn separate scale and shift parameters per style

\[
\begin{align*}
\{\mu, \sigma\} & \quad \Rightarrow \quad \gamma \\
X & \quad \Rightarrow \quad x_{\text{norm}} = (X - \mu) / \sigma \\
\beta & \quad \Rightarrow \quad z = \gamma \cdot x_{\text{norm}} + \beta \\
\end{align*}
\]

At test-time, blend scale and shift parameters for realtime style blending!

https://research.googleblog.com/2016/10/supercharging-style-transfer.html
Fast Style Transfer: Multiple styles with one network

https://research.googleblog.com/2016/10/supercharging-style-transfer.html
For more details on CNNs, take CS 231n in Spring!