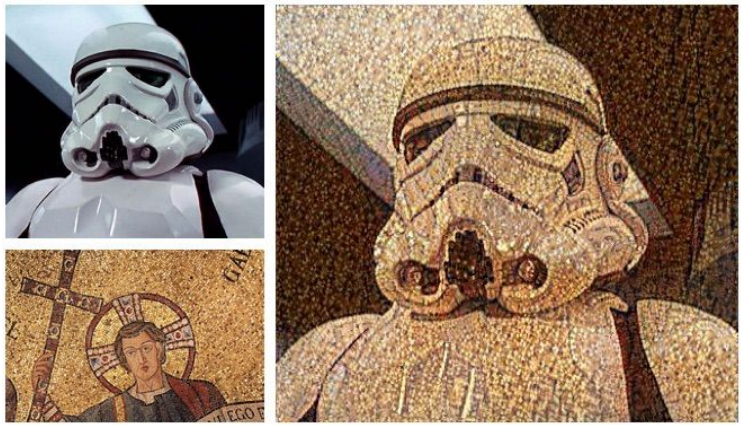
Spatially Factorized Neural Style Transfer

Ian Timmis

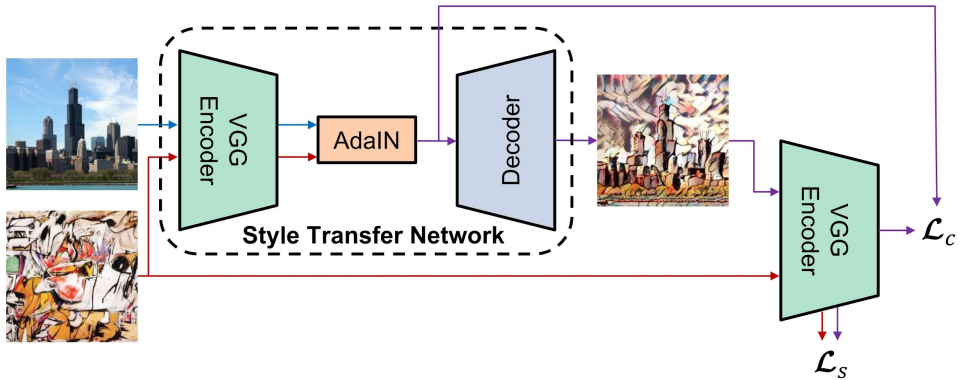
Background - Neural Style transfer

Neural style transfer is the process of using neural networks to transfer the style of one image onto the content another image.



Adaptive Instance Normalization

A modern technique for performing Neural Style transfer is to encode your style and content images with a VGG network, align the mean and variance (this is called adaptive instance normalization), and then decode with convolutional transpose operations. The decoder architecture is essentially a VGG flipped around.



[Xun Huang et al. Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization]

Adaptive Instance Normalization

Key equations:

$$Adain(x, y) = \sigma(y)(\frac{x - \mu(x)}{\sigma(x)}) + \mu(y)$$

$$t = Adain(f(c), f(s))$$

$$T(c, s) = g(t)$$

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

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

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

Adaptive Instance Normalization

Style Interpolation:

$$T(c, s, \alpha) = g((1 - \alpha)f(c) + \alpha Adain(f(c), f(s)))$$

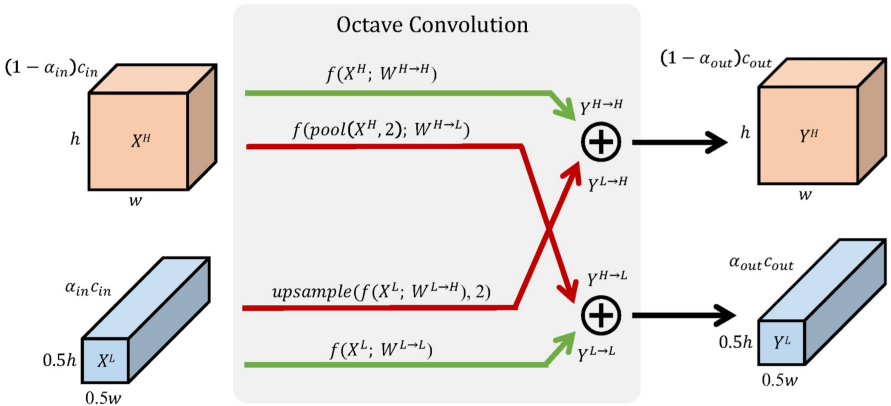
Question

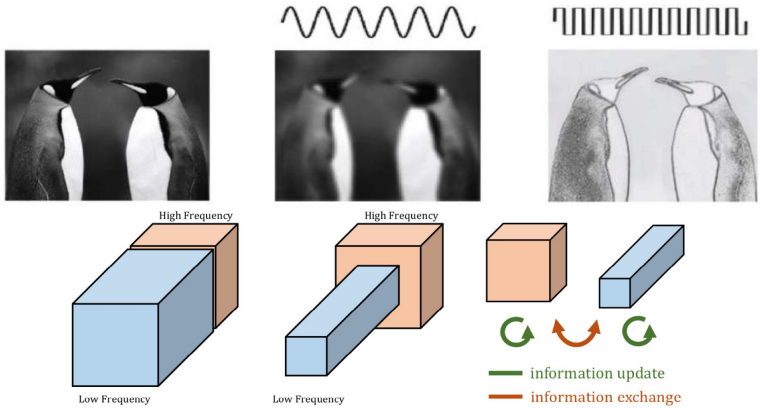
Can we create multiple meaningful axes of style?

I.e. Can we give the user multiple “sliders” in which they can have more control over stylization of their photos and video?

Background - Octave Convolution

Octave convolutions work more efficiently than ordinary convolutional layers by separating the embeddings into high and low frequency regimes by spatially factorizing the input image.

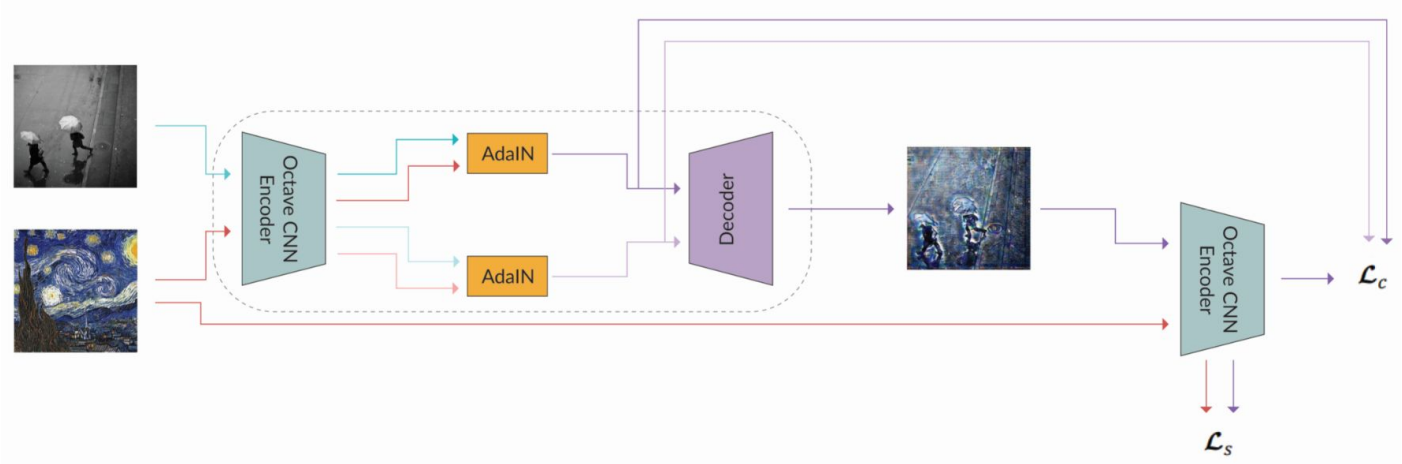




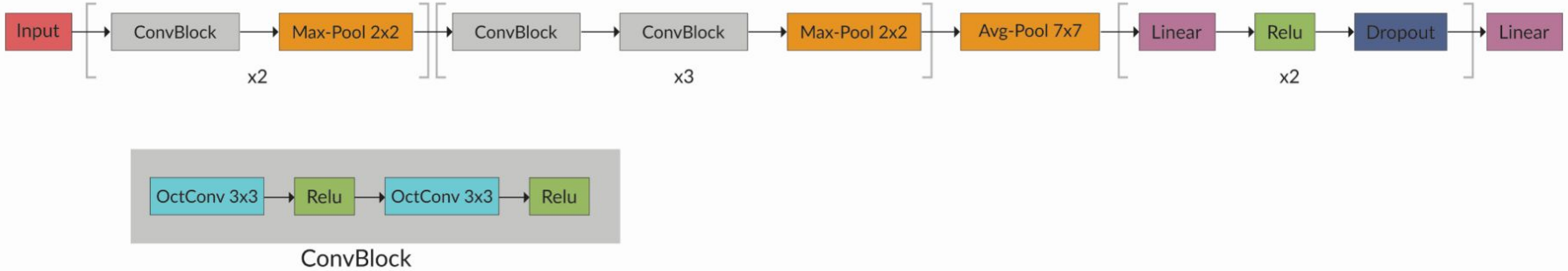
[Yunpeng Chen et al. Drop an Octave: Reducing Spatial Redundancy in Convolutional Neural Networks with Octave Convolution]

Spatially Factorized AdaIN

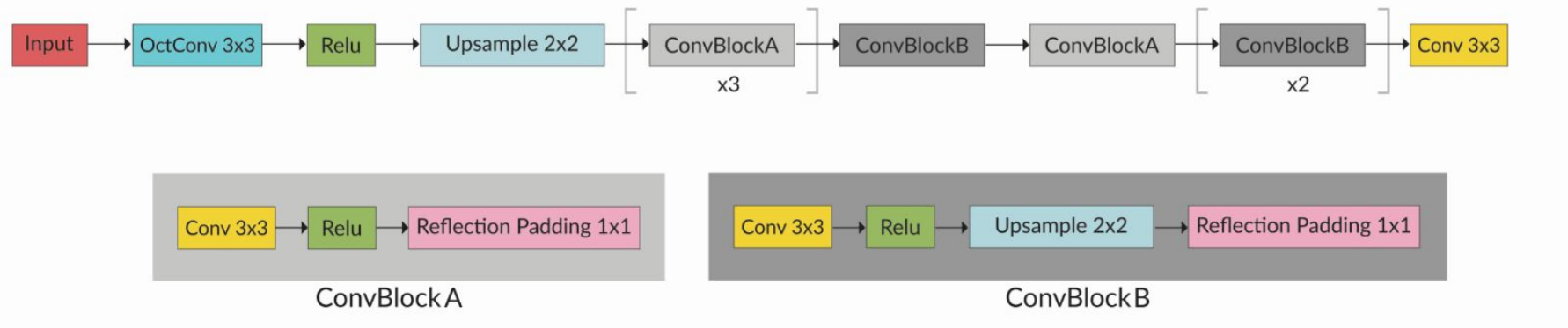
In an attempt to gain more control over stylization outputs, one can spatially factorize the content and style images with Octave Convolution and perform adaptive instance normalization on the high and low frequency features separately.



Encoder Architecture



Decoder Architecture



Spatially Factorized AdaIN - equations

$$Adain(x, y) = \sigma(y)(\frac{x - \mu(x)}{\sigma(x)}) + \mu(y)$$

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

$$t = Adain(f(c), f(s)) \longrightarrow t^l = Adain(f(c)^l, f(s)^l)$$

$$t^h = Adain(f(c)^h, f(s)^h)$$

$$T(c, s) = g(t) \longrightarrow T(c, s) = g(t^h, t^l)$$

$$\mathcal{L}_c = \|f(g(t)) - t\|_2 \longrightarrow \mathcal{L}_c = \beta_c \left\|f(g(t^h, t^l))^h - t^h\right\|_2 + \left\|f(g(t^h, t^l))^l - t^l\right\|_2$$

Spatially Factorized AdaIN - equations

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

$$\mathcal{L}_s = \beta_s(\sum_{i=1}^{L} \left\|\mu(\phi_i(g(t^h, t^l)))^h - \mu(\phi_i(s)^h)\right\|_2 + \sum_{i=1}^{L} \left\|\sigma(\phi_i(g(t^h, t^l))^h) - \sigma(\phi_i(s)^h)\right\|_2) +$$

$$\sum_{i=1}^{L} \left\|\mu(\phi_i(g(t^h, t^l))^l) - \mu(\phi_i(s)^l)\right\|_2 + \sum_{i=1}^{L} \left\|\sigma(\phi_i(g(t^h, t^l))^l) - \sigma(\phi_i(s)^l)\right\|_2$$

Spatially Factorized AdaIN - equations

$$T(c, s, \alpha) = g((1-\alpha)f(c) + \alpha Adain(f(c), f(s)))$$

$$T(c, s, \alpha^h, \alpha^l) = g((1-\alpha^h)f(c)^h + \alpha^h Adain(f(c)^h, f(s)^h), (1-\alpha^l)f(c)^l + \alpha^l Adain(f(c)^l, f(s)^l))$$

Encoder Pre-Training

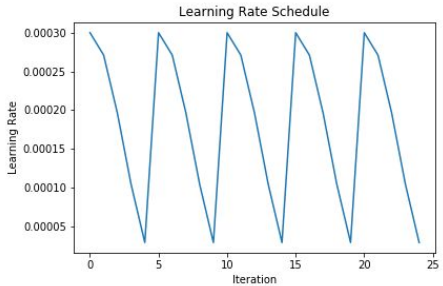
Data (ImageNet):

- Training images: 1,281,167
- Validation images: 50,000
- Classes: 1000

Hyperparameters:

- Optimizer: Ranger (RAdam + Lookahead)
- Weight Decay: 0.1
- LR Schedule: 3e-4 cosine anneal + restarts every 5 epochs
- Batch size: 64

Results: 61% top-1 accuracy



System-level Training

Data:

- MSCOCO: ~80,000 content images
- WikiArt: ~80,000 style images

Hyperparameters:

- Optimizer: RAdam
- Learning Rate: 1e-4
- Learning Rate Decay: 5e-5
- Batch size: 24 content images + 24 style images

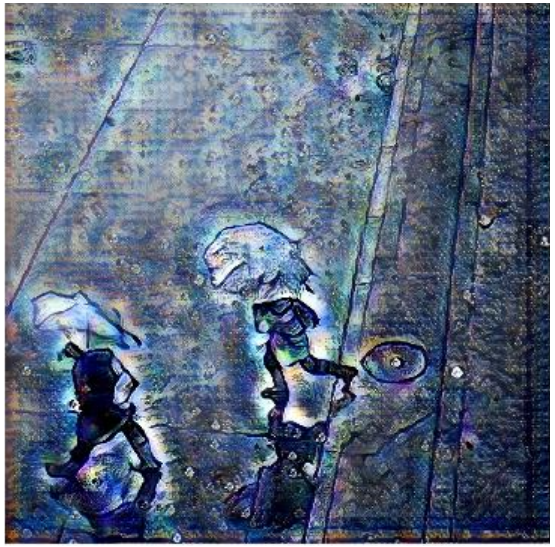
Notes:

- The encoder parameters are frozen during system-level training. Only decoder parameters are trained.

Ablation Study on Loss Weights

Style weight: λ	Content HF weight: β_c	Style HF weight: β_s
10	1	1
5	1	1
.01	1	1
1	1	1
1	5	5
0.1	5	5
1	0.2	0.2
0.1	0.2	0.2
10	5	5

Results



Results

Open Challenges

Issue: Checkerboard artifacts

Possible Solution: Requires reflective-padding in the encoder (already exists in the decoder)

Blocker: Hard to pad OctConv properly

Issue: Imperfect reconstruction

Possible Solution: Requires even more extensive experimentation and examination of loss function. Must be some non-obvious interaction with the LF and HF embeddings, or perhaps a training strategy left out of original AdaIN paper.

Blocker: No obvious path forward. Could be time-consuming

Open Challenges

Issue: Style “shapes” not transferring well

Possible Cause: OctConv was designed to reduce “feature redundancy.” Is this redundancy the reason why AdaIN worked in the first place?

Blocker: