Photography Made Easy

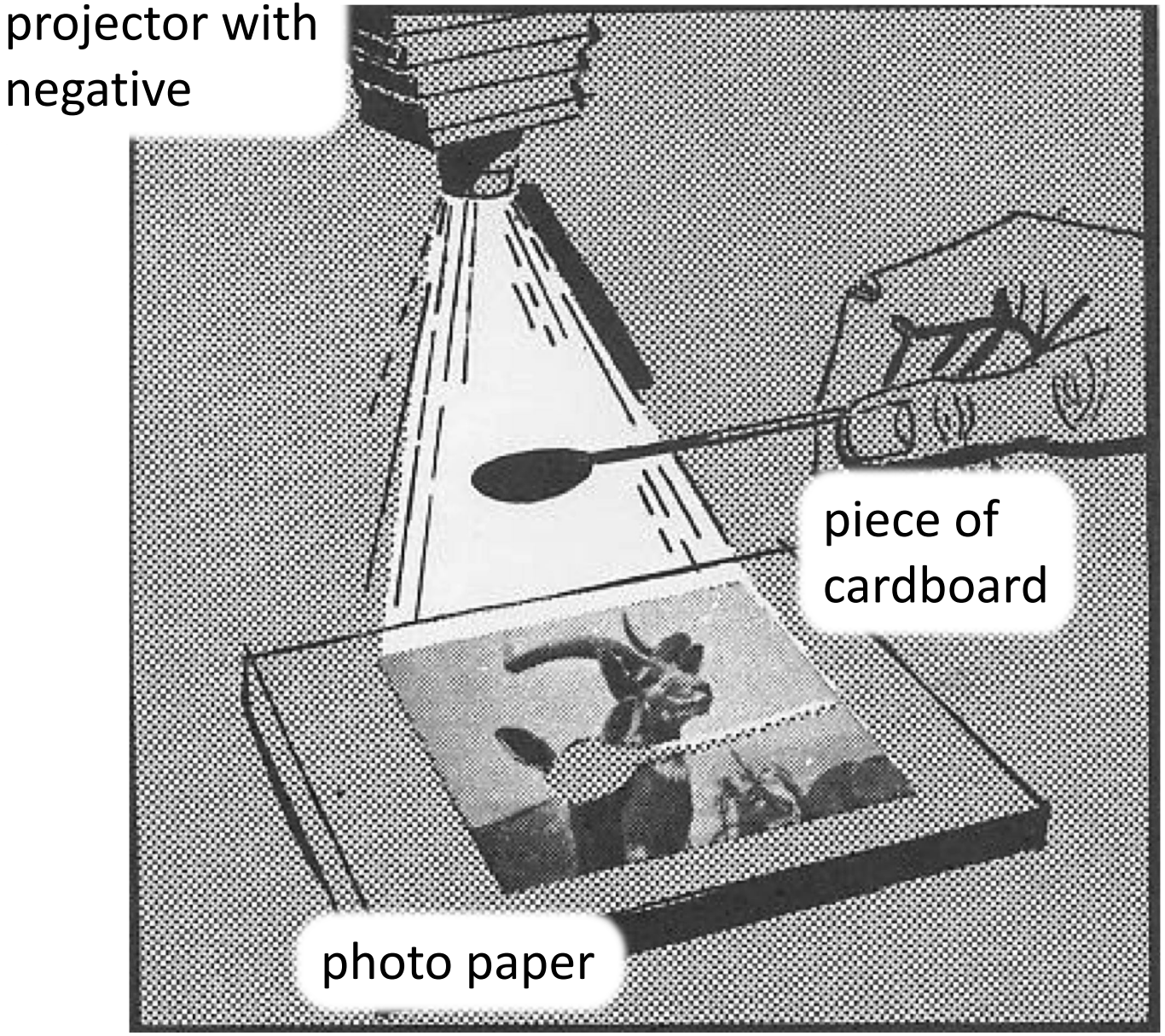
Sylvain Paris, Adobe

Photo before retouching

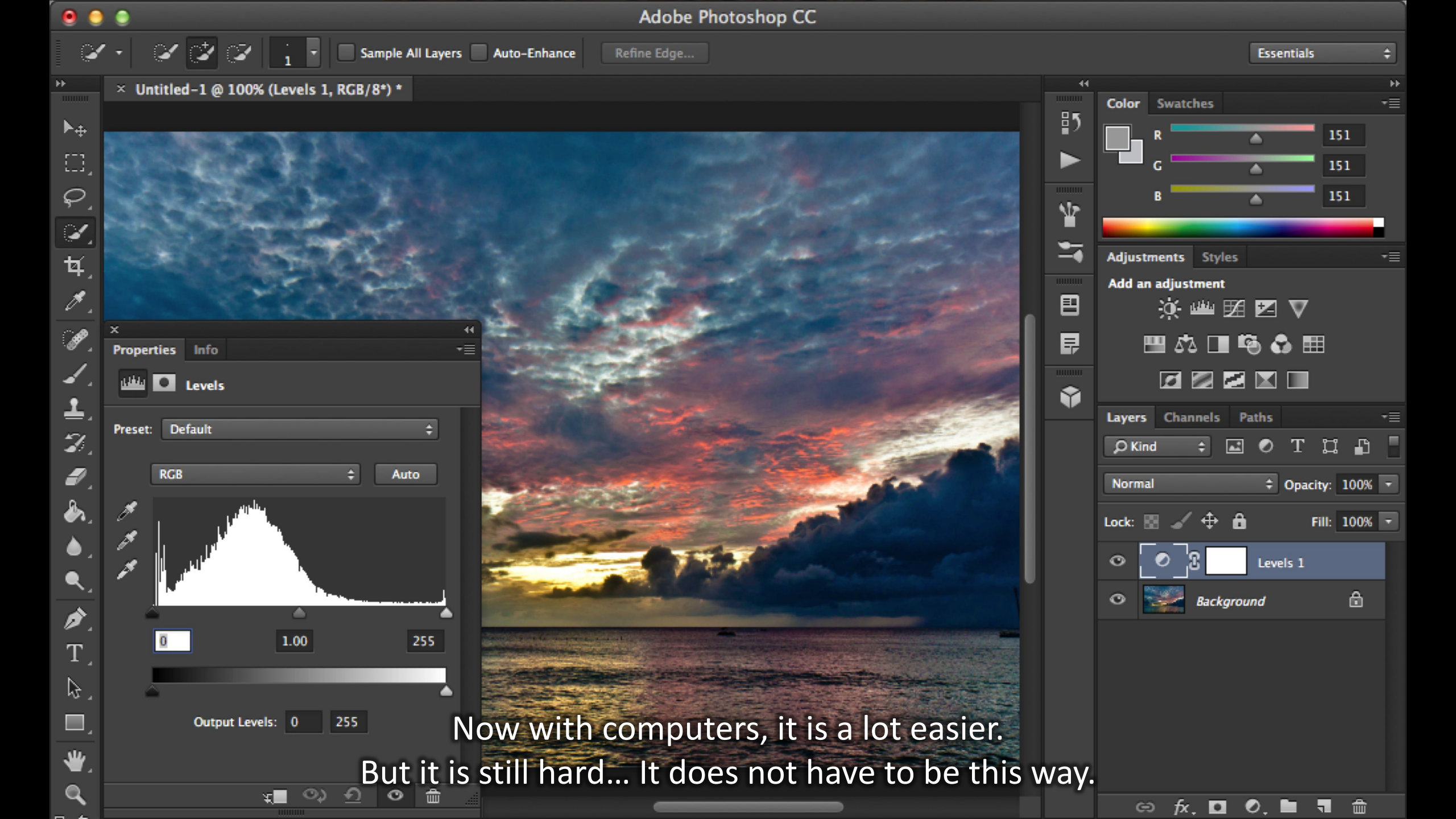
After retouching

Photo before retouching

After retouching



Printing photos used to be hard


Adobe Photoshop CC
Sample All Layers
Auto-Enhance
Refine Edge...
Essentials
Untitled-1 @ 100% (Levels 1, RGB/8*) *
Color
Swatches
R
G
B
151
151
151
Adjustments
Styles
Add an adjustment
Properties
Info
Levels
Preset:
Default
RGB
Auto
0
1.00
255
Output Levels:
0
255
Layers
Channels
Paths
Kind
Normal
Opacity:
100%
Lock:
Fill:
100%
Levels 1
Background
Now with computers, it is a lot easier.
But it is still hard... It does not have to be this way.

Many photos can become great after retouching.

I want to make it easy.

How to Make Photo Retouching Easier

- Some people know how to do it: photographers and artists
- There are plenty of examples of good photos available on the web
- Our strategy
    1. Develop algorithms that transfer the statistics of good photos.
    2. Learn how to retouch from examples of good photos

Photographic Style Transfer

SIGGRAPH 2006

Make this photo

look like

Tonal Aspects of Look: Global Contrast

Ansel Adams

Kenro Izu

High global contrast **Low global contrast**

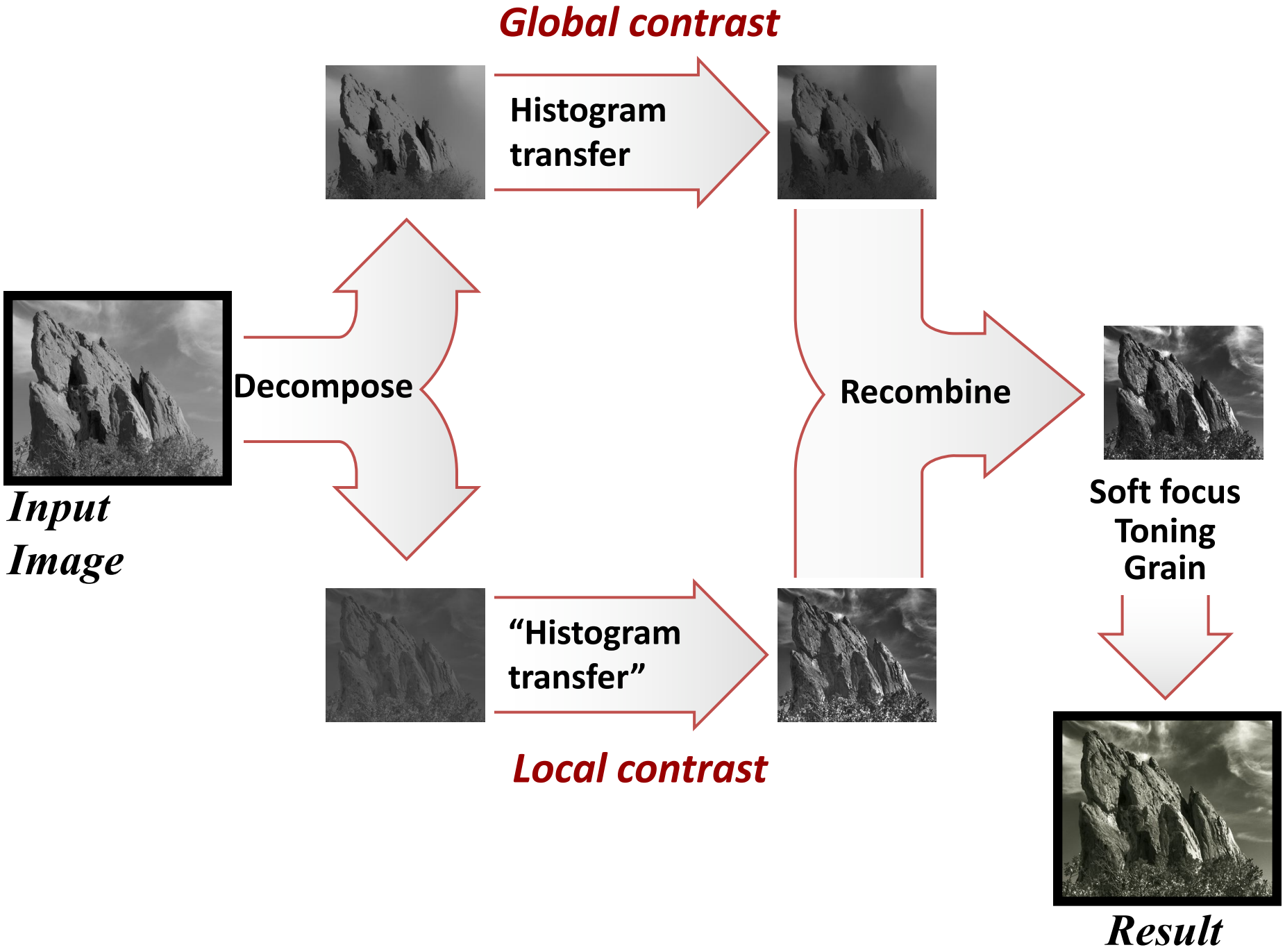
Tonal Aspects of Look: Local Contrast

Ansel Adams

Kenro Izu

Variable amount of texture **Texture everywhere**

Pipeline

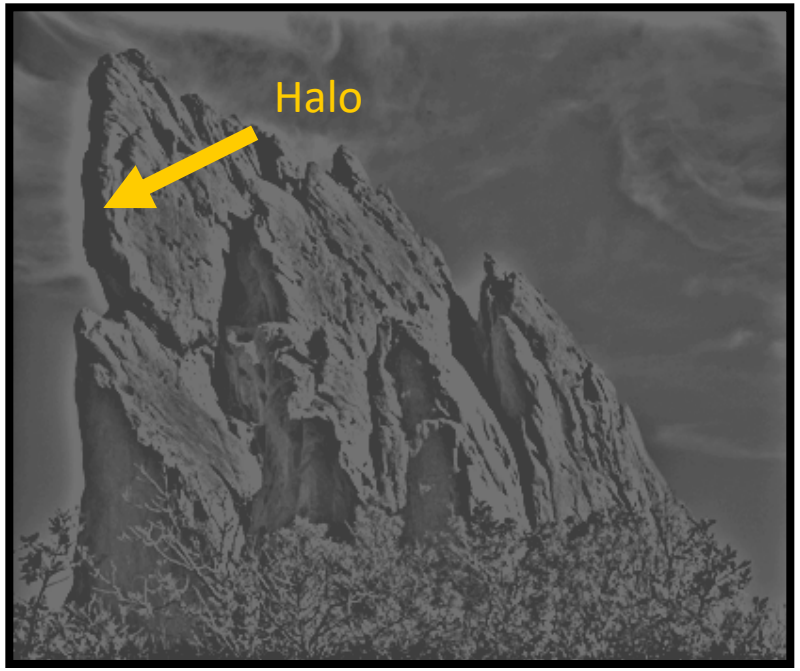

Global contrast
Histogram transfer
Input Image
Decompose
Recombine
Soft focus
Toning
Grain
"Histogram transfer"
Local contrast
Result

Naïve Decomposition: Low vs. High Frequency

- Problem: introduce blur & halos

Low frequency

Global contrast



High frequency

Local contrast

Edge-Preserving Decomposition: Bilateral Filter

Bilateral filter output

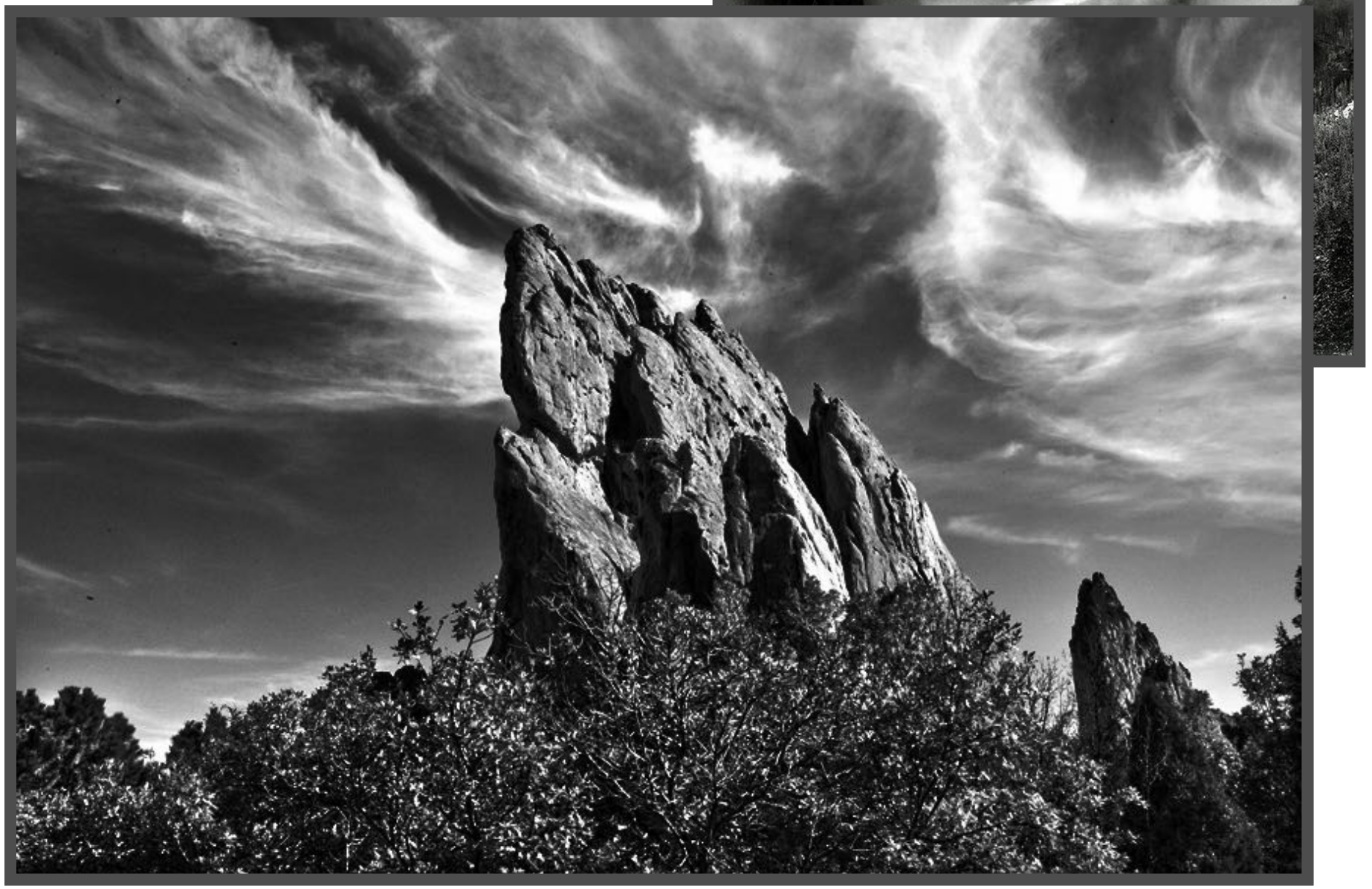
Global contrast

Bilateral filter residual

Local contrast

Result

Model



Result

Photoshop Demo

Local Laplacian Filters

A Better Decomposition

SIGGRAPH 2011

Background on Gaussian Pyramids

- Resolution halved at each level using Gaussian kernel

level 0

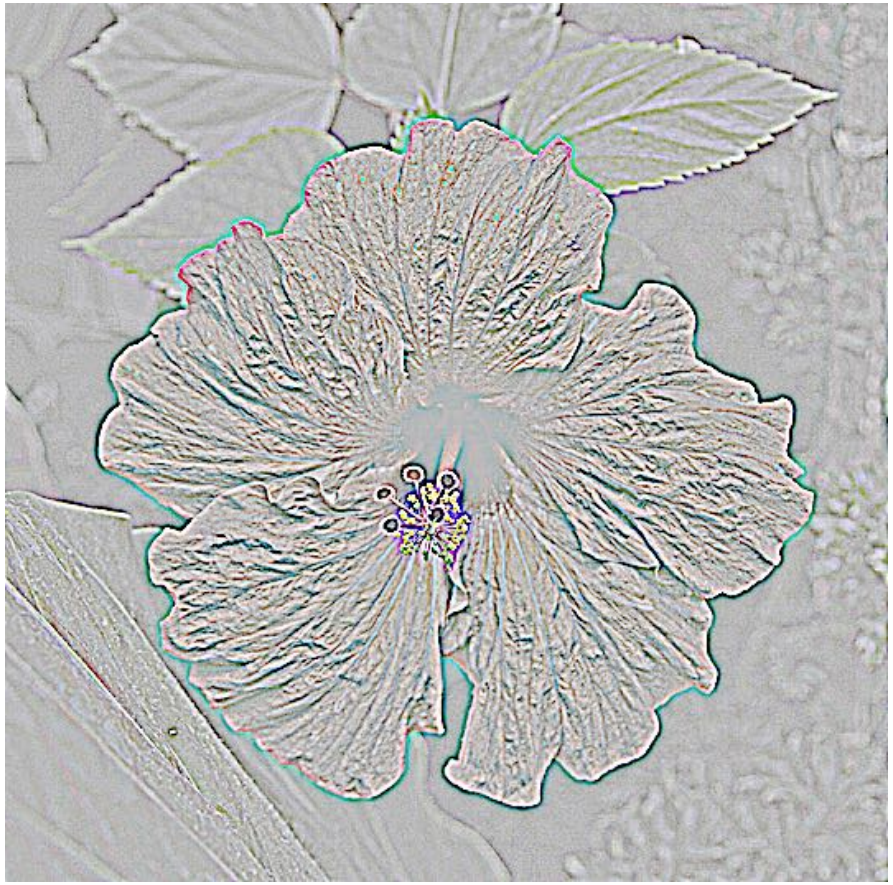
level 1

level 2

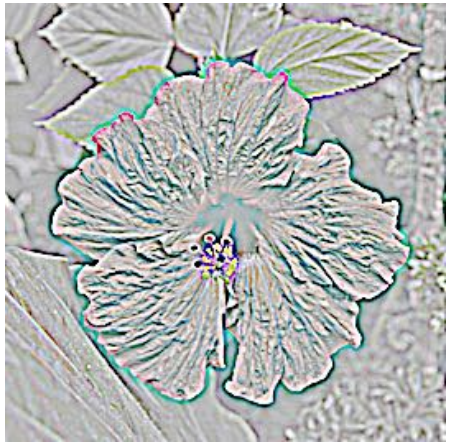
level 3
(residual)

Background on Laplacian Pyramids

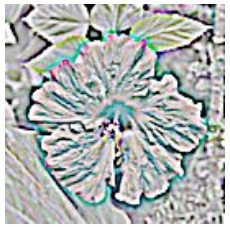
- Difference between adjacent Gaussian levels



level 0



level 1



level 2

level 3
(residual)

Pros and Cons of Pyramids

☺ Useful for compression [Burt 83], texture synthesis [Heeger 95], harmonization [Sunkavalli 10]…

☹ Believed to be unsuitable for edge-aware processing

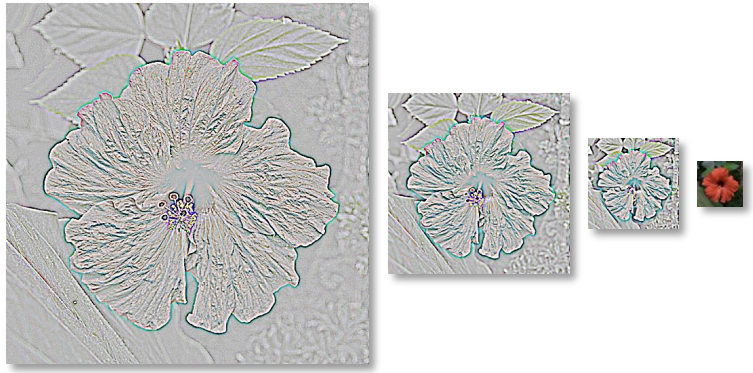
- Use isotropic & spatially invariant kernels
- But edges are anisotropic & well located
- "*Manipulating pyramids generate halos*"

▶ We show otherwise.

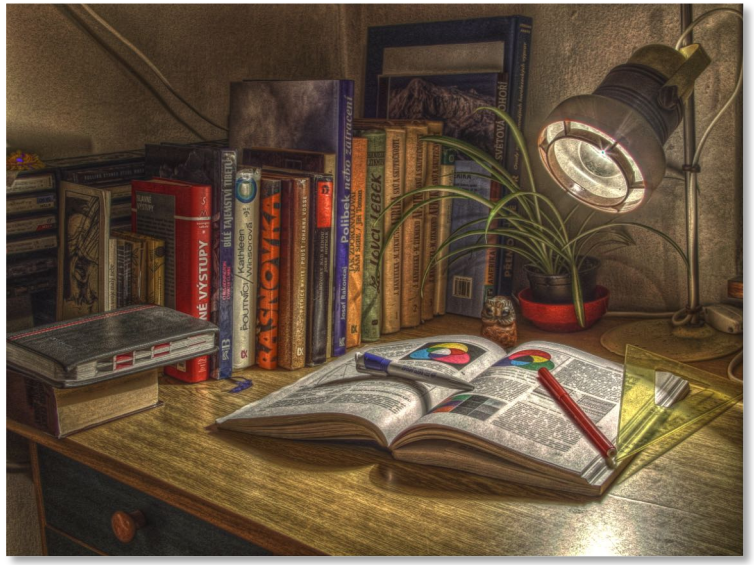
boosting high-frequency bands

Our Contributions

- **Edge-aware editing with Laplacian pyramids**
  - We use a classical multi-scale representation



- **Robustness**: strong effects, no artifacts
  - We achieve extreme enhancements where other methods fail



Our Strategy: Local Adaptation

1. Generate an image that looks good for a small neighborhood
   - It may look bad elsewhere

2. “Combine data from all neighborhoods”

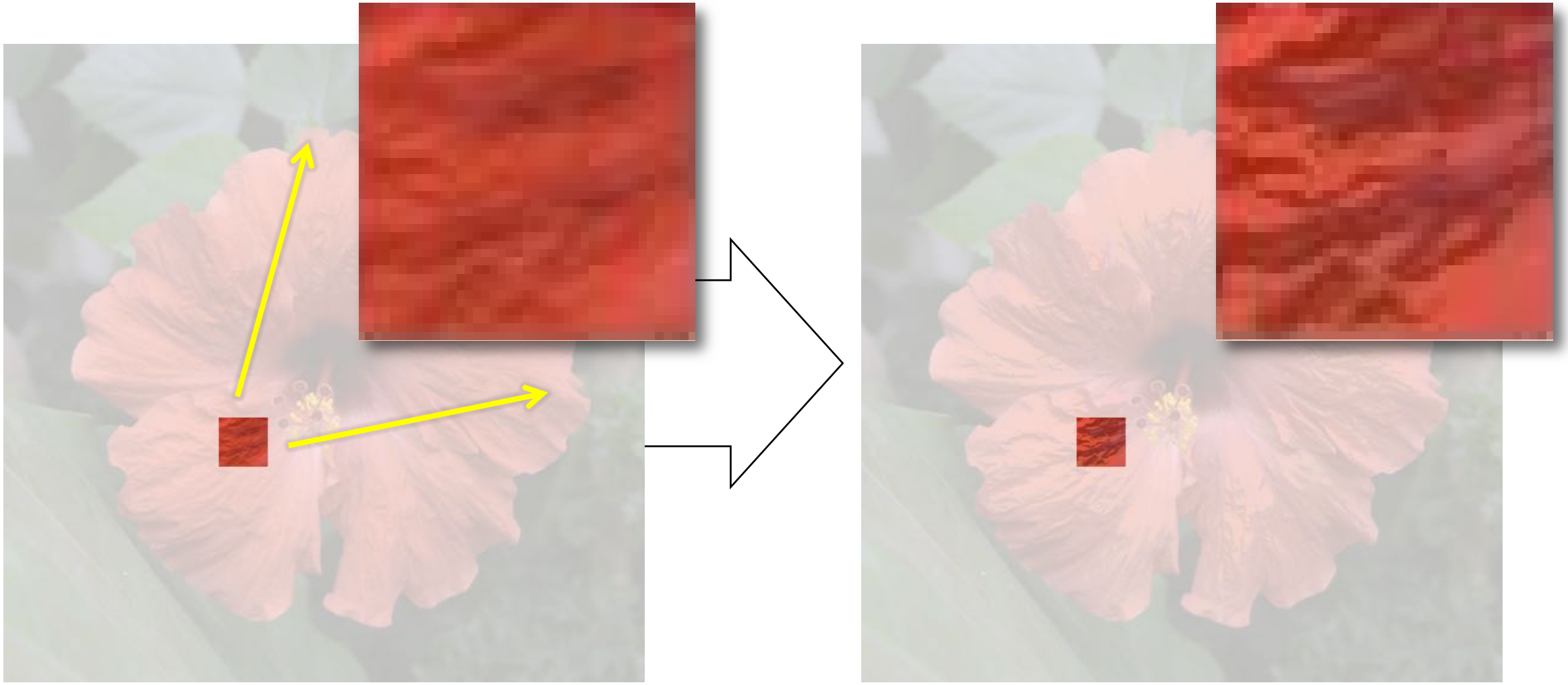
Example: Local Contrast Increase

input

Example: Local Contrast Increase

input

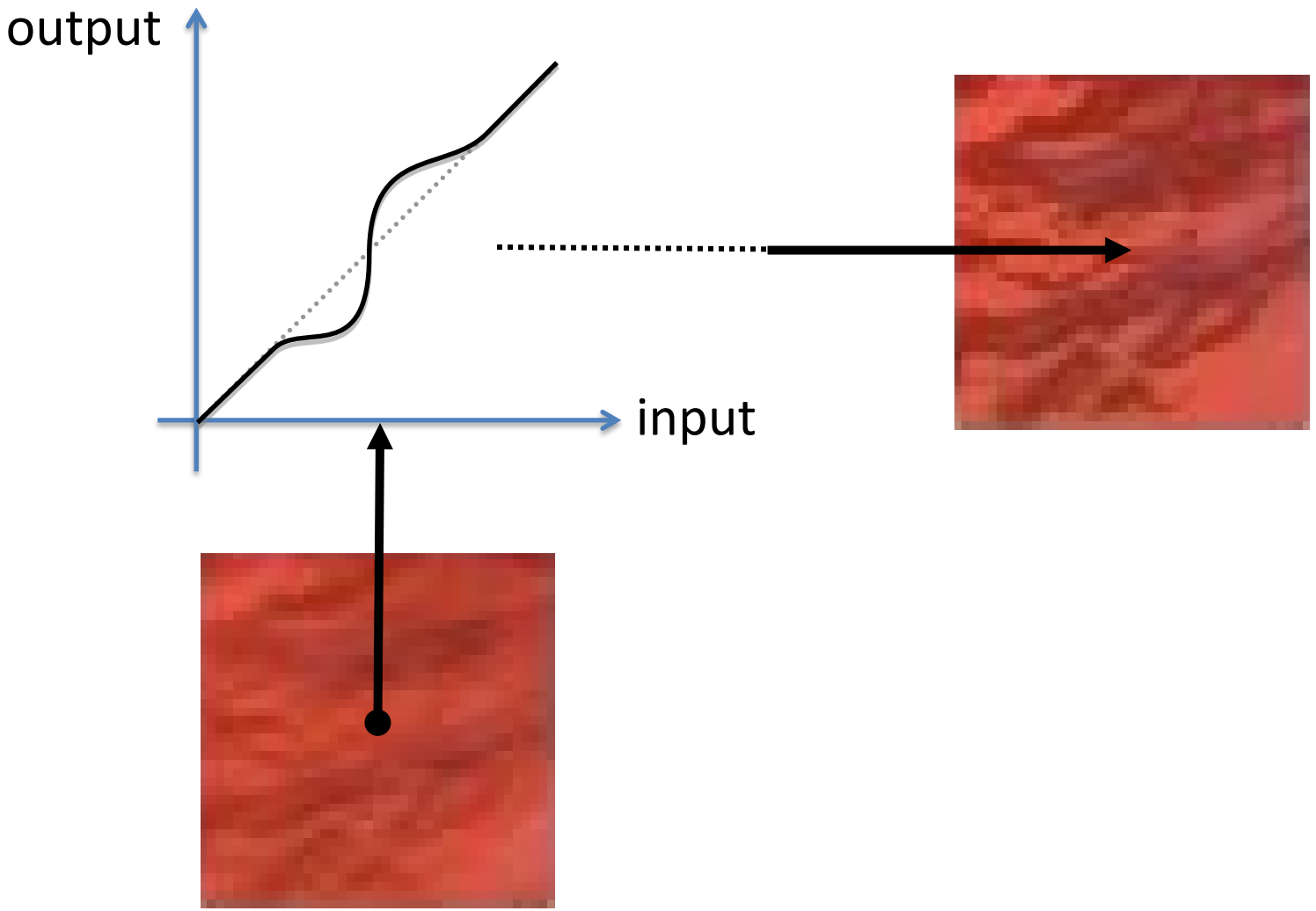
Example: Local Contrast Increase



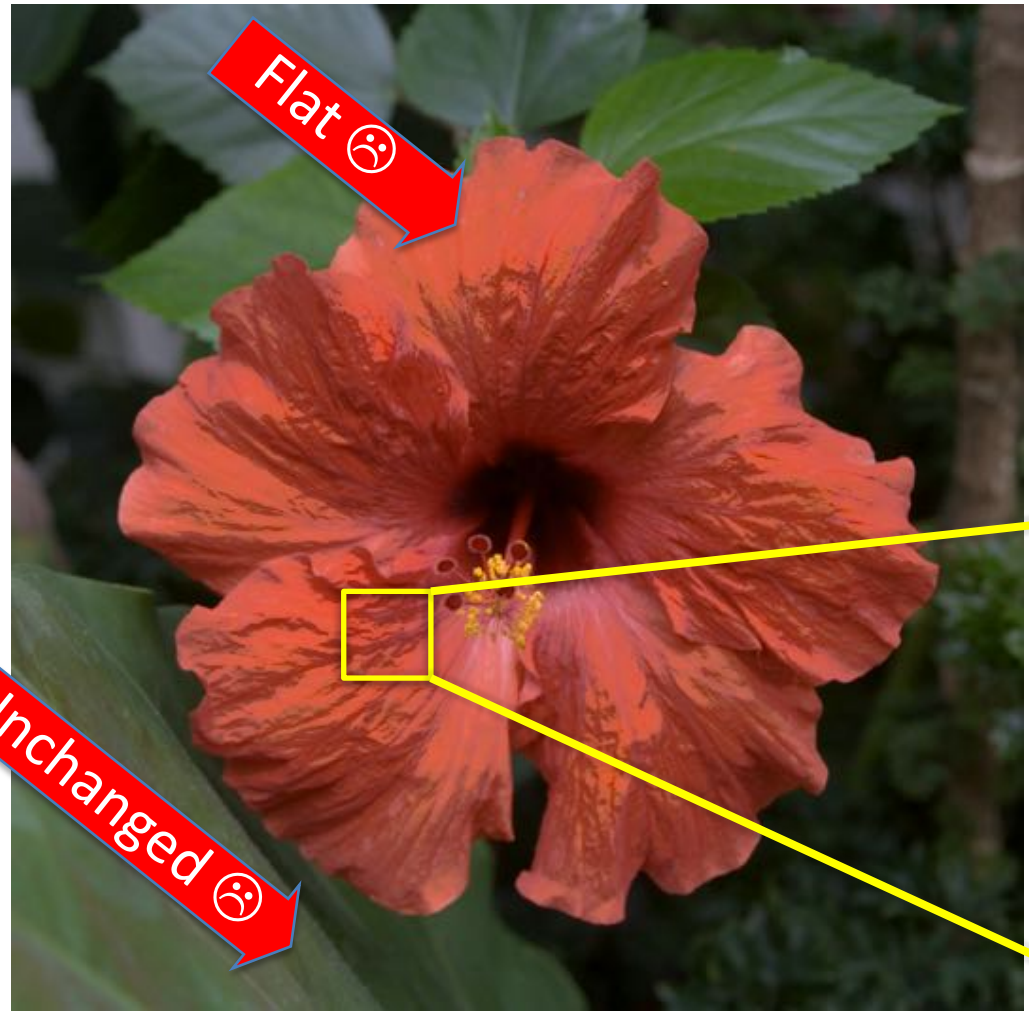
input

output with local contrast increased

Simple Local S-shaped Curve



Only Local Result Matters

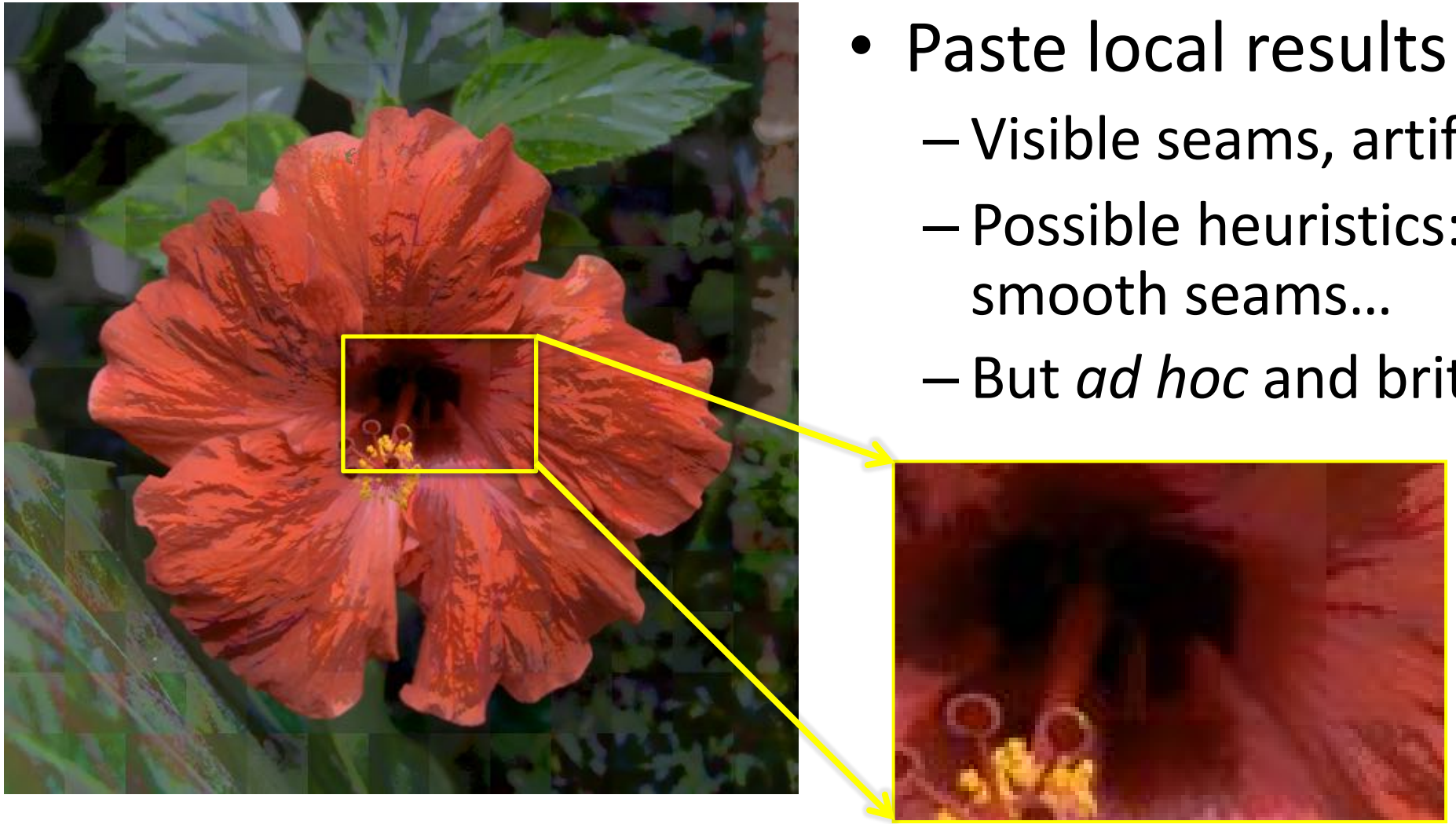


output with local contrast increased

- Artifacts appear elsewhere
- Not a problem, we use only local data

The processed image only needs to look good locally

Naïve Stitching



- Paste local results side by side?
  - Visible seams, artifacts…
  - Possible heuristics: vary patch size, smooth seams…
  - But *ad hoc* and brittle

Our Multi-scale Approach

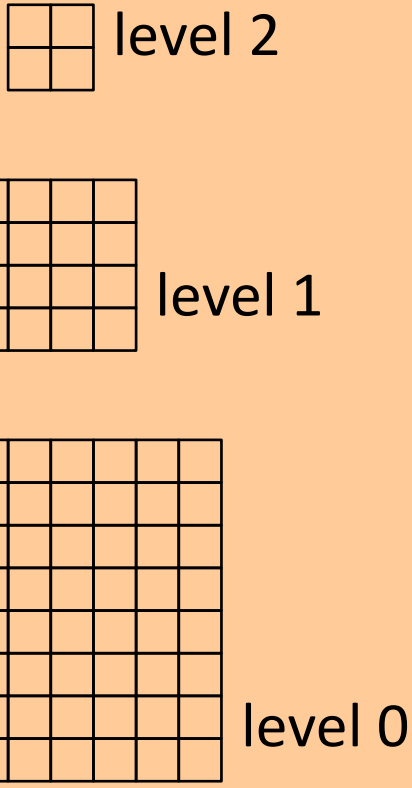
- We build the Laplacian pyramid of the output coefficient by coefficient

```
For each coefficient
   1. Generate "locally good" image
   2. Compute Laplacian pyramid of that image
   3. Copy coefficient to output pyramid
```

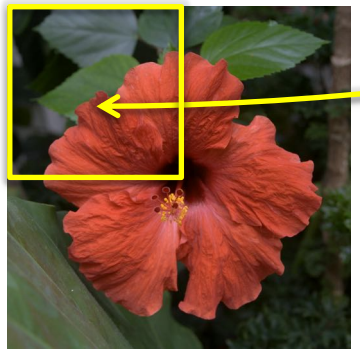
Illustration

input image

Output
Laplacian pyramid



Illustration



input image

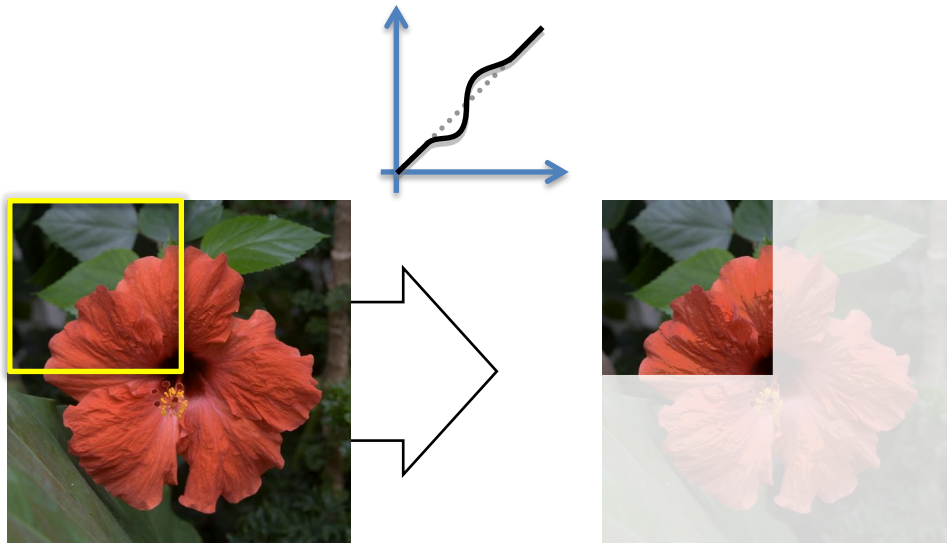
Output
Laplacian pyramid

level 2

level 1

level 0

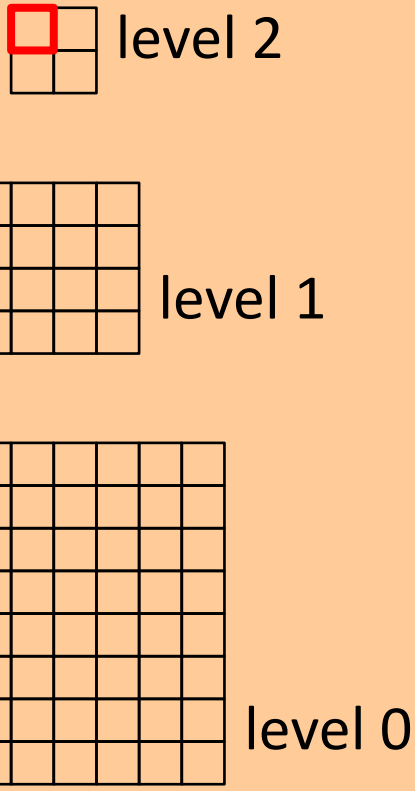
Illustration



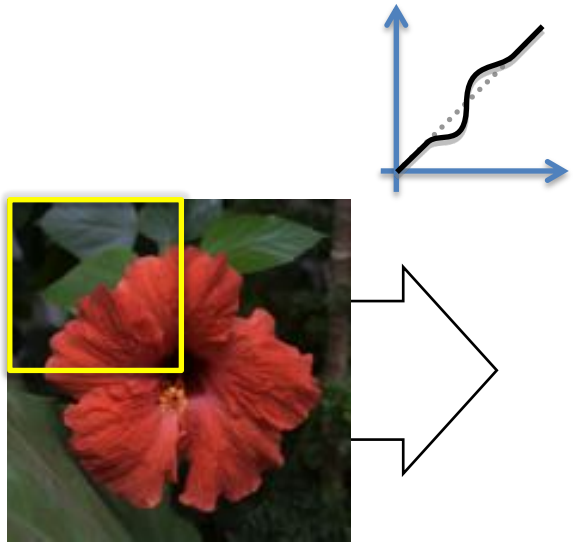
input image

“locally good” image

Output
Laplacian pyramid

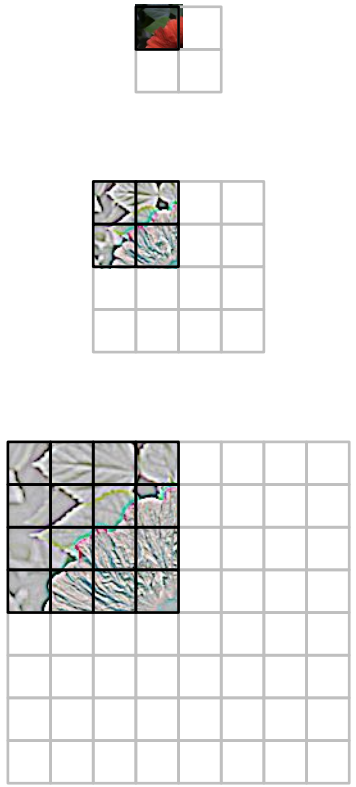


Illustration



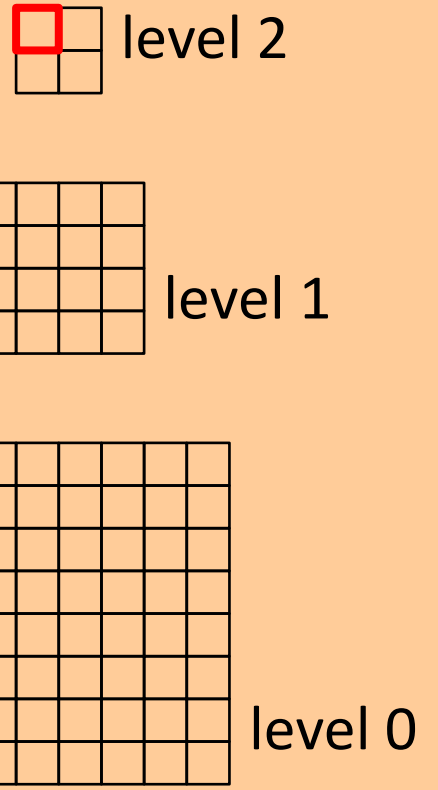
input image

“locally good” image

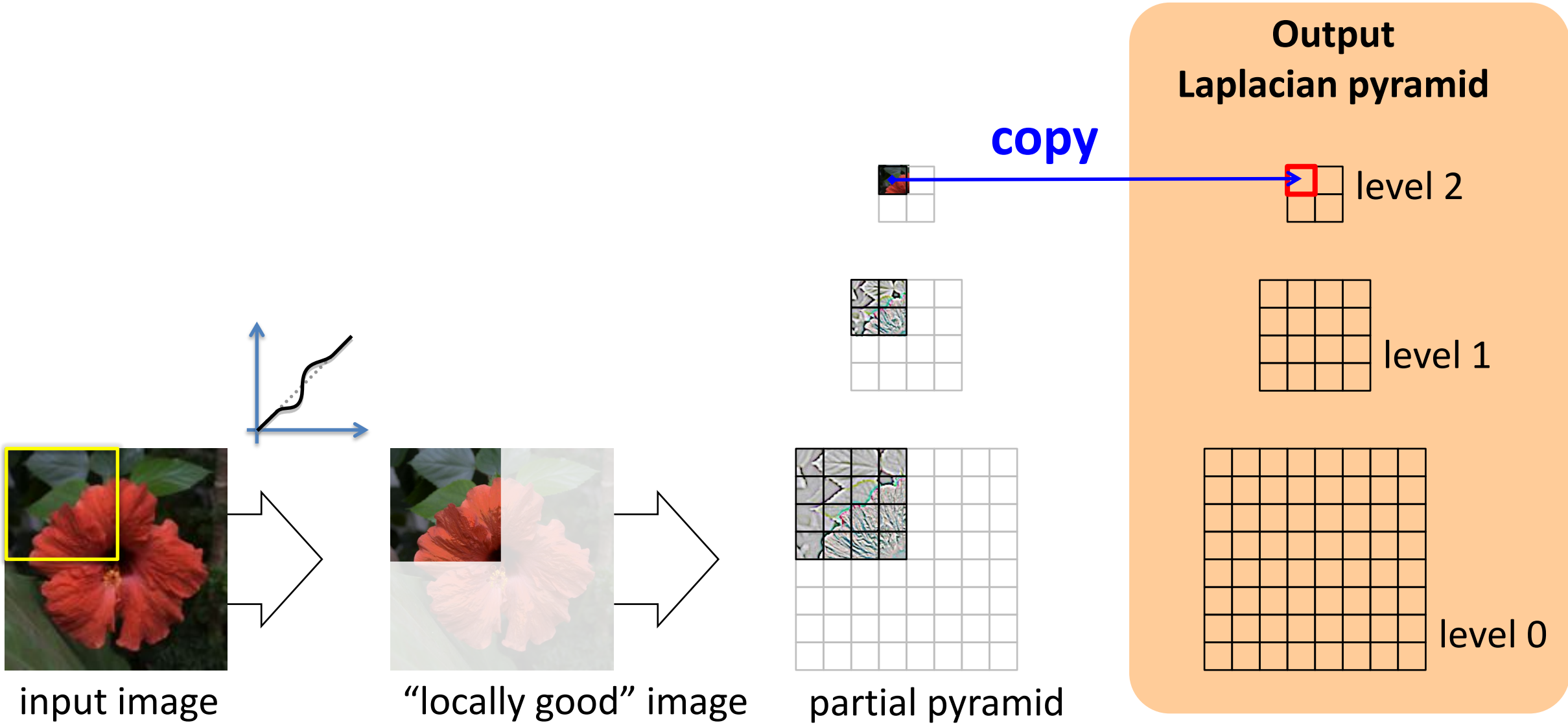


partial pyramid

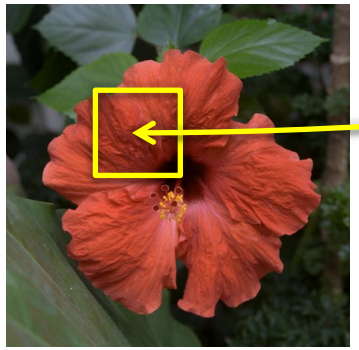
Output
Laplacian pyramid



Illustration



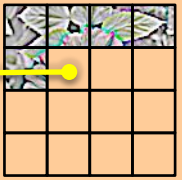
Illustration



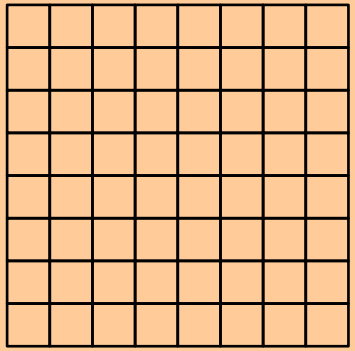
input image

Output
Laplacian pyramid

level 2

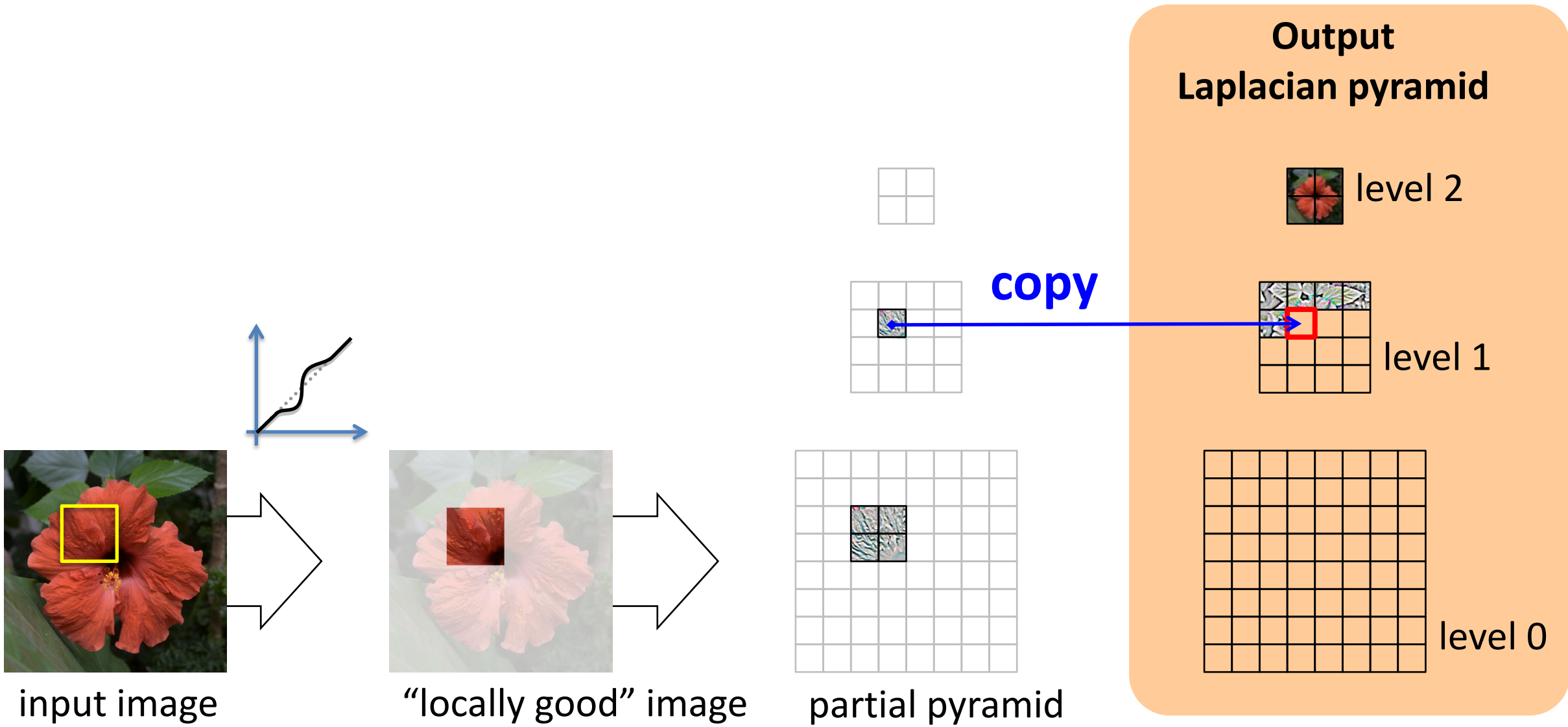


level 1

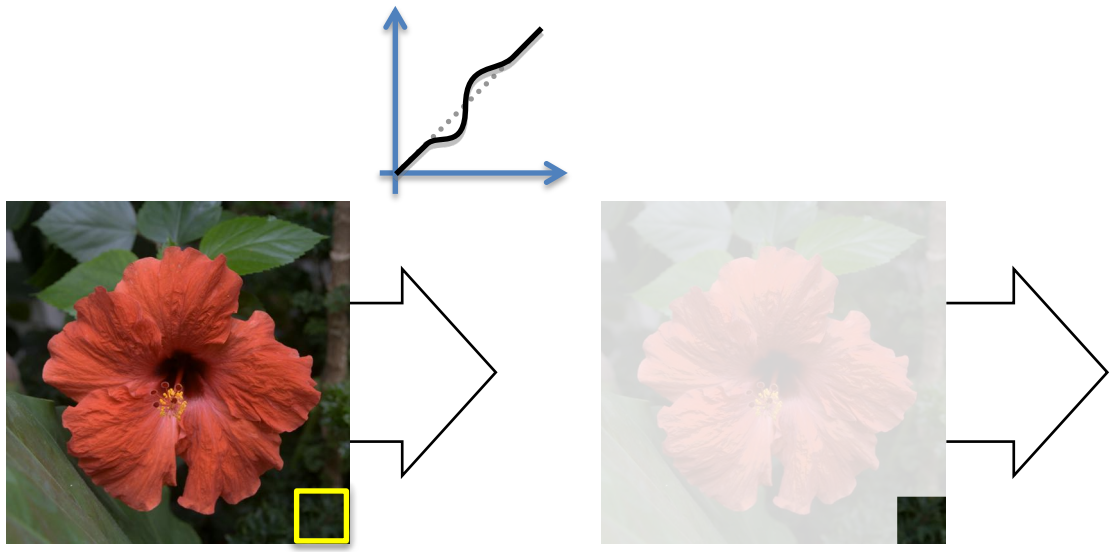


level 0

Illustration

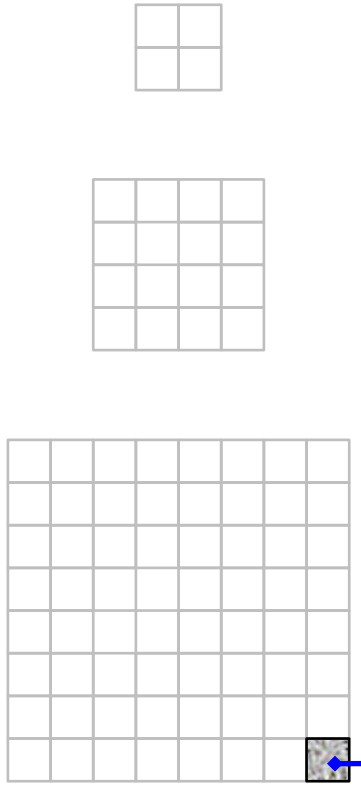


Illustration



input image

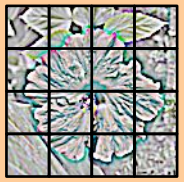
“locally good” image



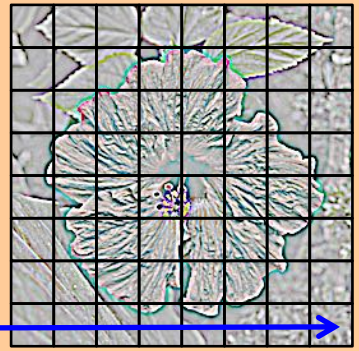
partial pyramid

Output
Laplacian pyramid

level 2



level 1

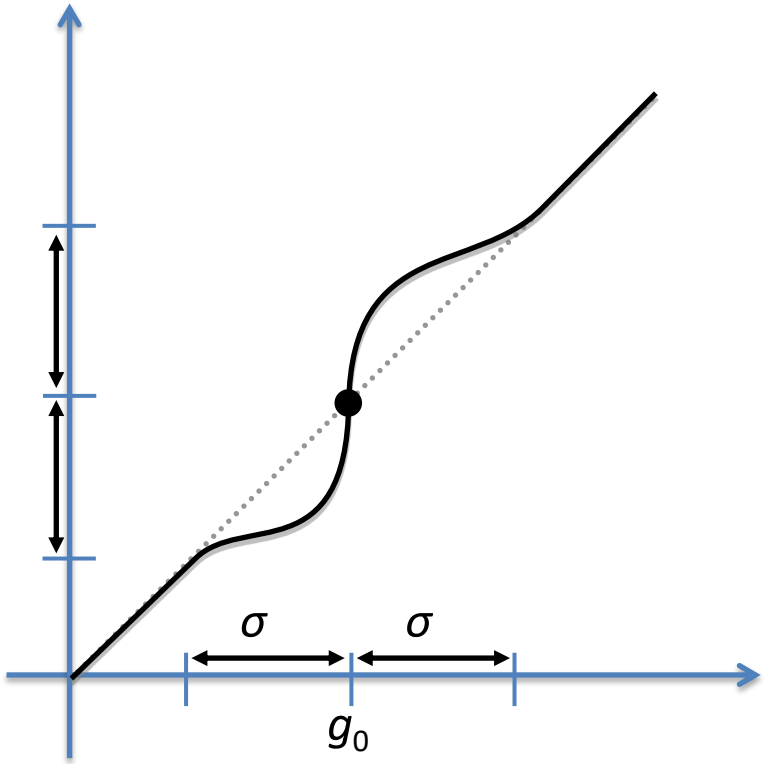


level 0

copy

Possible Nonlinearities

- Detail manipulation

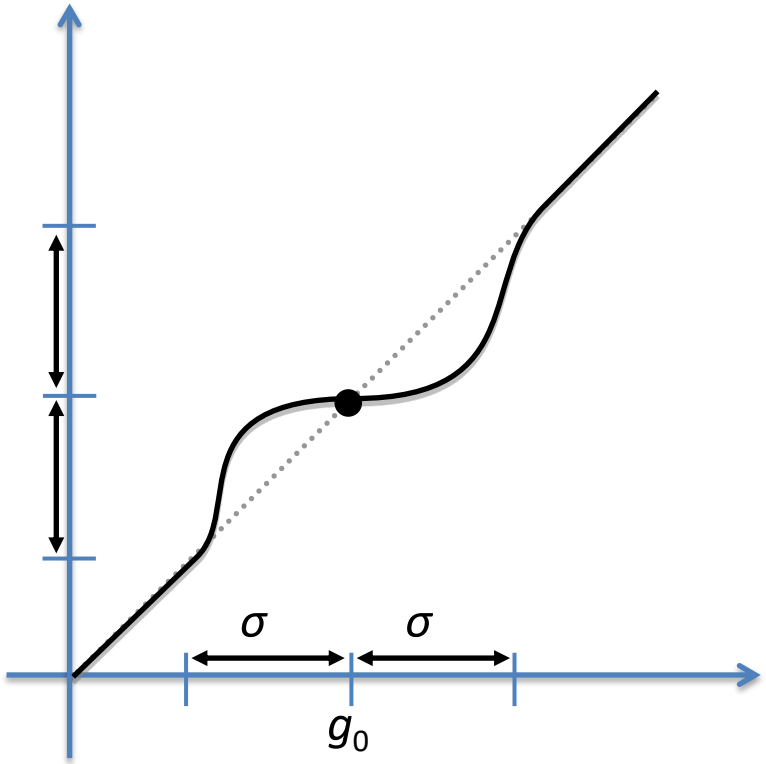


input image

details enhanced

Possible Nonlinearities

- Detail manipulation

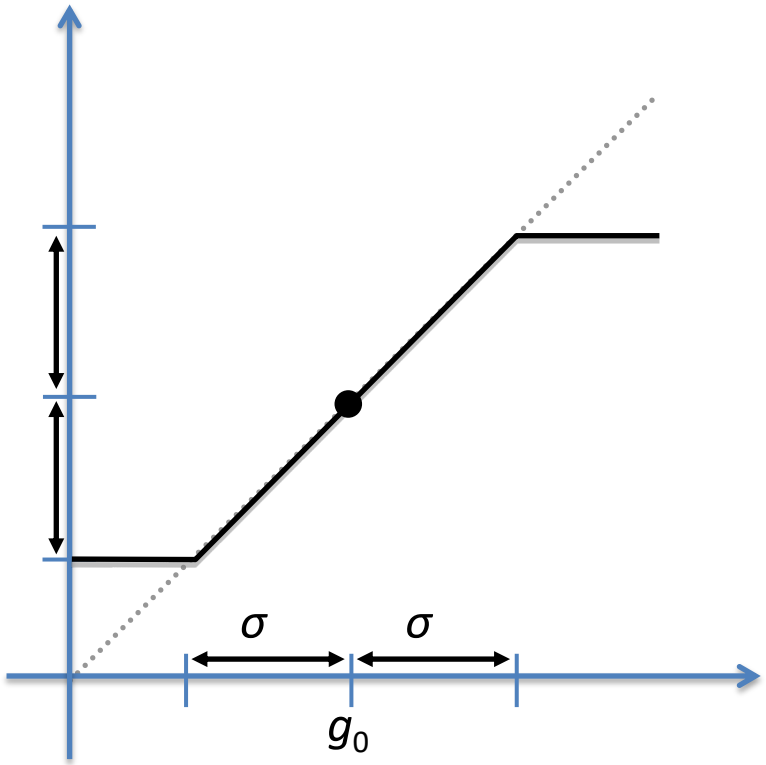


input image

details reduced

Possible Nonlinearities

- Dynamic range manipulation

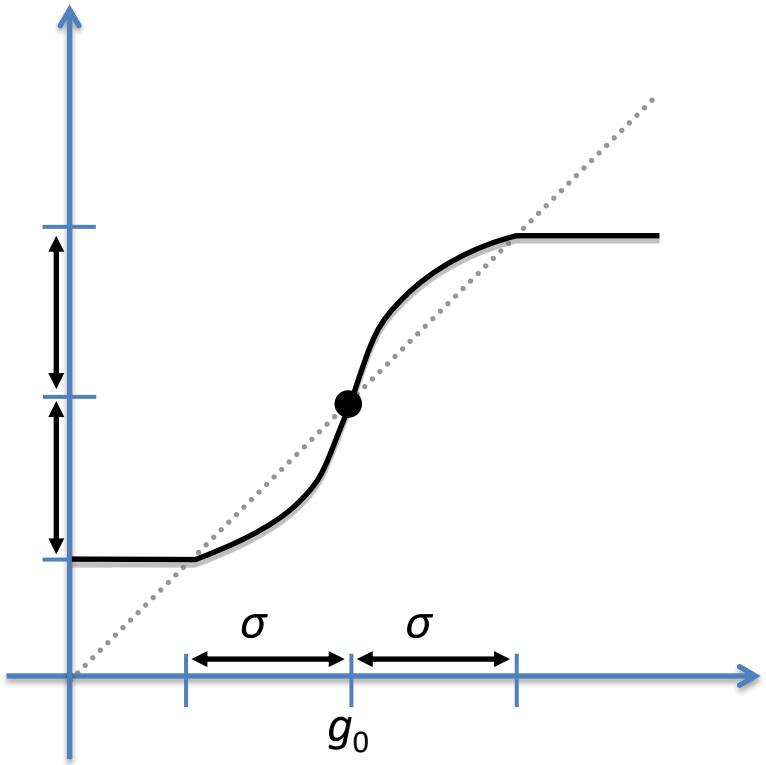


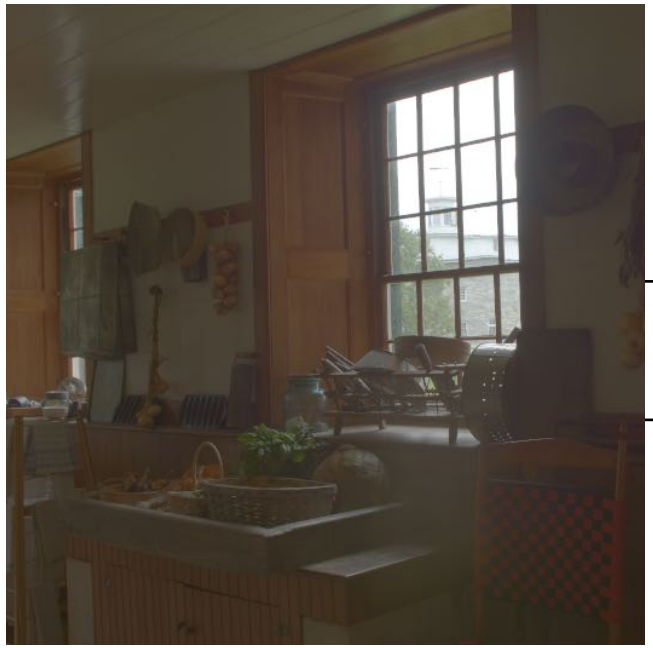
input image

tone mapped

Possible Nonlinearities

- Can be combined, e.g. tone map + boost details





input image

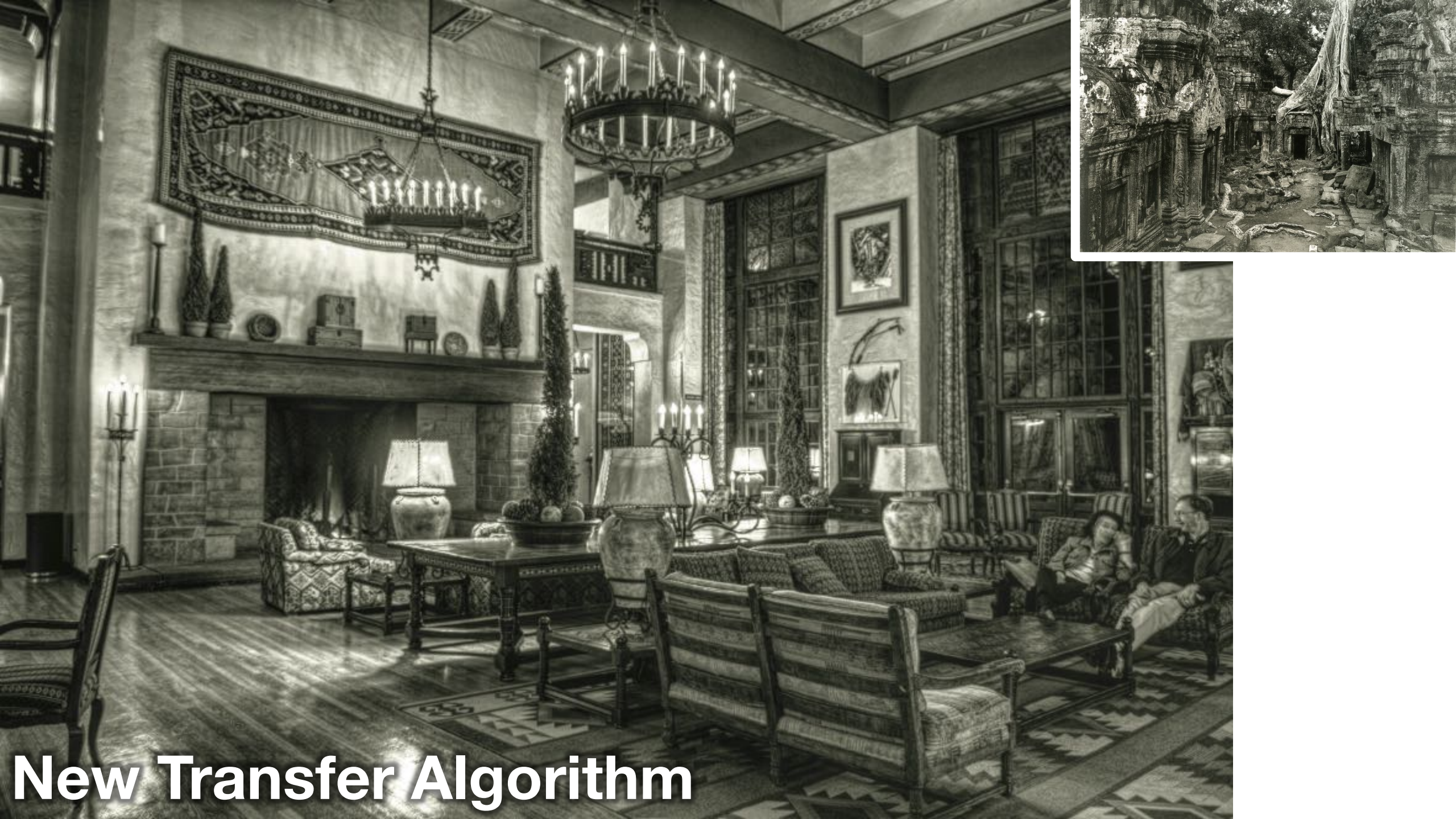
tone mapped +
details enhanced

Photoshop Demo

Back to Photo Style Transfer

Transactions on Graphics 2014

Old Transfer Algorithm



New Transfer Algorithm

Close-up

old algorithm

new algorithm

a lot more details

higher quality

Takeaway message #1

Photo style transfer works.

Takeaway message #2

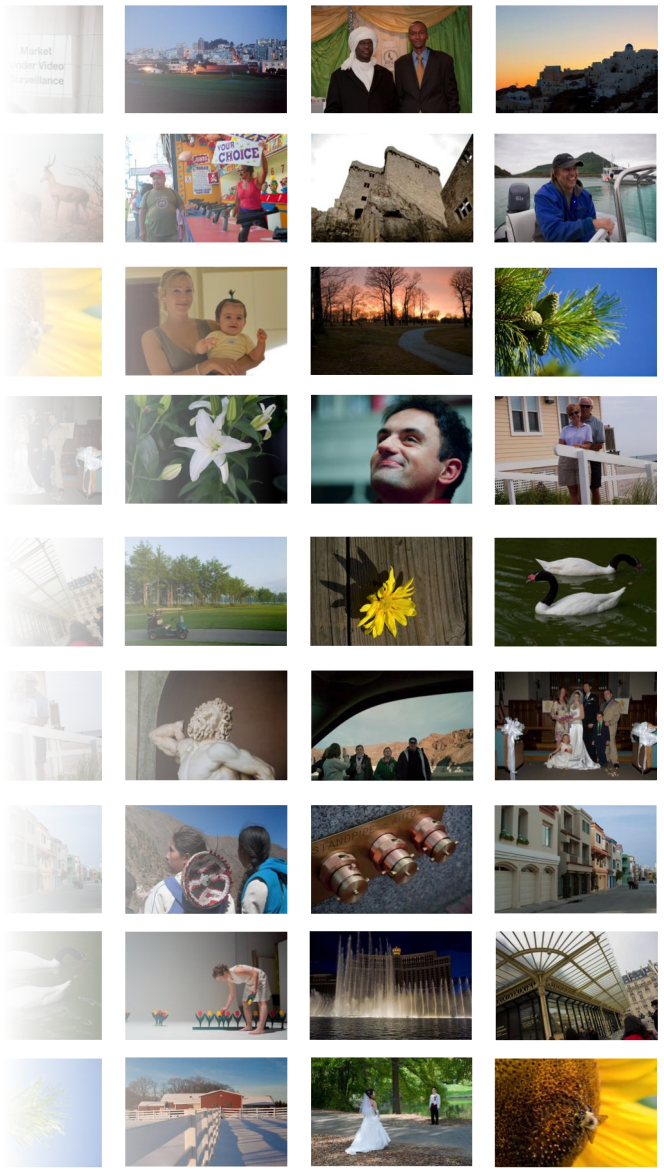
Low-level aspects matter.

Learning from a Dataset

CVPR 2011

5000 reference photos
adjusted by a pro

input



5000 reference photos
adjusted by a pro

input

5000 reference photos
adjusted by a pro

our result

Result

Result

Product Impact

- Transferred into Photoshop CS6
  as “Auto” in Brightness & Contrast, Levels, Curves, and Auto Tone.

- Can also learn from users on the fly,
  transferred as Auto Smart Tone into Photoshop Elements 12.

Advantages of Gaussian Processes

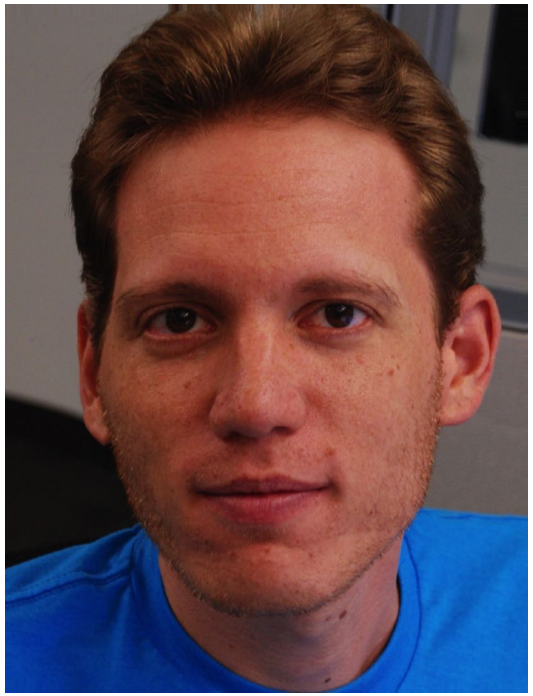
- Can learn from a few hundred examples “only”
  - Hours of work by a professional → quality training data
- Revert to the mean when “unsure”
  - Does not ruin the image
- Easy to debug by looking at how training data are used

Back to Transferring from an Example

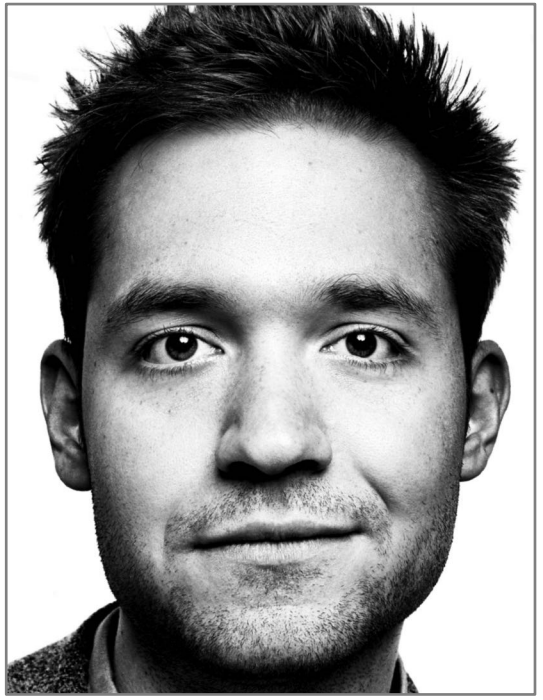
Previous Techniques Fail on Portraits

- Two-scale model not sufficient for skin texture
- Image-level statistics too coarse for portraits

Make this portrait



look like this one



A Local and Multi-scale Model

Input

Target

A Local and Multi-scale Model

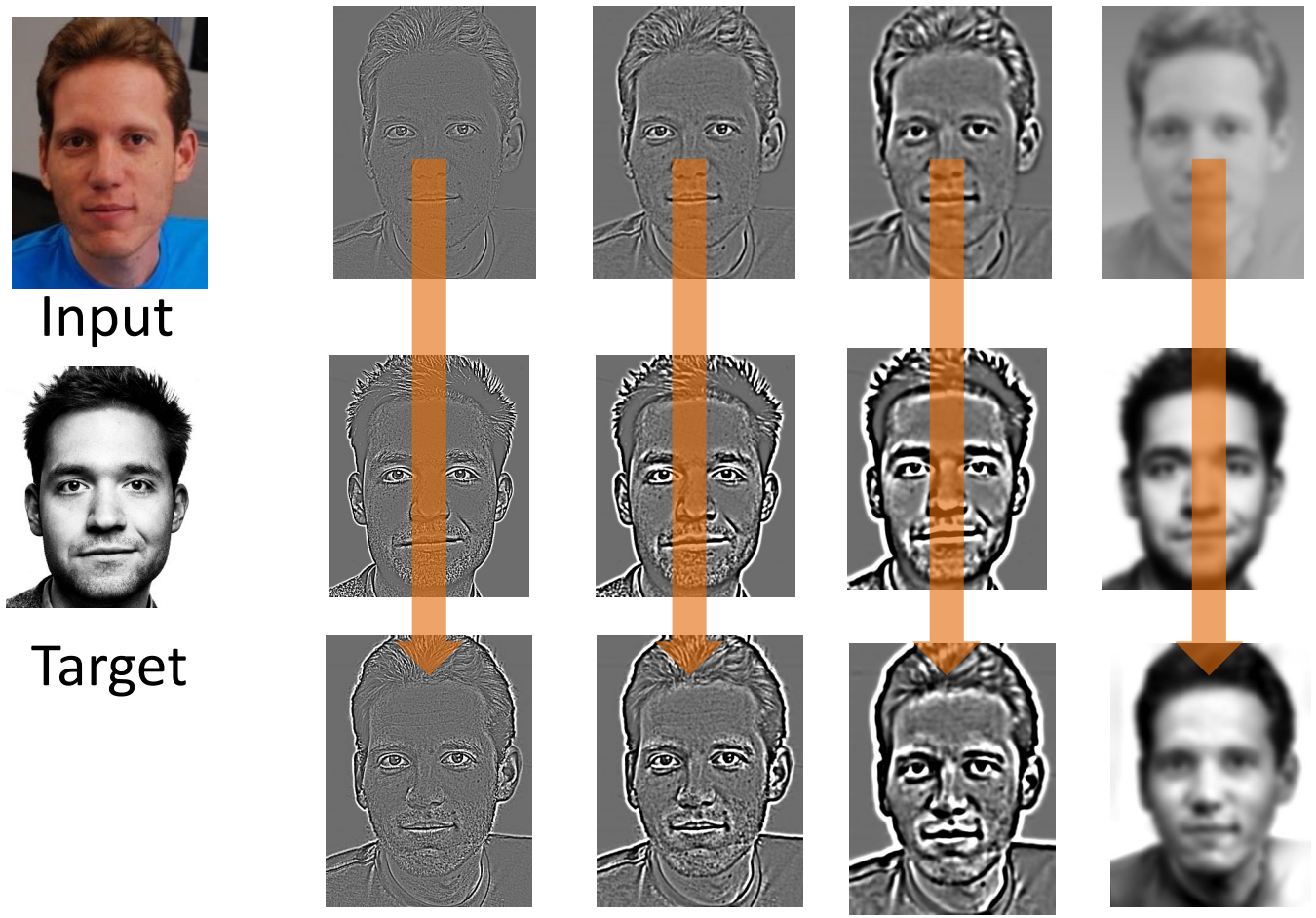
1. Construct Laplacian stacks for the input and the example

Input

Target

A Local and Multi-scale Model

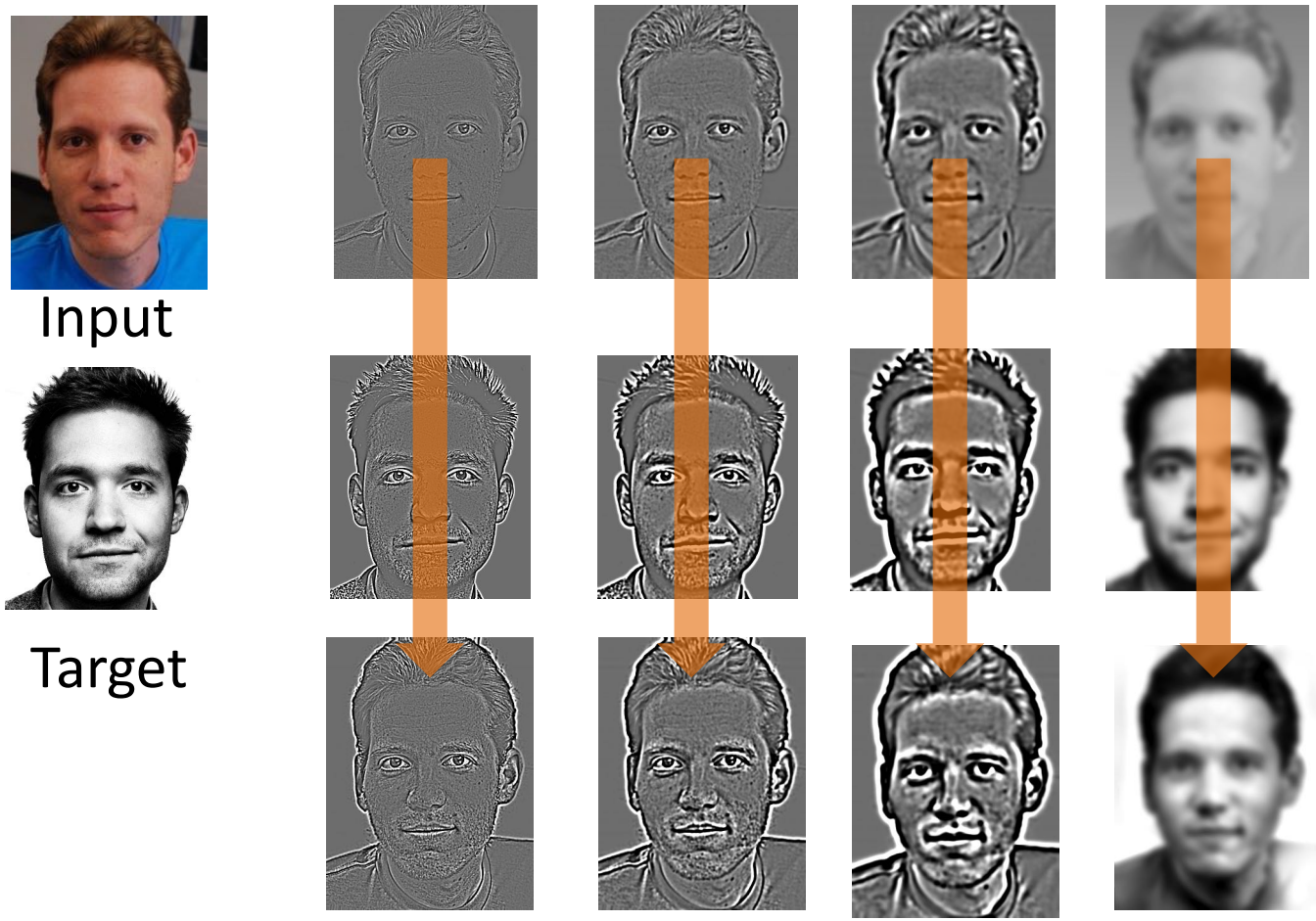
1. Construct Laplacian stacks for the input and the example



2. Local match at each scale

A Local and Multi-scale Model

1. Construct Laplacian stacks for the input and the example



Input

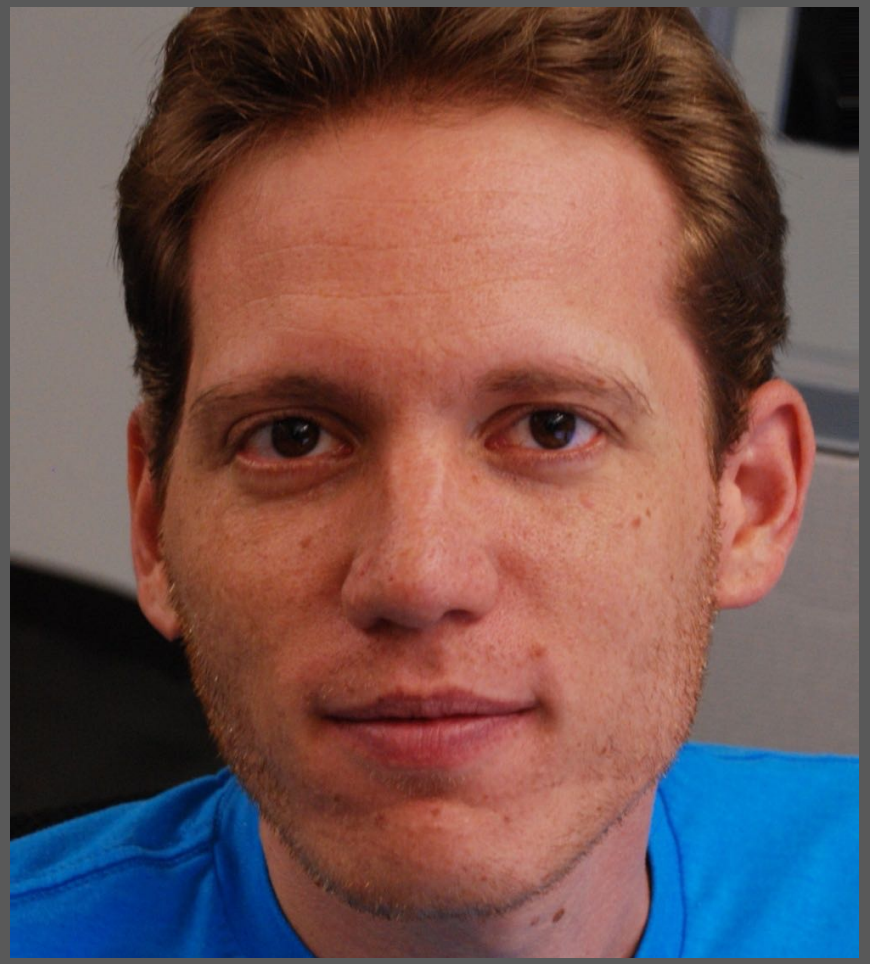
Target

2. Local match at each scale

Output

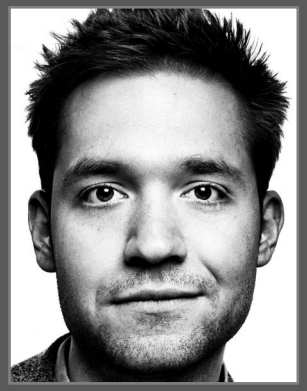
3. Collapse the matched stacks to create the output of this step

Result (Plato)



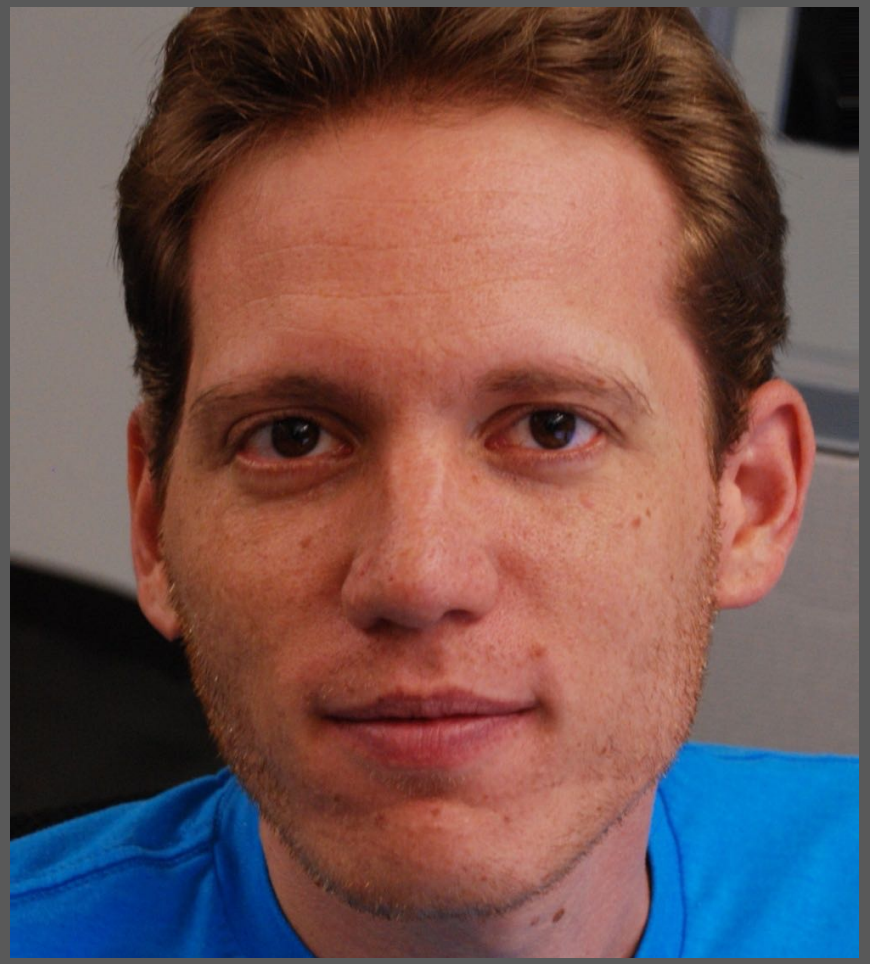
input

output

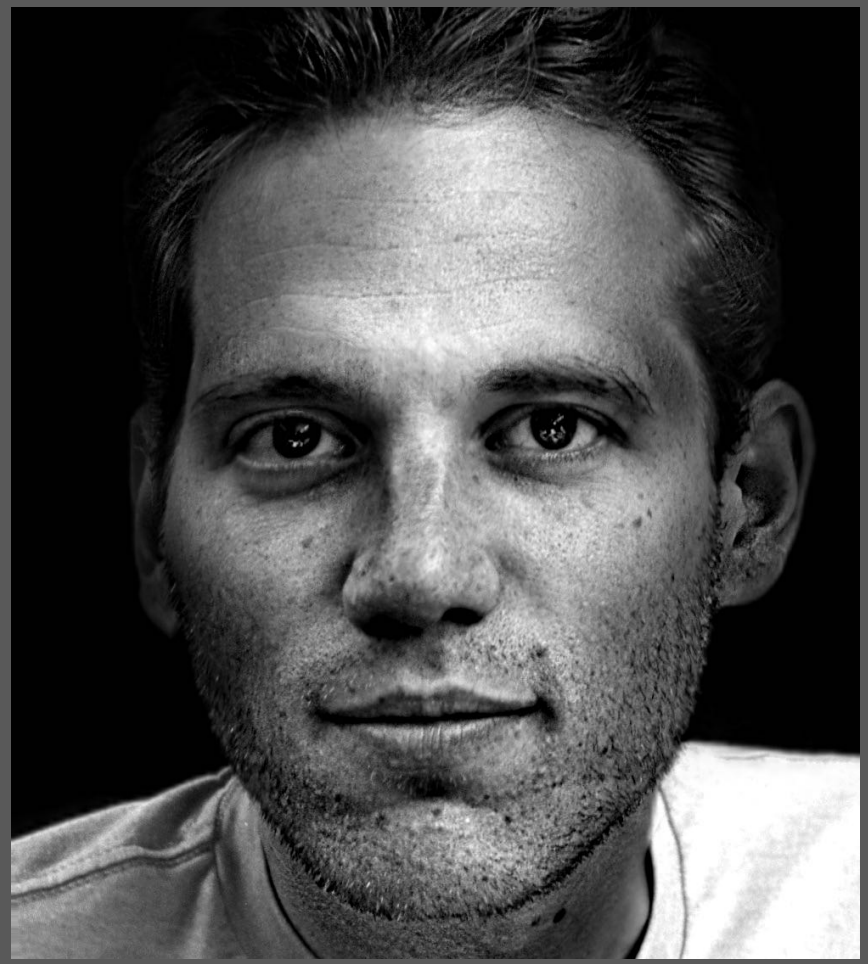


example

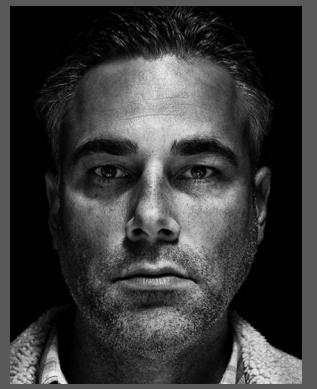
Result (Kelly Castro)



input

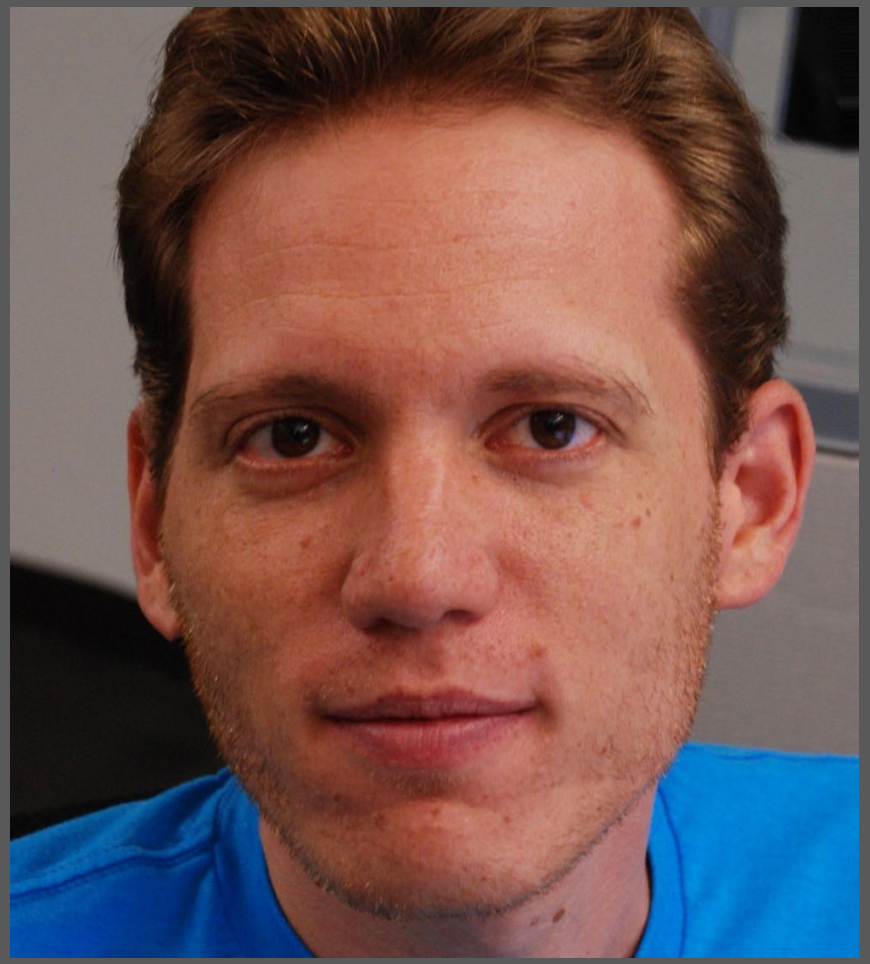


output



example

Result (Martin Schoeller)



input

output

example

Input sequence with extreme facial expressions

Our style transfer result using the example in the gray box

Transferring Other Properties

Time-of-Day Hallucination

SIGGRAPH Asia 2013

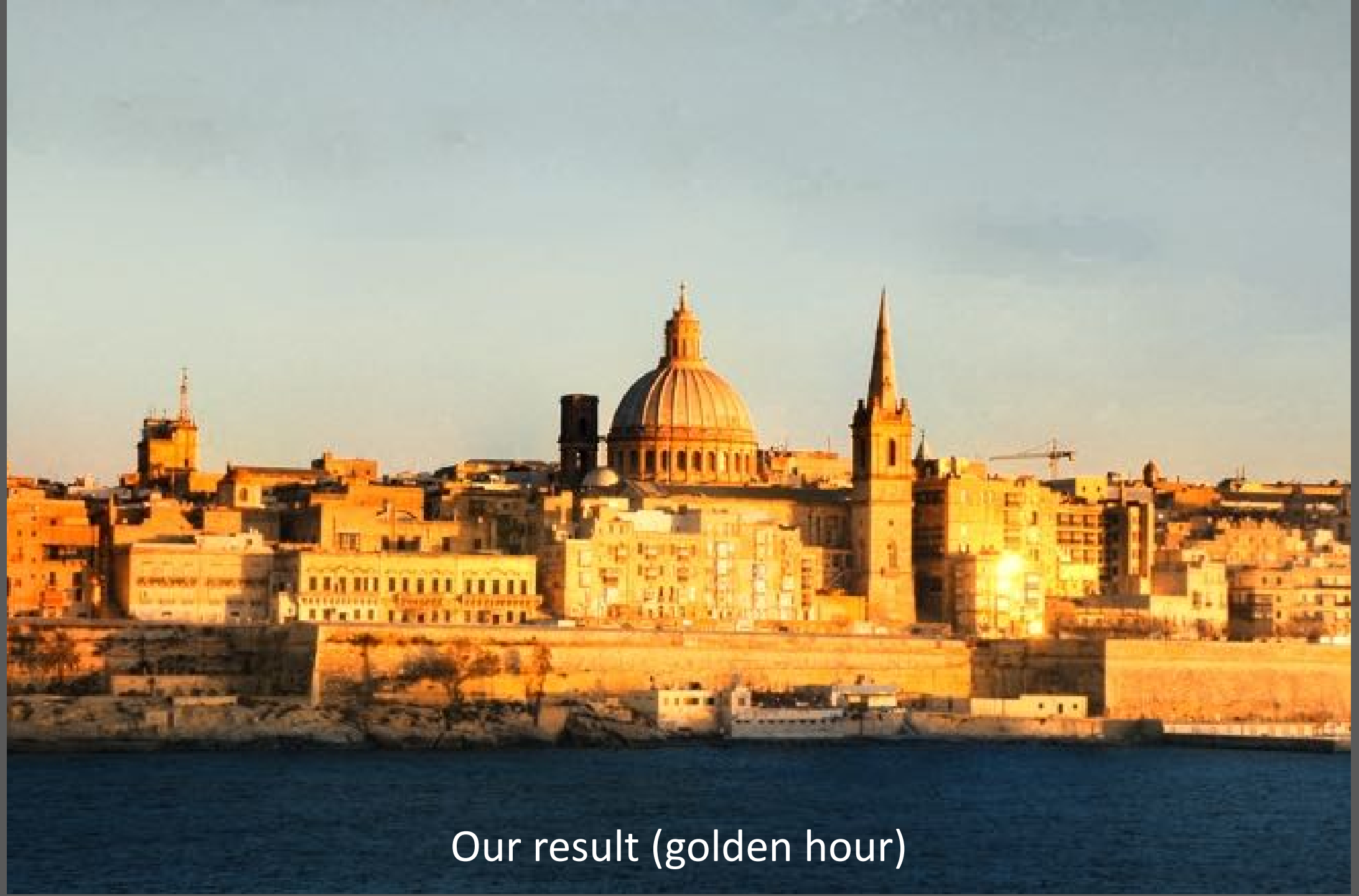
Time-lapse Videos as Examples

- A database of 400+ time-lapse videos
- Shows how scenes change during the day

Input photo

Our result

Input



Our result (golden hour)

Deep Photo Style Transfer

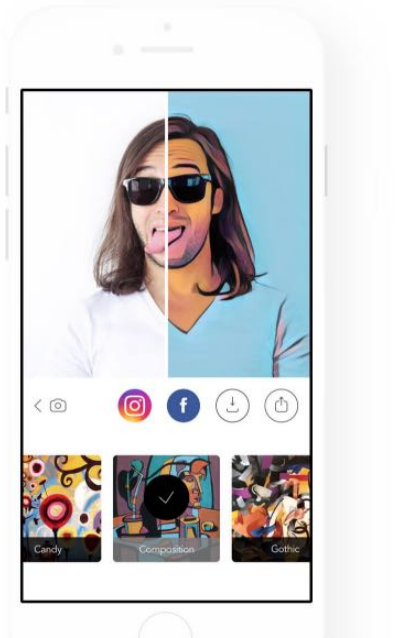
A First Step Toward a Unified Approach

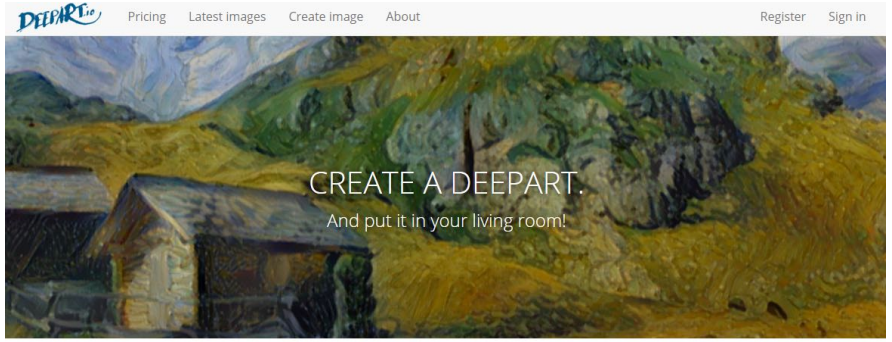
To appear at CVPR 2017

Motivation

- A Neural Algorithm of Artistic Style
  [Gatys et al. 2016]

- Several apps and websites
  - Prisma
  - DeepArt.io





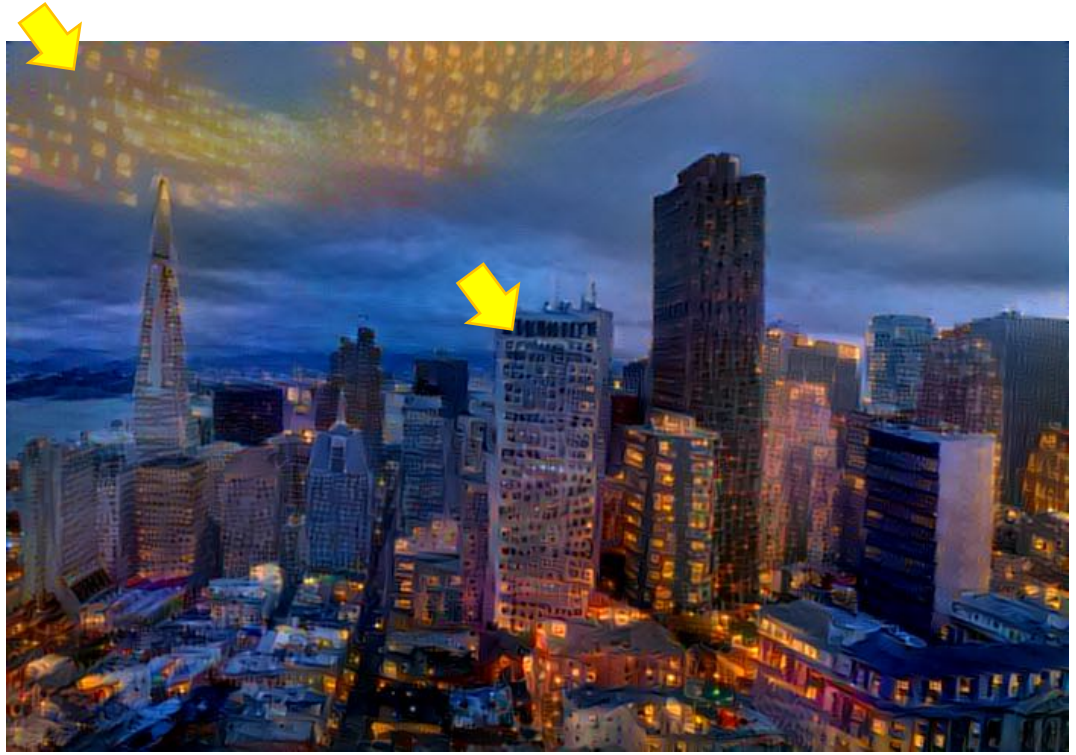
The neural style algorithm works well for paintings.

What about photos?

So we tried…

Model

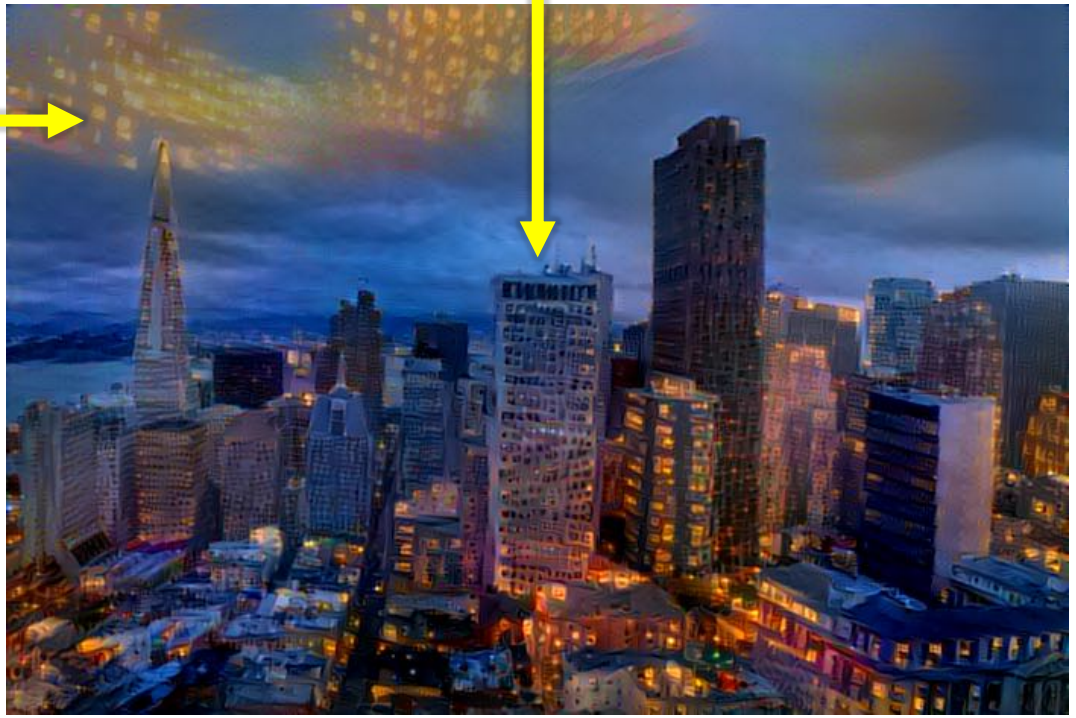
Input



Output of Neural Style Transfer

Two Problems to Solve

- Undesired distortion
  - Result still looks like a painting, not a photo
- Semantic mismatch
  - E.g., ground texture can appear on the sky



Background on Neural Style Transfer

1. Decompose the input and model images using a neural network

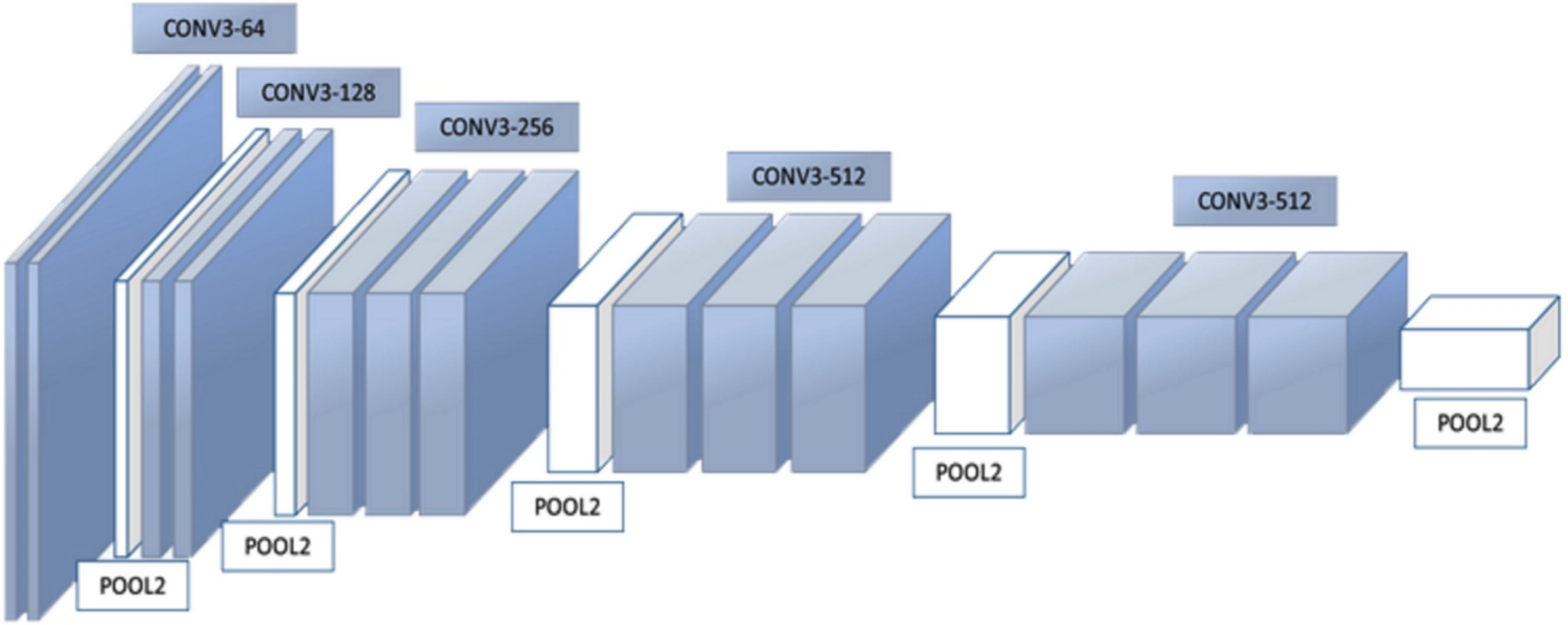


Diagram from http://www.justindchien.com/

Background on Neural Style Transfer

1. Decompose the input and model images using a neural network

2. Impose some of the statistics of the model image

Background on Neural Style Transfer

1. Decompose the input and model images using a neural network

2. Impose some of the statistics of the model image

3. Reconstruct the output image

Background on Neural Style Transfer

1. Decompose the input photo using a neural network

2. Impose some of the statistics of the model image

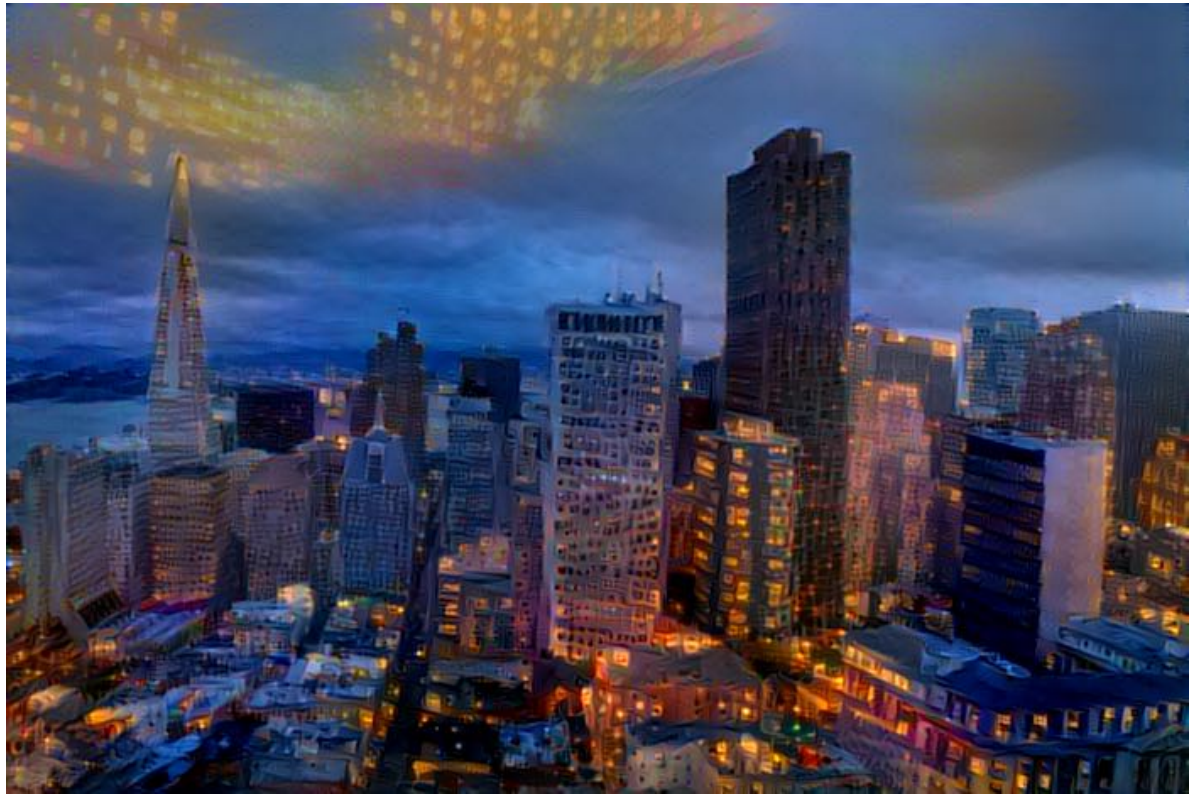
3. Reconstruct the output image

Preventing Distortion

- We tried many options
  - Constraining the gradients
  - Multi-scale constraints akin to Portrait Style Transfer
  - Band-limiting the transformation
- All helped but none was 100% successful

- What worked was forcing the color transformation to be locally affine [Levin et al. 2006]

$$\begin{pmatrix} r_{out} \\ g_{out} \\ b_{out} \end{pmatrix} = A_{3\times3} \begin{pmatrix} r_{in} \\ g_{in} \\ b_{in} \end{pmatrix} + B_{3\times1}$$

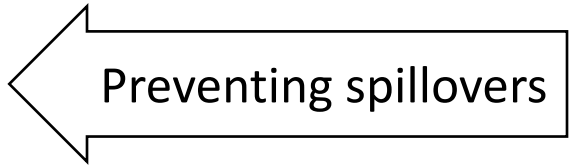


Output of Neural Style Transfer

Neural Style Transfer + Locally Affine Transformation

Background on Neural Style Transfer

1. Decompose the input photo using a neural network
2. Impose some of the statistics of the model image
3. Reconstruct the output image



Original Algorithm and Its Limitation

- Transfer the Gram matrix of the model to the input

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

ℓ layer index in the network

i, j map index in the layer

k pixel index in the map

Original Algorithm and Its Limitation

- Transfer the Gram matrix of the model to the input

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

G does not depend on the pixel position

ℓ layer index in the network

i, j map index in the layer

k pixel index in the map

Original Algorithm and Its Limitation

- Transfer the Gram matrix of the model to the input

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

G does not depend on the pixel position

ℓ layer index in the network

i, j map index in the layer

k pixel index in the map

- Property: the Gram matrix defines the vectors up to an isometry
  - “the neural responses can move but their distribution remains the same”
  - E.g., the G matrix implicitly encodes the “quantity of sky” in a photo.

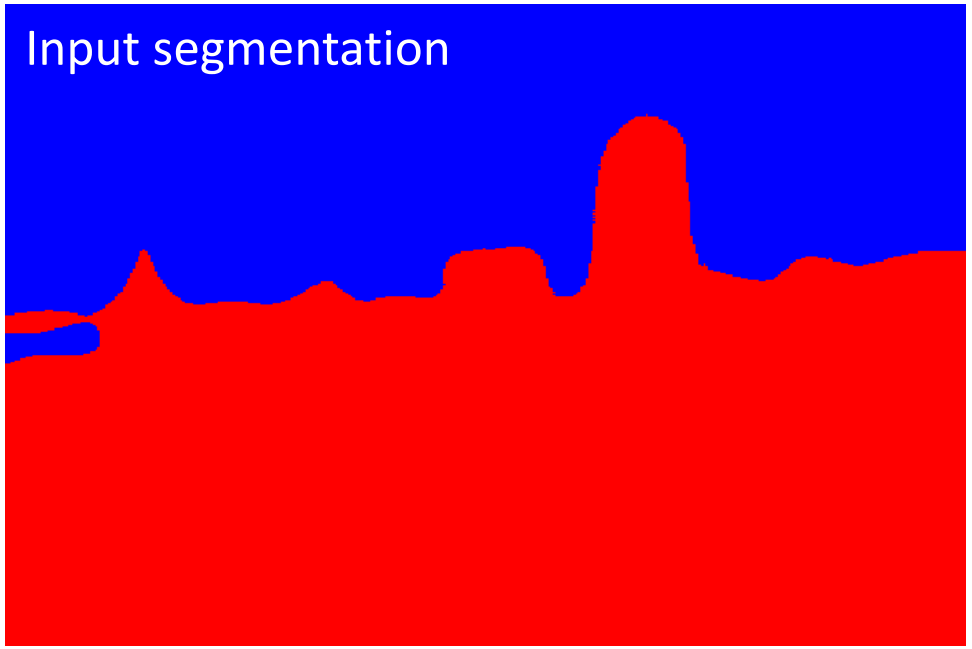
Different Horizon Lines

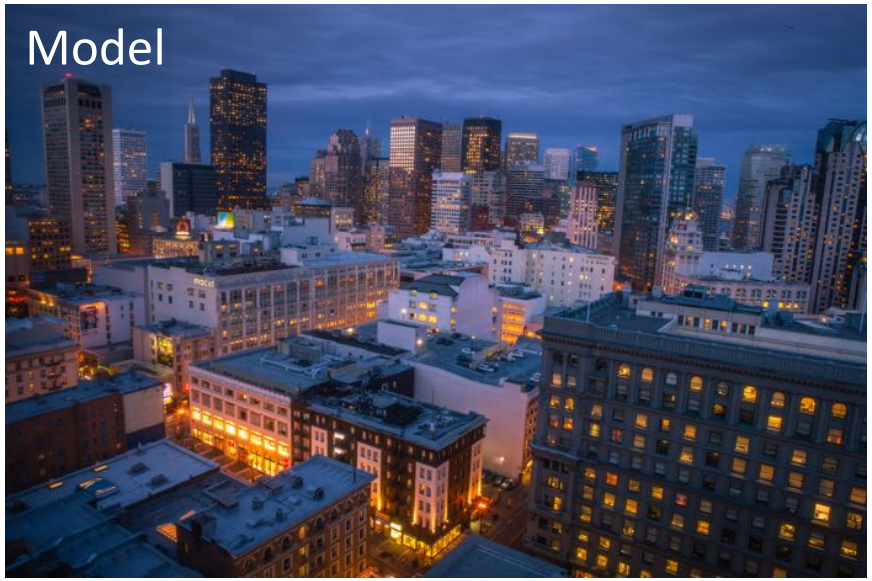
Our Strategy to Ensure Consistent Matching

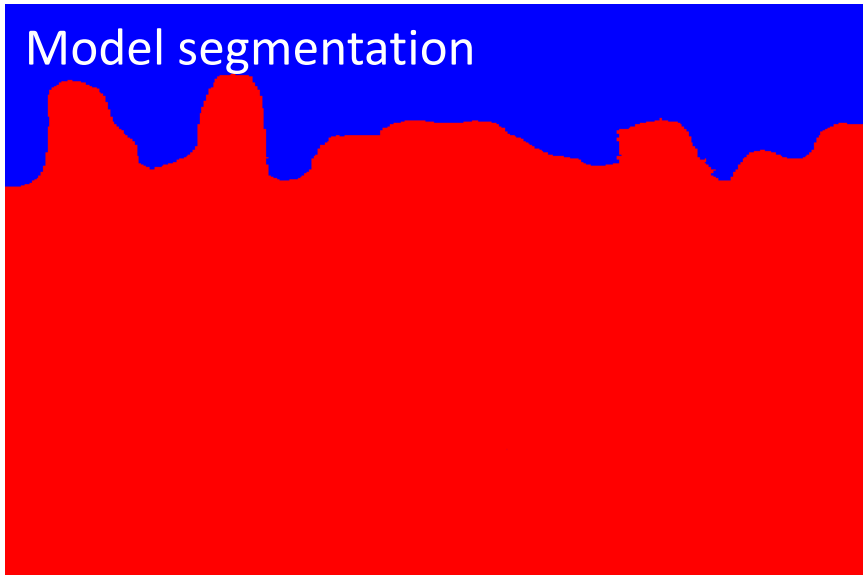
1. Run semantic segmentation on input and model [Chen et al. 2016]
   - Merge “visually equivalent” categories, e.g., sea and lake
   - Constrain the input labels to be a subset of the model labels

2. Transfer statistics within each category, e.g., sky to sky


Input


Input segmentation


Model


Model segmentation

Output of Neural Style Transfer

Model

Output of Neural Style Transfer + Locally Affine Transformation

Model

Model

Our result: Neural Style Transfer + Locally Affine Transformation + Semantic Matching

Model

Model

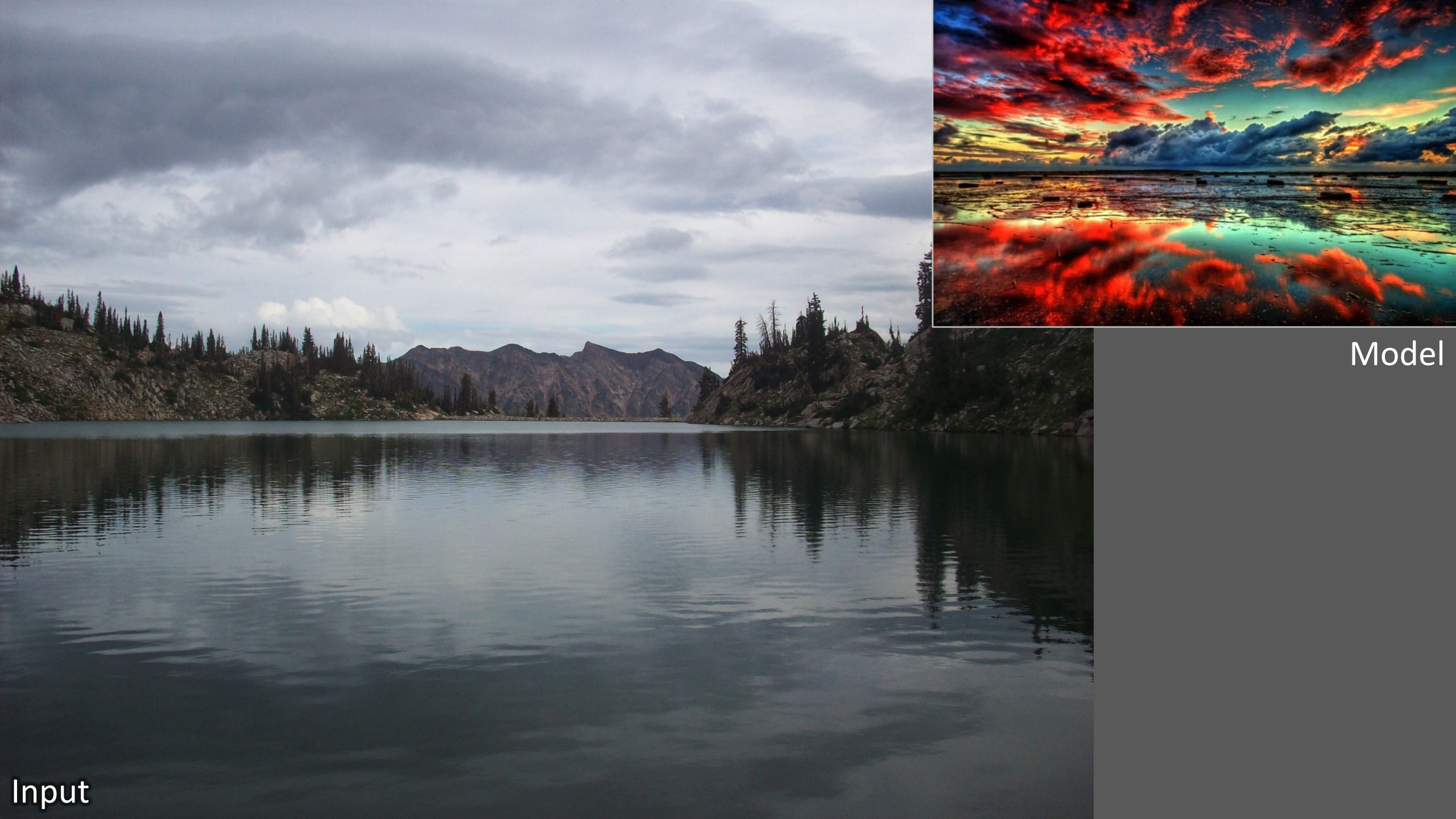
Our result: Neural Style Transfer + Locally Affine Transformation + Semantic Matching


Input


Model


Model
Neural Style Transfer


Our result
Model


Input
Model

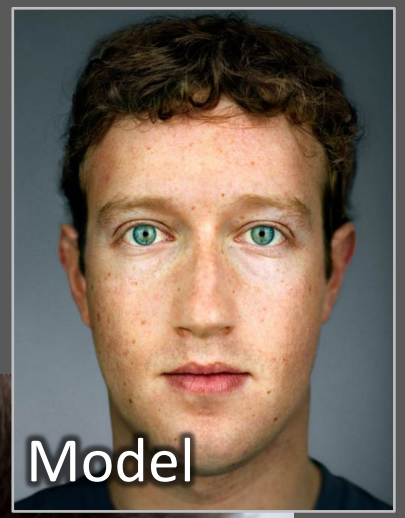

Neural Style Transfer
Model

Color Histogram Transfer

Model

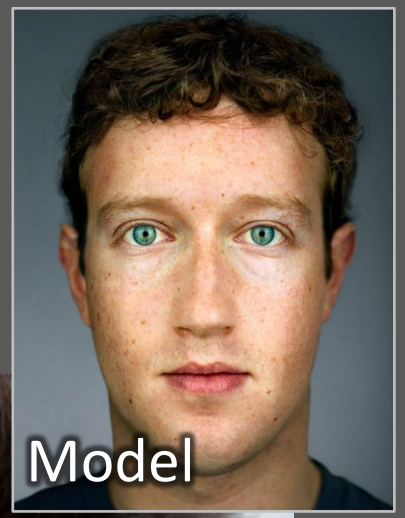
Model

Our result


Model


Input

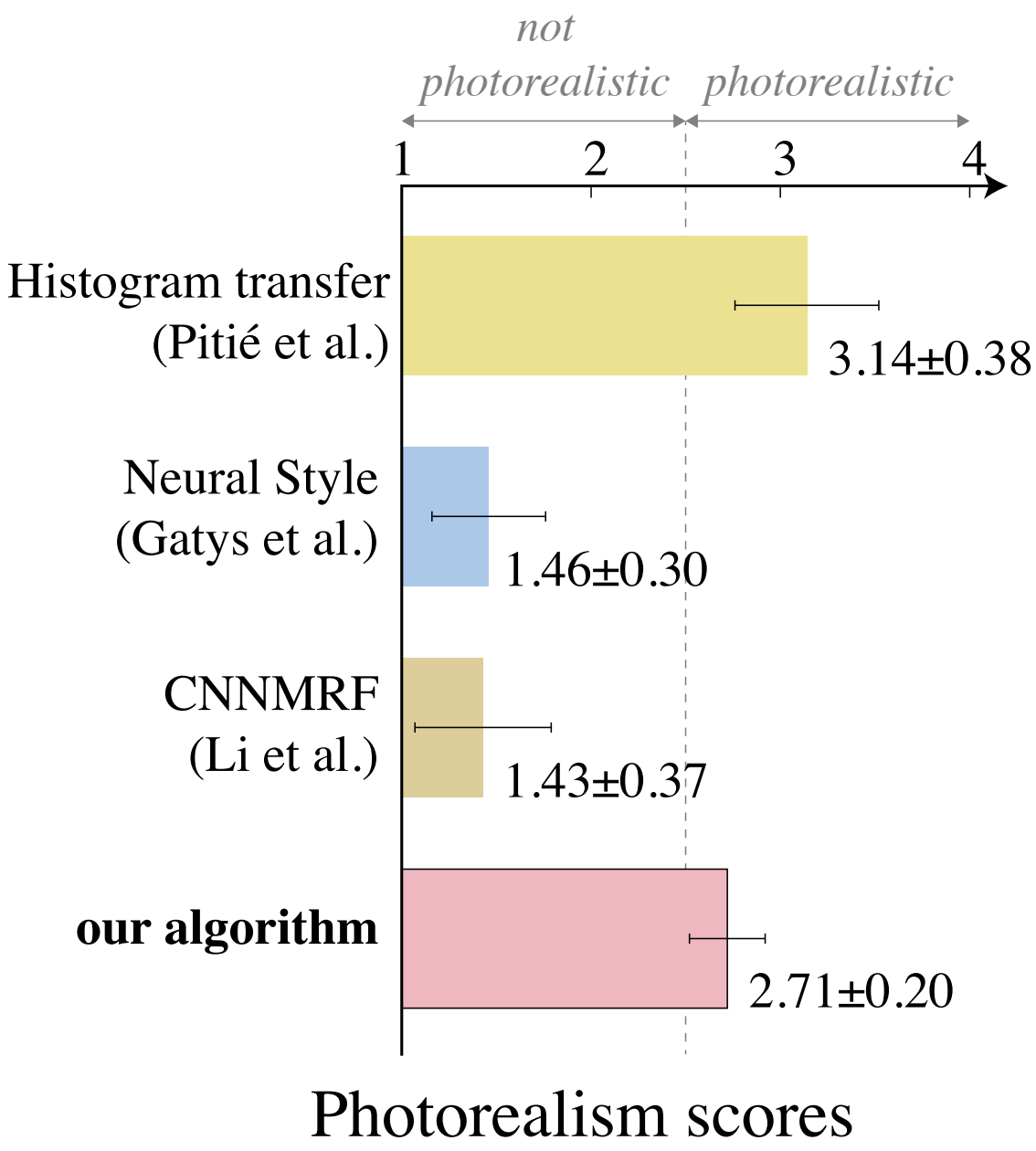

Our result

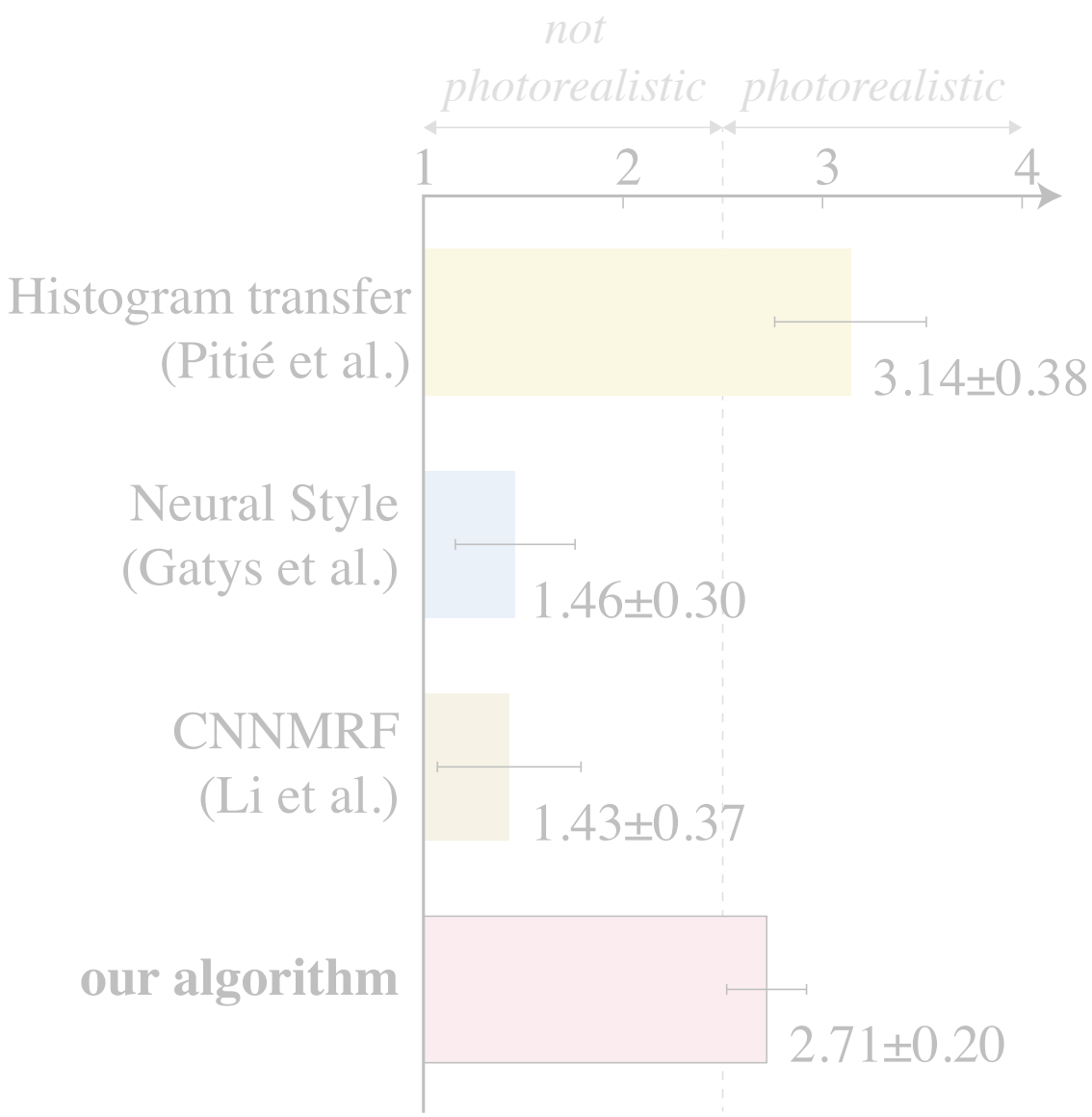

Model


Input

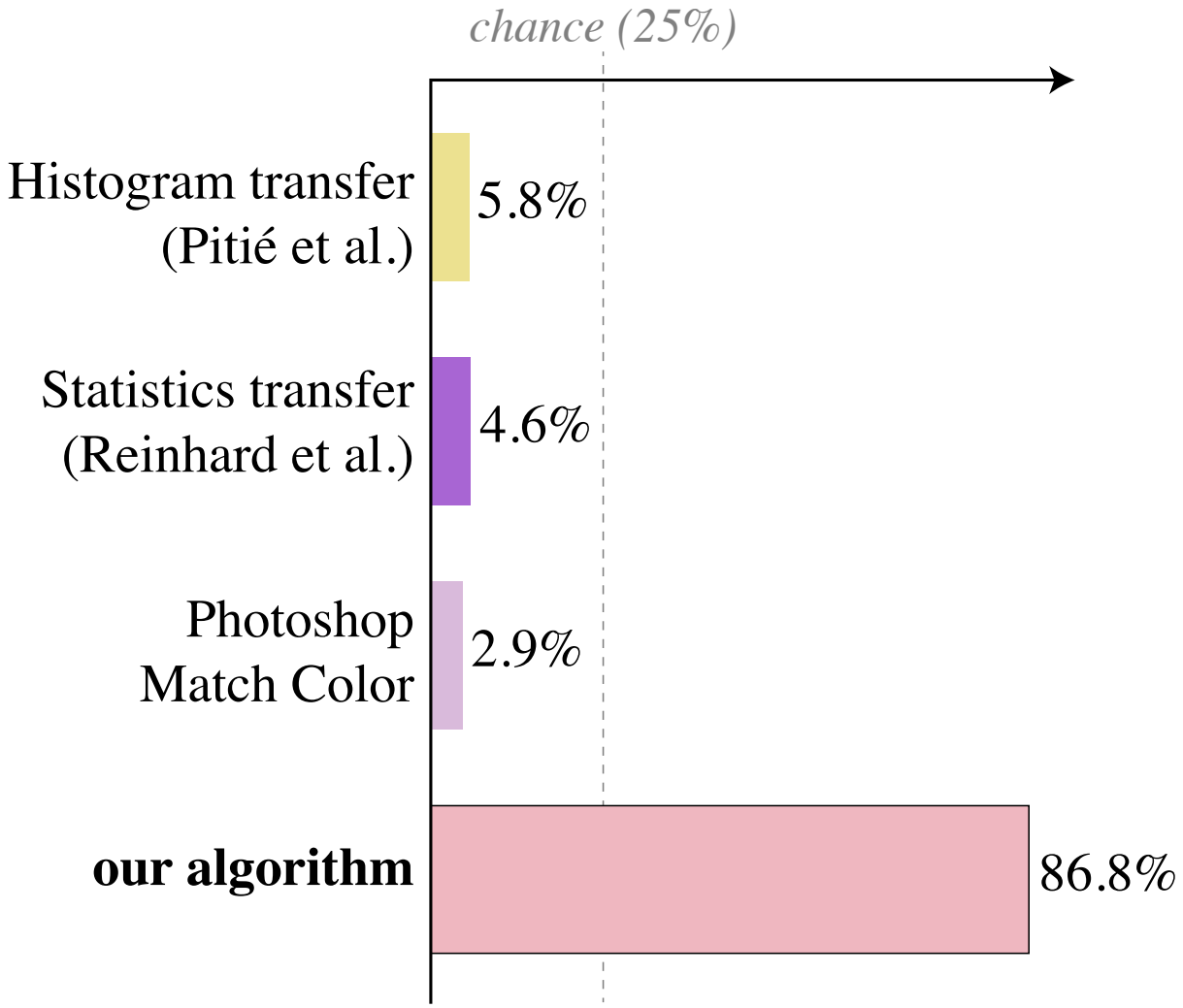

Our result


Portrait Style Transfer


not photorealistic
photorealistic
1
2
3
4
Histogram transfer (Pitié et al.)
3.14±0.38
Neural Style (Gatys et al.)
1.46±0.30
CNNMRF (Li et al.)
1.43±0.37
our algorithm
2.71±0.20
Photorealism scores



Photorealism scores



Style faithfulness preference

Disclaimer: specialized algorithms like to perform as well or better.

Required Steps for Productization

- Faster performance
  - Research prototype is too slow, on the order of minutes per photo
  - Feed-forward network, GPU, cloud…
- Smaller memory footprint
  - Currently based on VGG-19
  - Shallow network [Ustyuzhaninov et al., 2016]

What’s next?

Deep Photo Style Transfer is only a First Step

- Vast algorithm design space to explore

Background on Neural Style Transfer

1. Decompose the input and model images using a neural network
2. Impose some of the statistics of the model image
3. Reconstruct the output image

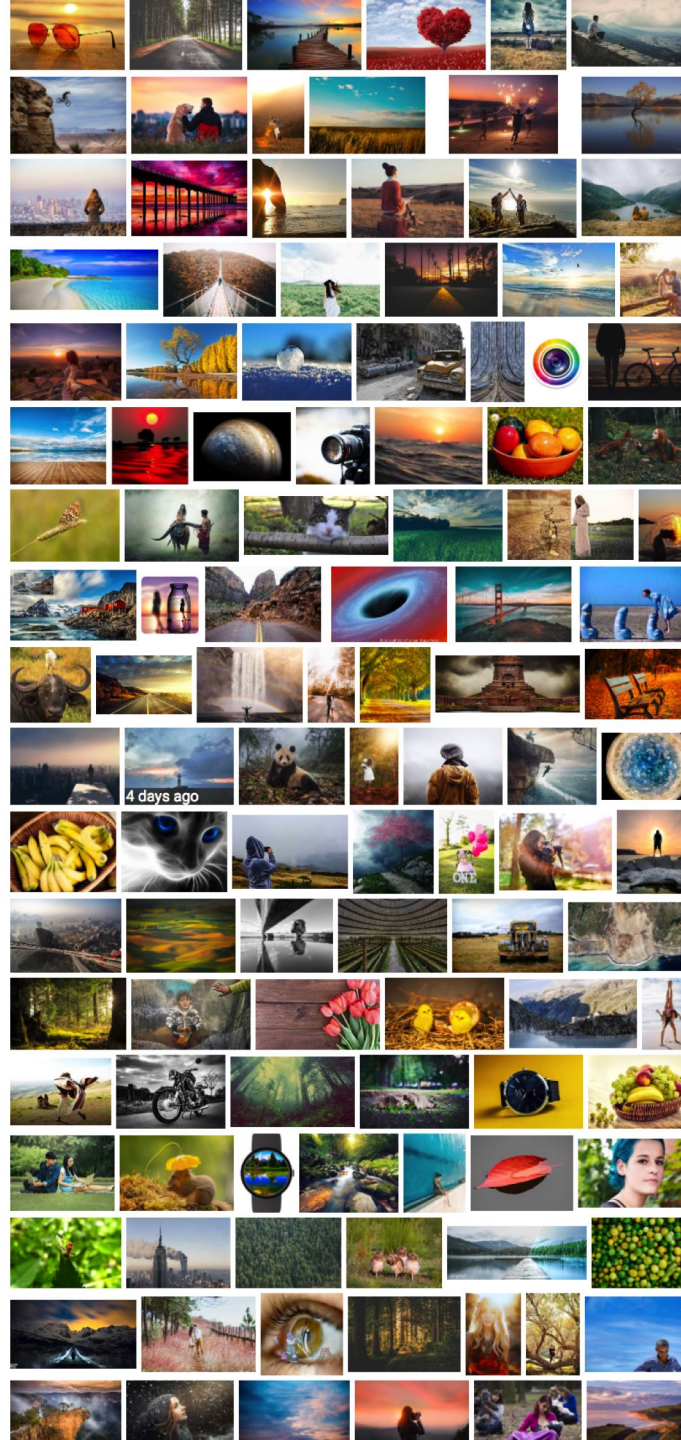
- Many questions to answer
  - Why a network trained for classification? Can we do better?
  - “Spillovers are obvious”, can we fix them without a full semantic analysis?
  - Where does the distortion come from?

Learning Creative Edits from Experts

- Challenge #1: experts cannot create 1 million+ photos for training a CNN
- Challenge #2: the range of effects is large
- Challenge #3: not all styles apply to all photos

Exploiting On-line Collections

- What can we do with 1 million or more photos?
- How do we find the “good ones” and exploit them?
- Working with sets instead of samples à la CycleGAN or Exposure is promising but still needs work
  [Zhu et al. 2017, Zhang et al. 2018]



A Great Photo from Every Shot

Better Photos, Easily

Photo Style Transfer

- A simple and powerful way to edit photos
- Challenges us to better understand images
  - What is style? What is content?
  - What makes an image look like it does?
  - How to preserve the image structure?
- What I did not show: videos, panoramas, weather, lighting, perception, performance…

Thanks!

Nothing would have been possible without my collaborators:
Mathieu Aubry, Soonmin Bae, Kavita Bala, Connelly Barnes,
Adrien Bousseau, Vladimir Bychkovsky, Eric Chan, Jiawen Chen,
Jeff Chien, George Drettakis, Frédo Durand, Bill Freeman, Sam Hasinoff,
Jan Kautz, Fujun Luan, Eli Shechtman, and Yichang Shih