From **TensorFlow** to **Taichi**: A **GAN** for Computational Photography and A **Library** for Computer Graphics

Presented by
Yuanming Hu 胡渊鸣 , MIT CSAIL

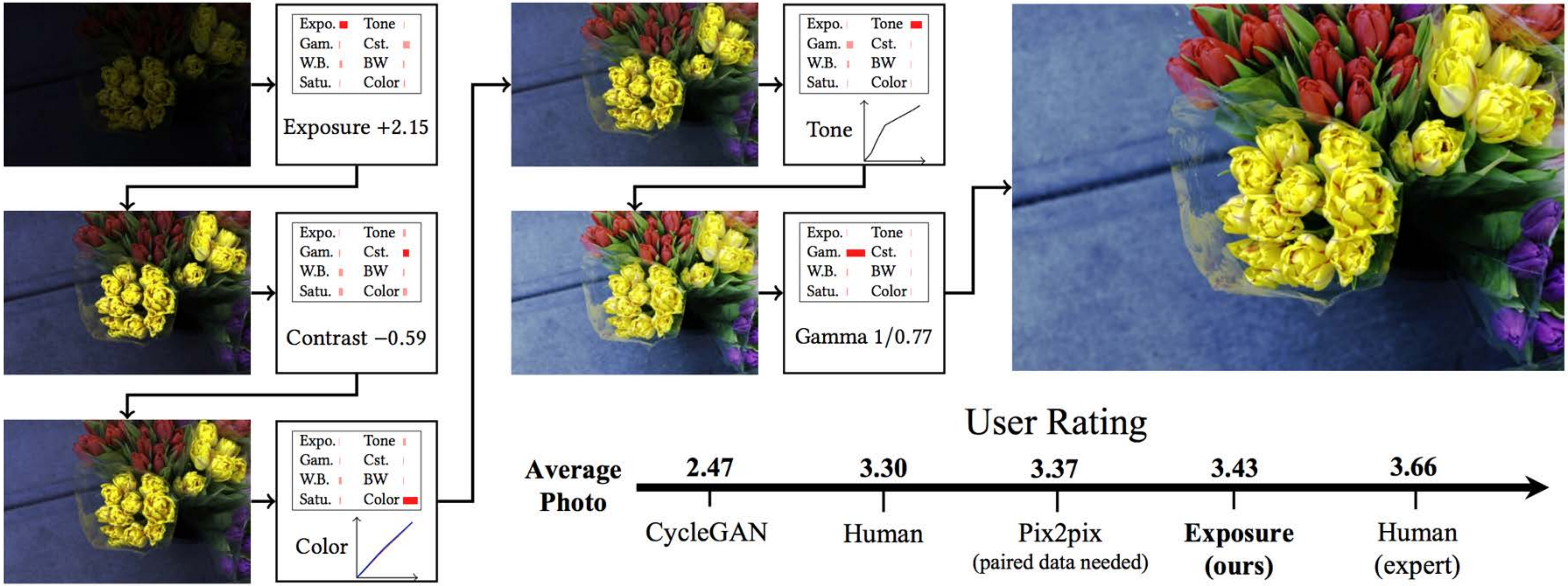
Part I

Exposure: A White-Box Photo Post-Processing Framework

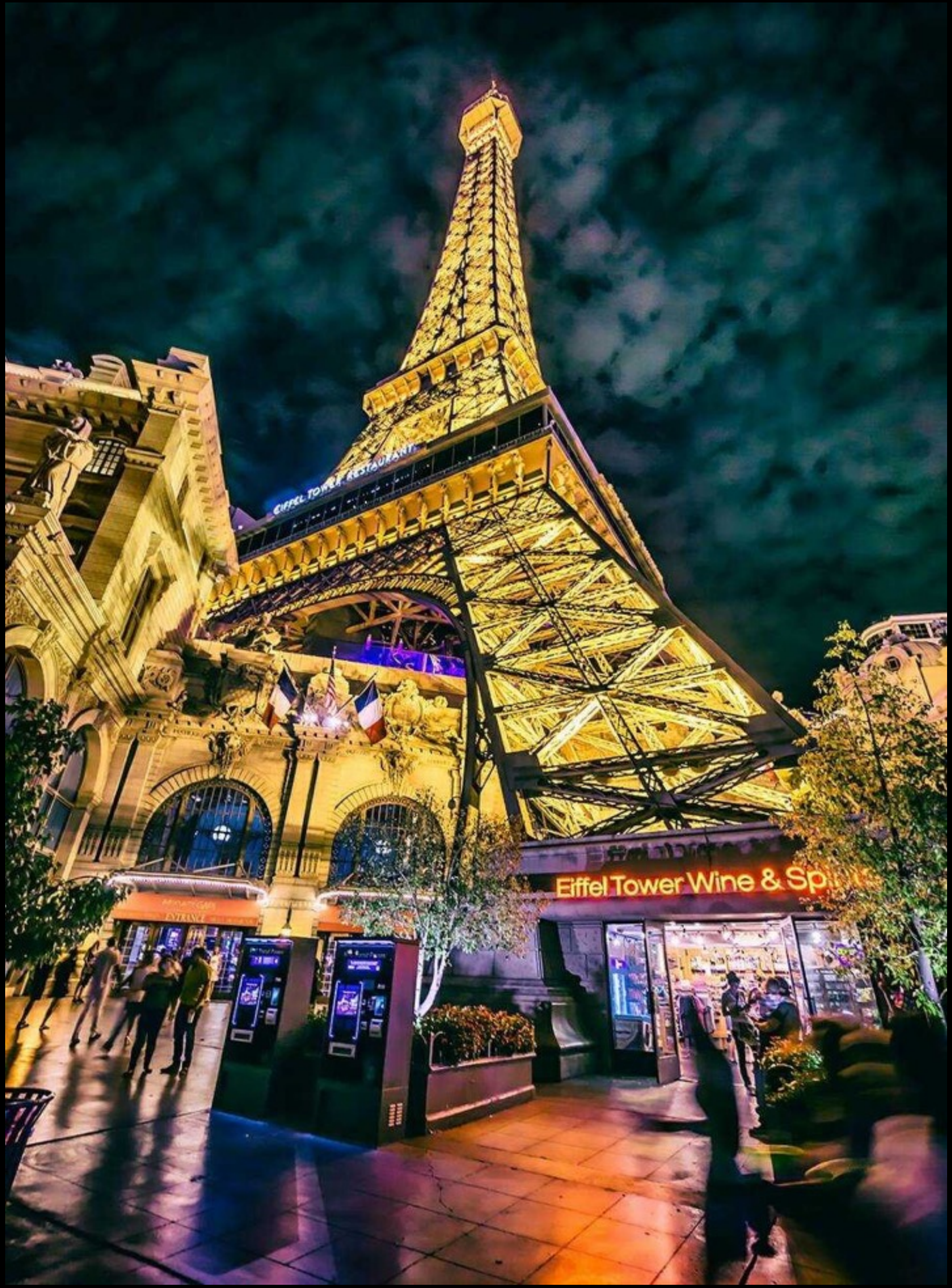
ACM Transactions on Graphics, to be presented at SIGGRAPH 2018

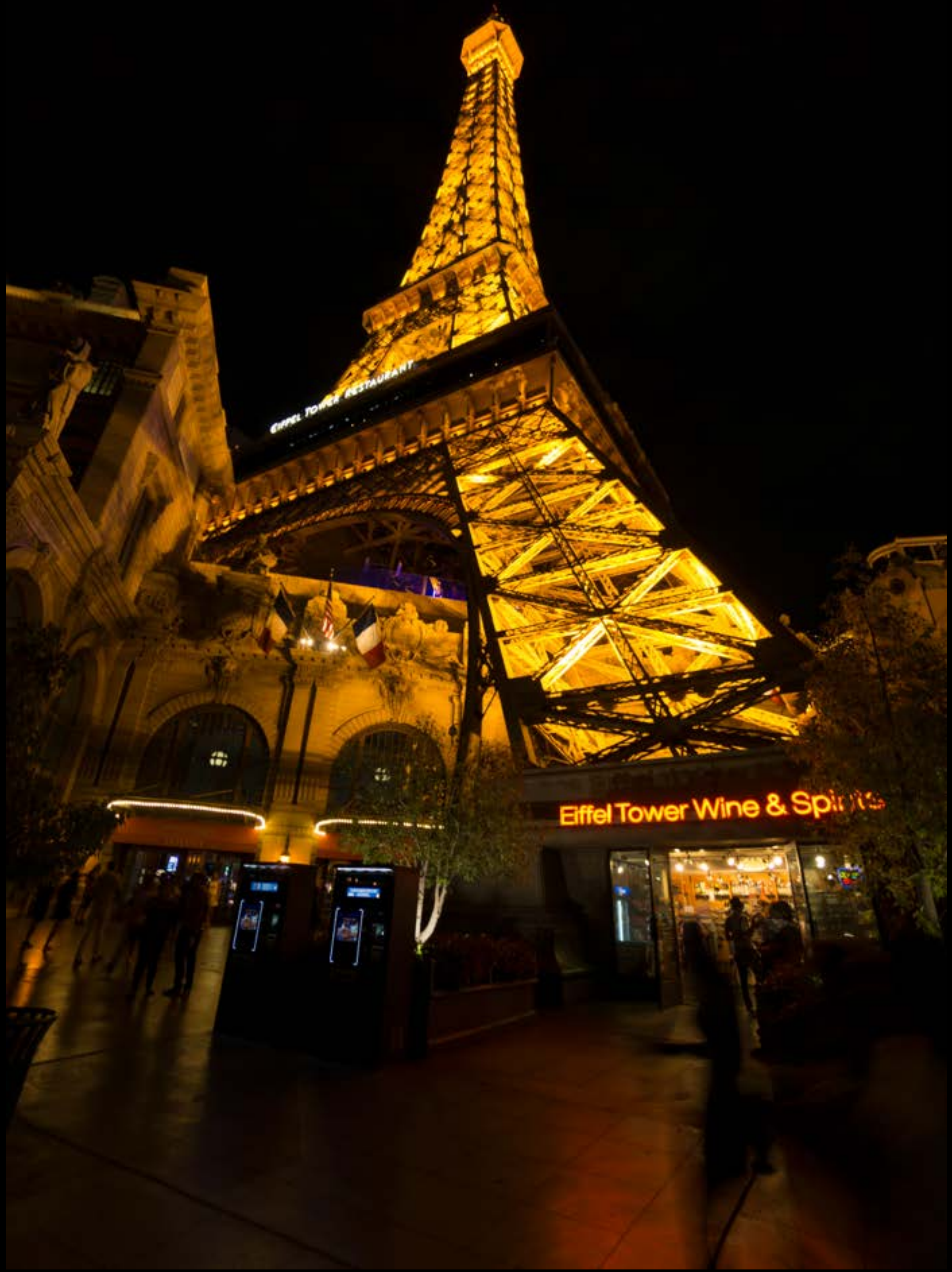
Yuanming Hu[1,2] Hao He[1,2] Chenxi Xu[1,3] Baoyuan Wang[1] Stephen Lin[1]

[1]Microsoft Research [2]MIT CSAIL [3]Peking University



“Magic”





Exposure + 2.40

Highlight -78

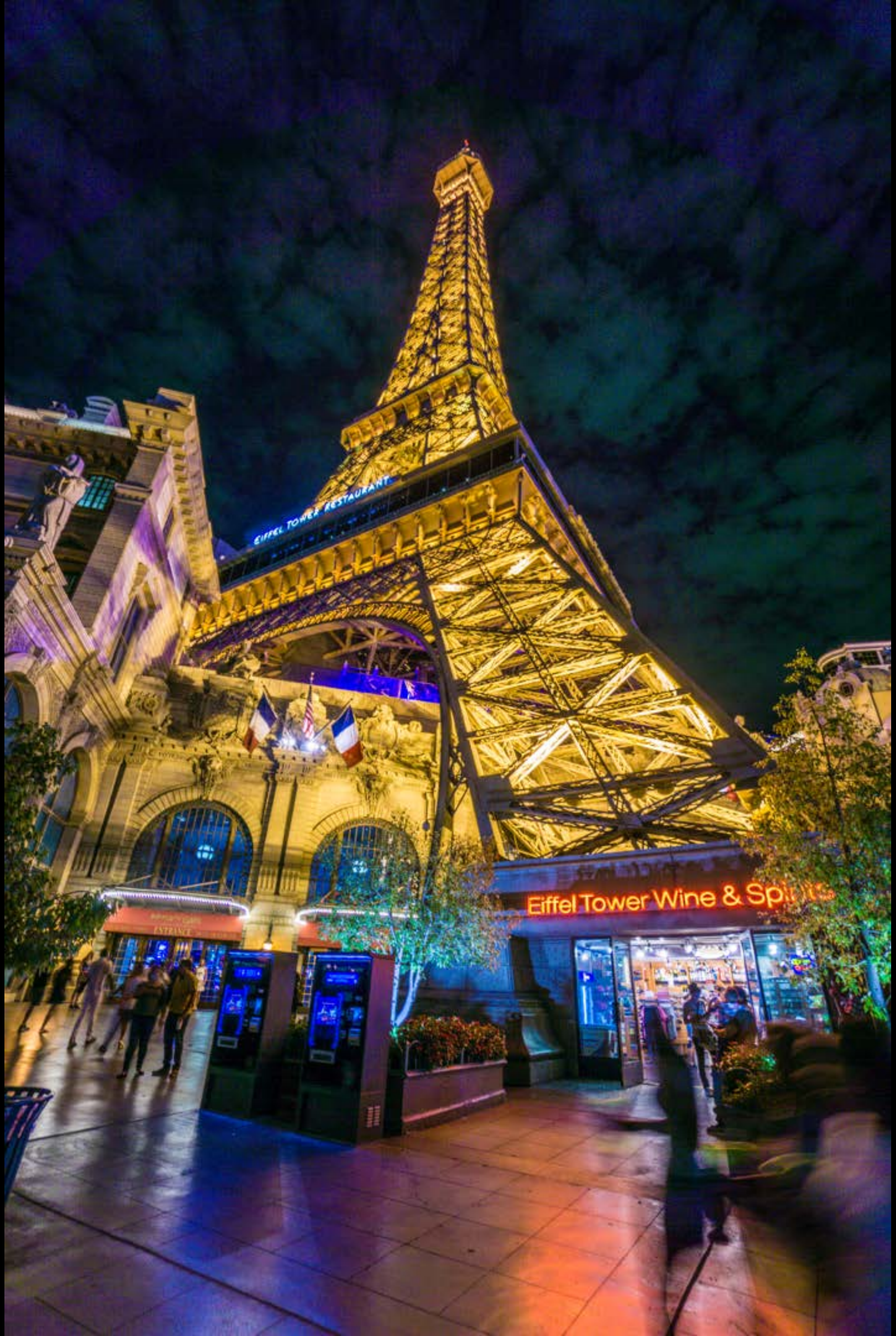
White balance

→

Temperature 2600
Tint +23

Clarity + 63

Vibrance +75

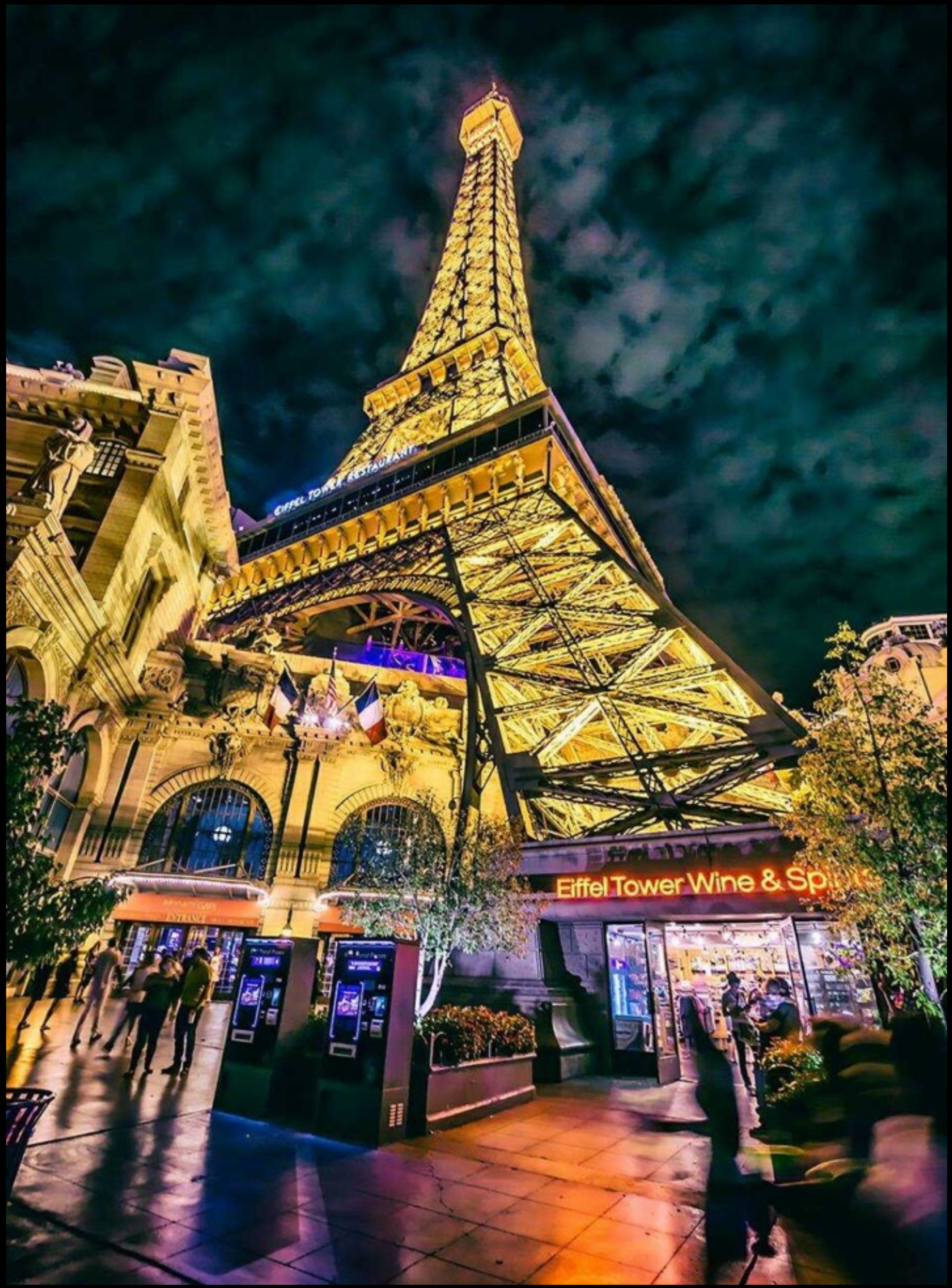

EIFFEL TOWER RESTAURANT
Eiffel Tower Wine & Spirits

Shadow + 70


EIFFEL TOWER RESTAURANT
Eiffel Tower Wine & Spirits

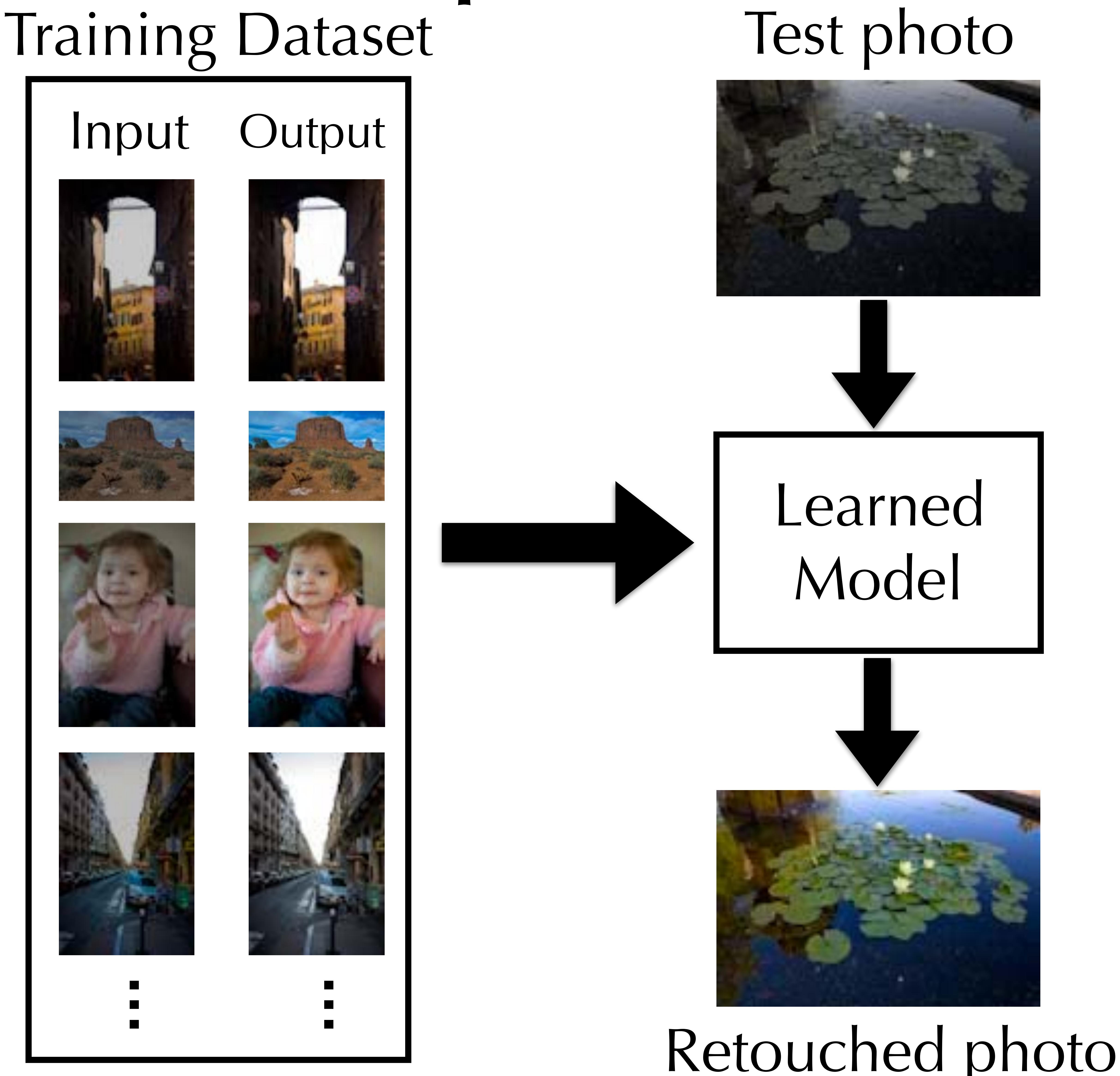

Eiffel Tower Wine &

...


Eiffel Tower Wine &

Can machines learn this process?

- **Input dataset:**
  - A set of RAW photos
  - A set of retouched target photos
- **Goal:**
  - Post-process raw photos following the style similar to the training dataset



Learning-based Photo Processing

Bychkovsky et al. 2011, **Learning Photographic Global Tonal Adjustment with a Database of Input / Output Image Pairs**

MIT-Adobe FiveK Dataset

x5000

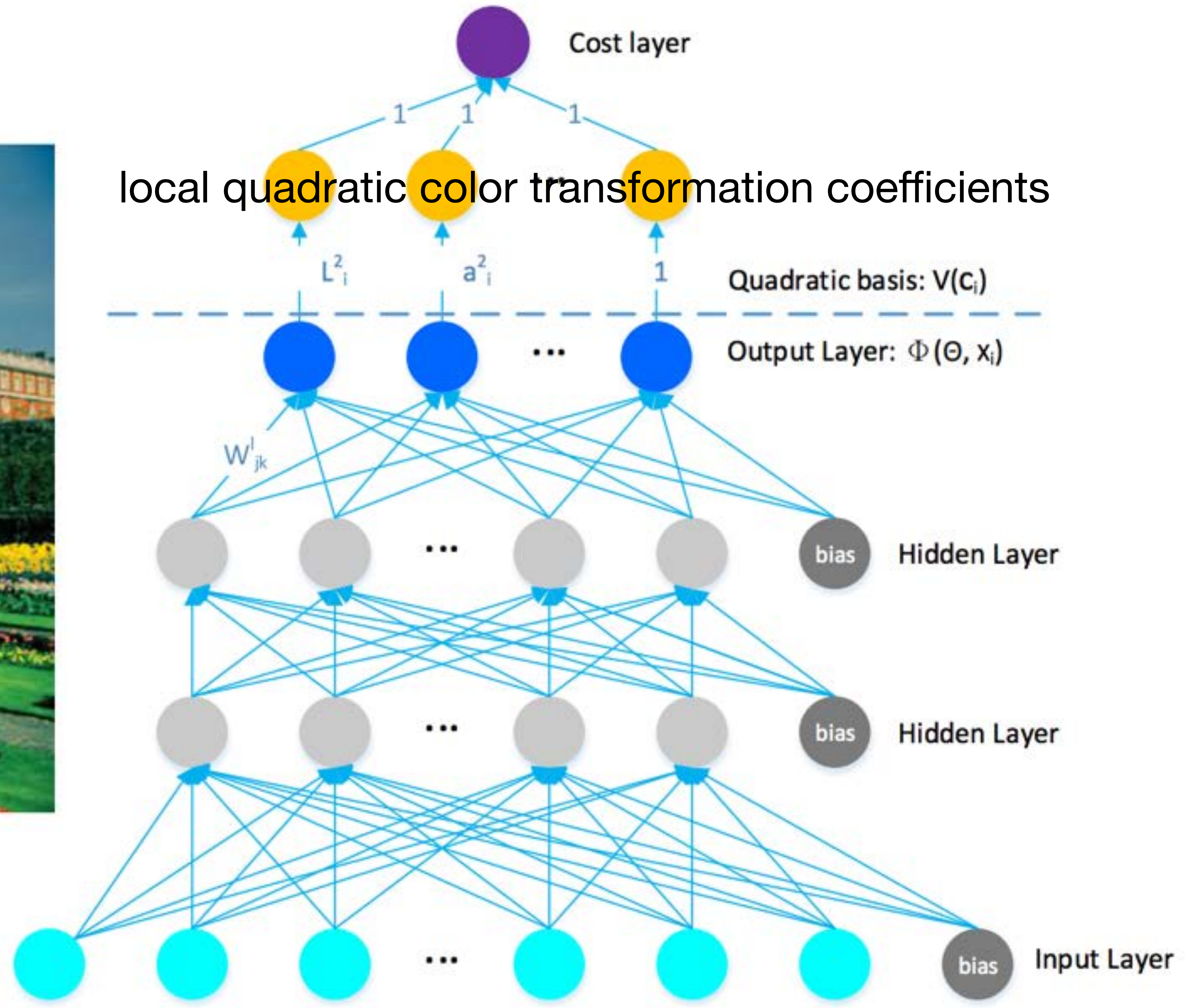
+

Learning-based Global Tonal Adjustment

Learning-based Photo Processing

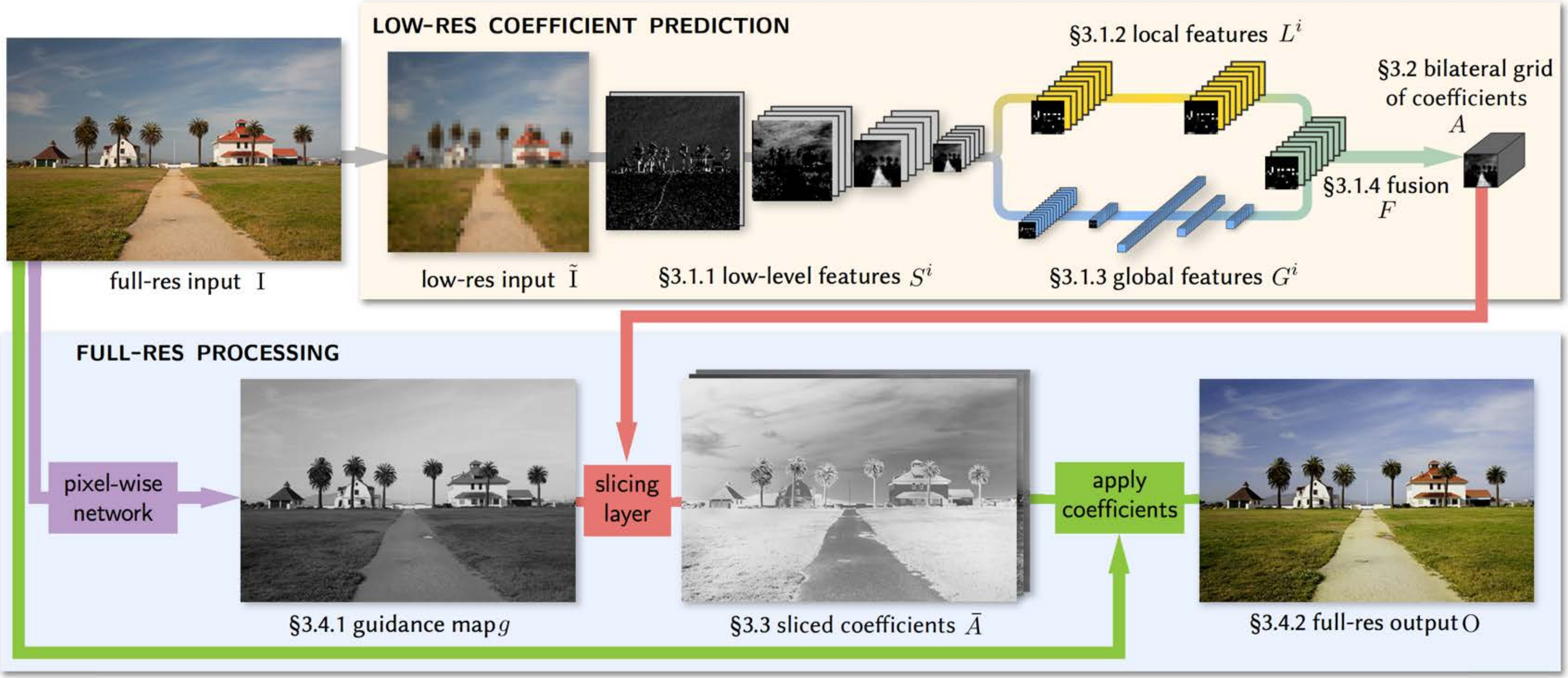
Yan et al. 2014, **Automatic Photo Adjustment Using Deep Neural Networks**

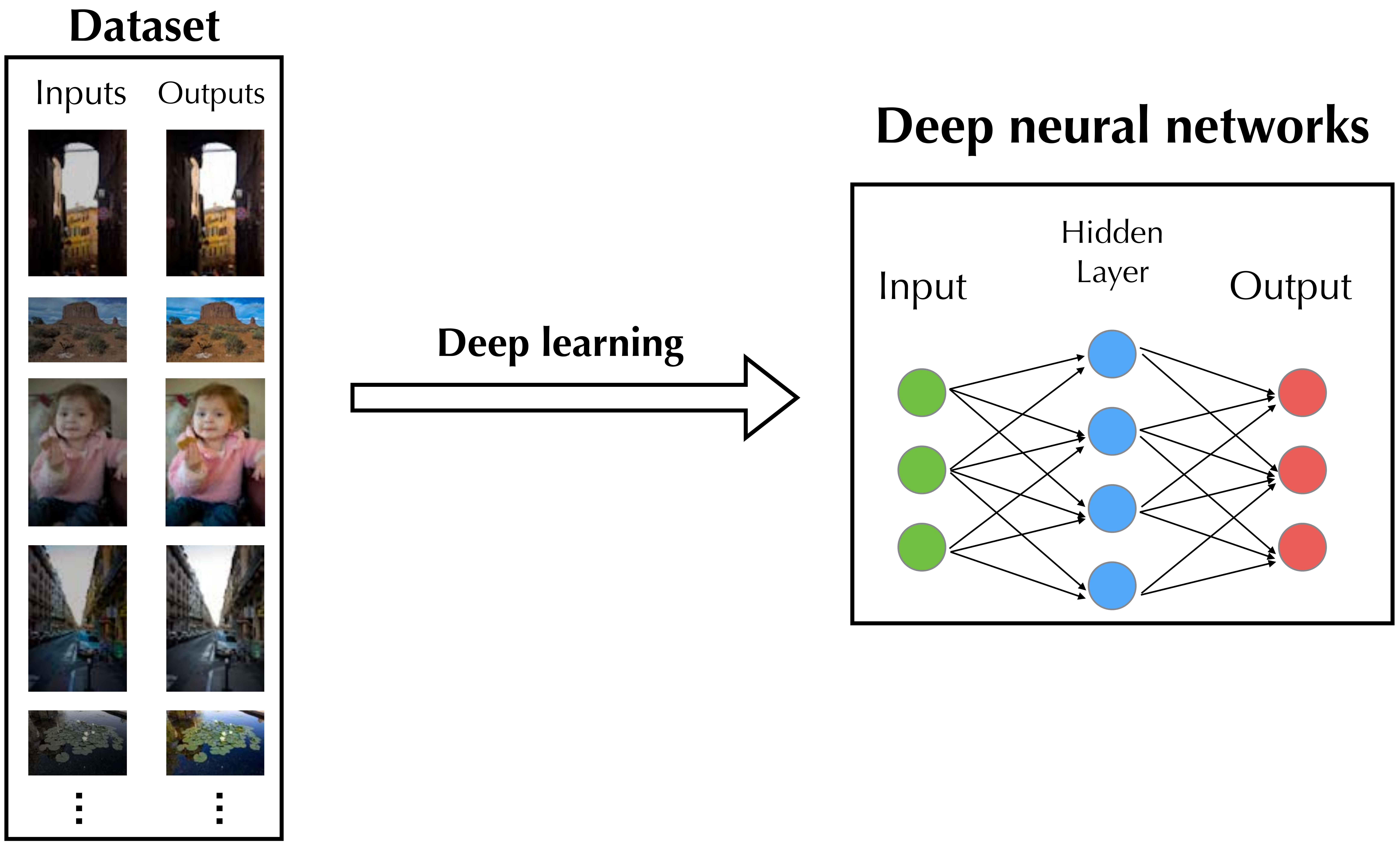
(a) (b)



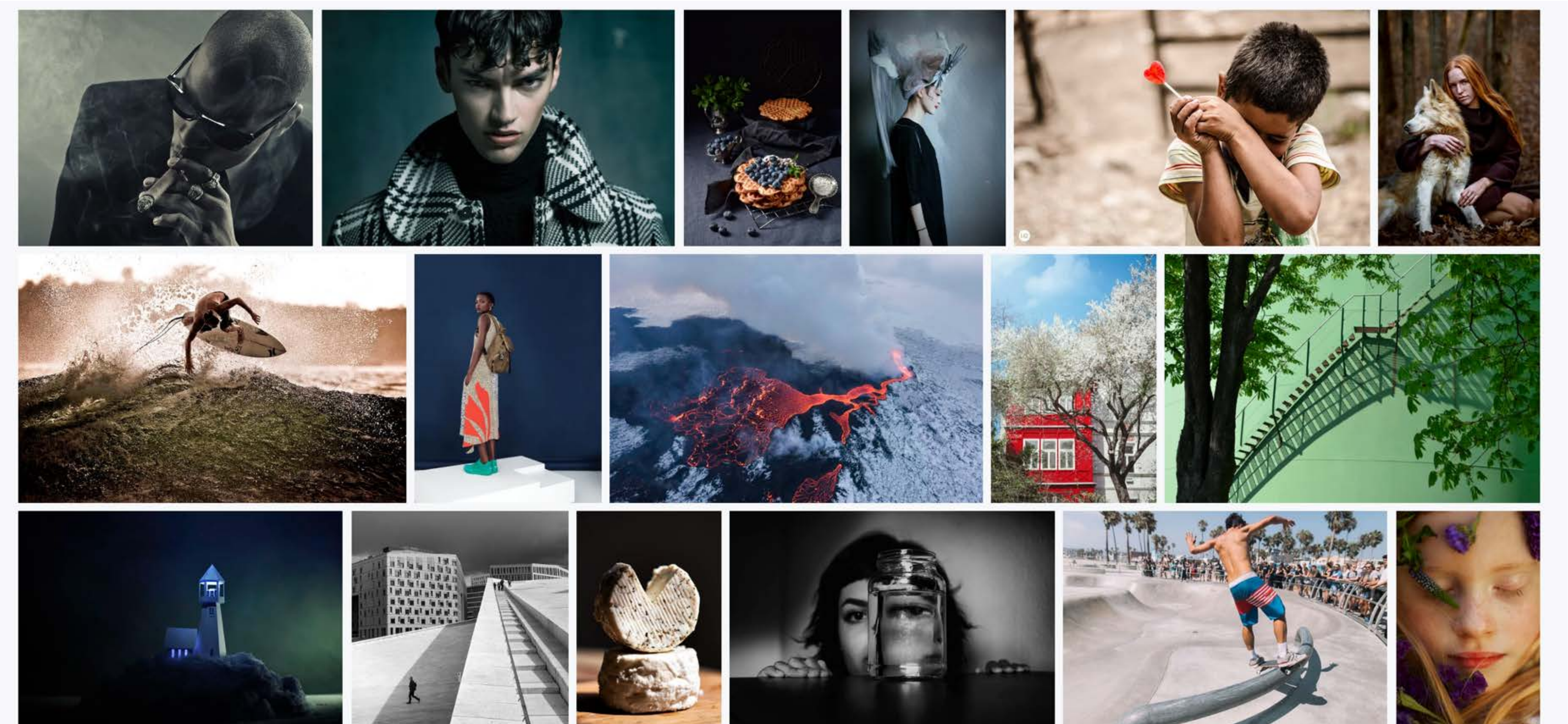
Learning-based Photo Processing

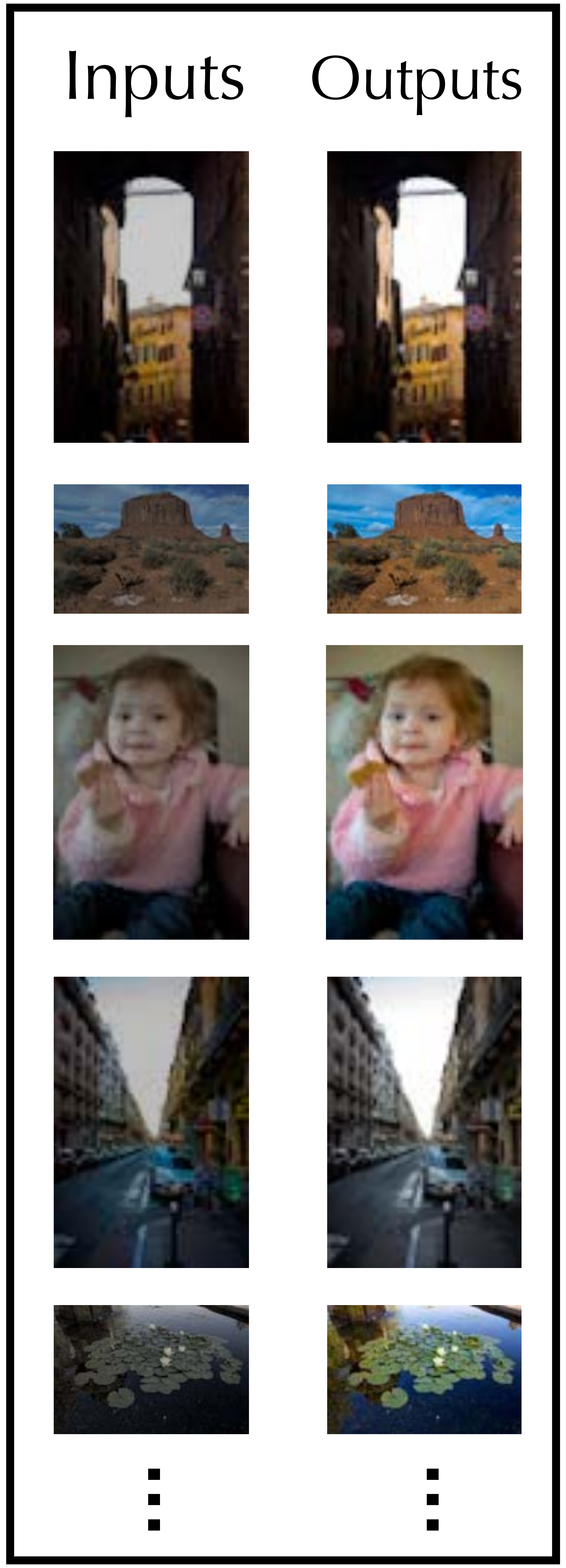
Gharbi et al., **Deep Bilateral Learning for Real-Time Image Enhancement**




Dataset
Inputs
Outputs
Deep learning
Deep neural networks
Input
Hidden
Layer
Output

500px.com





Outputs

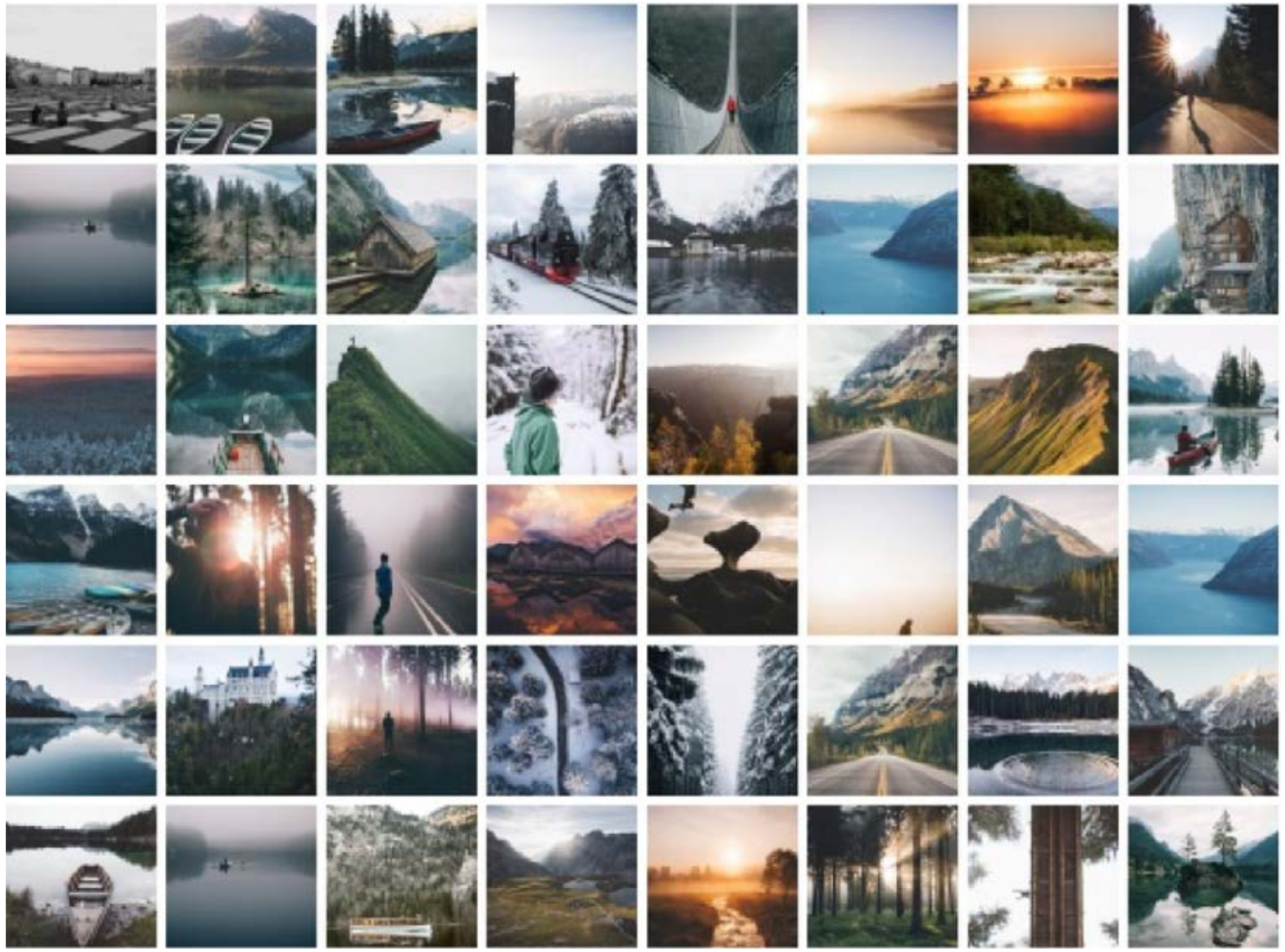
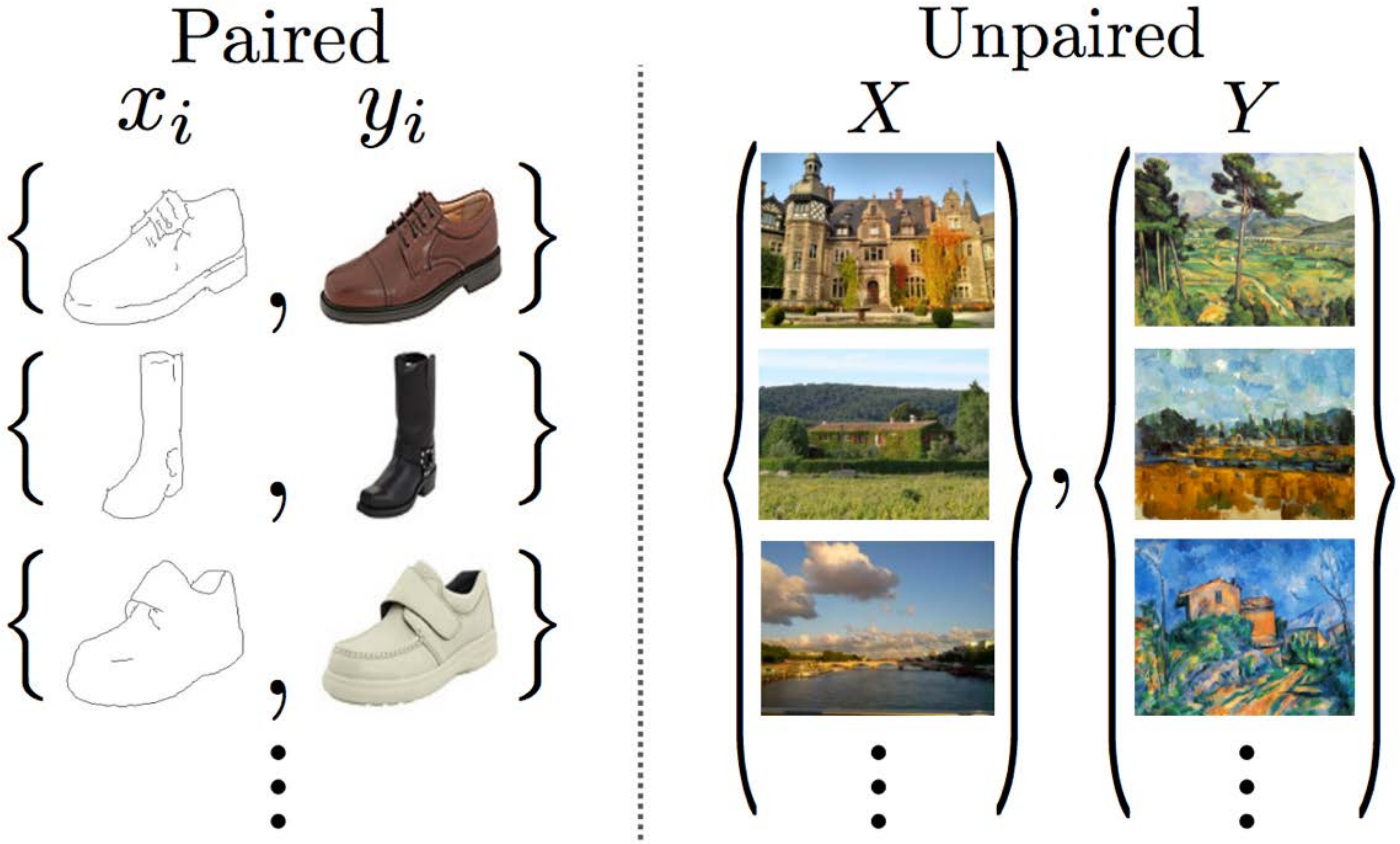


Image Translation

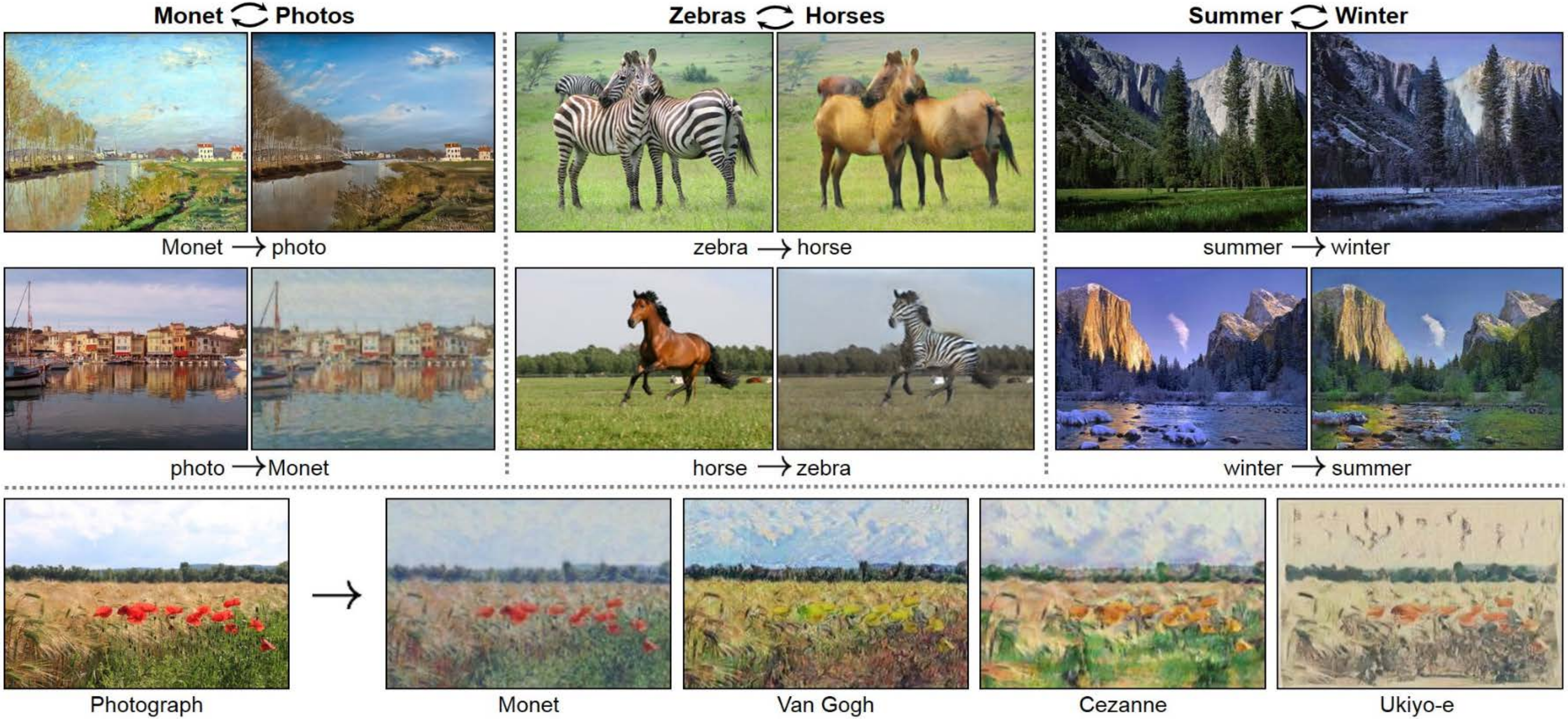


[Isola et al. 2017, Image-to-Image Translation with Conditional Adversarial Networks]

[Zhu et al. 2017, Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks]

CycleGAN

[Zhu et al. 2017, Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks]

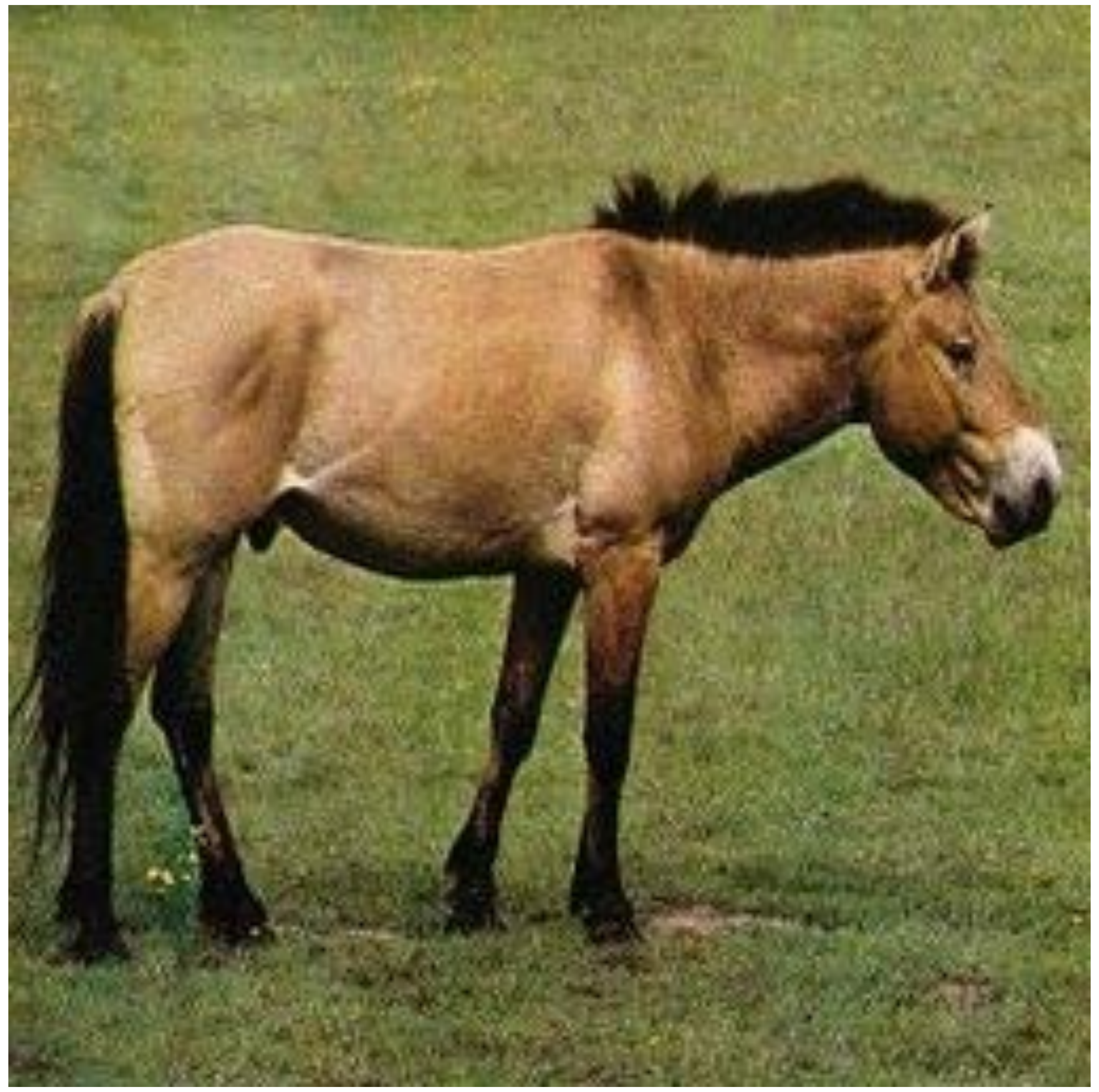


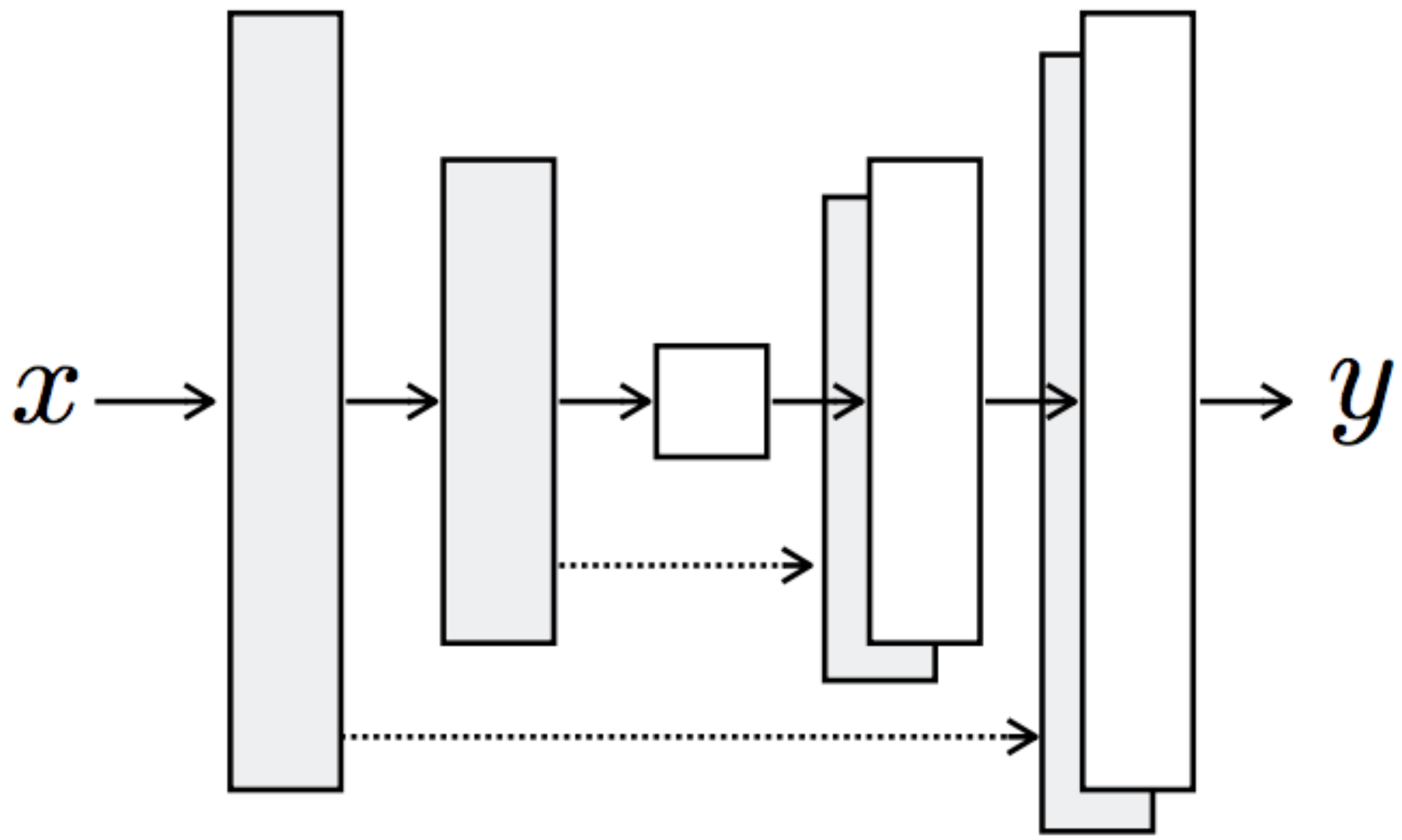
(Conditional) Generative Adversarial Networks (c-GANs)

Generator

Encoder/
decoder-based
CNN

256x256 px





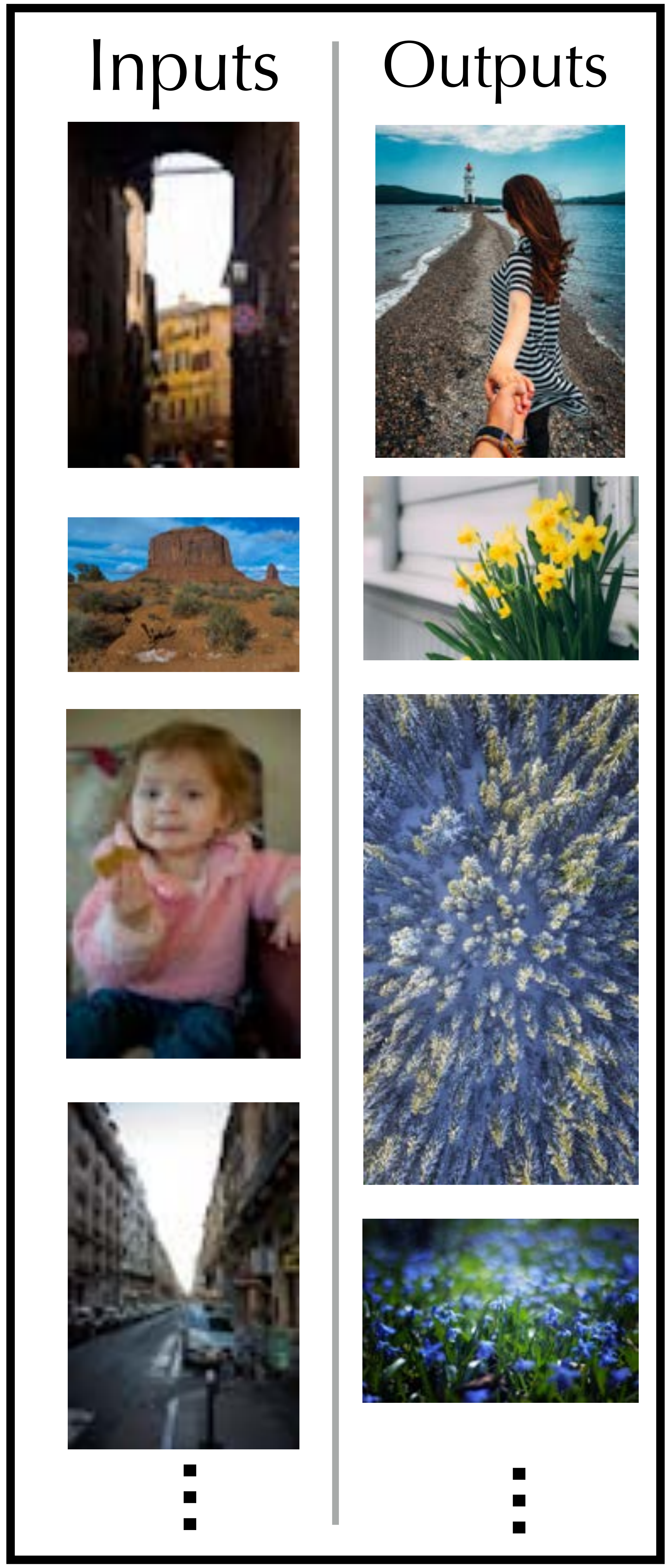
256x256 px

[Zhu et al. 2017, Unpaired Image-to-Image Translation
using Cycle-Consistent Adversarial Networks]

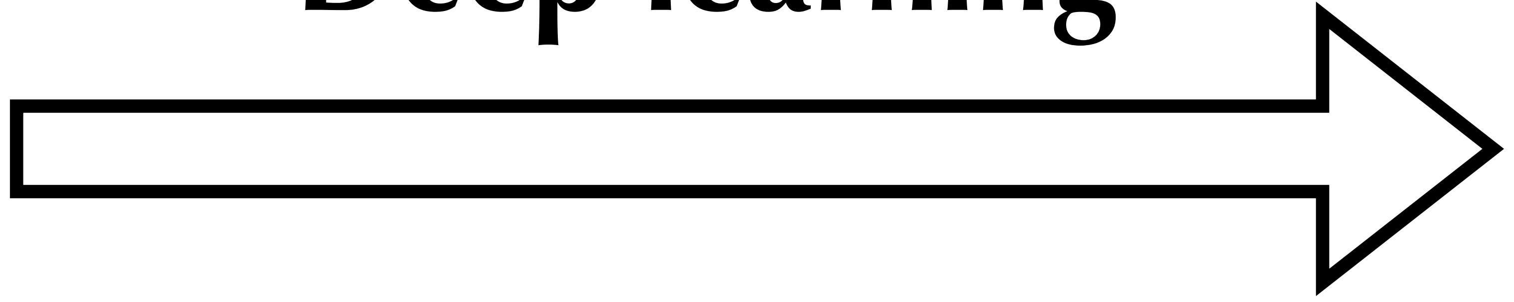
	High Resolution	Unpaired Training	Human Understandable	End-to-end Processing
Tonal Adjustment Learning Bychkovsky et al. 2011	○		○	
Local color transform learning Yan et al.	○			○
Deep Bilateral Learning Gharbi et al.	○			○
CycleGAN, Zhu et al.		○		○
?	○	○	○	○

Black Box A

(Unpaired data)

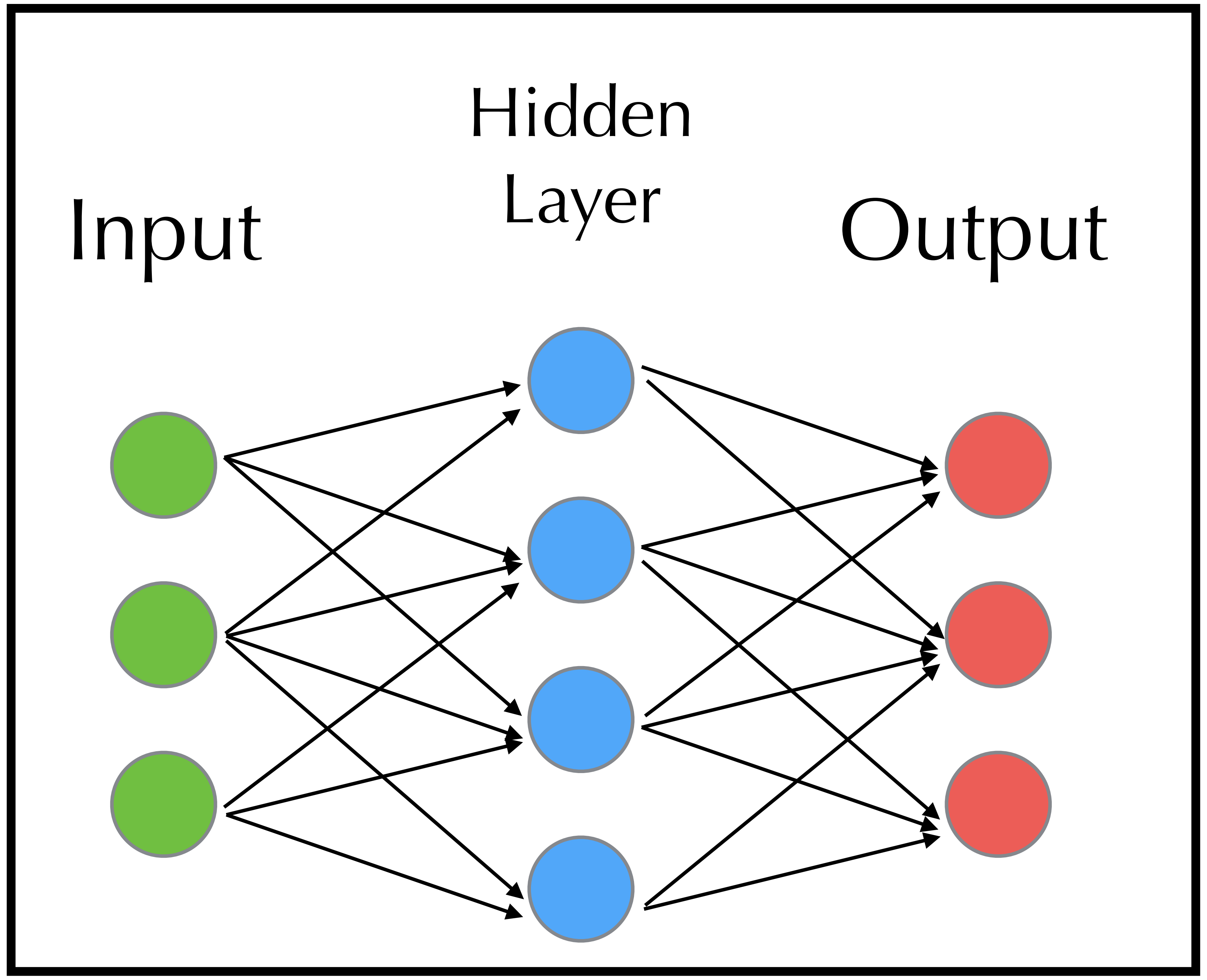


Deep learning



Black Box B

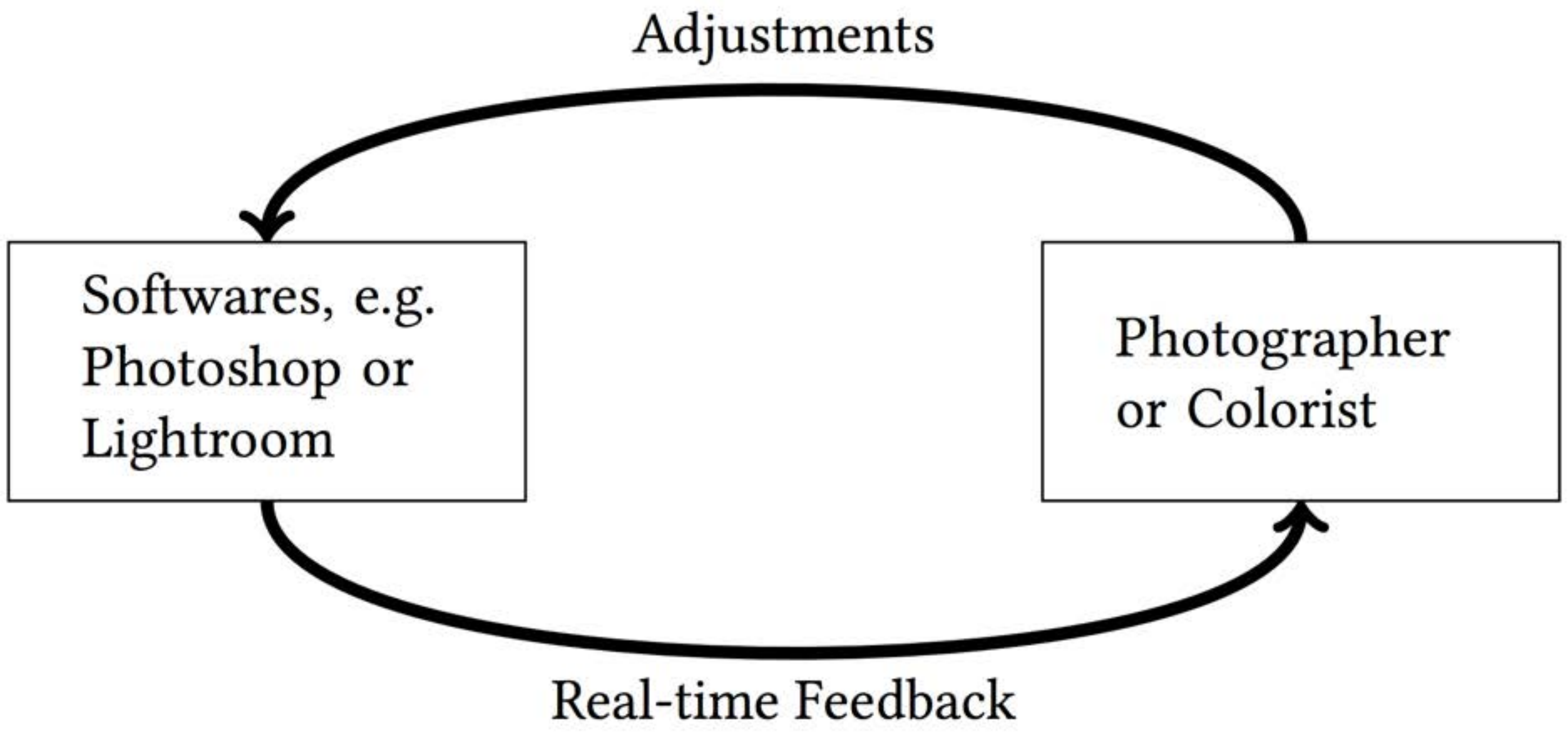
(deep neural networks)



Traditional deep-learning approaches generate black boxes (CNNs) out of existing ones (datasets).

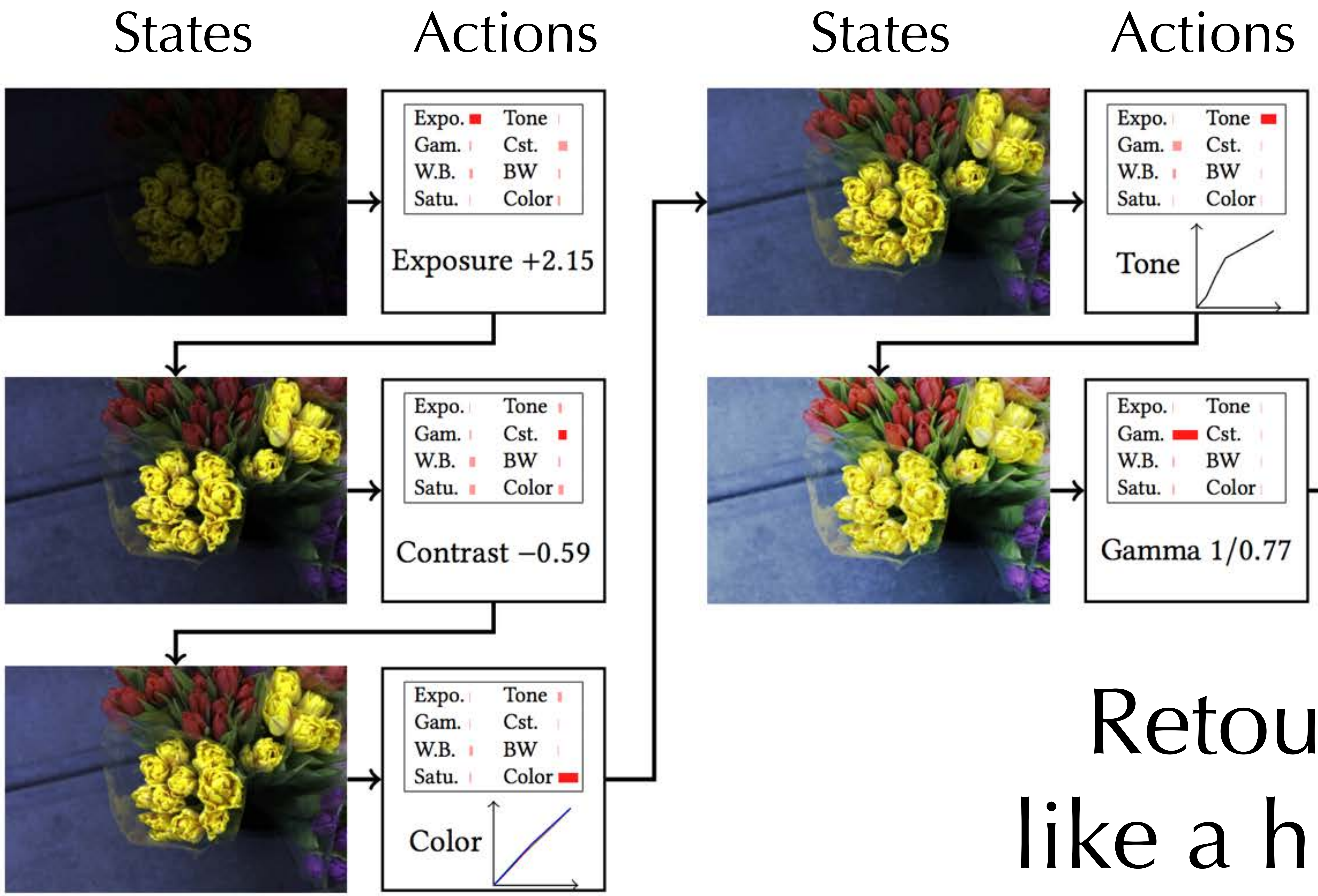
To understand the magic of photo retouching, we need a **white box** result.

Modelling Photo Post-Processing



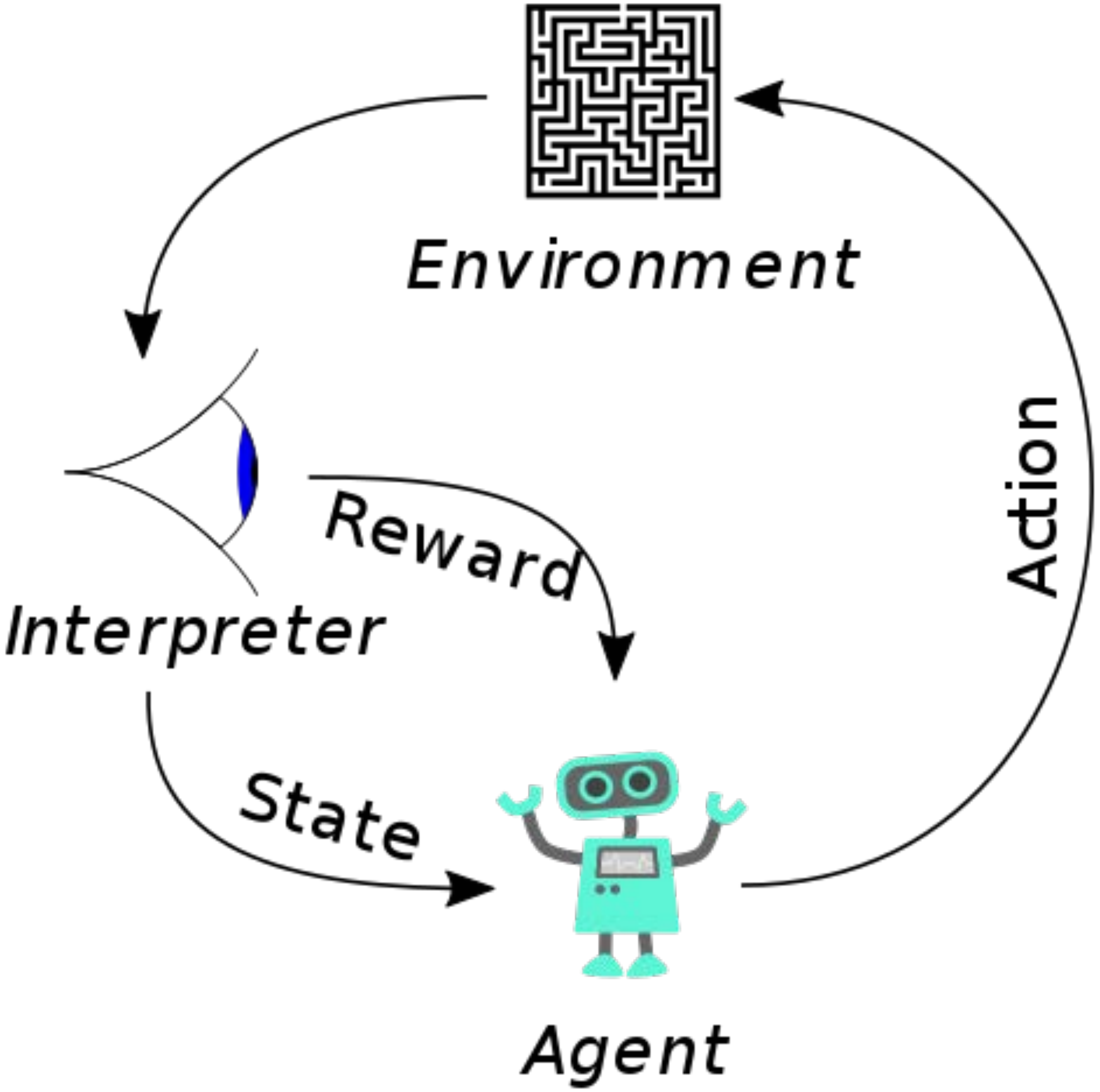
- **People retouch photos step-by-step**
- **Feedback is important**
  - In many software such feedback is done in real-time
  - Human usually does not specify a concrete adjustment number (say, “Exposure + 1.32”)

Modelling Photo Post-Processing



Retouch photos
like a human artist!

Reinforcement Learning



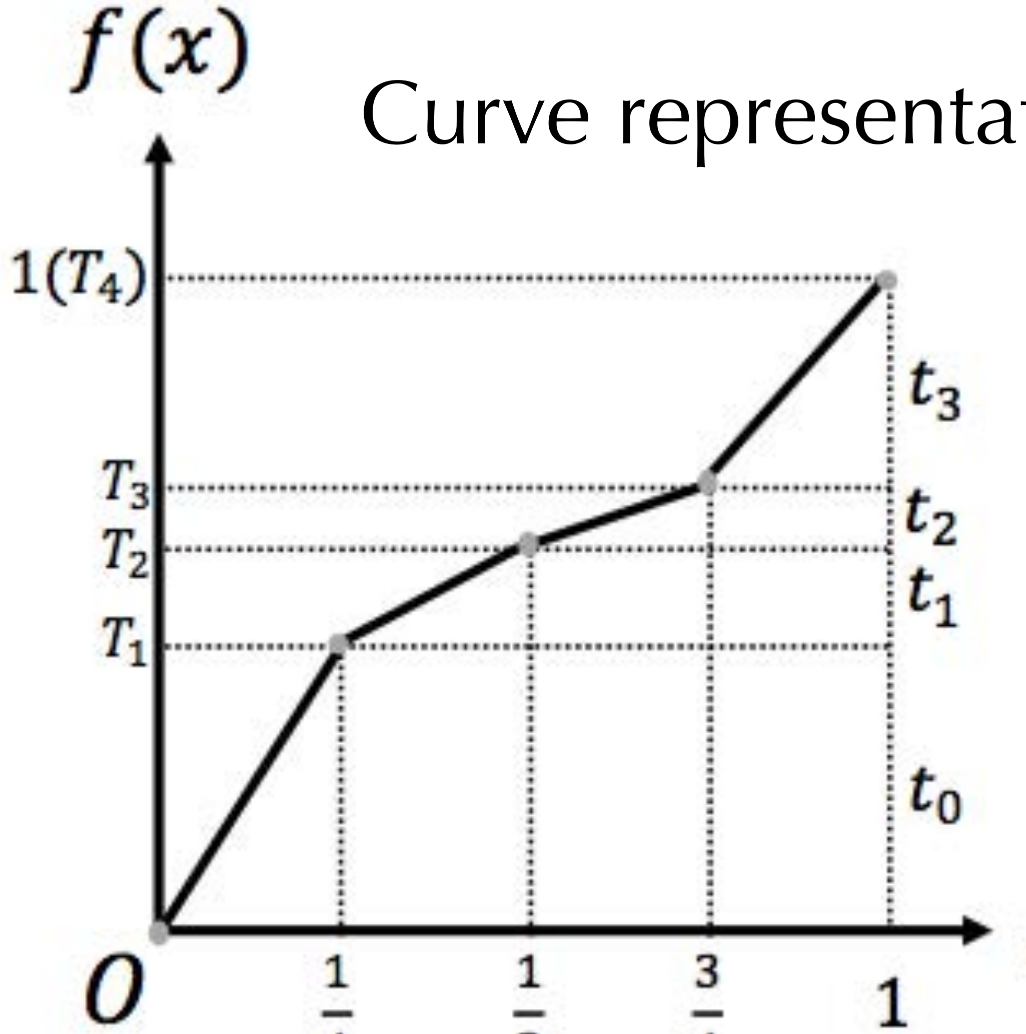
- ✦ **People retouch photos step-by-step**
  - I.e., transit from one state to another
- ✦ **Feedback is important**
  - Adjust (e.g., using **policy gradients**) the behaviour according to **rewards**

$$\nabla_{\theta_1} J(\pi_\theta) = \mathop{\mathbb{E}}_{\substack{s\sim\rho^\pi \\ a_1\sim\pi_1(s) \\ a_2=\pi_2(s,a_1)}} [\nabla_{\theta_1} \log \pi_1(a_1|s) Q(s,(a_1,a_2))],$$

$$\nabla_{\theta_2} J(\pi_\theta) = \mathop{\mathbb{E}}_{\substack{s\sim\rho^\pi \\ a_2=\pi_2(s,a_1)}} [\nabla_{a_2} Q(s,(a_1,a_2)) \nabla_{\theta_2} \pi_2(s,a_1)],$$

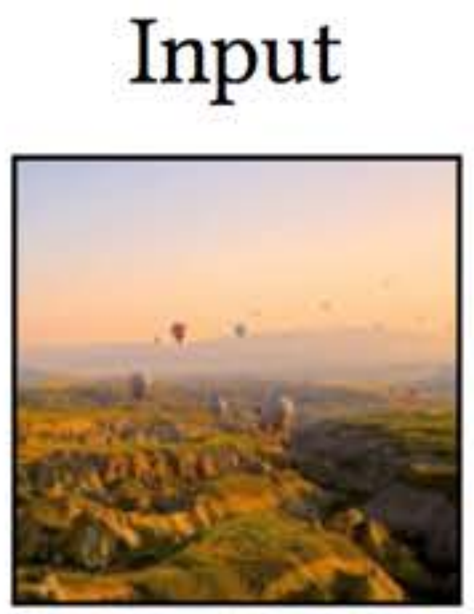
Actions: Filters with Their Gradients

Curve representation



Filters

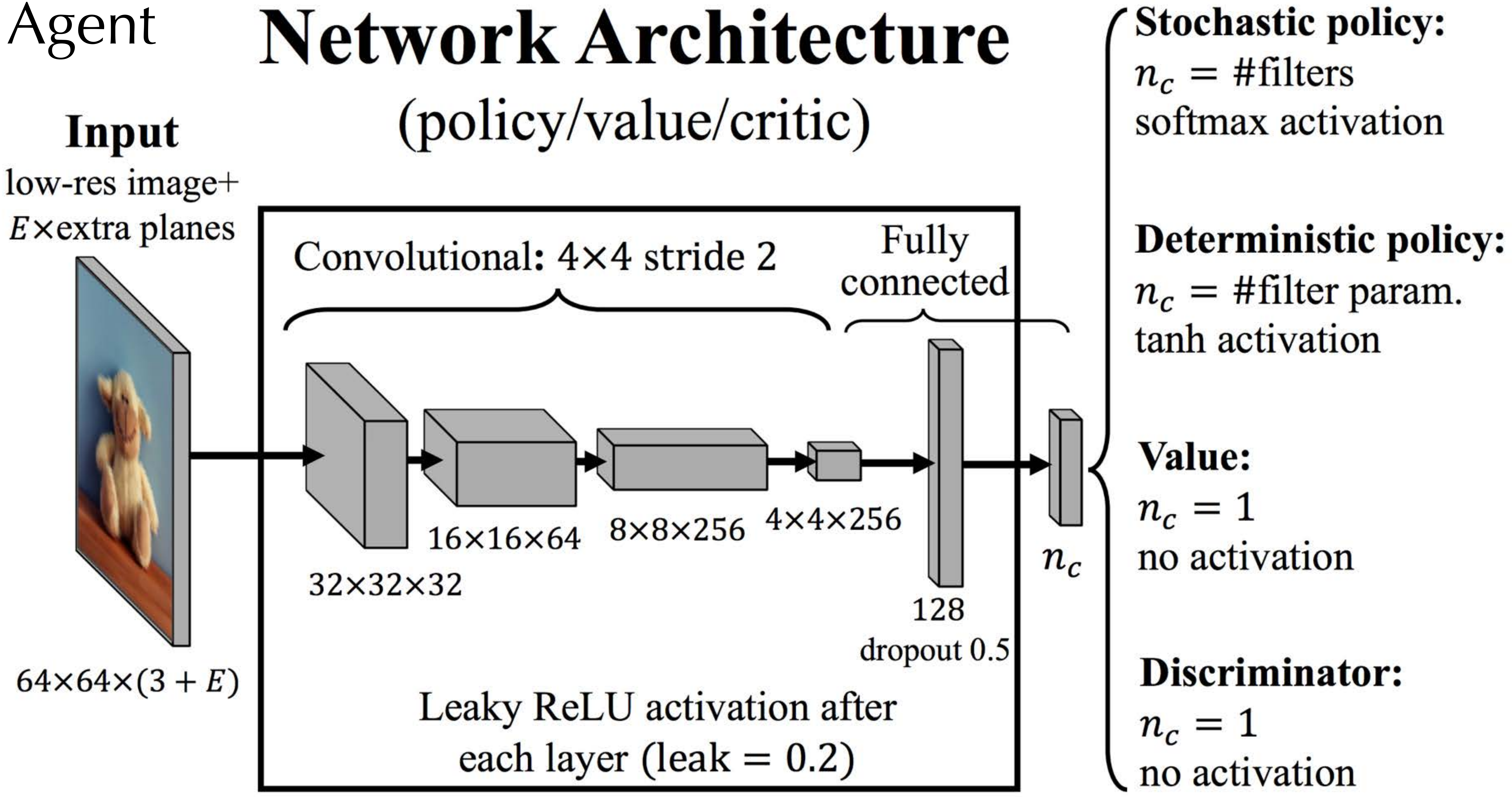
Output

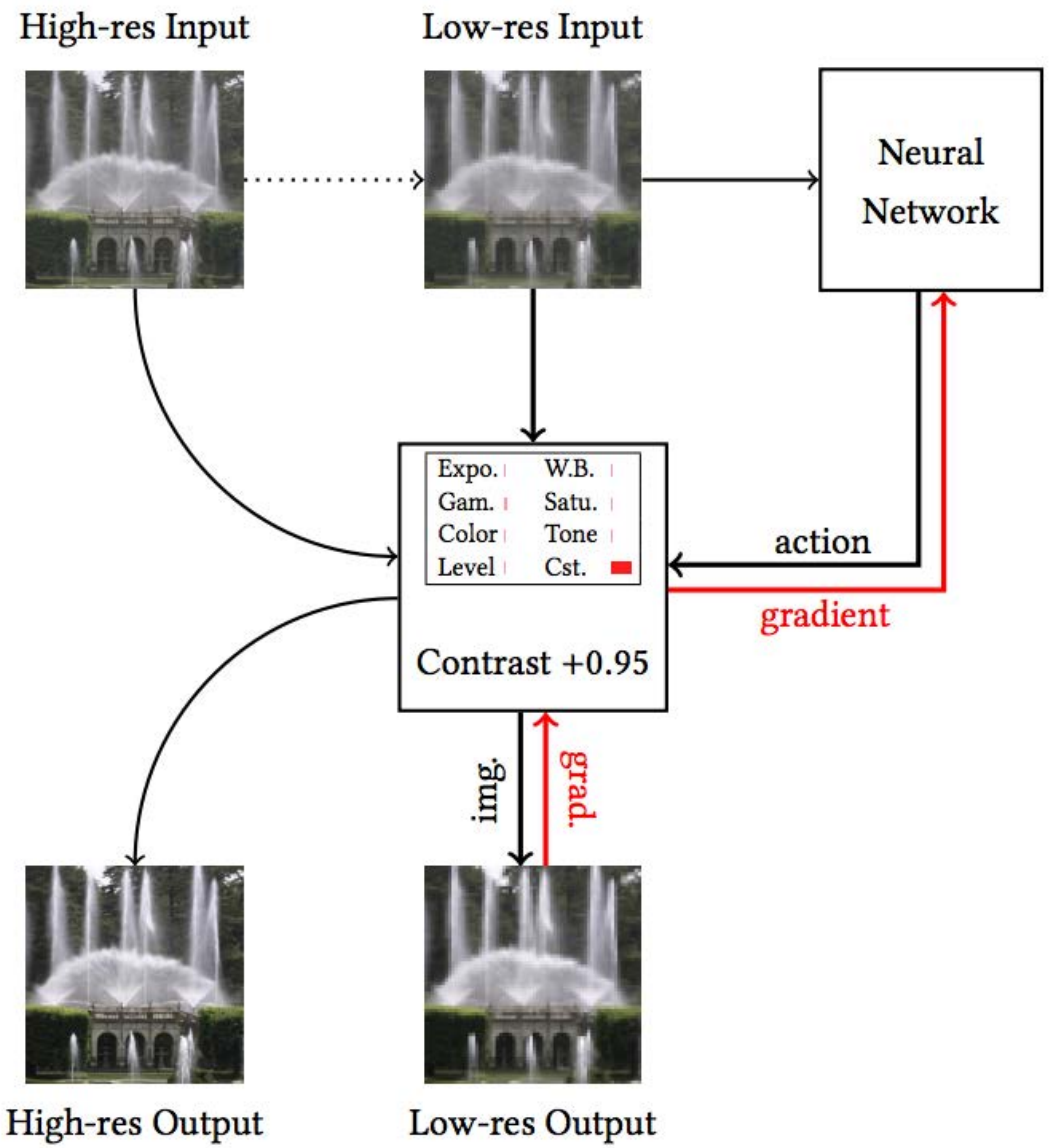


Environment: Wasserstein GAN-GP

Agent

Network Architecture
(policy/value/critic)





ALGORITHM 1: Training procedure

Input: Input datasets D_{RAW} and $D_{\text{retouched}}$; batch size $b = 64$, learning rates $\alpha_\theta = 1.5 \times 10^{-5}$, $\alpha_\omega = 5 \times 10^{-5}$, $\alpha_\nu = 5 \times 10^{-4}$, $n_{\text{critic}} = 5$

Output: Actor model $\theta = (\theta_1, \theta_2)$, critic model v, and discriminator model w

Initialize the trajectory buffer with 2, 048 RAW images;

while *θ has not converged* **do**

  for *i in $1..n_{critic}$* **do**

    Sample a batch of b finished images from the trajectory buffer;

    Sample a batch of b target images from $\mathcal{D}_{\text{target}}$;

    $w \leftarrow w - \alpha_w \nabla_w L_w$;

  end

  Draw a batch B of b images from the trajectory buffer;

  Delete images in the batch that are already finished;

  Refill deleted images in the batch using those from D_{RAW};

  Apply one step of operation to the images: $B' = \text{Actor}(B)$;

  $\theta_1 \leftarrow \theta_1 + \alpha_\theta \nabla_{\theta_1} J(\pi_\theta)$;

  $\theta_2 \leftarrow \theta_2 + \alpha_\theta \nabla_{\theta_2} J(\pi_\theta)$;

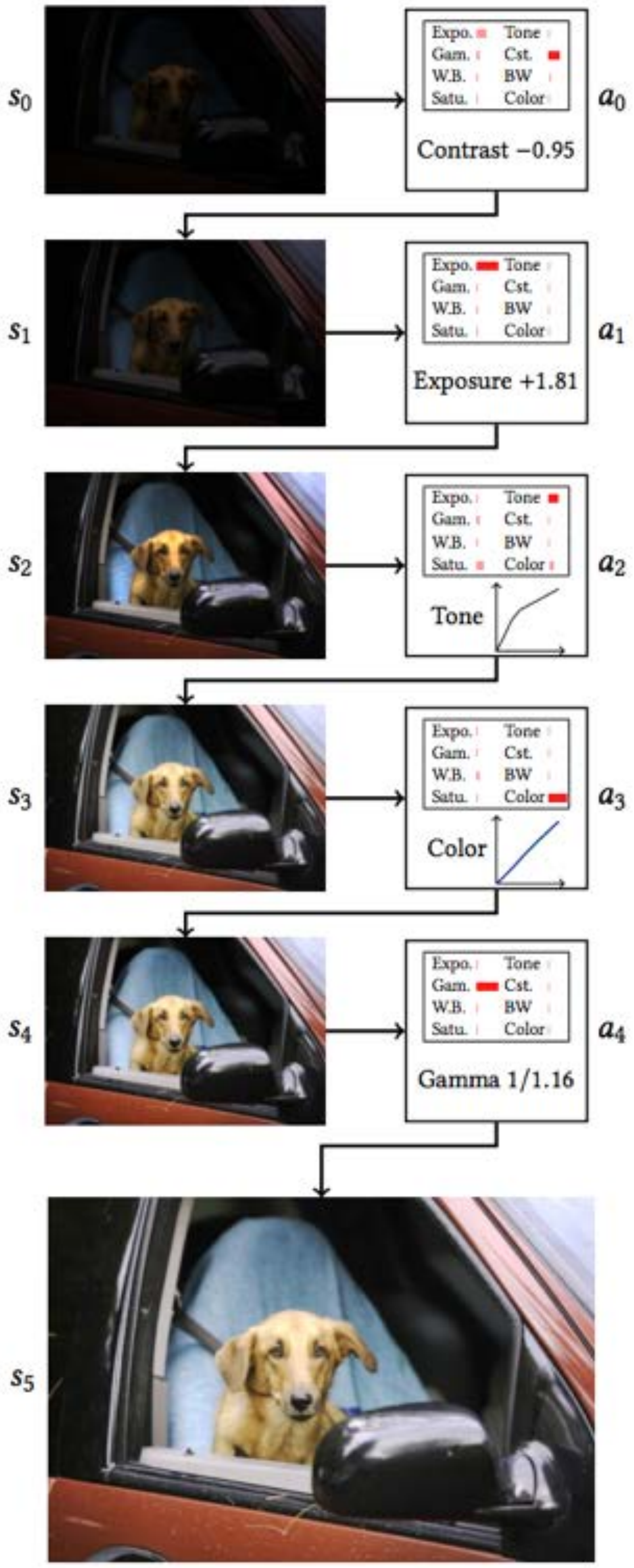
  $v \leftarrow v - \alpha_v \nabla_v L_v$;

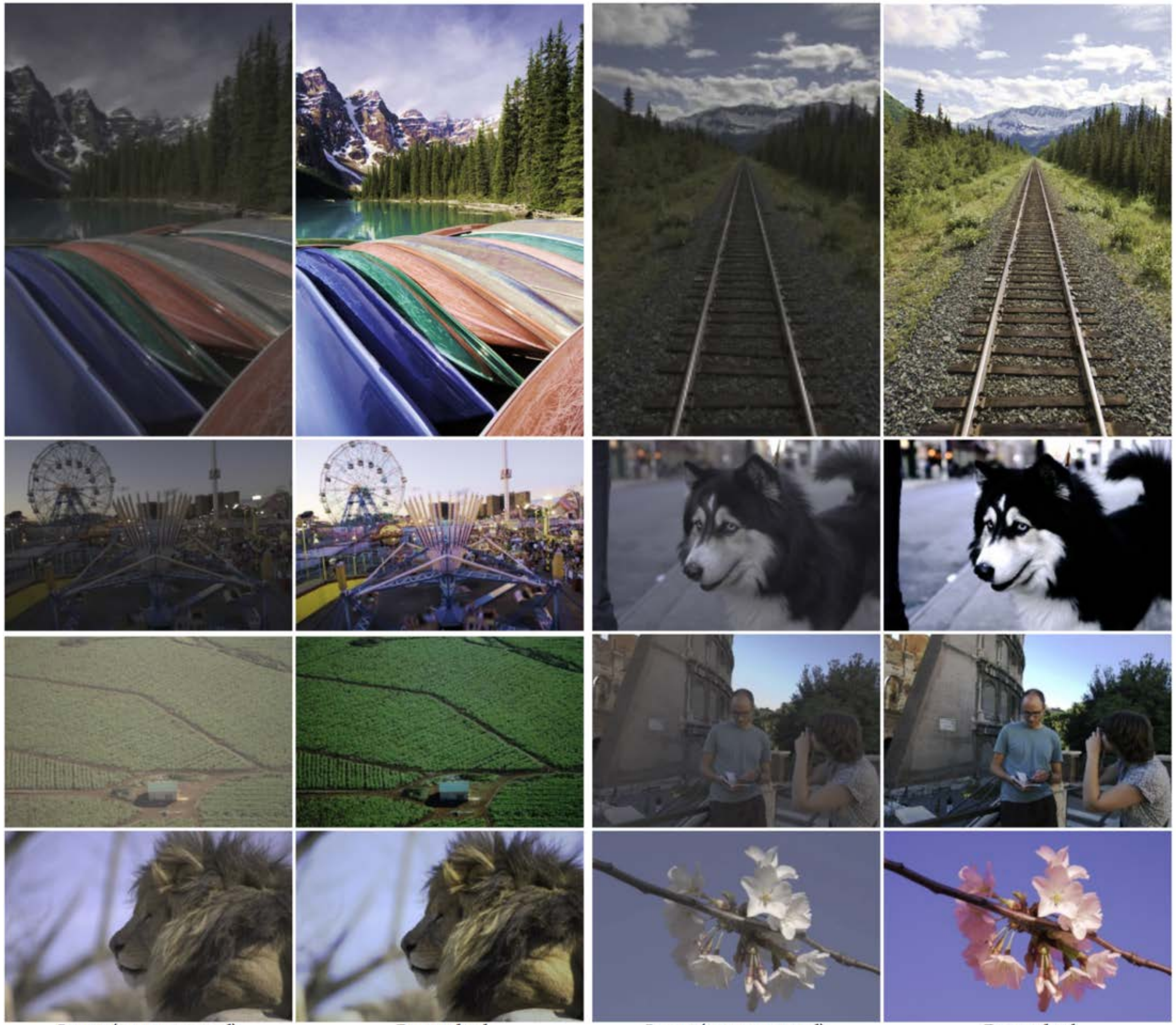
  Put new images B' back into the trajectory buffer;

end

Results

Stylized (500px artist A) Stylized (500px artist B)



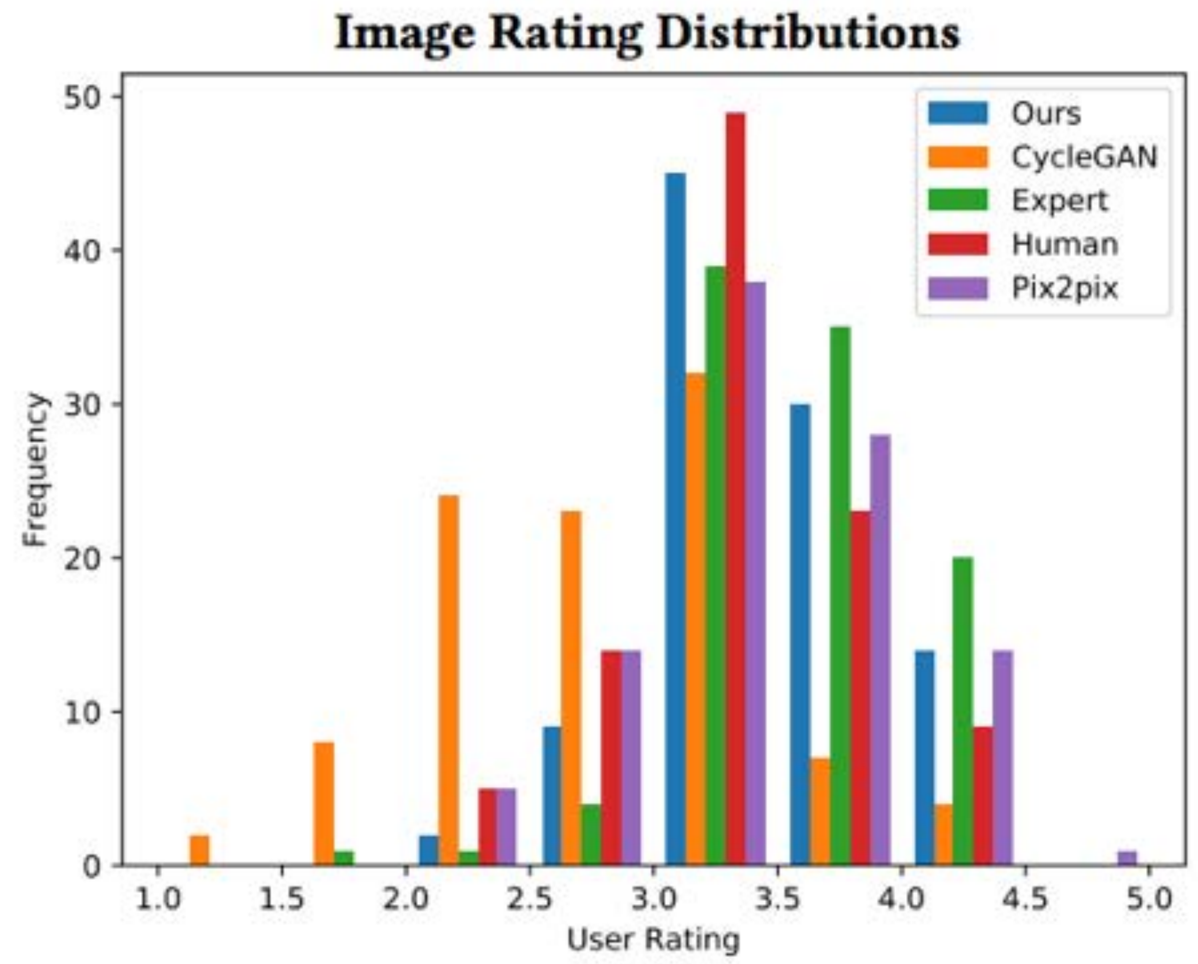


Input (tone-mapped) Retouched Input (tone-mapped) Retouched

Comparisons with deconvolution-based methods

- **Higher quality, resolution**

Approach	Histogram Intersection			AMT
	Luminance	Contrast	Saturation	User Rating
Ours	71.3%	83.7%	69.7%	3.43
CycleGAN	61.4%	71.1%	82.6%	2.47
Pix2pix	92.4%	83.3%	86.5%	3.37
Human	-	-	-	3.30
Expert C	100%	100%	100%	3.66



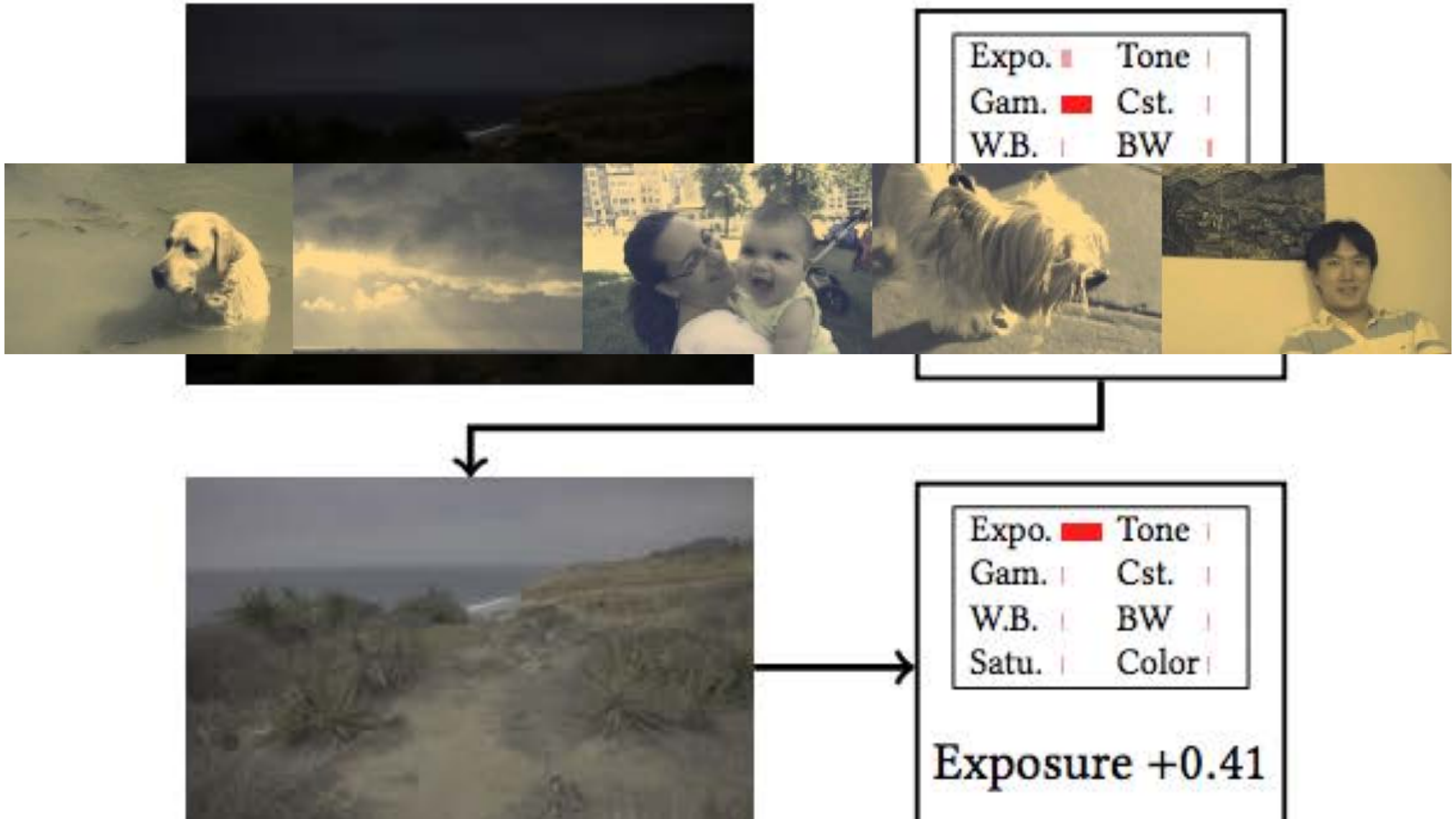
Input **CycleGAN** **Pix2pix** **Ours**

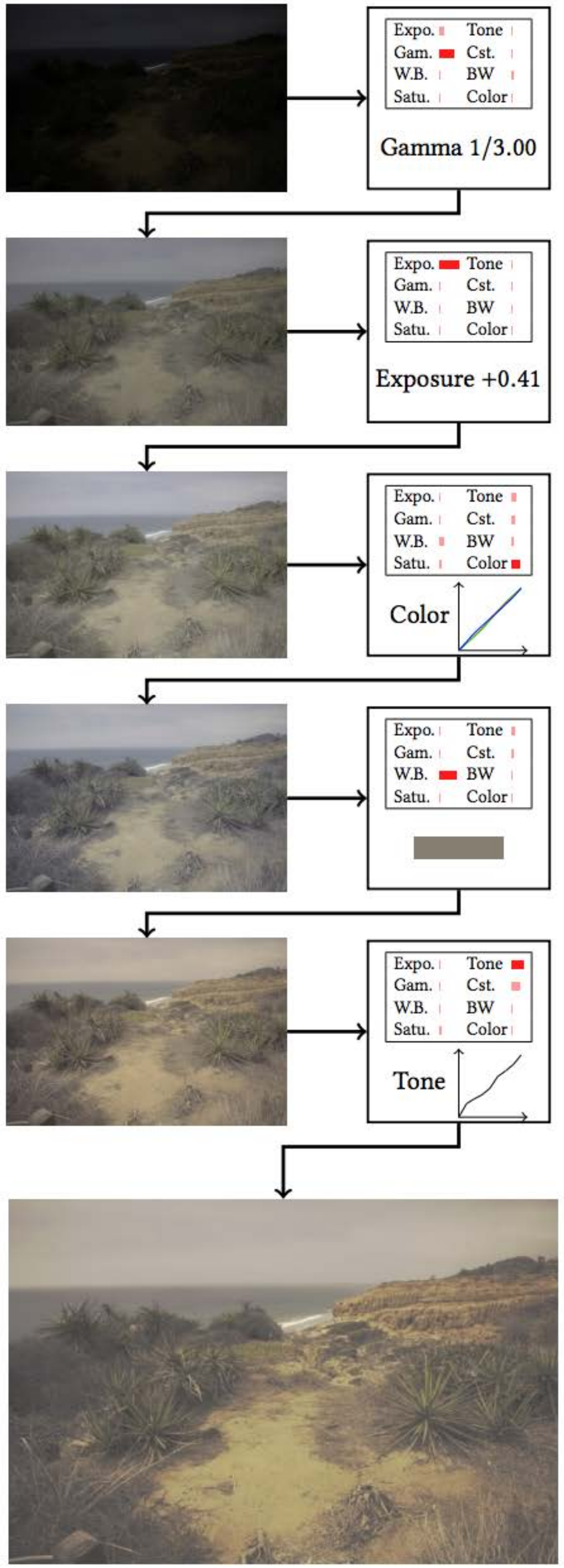
Zoom-in views

An "Infinite-Resolution" GAN

An “Infinite-Resolution” GAN

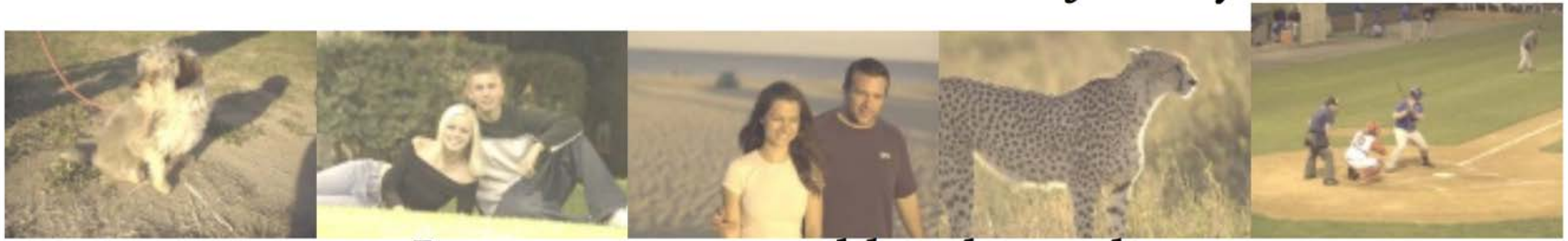
Reverse Engineering






```
# Step 1: Gamma
image = image ** (1 / 3.0)
# Step 2: Exposure
image = image / image.mean() * 0.6
# Step 3: Boost blue shadow
blue_shadow = image[:, :, 2] < 0.5
blue = image[:, :, 2]
blue = blue_shadow * (blue * 2) ** 0.7 / 2 + blue * (1 - blue_shadow)
image[:, :, 2] = blue
# Step 4: White balance
image = image * np.array((1.055, 0.984, 0.886)).reshape((1, 1, 3))
# Step 5: Boost shadow
shadow = image < 0.33
image = ((image * shadow * 3) ** 0.8 / 3) + image * (1 - shadow)
```

Code based on the learned trajectory



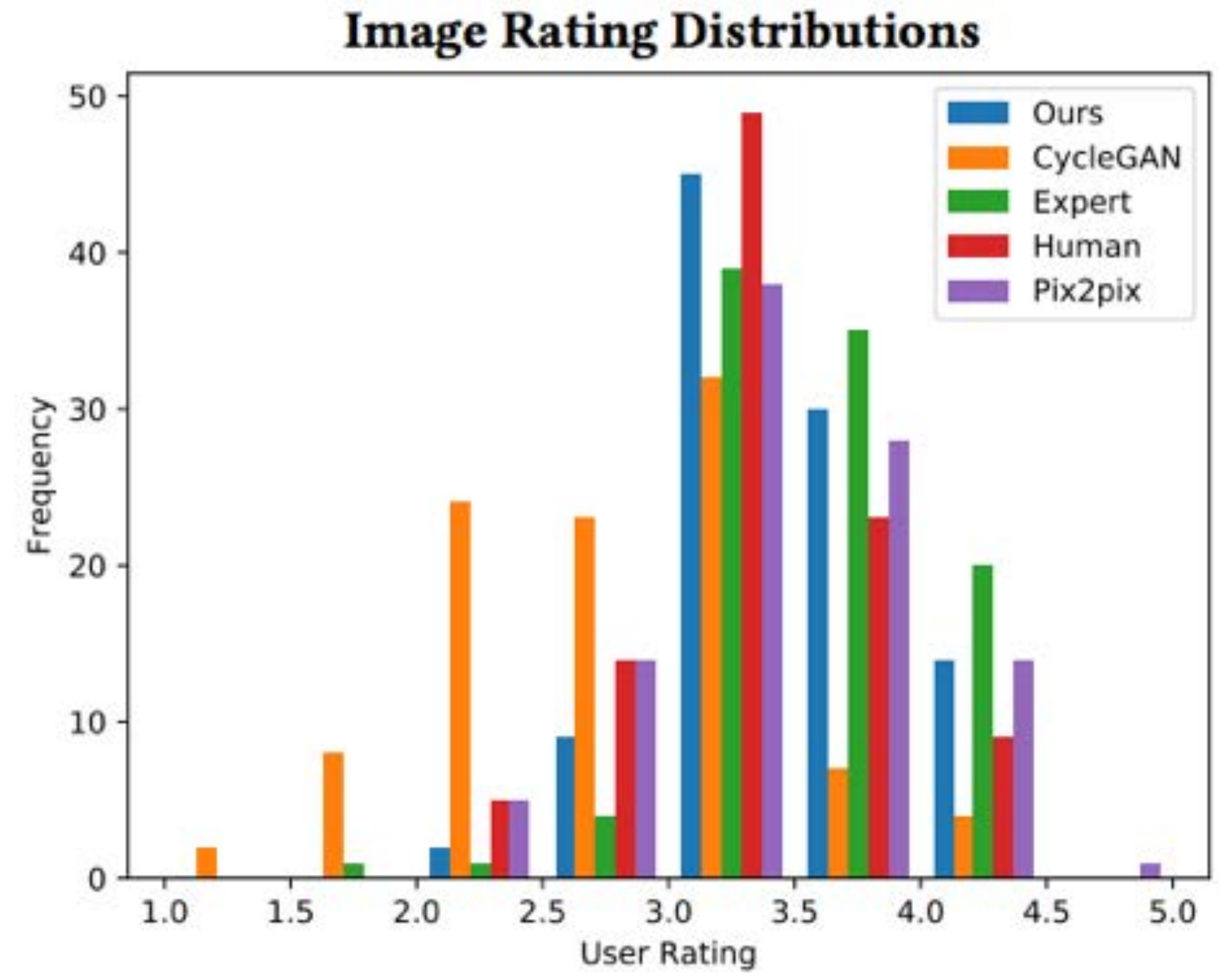
Images generated by the code

Images generated by the black-box filter

Summary: A White-box Framework

- **A learnable model for photo post-processing**
  - Resolution independent
  - Content preserving
    - No need for cycle-consistency
  - Human-understandable
  - "Reverse-engineering"
- **RL+GAN for optimisation**
- **What's next?**
  - More robust learning
  - Better face?
- **Open-source: https://github.com/yuanming-hu/exposure**

Approach	Histogram Intersection			AMT
	Luminance	Contrast	Saturation	User Rating
Ours	71.3%	83.7%	69.7%	3.43
CycleGAN	61.4%	71.1%	82.6%	2.47
Pix2pix	92.4%	83.3%	86.5%	3.37
Human	-	-	-	3.30
Expert C	100%	100%	100%	3.66

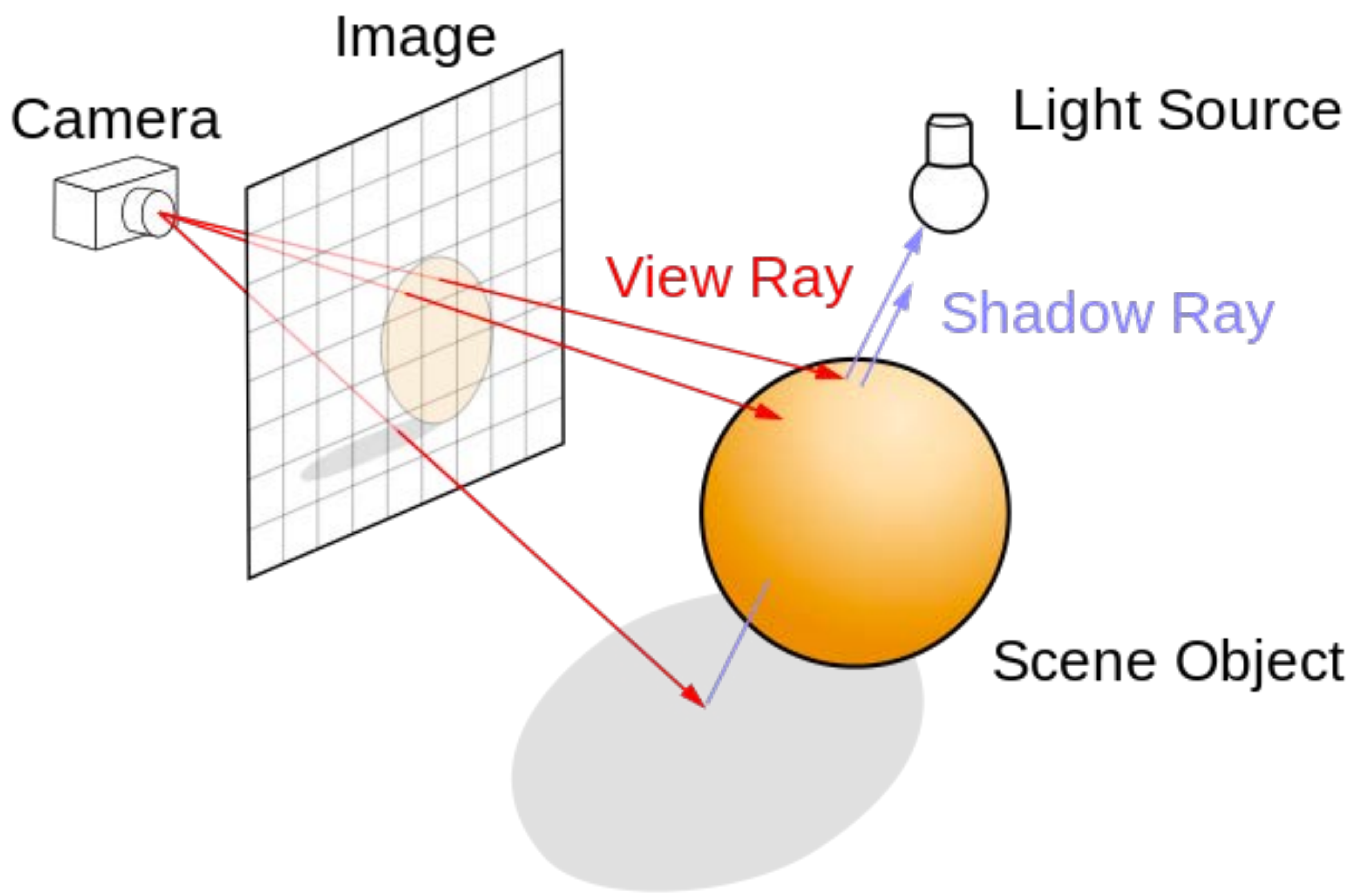


Part II
Taichi: An Open-Source Computer Graphics Library

Yuanming Hu, MIT CSAIL

http://taichi.graphics/

Your amazing ray tracer




```
float output[1920][1080][3]
```

How to display this image on screen?
How to save this image on disk?
How to …?

(Fundamentals of Computer Graphics, Course Website)

(Students' Feedbacks)

学生留言

Q: 光线跟踪结果如何显示?

A: 部分同学反映对GUI编程不熟悉。我们推荐是用下面的一个库：OpenCV。 推荐原因：简单易学，使用方便，读入图片并显示只需要20多行代码。在编程实现过程中，光线跟踪算法的主要操作就是设定输出图像上每个pixel的值，请大家把精力专注在算法部分。

Q: How can I display the image rendered by my ray tracer?
A: …We recommend using the library **OpenCV**. Reason: OpenCV is easy to learn and use. With only 20 lines of code you can read and display an image…. Please focus your time on implementing the ray tracer itself.

OpenCV (Open Source Computer **Vision** Library)

We do not even have a light-weight library to programmatically display an image.

Don't we have such a library?

学生留言

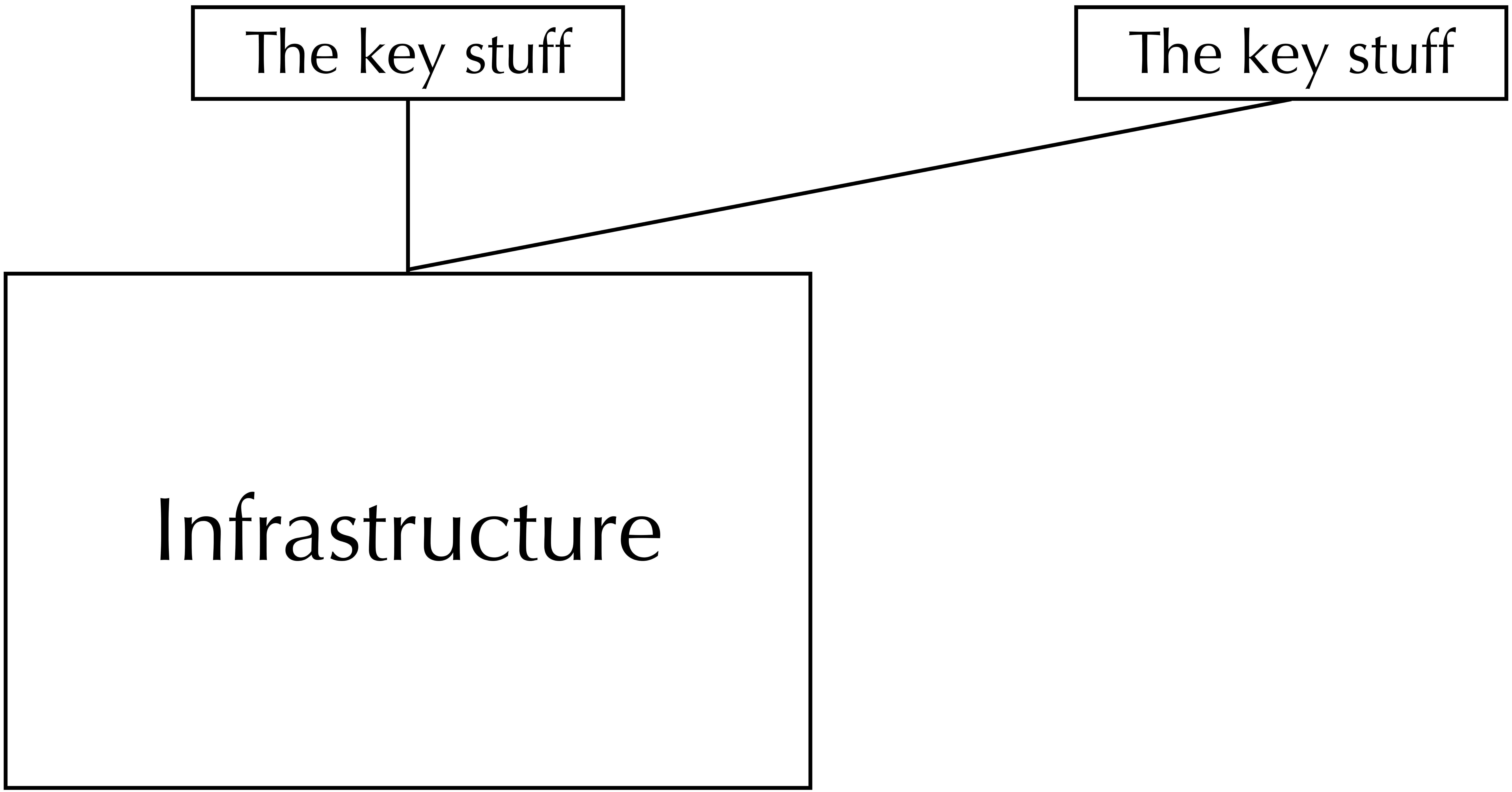
Q: 光线跟踪结果如何显示?

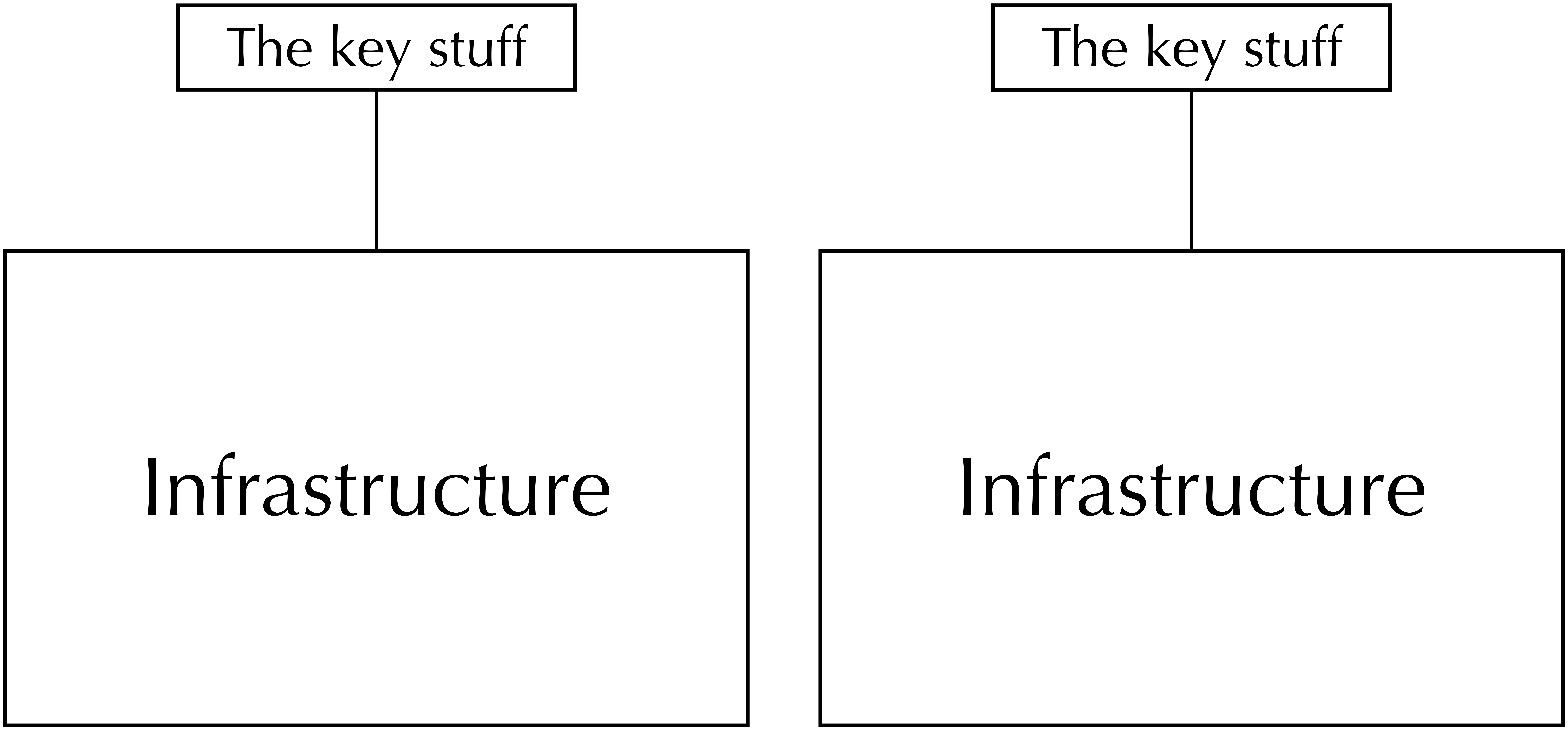
A: 部分同学反映对GUI编程不熟悉。我们推荐是用下面的一个库：OpenCV。 推荐原因：简单易学，使用方便，读入图片并显示只需要20多行代码。在编程实现过程中，光线跟踪算法的主要操作就是设定输出图像上每个pixel的值，请大家把精力专注在算法部分。

OpenGL? Qt? SDL?
Unity?

Don't we have such a library?

- **Rendering:** Mitsuba [Jakob 2010], PBRT [Pharr et al. 2016], Lightmetrica [Otsu 2015], POV-Ray [Buck and Collins 2004] …
- **Geometry processing:** libigl [Jacobson et al. 2013], MeshLab [Cignoni et al. 2008], CGAL [Fabri and Pion 2009] …
- **Simulation:** Bullet [Coumans et al. 2013], ODE [Smith et al. 2005], ArcSim [Narain et al. 2004], VegaFEM [Sin et al. 2013], MantaFlow [Thuerey and Pfa 2017], Box2D [Cao 2011], PhysBAM [Dubey et al. 2011], SPlisHSPlasH [Bender et al. 2016] …
- Unfortunately, more frequently we need to build our own system **(low-level engineering)** instead of reusing **(at a high level)** the aforementioned libraries reuse


The key stuff
The key stuff
Infrastructure


The key stuff
Infrastructure
The key stuff
Infrastructure


CNN tech
YouTube | SpaceX

Reusability: “I can’t even build it.”

[Graphics-students] [Graphics] Anyone have experience with compiling CGAL on Windows? Inbox x

[redacted]
to Graphics

Jan 2

Tried CGAL 4.11 this morning on Windows 10 + Visual Studio 2015 + Boost 1.59. Turned out to be a huge pain. If anyone in our group has tried this before could I come over to ask you a few questions? Thanks!

Thanks,
[redacted]

Yuanming Hu <yuanming@mit.edu>
to [redacted] Graphics

I did this and I agree that the experience was terrible. Maybe we can talk about it later in the afternoon.

[redacted]@csail.mit.edu>
to Yuanming, Graphics

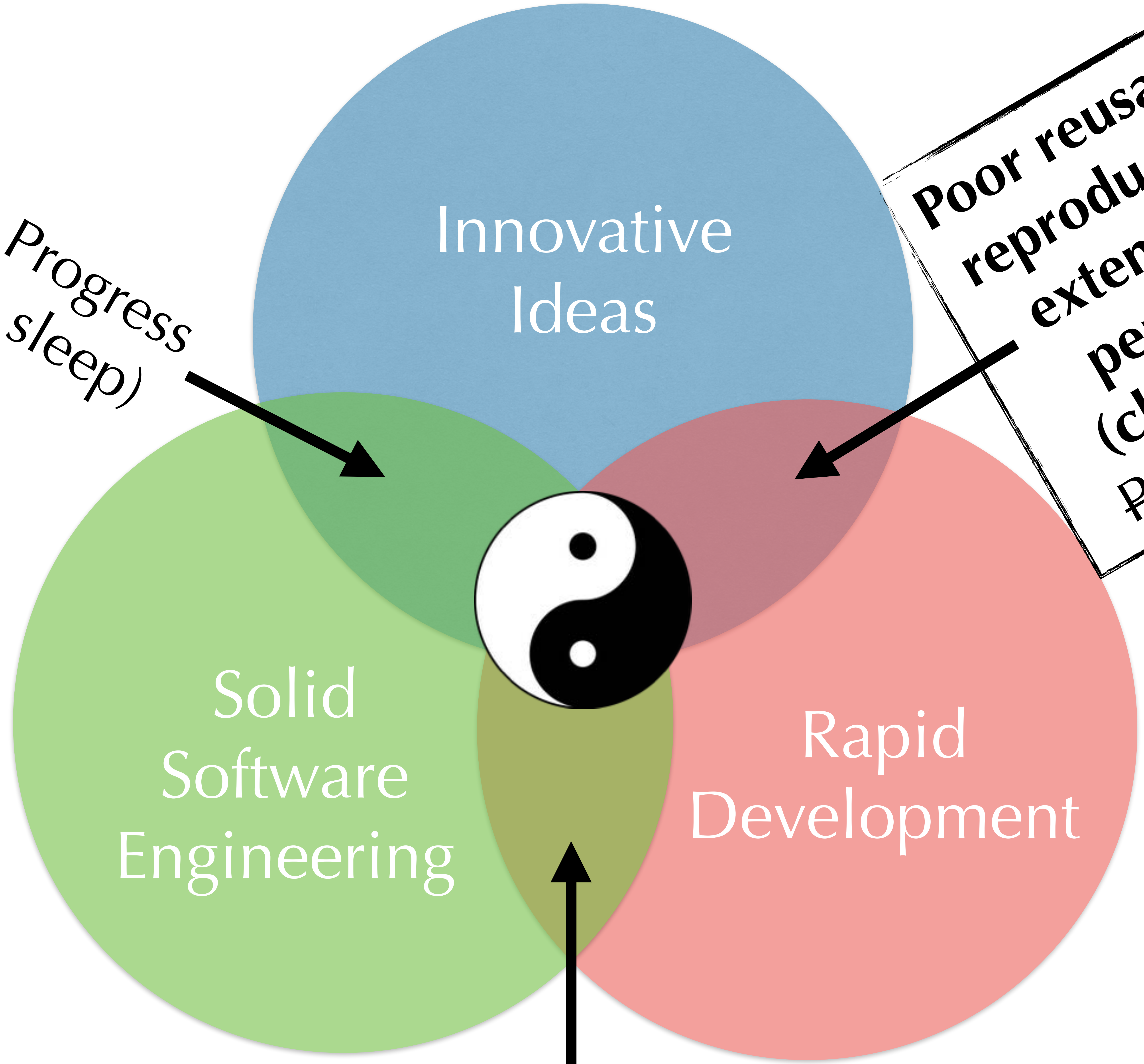
Resolved. Yuanming is a genius.

Question: Why do you have to be a “genius” just to compile a software??

The trade-off…

Reusable infrastructure that provides good software engineering (for free)

Slow Progress
(or no sleep)



Poor reusability or reproducibility or extensibility or performance (closed-source)
~~People's choice?~~

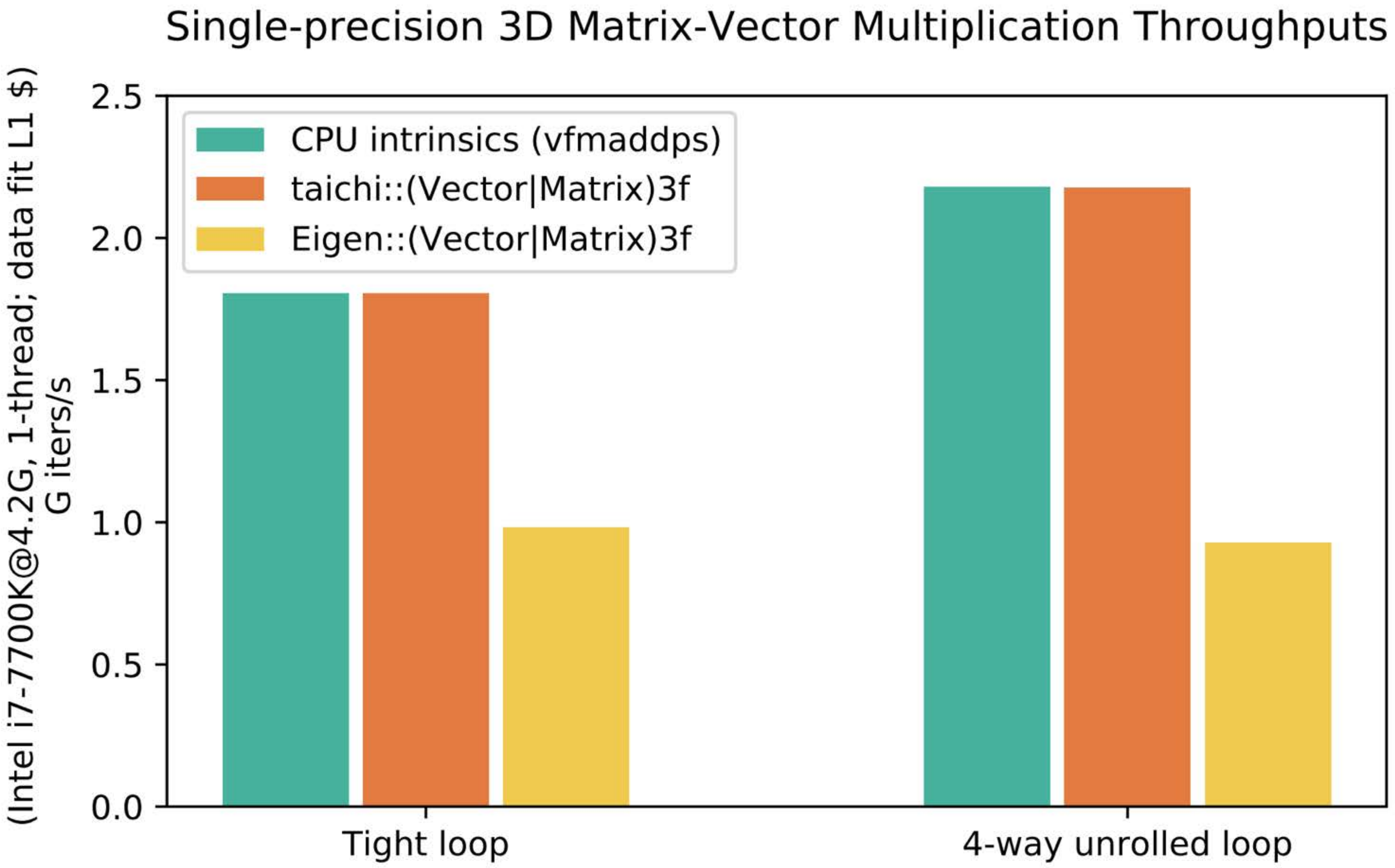
Hard to achieve high novelty
(i.e., hard to have your paper accepted)

Building a Reusable Infrastructure

- **Accessible, portable, extensible, and high-performance infrastructure, that is reusable and tailored for researchers in computer graphics-related fields**
- **Easy to achieve some of the features, but having them all is hard.**
- **Reusability** is especially hard.
- **More discussions: https://arxiv.org/abs/1804.09293**

“Why do we need something tailored for graphics? Why not just reuse **Boost** or **Eigen**?”

Eigen?



Single-precision 3D Matrix-Vector Multiplication Throughputs
G iters/s
(Intel i7-7700K@4.2G, 1-thread; data fit L1 $)
2.5
2.0
1.5
1.0
0.5
0.0
CPU intrinsics (vfmaddps)
taichi::(Vector|Matrix)3f
Eigen::(Vector|Matrix)3f
Tight loop
4-way unrolled loop

“Is it possible to get performance and user-friendliness simultaneously?”

The cost of performance

"C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off" - Bjarne Stroustrup
http://www.stroustrup.com/bs_faq.html#really-say-that

"Heisenbugs"

Portability
(E.g. how to create a folder using portable code? No answer until C++17 (std::filesystem))

Complexity:
SFINAE
RAII
RTTI
ABI

Long Compilation Time

Hard-to-read error message


pybind11

What do we need Taichi for?

- **Research**
- **Education**
  - I.e., do not let graphics students start by using OpenCV
- **Propagation**
  - Elegant ideas should have simple code
  - which can be implemented easily
- **Deployment**

Borrow some efforts from the industry (to benefit the academia)

An infrastructure for graphics (commercial) deployment

An code-base for graphics education & propagation (`#include "taichi.h"`)

~~A library of SIGGRAPH papers~~

An infrastructure for computer graphics research

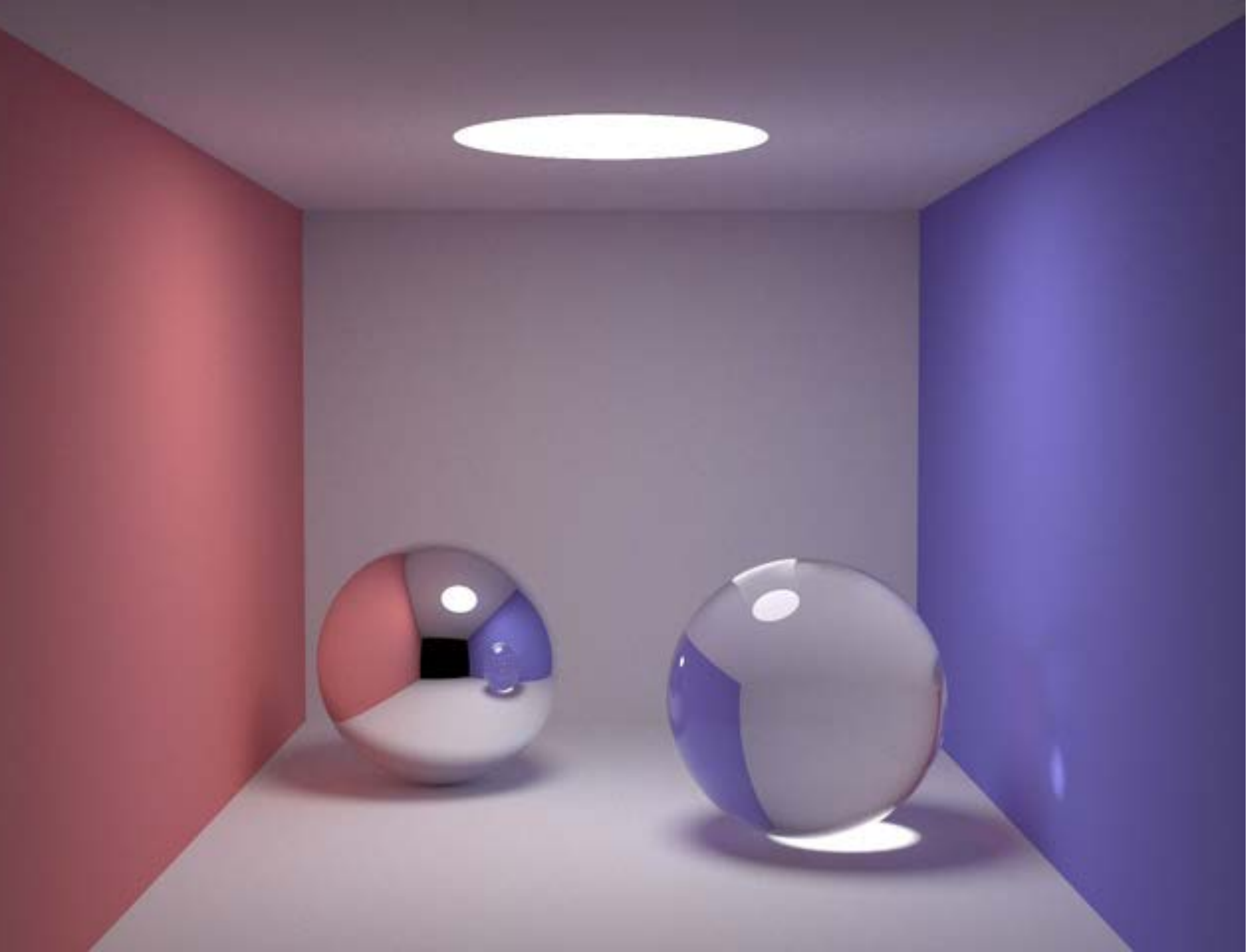
2016 2017 2018 2019 2020 2021

2019: doc, testing ready

Reproducibility

- **Good research should be easily reproducible**
  - Hard-to-reproduce projects intrinsically set barriers for people to follow up
  - … and hinder further developments
  - … even within a group
- **Ease of implementation greatly helps reproducibility**
  - The core idea should be easily reproduced
  - Maybe no need for performance

A 99 line topology optimization code written in Matlab | SpringerLink
https://link.springer.com/article/10.1007/s001580050176
by O Sigmund - 2001 - Cited by 1335 - Related articles



```cpp
#include <math.h>   // smallpt, a Path Tracer by Kevin Beason, 2008
#include <stdlib.h> // Make : g++ -O3 -fopenmp smallpt.cpp -o smallpt
#include <stdio.h>  //        Remove "-fopenmp" for g++ version < 4.2
struct Vec {        // Usage: time ./smallpt 5000 && xv image.ppm
  double x, y, z;                  // position, also color (r,g,b)
  Vec(double x_=0, double y_=0, double z_=0){ x=x_; y=y_; z=z_; }
  Vec operator+(const Vec &b) const { return Vec(x+b.x,y+b.y,z+b.z); }
  Vec operator-(const Vec &b) const { return Vec(x-b.x,y-b.y,z-b.z); }
  Vec operator*(double b) const { return Vec(x*b,y*b,z*b); }
  Vec mult(const Vec &b) const { return Vec(x*b.x,y*b.y,z*b.z); }
  Vec& norm(){ return *this = *this * (1/sqrt(x*x+y*y+z*z)); }
  double dot(const Vec &b) const { return x*b.x+y*b.y+z*b.z; } // cross:
  Vec operator%(Vec&b){return Vec(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);}
};
struct Ray { Vec o, d; Ray(Vec o_, Vec d_) : o(o_), d(d_) {} };
enum Refl_t { DIFF, SPEC, REFR };  // material types, used in radiance()
struct Sphere {
  double rad;       // radius
  Vec p, e, c;      // position, emission, color
  Refl_t refl;      // reflection type (DIFFuse, SPECular, REFRactive)
  Sphere(double rad_, Vec p_, Vec e_, Vec c_, Refl_t refl_):
    rad(rad_), p(p_), e(e_), c(c_), refl(refl_) {}
  double intersect(const Ray &r) const { // returns distance, 0 if nohit
    Vec op = p-r.o; // Solve t^2*d.d + 2*t*(o-p).d + (o-p).(o-p)-R^2 = 0
    double t, eps=1e-4, b=op.dot(r.d), det=b*b-op.dot(op)+rad*rad;
    if (det<0) return 0; else det=sqrt(det);
    return (t=b-det)>eps ? t : ((t=b+det)>eps ? t : 0);
  }
};
Sphere spheres[] = {//Scene: radius, position, emission, color, material
  Sphere(1e5, Vec( 1e5+1,40.8,81.6), Vec(),Vec(.75,.25,.25),DIFF),//Left
  Sphere(1e5, Vec(-1e5+99,40.8,81.6),Vec(),Vec(.25,.25,.75),DIFF),//Rght
  Sphere(1e5, Vec(50,40.8, 1e5),     Vec(),Vec(.75,.75,.75),DIFF),//Back
  Sphere(1e5, Vec(50,40.8,-1e5+170), Vec(),Vec(),           DIFF),//Frnt
  Sphere(1e5, Vec(50, 1e5, 81.6),    Vec(),Vec(.75,.75,.75),DIFF),//Botm
  Sphere(1e5, Vec(50,-1e5+81.6,81.6),Vec(),Vec(.75,.75,.75),DIFF),//Top
  Sphere(16.5,Vec(27,16.5,47),       Vec(),Vec(1,1,1)*.999, SPEC),//Mirr
  Sphere(16.5,Vec(73,16.5,78),       Vec(),Vec(1,1,1)*.999, REFR),//Glas
  Sphere(600, Vec(50,681.6-.27,81.6),Vec(12,12,12),  Vec(), DIFF) //Lite
};
inline double clamp(double x){ return x<0 ? 0 : x>1 ? 1 : x; }
inline int toInt(double x){ return int(pow(clamp(x),1/2.2)*255+.5); }
inline bool intersect(const Ray &r, double &t, int &id){
  double n=sizeof(spheres)/sizeof(Sphere), d, inf=t=1e20;
  for(int i=int(n);i--;) if((d=spheres[i].intersect(r))&&d<t){t=d;id=i;}
  return t<inf;
}
Vec radiance(const Ray &r, int depth, unsigned short *Xi){
  double t;                               // distance to intersection
  int id=0;                               // id of intersected object
  if (!intersect(r, t, id)) return Vec(); // if miss, return black
  const Sphere &obj = spheres[id];        // the hit object
  Vec x=r.o+r.d*t, n=(x-obj.p).norm(), nl=n.dot(r.d)<0?n:n*-1, f=obj.c;
  double p = f.x>f.y && f.x>f.z ? f.x : f.y>f.z ? f.y : f.z; // max refl
  if (++depth>5) if (erand48(Xi)<p) f=f*(1/p); else return obj.e; //R.R.
  if (obj.refl == DIFF){                  // Ideal DIFFUSE reflection
    double r1=2*M_PI*erand48(Xi), r2=erand48(Xi), r2s=sqrt(r2);
    Vec w=nl, u=((fabs(w.x)>.1?Vec(0,1):Vec(1))%w).norm(), v=w%u;
    Vec d = (u*cos(r1)*r2s + v*sin(r1)*r2s + w*sqrt(1-r2)).norm();
    return obj.e + f.mult(radiance(Ray(x,d),depth,Xi));
  } else if (obj.refl == SPEC)            // Ideal SPECULAR reflection
    return obj.e + f.mult(radiance(Ray(x,r.d-n*2*n.dot(r.d)),depth,Xi));
  Ray reflRay(x, r.d-n*2*n.dot(r.d));     // Ideal dielectric REFRACTION
  bool into = n.dot(nl)>0;                // Ray from outside going in?
  double nc=1, nt=1.5, nnt=into?nc/nt:nt/nc, ddn=r.d.dot(nl), cos2t;
  if ((cos2t=1-nnt*nnt*(1-ddn*ddn))<0)    // Total internal reflection
    return obj.e + f.mult(radiance(reflRay,depth,Xi));
  Vec tdir = (r.d*nnt - n*((into?1:-1)*(ddn*nnt+sqrt(cos2t)))).norm();
  double a=nt-nc, b=nt+nc, R0=a*a/(b*b), c = 1-(into?-ddn:tdir.dot(n));
  double Re=R0+(1-R0)*c*c*c*c*c,Tr=1-Re,P=.25+.5*Re,RP=Re/P,TP=Tr/(1-P);
  return obj.e + f.mult(depth>2 ? (erand48(Xi)<P ?   // Russian roulette
    radiance(reflRay,depth,Xi)*RP:radiance(Ray(x,tdir),depth,Xi)*TP) :
    radiance(reflRay,depth,Xi)*Re+radiance(Ray(x,tdir),depth,Xi)*Tr);
}
int main(int argc, char *argv[]){
  int w=1024, h=768, samps = argc==2 ? atoi(argv[1])/4 : 1; // # samples
  Ray cam(Vec(50,52,295.6), Vec(0,-0.042612,-1).norm()); // cam pos, dir
  Vec cx=Vec(w*.5135/h), cy=(cx%cam.d).norm()*.5135, r, *c=new Vec[w*h];
#pragma omp parallel for schedule(dynamic, 1) private(r)       // OpenMP
  for (int y=0; y<h; y++){                       // Loop over image rows
    fprintf(stderr,"\rRendering (%d spp) %5.2f%%",samps*4,100.*y/(h-1));
    for (unsigned short x=0, Xi[3]={0,0,y*y*y}; x<w; x++)   // Loop cols
      for (int sy=0, i=(h-y-1)*w+x; sy<2; sy++)     // 2x2 subpixel rows
        for (int sx=0; sx<2; sx++, r=Vec()){        // 2x2 subpixel cols
          for (int s=0; s<samps; s++){
            double r1=2*erand48(Xi), dx=r1<1 ? sqrt(r1)-1: 1-sqrt(2-r1);
            double r2=2*erand48(Xi), dy=r2<1 ? sqrt(r2)-1: 1-sqrt(2-r2);
            Vec d = cx*( ( (sx+.5 + dx)/2 + x)/w - .5) +
                    cy*( ( (sy+.5 + dy)/2 + y)/h - .5) + cam.d;
            r = r + radiance(Ray(cam.o+d*140,d.norm()),0,Xi)*(1./samps);
          } // Camera rays are pushed ^^^^^ forward to start in interior
          c[i] = c[i] + Vec(clamp(r.x),clamp(r.y),clamp(r.z))*.25;
        }
  }
  FILE *f = fopen("image.ppm", "w");         // Write image to PPM file.
  fprintf(f, "P3\n%d %d\n%d\n", w, h, 255);
  for (int i=0; i<w*h; i++)
    fprintf(f,"%d %d %d ", toInt(c[i].x), toInt(c[i].y), toInt(c[i].z));
}
```

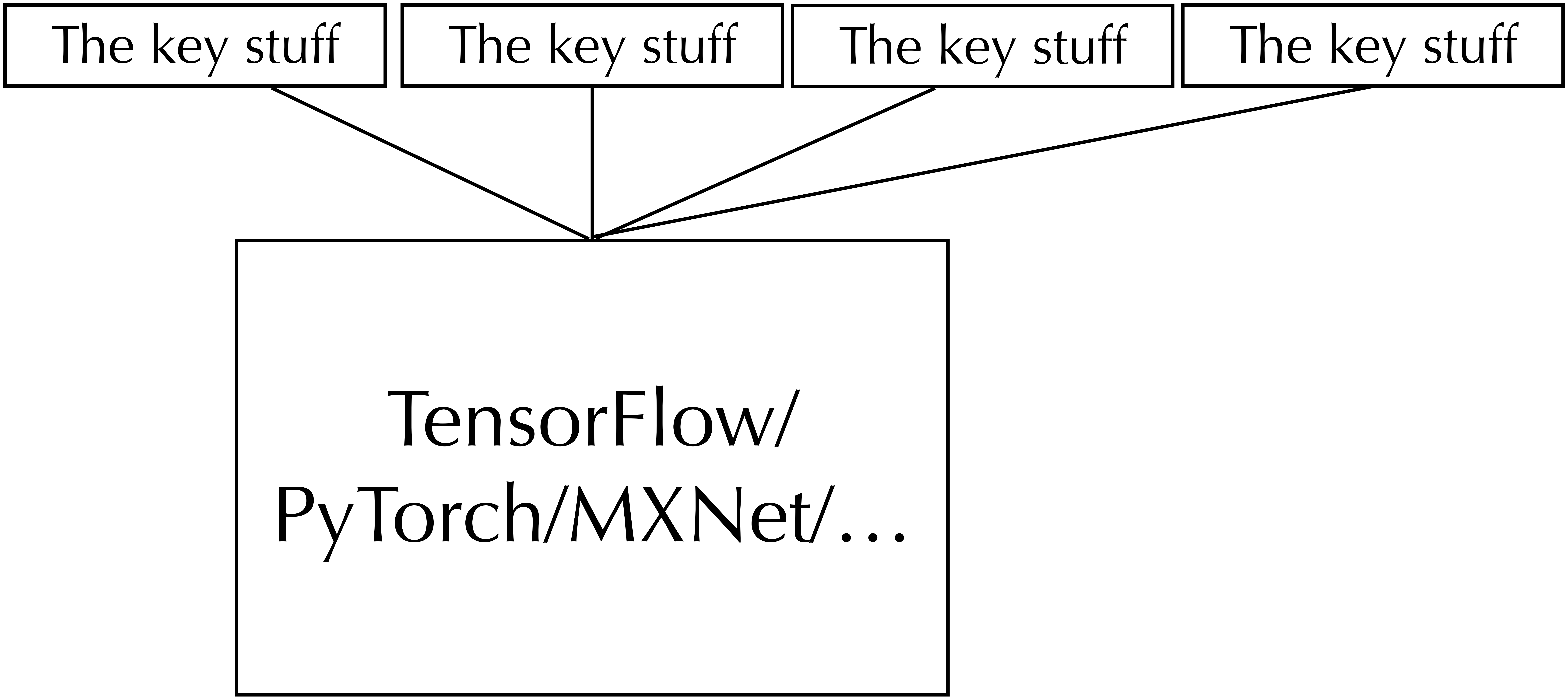
```
#include <math.h>   // smallpt, a Path Tracer by Kevin Beason, 2008
#include <stdlib.h> // Make : g++ -O3 -fopenmp smallpt.cpp -o smallpt
#include <stdio.h>  //        Remove "-fopenmp" for g++ version < 4.2
struct Vec {        // Usage: time ./smallpt 5000 && xv image.ppm
  double x, y, z;                  // position, also color (r,g,b)
  Vec(double x_=0, double y_=0, double z_=0){ x=x_; y=y_; z=z_; }
  Vec operator+(const Vec &b) const { return Vec(x+b.x,y+b.y,z+b.z); }
  Vec operator-(const Vec &b) const { return Vec(x-b.x,y-b.y,z-b.z); }
  Vec operator*(double b) const { return Vec(x*b,y*b,z*b); }
  Vec mult(const Vec &b) const { return Vec(x*b.x,y*b.y,z*b.z); }
  Vec& norm(){ return *this = *this * (1/sqrt(x*x+y*y+z*z)); }
  double dot(const Vec &b) const { return x*b.x+y*b.y+z*b.z; } // cross:
  Vec operator%(Vec&b){return Vec(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);}
};
struct Ray { Vec o, d; Ray(Vec o_, Vec d_) : o(o_), d(d_) {} };
enum Refl_t { DIFF, SPEC, REFR };  // material types, used in radiance()
struct Sphere {
  double rad;       // radius
  Vec p, e, c;      // position, emission, color
  Refl_t refl;      // reflection type (DIFFuse, SPECular, REFRactive)
  Sphere(double rad_, Vec p_, Vec e_, Vec c_, Refl_t refl_):
    rad(rad_), p(p_), e(e_), c(c_), refl(refl_) {}
  double intersect(const Ray &r) const { // returns distance, 0 if nohit
    Vec op = p-r.o; // Solve t^2*d.d + 2*t*(o-p).d + (o-p).(o-p)-R^2 = 0
    double t, eps=1e-4, b=op.dot(r.d), det=b*b-op.dot(op)+rad*rad;
    if (det<0) return 0; else det=sqrt(det);
    return (t=b-det)>eps ? t : ((t=b+det)>eps ? t : 0);
  }
};
Sphere spheres[] = {//Scene: radius, position, emission, color, material
  Sphere(1e5, Vec( 1e5+1,40.8,81.6), Vec(),Vec(.75,.25,.25),DIFF),//Left
  Sphere(1e5, Vec(-1e5+99,40.8,81.6),Vec(),Vec(.25,.25,.75),DIFF),//Rght
  Sphere(1e5, Vec(50,40.8, 1e5),     Vec(),Vec(.75,.75,.75),DIFF),//Back
  Sphere(1e5, Vec(50,40.8,-1e5+170), Vec(),Vec(),           DIFF),//Frnt
  Sphere(1e5, Vec(50, 1e5, 81.6),    Vec(),Vec(.75,.75,.75),DIFF),//Botm
  Sphere(1e5, Vec(50,-1e5+81.6,81.6),Vec(),Vec(.75,.75,.75),DIFF),//Top
  Sphere(16.5,Vec(27,16.5,47),       Vec(),Vec(1,1,1)*.999, SPEC),//Mirr
  Sphere(16.5,Vec(73,16.5,78),       Vec(),Vec(1,1,1)*.999, REFR),//Glas
  Sphere(600, Vec(50,681.6-.27,81.6),Vec(12,12,12),  Vec(), DIFF) //Lite
};
inline double clamp(double x){ return x<0 ? 0 : x>1 ? 1 : x; }
inline int toInt(double x){ return int(pow(clamp(x),1/2.2)*255+.5); }
inline bool intersect(const Ray &r, double &t, int &id){
  double n=sizeof(spheres)/sizeof(Sphere), d, inf=t=1e20;
  for(int i=int(n);i--;) if((d=spheres[i].intersect(r))&&d<t){t=d;id=i;}
  return t<inf;
}
Vec radiance(const Ray &r, int depth, unsigned short *Xi){
  double t;                               // distance to intersection
  int id=0;                               // id of intersected object
  if (!intersect(r, t, id)) return Vec(); // if miss, return black
  const Sphere &obj = spheres[id];        // the hit object
  Vec x=r.o+r.d*t, n=(x-obj.p).norm(), nl=n.dot(r.d)<0?n:n*-1, f=obj.c;
  double p = f.x>f.y && f.x>f.z ? f.x : f.y>f.z ? f.y : f.z; // max refl
  if (++depth>5) if (erand48(Xi)<p) f=f*(1/p); else return obj.e; //R.R.
  if (obj.refl == DIFF){                  // Ideal DIFFUSE reflection
    double r1=2*M_PI*erand48(Xi), r2=erand48(Xi), r2s=sqrt(r2);
    Vec w=nl, u=((fabs(w.x)>.1?Vec(0,1):Vec(1))%w).norm(), v=w%u;
    Vec d = (u*cos(r1)*r2s + v*sin(r1)*r2s + w*sqrt(1-r2)).norm();
    return obj.e + f.mult(radiance(Ray(x,d),depth,Xi));
  } else if (obj.refl == SPEC)            // Ideal SPECULAR reflection
    return obj.e + f.mult(radiance(Ray(x,r.d-n*2*n.dot(r.d)),depth,Xi));
  Ray reflRay(x, r.d-n*2*n.dot(r.d));     // Ideal dielectric REFRACTION
  bool into = n.dot(nl)>0;                // Ray from outside going in?
  double nc=1, nt=1.5, nnt=into?nc/nt:nt/nc, ddn=r.d.dot(nl), cos2t;
  if ((cos2t=1-nnt*nnt*(1-ddn*ddn))<0)    // Total internal reflection
    return obj.e + f.mult(radiance(reflRay,depth,Xi));
  Vec tdir = (r.d*nnt - n*((into?1:-1)*(ddn*nnt+sqrt(cos2t)))).norm();
  double a=nt-nc, b=nt+nc, R0=a*a/(b*b), c = 1-(into?-ddn:tdir.dot(n));
  double Re=R0+(1-R0)*c*c*c*c*c,Tr=1-Re,P=.25+.5*Re,RP=Re/P,TP=Tr/(1-P);
  return obj.e + f.mult(depth>2 ? (erand48(Xi)<P ?   // Russian roulette
    radiance(reflRay,depth,Xi)*RP:radiance(Ray(x,tdir),depth,Xi)*TP) :
    radiance(reflRay,depth,Xi)*Re+radiance(Ray(x,tdir),depth,Xi)*Tr);
}
int main(int argc, char *argv[]){
  int w=1024, h=768, samps = argc==2 ? atoi(argv[1])/4 : 1; // # samples
  Ray cam(Vec(50,52,295.6), Vec(0,-0.042612,-1).norm()); // cam pos, dir
  Vec cx=Vec(w*.5135/h), cy=(cx%cam.d).norm()*.5135, r, *c=new Vec[w*h];
#pragma omp parallel for schedule(dynamic, 1) private(r)       // OpenMP
  for (int y=0; y<h; y++){                       // Loop over image rows
    fprintf(stderr,"\rRendering (%d spp) %5.2f%%",samps*4,100.*y/(h-1));
    for (unsigned short x=0, Xi[3]={0,0,y*y*y}; x<w; x++)   // Loop cols
      for (int sy=0, i=(h-y-1)*w+x; sy<2; sy++)     // 2x2 subpixel rows
        for (int sx=0; sx<2; sx++, r=Vec()){        // 2x2 subpixel cols
          for (int s=0; s<samps; s++){
            double r1=2*erand48(Xi), dx=r1<1 ? sqrt(r1)-1: 1-sqrt(2-r1);
            double r2=2*erand48(Xi), dy=r2<1 ? sqrt(r2)-1: 1-sqrt(2-r2);
            Vec d = cx*( ( (sx+.5 + dx)/2 + x)/w - .5) +
                    cy*( ( (sy+.5 + dy)/2 + y)/h - .5) + cam.d;
            r = r + radiance(Ray(cam.o+d*140,d.norm()),0,Xi)*(1./samps);
          } // Camera rays are pushed ^^^^^ forward to start in interior
          c[i] = c[i] + Vec(clamp(r.x),clamp(r.y),clamp(r.z))*.25;
        }
  }
  FILE *f = fopen("image.ppm", "w");         // Write image to PPM file.
  fprintf(f, "P3\n%d %d\n%d\n", w, h, 255);
  for (int i=0; i<w*h; i++)
    fprintf(f,"%d %d %d ", toInt(c[i].x), toInt(c[i].y), toInt(c[i].z));
}
```

#include <taichi.h>

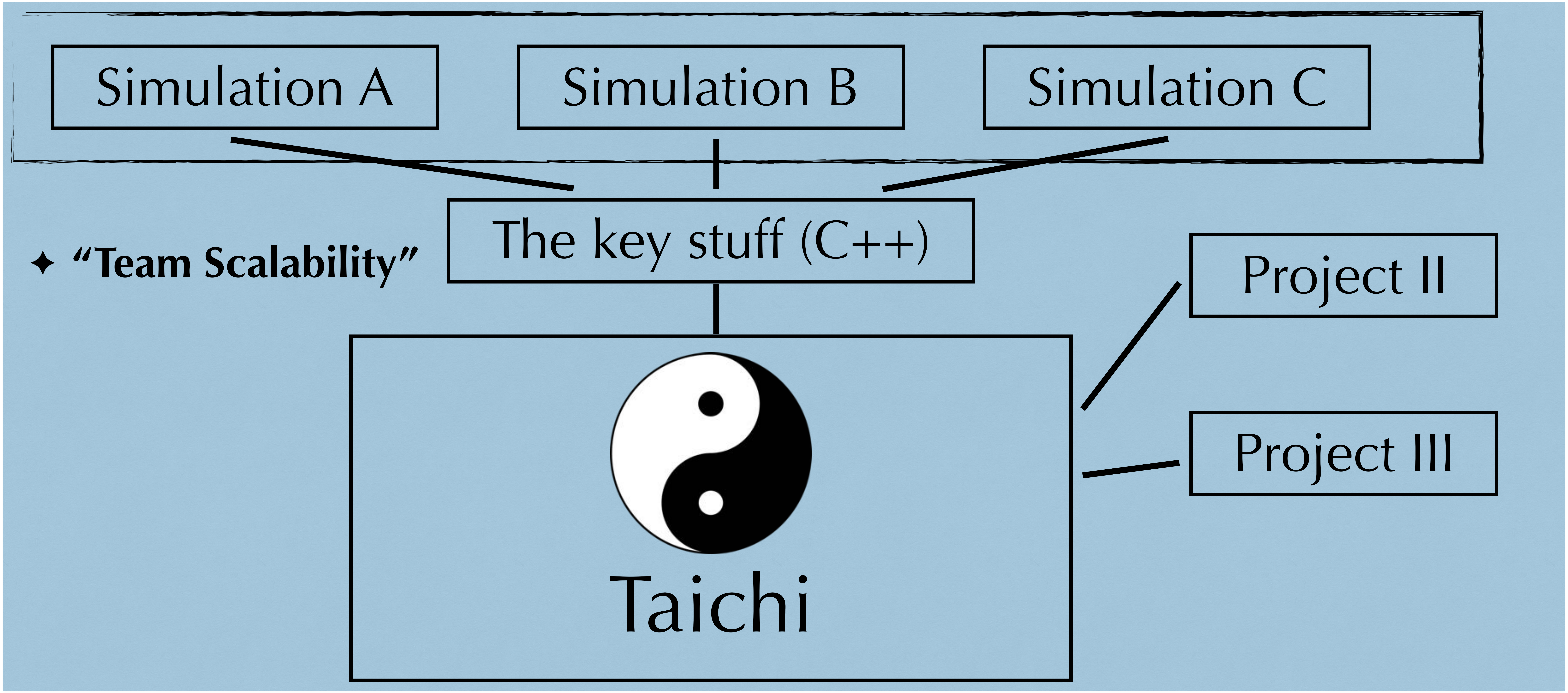
- **88-line implementations**
  - E.g. MLS-MPM
- **Perfectly portable (with GUI!)**
  - Two files are enough for a self-contained demo
  - No need for Makefiles, CMakeLists.txt
  - g++ mpm.cpp -std=c++14 -lX11 -lpthread -O2 -o mpm
  - Portability ensured by taichi.h
- **Not parallelized, but already much faster than Python/ matlab**

```cpp
// The Moving Least Squares Material Point Method in 88 LoC (with comments)
//   To compile:   g++ mpm.cpp -std=c++14 -g -lX11 -lpthread -O2 -o mpm
#include "taichi.h" // Single header version of (a small part of) taichi
using namespace taichi;
const int n = 64 /*grid resolution (cells)*/, window_size = 500;
const real dt = 1e-4_f, frame_dt = 1e-3_f, dx = 1.0_f / n, inv_dx = 1.0_f / dx;
real mass = 1.0_f, vol = 1.0_f; // Particle mass and volume
real hardening = 10, E = 1e4 /* Young's Modulus*/, nu = 0.2 /*Poisson's Ratio*/;
real mu_0 = E/(2*(1+nu)), lambda_0=E*nu/((1+nu)*(1-2*nu));    // Lame parameters
using Vec = Vector2; using Mat = Matrix2; //Handy abbriviations for lin. algebra
struct Particle {Vec x/*position*/, v/*velocity*/; Mat B/*affine momentum*/;
  Mat F/*elastic deformation grad.*/;   real Jp /*det(plastic def. grad.)*/;
  Particle(Vec x, Vec v=Vec(0)) : x(x), v(v), B(0), F(1), Jp(1) {} };
std::vector<Particle> particles; // Particle states
Vector3 grid[n + 1][n + 1];// velocity with mass, note that node res=cell res+1

void advance(real dt) {                     // Simulation
  std::memset(grid, 0, sizeof(grid));  // Reset grid
  for (auto &p : particles) {              // P2G
    Vector2i base_coord = (p.x*inv_dx-Vec(0.5_f)).cast<int>();
    Vec fx = p.x * inv_dx - base_coord.cast<real>();
    // Quadratic kernels, see http://mpm.graphics Formula (123)
    Vec w[3]{Vec(0.5) * sqr(Vec(1.5) - fx), Vec(0.75) - sqr(fx - Vec(1.0)),
             Vec(0.5) * sqr(fx - Vec(0.5))};
    auto e = std::exp(hardening * (1.0_f - p.Jp)), mu=mu_0*e, lambda=lambda_0*e;
    real J = determinant(p.F);             //Current volume
    Mat r, s; polar_decomp(p.F, r, s); //Polor decomp. for Fixed Corotated Model
    auto force =                          // Negative Cauchy stress times dt and inv_dx
        inv_dx*dt*vol*(2*mu * (p.F-r) * transposed(p.F) + lambda * (J-1) * J);
    for (int i = 0; i < 3; i++) for (int j = 0; j < 3; j++) { // Scatter to grid
      auto dpos = fx - Vec(i, j);
      Vector3 contrib(p.v * mass, mass);
      grid[base_coord.x + i][base_coord.y + j] +=
          w[i].x*w[j].y*(contrib+Vector3(4.0_f*(force+p.B*mass)*dpos));
    }
  }
  for(int i = 0; i <= n; i++) for(int j = 0; j <= n; j++) { //For all grid nodes
    auto &g = grid[i][j];
    if (g[2] > 0) {                                // No need for epsilon here
      g /= g[2];                                   // Normalize by mass
      g += dt * Vector3(0, -100, 0);               // Apply gravity
      real boundary=0.05,x=(real)i/n,y=real(j)/n;//boundary thickness,node coord
      if (x < boundary||x > 1-boundary||y > 1-boundary) g=Vector3(0);//Sticky BC
      if (y < boundary) g[1]=std::max(0.0_f, g[1]);             //"Separate" BC
    }   // "BC" stands for "boundary condition", which is applied to grid nodes
  }
  for (auto &p : particles) { // Grid to particle
    Vector2i base_coord = (p.x * inv_dx - Vec(0.5_f)).cast<int>();
    Vec fx = p.x * inv_dx - base_coord.cast<real>();
    Vec w[3]{Vec(0.5) * sqr(Vec(1.5) - fx), Vec(0.75) - sqr(fx - Vec(1.0)),
             Vec(0.5) * sqr(fx - Vec(0.5))};
    p.B = Mat(0); p.v = Vec(0);
    for (int i = 0; i < 3; i++) for (int j = 0; j < 3; j++) {
      auto dpos = fx - Vec(i, j),
           grid_v = Vec(grid[base_coord.x + i][base_coord.y + j]);
      auto weight = w[i].x * w[j].y;
      p.v += weight * grid_v;                                      //  Velocity
      p.B += Mat::outer_product(weight * grid_v, dpos);            //    APIC B
    }
    p.x += dt * p.v;                                               // Advection
    auto F = (Mat(1) - (4 * inv_dx * dt) * p.B) * p.F;       // MLS-MPM F-update
    Mat svd_u, sig, svd_v; svd(F, svd_u, sig, svd_v); // SVD for snow Plasticity
    for (int i = 0; i < 2; i++)    // See SIGGRAPH 2013: MPM for Snow Simulation
      sig[i][i] = clamp(sig[i][i], 1.0_f - 2.5e-2_f, 1.0_f + 7.5e-3_f);
    real oldJ = determinant(F); F = svd_u * sig * transposed(svd_v);
    real Jp_new = clamp(p.Jp * oldJ / determinant(F), 0.6_f, 20.0_f);
    p.Jp = Jp_new; p.F = F;
  }
}

void add_object(Vec center) {    // Seed particles
  for (int i = 0; i < 1000; i++) // Randomly sample 1000 particles in the square
    particles.push_back(Particle((Vec::rand()*2.0_f-Vec(1))*0.08_f+center));  }

int main() {
  GUI gui("Taichi Demo: Real-time MLS-MPM 2D ", window_size, window_size);
  add_object(Vec(0.5,0.4));add_object(Vec(0.45,0.6));add_object(Vec(0.55,0.8));
  for (int i = 0;; i++) {                              // Main Loop
    advance(dt);                                       // Advance simulation
    if (i % int(frame_dt / dt) == 0) {                 // Redraw frame
      gui.canvas->clear(Vector4(0.7, 0.4, 0.2, 1.0_f)); // Clear background
      for (auto p : particles)                         // Draw particles
        gui.buffer[(p.x * (inv_dx*window_size/n)).cast<int>()] = Vector4(0.8);
      gui.update();                                    // Update GUI
    }//Reference: A Moving Least Squares Material Point Method with Displacement
  } //              Discontinuity and Two-Way Rigid Body Coupling (SIGGRAPH 2018)
}  //  By Yuanming Hu (who also wrote this 88-line version), Yu Fang, Ziheng Ge,
  //                          Ziyin Qu, Yixin Zhu, Andre Pradhana, Chenfanfu Jiang
```

The Computer Vision/Deep Learning World



Case study:
MLS-MPM-CPIC Development



What are included as the infrastructure?

- **Logging & Fomatting**
  - Essential for long-running tasks
  - No more std::cout or std::printf
- **(De)serialization**
- **Profiling**
- **Better debugging and testing**
  - Automatic stack back-trace
  - Email you when the program crashes
- **File IO support (ply, jpg, png, bmp, ttf etc.)**
- **High-performance small-size linear algebra**
- **Scripting**
- **Portable GUI**
- **Plugin system**
- **…**

The Mission of Taichi

1. **Provide an accessible, portable, extensible, and high-performance infrastructure, that is reusable and tailored for researchers in computer graphics-related fields;**
2. **Lower the barrier for computer graphics beginners by providing an easy-to-use code-base that includes demonstrative implementations of state-of-the-art research projects;**
3. **Help improve reproducibility of computer graphics research by simplifying and promoting open-sourcing.**

TAICHI

COMPUTER GRAPHICS LIBRARY

The End

>> import tensorflow as tf
>> import taichi as tc

Questions are welcome!