

[Kim16-CVPR] Accurate Image Super-Resolution Using Very Deep Convolutional Networks

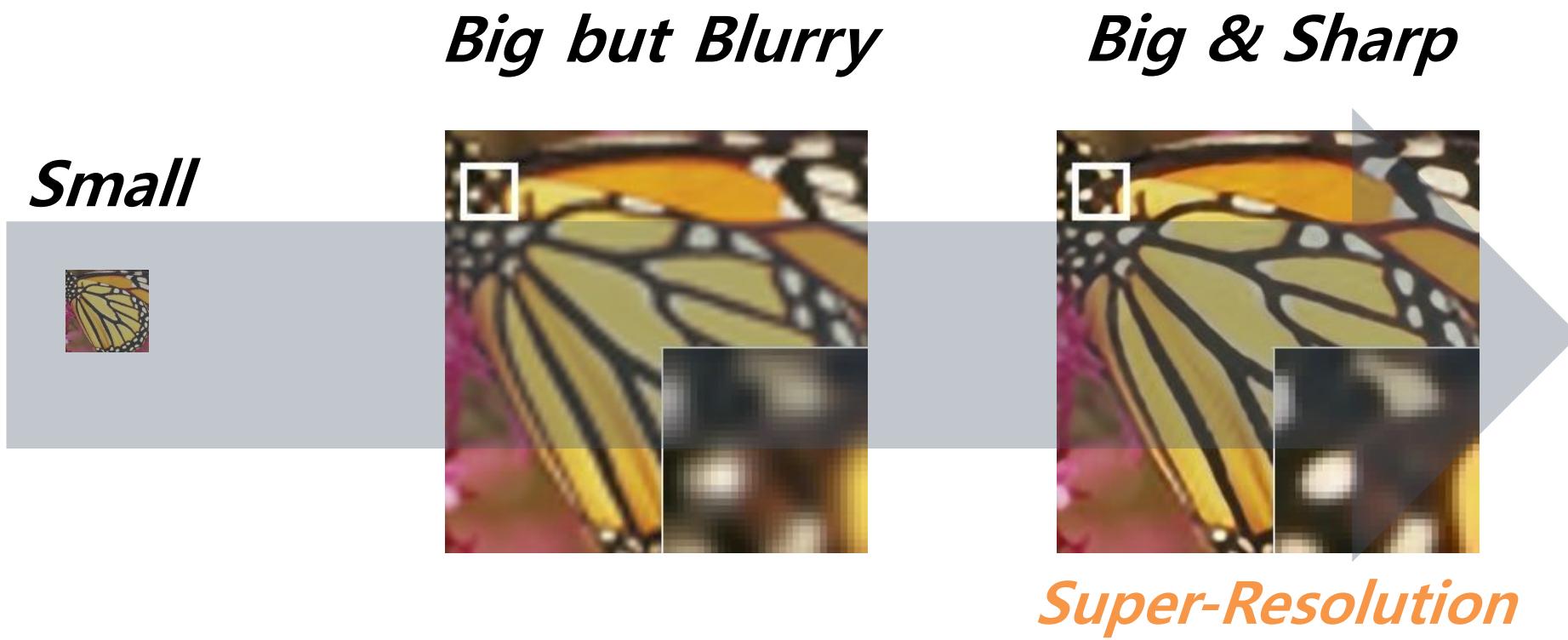
Presenter : Jiin Kim

Contents

1. Introduction
2. Related Work
3. Proposed Method
4. Understanding Properties
5. Experimental Results
6. Conclusion

1. Introduction

- Super-Resolution 이란?



Introduction

SR은 오래된 기술이지만 여전히 HOT 하다.
5년간 266,000개의 논문이 출간되었다.

The screenshot shows a Google Scholar search interface. The search query 'image super resolution' is entered in the search bar. The results page displays four academic papers related to image super-resolution, each with a title, abstract, and download link. On the left sidebar, there are filters for year (2016-2020), language (Korean), and inclusion criteria (checked for '특허 포함' and '서지정보 포함'). The year filter is highlighted with a red box.

검색 결과 약 266,000개 (0.07초)

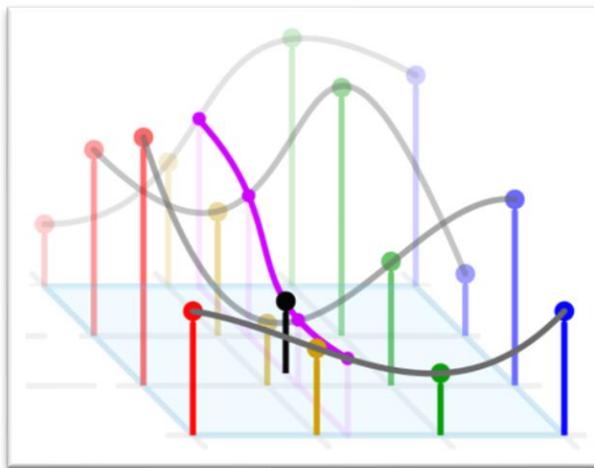
Feedback network for image super-resolution
Z Li, J Yang, Z Liu, X Yang... - Proceedings of the ..., 2019 - openaccess.thecvf.com
Recent advances in **image super-resolution** (SR) explored the power of deep learning to achieve a better reconstruction performance. However, the feedback mechanism, which commonly exists in human visual system, has not been fully exploited in existing deep ...
☆ 99 192회 인용 관련 학술자료 전체 7개의 버전 » [PDF] thecvf.com

Residual dense network for image super-resolution
Y Zhang, Y Tian, Y Kong... - Proceedings of the ..., 2018 - openaccess.thecvf.com
In this paper, we propose dense feature fusion (DFF) for **image super-resolution** (SR). As the same content in different natural images often have various scales and angles of view, jointly leaning hierarchical features is essential for **image SR**. On the other hand, very deep ...
☆ 99 1050회 인용 관련 학술자료 전체 14개의 버전 » [PDF] thecvf.com

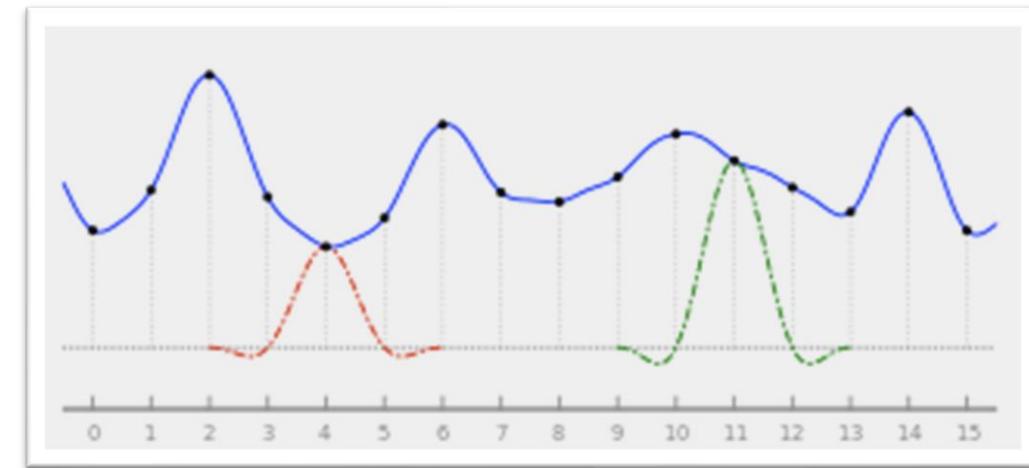
Deep learning for image super-resolution: A survey
Z Wang, J Chen, SCH Hoi - IEEE transactions on pattern ..., 2020 - ieeexplore.ieee.org
Image Super-Resolution (SR) is an important class of **image** processing techniques to enhance the **resolution** of images and videos in computer vision. Recent years have witnessed remarkable progress of **image super-resolution** using deep learning techniques ...
☆ 99 181회 인용 관련 학술자료 전체 6개의 버전 » [PDF] arxiv.org

Multi-scale residual network for image super-resolution
J Li, F Fang, K Mei, G Zhang - Proceedings of the European ..., 2018 - openaccess.thecvf.com
Recent studies have shown that deep neural networks can significantly improve the quality of **single-image super-resolution**. Current researches tend to use deeper convolutional neural networks to enhance performance. However, blindly increasing the depth of the ...
☆ 99 174회 인용 관련 학술자료 전체 9개의 버전 » [PDF] thecvf.com

- 초기의 Single Image Super-Resolution(SISR)은 interpolation 을 사용했다.



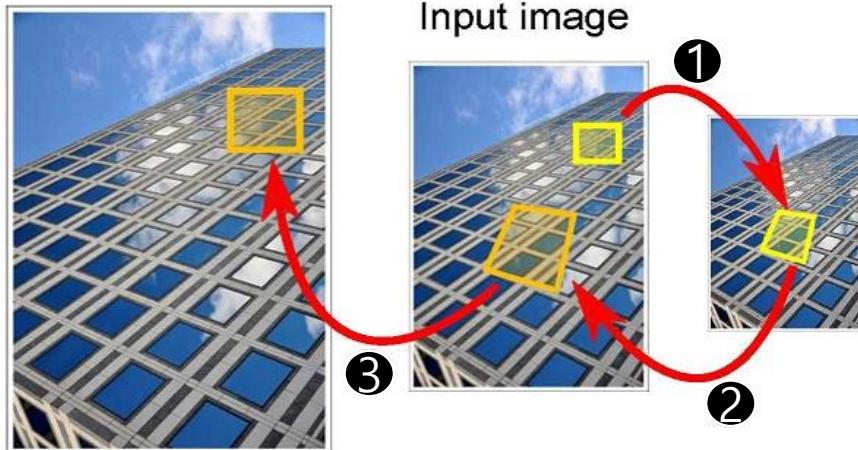
Bicubic interpolation



Lanczos resampling

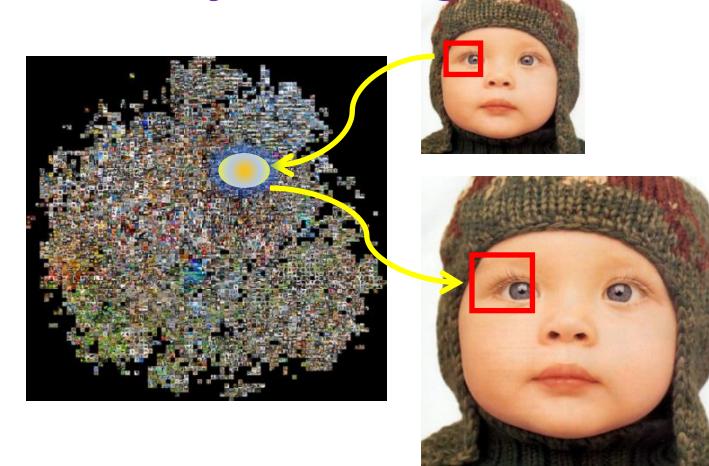
- 최근의 SISR 은 LR 과 HR 사이에서의 mapping 을 사용한다.

Self Similarity based method



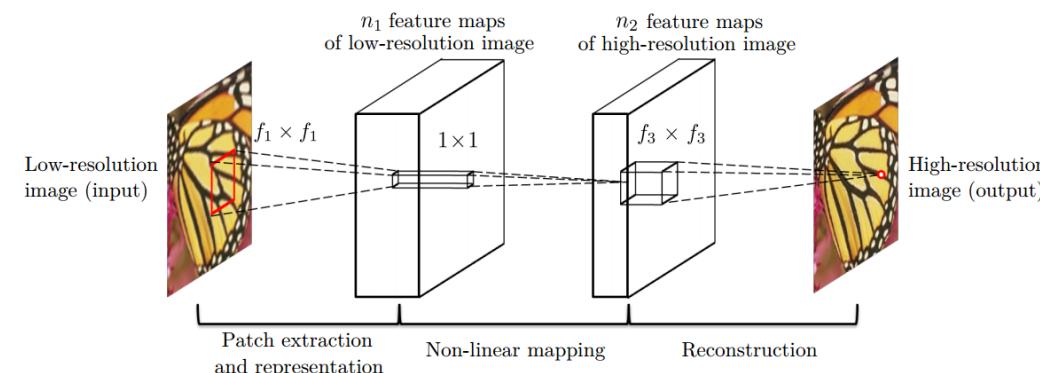
Jia-bin Huang et al. Single image Super-Resoltuion from Transformed Self-Exemplars, CVPR 2015 (SelfEx)

Dictionary-learning based method



R. Timofte et al., A+: Adjusted anchored neighborhood regression for fast super-resoltuion, ACCV 2014 (A+)

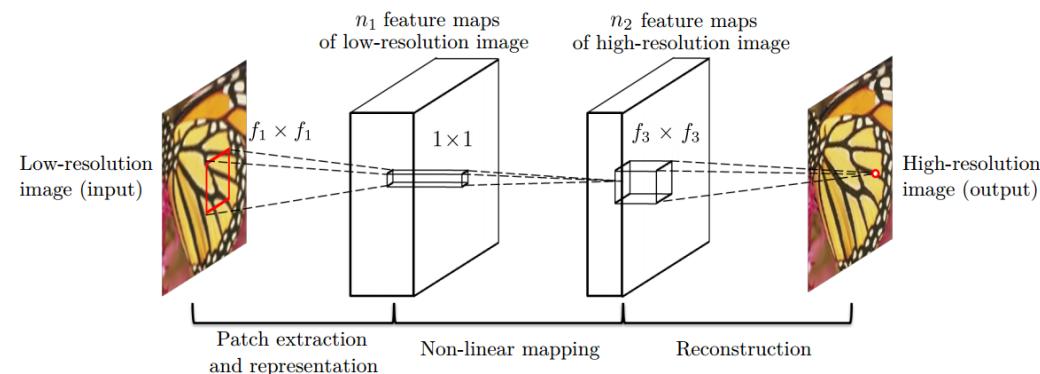
Deep learning based method



C. Dong et al. , Learning a deep convolutional network for image super-resolution, ECCV 2014 (SRCNN)

• Motivation

- SRCNN 에는 세 가지 제약이 있다.
 1. SRCNN 은 이미지의 작은 영역에서의 context 에 의존한다.
 - SRCNN 은 오직 3개의 레이어 만을 가지기 때문에 receptive field 의 크기가 작다.
 2. 훈련이 너무 느리게 수렴한다.
 - SRCNN 의 learning rate 는 10^{-5} 이기 때문에, 수렴하려면 며칠이 걸린다.
 3. SRCNN 은 오직 한가지 스케일에서만 동작한다.



• VDSR (Ours)

• Context

- 큰 receptive field 를 사용하는 매우 깊은 네트워크인 VDSR 은 이미지의 넓은 영역에 퍼져 있는 context 정보를 사용한다.

• Converge

- 빠르게 훈련하기 위해 residual-learning 과 높은 학습률 (SRCNN 의 10^4 배)을 사용한다.
- 매우 깊은 네트워크에서 작은 학습률을 사용한다면 매우 느리게 수렴 할 것이고, 높은 학습률로 수렴률을 빠르게 만들면 exploding gradients 를 발생시킨다. 이 문제를 residual-learning 과 gradient clipping 으로 해결한다.
- LR 이미지와 HR 이미지는 주로 같은 정보를 공유하기 때문에, Residual image($|HR \text{ image}-LR \text{ image}|$) 를 모델링 하는 것은 장점이 있다.

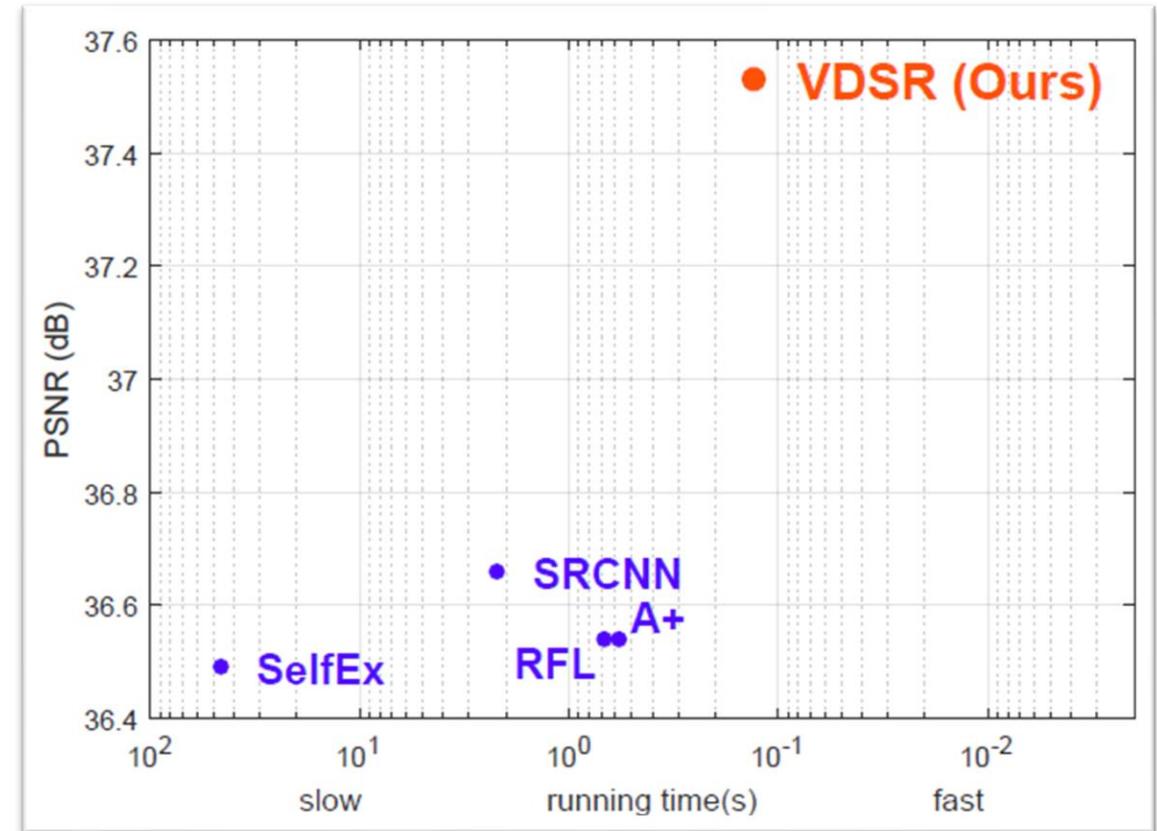
• Scale Factor (어떤 양을 늘리거나 줄이거나 또는 곱하는 수)

- Single-model SR 접근법을 제안한다.
- 하나의 네트워크로 다양한 스케일을 가진 이미지에 대해 SR 을 할 수 있다.

PSNR : Peak Signal-to-Noise Ratio

사용 목적 : 생성 혹은 압축된 영상의 화질에 대한 손실 정보를 평가한다.

의미 : 손실이 적을 수록 (화질이 좋을 수록) 높은 값을 가진다.



VDSR 의 PSNR 은 SRCNN 보다 높은 값을 가진다. (+ 0.87 dB)

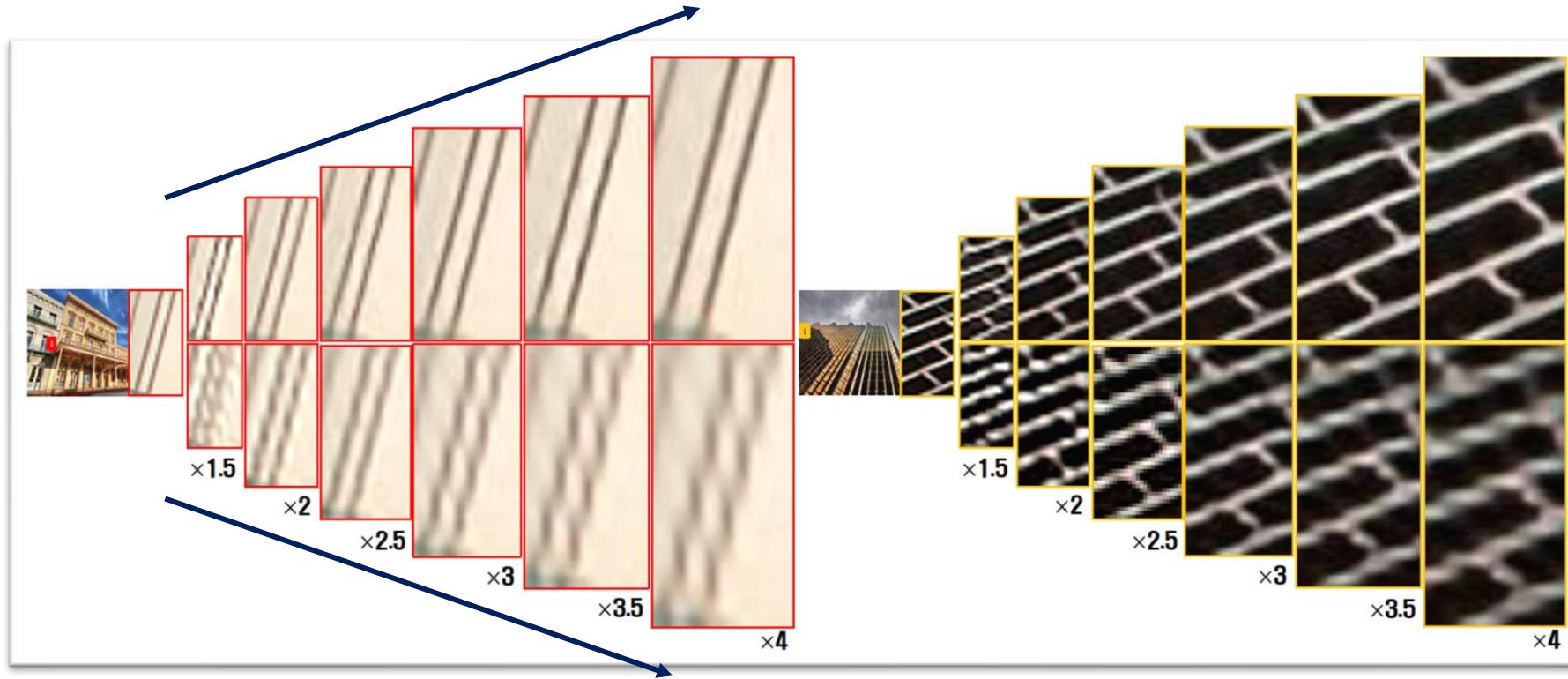
2. Related Work

2.1. Convolutional Network for Image Super-Resolution

- 딥러닝 기반의 최신 SR 방법인 **SRCNN** 과 **VDSR(Ours)** 를 비교해보자.

	SRCNN	VDSR
Model	3개의 레이어	20개의 레이어
Training	HR image 를 모델링한다. Auto-encoder 의 동작과 유사하다. 훈련 시간의 상당 부분을 이 auto-encoder 를 학습 시키는 데 에 사용하기 때문에 이미지의 세부 정보를 잘 학습하지 못한 다.	Residual image 를 모델링한다. 더 나은 정확도를 얻을 수 있고 더 빠르게 수렴한다.
Scale	Single scale factor 에 대해서 훈련되고 오직 특정 스케일에 서만 동작한다. 만약 새로운 스케일이 요구된다면, 새로운 모 델이 훈련 되어야 한다.	여러 스케일에 대한 SR 문제를 효율적으로 해결하기 위해 단일 네트워크를 설계하고 훈련한다
Size of output	출력 이미지의 크기는 입력 이미지보다 작다.	모든 레이어에 zero-padding 을 주어 입력 이미지와 출력 이미 지의 크기는 동일하다.
Learning rate	안정적인 수렴을 위해 다른 레이어에는 다른 학습률을 사용 한다.	모든 레이어에 같은 학습률을 사용한다.

Experiments results

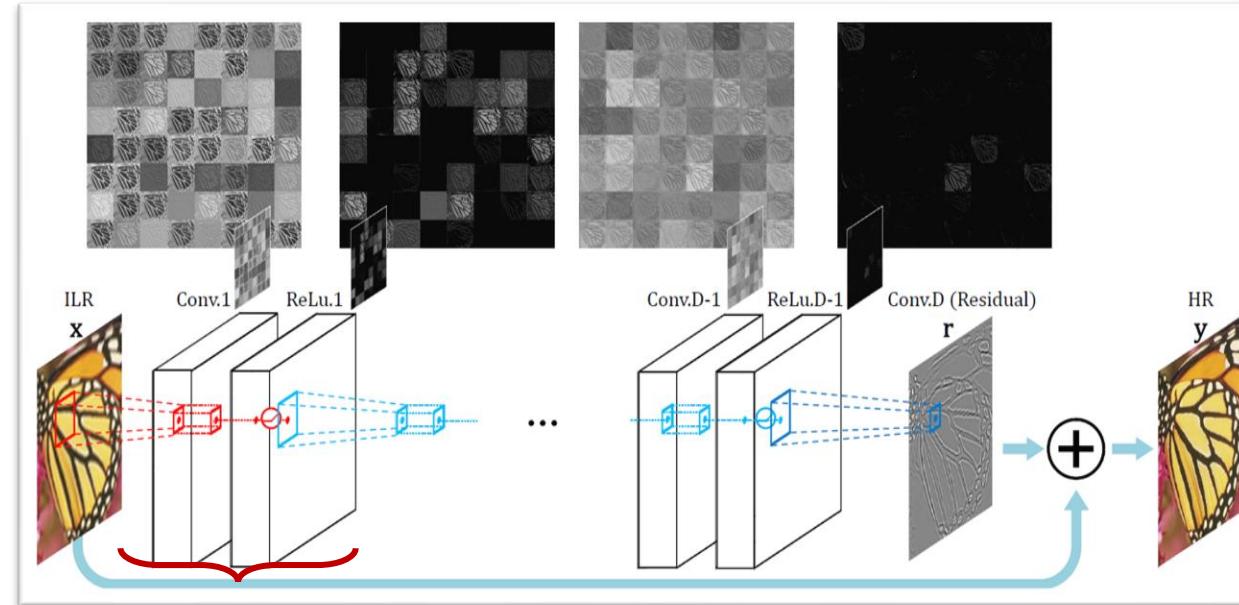


위 : VDSR → Super-resolved images 는 모두 깨끗하고 분명하다.
아래 : SRCNN → 결과가 시각적으로 좋지 않다.

3. Proposed Method

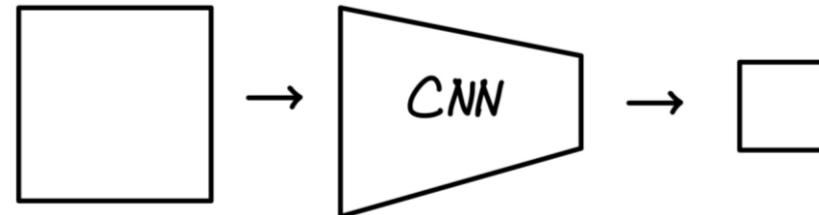
3.1. Proposed Network

- SR image reconstruction 을 위해, 매우 깊은 convolutional network 을 사용한다.

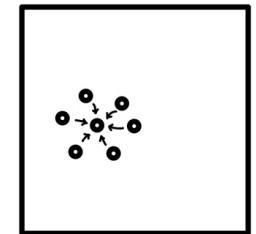


- 한 쌍의 레이어 (convolutional & nonlinear) 를 반복적으로 연결한다.
- ILR (Interpolated Low-Resolution) 이미지는 레이어들을 통과하면 HR 이미지로 변형된다.
- 네트워크는 residual image 를 예측하고, ILR 과 residual 을 더하면 원하는 결과가 나온다.
- 각 합성곱 층에 64 차원의 filter 를 사용하고, 몇 개의 feature map 들을 예시로 나타냈다.
- ReLU 를 적용한 후의 대부분의 features 는 0이다.

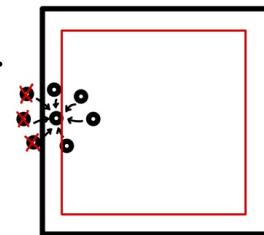
- 매우 깊은 네트워크를 사용하면, 합성곱 연산이 적용될 때마다 feature map의 크기가 줄어든다는 단점이 있다.



- 많은 SR 방법들은 center pixels를 정확히 예측하기 위해 surrounding pixels를 요구한다.

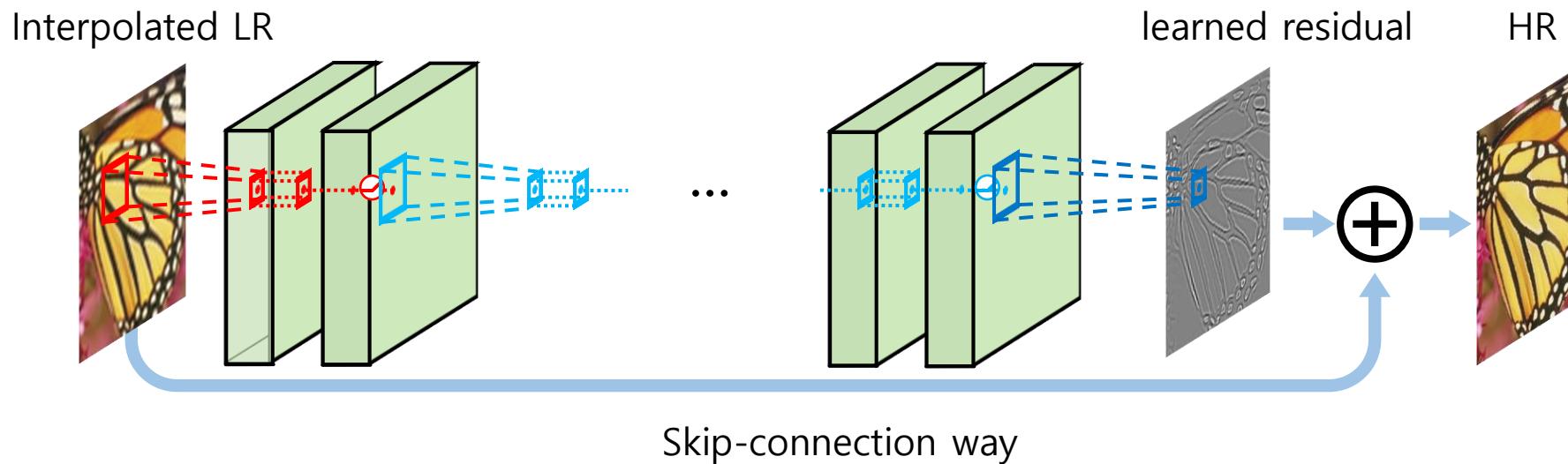


- 이미지 경계 가까이에 있는 픽셀들에 대해서는 이 방법을 사용할 수 없기 때문에, 많은 SR 방법들은 결과 이미지를 자른다.



- 이러한 방법은 요구하는 surround region이 매우 크다면 cropping 후에 최종 이미지가 너무 작아지게 된다.

- 이 문제를 해결하기 위해, 우리는 모든 feature maps (출력 이미지를 포함)의 크기를 같게 유지하고 합성곱 연산 전에 zero-padding 을 적용했다.
- 이러한 이유로, 우리의 방법은 이미지 경계 가까이에 있는 픽셀들도 정확히 예측한다.
- 예측된 이미지의 디테일은 input ILR image 에 더해져 final image (HR)을 만든다.

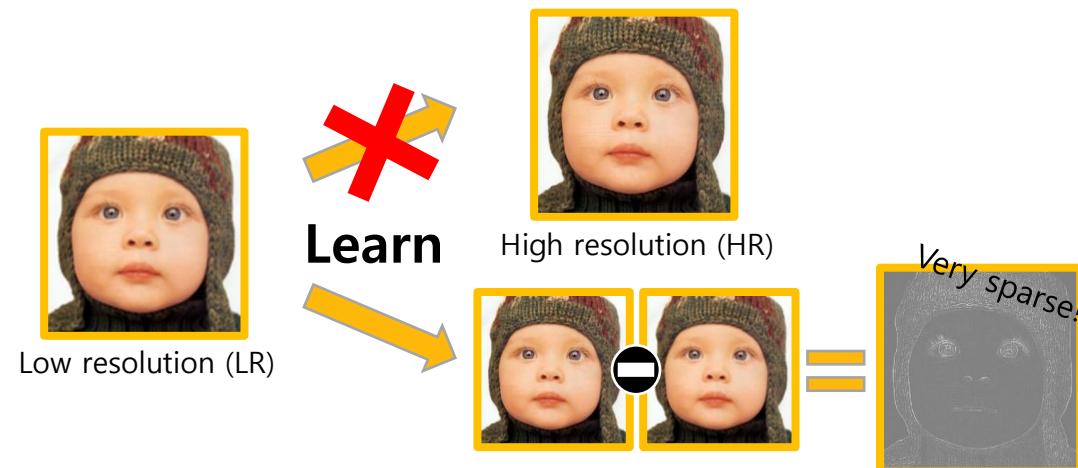


3.2. Training

- Training dataset : $\{x^{(i)}, y^{(i)}\}_{i=1}^N$
 - x : Interpolated low-resolution image
 - y : High-resolution image
- Training details
 - Residual learning
 - High Learning Rates for Very Deep Networks
 - Adjustable Gradient Clipping
 - Multi-Scale

• Residual-Learning

- Residual learning 으로 Vanishing/Exploding gradients 문제를 해결한다.
- Residual image
 - $r = y - x$ (ILR - HR)
 - Residual image 의 대부분의 값들은 0 이거나 작은 수이다.



• Residual-Learning

- Loss 함수 : $\frac{1}{2} \|r - f(x)\|^2$

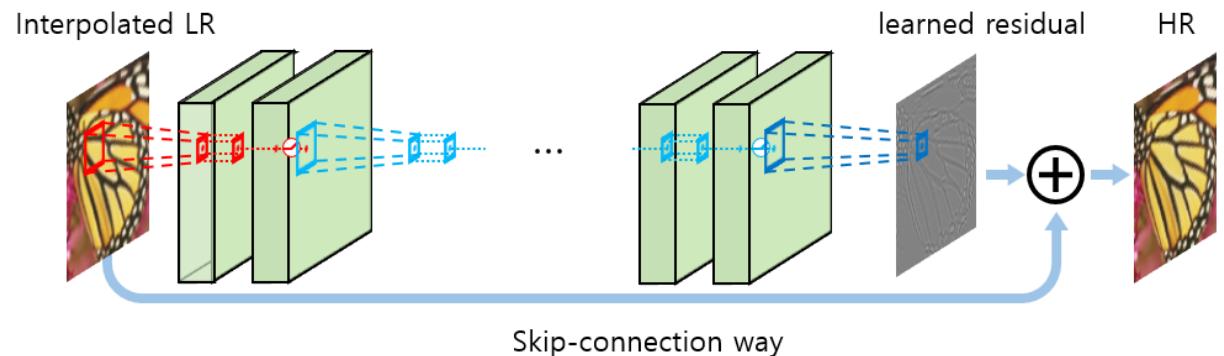
- r : target residual image
- f : the network prediction (residual image를 예측한다.)

- x : interpolated low-resolution image (i.e. bicubic)

- Final super-resolution result : $f(x) + x$

- SGD 사용(momentum = 0.9)

- Weight decay로 정규화 한다. (L_2 penalty multiplied by 0.0001)

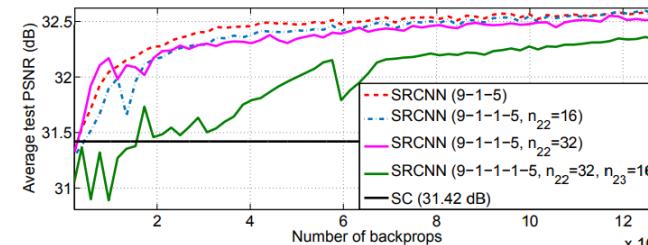
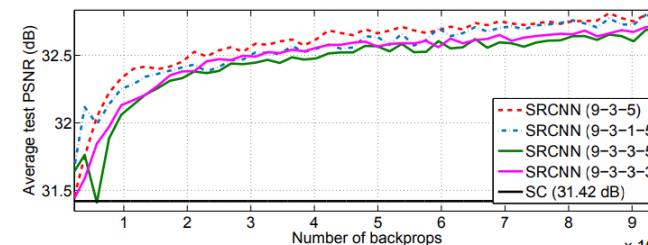


- **High Learning Rates for Very Deep Networks**

- SRCNN 은 3개 이상의 레이어를 사용한다고 해서 우수한 성능을 보여주지 못한다.
- 그것에 대한 한가지 가능한 이유는 그들이 네트워크가 수렴하기 전에 훈련을 멈추기 때문이다.
- SRCNN 의 학습률인 10^{-5} 는 일반적인 GPU 에서 일주일 내에 네트워크가 수렴하기에 너무 작다.

• High Learning Rates for Very Deep Networks

- SRCNN[6] 의 Fig. 9 를 보면, SRCNN 의 deeper networks 가 수렴되었다고 말할 수 없고 SRCNN 의 성능은 포화되었다.

(a) 9-1-1-5 ($n_{22} = 32$) and 9-1-1-1-5 ($n_{22} = 32, n_{23} = 16$)

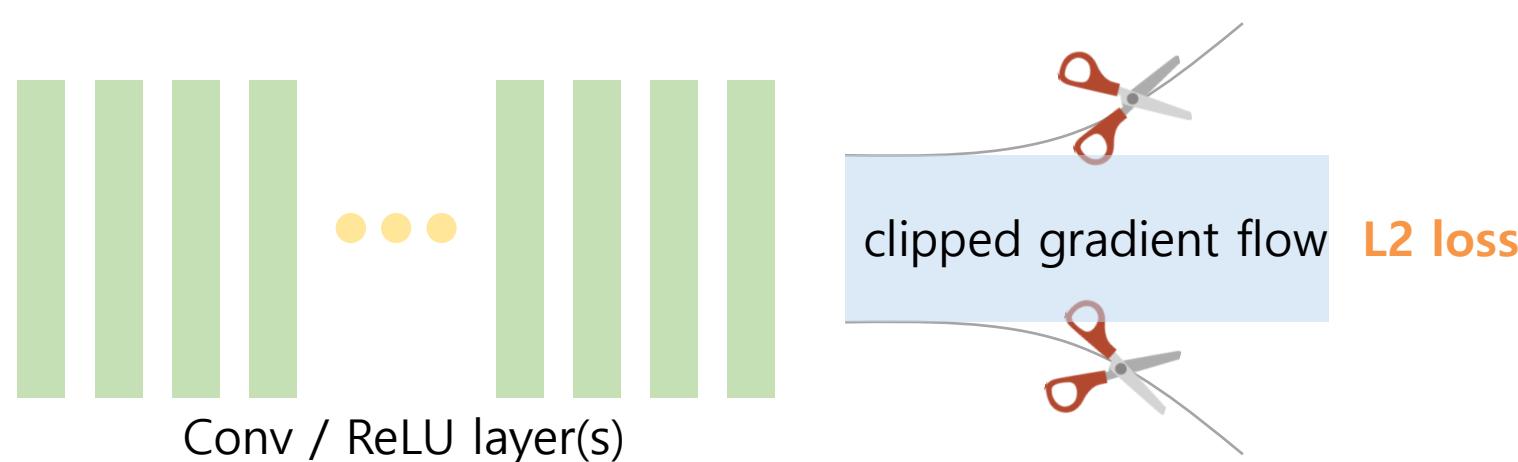
(b) 9-3-3-5 and 9-3-3-3

Fig. 9. Deeper structure does not always lead to better results.

- 훈련을 빠르게 하기 위해 학습률을 높이는 것이 기본적인 규칙이다. 그러나 간단히 학습률만 높게 만드는 것은 vanishing/exploding gradients 를 발생시킨다.
- Exploding gradients 는 줄이고 속도는 높이기 위해 gradient clipping 을 제안한다.

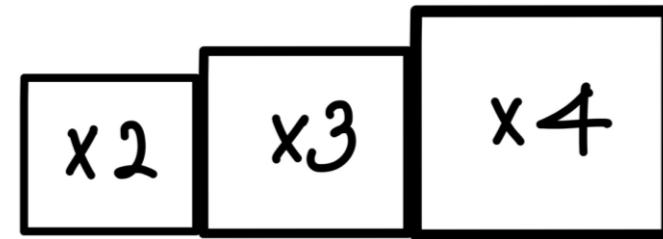
• Adjustable Gradient Clipping

- Adjustable gradient clipping 은 vanishing/exploding gradient problem 을 해결하는데에 효율적이다.
- 최대 속도의 수렴을 위해, gradients 를 $[-\frac{\theta}{\gamma}, \frac{\theta}{\gamma}]$ 로 만든다. γ 은 현재의 학습률이다.
- 20개의 레이어를 가진 VDSR 은 훈련에 4시간이 걸리고, 3개의 레이어를 가진 SRCNN 은 훈련에 며칠이 걸린다.



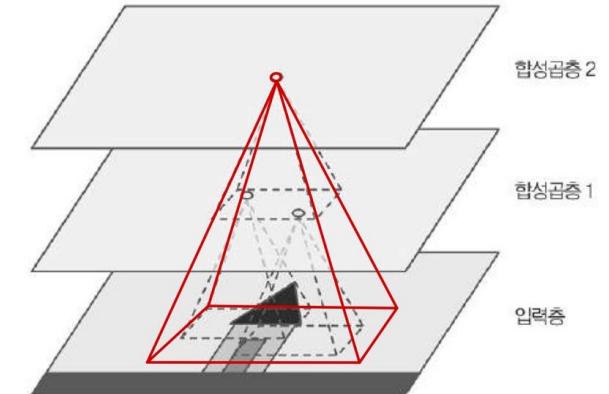
- **Multi-Scale**

- Multi-scale model 을 훈련한다.
- 몇 개의 특정 크기를 위한 훈련 데이터셋들은 하나의 큰 데이터셋으로 합쳐진다.
- 이미지들은 중복없이 sub-images 로 나눠진다.
- 배치 사이즈는 64이고, 다른 크기의 sub-images 는 같은 배치 안에 있을 수 있다.



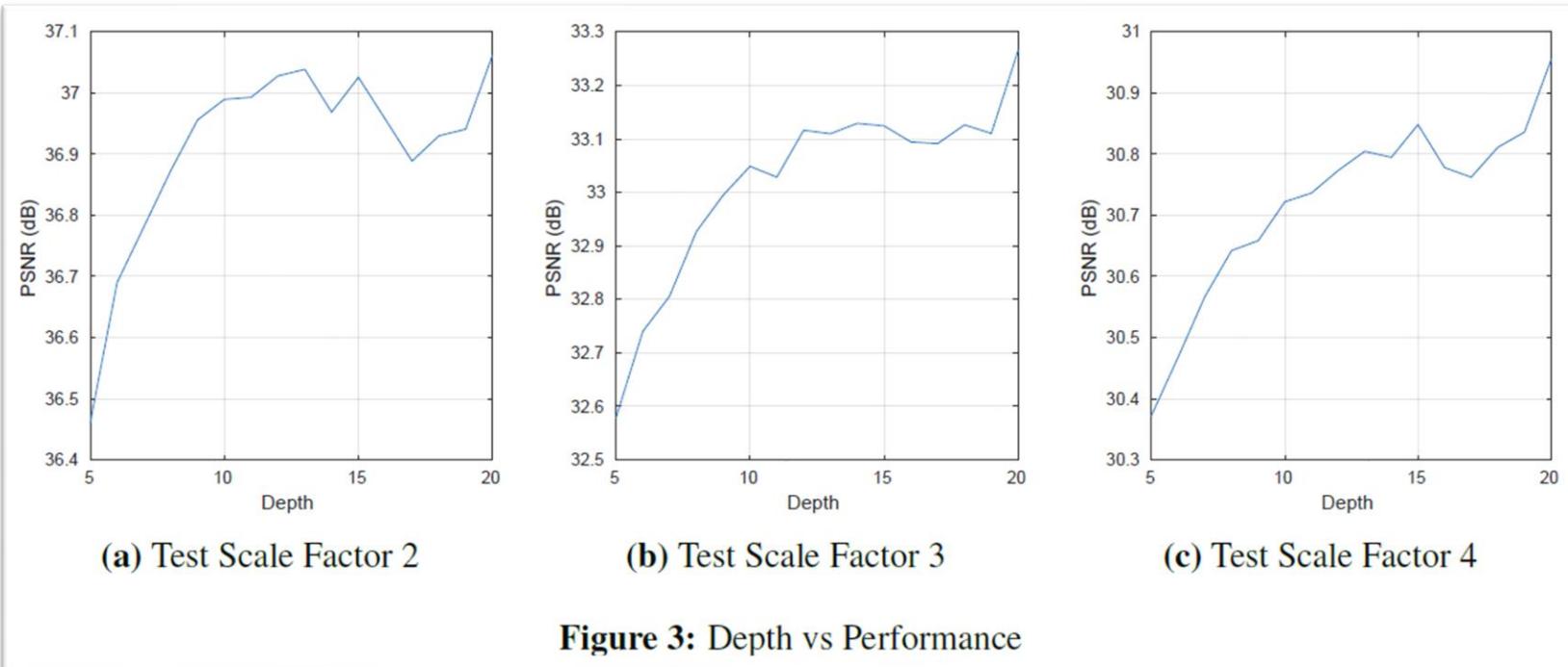
4. Understanding Properties

4.1. The Deeper, the Better



- 레이어를 많이 쌓는 것은 filters 를 global하게 만든다. (pixel space 의 더 큰 영역에 반응한다.)
- 모든 레이어에 64 개의 channel 을 가진 3×3 크기의 filter 를 사용한다.
- 깊이가 D 인 네트워크의 receptive field 크기 : $(2D + 1) \times (2D + 1)$
- 큰 receptive field 일 수록 더 많은 context 정보를 이미지 세부사항을 예측하는데 사용한다.
- 매우 깊은 네트워크는 high nonlinearities 를 활용할 수 있다. ReLU 를 19번 사용하고 channels 의 수를 조정해서 매우 복잡한 함수를 모델링한다.

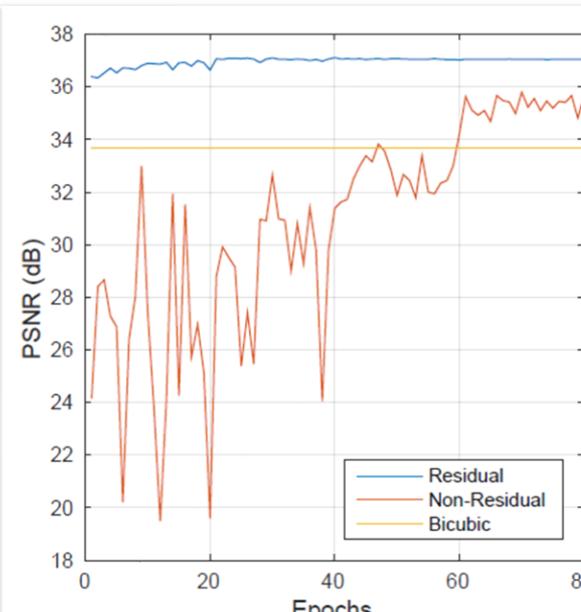
Understanding Properties



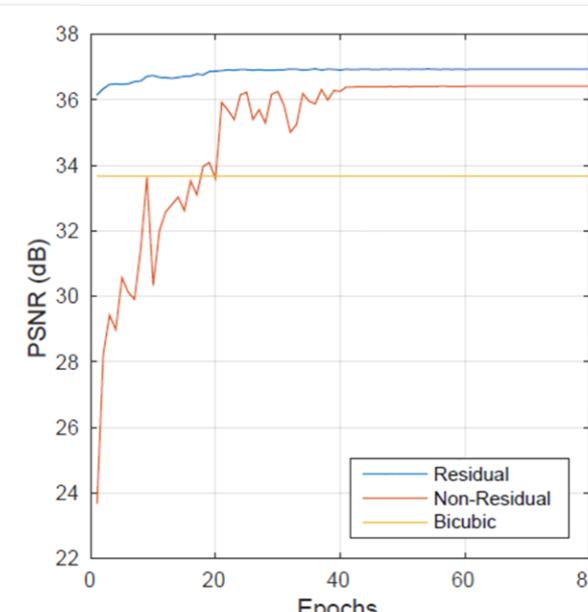
- 매우 깊은 네트워크가 SR 성능을 상당히 높이는 것을 보여준다.
- 5에서 20 사이의 범위에 있는 깊이로 네트워크를 훈련하고 테스트한다.
- 깊이가 깊어질 수록 성능이 좋아진다.

4.2. Residual-Learning

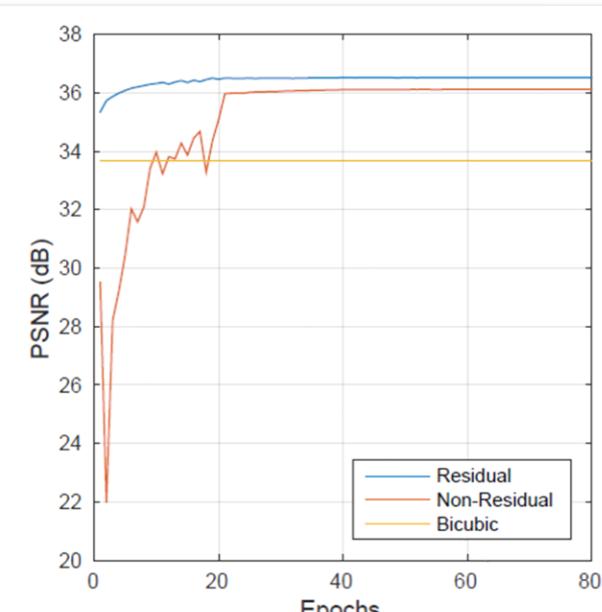
- Residual network(residual image 를 모델링)와 non-residual network(high-resolution image 를 모델링)의 Performance curve
 - Scale factor = 2 인 'Set5'에서 두 개의 네트워크를 테스트한다.



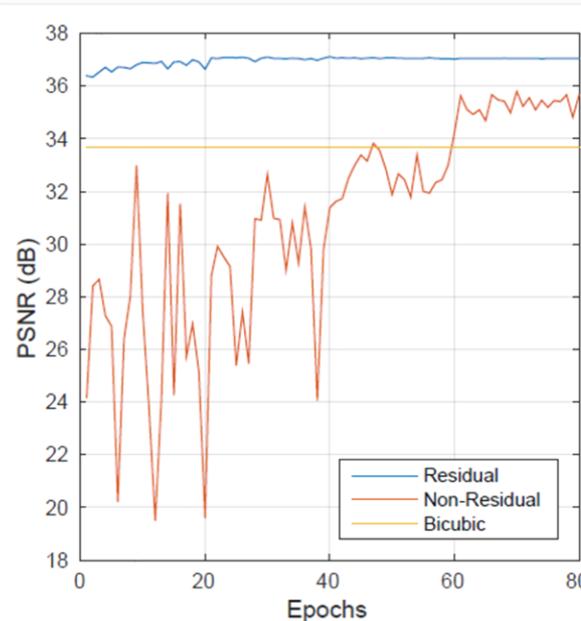
(a) Initial learning rate 0.1



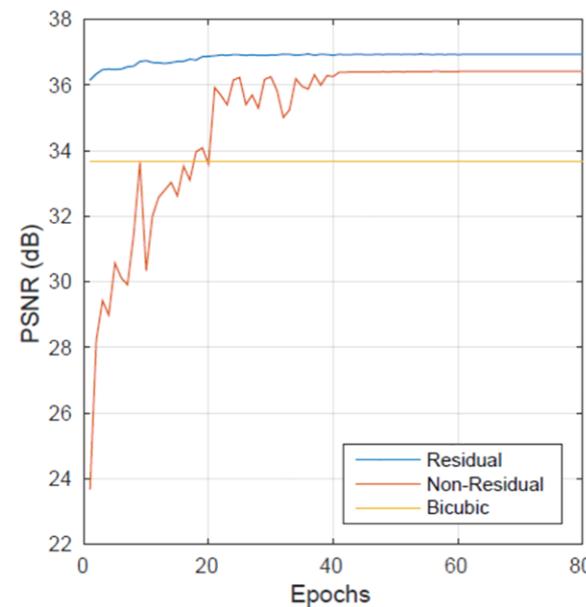
(b) Initial learning rate 0.01



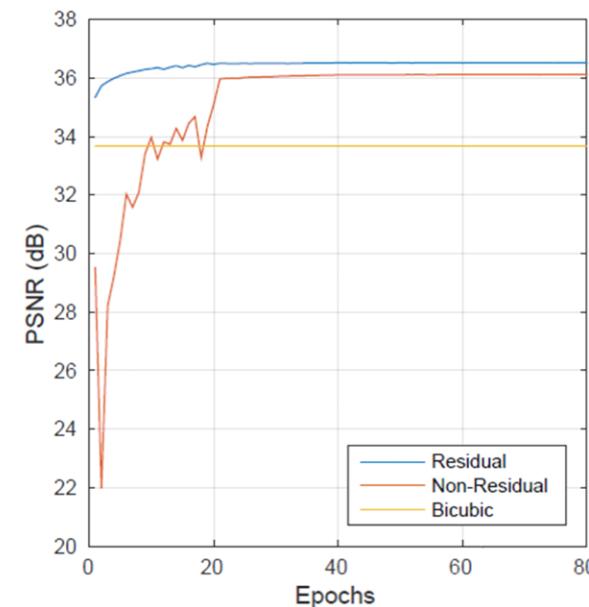
(c) Initial learning rate 0.001



(a) Initial learning rate 0.1

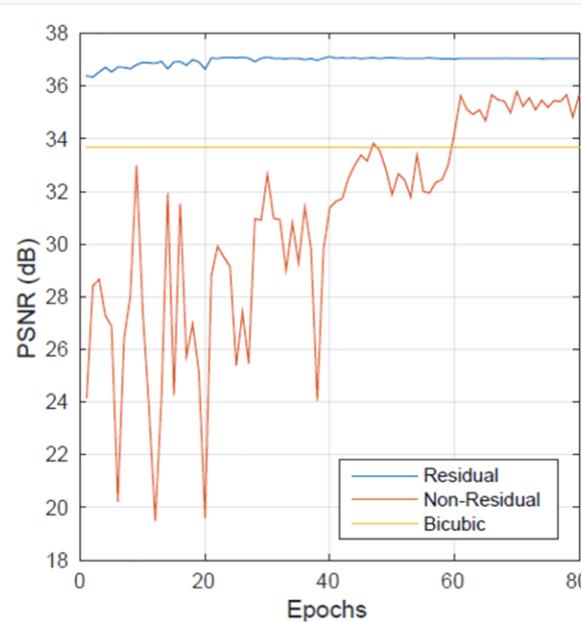


(b) Initial learning rate 0.01

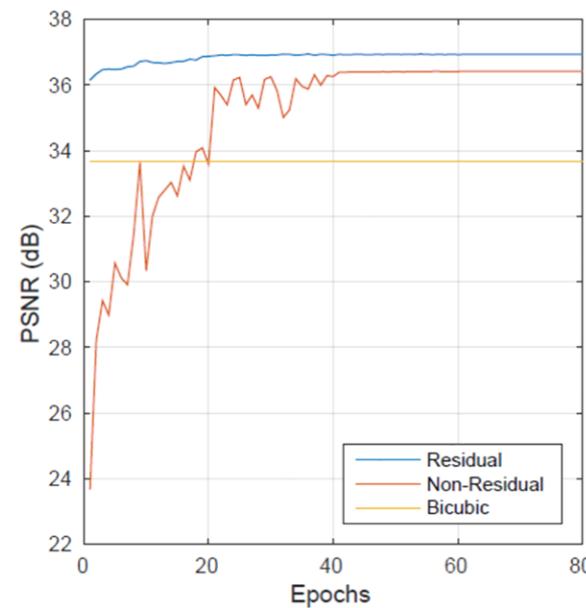


(c) Initial learning rate 0.001

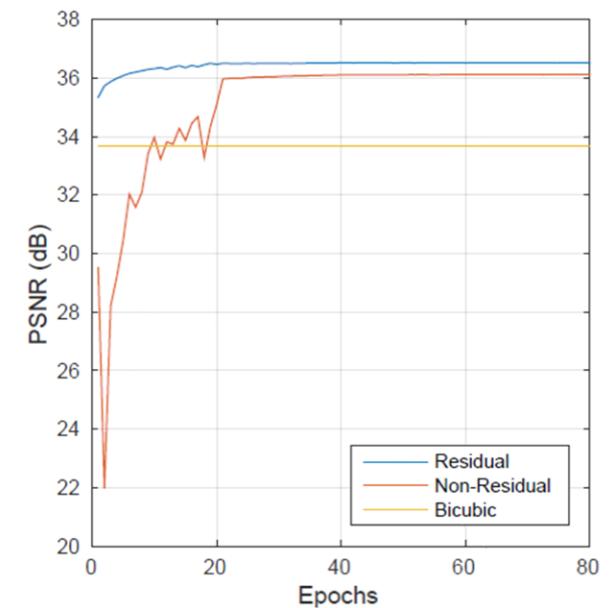
- Residual network 가 Non-Residual network 보다 더 빠르게 수렴한다.
 - Residual network 는 몇 epoch 만에 최고 성능에 도달하지만, Non-Residual networks 는 최고 성능에 도달하기 위해 많은 epoch 가 필요하다.



(a) Initial learning rate 0.1



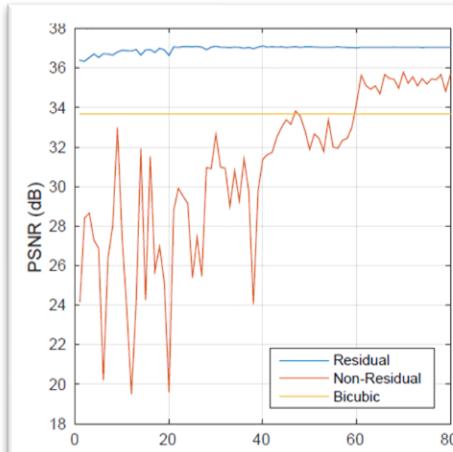
(b) Initial learning rate 0.01



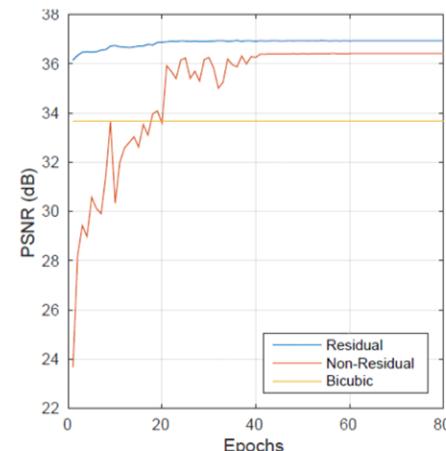
(c) Initial learning rate 0.001

- 훈련이 끝났을 때(수렴 후) Residual network 는 우수한 성능을 보여준다.
 - Residual networks 의 PSNR 가 Non-Residual network 보다 더 높다.

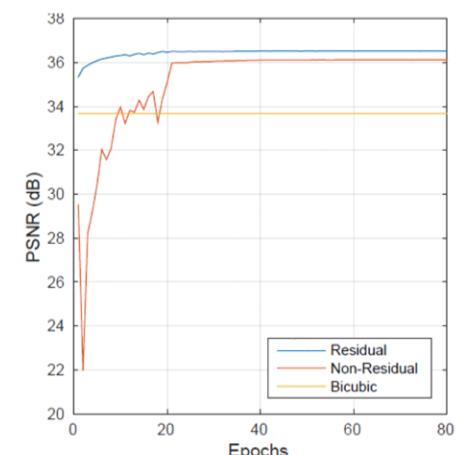
Understanding Properties



(a) Initial learning rate 0.1



(b) Initial learning rate 0.01



(c) Initial learning rate 0.001

Epoch	10	20	40	80
Residual	36.90	36.64	37.12	37.05
Non-Residual	27.42	19.59	31.38	35.66
Difference	9.48	17.05	5.74	1.39

(a) Initial learning rate 0.1

Epoch	10	20	40	80
Residual	36.74	36.87	36.91	36.93
Non-Residual	30.33	33.59	36.26	36.42
Difference	6.41	3.28	0.65	0.52

(b) Initial learning rate 0.01

Epoch	10	20	40	80
Residual	36.31	36.46	36.52	36.52
Non-Residual	33.97	35.08	36.11	36.11
Difference	2.35	1.38	0.42	0.40

(c) Initial learning rate 0.001

Residual network 와 non-residual network 의 PSNR

- 초기 학습률을 작게 설정하면 빠르게 수렴하지 않는다. Residual network 의 PSNR 을 살펴보자.
 - 초기 학습률 = 0.1 → 10 epochs 내에 36.90 에 도달한다.
 - 초기 학습률 = 0.001 → 80 epochs 후에 36.52 에 도달한다.

Understanding Properties

Epoch	10	20	40	80
Residual	36.90	36.64	37.12	37.05
Non-Residual	27.42	19.59	31.38	35.66
Difference	9.48	17.05	5.74	1.39

(a) Initial learning rate 0.1

Epoch	10	20	40	80
Residual	36.74	36.87	36.91	36.93
Non-Residual	30.33	33.59	36.26	36.42
Difference	6.41	3.28	0.65	0.52

(b) Initial learning rate 0.01

Epoch	10	20	40	80
Residual	36.31	36.46	36.52	36.52
Non-Residual	33.97	35.08	36.11	36.11
Difference	2.35	1.38	0.42	0.40

(c) Initial learning rate 0.001

Residual network 와 non-residual network 의 PSNR

- Residual network 와 Non-Residual networks 는 10 epochs 후에 급격한 성능의 차이를 보여준다.
- Initial learning rate = 0.1 의 경우,

Residual 의 PSNR : 36.90 & Non-residual 의 PSNR : 27.42

4.3. Single Model for Multiple Scales

- 우리의 네트워크를 single scale factor s_{train} 으로 훈련하고 다른 scale factor s_{test} 로 테스트한다.

Test / Train	$\times 2$	$\times 3$	$\times 4$	$\times 2,3$	$\times 2,4$	$\times 3,4$	$\times 2,3,4$	Bicubic
$\times 2$	37.10	30.05	28.13	37.09	37.03	32.43	37.06	33.66
$\times 3$	30.42	32.89	30.50	33.22	31.20	33.24	33.27	30.39
$\times 4$	28.43	28.73	30.84	28.70	30.86	30.94	30.95	28.42

- Scale Factor 실험 (PSNR 측정)
- 몇 개의 모델들이 다른 scale sets 에서 훈련된다.
- 데이터셋 'Set5'을 사용한다.
- Red color 는 test scale 이 훈련에 포함되는 것을 의미한다.

Test / Train	$\times 2$	$\times 3$	$\times 4$	$\times 2,3$	$\times 2,4$	$\times 3,4$	$\times 2,3,4$	Bicubic
$\times 2$	37.10	30.05	28.13	37.09	37.03	32.43	37.06	33.66
$\times 3$	30.42	32.89	30.50	33.22	31.20	33.24	33.27	30.39
$\times 4$	28.43	28.73	30.84	28.70	30.86	30.94	30.95	28.42

- 만약 $s_{train} \neq s_{test}$ 이라면 성능은 저하된다.
- Scale factor 2에 대해서 factor 2로 훈련된 모델은 PSNR이 37.10인 반면, factor 3과 factor 4로 훈련된 모델들은 각각 30.05와 28.13이다.

Test / Train	$\times 2$	$\times 3$	$\times 4$	$\times 2,3$	$\times 2,4$	$\times 3,4$	$\times 2,3,4$	Bicubic
$\times 2$	37.10	30.05	28.13	37.09	37.03	32.43	37.06	33.66
$\times 3$	30.42	32.89	30.50	33.22	31.20	33.24	33.27	30.39
$\times 4$	28.43	28.73	30.84	28.70	30.86	30.94	30.95	28.42

- Single-scale data에서 훈련된 네트워크는 다른 scales를 처리할 수 없다. 심지어 입력 이미지를 생성하기 위해 사용되는 방법인 bicubic interpolation 보다 성능이 좋지 않다.

- Scale augmentation 으로 훈련된 모델이 다양한 scale factors 에서 SR 을 수행할 수 있는지 테스트 한다.

Test / Train	$\times 2$	$\times 3$	$\times 4$	$\times 2,3$	$\times 2,4$	$\times 3,4$	$\times 2,3,4$	Bicubic
$\times 2$	37.10	30.05	28.13	37.09	37.03	32.43	37.06	33.66
$\times 3$	30.42	32.89	30.50	33.22	31.20	33.24	33.27	30.39
$\times 4$	28.43	28.73	30.84	28.70	30.86	30.94	30.95	28.42

- $s_{train} = \{2, 3, 4\}$ 의 경우 각 scale 에 대해서 single-scale network 에 대응하는 결과와 PSNR 을 비교할 만하다.
- 37.06 vs. 37.10 ($\times 2$) / 33.27 vs. 32.89 ($\times 3$) / 30.95 vs. 30.84 ($\times 4$)

Test / Train	$\times 2$	$\times 3$	$\times 4$	$\times 2,3$	$\times 2,4$	$\times 3,4$	$\times 2,3,4$	Bicubic
$\times 2$	37.10	30.05	28.13	37.09	37.03	32.43	37.06	33.66
$\times 3$	30.42	32.89	30.50	33.22	31.20	33.24	33.27	30.39
$\times 4$	28.43	28.73	30.84	28.70	30.86	30.94	30.95	28.42

- 우리의 multi-scale network 는 single-scale network 보다 우수하다.
- test scale 3 에 대해서 우리의 모델 ($\times 2, 3$), ($\times 3, 4$), ($\times 2, 3, 4$) 는 PSNRs 가 33.22, 33.24, 33.27 이고, 반면 ($\times 3$) 은 32.89이다.

Test / Train	$\times 2$	$\times 3$	$\times 4$	$\times 2,3$	$\times 2,4$	$\times 3,4$	$\times 2,3,4$	Bicubic
$\times 2$	37.10	30.05	28.13	37.09	37.03	32.43	37.06	33.66
$\times 3$	30.42	32.89	30.50	33.22	31.20	33.24	33.27	30.39
$\times 4$	28.43	28.73	30.84	28.70	30.86	30.94	30.95	28.42

- 유사하게, $(\times 2, 4)$, $(\times 3, 4)$, $(\times 2, 3, 4)$ 는 30.86, 30.94, 30.95 인 반면, $(\times 4)$ 는 30.84 이다.
- 이것으로 multiple scales 로 훈련하는 것이 large scales 를 위한 성능을 증가시킨다는 것을 관찰했다.

5. Experimental Results

5.1. Datasets for Training and Testing

- **Training dataset**

- 다른 learning-based 방법들은 다른 training images 를 사용한다.
- RFL [18] 은 두 가지 버전이 있다.
 - 첫 번째 버전 : Yang et al. [25] 의 91 images 를 사용한다.
 - 두 번째 버전 : Berkeley Segmentation Dataset [16] 으로부터 나온 200 images 를 함께 사용한다.
 - 총 291개의 이미지를 사용한다.
- SRCNN [6]
 - ImageNet dataset 을 사용한다.
 - 우리는 다른 방법들과 기준을 맞추기 위해서 [18]의 291 images 를 사용한다.
 - 게다가, data augmentation(rotation or flip)을 사용한다.

- **Test dataset**

- Datasets 'Set5' [15] 와 'Set14' [26]
 - 다른 연구에서 [22, 21, 5] benchmark 를 위해 자주 사용된다.
- Dataset 'Urban100'
 - Urban images
- Dataset 'B100'
 - Berkeley Segmentation Dataset 의 natural images

5.2. Training Parameters

- depth = 20
- batch size = 64
- momentum = 0.9
- weight decay parameters = 0.0001
- He et al. [10]를 사용하여 가중치를 초기화한다.
- 모든 실험들을 80 epochs 이상 훈련한다. (9960 iterations with batch size 64).
- learning rate 는 초기에 0.1로 설정되었고 20 epochs 마다 10배씩 감소한다.
- 전체적으로, 학습률은 3배 감소되고, 학습은 80 epochs 후에 중단된다.
- 훈련은 GPU Titan Z 에서 대략 4시간이 걸린다.

5.3. Benchmark

- Benchmark 를 위해, 우리는 공적으로 이용가능한 프레임워크인 Huang et al. [21] 을 따른다. 이것은 같은 평가 절차를 가진 많은 최신 결과들과 비교할 수 있게 한다.
- 프레임워크는 이미지의 색상 성분에 bicubic interpolation 을 적용하고 휘도 성분에 복잡한 모델들을 적용한다. 인간의 시각은 색상보다 강도에서 더 민감하기 때문이다.
- 이 프레임워크는 이미지 경계에 있는 픽셀들을 잘라낸다. 픽셀을 잘라내는 것에 있어서 VDSR 은 네트워크가 full-sized 이미지를 출력하기 때문에 불필요하지만,公正한 비교를 위해 같은 양의 픽셀들을 잘라낸다.

5.4. Comparisons with State-of-Art Methods

PSNR : 생성 혹은 압축된 영상의 화질에 대한 손실 정보를 평가
SSIM : 영상 품질을 측정하는 구조적 유사도(SSIM) 지수

- 정량적 비교와 질적 비교를 한다.
- 몇 개의 데이터셋에서 summary or quantitative evaluation 을 제공한다.
- 우리의 방법은 이 데이터셋을 사용한 모든 이전 방법들을 능가하고, 속도도 비교적 빠르다.

Dataset	Scale	Bicubic PSNR/SSIM/time	A+ [22] PSNR/SSIM/time	RFL [18] PSNR/SSIM/time	SelfEx [11] PSNR/SSIM/time	SRCNN [5] PSNR/SSIM/time	VDSR (Ours) PSNR/SSIM/time
Set5	×2	33.66/0.9299/0.00	36.54/ 0.9544/0.58	36.54/0.9537/0.63	36.49/0.9537/45.78	36.66 /0.9542/2.19	37.53/0.9587/0.13
	×3	30.39/0.8682/0.00	32.58/0.9088/ 0.32	32.43/0.9057/0.49	32.58/ 0.9093 /33.44	32.75 /0.9090/2.23	33.66/0.9213/0.13
	×4	28.42/0.8104/0.00	30.28/0.8603/ 0.24	30.14/0.8548/0.38	30.31/0.8619/29.18	30.48/0.8628 /2.19	31.35/0.8838/0.12
Set14	×2	30.24/0.8688/0.00	32.28/0.9056/ 0.86	32.26/0.9040/1.13	32.22/0.9034/105.00	32.42/0.9063 /4.32	33.03/0.9124/0.25
	×3	27.55/0.7742/0.00	29.13/0.8188/ 0.56	29.05/0.8164/0.85	29.16/0.8196/74.69	29.28/0.8209 /4.40	29.77/0.8314/0.26
	×4	26.00/0.7027/0.00	27.32/0.7491/ 0.38	27.24/0.7451/0.65	27.40/ 0.7518 /65.08	27.49/0.7503 /4.39	28.01/0.7674/0.25
B100	×2	29.56/0.8431/0.00	31.21/0.8863/ 0.59	31.16/0.8840/0.80	31.18/0.8855/60.09	31.36/0.8879 /2.51	31.90/0.8960/0.16
	×3	27.21/0.7385/0.00	28.29/0.7835/ 0.33	28.22/0.7806/0.62	28.29/0.7840/40.01	28.41/0.7863 /2.58	28.82/0.7976/0.21
	×4	25.96/0.6675/0.00	26.82/0.7087/ 0.26	26.75/0.7054/0.48	26.84/ 0.7106 /35.87	26.90/0.7101 /2.51	27.29/0.7251/0.21
Urban100	×2	26.88/0.8403/0.00	29.20/0.8938/ 2.96	29.11/0.8904/3.62	29.54/0.8967 /663.98	29.50/0.8946/22.12	30.76/0.9140/0.98
	×3	24.46/0.7349/0.00	26.03/0.7973/ 1.67	25.86/0.7900/2.48	26.44/0.8088 /473.60	26.24/0.7989/19.35	27.14/0.8279/1.08
	×4	23.14/0.6577/0.00	24.32/0.7183/ 1.21	24.19/0.7096/1.88	24.79/0.7374 /394.40	24.52/0.7221/18.46	25.18/0.7524/1.06

Table 3: Average PSNR/SSIM for scale factor ×2, ×3 and ×4 on datasets Set5, Set14, B100 and Urban100. Red color indicates the best performance and blue color indicates the second best performance.

5.4. Comparisons with State-of-Art Methods

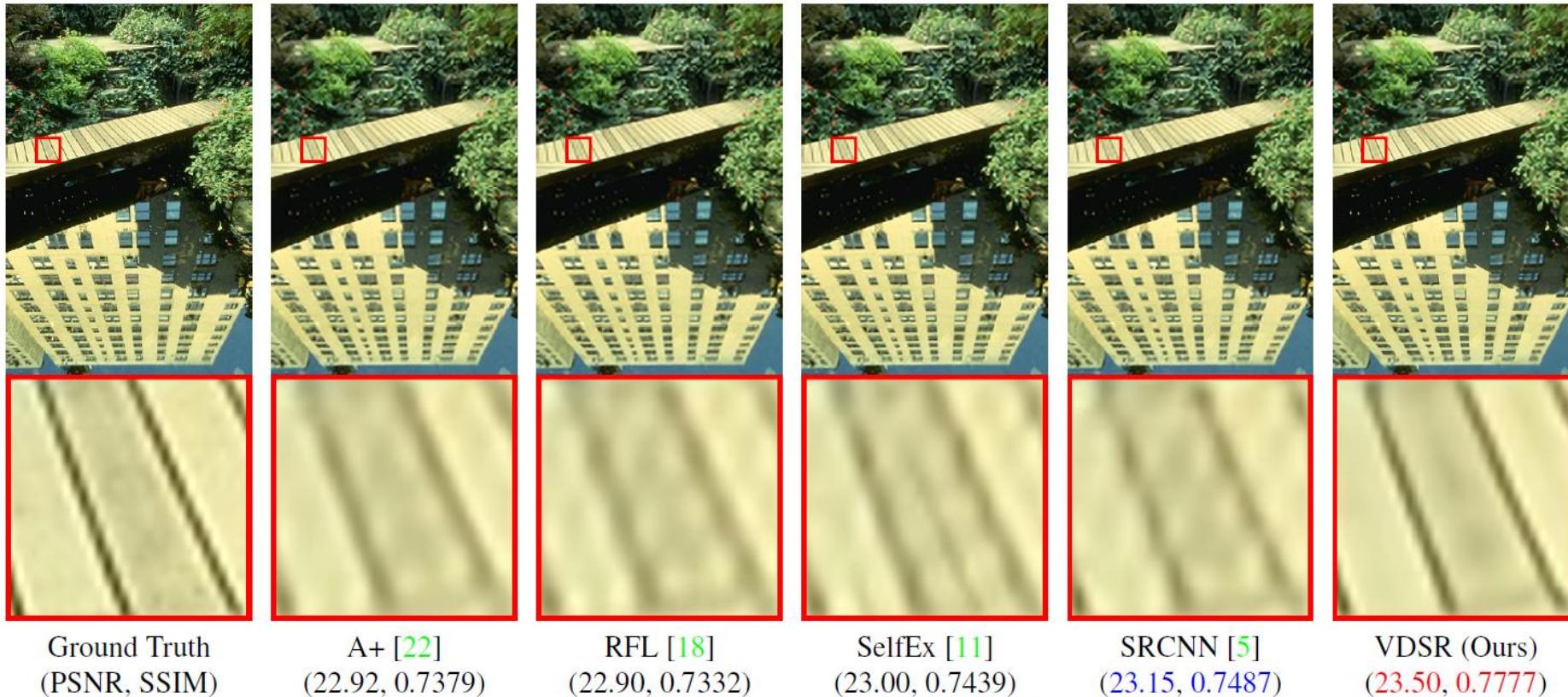


Figure 6: Super-resolution results of “148026” (*B100*) with scale factor $\times 3$. VDSR recovers sharp lines.

Experiments results

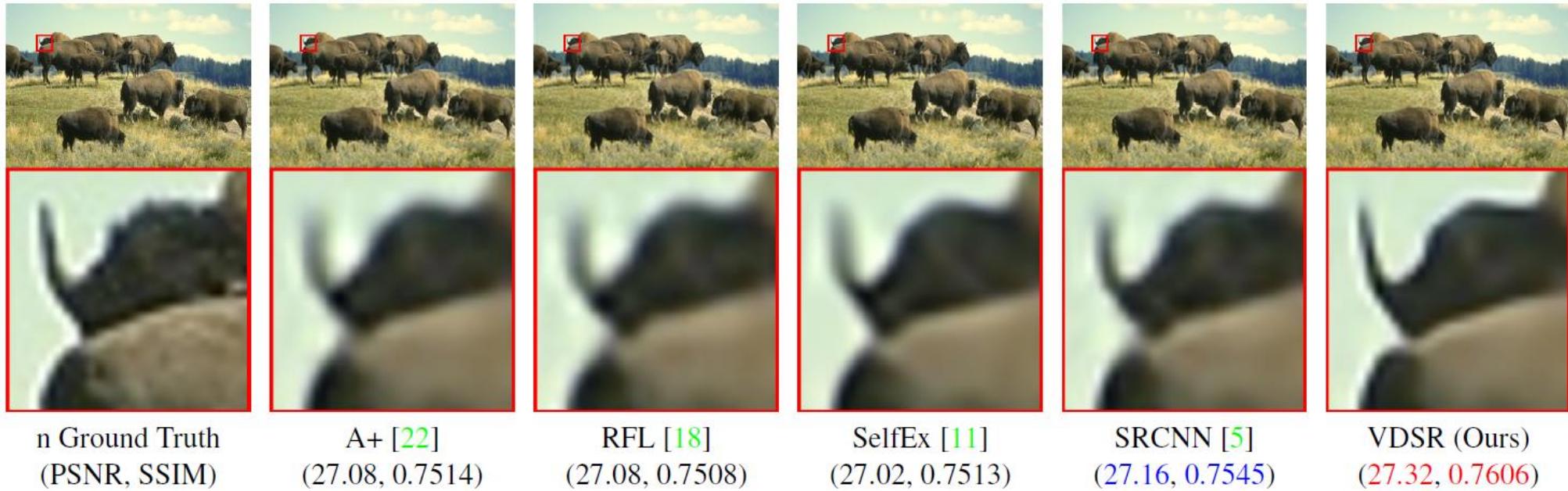


Figure 7: Super-resolution results of “38092” (*B100*) with scale factor $\times 3$. The horn in the image is sharp in the result of VDSR.





Ground truth
(PSNR/SSIM)



A+ [1]
30.00/0.7878



RFL [2]
29.86/0.7830



SelfEx [3]
29.93/0.7883



SRCNN [4]
29.98/0.7867



VDSR (ours)
30.36/0.8005



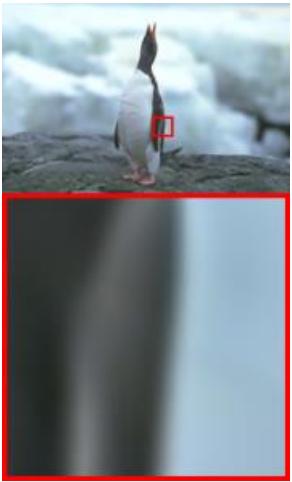
Ground truth
(PSNR/SSIM)



A+ [1]
32.76/0.8859



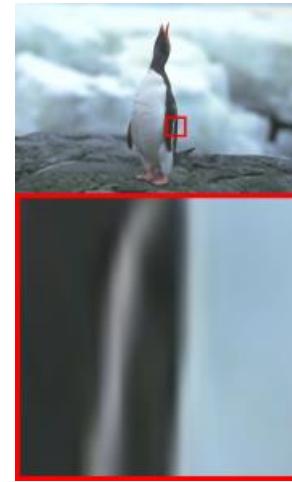
RFL [2]
32.65/0.8838



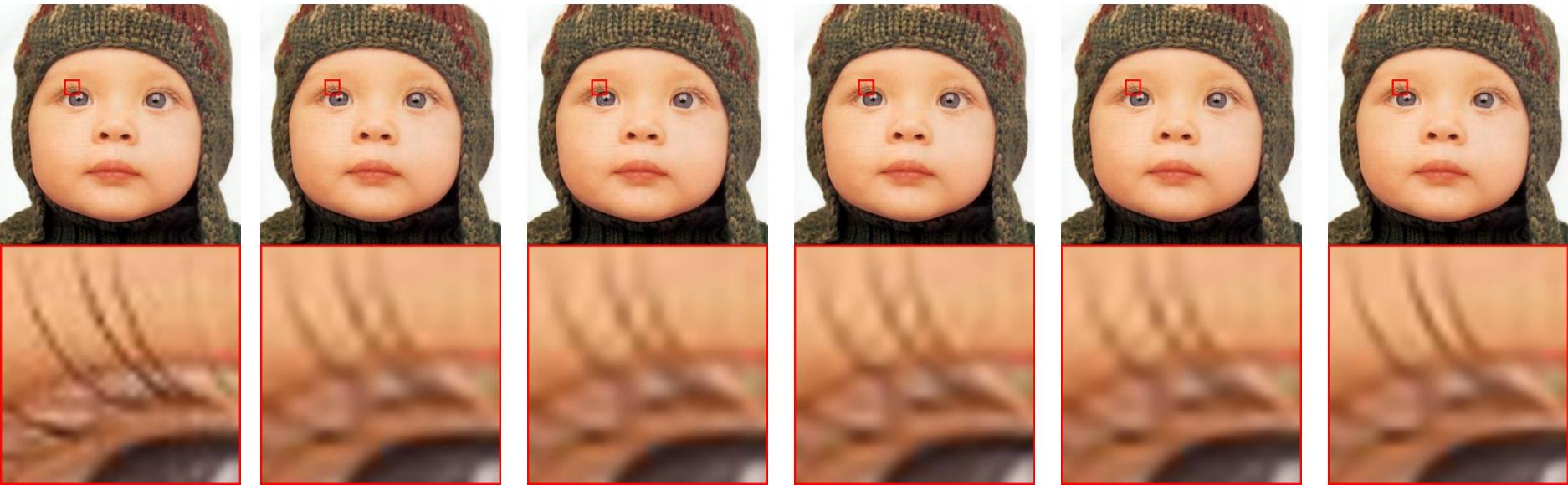
SelfEx [3]
32.89/0.8865



SRCNN [4]
32.99/0.8856



VDSR (ours)
33.77/0.8938



Ground truth
(PSNR/SSIM)

A+ [1]
38.52/0.9651

RFL [2]
38.51/0.9650

SelfEx [3]
38.44/0.9644

SRCNN [4]
38.54/0.9655

VDSR (ours)
38.75/0.9667



Ground truth
(PSNR/SSIM)

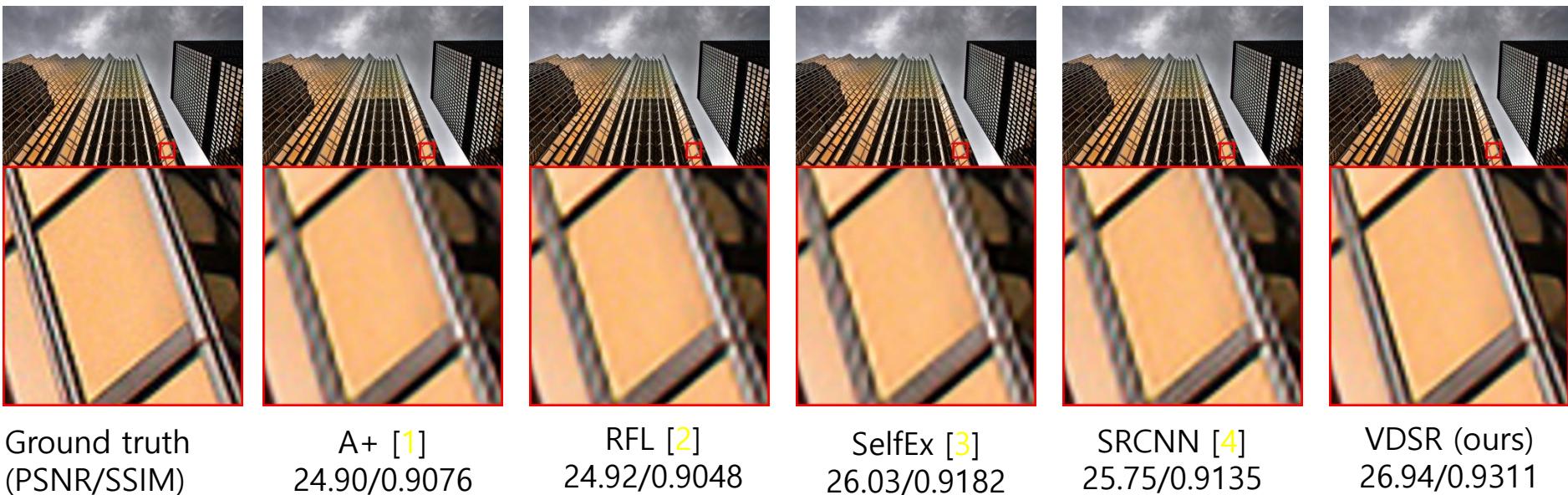
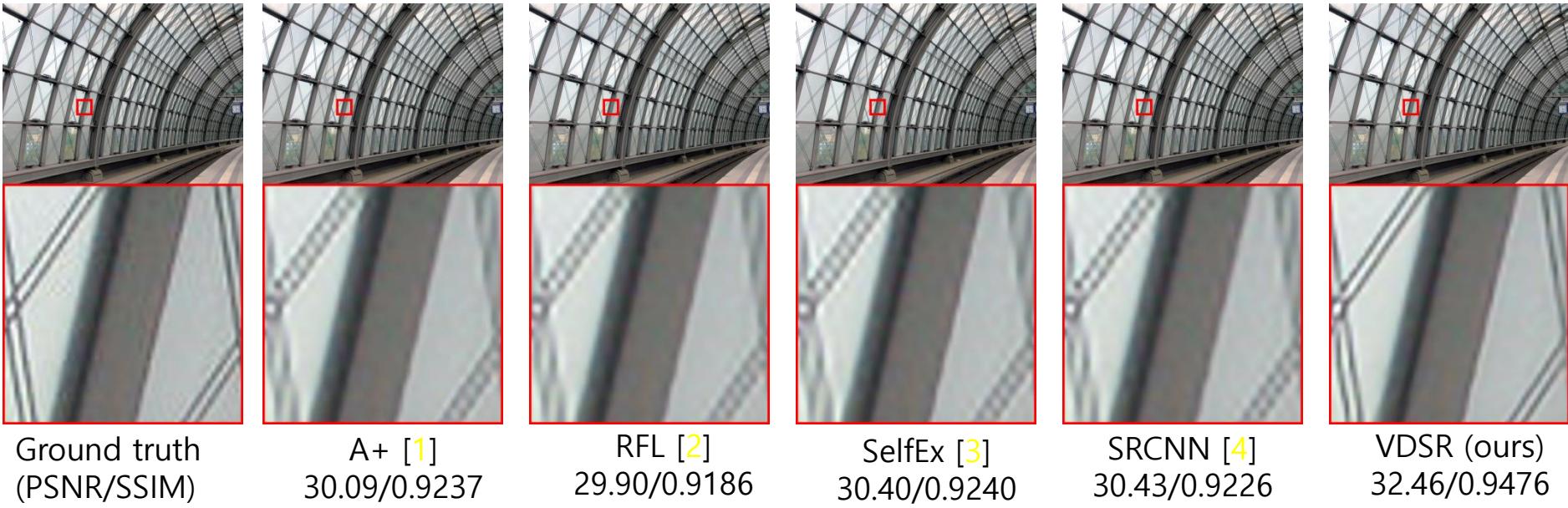
A+ [1]
34.29/0.9400

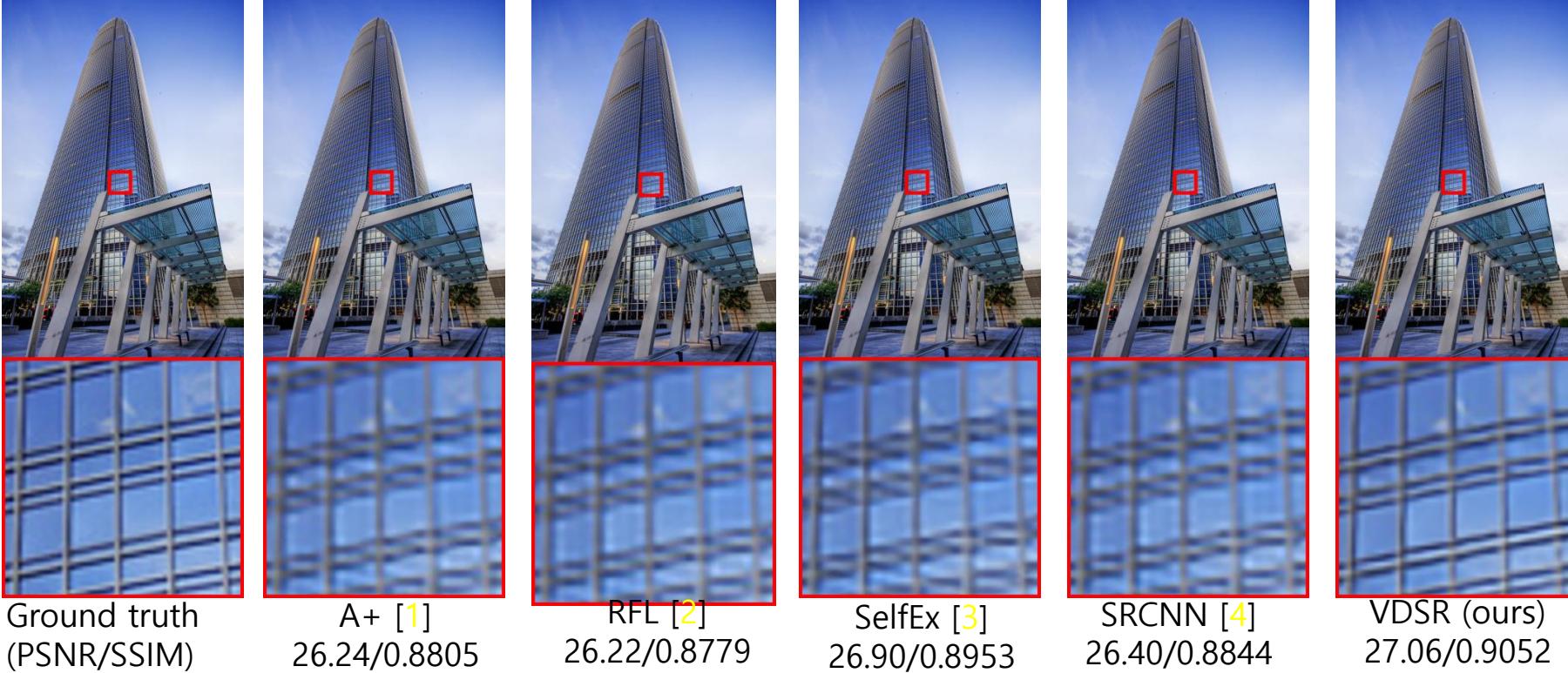
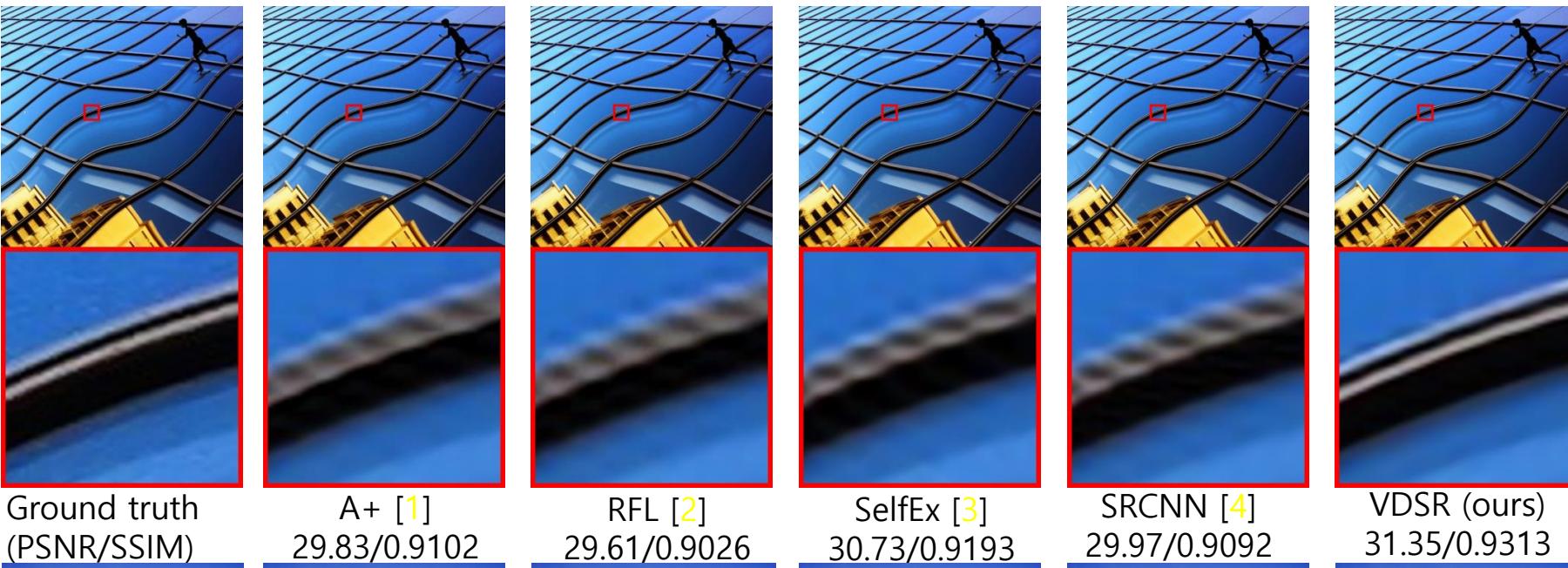
RFL [2]
33.90/0.9372

SelfEx [3]
34.21/0.9389

SRCNN [4]
33.81/0.9374

VDSR (ours)
34.99/0.9484







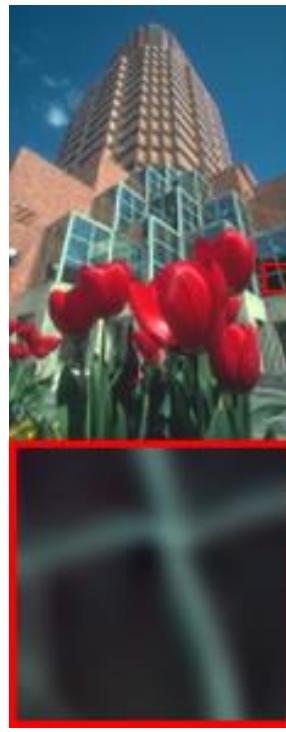
Ground truth
(PSNR/SSIM)



A+ [1]
25.04/0.7465



RFL [2]
24.94/0.7394



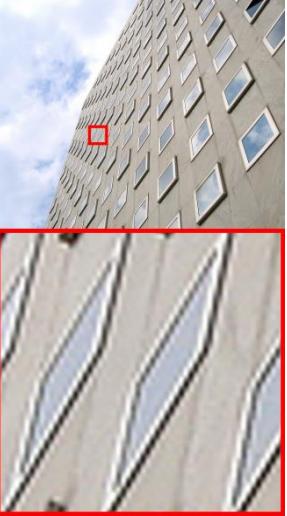
SelfEx [3]
25.39/0.7637



SRCNN [4]
25.00/0.7448



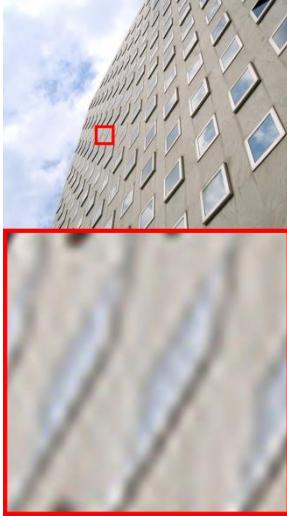
VDSR (ours)
25.66/0.7768



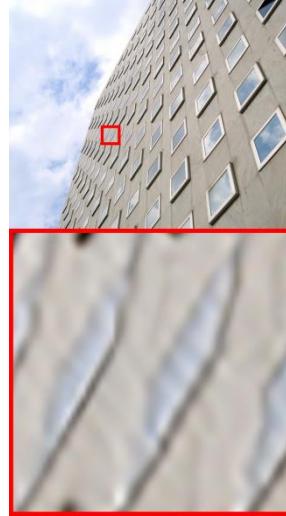
Ground truth
(PSNR/SSIM)



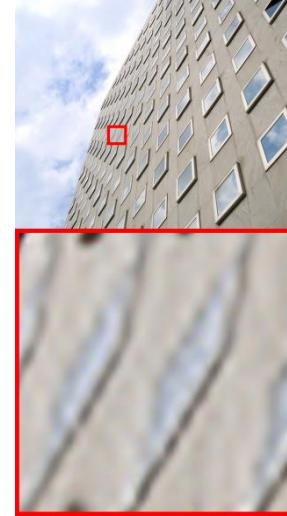
A+ [1]
26.15/0.8692



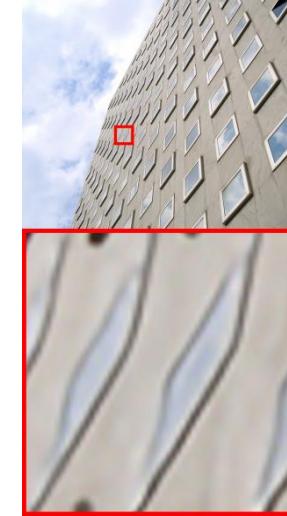
RFL [2]
25.79/0.8588



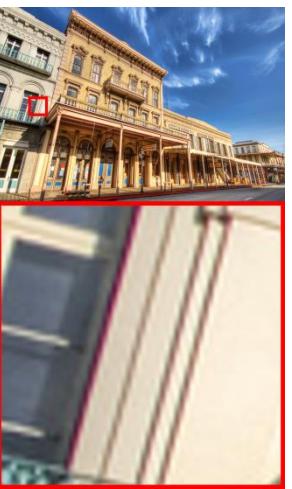
SelfEx [3]
26.33/0.8733



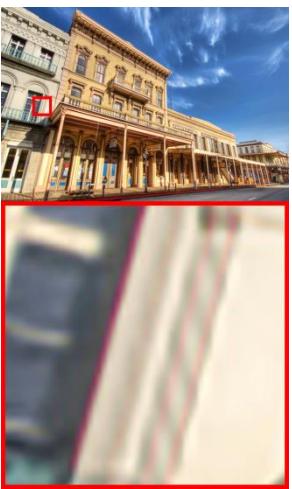
SRCNN [4]
26.39/0.8684



VDSR (ours)
28.18/0.9073



Ground truth
(PSNR/SSIM)



A+ [1]
25.78/0.8330



RFL [2]
25.69/0.8289



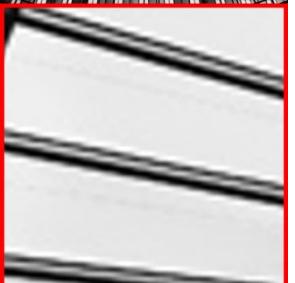
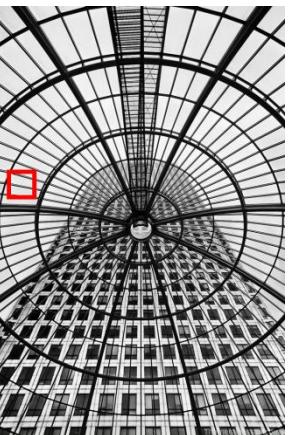
SelfEx [3]
25.89/0.8365



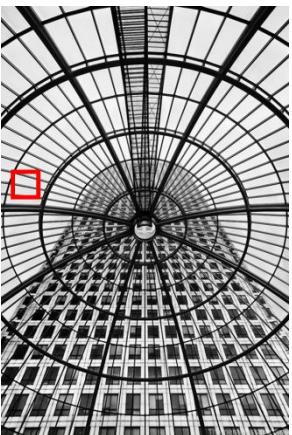
SRCNN [4]
25.93/0.8363



VDSR (ours)
26.91/0.8695



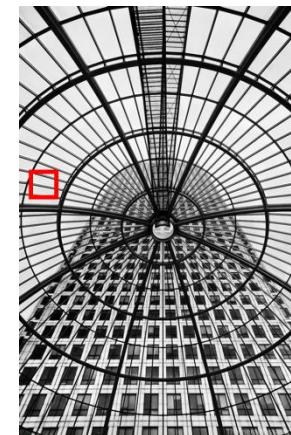
Ground truth
(PSNR/SSIM)



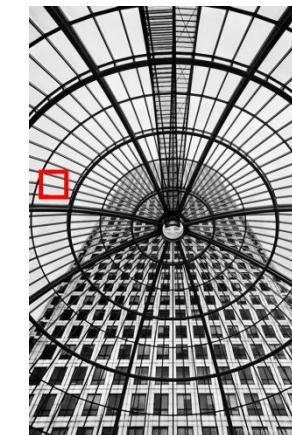
A+ [1]
22.84/0.9058



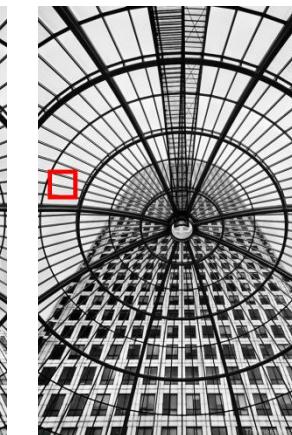
RFL [2]
22.77/0.9037



SelfEx [3]
22.89/0.9028



SRCNN [4]
23.54/0.9162



VDSR (ours)
25.13/0.9439

6. Conclusion

- 이번 연구에서 매우 깊은 네트워크를 사용한 super-resolution method 를 제시했다.
- 매우 깊은 네트워크에서의 훈련은 느린 수렴률 때문에 어렵다. 우리는 매우 깊은 네트워크를 빠르게 최적화하기 위해 **residual-learning** 과 **극도로 높은 학습률**을 사용한다.
- 수렴 속도는 최대화하고 훈련의 안정성을 보장하기 위해 **gradient clipping** 을 사용한다.
- Benchmarked images 에서의 large margin 에 의해 우리의 방법이 현존하는 방법들을 능가한다.
- VDSR 이 denoising 과 compression artifact removal 과 같은 다른 이미지 복구 문제에 순조롭게 적용할 수 있다고 생각한다.

Results

Results





Results

In [9]:

```
1 # Let's see how long does it take for processing
2 start_time = time.time()
3 out = model(im_input)
4 elapsed_time = time.time() - start_time
5 print("It takes {}s for processing in cpu mode".format(elapsed_time))
```

It takes 5.04360198975s for processing in cpu mode

In [11]:

```
1 # Let's see how long does it take for processing in gpu mode
2 start_time = time.time()
3 out = model(im_input)
4 elapsed_time = time.time() - start_time
5 print("It takes {}s for processing in gpu mode".format(elapsed_time))
```

It takes 0.272584915161s for processing in gpu mode

Results





Results

In [9]:

```
1 # Let's see how long does it take for processing
2 start_time = time.time()
3 out = model(im_input)
4 elapsed_time = time.time() - start_time
5 print("It takes {}s for processing in cpu mode".format(elapsed_time))
```

It takes 5.06392908096s for processing in cpu mode

In [11]:

```
1 # Let's see how long does it take for processing in gpu mode
2 start_time = time.time()
3 out = model(im_input)
4 elapsed_time = time.time() - start_time
5 print("It takes {}s for processing in gpu mode".format(elapsed_time))
```

It takes 0.464923858643s for processing in gpu mode

Results





Results

In [9]:

```
1 # Let's see how long does it take for processing
2 start_time = time.time()
3 out = model(im_input)
4 elapsed_time = time.time() - start_time
5 print("It takes {}s for processing in cpu mode".format(elapsed_time))
```

It takes 4.94673991203s for processing in cpu mode

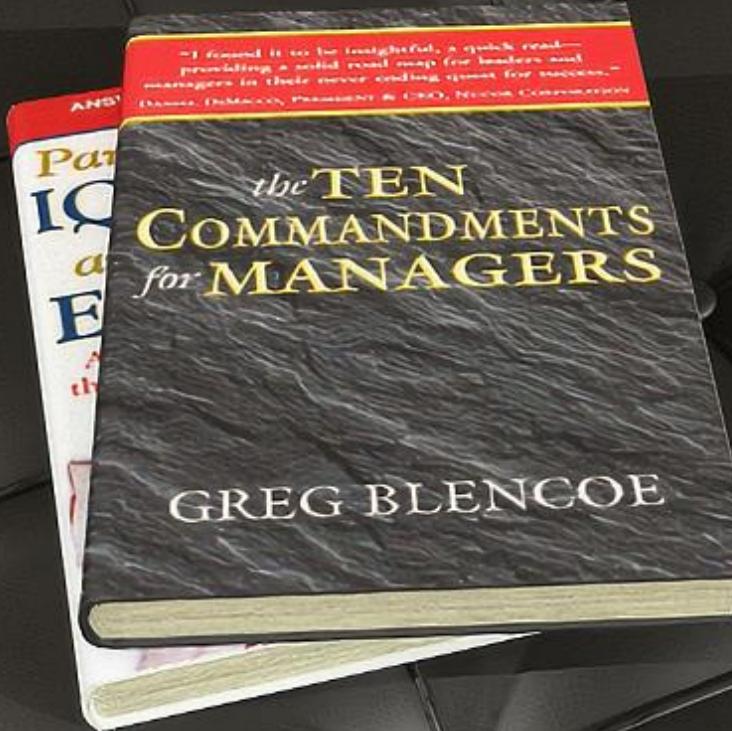
In [11]:

```
1 # Let's see how long does it take for processing in gpu mode
2 start_time = time.time()
3 out = model(im_input)
4 elapsed_time = time.time() - start_time
5 print("It takes {}s for processing in gpu mode".format(elapsed_time))
```

It takes 0.514549970627s for processing in gpu mode

Results





ANSI

Part
IC
a
E

GREG BLENCOE

Results

In [9]:

```
1 # Let's see how long does it take for processing
2 start_time = time.time()
3 out = model(im_input)
4 elapsed_time = time.time() - start_time
5 print("It takes {}s for processing in cpu mode".format(elapsed_time))
```

It takes 5.00729489326s for processing in cpu mode

In [11]:

```
1 # Let's see how long does it take for processing in gpu mode
2 start_time = time.time()
3 out = model(im_input)
4 elapsed_time = time.time() - start_time
5 print("It takes {}s for processing in gpu mode".format(elapsed_time))
```

It takes 0.455554008484s for processing in gpu mode

Results





Results

In [9]:

```
1 # Let's see how long does it take for processing
2 start_time = time.time()
3 out = model(im_input)
4 elapsed_time = time.time() - start_time
5 print("It takes {}s for processing in cpu mode".format(elapsed_time))
```

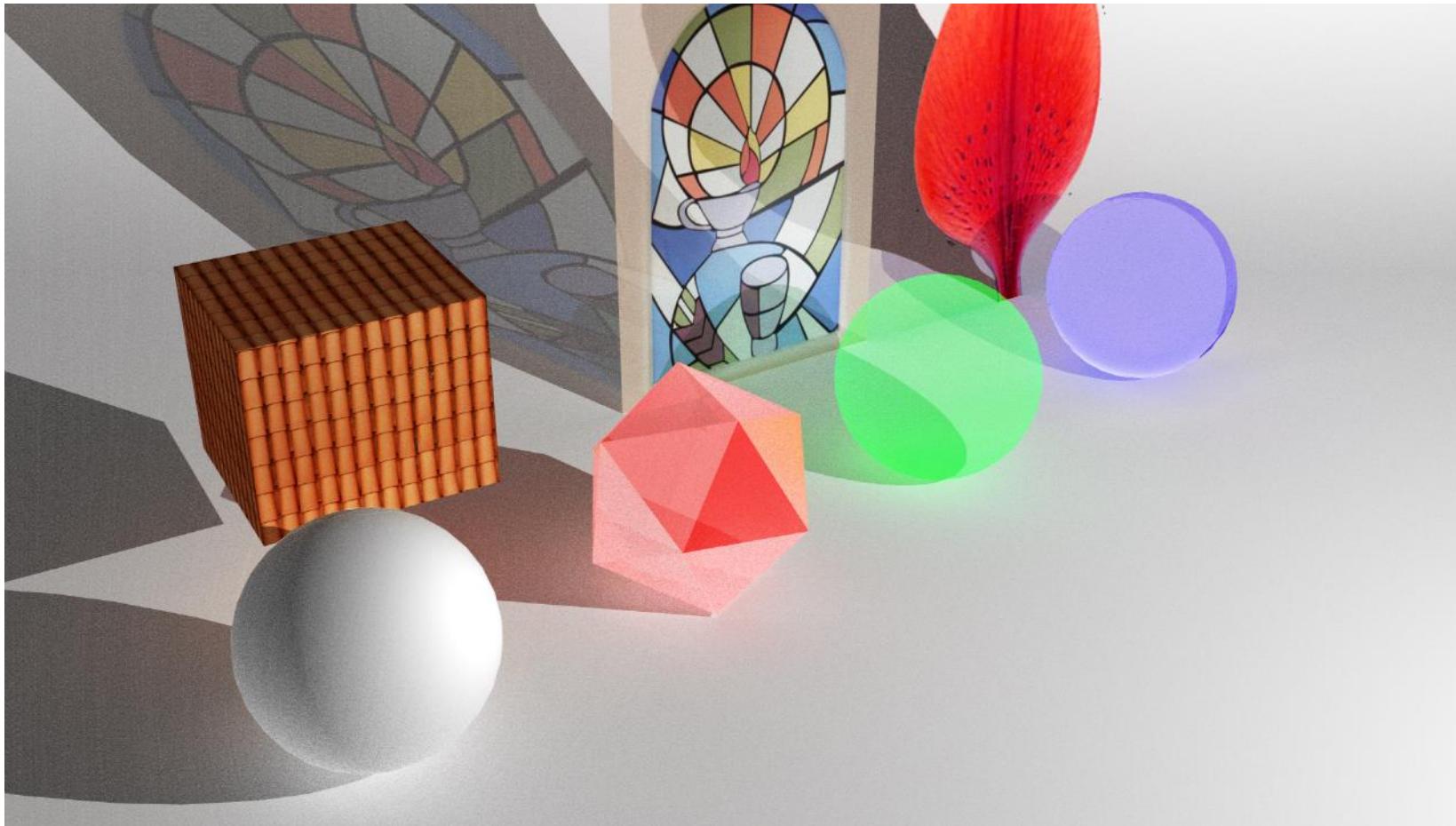
It takes 4.90062594414s for processing in cpu mode

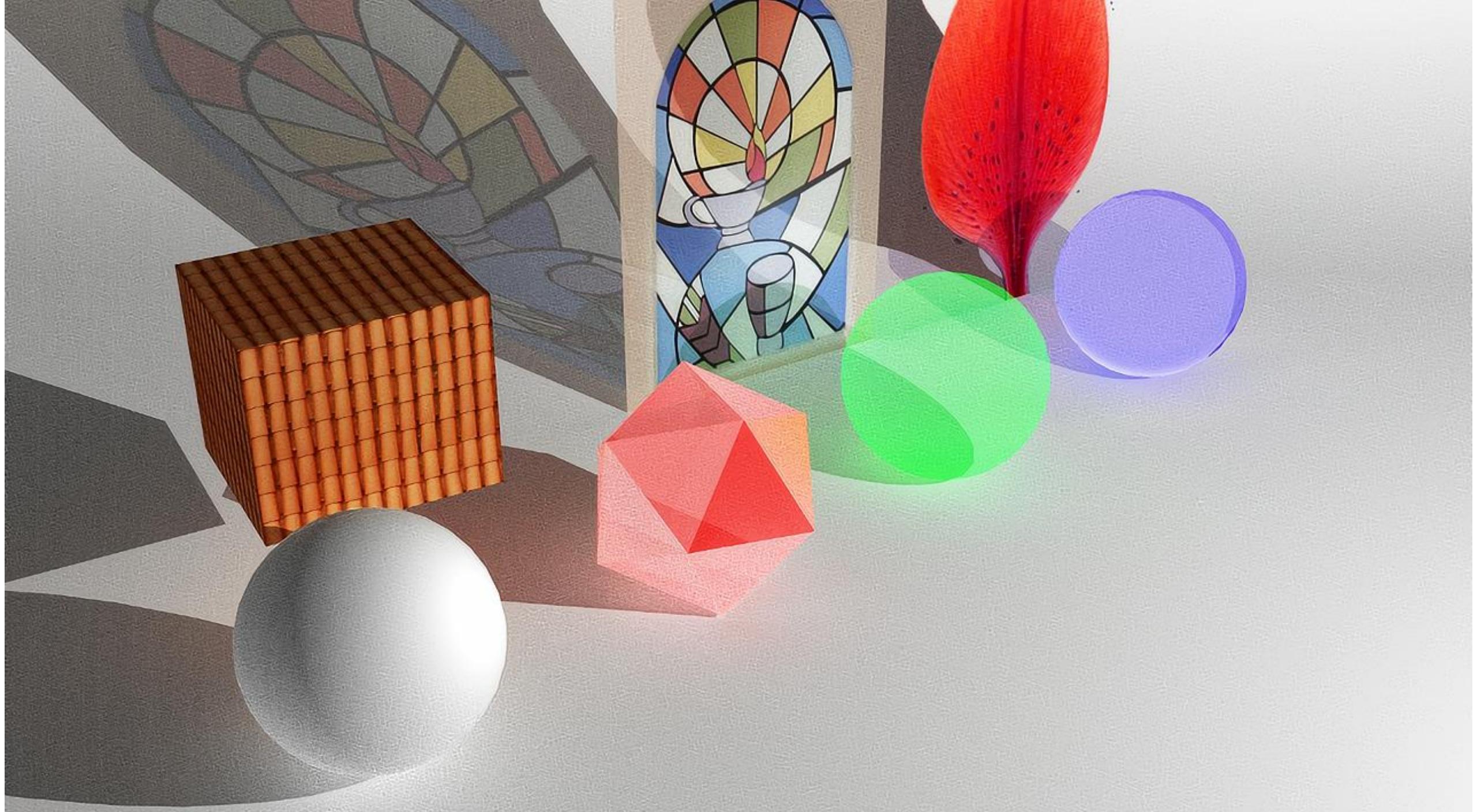
In [11]:

```
1 # Let's see how long does it take for processing in gpu mode
2 start_time = time.time()
3 out = model(im_input)
4 elapsed_time = time.time() - start_time
5 print("It takes {}s for processing in gpu mode".format(elapsed_time))
```

It takes 0.41789317131s for processing in gpu mode

Results





Results

In [9]:

```
1 # Let's see how long does it take for processing
2 start_time = time.time()
3 out = model(im_input)
4 elapsed_time = time.time() - start_time
5 print("It takes {}s for processing in cpu mode".format(elapsed_time))
```

It takes 4.95740318298s for processing in cpu mode

In [11]:

```
1 # Let's see how long does it take for processing in gpu mode
2 start_time = time.time()
3 out = model(im_input)
4 elapsed_time = time.time() - start_time
5 print("It takes {}s for processing in gpu mode".format(elapsed_time))
```

It takes 0.263444900513s for processing in gpu mode

Thank you for listening
