AI VIETNAM
All-in-One Course

Image Domain Conversion

Upsampling Alternatives and Applications

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2023

Outline

- Segmentation (Unet Using Interpolation)
- Alternatives to Increate Feature Resolution
- Colorization
- Super-resolution
- Denoising Survey
- Super-resolution Survey

Discussion – Segmentation Problem

Input/Output + Loss Function

Model

Inference

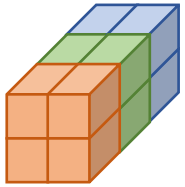
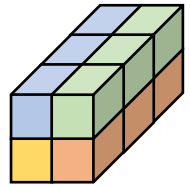


image
(2x2)

Model



Output
(3x2x2)

0	2
2	1

Predicted
Segmentation

Training

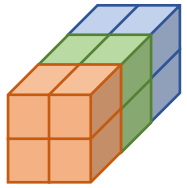
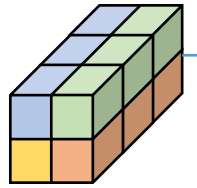


image
(2x2)

Model



Output
(3x2x2)

Cross
Entropy

1	0
2	0

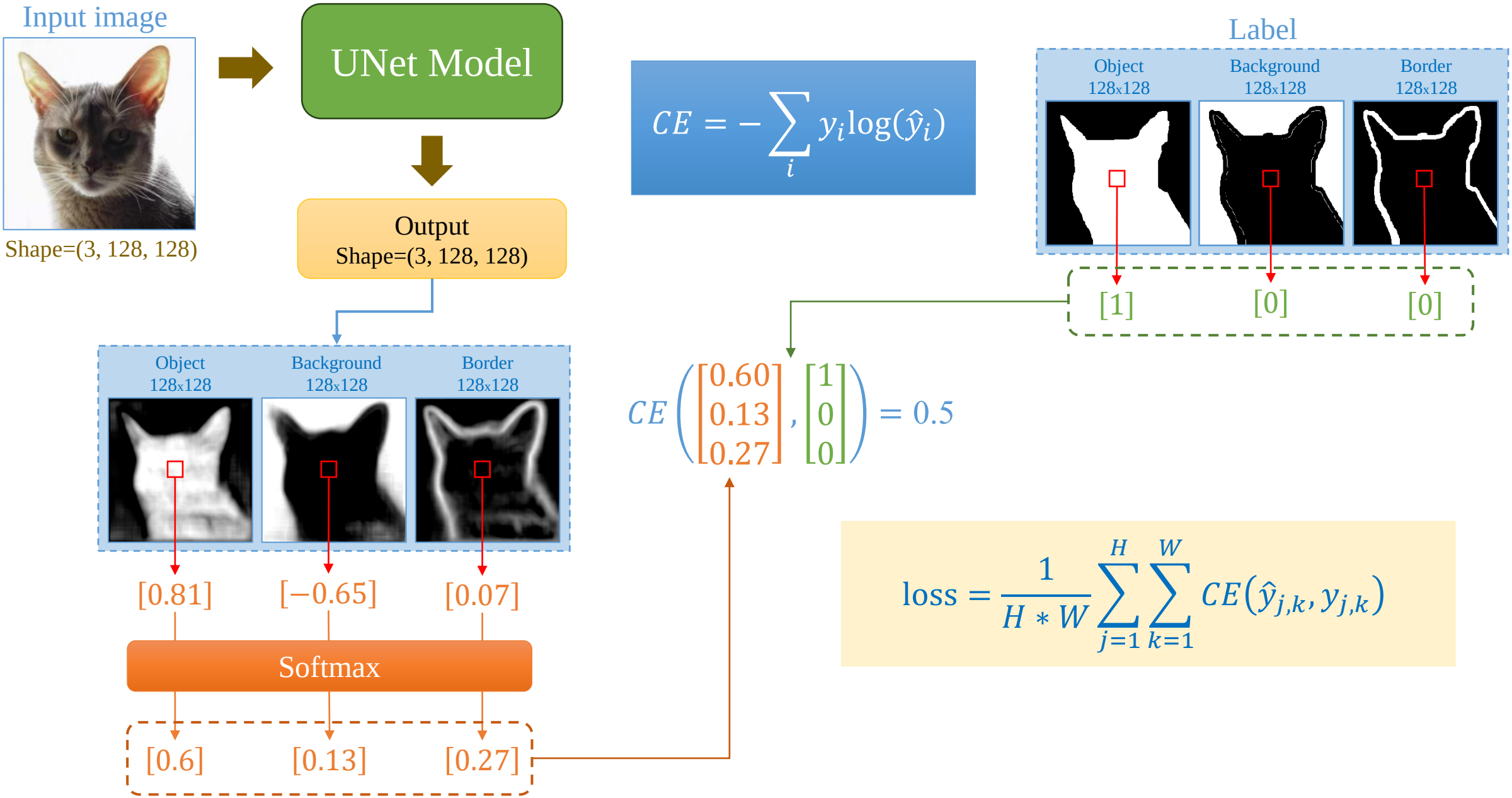
Ground-truth

one-hot encoding

0	1	0
0	0	1

1	0	0
1	0	0

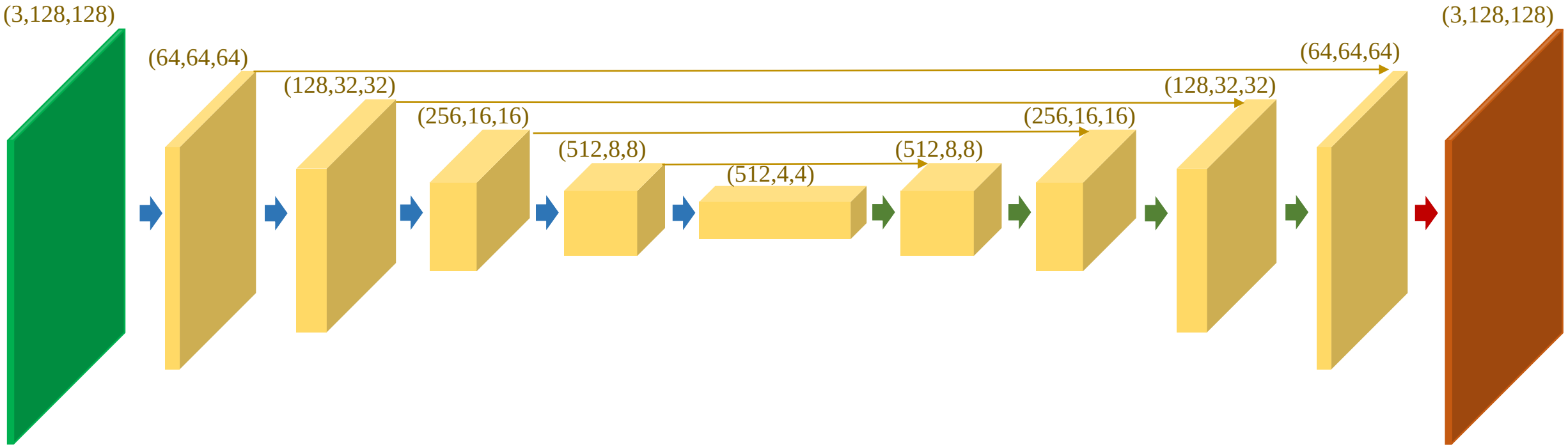
Loss Function



Segmentation (6)

Using skip connections

Code Reading



(3x3) Convolution padding = 'same' stride = 1 + ReLU + Batch Norm + (2x2) max pooling

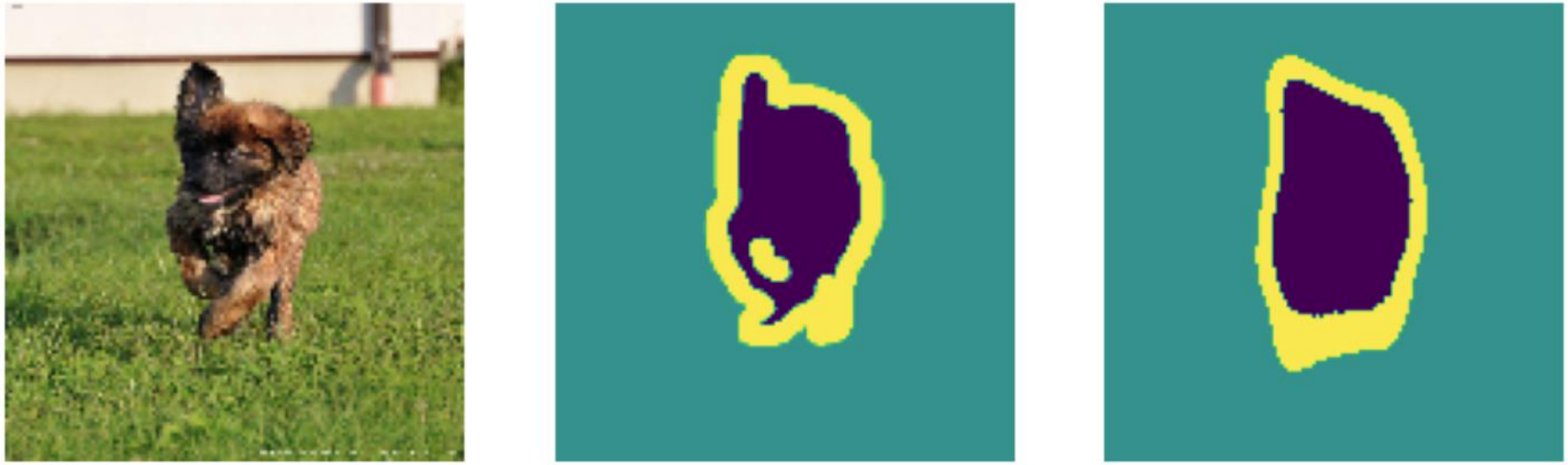
Feature Map Upsamping + (3x3) Convolution padding = 'same' stride = 1 + ReLU + Batch Norm

Feature Map Upsamping + (3x3) Convolution padding = 'same' softmax

Experimental Results

With Skip Connections

Without Skip Connections

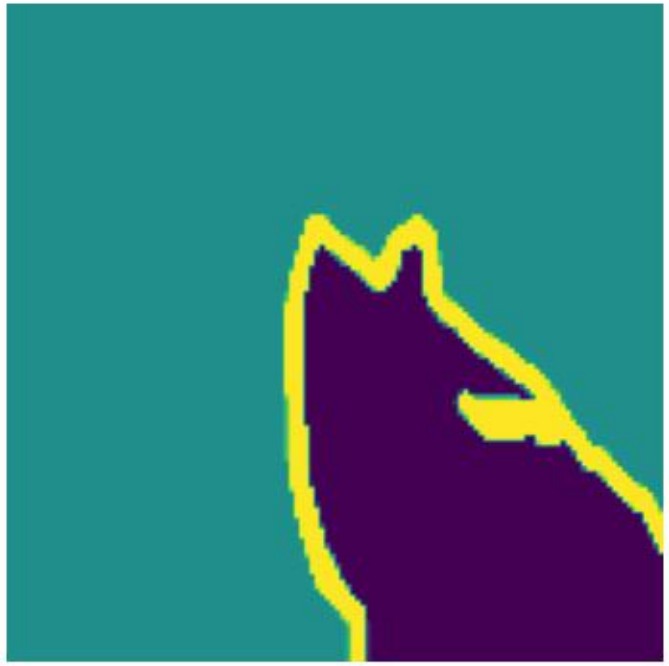


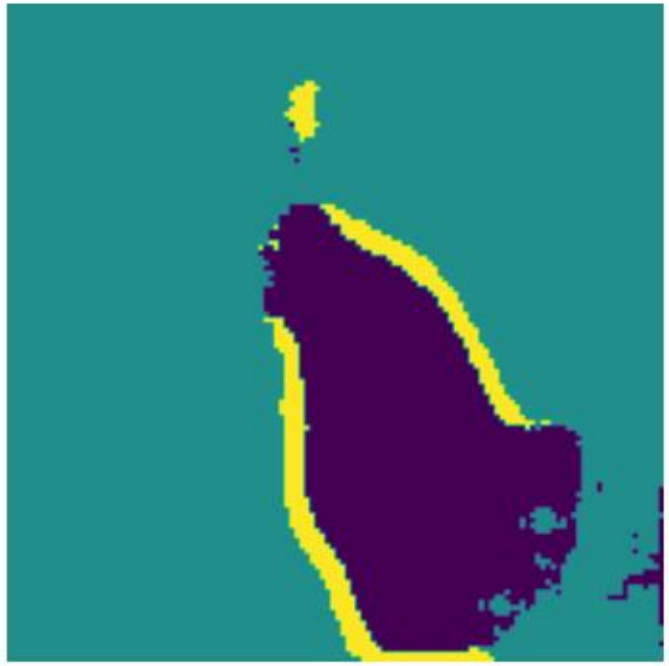
Input Image

True Mask

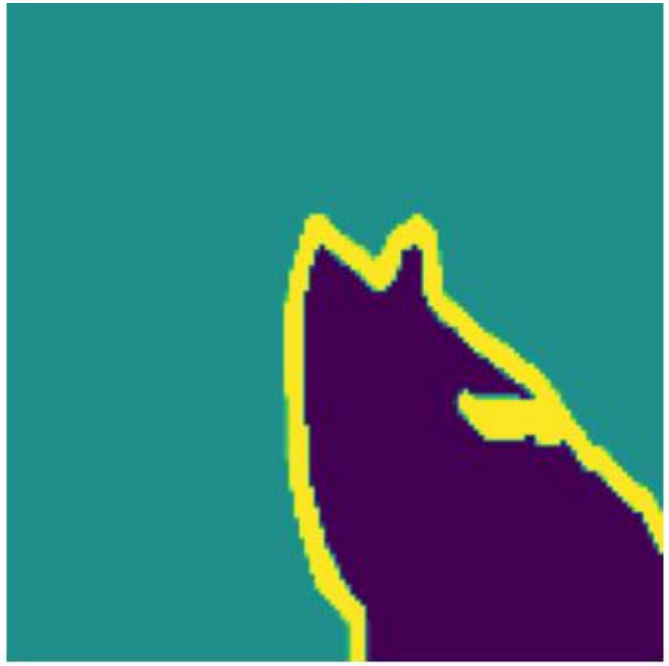
Predicted Mask

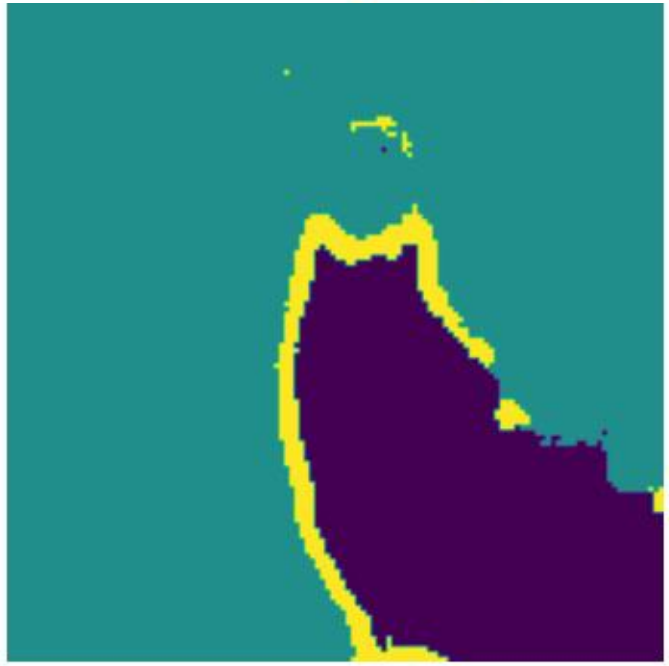
Without Skip Connections





With Skip Connections



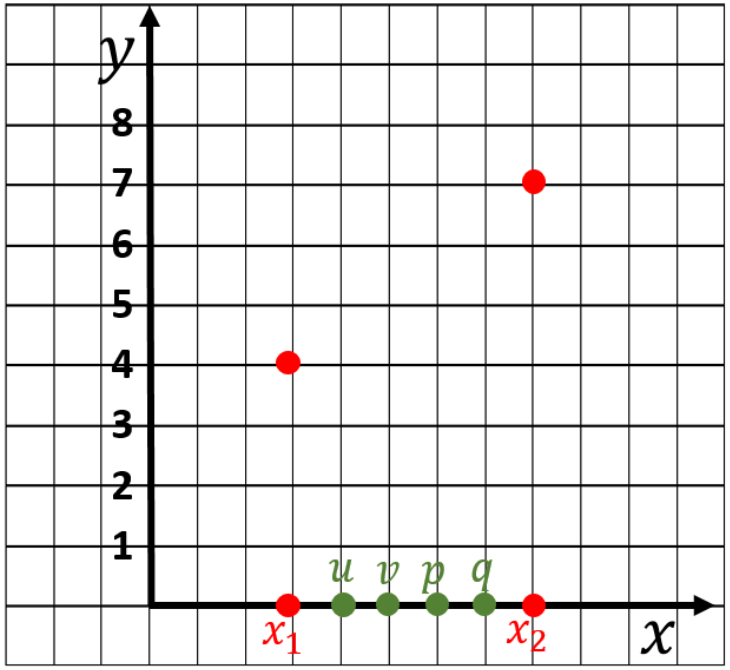


Outline

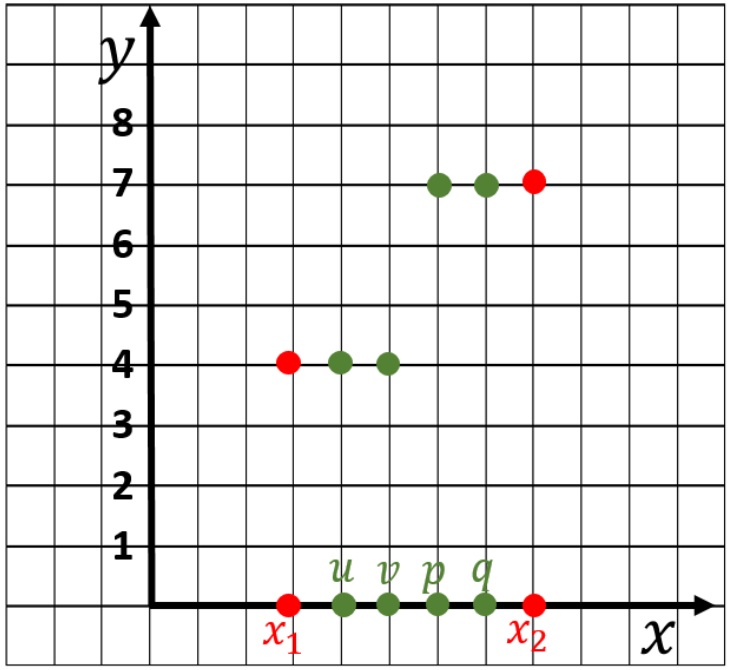
- Segmentation (Unet Using Interpolation)
- Alternatives to Increate Feature Resolution
- Colorization
- Super-resolution
- Denoising Survey
- Super-resolution Survey

How to Increase Feature Map

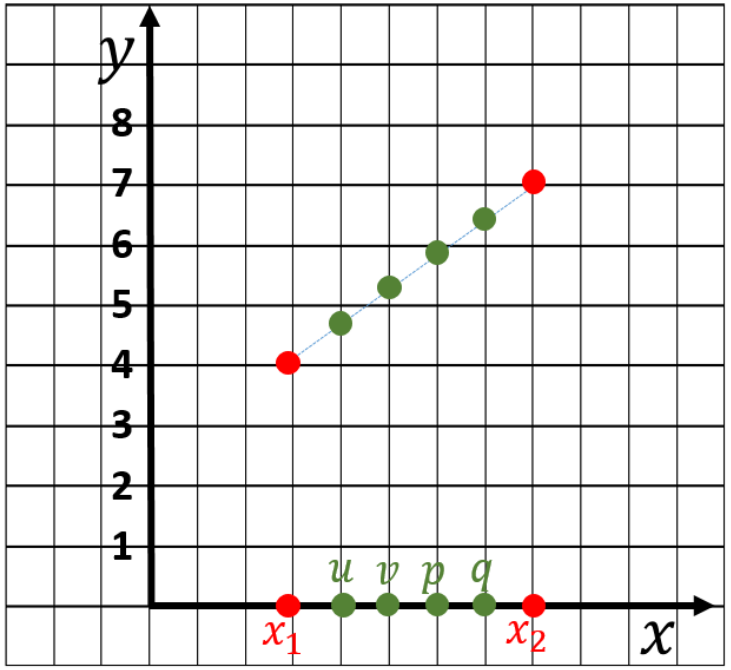
- Image upsampling
  - Data interpolation



Tìm giá trị cho các vị trí u, v, p và q



Nearest neighbor: Tính khoảng cách đến x_1 và x_2, và lấy giá trị của x gần hơn

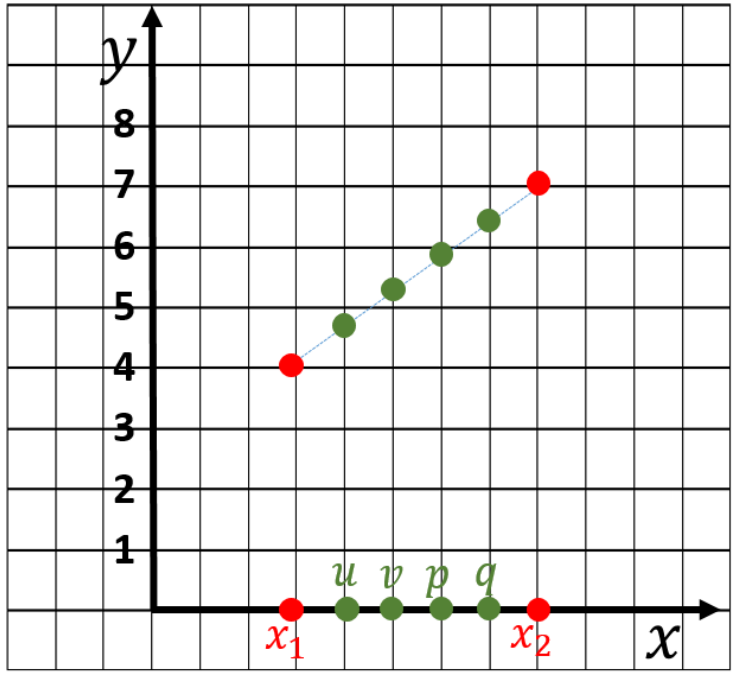


Nội suy theo hàm tuyến tính

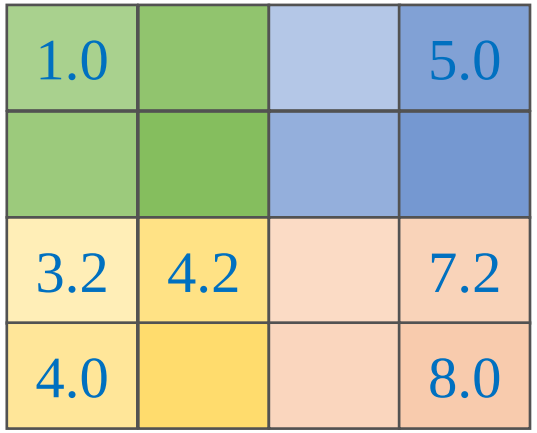
How to Increase Feature Map

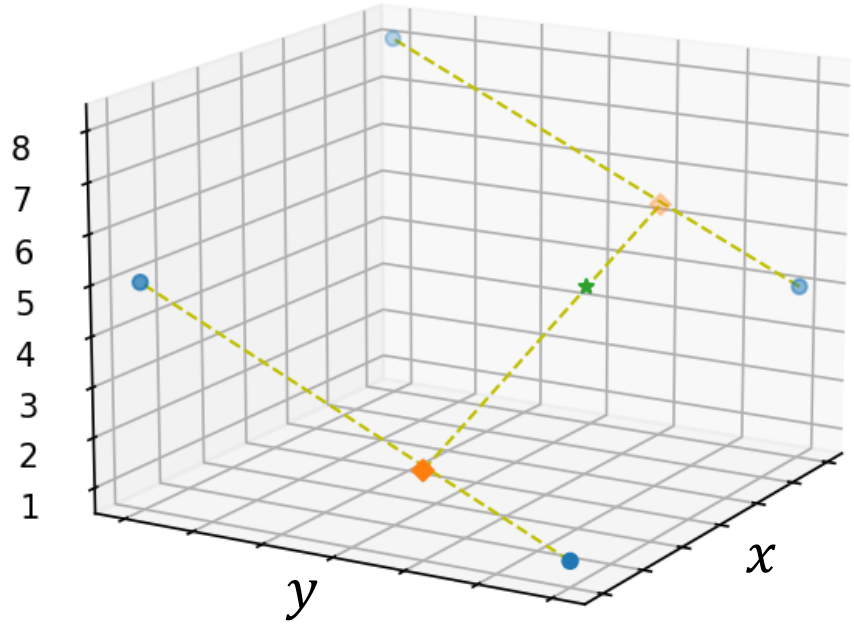
❖ Image upsampling

❖ Data interpolation



Nội suy theo hàm tuyến tính

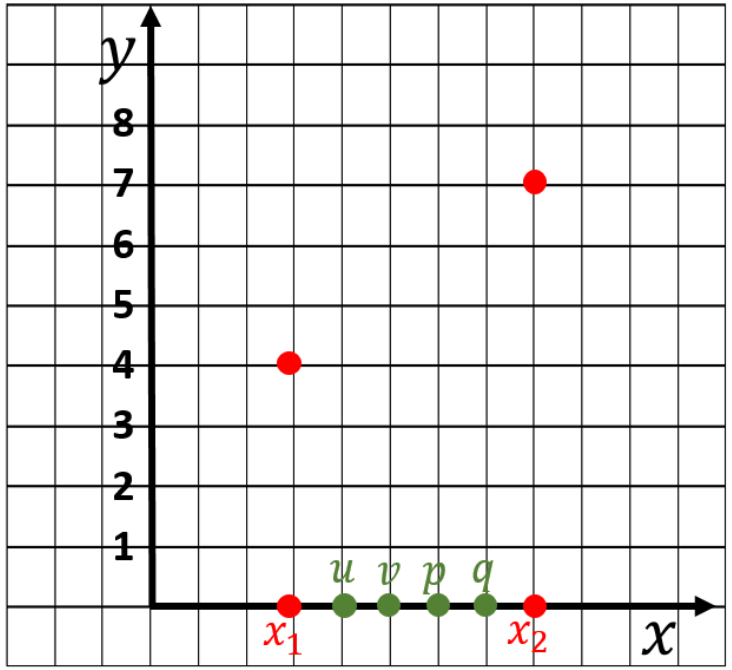




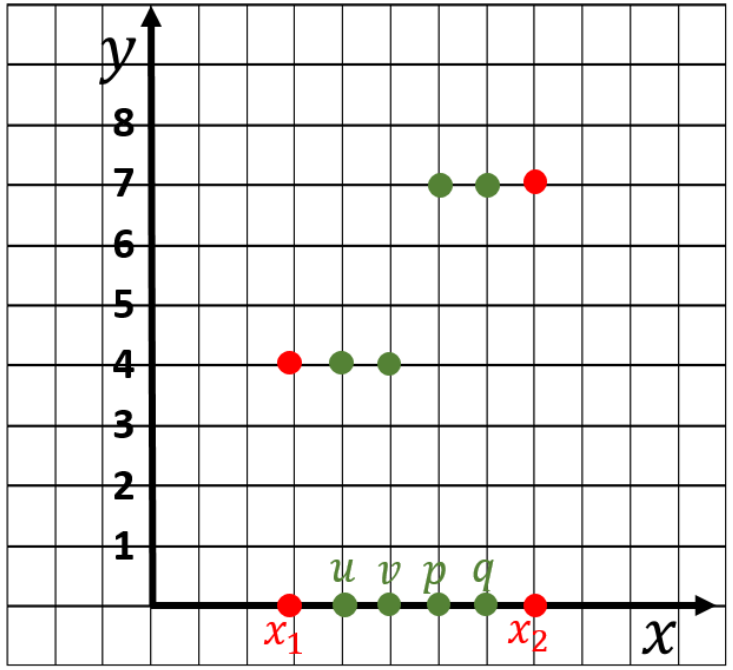
Bilinear interpolation

How to Increase Feature Map

❖ Nearest neighbor interpolation



Tìm giá trị cho các vị trí u, v, p và q



Nearest neighbor: Tính khoảng cách đến x_1 và x_2, và lấy giá trị của x gần hơn

data =

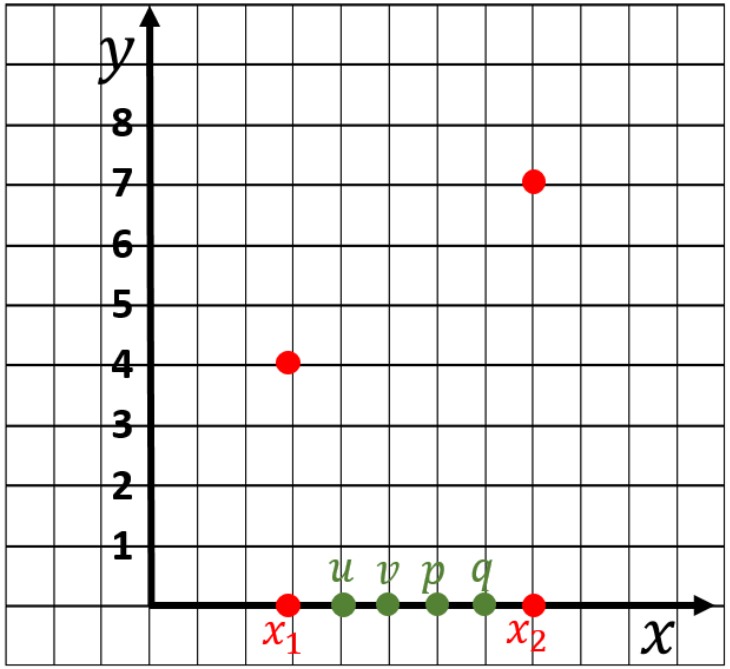
1	5
4	8

1	1	5	5
1	1	5	5
4	4	8	8
4	4	8	8

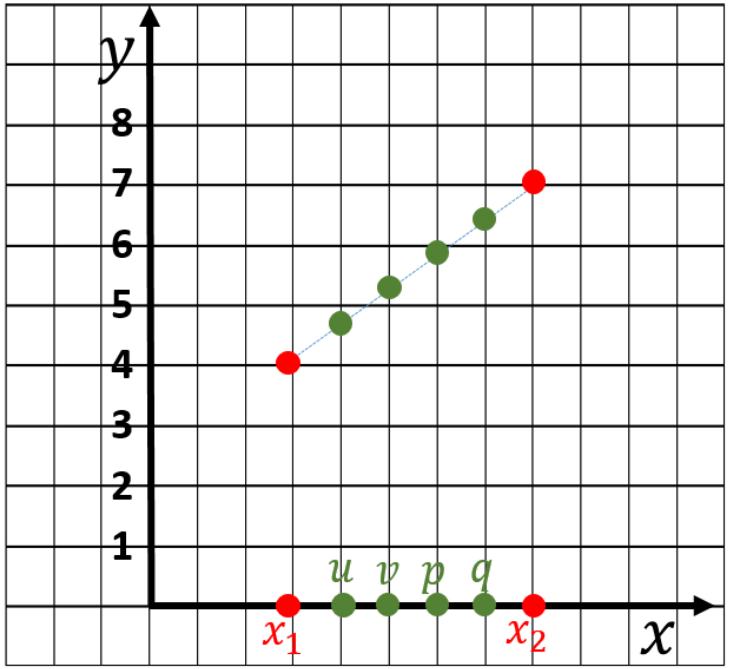
Nearest neighbor interpolation

How to Increase Feature Map

❖ Bilinear interpolation



Tìm giá trị cho các vị trí u, v, p và q



Nội suy theo hàm tuyến tính

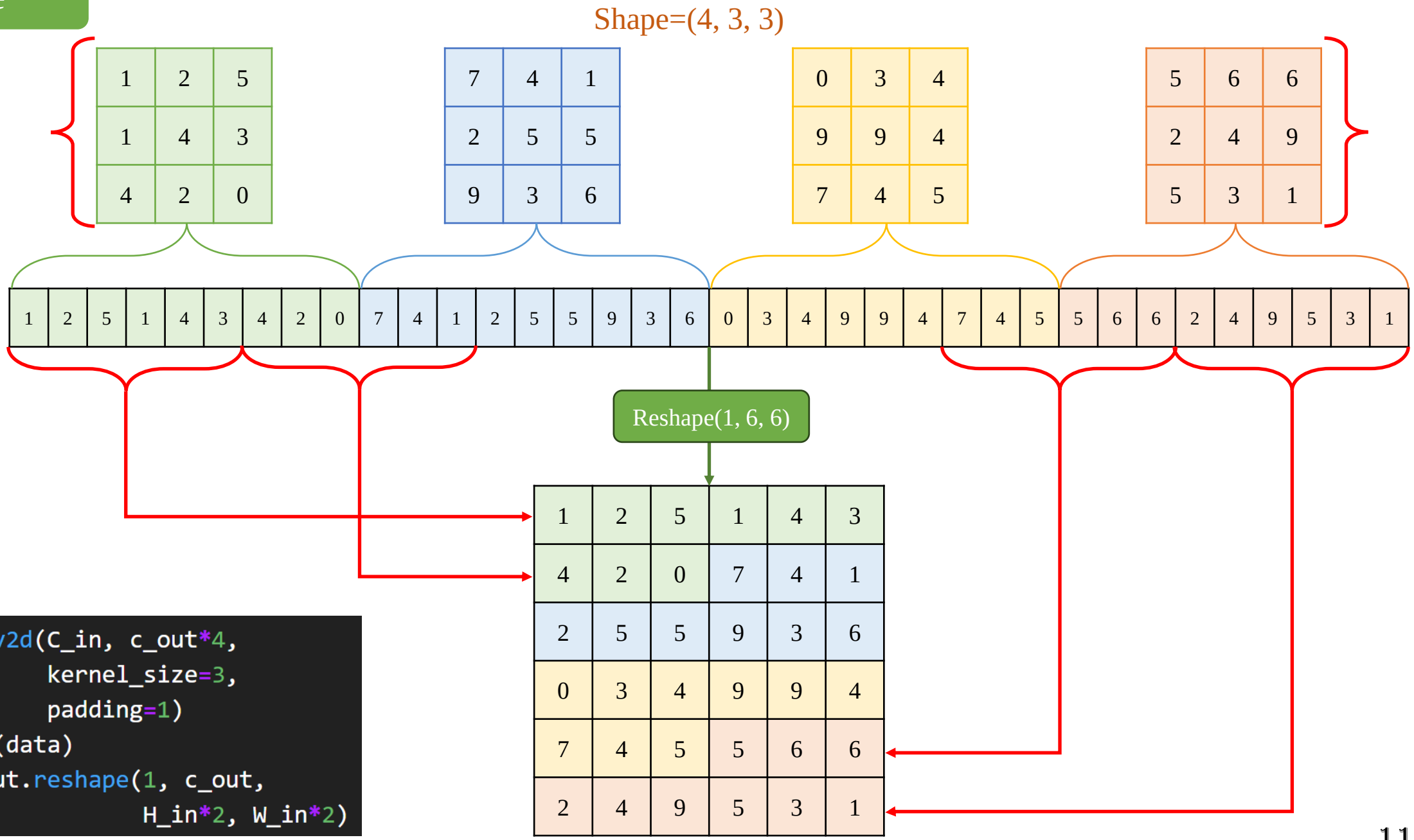
data =

1	5
4	8

1.0	2.0	4.0	5.0
1.7	2.7	4.7	5.7
3.2	4.2	6.2	7.2
4.0	5.0	7.0	8.0

Bilinear interpolation

Reshape



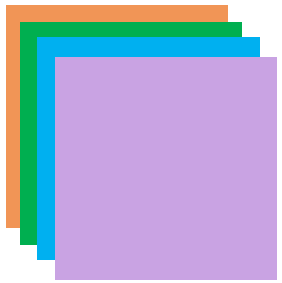

```
conv = nn.Conv2d(C_in, c_out*4,
                 kernel_size=3,
                 padding=1)
output = conv(data)
output = output.reshape(1, c_out,
                        H_in*2, W_in*2)
```

How to Increase Feature Map

❖ Pixel Shuffle

$$(B, C \times r^2, H, W) \Rightarrow (B, C, H \times r, W \times r)$$

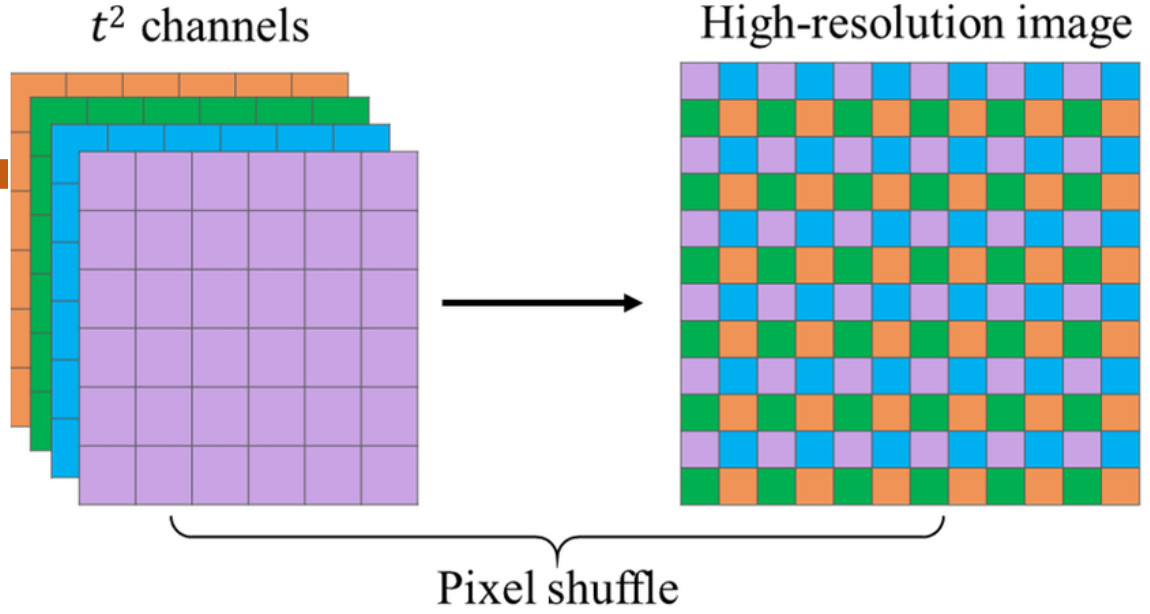
7	5	6	4	7	8	3	8
9	5	0	4	2	0	6	9



data

7	6	5	4
7	3	8	8
9	0	5	4
2	6	0	9

output



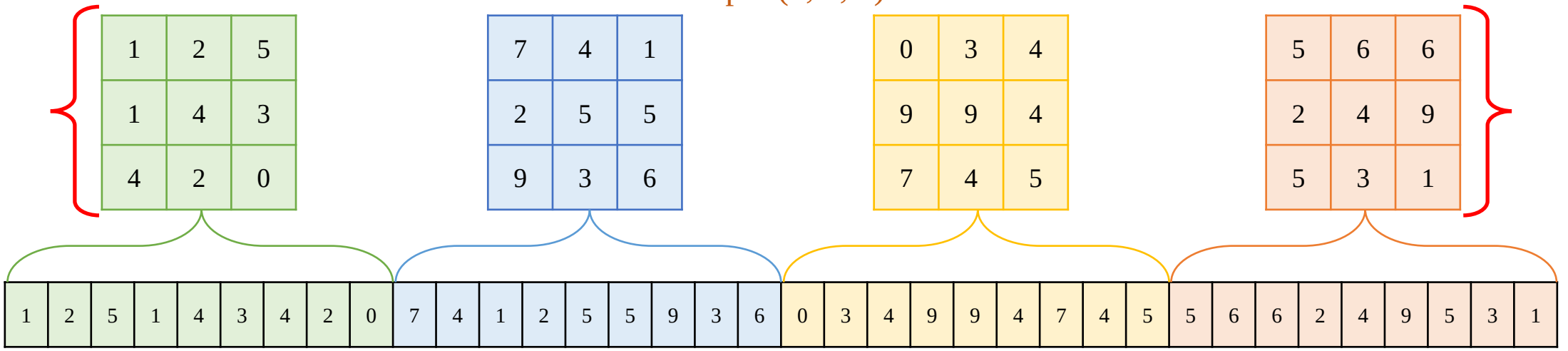
https://www.researchgate.net/figure/The-pixel-shuffle-layer-transforms-feature-maps-from-the-LR-domain-to-the-HR-image_fig3_339531308

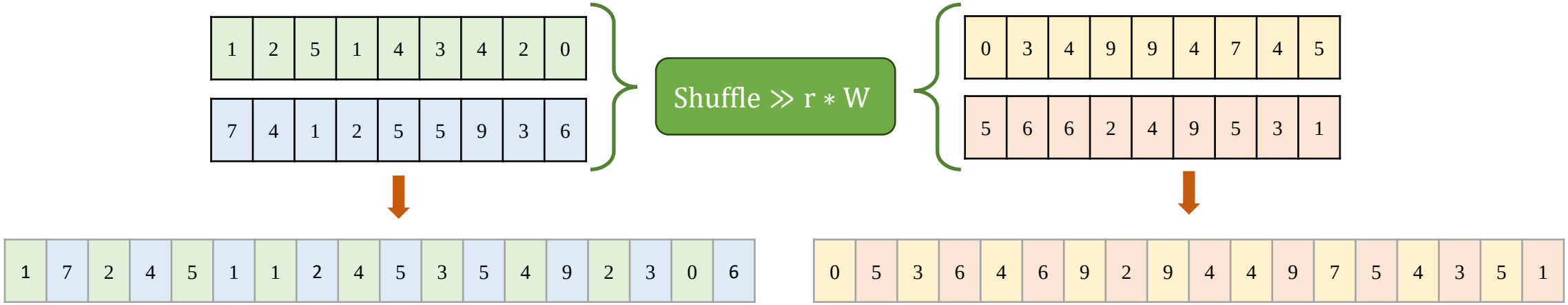
```
decoder = nn.Sequential(
            nn.Conv2d(in_channels,
                      out_channels*4,
                      kernel_size=3,
                      padding=1),
            # ...
            nn.PixelShuffle(2))
# forward
x = decoder(x)
```

PixelShuffle

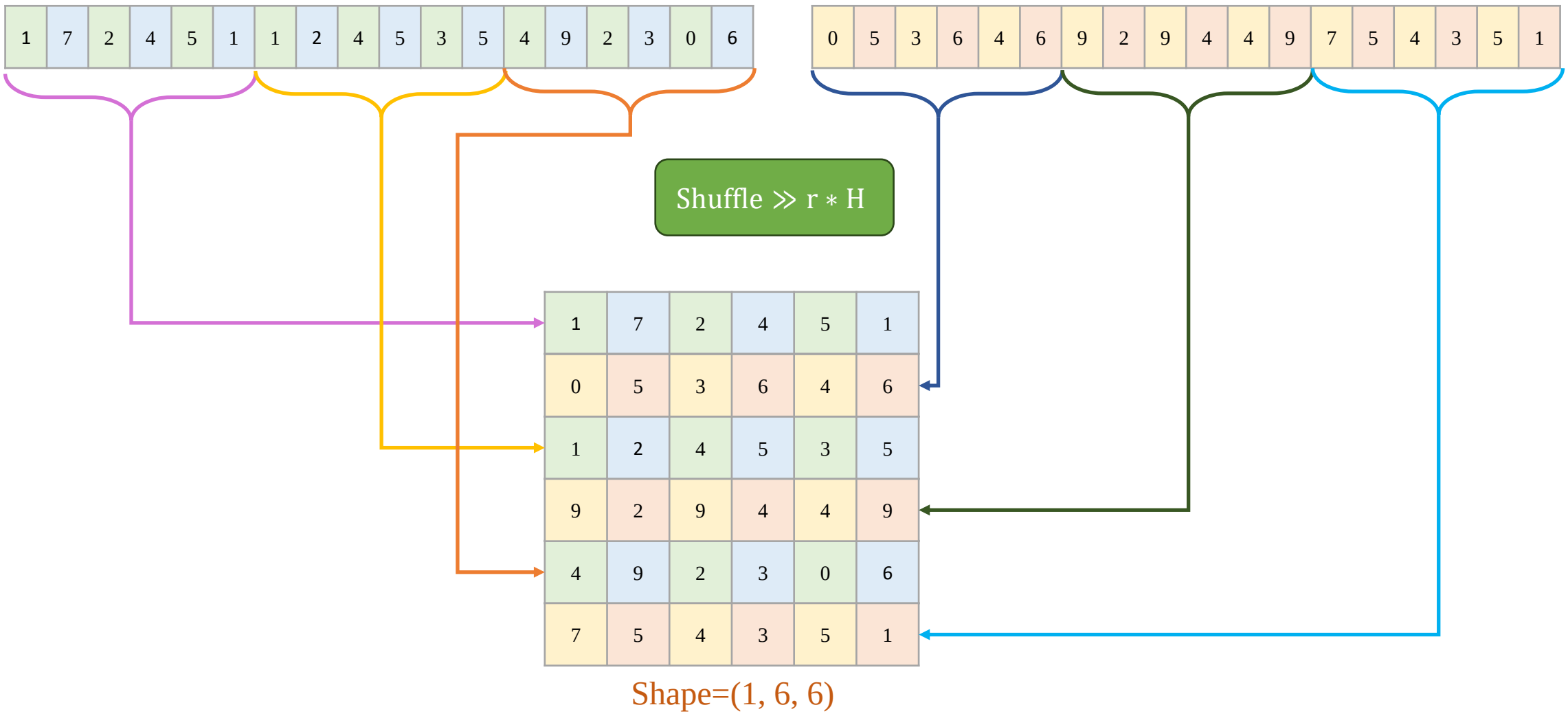
$(C * r^2, W, H) \rightarrow (C, r * W, r * H)$

Shape=(4, 3, 3)

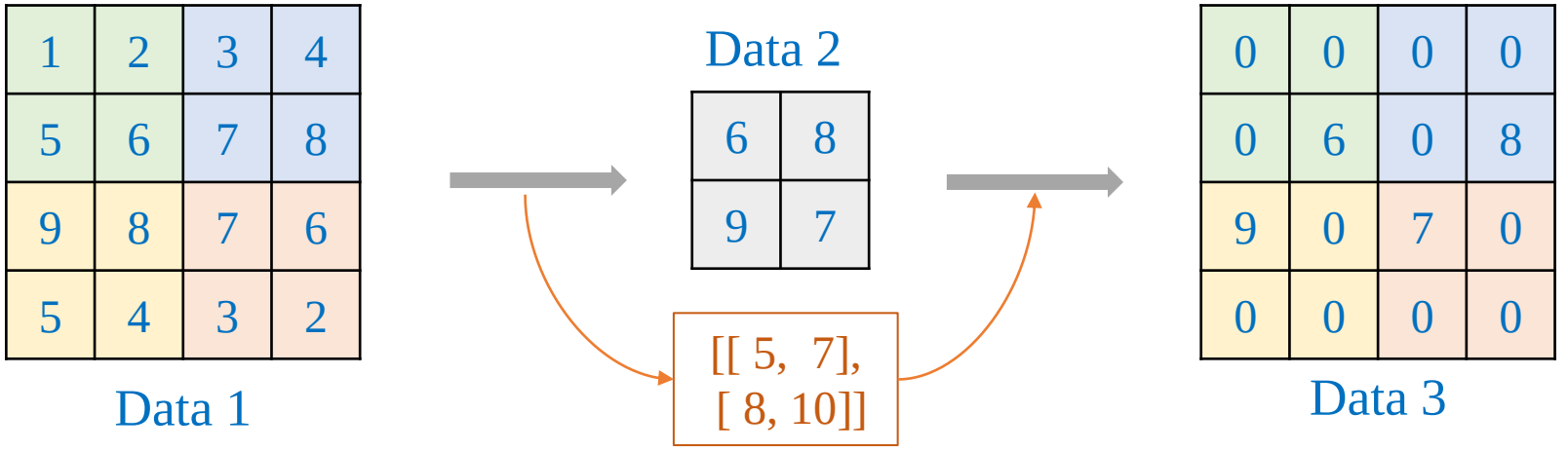




PixelShuffle



Unpooling




```
data = torch.Tensor([[1, 2, 3, 4],
                     [5, 6, 7, 8],
                     [9, 8, 7, 6],
                     [5, 4, 3, 2]])
data = data.reshape(1, 1, 4, 4)

mpool = nn.MaxPool2d(2, return_indices=True)
output_mp, indices = mpool(data)

unpooling = nn.MaxUnpool2d(kernel_size=2)
output_mup = unpooling(output_mp, indices)
```

```
# declaration
encoder = nn.Sequential(
            nn.Conv2d(in_channels, out_channels,
                      kernel_size=3, padding=1),
            # ...
            nn.MaxPool2d(2, return_indices=True))
decoder = nn.Sequential(
            nn.MaxUnpool2d(kernel_size=2),
            # ...
            nn.Conv2d(in_channels, out_channels,
                      kernel_size=3, padding=1))
# forward
x, indices = encoder(x)
x = decoder(x, indices)
```

Convolution Transpose

❖ Example 1

tensorflow

```
Conv2DTranspose(filters=1, kernel_size=2, strides=1, padding='valid')
```

pytorch

```
ConvTranspose2d(in_channels=1, out_channels=1, kernel_size=2, stride=1, padding=0)
```

Stride S = 1

Padding p = 0

Transpose padding
$p' = K - 1 = 1$

!

data =

1	2
3	4

$D = 2$

0	0	0	0
0	1	2	0
0	4	3	0
0	0	0	0

kernel =

0.1	-0.2
0.3	-0.1

K = 2

0	0	0	0
0	1	2	0
0	4	3	0
0	0	0	0

*

0.1	-0.2
0.3	-0.1

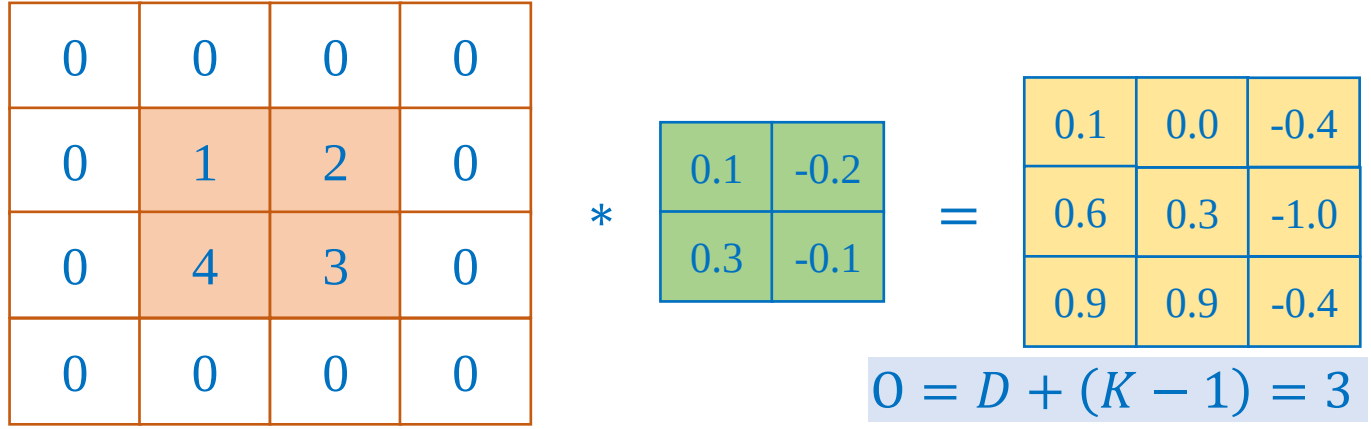
=

Output

0.1	0.0	-0.4
0.6	0.3	-1.0
0.9	0.9	-0.4

Convolution Transpose

❖ Example 1



$$O = D + (K - 1) = 3$$

Numpy

```
data_v1 = np.pad(data, 1)
print(data_v1)
```

```
[[0. 0. 0. 0.]
 [0. 1. 2. 0.]
 [0. 3. 4. 0.]
 [0. 0. 0. 0.]]
```

```
kernel = np.array([[ 0.1, -0.2],
                   [ 0.3, -0.1]])
print(kernel)
```

```
[[ 0.1 -0.2]
 [ 0.3 -0.1]]
```

```
from scipy import signal
signal.convolve2d(data_v1, kernel, mode='valid')
```

```
array([[ 0.1,  0. , -0.4],
       [ 0.6,  0.3, -1. ],
       [ 0.9,  0.9, -0.4]])
```

Pytorch

```
conv_transpose = torch.nn.ConvTranspose2d(in_channels=1, out_channels=1,
                                          kernel_size=2, stride=1, padding=0)
output = conv_transpose(data)
print(output[0,0,:,:])
```

```
tensor([[ 0.1000,  0.0000, -0.4000],
        [ 0.6000,  0.3000, -1.0000],
        [ 0.9000,  0.9000, -0.4000]], grad_fn=<SliceBackward0>)
```

Tensorflow

```
data = np.array([[1.0, 2.0],
                 [3.0, 4.0]])
data = data.reshape(1, 2, 2, 1)

conv_transpose = tf.keras.layers.Conv2DTranspose(filters=1, kernel_size=2,
                                                 strides=1, padding='valid')
```

```
output = conv_transpose(data)
print(output[0,:,:,0])
```

```
tf.Tensor(
[[ 0.1         0.         -0.4       ]
 [ 0.6         0.29999998 -1.        ]
 [ 0.90000004  0.90000004 -0.4       ]], shape=(3, 3), dtype=float32)
```

Convolution Transpose: Example 1 (Different view)

Stride S = 1

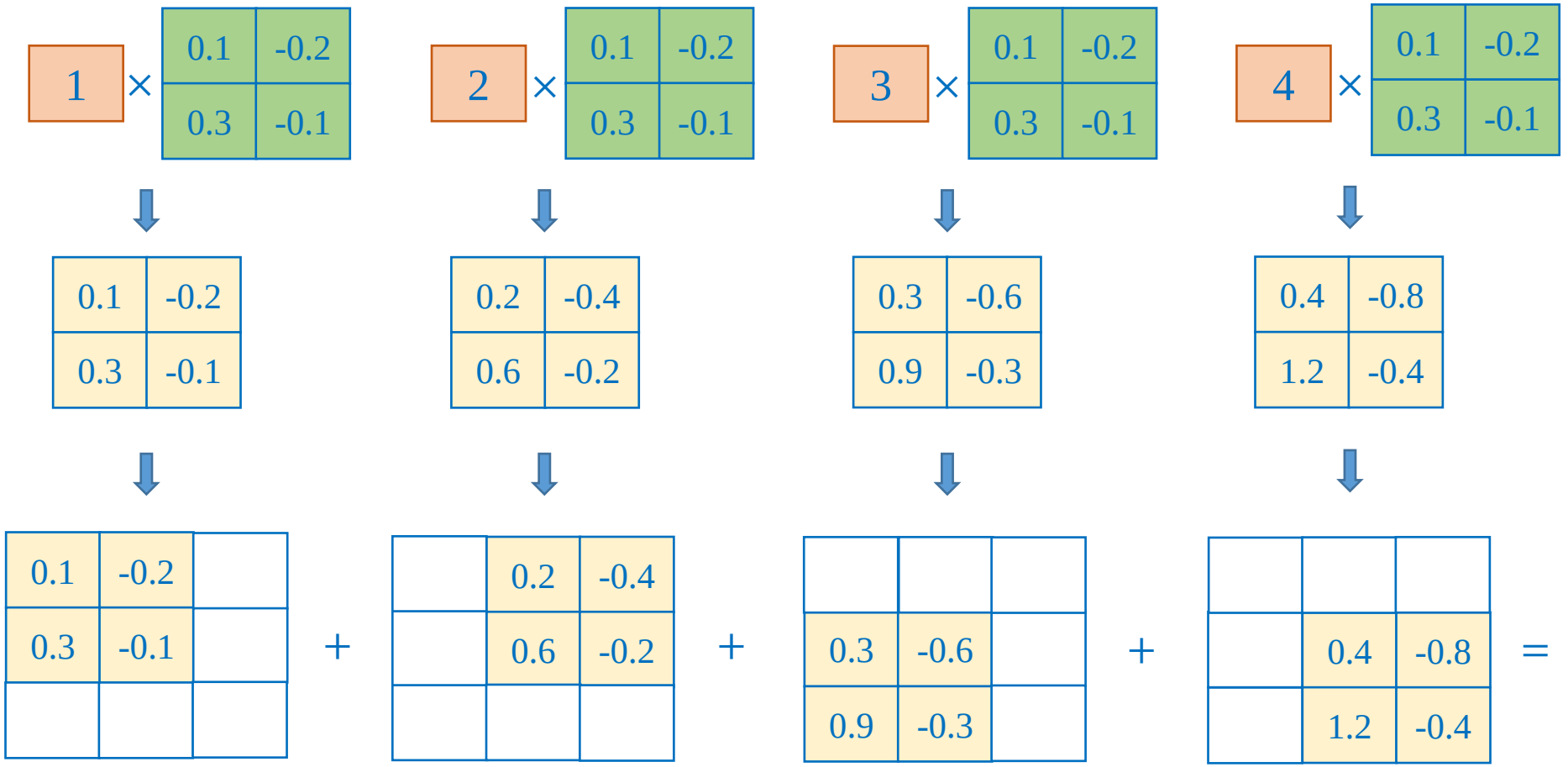
Padding p = 0

data =

1	2
3	4

kernel =

0.1	-0.2
0.3	-0.1



Convolution Transpose

❖ Example 2

Padding $p = 0$
(padding='valid')

$(D - K)\%S == 0$

data =

1	2
3	4

$D = 2$

kernel =

0.1	-0.2
0.3	-0.1

$K = 2$

With stride $S = 2$

1	2
3	4

→

1	0	2
0	0	0
4	0	3

Transpose padding
$p' = K - 1 = 1$

→

0	0	0	0	0
0	1	0	2	0
0	0	0	0	0
0	4	0	3	0
0	0	0	0	0

0.1	-0.2	0.2	-0.4
0.3	-0.1	0.6	-0.2
0.3	-0.6	0.4	-0.8
0.9	-0.3	1.2	-0.4

$=$

$*$

0.1	-0.2
0.3	-0.1

Convolution Transpose

❖ Example 2

```
data_v3 = np.insert(data, obj=[1], values=0, axis=1)
data_v3 = np.insert(data_v3, obj=[1], values=0, axis=0)
print(data_v3)
```

```
[[1. 0. 2.]
 [0. 0. 0.]
 [3. 0. 4.]]
```

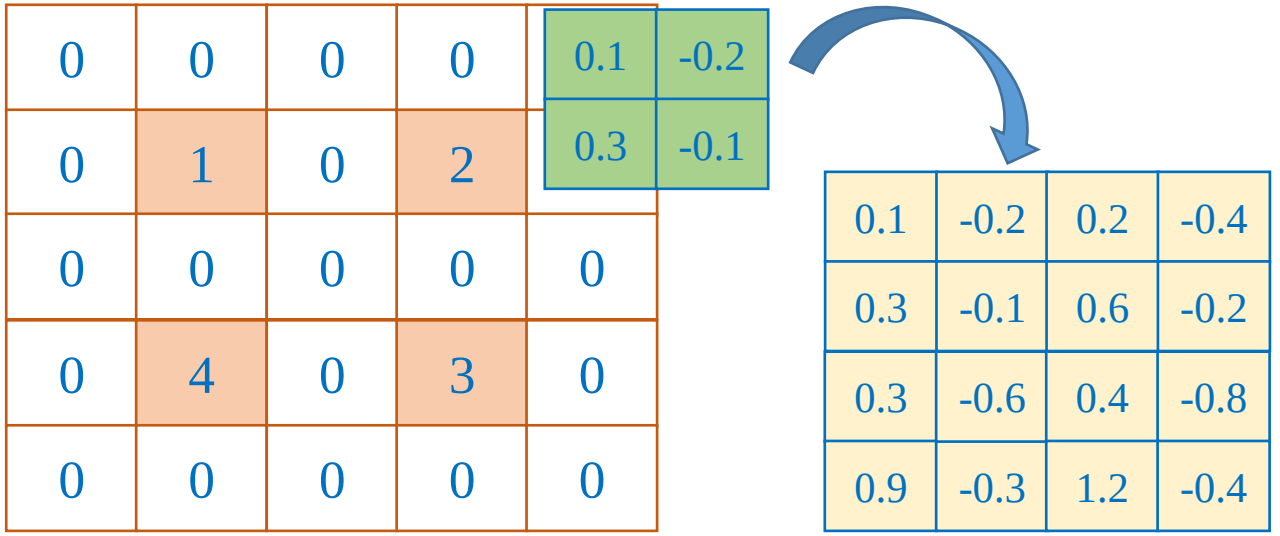
```
data_v3 = np.pad(data_v3, 1)
print(data_v3)
```

```
[[0. 0. 0. 0. 0.]
 [0. 1. 0. 2. 0.]
 [0. 0. 0. 0. 0.]
 [0. 3. 0. 4. 0.]
 [0. 0. 0. 0. 0.]]
```

```
signal.convolve2d(data_v3, kernel, mode='valid')
```

```
array([[ 0.1, -0.2,  0.2, -0.4],
       [ 0.3, -0.1,  0.6, -0.2],
       [ 0.3, -0.6,  0.4, -0.8],
       [ 0.9, -0.3,  1.2, -0.4]])
```

Numpy




```
#torch.nn.ConvTranspose2d(in_channels=C, out_channels=1,
#                          kernel_size=(2,2), stride=2, padding=0)

output = conv_transpose(data)
```

Pytorch

```
tensor([[ 0.1000, -0.2000,  0.2000, -0.4000],
        [ 0.3000, -0.1000,  0.6000, -0.2000],
        [ 0.3000, -0.6000,  0.4000, -0.8000],
        [ 0.9000, -0.3000,  1.2000, -0.4000]], grad_fn=<SliceBackwa
```

```
conv_transpose = Conv2DTranspose(filters=1, kernel_size=2,
                                 strides=2, padding='valid')
output = conv_transpose(data)
```

Tensorflow

```
tf.Tensor(
[[ 0.1        -0.2         0.2        -0.4       ]
 [ 0.3        -0.1         0.6        -0.2       ]
 [ 0.3        -0.6         0.4        -0.8       ]
 [ 0.90000004 -0.3         1.2        -0.4       ]], shape=(4, 4),
```

Convolution Transpose

Example 2 (Different view)

data =

1	2
3	4

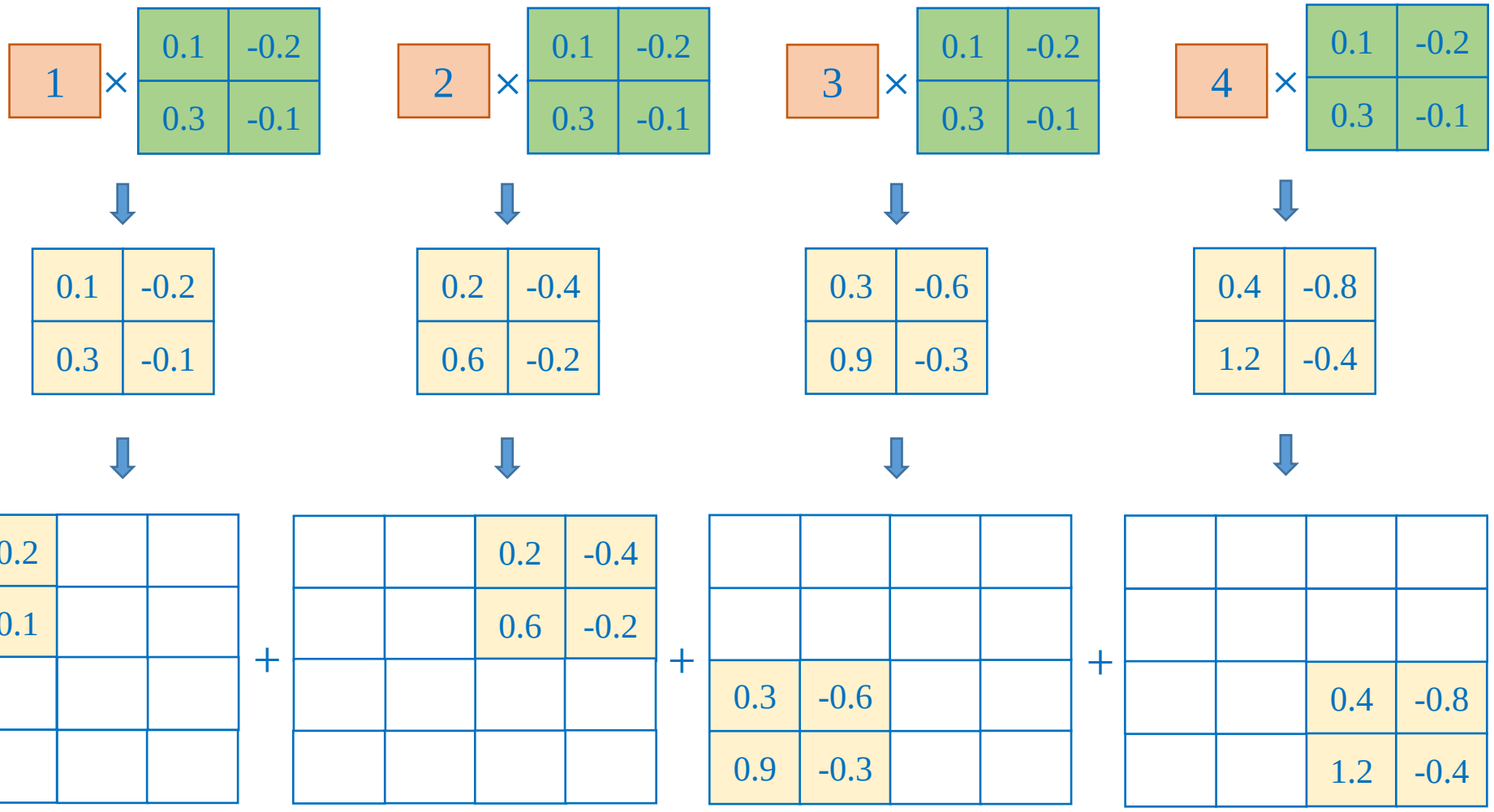
kernel =

0.1	-0.2
0.3	-0.1

Padding p = 0

(D − K)%S == 0

With stride S = 2



ConvTranspose

Input

1.	**2.**
3.	**4.**

Shape=(2,2)

Kernel

3.	0.	-2.
1.	1.	1.
-3.	0.	2.

Shape=(3,3)

1. *

3.	0.	-2.
1.	1.	1.
-3.	0.	2.

=

3.	0.	-2.
1.	1.	1.
-3.	0.	2.

2. *

3.	0.	-2.
1.	1.	1.
-3.	0.	2.

=

6.	0.	-4.
2.	2.	2.
-6.	0.	4.

3. *

3.	0.	-2.
1.	1.	1.
-3.	0.	2.

=

9.	0.	-6.
3.	3.	3.
-9.	0.	6.

4. *

3.	0.	-2.
1.	1.	1.
-3.	0.	2.

=

12.	0.	-8.
4.	4.	4.
-12.	0.	8.

Stride = 2

3.	0.	-2.	6.	0.	-4.
1.	1.	1.	2.	2.	2.
-3.	0.	2.	-6.	0.	4.
9.	0.	-6.	12.	0.	-8.
3.	3.	3.	4.	4.	4.
-9.	0.	6.	-12.	0.	8.

Sum

3.	0.	4.	0.	-4.
1.	1.	3.	2.	2.
6.	0.	2.	0.	-4.
3.	3.	7.	4.	4.
-9.	0.	-6.	0.	8.

Shape=(5,5)

ConvTranspose

$$O = S(D-1) + (K-1) - 2 \times P + P_o + 1$$

padding = 0

output_padding = 0

3.	0.	4.	0.	-4.
1.	1.	3.	2.	2.
6.	0.	2.	0.	-4.
3.	3.	7.	4.	4.
-9.	0.	-6.	0.	8.

Shape=(5,5)

padding = 1

output_padding = 0

1.	3.	2.
0.	2.	0.
3.	7.	4.

Shape=(3,3)

Bias

+0.1

Output

1.1	3.1	2.1
0.1	2.1	0.1
3.1	7.1	4.1

Shape=(3,3)

padding = 1

output_padding = 1

1.	3.	2.	2.
0.	2.	0.	-4.
3.	7.	4.	4.
0.	-6.	0.	8.

Shape=(4,4)

Bias

+0.1

Output

1.1	3.1	2.1	2.1
0.1	2.1	0.1	-3.9
3.1	7.1	4.1	4.1
0.1	-5.9	0.1	8.1

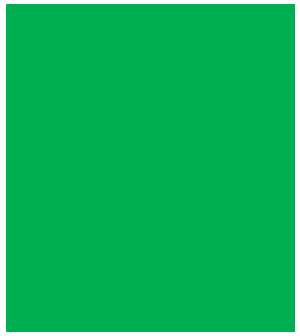
Shape=(4,4)

Convolution Transpose

```
nn.ConvTranspose2d(in_channels, out_channels,
                   kernel_size=4, stride=2,
                   padding=1, output_padding=0)
```

❖ **To upsample feature maps 2x**

Transpose padding: $P_o = 0$



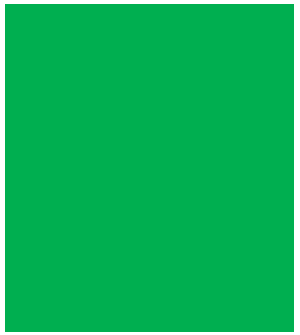
64x64x1

D = 64 K = 4

S = 2 P = 1

Output size

$$O = S(D-1) + (K-1) - 2 \times P + P_o + 1$$
$$= 2 * 63 + 3 - 2 + 1 + 1 = 128$$



64x64x1

D = 64 K = 3

S = 2 P = 1

```
nn.ConvTranspose2d(in_channels, out_channels,
                   kernel_size=3, stride=2,
                   padding=1, output_padding=0)
```

Output size

$$O = S(D-1) + (K-1) - 2 \times P + P_o + 1$$
$$= 2 * 63 + 2 - 2 + 1 + 1 = 127$$

Further reading:
https://arxiv.org/pdf/1603.07285v1.pdf

Convolution Transpose

$D = 64$ $K = 3$

$S = 2$ $P = 1$

```python
import torch
import numpy as np

N, H, W, C = 1, 64, 64, 1
data = np.random.randn(N, C, H, W)
data = torch.tensor(data, dtype=torch.float)

conv_transpose = torch.nn.ConvTranspose2d(in_channels=1, out_channels=1,
                                          kernel_size=(3,3), stride=2, padding=1)

output = conv_transpose(data)
print(output.shape)
```

```
torch.Size([1, 1, 127, 127])
```

$D = 64$ $K = 4$

$S = 2$ $P = 1$

```python
import torch
import numpy as np

N, H, W, C = 1, 64, 64, 1
data = np.random.randn(N, C, H, W)
data = torch.tensor(data, dtype=torch.float)

conv_transpose = torch.nn.ConvTranspose2d(in_channels=1, out_channels=1,
                                          kernel_size=(4,4), stride=2, padding=1)

output = conv_transpose(data)
print(output.shape)
```

```
torch.Size([1, 1, 128, 128])
```

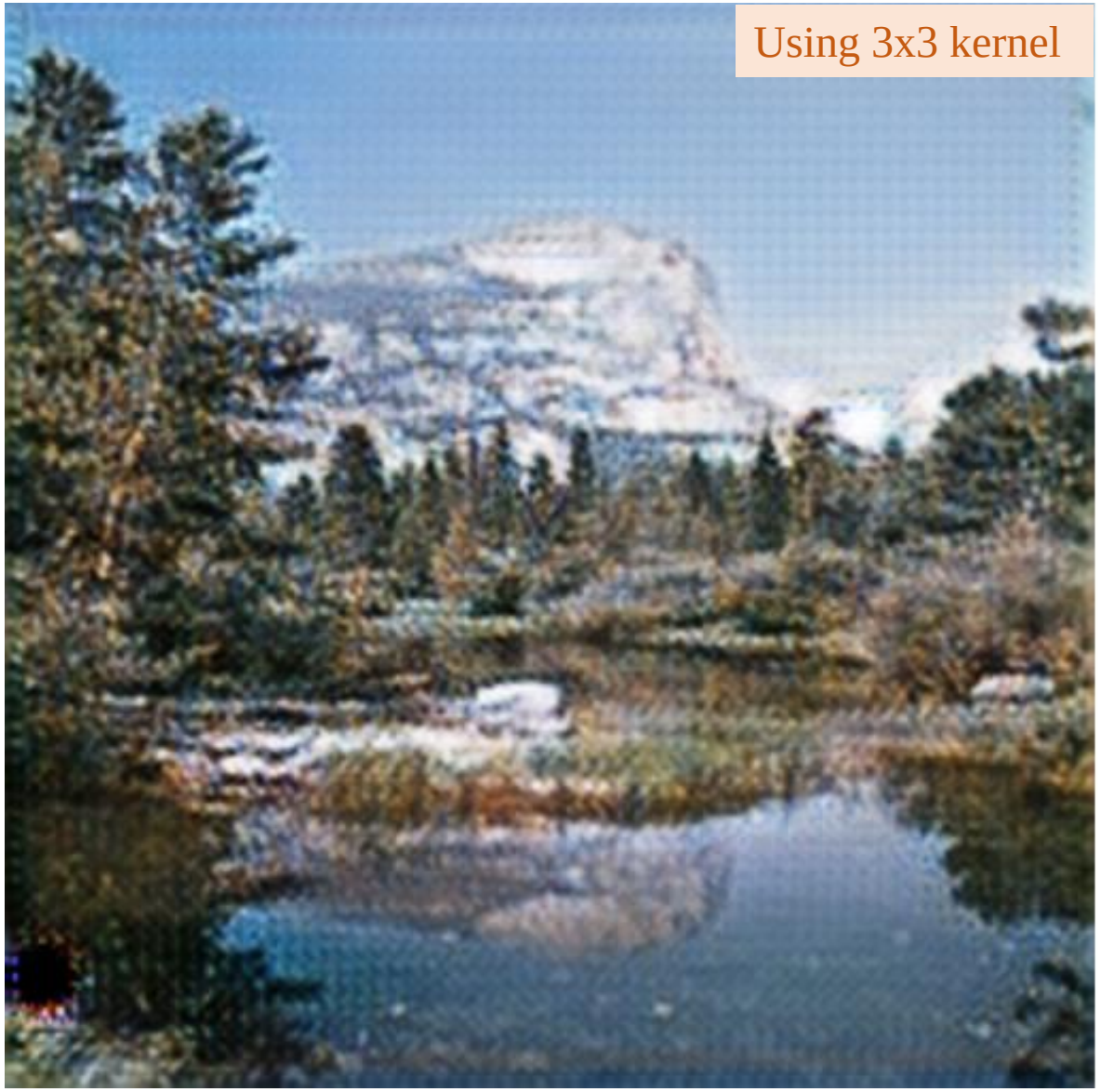
Further reading:
https://arxiv.org/pdf/1603.07285v1.pdf

Using 3x3 or 4x4

❖ Convolution Transpose

```
nn.ConvTranspose2d(in_channels, out_channels,
                   kernel_size=4, stride=2,
                   padding=1, output_padding=0)
```

https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix/issues/78



Outline

- Segmentation (Unet Using Interpolation)
- Alternatives to Increate Feature Resolution
- Colorization
- Super-resolution
- Denoising Survey
- Super-resolution Survey

Image Denoising and Colorization

Noisy images

Model

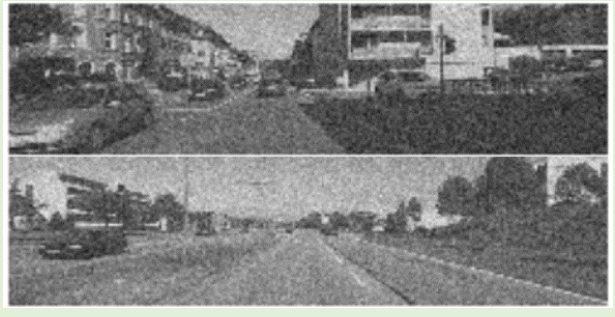
Clean images

Grayscale images

Model

Color images

Noisy and grayscale images



Model

Clean and color images

Image Translation

❖ Super-resolution

Low Resolution

Predicted High Resolution

Low Resolution

Predicted High Resolution

Low Resolution

Predicted High Resolution

Image Translation

❖ Edge2Scene

Edge

Scene

Edge

Scene

Edge

Scene

Image Translation

Deblur

1	Data preparation	3	Loss and optimizer
2	Network construction	4	Training

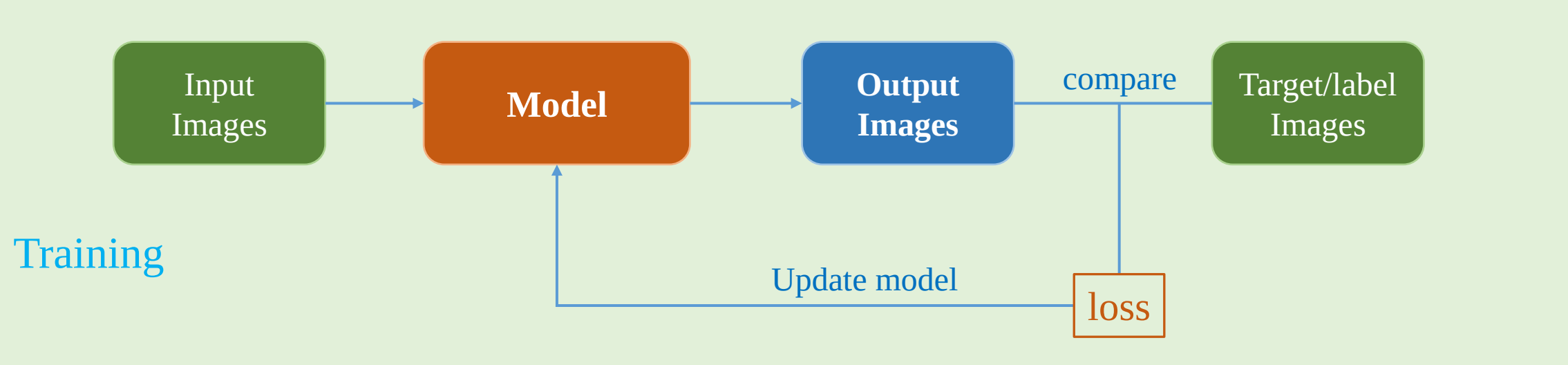

Input Images

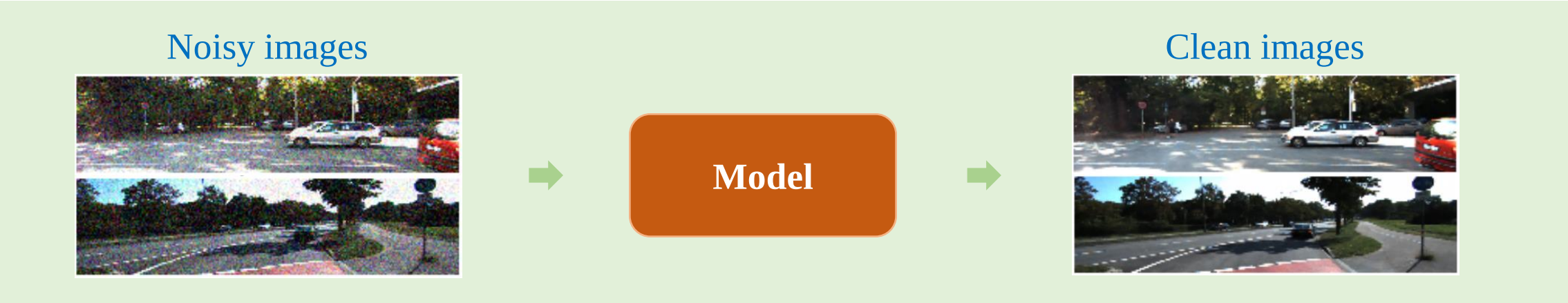

Target Images


Predicted Images

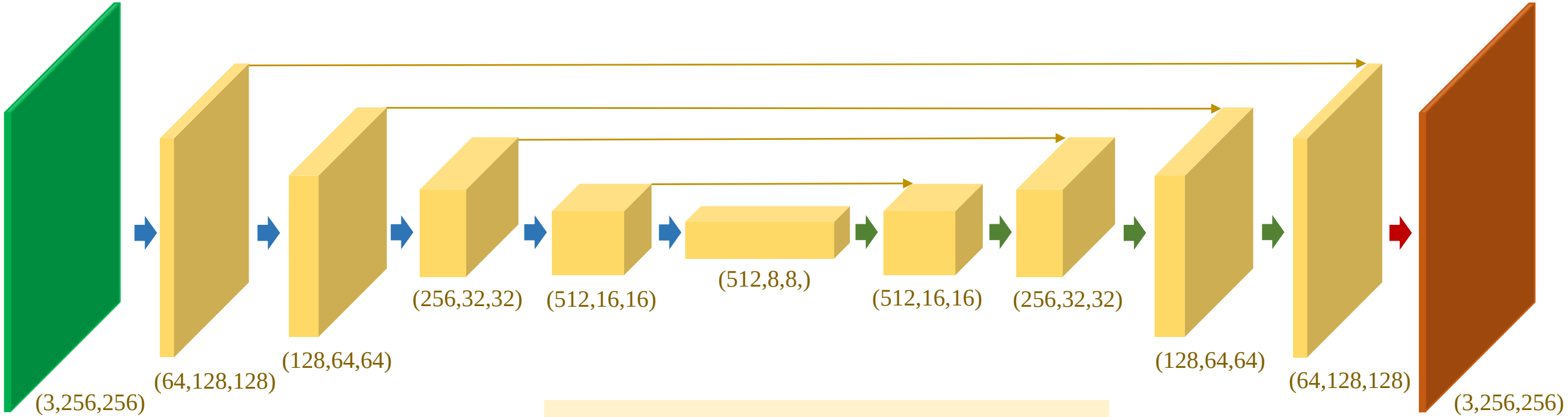
Image Denoising and Colorization

❖ Flowchart





Denoising/Colorization/Deblur/…



Input Image
[-1,1]

Skip connection (UNet)

Output Image
[-1,1]

(3x3) Convolution padding = 'same' stride = 1 + ReLU + Batch Norm + (2x2) max pooling

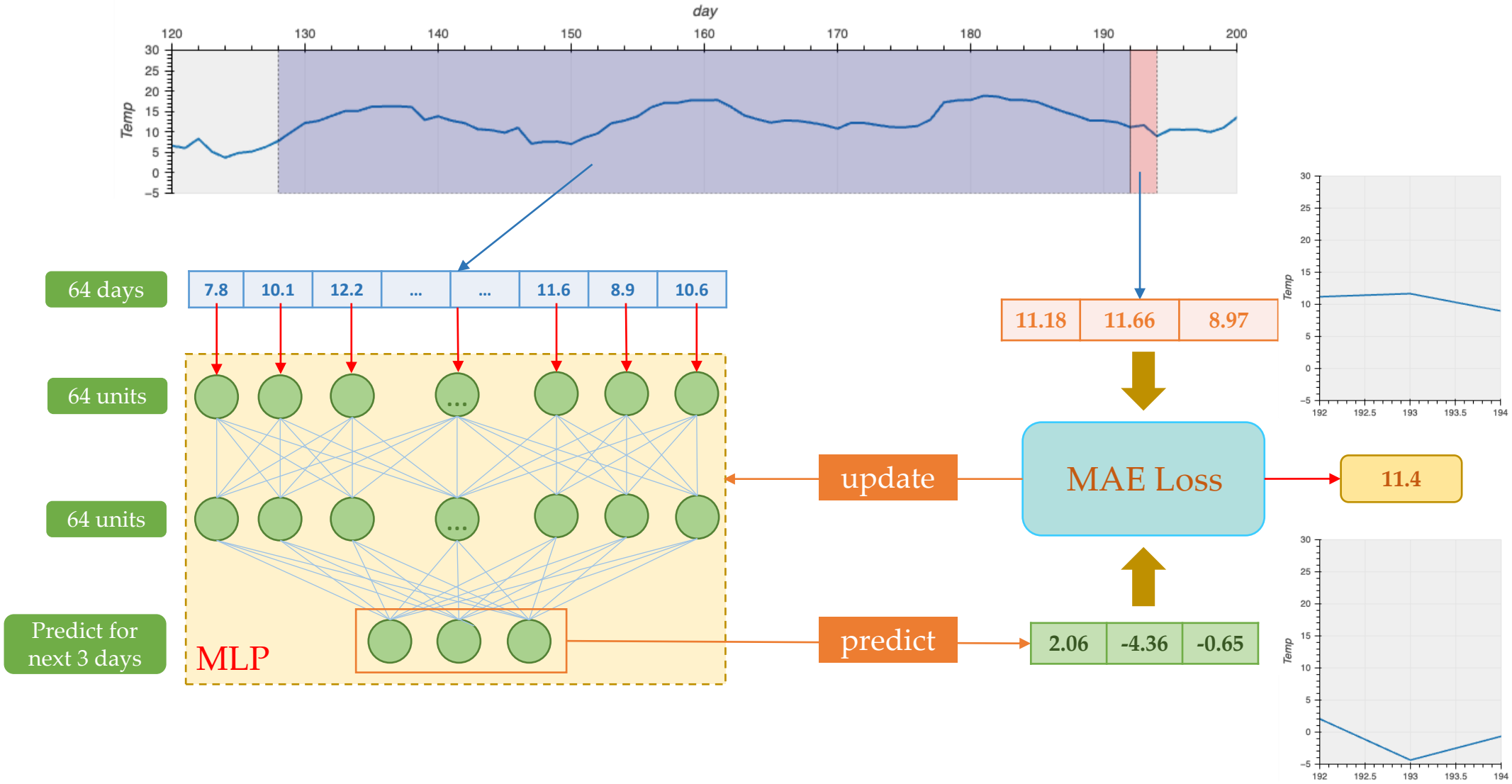
4x4 ConvTranspose padding = 'same' stride = 2 + ReLU + Batch Norm

4x4 ConvTranspose padding = 'same' stride = 2 + Tanh

Regression Problem

$$L1(I1, I2) = \frac{1}{H * W * C} \sum_{h=1}^{H} \sum_{w=1}^{W} \sum_{c=1}^{C} |I1_{hwc} - I2_{hwc}|$$

Input/Output + Loss Function



Further Reading

Unet-related Papers

U-Net: Convolutional Networks for Biomedical Image Segmentation

Olaf Ronneberger, Philipp Fischer, and Thomas Brox

Computer Science Department and BIOSS Centre for Biological Signalling Studies,
University of Freiburg, Germany
ronneber@informatik.uni-freiburg.de,
WWW home page: http://lmb.informatik.uni-freiburg.de/

TransUNet: Transformers Make Strong Encoders for Medical Image Segmentation

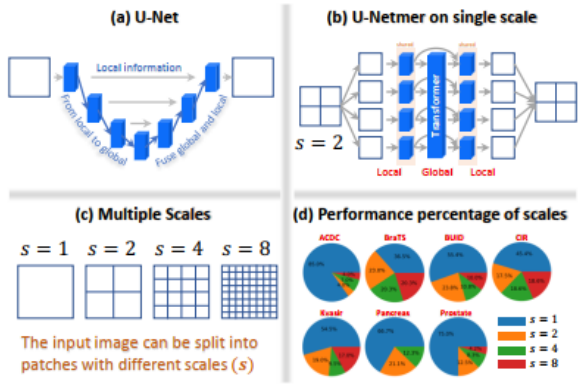
Jieneng Chen[1], Yongyi Lu[1], Qihang Yu[1], Xiangde Luo[2],
Ehsan Adeli[3], Yan Wang[4], Le Lu[5], Alan L. Yuille[1], and Yuyin Zhou[3]

[1]Johns Hopkins University
[2]University of Electronic Science and Technology of China
[3]Stanford University
[4] East China Normal University
[5]PAII Inc.

U-Netmer: U-Net meets Transformer for medical image segmentation

Sheng He, Rina Bao, P. Ellen Grant, Yangming Ou

***Abstract*—The combination of the U-Net based deep learning models and Transformer is a new trend for medical image segmentation. U-Net can extract the detailed local semantic and texture information and Transformer can learn the long-rang dependencies among pixels in the input image. However, directly adapting the Transformer for segmentation has "token-flatten" problem (flattens the local patches into 1D tokens which losses the interaction among pixels within local patches) and "scale-sensitivity" problem (uses a fixed scale to split the input image into local patches). Compared to directly combining U-Net and Transformer, we propose a new global-local fashion combination of U-Net and Transformer, named U-Netmer, to solve the two problems. The proposed U-Netmer splits an input image into local patches. The global-context information among local patches is learnt by the self-attention mechanism in Transformer and U-Net segments each local patch instead of flattening into tokens to solve the 'token-flatten" problem. The U-Netmer can segment the input image with different patch sizes with the identical**



3D TransUNet: Advancing Medical Image Segmentation through Vision Transformers

Jieneng Chen, Jieru Mei, Xianhang Li, Yongyi Lu, Qihang Yu, Qingyue Wei, Xiangde Luo, Yutong Xie,
Ehsan Adeli, Yan Wang, Matthew Lungren, Lei Xing, Le Lu, Alan Yuille, Yuyin Zhou

***Abstract*—Medical image segmentation plays a crucial role in advancing healthcare systems for disease diagnosis and treatment planning. The u-shaped architecture, popularly known as U-Net, has proven highly successful for various medical image segmentation tasks. However, U-Net's convolution-based operations inherently limit its ability to model long-range dependencies effectively.**

To address these limitations, researchers have turned to Transformers, renowned for their global self-attention mechanisms, as alternative architectures. One popular network is our previous TransUNet, which leverages Transformers' self-attention to complement U-Net's localized information with the global context. In this paper, we extend encoder and decoder into the u-shaped medical image segmentation architecture. TransUNet outperforms competitors in various medical applications, including multi-organ segmentation, pancreatic tumor segmentation, and hepatic vessel segmentation. It notably surpasses the top solution in the BrasTS2021 challenge. Code and models are available at https://github.com/Beckschen/3D-TransUNet.

I. Introduction

Outline

- Segmentation (Unet Using Interpolation)
- Alternatives to Increate Feature Resolution
- Colorization
- Super-resolution
- Denoising Survey
- Super-resolution Survey

Image Denoising

Denoising

Noisy Images

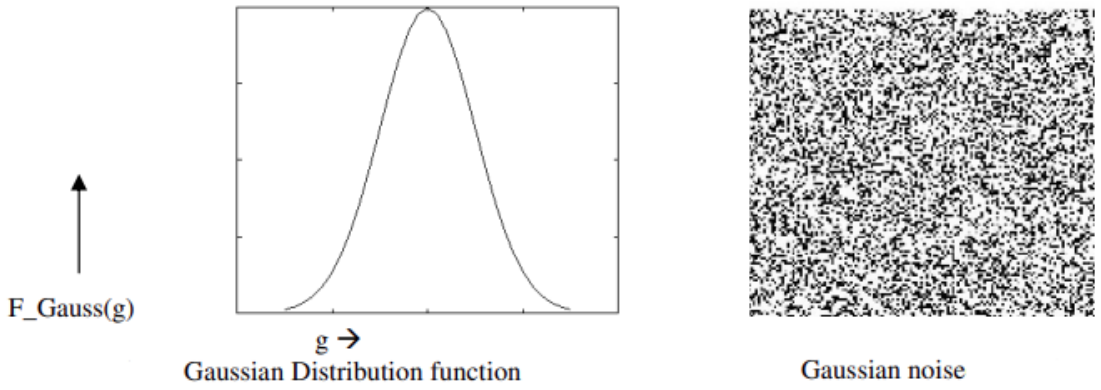
Denoised Images

Wetzler, Aaron & Kimmel, Ron. (2011). Efficient Beltrami Flow in Patch-Space. 6667. 134-143. 10.1007/978-3-642-24785-9_12.

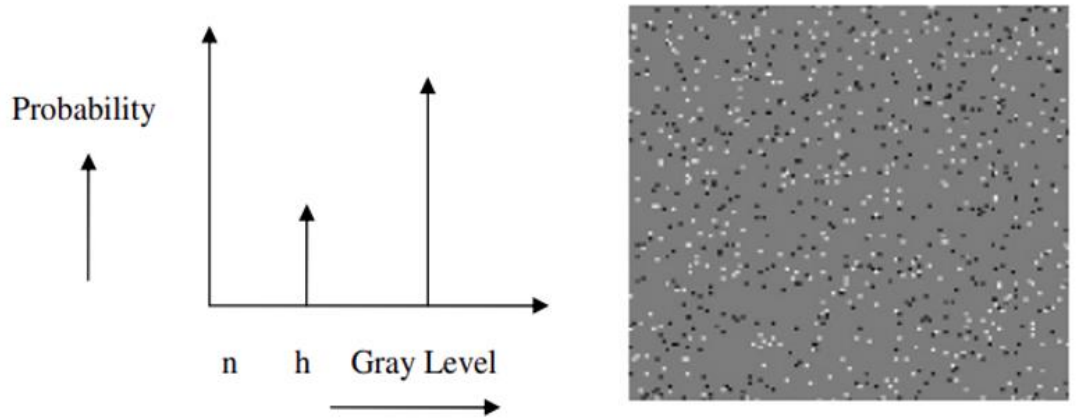
Image Denoising

Denoising

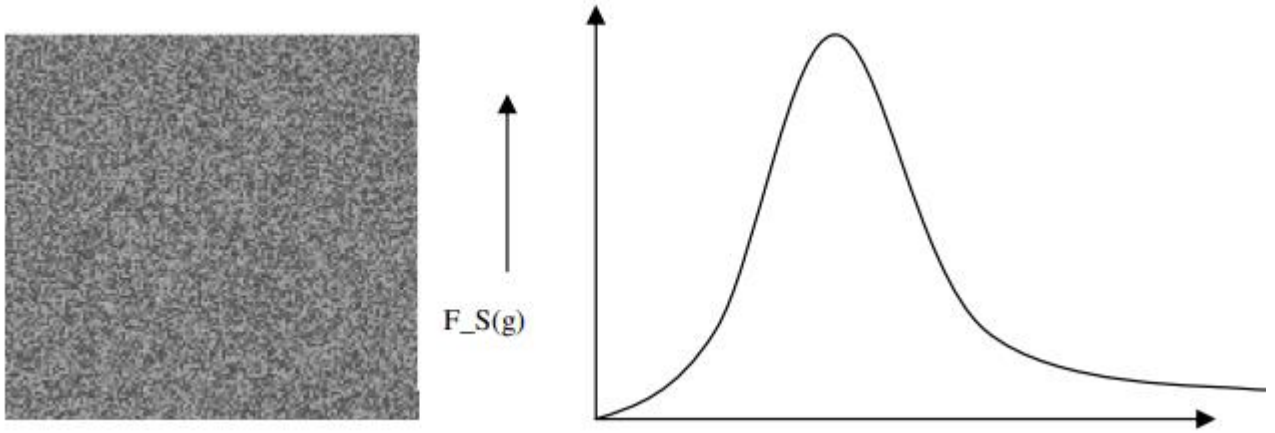
Gaussian Noise



Salt and Pepper Noise



Speckle Noise



https://analyticsindiamag.com/a-guide-to-different-types-of-noises-and-image-denoising-methods/

Image Denoising

Traditional Image Denoising Technique

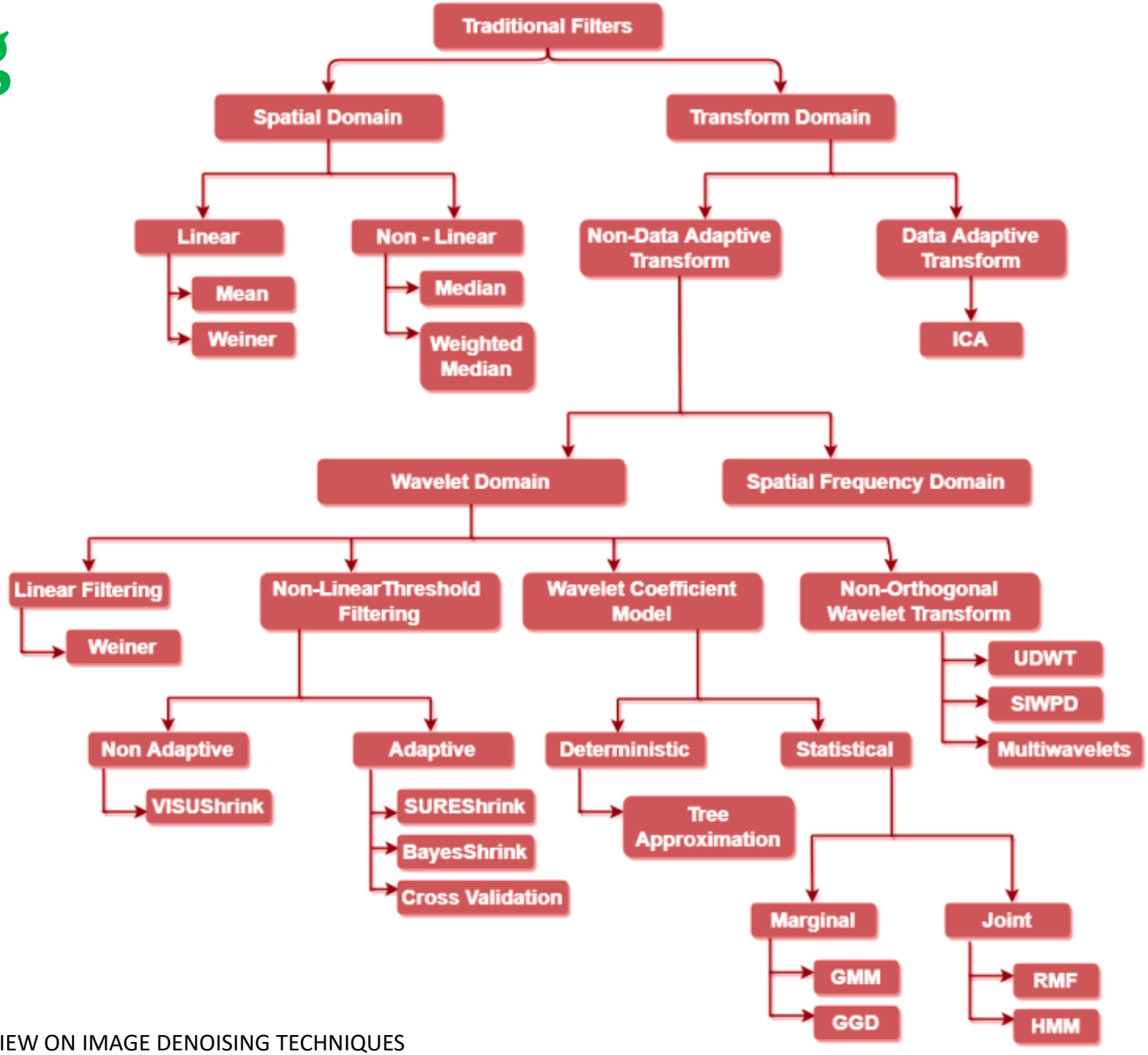
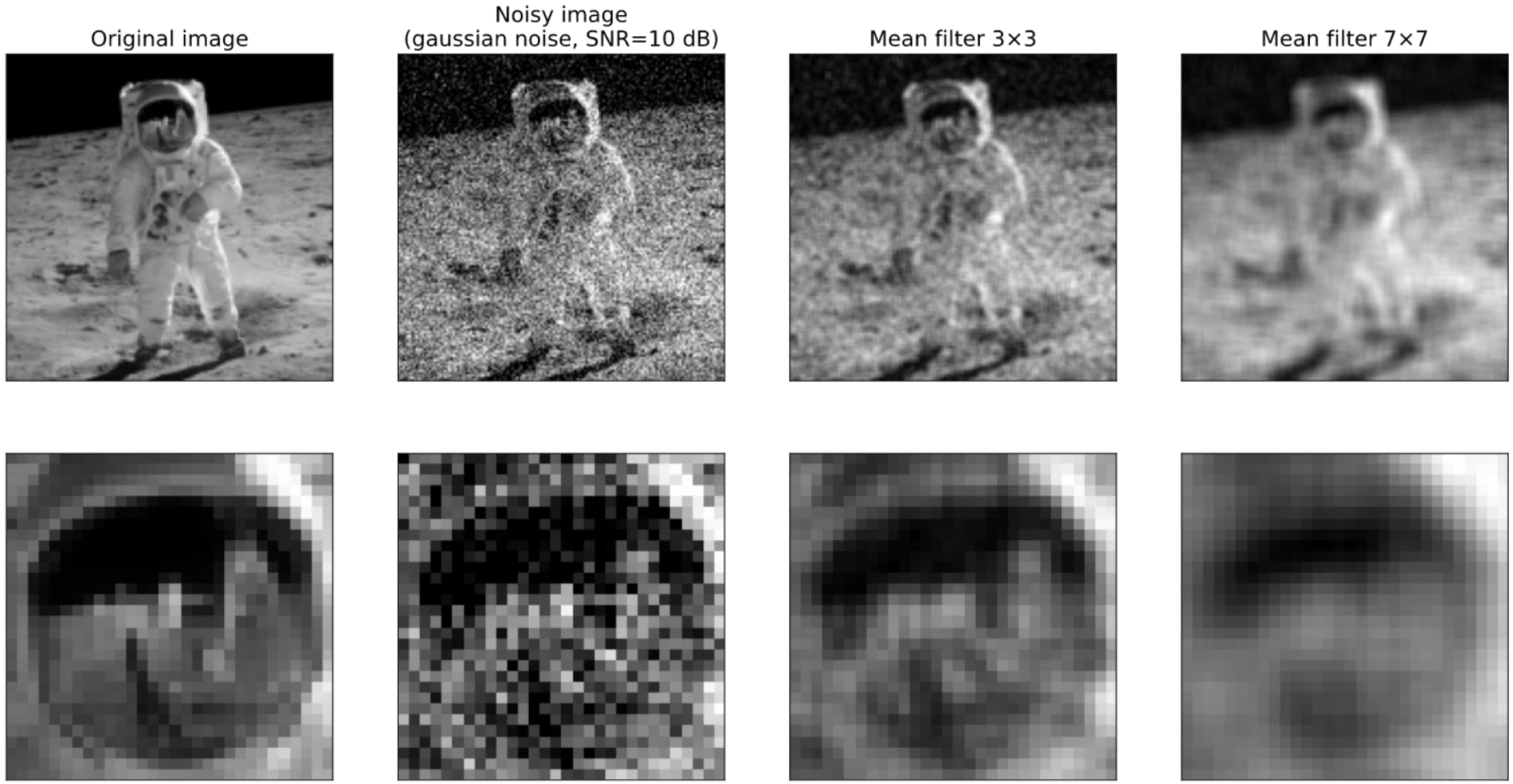


Image Denoising

❖ Traditional Image Denoising Technique

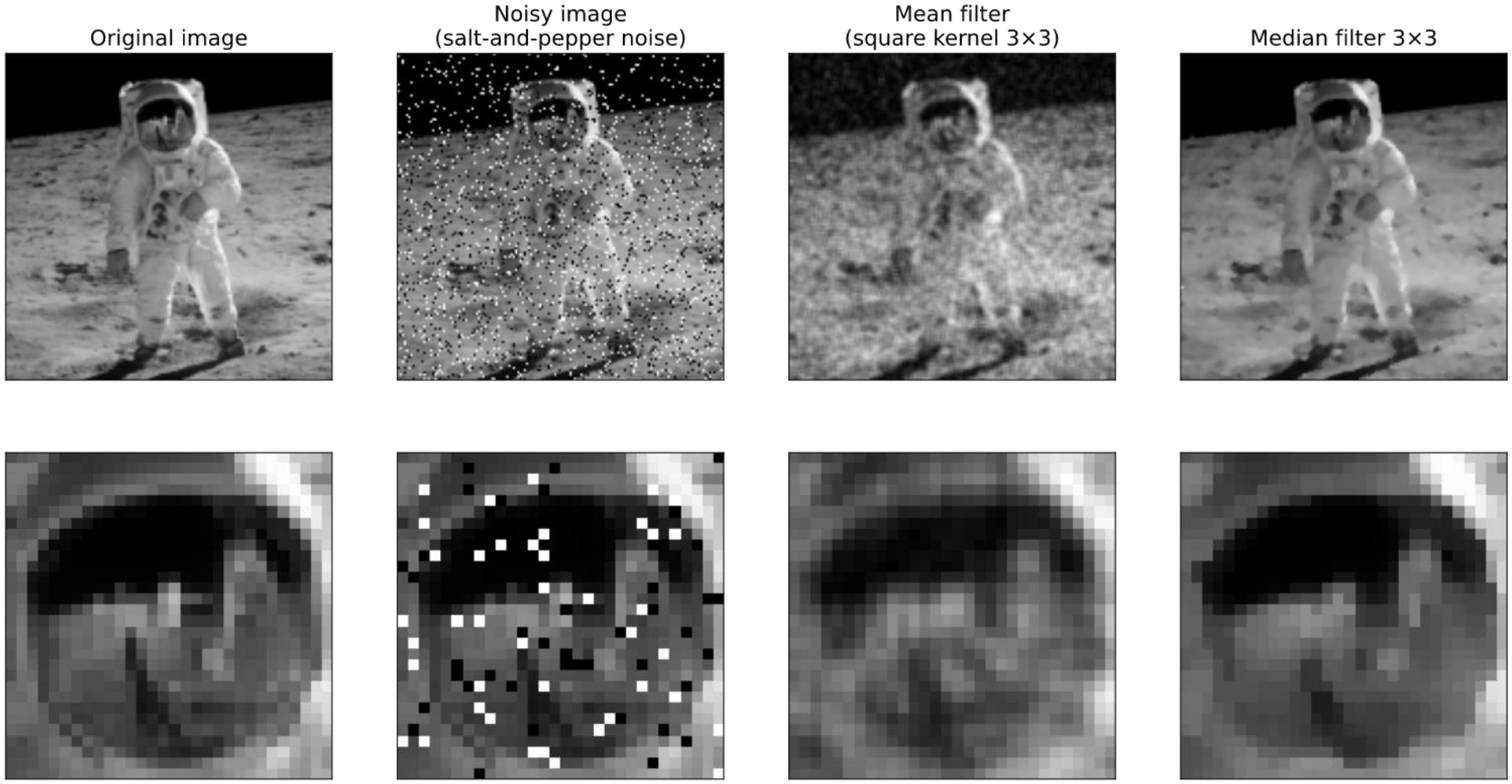


Mean filter

https://vincmazet.github.io/bip/restoration/denoising.html

Image Denoising

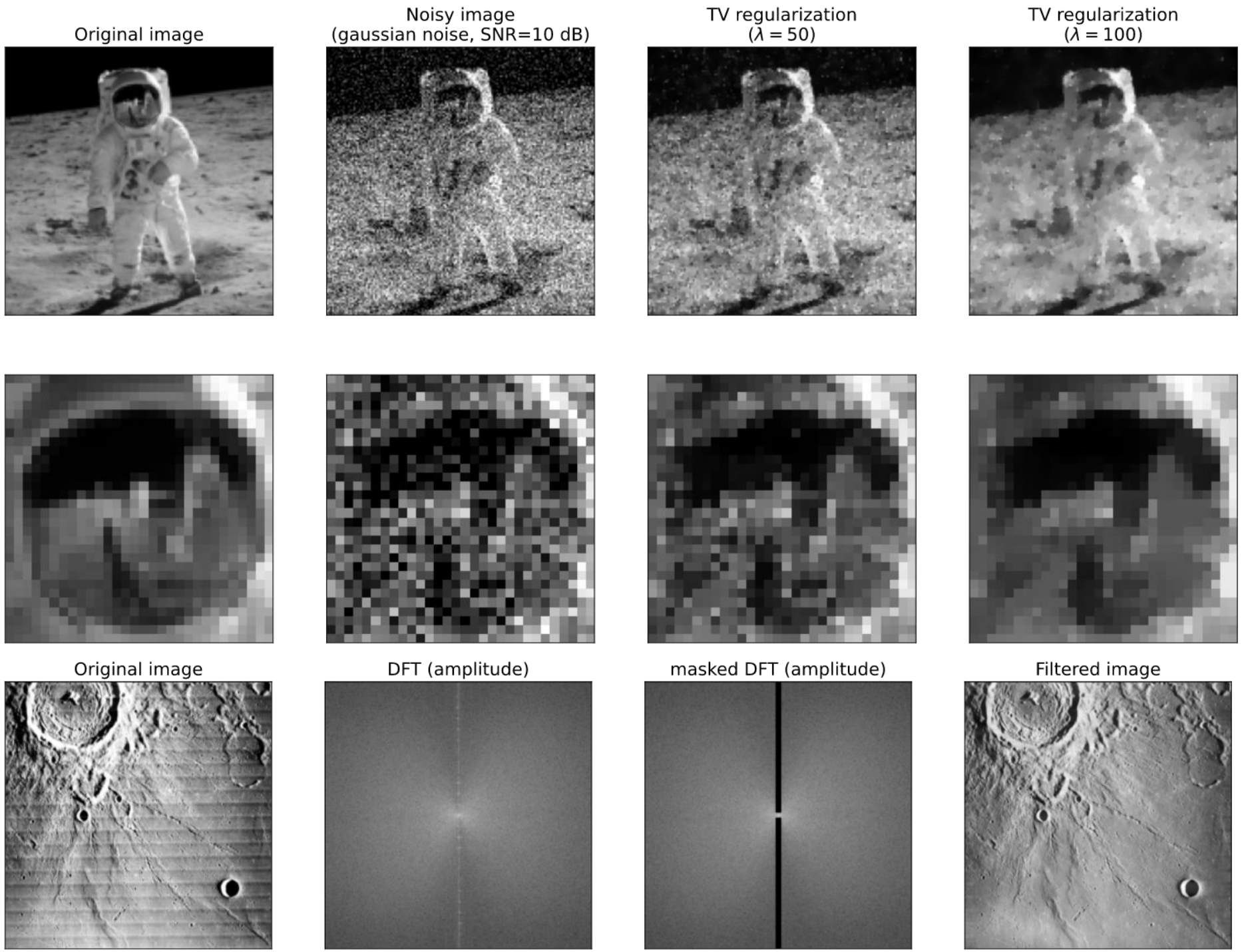
❖ Traditional Image Denoising Technique



Median filter

https://vincmazet.github.io/bip/restoration/denoising.html

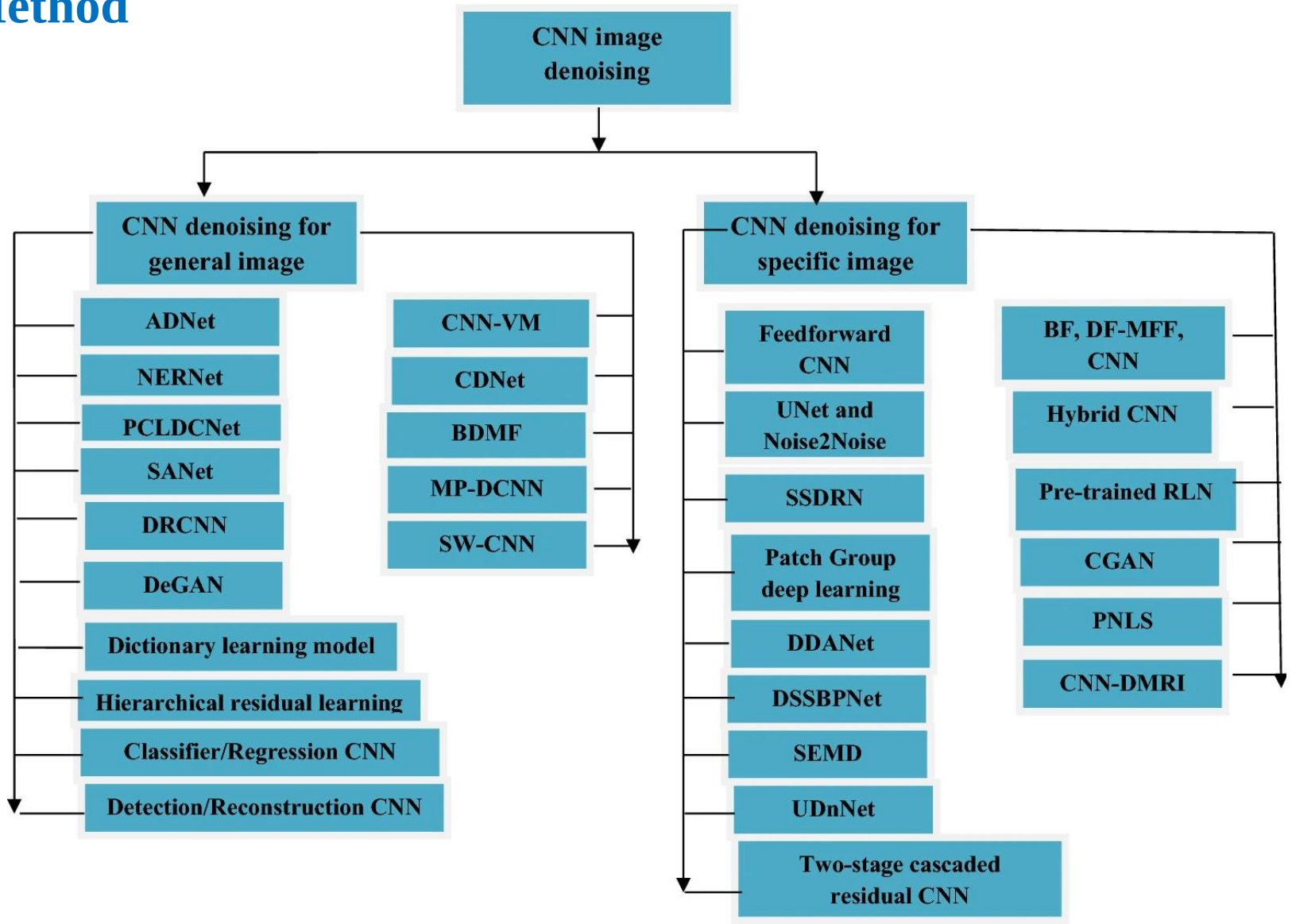
Image Denoising



https://vincmazet.github.io/bip/restoration/denoising.html

Image Denoising

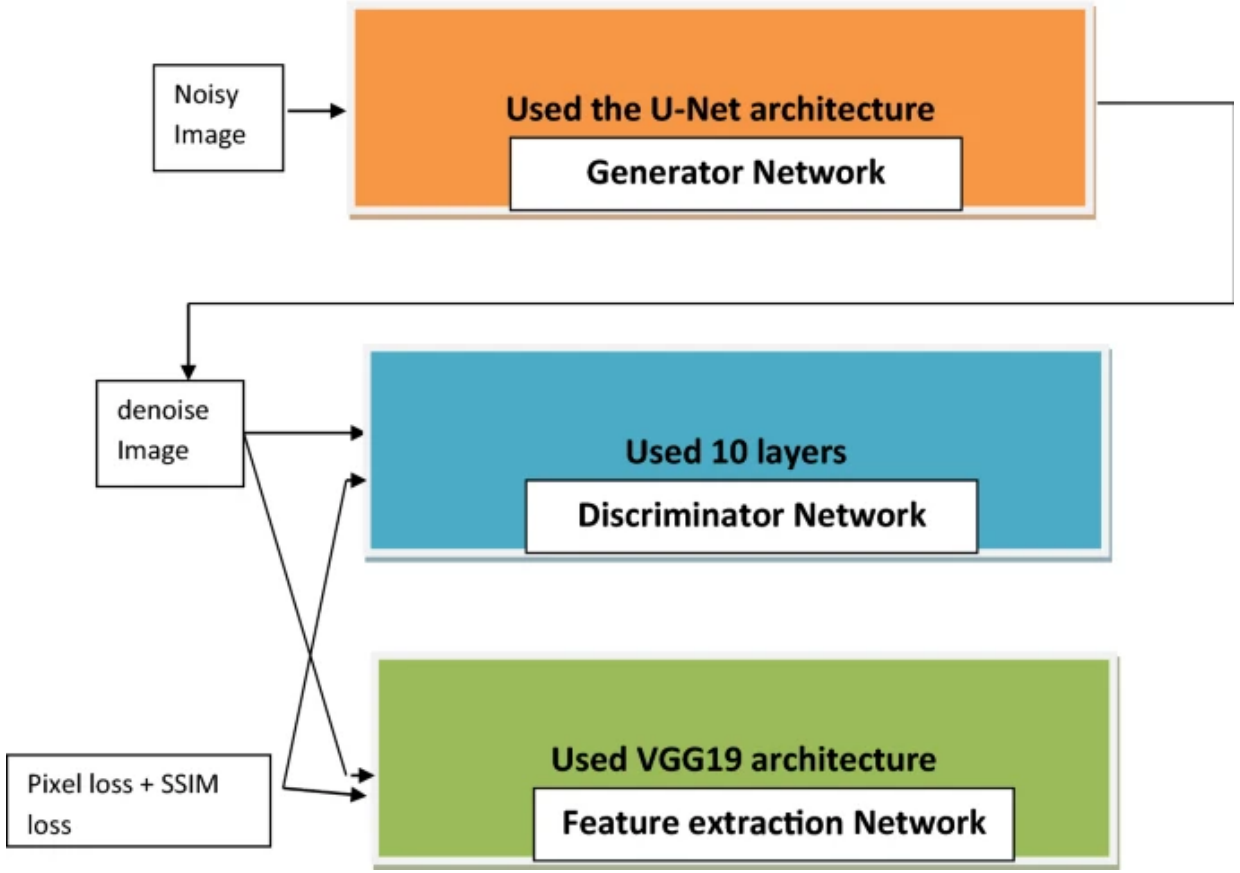
❖ Based Deep-Learning Method



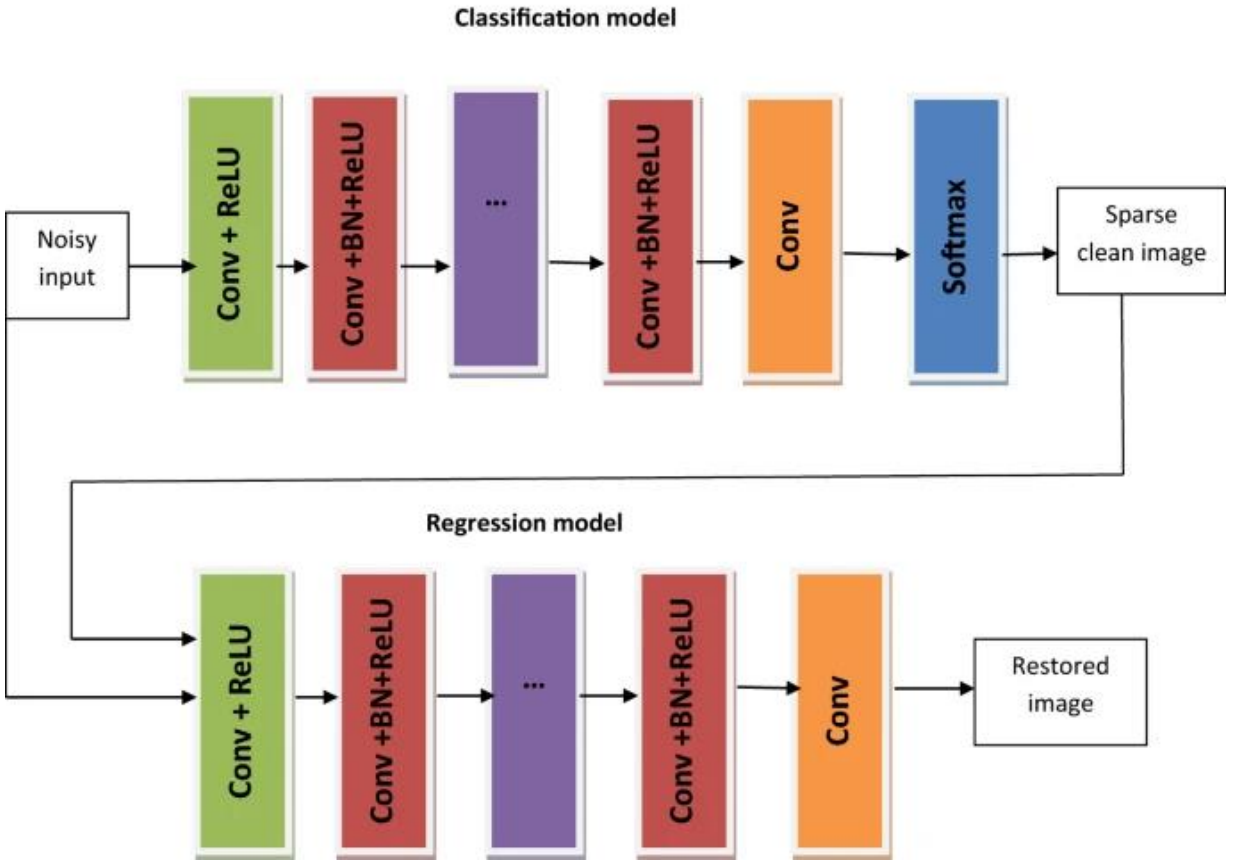
Ilesanmi, A.E., Ilesanmi, T.O. Methods for image denoising using convolutional neural network: a review. Complex Intell. Syst. 7, 2179–2198 (2021)

Image Denoising

❖ Based Deep-Learning Method



DeGAN



Classifier/regression CNN

Ilesanmi, A.E., Ilesanmi, T.O. Methods for image denoising using convolutional neural network: a review. Complex Intell. Syst. 7, 2179–2198 (2021)

Outline

- Segmentation (Unet Using Interpolation)
- Alternatives to Increate Feature Resolution
- Colorization
- Super-resolution
- Denoising Survey
- Super-resolution Survey

Image Super-Resolution

https://towardsdatascience.com/deep-learning-based-super-resolution-without-using-a-gan-11c9bb5b6cd5

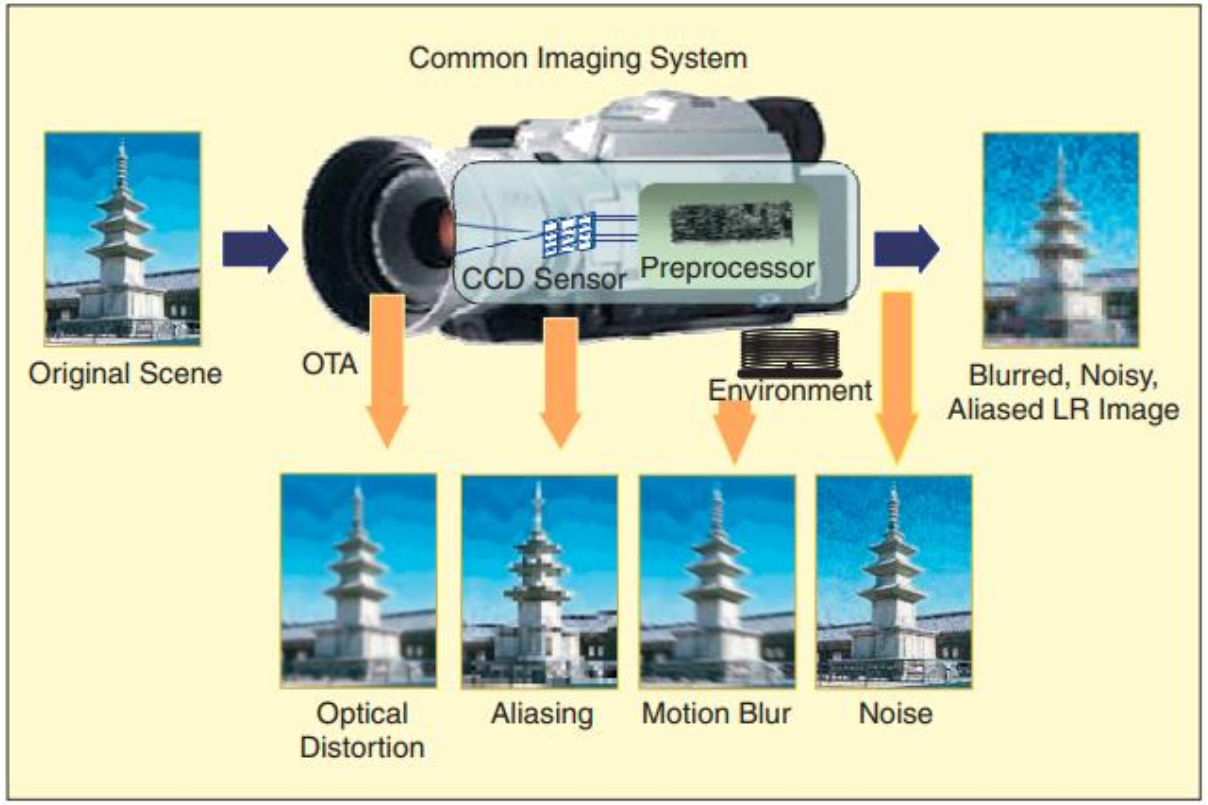
❖ Image Super-Resolution

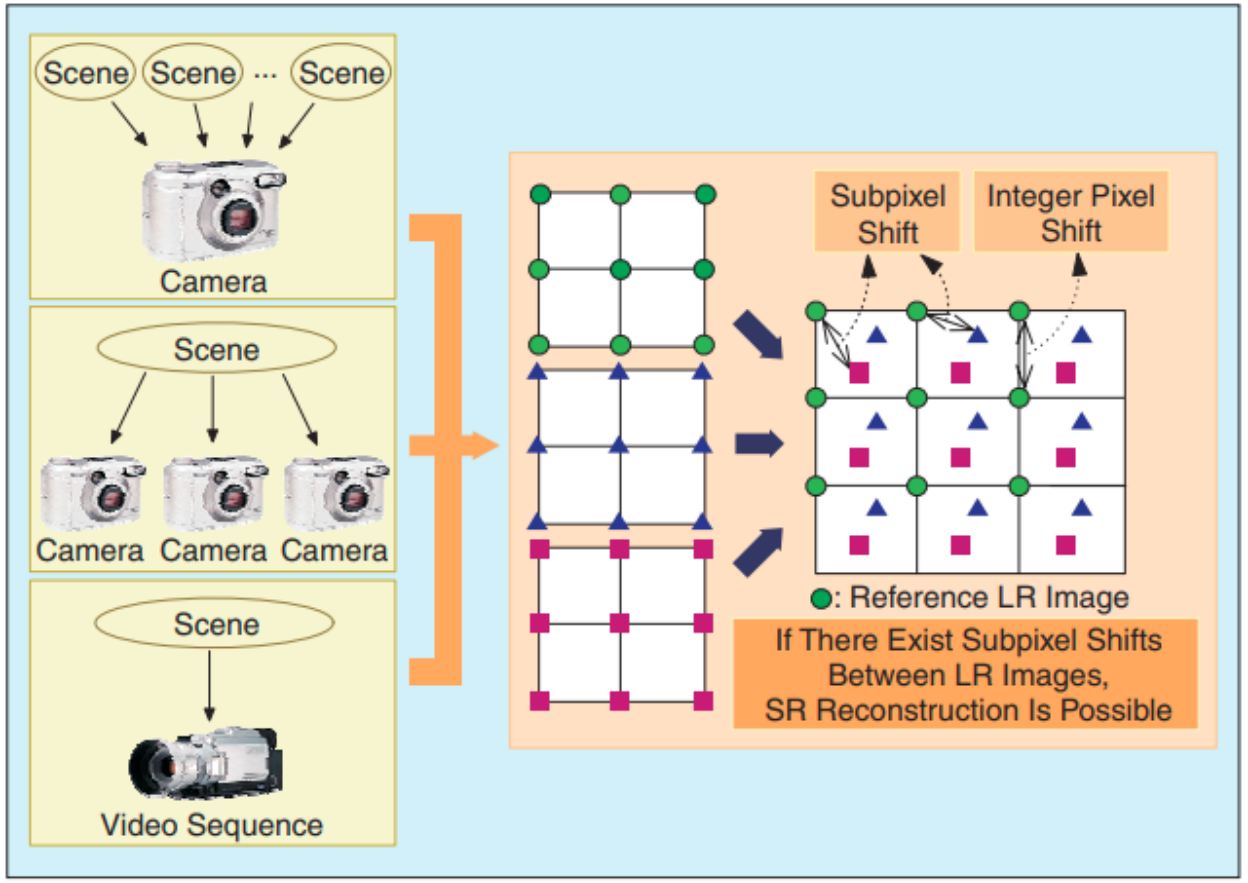
Low-resolution High-resolution

Low-resolution High-resolution

Image Super-Resolution

❖ Image Super-Resolution

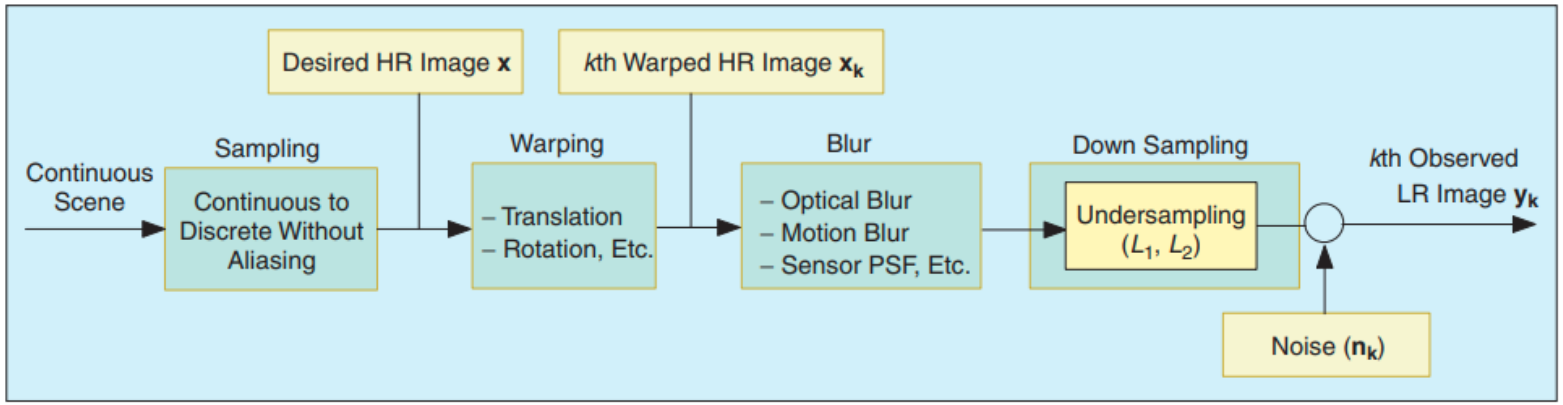


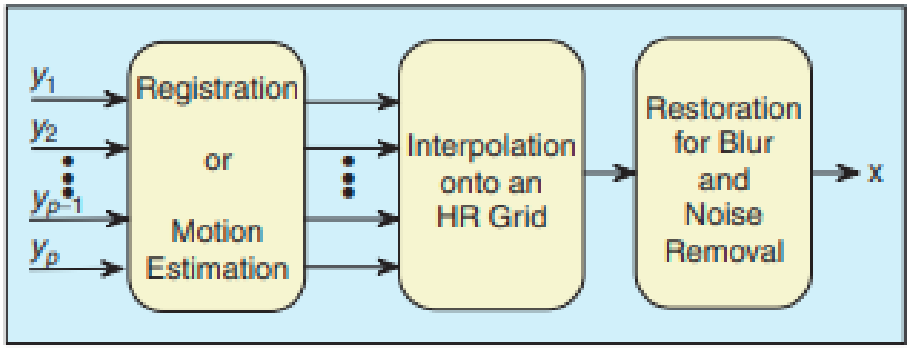


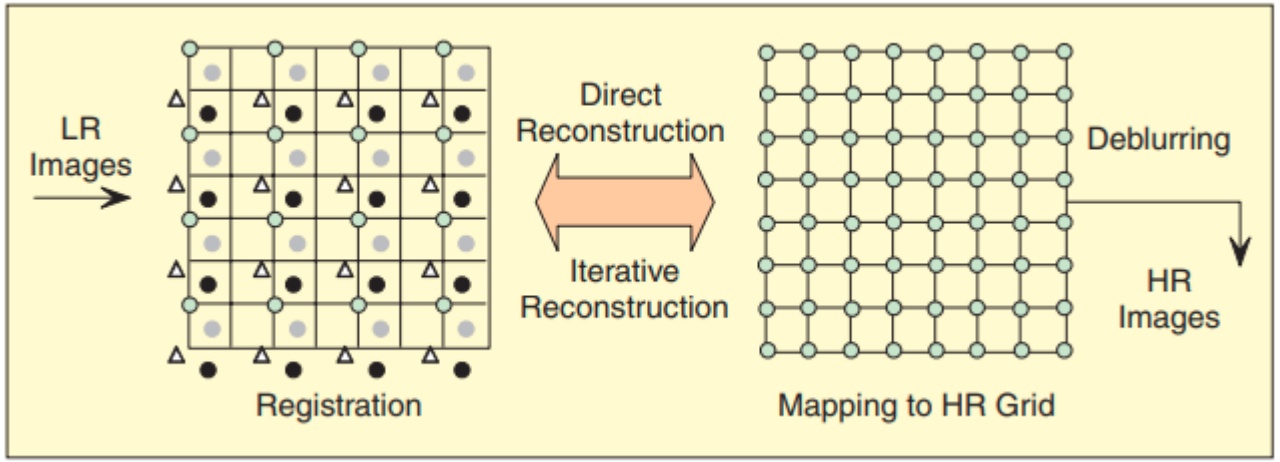
. Park, S.C., Park, M.K., Kang, M.G.: Super-resolution image reconstruction: a technical overview. IEEE Signal Process. Mag. 20(3), 21–36 (2003)

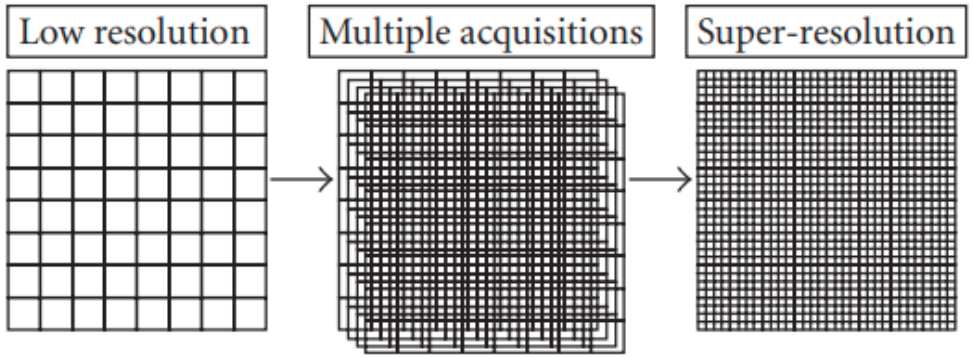
Image Super-Resolution

❖ Traditional Method





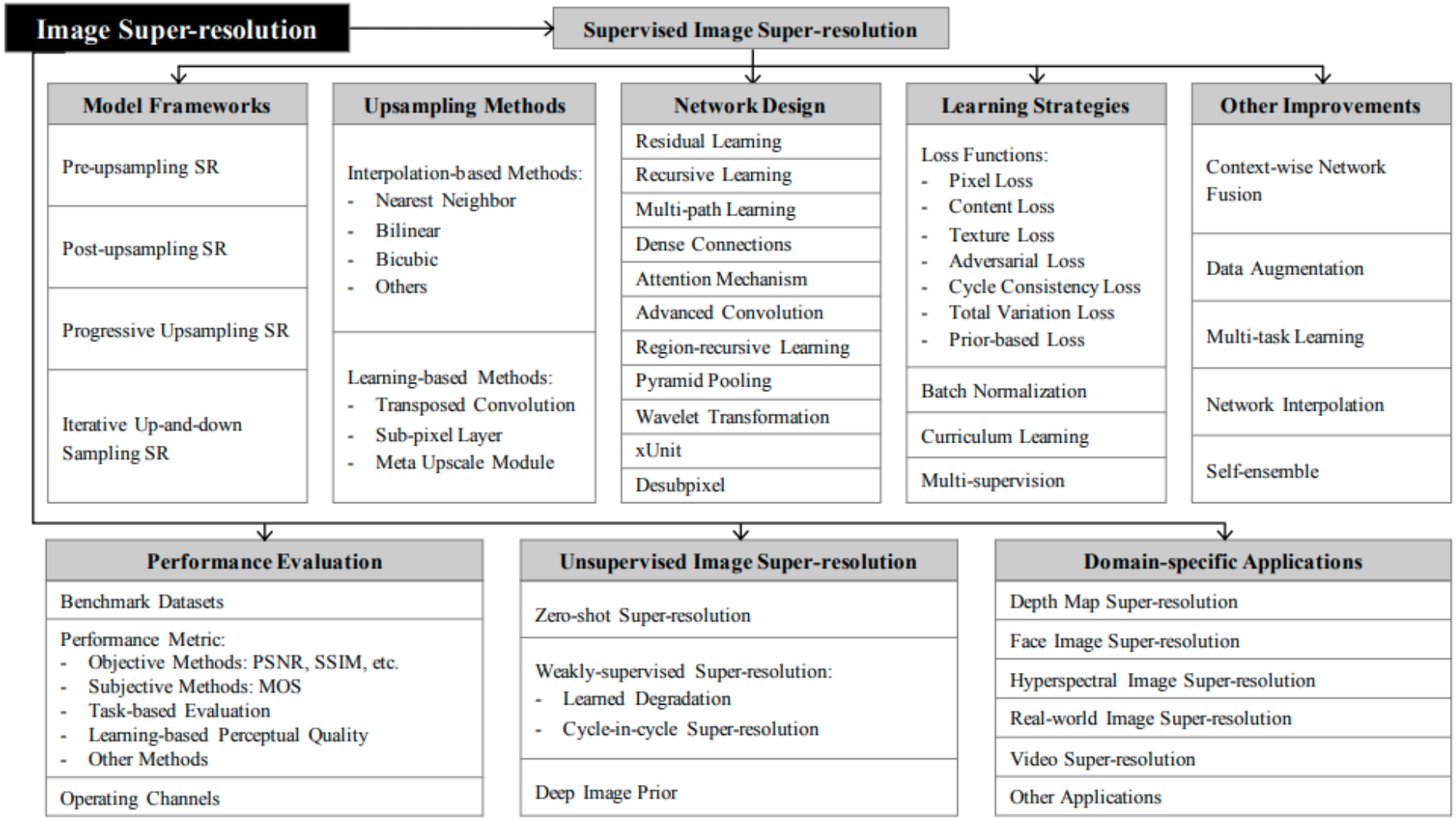




Park, S.C., Park, M.K., Kang, M.G.: Super-resolution image reconstruction: a technical overview. IEEE Signal Process. Mag. 20(3), 21–36 (2003)
Kennedy, John & Israel, Ora & Frenkel, Alex & bar-shalom, Rachel & Azhari, Haim. (2007)
Improved Image Fusion in PET/CT Using Hybrid Image Reconstruction and Super-Resolution. International journal of biomedical imaging. 2007

Image Super-Resolution

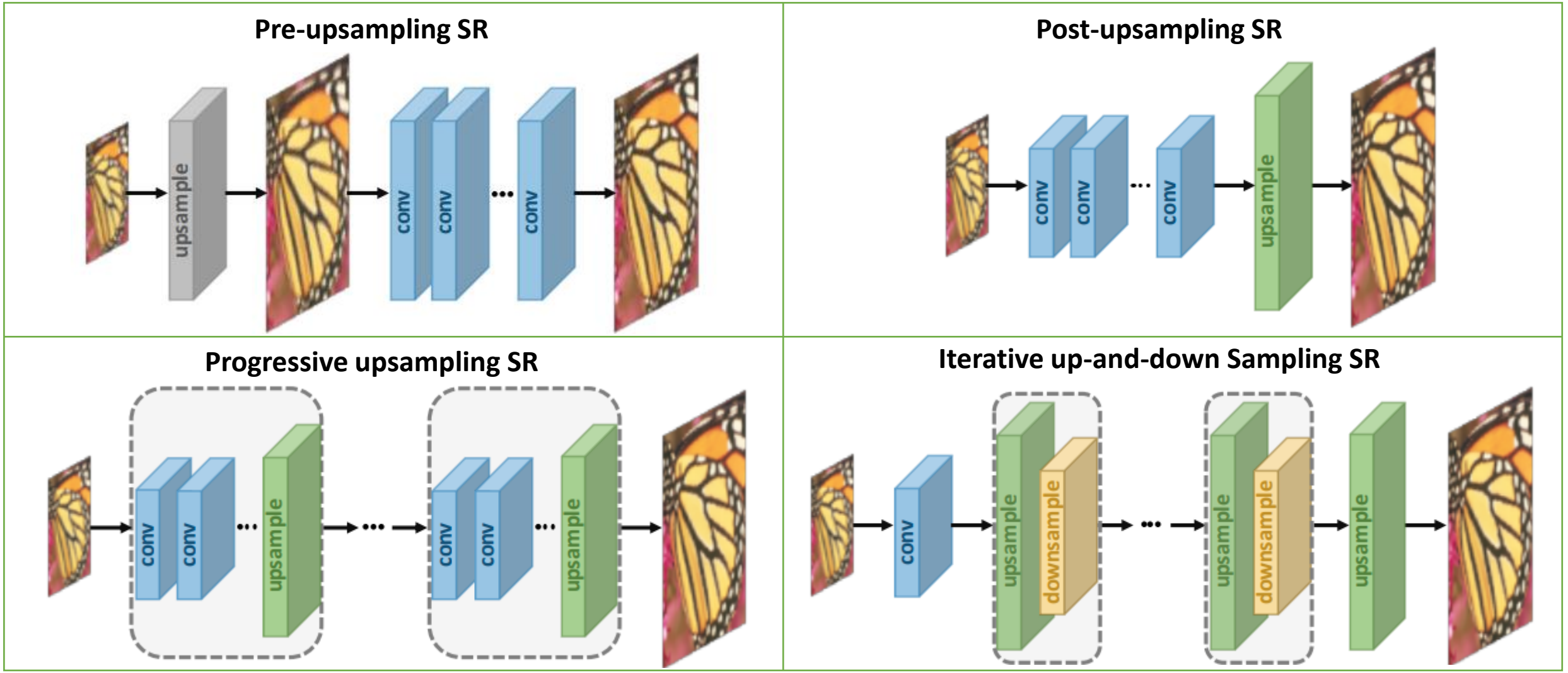
❖ Based Deep Learning Method



Wang, Z., Chen, J., & Hoi, S.C. (2021). Deep Learning for Image Super-Resolution: A Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 43, 3365-3387

Image Super-Resolution

❖ Model Frameworks

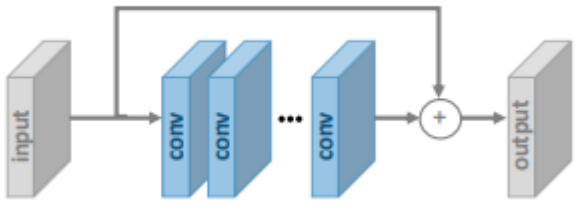


Wang, Z., Chen, J., & Hoi, S.C. (2021). Deep Learning for Image Super-Resolution: A Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 43, 3365-3387.

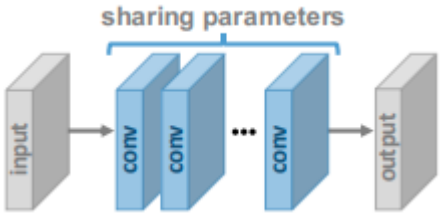
Image Super-Resolution

❖ Network Design

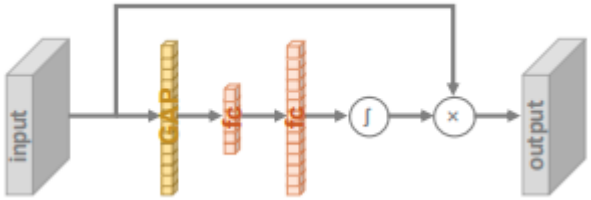
Residual Learning



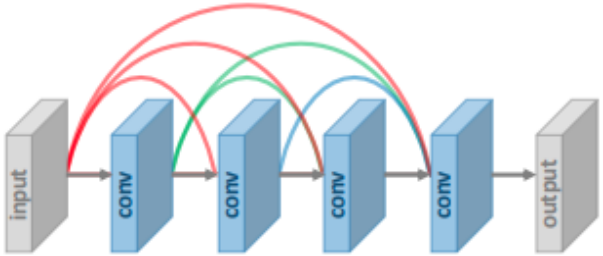
Recursive learning



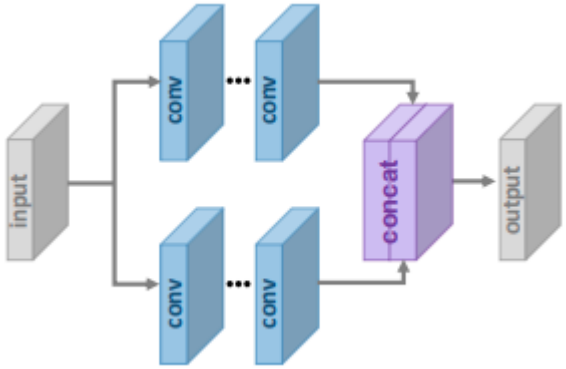
Channel attention



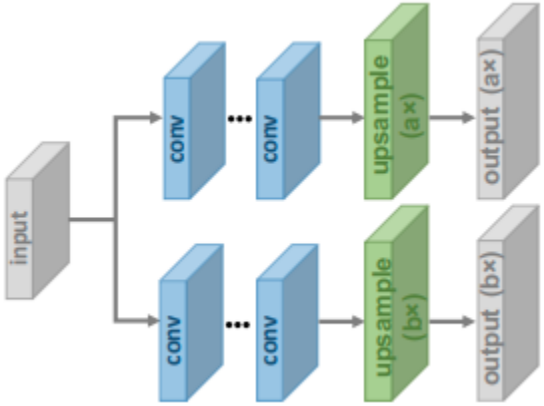
Dense connections



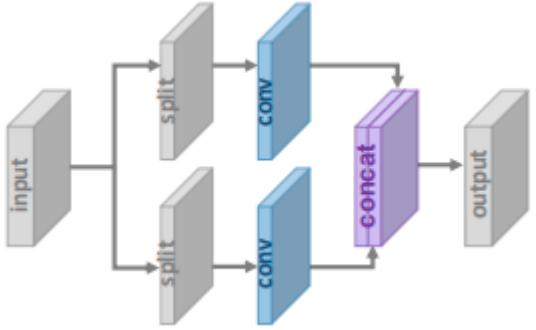
Local multi-path learning



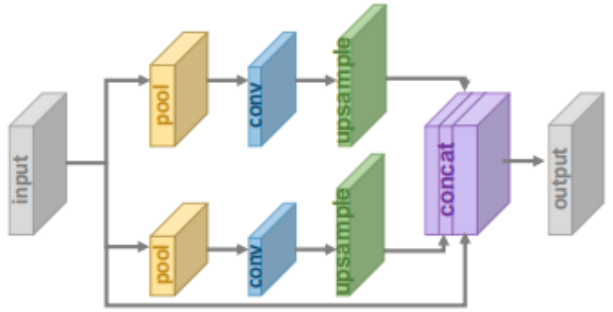
Scale-specific multi-path learning



Group convolution

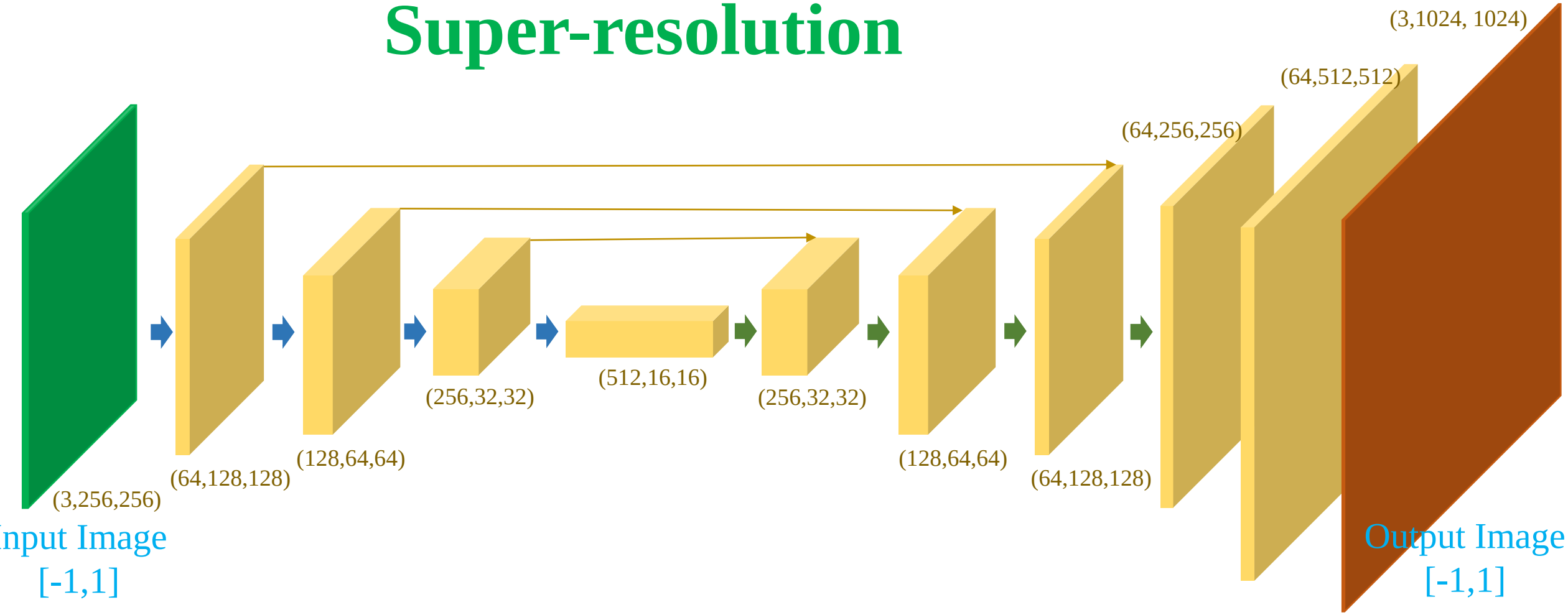


Pyramid pooling



Wang, Z., Chen, J., & Hoi, S.C. (2021). Deep Learning for Image Super-Resolution: A Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 43, 3365-3387.

Super-resolution



(3x3) Convolution padding = 'same' stride = 1 + ReLU + Batch Norm + (2x2) max pooling

4x4 ConvTranspose padding = 'same' stride = 2 + ReLU + Batch Norm

4x4 ConvTranspose padding = 'same' stride = 2 + Tanh