Adaptive Super-Resolution for ocean bathymetric maps using a deep neural network and data augmentation

Program Documentation

2026.02

JAMSTEC
Japan Agency for Marine-Earth Science and Technology

ASTOM R&D

Table of contents

1. sr_dataset_builder

1.1. Program Overview

sr_dataset_builder is a program for generating random image datasets based on block-wise partitioning. The overall procedure for creating the image dataset is described below.

1. Low-Resolution Patch Generation (16 × 16)

1-1. Missing pixels are restored using the mean of non-missing values within a **5 × 5** window.

1-2. The input bathymetry is divided into 44 × 40 blocks at 5-km intervals.

1-3. For each block, 25 low-resolution images are extracted by applying offsets at intervals of 2 pixels.

2. High-Resolution Patch Generation (64 × 64)

2-1. Missing pixels are restored using the mean of non-missing values within a **5 × 5** window.

2-2. The input bathymetry is divided into 44 × 40 blocks at 5-km intervals.

2-3. For each block, 25 low-resolution images are extracted by applying offsets at intervals of 8 pixels.

3. Filtering and Dataset Split

3-1. LR–HR patch pairs containing missing values are removed.

3-2. The dataset is divided into training data (80%), validation data (10%), and test data (10%).

3-3. When enabled, data augmentation (flip, rotation, Adaptive Mixup) is applied to training data.

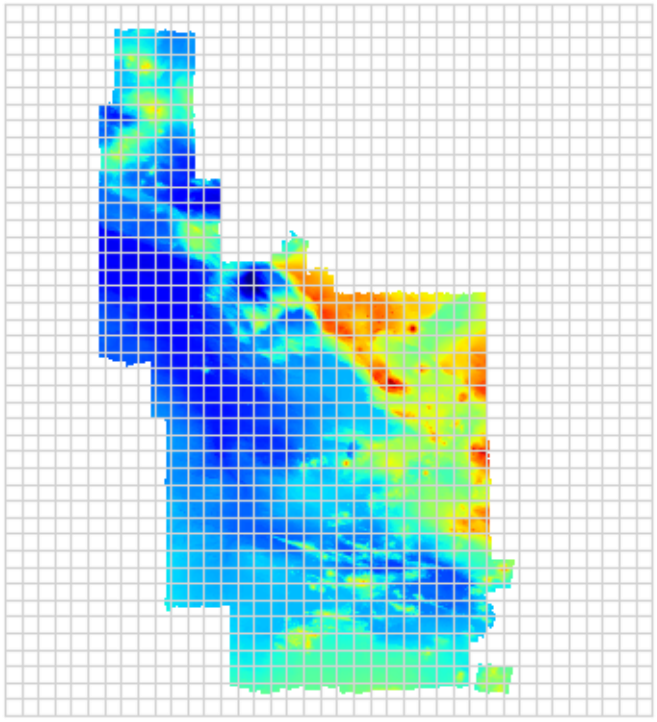

Figure 1 Example of dividing the bathymetric map into 44 × 40 blocks at 5 km intervals.

1.1.1. Adaptive Mixup: Adaptive Data Augmentation for Bathymetric Super-Resolution

Adaptive Mixup [1] is an adaptive data augmentation method consisting of Mixup-based augmentation applied to training data and data sampling adjusted to match the characteristics of the test data.
Mixup is a data augmentation technique that generates new training samples by linearly combining two different images.

Conventional data augmentation methods such as flipping and rotation only change the orientation of an image and do not alter topographic feature quantities such as slope gradient. Therefore, when the feature distributions of the training data and the target data differ, Super-Resolution performance tends to degrade.

In Adaptive Mixup, flipping and rotation are first applied to pairs of low-resolution and high-resolution images. Subsequently, the augmented data are divided into two groups based on a topographic feature (in the paper, the Mean Slope Gradient, MSG).

Image pairs are then sampled from each group, and new low-resolution and high-resolution image pairs are generated through linear combination using the following equations:

$$\tilde{x} = \lambda x_i + (1-\lambda)x_j$$
$$\tilde{y} = \lambda y_i + (1-\lambda)y_j$$

Here, x_i, x_j denote low-resolution images, y_i, y_j denote the corresponding high-resolution images, and $\lambda \in [0,1]$ represents the mixing ratio.

Through this operation, it becomes possible to artificially generate training data containing intermediate topographic features that did not exist in the original dataset.

Furthermore, the generated mixup data are sampled so that their feature distribution approximates that of the target data. This reduces the mismatch between the feature distributions of the training and target datasets and improves Super-Resolution accuracy under specific topographic conditions.

Compared with conventional data augmentation methods using only flipping and rotation, this method has been reported to improve RMSE by up to 14.3% in regions containing steep terrain while maintaining spatial structural consistency.

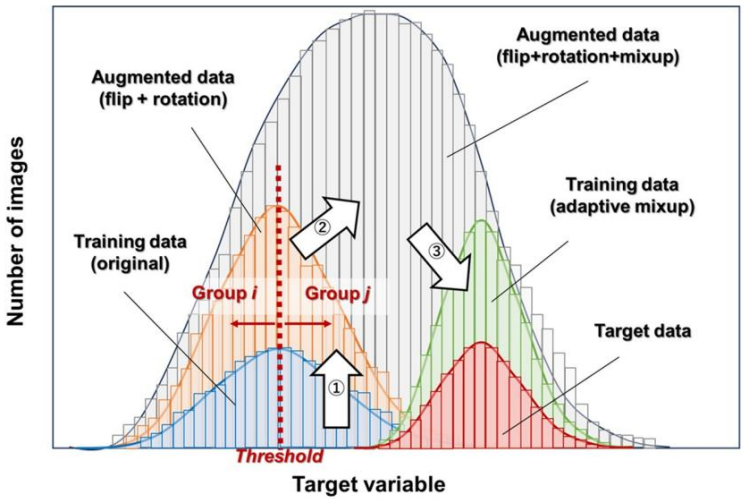

Figure 2. Conceptual diagram of adaptive data augmentation, reproduced from [1]

1.2. Functional Overview

1.2.1. dataset.Dataset Class

The Dataset class creates randomized image datasets using block-based extraction.

Attribute	Description
x_train	Low-resolution training data (e.g. shape = (num_images, 16, 16))
y_train	High-resolution training data (e.g. shape = (num_images, 64, 64))
x_validation	Low-resolution validation data／e.g. shape = (num_images, 16, 16)
y_validation	High-resolution validation data／e.g. shape = (num_images, 64, 64)
x_test	Low-resolution test data／e.g. shape = (num_images, 16, 16)
y_test	High-resolution test data／e.g. shape = (num_images, 64, 64)

__init__()

Initializes and constructs the dataset.

Argument	Description
lr_fname	Filename of the LR pickle file (e.g. shape = (1100, 1000))
hr_fname	Filename of the HR pickle file (e.g. shape = (4400, 4000))
block_shape	Number of spatial blocks／e.g. = (44, 40)

cleansing()

Restores missing values in an input bathymetric grid.

Argument	Description
img	Input image／e.g. shape = (1100, 1000) or (4400, 4000)

kernel_size	Window size for mean-value restoration／e.g. = (5, 5)

Return Value	Description
cimg	Image with missing values restored／e.g. shape = (1100, 1000) or (4400, 4000)

block_split()

Divides an input image into multiple blocks.

Argument	Description
base_img	Full input image／e.g. shape = (1100, 1000) or (4400, 4000)
block_shape	Number of blocks／e.g. = (44, 40)
img_shape	Patch size／e.g. shape = (16, 16) or (64, 64)
stride	Extraction stride／e.g. = (2, 2) or (8, 8)

Return Value	Description
-	Extracted images for each block／e.g. shape = (num_blocks, num_patches, width, height)

rm_nan()

Filters out LR/HR patch pairs that contain missing values.

Argument	Description
lr_blocks	LR patch groups／e.g. shape = (num_blocks, num_patches, width, height)
hr_blocks	HR patch groups／e.g. shape = (num_blocks, num_patches, width, height)

Return Value	Description
list[0]	filtered LR blocks／e.g. shape = (num_blocks, num_patches, width, height)
list[1]	filtered HR blocks／e.g. shape = (num_blocks, num_patches, width, height)

train_validation_test_split()

Splits LR/HR patch groups into training, validation, and test datasets.

Argument	Description
x	LR patch groups／e.g. shape = (num_blocks, num_patches, width, height)
y	HR patch groups／e.g. shape = (num_blocks, num_patches, width, height)
validation_size	Validation ratio（0～1）／e.g. = 0.1
test_size	Test ratio（0～1）／e.g. = 0.1

Return Value	Description
list[0]	Training LR patches／e.g. shape = (num_blocks, num_patches, width, height)
list[1]	Validation LR patches／e.g. shape = (num_blocks, num_patches, width, height)
list[2]	Test LR patches／e.g. shape = (num_blocks, num_patches, width, height)
list[3]	Training HR patches／e.g. shape = (num_blocks, num_patches, width, height)
list[4]	Validation HR patches／e.g. shape = (num_blocks, num_patches, width, height)
list[5]	Test HR patches／e.g. shape = (num_blocks, num_patches, width, height)

save()

Saves LR/HR datasets to the specified directory.

- Creates train, validation, and test subdirectories.
- Outputs:
  - data_LR.pkl
  - data_HR.pkl

Patch shape follows: (num_images, width, height, channels)

Argument	Description
save_dir	directory name for data output

1.3. Utility Programs

This section describes the individual utility scripts used for data augmentation and feature extraction within the bathymetric super-resolution workflow.

resize.py

Performs image resizing.

The script loads an original pickle file, converts the images to one-quarter resolution, and saves the downsampled data.

flip.py

Generates flipped images for offline data augmentation.

Within the train directory under save_dir, LR and HR image files are saved with the filenames specified in the YAML configuration.

The script accepts a YAML configuration file as its command-line argument.

YAML configuration example

```yaml
train_dir: output_origin/train
train_data: ['data_HR.pkl','data_LR.pkl']
train_savedata: ['data_HR_flip.pkl','data_LR_flip.pkl']
```

Attribute	Description
train_dir	Directory path of the input training data
train_data	Filenames of input training data
train_savedata	Output filenames for the augmented data (saved in train_dir)

rotate.py

Generates rotated images for offline data augmentation.

Augmented LR and HR images are saved in the train directory under save_dir, according to filenames set in the YAML file.

This script also accepts a YAML configuration file as a command-line argument.

YAML configuration example

```yaml
train_dir: output_cv2/train
train_data: ['data_HR.pkl','data_LR.pkl']
train_savedata: ['data_HR_rotate30_reflect.pkl','data_LR_rotate30_reflect.pkl']
#  'nearest', 'reflect', 'wrap'
fill_mode: 'reflect'
angles: [30,60,90,120,150,180,210,240,270,300,330]
```

Attribute	Description
train_dir	Directory path of input training data
train_data	Filenames of input data
train_savedata	Output filenames for the augmented dataset (saved in train_dir)
fill_mode	Fill method for areas exposed by rotation (nearest, reflect, or wrap)
angles	List of rotation angles (degrees)

scale.py

Generates scaled images in horizontal and vertical directions.
Horizontal scaling applies isotropic resizing.
Used for offline data augmentation.
The script loads LR/HR data from train_dir, applies scaling based on parameters specified in the YAML file, and saves the results in the same directory.
The script accepts a YAML configuration file as its command-line argument.

YAML configuration example

```yaml
train_dir: output_origin/train
train_data: ['data_HR.pkl','data_LR.pkl']
train_savedata: ['data_HR_scale.pkl','data_LR_scale.pkl']
# Vertical scaling factors
dscales: [0.5, 0.8, 1.2, 1.5]
# Horizontal scaling factors
xyscales: [0.5, 0.7, 0.9]
```

Attribute	Description
train_dir	Directory path of input training data
train_data	Input filenames
train_savedata	Output filenames for the augmented dataset (saved in train_dir)
dscales	Vertical scaling factors
xyscales	Horizontal scaling factors

bathymetric_map_feature.py

Computes a set of topographic features for bathymetric map data and outputs them as a CSV file.

The script takes a pickle format bathymetry dataset as input.

Execution example

```
$ python bathymetric_map_feature.py data_HR.pkl
```

This produces a CSV with the same base filename containing the following columns:

Column name	Description
index	Image index in the pickle file
mean_slope_gradient	Slope gradient (before normalization)
altitude_difference	Maximum depth difference
mean_altitude	Mean depth
mean_slope_gradient_norm_data	Slope gradient (after normalization)

This CSV is required when evaluating test data based on slope metrics.

test_da.py

Tests data augmentation operations on bathymetric patch data and visualizes the results.

The script takes a pickle format bathymetry dataset as input.

Execution example

```
$ python test_da.py data_HR.pkl
```

Running this script displays an image summarizing augmentation results.

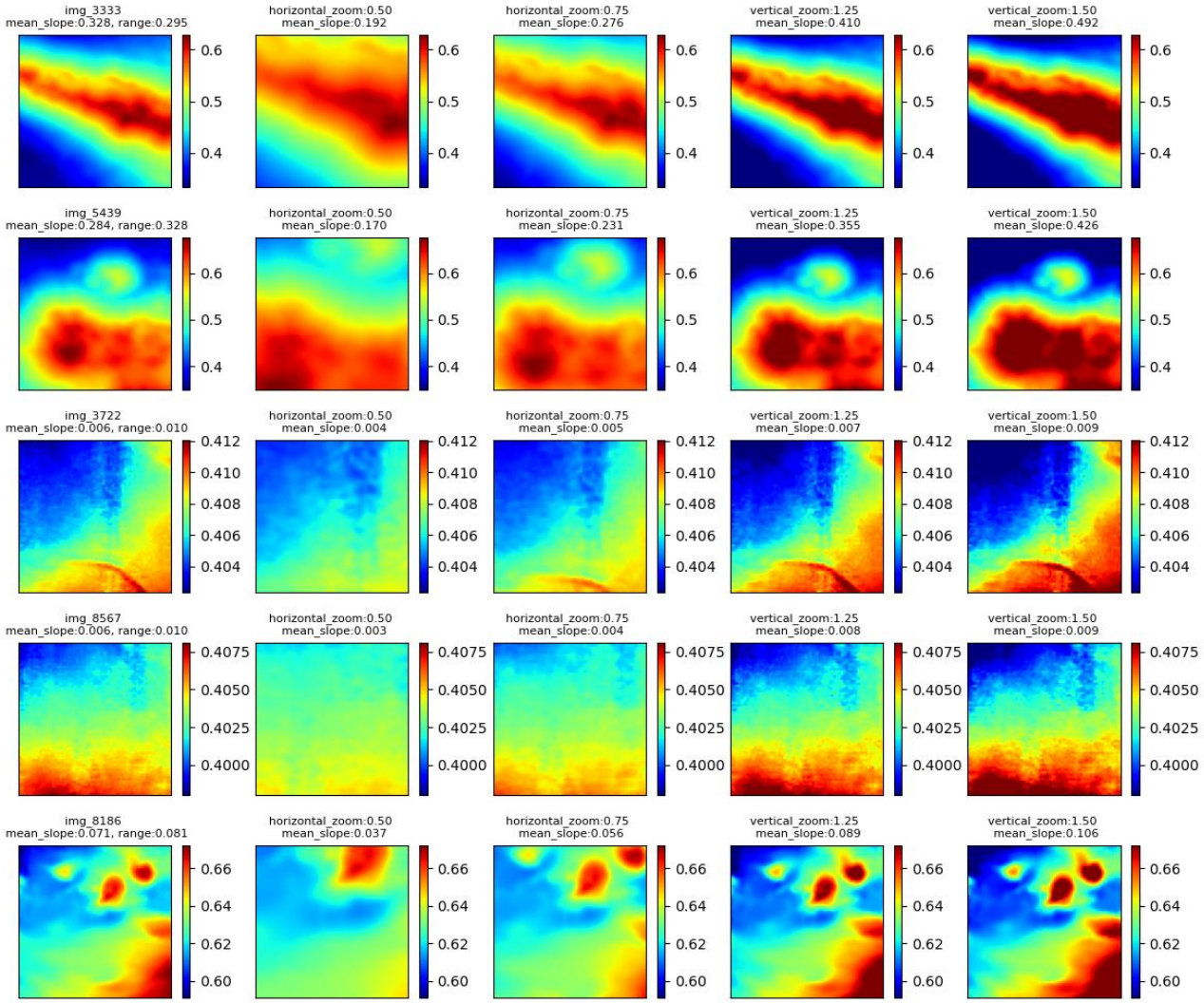

Figure 3 Output of test_da.py

bathymetric_map_feature_LR.py

Computes topographic features for low-resolution (16 × 16) bathymetric patches and outputs them as a CSV file.

This script is intended for LR patches and calculates slope gradient accordingly.

The script takes a pickle format bathymetry dataset as input.

Execution example

```
$ python bathymetric_map_feature_LR.py data_LR.pkl
```

This produces a CSV with the same base filename containing the following columns:

Column name	Description
index	Image index in the pickle file

mean_slope_gradient	Slope gradient (before normalization)
altitude_difference	Maximum depth difference
mean_altitude	Mean depth
mean_slope_gradient_norm_data	Slope gradient (after normalization)
normalized_mean_slope_gradient	Image index in the pickle file

This script determines the normalization range for the entire input dataset using nanmin and nanmax and also computes the slope gradient for the normalized data.
The CSV generated by this script is required for slope-based data distribution analysis or data augmentation (mixup.py).

mixup.py

Performs Adaptive Mixup, a data augmentation method that adjusts the training dataset's topographic-feature distribution.
Using topographic feature metrics (such as the mean slope gradient), LR/HR image pairs from groups with higher and lower mean slope gradient values are combined to generate synthetic data aligned with the target distribution.

YAML configuration example

```yaml
train_dir: output/train
train_data: ['data_HR.pkl','data_LR.pkl']
target_data_csv: data_LR.csv
# 0: normalized_mean_slope_gradient, 1: altitude_difference, 2:
mean_slope_gradient_norm_data, 3: mean_slope_gradient
target_value: 3
threshold: 0.1
std: 0.05
save_dir_name: mixup_data
random_split: False
alpha: 0.5
lamb: 0.8
plot_figure: False
original_image_num: 11053
increase_rate: 2

get_slope_difference:
  valid: True
  input_data_dir: oki_test_inter/test
```

```
delta: 50
```

Attribute	Description
train_dir	Directory path containing the input training data.
train_data	Filenames of the input LR and HR training data.
target_data_csv	CSV file containing the topographic-feature values (e.g., mean slope gradient) used as the target distribution.
target_value	Index specifying which feature column to use for distribution control （0: normalized_mean_slope_gradient, 1: altitude_difference, 2: mean_slope_gradient_norm_data, 3: mean_slope_gradient）
threshold	Threshold value used to divide samples into groups based on the selected topographic feature.
std	Standard deviation value used when forming the distribution boundary for grouping.
save_dir_name	Name of the output directory that will be created under train_dir for Mixup results.
random_split	Enables or disables random grouping.
alpha	Mixup parameter.
lamb	Mixup parameter.
plot_figure	If set to True, the script outputs visualization figures of Mixup results.
original_image_num	Number of original images before any data augmentation.
increase_rate	Factor specifying how many times the dataset should be expanded compared to the original

get_slope_difference section

Attribute	Description
get_slope_difference.valid	Enables or disables slope-difference calculation.
get_slope_difference.input_data_dir	Directory containing input data used for calculating slope differences.
get_slope_difference.delta	Grid spacing used in slope-difference computation.

load_csv_norm()

Reads the feature CSV file and extracts groups of high–mean-slope-gradient data and low–mean-slope-gradient data based on the specified feature column.
Original, flipped, and rotated images are treated as belonging to the same group.
The method normalizes the feature values and generates reference data for distribution matching.

Attribute	Description
data_csv	Path to the feature CSV for the training dataset／e.g. = "train/data_LR.csv"
target_key	Feature to be used for distribution control. (mean_slope_gradient, mean_slope_gradient_norm_data, altitude_difference or normalized_mean_slope_gradient)
target_data_csv	Feature CSV used as the target distribution ／ e.g. = "target/data_LR.csv"
save_dir	Output directory／e.g. = "train/mixup1"
original_image_num	Number of original (pre-augmentation) images／e.g. = 1000

Return Value	Description
index_high	indices of high-slope patches ／e.g. shape = (num_augmentation_type, num_high_slope_patches)
index_low	indices of low-slope patches ／e.g. shape = (num_augmentation_type, num_low_slope_patches)
slope_high	feature values of high-slope patches ／e.g. shape = (num_augmentation_type, num_high_slope_patches)
slope_low	feature values of low-slope patches ／e.g. shape = (num_augmentation_type, num_low_slope_patches)
df	DataFrame of the full input dataset
target_df	DataFrame of the target distribution

load_imgs()

Loads LR/HR images stored in pickle format and converts them into a consistent array structure for processing.

Attribute	Description
imgs_file	Path to the pickle file／e.g. = "train/data_LR.pkl"

Return Value	Description
imgs	processed image array／e.g. shape = (num_images, width, height) or

	(num_images, width, height, num_channels)
imgs_org	original array loaded from pickle／e.g. shape = (num_images, width, height) or (num_images, width, height, num_channels)

mixup_norm()

Applies Adaptive Mixup based on topographic features.
Pairs of LR/HR images from the high-slope and low-slope groups are linearly combined to generate synthetic patches whose topographic-feature distribution is closer to the target.

Attribute	Description
imgs_HR	normalized HR images／e.g. shape = (None, 64, 64) or (None, 64, 64, C)
imgs_HR_org	original-scale HR images／e.g. shape = (None, 64, 64) or (None, 64, 64, C)
imgs_LR	normalized LR images／e.g. shape = (None, 16, 16) or (None, 16, 16, C)
imgs_LR_org	original-scale LR images／e.g. shape = (None, 16, 16) or (None, 16, 16, C)
index_high	indices of high-slope images ／e.g. shape = (num_augmentation_type, num_high_slope_patches)
index_low	indices of low-slope images ／e.g. shape = (num_augmentation_type, num_low_slope_patches)
train_dir	training data directory／e.g. = "train"
target_df	target distribution DataFrame
target_key	Feature to be used for distribution control. (mean_slope_gradient, mean_slope_gradient_norm_data, altitude_difference or normalized_mean_slope_gradient)
slope_normalized	flag indicating whether normalized slope values are used／e.g. = True

Return Value	Description
-	A set of Mixup-generated LR/HR data and intermediate results

get_slope()

Computes the mean slope gradient for a high-resolution (64 × 64) bathymetric patch using finite-difference approximations.

Attribute	Description

h	input HR bathymetric patch ／ shape = (64, 64)
delta	grid spacing in original scale ／default = 50

Return Value	Description
-	Mean slope gradient value／e.g. = 0.03

get_slope_LR()

Computes the mean slope gradient for a low-resolution (16 × 16) bathymetric patch using finite-difference approximations.

Attribute	Description
h	input LR bathymetric patch／shape = (16, 16)
delta	grid spacing in original scale

Return Value	Description
-	Mean slope gradient value／e.g. = 0.02

pick_images()

Selects LR/HR images from the mixup-generated dataset so that the overall feature distribution (e.g., mean slope gradient) matches the target distribution.
The method:

- Reads data_LR.csv generated in the mixup output directory
- Determines the required number of samples for each histogram bin
- Prioritizes Mixup images and flip/rotation images
- Outputs selected images into saved_dir/picked_images/ directory
- Computes feature values for the selected LR patches

Attribute	Description
target_df	target feature distribution
save_dir	directory containing Mixup results ／e.g. = "train/mixup1"
imgs_HR_mixup	HR images including Mixup results ／e.g. shape = (None, 64, 64, 1)
imgs_LR_mixup	LR images including Mixup results ／e.g. shape = (None, 16, 16, 1)
original_image_num	number of original images ／e.g. = 1000
target_key	Feature to be used for distribution control.

	(mean_slope_gradient, mean_slope_gradient_norm_data, altitude_difference or normalized_mean_slope_gradient)
data_csv	CSV file of the original dataset ／e.g. = "train/data_LR.csv"
increase_rate	multiplier indicating how many times the target distribution should be amplified

idx_of_the_nearest()

Returns the index of the element in an array that is closest to a given value.

Attribute	Description
data	array of values (list, numpy.ndarray)
value	target value ／e.g. = 0.35

Return Value	Description
-	Index of the nearest element

indices_of_the_nearest()

Returns all indices of elements whose value is closest to a specified target value. (assuming that multiple elements may have the same distance)

Attribute	Description
data	array of values (list, numpy.ndarray)
value	target value ／e.g. = 0.35

Return Value	Description
-	Array of indices with the smallest absolute difference／e.g. shape = (L,) (L:number of elements with same distance)

1.4. Recommended Environment

The program runs on Python 3.8 and requires the following packages:

Package	Version (recommended)
pickle	*
pathlib	*
numpy	1.21.2
tqdm	4.62.3

1.5. Usage

Initialize the Dataset class and construct the dataset:

```
from dataset import Dataset
lr_fname = 'input/okinawa_2D_100m.pkl'
hr_fname = 'input/okinawa_2D_50m.pkl'
dataset = Dataset(lr_fname, hr_fname)
```

Save the generated training, validation, and test datasets:

```
dataset.save(save_dir='output')
```

When executed successfully, progress information is displayed as follows:

```
loading... (input/okinawa_2D_100m.pkl)
cleansing: 100%|████████████████████| 2908486/2908486 [00:15<00:00, 189839.98it/s]
splitting: 100%|████████████████████████████| 1760/1760 [00:00<00:00, 51791.70it/s]
loading... (input/okinawa_2D_50m.pkl)
cleansing: 100%|████████████████████| 11659333/11659333 [00:59<00:00, 197567.03it/s]
splitting: 100%|████████████████████████████| 1760/1760 [00:00<00:00, 53420.96it/s]
removing nan... done
splitting data... done
saving (output)... done
------------------------------------------------------------------
    train: x = (11022, 32, 32, 1), y = (11022, 64, 64, 1)
validation: x = (1333, 32, 32, 1), y = (1333, 64, 64, 1)
     test: x = (1368, 32, 32, 1), y = (1368, 64, 64, 1)
------------------------------------------------------------------
```

2. sr_trainer

2.1. Program Overview

sr_trainer is a program that performs deep learning-based Super-Resolution training and testing using the following algorithms: SRCNN, FSRCNN, ESPCN, SRGAN, and ESRGAN.

2.1.1. SRCNN: Super-Resolution Convolutional Neural Network

SRCNN [2] is a three-layer convolutional neural network.

The input and output image sizes within the network are identical, and therefore the low-resolution image must be upscaled in advance using methods such as bicubic interpolation.

Super-Resolution approaches that enlarge the image prior to network input are referred to as Pre-upsampling SR.

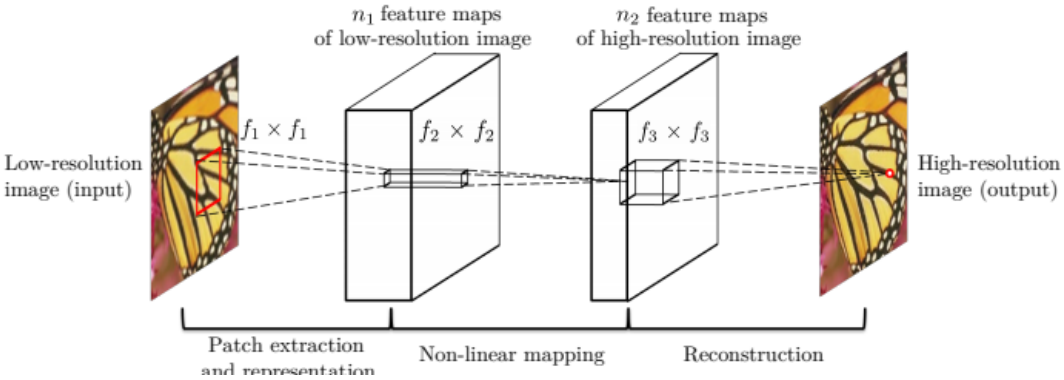

Figure 4 Conceptual diagram of the SRCNN (Super-Resolution Convolutional Neural Network) architecture, reproduced from [2].

2.1.2. FSRCNN: Fast Super-Resolution Convolutional Neural Network

FSRCNN [3] is an algorithm designed to accelerate SRCNN.
Since SRCNN takes pre-upsampled images as input, the input image size becomes large. In contrast, FSRCNN directly applies a convolutional neural network to low-resolution images and enlarges the images at the final layer using deconvolution.

Super-Resolution approaches that upscale the image after passing through the network are referred to as Post-upsampling SR.

In the original FSRCNN paper, kernel sizes and other architectural parameters were jointly optimized, achieving speed improvements of several tens of times compared to SRCNN.Input image tensor

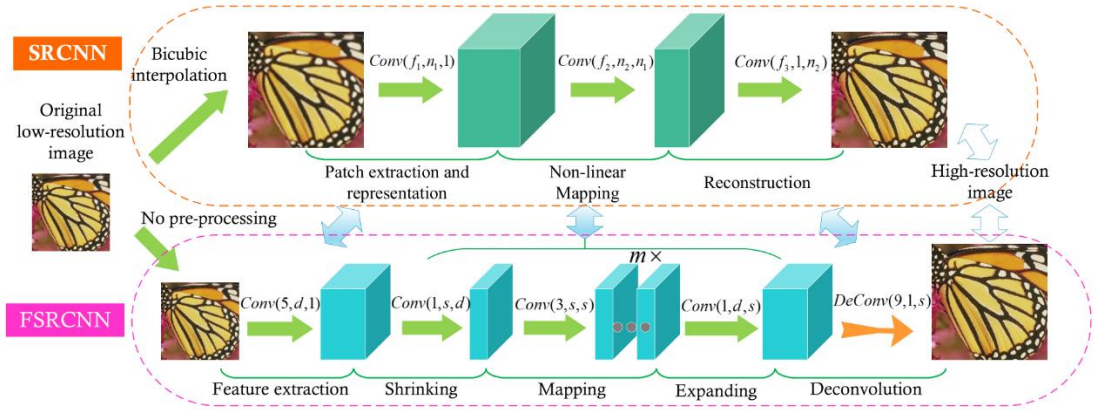

Figure 5 Network architecture of FSRCNN (Fast Super-Resolution Convolutional Neural Network), reproduced from [3].

2.1.3. ESPCN: Efficient Sub-Pixel Convolutional Neural Network

ESPCN [4], similar to FSRCNN, adopts a Post-upsampling SR approach. However, instead of using deconvolution, ESPCN performs image upscaling through Sub-Pixel Convolution.

As illustrated in the figure below, deconvolution refers to a convolution operation in which the number of referenced pixels varies depending on spatial location, making it prone to the manifestation of checkerboard artifacts. Furthermore, since the pixels inserted during the deconvolution process do not contain intrinsic information, they may lead to inefficient learning.

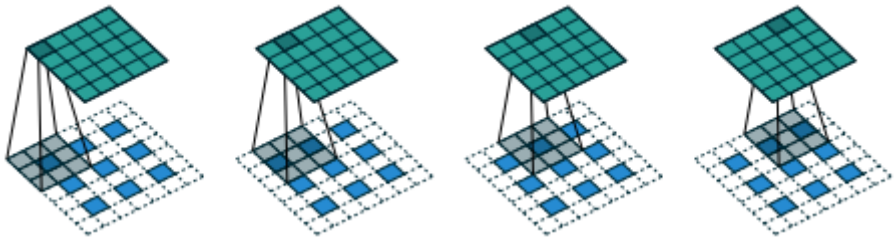

Figure 6 Conceptual diagram of upsampling using Sub-Pixel Convolution, reproduced from [5].

The Sub-Pixel Convolution employed in ESPCN enlarges images by rearranging channel-wise pixel values without inserting interpolated pixels. In this process, the number of referenced pixels becomes spatially uniform, thereby suppressing checkerboard artifacts, which are a side effect of deconvolution.

Moreover, since no additional pixels are inserted, ESPCN achieves further acceleration compared to FSRCNN.

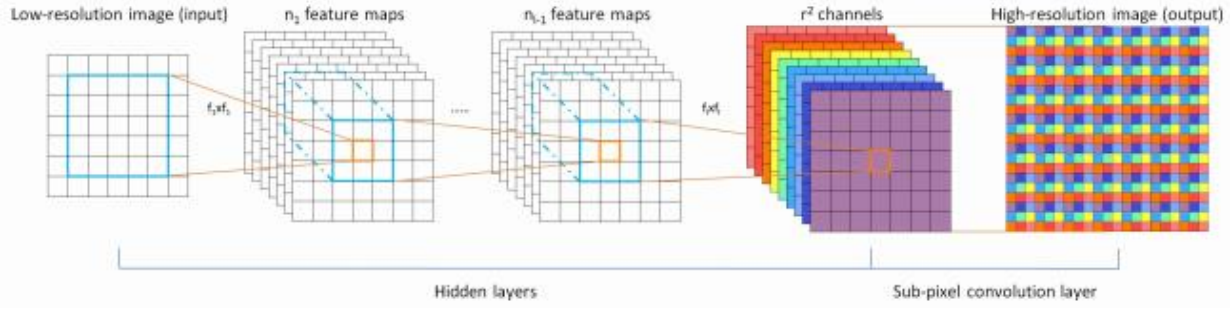

Figure 7 Network architecture of ESPCN (Efficient Sub-Pixel Convolutional Neural Network), reproduced from [4].

2.1.4. SRGAN: Super-Resolution using Generative Adversarial Network

SRGAN [6] is a method that improves Super-Resolution quality by enhancing performance through the cooperative training of a Generator and a Discriminator.

The Generator produces high-resolution images from low-resolution inputs. The Discriminator, on the other hand, determines whether an input image is a super-resolved image generated by the Generator or an original high-resolution image.

By alternately training the Generator and the Discriminator and balancing the image reconstruction loss of the Generator with the classification loss of the Discriminator, the method surpasses the performance limitations of using the Generator alone.Input image tensor

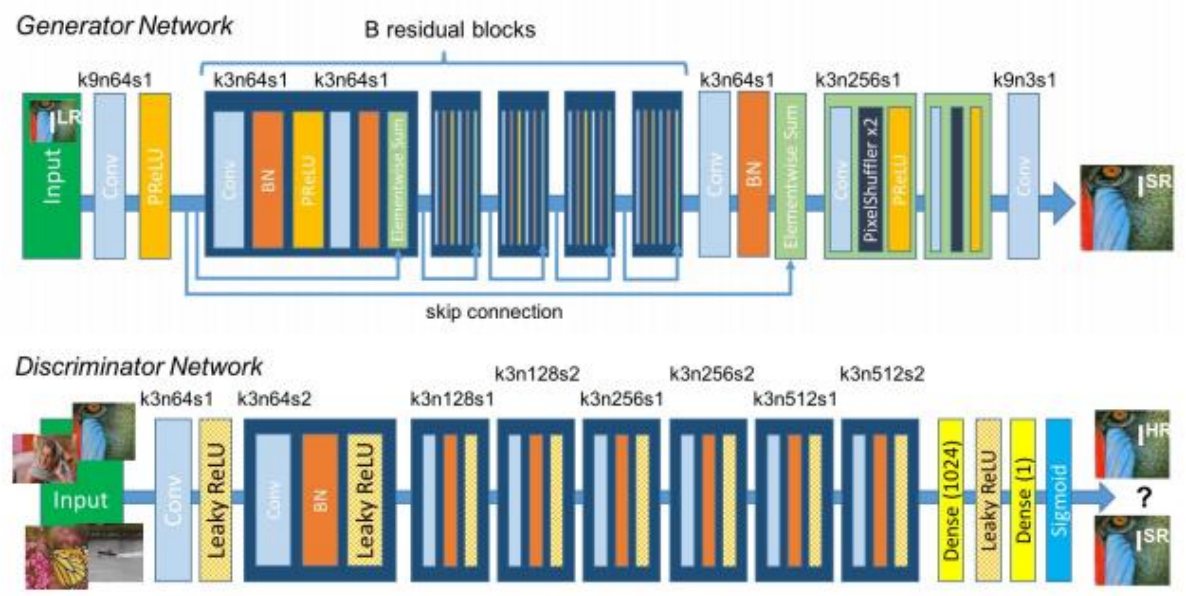

Figure 8 Network architecture of SRGAN (Super-Resolution Generative Adversarial Network), reproduced from [6].

2.1.5. ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks

ESRGAN [7] is an improved version of SRGAN designed to enhance representational capacity and training efficiency. The following describes three key modifications introduced in ESRGAN compared to SRGAN.

(1) Removal of Batch Normalization in the Generator

As noted in the Supplemental Material of the ESRGAN paper, Batch Normalization may introduce unnatural artifacts. This issue arises because the batch statistics used during training differ from those used during testing. Specifically, during training, normalization is performed using the exact batch mean and variance computed from the current batch, whereas during testing, estimated (running) mean and variance are used. For example, when the model is trained with a batch size of 16 but tested using a single image, sufficient statistical information equivalent to 16 images cannot be obtained from the single test image.

For this reason, Batch Normalization is removed from the Generator in ESRGAN.

(2) Extension of the Residual Block

In ESRGAN, the Residual Block used in SRGAN is extended to the Residual-in-Residual Dense Block (RRDB), as illustrated in the figure below. In RRDB, weak residual connections scaled by a coefficient β (β ≈ 0.2) are introduced, allowing features from a broader range of preceding and subsequent layers to be effectively reflected.

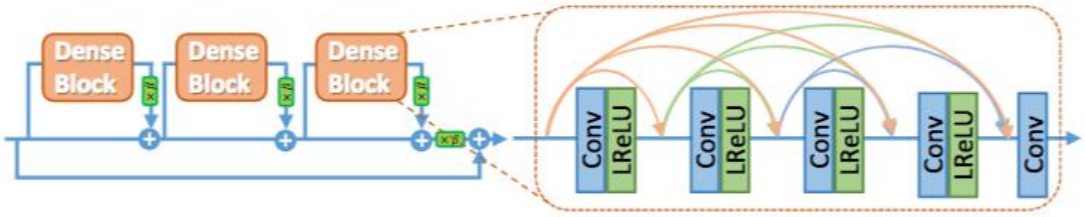

Figure 9 Conceptual diagram of the Residual-in-Residual Dense Block (RRDB) architecture, reproduced from [7].

(3) Introduction of the Relativistic Discriminator

In SRGAN, once the Discriminator training progresses sufficiently, it rarely misclassifies real images. As a result, the Discriminator improves only when it mistakenly classifies fake images as real. The behavior of the Discriminator $D(x)$ for a batch of real images x_r and fake images x_f is expressed as follows:

$$D(x_r) = \sigma\big(C(x_r)\big) \to 1 = Real$$
$$D\big(x_f\big) = \sigma(C\big(x_f\big)) \to 0 = Fake$$

Here, σ denotes the sigmoid function, and C represents the output value of the Discriminator. However, under this formulation, only approximately half of the inputs contribute effectively to learning in the later training stages, resulting in inefficiency.

To address this issue, ESRGAN introduces the following Relativistic Discriminator$D_{Ra}(x)$:

$$D_{Ra}\big(x_r, x_f\big) = \sigma\big(C(x_r) - \mathrm{E}\big[C\big(x_f\big)\big]\big) \to 1 = Real$$
$$D_{Ra}\big(x_f, x_r\big) = \sigma\big(C\big(x_f\big) - \mathrm{E}[C(x_r)]\big) \to 0 = Fake$$

Here, E denotes the batch mean.

The first equation enforces that the predicted value for real images should be higher than the average predicted value for fake images, while the second equation enforces that the predicted value for fake images should be lower than the average predicted value for real images. Consequently, unless classification is perfect, learning progresses for both real and fake images.

The loss function of the Relativistic Discriminator $L_D^{Ra}(x)$ is defined by combining the two expressions in parallel:

$$L_D^{Ra}\big(x_r, x_f\big) = -\mathrm{E}\left[\log\Big(D_{Ra}\big(x_r, x_f\big)\Big)\right] - \mathrm{E}\big[\log\big(1 - D_{Ra}(x_f, x_r)\big)\big]$$

During Generator training, in order to deceive the Relativistic Discriminator, x_r and x_f are exchanged in the loss computation.

$$L_D^{Ra}\big(x_r, x_f\big) = -\mathrm{E}\left[\log\Big(D_{Ra}\big(x_r, x_f\big)\Big)\right] - \mathrm{E}\big[\log\big(1 - D_{Ra}(x_f, x_r)\big)\big]$$

2.2. Functions

2.2.1. Class Diagram

The relationships among the classes in sr_trainer are shown below as a UML class diagram (attributes and methods are omitted).

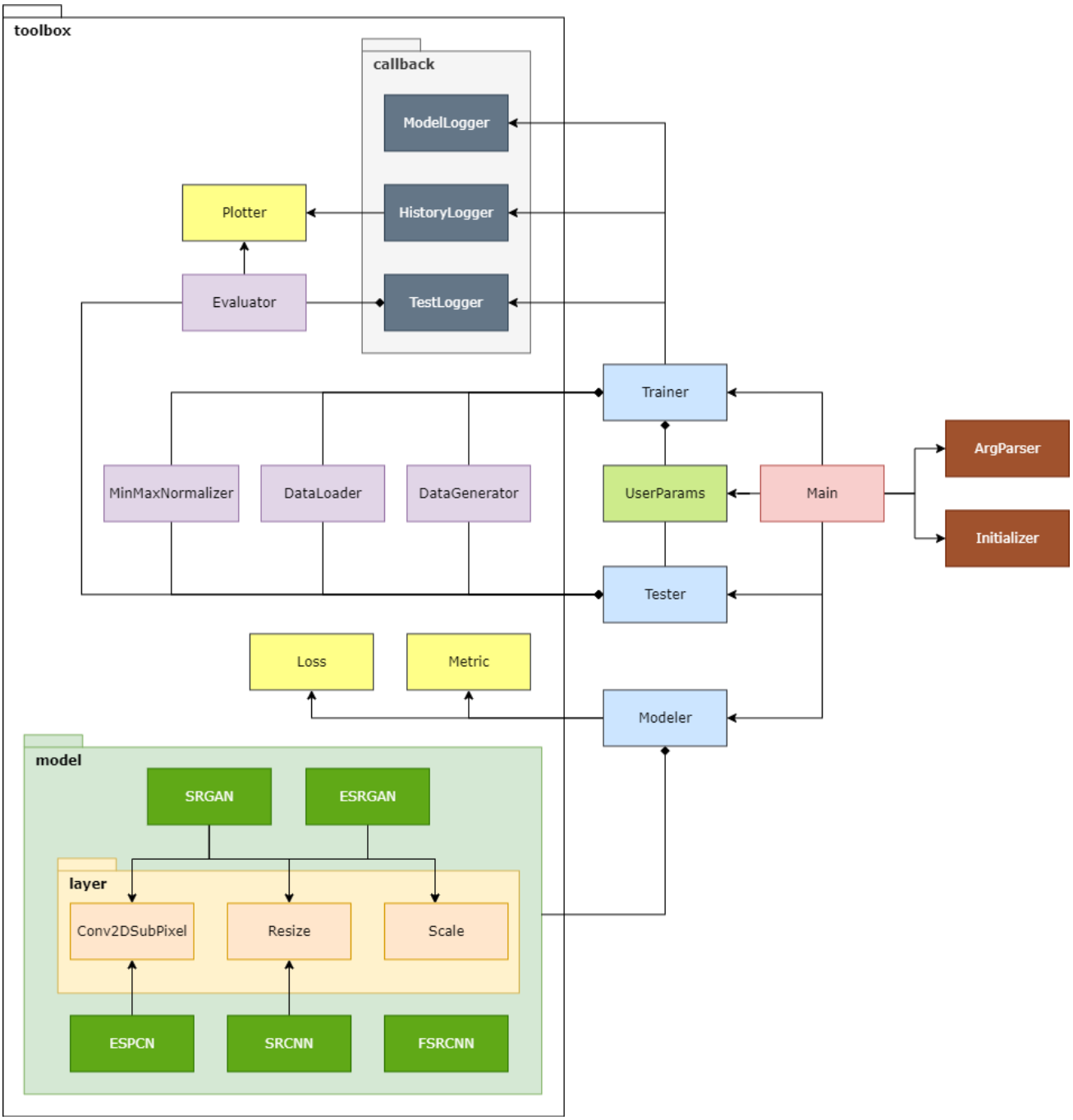

Figure 10 Class Diagram

2.2.2. toolbox.model.layer.Resize Class

This class inherits from keras.engine.topology.Layer and implements a custom layer for image upscaling (e.g., bilinear and bicubic interpolation).

Attribute	Description
scale	Upscaling factor／e.g. = 2
method	Image interpolation method ／ default = tf.image.ResizeMethod.BICUBIC

__init__()

Initializes the class attributes and the parent class keras.engine.topology.Layer.

Argument	Description
scale	Upscaling factor／e.g. = 2
method	Image interpolation method ／ default = tf.image.ResizeMethod.BICUBIC
trainable	Whether the layer is trainableTrainability／default = False
margin	Margin width (in pixels) applied around the resized image／default = 0
**kwargs	Additional keyword arguments

resized_shape()

Obtains the image shape after upscaling (excluding the leading batch dimension and the trailing channel dimension).

Argument	Description
input_shape	Input image tensorInput image shape／e.g. shape = (None, 32, 32, 1)

Return Value	Description
-	Upscaled image shape／e.g. shape = (64, 64)

call()

Performs image upscaling using tf.image.resize_image.

Argument	Description
x	Input image tensor／e.g. shape = (None, 32, 32, 1)

Return Value	Description
-	Upscaled image tensor／e.g. shape = (None, 64, 64, 1)

compute_output_shape()

Returns the output image shape.

Argument	Description
input_shape	Input image tensorInput image shape／e.g. shape = (None, 32, 32, 1)

Return Value	Description
-	Output image shape／e.g. shape = (None, 64, 64, 1)

get_config()

Returns the configuration of the layer with the added attributes.

Return Value	Description
-	Dictionary of layer configurationDictionary of layer configuration

2.2.3. toolbox.model.layer.Conv2DSubPixel Class

This class inherits from keras.engine.topology.Layer and implements a custom layer that performs Sub-Pixel Convolution.

Attribute	Description
scale	Upscaling factor／e.g. = 2

__init__()

Initializes the class attributes and the parent class keras.engine.topology.Layer.。

Argument	Description
scale	Upscaling factor／e.g. = 2
trainable	Trainability／default = False
**kwargs	Additional keyword arguments

call()

Performs Sub-Pixel Convolution by utilizing tf.depth_to_space.

Argument	Description
x	Input image tensor／e.g. shape = (None, 16, 16, 1)

Return Value	Description
-	Upscaled image tensor／e.g. shape = (None, 64, 64, 1)

compute_output_shape()

Obtains the shape of the output image.

Argument	Description
input_shape	Input image tensorInput image shape／e.g. shape = (None, 16, 16, 1)

Return Value	Description
-	Output image shape／e.g. shape = (None, 64, 64, 1)

get_config()

Retrieves the configuration of the layer, including the added attributes.

Return Value	Description
-	Dictionary of layer configuration

2.2.4. toolbox.model.layer.Scale Class

This class inherits from keras.engine.topology.Layer and implements a custom layer that scales the image by a constant factor.

Attribute	Description
factor	Scaling factor／e.g. = 0.2

__init__()

Initializes the class attributes and the parent class keras.engine.topology.Layer.

Argument	Description
factor	Scaling factor／e.g. = 0.2
trainable	Trainability／default = False
**kwargs	Additional keyword arguments

call()

Scales the image by a constant factor using a Lambda layer.

Argument	Description
x	Input image tensor／e.g. shape = (None, 16, 16, 1)

Return Value	Description
-	Image scaled by a constant factor／e.g. shape = (None, 16, 16, 1)

compute_output_shape()

Obtains the shape of the output image.

Argument	Description
input_shape	Input image shape／e.g. shape = (None, 16, 16, 1)

Return Value	Description
-	Output image shape／e.g. shape = (None, 16, 16, 1)

get_config()

Retrieves the configuration of the layer, including the added attributes.

Return Value	Description

-	Dictionary of layer configuration

2.2.5. toolbox.model.srcnn.SRCNN Class

This class inherits from keras.models.Model and constructs the SRCNN model.

Attribute	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2
params	Parameter configuration

__init__()

After initializing the keras.models.Model class, the __build__ method is invoked to construct the SRCNN model.

Argument	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2

__build__()

Constructs the SRCNN model. The number of channels and kernel sizes follow the original paper. Pre-upsampling using bicubic interpolation is implemented through the custom layer toolbox.model.layer.Resize.

Argument	Description
filters	Number of channels in the convolutional layer／default = [32, 16]
kernels	Kernel size of the convolutional layer／default = [9, 1, 5]

2.2.6. toolbox.model.fsrcnn.FSRCNN Class

This class inherits from keras.models.Model and constructs the FSRCNN model.

Attribute	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2

__init__()

After initializing the keras.models.Model class, the __build__ method is invoked to construct the FSRCNN model.

Argument	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2

__build__()

Constructs the FSRCNN model. The number of channels and kernel sizes follow the original paper.

Argument	Description
d	Number of channels in the first and last convolutional layers／default = 56
s	Number of channels in the intermediate convolutional layers／default = 12
m	Number of repetitions of the intermediate convolutional layers／default = 4

2.2.7. toolbox.model.espcn.ESPCN Class

This class inherits from keras.models.Model and constructs the ESPCN model.

Attribute	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2

__init__()

After initializing the keras.models.Model class, the __build__ method is invoked to construct the ESPCN model.

Argument	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2

__build__()

Constructs the ESPCN model. The number of channels and kernel sizes follow the original paper. Sub-Pixel Convolution is implemented using the custom layer toolbox.model.layer.Conv2DSubPixel.

Argument	Description
filters	Number of channels in the convolutional layer／default = [64, 32]
kernels	Kernel size of the convolutional layer／default = [5, 3, 3]

2.2.8. toolbox.model.srgan.Generator Class

This class inherits from keras.models.Model and constructs the Generator model of SRGAN.

Attribute	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2

__init__()

After initializing the keras.models.Model class, the __build__ method is invoked to construct the Generator model of SRGAN.

Argument	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2

__residual_block()

Constructs a residual block composed of Conv2D, BatchNormalization, and PReLU layers.

Argument	Description
filters	Number of channels in the convolutional layer／e.g. = 32
kernel_initializer	Kernel initialization／e.g. = 'he_normal'
momentum	Momentum coefficient for the moving average in BatchNormalization／default = 0.8

Return Value	Description
-	Residual block

__upsampling_block()

Constructs an upsampling block for image enlargement. Sub-Pixel Convolution is employed for the upsampling process. However, as pointed out in distill.pub (https://distill.pub/2016/deconv-checkerboard/
), upsampling using nearest-neighbor or bilinear interpolation is effective in preventing checkerboard artifacts. Nevertheless, such approaches may exacerbate noise near image boundaries. Therefore, the internal function resize_conv is currently left unused.

Argument	Description
filters	Number of channels in the convolutional layer／e.g. = 32
kernel_initializer	Kernel initialization／e.g. = 'he_normal'

Return Value	Description
-	Upsampling block

__build__()

Constructs the Generator model of SRGAN.

Argument	Description
filters	Number of channels in the convolutional layer／default = 32
num_residual_blocks	Number of residual blocks／default = 1
kernel_initializer	Kernel initialization／e.g. = 'he_normal'
momentum	BatchNormalization moving average momentum coefficient／default = 0.8

2.2.9. toolbox.model.srgan.Discriminator Class

This class inherits from keras.models.Model and constructs the Discriminator model of SRGAN.

Attribute	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2

__init__()

After initializing the keras.models.Model class, the __build__ method is invoked to construct the Discriminator model of SRGAN.

Argument	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2

__conv2d_block()

Constructs a convolutional block composed of Conv2D, BatchNormalization, and LeakyReLU layers.

Argument	Description
filters	Number of channels in the convolutional layer／e.g. = 4
kernel_size	Kernel size of the convolutional layer／default = 3
strides	Convolutional layer stride／default = 1
use_bn	Use BatchNormalization or not／default = True
kernel_initializer	Kernel initialization／e.g. = 'he_normal'
momentum	BatchNormalization moving average momentum coefficient／default = 0.8
alpha	Negative slope of LeakyReLU／default = 0.2

Return Value	Description
-	Convolutional block

__build__()

Constructs the Discriminator model of SRGAN.

Argument	Description
filters	Number of channels in the convolutional layer／default = 4
num_downsampling_blocks	Number of downsampling blocks／default = 1
kernel_initializer	Kernel initialization／e.g. = 'he_normal'
momentum	BatchNormalization moving average momentum coefficient／default = 0.8
alpha	Negative slope of LeakyReLU／default = 0.2

2.2.10. toolbox.model.srgan.SRGAN Class

This class inherits from keras.models.Model and constructs the SRGAN model.

Attribute	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2
label_noise	Noise ratio for real and fake labels／default = 1e-6
loss_weight	Loss weighting coefficients for the Generator and Discriminator／default = [1, 1e-7]
generator	toolbox.model.srgan.Generator Model
discriminator	toolbox.model.srgan.Discriminator Model

__init__()

After initializing the keras.models.Model class, the __build__ method is invoked to construct the SRGAN model.

Argument	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2
label_noise	Noise ratio for real and fake labels／default = 1e-6
loss_weight	Loss weighting coefficients for the Generator and Discriminator／default = [1, 1e-7]

__build__()

Constructs a GAN model by connecting the Generator and the Discriminator.

summary()

Overrides the summary method of keras.models.Model to display the layers of the Generator, Discriminator, and GAN models.

compile()

Overrides the compile method of keras.models.Model to compile the Generator, Discriminator, and GAN models. In the GAN model used for training the Generator, the weights of the Discriminator are frozen. The loss function of the Discriminator is set to binary cross-entropy.

Argument	Description
loss	Loss function for the Generator output images／e.g. = 'mse'
optimizer	Optimizer／e.g. = 'adam'
metrics	LIST OF EVALUATION METRICS (E.G., PSNR, DSSIM)／default = None

sample_labels()

Generates real and fake labels for the Discriminator and GAN models.

Argument	Description
batch_size	Vector length of real and fake labels (batch size)
noise	Noise ratio for real and fake labels／e.g. = 1e-6

Return Value	Description
list[0]	Real labels／e.g. shape = (32,)
list[1]	Fake labels／e.g. shape = (32,)

train_on_batch()

Overrides the train_on_batch method of keras.models.Model to perform batch-wise training of the Discriminator and the GAN model.

Argument	Description
lr_imgs	Low-resolution image batch／e.g. shape = (Batch size, 16, 16, 1)
hr_imgs	High-resolution image batch／e.g. shape = (Batch size, 64, 64, 1)
sample_weight	Optional parameters of the parent class／default = None
class_weight	Optional parameters of the parent class／default = None
reset_metrics	Optional parameters of the parent class／default = True

Return Value	Description
-	List of losses and evaluation metrics

test_on_batch()

Overrides the test_on_batch method of keras.models.Model to perform batch-wise testing of the GAN model.

Argument	Description
lr_imgs	Low-resolution image batch／e.g. shape = (Batch size, 16, 16, 1)
hr_imgs	High-resolution image batch／e.g. shape = (Batch size, 64, 64, 1)
sample_weight	Optional parameters of the parent class／default = None
reset_metrics	Optional parameters of the parent class／default = True

Return Value	Description
-	List of losses and evaluation metrics

predict()

Overrides the test_on_batch method of keras.models.Model to perform batch-wise testing of the GAN model.

Argument	Description
x	Low-resolution image batch／e.g. shape = (Batch size, 16, 16, 1)
batch_size	Batch size
verbose	Progress bar (0: disabled, 1: enabled)／default = 0
steps	Optional parameters of the parent class／default = None

Return Value	Description
-	Batch of super-resolved images／e.g. shape = (Batch size, 64, 64, 1)

2.2.11. toolbox.model.esrgan.Generator Class

This class inherits from keras.models.Model and constructs the Generator model of ESRGAN.

Attribute	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2

__init__()

After initializing the keras.models.Model class, the __build__ method is invoked to construct the Generator model of ESRGAN.

Argument	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2
filters	Number of channels in the convolutional layer
num_residual_blocks	Number of RRDB blocks

__DB()

Constructs the Dense Block (DB) layer of ESRGAN.

Argument	Description
filters	Number of channels in the convolutional layer／e.g. = 32
kernel_initializer	Kernel initialization／e.g. = 'he_normal'
alpha	Negative slope of LeakyReLU／e.g. = 0.2
kernel_size	Kernel size of the convolutional layer／default = 3
strides	Convolutional layer stride／default = 1

Return Value	Description
-	Dense Block (DB) layer

__RRDB()

Constructs the Residual-in-Residual Dense Block (RRDB) layer of ESRGAN.

Argument	Description
filters	Number of channels in the convolutional layer／e.g. = 32
kernel_initializer	Kernel initialization／e.g. = 'he_normal'
alpha	Negative slope of LeakyReLU／e.g. = 0.2
beta	Residual scaling coefficient／e.g. = 0.2

Return Value	Description
-	Residual-in-Residual Dense Block (RRDB) layer

__upsampling_block()

Constructs an upsampling block for image enlargement. Sub-Pixel Convolution is adopted for the upsampling process. However, as pointed out in distill.pub (https://distill.pub/2016/deconv-checkerboard/
), upsampling using nearest-neighbor or bilinear interpolation is effective in preventing checkerboard artifacts. Nevertheless, such approaches may exacerbate noise near image boundaries. Therefore, the internal function resize_conv is currently left unused.

Argument	Description
filters	Number of channels in the convolutional layer／e.g. = 32
kernel_initializer	Kernel initialization／e.g. = 'he_normal'

Return Value	Description
-	Upsampling block

__build__()

Constructs the Generator model of ESRGAN.

Argument	Description
filters	Number of channels in the convolutional layer／default = 32
num_residual_blocks	NUMBER OF RRDB BLOCKS／default = 2
kernel_initializer	Kernel initialization／e.g. = 'he_normal'
alpha	Negative slope of LeakyReLU／e.g. = 0.2
beta	Residual scaling coefficient／e.g. = 0.2

2.2.12. toolbox.model.esrgan.Discriminator Class

This class inherits from keras.models.Model and constructs the Discriminator model of ESRGAN. It differs from the SRGAN Discriminator in that dropout layers are added.

Attribute	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2

__init__()

After initializing the keras.models.Model class, the __build__ method is invoked to construct the Discriminator model of ESRGAN.

Argument	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2
d_filters	／e.g. d_filters = 4
d_num_downsampling_blocks	／e.g. d_num_downsampling_blocks = 2

__conv2d_block()

Constructs a convolutional block composed of Conv2D, BatchNormalization, and LeakyReLU layers.

Argument	Description
filters	Number of channels in the convolutional layer／e.g. = 4
kernel_size	Kernel size of the convolutional layer／default = 3
strides	Convolutional layer stride／default = 1
use_bn	Use BatchNormalization or not／default = True
kernel_initializer	Kernel initialization／e.g. = 'he_normal'
momentum	BatchNormalization moving average momentum coefficient
alpha	Negative slope of LeakyReLU／default = 0.2

Return Value	Description
-	Convolutional block

__build__()

Constructs the Discriminator model of ESRGAN.

Argument	Description
filters	Number of channels in the convolutional layer／default = 4
num_downsampling_blocks	Number of downsampling blocks／default = 2
kernel_initializer	Kernel initialization／e.g. = 'he_normal'
momentum	BatchNormalization moving average momentum coefficient
alpha	Negative slope of LeakyReLU
dropout_rate	Dropout rate／default = 0.4

2.2.13. toolbox.model.esrgan.RelativisticDiscriminator Class

This class inherits from keras.models.Model and constructs the Relativistic Discriminator model of ESRGAN.

__init__()

After initializing the keras.models.Model class, the __build__ method is invoked to construct the Discriminator model of ESRGAN.

Argument	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2
d_filters	／e.g. d_filters = 4
d_num_downsampling_blocks	／e.g. d_num_downsampling_blocks = 2

__ra_loss()

Computes the Relativistic Average Loss.

Argument	Description
x	Real and fake labels output by the Discriminator
noise	Noise coefficient for preventing divergence in the logarithm／default = 1e-6

Return Value	Description
-	Relativistic Average Loss

__build__()

Constructs the Relativistic Discriminator model of ESRGAN.

2.2.14. toolbox.model.esrgan.ESRGAN Class

This class inherits from keras.models.Model and constructs the ESRGAN model.

Attribute	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2
loss_weight	Loss weighting coefficients for the Generator and Discriminator／default = [1, 1e-7]
generator	toolbox.model.esrgan.Generator Model
discriminator	toolbox.model.esrgan.RelativisticDiscriminator Model

__init__()

After initializing the keras.models.Model class, the __build__ method is invoked to construct the ESRGAN model.

Argument	Description
img_shape	Shape of the low-resolution image／e.g. shape = (16, 16, 1)
scale	Super-Resolution scaling factor／e.g. = 2
loss_weight	Loss weighting coefficients for the Generator and Discriminator／default = [1, 1e-7]
params	parameter／default = None

__build__()

Constructs a GAN model by connecting the Generator and the Relativistic Discriminator.

summary()

Overrides the summary method of keras.models.Model to display the layers of the Generator, Relativistic Discriminator, and GAN models.

compile()

Overrides the compile method of keras.models.Model to compile the Generator, Relativistic Discriminator, and GAN models. In the GAN model used for training the Generator, the weights of the Relativistic Discriminator are frozen. In ESRGAN, both the Generator and Discriminator explicitly define loss functions computed from the inputs using the add_loss() method. Accordingly, in the train_on_batch() method, the output argument (outputs) is not used (i.e., outputs=None).

Argument	Description
loss	Loss function for the Generator output images／e.g. = 'mse'
optimizer	Optimizer／e.g. = 'adam'
metrics	LIST OF EVALUATION METRICS (E.G., PSNR, DSSIM)／default = None

train_on_batch()

Overrides the train_on_batch method of keras.models.Model to perform batch training for the Relativistic Discriminator and the GAN model.

Argument	Description
lr_imgs	Low-resolution image batch／e.g. shape = (Batch size, 16, 16, 1)
hr_imgs	High-resolution image batch／e.g. shape = (Batch size, 64, 64, 1)
sample_weight	Optional parameters of the parent class／default = None
class_weight	Optional parameters of the parent class／default = None
reset_metrics	Optional parameters of the parent class／default = True

Return Value	Description
-	List of losses and evaluation metrics

test_on_batch()

Overrides the test_on_batch method of keras.models.Model to perform batch testing of the GAN model.

Argument	Description
lr_imgs	Low-resolution image batch／e.g. shape = (Batch size, 16, 16, 1)
hr_imgs	High-resolution image batch／e.g. shape = (Batch size, 64, 64, 1)
sample_weight	Optional parameters of the parent class／default = None
reset_metrics	Optional parameters of the parent class／default = True

Return Value	Description
-	List of losses and evaluation metrics

predict()

Overrides the predict method of keras.models.Model to perform Super-Resolution processing using the Generator.

Argument	Description
x	Low-resolution image batch／e.g. shape = (Batch size, 16, 16, 1)
batch_size	Batch size
verbose	Progress bar (0: disabled, 1: enabled)／default = 0
steps	Optional parameters of the parent class／default = None

Return Value	Description
-	Batch of super-resolved images／e.g. shape = (Batch size, 64, 64, 1)

2.2.15. toolbox.metric.Metric Class

Defines evaluation metrics for image quality.

psnr()

Defines the evaluation function for Peak Signal-to-Noise Ratio (PSNR).

Argument	Description
y_true	Ground-truth data／e.g. shape = (Batch size, 64, 64, 1)
y_pred	Predicted data／e.g. shape = (Batch size, 64, 64, 1)

Return Value	Description
-	PSNR

dssim()

Defines the evaluation function for Structural Dissimilarity (DSSIM). DSSIM is derived from Structural Similarity (SSIM) as DSSIM = (1 – SSIM) / 2.

Argument	Description
y_true	Ground-truth data／e.g. shape = (Batch size, 64, 64, 1)
y_pred	Predicted data／e.g. shape = (Batch size, 64, 64, 1)

Return Value	Description
-	DSSIM

2.2.16. toolbox.loss.Loss Class

Defines the loss function to be minimized during training.

psnr()

Defines a loss function that minimizes the reciprocal of the Peak Signal-to-Noise Ratio (PSNR).

Argument	Description
y_true	Ground-truth data／e.g. shape = (Batch size, 64, 64, 1)
y_pred	Predicted data／e.g. shape = (Batch size, 64, 64, 1)

Return Value	Description
-	1 / PSNR

dssim()

Defines a loss function that minimizes Structural Dissimilarity (DSSIM).

Argument	Description
y_true	Ground-truth data／e.g. shape = (Batch size, 64, 64, 1)
y_pred	Predicted data／e.g. shape = (Batch size, 64, 64, 1)

Return Value	Description
-	DSSIM

vgg()

Defines a loss function that minimizes the content loss computed at an intermediate layer of the VGG19 model. The __preprocess function normalizes data for input to the VGG19 model. The input to VGG19 is an RGB image in the range 0–255. The penalty_l1 and penalty_l2 functions implement L1 and L2 regularization terms for predictions that have not been normalized to the range 0–1 (required when using a linear activation, rather than tanh which may cause vanishing gradients, as the activation function in the final layer of the Generator).

Argument	Description
y_true	Ground-truth data／e.g. shape = (Batch size, 64, 64, 1)
y_pred	Predicted data／e.g. shape = (Batch size, 64, 64, 1)
feature_index	Layer index used for evaluating the content loss／default = 20

Return Value	Description
-	Regularized content loss at the feature_index layer

2.2.17. toolbox.callbacks.ModelLogger Class

This class inherits from keras.callbacks.ModelCheckpoint and saves the model weights at each epoch to weights_*.h5. In addition, the weights from the epoch with the minimum val_loss are saved to weights_best.h5.

Attribute	Description
log_dir	Output directory name for the data／e.g. = 'log'
log_period	Output frequency of the data／e.g. = 10

__init__()

Initializes the attributes and the keras.callbacks.ModelCheckpoint class.

Argument	Description
log_dir	Output directory name for the data／e.g. = 'log'
log_period	Output frequency of the data／e.g. = 10

on_epoch_end()

Overrides the on_epoch_end method of keras.callbacks.ModelCheckpoint to save the model weights at the end of each epoch.

Argument	Description
epoch	Current epoch number
logs	Dictionary of current loss and evaluation metrics／e.g. = { 'loss':1e-3, 'psnr':51.1, ...}

2.2.18. toolbox.callbacks.HistoryLogger Class

This class inherits from keras.callbacks.CSVLogger and saves the training history plots for each epoch to history_*.png. In addition, the training history over all epochs is saved to history.csv.

Attribute	Description
log_dir	Output directory name for the data／e.g. = 'log'
log_period	Output frequency of the data／e.g. = 10
epochs	Epoch history
logs	Dictionary of loss and evaluation metric history／e.g. = { 'loss':[], 'psnr':[], ...}

__init__()

Initializes the attributes and the keras.callbacks.CSVLogger class.

Argument	Description
log_dir	Output directory name for the data／e.g. = 'log'
log_period	Output frequency of the data／e.g. = 10

on_epoch_end()

Overrides the on_epoch_end method of keras.callbacks.CSVLogger to append the training history to history.csv at the end of each epoch and to save the training history plot.

Argument	Description
epoch	Current epoch number
logs	Dictionary of current loss and evaluation metrics／e.g. = { 'loss':1e-3, 'psnr':51.1, ...}

2.2.19. toolbox.callbacks.TestLogger Class

This class inherits from keras.callbacks.Callback and performs performance testing of the Super-Resolution model at each epoch, saving the test results to test_*.png.

Attribute	Description
log_dir	Output directory name for the data／e.g. = 'log'
log_period	Output frequency of the data／e.g. = 10
data_generator	Validation data generator
denorm_func	Denormalization function for normalized data
num_samples	Number of test image samples／e.g. = 4
evaluator	toolbox.evaluator.Evaluator class object (for model performance evaluation)

__init__()

Initializes the attributes.

Argument	Description
log_dir	Output directory name for the data／e.g. = 'log'
log_period	Output frequency of the data／e.g. = 10
data_generator	Validation data generator
denorm_func	Denormalization function for normalized data
num_samples	Number of test image samples／e.g. = 4

on_epoch_end()

Overrides the on_epoch_end method of keras.callbacks.Callback to perform performance testing of the Super-Resolution model at the end of each epoch and to save the test results to test_*.png.

Argument	Description
epoch	Current epoch number
logs	Dictionary of current loss and evaluation metrics／e.g. = { 'loss':1e-3, 'psnr':51.1, ...}

2.2.20. toolbox.data_loader.DataLoader Class

Loads the training, validation, and test data from the datasets stored in the train, validation, and test directories.

Attribute	Description
dataset	Dictionary of training, validation, and test datasets／e.g. dataset['train']

__init__()

Loads the training, validation, and test data from the datasets stored in the train, validation, and test directories, and initializes the dataset attribute.

Argument	Description
data_dir	Output directory name for the data／e.g. = 'log'
sub_dirs	Names of the training, validation, and test data directories
sr_scale	Super-Resolution scale factor／default = None
normalize	Flag indicating whether to perform normalization／default = False

train()

Retrieves the training dataset.

Return Value	Description
tuple[0]	Low-resolution image set for training／e.g. shape = (Number of images, 16, 16, 1)
tuple[1]	High-resolution image set for training／e.g. shape = (Number of images, 64, 64, 1)

validation()

Retrieves the validation dataset.

Return Value	Description
tuple[0]	Low-resolution image set for validation／e.g. shape = (Number of images, 16, 16, 1)
tuple[1]	High-resolution image set for validation／e.g. shape = (Number of images, 64, 64, 1)

test()

Retrieves the test dataset.

Return Value	Description
tuple[0]	Low-resolution image set for test／e.g. shape = (Number of images, 16, 16, 1)

tuple[1]	High-resolution image set for test／e.g. shape = (Number of images, 64, 64, 1)

2.2.21. toolbox.data_generator.DataGenerator Class

Normalizes the low-resolution and high-resolution image datasets and defines a generator function.

Attribute	Description
lr_imgs	Low-resolution image set／e.g. shape = (Number of images, 16, 16, 1)
hr_imgs	High-resolution image set／e.g. shape = (Number of images, 64, 64, 1)
num_imgs	Number of images
batch_size	Batch size

__init__()

Removes data containing NaN values from the low-resolution and high-resolution image sets and initializes the attributes.

Argument	Description
x	Low-resolution image set／e.g. shape = (Number of images, 16, 16, 1)
y	High-resolution image set／e.g. shape = (Number of images, 64, 64, 1)
norm_func	Normalization function
batch_size	Batch size

sample()

Samples the specified number of low-resolution and high-resolution images.

Argument	Description
num_samples	Number of images to sample
replace	Flag indicating whether sampling with replacement is allowed／default = False
indices	Manually specified indices／default = []

Return Value	Description
tuple[0]	Low-resolution image set／e.g. shape = (num_samples, 16, 16, 1)
tuple[1]	High-resolution image set／e.g. shape = (num_samples, 64, 64, 1)

generator()

Samples a batch of low-resolution and high-resolution images.

Return Value	Description
tuple[0]	Low-resolution image set／e.g. shape = (batch_size, 16, 16, 1)
tuple[1]	High-resolution image set／e.g. shape = (batch_size, 64, 64, 1)

steps_per_epoch()

Retrieves the number of batch training iterations per epoch.

Return Value	Description
-	Number of batch training iterations per step (num_imgs // batch_size)

2.2.22. toolbox.normalizer.MinMaxNormalizer

Normalizes the data elements to the range 0–1.

Attribute	Description
xmin	Minimum value of data x
xmax	Maximum value of data x
ymin	Minimum value of data y
ymax	Maximum value of data y

__init__()

Initializes the min and max attributes.

Argument	Description
xMin	Minimum value of data x
xMax	Maximum value of data x
ymin	Minimum value of data y
ymax	Maximum value of data y

normalize()

Defines a data normalization function.

Argument	Description
imgs	Data before normalization
min	Minimum value of the data
max	Maximum value of the data

Return Value	Description
-	Normalized data

denormalize_x()

Defines the inverse transformation of the normalization function.

Argument	Description
imgs	Normalized data

Return Value	Description
-	Data before normalization

denormalize_y()

Defines the inverse transformation of the normalization function.

Argument	Description
imgs	Normalized data

Return Value	Description
-	Data before normalization

2.2.23. toolbox.evaluator.Evaluator Class

Generates bicubic-interpolated images and super-resolved images from low-resolution inputs, and visualizes them in comparison with the corresponding high-resolution images. Image quality is evaluated by computing PSNR and DSSIM with respect to the high-resolution images.

Attribute	Description
X	Placeholder for evaluation metric computation
Y	Placeholder for evaluation metric computation
metric_ops	Operators for computing PSNR and DSSIM

__init__()

Initializes the X, Y, and metric_ops attributes.

Argument	Description
img_shape	Image shape／e.g. shape = (64, 64, 1)

evaluate()

For the test data, visualizes the low-resolution images, bicubic-interpolated images, super-resolved images, and high-resolution images side by side, and displays PSNR and DSSIM values alongside them.

Argument	Description
x_test	Low-resolution image set／e.g. shape = (Number of images, 16, 16,

	1)
y_test	High-resolution image set／e.g. shape = (Number of images, 64, 64, 1)
batch_size	Batch size
denorm_func_x	Inverse transformation function for normalized data x
denorm_func_y	Inverse transformation function for normalized data y
save_fname	Output filename for saving the results (if None, the results are displayed using matplotlib.pyplot.show)

2.2.24. toolbox.plotter.Plotter Class

Visualizes the training history and test results using matplotlib.pyplot.

__setup_ax()

Configures the display settings of the matplotlib.axes.Axes object.

Argument	Description
ax	Matplotlib.axes.Axes object
xlabel	Label of the x-axis
ylabel	Label of the y-axis
xscale_log	Flag indicating whether to use logarithmic scaling on the x-axis／default = False
yscale_log	Flag indicating whether to use logarithmic scaling on the y-axis／default = False
fontsize	Font size for axis labels, etc.／default = 9
labelsize	Font size for tick labels, etc.／default = 8

__pair_plot()

Plots training history in pairs (e.g., loss and val_loss).

Argument	Description
ax	Matplotlib.axes.Axes object
x	Data for the x-axis
y	List of data for the y-axis
labels	List of labels corresponding to the y-axis data
color	Plot color

plot_history_graph()

Plots the history of loss, PSNR, and DSSIM. As shown in the figure below, the performance on the training data (loss, psnr, dssim) and the performance on the validation data (val_loss, val_psnr, val_dssim) are plotted in three separate graphs.

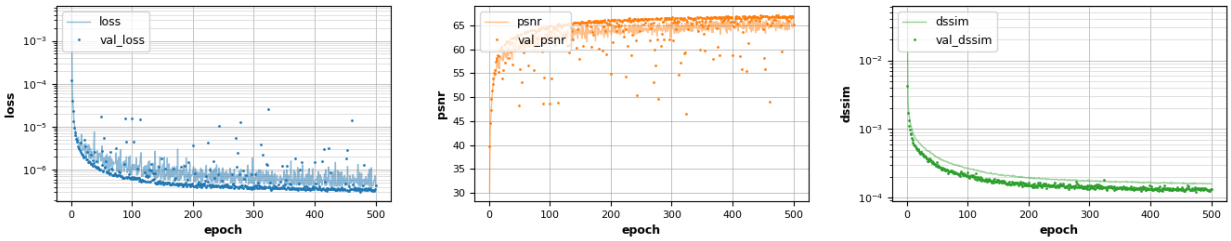

Figure 11 Example of graph plotting

However, in the SRGAN model, instead of plotting loss and val_loss, the Generator losses (g_loss, val_g_loss) and the Discriminator losses (d_loss, val_d_loss) are plotted, as shown in the figure below.

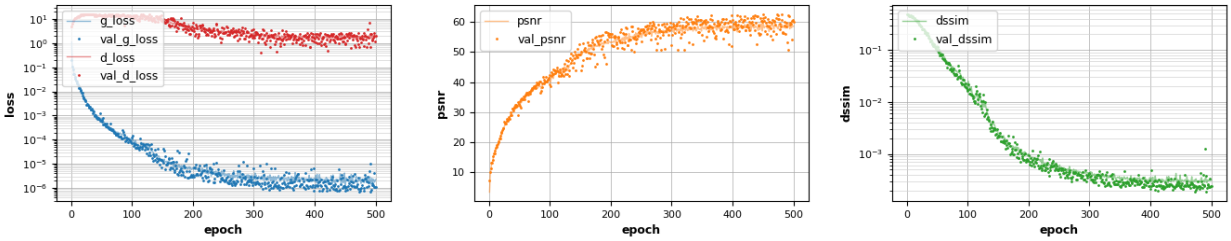

Figure 12 Example of graph plotting (SRGAN)

Argument	Description
epochs	Number of epochs (x-axis)
logs	Dictionary of training history over all epochs／e.g. { 'loss'=[1e-2, 1e-3, ...], ... }
save_fname	Output filename for saving the results (if None, the results are displayed using matplotlib.pyplot.show)

plot_test_imgs()

Visualizes the test results. As shown in the figure below, the toolbox.evaluator.Evaluator class visualizes the low-resolution images, bicubic-interpolated images, super-resolved images, and high-resolution images, and displays PSNR and DSSIM values alongside them.

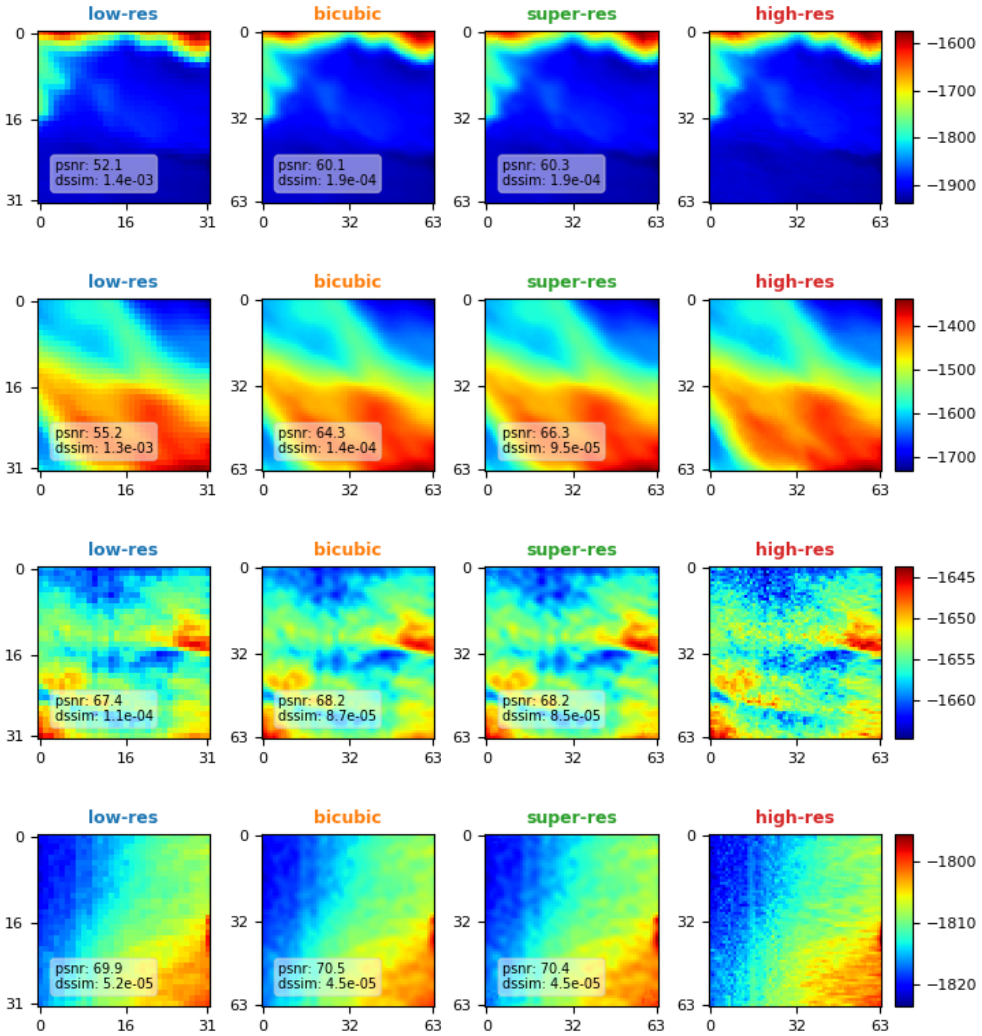

Figure 13 Visualization of test results

Argument	Description
imgs	Images to visualize / e.g. shape = (Group, Number of images, 64, 64, 1)
labels	Group names／e.g. = ['low-res', 'bicubic', 'super-res', 'high-res']
save_fname	Output filename for saving the results (if None, the results are displayed using matplotlib.pyplot.show)
figsize	Figure size (x 100px) ／default = (8, 8)
fontsize	Font size for titles, etc.／default = 9

labelsize	Font size for tick labels, etc.／default = 8
textsize	Text size for PSNR, DSSIM, etc.／default = 7

2.2.25. arg_parser.ArgParser Class

The ArgParser class handles command-line argument parsing for the program.

The parser reads:

- yml_fname – Input YAML filename containing user parameters
- mode – Execution mode (train or test)

Attribute	Description
Args	Parsed command-line arguments

__init__()

Initializes the argument parser and stores the parsed arguments.

get_fname()

Returns the input YAML filename.

Return Value	Description
-	Filename

get_mode()

Returns the execution mode.

Return Value	Description
-	Execution mode ('train' or 'test')

2.2.26. Initializer.Initializer Class

The Initializer class performs system-level initialization for TensorFlow.

tf_init()

Configures TensorFlow settings including logging settings, memory configuration and initializes the random seed.

Argument	Description
log_level	Logging level for TensorFlow warning messages／default = '3'
seed	Random seed value. If set to None, the random seed is not fixed.

2.2.27. user_params.UserParams Class

The UserParams class loads user-defined parameters from a YAML file and exposes them as attributes.

Attribute	Description
input_shape	Input LR image shape／e.g. = (16, 16, 1)
scale	Super-resolution scale factor／e.g. = 2
data_dir	Directory that contains the train, validation, and test subdirectories.
norm_range	Value range used for normalization.／e.g. = (-2500, 0)
fname_model_weights	Filename of pre-trained model weights for test run.
batch_size	Batch size.
epochs	Number of training epochs.
model	Super-resolution model type （srcnn, fsrcnn, espcn, srgan, esrgan）.
loss	Loss function（mae, mse, psnr, dssim, vgg）
optimizer	Optimization method（sgd, rmsprop, adam, etc.）
learning_rate	Learning rate／recommended = 1e-4
log_period	Frequency for writing log outputs.
log_dir	Directory for saving log files (if empty, yaml filename is used).
patience	Early stopping parameter. (null disables early stopping and the program runs normally) ／e.g. = 500
srcnn	srcnn settings
filters	Filter sizes.
kernels	Kernel sizes.
espcn	espcn settings
filters	Filter sizes.
kernels	Kernel sizes.
esrgan	esrgan settings
filters	Filter sizes for the Generator
num_residual_blocks	Number of residual blocks in the Generator.
d_filters	Filter sizes for the Discriminator.
d_num_downsampling_blocks	Number of convolution blocks in the Discriminator.
loss_weights	Loss-weighting ratio for Generator and Discriminator.

__init__()

Loads user parameters from a YAML file and initializes them as attributes. Parameters are assigned to __dict__, allowing direct attribute-style access throughout the program.

Argument	Description
fname	Filename of the YAML configuration file.

2.2.28. modeler.Modeler Class

The Modeler class constructs the super-resolution model based on the user parameters provided through the UserParams class.

Attribute	Description
u_params	user_params.UserParams object.
models	A dictionary mapping model names to super-resolution model classes { 'srcnn':SRCNN, 'fsrcnn':FSRCNN, ... }
custom_layers	A dictionary of custom layer classes from toolbox.model.layer.
loss_funcs	A dictionary mapping loss function names to loss implementations { 'psnr':Loss.psnr, 'dssim':Loss.dssim, ... }
optimizers	A dictionary mapping optimizer names to optimizer implementations { 'sgd':keras.optimizers.SGD, ... }

__init__()

Initializes all attributes.

Argument	Description
u_params	user_params.UserParams object.

build()

Builds and compiles the super-resolution model.

Argument	Description
verbose	If set to 1, the model structure (layers) is displayed; if 0, the structure is hidden.／default = 1

Return Value	Description
-	The compiled super-resolution model.

2.2.29. trainer.Trainer Class

The Trainer class performs super-resolution model training based on the user parameters defined in the UserParams class.

Attribute	Description
u_params	user_params.UserParams object.
normalizer	Normalization handler
train_data_generator	toolbox.data_generator.DataGenerator object.
validation_data_generator	toolbox.data_generator.DataGenerator object.

__init__()

Initializes all attributes.

Argument	Description
u_params	user_params.UserParams object.

reset_weights()

Initializes the weights (kernel and bias parameters) of the model .

Argument	Description
model	The super-resolution model whose weights will be reset.

train()

Executes mtraining of the super-resolution model.

Argument	Description
model	The super-resolution model to be trained.

2.2.30. tester.Tester Class

The Tester class evaluates the performance of a trained super-resolution model based on the user parameters provided through the UserParams class.

Attribute	Description
u_params	user_params.UserParams object.
normalizer	Normalization handler
test_data_generator	toolbox.data_generator.DataGenerator object.
evaluator	toolbox.evaluator.Evaluator object.

__init__()

Initializes all attributes.

Argument	Description
u_params	user_params.UserParams object.

test()

Executes the performance evaluation of the super-resolution model.

Argument	Description
model	The trained super-resolution model to be evaluated.
num_samples	Number of test images／default = 4

2.2.31. Main Class

The Main class executes the complete workflow for super-resolution training and testing.

run()

Executes the super-resolution learning and testing processes in the following order:

(1) Reads command-line arguments using the ArgParser class.

(2) Initializes user parameters via the UserParams class.

(3) Initializes TensorFlow using the Initializer class.

(4) Builds the super-resolution model.

(5) Executes training or testing.

2.3. Recommended Environment

The program runs on Python 3.8 and requires the following packages:

Package	Version (recommended)
argparse	*
matplotlib	3.3.4
numpy	1.19.5
opencv	4.5.2.54
optuna	2.8.0
pandas	1.1.5
pathlib	*
pickle	*
pyyaml	5.4.1
tensorflow	2.5.0

The detailed package requirements are listed in the requirements.txt file included in the program directory.

2.4. Usage

2.4.1. User Parameter Settings

User parameters are defined in a YAML file.

```yaml
# input image shape. (32, 32, 1) is recommended.
input_shape: !!python/tuple [16, 16, 1]
# Super-Resolution scale. 2x is recommended.
scale: 4
# dataset directory including train, validation and test directories
data_dir: data/output_origin
# data range for normalization
norm_range: !!python/tuple [-2500, 0]
# model type (srcnn, fsrcnn, espcn, srgan, esrgan)
model: esrgan
# filename of trained weights (None for new model)
fname_model_weights: None
# tester.py read weights
read_test_weights: weights_best.h5
# training batch size
batch_size: 16
# number of training epochs
epochs: 5000
# loss function (mae, mse, psnr, ssim, vgg)
loss: mse
# optimizer (sgd, rmsprop, adagrad, adadelta, adam, adamax, nadam)
optimizer: adam
# learning rate
learning_rate: 0.0001
# training log directory. if null, yml_filename_log is used
log_dir:
# logging perieod
log_period: 20
# early stopping(default:null) Number of epochs with no improvement after which tr
aining will be stopped.
patience: 500
online_da:
  valid: false
  zoom_range: !!python/tuple [0.5,1.0]
```

```
  zoom_iso: true
  depth_scale_range: !!python/tuple [0.5,1.5]
  use_depth_scale: true
  horizontal_flip: false
  vertical_flip: false
  rotation_range: 0
  steps_per_epoch_mag: 1

srcnn:
  filters: !!python/tuple [64,32]
  kernels: !!python/tuple [9,1,5]

espcn:
  filters: !!python/tuple [64,32]
  kernels: !!python/tuple [5,3,3]

esrgan:
  filters: 32
  num_residual_blocks: 2
  d_filters: 4
  d_num_downsampling_blocks: 1
  loss_weights: [1, 1e-7]
```

The table below summarizes the main parameters.

User Params	Description
input_shape	Shape of the input LR image.／e.g. = (16, 16, 1)
scale	Super-resolution scaling factor.／e.g. = 2
data_dir	Directory containing the train, validation, and test subdirectories.
norm_range	Value range used for normalization.／e.g. = (-2500, 0)
fname_model_weights	Filename of pre-trained model weights for test run.
batch_size	Batch size.
epochs	Number of training epochs.
model	Super-resolution model type （srcnn, fsrcnn, espcn, srgan, esrgan）.
loss	Loss function（mae, mse, psnr, dssim, vgg）
optimizer	Optimization method（sgd, rmsprop, adam, etc.）
learning_rate	Learning rate／recommended = 1e-4
log_period	Frequency for writing log outputs.
log_dir	Directory for saving log files (if empty, yaml filename is used).
patience	Early stopping parameter. (null disables early stopping and the program runs normally) ／e.g. = 500
online_da	online data augmentation settings
valid	Enables/disables online data augmentation. (true or false)
zoom_range	Range of horizontal scaling factors [min, max].
zoom_iso	Whether scaling is applied isotropically. (true or false)
depth_scale_range	Range of vertical scaling factors [min, max].
use_depth_scale	Enables/disables vertical scaling.(true or false)
horizontal_flip	Enables/disables horizontal flipping (unused in this version).
vertical_flip	Enables/disables vertical flipping (unused in this version).
rotation_range	Random rotation range (0 = no rotation; unused in this version).
steps_per_epoch_mag	Multiplier for steps per epoch (baseline = number of images ÷ batch size).
srcnn	srcnn settings

```
  filters                   Filter sizes.
  kernels                   Kernel sizes.
espcn                       espcn settings
  filters                   Filter sizes.
  kernels                   Kernel sizes.
esrgan                      esrgan settings
  filters                   Filter sizes for the Generator
  num_residual_blocks       Number of residual blocks in the Generator.
  d_filters                 Filter sizes for the Discriminator.
  d_num_downsampling_blocks Number of convolution blocks in the Discriminator.
  loss_weights              Loss-weighting ratio for Generator and Discriminator.
```

2.4.2. Command-Line Execution

main.py is the entry-point script.

It takes two arguments: the YAML filename and the execution mode (train or test).

```
$ python main.py -h

usage: main.py [-h] yml_fname {train,test}

positional arguments:
  yml_fname     filename including input parameters (.yml)
  {train,test}  execution mode: train or test

optional arguments:
  -h, --help  show this help message and exit
```

2.4.3. Model Training

Set fname_model_weights: None in the YAML file, then run:

```
$ python main.py input.yml train
```

During execution, the model structure and training progress are displayed.

```
Using TensorFlow backend.
building model... done
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 32, 32, 1)         0
_________________________________________________________________
resize_1 (Resize)            (None, 64, 64, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 64, 64)        5248
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 32)        2080
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 1)         801
=================================================================
Total params: 8,129
Trainable params: 8,129
Non-trainable params: 0
_________________________________________________________________
Epoch 1/2000
345/345 [==============================] - 5s 13ms/step - loss: 0.0042 - psnr: 29.8367
- dssim: 0.0476 - val_loss: 1.3553e-04 - val_psnr: 39.3511 - val_dssim: 0.0056
Epoch 2/2000
345/345 [==============================] - 3s 9ms/step - loss: 8.2348e-05 - psnr: 41.7771
- dssim: 0.0038 - val_loss: 4.7004e-05 - val_psnr: 44.0090 - val_dssim: 0.0025
Epoch 3/2000
345/345 [==============================] - 3s 9ms/step - loss: 4.0654e-05 - psnr: 44.8862
- dssim: 0.0030 - val_loss: 2.8650e-05 - val_psnr: 46.4466 - val_dssim: 0.0020
...
```

During training, the following files are automatically saved to the directory specified by log_dir (* denotes epoch number):

- history.csv: Contains the complete training and validation history for all epochs, including loss, PSNR, DSSIM, and other metrics.
- history_*.png: Plots of training and validation metrics generated at the logging interval specified by log_period.
- weights_best.h5: Model weights for the epoch in which the validation loss reached its minimum.
- weights_*.h5: Model weights saved at each logging interval during training.
- test_*.png: Visualization of super-resolution results for selected samples,

generated at each logging interval.

2.4.4. Model Testing

Specify the path to the trained model in fname_model_weights in the input file (input.yml) (e.g., fname_model_weights: log/weights_best.h5), and execute the training with the following command:

```
$ python main.py input.yml test
```

For testing, it is assumed that the topographic-feature CSV (generated by bathymetric_map_feature.py) already exists for the high-resolution data specified in the YAML parameter data_dir.

Upon execution, a new directory named test_result is created inside the log directory, and the following items are output:

- test_results.csv: Evaluation results for all test cases.
- test_results_describe.csv: Statistical summary of the evaluation results (equivalent to the output of pandas.DataFrame.describe()).
- test_resultsslope0-0.05_describe.csv:Summary of evaluation results for test cases whose slope values fall within 0–0.05.
- test_resultsslope0.05-0.1_describe.csv: Summary of evaluation results for test cases whose slope values fall within 0.05–0.1.
- test_resultsslope0.1-_describe.csv: Summary of evaluation results for test cases whose slope values 0.1 and above.
- data_SR.pkl: Predicted results for all test data.
- test_result_N-M.png: Visualization output of the predictions for test data N to M
- (one figure is generated for every four test images).

An example of test_result_N–M.png is shown below.

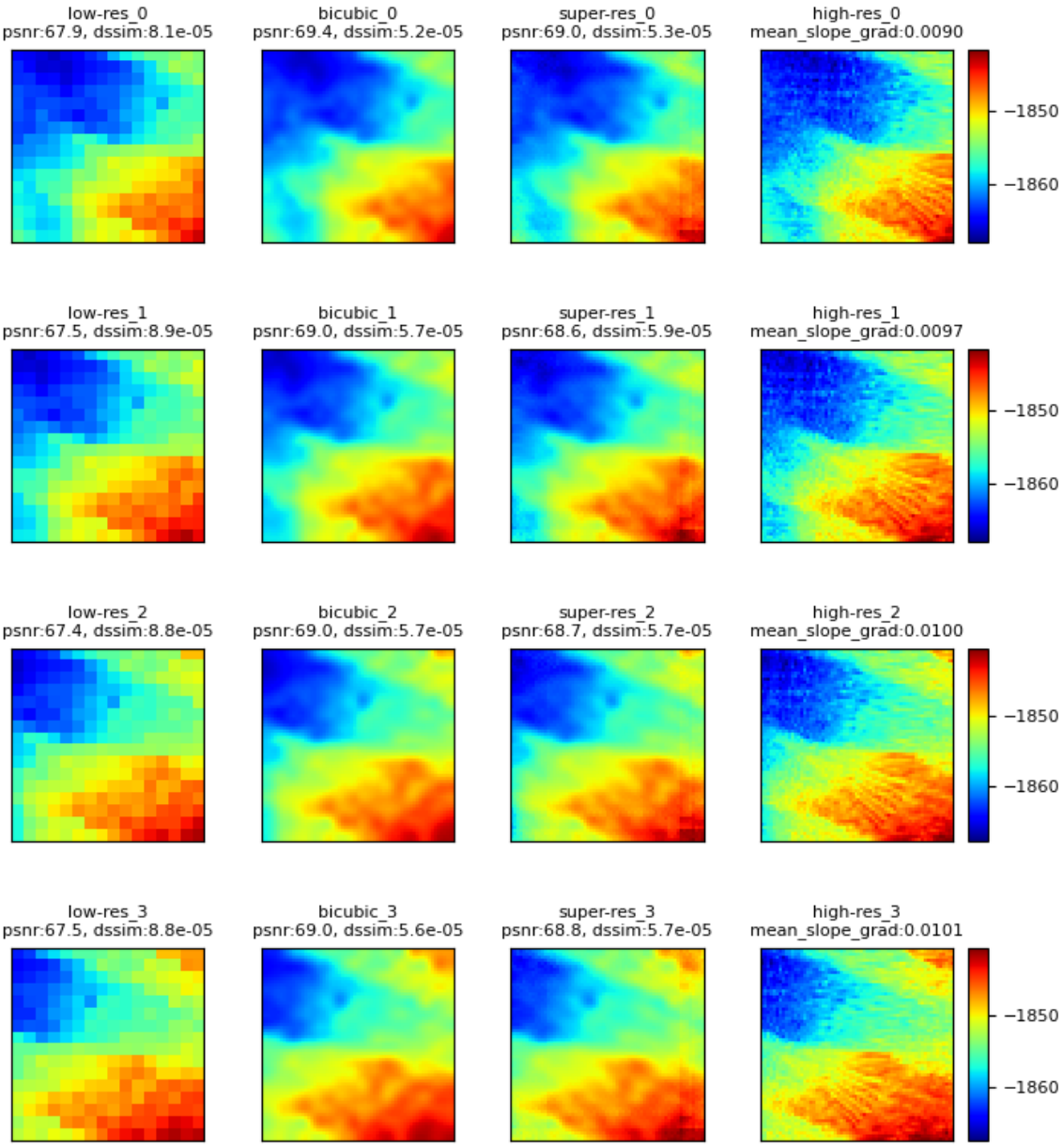

Figure 14 Example of `test_result_N-M.png`

3. References

[1] K. Murakami, D. Matsuoka, N. Takatsuki, M. Hidaka, J. Kaneko, Y. Kido , E. Kikawa, “Adaptive Super-Resolution for Ocean Bathymetric Maps Using a Deep Neural Network and Data Augmentation,” Earth and Space Science, 2025.

[2] C. Dong, C. Loy , X. Tang, “Accelerating the Super-Resolution Convolutional Neural Network,” in Proceedings of European Conference on Computer Vision (ECCV), 2016.

[3] C. Dong, C. Loy, X. Tang , H. Kaiming, “Image Super-Resolution Using Deep Convolutional Networks,” arXiv: 1501.00092 [cs.CV], 2014.

[4] L. Christian, T. Lucas, H. Ferenc, C. Jose, C. Andrew, A. Alejandro, A. Andrew, T. Alykhan, T. Johannes, W. Zehan , S. Wenzhe, “ Photo-realistic single image Super-Resolution using a generative adversarial network,” Proceedings of the IEEE conference on computer vision and pattern recognition, 2017.

[5] S. Wenzhe, C. Jose, H. Ferenc, T. Johannes, A. P. Andrew, B. Rob, R. Daniel , W. Zehan, “Real-time single image and video Super-Resolution using an efficient sub-pixel convolutional neural network,” Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[6] D. Vincent , V. Francesco, “ A guide to convolution arithmetic for deep learning, ” arXiv:1603.07285, 2016.

[7] X. Wang, Y. Ke, W. Shixiang, G. Jinjin, L. Yihao, D. Chao, L. C. Chen, Q. Yu , T. Xiaoou, “ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks,” arXiv: 1809.00219, 2018.