深層ニューラルネットワークとデータ拡張を用いた
海底地形図の適応的超解像

プログラム説明書

2026.02

国立研究開発法人海洋研究開発機構
JAMSTEC
JAPAN AGENCY FOR MARINE-EARTH SCIENCE AND TECHNOLOGY

ASTOM R&D

目次

1. sr_dataset_builder 1

1.1. プログラム概要 1

1.1.1. Adaptive Mixup: Adaptive Data Augmentation for Bathymetric Super-Resolution 2

1.2. 各種機能 3

1.2.1. dataset.Dataset クラス 3

__init__() 3

cleansing() 3

block_split() 4

rm_nan() 4

train_validation_test_split() 5

save() 5

1.3. ユーティリティプログラム 5

resize.py 5

flip.py 5

rotate.py 6

scale.py 7

bathymetric_map_feature.py 7

test_da.py 8

bathymetric_map_feature_LR.py 9

mixup.py 10

load_csv_norm() 11

load_imgs() 12

mixup_norm() 12

get_slope() 13

get_slope_LR() 13

pick_images() 13

idx_of_the_nearest() 14

indices_of_the_nearest() 14

1.4. 推奨動作環境 16

1.5. 使用方法 16

2. sr_trainer 17

2.1. プログラムの概要 17

2.1.1. SRCNN: Super-Resolution Convolutional Neural Network 17
2.1.2. FSRCNN: Fast Super-Resolution Convolutional Neural Network 18
2.1.3. ESPCN: Efficient Sub-Pixel Convolutional Neural Network 18
2.1.4. SRGAN: Super-Resolution using Generative Adversarial Network 19
2.1.5. ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks 20

2.2. 各種機能 22

2.2.1. クラス図 22
2.2.2. toolbox.model.layer.Resize クラス 23
__init__() 23
resized_shape() 23
call() 23
compute_output_shape() 24
get_config() 24
2.2.3. toolbox.model.layer.Conv2DSubPixel クラス 24
__init__() 24
call() 25
compute_output_shape() 25
get_config() 25
2.2.4. toolbox.model.layer.Scale クラス 26
__init__() 26
call() 26
compute_output_shape() 26
get_config() 26
2.2.5. toolbox.model.srcnn.SRCNN クラス 27
__init__() 27
__build__() 27
2.2.6. toolbox.model.fsrcnn.FSRCNN クラス 28
__init__() 28
__build__() 28
2.2.7. toolbox.model.espcn.ESPCN クラス 29
__init__() 29
__build__() 29
2.2.8. toolbox.model.srgan.Generator クラス 30
__init__() 30

__residual_block() ... 30
__upsampling_block() ... 31
__build__() ... 31
2.2.9. toolbox.model.srgan.Discriminator クラス ... 32
__init__() ... 32
__conv2d_block() ... 32
__build__() ... 33
2.2.10. toolbox.model.srgan.SRGAN クラス ... 33
__init__() ... 33
__build__() ... 33
summary() ... 34
compile() ... 34
sample_labels() ... 34
train_on_batch() ... 34
test_on_batch() ... 35
predict() ... 35
2.2.11. toolbox.model.esrgan.Generator クラス ... 36
__init__() ... 36
__DB() ... 36
__RRDB() ... 37
__upsampling_block() ... 37
__build__() ... 37
2.2.12. toolbox.model.esrgan.Discriminator クラス ... 38
__init__() ... 38
__conv2d_block() ... 38
__build__() ... 39
2.2.13. toolbox.model.esrgan.RelativisticDiscriminator クラス ... 39
__init__() ... 39
__ra_loss() ... 39
__build__() ... 39
2.2.14. toolbox.model.esrgan.ESRGAN クラス ... 40
__init__() ... 40
__build__() ... 40
summary() ... 40
compile() ... 41
train_on_batch() ... 41
test_on_batch() ... 41
predict() ... 42

2.2.15. toolbox.metric.Metric クラス 43
psnr() *43*
dssim() *43*
2.2.16. toolbox.loss.Loss クラス 44
psnr() *44*
dssim() *44*
vgg() *45*
2.2.17. toolbox.callbacks.ModelLogger クラス 46
__init__() *46*
on_epoch_end() *46*
2.2.18. toolbox.callbacks.HistoryLogger クラス 47
__init__() *47*
on_epoch_end() *47*
2.2.19. toolbox.callbacks.TestLogger クラス 48
__init__() *48*
on_epoch_end() *48*
2.2.20. toolbox.data_loader.DataLoader クラス 49
__init__() *49*
train() *49*
validation() *49*
test() *49*
2.2.21. toolbox.data_generator.DataGenerator クラス 51
__init__() *51*
sample() *51*
generator() *51*
steps_per_epoch() *52*
2.2.22. toolbox.normalizer.MinMaxNormalizer 52
__init__() *52*
normalize() *52*
denormalize_x() *52*
denormalize_y() *53*
2.2.23. toolbox.evaluator.Evaluator クラス 53
__init__() *53*
evaluate() *53*
2.2.24. toolbox.plotter.Plotter クラス 55
__setup_ax() *55*
__pair_plot() *55*
plot_history_graph() *56*

plot_test_imgs() 57
2.2.25. arg_parser.ArgParser クラス 59
__init__() 59
get_fname() 59
get_mode() 59
2.2.26. Initializer.Initializer クラス 59
tf_init() 59
2.2.27. user_params.UserParams クラス 60
__init__() 60
2.2.28. modeler.Modeler クラス 62
__init__() 62
build() 62
2.2.29. trainer.Trainer クラス 63
__init__() 63
reset_weights() 63
train() 63
2.2.30. tester.Tester クラス 64
__init__() 64
test() 64
2.2.31. Main クラス 64
run() 64
2.3. 推奨動作環境 **65**
2.4. 使用方法 **66**
2.4.1. ユーザパラメータの設定 66
2.4.2. コマンドライン引数 69
2.4.3. モデルの学習 70
2.4.4. モデルのテスト 71
3. 引用文献 **73**

1. sr_dataset_builder

1.1. プログラム概要

sr_dataset_builder は、ブロック分割によるランダムな画像データセットを作成するプログラムである。画像データセットの作成までの流れを以下に示す。

1. 低解像度画像の取得（解像度 16 x 16）
   1-1. 欠損画素を周辺 5 x 5 ピクセル中の非欠損画素の平均値で修復
   1-2. 入力画像を 5km 間隔で 44 x 40 ブロックに分割（下図参考）
   1-3. 各ブロックを 2 画素の間隔でオフセットしながら、ブロック毎に 25 枚の低解像度画像を取得
2. 高解像度画像の取得（解像度 64 x 64）
   2-1. 欠損画素を周辺 5 x 5 ピクセル中の非欠損画素の平均値で修復
   2-2. 入力画像を 5km 間隔で 44 x 40 ブロックに分割（下図参考）
   2-3. 各ブロックを 8 画素の間隔でオフセットしながら、ブロック毎に 25 枚の高解像度画像を取得
3. 低解像度・高解像度画像ペアのいずれかに欠損画素がある場合、データセットから除外
4. 全体の 80%を教師データ、10%を検証データ、10%をテストデータに分割
5. データ拡張を行う場合は、教師データに対して任意のデータ拡張を行う。(回転、反転、Adaptive Mixup)

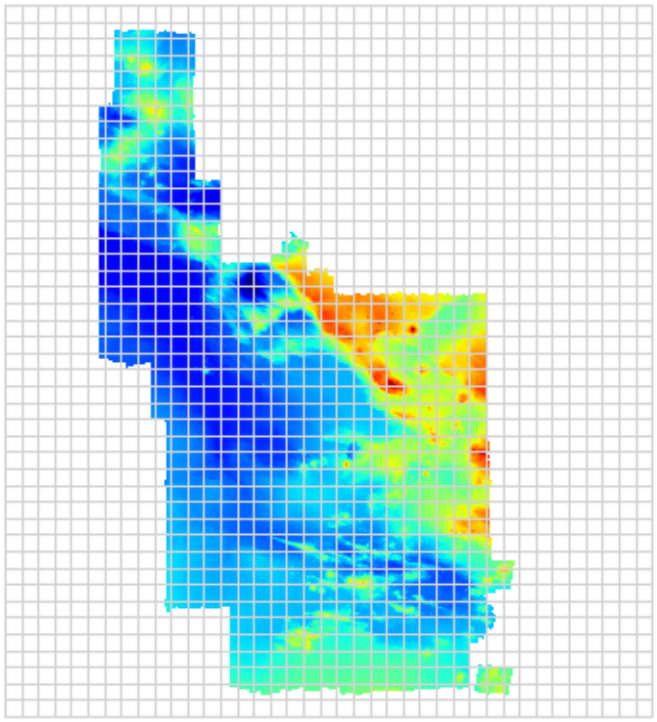

図 1 5km 間隔で 44 x 40 ブロックに分割するイメージ図

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

Adaptive Mixup [1] は、Mixup を用いた訓練データに対するデータ拡張とテストデータの特徴に合わせたデータサンプリングからなる適応的なデータ拡張手法である。Mixup は、異なる 2 枚の画像を線形結合することで新たな学習データを生成するデータ拡張手法である。
従来の反転や回転によるデータ拡張は、画像の向きを変えるのみであり、傾斜度などの地形的特徴量そのものは変化しない。そのため、訓練データと適用対象データの特徴分布が異なる場合、超解像性能が低下するという課題があった。
Adaptive Mixup では、まず低解像度・高解像度画像ペアに対して反転および回転による拡張を行い、その後、地形特徴量（論文中では平均傾斜度：Mean Slope Gradient, MSG）に基づいてデータを 2 つのグループに分割する。
それぞれのグループから画像ペアを抽出し、以下の式により線形結合を行うことで、新たな低解像度・高解像度画像ペアを生成する。

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

ここで、x_i, x_jは低解像度画像、y_i, y_jは対応する高解像度画像を表し、$\lambda \in [0,1]$は混合比率を示す。
この操作により、元データには存在しなかった中間的な地形特徴を持つ学習データを人工的に生成することが可能となる。
さらに、生成された mixup データは、適用対象データの特徴量分布に近づくようにサンプリングされる。これにより、訓練データと対象データ間の特徴量分布の不一致を緩和し、特定の地形条件に対する超解像精度を向上させることができる。
本手法は、反転・回転のみを用いた従来のデータ拡張と比較して、急峻な地形を含む領域において RMSE を最大 14.3%改善しつつ、空間構造の一貫性を維持できることが報告されている。

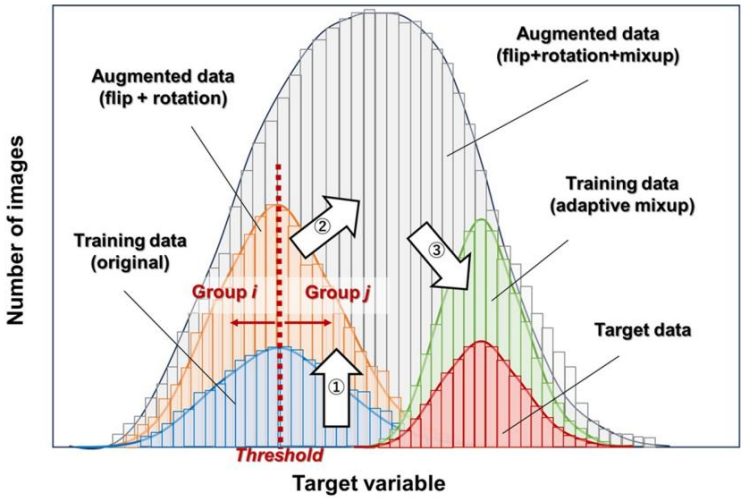

図 **2. Target variable** に基づく適応的データ拡張の概念図 **[1]**より引用

1.2. 各種機能

1.2.1. dataset.Dataset クラス

ブロック分割によるランダムな画像データセットを作成する。

Attribute	Description
x_train	低解像度教師データ／e.g. shape = (画像数, 16, 16)
y_train	高解像度教師データ／e.g. shape = (画像数, 64, 64)
x_validation	低解像度検証データ／e.g. shape = (画像数, 16, 16)
y_validation	高解像度検証データ／e.g. shape = (画像数, 64, 64)
x_test	低解像度テストデータ／e.g. shape = (画像数, 16, 16)
y_test	高解像度テストデータ／e.g. shape = (画像数, 64, 64)

__init__()

データセットを初期化・構築する。

Argument	Description
lr_fname	低解像度 pkl ファイル名／e.g. shape = (1100, 1000)
hr_fname	高解像度 pkl ファイル名／e.g. shape = (4400, 4000)
block_shape	ブロック分割数／e.g. = (44, 40)

cleansing()

欠損画素を修復する。

Argument	Description
img	入力画像／e.g. shape = (1100, 1000) or (4400, 4000)
kernel_size	非欠損画素の平均値で修復する窓サイズ／e.g. = (5, 5)

Return Value	Description
cimg	欠損画素を修復した画像／e.g. shape = (1100, 1000) or (4400, 4000)

block_split()

画像をブロック分割する。

Argument	Description
base_img	全体入力画像／e.g. shape = (1100, 1000) or (4400, 4000)
block_shape	ブロック分割数／e.g. = (44, 40)
img_shape	抽出画像サイズ／e.g. shape = (16, 16) or (64, 64)
stride	オフセット間隔／e.g. = (2, 2) or (8, 8)

Return Value	Description
-	各ブロックの全画像／e.g. shape = (ブロック数, 画像数, 幅, 高さ)

rm_nan()

低解像度・高解像度画像ペアのいずれかに欠損画素がある場合、画像を除外する。

Argument	Description
lr_blocks	低解像度画像群／e.g. shape = (ブロック数, 画像数, 幅, 高さ)
hr_blocks	高解像度画像群／e.g. shape = (ブロック数, 画像数, 幅, 高さ)

Return Value	Description
list[0]	低解像度画像群／e.g. shape = (ブロック数, 画像数, 幅, 高さ)
list[1]	高解像度画像群／e.g. shape = (ブロック数, 画像数, 幅, 高さ)

train_validation_test_split()

データを教師・検証・テストデータに分割する。

Argument	Description
x	低解像度画像群／e.g. shape = (ブロック数, 画像数, 幅, 高さ)
y	高解像度画像群／e.g. shape = (ブロック数, 画像数, 幅, 高さ)
validation_size	検証データの割合（0〜1）／e.g. = 0.1
test_size	テストデータの割合（0〜1）／e.g. = 0.1

Return Value	Description
list[0]	教師・低解像度画像群／e.g. shape = (ブロック数, 画像数, 幅, 高さ)
list[1]	検証・低解像度画像群／e.g. shape = (ブロック数, 画像数, 幅, 高さ)
list[2]	テスト・低解像度画像群／e.g. shape = (ブロック数, 画像数, 幅, 高さ)
list[3]	教師・高解像度画像群／e.g. shape = (ブロック数, 画像数, 幅, 高さ)
list[4]	検証・高解像度画像群／e.g. shape = (ブロック数, 画像数, 幅, 高さ)
list[5]	テスト・高解像度画像群／e.g. shape = (ブロック数, 画像数, 幅, 高さ)

save()

教師・検証・テスト画像を保存する。save_dir 内に train, validation, test ディレクトリが生成され、それぞれの中に低解像度画像データ data_LR.pkl と高解像度画像データ data_HR.pkl が作られる。画像データの shape は、(画像数, 幅, 高さ, チャネル数)となる。

Argument	Description
save_dir	データ保存先のディレクトリ名前

1.3. ユーティリティプログラム

データ拡張やデータ特徴量計算のための個別スクリプトについて以下に説明する。

resize.py

画像のリサイズを行う。オリジナルの pickle ファイルを読み込み、1/4 の低解像度に変換を行った画像を生成して保存する。

flip.py

反転画像の生成を行う。オフラインデータ拡張に用いる。save_dir 内の train ディレクトリに低解像度画像データ、高解像度画像データのそれぞれ yaml で指定された名前の pickle ファイルが作られる。画像データの shape は、(画像数, 幅, 高さ, チャネル数)となる。
反転画像生成スクリプトである flip.py は、入力設定ファイル名（.yml）をコマンドライン引数として

受け取る。

flip.py の入力 yaml ファイルは以下の構成となっている。

```
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	入力学習データのディレクトリパス
train_data	入力学習データのファイル名
train_savedata	出力される拡張データのファイル名　train_dir にここで指定された名前で保存される。

rotate.py

回転画像の生成を行う。オフラインデータ拡張に用いる。save_dir 内の train ディレクトリに低解像度画像データ、高解像度画像データのそれぞれ yaml で指定された名前の pickl ファイルが作られる。画像データの shape は、(画像数，幅，高さ，チャネル数)となる。
回転画像生成スクリプトである rotate.py は、入力設定ファイル名（.yml）をコマンドライン引数として受け取る。

roate.py の入力 yaml ファイルは以下の構成となっている。

```
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	入力学習データのディレクトリパス
train_data	入力学習データのファイル名
train_savedata	出力される拡張データのファイル名　train_dir にここで指定された名前で保存される。
fill_mode	回転により発生する余白のフィル方法を指定　nearest，reflect，wrap　のいずれか
angles	回転角度を配列形式で複数指定（単位:degree）

scale.py

水平、鉛直方向のスケール画像の生成を行う。水平方向は等方的な拡大縮小となる。オフラインデータ拡張に用いる。save_dir 内の train ディレクトリに低解像度画像データ、高解像度画像データのそれぞれ yaml で指定された名前の pickl ファイルが作られる。画像データの shape は、(画像数，幅，高さ，チャネル数)となる。
スケールパラメータは現状スクリプト内にて直接指定している。
スケール画像生成スクリプトである scale.py は、入力設定ファイル名（.yml）をコマンドライン引数として受け取る。

scale.py の入力 yaml ファイルは以下の構成となっている。

```
train_dir: output_origin/train
train_data: ['data_HR.pkl','data_LR.pkl']
train_savedata: ['data_HR_scale.pkl','data_LR_scale.pkl']
# 鉛直方向の拡大縮小倍率
dscales: [0.5, 0.8, 1.2, 1.5]
# 水平方向の拡大縮小倍率
xyscales: [0.5, 0.7, 0.9]
```

Attribute	Description
train_dir	入力学習データのディレクトリパス
train_data	入力学習データのファイル名
train_savedata	出力される拡張データのファイル名 train_dir にここで指定された名前で保存される。
dscales	鉛直方向の拡大縮小倍率を配列形式で複数指定
xyscales	水平方向の拡大縮小倍率を配列形式で複数指定

bathymetric_map_feature.py

海底地形図データに対する各種特徴量を算出し、CSV として保存する。引数として pkl 形式の海底地形図データを指定する。
実行例を以下に示す。

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

実行すると、入力 pkl と同じファイル名の CSV ファイルが生成される。CSV ファイルには以下のカラムが含まれる。

Column name	Description
index	pkl 内の画像インデックス

mean_slope_gradient	傾斜度（規格化前のデータに対する算出結果）
altitude_difference	最大水深差
mean_altitude	平均水深
mean_slope_gradient_norm_data	傾斜度（規格化後のデータに対する算出結果）

傾斜度を用いた test データ評価の際には、本スクリプトにより生成された CSV が存在することが前提となる。

test_da.py

海底地形図データに対するデータ拡張を試行し、結果を画像表示する。
引数として pkl 形式の海底地形図データを指定する。
実行例を以下に示す。

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

実行すると、以下のような画像が表示される。

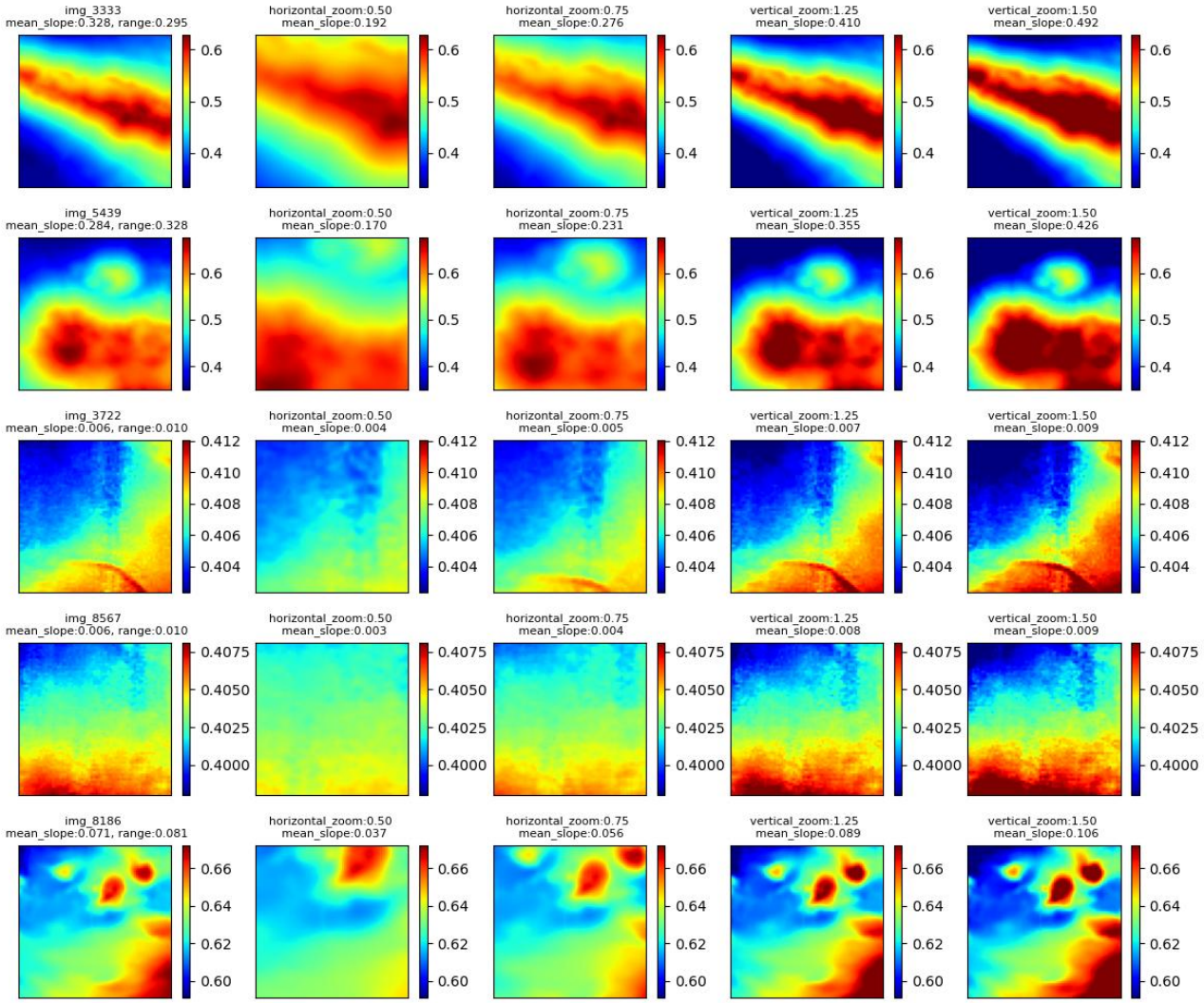

図 **3 test_da.py** の出力結果例

bathymetric_map_feature_LR.py

海底地形図データに対する各種特徴量を算出し、CSV として保存する。引数として pkl 形式の海底地形図データを指定する。

（想定する入力は低解像度パッチデータであり、傾斜度算出は 16×16 を前提としている。）

実行例を以下に示す。

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

実行すると、入力 pkl と同じファイル名の CSV ファイルが生成される（例：data_LR.csv）。

CSV ファイルには以下のカラムが含まれる。

Column name	Description
index	pkl 内の画像インデックス
mean_slope_gradient	傾斜度（規格化前のデータに対する算出結果）
altitude_difference	最大水深差

mean_altitude	平均水深
mean_slope_gradient_norm_data	傾斜度（規格化後のデータに対する算出結果）
normalized_mean_slope_gradient	傾斜度（mean_slope_gradient を 0〜1 に正規化した値）

本スクリプトでは、入力データ全体に対して nanmin / nanmax により正規化範囲を求め、正規化後データに対する傾斜度も併せて算出する。
傾斜度を用いたデータ分布評価やデータ拡張（mixup.py）においては、本スクリプトにより生成された CSV が存在することが前提となる。

mixup.py

地形特徴量分布を考慮した Adaptive Mixup によるデータ拡張を行う。
高解像度画像および低解像度画像を対象に、傾斜度などの地形特徴量を指標として高傾斜データと低傾斜データを組み合わせ、教師データの特徴量分布を制御した拡張データを生成する。
生成されたデータは pickle 形式で保存され、特徴量の再計算および分布調整後のデータ選別処理にも利用される。

入力 yaml ファイルは以下の構成となっている。

```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	入力学習データのディレクトリパス
train_data	入力学習データのファイル名
target_data_csv	目標とする特徴量分布を持つ CSV ファイル
target_value	分布制御に用いる特徴量の指定（0: normalized_mean_slope_gradient, 1: altitude_difference, 2: mean_slope_gradient_norm_data, 3: mean_slope_gradient）
threshold	閾値として使用するパラメータ
std	標準偏差として使用するパラメータ
save_dir_name	出力ディレクトリ名（train_dir 配下に作成される）
random_split	ランダム分割の有効/無効を指定
alpha	Mixup に関連するパラメータとして使用
lamb	Mixup に関連するパラメータとして使用
plot_figure	True の場合、Mixup 結果の可視化画像を出力
original_image_num	元画像枚数（データ拡張前）
increase_rate	データ拡張倍率

get_slope_difference セクション

Attribute	Description
get_slope_difference.valid	本機能の有効/無効
get_slope_difference.input_data_dir	参照入力データのディレクトリ
get_slope_difference.delta	傾斜度算出に用いる格子間隔（delta）

load_csv_norm()

特徴量 CSV を読み込み、傾斜度分布に基づいて高傾斜データ群・低傾斜データ群を抽出・正規化する。元画像・反転画像・回転画像を同一グループとして扱い、分布の偏りを補正するための基準データを生成する。

Attribute	Description
data_csv	学習用データセットの特徴量 CSV ファイルパス／e.g. = "train/data_LR.csv"
target_key	分布制御に用いる特徴量名／次のいずれか mean_slope_gradient, mean_slope_gradient_norm_data, altitude_difference, normalized_mean_slope_gradient
target_data_csv	目標分布として参照する特徴量 CSV パス／e.g. = "target/data_LR.csv"
save_dir	出力先ディレクトリ／e.g. = "train/mixup1"
original_image_num	元画像枚数（データ拡張前）／e.g. = 1000

Return Value	Description
index_high	高傾斜データのインデックス群／e.g. shape = (拡張種別数, 高傾斜枚数)
index_low	低傾斜データのインデックス群／e.g. shape = (拡張種別数, 低傾斜枚数)
slope_high	高傾斜データの特徴量値リスト／e.g. shape = (拡張種別数, 高傾斜枚数)
slope_low	低傾斜データの特徴量値リスト／e.g. shape = (拡張種別数, 低傾斜枚数)
df	入力全体の DataFrame
target_df	目標分布の DataFrame／

load_imgs()

pickle 形式で保存された画像配列を読み込み、処理しやすい形に変換して返す。

Attribute	Description
imgs_file	入力 pickle ファイルパス。pickle 内には画像配列（複数枚）が格納されていること。／e.g. = "train/data_LR.pkl"

Return Value	Description
imgs	整形後画像配列（numpy）／e.g. shape = (画像数, 幅, 高さ) または (画像数, 幅, 高さ, チャネル数)
imgs_org	pickle から読み込んだ元データ（変換前の配列）／e.g. shape = (画像数, 幅, 高さ) または (画像数, 幅, 高さ, チャネル数)

mixup_norm()

傾斜・低傾斜地形画像を組み合わせて特徴量分布を補正する Mixup データ拡張を実施する。
Beta 分布に基づく重みを用い、LR/HR 画像を線形結合することで新規サンプルを生成する。

Attribute	Description
imgs_HR	高解像度画像（正規化後）／e.g. shape = (None, 64, 64) または (None, 64, 64, C)
imgs_HR_org	高解像度画像（元スケール）／e.g. shape = (None, 64, 64) または (None, 64, 64, C)
imgs_LR	低解像度画像（正規化後）／e.g. shape = (None, 16, 16) または (None, 16, 16, C)
imgs_LR_org	低解像度画像（元スケール）／e.g. shape = (None, 16, 16) または (None, 16, 16, C)
index_high	高傾斜データインデックス群　／e.g. shape = (拡張種別数, 高傾斜枚数)
index_low	低傾斜データインデックス群　／e.g. shape = (拡張種別数, 低傾斜枚数)
train_dir	学習データディレクトリ／e.g. = "train"
target_df	目標分布データフレーム

target_key	分布制御に用いる特徴量名／次のいずれか　mean_slope_gradient, mean_slope_gradient_norm_data, altitude_difference, normalized_mean_slope_gradient
slope_normalized	正規化傾斜度を使用するかどうか／e.g. = True

Return Value	Description
-	Mixup 後の HR/LR データおよび中間生成物一式

get_slope()

高解像度（HR）地形パッチ（64×64）に対する平均傾斜度（mean slope gradient）を算出する。
差分近似により x/y 方向の勾配を算出し、各画素の傾斜度を求め、その平均を返す。

Attribute	Description
h	入力地形データ／ shape = (64, 64) を想定
delta	格子間隔（実距離スケール）。／default = 50

Return Value	Description
-	地形パッチ全体の平均傾斜度／e.g. = 0.03

get_slope_LR()

低解像度（LR）地形パッチ（16×16）に対する平均傾斜度（mean slope gradient）を算出する。
HR 用 `get_slope` と同様に差分近似で勾配を求め、各画素の傾斜度の平均を返す。

Attribute	Description
h	入力地形データ／shape = (16, 16) を想定
delta	格子間隔（実距離スケール）。

Return Value	Description
-	地形パッチ全体の平均傾斜度／e.g. = 0.02

pick_images()

Mixup 等で生成・拡張したデータセットから、目標とする特徴量分布に近づくように学習用画像を選別（サンプリング）し、pickle として保存する。
内部では、拡張後データの特徴量 CSV（save_dir/data_LR.csv）と、元データ特徴量 CSV（data_csv）および目標分布（target_df）を参照し、各ビン（ヒストグラム領域）ごとに必要枚数を決定する。
Mixup 生成データと flip 等のデータ拡張データを優先度つきで採用し、saved_dir/picked_images/ 配下へ HR/LR データを保存する。また保存後の picked_images/ の LR pickle に対して

bathymetric_map_feature_LR.py を呼び出し、特徴量 CSV を生成する。

Attribute	Description
target_df	目標とする分布を持つ特徴量データフレーム。
save_dir	出力先ディレクトリ。この中に data_LR.csv 等が存在する想定。／e.g. = "train/mixup1"
imgs_HR_mixup	拡張後の HR 画像配列（元 HR + Mixup HR を含む）。／e.g. shape = (None, 64, 64, 1)
imgs_LR_mixup	拡張後の LR 画像配列（元 LR + Mixup LR を含む）。／e.g. shape = (None, 16, 16, 1)
original_image_num	元画像枚数（データ拡張前）。index 判定や区間分割の基準に用いる。／e.g. = 1000
target_key	分布制御に用いる特徴量名／次のいずれか mean_slope_gradient, mean_slope_gradient_norm_data, altitude_difference, normalized_mean_slope_gradient
data_csv	元データ（もしくは参照元）の特徴量 CSV パス／e.g. = "train/data_LR.csv"
increase_rate	分布調整で目標とする増加倍率（ビンごとの必要枚数の基準に使用）

idx_of_the_nearest()

配列 data の中から value に最も近い要素のインデックスを返す。

Attribute	Description
data	数値の配列（list, numpy.ndarray 等）。
value	探索対象の値。／e.g. = 0.35

Return Value	Description
-	value に最も近い要素のインデックス（最小の絶対誤差を与える要素）

indices_of_the_nearest()

配列 data の中から value に最も近い要素のインデックスをすべて返す（同距離がある場合を想定）。

Attribute	Description
data	数値の配列（list, numpy.ndarray 等）。
value	探索対象の値。／e.g. = 0.35

Return Value	Description
-	value に最も近い要素のインデックス配列（同距離の要素が複数ある場合は複数

要素を返す)。／e.g. shape = (L,)（L は同距離要素数）

1.4. 推奨動作環境

プログラムは、Python 3.8 の環境上で、以下のパッケージを利用して動作する。

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

1.5. 使用方法

Dataset クラスを初期化し、データセットを構築する。

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

構築したデータセットを保存する。

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

正常に動作すると、下記のとおりプログレスが表示される。

```
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. プログラムの概要

sr_trainer は、下記のアルゴリズム（SRCNN・FSRCNN・ESPCN・SRGAN・ESRGAN）を用いた超解像の深層学習・テストを行うプログラムである。

2.1.1. SRCNN: Super-Resolution Convolutional Neural Network

SRCNN [2]は、3 層の畳み込みニューラルネットワークである。ネットワーク中の入出力画像のサイズは同一であり、事前に Bicubic 補間等で低解像度画像を拡大する必要がある。ネットワークの入力前に画像を拡大する超解像手法は、Pre-upsampling SR と呼ばれる。

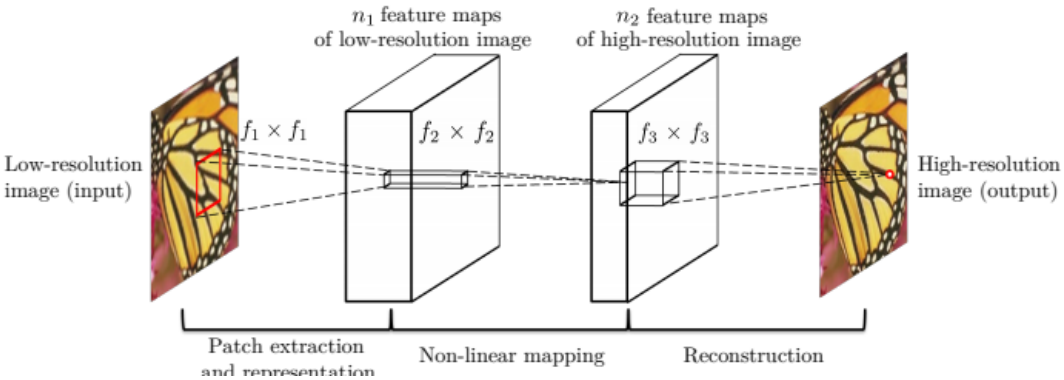

図 4 SRCNN（Super-Resolution Convolutional Neural Network）の構造概念図 [2]より引用

2.1.2. FSRCNN: Fast Super-Resolution Convolutional Neural Network

FSRCNN [3]は、SRCNN を高速化したアルゴリズムである。SRCNN は Pre-upsampling 画像を入力とするため、入力画像サイズが大きい。一方、FSRCNN は、低解像度画像に直接畳み込みニューラルネットワークを適用し、最終層で Deconvolution を用いて画像を拡大する。このネットワーク通過後に画像を拡大する超解像手法は、Post-upsampling SR と呼ばれる。FSRCNN の論文では、カーネルサイズ等も合わせてチューニングし、SRCNN の数十倍の高速化を達成している。

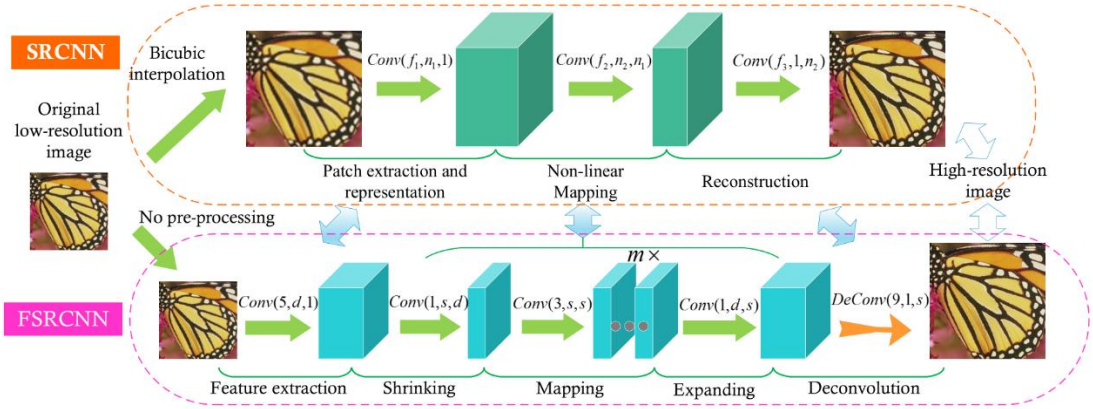

図 5　FSRCNN（Fast Super-Resolution Convolutional Neural Network）のネットワーク構造 [3]より引用

2.1.3. ESPCN: Efficient Sub-Pixel Convolutional Neural Network

ESPCN [4]は、FSRCNN と同じく Post-upsampling SR である。ただし、Deconvolution ではなく、Sub-Pixel Convolution を用いて画像拡大を行う。Deconvolution は、下図に示すように畳み込み時に参照する画素数が場所によって異なるため、Checkerboard Artifact が顕在化しやすい。また、Deconvolution 時に挿入された画素は本質的な情報ではないため、非効率な学習を招く。

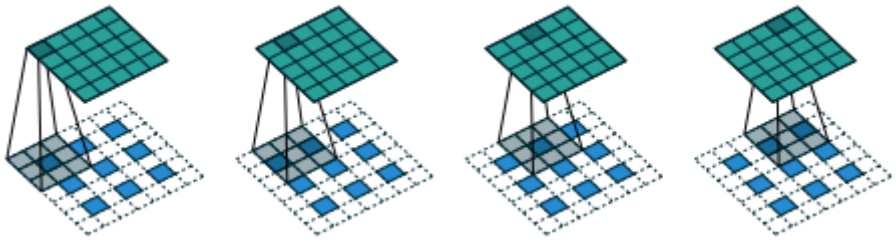

図 6 Sub-Pixel Convolution によるアップサンプリングの概念図 [5]より引用

ESPCN で採用する Sub-Pixel Convolution は、補間画素を挿入することなく、チャネルの画素を並び替えて画像を拡大する。このとき、参照する画素数が空間的に均一になるため、Deconvolution の副作用である Checkerboard Artifact が抑制される。また、画素の挿入がないため、FSRCNN よりもさらに高速化される。

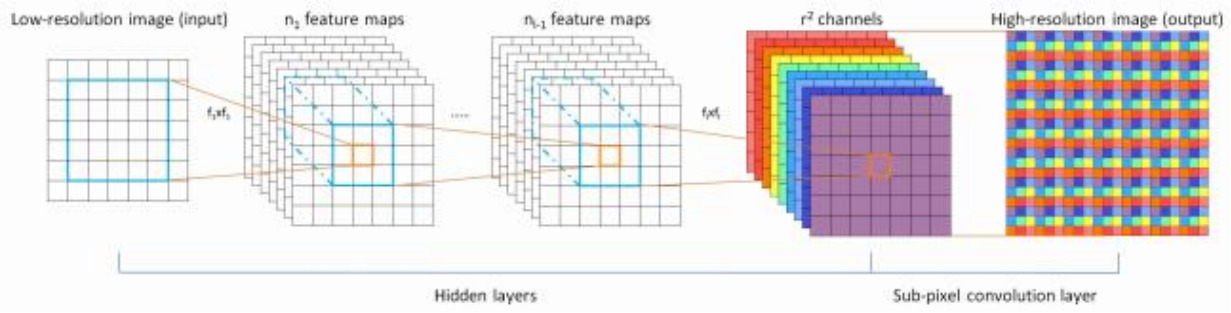

図 7 ESPCN（Efficient Sub-Pixel Convolutional Neural Network）のネットワーク構造 [4]より引用

2.1.4. SRGAN: Super-Resolution using Generative Adversarial Network

SRGAN [6]は、Generator と Discriminator が協調して性能を高めることにより、超解像度の品質を改善する手法である。Generator は、低解像度画像から高解像度画像を生成する。Discriminator は、入力画像が Generator の生成した超解像度画像であるか、本来の高解像度画像であるかを識別する。この Generator と Discriminator の学習を交互に行い、Generator の画像誤差と Discriminator の識別誤差をバランスさせることによって Generator 単体の性能限界を突破する。

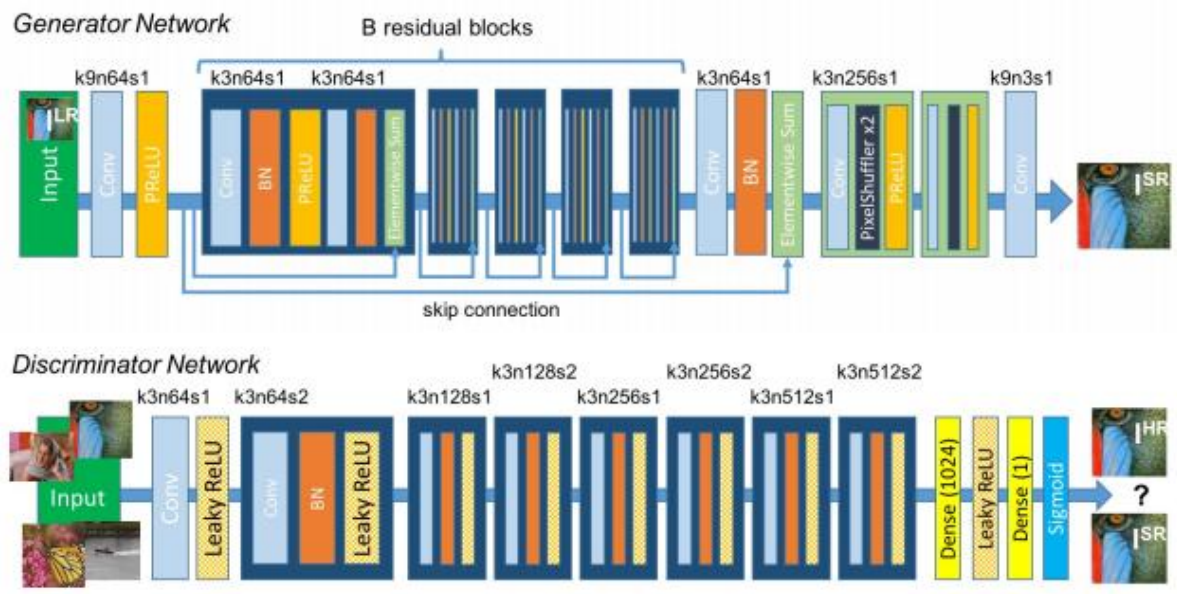

図 8 SRGAN（Super-Resolution Generative Adversarial Network）のネットワーク構造 [6]より引用

2.1.5. ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks

ESRGAN [7]は、SRGAN を改良して表現力と学習効率を高めたモデルである。以下では、ESRGAN において SRGAN からの変更された 3 点について記述する。

(1) Generator の Batch Normalization を除去

ESRGAN 論文の Supplemental Material で指摘されているように、Batch Normalization は、不自然なアーティファクトを発生させる。これは、学習時とテスト時のバッチの統計量が異なるために生じる。つまり、学習時は正確なバッチの平均と分散から正規化を行う一方、テスト時は推定した平均と分散を使うことになる（例えば、16 バッチで学習させて 1 枚の画像をテストする場合、1 枚のテスト画像からは 16 枚分の統計情報は得られない）。

この理由により、ESRGAN では、Generator から Batch Normalization が除去された。

(2) Residual Block の拡張

ESRGAN では、SRGAN の Residual Block を下図に示す Residual in Residual Dense Block (RRDB)に拡張している。RRDB では、係数 β 倍（β ～ 0.2）した弱い残差結合により、より広範囲の前後層の特徴が反映される。

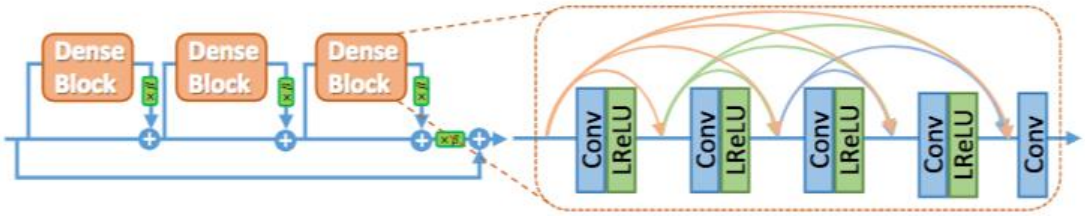

図 9 Residual in Residual Dense Block (RRDB)の構造（概念図） [7]より引用

(3) Relativistic Discriminator の導入

SRGAN では、Discriminator の学習が進むと、Real 画像の判定に失敗することがなくなる。そして、Fake 画像を Real と誤判定した場合のみ、Discriminator の性能が改善する。この Discriminator $D(x)$ の振る舞いは、Real バッチ画像 x_r と Fake バッチ画像 x_f に対して次式で表される。

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

ここで、 σ はシグモイド関数、 C は Discriminator の出力値である。しかしながら、上式では終盤に入力の半分しか学習に反映されず、非効率である。そこで ESRGAN では、次式の Relativistic Discriminator $D_{Ra}(x)$ を導入する。

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

ここで、 E はバッチ平均を示す。第 1 式は、Fake 画像の平均予測値よりも Real 画像の予測値が大きいことを強制し、第 2 式は、Real 画像の平均予測値よりも Fake 画像の予測値が小さいことを強制する。これにより、判定が完全でない限り、Real・Fake 画像の両方に対して学習が進む。Relativistic Discriminator の損失関数 $L_D^{Ra}(x)$ は、この両式を並列させる。

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

Generator の学習では、Relativistic Discriminator を欺くため、x_r と x_f を交換して学習する。

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

2.2. 各種機能

2.2.1. クラス図

sr_trainer のクラス間の関係を UML クラス図（属性・メソッドは省略）として以下に示す。

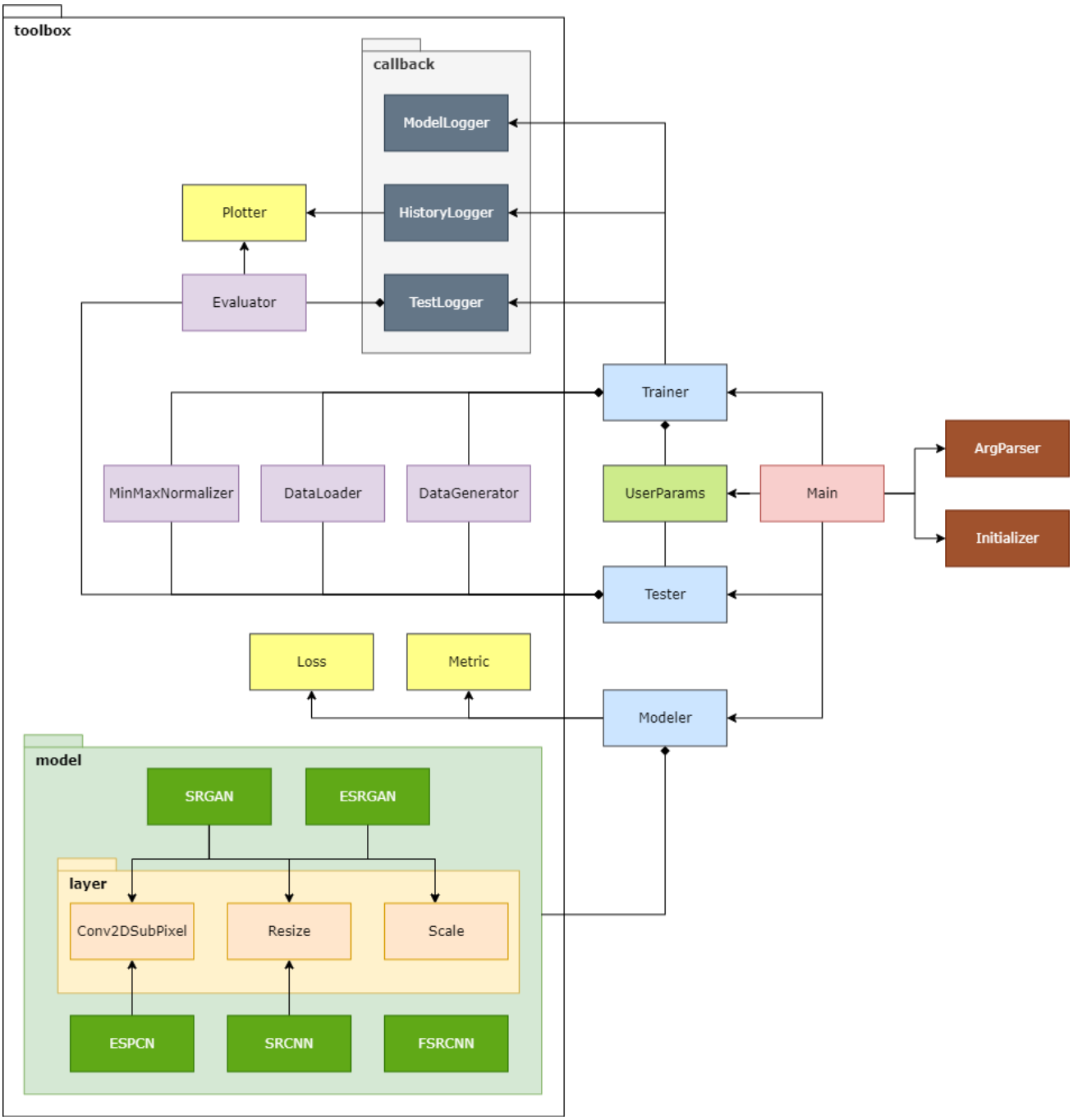

図 10 クラス図

2.2.2. toolbox.model.layer.Resize クラス

keras.engine.topology.Layer クラスを継承して画像拡大（Bilinear, Bicubic 補間等）を行うカスタムレイヤーを構築する。

Attribute	Description
scale	画像の拡大倍率／e.g. = 2
method	画像処理アルゴリズム／default = tf.image.ResizeMethod.BICUBIC

__init__()

属性と keras.engine.topology.Layer クラスを初期化する。

Argument	Description
scale	画像の拡大倍率／e.g. = 2
method	画像処理アルゴリズム／default = tf.image.ResizeMethod.BICUBIC
trainable	学習の可否／default = False
margin	リサイズ後画像の周囲に設定するマージン幅（ピクセル数）／default = 0
**kwargs	その他の可変長引数

resized_shape()

拡大後の画像形状を取得する（先頭のバッチ次元と末尾のチャネル次元は削除）。

Argument	Description
input_shape	入力画像形状／e.g. shape = (None, 32, 32, 1)

Return Value	Description
-	拡大後の画像形状／e.g. shape = (64, 64)

call()

tf.image.resize_image を使い、画像を拡大する。

Argument	Description
x	入力画像／e.g. shape = (None, 32, 32, 1)

Return Value	Description
-	拡大画像／e.g. shape = (Nome, 64, 64, 1)

compute_output_shape()

出力画像の形状を取得する。

Argument	Description
input_shape	入力画像形状／e.g. shape = (None, 32, 32, 1)

Return Value	Description
-	出力画像形状／e.g. shape = (None, 64, 64, 1)

get_config()

属性を追加したレイヤーの設定を取得する。

Return Value	Description
-	レイヤー設定の辞書

2.2.3. toolbox.model.layer.Conv2DSubPixel クラス

keras.engine.topology.Layer クラスを継承して Sub-Pixel Convolution を行うカスタムレイヤーを構築する。

Attribute	Description
scale	画像の拡大倍率／e.g. = 2

__init__()

属性と keras.engine.topology.Layer クラスを初期化する。

Argument	Description
scale	画像の拡大倍率／e.g. = 2
trainable	学習の可否／default = False
**kwargs	その他の可変長引数

call()

tf.depth_to_space を使い、Sub-Pixel Convolution を適用する。

Argument	Description
x	入力画像／e.g. shape = (None, 16, 16, 1)

Return Value	Description
-	拡大画像／e.g. shape = (Nome, 64, 64, 1)

compute_output_shape()

出力画像の形状を取得する。

Argument	Description
input_shape	入力画像形状／e.g. shape = (None, 16, 16, 1)

Return Value	Description
-	出力画像形状／e.g. shape = (None, 64, 64, 1)

get_config()

属性を追加したレイヤーの設定を取得する。

Return Value	Description
-	レイヤー設定の辞書

2.2.4. toolbox.model.layer.Scale クラス

keras.engine.topology.Layer クラスを継承して画像を定数倍するカスタムレイヤーを構築する。

Attribute	Description
factor	倍率／e.g. = 0.2

__init__()

属性と keras.engine.topology.Layer クラスを初期化する。

Argument	Description
factor	倍率／e.g. = 0.2
trainable	学習の可否／default = False
**kwargs	その他の可変長引数

call()

ラムダレイヤーを使い、画像を定数倍する。

Argument	Description
x	入力画像／e.g. shape = (None, 16, 16, 1)

Return Value	Description
-	定数倍した画像／e.g. shape = (Nome, 16, 16, 1)

compute_output_shape()

出力画像の形状を取得する。

Argument	Description
input_shape	入力画像形状／e.g. shape = (None, 16, 16, 1)

Return Value	Description
-	出力画像形状／e.g. shape = (None, 16, 16, 1)

get_config()

属性を追加したレイヤーの設定を取得する。

Return Value	Description
-	レイヤー設定の辞書

2.2.5. toolbox.model.srcnn.SRCNN クラス

keras.models.Model クラスを継承し、SRCNN モデルを構築する。

Attribute	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2
params	パラメータ設定

__init__()

keras.models.Model クラスを初期化後、*__build__*メソッドを呼び出し、SRCNN モデルを構築する。

Argument	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2

__build__()

SRCNN モデルを構築する。チャネル数とカーネルサイズは、原著論文に準ずる。Bicubic 補間による Pre-upsampling 処理は、toolbox.model.layer.Resize カスタムレイヤーを利用する。

Argument	Description
filters	畳み込み層のチャネル数／default = [32, 16]
kernels	畳み込み層のカーネルサイズ／default = [9, 1, 5]

2.2.6. toolbox.model.fsrcnn.FSRCNN クラス

keras.models.Model クラスを継承し、FSRCNN モデルを構築する。

Attribute	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2

__init__()

keras.models.Model クラスを初期化後、*__build__*メソッドを呼び出し、FSRCNN モデルを構築する。

Argument	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2

__build__()

FSRCNN モデルを構築する。チャネル数とカーネルサイズは、原著論文に準ずる。

Argument	Description
d	先頭と末尾の畳み込み層のチャネル数／default = 56
s	中間畳み込み層のチャネル数／default = 12
m	中間畳み込み層の繰り返し数／default = 4

2.2.7. toolbox.model.espcn.ESPCN クラス

keras.models.Model クラスを継承し、ESPCN モデルを構築する。

Attribute	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2

__init__()

keras.models.Model クラスを初期化後、*__build__*メソッドを呼び出し、ESPCN モデルを構築する。

Argument	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2

__build__()

ESPCN モデルを構築する。チャネル数とカーネルサイズは、原著論文に準ずる。Sub-Pixel Convolution には、toolbox.model.layer.Conv2DSubPixel カスタムレイヤーを利用する。

Argument	Description
filters	畳み込み層のチャネル数／default = [64, 32]
kernels	畳み込み層のカーネルサイズ／default = [5, 3, 3]

2.2.8. toolbox.model.srgan.Generator クラス

keras.models.Model クラスを継承し、SRGAN の Generator モデルを構築する。

Attribute	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2

__init__()

keras.models.Model クラスを初期化後、*__build__*メソッドを呼び出し、SRGAN の Generator モデルを構築する。

Argument	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2

__residual_block()

Conv2D, BatchNormalization, PReLU を組み合わせた残差ブロックを構築する。

Argument	Description
filters	畳み込み層のチャネル数／e.g. = 32
kernel_initializer	カーネル初期化子／e.g. = 'he_normal'
momentum	BatchNormalization の移動平均における運動量係数／default = 0.8

Return Value	Description
-	残差ブロック

__upsampling_block()

画像拡大を行う Upsampling ブロックを構築する。Upsampling には、Sub-Pixel Convolution を採用する。ただし、distill.pub（https://distill.pub/2016/deconv-checkerboard/）で指摘されているように、Checkerboard Artifact の防止には、Nearest-Neighbor もしくは Bilinear 補間による拡大が効果的である。しかし、その副作用として画像端部のノイズを助長するため、現在は *resize_conv* 内部関数を未使用の状態に留めている。

Argument	Description
filters	畳み込み層のチャネル数／e.g. = 32
kernel_initializer	カーネル初期化子／e.g. = 'he_normal'

Return Value	Description
-	Upsampling ブロック

__build__()

SRGAN の Generator モデルを構築する。

Argument	Description
filters	畳み込み層のチャネル数／default = 32
num_residual_blocks	残差ブロック数／default = 1
kernel_initializer	カーネル初期化子／e.g. = 'he_normal'
momentum	BatchNormalization の移動平均における運動量係数／default = 0.8

2.2.9. toolbox.model.srgan.Discriminator クラス

keras.models.Model クラスを継承し、SRGAN の Discrimiantor モデルを構築する。

Attribute	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2

__init__()

keras.models.Model クラスを初期化後、*__build__*メソッドを呼び出し、SRGAN の Discriminator モデルを構築する。

Argument	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2

__conv2d_block()

Conv2D, BatchNormalization, LeakyReLU を組み合わせた畳み込みブロックを構築する。

Argument	Description
filters	畳み込み層のチャネル数／e.g. = 4
kernel_size	畳み込み層のカーネルサイズ／default = 3
strides	畳み込み層のストライド長／default = 1
use_bn	BatchNormalization 層の挿入の可否／default = True
kernel_initializer	カーネル初期化子／e.g. = 'he_normal'
momentum	BatchNormalization の移動平均における運動量係数／default = 0.8
alpha	LeakyReLU の傾き／default = 0.2

Return Value	Description
-	畳み込みブロック

__build__()

SRGAN の Discriminator モデルを構築する。

Argument	Description
filters	畳み込み層のチャネル数／default = 4
num_downsampling_blocks	Downsampling ブロック数／default = 1
kernel_initializer	カーネル初期化子／e.g. = 'he_normal'
momentum	BatchNormalization の移動平均における運動量係数／default = 0.8
alpha	LeakyReLU の傾き／default = 0.2

2.2.10. toolbox.model.srgan.SRGAN クラス

keras.models.Model クラスを継承し、SRGAN モデルを構築する。

Attribute	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2
label_noise	正解・不正解ラベルのノイズ比／default = 1e-6
loss_weight	Generator と Discriminator の損失係数／default = [1, 1e-7]
generator	toolbox.model.srgan.Generator モデル
discriminator	toolbox.model.srgan.Discriminator モデル

__init__()

keras.models.Model クラスを初期化後、*__build__*メソッドを呼び出し、SRGAN モデルを構築する。

Argument	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2
label_noise	正解・不正解ラベルのノイズ比／default = 1e-6
loss_weight	Generator と Discriminator の損失係数／default = [1, 1e-7]

__build__()

Generator と Discriminator を繋げた GAN モデルを構築する。

summary()

keras.models.Model クラスの *summary* メソッドをオーバーロードし、Generator, Discriminator, GAN モデルのレイヤーを表示する。

compile()

keras.models.Model クラスの *compile* メソッドをオーバーロードし、Generator, Discriminator, GAN モデルをコンパイルする。Generator を学習する GAN モデルでは、Discriminator の重みを凍結する。Discriminator の損失関数は、binary_crossentropy とする。

Argument	Description
loss	Generator 画像の損失関数／e.g. = 'mse'
optimizer	最適化関数／e.g. = 'adam'
metrics	PSNR や DSSIM 等の評価関数リスト／default = None

sample_labels()

Discriminator, GAN モデルの正解・不正解ラベルを生成する。

Argument	Description
batch_size	正解・不正解ラベルのベクトル長（バッチサイズ）
noise	正解・不正解ラベルのノイズ比／e.g. = 1e-6

Return Value	Description
list[0]	正解ラベル／e.g. shape = (32,)
list[1]	不正解ラベル／e.g. shape = (32,)

train_on_batch()

keras.models.Model クラスの *train_on_batch* メソッドをオーバーロードし、Discriminator と GAN モデルのバッチ学習を行う。

Argument	Description
lr_imgs	低解像度画像バッチ／e.g. shape = (バッチサイズ, 16, 16, 1)
hr_imgs	高解像度画像バッチ／e.g. shape = (バッチサイズ, 64, 64, 1)
sample_weight	親クラスのオプションパラメータ／default = None
class_weight	親クラスのオプションパラメータ／default = None
reset_metrics	親クラスのオプションパラメータ／default = True

Return Value	Description
-	損失と評価値のリスト

test_on_batch()

keras.models.Model クラスの *test_on_batch* メソッドをオーバーロードし、GAN モデルのバッチテストを行う。

Argument	Description
lr_imgs	低解像度画像バッチ／e.g. shape = (バッチサイズ, 16, 16, 1)
hr_imgs	高解像度画像バッチ／e.g. shape = (バッチサイズ, 64, 64, 1)
sample_weight	親クラスのオプションパラメータ／default = None
reset_metrics	親クラスのオプションパラメータ／default = True

Return Value	Description
-	損失と評価値のリスト

predict()

keras.models.Model クラスの *predict* メソッドをオーバーロードし、Generator による超解像処理を実行する。

Argument	Description
x	低解像度画像バッチ／e.g. shape = (バッチサイズ, 16, 16, 1)
batch_size	バッチサイズ
verbose	プログレスバー（0: 表示なし, 1: 表示）／default = 0
steps	親クラスのオプションパラメータ／default = None

Return Value	Description
-	超解像画像バッチ／e.g. shape = (バッチサイズ, 64, 64, 1)

2.2.11. toolbox.model.esrgan.Generator クラス

keras.models.Model クラスを継承し、ESRGAN の Generator モデルを構築する。

Attribute	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2

__init__()

keras.models.Model クラスを初期化後、*__build__*メソッドを呼び出し、ESRGAN の Generator モデルを構築する。

Argument	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2
filters	畳み込み層のチャネル数
num_residual_blocks	RRDB ブロック数

__DB()

ESRGAN の Dense Block (DB)レイヤーを構築する。

Argument	Description
filters	畳み込み層のチャネル数／e.g. = 32
kernel_initializer	カーネル初期化子／e.g. = 'he_normal'
alpha	LeakyReLU の傾き／e.g. = 0.2
kernel_size	畳み込み層のカーネルサイズ／default = 3
strides	畳み込み層のストライド長／default = 1

Return Value	Description
-	Dense Block (DB)レイヤー

__RRDB()

ESRGAN の Residual-in-Residual Dense block (RRDB)レイヤーを構築する。

Argument	Description
filters	畳み込み層のチャネル数／e.g. = 32
kernel_initializer	カーネル初期化子／e.g. = 'he_normal'
alpha	LeakyReLU の傾き／e.g. = 0.2
beta	残差結合係数／e.g. = 0.2

Return Value	Description
-	Residual-in-Residual Dense Block (RRDB)レイヤー

__upsampling_block()

画像拡大を行う Upsampling ブロックを構築する。Upsampling には、Sub-Pixel Convolution を採用する。ただし、distill.pub（https://distill.pub/2016/deconv-checkerboard/）で指摘されているように、Checkerboard Artifact の防止には、Nearest-Neighbor もしくは Bilinear 補間による拡大が効果的である。しかし、その副作用として画像端部のノイズを助長するため、現在は *resize_conv* 内部関数を未使用の状態に留めている。

Argument	Description
filters	畳み込み層のチャネル数／e.g. = 32
kernel_initializer	カーネル初期化子／e.g. = 'he_normal'

Return Value	Description
-	Upsampling ブロック

__build__()

ESRGAN の Generator モデルを構築する。

Argument	Description
filters	畳み込み層のチャネル数／default = 32
num_residual_blocks	RRDB ブロック数／default = 2
kernel_initializer	カーネル初期化子／e.g. = 'he_normal'
alpha	LeakyReLU の傾き／e.g. = 0.2
beta	残差結合係数／e.g. = 0.2

2.2.12. toolbox.model.esrgan.Discriminator クラス

keras.models.Model クラスを継承し、ESRGAN の Discrimiantor モデルを構築する。SRGAN の Discriminator とは、ドロップアウト層を追加した点が異なる。

Attribute	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2

__init__()

keras.models.Model クラスを初期化後、*__build__*メソッドを呼び出し、ESRGAN の Discriminator モデルを構築する。

Argument	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2
d_filters	／e.g. d_filters = 4
d_num_downsampling_blocks	／e.g. d_num_downsampling_blocks = 2

__conv2d_block()

Conv2D, BatchNormalization, LeakyReLU を組み合わせた畳み込みブロックを構築する。

Argument	Description
filters	畳み込み層のチャネル数／e.g. = 4
kernel_size	畳み込み層のカーネルサイズ／default = 3
strides	畳み込み層のストライド長／default = 1
use_bn	BatchNormalization 層の挿入の可否／default = True
kernel_initializer	カーネル初期化子／e.g. = 'he_normal'
momentum	BatchNormalization の移動平均における運動量係数
alpha	LeakyReLU の傾き／default = 0.2

Return Value	Description
-	畳み込みブロック

__build__()

ESRGAN の Discriminator モデルを構築する。

Argument	Description
filters	畳み込み層のチャネル数／default = 4
num_downsampling_blocks	Downsampling ブロック数／default = 2
kernel_initializer	カーネル初期化子／e.g. = 'he_normal'
momentum	BatchNormalization の移動平均における運動量係数
alpha	LeakyReLU の傾き
dropout_rate	ドロップアウトの比率／default = 0.4

2.2.13. toolbox.model.esrgan.RelativisticDiscriminator クラス

keras.models.Model クラスを継承し、ESRGAN の Relativistic Discriminator モデルを構築する。

__init__()

keras.models.Model クラスを初期化後、*__build__*メソッドを呼び出し、ESRGAN の Discriminator モデルを構築する。

Argument	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2
d_filters	／e.g. d_filters = 4
d_num_downsampling_blocks	／e.g. d_num_downsampling_blocks = 2

__ra_loss()

Relativistic Average Loss を計算する。

Argument	Description
x	Discriminator が出力する正解・不正解ラベル
noise	Log の発散防止用ノイズ係数／default = 1e-6

Return Value	Description
-	Relativistic Average Loss

__build__()

ESRGAN の Relativistic Discriminator モデルを構築する。

2.2.14. toolbox.model.esrgan.ESRGAN クラス

keras.models.Model クラスを継承し、ESRGAN モデルを構築する。

Attribute	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2
loss_weight	Generator と Discriminator の損失係数／default = [1, 1e-7]
generator	toolbox.model.esrgan.Generator モデル
discriminator	toolbox.model.esrgan.RelativisticDiscriminator モデル

__init__()

keras.models.Model クラスを初期化後、*__build__*メソッドを呼び出し、ESRGAN モデルを構築する。

Argument	Description
img_shape	低解像度画像の形状／e.g. shape = (16, 16, 1)
scale	超解像の拡大倍率／e.g. = 2
loss_weight	Generator と Discriminator の損失係数／default = [1, 1e-7]
params	パラメータ／default = None

__build__()

Generator と Relativistic Discriminator を繋げた GAN モデルを構築する。

summary()

keras.models.Model クラスの *summary* メソッドをオーバーロードし、Generator, Relativistic Discriminator, GAN モデルのレイヤーを表示する。

compile()

keras.models.Model クラスの *compile* メソッドをオーバーロードし、Generator, Relativistic Discriminator, GAN モデルをコンパイルする。Generator を学習する GAN モデルでは、Relativistic Discriminator の重みを凍結する。ESRGAN の Generator と Discriminator では、*add_loss()*メソッドを用いて、入力から算出される損失関数を明示的に指定する。すなわち、*train_on_batch()*メソッドにおいて、出力項 outputs は使用しない（outputs=None とする）。

Argument	Description
loss	Generator 画像の損失関数／e.g. = 'mse'
optimizer	最適化関数／e.g. = 'adam'
metrics	PSNR や DSSIM 等の評価関数リスト／default = None

train_on_batch()

keras.models.Model クラスの *train_on_batch* メソッドをオーバーロードし、Relativistic Discriminator と GAN モデルのバッチ学習を行う。

Argument	Description
lr_imgs	低解像度画像バッチ／e.g. shape = (バッチサイズ, 16, 16, 1)
hr_imgs	高解像度画像バッチ／e.g. shape = (バッチサイズ, 64, 64, 1)
sample_weight	親クラスのオプションパラメータ／default = None
class_weight	親クラスのオプションパラメータ／default = None
reset_metrics	親クラスのオプションパラメータ／default = True

Return Value	Description
-	損失と評価値のリスト

test_on_batch()

keras.models.Model クラスの *test_on_batch* メソッドをオーバーロードし、GAN モデルのバッチテストを行う。

Argument	Description
lr_imgs	低解像度画像バッチ／e.g. shape = (バッチサイズ, 16, 16, 1)
hr_imgs	高解像度画像バッチ／e.g. shape = (バッチサイズ, 64, 64, 1)
sample_weight	親クラスのオプションパラメータ／default = None
reset_metrics	親クラスのオプションパラメータ／default = True

Return Value	Description
-	損失と評価値のリスト

predict()

keras.models.Model クラスの *predict* メソッドをオーバーロードし、Generator による超解像処理を実行する。

Argument	Description
x	低解像度画像バッチ／e.g. shape = (バッチサイズ, 16, 16, 1)
batch_size	バッチサイズ
verbose	プログレスバー（0: 表示なし, 1: 表示）／default = 0
steps	親クラスのオプションパラメータ／default = None

Return Value	Description
-	超解像画像バッチ／e.g. shape = (バッチサイズ, 64, 64, 1)

2.2.15. toolbox.metric.Metric クラス

画像品質の評価関数を定義する。

psnr()

Peak Signal-To-Noise Ratio (PSNR)の評価関数を定義する。

Argument	Description
y_true	正解データ／e.g. shape = (バッチサイズ, 64, 64, 1)
y_pred	予測データ／e.g. shape = (バッチサイズ, 64, 64, 1)

Return Value	Description
-	PSNR

dssim()

Structural Disimilarity (DSSIM)の評価関数を定義する。DSSIM は、Strucural Similarity (SSIM)から DSSIM = (1 – SSIM) / 2 と与えられる。

Argument	Description
y_true	正解データ／e.g. shape = (バッチサイズ, 64, 64, 1)
y_pred	予測データ／e.g. shape = (バッチサイズ, 64, 64, 1)

Return Value	Description
-	DSSIM

2.2.16. toolbox.loss.Loss クラス

学習で最小化する損失関数を定義する。

psnr()

Peak Signal-To-Noise Ratio (PSNR)の逆数を最小化する損失関数を定義する。

Argument	Description
y_true	正解データ／e.g. shape = (バッチサイズ, 64, 64, 1)
y_pred	予測データ／e.g. shape = (バッチサイズ, 64, 64, 1)

Return Value	Description
-	1 / PSNR

dssim()

Structural Disimilarity (DSSIM)を最小化する損失関数を定義する。

Argument	Description
y_true	正解データ／e.g. shape = (バッチサイズ, 64, 64, 1)
y_pred	予測データ／e.g. shape = (バッチサイズ, 64, 64, 1)

Return Value	Description
-	DSSIM

vgg()

VGG19 モデルにおける中間層のコンテンツ誤差を最小化する損失関数を定義する。__preprocess 関数は、VGG19 モデルの入力用にデータを規格化する。VGG19 モデルの入力は、0-255 範囲の RGB 画像である。penalty_l1, pnalty_l2 関数は、0-1 の範囲に規格化されていない予測値に対する L1, L2 正則化項の実装である（Generator の最終層の活性化関数として勾配消失を招く tanh ではなく、linear を使用する場合に必要）。

Argument	Description
y_true	正解データ／e.g. shape = (バッチサイズ, 64, 64, 1)
y_pred	予測データ／e.g. shape = (バッチサイズ, 64, 64, 1)
feature_index	コンテンツ誤差を評価するレイヤー番号／default = 20

Return Value	Description
-	正則化した feature_index 層のコンテンツ誤差

2.2.17. toolbox.callbacks.ModelLogger クラス

keras.callbacks.ModelCheckPoint クラスを継承し、各エポックにおけるモデルの重みを weights_*.h5 に保存する。また、val_loss が最小となったエポックの重みも weights_best.h5 に保存する。

Attribute	Description
log_dir	データの出力ディレクトリ名／e.g. = 'log'
log_period	データの出力頻度／e.g. = 10

__init__()

属性と keras.callbacks.ModelCheckPoint クラスを初期化する。

Argument	Description
log_dir	データの出力ディレクトリ名／e.g. = 'log'
log_period	データの出力頻度／e.g. = 10

on_epoch_end()

keras.callbacks.ModelCheckPoint クラスの *on_epoch_end* メソッドをオーバーロードし、エポック終了時にモデルの重みを保存する。

Argument	Description
epoch	現在のエポック数
logs	現在の損失と評価値の辞書／e.g. = { 'loss':1e-3, 'psnr':51.1, ...}

2.2.18. toolbox.callbacks.HistoryLogger クラス

keras.callbacks.CSVLogger クラスを継承し、各エポックにおける学習履歴グラフを history_*.png に保存する。また、全エポックの学習履歴を history.csv に保存する。

Attribute	Description
log_dir	データの出力ディレクトリ名／e.g. = 'log'
log_period	データの出力頻度／e.g. = 10
epochs	エポック履歴
logs	損失と評価値の履歴辞書／e.g. = { 'loss':[], 'psnr':[], ...}

__init__()

属性と keras.callbacks.CSVLogger クラスを初期化する。

Argument	Description
log_dir	データの出力ディレクトリ名／e.g. = 'log'
log_period	データの出力頻度／e.g. = 10

on_epoch_end()

keras.callbacks.CSVLogger クラスの *on_epoch_end* メソッドをオーバーロードし、エポック終了時に学習履歴を history.csv に追記し、学習履歴グラフを保存する。

Argument	Description
epoch	現在のエポック数
logs	現在の損失と評価値の辞書／e.g. = { 'loss':1e-3, 'psnr':51.1, ...}

2.2.19. toolbox.callbacks.TestLogger クラス

keras.callbacks.Callback クラスを継承し、各エポックにおける超解像モデルの性能テストを実行し、テスト結果を test_*.png に保存する。

Attribute	Description
log_dir	データの出力ディレクトリ名／e.g. = 'log'
log_period	データの出力頻度／e.g. = 10
data_generator	検証データ生成器
denorm_func	正規化データの逆変換関数
num_samples	テスト画像のサンプル数／e.g. = 4
evaluator	toolbox.evaluator.Evaluator クラスオブジェクト（モデルの性能評価）

__init__()

属性を初期化する。

Argument	Description
log_dir	データの出力ディレクトリ名／e.g. = 'log'
log_period	データの出力頻度／e.g. = 10
data_generator	検証データ生成器
denorm_func	正規化データの逆変換関数
num_samples	テスト画像のサンプル数／e.g. = 4

on_epoch_end()

keras.callbacks.Callback クラスの *on_epoch_end* メソッドをオーバーロードし、エポック終了時に超解像モデルの性能テストを実行し、テスト結果を test_*.png に保存する。

Argument	Description
epoch	現在のエポック数
logs	現在の損失と評価値の辞書／e.g. = { 'loss':1e-3, 'psnr':51.1, ...}

2.2.20. toolbox.data_loader.DataLoader クラス

train, validation, test ディレクトリに保存されたデータセットから、教師・検証・テストデータを読み込む。

Attribute	Description
dataset	教師・検証・テストデータセットの辞書／e.g. dataset['train']

__init__()

train, validation, test ディレクトリに保存されたデータセットから、教師・検証・テストデータを読み込み、dataset 属性を初期化する。

Argument	Description
data_dir	データの出力ディレクトリ名／e.g. = 'log'
sub_dirs	教師・検証・テストデータディレクトリ名
sr_scale	超解像倍率。／default = None
normalize	正規化処理を行うかどうかのフラグ。／default = False

train()

教師データセットを取得する。

Return Value	Description
tuple[0]	教師用の低解像度画像群／e.g. shape = (画像数, 16, 16, 1)
tuple[1]	教師用の高解像度画像群／e.g. shape = (画像数, 64, 64, 1)

validation()

検証データセットを取得する。

Return Value	Description
tuple[0]	検証用の低解像度画像群／e.g. shape = (画像数, 16, 16, 1)
tuple[1]	検証用の高解像度画像群／e.g. shape = (画像数, 64, 64, 1)

test()

テストデータセットを取得する。

Return Value	Description
tuple[0]	テスト用の低解像度画像群／e.g. shape = (画像数, 16, 16, 1)
tuple[1]	テスト用の高解像度画像群／e.g. shape = (画像数, 64, 64, 1)

2.2.21. toolbox.data_generator.DataGenerator クラス

低解像度・高解像度画像データセットを正規化し、ジェネレータ関数を定義する。

Attribute	Description
lr_imgs	低解像度画像群／e.g. shape = (画像数, 16, 16, 1)
hr_imgs	高解像度画像群／e.g. shape = (画像数, 64, 64, 1)
num_imgs	画像数
batch_size	バッチサイズ

__init__()

低解像度・高解像度画像群から nan を含むデータを除外し、属性を初期化する。

Argument	Description
x	低解像度画像群／e.g. shape = (画像数, 16, 16, 1)
y	高解像度画像群／e.g. shape = (画像数, 64, 64, 1)
norm_func	正規化関数
batch_size	バッチサイズ

sample()

指定枚数の低解像度・高解像度画像をサンプルする。

Argument	Description
num_samples	サンプルする画像枚数
replace	重複サンプリングの可否／default = False
indices	手動指定のインデックス／default = []

Return Value	Description
tuple[0]	低解像度画像群／e.g. shape = (num_samples, 16, 16, 1)
tuple[1]	高解像度画像群／e.g. shape = (num_samples, 64, 64, 1)

generator()

バッチサイズの低解像度・高解像度画像をサンプルする。

Return Value	Description
tuple[0]	低解像度画像群／e.g. shape = (batch_size, 16, 16, 1)
tuple[1]	高解像度画像群／e.g. shape = (batch_size, 64, 64, 1)

steps_per_epoch()

エポックあたりのバッチ学習回数を取得する。

Return Value	Description
-	ステップあたりのバッチ学習回数 (num_imgs // batch_size)

2.2.22. toolbox.normalizer.MinMaxNormalizer

データの要素を 0-1 の範囲に正規化する。

Attribute	Description
xmin	データ x の最小値
xmax	データ x の最大値
ymin	データ y の最小値
ymax	データ y の最大値

__init__()

min, max 属性を初期化する。

Argument	Description
xMin	データ x の最小値
xMax	データ x の最大値
ymin	データ y の最小値
ymax	データ y の最大値

normalize()

データの正規化関数を定義する。

Argument	Description
imgs	正規化前のデータ
min	データの最小値
max	データの最大値

Return Value	Description
-	正規化後のデータ

denormalize_x()

正規化関数の逆変換を定義する。

Argument	Description
imgs	正規化後のデータ

Return Value	Description
-	正規化前のデータ

denormalize_y()

正規化関数の逆変換を定義する。

Argument	Description
imgs	正規化後のデータ

Return Value	Description
-	正規化前のデータ

2.2.23. toolbox.evaluator.Evaluator クラス

低解像度画像から Bicubic 補間画像と超解像画像を生成し、高解像度画像と対比して可視化する。画像品質は、高解像度画像に対する PSNR と DSSIM を計算する。

Attribute	Description
X	評価関数計算用のプレースホルダー
Y	評価関数計算用のプレースホルダー
metric_ops	PSNR と DSSIM の演算オペレータ

__init__()

X, Y, metric_ops 属性を初期化する。

Argument	Description
img_shape	画像の形状／e.g. shape = (64, 64, 1)

evaluate()

テストデータに対して、低解像度画像・Bicubic 補間画像・超解像度画像・高解像度画像を対比して可視化し、PSNR と DSSIM を併記する。

Argument	Description
x_test	低解像度画像群／e.g. shape = (画像数, 16, 16, 1)

y_test	高解像度画像群／e.g. shape = (画像数, 64, 64, 1)
batch_size	バッチサイズ
denorm_func_x	正規化データ x の逆変換関数
denorm_func_y	正規化データ y の逆変換関数
save_fname	結果の保存ファイル名（None の場合、matplotlib.pyplot.show で可視化）

2.2.24. toolbox.plotter.Plotter クラス

matplotlib.pyplot を使用して、学習履歴とテスト結果を可視化する。

__setup_ax()

matplotlib.axes.Axes オブジェクトの表示設定を行う。

Argument	Description
ax	Matplotlib.axes.Axes オブジェクト
xlabel	横軸のラベル
ylabel	縦軸のラベル
xscale_log	横軸のログ表示の可否／default = False
yscale_log	縦軸のログ表示の可否／default = False
fontsize	ラベル等のフォントサイズ／default = 9
labelsize	目盛等のラベルサイズ／default = 8

__pair_plot()

学習履歴をペアでプロットする（loss と val_loss 等）。

Argument	Description
ax	Matplotlib.axes.Axes オブジェクト
x	横軸データ
y	縦軸データのリスト
labels	縦軸データに対応するラベルリスト
color	プロット色

plot_history_graph()

損失値・PSNR・DSSIM の履歴をプロットする。下図のとおり、教師データの性能（loss, psnr, dssim）と検証データの性能（val_loss, val_psnr, val_dssim）が 3 つのグラフにプロットされる。

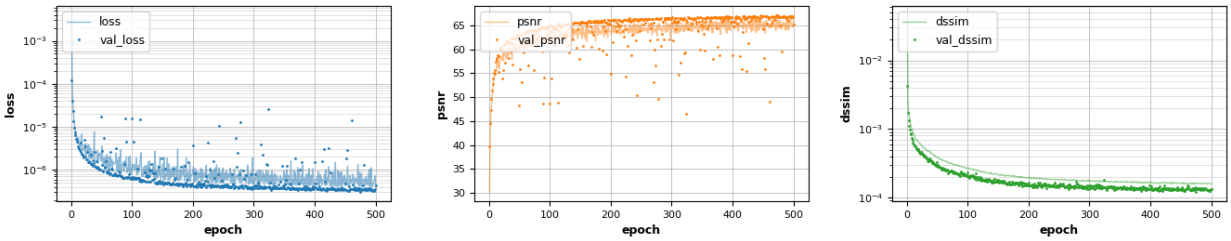

図 11 グラフプロット例

ただし、SRGAN モデルでは、下図のように loss, val_loss ではなく、Generator モデルの損失 g_loss, val_g_loss および Discriminator モデルの損失 d_loss, val_d_loss がプロットされる。

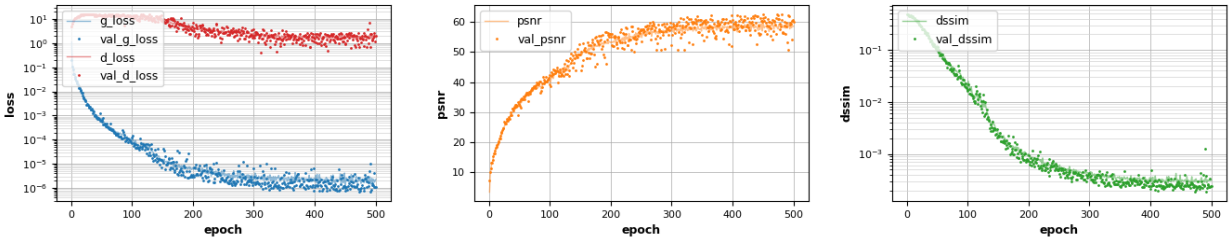

図 12 グラフプロット例(SRGAN)

Argument	Description
epochs	エポック数（横軸）
logs	全エポックの学習履歴の辞書／e.g. { 'loss'=[1e-2, 1e-3, ...], ... }
save_fname	結果の保存ファイル名（None の場合、matplotlib.pyplot.show で可視化）

plot_test_imgs()

テスト結果を可視化する。toolbox.evaluator.Evaluator クラスでは、下図のとおり、低解像度画像・Bicubic 補間画像・超解像画像・高解像度画像を可視化し、PSNR と DSSIM を併記する。

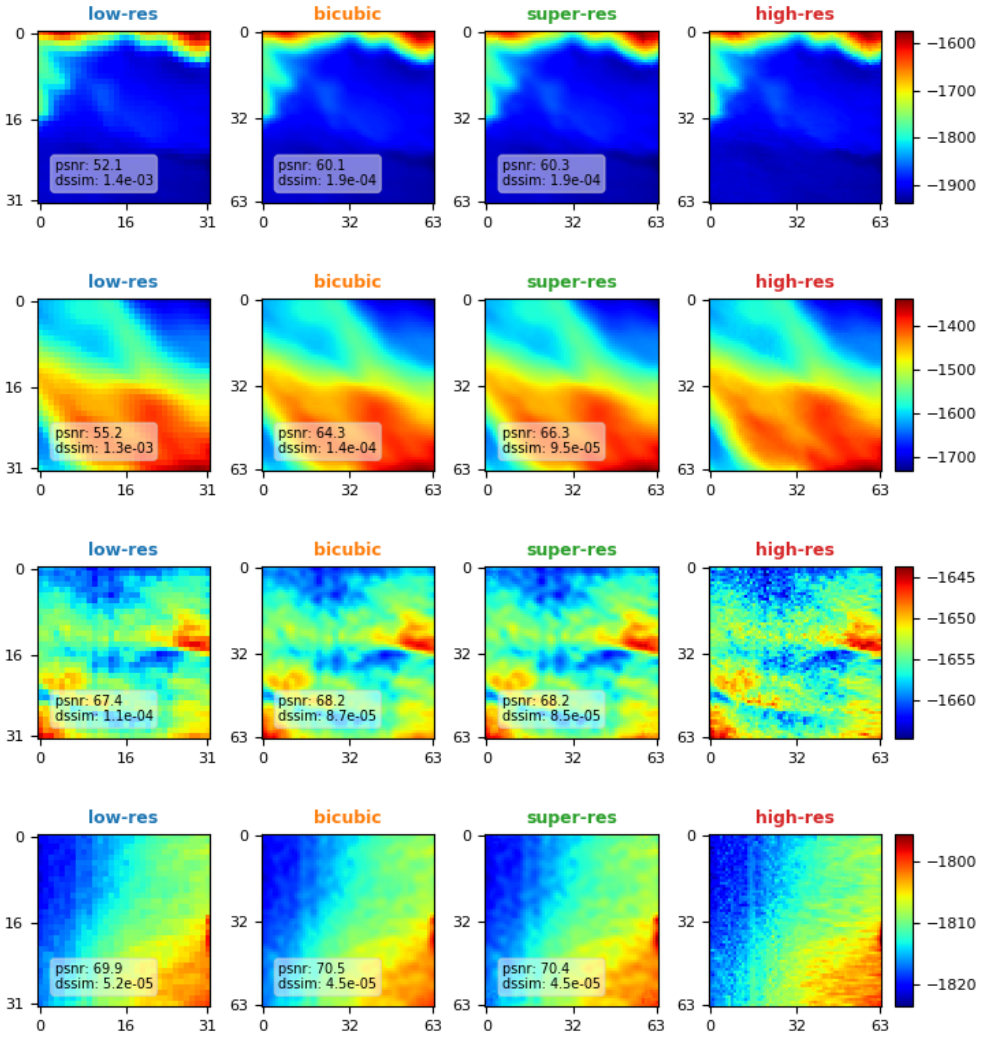

図 **13** テスト結果の可視化

Argument	Description
imgs	可視化画像／e.g. shape = (グループ, 画像数, 64, 64, 1)
labels	グループ名／e.g. = ['low-res', 'bicubic', 'super-res', 'high-res']
save_fname	結果の保存ファイル名（None の場合、matplotlib.pyplot.show で可視化）
figsize	ウィンドウサイズ（x 100px）／default = (8, 8)
fontsize	タイトル等フォントサイズ／default = 9
labelsize	目盛等ラベルサイズ／default = 8
textsize	PSNR, DSSIM 等テキストサイズ／default = 7

2.2.25. arg_parser.ArgParser クラス

コマンドライン引数の構文解析を行う。args 属性として、入力ファイル名 yml_fname、実行モード mode を定義する。

Attribute	Description
Args	コマンドライン引数

__init__()

入力ファイル名 yml_fname、実行モードを args 属性として初期化する。

get_fname()

入力ファイル名 yml_fname を取得する。

Return Value	Description
-	入力ファイル名

get_mode()

実行モード mode を取得する。

Return Value	Description
-	実行モード（'train' or 'test'）

2.2.26. Initializer.Initializer クラス

システムの初期化を行う。

tf_init()

TensorFlow の警告メッセージ・メモリ管理設定を行い、乱数シードを初期化する。

Argument	Description
log_level	TensorFlow の警告メッセージの表示レベル／default = '3'
seed	乱数シード（None の場合、シードは固定しない）

2.2.27. user_params.UserParams クラス

yml 形式の入力ファイルに記載されたユーザパラメータを読み込む。

Attribute	Description
input_shape	入力画像形状／e.g. = (16, 16, 1)
scale	超解像倍率／e.g. = 2
data_dir	train, validation, test ディレクトリを含むディレクトリ名
norm_range	正規化のためのデータ範囲／e.g. = (-2500, 0)
fname_model_weights	学習済みモデルの重みファイル名（test モード用）
batch_size	バッチサイズ
epochs	学習エポック数
model	超解像モデル（'srcnn', 'fsrcnn', 'espcn', 'srgan'）
loss	損失関数（'mae', 'mse', 'psnr', 'dssim', 'vgg'）
optimizer	最適化関数（'sgd', 'rmsprop', 'adam', etc.）
learning_rate	学習率／recommended = 1e-4
log_period	学習履歴の出力頻度
log_dir	Log 出力ディレクトリ名(nul の場合は yaml ファイルと同じ名前になる)
patience	early stopping 機能(null の場合は通常の動作になる)
srcnn	以下、srcnn の設定を行う
filters	フィルター数
kernels	カーネルサイズ
espcn	以下、espcn の設定を行う
filters	フィルター数
kernals	カーネルサイズ
esrgan	以下、esrgan の設定を行う
filters	Generator のフィルター数
num_residual_blocks	残差ブロック
d_filters	Discriminator のフィルター数
d_num_downsampling_blocks	Downsampling ブロック数
loss_weights	Generator と discriminator の loss weights

__init__()

yml 形式の入力ファイルに記載されたユーザパラメータを読み込む。ユーザパラメータは、__dict__に代入することにより、属性変数として初期化する。

Argument	Description
fname	yml 形式の入力ファイル名

2.2.28. modeler.Modeler クラス

user_params.UserParams クラスで与えられるユーザパラメータに基づき、超解像モデルを構築する。

Attribute	Description
u_params	user_params.UserParams クラスオブジェクト
models	超解像モデル辞書 { 'srcnn':SRCNN, 'fsrcnn':FSRCNN, ... }
custom_layers	toolbox.model.layer モジュールのカスタムレイヤー辞書
loss_funcs	損失関数辞書 { 'psnr':Loss.psnr, 'dssim':Loss.dssim, ... }
optimizers	最適化関数辞書 { 'sgd':keras.optimizers.SGD, ... }

__init__()

各種属性を初期化する。

Argument	Description
u_params	user_params.UserParams クラスオブジェクト

build()

超解像モデルを構築・コンパイルする。

Argument	Description
verbose	モデルレイヤーの表示(1)・非表示(0)／default = 1

Return Value	Description
-	超解像モデル

2.2.29. trainer.Trainer クラス

user_params.UserParams クラスで与えられるユーザパラメータに基づき、超解像モデルの学習を行う。

Attribute	Description
u_params	user_params.UserParams クラスオブジェクト
normalizer	データの初期化子
train_data_generator	toolbox.data_generator.DataGenerator クラスオブジェクト
validation_data_generator	toolbox.data_generator.DataGenerator クラスオブジェクト

__init__()

各種属性を初期化する。

Argument	Description
u_params	user_params.UserParams クラスオブジェクト

reset_weights()

モデルの重み（カーネル・バイアス）を初期化する

Argument	Description
model	超解像モデル

train()

超解像モデルの学習を実行する。

Argument	Description
model	超解像モデル

2.2.30. tester.Tester クラス

user_params.UserParams クラスで与えられるユーザパラメータに基づき、超解像モデルの性能テストを行う。

Attribute	Description
u_params	user_params.UserParams クラスオブジェクト
normalizer	データの初期化子
test_data_generator	toolbox.data_generator.DataGenerator クラスオブジェクト
evaluator	toolbox.evaluator.Evaluator クラスオブジェクト

__init__()

各種属性を初期化する。

Argument	Description
u_params	user_params.UserParams クラスオブジェクト

test()

超解像モデルの性能テストを実行する。

Argument	Description
model	超解像モデル
num_samples	テストで比較する画像のサンプル数／default = 4

2.2.31. Main クラス

超解像の学習・テストを実行する。

run()

超解像の学習・テストを次のフローで実行する。

(1) ArgParser クラスでコマンドライン引数を読み込む

(2) UserParams クラスを初期化する

(3) Initializer クラスで TensorFlow を初期化する

(4) 超解像モデルを生成する

(5) 学習・テストを実行する

2.3. 推奨動作環境

プログラムは、Python 3.8 の環境上で、以下のパッケージを利用して動作する。

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

詳細なパッケージ情報は、プログラムフォルダの requiremtns.txt に記載されている。

2.4. 使用方法

2.4.1. ユーザパラメータの設定

ユーザパラメータは、yml 形式のファイルに以下の内容を記述する。

```
# 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]
```

各パラメータの概略を下表に示す。

User Params	Description
input_shape	入力画像形状／e.g. = (16, 16, 1)
scale	超解像倍率／e.g. = 2
data_dir	train, validation, test ディレクトリを含むディレクトリ名
norm_range	正規化のためのデータ範囲／e.g. = (-2500, 0)
fname_model_weights	学習済みモデルの重みファイル名（test モード用）
batch_size	バッチサイズ
epochs	学習エポック数
model	超解像モデル（srcnn, fsrcnn, espcn, srgan, esrgan）
loss	損失関数（mae, mse, psnr, dssim, vgg）
optimizer	最適化関数（sgd, rmsprop, adam, etc.）
learning_rate	学習率／recommended = 1e-4
log_period	学習履歴の出力頻度
log_dir	Log 出力ディレクトリ名(nul の場合は yaml ファイルと同じ名前になる)
patience	early stopping 機能(null の場合は通常の動作になる)／e.g = 500
online_da	オンラインデータ拡張に関する設定を行う
valid	true or false. オンライン拡張を行うか
zoom_range	水平方向の拡大縮小倍率 [最小倍率、最大倍率]で指定
zoom_iso	true or false. 等方拡大縮小とするか
depth_scale_range	鉛直方向の拡大縮小倍率 [最小倍率、最大倍率]で指定
use_depth_scale	true or false 鉛直方向の拡大縮小を行うか
horizontal_flip	左右反転を行うか 今回は未使用
vertical_flip	上下反転を行うか 今回は未使用
rotation_range	ランダム回転範囲 0 の場合は回転なし 今回は未使用
steps_per_epoch_mag	エポックごとの学習ステップ数を何倍するか 学習データ数÷バッチサイズが基準ステップ数となる
srcnn	以下、srcnn の設定を行う
filters	フィルター数
kernels	カーネルサイズ
espcn	以下、espcn の設定を行う
filters	フィルター数
kernals	カーネルサイズ
esrgan	以下、esrgan の設定を行う
filters	Generator のフィルター数

num_residual_blocks	Generator の residual ブロック数
d_filters	Discriminator のフィルター数
d_num_downsampling_blocks	Discriminator の畳み込みブロック数
loss_weights	Generator と discriminator の loss weights の比

2.4.2. コマンドライン引数

実行用スクリプトである main.py は、入力ファイル名（.yml）と実行モード（train or test）をコマンドライン引数として受け取る。コマンドライン引数は、次のコマンドでヘルプから確認できる。

```
$ 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. モデルの学習

入力ファイル（input.yml）の fname_model_weights に None を指定し、以下のコマンドで学習を実行する。

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

学習が正常に実行されると、以下の超解像モデルのレイヤー構成と学習プログレスが表示される。

```
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
...
```

学習履歴は、入力ファイルの log_dir で指定したディレクトリ内に保存される。保存されるファイルを以下に示す（*はエポック数）

- history.csv: 全エポックの教師・検証データに対する損失・PSNR・DSSIM の一覧
- history_*.png: 各エポックの教師・検証データに対する損失・PSNR・DSSIM の履歴
- weights_best.h5: val_loss が最小となったエポックにおけるベストモデルの重み
- weights_*.h5: 各エポックのモデルの重み
- test_*.png: 各エポックの性能テスト結果

2.4.4. モデルのテスト

入力ファイル（input.yml）の fname_model_weights に学習済みモデルのパスを指定し（fname_model_weights: log/weights_best.h5)、以下のコマンドで学習を実行する。

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

test を実行する際に、入力データ(yaml の data_dir で指定されるデータ)の高解像度データに対して、bathymetric_map_feature.py による傾斜度情報 CSV が生成されていることが前提となる。
実行すると、test データに対して評価が実行され、ログディレクトリ内に test_result フォルダが新規に作成され、以下の項目が出力される。

- test_results.csv: 全テストケースに対する各指標の評価結果
- test_results_describe.csv: 全テストケースに対する評価結果のサマリ（最大、最小、平均など、pandas.DataFrame.describe()結果相当）
- test_resultsslope0-0.05_describe.csv: 傾斜度 0～0.05 のテストケースに対する評価結果のサマリ
- test_resultsslope0.05-0.1_describe.csv: 傾斜度 0.05～0.1 のテストケースに対する評価結果のサマリ
- test_resultsslope0.1-_describe.csv: 傾斜度 0.1～のテストケースに対する評価結果のサマリ
- data_SR.pkl: 全テストデータに対する予測結果
- test_result_N-M.png テストデータ N～M に対する予測の可視化結果 4 データごとに 1 画像として出力

以下に test_result_N-M.png の例を示す。

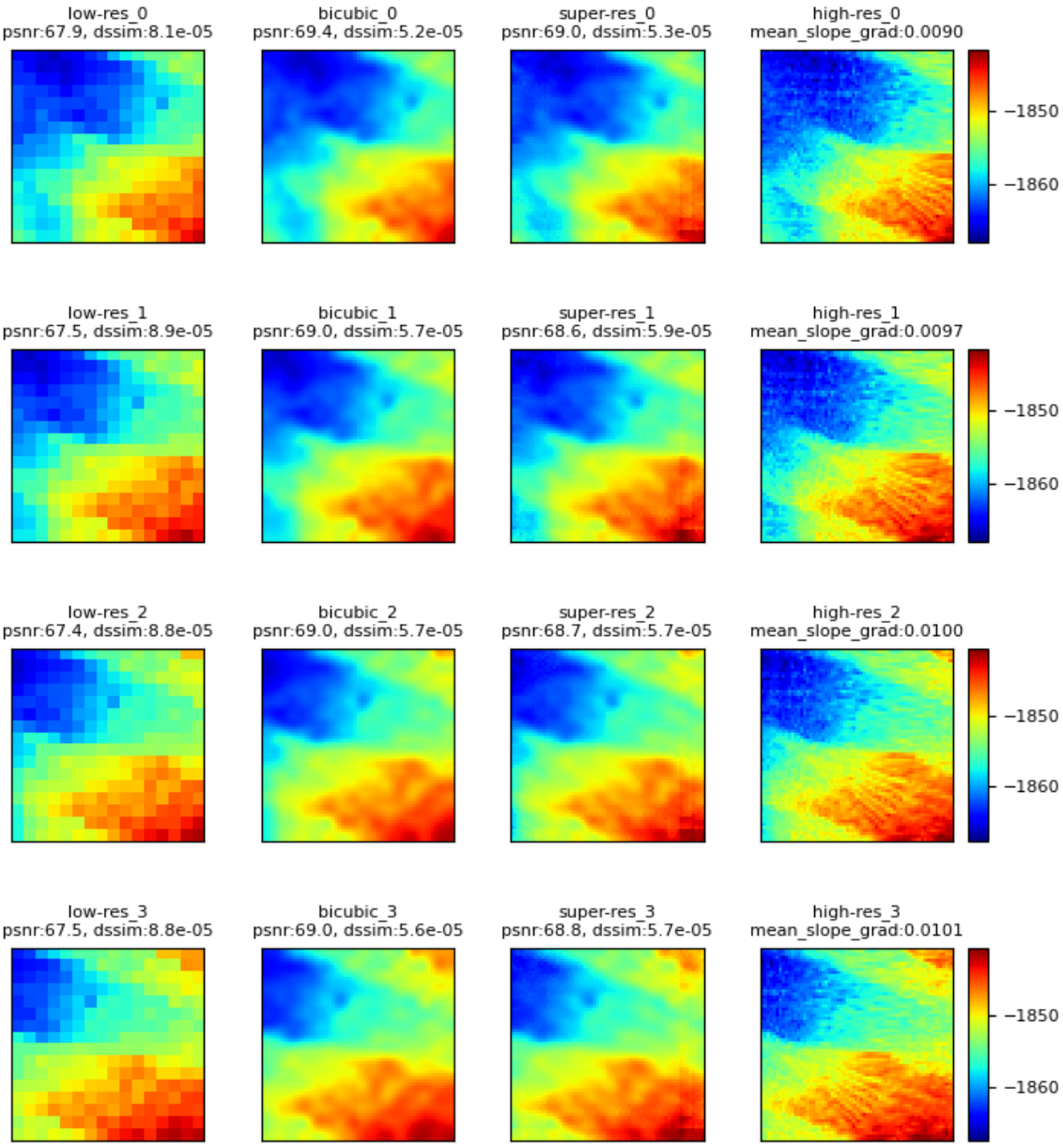

図 14 **test_result_N-M.png** の例

3. 引用文献

[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 , H. Kaiming, “Image Super-Resolution Using Deep Convolutional Networks,” arXiv: 1501.00092 [cs.CV], 2014.

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

[4] 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.

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

[6] 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.

[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.