Segmentation Models Documentation

Release 0.1.2

Pavel Yakubovskiy

Aug 02, 2019

Contents:

1 Installation **1**

2 Tutorial **3**
- 2.1 Quick start . . . 3
- 2.2 Simple training pipeline . . . 4
- 2.3 Models and Backbones . . . 4
- 2.4 Fine tuning . . . 5
- 2.5 Training with non-RGB data . . . 5

3 Segmentation Models Python API **7**
- 3.1 Unet . . . 7
- 3.2 Linknet . . . 8
- 3.3 FPN . . . 9
- 3.4 PSPNet . . . 9
- 3.5 metrics . . . 10
- 3.6 losses . . . 11
- 3.7 utils . . . 12

4 Support **13**

5 Indices and tables **15**

Index **17**

CHAPTER 1

Installation

Requirements

1) Python 3.5+
2) Keras >= 2.2.0
3) Keras Applications >= 1.7.0
4) Image Classifiers == 0.2.0
5) Tensorflow 1.9 (tested)

Note: This library does not have Tensorflow in a requirements.txt for installation. Please, choose suitable version ('cpu'/'gpu') and install it manually using official Guide.

Pip package

```
$ pip install segmentation-models
```

Latest version

```
$ pip install git+https://github.com/qubvel/segmentation_models
```

CHAPTER 2

Tutorial

Segmentation models is python library with Neural Networks for Image Segmentation based on Keras (Tensorflow) framework.

The main features of this library are:

- High level API (just two lines to create NN)
- **4** models architectures for binary and multi class segmentation (including legendary **Unet**)
- **25** available backbones for each architecture
- All backbones have **pre-trained** weights for faster and better convergence

2.1 Quick start

Since the library is built on the Keras framework, created segmentaion model is just a Keras Model, which can be created as easy as:

```
from segmentation_models import Unet

model = Unet()
```

Depending on the task, you can change the network architecture by choosing backbones with fewer or more parameters and use pretrainded weights to initialize it:

```
model = Unet('resnet34', encoder_weights='imagenet')
```

Change number of output classes in the model:

```
model = Unet('resnet34', classes=3, activation='softmax')
```

Change input shape of the model:

```
model = Unet('resnet34', input_shape=(None, None, 6), encoder_weights=None)
```

2.2 Simple training pipeline

```
from segmentation_models import Unet
from segmentation_models.backbones import get_preprocessing
from segmentation_models.losses import bce_jaccard_loss
from segmentation_models.metrics import iou_score

BACKBONE = 'resnet34'
preprocess_input = get_prepocessing(BACKBONE)

# load your data
x_train, y_train, x_val, y_val = load_data(...)

# preprocess input
x_train = preprocess_input(x_train)
x_val = preprocess_input(x_val)

# define model
model = Unet(BACKBONE, encoder_weights='imagenet')
model.complile('Adam', loss=bce_jaccard_loss, metrics=[iou_score])

# fit model
model.fit(
    x=x_train,
    y=y_train,
    batch_size=16,
    epochs=100,
    validation_data=(x_val, y_val),
)
```

Same manimulations can be done with `Linknet`, `PSPNet` and `FPN`. For more detailed information about models API and use cases Read the Docs.

2.3 Models and Backbones

Models

- Unet
- FPN
- Linknet
- PSPNet

Unet	Linknet

PSPNet	FPN

Backbones

Type	Names
VGG	`'vgg16' 'vgg19'`
ResNet	`'resnet18' 'resnet34' 'resnet50' 'resnet101' 'resnet152'`
SE-ResNet	`'seresnet18' 'seresnet34' 'seresnet50' 'seresnet101' 'seresnet152'`
ResNeXt	`'resnext50' 'resnet101'`
SE-ResNeXt	`'seresnext50' 'seresnet101'`
SENet154	`'senet154'`
DenseNet	`'densenet121' 'densenet169' 'densenet201'`
Inception	`'inceptionv3' 'inceptionresnetv2'`
MobileNet	`'mobilenet' 'mobilenetv2'`

All backbones have weights trained on 2012 ILSVRC ImageNet dataset (`encoder_weights='imagenet'`).

2.4 Fine tuning

Some times, it is useful to train only randomly initialized *decoder* in order not to damage weights of properly trained *encoder* with huge gradients during first steps of training. In this case, all you need is just pass `freeze_encoder = True` argument while initializing the model.

```
from segmentation_models import Unet
from segmentation_models.utils import set_trainable

model = Unet(backbone_name='resnet34', encoder_weights='imagenet', freeze_
↪encoder=True)
model.compile('Adam', 'binary_crossentropy', ['binary_accuracy'])

# pretrain model decoder
model.fit(x, y, epochs=2)

# release all layers for training
set_trainable(model) # set all layers trainable and recompile model

# continue training
model.fit(x, y, epochs=100)
```

2.5 Training with non-RGB data

In case you have non RGB images (e.g. grayscale or some medical/remote sensing data) you have few different options:

1. Train network from scratch with randomly initialized weights

```
from segmentation_models import Unet

# read/scale/preprocess data
x, y = ...
```

(continues on next page)

(continued from previous page)

```
# define number of channels
N = x.shape[-1]

# define model
model = Unet(backbone_name='resnet34', encoder_weights=None, input_shape=(None, None,␣
↪N))

# continue with usual steps: compile, fit, etc..
```

2. Add extra convolution layer to map `N -> 3` channels data and train with pretrained weights

```
from segmentation_models import Unet
from keras.layers import Input, Conv2D
from keras.models import Model

# read/scale/preprocess data
x, y = ...

# define number of channels
N = x.shape[-1]

base_model = Unet(backbone_name='resnet34', encoder_weights='imagenet')

inp = Input(shape=(None, None, N))
l1 = Conv2D(3, (1, 1))(inp) # map N channels data to 3 channels
out = base_model(l1)

model = Model(inp, out, name=base_model.name)

# continue with usual steps: compile, fit, etc..
```

CHAPTER 3

Segmentation Models Python API

Getting started with segmentation models is easy.

3.1 Unet

segmentation_models.**Unet**(*backbone_name='vgg16'*, *input_shape=(None*, *None*, *3)*, *classes=1*, *activation='sigmoid'*, *encoder_weights='imagenet'*, *encoder_freeze=False*, *encoder_features='default'*, *decoder_block_type='upsampling'*, *decoder_filters=(256*, *128*, *64*, *32*, *16)*, *decoder_use_batchnorm=True*, ***kwargs*)

Unet is a fully convolution neural network for image semantic segmentation

Parameters

- **backbone_name** – name of classification model (without last dense layers) used as feature extractor to build segmentation model.
- **input_shape** – shape of input data/image `(H, W, C)`, in general case you do not need to set `H` and `W` shapes, just pass `(None, None, C)` to make your model be able to process images af any size, but `H` and `W` of input images should be divisible by factor `32`.
- **classes** – a number of classes for output (output shape - `(h, w, classes)`).
- **activation** – name of one of `keras.activations` for last model layer (e.g. `sigmoid`, `softmax`, `linear`).
- **encoder_weights** – one of `None` (random initialization), `imagenet` (pre-training on ImageNet).
- **encoder_freeze** – if `True` set all layers of encoder (backbone model) as non-trainable.
- **encoder_features** – a list of layer numbers or names starting from top of the model. Each of these layers will be concatenated with corresponding decoder block. If `default` is used layer names are taken from `DEFAULT_SKIP_CONNECTIONS`.
- **decoder_block_type** – one of blocks with following layers structure:

- *upsampling*: `Upsampling2D` -> `Conv2D` -> `Conv2D`
- *transpose*: `Transpose2D` -> `Conv2D`

- **decoder_filters** – list of numbers of `Conv2D` layer filters in decoder blocks
- **decoder_use_batchnorm** – if `True`, `BatchNormalisation` layer between `Conv2D` and `Activation` layers is used.

Returns Unet

Return type `keras.models.Model`

3.2 Linknet

`segmentation_models.`**Linknet**(*backbone_name='vgg16'*, *input_shape=(None, None, 3)*, *classes=1*, *activation='sigmoid'*, *encoder_weights='imagenet'*, *encoder_freeze=False*, *encoder_features='default'*, *decoder_filters=(None, None, None, None, 16)*, *decoder_use_batchnorm=True*, *decoder_block_type='upsampling'*, ***kwargs*)

Linknet is a fully convolution neural network for fast image semantic segmentation

Note: This implementation by default has 4 skip connections (original - 3).

Parameters

- **backbone_name** – name of classification model (without last dense layers) used as feature extractor to build segmentation model.
- **input_shape** – shape of input data/image `(H, W, C)`, in general case you do not need to set `H` and `W` shapes, just pass `(None, None, C)` to make your model be able to process images af any size, but `H` and `W` of input images should be divisible by factor `32`.
- **classes** – a number of classes for output (output shape - `(h, w, classes)`).
- **activation** – name of one of `keras.activations` for last model layer (e.g. `sigmoid`, `softmax`, `linear`).
- **encoder_weights** – one of `None` (random initialization), `imagenet` (pre-training on ImageNet).
- **encoder_freeze** – if `True` set all layers of encoder (backbone model) as non-trainable.
- **encoder_features** – a list of layer numbers or names starting from top of the model. Each of these layers will be concatenated with corresponding decoder block. If `default` is used layer names are taken from `DEFAULT_SKIP_CONNECTIONS`.
- **decoder_filters** – list of numbers of `Conv2D` layer filters in decoder blocks, for block with skip connection a number of filters is equal to number of filters in corresponding encoder block (estimates automatically and can be passed as `None` value).
- **decoder_use_batchnorm** – if `True`, `BatchNormalisation` layer between `Conv2D` and `Activation` layers is used.
- **decoder_block_type** – one of - *upsampling*: use `Upsampling2D` keras layer - *transpose*: use `Transpose2D` keras layer

Returns Linknet

Return type `keras.models.Model`

3.3 FPN

`segmentation_models.`**`FPN`**(*backbone_name='vgg16'*, *input_shape=(None, None, 3)*, *input_tensor=None*, *classes=21*, *activation='softmax'*, *encoder_weights='imagenet'*, *encoder_freeze=False*, *encoder_features='default'*, *pyramid_block_filters=256*, *pyramid_use_batchnorm=True*, *pyramid_dropout=None*, *final_interpolation='bilinear'*, ***kwargs*)

FPN is a fully convolution neural network for image semantic segmentation

Parameters

- **`backbone_name`** – name of classification model (without last dense layers) used as feature extractor to build segmentation model.
- **`input_shape`** – shape of input data/image `(H, W, C)`, in general case you do not need to set `H` and `W` shapes, just pass `(None, None, C)` to make your model be able to process images af any size, but `H` and `W` of input images should be divisible by factor `32`.
- **`input_tensor`** – optional Keras tensor (i.e. output of *layers.Input()*) to use as image input for the model (works only if `encoder_weights` is `None`).
- **`classes`** – a number of classes for output (output shape - `(h, w, classes)`).
- **`activation`** – name of one of `keras.activations` for last model layer (e.g. `sigmoid`, `softmax`, `linear`).
- **`encoder_weights`** – one of `None` (random initialization), `imagenet` (pre-training on ImageNet).
- **`encoder_freeze`** – if `True` set all layers of encoder (backbone model) as non-trainable.
- **`encoder_features`** – a list of layer numbers or names starting from top of the model. Each of these layers will be used to build features pyramid. If `default` is used layer names are taken from `DEFAULT_FEATURE_PYRAMID_LAYERS`.
- **`pyramid_block_filters`** – a number of filters in Feature Pyramid Block of FPN.
- **`pyramid_use_batchnorm`** – if `True`, `BatchNormalisation` layer between `Conv2D` and `Activation` layers is used.
- **`pyramid_dropout`** – spatial dropout rate for feature pyramid in range (0, 1).
- **`final_interpolation`** – interpolation type for upsampling layers, on of `nearest`, `bilinear`.

Returns FPN

Return type `keras.models.Model`

3.4 PSPNet

`segmentation_models.`**`PSPNet`**(*backbone_name='vgg16'*, *input_shape=(384, 384, 3)*, *classes=21*, *activation='softmax'*, *encoder_weights='imagenet'*, *encoder_freeze=False*, *downsample_factor=8*, *psp_conv_filters=512*, *psp_pooling_type='avg'*, *psp_use_batchnorm=True*, *psp_dropout=None*, *final_interpolation='bilinear'*, ***kwargs*)

PSPNet is a fully convolution neural network for image semantic segmentation

Parameters

- **backbone_name** – name of classification model used as feature extractor to build segmentation model.
- **input_shape** – shape of input data/image `(H, W, C)`. `H` and `W` should be divisible by `6 * downsample_factor` and **NOT** `None`!
- **classes** – a number of classes for output (output shape - `(h, w, classes)`).
- **activation** – name of one of `keras.activations` for last model layer (e.g. `sigmoid`, `softmax`, `linear`).
- **encoder_weights** – one of `None` (random initialization), `imagenet` (pre-training on ImageNet).
- **encoder_freeze** – if `True` set all layers of encoder (backbone model) as non-trainable.
- **downsample_factor** – one of 4, 8 and 16. Downsampling rate or in other words backbone depth to construct PSP module on it.
- **psp_conv_filters** – number of filters in `Conv2D` layer in each PSP block.
- **psp_pooling_type** – one of 'avg', 'max'. PSP block pooling type (maximum or average).
- **psp_use_batchnorm** – if `True`, `BatchNormalisation` layer between `Conv2D` and `Activation` layers is used.
- **psp_dropout** – dropout rate between 0 and 1.
- **final_interpolation** – `duc` or `bilinear` - interpolation type for final upsampling layer.

Returns PSPNet

Return type `keras.models.Model`

3.5 metrics

`segmentation_models.metrics.`**iou_score**(*gt*, *pr*, *class_weights=1.0*, *smooth=1.0*, *per_image=True*, *threshold=None*)

The Jaccard index, also known as Intersection over Union and the Jaccard similarity coefficient (originally coined coefficient de communauté by Paul Jaccard), is a statistic used for comparing the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A, B) = \frac{A \cap B}{A \cup B}$$

Parameters

- **gt** – ground truth 4D keras tensor (B, H, W, C)
- **pr** – prediction 4D keras tensor (B, H, W, C)
- **class_weights** –
  1. or list of class weights, len(weights) = C
- **smooth** – value to avoid division by zero

- **per_image** – if `True`, metric is calculated as mean over images in batch (B), else over whole batch
- **threshold** – value to round predictions (use > comparison), if `None` prediction prediction will not be round

Returns IoU/Jaccard score in range [0, 1]

`segmentation_models.metrics.`**f_score**(*gt*, *pr*, *class_weights=1*, *beta=1*, *smooth=1.0*, *per_image=True*, *threshold=None*)

The F-score (Dice coefficient) can be interpreted as a weighted average of the precision and recall, where an F-score reaches its best value at 1 and worst score at 0. The relative contribution of `precision` and `recall` to the F1-score are equal. The formula for the F score is:

$$F_\beta(precision, recall) = (1 + \beta^2)\frac{precision \cdot recall}{\beta^2 \cdot precision + recall}$$

The formula in terms of *Type I* and *Type II* errors:

$$F_\beta(A, B) = \frac{(1 + \beta^2)TP}{(1 + \beta^2)TP + \beta^2 FN + FP}$$

where: TP - true positive; FP - false positive; FN - false negative;

Parameters

- **gt** – ground truth 4D keras tensor (B, H, W, C)
- **pr** – prediction 4D keras tensor (B, H, W, C)
- **class_weights** –
  1. or list of class weights, len(weights) = C
- **beta** – f-score coefficient
- **smooth** – value to avoid division by zero
- **per_image** – if `True`, metric is calculated as mean over images in batch (B), else over whole batch
- **threshold** – value to round predictions (use > comparison), if `None` prediction prediction will not be round

Returns F-score in range [0, 1]

3.6 losses

`segmentation_models.losses.`**jaccard_loss**(*gt*, *pr*, *class_weights=1.0*, *smooth=1.0*, *per_image=True*)

Jaccard loss function for imbalanced datasets:

$$L(A, B) = 1 - \frac{A \cap B}{A \cup B}$$

Parameters

- **gt** – ground truth 4D keras tensor (B, H, W, C)
- **pr** – prediction 4D keras tensor (B, H, W, C)
- **class_weights** –

1. or list of class weights, len(weights) = C

- **smooth** – value to avoid division by zero
- **per_image** – if `True`, metric is calculated as mean over images in batch (B), else over whole batch

Returns Jaccard loss in range [0, 1]

segmentation_models.losses.**dice_loss**(*gt*, *pr*, *class_weights=1.0*, *smooth=1.0*, *per_image=True*, *beta=1.0*)

Dice loss function for imbalanced datasets:

$$L(precision, recall) = 1 - (1 + \beta^2) \frac{precision \cdot recall}{\beta^2 \cdot precision + recall}$$

Parameters

- **gt** – ground truth 4D keras tensor (B, H, W, C)
- **pr** – prediction 4D keras tensor (B, H, W, C)
- **class_weights** –

1. or list of class weights, len(weights) = C

- **smooth** – value to avoid division by zero
- **per_image** – if `True`, metric is calculated as mean over images in batch (B), else over whole batch
- **beta** – coefficient for precision recall balance

Returns Dice loss in range [0, 1]

3.7 utils

segmentation_models.backbones.**get_preprocessing**(*name*)

segmentation_models.utils.**set_trainable**(*model*)

Set all layers of model trainable and recompile it

Note: Model is recompiled using same optimizer, loss and metrics:

```
model.compile(model.optimizer, model.loss, model.metrics)
```

Parameters **model** (`keras.models.Model`) – instance of keras model

CHAPTER 4

Support

The easiest way to get help with the project is to create issue or PR on github.

Github: http://github.com/qubvel/segmentation_models/issues

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index

D

dice_loss() (*in module segmentation_models.losses*), 12

F

f_score() (*in module segmentation_models.metrics*), 11

FPN() (*in module segmentation_models*), 9

G

get_preprocessing() (*in module segmentation_models.backbones*), 12

I

iou_score() (*in module segmentation_models.metrics*), 10

J

jaccard_loss() (*in module segmentation_models.losses*), 11

L

Linknet() (*in module segmentation_models*), 8

P

PSPNet() (*in module segmentation_models*), 9

S

set_trainable() (*in module segmentation_models.utils*), 12

U

Unet() (*in module segmentation_models*), 7