loss combinations

bce_dice_loss = binary_crossentropy + dice_loss bce_jaccard_loss = binary_crossentropy + jaccard_loss

cce_dice_loss = categorical_crossentropy + dice_loss cce_jaccard_loss = categorical_crossentropy + jaccard_loss

binary_focal_dice_loss = binary_focal_loss + dice_loss binary_focal_jaccard_loss = binary_focal_loss + jaccard_loss

categorical_focal_dice_loss = categorical_focal_loss + dice_loss categorical_focal_jaccard_loss = categorical_focal_loss + jaccard_loss

==========

Scores

iou_score = IOUScore() f1_score = FScore(beta=1) f2_score = FScore(beta=2) precision = Precision() recall = Recall()

Tricks

```
# Loading trained model
base_model = load_model('path/to/trained/model.h5')

# Modify base model (change number of output classes)
x = base_model.layers[-2].output  # remove activation and last conv layer
x = keras.layers.Conv2D(n_classes, (1, 1))(x)
output = keras.layers.Activation(<activation_name>)(x)

new_model = keras.models.Model(base_model.input, output)
```

,

```
import segmentation_models as sm
from segmentation_models import Unet
from segmentation_models import get_preprocessing
from segmentation_models.losses import bce_jaccard_loss
from segmentation_models.metrics import iou_score

import keras
# or from tensorflow import keras

keras.backend.set_image_data_format('channels_last')
# or keras.backend.set_image_data_format('channels_first')
model = sm.Unet()
model = Unet('resnet34', encoder_weights='imagenet')
model = Unet('resnet34', classes=3, activation='softmax')
```

```
#==============================

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..
```

,

```
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 = sm.Unet('resnet34', encoder_weights='imagenet')

Change number of output classes in the model (choose your case):

# binary segmentation (this parameters are default when you call Unet('resnet34')
model = sm.Unet('resnet34', classes=1, activation='sigmoid')

# multiclass segmentation with non overlapping class masks (your classes +
background)
model = sm.Unet('resnet34', classes=3, activation='softmax')

# multiclass segmentation with independent overlapping/non-overlapping class masks
model = sm.Unet('resnet34', classes=3, activation='sigmoid')

Change input shape of the model:

# if you set input channels not equal to 3, you have to set encoder_weights=None
# how to handle such case with encoder_weights='imagenet' described in docs
model = Unet('resnet34', input_shape=(None, None, 6), encoder_weights=None)

Simple training pipeline

import segmentation_models as sm
```

```
BACKBONE = 'resnet34'
preprocess_input = sm.get_preprocessing(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 = sm.Unet(BACKBONE, encoder_weights='imagenet')
model.compile(
    'Adam',
    loss=sm.losses.bce_jaccard_loss,
    metrics=[sm.metrics.iou_score],
)

# fit model
# if you use data generator use model.fit_generator(...) instead of model.fit(...)
# more about `fit_generator` here:
https://keras.io/models/sequential/#fit_generator
model.fit(
   x=x_train,
   y=y_train,
   batch_size=16,
   epochs=100,
   validation_data=(x_val, y_val),
)
```

Example

```
dice_loss = sm.losses.DiceLoss()
focal_loss = sm.losses.BinaryFocalLoss()
total_loss = dice_loss + (1 * focal_loss)

opt = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999)

metrics = [sm.metrics.IOUScore(threshold=0.5), sm.metrics.FScore(threshold=0.5)]

model = sm.PSPNet(BACKBONE, encoder_weights = 'imagenet', classes = 1,
encoder_freeze=False, activation='sigmoid', downsample_factor=16, input_shape=
(960,960,3),
psp_conv_filters=1024, psp_pooling_type='avg')
```