import keras.backend as K

import tensorflow as tf

from keras.layers import \*

**def resize\_images\_bilinear(X, height\_factor=1, width\_factor=1, target\_height=None, target\_width=None, data\_format='default'):**

'''

Resize the images contained in a 4D tensor of shape

- [batch, channels, height, width] (for 'channels\_first' data\_format)

- [batch, height, width, channels] (for 'channels\_last' data\_format)

by a factor of (height\_factor, width\_factor). Both factors should be

positive integers.

'''

if data\_format == 'default':

data\_format = K.image\_data\_format()

if data\_format == 'channels\_first':

original\_shape = K.int\_shape(X)

if target\_height and target\_width:

new\_shape = tf.constant(np.array((target\_height, target\_width)).astype('int32'))

else:

new\_shape = tf.shape(X)[2:]

new\_shape \*= tf.constant(np.array([height\_factor, width\_factor]).astype('int32'))

X = permute\_dimensions(X, [0, 2, 3, 1])

X = tf.image.resize\_bilinear(X, new\_shape)

X = permute\_dimensions(X, [0, 3, 1, 2])

if target\_height and target\_width:

X.set\_shape((None, None, target\_height, target\_width))

else:

X.set\_shape((None, None, original\_shape[2] \* height\_factor, original\_shape[3] \* width\_factor))

return X

elif data\_format == 'channels\_last':

original\_shape = K.int\_shape(X)

if target\_height and target\_width:

new\_shape = tf.constant(np.array((target\_height, target\_width)).astype('int32'))

else:

new\_shape = tf.shape(X)[1:3]

new\_shape \*= tf.constant(np.array([height\_factor, width\_factor]).astype('int32'))

X = tf.image.resize\_bilinear(X, new\_shape)

if target\_height and target\_width:

X.set\_shape((None, target\_height, target\_width, None))

else:

X.set\_shape((None, original\_shape[1] \* height\_factor, original\_shape[2] \* width\_factor, None))

return X

else:

raise Exception('Invalid data\_format: ' + data\_format)

**class BilinearUpSampling2D(Layer):**

def \_\_init\_\_(self, size=(1, 1), target\_size=None, data\_format='default', \*\*kwargs):

if data\_format == 'default':

data\_format = K.image\_data\_format()

self.size = tuple(size)

if target\_size is not None:

self.target\_size = tuple(target\_size)

else:

self.target\_size = None

assert data\_format in {'channels\_last', 'channels\_first'}, 'data\_format must be in {tf, th}'

self.data\_format = data\_format

self.input\_spec = [InputSpec(ndim=4)]

super(BilinearUpSampling2D, self).\_\_init\_\_(\*\*kwargs)

def compute\_output\_shape(self, input\_shape):

if self.data\_format == 'channels\_first':

width = int(self.size[0] \* input\_shape[2] if input\_shape[2] is not None else None)

height = int(self.size[1] \* input\_shape[3] if input\_shape[3] is not None else None)

if self.target\_size is not None:

width = self.target\_size[0]

height = self.target\_size[1]

return (input\_shape[0],

input\_shape[1],

width,

height)

elif self.data\_format == 'channels\_last':

width = int(self.size[0] \* input\_shape[1] if input\_shape[1] is not None else None)

height = int(self.size[1] \* input\_shape[2] if input\_shape[2] is not None else None)

if self.target\_size is not None:

width = self.target\_size[0]

height = self.target\_size[1]

return (input\_shape[0],

width,

height,

input\_shape[3])

else:

raise Exception('Invalid data\_format: ' + self.data\_format)

def call(self, x, mask=None):

if self.target\_size is not None:

return resize\_images\_bilinear(x, target\_height=self.target\_size[0], target\_width=self.target\_size[1],

data\_format=self.data\_format)

else:

return resize\_images\_bilinear(x, height\_factor=self.size[0], width\_factor=self.size[1], data\_format=self.data\_format)

def get\_config(self):

config = {'size': self.size, 'target\_size': self.target\_size}

base\_config = super(BilinearUpSampling2D, self).get\_config()

return dict(list(base\_config.items()) + list(config.items()))