import tensorflow as tf

from keras.utils import tf\_utils

from keras.utils import losses\_utils

from keras import backend

def inv\_kl\_divergence(y\_true, y\_pred):

y\_pred = tf.convert\_to\_tensor(y\_pred)

y\_true = tf.cast(y\_true, y\_pred.dtype)

y\_true = backend.clip(y\_true, backend.epsilon(), 1)

y\_pred = backend.clip(y\_pred, backend.epsilon(), 1)

return tf.reduce\_sum(y\_pred \* tf.math.log(y\_pred / y\_true), axis=-1)

def masked\_bce(y\_true, y\_pred):

y\_true = tf.cast(y\_true, dtype=tf.float32)

mask = y\_true != -1

return tf.keras.metrics.binary\_crossentropy(tf.boolean\_mask(y\_true, mask),

tf.boolean\_mask(y\_pred, mask))

class LossFunctionWrapper(tf.keras.losses.Loss):

def \_\_init\_\_(self,

fn,

reduction=losses\_utils.ReductionV2.AUTO,

name=None,

\*\*kwargs):

super().\_\_init\_\_(reduction=reduction, name=name)

self.fn = fn

self.\_fn\_kwargs = kwargs

def call(self, y\_true, y\_pred):

if tf.is\_tensor(y\_pred) and tf.is\_tensor(y\_true):

y\_pred, y\_true = losses\_utils.squeeze\_or\_expand\_dimensions(y\_pred, y\_true)

ag\_fn = tf.\_\_internal\_\_.autograph.tf\_convert(self.fn, tf.\_\_internal\_\_.autograph.control\_status\_ctx())

return ag\_fn(y\_true, y\_pred, \*\*self.\_fn\_kwargs)

def get\_config(self):

config = {}

for k, v in self.\_fn\_kwargs.items():

config[k] = backend.eval(v) if tf\_utils.is\_tensor\_or\_variable(v) else v

base\_config = super().get\_config()

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

class InvKLD(LossFunctionWrapper):

def \_\_init\_\_(self,

reduction=losses\_utils.ReductionV2.AUTO,

name='inv\_kl\_divergence'):

super().\_\_init\_\_(inv\_kl\_divergence, name=name, reduction=reduction)

class MaskedBCE(LossFunctionWrapper):

def \_\_init\_\_(self,

reduction=losses\_utils.ReductionV2.AUTO,

name='masked\_bce'):

super().\_\_init\_\_(masked\_bce, name=name, reduction=reduction)