"""

Module containing ClemNet.

"""

from typing import Any

from .layers import ChannelWiseDense, KerasConv1D, ResidualLayer

from .params import BlockParams, ClemNetParams

import tensorflow as tf

import tensorflow.compat.v1 as tf1

class Block2(tf.keras.layers.Layer):

"""

Possible ClemNet block. Architecture is as follow:

Optional(DenseLayer + BN + Act)

Optional(ConvLayer + BN + Act)

Optional(Residual Layer)

"""

def \_\_init\_\_(self, params: BlockParams, \*\*kwargs: Any):

super(Block2, self).\_\_init\_\_(\*\*kwargs)

self.params = params

def build(self, input\_shape: tf.TensorShape) -> None:

assert (

len(input\_shape) == 3

), f"Tensor shape must be of length 3. Passed tensor of shape {input\_shape}."

def call(self, inputs: tf.Tensor, training: bool) -> tf.Tensor:

x = inputs

if self.params.dense:

x = ChannelWiseDense(\*\*self.params.dense.dict())(inputs=x, training=training)

x = tf1.layers.batch\_normalization(x, momentum=0.9999, training=training, axis=1)

x = tf.keras.layers.Activation(self.params.activation)(x)

if self.params.conv:

x = KerasConv1D(\*\*self.params.conv.dict())(inputs=x, training=training)

x = tf1.layers.batch\_normalization(x, momentum=0.9999, training=training, axis=1)

x = tf.keras.layers.Activation(self.params.activation)(x)

if self.params.residual:

x = ResidualLayer()(inputs=inputs, residual=x)

return x

class ClemNet(tf.keras.layers.Layer):

"""

A residual network stacking residual blocks composed of dense layers and convolutions.

"""

def \_\_init\_\_(self, params: ClemNetParams, \*\*kwargs: Any):

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

self.params = params

def build(self, input\_shape: tf.TensorShape) -> None:

assert len(input\_shape) in (

2,

3,

), f"Tensor shape must be of length 3. Passed tensor of shape {input\_shape}."

def call(self, inputs: tf.Tensor, training: bool) -> tf.Tensor:

if len(inputs.shape) < 3:

inputs = tf.expand\_dims(inputs, axis=-1)

x = inputs

for block\_params in self.params.blocks:

x = Block2(block\_params)(inputs=x, training=training)

x = tf.keras.layers.Flatten(name="flattened")(x)

if self.params.top:

x = tf.keras.layers.Dense(units=self.params.top.n\_labels, name="logits")(x)

return x