```
import os
os.environ['TF_ENABLE_ONEDNN_
OPTS'] = '0'
import tensorflow as tf
import numpy as np
print(tf.__version__)
for _ in range(20):
    try:
        with tf.device("GPU:0"):
            data_format = "NCHW"
            out_backprop =
tf.saturate_cast(tf.random.
uniform([13, 6], minval=0,
maxval=64, dtype=tf.int64),
dtype=tf.half)
            res =
tf.raw_ops.BiasAddGrad(

data_format=data_format,

out_backprop=out_backprop,
            )
    except:
        pass
```

```
  def
testDilationBfloat16(self):
+    for use_gpu in True, False:
+      self._VerifyValues(
+          image=[[[[.1], [.2]],
[[.3], [.4]]]],
+          kernel=[[[.4], [.3]],
[[.1], [.0]]],
+          strides=[1, 1],
+          rates=[1, 1],
+          padding="VALID",
```

```
+           out=[[[[.5]]]],
+           use_gpu=use_gpu,
+
dtype=dtypes.bfloat16)
```

```
import tensorflow as tf
import keras
import keras.layers

class
CosineSimilarityLayer(keras.
layers.Layer):
    def __init__(
        self, num_classes:
int, name: str = None
    ):

super().__init__(name=name)
        self.num_classes =
num_classes
        self._weights = None

    def build(self,
input_shape):
        self._weights =
self.add_weight(
            name="W",
            shape=(

input_shape[-1],

self.num_classes,
            ),

initializer="glorot_normal",
            trainable=True,
```

```
            dtype=self.dtype
        )

super().build(input_shape)

    def
compute_output_shape(self):
        return None,
self.num_classes

    # If you comment out
tf.function here, it works.
    @tf.function
    def call(self, inputs:
tf.Tensor):
        embedding = inputs
        # normalize feature
        embedding_normalized =
tf.nn.l2_normalize(embedding,
axis=1)
        # get centroids
        weights_normalized =
tf.nn.l2_normalize(self._
weights, axis=1, )

        logits =
embedding_normalized @
weights_normalized

        return logits

    def get_config(self):
        config =
super().get_config().copy()
        config.update(
            {
                "num_classes":
self.num_classes,
            }
```

```
        )
        return config

def main():

tf.keras.mixed_precision.set_
global_policy("mixed_float16")

    layer =
CosineSimilarityLayer(num_
classes=100)

    input = tf.zeros(shape=
(10, 100), dtype=tf.float16)

    model =
keras.models.Sequential([
        layer
    ])

    model(input)


model.save("autocast_issue")

if __name__ == "__main__":
    main()
```

1. get_concrete_function()

```
import coremltools as ct
import tensorflow as tf
import talib as ta
import numpy as np
import os

tf.config.experimental_run_
functions_eagerly(True)
```

```
@tf.function(input_signature=[
tf.TensorSpec(shape=None,
dtype=tf.float32)])


def
gelu_tanh_activation(features)
:
    featureSize = 6
    xFeatures =
[0]*featureSize
    for i in range(0,
featureSize):
        xFeatures[i] =
float(features[i])
    return
ta.MA(np.array(xFeatures).
astype('double'), timeperiod =
2)

conc_func =
gelu_tanh_activation.get_
concrete_function()

# # provide the concrete
fucntion as a list
mlmodel =
ct.convert([conc_func])
mlmodel.save(os.environ['HOME'
] +
'/ModelSHSZ/release/ConcFunc’)



def get_timestamp_millis():
  return
math_ops.cast(logging_ops.
timestamp() * 1000,
dtype=dtypes.int64)
```

```
def weps_inject_get_dense_ops(
infer_model_id=None,
op_name_compat=True,
patterns=None):
    patterns = patterns if
patterns else {}
    _vars =
get_vars_to_sync(patterns)
    _ops = [op for op in
ops.get_default_graph().get_
operations()]

    def
update_dense_ops(infer_model_
id, op_name_compat):
        get_ops = []
        for var in _vars:
            if isinstance(var,
de.TrainableWrapper):
                pass
            else:

tf_logging.info('
get_dense_v2 for: %s(%r)' %
(var.name, type(var)))
                get_op =
gen_weps_ops.ps_get_dense_v2
                gt =
get_op(input_name=var.name,
shape=var.shape.as_list())

get_ops.append(state_ops.
assign(var, gt))
        #
get_ops.append(logging_ops.
Print(logging_ops.timestamp(),
[logging_ops.timestamp()],
        #
```

```
message="================ time
to update dense"))
        return get_ops

    dense_var_update_interval
= int(

os.environ.get('TTF_WEPS_
DENSE_VAR_UPDATE_INTERVAL_
MILLIS', '5000'))
    if
dense_var_update_interval > 0:
        last_update_time =
variables.VariableV1(
            0,
            shape=(),

dtype=dtypes.int64,

name='dense_last_update_time',
            trainable=False,
            use_resource=True)

        def
update_ops(infer_model_id,
op_name_compat):
            refresh_time_op =
state_ops.assign(last_update_
time,

get_timestamp_millis(),

use_locking=True)
            with
ops.control_dependencies([
refresh_time_op]):

update_dense_op =
control_flow_ops.group(*update
```

```
_dense_ops(infer_model_id,
op_name_compat))
            return
refresh_time_op,
update_dense_op

        get_group =
tf.group(update_ops(infer_
model_id, op_name_compat))
    else:
        get_group =
control_flow_ops.group(*update
_dense_ops(infer_model_id,
op_name_compat))

    for op in _ops:
        if op.type == 'Const':

tf_logging.info('injected
before: %s(%s)' % (op.name,
op.type))

op._add_control_input(get_
group)
```