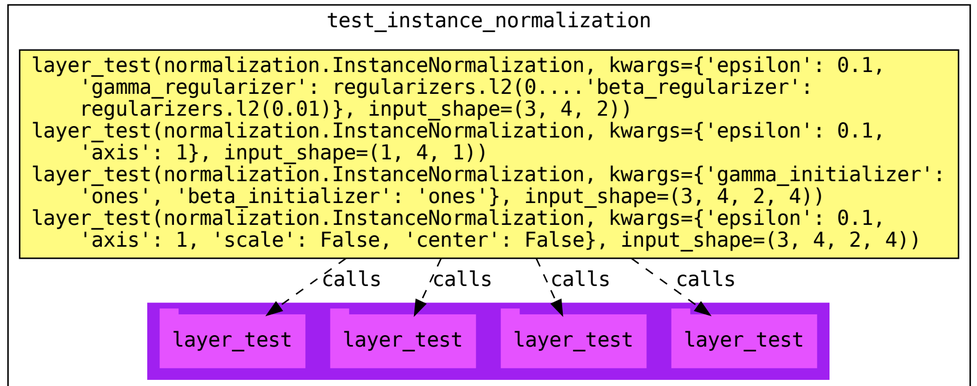
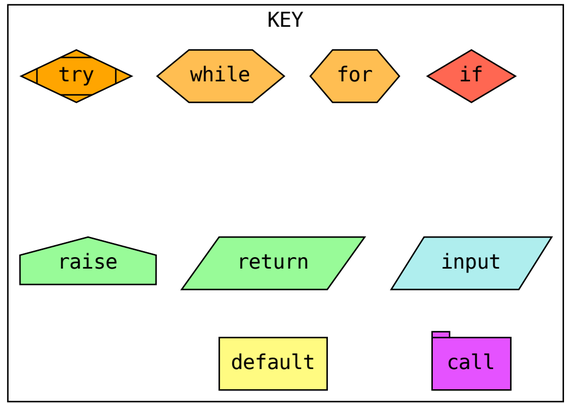
```
from keras import regularizers
import pytest
from lib.model import normalization
from lib.utils import get_backend
from .layers_test import layer_test
@pytest.mark.parametrize('dummy', [None], ids=[get_backend().upper()])...
```





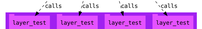
CFG1

```
from itertools import product
import numpy as np
import pytest
from keras import regularizers, models, layers
from tib.model import normalization
from tib.utils import get_backend
from tests.tib.model.layers_test import layer_test
@pytest.mark.parametrize('dummy', [None], ids=[get_backend().upper()])...
@pytest.mark.parametrize('dummy', [None], ids=[get_backend().upper()])...
_PARAMS = ('center', 'scale')
_VALUES = list(product([True, False], repeat=len(_PARAMS)))
_IDS = ['{}{}'.format('_'.join([_PARAMS[idx] for idx, b in enumerate(v) if b]), get_backend().upper()) for v in _VALUES]
@pytest.mark.parametrize(_PARAMS, _VALUES, ids=_IDS)...
@pytest.mark.parametrize(_PARAMS, _VALUES, ids=_IDS)...
_PARAMS = ('partial', 'bias')
_VALUES = [(0.0, False), (0.25, False), (0.5, True), (0.75, False), (1.0, True)]

_IDS = ['partial={}|bias={}{{...}}'.format(v[0], v[1], get_backend().upper()) for v in _VALUES]
@pytest.mark.parametrize(_PARAMS, _VALUES, ids=_IDS)...
```

test_instance_normalization

```
layer_test(normalization.InstanceNormalization, kwargs={'epsilon': 0.1,
           'gamma_regularizer': regularizers.l2(0.01),
           'beta_regularizer': regularizers.l2(0.01)}, input_shape=(3, 4, 2))
layer_test(normalization.InstanceNormalization, kwargs={'epsilon': 0.1,
           'axis': 1}, input_shape=(1, 4, 1))
layer_test(normalization.InstanceNormalization, kwargs={'gamma_initializer':
           'ones', 'beta_initializer': 'ones'}, input_shape=(3, 4, 2, 4))
layer_test(normalization.InstanceNormalization, kwargs={'epsilon': 0.1,
           'axis': 1, 'scale': False, 'center': False}, input_shape=(3, 4, 2, 4))
```



test_group_normalization

```
layer_test(normalization.GroupNormalization, kwargs={'epsilon': 0.1,
           'gamma_regularizer': regularizers.l2(0.01),
           'beta_regularizer': regularizers.l2(0.01)}, input_shape=(3, 4, 2))
layer_test(normalization.GroupNormalization, kwargs={'epsilon': 0.1, 'axis':
           1}, input_shape=(4, 1, 4, 256))
layer_test(normalization.GroupNormalization, kwargs={'gamma_init': 'ones',
           'beta_init': 'ones'}, input_shape=(4, 64))
layer_test(normalization.GroupNormalization, kwargs={'epsilon': 0.1, 'axis':
           1, 'group': 16}, input_shape=(3, 64))
```

test_adain_normalization

```
norm = normalization.AdaInstanceNormalization(center=center, scale=scale)
shapes = [(4, 8, 8, 128D), (4, 1, 1, 128D), (4, 1, 1, 128D)]
norm.build(shapes)
expected_output_shape = norm.compute_output_shape(shapes)
inputs = [layers.Input(shape=shapes[0])[:,:], layers.Input(shape=shapes[1])[1
          ]], layers.Input(shape=shapes[2])[:1]]
model = models.Model(inputs, norm(inputs))
data = [(10 * np.random.random(shape)).for shape in shapes]
actual_output = model.predict(data)
actual_output_shape = actual_output.shape
```

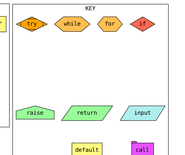
CFG2

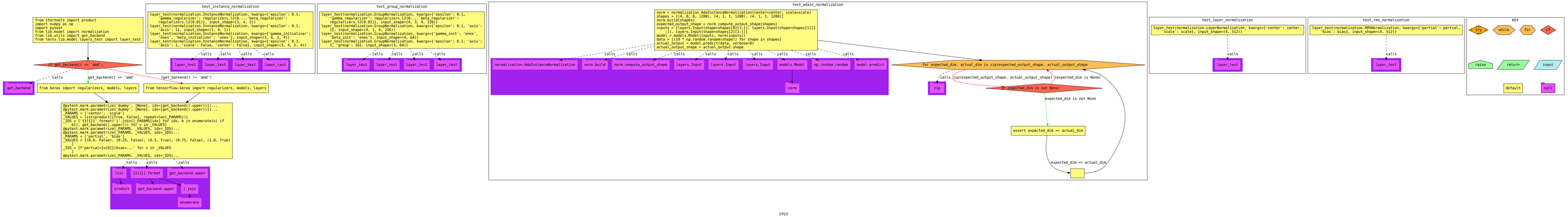
test_layer_normalization

```
layer_test(normalization.LayerNormalization, kwargs={'center': center,
           'scale': scale}, input_shape=(4, 512))
```

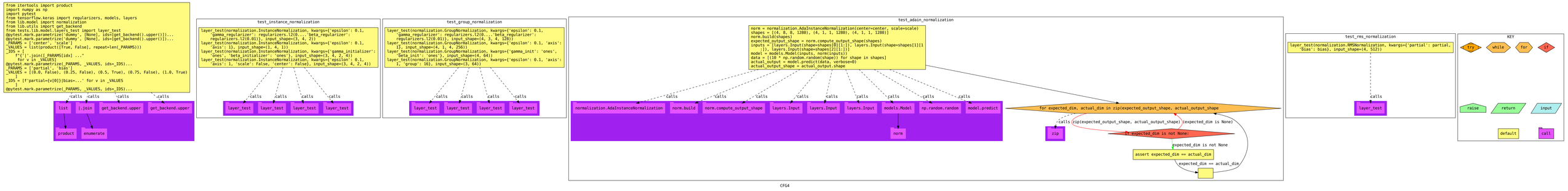
test_rms_normalization

```
layer_test(normalization.RMSNormalization, kwargs={'partial': partial,
           'bias': bias}, input_shape=(4, 512))
```




from itertools import product
import numpy as np
import pytest
from lib.model import normalization
from lib.utils import get_backend
from tests.lib.model.layers_test import layer_test
if get_backend() == 'amd':
calls
get_backend
get_backend() == 'amd'
from keras import regularizers, models, layers
get_backend() != 'amd')
from tensorflow.keras import regularizers, models, layers
@pytest.mark.parametrize('dummy', [None], ids=[get_backend().upper()])...
@pytest.mark.parametrize('dummy', [None], ids=[get_backend().upper()])...
_PARAMS = ['center', 'scale']
_VALUES = list(product([True, False], repeat=len(_PARAMS)))
_IDS = [f"{i}{j}".format("".join([_PARAMS[idx] for idx, b in enumerate(v) if b]), get_backend().upper()) for v in _VALUES]
@pytest.mark.parametrize('_PARAMS, _VALUES, ids=_IDS)...
@pytest.mark.parametrize('_PARAMS, _VALUES, ids=_IDS)...
_PARAMS = ['partial', 'bias']
_VALUES = [(0.0, False), (0.25, False), (0.5, True), (0.75, False), (1.0, True)]
_IDS = [f'partial={v[0]}|bias=...' for v in _VALUES
@pytest.mark.parametrize('_PARAMS, _VALUES, ids=_IDS)...
calls
list
{}{}{}.format
get_backend.upper
product
get_backend.upper
|.join
enumerate
test_instance_normalization
layer_test(normalization.InstanceNormalization, kwargs={'epsilon': 0.1,
'gamma_regularizer': regularizers.l2(0.),...'beta_regularizer':
regularizers.l2(0.01)}, input_shape=(3, 4, 2))
layer_test(normalization.InstanceNormalization, kwargs={'epsilon': 0.1,
'axis': 1}, input_shape=(1, 4, 1))
layer_test(normalization.InstanceNormalization, kwargs={'gamma_initializer':
'ones', 'beta_initializer': 'ones'}, input_shape=(3, 4, 2, 4))
layer_test(normalization.InstanceNormalization, kwargs={'epsilon': 0.1,
'axis': 1, 'scale': False, 'center': False}, input_shape=(3, 4, 2, 4))
calls
layer_test
layer_test
layer_test
layer_test
test_group_normalization
layer_test(normalization.GroupNormalization, kwargs={'epsilon': 0.1,
'gamma_regularizer': regularizers.l2(0.),'beta_regularizer':
regularizers.l2(0.01)}, input_shape=(4, 3, 4, 128))
layer_test(normalization.GroupNormalization, kwargs={'epsilon': 0.1, 'axis':
1}, input_shape=(4, 1, 4, 256))
layer_test(normalization.GroupNormalization, kwargs={'gamma_init': 'ones',
'beta_init': 'ones'}, input_shape=(4, 64))
layer_test(normalization.GroupNormalization, kwargs={'epsilon': 0.1, 'axis':
1, 'group': 16}, input_shape=(3, 64))
calls
layer_test
layer_test
layer_test
layer_test
test_adain_normalization
norm = normalization.AdaInstanceNormalization(center=center, scale=scale)
shapes = [(4, 8, 8, 128), (4, 1, 1, 128), (4, 1, 1, 128)]
norm.build(shapes)
expected_output_shape = norm.compute_output_shape(shapes)
inputs = [layers.Input(shape=shapes[0][1:]), layers.Input(shape=shapes[1][1
:]), layers.Input(shapes=shapes[2][1:])]
model = models.Model(inputs, norm(inputs))
data = [(0 * np.random.random(shape)) for shape in shapes]
actual_output = model.predict(data, verbose=0)
actual_output_shape = actual_output.shape
calls
normalization.AdaInstanceNormalization
norm.build
norm.compute_output_shape
layers.Input
layers.Input
layers.Input
models.Model
np.random.random
model.predict
norm
for expected_dim, actual_dim in zip(expected_output_shape, actual_output_shape
calls zip(expected_output_shape, actual_output_shape)
(expected_dim is None)
zip
if expected_dim is not None:
expected_dim is not None
assert expected_dim == actual_dim
expected_dim == actual_dim
test_layer_normalization
layer_test(normalization.LayerNormalization, kwargs={'center': center,
'scale': scale}, input_shape=(4, 512))
calls
layer_test
test_rms_normalization
layer_test(normalization.RMSNormalization, kwargs={'partial': partial,
'bias': bias}, input_shape=(4, 512))
calls
layer_test
KEY
try
while
for
if
raise
return
input
default
call

CFG3


from itertools import product
import numpy as np
import pytest
from tensorflow.keras import regularizers, models, layers
from lib.model import normalization
from lib.utils import get_backend
from tests.lib.model.layers_test import layer_test
@pytest.mark.parametrize('dummy', [None], ids=[get_backend().upper()])...
@pytest.mark.parametrize('dummy', [None], ids=[get_backend().upper()])...
_PARAMS = ['center', 'scale']
_VALUES = list(product([True, False], repeat=len(_PARAMS)))
_IDS = [
f'{'.join([_PARAMS[idx] ..."
for v in _VALUES]
@pytest.mark.parametrize(_PARAMS, _VALUES, ids=_IDS)...
_PARAMS = ['partial', 'bias']
_VALUES = [(0.0, False), (0.25, False), (0.5, True), (0.75, False), (1.0, True)
]
_IDS = [f'partial={v[0]}|bias=...' for v in _VALUES
]
@pytest.mark.parametrize(_PARAMS, _VALUES, ids=_IDS)...
calls
calls
calls
calls
list
|.join
get_backend.upper
get_backend.upper
product
enumerate
test_instance_normalization
layer_test(normalization.InstanceNormalization, kwargs={'epsilon': 0.1,
'gamma_regularizer': regularizers.l2(0...,'beta_regularizer':
regularizers.l2(0.01)}, input_shape=(3, 4, 2))
layer_test(normalization.InstanceNormalization, kwargs={'epsilon': 0.1,
'axis': 3}, input_shape=(4, 4, 3))
layer_test(normalization.InstanceNormalization, kwargs={'gamma_initializer':
'ones', 'beta_initializer': 'ones'}, input_shape=(3, 4, 2, 4))
layer_test(normalization.InstanceNormalization, kwargs={'epsilon': 0.1,
'axis': 1, 'scale': False, 'center': False}, input_shape=(3, 4, 2, 4))
calls
calls
calls
calls
layer_test
layer_test
layer_test
layer_test
test_group_normalization
layer_test(normalization.GroupNormalization, kwargs={'epsilon': 0.1,
'gamma_regularizer': regularizers.l2(0..., 'beta_regularizer':
regularizers.l2(0.01)}, input_shape=(4, 3, 4, 128))
layer_test(normalization.GroupNormalization, kwargs={'epsilon': 0.1, 'axis':
-1}, input_shape=(4, 4, 3, 256))
layer_test(normalization.GroupNormalization, kwargs={'gamma_init': 'ones',
'beta_init': 'ones'}, input_shape=(4, 64))
layer_test(normalization.GroupNormalization, kwargs={'epsilon': 0.1, 'axis':
1, 'group': 16}, input_shape=(3, 64))
calls
calls
calls
calls
layer_test
layer_test
layer_test
layer_test
test_adain_normalization
norm = normalization.AdaInstanceNormalization(center=center, scale=scale)
shapes = [(4, 8, 8, 128), (4, 1, 1, 128), (4, 1, 1, 128)]
norm.build(shapes)
expected_output_shape = norm.compute_output_shape(shapes)
inputs = [layers.Input(shape=shapes[0][1:]), layers.Input(shape=shapes[1][1
:]), layers.Input(shape=shapes[2][1:])]
model = models.Model(inputs, norm(inputs))
data = [128 * np.random.random(shape)) for shape in shapes]
actual_output = model.predict(data, verbose=0)
actual_output_shape = actual_output.shape
calls
calls
calls
calls
calls
calls
calls
calls
calls
calls
normalization.AdaInstanceNormalization
norm.build
norm.compute_output_shape
layers.Input
layers.Input
layers.Input
models.Model
np.random.random
model.predict
norm
for expected_dim, actual_dim in zip(expected_output_shape, actual_output_shape
calls
zip(expected_output_shape, actual_output_shape)
(expected_dim is None)
if expected_dim is not None:
expected_dim is not None
assert expected_dim == actual_dim
expected_dim == actual_dim
zip
test_rms_normalization
layer_test(normalization.RMSNormalization, kwargs={'partial': partial,
'bias': bias}, input_shape=(4, 512))
calls
layer_test
KEY
try
while
for
if
raise
return
input
default
call
CFG4