tf.TensorShape

Contents

Class **TensorShape**

Defined in `tensorflow/python/framework/tensor_shape.py`.

See the guide: Building Graphs > Defining new operations

Represents the shape of a `Tensor`.

A `TensorShape` represents a possibly-partial shape specification for a `Tensor`. It may be one of the following:

- *Fully-known shape:* has a known number of dimensions and a known size for each dimension. e.g. `TensorShape([16, 256])`
- *Partially-known shape:* has a known number of dimensions, and an unknown size for one or more dimension. e.g. `TensorShape([None, 256])`
- *Unknown shape:* has an unknown number of dimensions, and an unknown size in all dimensions. e.g. `TensorShape(None)`

If a tensor is produced by an operation of type `"Foo"`, its shape may be inferred if there is a registered shape function for `"Foo"`. See `Shape functions in C++` for details of shape functions and how to register them. Alternatively, the shape may be set explicitly using `tf.Tensor.set_shape`.

Properties

`dims`

Returns a list of Dimensions, or None if the shape is unspecified.

`ndims`

Returns the rank of this shape, or None if it is unspecified.

Methods

`__init__`

```
__init__(dims)
```

Creates a new TensorShape with the given dimensions.

Args:

- `dims`: A list of Dimensions, or None if the shape is unspecified.
- `DEPRECATED`: A single integer is treated as a singleton list.

Raises:

- `TypeError`: If dims cannot be converted to a list of dimensions.

__bool__

```
__bool__()
```

Returns True if this shape contains non-zero information.

__eq__

```
__eq__(other)
```

Returns True if **`self`** is equivalent to **`other`**.

__getitem__

```
__getitem__(key)
```

Returns the value of a dimension or a shape, depending on the key.

Args:

- `key`: If **`key`** is an integer, returns the dimension at that index; otherwise if **`key`** is a slice, returns a TensorShape whose dimensions are those selected by the slice from **`self`**.

Returns:

A dimension if **`key`** is an integer, or a **`TensorShape`** if **`key`** is a slice.

Raises:

- `ValueError`: If **`key`** is a slice, and any of its elements are negative, or if **`self`** is completely unknown and the step is set.

__iter__

```
__iter__()
```

Returns **`self.dims`** if the rank is known, otherwise raises ValueError.

__len__

```
__len__()
```

Returns the rank of this shape, or raises ValueError if unspecified.

`__ne__`

```
__ne__(other)
```

Returns True if `self` is known to be different from `other`.

`__nonzero__`

```
__nonzero__()
```

Returns True if this shape contains non-zero information.

`as_list`

```
as_list()
```

Returns a list of integers or `None` for each dimension.

Returns:

A list of integers or `None` for each dimension.

Raises:

- `ValueError`: If `self` is an unknown shape with an unknown rank.

`as_proto`

```
as_proto()
```

Returns this shape as a `TensorShapeProto`.

`assert_has_rank`

```
assert_has_rank(rank)
```

Raises an exception if `self` is not compatible with the given `rank`.

Args:

- `rank`: An integer.

Raises:

- `ValueError`: If `self` does not represent a shape with the given `rank`.

assert_is_compatible_with

```
assert_is_compatible_with(other)
```

Raises exception if **self** and **other** do not represent the same shape.

This method can be used to assert that there exists a shape that both **self** and **other** represent.

Args:

- **other**: Another TensorShape.

Raises:

- **ValueError**: If **self** and **other** do not represent the same shape.

assert_is_fully_defined

```
assert_is_fully_defined()
```

Raises an exception if **self** is not fully defined in every dimension.

Raises:

- **ValueError**: If **self** does not have a known value for every dimension.

assert_same_rank

```
assert_same_rank(other)
```

Raises an exception if **self** and **other** do not have compatible ranks.

Args:

- **other**: Another **TensorShape**.

Raises:

- **ValueError**: If **self** and **other** do not represent shapes with the same rank.

concatenate

```
concatenate(other)
```

Returns the concatenation of the dimension in **self** and **other**.

N.B. If either **self** or **other** is completely unknown, concatenation will discard information about the other shape. In future, we might support concatenation that preserves this information for use with slicing.

Args:

- `other`: Another `TensorShape`.

Returns:

A `TensorShape` whose dimensions are the concatenation of the dimensions in `self` and `other`.

`is_compatible_with`

```
is_compatible_with(other)
```

Returns True iff `self` is compatible with `other`.

Two possibly-partially-defined shapes are compatible if there exists a fully-defined shape that both shapes can represent. Thus, compatibility allows the shape inference code to reason about partially-defined shapes. For example:

- TensorShape(None) is compatible with all shapes.
- TensorShape([None, None]) is compatible with all two-dimensional shapes, such as TensorShape([32, 784]), and also TensorShape(None). It is not compatible with, for example, TensorShape([None]) or TensorShape([None, None, None]).
- TensorShape([32, None]) is compatible with all two-dimensional shapes with size 32 in the 0th dimension, and also TensorShape([None, None]) and TensorShape(None). It is not compatible with, for example, TensorShape([32]), TensorShape([32, None, 1]) or TensorShape([64, None]).
- TensorShape([32, 784]) is compatible with itself, and also TensorShape([32, None]), TensorShape([None, 784]), TensorShape([None, None]) and TensorShape(None). It is not compatible with, for example, TensorShape([32, 1, 784]) or TensorShape([None]).

The compatibility relation is reflexive and symmetric, but not transitive. For example, TensorShape([32, 784]) is compatible with TensorShape(None), and TensorShape(None) is compatible with TensorShape([4, 4]), but TensorShape([32, 784]) is not compatible with TensorShape([4, 4]).

Args:

- `other`: Another TensorShape.

Returns:

True iff `self` is compatible with `other`.

`is_fully_defined`

```
is_fully_defined()
```

Returns True iff `self` is fully defined in every dimension.

`merge_with`

```
merge_with(other)
```

Returns a `TensorShape` combining the information in `self` and `other`.

The dimensions in `self` and `other` are merged elementwise, according to the rules defined for `Dimension.merge_with()`.

Args:

- `other`: Another `TensorShape`.

Returns:

A `TensorShape` containing the combined information of `self` and `other`.

Raises:

- `ValueError`: If `self` and `other` are not compatible.

`most_specific_compatible_shape`

```
most_specific_compatible_shape(other)
```

Returns the most specific TensorShape compatible with `self` and `other`.

- TensorShape([None, 1]) is the most specific TensorShape compatible with both TensorShape([2, 1]) and TensorShape([5, 1]). Note that TensorShape(None) is also compatible with above mentioned TensorShapes.
- TensorShape([1, 2, 3]) is the most specific TensorShape compatible with both TensorShape([1, 2, 3]) and TensorShape([1, 2, 3]). There are more less specific TensorShapes compatible with above mentioned TensorShapes, e.g. TensorShape([1, 2, None]), TensorShape(None).

Args:

- `other`: Another `TensorShape`.

Returns:

A `TensorShape` which is the most specific compatible shape of `self` and `other`.

`num_elements`

```
num_elements()
```

Returns the total number of elements, or none for incomplete shapes.

`with_rank`

```
with_rank(rank)
```

Returns a shape based on `self` with the given rank.

This method promotes a completely unknown shape to one with a known rank.

Args:

- `rank`: An integer.

Returns:

A shape that is at least as specific as `self` with the given rank.

Raises:

- `ValueError`: If `self` does not represent a shape with the given `rank`.

`with_rank_at_least`

```
with_rank_at_least(rank)
```

Returns a shape based on `self` with at least the given rank.

Args:

- `rank`: An integer.

Returns:

A shape that is at least as specific as `self` with at least the given rank.

Raises:

- `ValueError`: If `self` does not represent a shape with at least the given `rank`.

`with_rank_at_most`

```
with_rank_at_most(rank)
```

Returns a shape based on `self` with at most the given rank.

Args:

- `rank`: An integer.

Returns:

A shape that is at least as specific as `self` with at most the given rank.

Raises:

- `ValueError`: If `self` does not represent a shape with at most the given `rank`.