tf.DType

Contents

Class **DType**

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

See the guide: Building Graphs > Tensor types

Represents the type of the elements in a `Tensor`.

The following `DType` objects are defined:

- `tf.float16`: 16-bit half-precision floating-point.
- `tf.float32`: 32-bit single-precision floating-point.
- `tf.float64`: 64-bit double-precision floating-point.
- `tf.bfloat16`: 16-bit truncated floating-point.
- `tf.complex64`: 64-bit single-precision complex.
- `tf.complex128`: 128-bit double-precision complex.
- `tf.int8`: 8-bit signed integer.
- `tf.uint8`: 8-bit unsigned integer.
- `tf.uint16`: 16-bit unsigned integer.
- `tf.int16`: 16-bit signed integer.
- `tf.int32`: 32-bit signed integer.
- `tf.int64`: 64-bit signed integer.
- `tf.bool`: Boolean.
- `tf.string`: String.
- `tf.qint8`: Quantized 8-bit signed integer.
- `tf.quint8`: Quantized 8-bit unsigned integer.
- `tf.qint16`: Quantized 16-bit signed integer.
- `tf.quint16`: Quantized 16-bit unsigned integer.
- `tf.qint32`: Quantized 32-bit signed integer.
- `tf.resource`: Handle to a mutable resource.
- `tf.variant`: Values of arbitrary types.

In addition, variants of these types with the `_ref` suffix are defined for reference-typed tensors.

The `tf.as_dtype()` function converts numpy types and string type names to a `DType` object.

Properties

as_datatype_enum

Returns a `types_pb2.DataType` enum value based on this `DType`.

as_numpy_dtype

Returns a `numpy.dtype` based on this `DType`.

base_dtype

Returns a non-reference `DType` based on this `DType`.

is_bool

Returns whether this is a boolean data type

is_complex

Returns whether this is a complex floating point type.

is_floating

Returns whether this is a (non-quantized, real) floating point type.

is_integer

Returns whether this is a (non-quantized) integer type.

is_numpy_compatible

is_quantized

Returns whether this is a quantized data type.

is_unsigned

Returns whether this type is unsigned.

Non-numeric, unordered, and quantized types are not considered unsigned, and this function returns `False`.

Returns:

Whether a `DType` is unsigned.

limits

Return intensity limits, i.e. (min, max) tuple, of the dtype.

Args:

- `clip_negative` : bool, optional If True, clip the negative range (i.e. return 0 for min intensity) even if the image dtype allows negative values. Returns min, max : tuple Lower and upper intensity limits.

`max`

Returns the maximum representable value in this data type.

Raises:

- `TypeError` : if this is a non-numeric, unordered, or quantized type.

`min`

Returns the minimum representable value in this data type.

Raises:

- `TypeError` : if this is a non-numeric, unordered, or quantized type.

`name`

Returns the string name for this `DType` .

`real_dtype`

Returns the dtype correspond to this dtype's real part.

`size`

Methods

`__init__`

```
__init__(type_enum)
```

Creates a new `DataType` .

NOTE(mrry): In normal circumstances, you should not need to construct a `DataType` object directly. Instead, use the `tf.as_dtype()` function.

Args:

- `type_enum` : A `types_pb2.DataType` enum value.

Raises:

- `TypeError` : If `type_enum` is not a value `types_pb2.DataType` .

__eq__

```
__eq__(other)
```

Returns True iff this DType refers to the same type as `other`.

__int__

```
__int__()
```

__ne__

```
__ne__(other)
```

Returns True iff self != other.

is_compatible_with

```
is_compatible_with(other)
```

Returns True if the `other` DType will be converted to this DType.

The conversion rules are as follows:

```
DType(T)       .is_compatible_with(DType(T))        == True
DType(T)       .is_compatible_with(DType(T).as_ref) == True
DType(T).as_ref.is_compatible_with(DType(T))        == False
DType(T).as_ref.is_compatible_with(DType(T).as_ref) == True
```

Args:

- `other`: A `DType` (or object that may be converted to a `DType`).

Returns:

True if a Tensor of the `other` `DType` will be implicitly converted to this `DType`.