Tensorflow Set Up

Instalation

```
pip install tensorflow
```

Import

```python
# Import the TensorFlow library with the alias tf
import tensorflow as tf
```

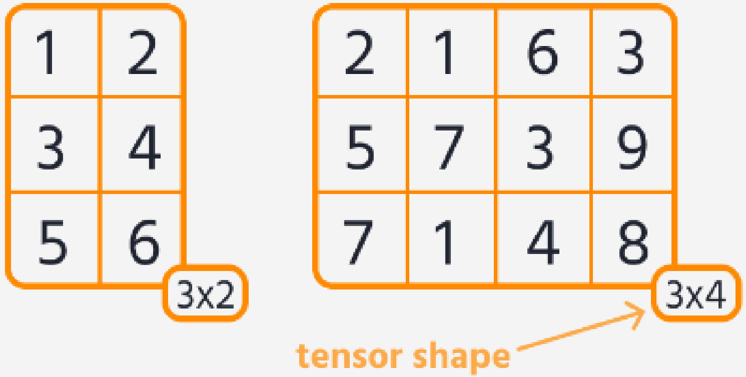
Tensor Types

Scalars (0D Tensors)

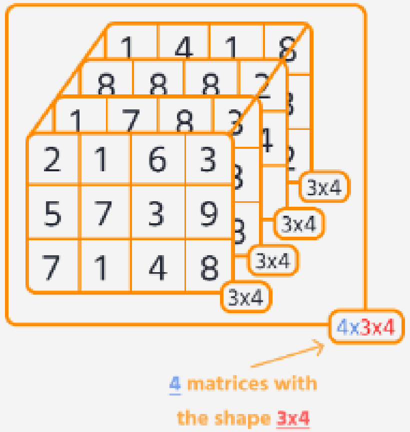
5 1 7

Vectors (1D Tensors)

Matrices (2D Tensors)

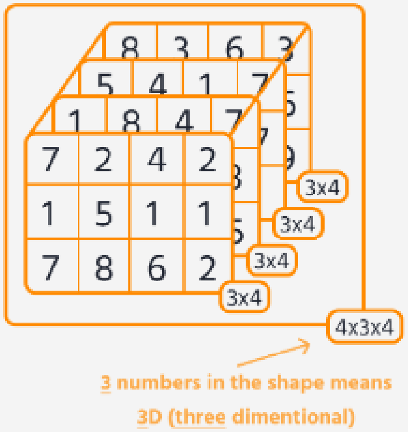


3D Tensor #1



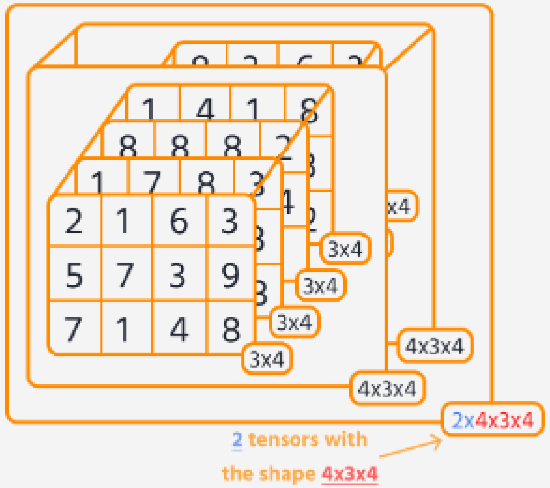
+

3D Tensor #2

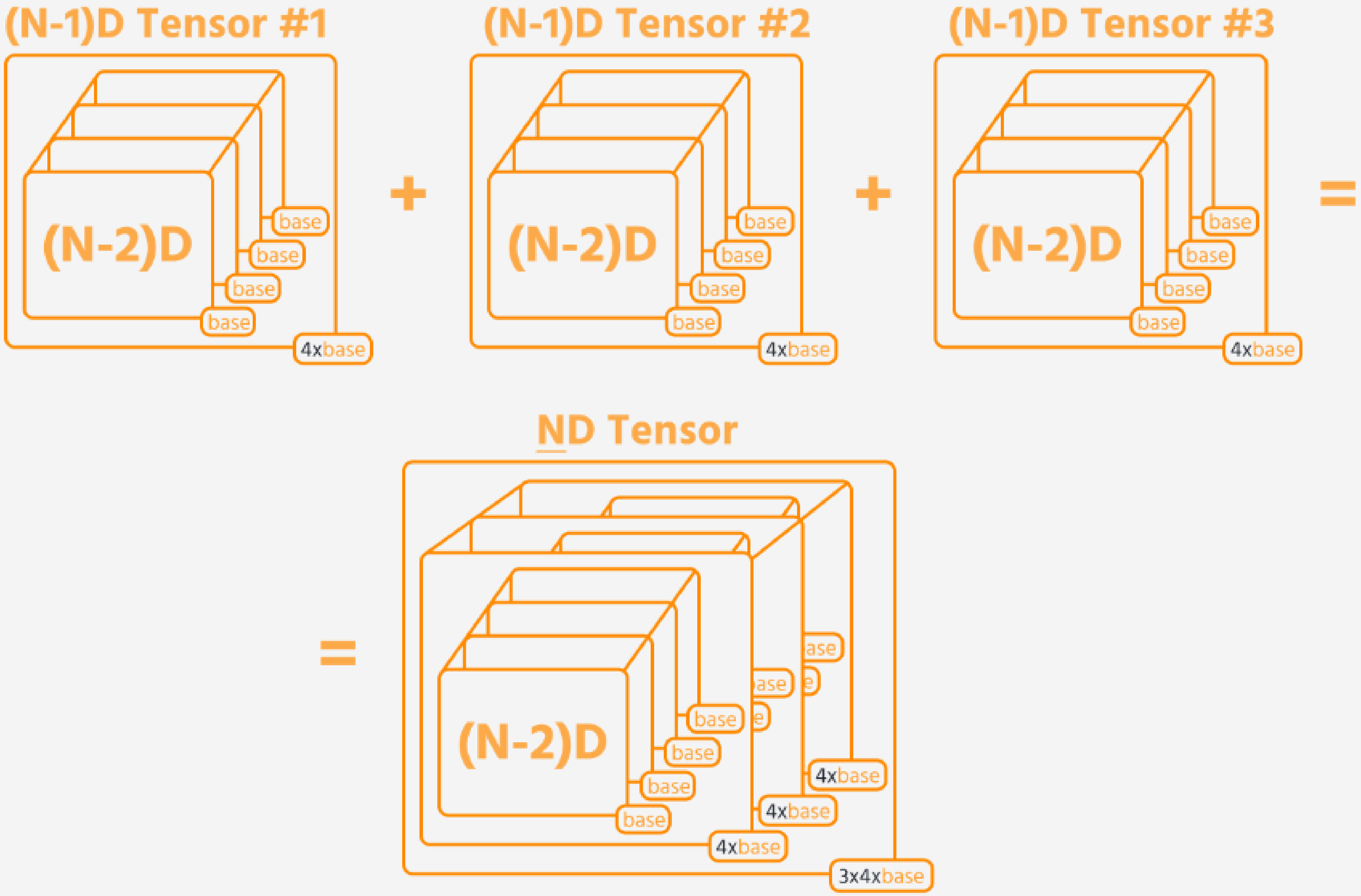


=

4D Tensor



Tensor Types



Simple Tensor Creation

```
# Create a 1D tensor
tensor_1D = tf.constant([1, 2, 3])

# Create a 2D tensor
tensor_2D = tf.constant([[1, 2, 3], [4, 5, 6]])

# Create a 3D tensor
tensor_3D = tf.constant([[[1, 2], [3, 4]], [[5, 6],[7, 8]]])
```

Tensor Properties

- **Rank**: It tells you the *number of dimensions* present in the tensor. For instance, a *matrix* has a rank of *2*. You can get the rank of the tensor using the `.ndim` attribute:

```
print(f'Rank of a tensor: {tensor.ndim}')
```

- **Shape**: This describes how many values exist in each dimension. A 2x3 matrix has a shape of `(2, 3)`. The length of the shape parameter matches the tensor's rank *(its number of dimensions)*. You can get the the shape of the tensor by the `.shape` attribute:

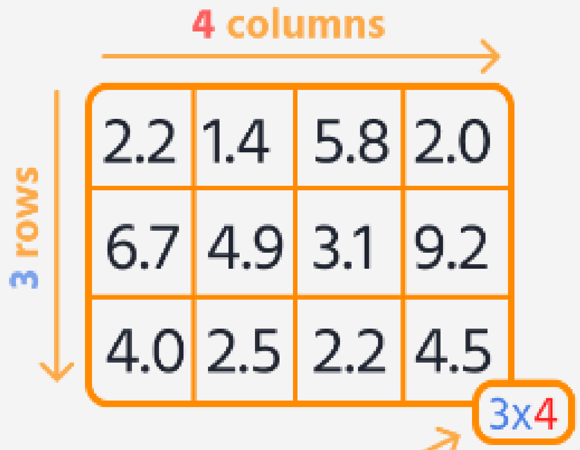
```
print(f'Shape of a tensor: {tensor.shape}')
```

- **Types**: Tensors come in various data types. While there are many, some common ones include `float32`, `int32`, and `string`. You can get the the data type of the tensor by the `.dtype` attribute:

```
print(f'Data type of a tensor: {tensor.dtype}')
```

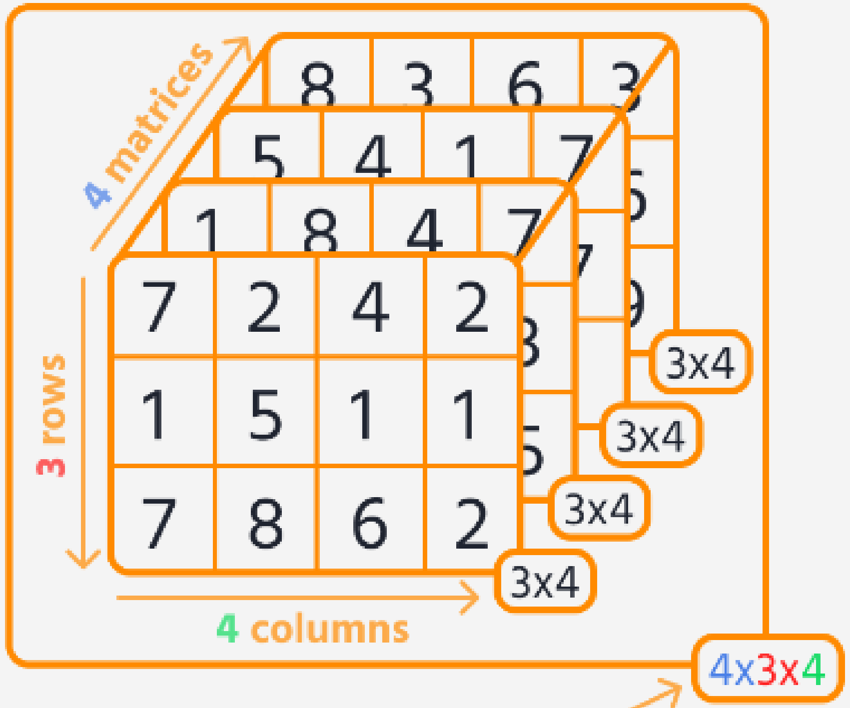
Tensor Properties

Matrix (2D Tensor)



Shape: (3x4)
Rank: 2
Data type: float32

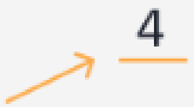
3D Tensor



Shape: (4x3x4)
Rank: 3
Data type: int32

Tensor Axes

Scalar (0D Tensor)



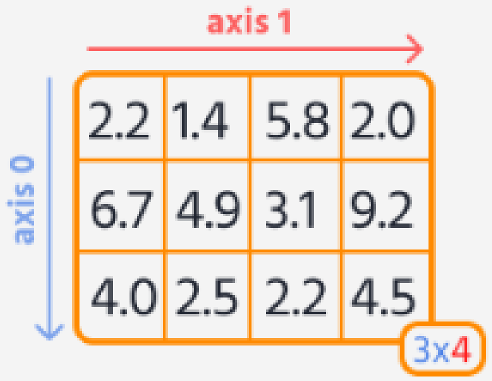
No Shape and No Axes

Rank: 0

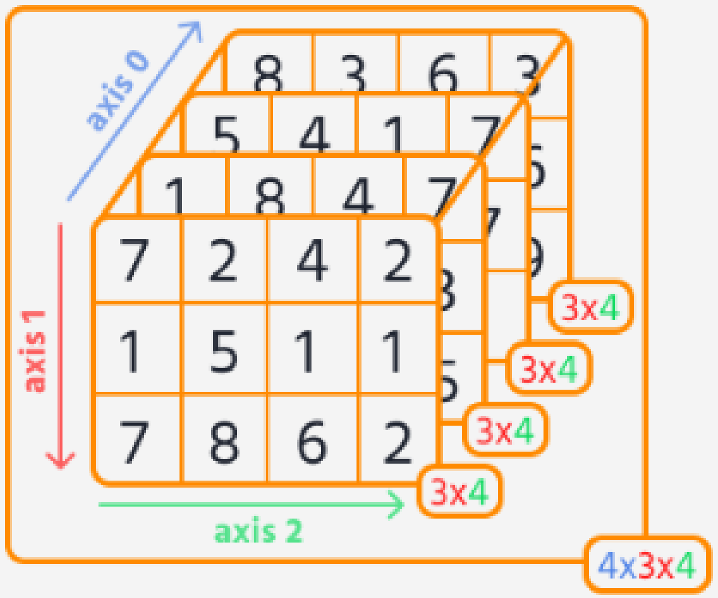
Data type: int32

Vector (1D Tensor)

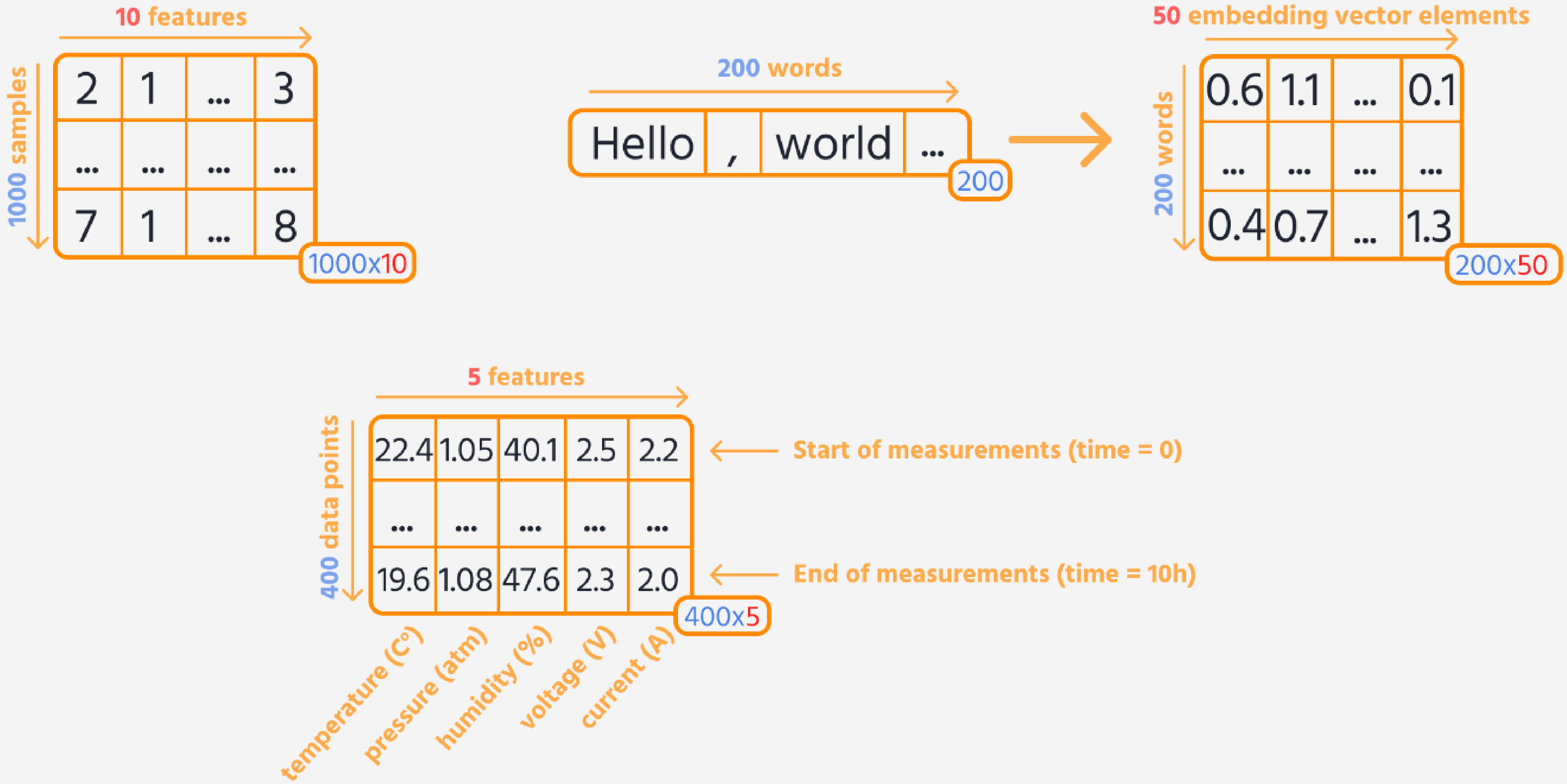
Matrix (2D Tensor)



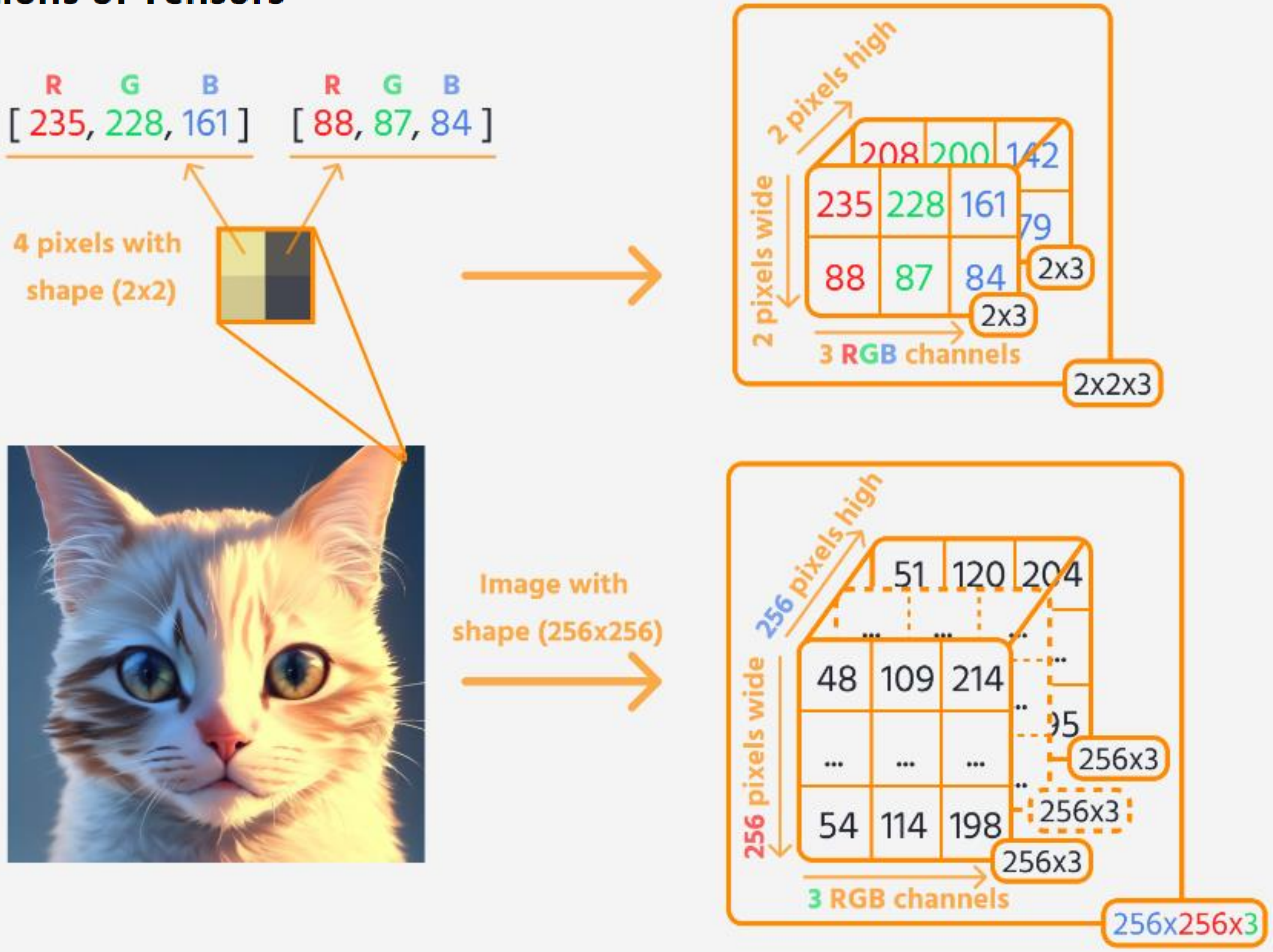
3D Tensor



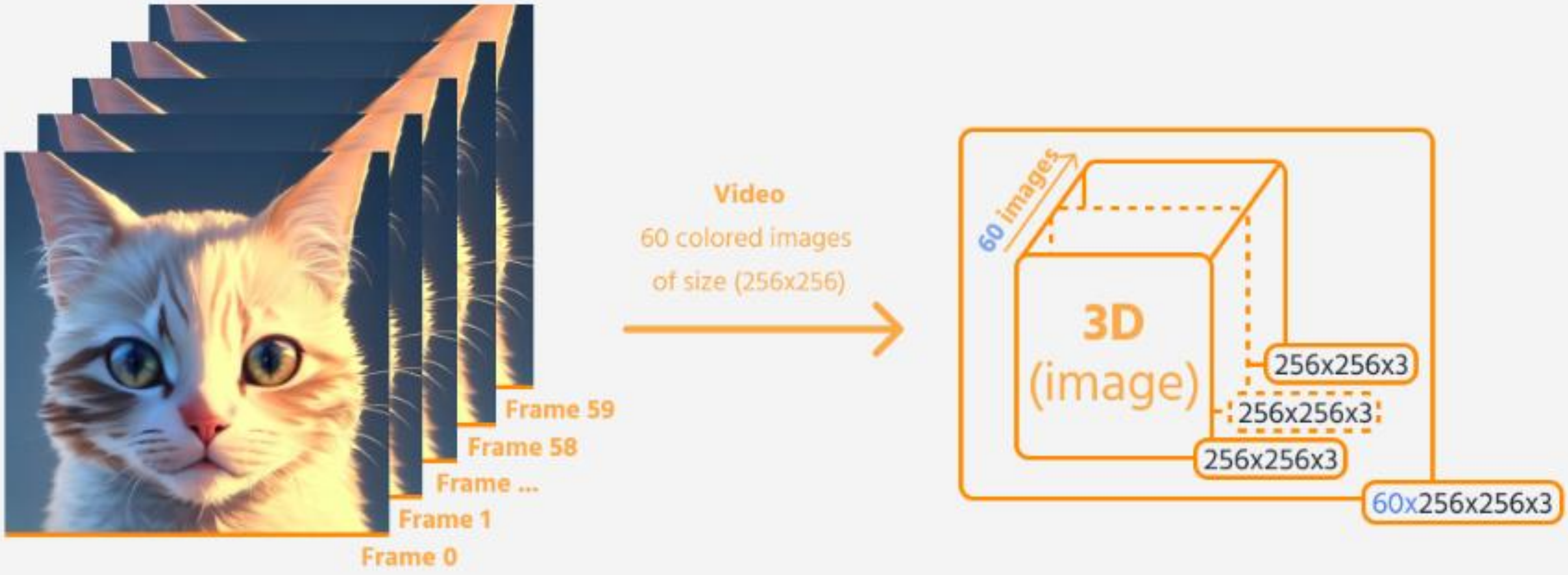
Applications of Tensors



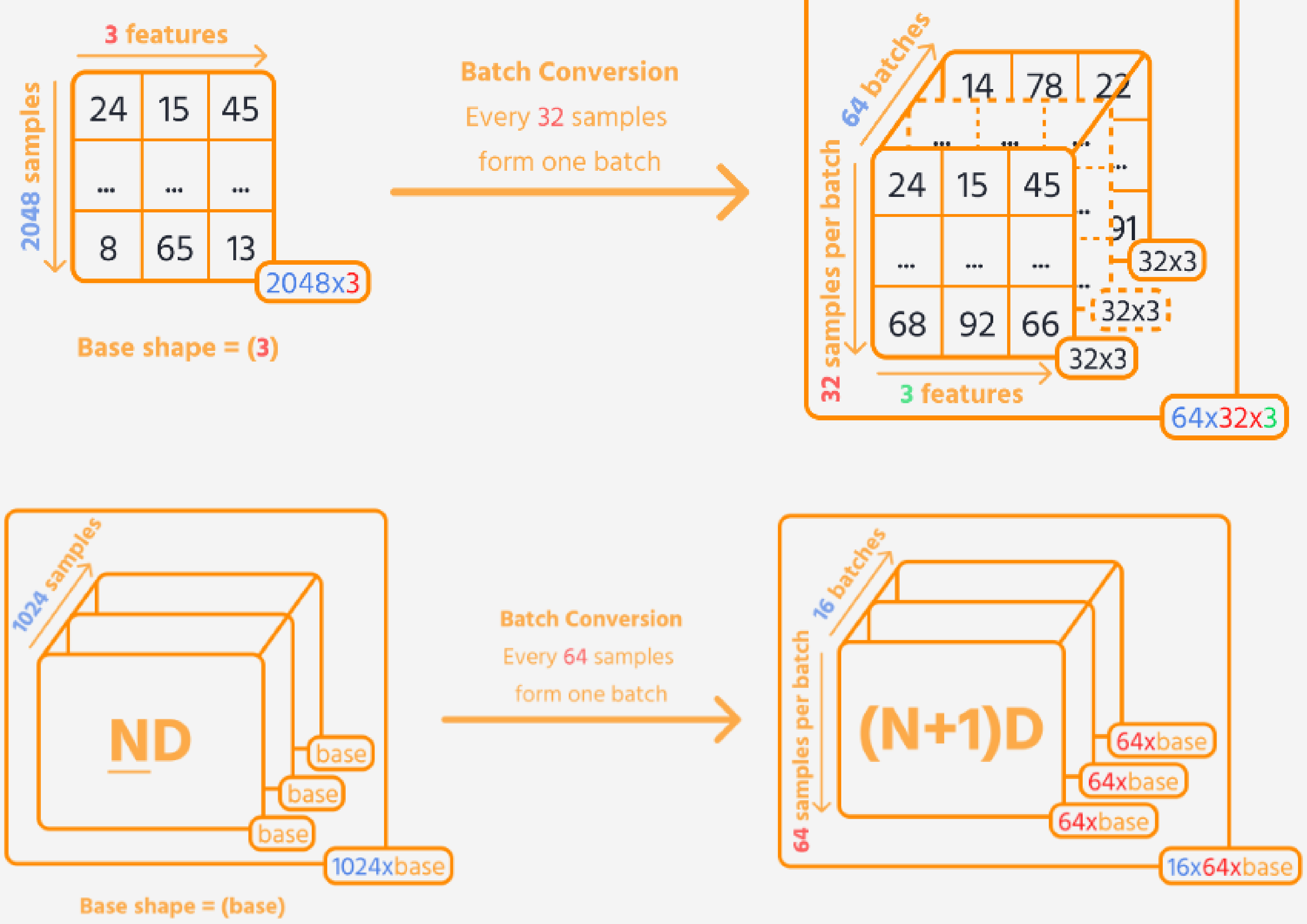
Applications of Tensors



Applications of Tensors



Batches



3 features
2048 samples
24 15 45
... ...
8 65 13
2048x3
Base shape = (3)
Batch Conversion
Every 32 samples
form one batch
64 batches
32 samples per batch
14 78 22
24 15 45
... ...
68 92 66
91
32x3
32x3
32x3
3 features
64x32x3
1024 samples
ND
base
base
base
1024xbase
Base shape = (base)
Batch Conversion
Every 64 samples
form one batch
16 batches
64 samples per batch
(N+1)D
64xbase
64xbase
64xbase
16x64xbase

Tensor Creation Methods

```python
# Create a 2x2 constant tensor
tensor_const = tf.constant([[1, 2], [3, 4]])

# Create a variable tensor
tensor_var = tf.Variable([[1, 2], [3, 4]])

# Zero tensor of shape (3, 3)
tensor_zeros = tf.zeros((3, 3))

# Ones tensor of shape (2, 2)
tensor_ones = tf.ones((2, 2))

# Tensor of shape (2, 2) filled with 6
tensor_fill = tf.fill((2, 2), 6)

# Generate a sequence of numbers starting from 0, ending at 9
tensor_range = tf.range(10)

# Create 5 equally spaced values between 0 and 10
tensor_linspace = tf.linspace(0, 10, 5)

# Tensor of shape (2, 2) with random values normally distributed
tensor_random = tf.random.normal((2, 2), mean=4, stddev=0.5)

# Tensor of shape (2, 2) with random values uniformly distributed
tensor_random = tf.random.uniform((2, 2), minval=-2, maxval=2)
```

Convertions

- NumPy to Tensor

```
# Create a NumPy array based on a Python list
numpy_array = np.array([[1, 2], [3, 4]])

# Convert a NumPy array to a tensor
tensor_from_np = tf.convert_to_tensor(numpy_array)
```

- Pandas to Tensor

```
# Create a DataFrame based on dictionary
df = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})

# Convert a DataFrame to a tensor
tensor_from_df = tf.convert_to_tensor(df.values)
```

- Constant Tensor to a Variable Tensor

```
# Create a variable from a tensor
tensor = tf.random.normal((2, 3))
variable_1 = tf.Variable(tensor)

# Create a variable based on other generator
variable_2 = tf.Variable(tf.zeros((2, 2)))
```

Data Types

Data Type	Range of Values
float32	$3.4\ e^{+/-38}$ (7 digits)
float64	$1.7\ e^{+/-308}$ (15 digits)
int8	-128 to 127
int32	-2,147,483,648 to 2,147,483,647
uint32	0 to 4,294,967,295
bool	True or False (1 or 0)
string	Limited by available memory

```python
# Creating a tensor of type float16
tensor_float = tf.constant([1.2, 2.3, 3.4], dtype=tf.float16)

# Convert tensor_float from float32 to int32
tensor_int = tf.cast(tensor_float, dtype=tf.int32)
```

Arithmetic

- Addition

```
c1 = tf.add(a, b)
c2 = a + b

# Changes the object inplace without creating a new one
a.assign_add(b)
```

- Subtraction

```
c1 = tf.subtract(a, b)
c2 = a - b

# Inplace substraction
a.assign_sub(b)
```

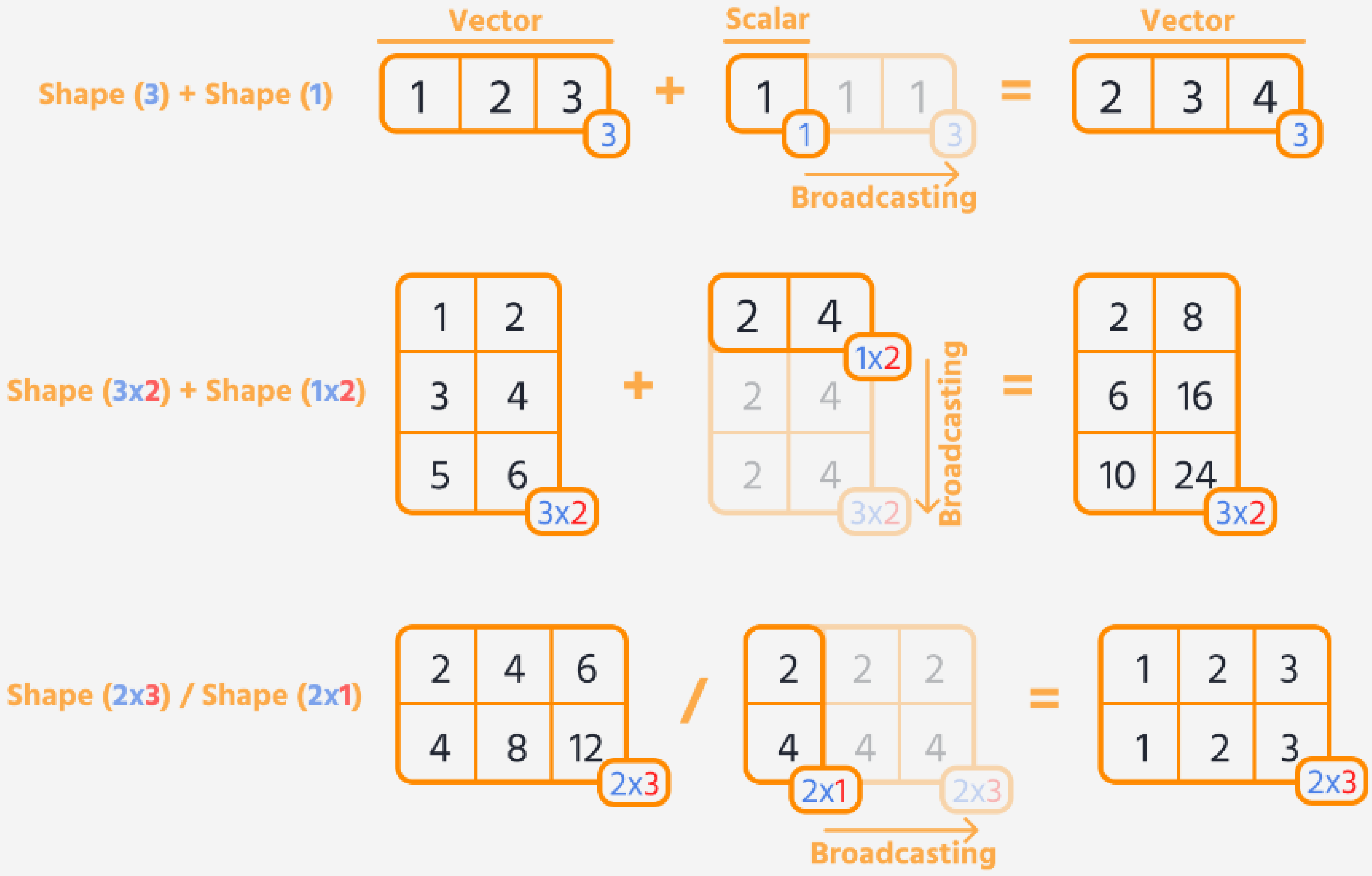
- Element-wise Multiplication

```
c1 = tf.multiply(a, b)
c2 = a * b
```

- Division

```
c1 = tf.divide(a, b)
c2 = a / b
```

Broadcasting



Linear Algebra

- Matrix Multiplication

```
product1 = tf.matmul(matrix1, matrix2)
product2 = matrix1 @ matrix2
```

- Matrix Inversion

```
inverse_mat = tf.linalg.inv(matrix)
```

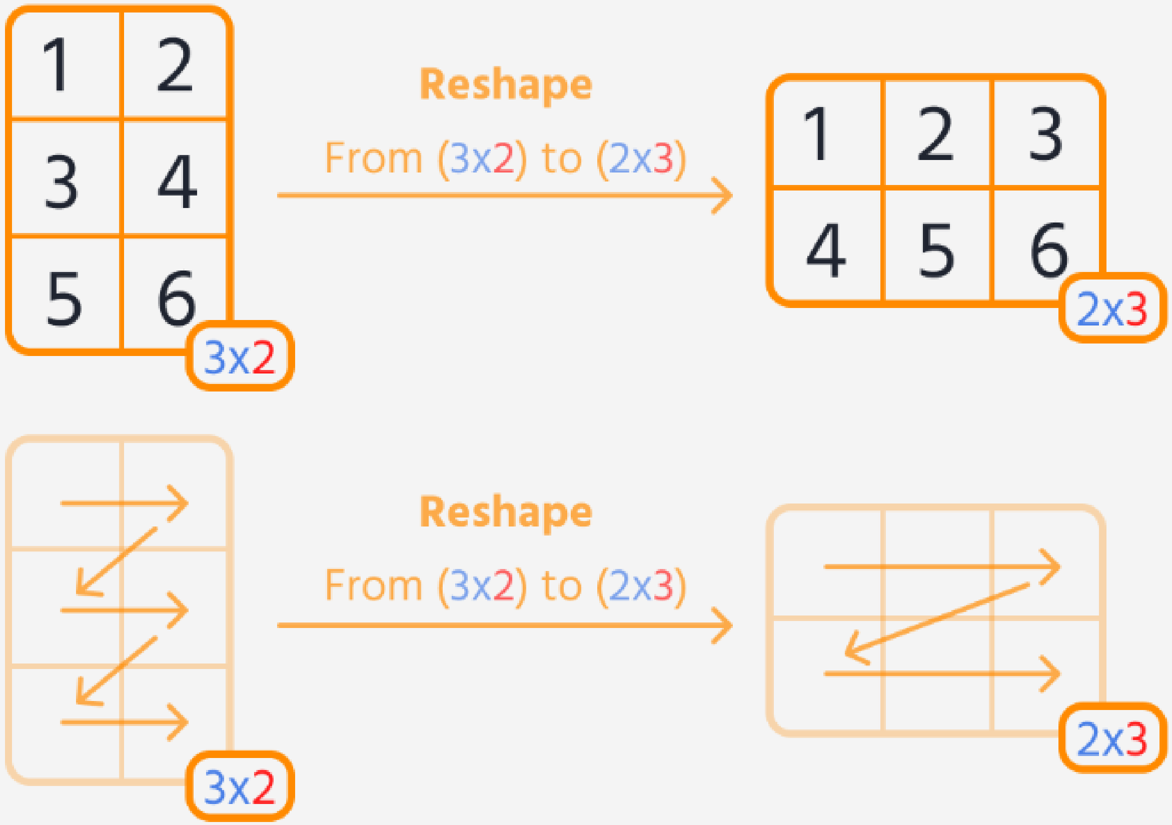
- Transpose

```
transposed = tf.transpose(matrix)
```

- Dot Product

```
# Dot product along axes
dot_product_axes1 = tf.tensordot(matrix1, matrix2, axes=1)
dot_product_axes0 = tf.tensordot(matrix1, matrix2, axes=0)
```

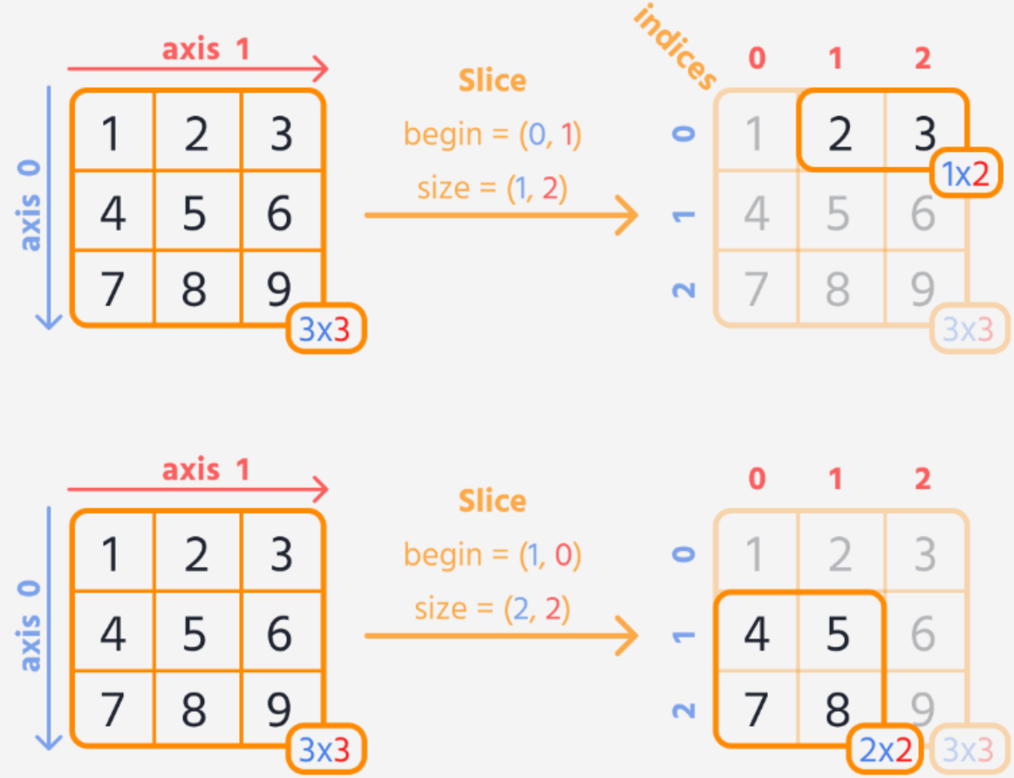
Reshape




```python
# Create a tensor with shape (3, 2)
tensor = tf.constant([[1, 2], [3, 4], [5, 6]])

# Reshape the tensor to shape (2, 3)
reshaped_tensor = tf.reshape(tensor, (2, 3))
```

Slicing

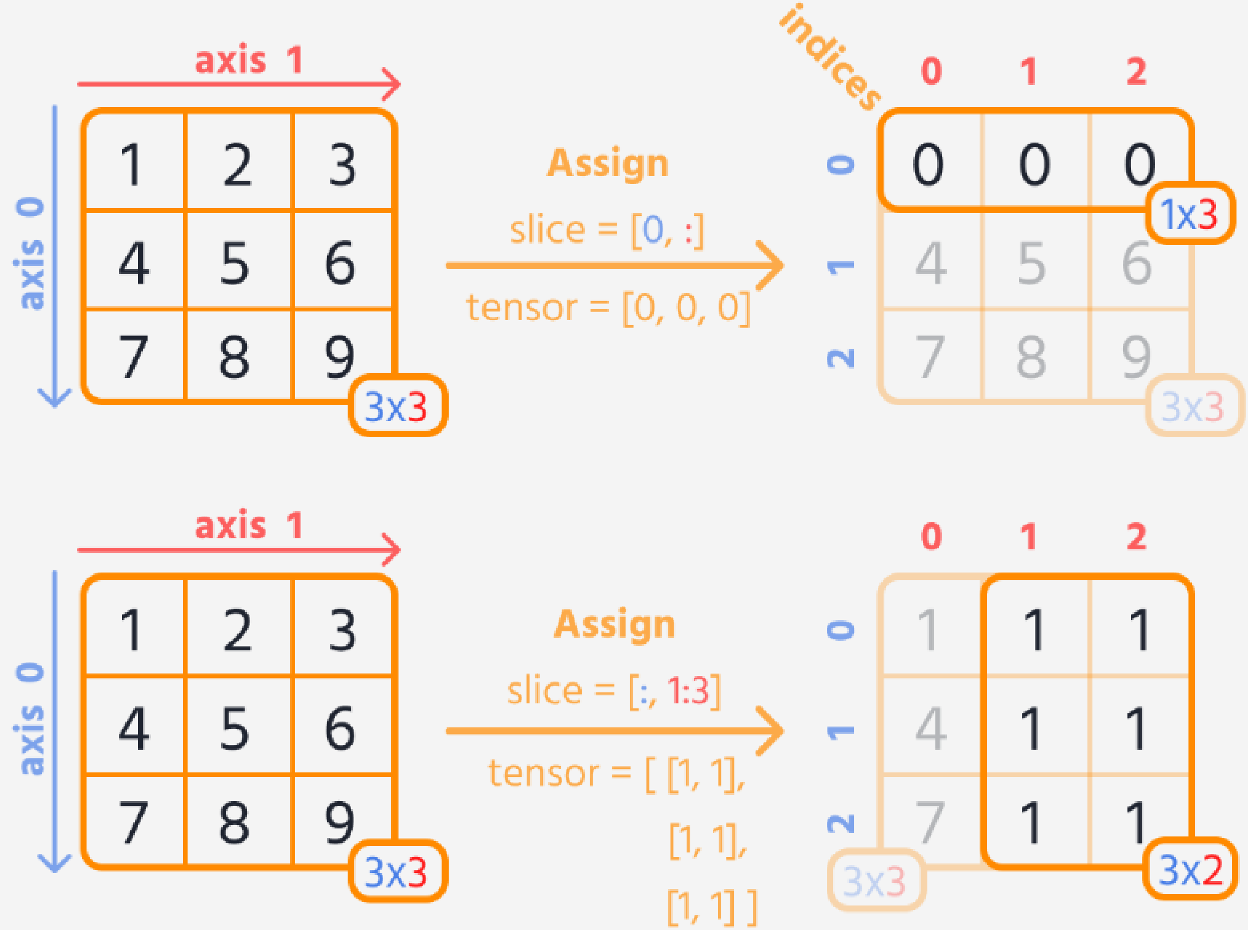



```python
# Create a tensor
tensor = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Slice tensor to extract sub-tensor from index (0, 1) of size (1, 2)
sliced_tensor = tf.slice(tensor, begin=(0, 1), size=(1, 2))

# Slice tensor to extract sub-tensor from index (1, 0) of size (2, 2)
sliced_tensor = tf.slice(tensor, (1, 0), (2, 2))
```

Modifying with Slicing

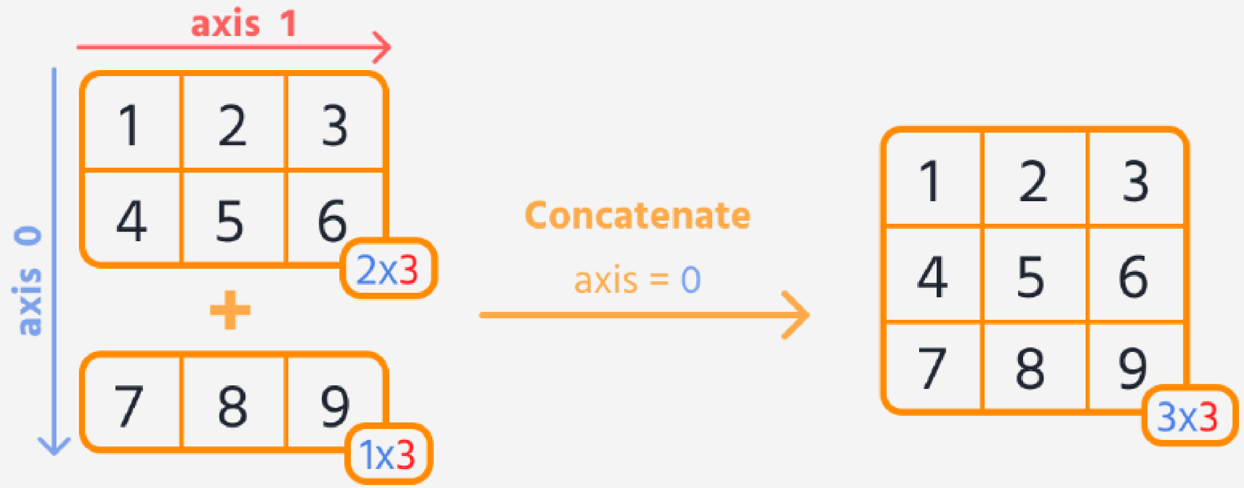


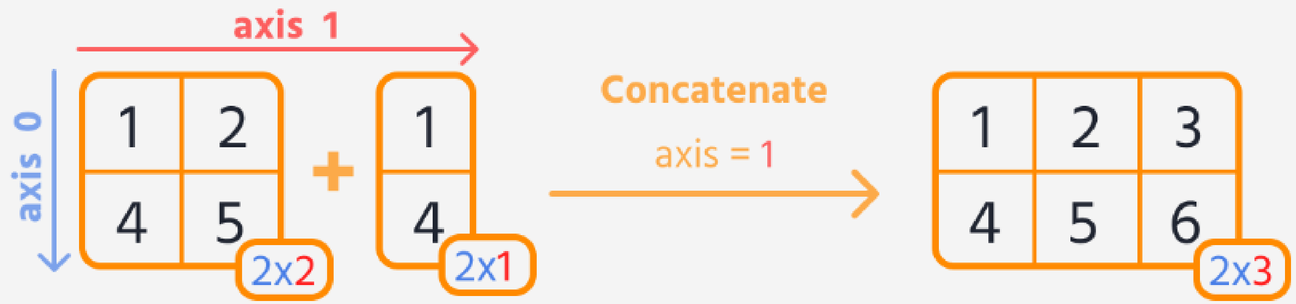

```python
# Create a tensor
tensor = tf.Variable([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Change the entire first row
tensor[0, :].assign([0, 0, 0])

# Modify the second and the third columns
tensor[:, 1:3].assign(tf.fill((3,2), 1))
```

Concatenating



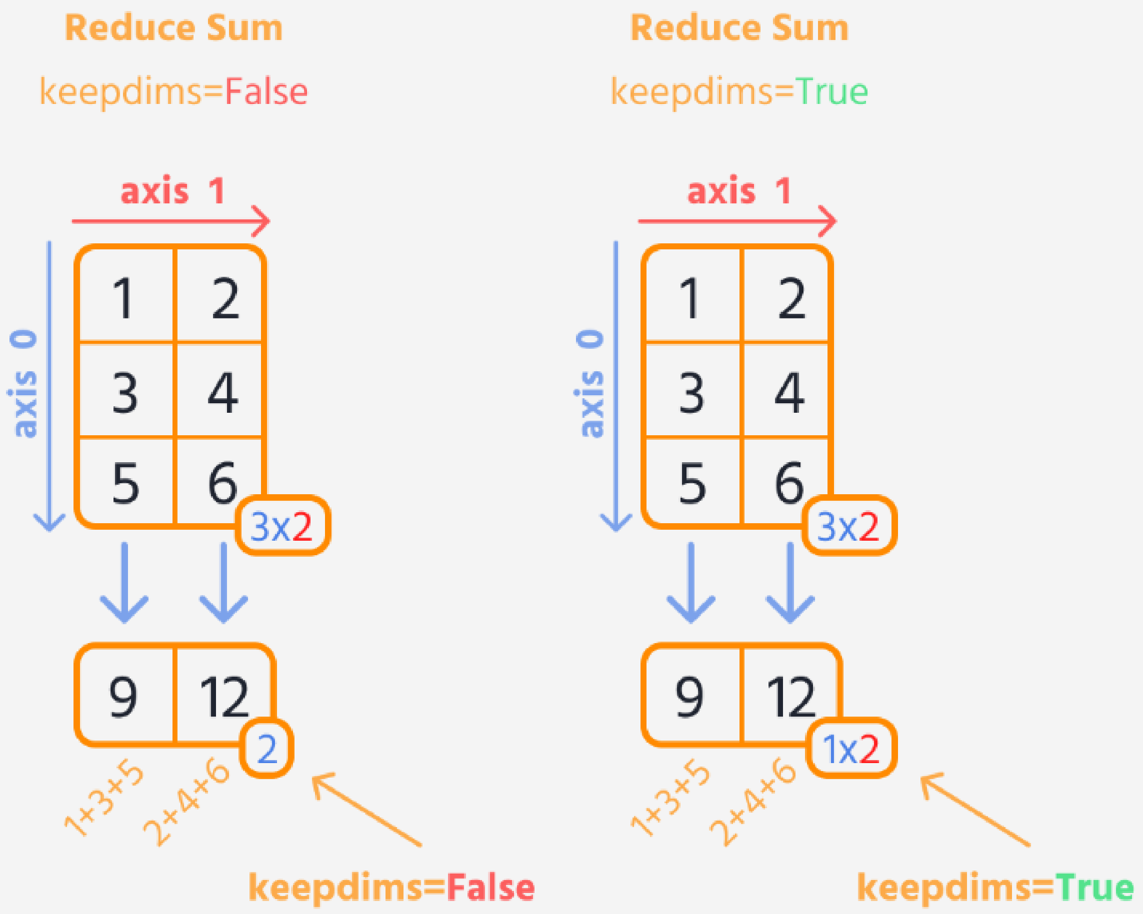



```
# Create two tensors
tensor1 = tf.constant([[1, 2, 3], [4, 5, 6]])
tensor2 = tf.constant([[7, 8, 9]])

# Concatenate tensors vertically (along rows)
concatenated_tensor = tf.concat([tensor1, tensor2], axis=0)

# Concatenate tensors horizontally (along columns)
concatenated_tensor = tf.concat([tensor3, tensor4], axis=1)
```

Reduction Operations




```python
# Calculate sum of all elements
total_sum = tf.reduce_sum(tensor)

# Calculate mean of all elements
mean_val = tf.reduce_mean(tensor)

# Determine the maximum value
max_val = tf.reduce_max(tensor)

# Find the minimum value
min_val = tf.reduce_min(tensor)
```

Gradient Tape

$$\frac{dy}{dX} = 2X = 2 * \underbrace{\begin{bmatrix} 3. & 3. & 3. \\ 3. & 3. & 3. \end{bmatrix}}_{2x3} = \underbrace{\begin{bmatrix} 6. & 6. & 6. \\ 6. & 6. & 6. \end{bmatrix}}_{2x3}$$

$$y = r\_sum(X^2 + 2z)$$

$$\frac{dy}{dz} = \overbrace{2*3}^{\text{number of elements in X matrix}} * 2 = 12$$

```python
# Define input variables
x = tf.Variable(tf.fill((2, 3), 3.0))
z = tf.Variable(5.0)

# Start recording the operations
with tf.GradientTape() as tape:
    # Define the calculations
    y = tf.reduce_sum(x * x + 2 * z)

# Extract the gradient for the specific inputs (x and z)
grad = tape.gradient(y, [x, z])

print(f"The gradient of y with respect to x is:\n{grad[0].numpy()}")
print(f"The gradient of y with respect to z is: {grad[1].numpy()}")
```

@tf.function

```
@tf.function
def compute_gradient_conditional(x):
    with tf.GradientTape() as tape:
        if tf.reduce_sum(x) > 0:
            y = x * x
        else:
            y = x * x * x
    return tape.gradient(y, x)

x = tf.constant([-2.0, 2.0])
grad = compute_gradient_conditional(x)
print(f"The gradient at x = {x.numpy()} is {grad.numpy()}")
```