在TensorRT中使用动态形状(Dynamic Shapes)

动态形状(Dynamic Shapes) 是延迟指定部分或全部张量维度直到运行时的能力。动态形状可以通过 C++ 和 Python 接口使用。
以下部分提供了更详细的信息；但是，这里概述了构建具有动态形状的引擎的步骤：

1.网络定义不得具有隐式批次维度。

C++

通过调用创建 `INetworkDefinition`

```
IBuilder::createNetworkV2(1U <<
        static_cast<int>(NetworkDefinitionCreationFlag::kEXPLICIT_BATCH))
```

Python

通过调用创建tensorrt.INetworkDefinition

```
create_network(1 <<
        int(tensorrt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
```

这些调用要求网络没有隐式批处理维度。

2. -1 作为维度的占位符来指定输入张量的每个运行时维度。

3.指定一个或多个优化配置文件，为具有运行时维度的输入指定允许的维度范围，以及自动调整器将优化的维度。有关详细信息，请参阅优化配置文件。

4.要使用引擎：

- 从引擎创建执行上下文，与没有动态形状的情况相同。
- 指定步骤 3 中涵盖输入维度的优化配置文件之一。
- 指定执行上下文的输入维度。设置输入维度后，您可以获得TensorRT针对给定输入维度计算的输出维度。
- Enqueue work。

8.1. Specifying Runtime Dimensions

构建网络时，使用 `-1` 表示输入张量的运行时维度。例如，要创建一个名为 `foo` 的 3D 输入张量，其中最后两个维度在运行时指定，第一个维度在构建时固定，请发出以下命令。

C++

```
networkDefinition.addInput("foo", DataType::kFLOAT, Dims3(3, -1, -1))
```

Python

```
network_definition.add_input("foo", trt.float32, (3, -1, -1))
```

在运行时，您需要在选择优化配置文件后设置输入维度（请参阅优化配置文件）。设输入`foo`的`bindingIndex`为`0`，输入的维度为`[3,150,250]`。在为前面的示例设置优化配置文件后，您将调用：

C++

```
context.setBindingDimensions(0, Dims3(3, 150, 250))
```

Python

```
context.set_binding_shape(0, (3, 150, 250))
```

在运行时，向引擎询问绑定维度会返回用于构建网络的相同维度，这意味着每个运行时维度都会得到`-1`。例如：

C++

```
engine.getBindingDimensions(0) returns a Dims with dimensions {3, -1, -1}
```

Python

```
engine.get_binding_shape(0) returns (3, -1, -1)
```

要获取特定于每个执行上下文的实际维度，请查询执行上下文：

C++

```
context.getBindingDimensions(0) returns a Dims with dimensions {3, 150, 250}.
```

Python

```
context.get_binding_shape(0) returns (3, 150, 250).
```

注意：输入的`setBindingDimensions`的返回值仅表明与为该输入设置的优化配置文件相关的一致性。指定所有输入绑定维度后，您可以通过查询网络输出绑定的维度来检查整个网络在动态输入形状方面是否一致。

```
nvinfer1::Dims out_dim = context->getBindingDimensions(out_index);

if (out_dim.nbDims == -1) {
gLogError << "Invalid network output, this might be caused by inconsistent input shapes." << std::endl;
// abort inference
}
```

8.2. Optimization Profiles

优化配置文件描述了每个网络输入的维度范围以及自动调谐器将用于优化的维度。使用运行时维度时，您必须在构建时创建至少一个优化配置文件。两个配置文件可以指定不相交或重叠的范围。

例如，一个配置文件可能指定最小尺寸`[3,100,200]`，最大尺寸`[3,200,300]`和优化尺寸`[3,150,250]`而另一个配置文件可能指定最小，最大和优化尺寸`[3,200,100]`，`[3,300,400]`，和`[3,250,250]`。

要创建优化配置文件，首先构造一个`IOptimizationProfile`。然后设置最小、优化和最大维度，并将其添加到网络配置中。优化配置文件定义的形状必须为网络定义有效的输入形状。以下是前面提到的第一个配置文件对输入`foo`的调用：

C++

```
IOptimizationProfile* profile = builder.createOptimizationProfile();
profile->setDimensions("foo", OptProfileSelector::kMIN, Dims3(3,100,200);
profile->setDimensions("foo", OptProfileSelector::kOPT, Dims3(3,150,250);
profile->setDimensions("foo", OptProfileSelector::kMAX, Dims3(3,200,300);

config->addOptimizationProfile(profile)
```

Python

```
profile = builder.create_optimization_profile();
profile.set_shape("foo", (3, 100, 200), (3, 150, 250), (3, 200, 300))
config.add_optimization_profile(profile)
```

在运行时，您需要在设置输入维度之前设置优化配置文件。配置文件按照添加的顺序编号，从0开始。请注意，每个执行上下文必须使用单独的优化配置文件。
要选择示例中的第一个优化配置文件，请使用：

C++
调用`context.setOptimizationProfileAsync(0, stream)`

其中stream是在此上下文中用于后续enqueue()或enqueueV2()调用的 CUDA 流。

Python
设置`context.set_optimization_profile_async(0, stream)`

如果关联的 CUDA 引擎具有动态输入，则必须使用唯一的配置文件索引至少设置一次优化配置文件，该唯一配置文件索引未被其他未销毁的执行上下文使用。对于为引擎创建的第一个执行上下文，隐式选择配置文件 0。

可以调用`setOptimizationProfileAsync()`在配置文件之间切换。它必须在当前上下文中的任何`enqueue()`或`enqueueV2()`操作完成后调用。当多个执行上下文同时运行时，允许切换到以前使用但已被具有不同动态输入维度的另一个执行上下文释放的配置文件。

`setOptimizationProfileAsync()`函数替换了现在已弃用的 API `setOptimizationProfile()`版本。使用`setOptimizationProfile()`在优化配置文件之间切换可能会导致后续`enqueue()`或`enqueueV2()`操作操作中的 GPU 内存复制操作。要在入队期间避免这些调用，请改用`setOptimizationProfileAsync()` API。

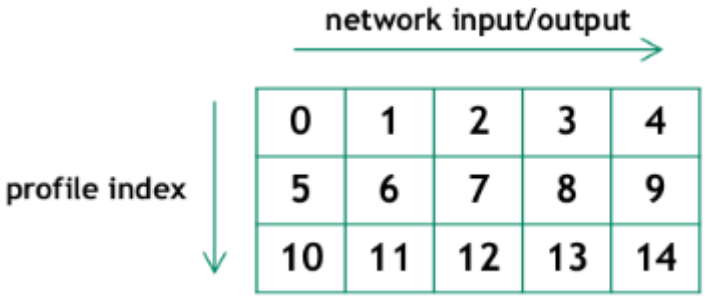
在由多个配置文件构建的引擎中，每个配置文件都有单独的绑定索引。第K个配置文件的输入/输出张量的名称附加了`[profile K]`，其中K以十进制表示。例如，如果`INetworkDefinition`的名称为“`foo`”，并且`bindingIndex`指的是优化配置文件中索引为`3`的张量，则`engine.getBindingName ( bindingIndex )`返回“`foo [profile 3]`”。

同样，如果使用`ICudaEngine::getBindingIndex(name)`获取第一个配置文件（`K=0`）之外的配置文件 K 的索引，请将“`[profile K]`”附加到`INetworkDefinition`中使用的名称。例如，如果张量在`INetworkDefinition`中被称为“`foo`”，则`engine.getBindingIndex ( “ foo [profile 3] ” )`在优化配置文件`3`中返回张量“`foo`”的绑定索引。

始终省略K=0的后缀。

8.2.1. Bindings For Multiple Optimization Profiles

考虑一个具有四个输入、一个输出、在`IBuilderConfig`中具有三个优化配置文件的网络。该引擎有`15`个绑定，每个优化配置文件有`5`个，在概念上组织为一个表：



每行都是一个配置文件。表中的数字表示绑定索引。第一个配置文件的绑定索引为 0..4，第二个配置文件为 5..9，第三个配置文件为 10..14。

对于绑定属于第一个配置文件但指定了另一个配置文件的情况，接口具有“自动更正”功能。在这种情况下，TensorRT 会警告错误，然后从同一列中选择正确的绑定索引。

为了向后半兼容，接口在绑定属于第一个配置文件但指定了另一个配置文件的情况下具有“自动更正”功能。在这种情况下，TensorRT 会警告错误，然后从同一列中选择正确的绑定索引。

8.3. Layer Extensions For Dynamic Shapes

一些层具有允许指定动态形状信息的可选输入，并且有一个新层IShapeLayer用于在运行时访问张量的形状。此外，一些层允许计算新的形状。下一节将讨论语义细节和限制。以下是与动态形状结合使用时可能有用的内容的摘要。

`IShapeLayer`输出一个包含输入张量尺寸的一维张量。例如，如果输入张量的维度为`[2,3,5,7]`，则输出张量是包含`{2,3,5,7}`的四元素一维张量。如果输入张量是标量，则它的维度为[]，输出张量是包含{}的零元素一维张量。

`IResizeLayer`接受包含所需输出尺寸的可选第二个输入。

`IShuffleLayer`接受包含重塑尺寸的可选第二个输入。例如，以下网络将张量Y重塑为与X具有相同的维度：

C++

```
auto* reshape = networkDefinition.addShuffle(Y);
reshape.setInput(1, networkDefintion.addShape(X)->getOutput(0));
```

Python

```
reshape = network_definition.add_shuffle(y)
reshape.set_input(1, network_definition.add_shape(X).get_output(0))
```

`ISliceLayer`接受可选的第二、第三和第四个输入，其中包含开始、大小和步幅。

`IConcatenationLayer, IElementWiseLayer, IGatherLayer, IIdentityLayer, and IReduceLayer`

可用于对形状进行计算并创建新的形状张量。

8.4. Restrictions For Dynamic Shapes

由于层的权重具有固定大小，因此会出现以下层限制：

- `IConvolutionLayer`和`IDeconvolutionLayer`要求通道维度是构建时常数。
- `IFullyConnectedLayer`要求最后三个维度是构建时常量。
- `Int8`要求通道维度是构建时常数。
- 接受额外形状输入的层（`IResizeLayer`、`IShuffleLayer`、`ISliceLayer`）要求额外的形状输入与最小和最大优化配置文件的尺寸以及运行时数据输入的尺寸兼容；否则，它可能导致构建时或运行时错误。

必须是构建时常量的值不必是 API 级别的常量。 TensorRT 的形状分析器通过进行形状计算的层进行逐个元素的常数传播。常量传播发现一个值是构建时常量就足够了。

8.5. Execution Tensors vs. Shape Tensors

使用动态形状的引擎采用两阶段执行策略。

1. 计算所有张量的形状
2. 将工作流式传输到 GPU。

阶段 1 是隐含的，由需求驱动，例如在请求输出维度时。第 2 阶段与之前版本的TensorRT 相同。两阶段执行对动态性施加了一些限制，这些限制对于理解是很重要的。

关键限制是：

- 张量的等级必须在构建时确定。
- 张量是执行张量、形状张量或两者兼而有之。归类为形状张量的张量受到限制。

执行张量是传统的TensorRT张量。形状张量是与形状计算相关的张量。它必须是`0D`或`1D`，类型为`Int32`、`Float`或`Bool`，并且其形状必须在构建时可确定。例如，有一个`IShapeLayer`，其输出是一维张量，其中包含输入张量的维度。输出是一个形状张量。`IShuffleLayer`接受一个可选的第二个输入，可以指定重塑尺寸。第二个输入必须是一个形状张量。

有些层在它们处理的张量类型方面是“多态的”。例如，`IElementWiseLayer`可以将两个`INT32`执行张量相加或将两个`INT32`形状张量相加。张量的类型取决于其最终用途。如果总和用于重塑另一个张量，那么它就是一个“形状张量”。

8.5.1. Formal Inference Rules

TensorRT 用于对张量进行分类的形式推理规则基于类型推理代数。令E表示执行张量， S表示形状张量。

IActivationLayer具有：

`IActivationLayer: E → E`

因为它将执行张量作为输入，将执行张量作为输出。`IElementWiseLayer`在这方面是多态的，有两个特点:

`IElementWiseLayer: S × S → S, E × E → E`

为简洁起见，让我们采用约定t是表示任一类张量的变量，并且特征中的所有t都指同一类张量。然后，前面的两个特征可以写成一个单一的多态特征：

`IElementWiseLayer: t × t → t`

双输入`IShuffleLayer`有一个形状张量作为第二个输入，并且相对于第一个输入是多态的：

`IShuffleLayer (two inputs): t × S → t`

`IConstantLayer`没有输入，但可以产生任何一种张量，所以它的特征是：

`IConstantLayer: → t`

`IShapeLayer`的特征允许所有四种可能的组合E→E 、 E→S 、 S→E和S→S ，因此可以用两个自变量编写：

`IShapeLayer: t1 → t2`

这是完整的规则集，它也可以作为可以使用哪些层来操纵形状张量的参考：

```
IAssertionLayer: S →
IConcatenationLayer: t × t × ...→ t
IIfConditionalInputLayer: t → t
IIfConditionalOutputLayer: t → t
IConstantLayer: → t
IActivationLayer: t → t
IElementWiseLayer: t × t → t
IFillLayer: S → t
IFillLayer: S × E × E → E
IGatherLayer: t × t → t
IIdentityLayer: t → t
IReduceLayer: t → t
IResizeLayer (one input): E → E
IResizeLayer (two inputs): E × S → E
ISelectLayer: t × t × t → t
IShapeLayer: t1 → t2
IShuffleLayer (one input): t → t
IShuffleLayer (two inputs): t × S → t
ISliceLayer (one input): t → t
ISliceLayer (two inputs): t × S → t
ISliceLayer (three inputs): t × S × S → t
ISliceLayer (four inputs): t × S × S × S → t
IUnaryLayer: t → t
all other layers: E × ... → E × ...
```

因为输出可以是多个后续层的输入，所以推断的“类型”不是唯一的。例如，一个`IConstantLayer`可能会馈入一个需要执行张量的用途和另一个需要形状张量的用途。 `IConstantLayer`的输出被归类为两者，可以在两阶段执行的阶段 1 和阶段 2 中使用。

在构建时知道形状张量的等级的要求限制了`ISliceLayer`可用于操纵形状张量的方式。具体来说，如果指定结果大小的第三个参数不是构建时常数，则生成的形状张量的长度在构建时将不再已知，从而打破形状张量对构建时形状的限制.更糟糕的是，它可能被用来重塑另一个张量，打破了在构建时必须知道张量等级的限制。

可以通过方法`ITensor::isShapeTensor()`和`ITensor::isExecutionTensor ()`方法检查 TensorRT 的推理，它为形状张量返回`true`，它为执行张量返回`true`。在调用这些方法之前先构建整个网络，因为它们的答案可能会根据添加的张量用途而改变。

例如，如果一个部分构建的网络将两个张量`T1`和`T2`相加来创建张量`T3`，并且还不需要任何形状张量，则`isShapeTensor()`对所有三个张量都返回`false`。将`IShuffleLayer`的第二个输入设置为`T3`会导致所有三个张量成为形状张量，因为`IShuffleLayer`要求其第二个可选输入是形状张量，如果`IElementWiseLayer`的输出是形状张量，那么它的输入也是形状张量。

8.6. Shape Tensor I/O (Advanced)

有时需要使用形状张量作为网络 I/O 张量。例如，考虑一个仅由`IshuffleLayer`组成的网络。TensorRT 推断第二个输入是一个形状张量。`ITensor::isShapeTensor`为它返回 true。因为它是一个输入形状张量，所以 TensorRT 需要两件事：

- 在构建时：形状张量的优化配置文件值。
- 在运行时：形状张量的值。

输入形状张量的形状在构建时始终是已知的。这是需要描述的值，因为它们可用于指定执行张量的维度。

可以使用`IOptimizationProfile::setShapeValues`设置优化配置文件值。类似于必须为具有运行时维度的执行张量提供最小、最大和优化维度的方式，必须在构建时为形状张量提供最小、最大和优化值。

对应的运行时方法是`IExecutionContext::setInputShapeBinding`，它在运行时设置形状张量的值。

因为“执行张量”与“形状张量”的推断是基于最终用途，所以 TensorRT无法推断网络输出是否为形状张量。您必须通过`INetworkDefinition::markOutputForShapes`方法告诉它。

除了让您输出形状信息以进行调试外，此功能对于编写引擎也很有用。例如，考虑构建三个引擎，每个引擎用于子网络`A、B、C`，其中从`A 到 B 或 B 到 C`的连接可能涉及形状张量。逆序构建网络：`C、B、A`。构建网络 C 后，可以使用`ITensor::isShapeTensor`判断输入是否为形状张量，并使用`INetworkDefinition::markOutputForShapes`标记网络中对应的输出张量B.然后检查B的哪些输入是形状张量，并在网络A中标记对应的输出张量。

网络边界处的形状张量必须具有`Int32`类型。它们不能具有`Float`或`Bool`类型。`Bool`的一种解决方法是使用`Int32`作为 I/O 张量，带有`0`和`1`，并且：

- 通过`ElementWiseOperation::kGREATER`转换为Bool，即`x > 0`。
- 通过`ISelectLayer`从`Bool`转换，即`y ? 1: 0`。

8.7. INT8 Calibration With Dynamic Shapes

要为具有动态形状的网络运行 INT8 校准，必须设置校准优化配置文件。使用配置文件的`kOPT`值执行校准。校准输入数据大小必须与此配置文件匹配。

要创建校准优化配置文件，首先，构造一个`IOptimizationProfile`，其方式与创建一般优化配置文件的方式相同。然后将配置文件设置为配置：

C++

```
config->setCalibrationProfile(profile)
```

Python

```
config.set_calibration_profile(profile)
```

校准配置文件必须有效或为 `nullptr` 。 `kMIN` 和 `kMAX` 值被 `kOPT` 覆盖。要检查当前校准配置文件，请使用l `BuilderConfig::getCalibrationProfile` 。
此方法返回指向当前校准配置文件的指针，如果未设置校准配置文件，则返回 `nullptr` 。为具有动态形状的网络运行校准时， `getBatchSize()` 校准器方法必须返回1 。

注意：如果未设置校准优化配置文件，则使用第一个网络优化配置文件作为校准优化配置文件。