1 ADDITIONAL EXAMPLES: TEMPORAL CONTRACTS

Temporal Contracts Sometimes DL APIs need to be called in a particular order to get the expected output. So, it is necessary to capture the API calling order (i.e., temporal properties) to prevent such kinds of performance-related issues. Traditional DbC techniques are insufficient to specify the order-related properties of an API method. For example, the batch normalization layer should be used before the dropout so that the "batchnorm" can compute the moving average and variance on the dropped outputs of the layer [1]. To capture such temporal properties, we introduce three *ML variable* for specifying the APIs and indicating the position of those APIs. We can inform the DL Contract checker through the *context* variable to check these temporal properties. In the following example, we can specify contract using `layer_1, layer_2, layer_position` *ML variable*.

```
@contract(context='temporal', layer_1='batch_normalization', layer_position ='before', layer_2 = 'dropout'
def compile(self, ...):
```

Example 1.1: Temporal contract on *Keras* `Compile` API

After executing buggy code with *DL Contract* enabled *Keras* library, we obtain the following contract violation message, which informs developers about the correct API calling order.

```
Contract violation for Model:Compile(). Contract violated: batch_normalization layer should be before dropout
layer.
```

Consider another example performance-affecting API calling order for a DNN. For instance, the dropout layer must be placed after the maximum pooling layer to be more effective [1]. Because dropping out activation before the pooling could have no effect except where the units correspond to maximums within the input pooling windows, the max-pooling would keep only these maximums as inputs for the next layers. To capture these temporal properties, we can specify using the same set of *ML variable* `layer_1, layer_2, layer_position` using the `dropout, max_pooling2d` which has been abstracted by *ML variable*. The library developer needs to specify the following contract on top of model compilation API e.g., `Compile`. By adding contracts to the temporal properties of a DNN, developers can detect mistakes early in the pipeline. Also, they can receive appropriate debugging information without the need for additional bug detection tools.

```
@contract(context='temporal', layer_1='dropout', layer_position ='after', layer_2 = 'max_pooling2d'
def compile(self, ...):
```

Example 1.2: Temporal contract on *Keras* `Compile` API

In case of a temporal contract violation on the dropout layer, the DL Application Developers using *DL Contract* enabled *Keras* library will receive the contract violation message below.

```
Contract violation for Model:Compile(). Contract violated: dropout layer should be after max_pooling2d layer.
```

REFERENCES

[1] Amin Nikanjam, Houssem Ben Braiek, Mohammad Mehdi Morovati, and Foutse Khomh. 2021. Automatic Fault Detection for Deep Learning Programs Using Graph Transformations. *ACM Trans. Softw. Eng. Methodol.* 30, 5 (2021), 26 pages.