Deep Learning Engineering - Robust network convolution block

OVERVIEW:

One way to design a flexible deep neural-network codebase is to construct networks out of configurable blocks. Each such network block can be designed as a class, with parameters to configure the block's layer configuration, sizes, activation types, etc.
In this exercise you're provided with a stripped down version of a convolution block (*conv_block)* by a DL researcher with poor coding practices that might have made some mistakes.
This class allows chaining of common layers, operations, etc. making it easier and quicker to prototype new network architectures.

GOAL:

The goal of this exercise is to review the code and improve it to a point where it's ready to be integrated into a larger ML code base. The end result should include safeguards and be robust to human error, which may cause the code to fail or elicit unintended behavior.
Using this exercise, we would like to learn how you think about designing and building robust DL code. In order to get to know you (and only you) better, *please complete this assignment by yourself.*

`CONVBLOCK` DESCRIPTION:

1. Code: The code for this exercise can be found here:
2. Code structure: The attached code includes the following classes (and files):
    a. `ConvBlock` main class (conv_block.py)
    b. Layers:
        i. `ConvLayer` (layers/convolutions.py)
        ii. `ActivationLayer` (layers/activation.py)
        iii. `PoolLayer` (layers/pooling.py)
3. Input parameters: The relevant input parameters and their types:
4.

```
    Parameter                 Type

    in_channels,              (int)
    out_channels,             (int)
    kernel_size=3,            (int | tuple)
    padding='SAME',           (str)
    stride=1,                 (int | tuple)
    dilation=1,               (int | tuple)
    tensor_type='2d',         (str)
    conv_block='ConvBnActiv', (str)
    causal=False,             (bool)
    batch_norm=True,          (bool)
    separable=False,          (bool)
    activation='relu',        (str)
```

```
    dropout=0.0,                (float)
    conv_init='kaiming',        (str)
    bn_init='default'):         (str)
```

a. *Additional details:*

i. *conv_block*: String that defines which operations, and their order, a given `ConvBlock` object implements. An arbitrary number of operations is allowed, but it must include at least one convolution. This string defines the operations as a concatenated string *list* (separated by) caps. E.g. `'ConvBnActiv'` defines a convolution => batch norm => activation `ConvBlock`.

ii. *activation*: String that defines the non-linearity(ies) in the chain of operations.

INSTRUCTIONS:

1. Misuse robustness:
   a. Make any code changes necessary to make sure the code:
      i. Doesn't allow any human error by providing the wrong inputs.
      ii. Doesn't allow any unintended behavior due to mistakes in the inputs.
2. Unit tests:
   a. Build unit-test(s) to confirm the correctness of the primary convolution layer's, `ConvLayer` (convolutions.py), functionality.

DELIVERABLES:

a. Your enhanced code, in a form we can run easily.
b. A summary of the rationale behind your changes.
c. All deliverables should be uploaded to a Git repository.