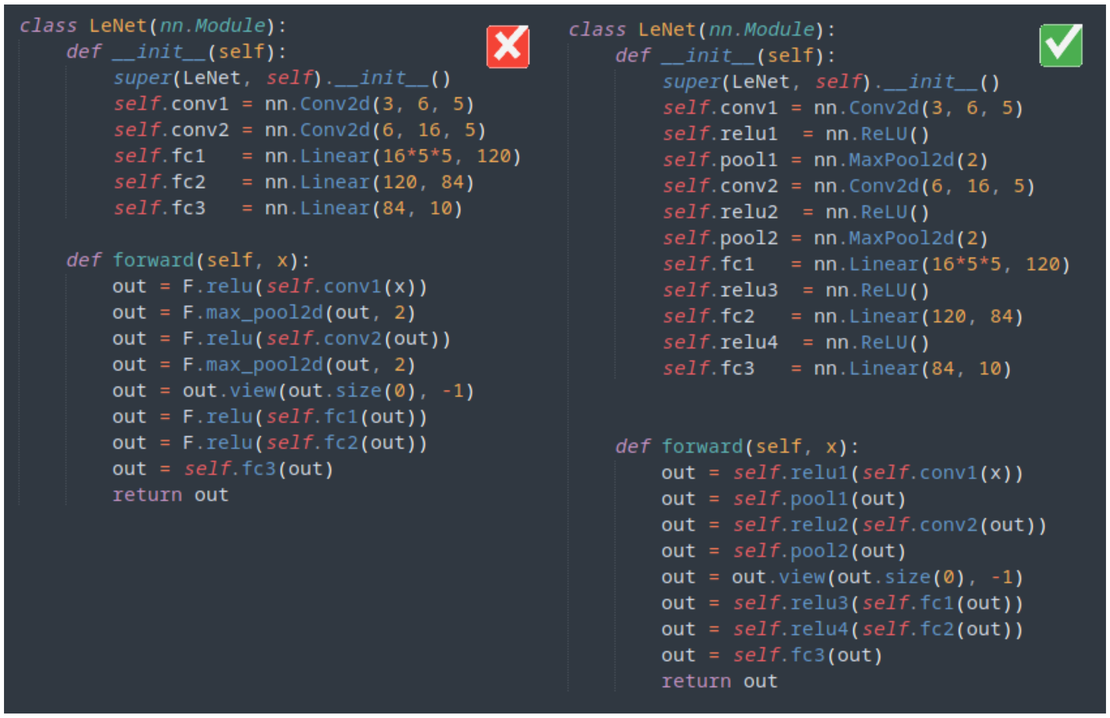
Brevitas Converter

Brevitas Converter are a set of Python3 scripts which turn a Convolutional Neural Network's description file using PyTorch's Framework to its quantized equivalent using Brevita's quantized layer functions to be used for quantization aware training. It is also possible to transfer the weights to the quantized net in order to reduce most of the training time, since Brevita's quantized layers take a considerable amount of time more to train than Pytorch's layers, this makes fine-tuning of quantization aware training faster.

brevitasConverter.py

Before running these scripts, your CNN description file must be in accordance with some rules:

- All layers must be separately defined in the initiation function, as shown in Figure 1.

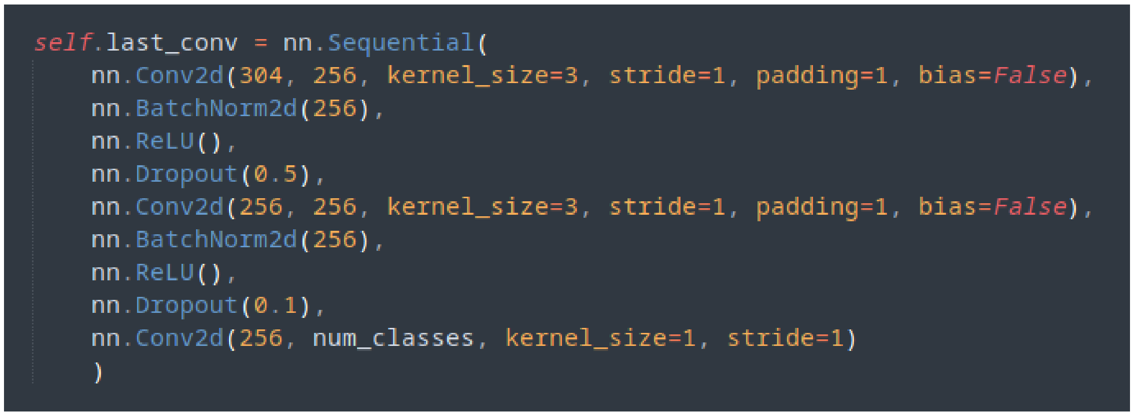


```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1   = nn.Linear(16*5*5, 120)
        self.fc2   = nn.Linear(120, 84)
        self.fc3   = nn.Linear(84, 10)

    def forward(self, x):
        out = F.relu(self.conv1(x))
        out = F.max_pool2d(out, 2)
        out = F.relu(self.conv2(out))
        out = F.max_pool2d(out, 2)
        out = out.view(out.size(0), -1)
        out = F.relu(self.fc1(out))
        out = F.relu(self.fc2(out))
        out = self.fc3(out)
        return out
```

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.relu1  = nn.ReLU()
        self.pool1 = nn.MaxPool2d(2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.relu2  = nn.ReLU()
        self.pool2 = nn.MaxPool2d(2)
        self.fc1    = nn.Linear(16*5*5, 120)
        self.relu3  = nn.ReLU()
        self.fc2    = nn.Linear(120, 84)
        self.relu4  = nn.ReLU()
        self.fc3    = nn.Linear(84, 10)


    def forward(self, x):
        out = self.relu1(self.conv1(x))
        out = self.pool1(out)
        out = self.relu2(self.conv2(out))
        out = self.pool2(out)
        out = out.view(out.size(0), -1)
        out = self.relu3(self.fc1(out))
        out = self.relu4(self.fc2(out))
        out = self.fc3(out)
        return out
```

Figure 1: Left side shows the relu and max pooling layers defined on the forward function and the right side shows every layer defined on the initiation function.

- Sequential blocks must have each layer separated by line and the closing parentheses must also be in a separate line, as shown in Figure 2.



```python
self.last_conv = nn.Sequential(
    nn.Conv2d(304, 256, kernel_size=3, stride=1, padding=1, bias=False),
    nn.BatchNorm2d(256),
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=False),
    nn.BatchNorm2d(256),
    nn.ReLU(),
    nn.Dropout(0.1),
    nn.Conv2d(256, num_classes, kernel_size=1, stride=1)
    )
```

Figure 2: Example of a Sequential function.

- The file's name should be the same as the net's name (case-insensitive) e.g lenet.py and class LeNet or lenet.py and class lenet .

- The whole net must be in just a single file i.e. it's not allowed to call other files functions.

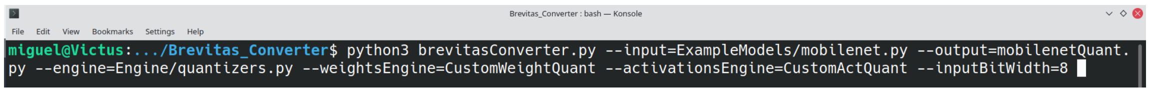
Several examples of Nets are provided in the **ExampleModels** folder.

A file including custom Brevitas quantization classes must also be provided to the scripts, an example of this is included in the **Engine** folder.

Running the **brevitasConverter.py** script requires a few parameters in order to work:

- **input** : The input file i.e. the file to be converted.
- **output** : The name of the converted file.
- **engine** : The quantization classes file.
- **weightsEngine** : The name of the quantization class to be used with weights.
- **activationsEngine** : The name of the quantization class to be used with the activations.
- **inputBitWidth** : The number of bits used in the input feature maps.

An example for running the brevitasConverter script in a terminal is as follows:



```
miguel@Victus:.../Brevitas_Converter$ python3 brevitasConverter.py --input=ExampleModels/mobilenet.py --output=mobilenetQuant.
py --engine=Engine/quantizers.py --weightsEngine=CustomWeightQuant --activationsEngine=CustomActQuant --inputBitWidth=8
```

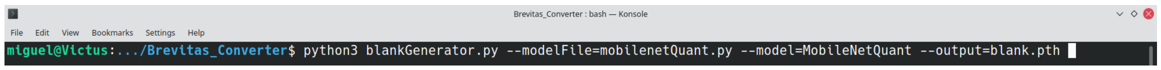
Figure 3: Example of brevitasConverter.py usage on MobileNet.

After running this script a file named after the **output** parameter is generated and it's ready to be used with your preferred training script, since the output file is a drop-in replacement. The name of the converted net is also changed by adding “Quant” to the previous name e.g. LeNet turns to LeNetQuant. The only thing needed is to add the LeNetQuant.setBitWidths(weights,activations) where LeNetQuant is the name of the converted net.

blankGenerator.py

In order convert the weights of an already trained net to a quantized net converted by **brevitasConverter.py**, a blank pth file of the converted net must be created. The script **blankGenerator.py** does this and requires the following parameters to work:

- **modelFile** : The name of the quantized net converted by **brevitasConverter**.
- **model** : The name of the net.
- **output** : The name of the output pth file.



```
miguel@Victus:.../Brevitas_Converter$ python3 blankGenerator.py --modelFile=mobilenetQuant.py --model=MobileNetQuant --output=blank.pth
```

Figure 4: Example of blankGenerator.py usage on MobileNetQuant.

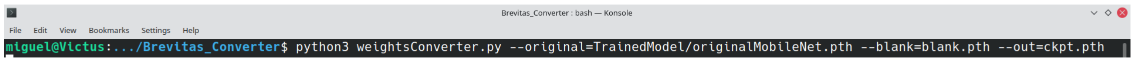
After running this script a file named after the **output** parameter is generated and ready to be used with the **weightsConverter.py** script.

weightsConverter.py

With the blank pth file generated it is now possible to convert the weights of a trained net to the quantized net. In order for this script to work both nets need to have the same names for every layer that uses weights.

Running the **weightsConverter.py** script requires a few parameters in order to work:

- **original** : The pth file of the already trained net.
- **blank** : The blank pth file of the quantized net generated by **blankGenerator.py** .
- **out** : The name of the output pth file.



```
miguel@Victus:.../Brevitas_Converter$ python3 weightsConverter.py --original=TrainedModel/originalMobileNet.pth --blank=blank.pth --out=ckpt.pth
```

Figure 5: Example of weightsConverter.py usage on a trained MobileNet pth file.

With the output pth file generated it is possible to resume training with the already trained weights. Due to the lower precision of the data format and the untrained activations the accuracy will be probably lower than on the trained net but much higher than starting anew.