Exercise 1: Implement ResNet-18

```
import torchvision.models as models
from torchsummary import summary
```

[25] ✓ 0.0s Python

```
resnet_18 = models.resnet18()
input = torch.randn(1, 3, 224, 224)
```

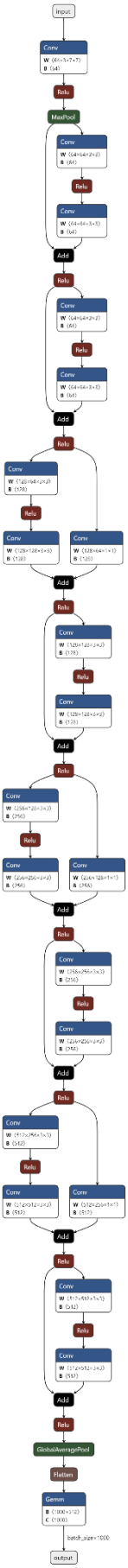
[26] ✓ 0.2s Python

```
summary(resnet_18, (3, 224, 224))
```

[ ] Python

```
torch.onnx.export(resnet_18,
                  input,
                  "resnet_18.onnx",
                  export_params=True,
                  opset_version=10,
                  do_constant_folding=True,
                  input_names = ['input'],
                  output_names = ['output'],
                  dynamic_axes={'input' : {0 : 'batch_size'},
                                'output' : {0 : 'batch_size'}})
```

[31] ✓ 0.7s Python


input
Conv
W (64×3×7×7)
B (64)
Relu
MaxPool
Conv
W (64×64×3×3)
B (64)
Relu
Conv
W (64×64×3×3)
B (64)
Add
Relu
Conv
W (64×64×3×3)
B (64)
Relu
Conv
W (64×64×3×3)
B (64)
Add
Relu
Conv
W (128×64×3×3)
B (128)
Relu
Conv
W (128×128×3×3)
B (128)
Conv
W (128×64×1×1)
B (128)
Add
Relu
Conv
W (128×128×3×3)
B (128)
Relu
Conv
W (128×128×3×3)
B (128)
Add
Relu
Conv
W (256×128×3×3)
B (256)
Relu
Conv
W (256×256×3×3)
B (256)
Conv
W (256×128×1×1)
B (256)
Add
Relu
Conv
W (256×256×3×3)
B (256)
Relu
Conv
W (256×256×3×3)
B (256)
Add
Relu
Conv
W (512×256×3×3)
B (512)
Relu
Conv
W (512×512×3×3)
B (512)
Conv
W (512×256×1×1)
B (512)
Add
Relu
Conv
W (512×512×3×3)
B (512)
Relu
Conv
W (512×512×3×3)
B (512)
Add
Relu
GlobalAveragePool
Flatten
Gemm
B (1000×1)
C (1000)
batch_size×1000
output

Exercise 2: Use thop library to verify the number of parameters & FLOPs of VGG16 and ResNet 101

```
import torch
from torchvision.models import vgg16, resnet101
from thop import profile
from thop import clever_format
```

[32] ✓ 0.0s Python

Params and FLOPs of VGG 16

```
vgg16_ = vgg16()
input = torch.randn(1, 3, 224, 224)
flops, params = profile(vgg16_, inputs=(input, ))
flops, params = clever_format([flops, params], "%.3f")
```

[33] ✓ 4.5s Python

```
[INFO] Register count_convNd() for <class 'torch.nn.modules.conv.Conv2d'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.activation.ReLU'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.pooling.MaxPool2d'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.container.Sequential'>.
[INFO] Register count_adap_avgpool() for <class 'torch.nn.modules.pooling.AdaptiveAvgPool2d'>.
[INFO] Register count_linear() for <class 'torch.nn.modules.linear.Linear'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.dropout.Dropout'>.
```

```
flops, params
```

[34] ✓ 0.0s Python

```
('15.470G', '138.358M')
```

Params and FLOPs of ResNet 101

```
vgg16_ = resnet101()
input = torch.randn(1, 3, 224, 224)
flops, params = profile(vgg16_, inputs=(input, ))
flops, params = clever_format([flops, params], "%.3f")
```

[35] ✓ 1.7s Python

```
[INFO] Register count_convNd() for <class 'torch.nn.modules.conv.Conv2d'>.
[INFO] Register count_normalization() for <class 'torch.nn.modules.batchnorm.BatchNorm2d'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.activation.ReLU'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.pooling.MaxPool2d'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.container.Sequential'>.
[INFO] Register count_adap_avgpool() for <class 'torch.nn.modules.pooling.AdaptiveAvgPool2d'>.
[INFO] Register count_linear() for <class 'torch.nn.modules.linear.Linear'>.
```

```
flops, params
```

[36] ✓ 0.0s Python

```
('7.866G', '44.549M')
```