Lecture 13

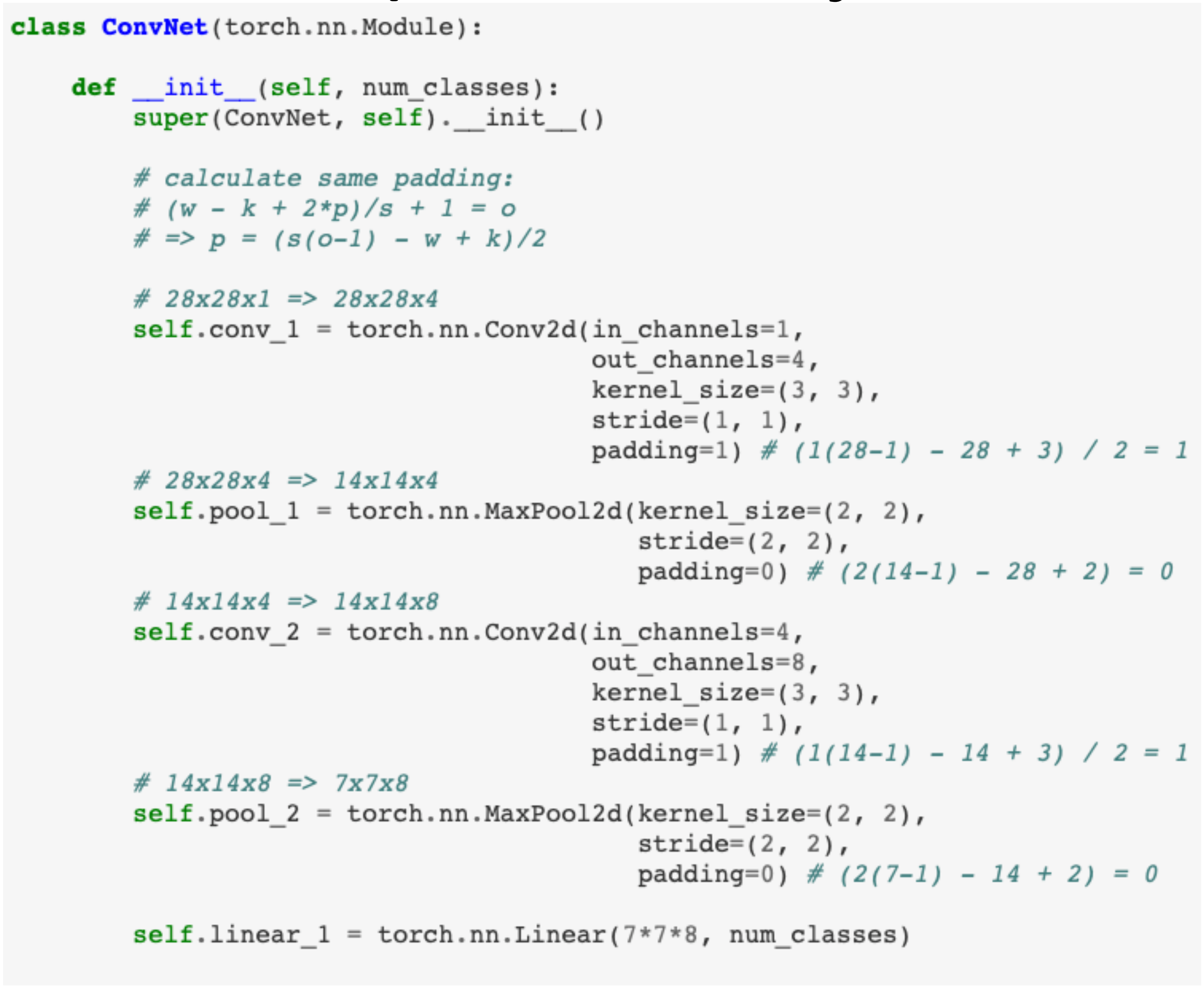
Introduction to Convolutional Neural Networks Part 2

STAT 479: Deep Learning, Spring 2019

Sebastian Raschka

http://stat.wisc.edu/~sraschka/teaching/stat479-ss2019/

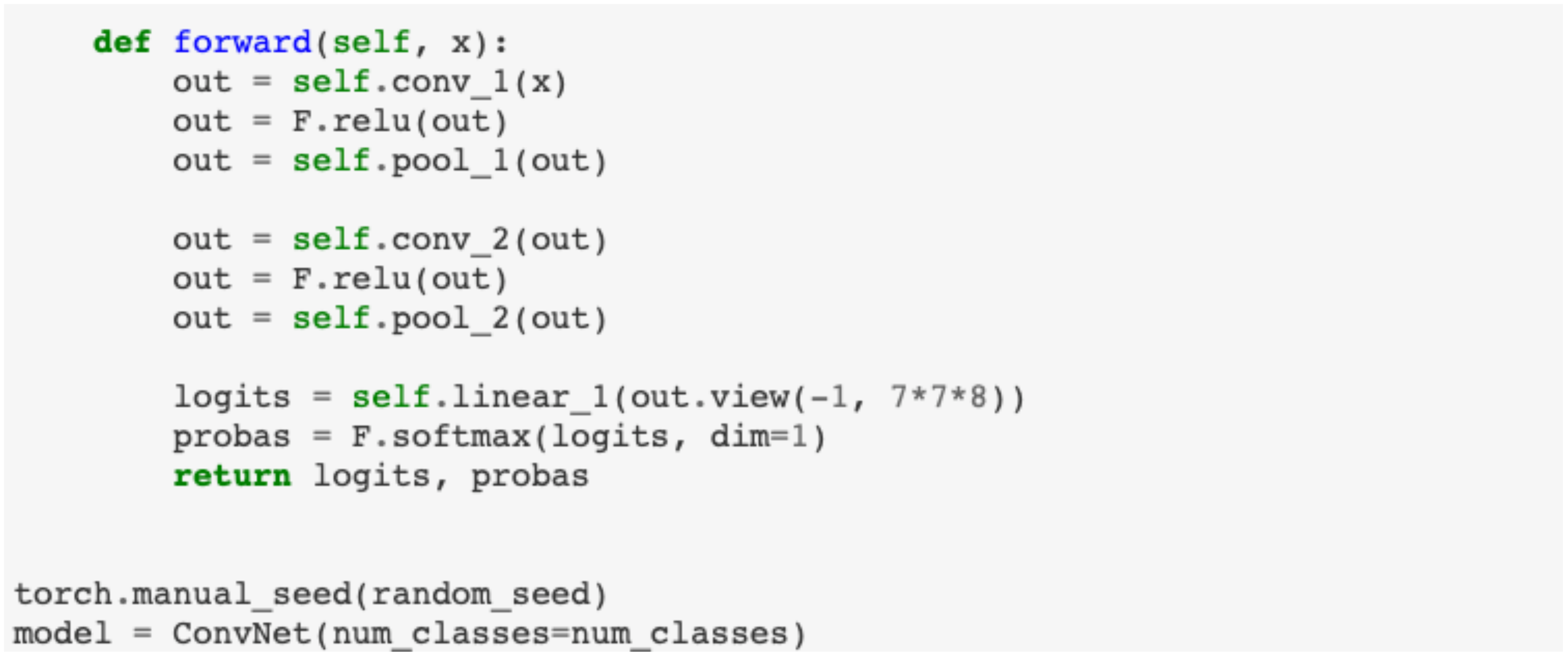
A Simple CNN in PyTorch



```python
class ConvNet(torch.nn.Module):

    def __init__(self, num_classes):
        super(ConvNet, self).__init__()

        # calculate same padding:
        # (w - k + 2*p)/s + 1 = o
        # => p = (s(o-1) - w + k)/2

        # 28x28x1 => 28x28x4
        self.conv_1 = torch.nn.Conv2d(in_channels=1,
                                      out_channels=4,
                                      kernel_size=(3, 3),
                                      stride=(1, 1),
                                      padding=1) # (1(28-1) - 28 + 3) / 2 = 1
        # 28x28x4 => 14x14x4
        self.pool_1 = torch.nn.MaxPool2d(kernel_size=(2, 2),
                                         stride=(2, 2),
                                         padding=0) # (2(14-1) - 28 + 2) = 0
        # 14x14x4 => 14x14x8
        self.conv_2 = torch.nn.Conv2d(in_channels=4,
                                      out_channels=8,
                                      kernel_size=(3, 3),
                                      stride=(1, 1),
                                      padding=1) # (1(14-1) - 14 + 3) / 2 = 1
        # 14x14x8 => 7x7x8
        self.pool_2 = torch.nn.MaxPool2d(kernel_size=(2, 2),
                                         stride=(2, 2),
                                         padding=0) # (2(7-1) - 14 + 2) = 0

        self.linear_1 = torch.nn.Linear(7*7*8, num_classes)
```

(forward method on the next slide)

A Simple CNN in PyTorch



```
    def forward(self, x):
        out = self.conv_1(x)
        out = F.relu(out)
        out = self.pool_1(out)

        out = self.conv_2(out)
        out = F.relu(out)
        out = self.pool_2(out)

        logits = self.linear_1(out.view(-1, 7*7*8))
        probas = F.softmax(logits, dim=1)
        return logits, probas


torch.manual_seed(random_seed)
model = ConvNet(num_classes=num_classes)
```

(model parameters on the previous slide)

Working example:
https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L13_intro-cnn/code/cnn-with-diff-init/default.ipynb

Padding

output size

output size

padding pixels per side

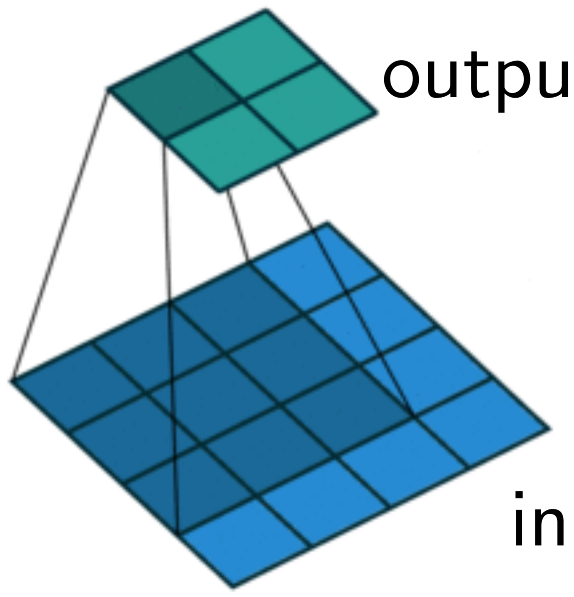
$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

kernel size

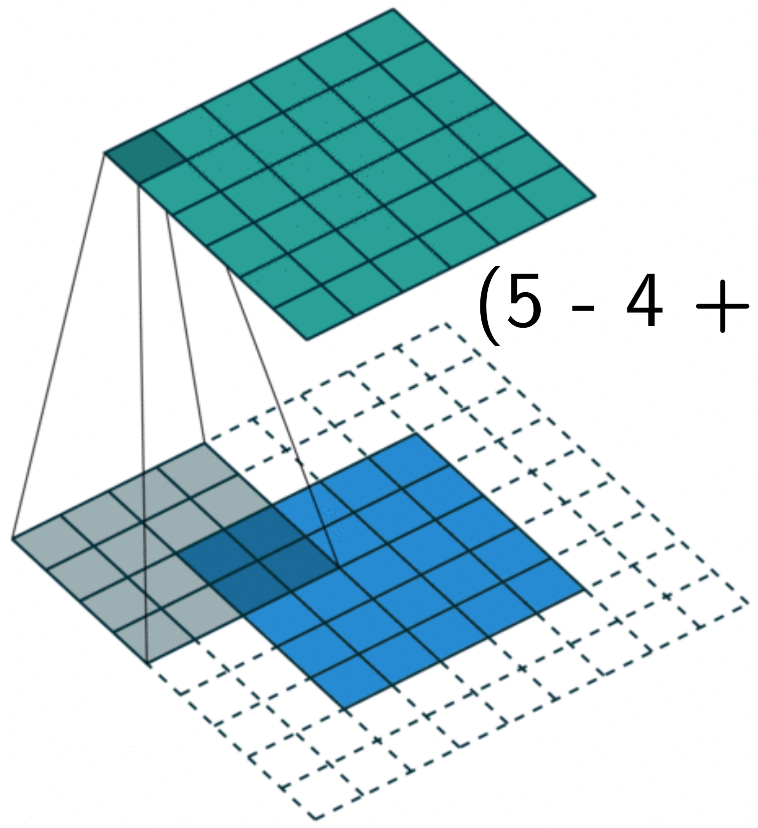
"floor" function

stride

$(4 - 3 + 2*0)/1 + 1 = 2$

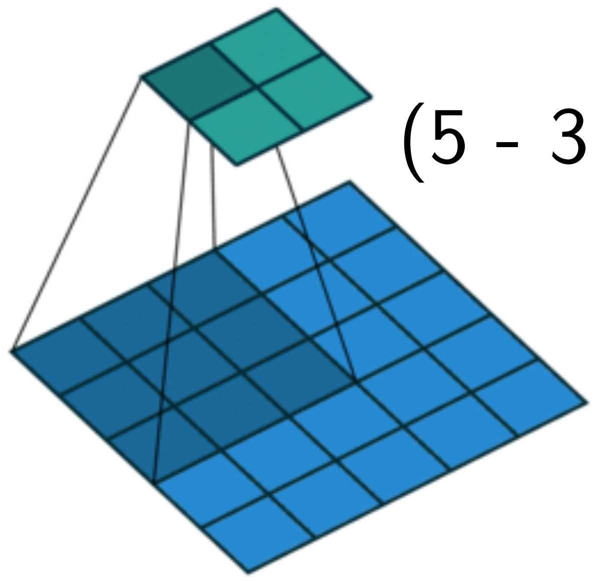


No padding, stride=1



$(5 - 4 + 2*2)/1 + 1 = 6$

padding=2, stride=1



$(5 - 3 + 2*0)/2 + 1 = 2$

No padding, stride=2

Highly recommended:

Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." *arXiv preprint arXiv:1603.07285* (2016).

Padding Jargon

"valid" convolution: no padding (feature map may shrink)

"same" convolution: padding such that the output size is equal to the input size

Common kernel size conventions:
3x3, 5x5, 7x7 (sometimes 1x1 in later layers to reduce channels)

Padding

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

Assume you want to use a convolutional operation with stride 1 and maintain the input dimensions in the output feature map:

How much padding do you need for "same" convolution?

$$o = i + 2p - k + 1$$

$$\Leftrightarrow p = (o - i + k - 1)/2$$

$$\Leftrightarrow p = (k - 1)/2$$

Padding

$$o = i + 2p - k + 1$$

$$\Leftrightarrow p = (o - i + k - 1)/2$$

$$\Leftrightarrow p = (k - 1)/2$$

Probably explains why common kernel size conventions are 3x3, 5x5, 7x7 (sometimes 1x1 in later layers to reduce channels)

Spatial Dropout -- Dropout2D

- Problem with regular dropout and CNNs: Adjacent pixels are likely highly correlated (thus, may not help with reducing the "dependency" much as originally intended by dropout)

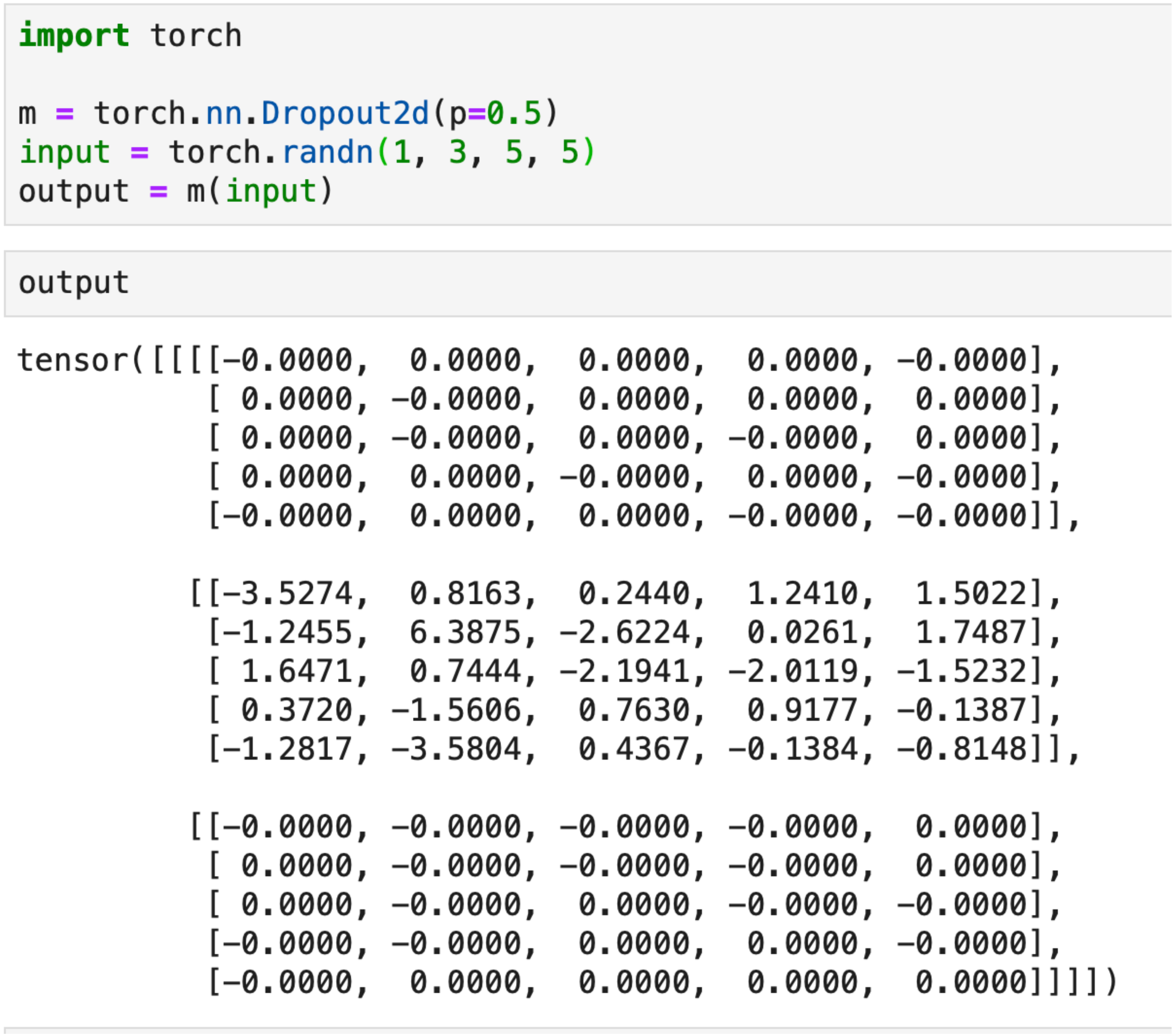
- Hence, it may be better to drop entire feature maps

Idea comes from

Tompson, Jonathan, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. "Efficient object localization using convolutional networks." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 648-656. 2015.

Spatial Dropout -- Dropout2D

- Dropout2d will drop full feature maps (channels)



```python
import torch

m = torch.nn.Dropout2d(p=0.5)
input = torch.randn(1, 3, 5, 5)
output = m(input)
```

```python
output
```

```
tensor([[[[-0.0000,  0.0000,  0.0000,  0.0000, -0.0000],
          [ 0.0000, -0.0000,  0.0000,  0.0000,  0.0000],
          [ 0.0000, -0.0000,  0.0000, -0.0000,  0.0000],
          [ 0.0000,  0.0000, -0.0000,  0.0000, -0.0000],
          [-0.0000,  0.0000,  0.0000, -0.0000, -0.0000]],

         [[-3.5274,  0.8163,  0.2440,  1.2410,  1.5022],
          [-1.2455,  6.3875, -2.6224,  0.0261,  1.7487],
          [ 1.6471,  0.7444, -2.1941, -2.0119, -1.5232],
          [ 0.3720, -1.5606,  0.7630,  0.9177, -0.1387],
          [-1.2817, -3.5804,  0.4367, -0.1384, -0.8148]],

         [[-0.0000, -0.0000, -0.0000, -0.0000,  0.0000],
          [ 0.0000, -0.0000, -0.0000, -0.0000,  0.0000],
          [ 0.0000, -0.0000,  0.0000, -0.0000, -0.0000],
          [-0.0000, -0.0000,  0.0000,  0.0000, -0.0000],
          [-0.0000,  0.0000,  0.0000,  0.0000,  0.0000]]]])
```

Spatial Dropout -- Dropout2D

- Problem with regular dropout and CNNs: Adjacent pixels are likely highly correlated (thus, may not help with reducing the "dependency" much as originally intended by dropout)

- Hence, it may be better to drop entire feature maps

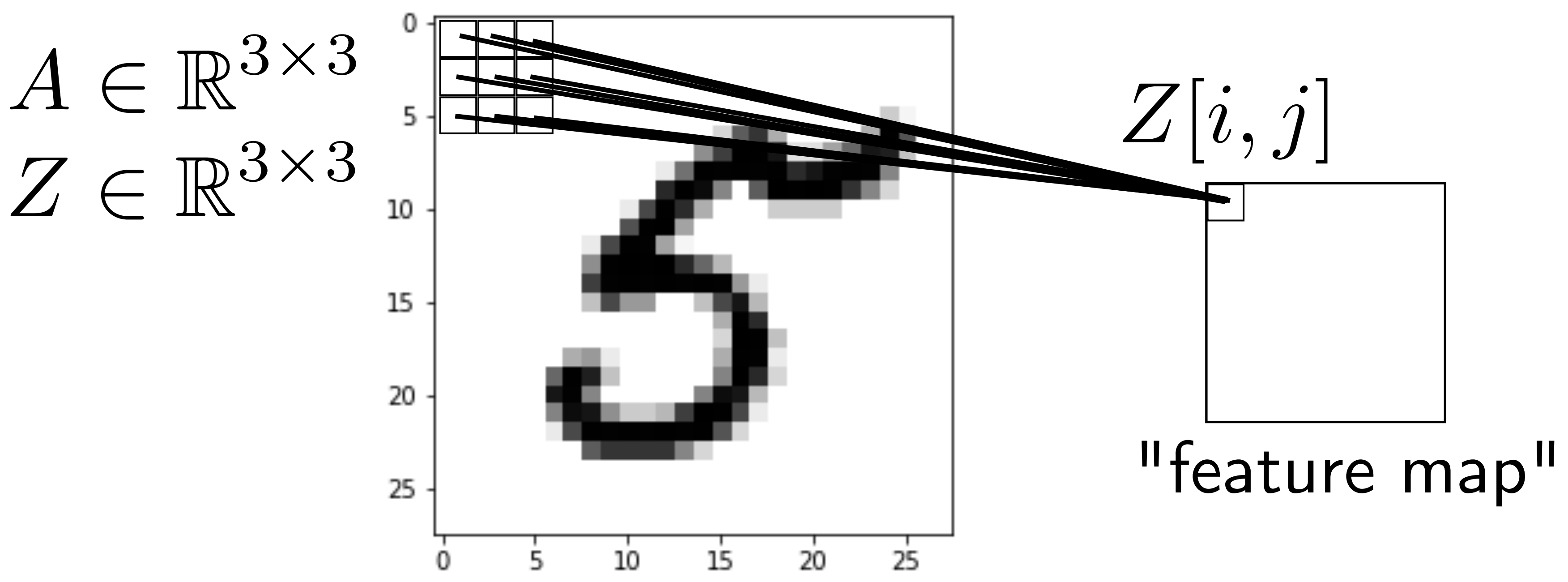
Idea comes from

Tompson, Jonathan, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. "Efficient object localization using convolutional networks." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 648-656. 2015.

Cross-Correlation vs Convolution

Deep Learning Jargon: convolution in DL is actually cross-correlation

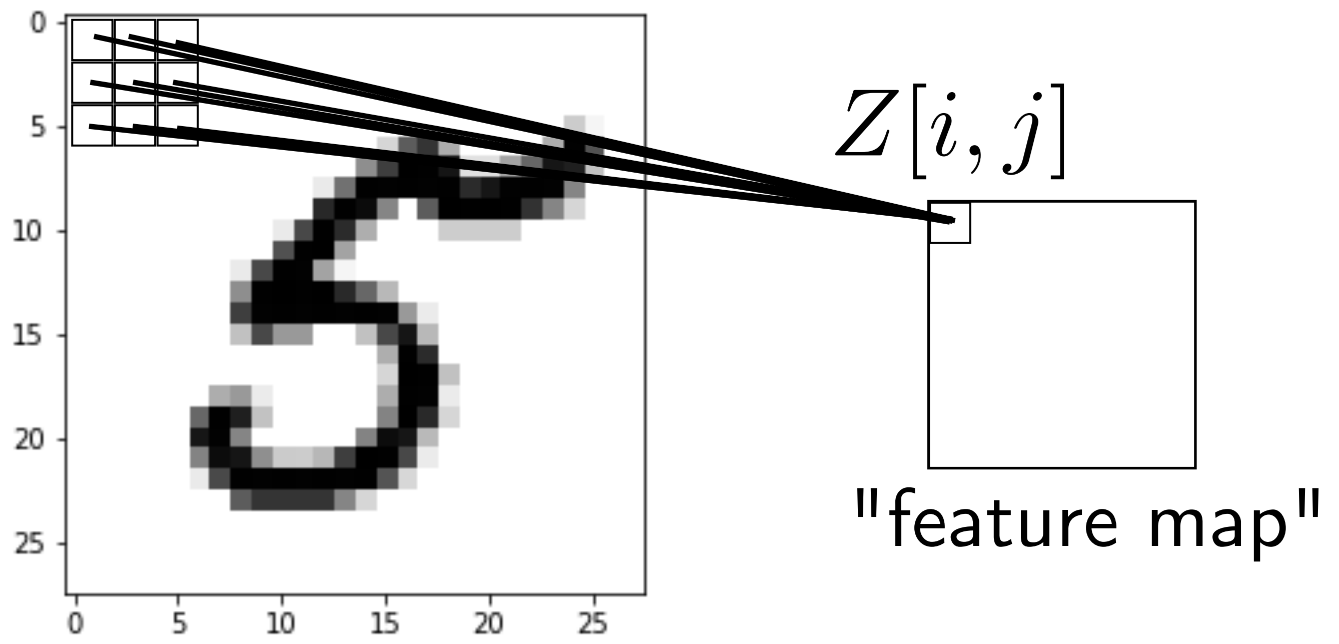
Cross-correlation is our sliding dot product over the image



Cross-Correlation vs Convolution

$A \in \mathbb{R}^{3\times 3}$

$Z \in \mathbb{R}^{3\times 3}$



<u>Cross-Correlation:</u>

$$Z[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} K[u,v] A[i+u, j+v]$$

$$Z[i,j] = K \otimes A$$

Cross-Correlation vs Convolution

Cross-Correlation:

$$Z[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} K[u,v] A[i+u, j+v]$$

$$Z[i,j] = K \otimes A$$

Looping direction indicated in red

1) -1,-1	2) -1,0	3) -1,1
4) 0,-1	5) 0,0	6) 0,1
7) 1,-1	8) 1,0	9) 1,1

Cross-Correlation vs Convolution

Cross-Correlation: $Z[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} K[u,v] A[i+u, j+v]$ $\qquad Z[i,j] = K \otimes A$

Convolution: $$Z[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} K[u,v] A[i-u, j-v]$$

$$Z[i,j] = K * A$$

Basically, we are flipping the kernel (or the receptive field) horizontally and vertically

9) -1,-1	8) -1,0	7) -1,1
6) 0,-1	5) 0,0	4) 0,1
3) 1,-1	2) 1,0	1) 1,1

Looping direction indicated in red

Cross-Correlation vs Convolution

Deep Learning Jargon: convolution in DL is actually cross-correlation

"Real" convolution has the nice associative property:
$(A * B) * C = A * (B * C)$

In DL, we usually don't care about that (as opposed to many traditional computer vision and signal processing applications).

Also, cross-correlation is easier to implement.

Maybe the term "convolution" for cross-correlation became popular, because "Cross-Correlational Neural Network" sounds weird ;)

Computing Convolutions on the GPU

- There are many different approaches to compute (approximate) convolution operations

- DL libraries usually use NVIDIA's CUDA & CuDNN libraries, which implement many different convolution algorithms

- These algorithms are usually more efficient than the CPU variants (convolutions on the CPU e.g., in CPU usually take up much more memory due to the algorithm choice compared to using the GPU)

If you are interested, you can find more info in:

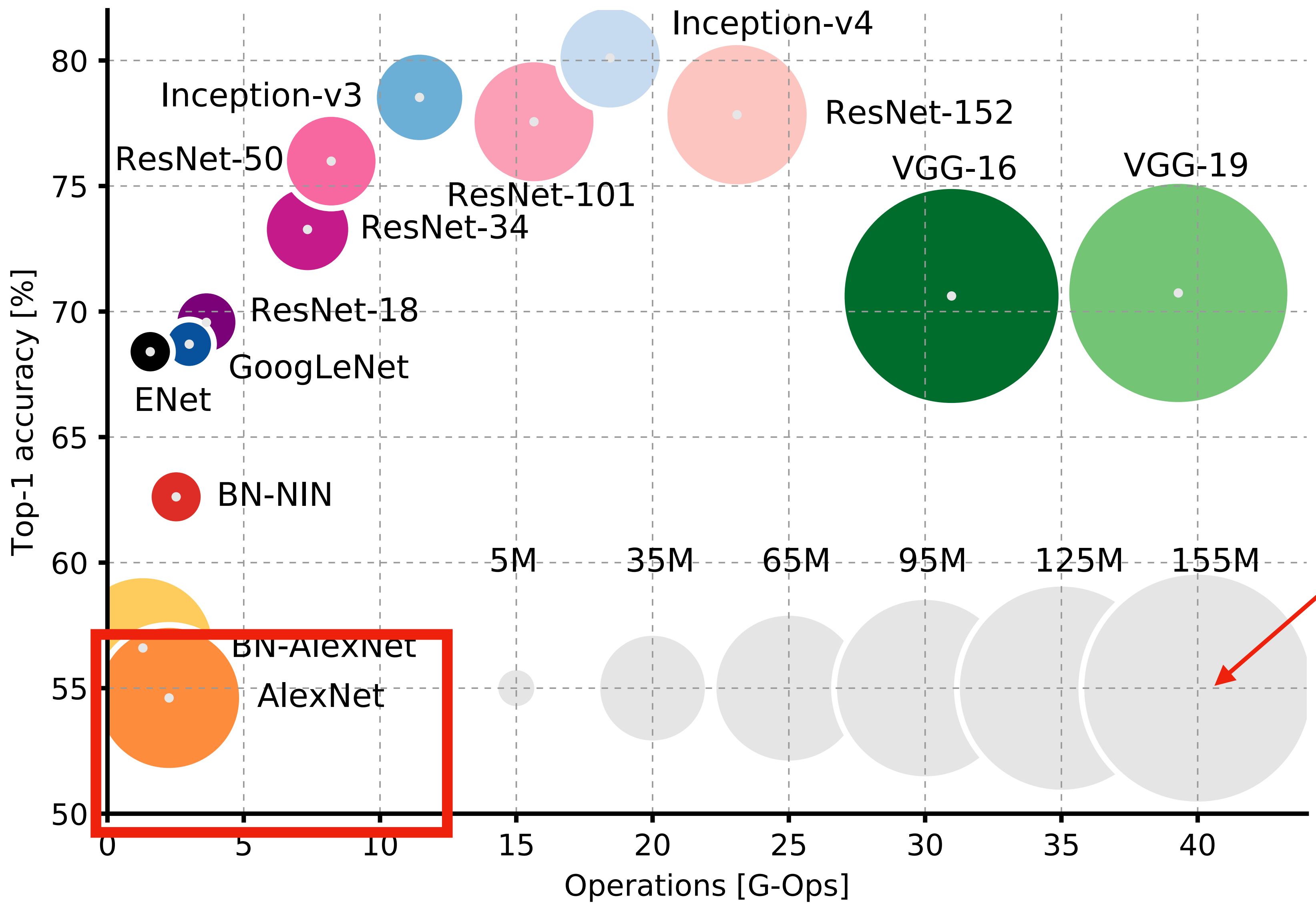
Lavin, Andrew, and Scott Gray. "Fast algorithms for convolutional neural networks." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.
https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Lavin_Fast_Algorithms_for_CVPR_2016_paper.pdf

Computing Convolutions on the GPU

- CuDNN is more geared towards engineers & speed rather than scientists and is unfortunately not deterministic/reproducible by default
- I.e., it determines which convolution algorithm to choose during run-time automatically, based on predicted speeds given the data flow
- For reproducibility and consistent results, I recommend setting the deterministic flag (speed is about the same, often even a bit faster, sometimes a bit slower)

```
import torch
import torch.nn as nn
import torch.nn.functional as F

if torch.cuda.is_available():
    torch.backends.cudnn.deterministic = True
```

Common Architectures Revisited

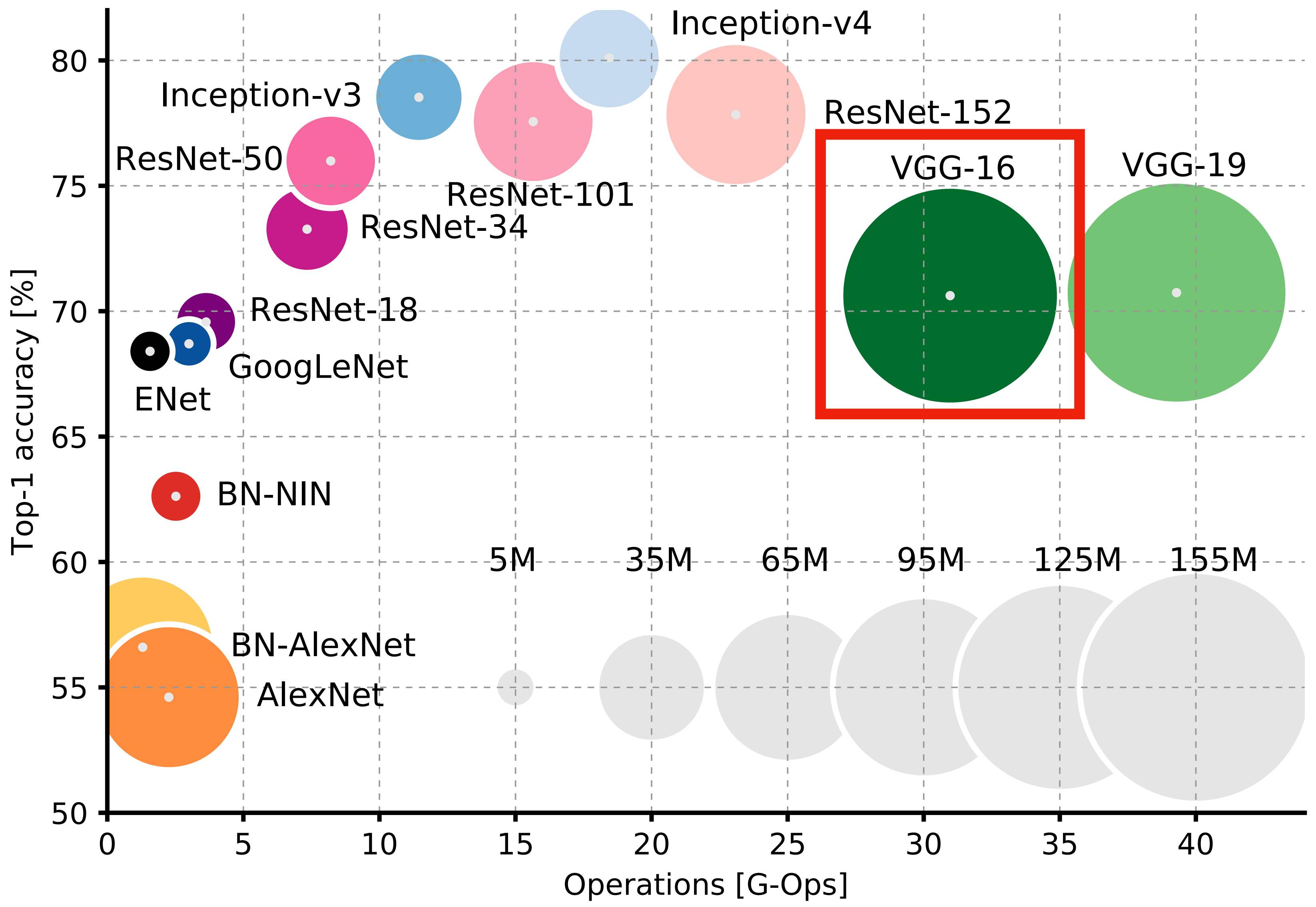


number of model parameters

You will work with this in HW4

Canziani, A., Paszke, A., & Culurciello, E. (2016). An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*.

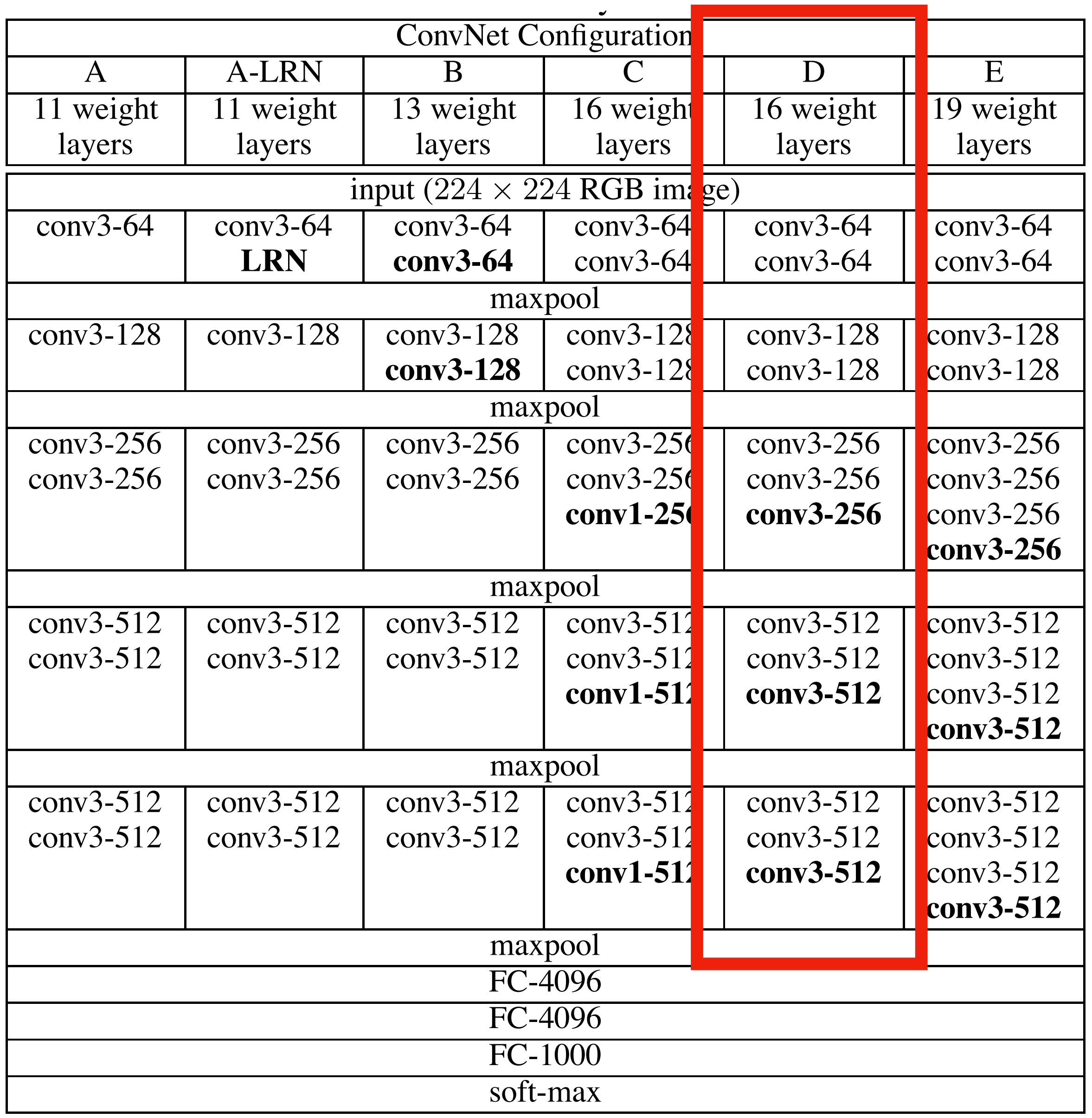
Common Architectures Revisited



Canziani, A., Paszke, A., & Culurciello, E. (2016). An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*.

VGG-16

PyTorch implementation: https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L13_intro-cnn/code/vgg16.ipynb



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-12 conv3-12	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-25 conv3-25 **conv1-25**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-51 conv3-51 **conv1-51**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-51 conv3-51 **conv1-51**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

<u>Advantages:</u>

very simple architecture,

3x3 convs, stride=1,

"same" padding, 2x2 max pooling

<u>Disadvantage:</u>

very large number of parameters

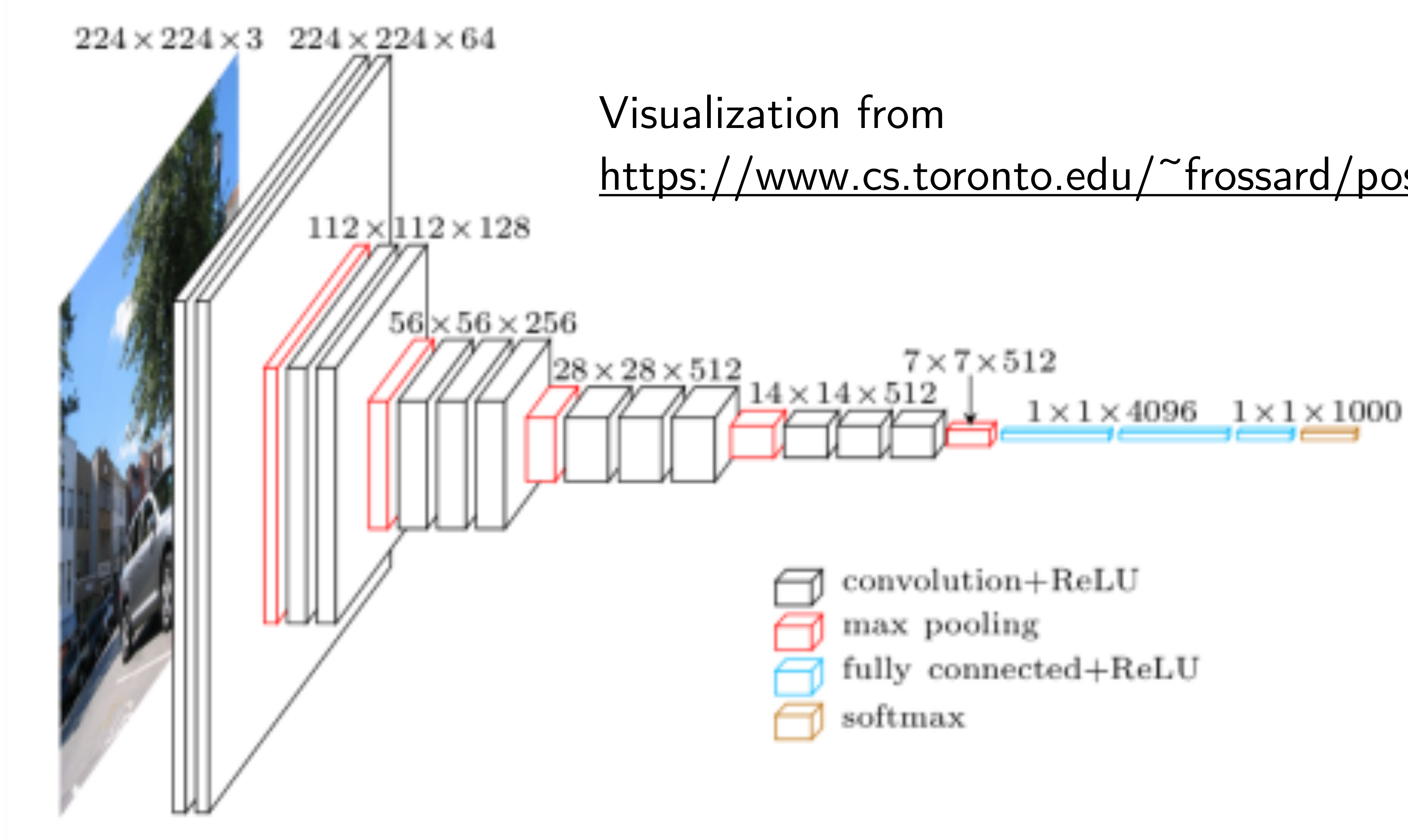
and slow

(see previous slide)

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

VGG-16

PyTorch implementation: https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L13_intro-cnn/code/vgg16.ipynb

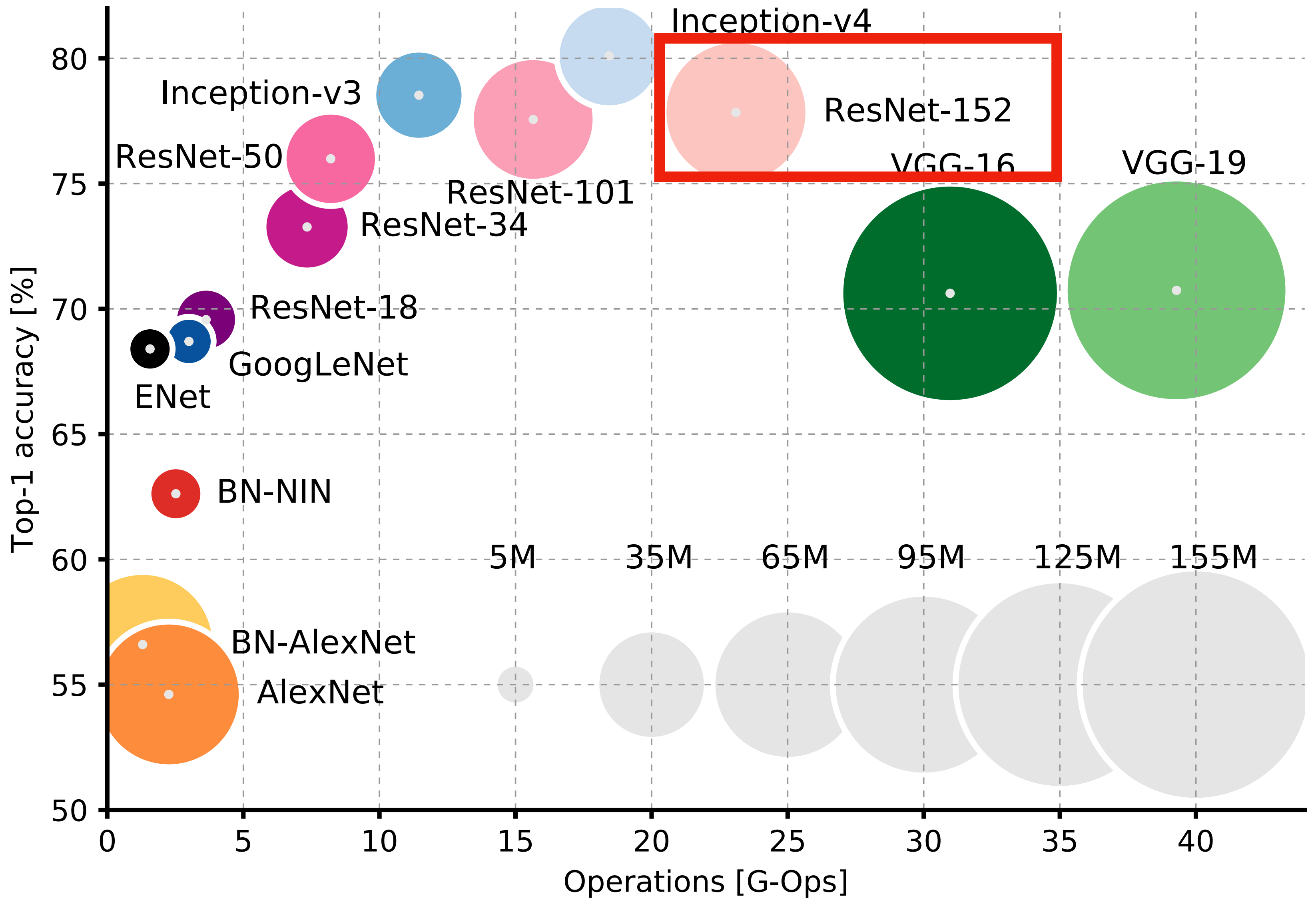


$224 \times 224 \times 3$
$224 \times 224 \times 64$
$112 \times 112 \times 128$
$56 \times 56 \times 256$
$28 \times 28 \times 512$
$14 \times 14 \times 512$
$7 \times 7 \times 512$
$1 \times 1 \times 4096$
$1 \times 1 \times 1000$
convolution+ReLU
max pooling
fully connected+ReLU
softmax

Visualization from
https://www.cs.toronto.edu/~frossard/post/vgg16/

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

Common Architectures Revisited



Canziani, A., Paszke, A., & Culurciello, E. (2016). An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*.

ResNets

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.

With their simple trick of allowing skip connections (the possibility to learn identity functions and skip layers that are not useful), ResNets allow us to to implement very, very deep architectures

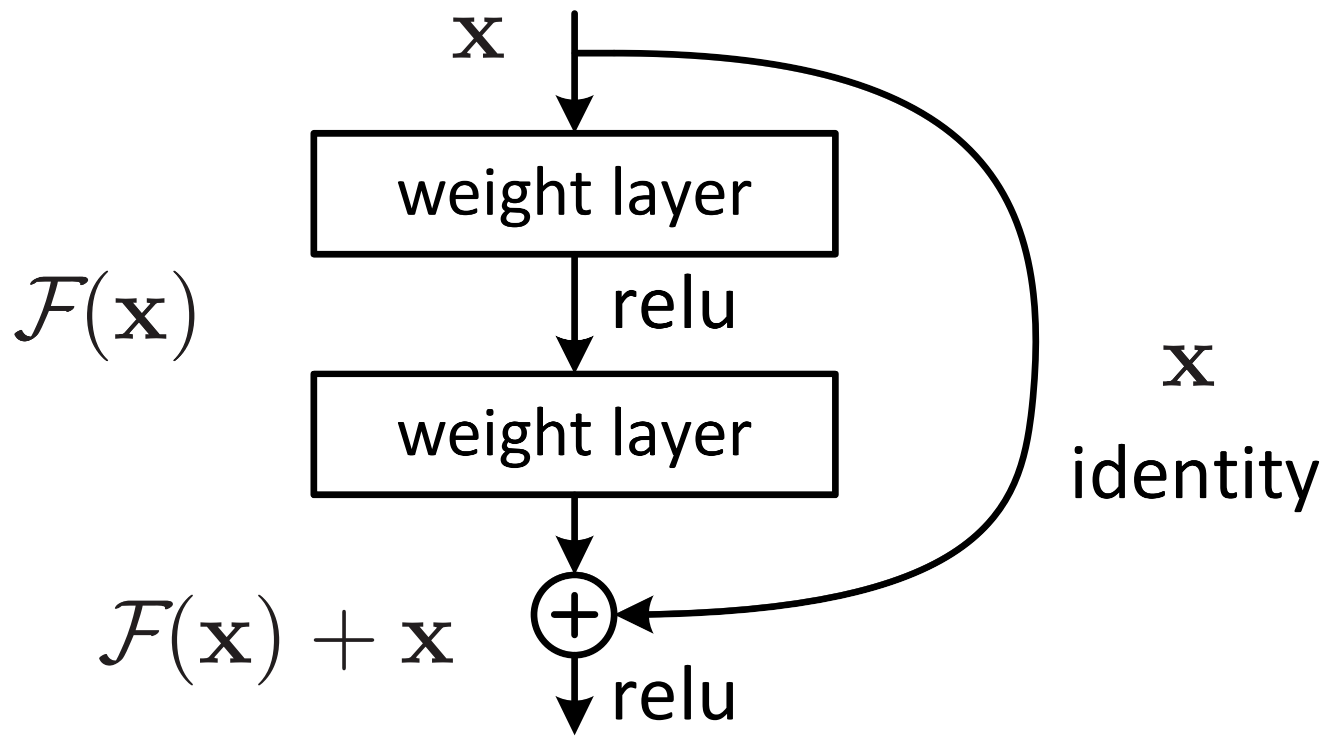
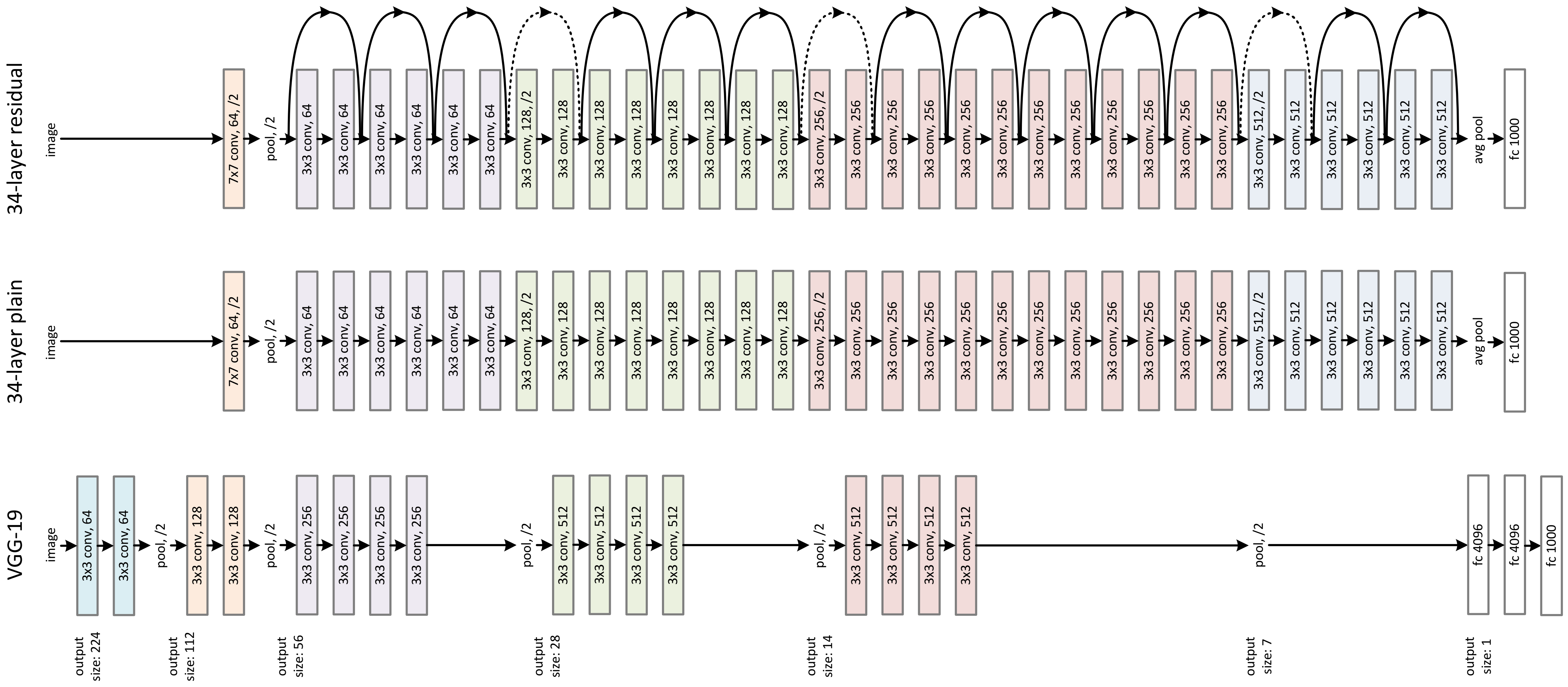


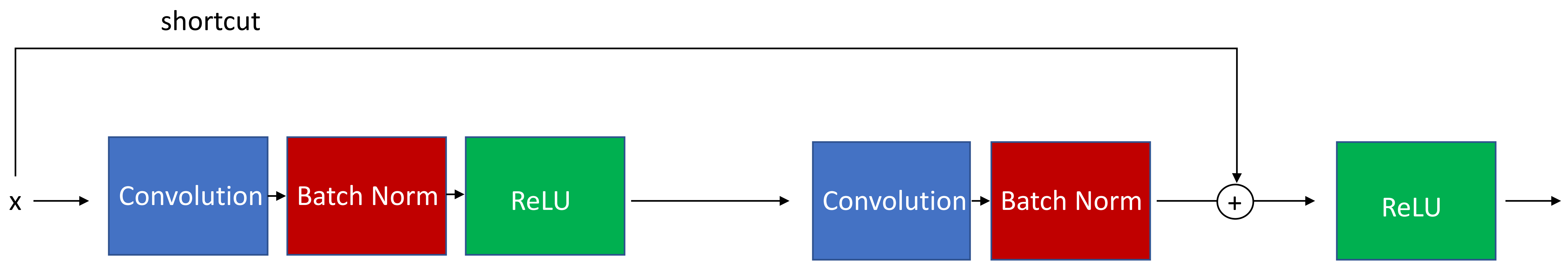
Figure 2. Residual learning: a building block.

ResNets

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.

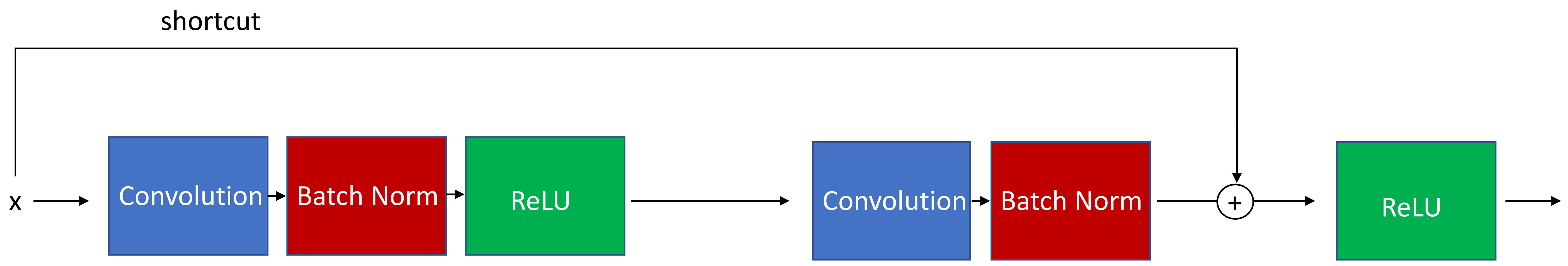


ResNets



In general: $$a^{(l+2)} = \sigma\left(z^{(l+2)} + a^{(l)}\right)$$

ResNets



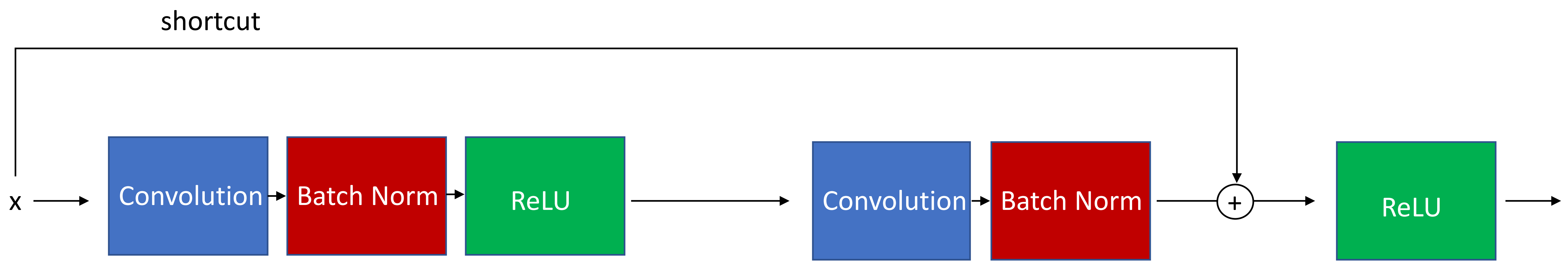
$$
\begin{aligned}
a^{(l+2)} &= \sigma\left(z^{(l+2)} + a^{(l)}\right) \\
&= \sigma\left(a^{(l+1)} W^{(l+2)} + b^{(l+2)} + a^{(l)}\right)
\end{aligned}
$$

If all weights and the bias are zero, then

$$= \sigma\left(a^{(l)}\right) \underset{\nwarrow \text{due to ReLU}}{=} a^{(l)}$$

(identity function)

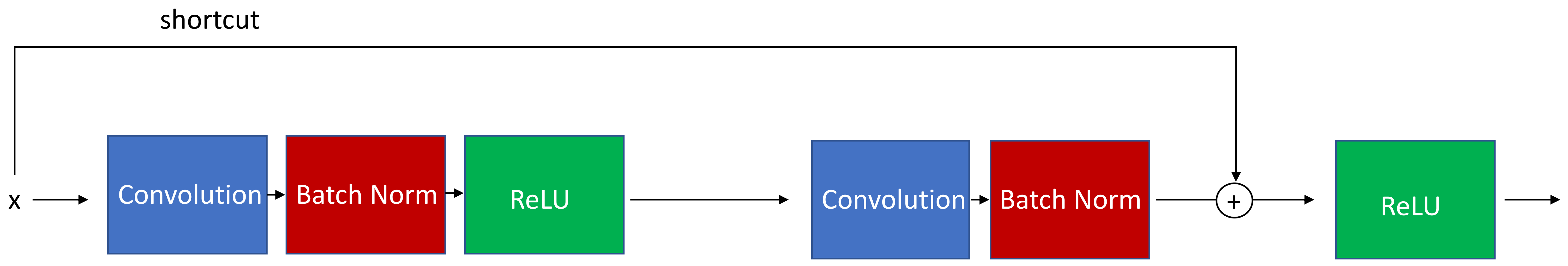
ResNets

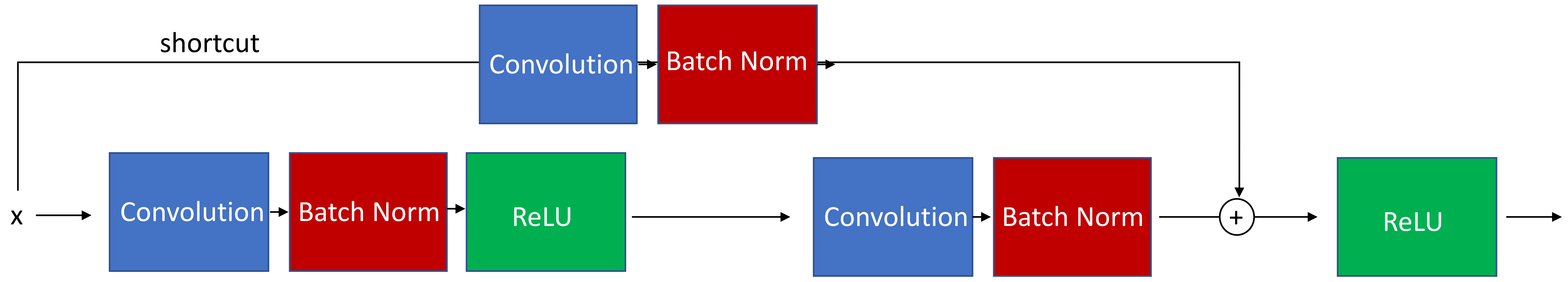


$$a^{(l+2)} = \sigma\big(\underbrace{z^{(l+2)} + a^{(l)}}\big)$$

We assume these have the same dimension
(e.g., via "same" convolution)

ResNets





alternative residual blocks with skip connections such that the input passed via the shortcut is resized to dimensions of the main path's output

ResNet Block Implementation

PyTorch implementations of the previous slides:

https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L13_intro-cnn/code/resnet-blocks.ipynb

ResNet-34 and ResNet-152

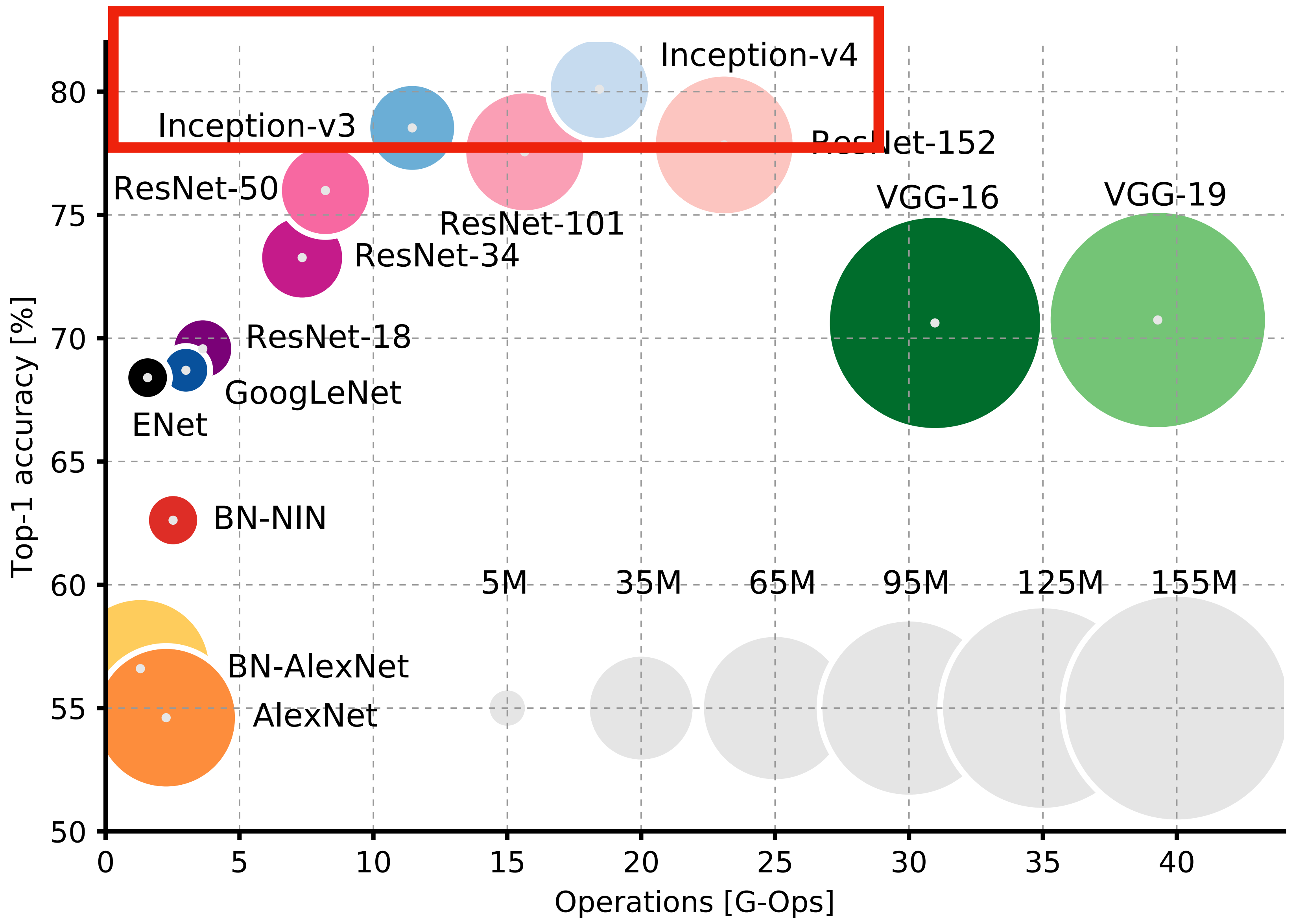
PyTorch implementations:

https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L13_intro-cnn/code/resnet-34.ipynb

https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L13_intro-cnn/code/resnet-152.ipynb

(Note that I had to implement them rather quickly yesterday; thus, I didn't tune hyperparameters, and the performance can be improved a lot, e.g., by using some of the image augmentation steps from your home work, among others)

Common Architectures Revisited



Canziani, A., Paszke, A., & Culurciello, E. (2016). An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*.

To be continued ...