Lecture 13

Introduction to Convolutional Neural Networks Part 2

STAT 453: Deep Learning, Spring 2020

Sebastian Raschka

http://stat.wisc.edu/~sraschka/teaching/stat453-ss2020/

https://github.com/rasbt/stat453-deep-learning-ss20/tree/master/L13-cnns-part2

Lecture Overview

1. **Padding (control output size in addition to stride)**
2. Spatial Dropout and BatchNorm
3. Considerations for CNNs on GPUs
4. Common Architectures
   A. VGG16 (simple, deep CNN)
   B. ResNet and skip connections
   C. Fully convolutional networks (no fully connected layers)
   D. Inception (parallel convolutions and auxiliary losses)
5. Transfer learning

Padding

output size

input size

padding pixels per side

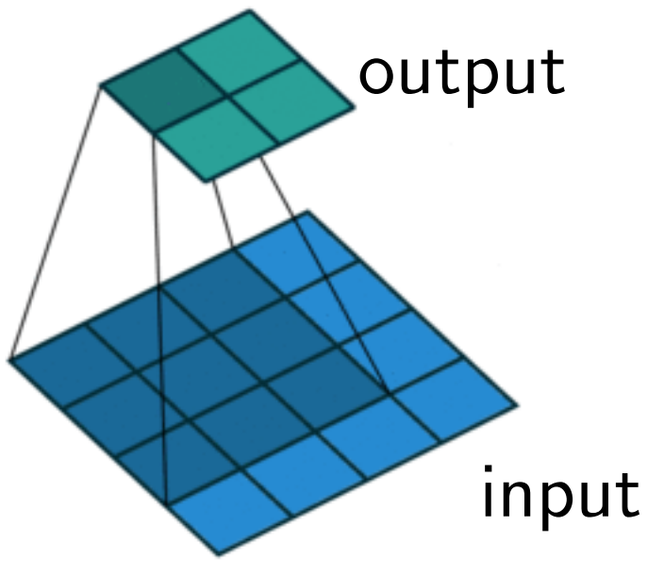
$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

"floor" function

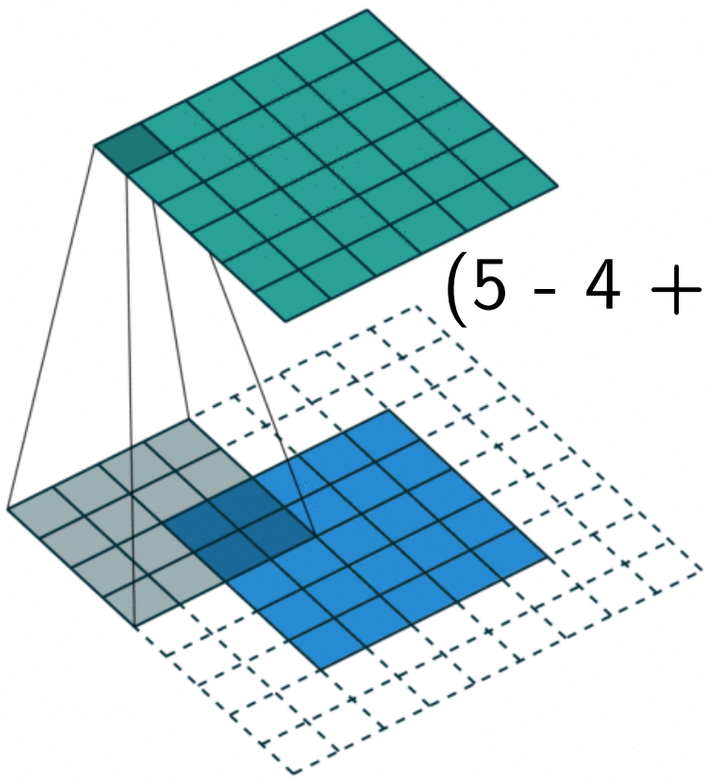
stride

kernel size

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

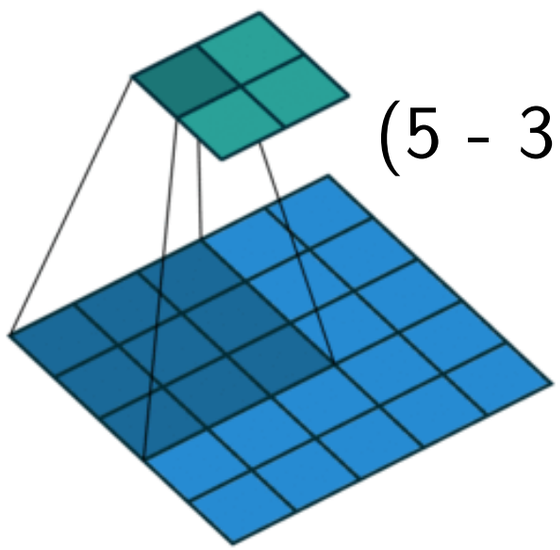


No padding, stride=1



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

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)

Lecture Overview

1. Padding (control output size in addition to stride)
2. **Spatial Dropout and BatchNorm**
3. Considerations for CNNs on GPUs
4. Common Architectures
    A. VGG16 (simple, deep CNN)
    B. ResNet and skip connections
    C. Fully convolutional networks (no fully connected layers)
    D. Inception (parallel convolutions and auxiliary losses)
5. Transfer learning

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)

```
import torch

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

```
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]],

         [[-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]]]])
```

BatchNorm 2D

BatchNorm1d

CLASS `torch.nn.BatchNorm1d(`*`num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True`*`)` [SOURCE]

Applies Batch Normalization over a 2D or 3D input (a mini-batch of 1D inputs with optional additional channel dimension) as described in the paper Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift .

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

BatchNorm2d

CLASS `torch.nn.BatchNorm2d(`*`num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True`*`)` [SOURCE]

Applies Batch Normalization over a 4D input (a mini-batch of 2D inputs with additional channel dimension) as described in the paper Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift .

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

Source: https://pytorch.org/docs/stable/nn.html

BatchNorm 2D

BatchNorm1d — Inputs are rank-2 tensors: [N, num_features)

CLASS `torch.nn.BatchNorm1d(num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)` [SOURCE]

Applies Batch Normalization over a 2D or 3D input (a mini-batch of 1D inputs with optional additional channel dimension) as described in the paper Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift .

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

BatchNorm2d — Inputs are rank-4 tensors: [N, C, H, W]

CLASS `torch.nn.BatchNorm2d(num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)` [SOURCE]

Applies Batch Normalization over a 4D input (a mini-batch of 2D inputs with additional channel dimension) as described in the paper Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift .

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

In BatchNorm2d, the mean and standard deviation are computed for N*H*W, i.e., over the channel dimension

Source: https://pytorch.org/docs/stable/nn.html

BatchNorm 2D

In BatchNorm2d, the mean and standard deviation are computed for N*H*W, i.e., over the channel dimension

[1]:
```
import torch.nn as nn
import torch.nn.functional as F


class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()

        self.cn1 = nn.Conv2d(3, 192, kernel_size=5,
                             stride=1, padding=2, bias=False)
        self.bn1 = nn.BatchNorm2d(192)

# ...
```

[2]:
```
model = Model()
```

[3]:
```
model.bn1.weight.size()
```

[3]:
```
torch.Size([192])
```

Lecture Overview

1. Padding (control output size in addition to stride)
2. Spatial Dropout and BatchNorm
3. **Considerations for CNNs on GPUs**
4. Common Architectures
    A. VGG16 (simple, deep CNN)
    B. ResNet and skip connections
    C. Fully convolutional networks (no fully connected layers)
    D. Inception (parallel convolutions and auxiliary losses)
5. Transfer learning

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)

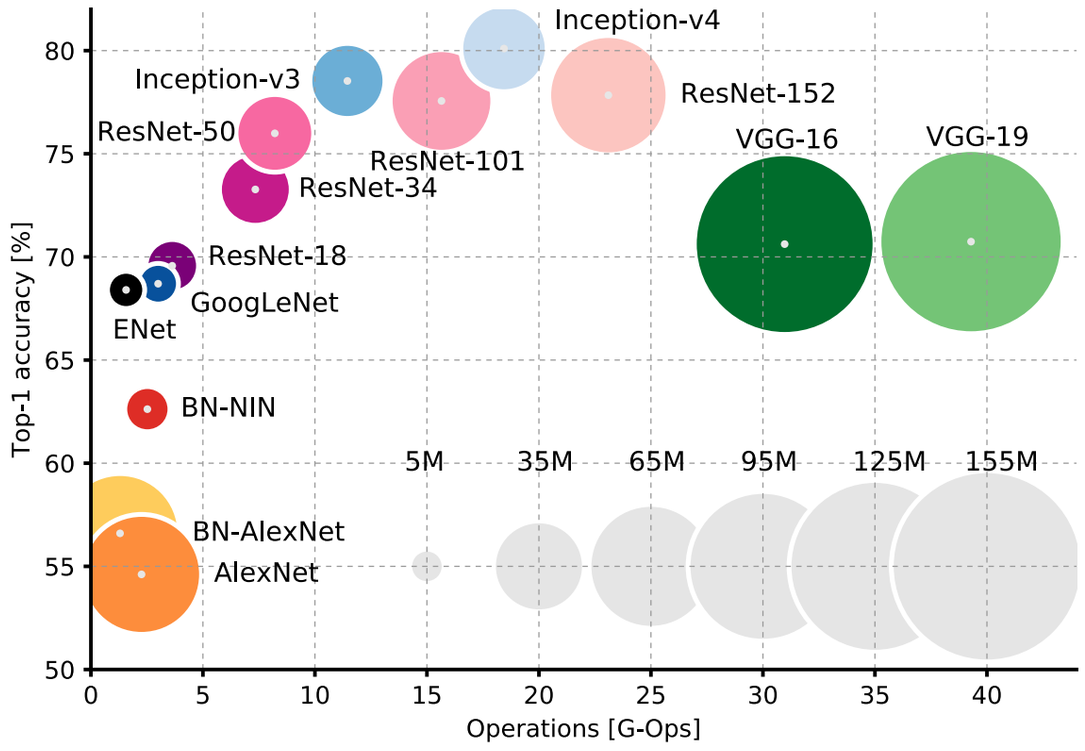
```
import torch
import torch.nn as nn
import torch.nn.functional as F

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

Lecture Overview

1. Padding (control output size in addition to stride)
2. Spatial Dropout and BatchNorm
3. Considerations for CNNs on GPUs
4. **Common Architectures**
    A. VGG16 (simple, deep CNN)
    B. ResNet and skip connections
    C. Fully convolutional networks (no fully connected layers)
    D. Inception (parallel convolutions and auxiliary losses)
5. Transfer learning

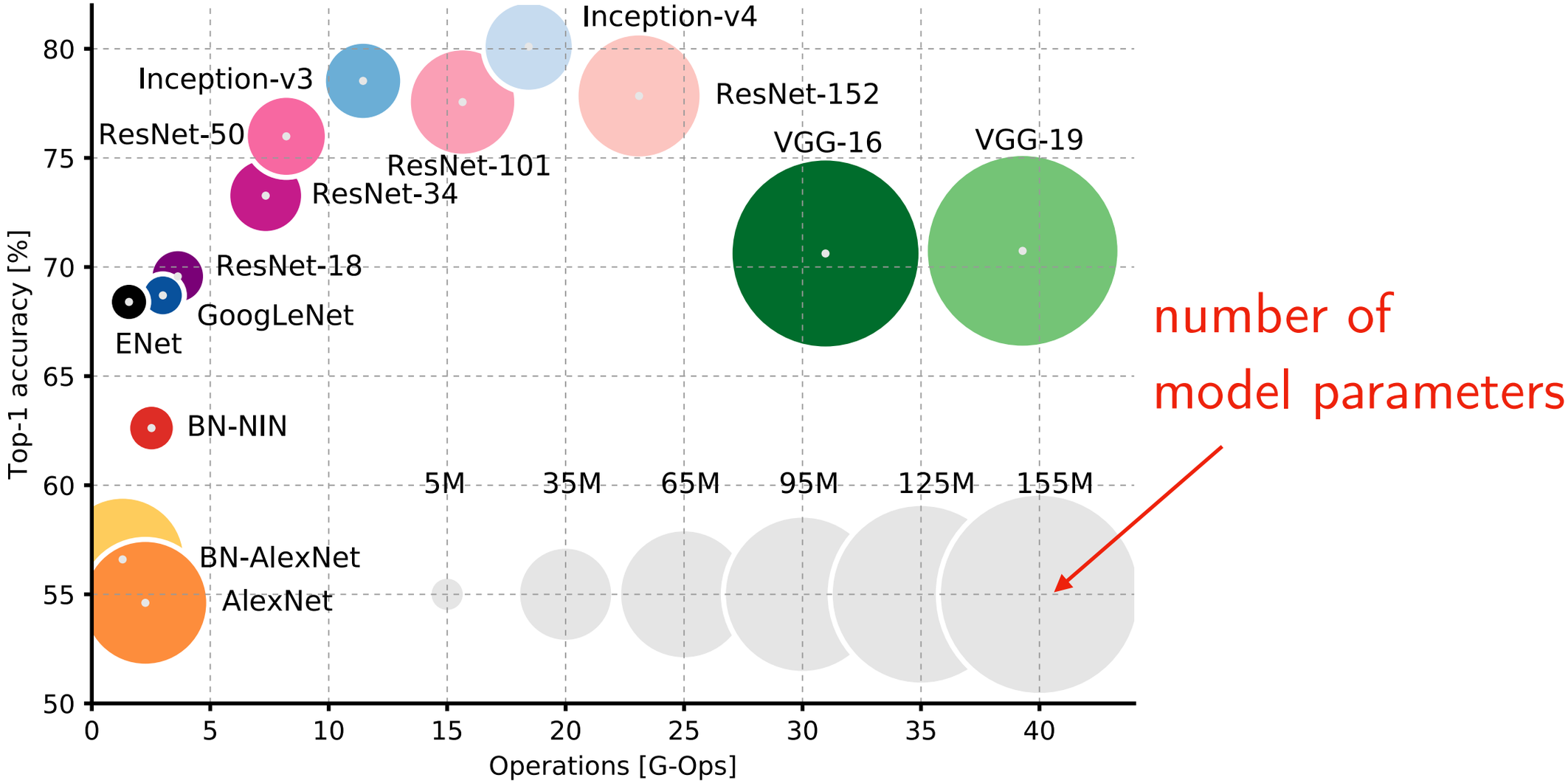
Common Architectures Revisited

We will discuss some additional common CNN architectures since the field evolved quite a bit since 2012 ...



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

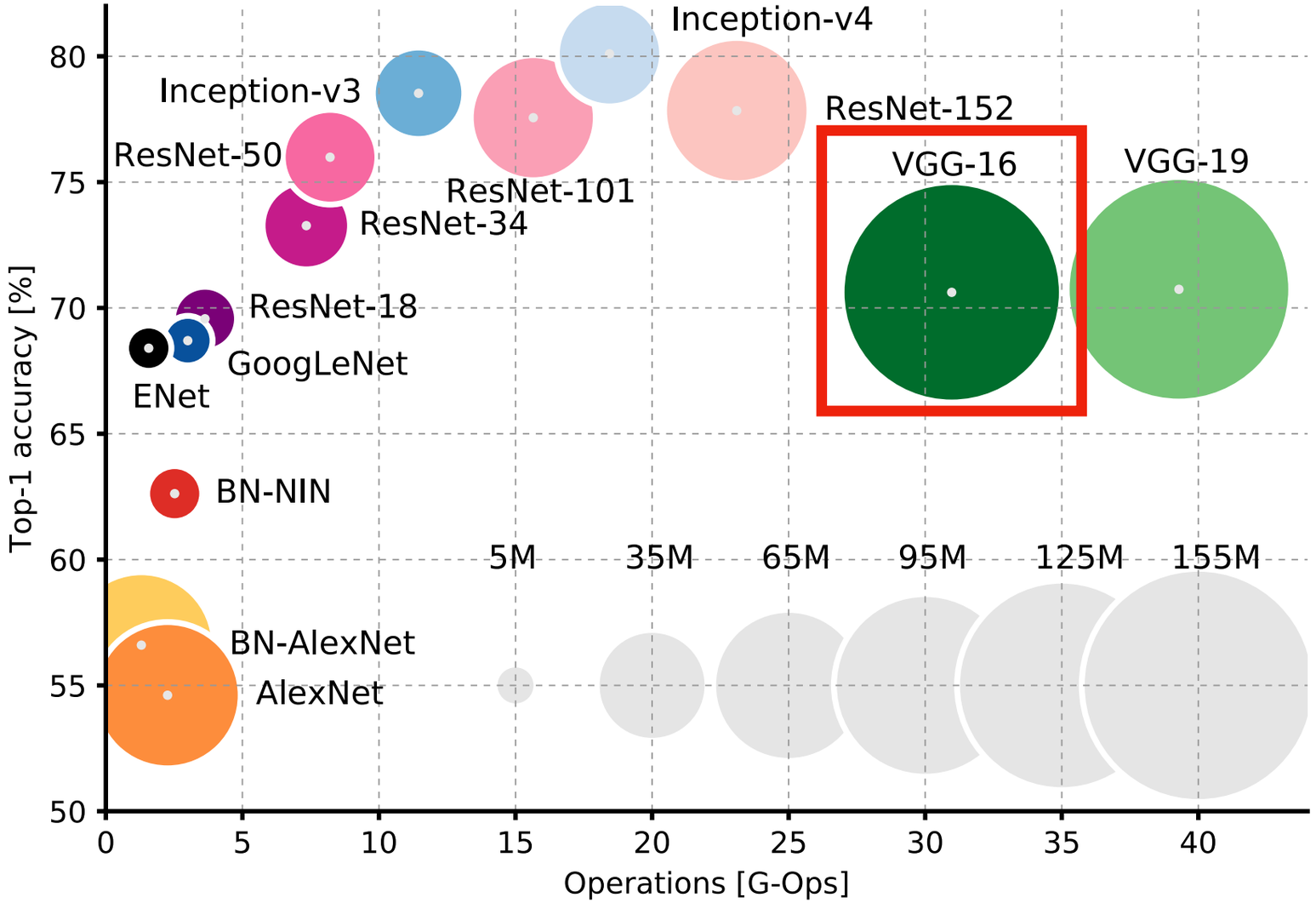


Canziani, A., Paszke, A., & Culurciello, E. (2016). An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*.

Lecture Overview

1. Padding (control output size in addition to stride)
2. Spatial Dropout and BatchNorm
3. Considerations for CNNs on GPUs
4. **Common Architectures**
   - **A. VGG16 (simple, deep CNN)**
   - B. ResNet and skip connections
   - C. Fully convolutional networks (no fully connected layers)
   - D. Inception (parallel convolutions and auxiliary losses)
5. Transfer learning

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/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/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**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	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	conv3-512 conv3-512 **conv1-512**	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	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Advantages:

very simple architecture,

3x3 convs, stride=1,

"same" padding, 2x2 max pooling

Disadvantage:

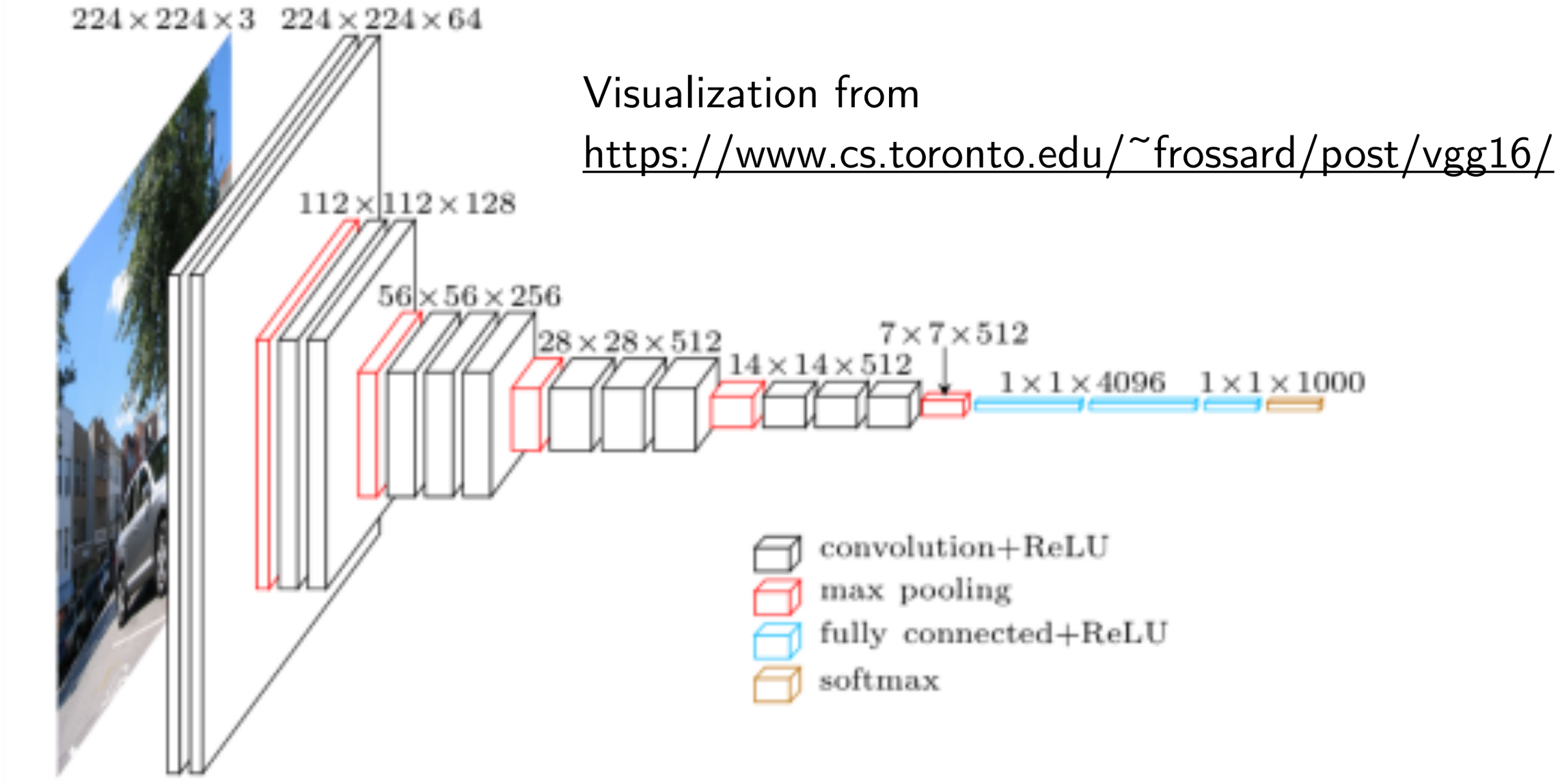
very large number of parameters and slow

(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/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/vgg16.ipynb



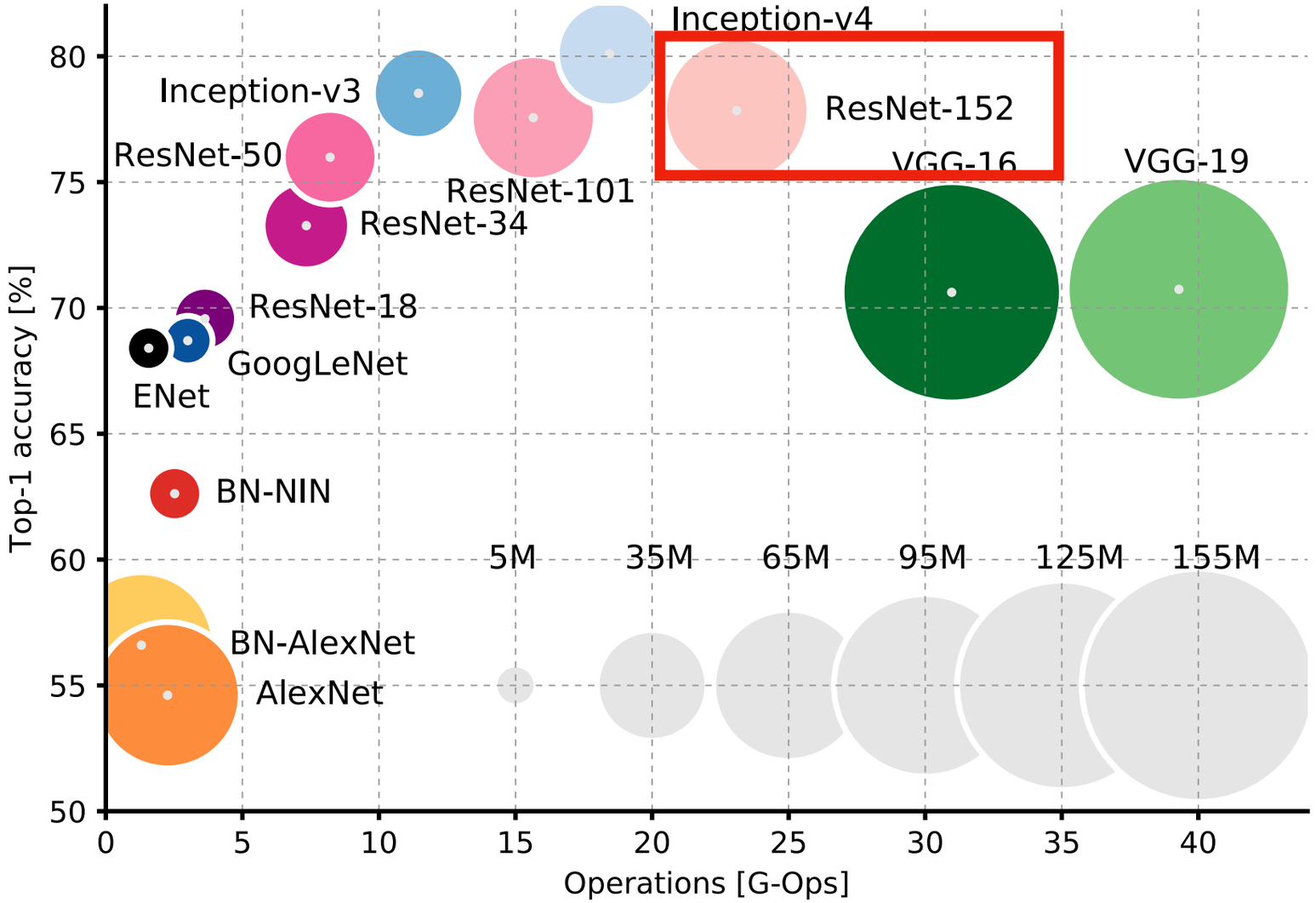
Visualization from
https://www.cs.toronto.edu/~frossard/post/vgg16/

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

Lecture Overview

1. Padding (control output size in addition to stride)
2. Spatial Dropout and BatchNorm
3. Considerations for CNNs on GPUs
4. **Common Architectures**
   - A. VGG16 (simple, deep CNN)
   - **B. ResNet and skip connections**
   - C. Fully convolutional networks (no fully connected layers)
   - D. Inception (parallel convolutions and auxiliary losses)
5. Transfer learning

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

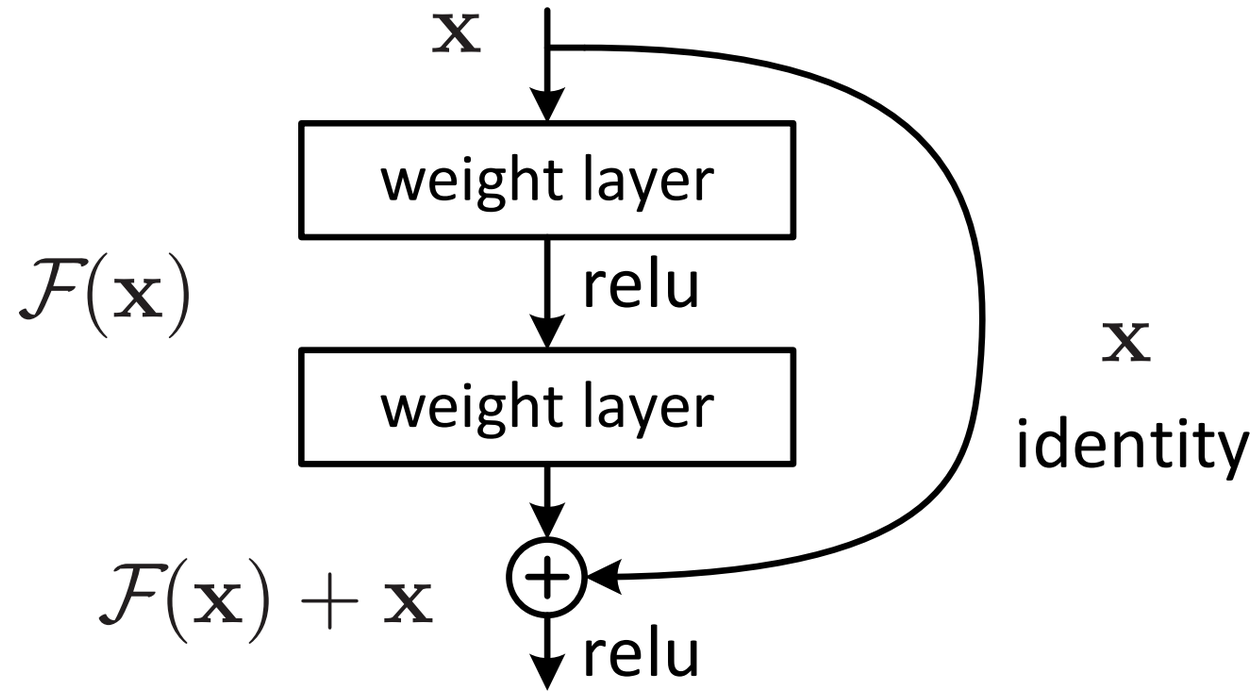
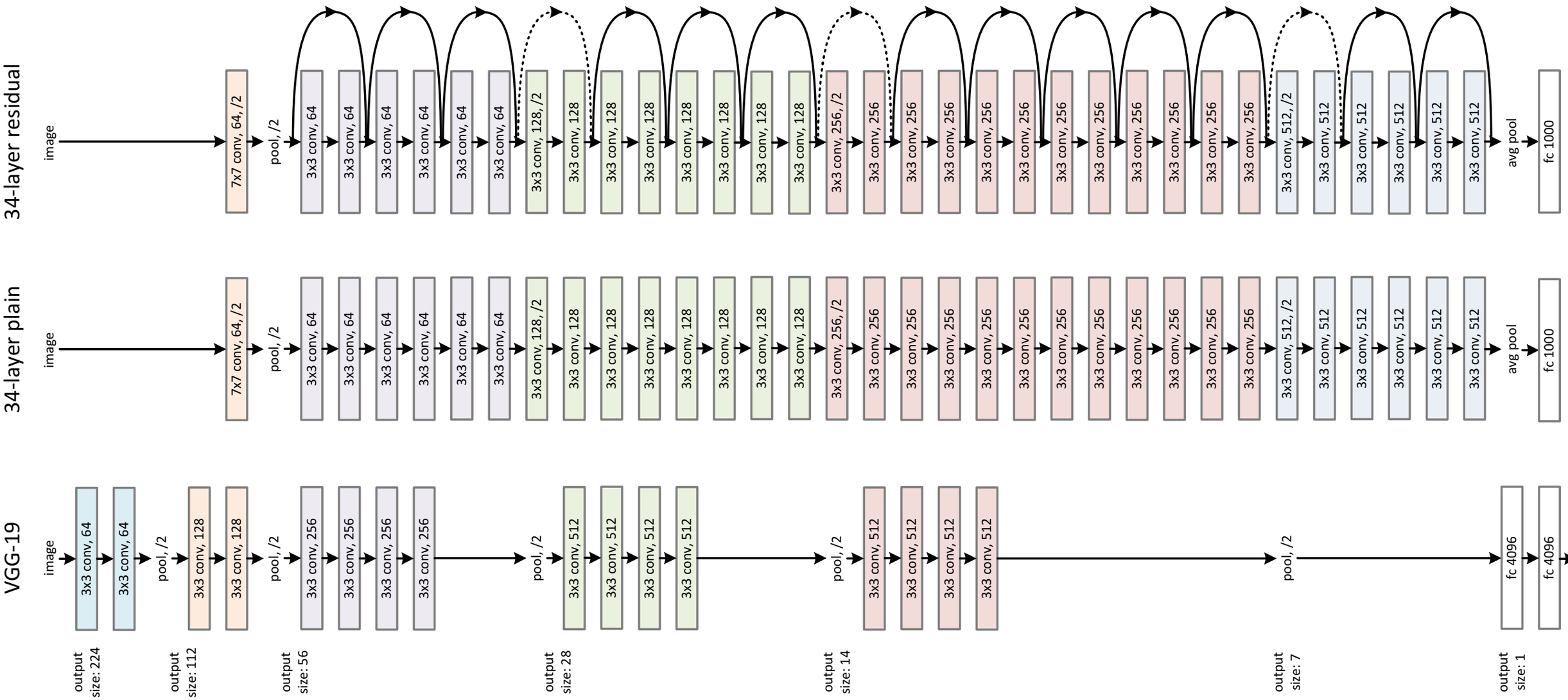


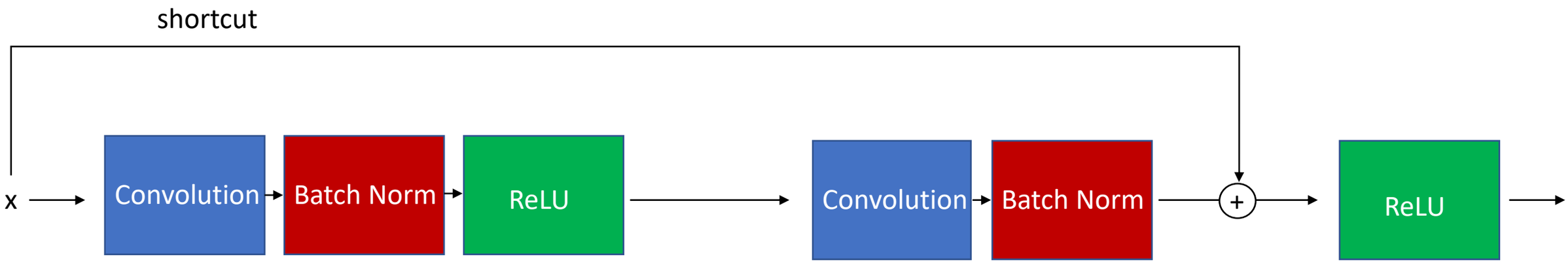
Figure 2. Residual learning: a building block.

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.

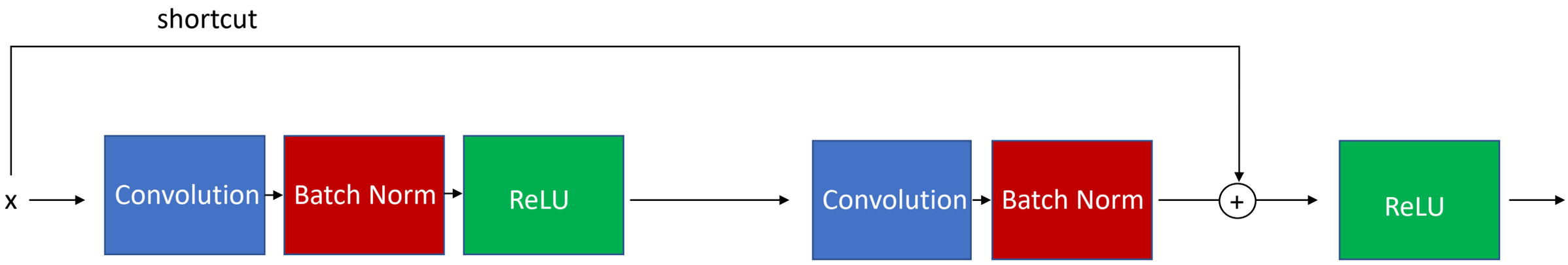


ResNets



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

ResNets



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

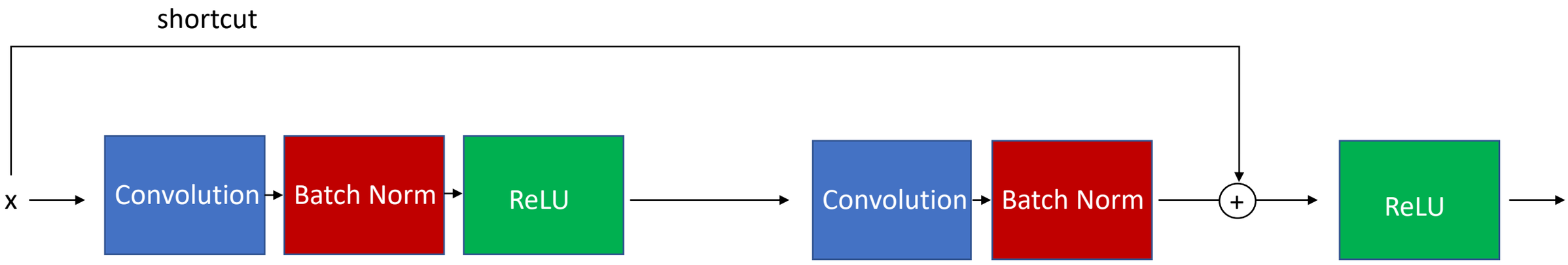
$$= \sigma\left(a^{(l+1)}W^{(l+2)} + b^{(l+2)} + a^{(l)}\right)$$

If all weights and the bias are zero, then

$$= \sigma\left(a^{(l)}\right) = a^{(l)} \qquad \text{(identity function)}$$

due to ReLU

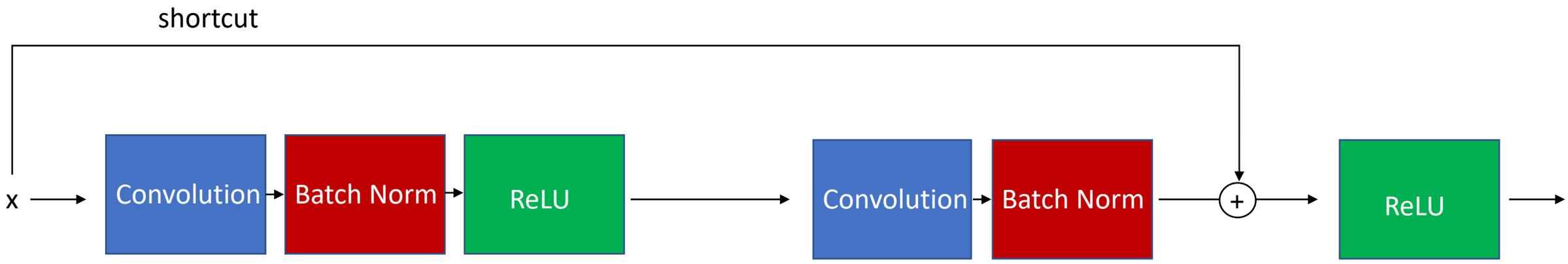
ResNets

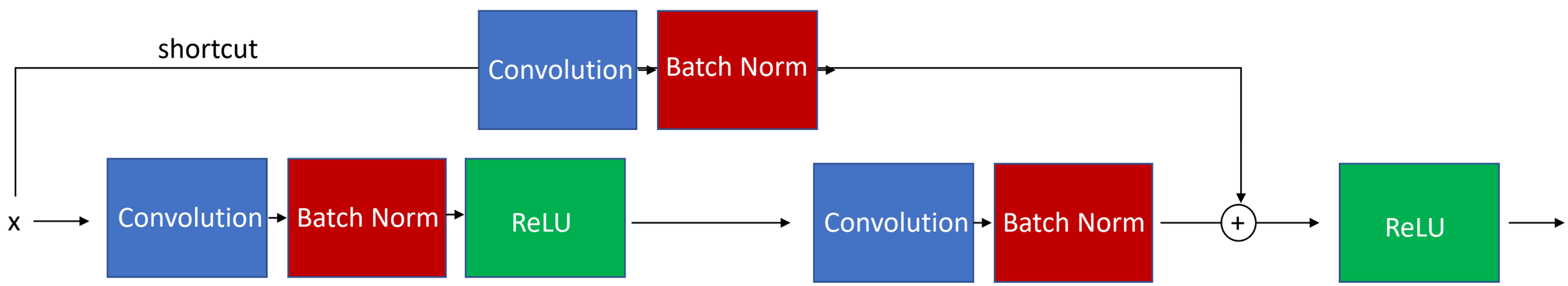


$$a^{(l+2)} = \sigma\big(z^{(l+2)} + a^{(l)}\big)$$

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/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/resnet-blocks.ipynb

ResNet-34 and ResNet-152

PyTorch implementations:

https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/resnet-34.ipynb

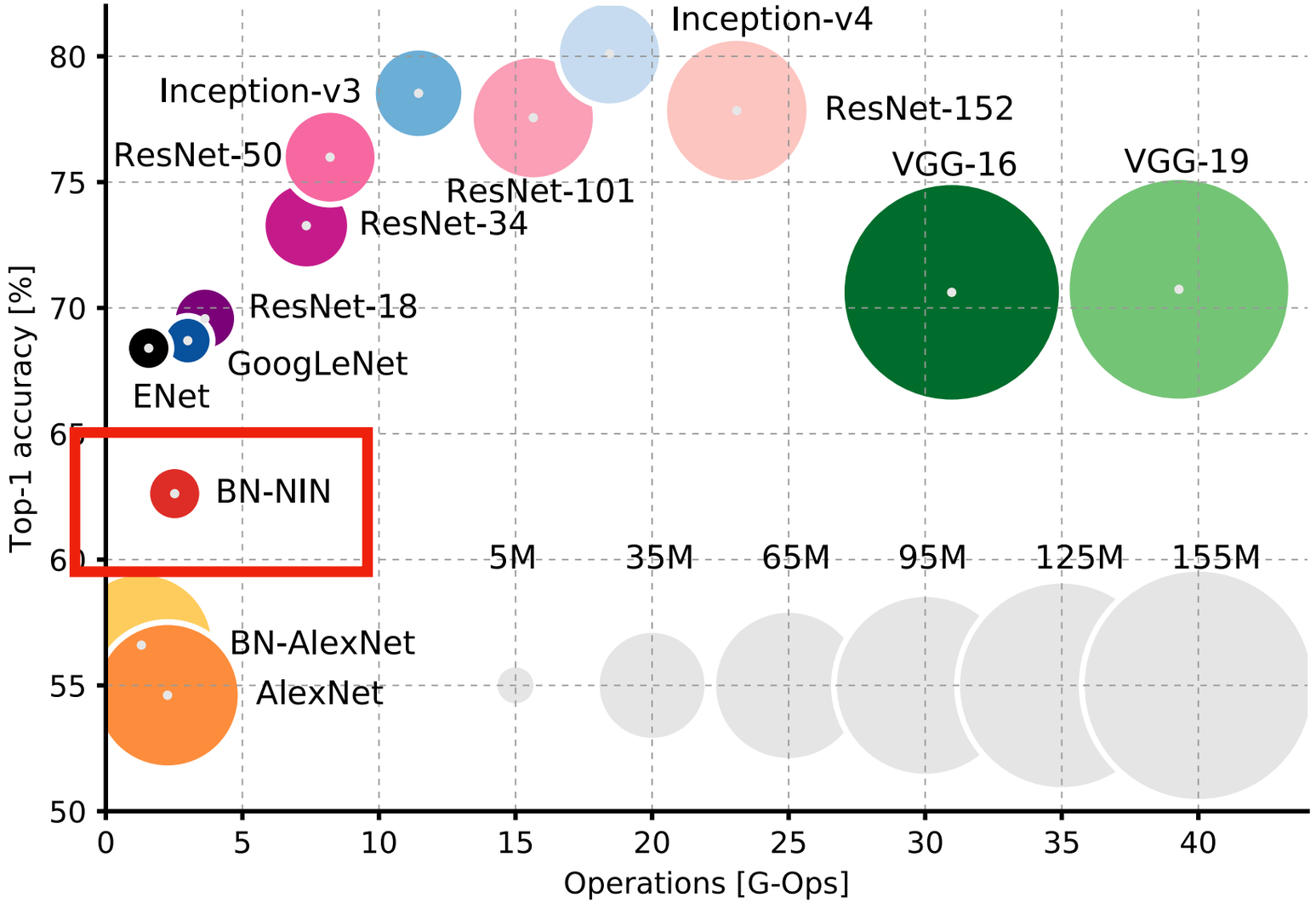
https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/resnet-152.ipynb

(Can be substantially improved with more hyperparameter tuning)

Lecture Overview

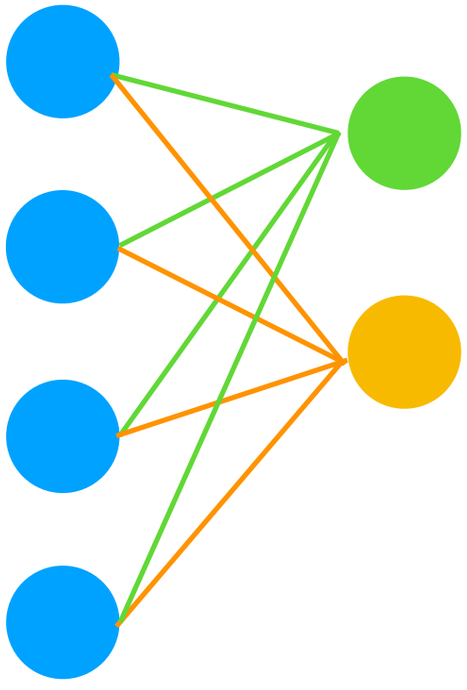
1. Padding (control output size in addition to stride)
2. Spatial Dropout and BatchNorm
3. Considerations for CNNs on GPUs
4. **Common Architectures**
   - A. VGG16 (simple, deep CNN)
   - B. ResNet and skip connections
   - **C. Fully convolutional networks (no fully connected layers)**
   - D. Inception (parallel convolutions and auxiliary losses)
5. Transfer learning

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*.

Side Note: It is Possible to Replace Fully Connected Layers by Convolutional Layers

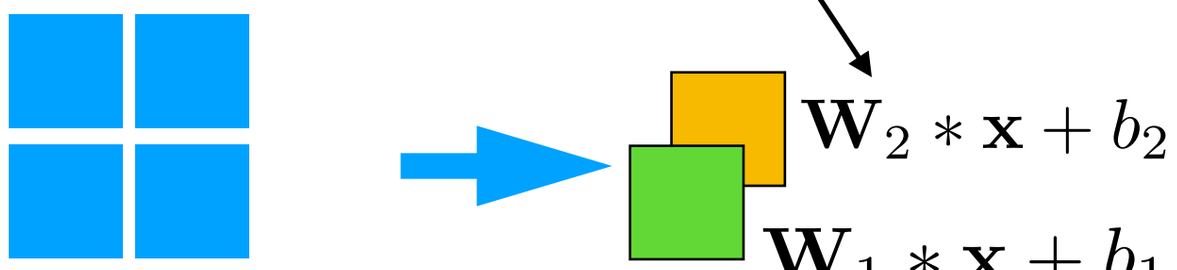


$$\mathbf{w}_1^T \mathbf{x} + b_1$$

$$\mathbf{w}_2^T \mathbf{x} + b_2$$

Fully connected layer

remember, these also involve dot products between the receptive fields and kernels



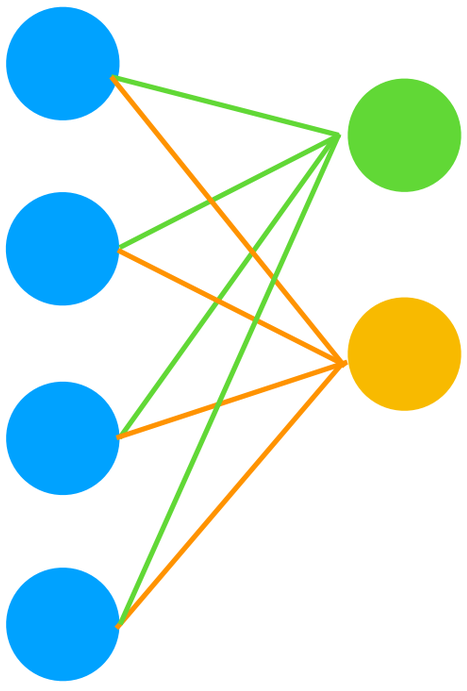
$$\mathbf{W}_2 * \mathbf{x} + b_2$$

$$\mathbf{W}_1 * \mathbf{x} + b_1$$

where $\mathbf{W}_1 = \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{1,3} & w_{1,4} \end{bmatrix}$

$$\mathbf{W}_2 = \begin{bmatrix} w_{2,1} & w_{2,2} \\ w_{2,3} & w_{2,4} \end{bmatrix}$$

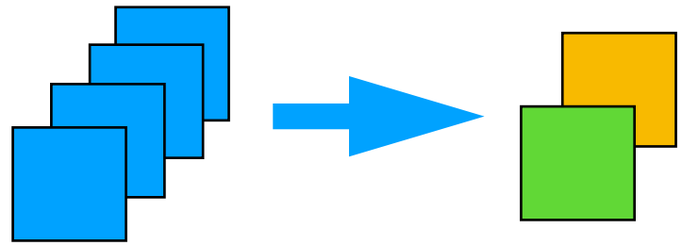
Side Note: It is Possible to Replace Fully Connected Layers by Convolutional Layers



$$\mathbf{w}_1^T \mathbf{x} + b_1$$

$$\mathbf{w}_2^T \mathbf{x} + b_2$$

Fully connected layer



Or, we can concatenate the inputs into 1x1 images with 4 channels and then use 2 kernels
(remember, each kernel then also has 4 channels)

Example: https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/fc-to-conv.ipynb

Network in Network (NiN)

Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *arXiv preprint arXiv:1312.4400* (2013).

Key Ideas

1)

- A convolution kernel can be thought of as a generalized linear model (GLM)
- Using a "sophisticated" nonlinear function approximator (e.g., an MLP) may enhance the abstraction ability of the local model
- => Replace GLM by "micro network" (sliding an MLP over the feature map)

Network in Network (NiN)

Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *arXiv preprint arXiv:1312.4400* (2013).

Key Ideas

1)

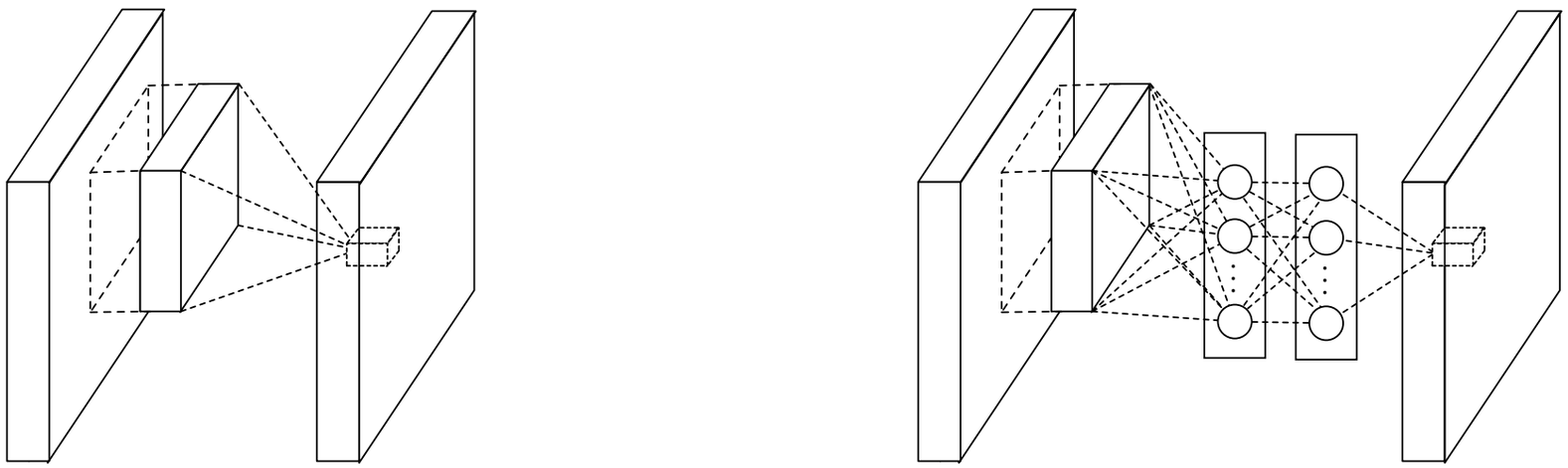
- A convolution kernel can be thought of as a generalized linear model (GLM)
- Using a "sophisticated" nonlinear function approximator (e.g., an MLP) may enhance the abstraction ability of the local model
- => Replace GLM by "micro network" (sliding an MLP over the feature map)

2)

- Replace the MLP "micro structure" via convolutions (explanation on the previous slides)
- Replace the fully connected layers in the last layers by *global average pooling*

Network in Network (NiN)

Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *arXiv preprint arXiv:1312.4400* (2013).



(a) Linear convolution layer (b) Mlpconv layer

Figure 1: Comparison of linear convolution layer and mlpconv layer. The linear convolution layer includes a linear filter while the mlpconv layer includes a micro network (we choose the multilayer perceptron in this paper). Both layers map the local receptive field to a confidence value of the latent concept.

- Using a "sophisticated" nonlinear function approximator (e.g., an MLP) may enhance the abstraction ability of the local model

Global Average Pooling in Last Layer

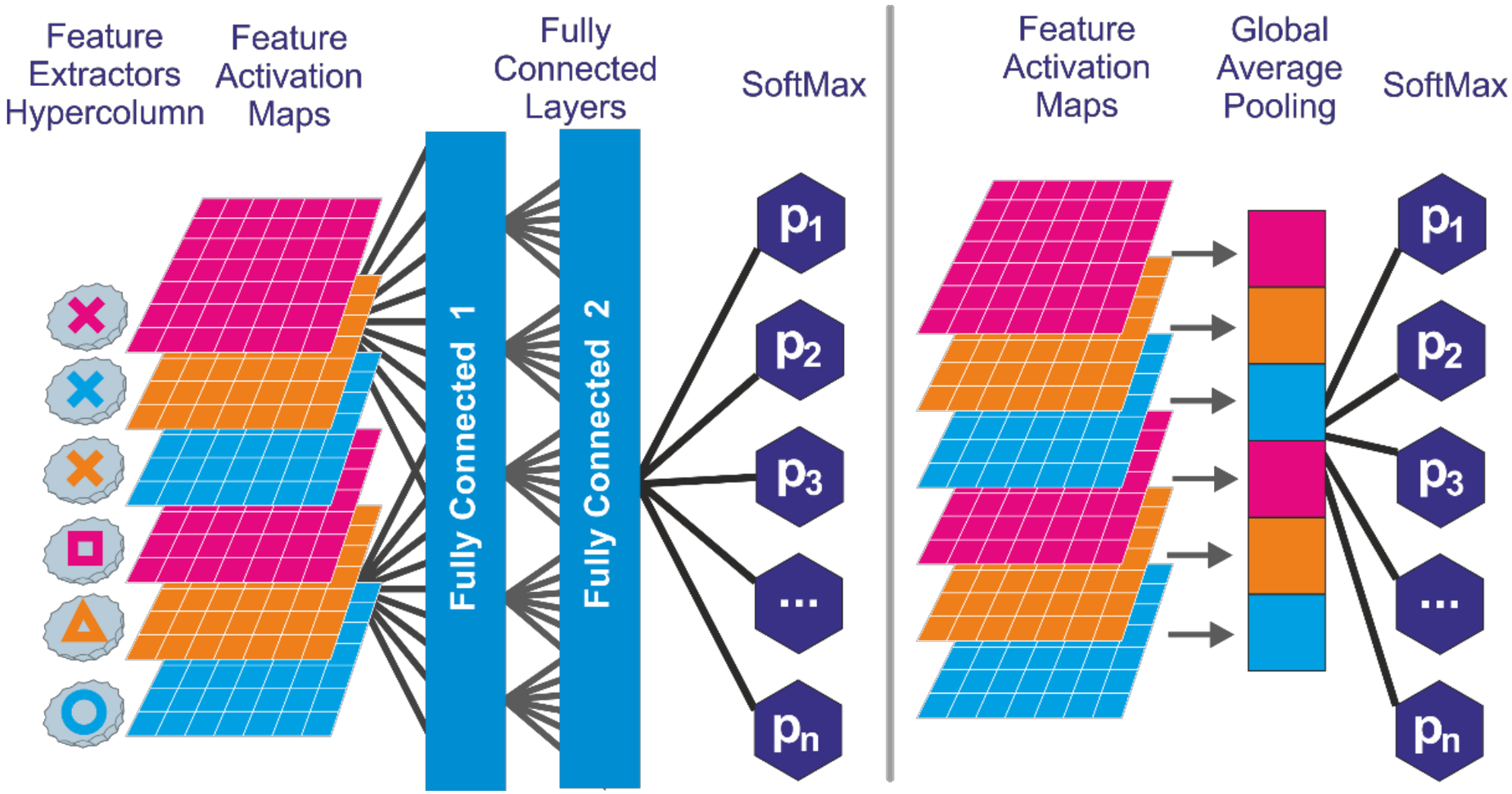


Figure 16: Global average pooling layer replacing the fully connected layers. The output layer implements a Softmax operation with $p_1, p_2, \ldots, p_n$ the predicted probabilities for each class.

Figure Source: Singh, Anshuman Vikram. "Content-based image retrieval using deep learning." (2015).

Network in Network (NiN)

Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *arXiv preprint arXiv:1312.4400* (2013).

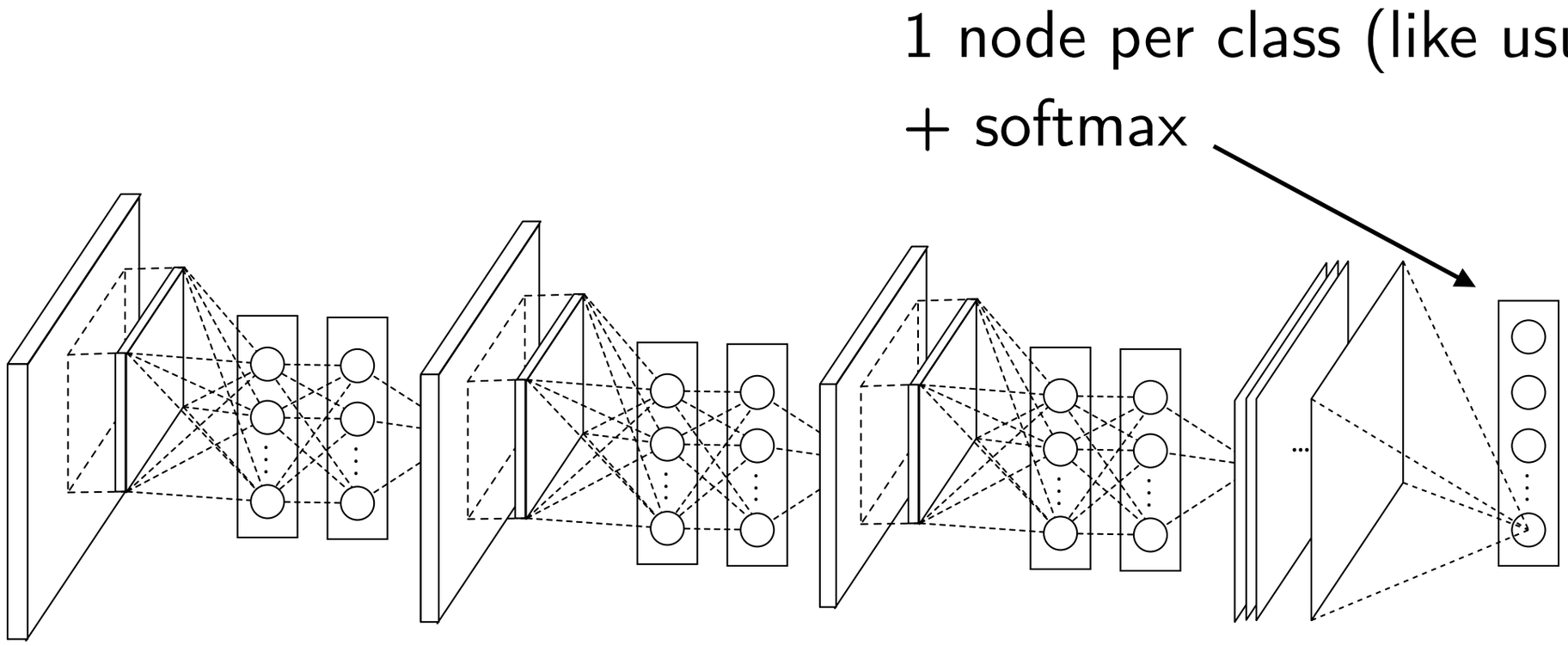


Figure 2: The overall structure of Network In Network. In this paper the NINs include the stacking of three mlpconv layers and one global average pooling layer.

- Replace the fully connected layers in the last layers by *global average pooling*

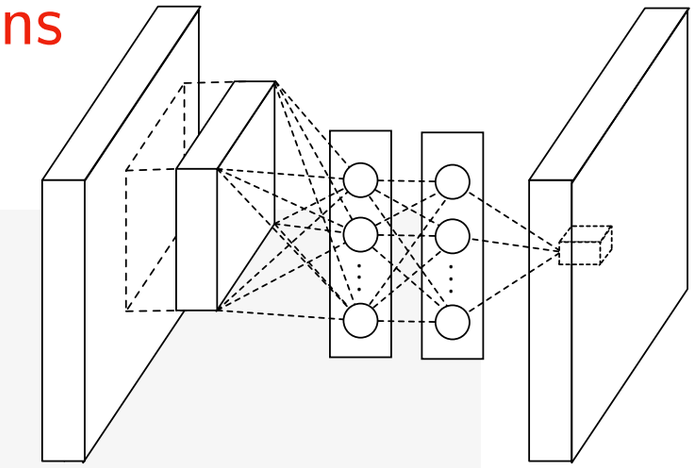
Network in Network (NiN)

Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *arXiv preprint arXiv:1312.4400* (2013).

Example Implementation:

https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/nin-cifar10.ipynb

- Replace the MLP "micro structure" with convolutions (explanation on the previous slides)



=

```
class NiN(nn.Module):
    def __init__(self, num_classes):
        super(NiN, self).__init__()
        self.num_classes = num_classes
        self.classifier = nn.Sequential(
                nn.Conv2d(3, 192, kernel_size=5, stride=1, padding=2),
                nn.ReLU(inplace=True),
                nn.Conv2d(192, 160, kernel_size=1, stride=1, padding=0),
                nn.ReLU(inplace=True),
                nn.Conv2d(160,  96, kernel_size=1, stride=1, padding=0),
                nn.ReLU(inplace=True),
                nn.MaxPool2d(kernel_size=3, stride=2, padding=1),
                nn.Dropout(0.5),

                nn.Conv2d(96, 192, kernel_size=5, stride=1, padding=2),
```

Network in Network (NiN)

Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *arXiv preprint arXiv:1312.4400* (2013).

Example Implementation:

https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/nin-cifar10.ipynb

- Replace fully connected layers with global average pooling (explanation on the previous slides)

```
            nn.Conv2d(192, 192, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(192, 192, kernel_size=1, stride=1, padding=0),
            nn.ReLU(inplace=True),
            nn.Conv2d(192,  10, kernel_size=1, stride=1, padding=0),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(kernel_size=8, stride=1, padding=0),

            )

    def forward(self, x):
        x = self.classifier(x)
        logits = x.view(x.size(0), self.num_classes)
        probas = torch.softmax(x, dim=1)
        return logits, probas
```

Network in Network (NiN)

Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *arXiv preprint arXiv:1312.4400* (2013).

Why it might work well in practice

Using the micro-networks allow us to extract more sophisticated features (non-linear functions); we may need fewer extractors and can avoid learning too simple or redundant abstractions

Fully-connected layers have a lot of parameters and may cause overfitting, replacing those by global average pooling might help with better generalization

(nice side-effect: we can make the network be somewhat agnostic to the input size)

Different but related idea: "All-Convolutional Network"

Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. "Striving for simplicity: The all convolutional net." *arXiv preprint arXiv:1412.6806* (2014).

Key Idea: Replace Maxpooling by strided convolutions

(i.e., conv layers with stride=2)

$$s_{i,j,u}(f) = \left(\sum_{h=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \sum_{w=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} |f_{g(h,w,i,j,u)}|^p \right)^{1/p},$$

definition of max-pooling with stride=2 to when $p \to \infty$

$$c_{i,j,o}(f) = \sigma \left(\sum_{h=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \sum_{w=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \sum_{u=1}^{N} \theta_{h,w,u,o} \cdot f_{g(h,w,i,j,u)} \right)$$

definition of a convolutional layer with stride=2

Different but related idea: "All-Convolutional Network"

Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. "Striving for simplicity: The all convolutional net." *arXiv preprint arXiv:1412.6806* (2014).

Key Idea: Replace Maxpooling by strided convolutions

(i.e., conv layers with stride=2)

$$s_{i,j,u}(f) = \left(\sum_{h=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \sum_{w=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} |f_{g(h,w,i,j,u)}|^p \right)^{1/p},$$

definition of max-pooling with stride=2 to when $p \to \infty$

$$c_{i,j,o}(f) = \sigma \left(\sum_{h=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \sum_{w=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \sum_{u=1}^{N} \theta_{h,w,u,o} \cdot f_{g(h,w,i,j,u)} \right)$$

definition of a convolutional layer with stride=2

We can think of "strided convolutions" as learnable pooling

"All-Convolutional Network"

Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. "Striving for simplicity: The all convolutional net." *arXiv preprint arXiv:1412.6806* (2014).

Experimental Ablation Study with 3 Base Models:

Table 1: The three base networks used for classification on CIFAR-10 and CIFAR-100.

Model		
A	B	C
Input 32×32 RGB image		
5×5 conv. 96 ReLU	5×5 conv. 96 ReLU 1×1 conv. 96 ReLU	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU
3×3 max-pooling stride 2		
5×5 conv. 192 ReLU	5×5 conv. 192 ReLU 1×1 conv. 192 ReLU	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU
3×3 max-pooling stride 2		
3×3 conv. 192 ReLU		
1×1 conv. 192 ReLU		
1×1 conv. 10 ReLU		
global averaging over 6×6 spatial dimensions		
10 or 100-way softmax		

"All-Convolutional Network"

Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. "Striving for simplicity: The all convolutional net." *arXiv preprint arXiv:1412.6806* (2014).

Experimental Ablation Study with 3 Base Models: Shown are the modifications for Model C

Model		
Strided-CNN-C	ConvPool-CNN-C	All-CNN-C
Input 32×32 RGB image		
3×3 conv. 96 ReLU 3×3 conv. 96 ReLU with stride $r = 2$	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU 3×3 conv. 96 ReLU	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU
	3×3 max-pooling stride 2	3×3 conv. 96 ReLU with stride $r = 2$
3×3 conv. 192 ReLU 3×3 conv. 192 ReLU with stride $r = 2$	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU 3×3 conv. 192 ReLU	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU
	3×3 max-pooling stride 2	3×3 conv. 192 ReLU with stride $r = 2$

$\vdots$

"All-Convolutional Network"

Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. "Striving for simplicity: The all convolutional net." *arXiv preprint arXiv:1412.6806* (2014).

Experimental Ablation Study with 3 Base Models: Shown are the modifications for Model C

Remove pooling & increase stride of the previous layer

Model		
Strided-CNN-C	ConvPool-CNN-C	All-CNN-C
Input 32×32 RGB image		
3×3 conv. 96 ReLU 3×3 conv. 96 ReLU with stride $r = 2$	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU 3×3 conv. 96 ReLU	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU
	3×3 max-pooling stride 2	3×3 conv. 96 ReLU with stride $r = 2$
3×3 conv. 192 ReLU 3×3 conv. 192 ReLU with stride $r = 2$	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU 3×3 conv. 192 ReLU	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU
	3×3 max-pooling stride 2	3×3 conv. 192 ReLU with stride $r = 2$

"All-Convolutional Network"

Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. "Striving for simplicity: The all convolutional net." *arXiv preprint arXiv:1412.6806* (2014).

Experimental Ablation Study with 3 Base Models: Shown are the modifications for Model C

Remove pooling & add a strided conv. layer

Model		
Strided-CNN-C	ConvPool-CNN-C	All-CNN-C
Input 32×32 RGB image		
3×3 conv. 96 ReLU 3×3 conv. 96 ReLU with stride $r = 2$	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU 3×3 conv. 96 ReLU	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU
	3×3 max-pooling stride 2	3×3 conv. 96 ReLU with stride $r = 2$
3×3 conv. 192 ReLU 3×3 conv. 192 ReLU with stride $r = 2$	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU 3×3 conv. 192 ReLU	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU
	3×3 max-pooling stride 2	3×3 conv. 192 ReLU with stride $r = 2$

"All-Convolutional Network"

Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. "Striving for simplicity: The all convolutional net." *arXiv preprint arXiv:1412.6806* (2014).

Table 3: Comparison between the base and derived models on the CIFAR-10 dataset.

CIFAR-10 classification error

Model	Error (%)	# parameters
without data augmentation		
Model A	12.47%	≈ 0.9 M
Strided-CNN-A	13.46%	≈ 0.9 M
ConvPool-CNN-A	**10.21%**	≈ 1.28 M
ALL-CNN-A	10.30%	≈ 1.28 M
Model B	10.20%	≈ 1 M
Strided-CNN-B	10.98%	≈ 1 M
ConvPool-CNN-B	9.33%	≈ 1.35 M
ALL-CNN-B	**9.10%**	≈ 1.35 M
Model C	9.74%	≈ 1.3 M
Strided-CNN-C	10.19%	≈ 1.3 M
ConvPool-CNN-C	9.31%	≈ 1.4 M
ALL-CNN-C	**9.08%**	≈ 1.4 M

"All-Convolutional Network"

Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. "Striving for simplicity: The all convolutional net." *arXiv preprint arXiv:1412.6806* (2014).

CIFAR-10 classification error

Model	Error (%)	# parameters
without data augmentation		
Model A	12.47%	≈ 0.9 M
Strided-CNN-A	13.46%	≈ 0.9 M
ConvPool-CNN-A	**10.21%**	≈ 1.28 M
ALL-CNN-A	10.30%	≈ 1.28 M
Model B	10.20%	≈ 1 M
Strided-CNN-B	10.98%	≈ 1 M
ConvPool-CNN-B	9.33%	≈ 1.35 M
ALL-CNN-B	**9.10%**	≈ 1.35 M
Model C	9.74%	≈ 1.3 M
Strided-CNN-C	10.19%	≈ 1.3 M
ConvPool-CNN-C	9.31%	≈ 1.4 M
ALL-CNN-C	**9.08%**	≈ 1.4 M

Removing maxpooling and increasing the stride of the previous layer performs worse

"All-Convolutional Network"

Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. "Striving for simplicity: The all convolutional net." *arXiv preprint arXiv:1412.6806* (2014).

CIFAR-10 classification error

Model	Error (%)	# parameters
without data augmentation		
Model A	12.47%	≈ 0.9 M
Strided-CNN-A	13.46%	≈ 0.9 M
ConvPool-CNN-A	**10.21%**	≈ 1.28 M
ALL-CNN-A	10.30%	≈ 1.28 M
Model B	10.20%	≈ 1 M
Strided-CNN-B	10.98%	≈ 1 M
ConvPool-CNN-B	9.33%	≈ 1.35 M
ALL-CNN-B	**9.10%**	≈ 1.35 M
Model C	9.74%	≈ 1.3 M
Strided-CNN-C	10.19%	≈ 1.3 M
ConvPool-CNN-C	9.31%	≈ 1.4 M
ALL-CNN-C	**9.08%**	≈ 1.4 M

Replacing maxpooling with an convolutional layer (stride=2) improves the performance

"All-Convolutional Network"

Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. "Striving for simplicity: The all convolutional net." *arXiv preprint arXiv:1412.6806* (2014).

CIFAR-10 classification error

Model	Error (%)	# parameters
without data augmentation		
Model A	12.47%	≈ 0.9 M
Strided-CNN-A	13.46%	≈ 0.9 M
ConvPool-CNN-A	**10.21%**	≈ 1.28 M
ALL-CNN-A	10.30%	≈ 1.28 M
Model B	10.20%	≈ 1 M
Strided-CNN-B	10.98%	≈ 1 M
ConvPool-CNN-B	9.33%	≈ 1.35 M
ALL-CNN-B	**9.10%**	≈ 1.35 M
Model C	9.74%	≈ 1.3 M
Strided-CNN-C	10.19%	≈ 1.3 M
ConvPool-CNN-C	9.31%	≈ 1.4 M
ALL-CNN-C	**9.08%**	≈ 1.4 M

Replacing maxpooling with an convolutional layer (stride=2) improves the performance
(this may be unfair because of the additional parameters)

Difference to "All-Convolutional Network"

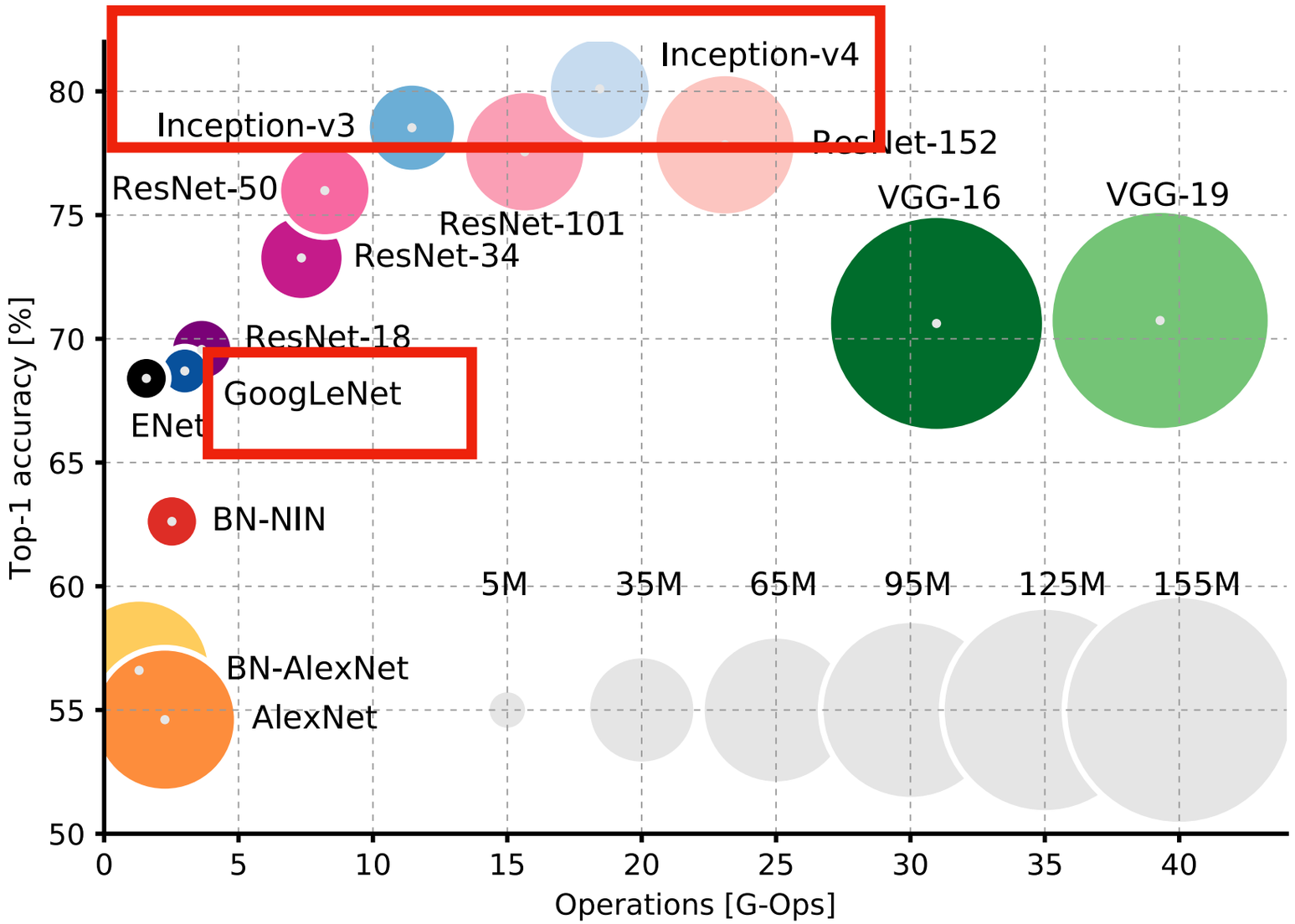
Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. "Striving for simplicity: The all convolutional net." *arXiv preprint arXiv:1412.6806* (2014).

Code example https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/convnet-allconv.ipynb

Lecture Overview

1. Padding (control output size in addition to stride)
2. Spatial Dropout and BatchNorm
3. Considerations for CNNs on GPUs
4. Common Architectures
   A. VGG16 (simple, deep CNN)
   B. ResNet and skip connections
   C. Fully convolutional networks (no fully connected layers)
   D. Inception (parallel convolutions and auxiliary losses)
5. Transfer learning

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*.

GoogLeNet / Inception v1

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9. 2015.

"In this paper, we will focus on an efficient deep neural network architecture for computer vision, codenamed Inception, which derives its name from the Network in network paper by Lin et al [12] in conjunction with the famous "we need to go deeper" internet meme"

GoogLeNet / Inception v1

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9. 2015.

Key Ideas/Features:

- 1x1 convolutions: An efficient way to reduce the number of channels (from NiN)
- Global Average Pooling at the last layer (from NiN)
- Use of auxiliary losses that are added to the total loss
- New: Inception module

GoogLeNet / Inception v1

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9. 2015.

Full Architecture

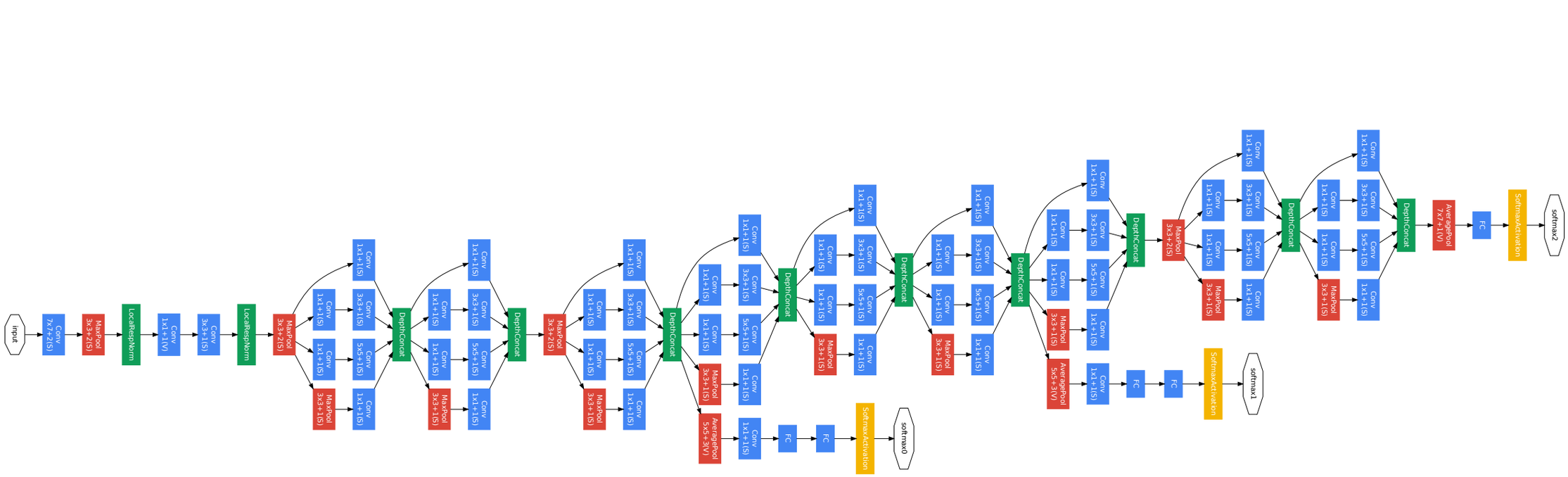


Figure 3: GoogLeNet network with all the bells and whistles.

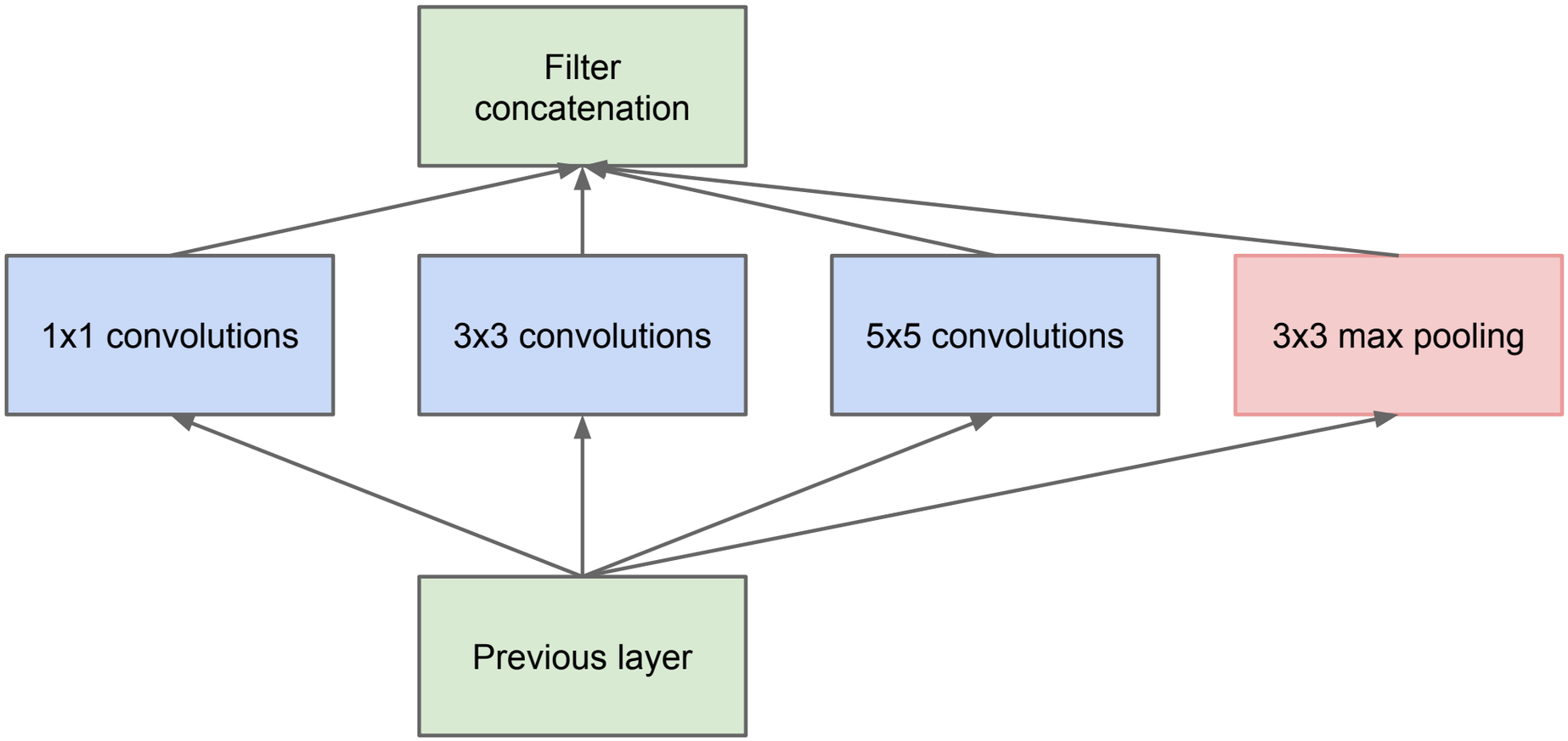
GoogLeNet / Inception v1

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9. 2015.

Zoomed in

GoogLeNet / Inception v1

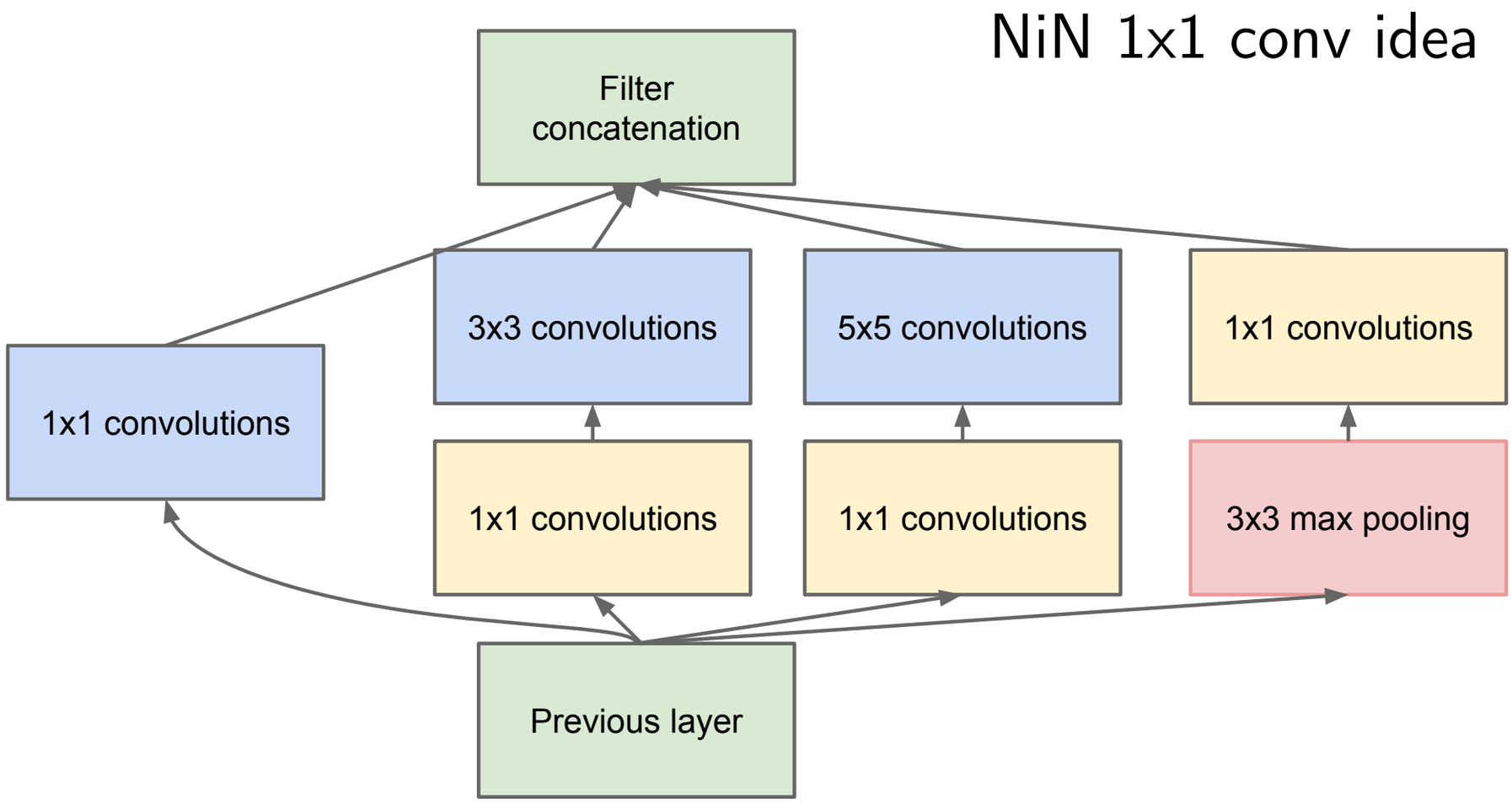
Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9. 2015.



(a) Inception module, naïve version

GoogLeNet / Inception v1

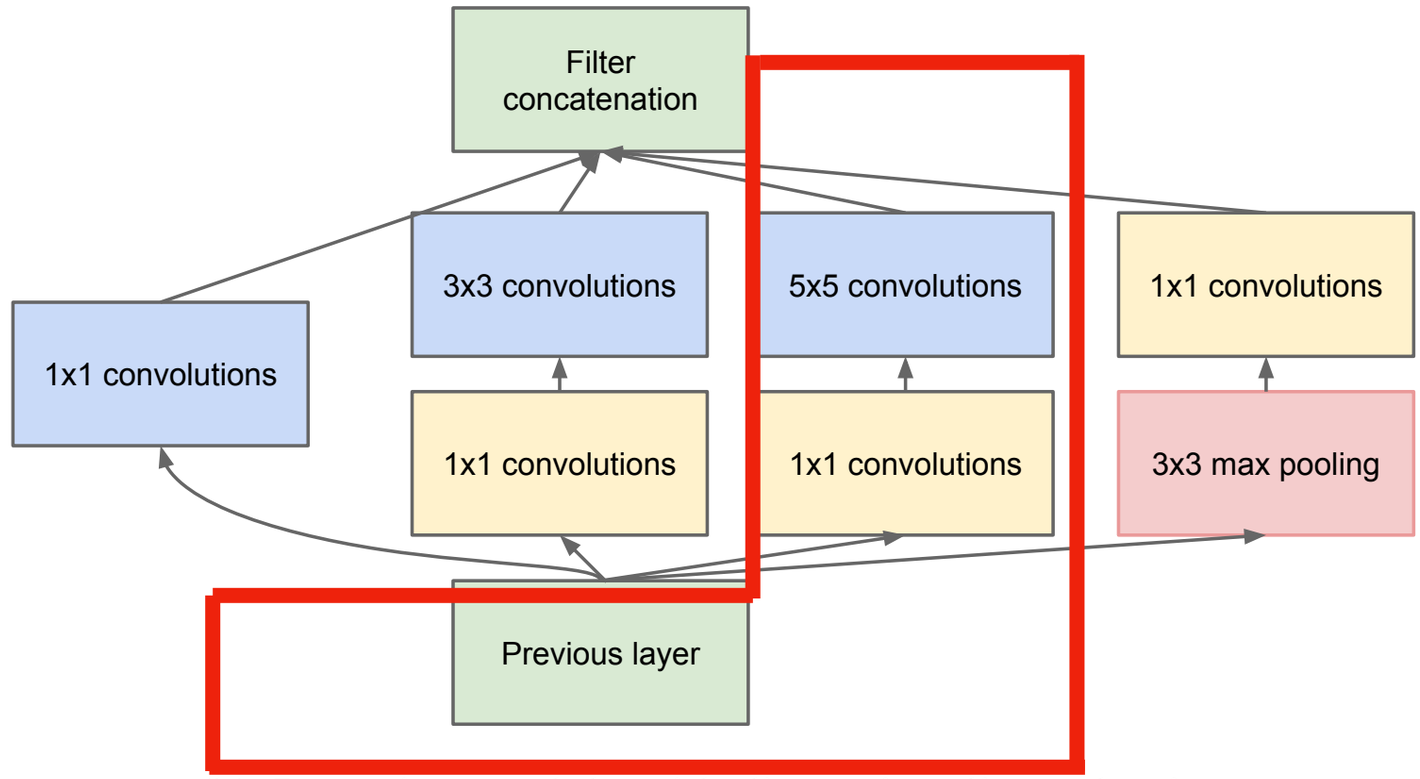
Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9. 2015.



(b) Inception module with dimensionality reduction

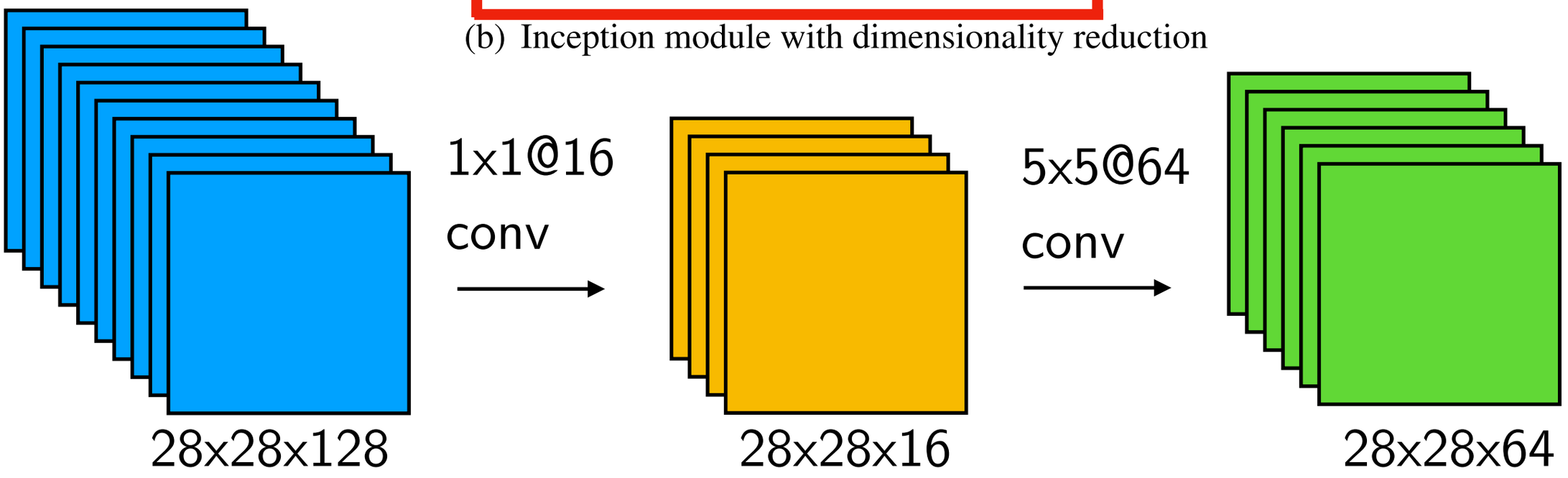
GoogLeNet / Inception v1

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9. 2015.



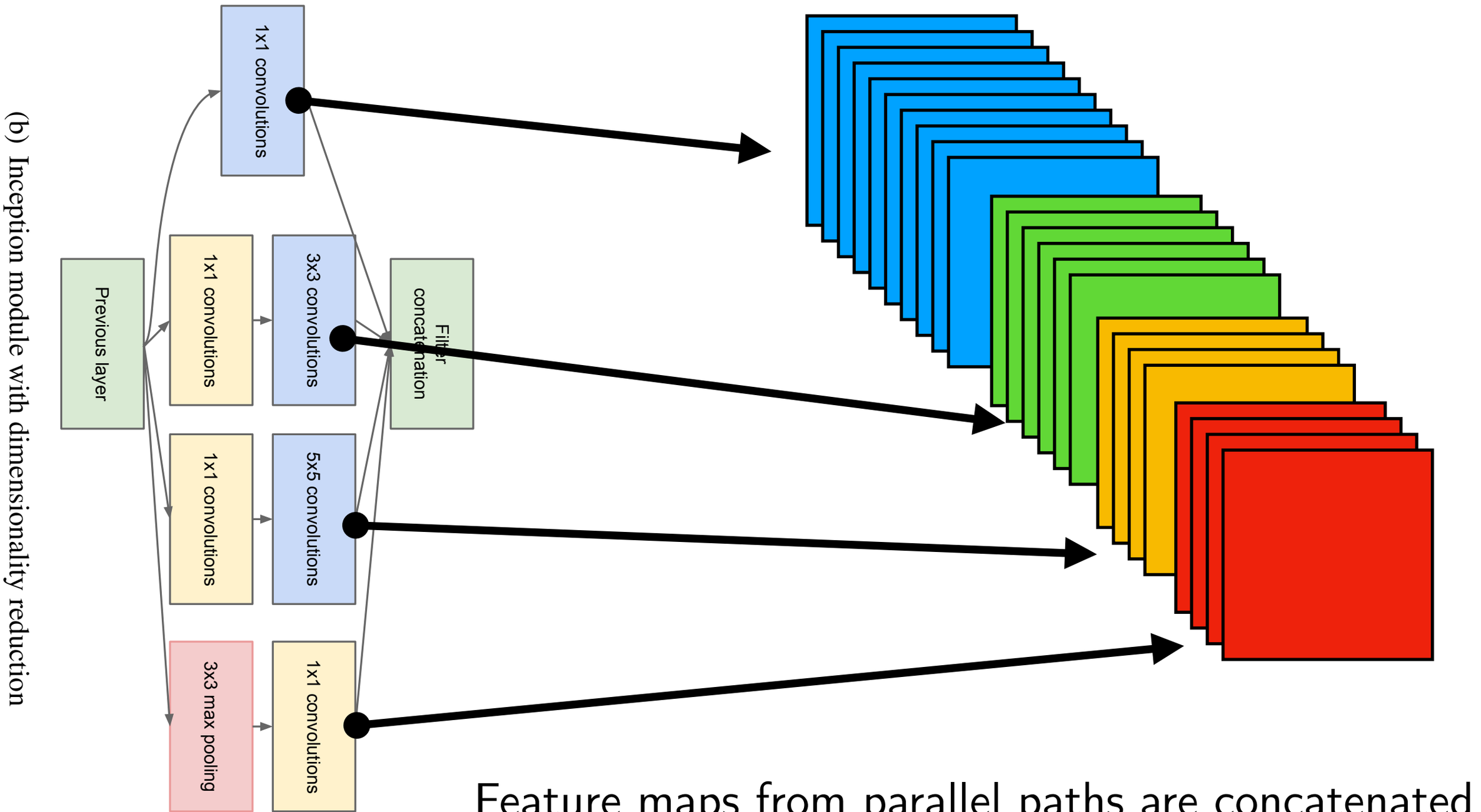
(b) Inception module with dimensionality reduction

Example:



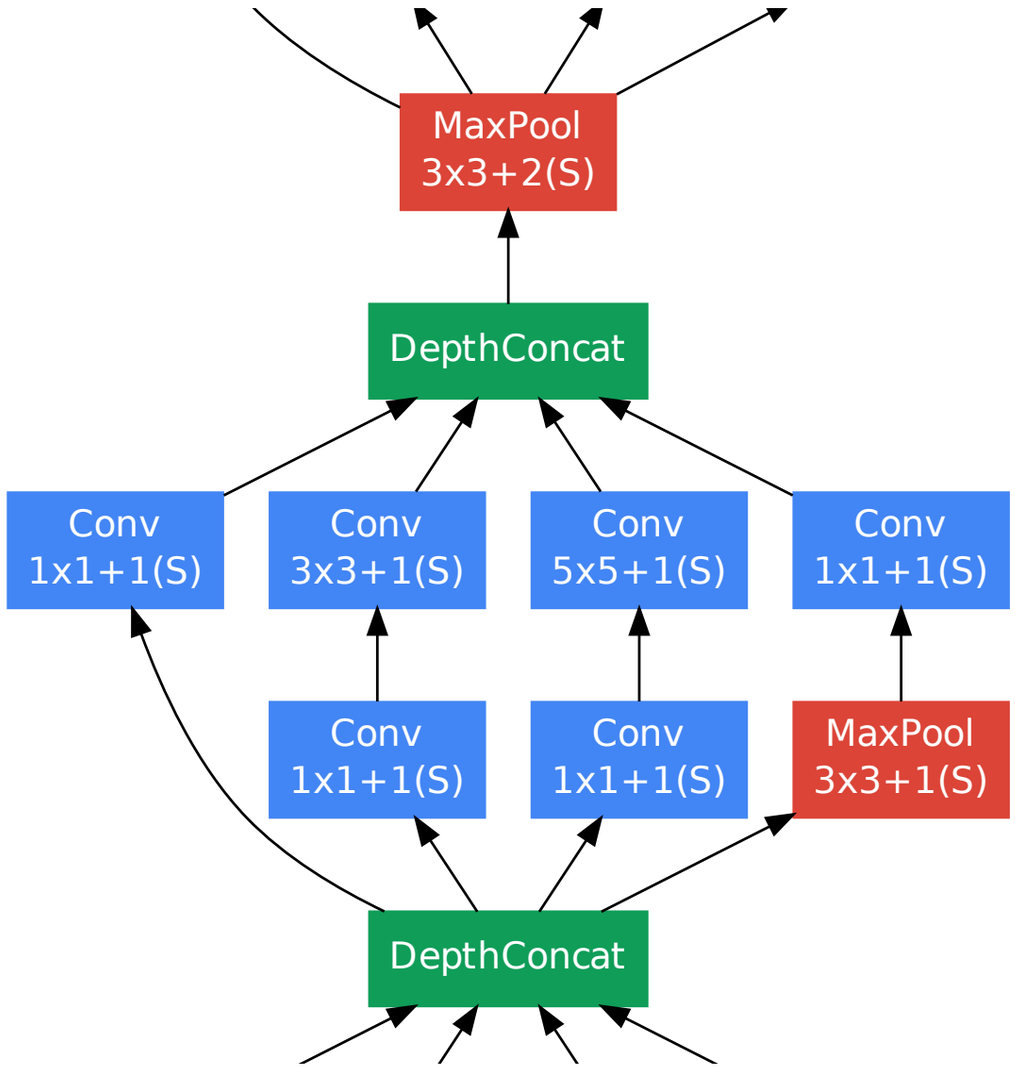
GoogLeNet / Inception v1

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9. 2015.



GoogLeNet / Inception v1

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9. 2015.



Inception v2

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (pp. 2818-2826).

Factorized the traditional large convolutions into multiple smaller convolutions

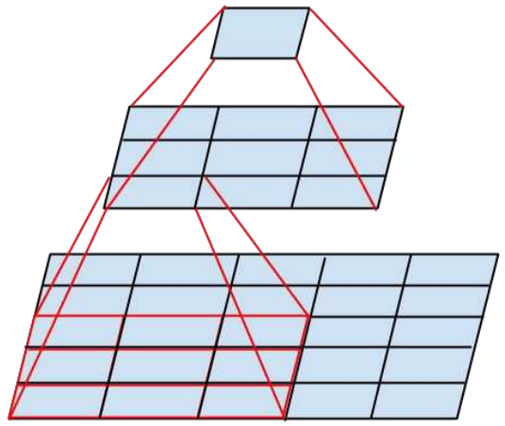


Figure 1. Mini-network replacing the 5×5 convolutions.

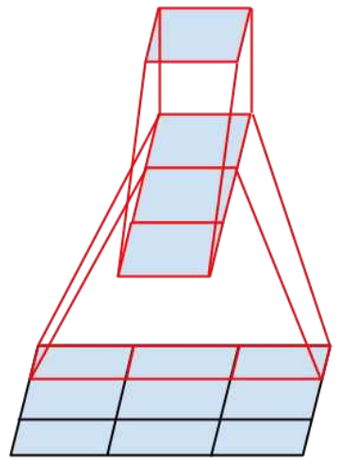


Figure 3. Mini-network replacing the 3×3 convolutions. The lower layer of this network consists of a 3×1 convolution with 3 output units.

Inception v2

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (pp. 2818-2826).

factorized the traditional large convolutions into multiple smaller convolutions

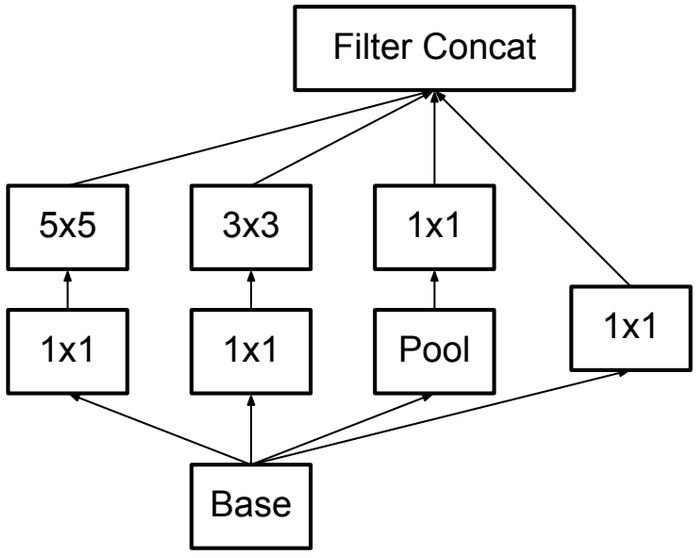


Figure 4. Original Inception module as described in [20].

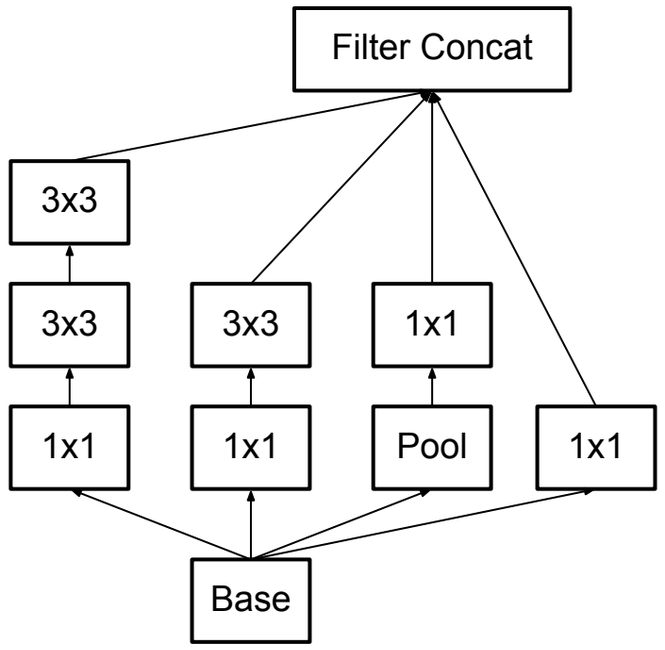


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

Inception v3

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (pp. 2818-2826).

Network	**Top-1** Error	**Top-5** Error	**Cost** Bn Ops
GoogLeNet [20]	29%	9.2%	1.5
BN-GoogLeNet	26.8%	-	**1.5**
BN-Inception [7]	25.2%	7.8	2.0
Inception-v3-basic	23.4%	-	3.8
Inception-v3-rmsprop RMSProp	23.1%	6.3	3.8
Inception-v3-smooth Label Smoothing	22.8%	6.1	3.8
Inception-v3-fact Factorized 7×7	21.6%	5.8	4.8
Inception-v3 BN-auxiliary	**21.2%**	**5.6%**	4.8

"Table 3 shows the experimental results about the recognition performance of our proposed architecture (Inception-v2) as described in Section 6. Each Inception-v2 line shows the result of the cumulative changes including the highlighted new modification plus all the earlier ones. [...]
BN auxiliary refers to the version in which the fully connected layer of the auxiliary classifier is also normalized, not just convolutions. We are referring to the model [Inception-v2 + BN auxiliary] as Inception-v3."

Inception v4

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017, February). Inception-v4, Inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI Conference on Artificial Intelligence*.

Here we give clear empirical evidence that training with residual connections accelerates the training of Inception networks significantly.

We also present several new streamlined architectures for both residual and non-residual Inception networks. These variations improve the single-frame recognition performance on the ILSVRC 2012 classification task significantly.

Computer Science > Computer Vision and Pattern Recognition

Exploring Randomly Wired Neural Networks for Image Recognition

Saining Xie, Alexander Kirillov, Ross Girshick, Kaiming He

(Submitted on 2 Apr 2019 (v1), last revised 8 Apr 2019 (this version, v2))

Neural networks for image recognition have evolved through extensive manual design from simple chain-like mc
paths. The success of ResNets and DenseNets is due in large part to their innovative wiring plans. Now, neural ar
exploring the joint optimization of wiring and operation types, however, the space of possible wirings is constrai
despite being searched. In this paper, we explore a more diverse set of connectivity patterns through the lens of
this, we first define the concept of a stochastic network generator that encapsulates the entire network generatic
view of NAS and randomly wired networks. Then, we use three classical random graph models to generate rando
are surprising: several variants of these random generators yield network instances that have competitive accura
results suggest that new efforts focusing on designing better network generators may lead to new breakthrough
spaces with more room for novel design.

Based on neural architecture search (NAS) and stochastic network generators

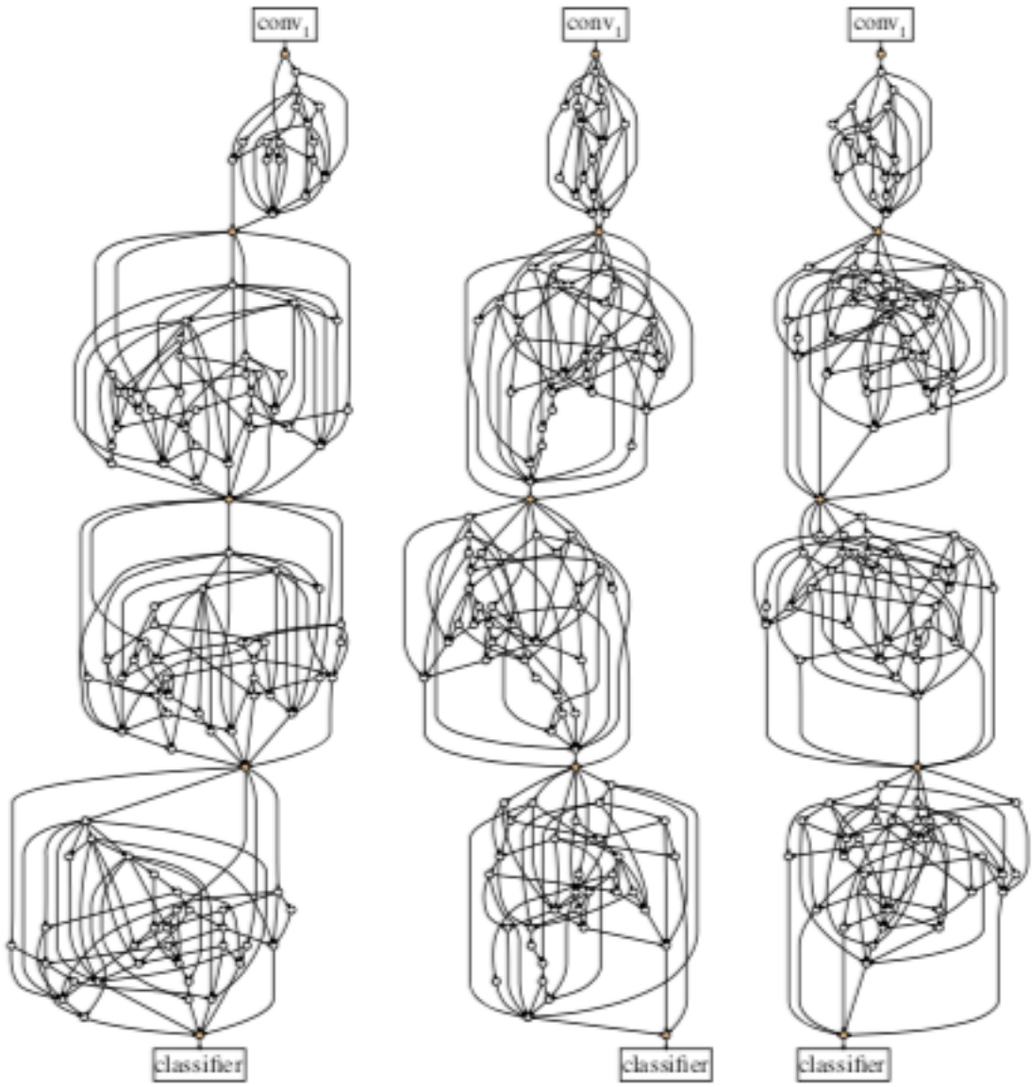
https://arxiv.org/abs/1904.01569

https://arxiv.org/abs/1904.01569

Exploring Randomly Wired Neural Networks for Image Recognition

Saining Xie, Alexander Kirillov, Ross Girshick, Kaiming He

(Submitted on 2 Apr 2019 (v1), last revised 8 Apr 2019 (this version, v2))



Based on neural architecture search (NAS) and stochastic network generators

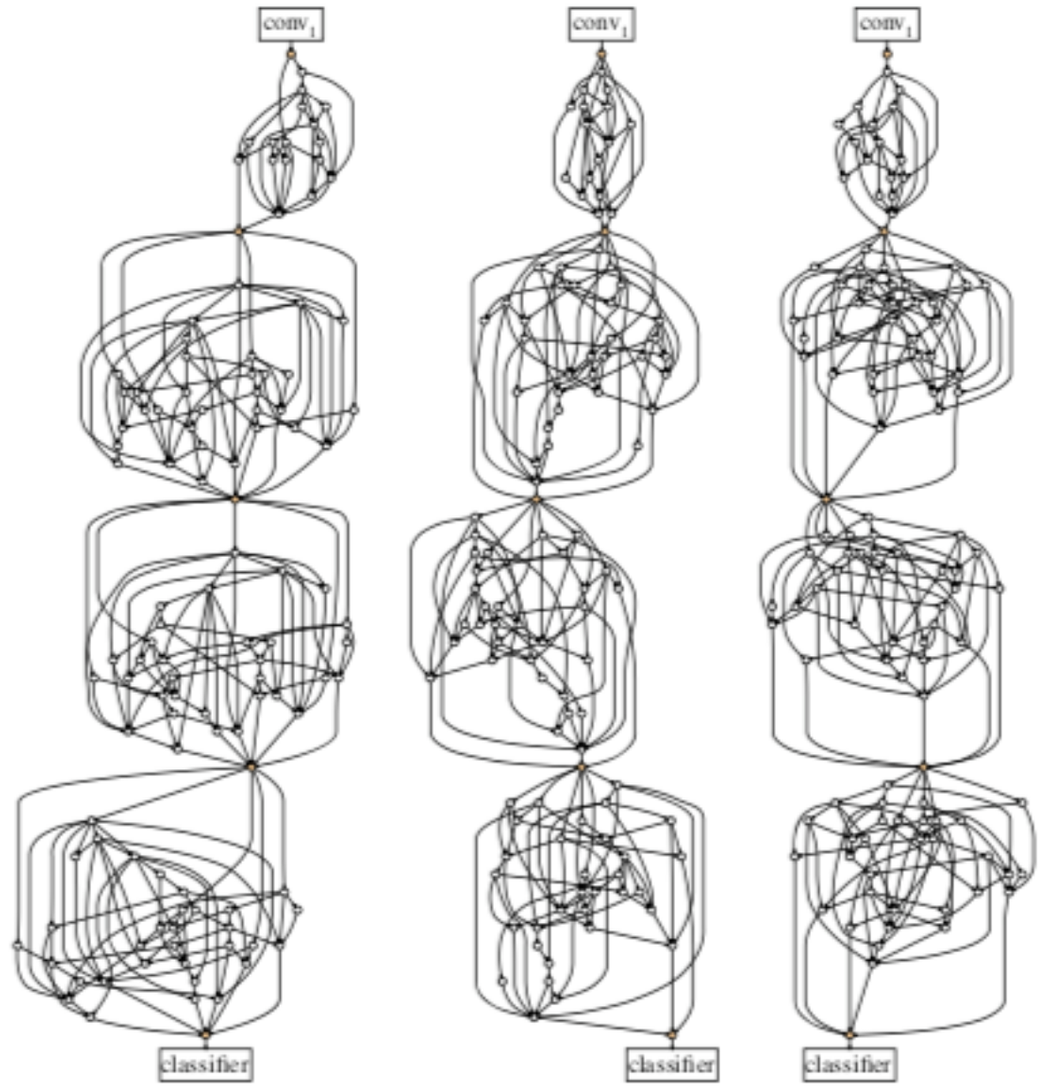
https://arxiv.org/abs/1904.01569

Exploring Randomly Wired Neural Networks for Image Recognition

Saining Xie, Alexander Kirillov, Ross Girshick, Kaiming He

(Submitted on 2 Apr 2019 (v1), last revised 8 Apr 2019 (this version, v2))

Also utilizes an
LSTM controller with
probabilistic behavior
(will discuss LSTMs in a different
context next lecture)



Based on neural architecture search (NAS) and stochastic network generators

arXiv.org > cs > arXiv:2004.02967

Computer Science > Machine Learning

Evolving Normalization-Activation Layers

Hanxiao Liu, Andrew Brock, Karen Simonyan, Quoc V. Le

(Submitted on 6 Apr 2020)

https://arxiv.org/abs/2004.02967

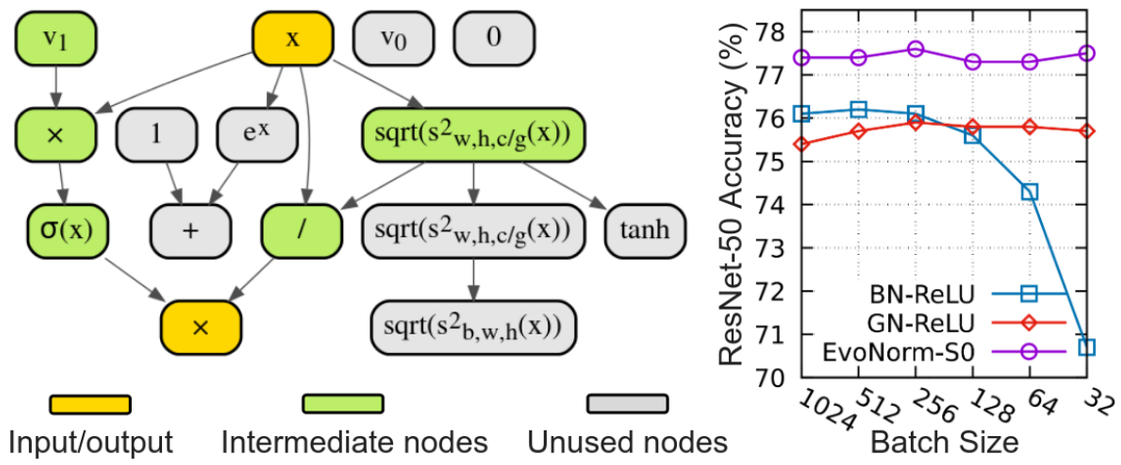


Figure 1. Left: Computation graph of a searched normalization-activation layer that is batch-independent, named EvoNorm-S0. The corresponding expression is $\sigma(v_1 x)\frac{x}{\sqrt{s^2_{w,h,c/g}(x)}}\gamma+\beta$, in con-

Lecture Overview

1. Padding (control output size in addition to stride)
2. Spatial Dropout and BatchNorm
3. Considerations for CNNs on GPUs
4. Common Architectures
    A. VGG16 (simple, deep CNN)
    B. ResNet and skip connections
    C. Fully convolutional networks (no fully connected layers)
    D. Inception (parallel convolutions and auxiliary losses)
5. **Transfer learning**

Transfer Learning

- A technique that may be useful for your class projects
- Key idea:
  - ✦ Feature extraction layers may be generally useful
  - ✦ Use a pre-trained model (e.g., pre-trained on ImageNet)
  - ✦ Freeze the weights: Only train last layer (or last few layers)
- Related approach: Fine-tuning, train a pre-trained network on your smaller dataset

Which Layers to Replace & Train?

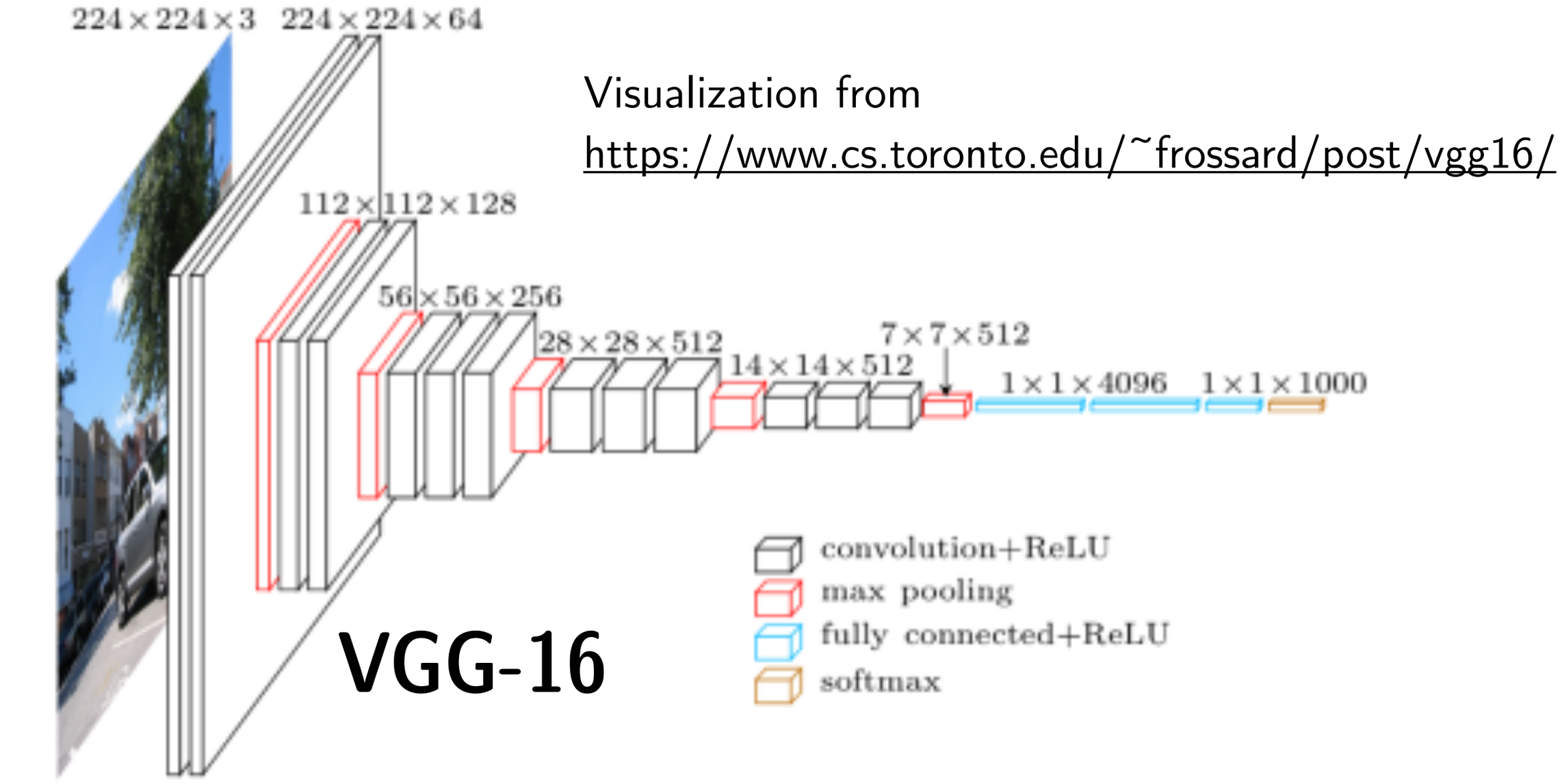
Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (pp. 1725-1732).

Model	**3-fold Accuracy**
Soomro et al [22]	43.9%
Feature Histograms + Neural Net	59.0%
Train from scratch	41.3%
Fine-tune top layer	64.1%
Fine-tune top 3 layers	**65.4**%
Fine-tune all layers	62.2%

Table 3: Results on UCF-101 for various Transfer Learning approaches using the Slow Fusion network.

Transfer Learning

PyTorch implementation: https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/vgg16-transferlearning.ipynb

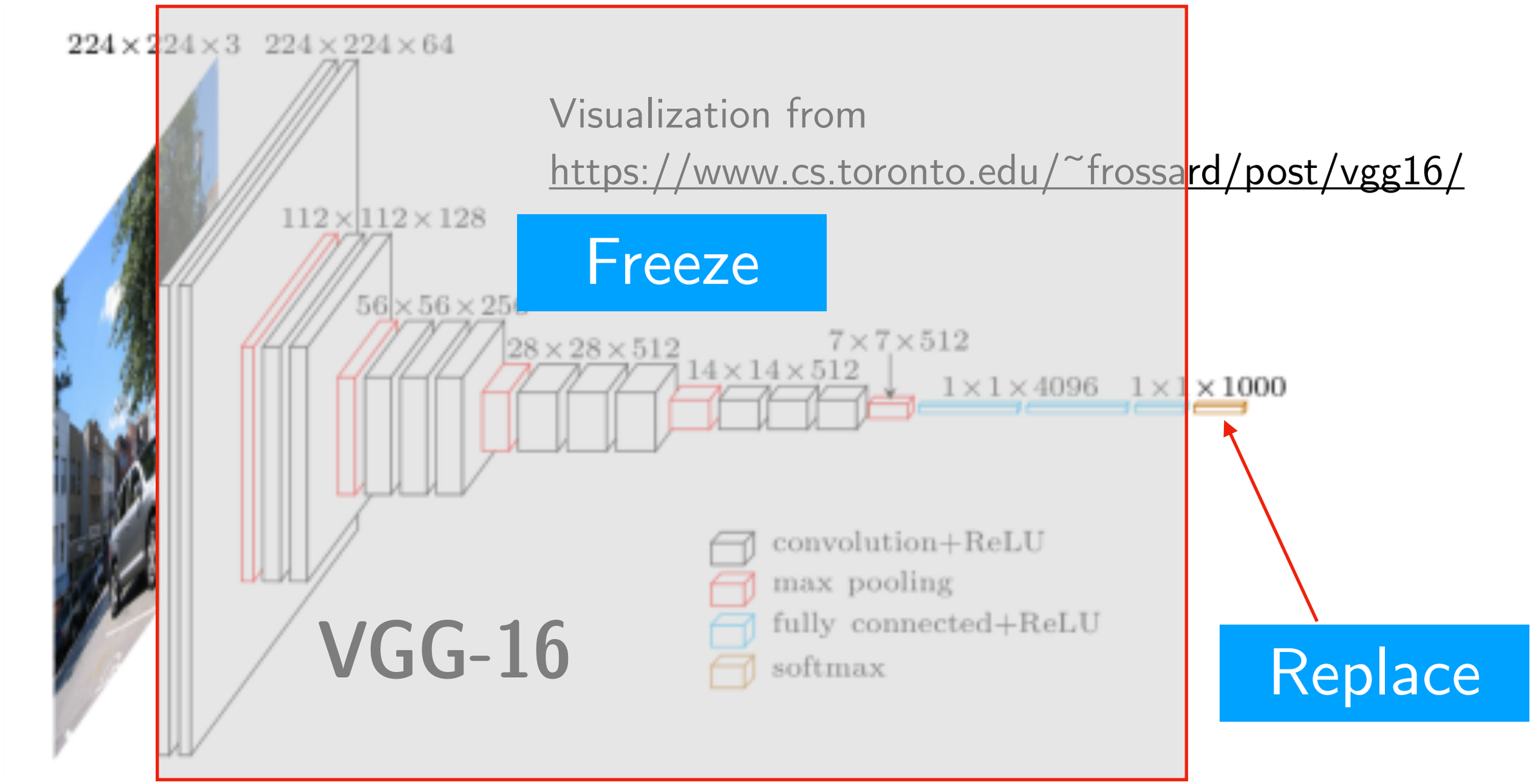


Visualization from
https://www.cs.toronto.edu/~frossard/post/vgg16/

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

Transfer Learning

PyTorch implementation: https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/vgg16-transferlearning.ipynb



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

Transfer Learning

https://pytorch.org/docs/stable/torchvision/models.html

TORCHVISION.MODELS

The models subpackage contains definitions of models for addressing different tasks, including: image classification, pixelwise semantic segmentation, object detection, instance segmentation, person keypoint detection and video classification.

Classification

The models subpackage contains definitions for the following model architectures for image classification:

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNet v2
- ResNeXt
- Wide ResNet
- MNASNet

Transfer Learning

https://pytorch.org/docs/stable/torchvision/models.html

Instancing a pre-trained model will download its weights to a cache directory. This directory can be set using the *TORCH_MODEL_ZOO* environment variable. See `torch.utils.model_zoo.load_url()` for details.

Some models use modules which have different training and evaluation behavior, such as batch normalization. To switch between these modes, use `model.train()` or `model.eval()` as appropriate. See `train()` or `eval()` for details.

All pre-trained models expect input images normalized in the same way, i.e. mini-batches of 3-channel RGB images of shape (3 x H x W), where H and W are expected to be at least 224. The images have to be loaded in to a range of [0, 1] and then normalized using `mean = [0.485, 0.456, 0.406]` and `std = [0.229, 0.224, 0.225]`. You can use the following transform to normalize:

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
```

Transfer Learning Example

PyTorch implementation: https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/vgg16-transferlearning.ipynb

Optional Reading Material

http://www.deeplearningbook.org/contents/convnets.html