Lecture 13

Introduction to Convolutional Neural Networks

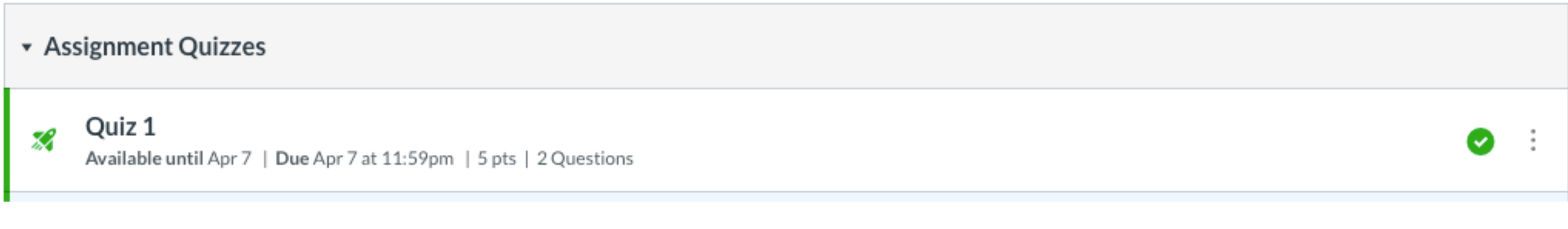
Part 2

STAT 453: Deep Learning, Spring 2020

Sebastian Raschka

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

Quiz


Assignment Quizzes
Quiz 1
Available until Apr 7 | Due Apr 7 at 11:59pm | 5 pts | 2 Questions

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

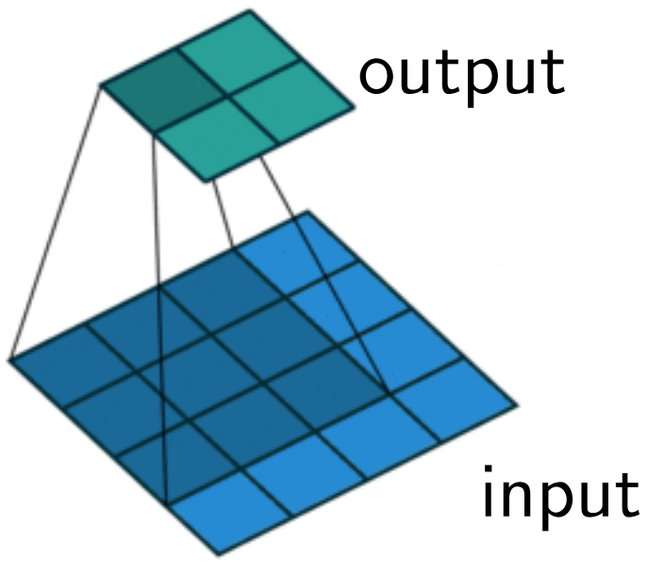
$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

"floor" function

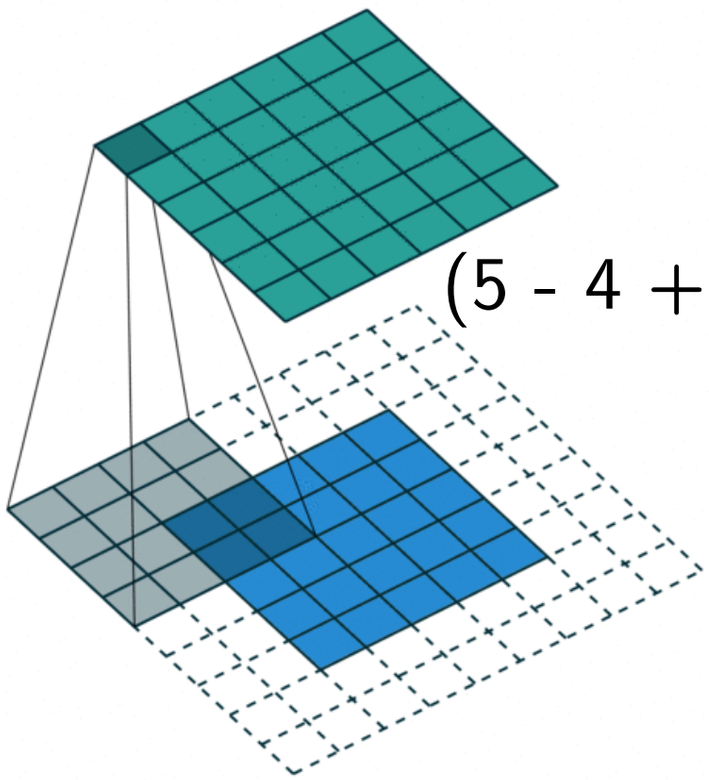
stride

kernel size

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

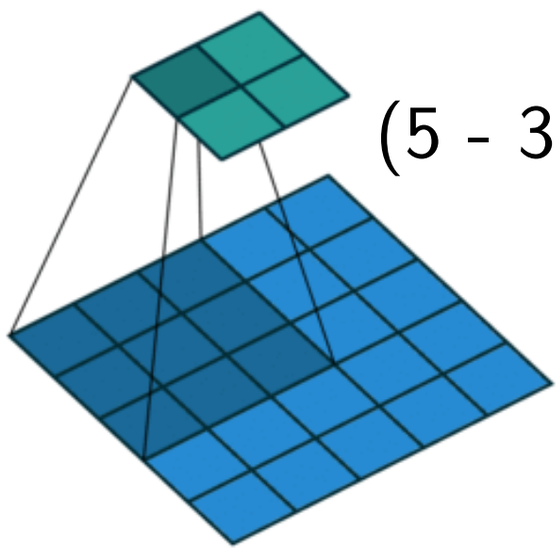


No padding, stride=1



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

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)

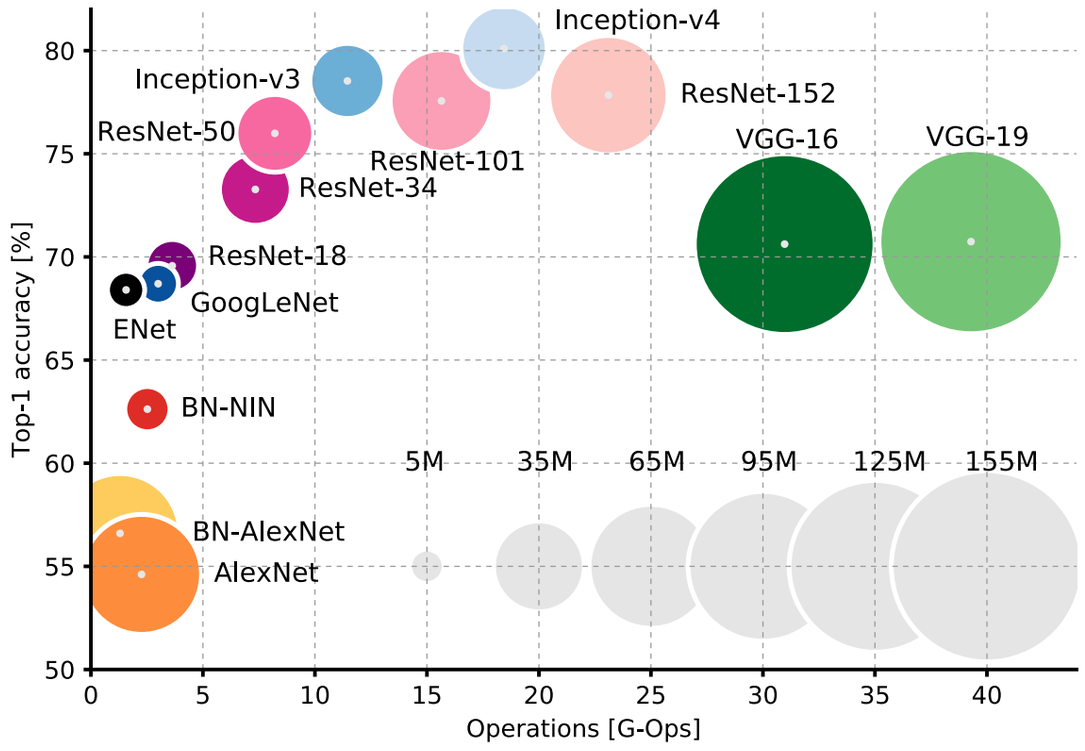
```
import torch
import torch.nn as nn
import torch.nn.functional as F

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

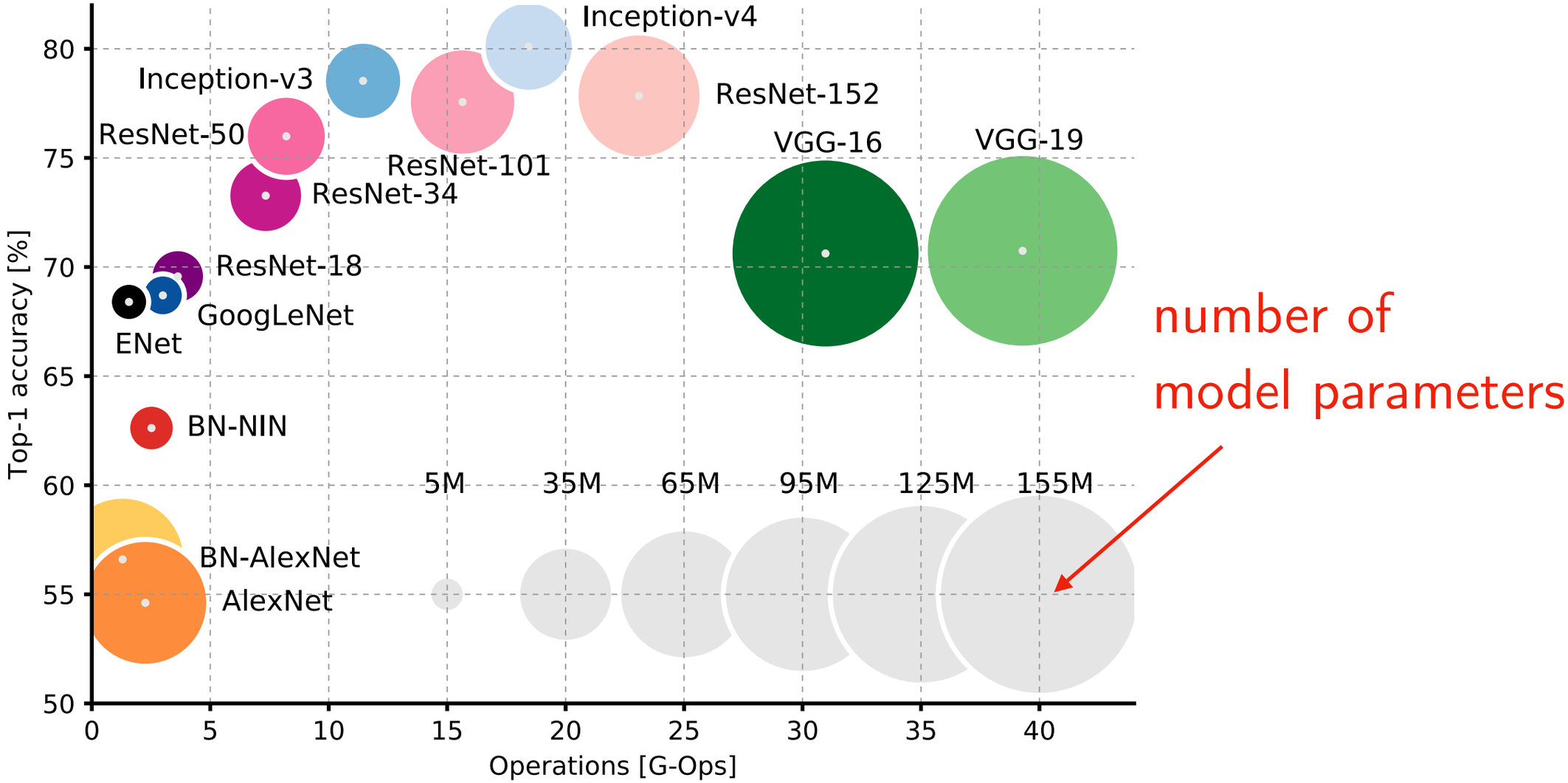
Common Architectures Revisited

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

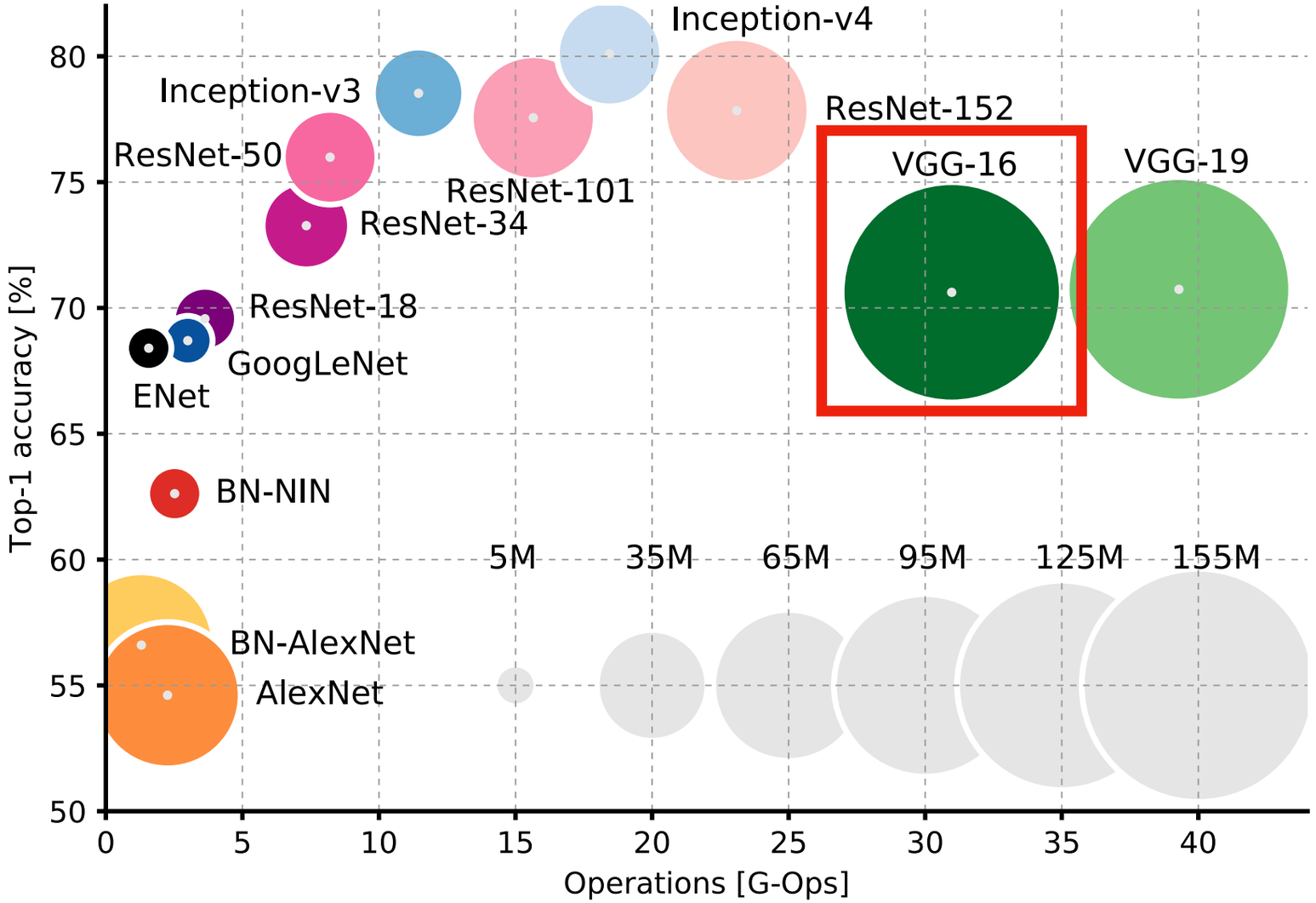


Canziani, A., Paszke, A., & Culurciello, E. (2016). An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*.

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:

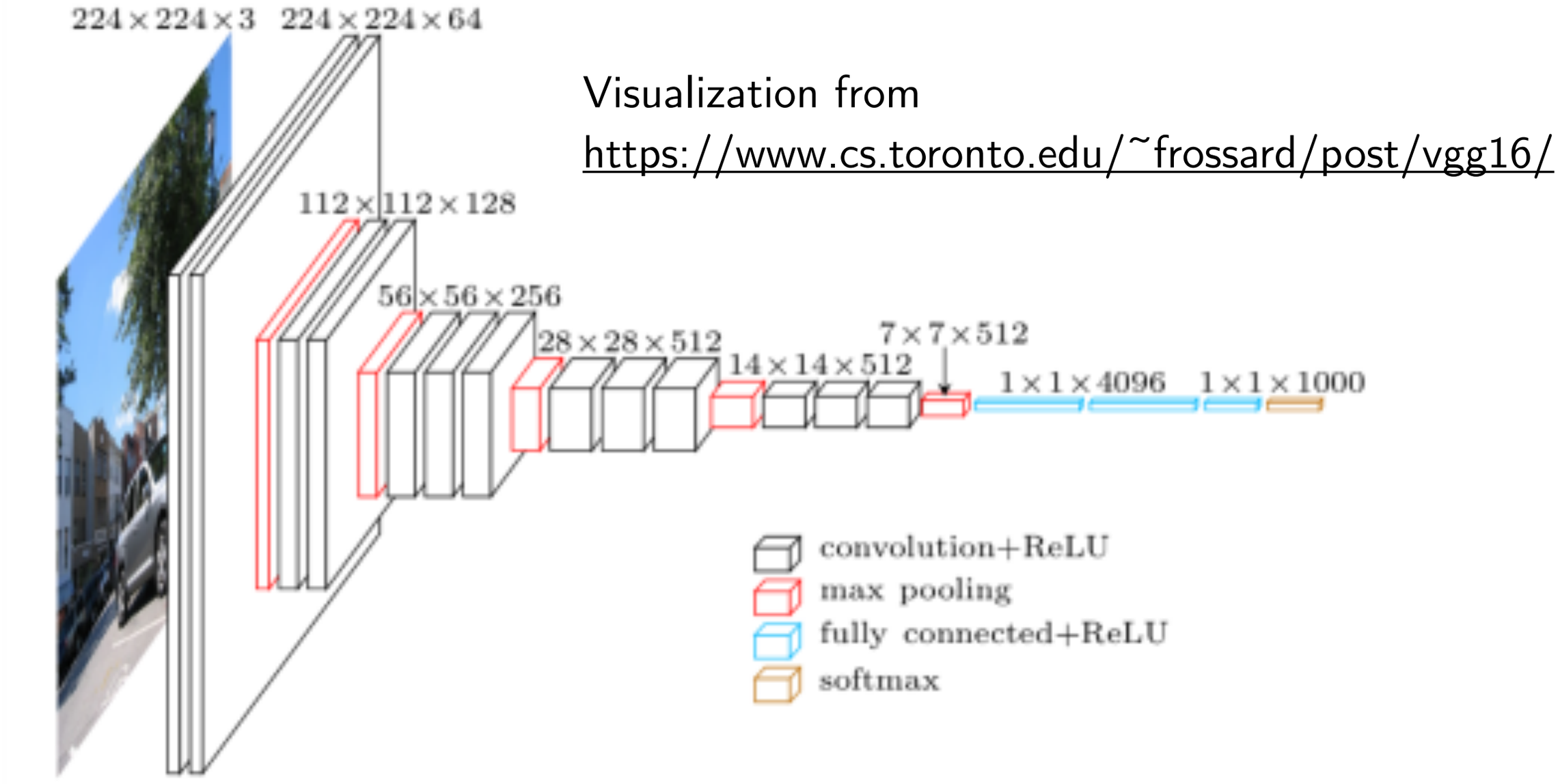
very large number of parameters and slow

(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



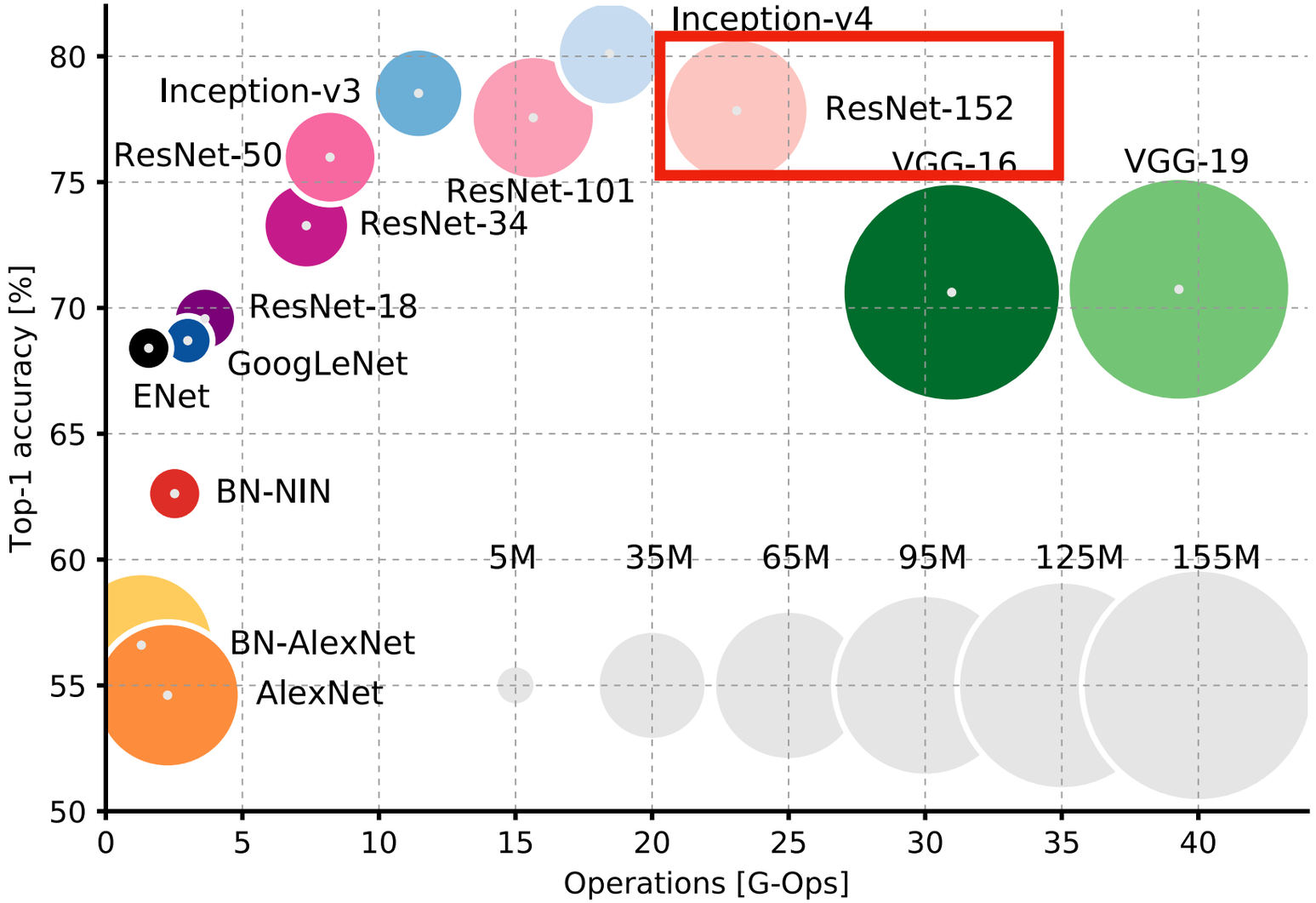
Visualization from
https://www.cs.toronto.edu/~frossard/post/vgg16/

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

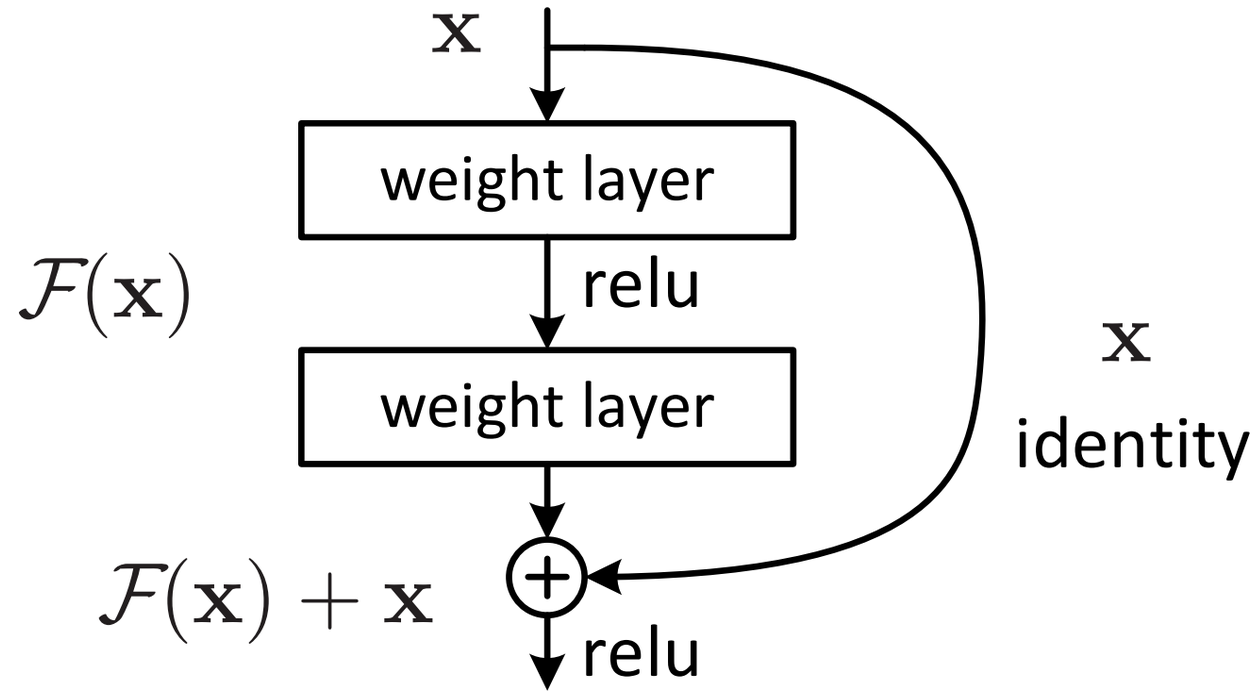
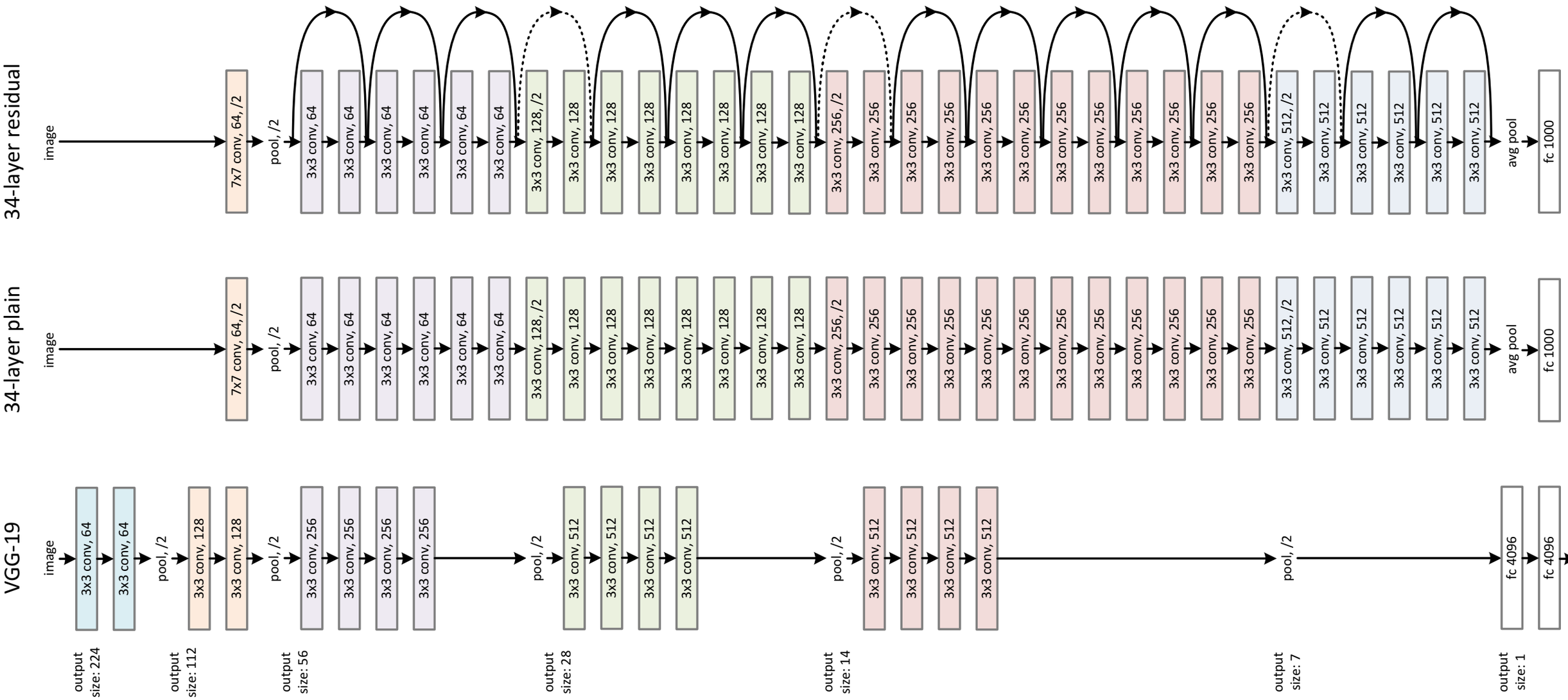


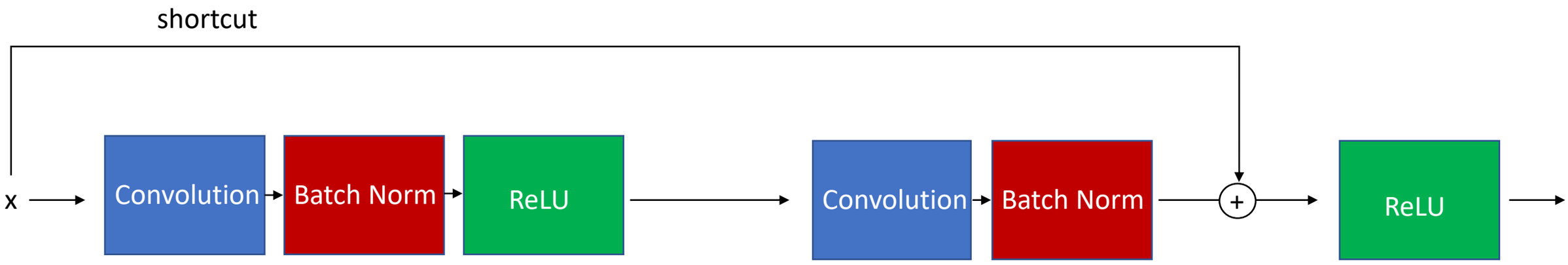
Figure 2. Residual learning: a building block.

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.

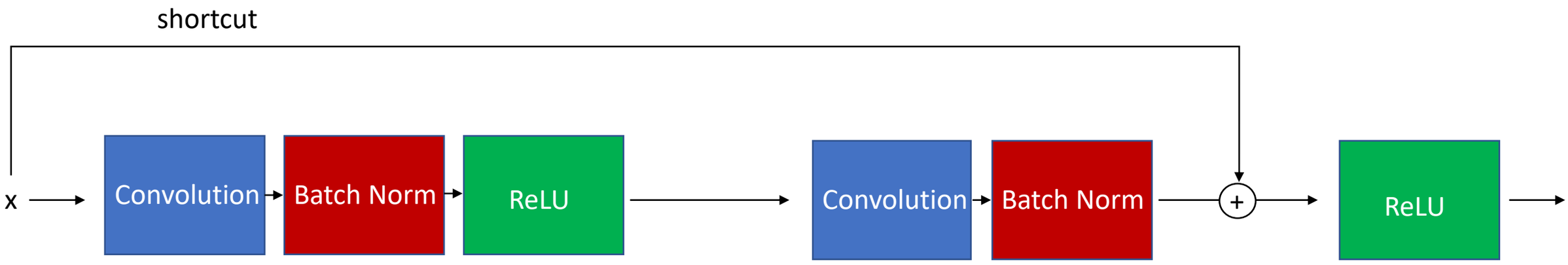


ResNets



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

ResNets



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

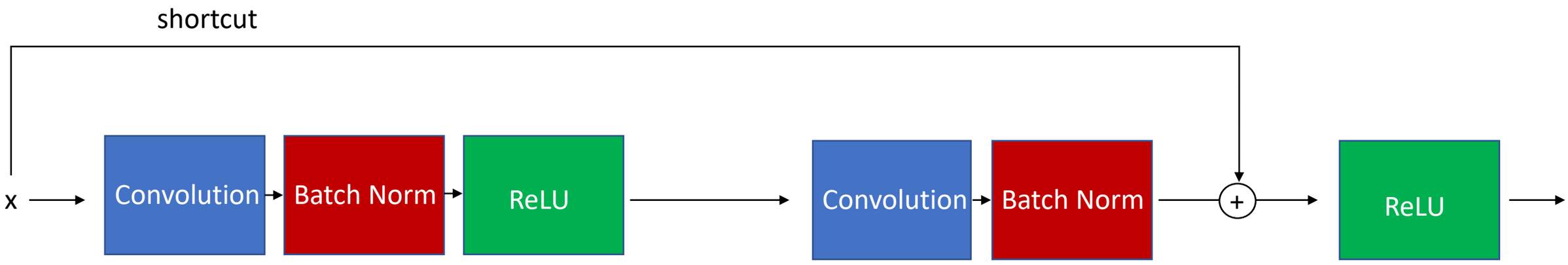
$$= \sigma\left(a^{(l+1)}W^{(l+2)} + b^{(l+2)} + a^{(l)}\right)$$

If all weights and the bias are zero, then

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

due to ReLU

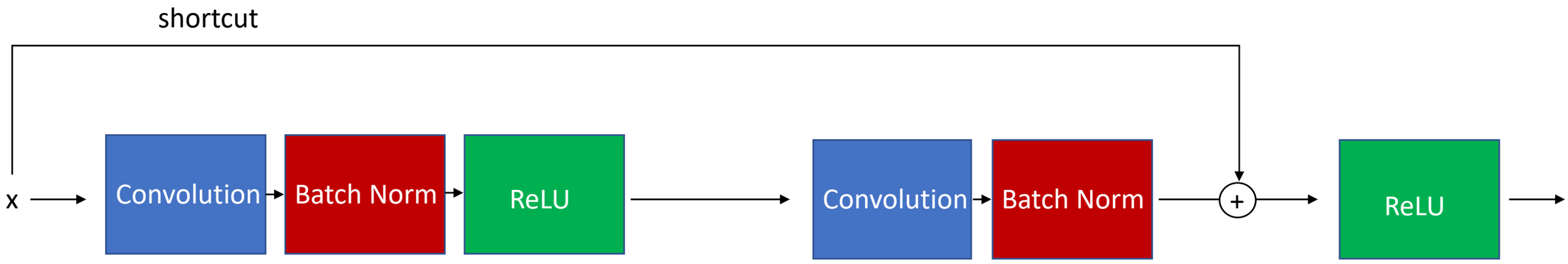
ResNets

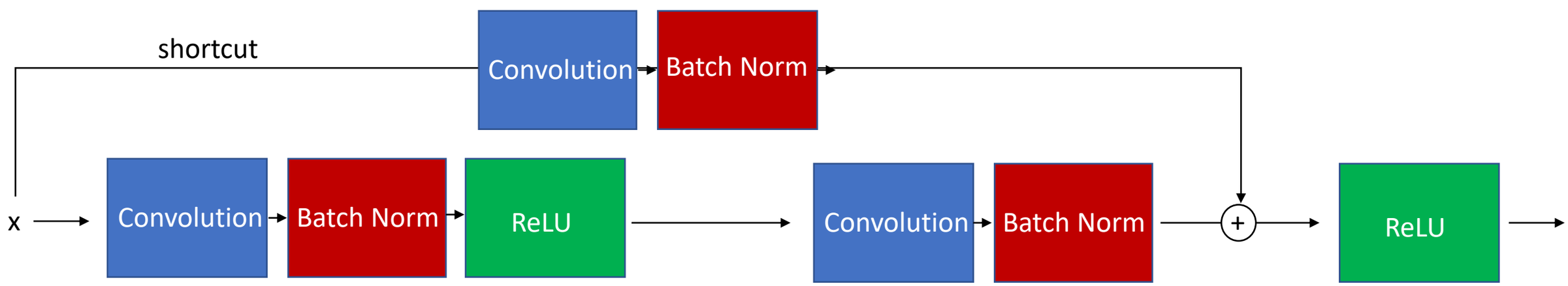


$$a^{(l+2)} = \sigma\big(z^{(l+2)} + a^{(l)}\big)$$

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

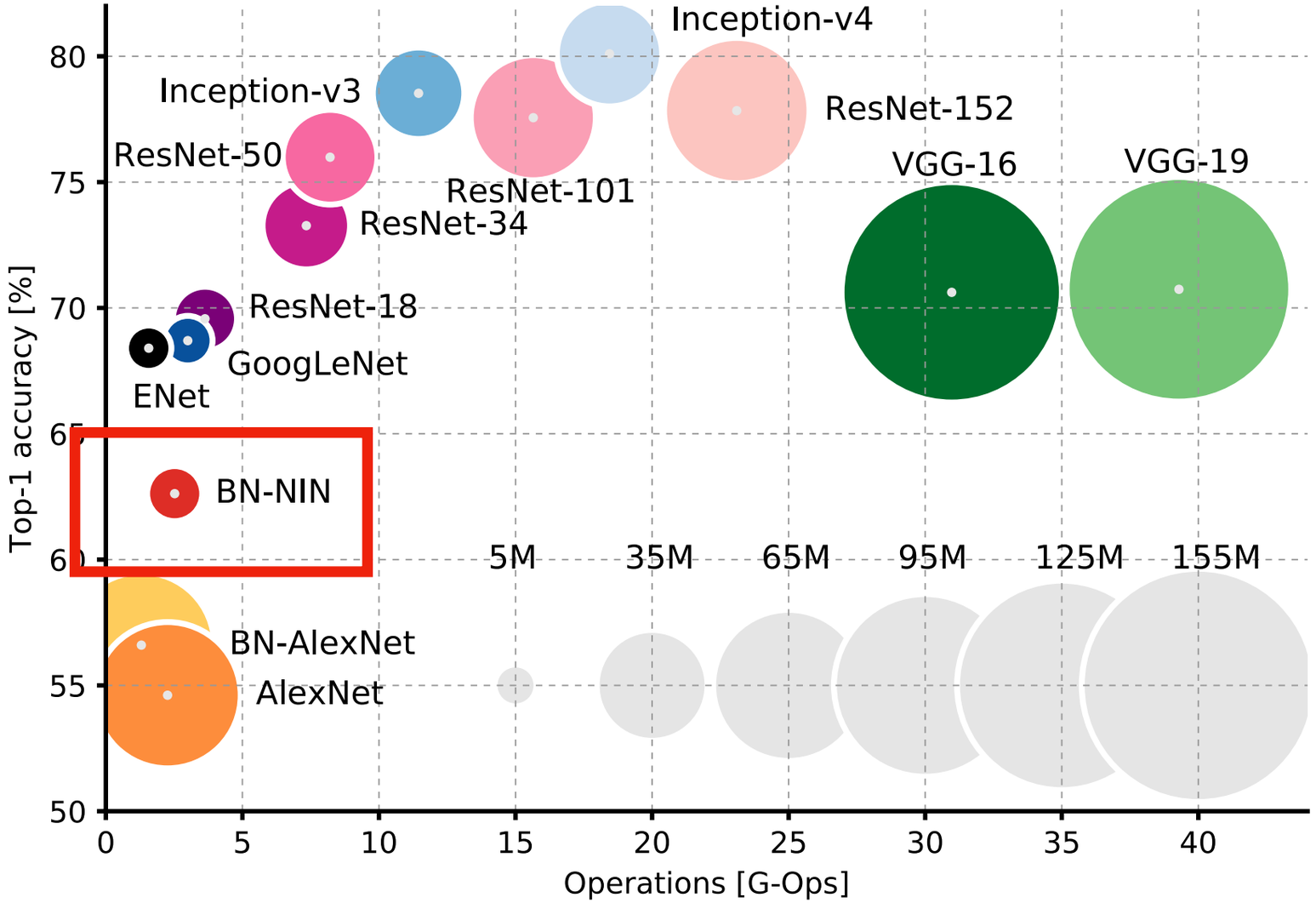
https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/resnet-152.ipynb

(Can be substantially improved with more hyperparameter tuning)

Lecture Overview

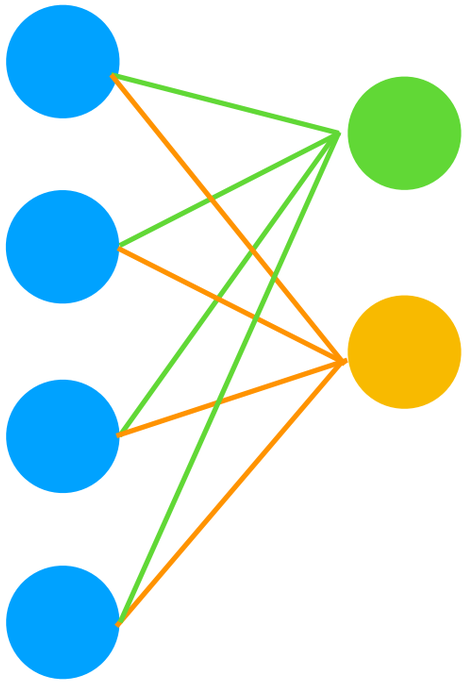
1. Padding (control output size in addition to stride)
2. Spatial Dropout and BatchNorm
3. Considerations for CNNs on GPUs
4. **Common Architectures**
   - A. VGG16 (simple, deep CNN)
   - B. ResNet and skip connections
   - **C. Fully convolutional networks (no fully connected layers)**
   - D. Inception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

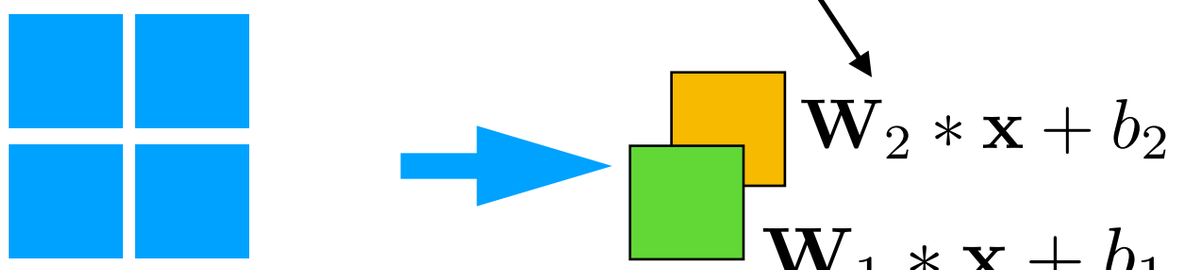


$$\mathbf{w}_1^T \mathbf{x} + b_1$$

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

Fully connected layer

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



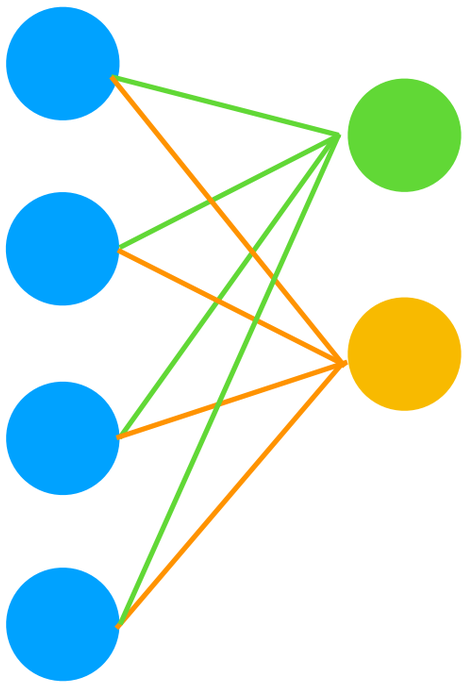
$$\mathbf{W}_2 * \mathbf{x} + b_2$$

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

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

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

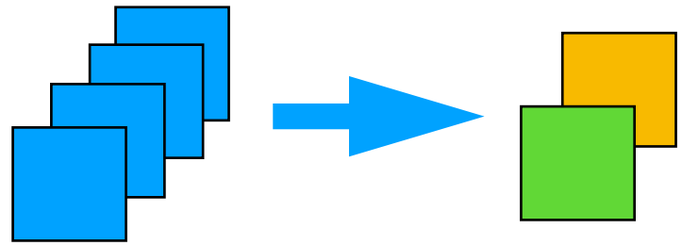
Side Note: It is Possible to Replace Fully Connected Layers by Convolutional Layers



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

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

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)

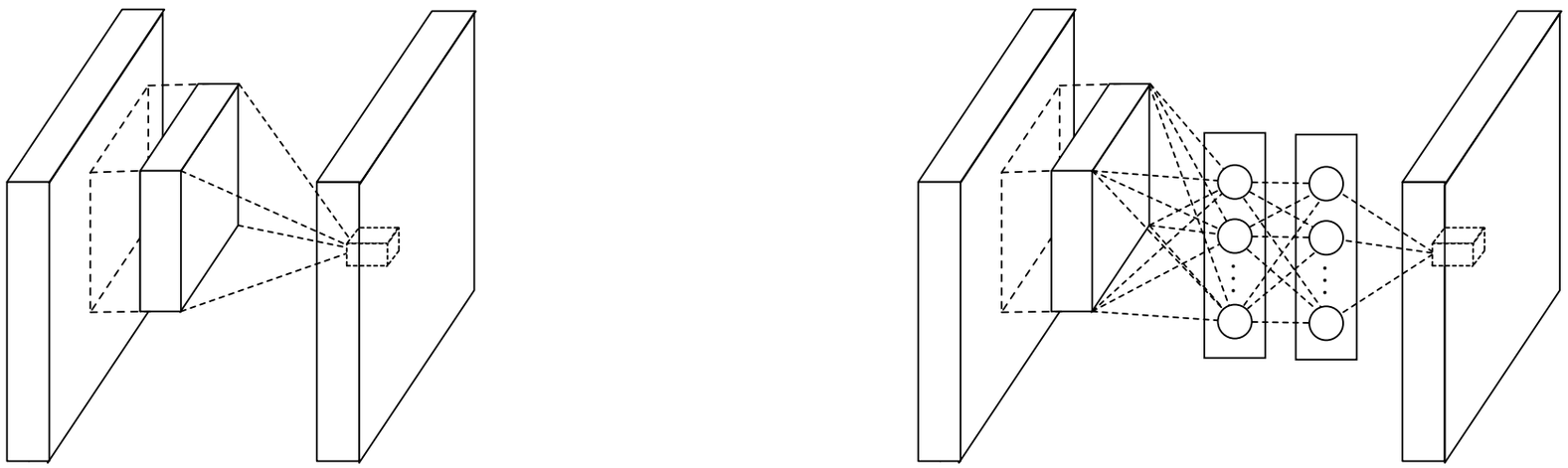
- A convolution kernel can be thought of as a generalized linear model (GLM)
- Using a "sophisticated" nonlinear function approximator (e.g., an MLP) may enhance the abstraction ability of the local model
- => Replace GLM by "micro network" (sliding an MLP over the feature map)

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

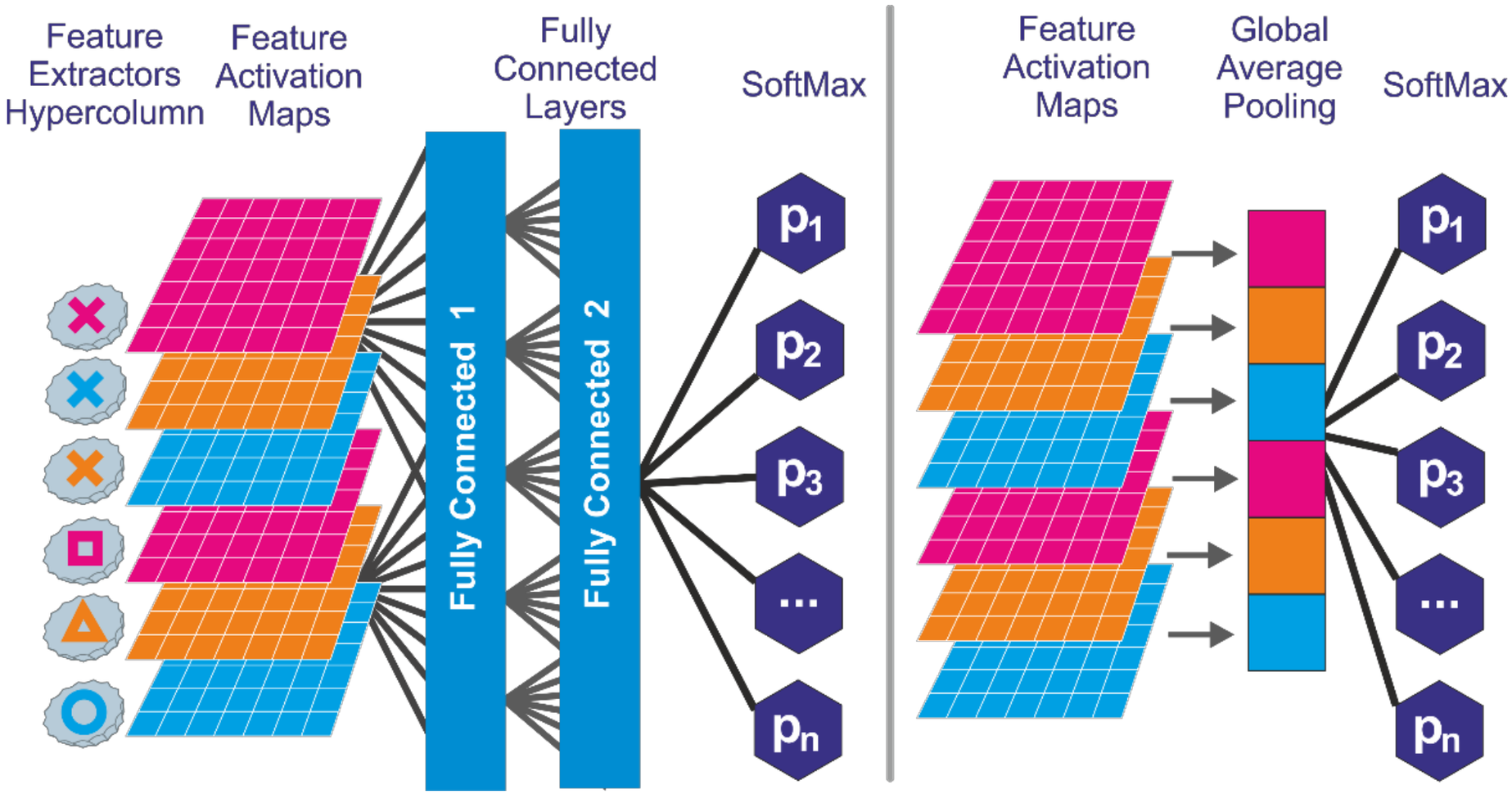


Figure 16: Global average pooling layer replacing the fully connected layers. The output layer implements a Softmax operation with $p_1, p_2, \ldots, p_n$ the predicted probabilities for each class.

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).

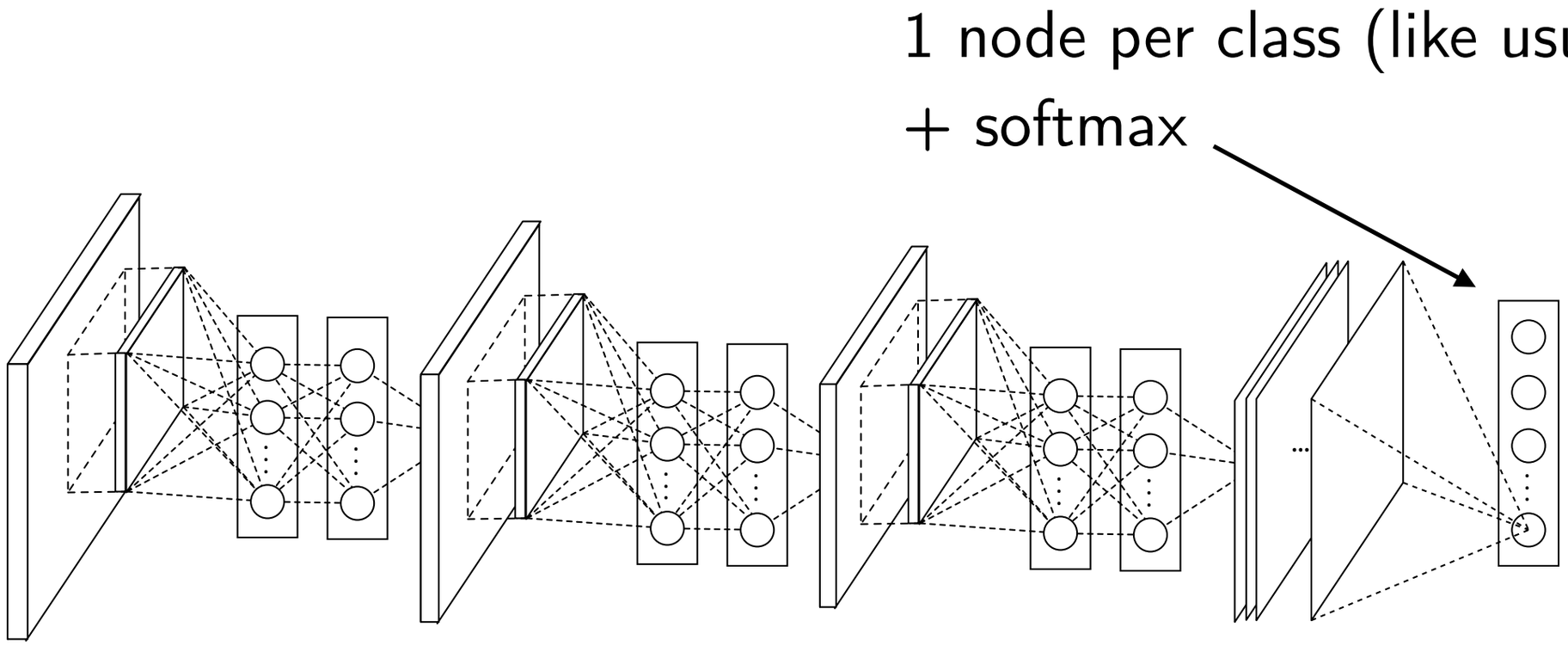


Figure 2: The overall structure of Network In Network. In this paper the NINs include the stacking of three mlpconv layers and one global average pooling layer.

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

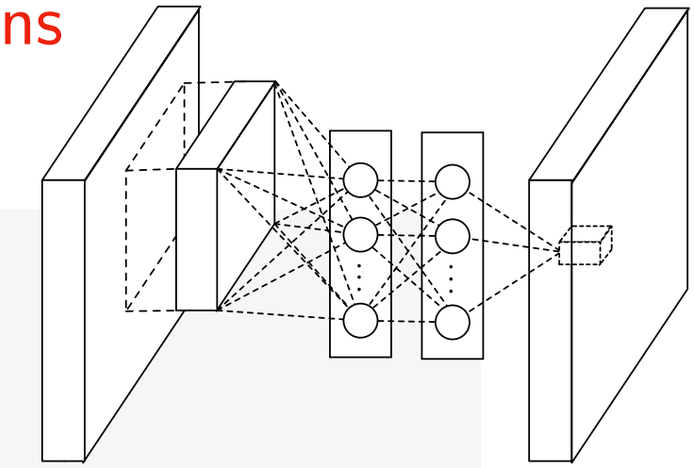
Network in Network (NiN)

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"

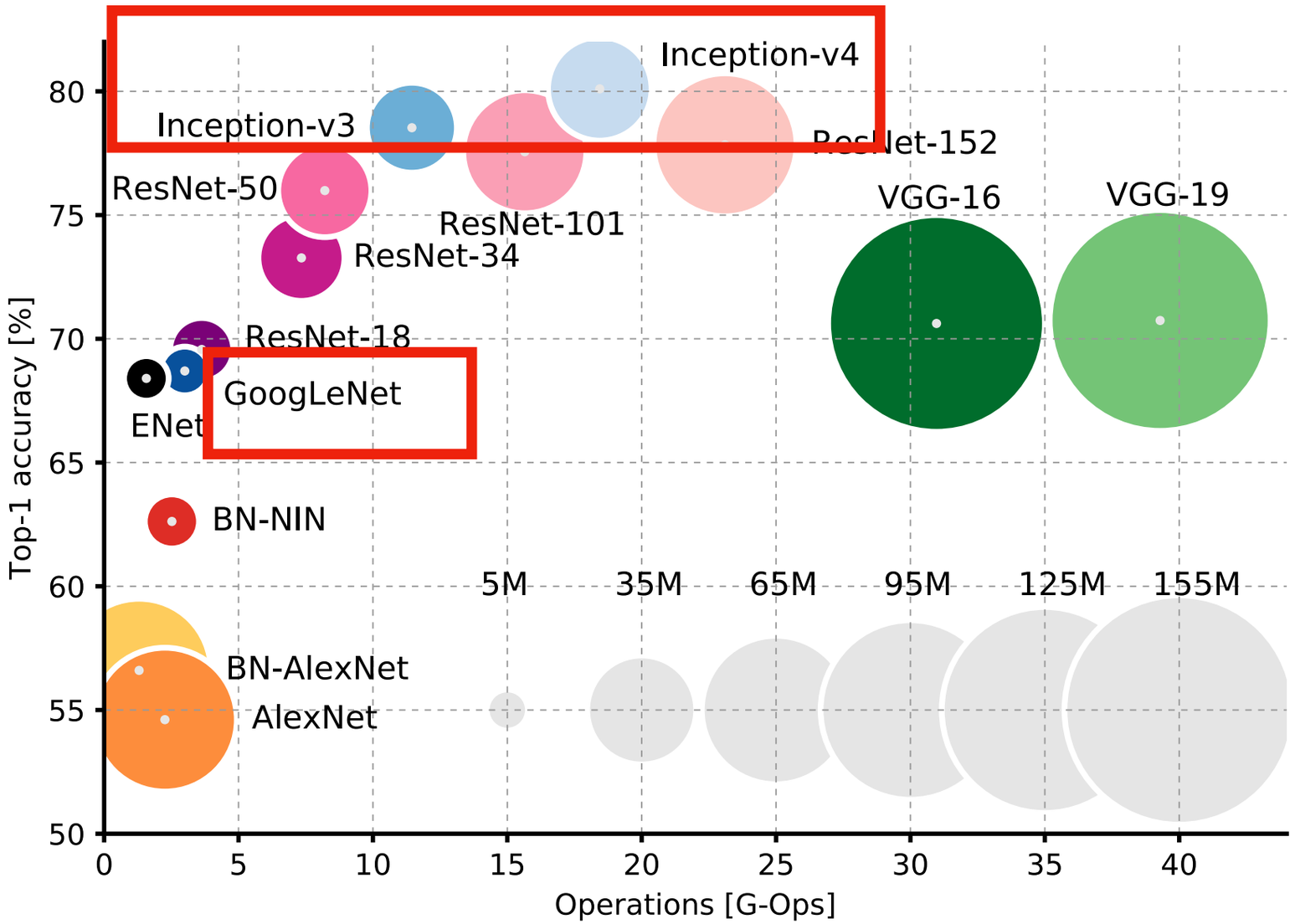
Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. "Striving for simplicity: The all convolutional net." *arXiv preprint arXiv:1412.6806* (2014).

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.

3 key ideas:

1. 1x1 in the middle of the network
2. global average pooling instead of fully connected layers
3. inception module

Note that the name inception doesn't have to make sense and is unrelated to the main concept; according to the authors, it comes from an internet meme that is in turn based on a movie called "Inception"

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

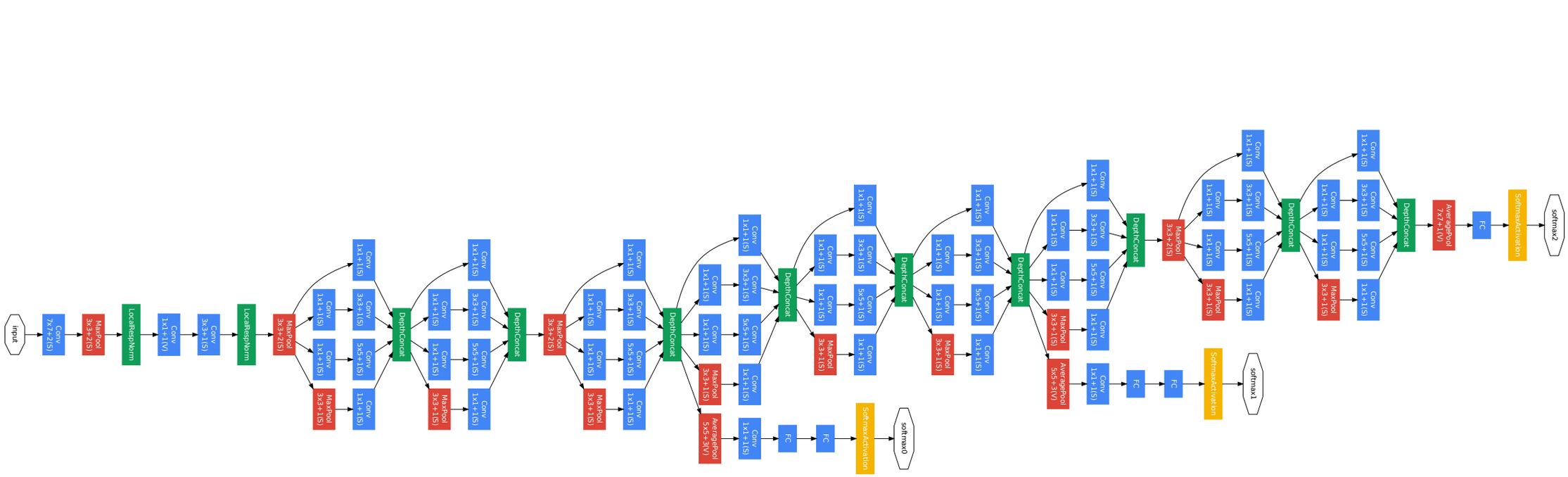


Figure 3: GoogLeNet network with all the bells and whistles.

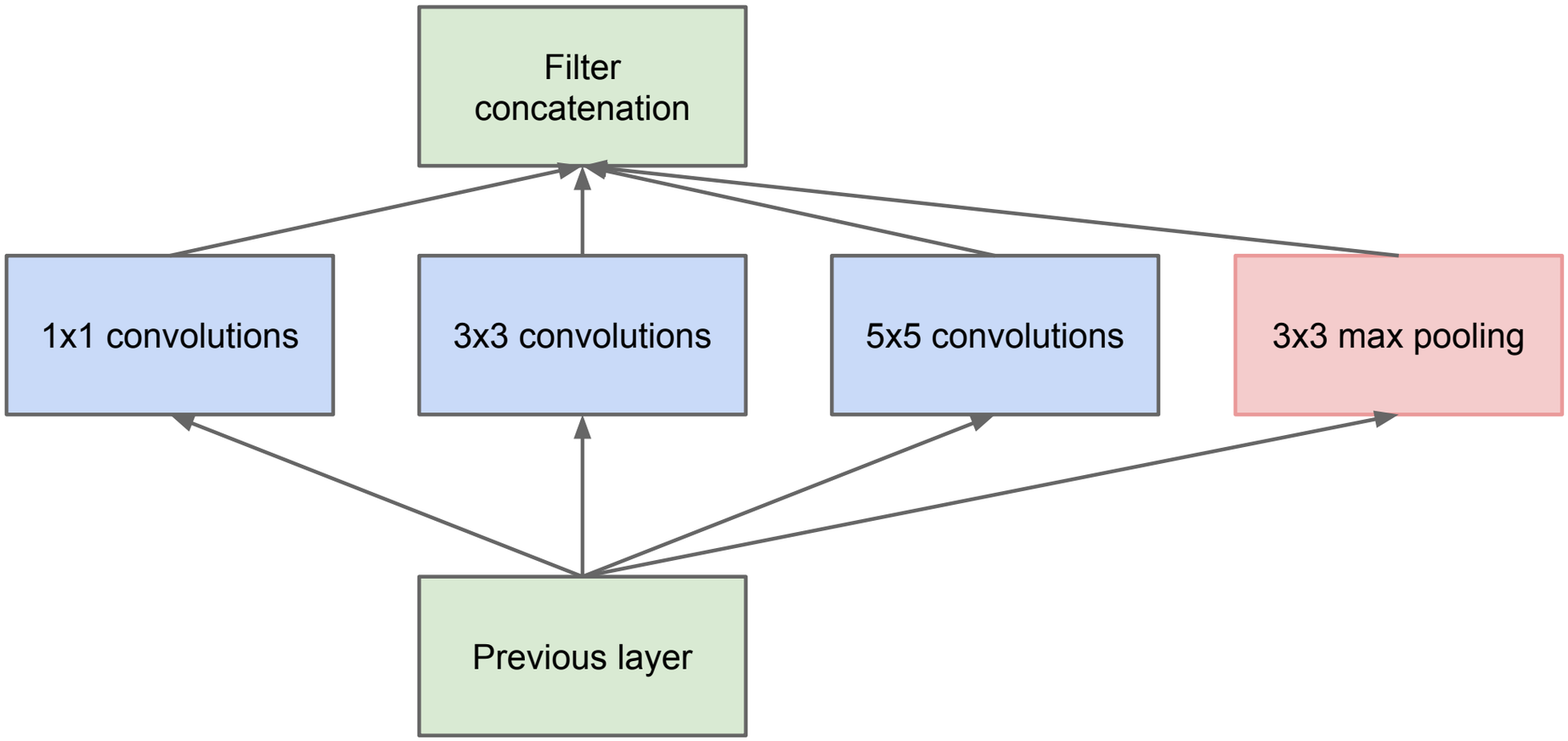
GoogLeNet / Inception v1

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

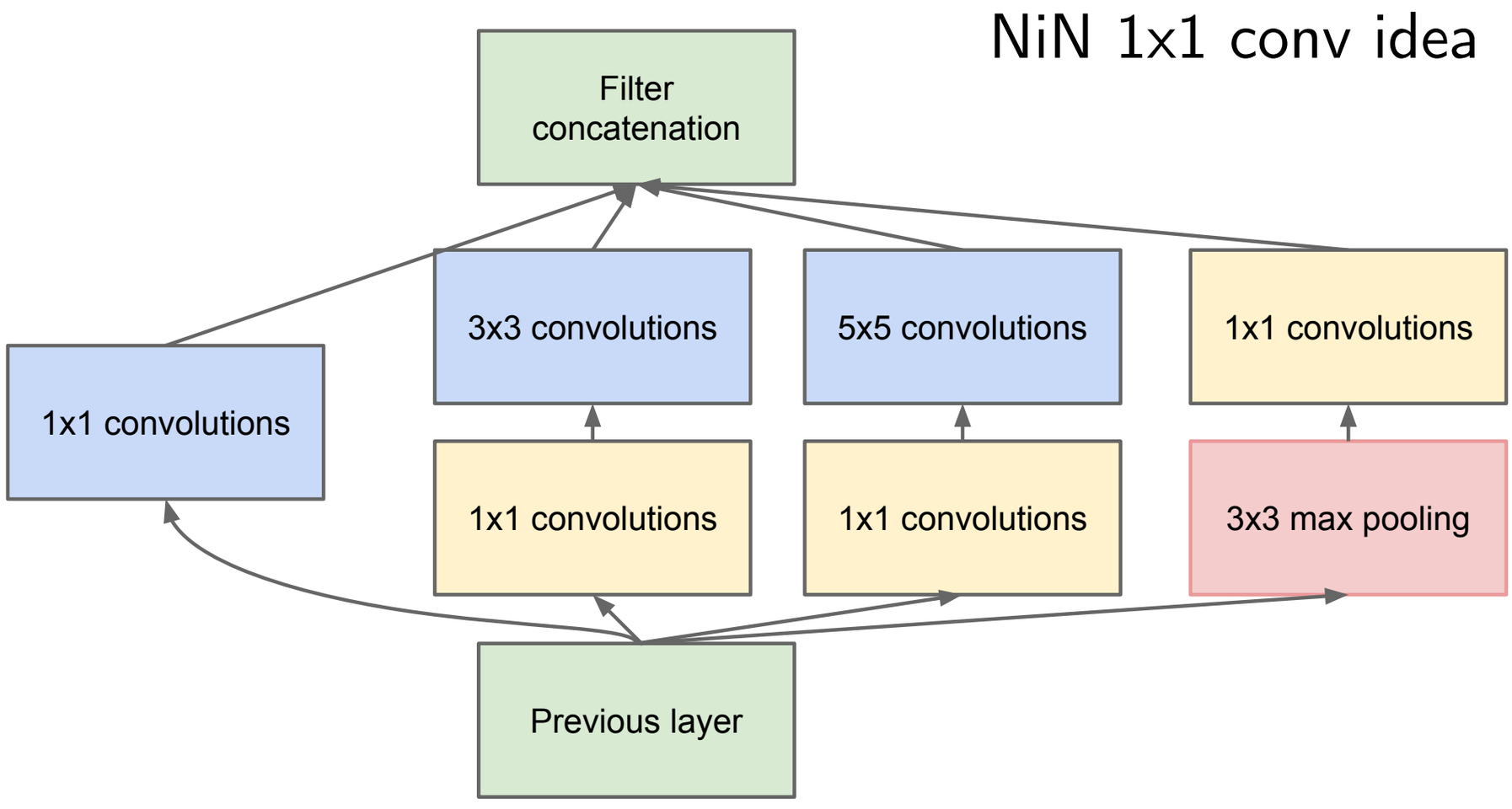
Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9. 2015.



(a) Inception module, naïve version

GoogLeNet / Inception v1

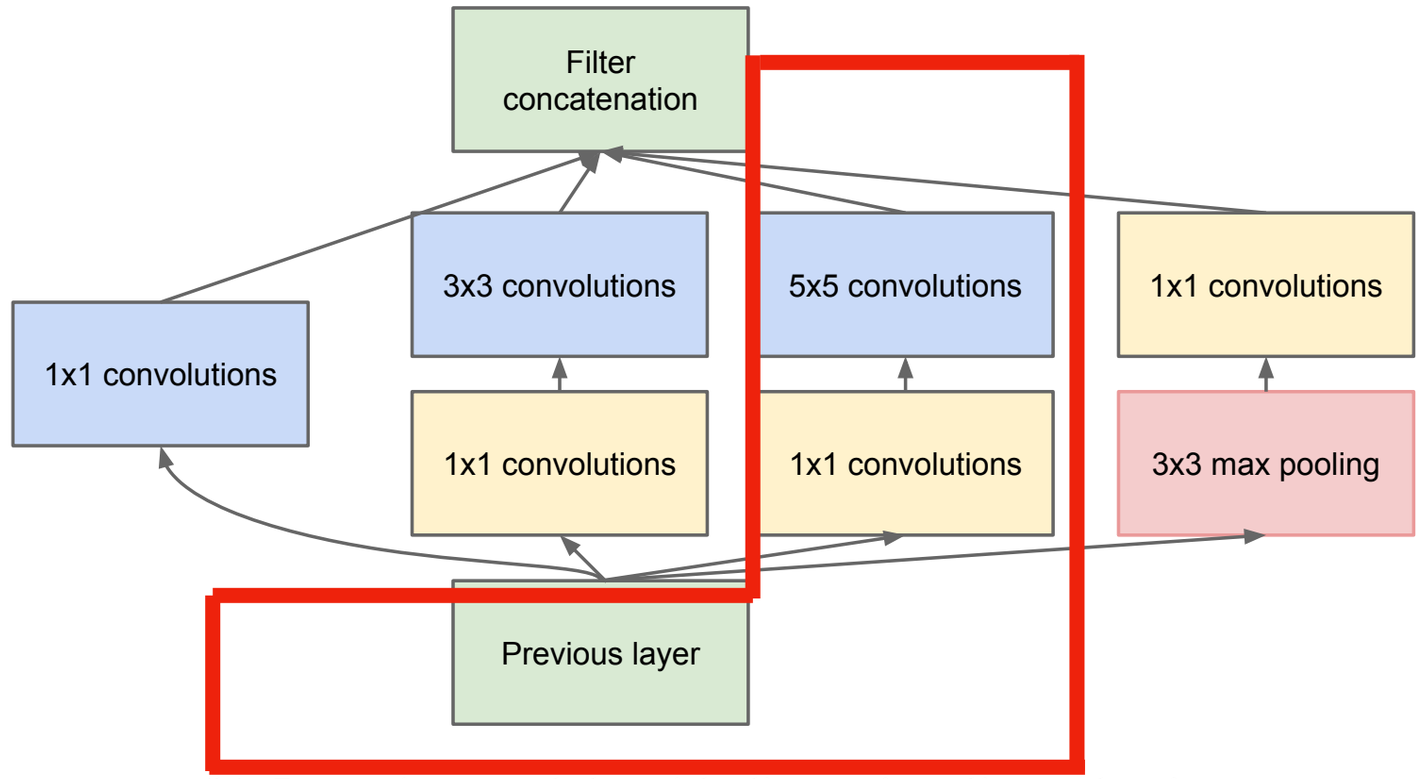
Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9. 2015.



(b) Inception module with dimensionality reduction

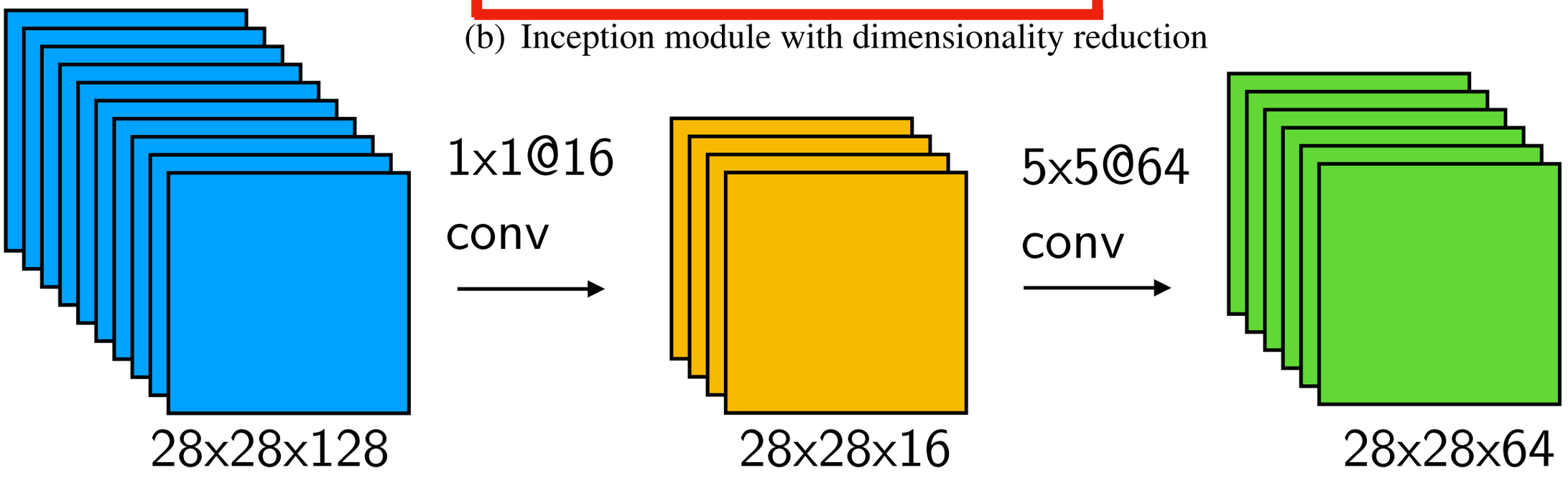
GoogLeNet / Inception v1

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.



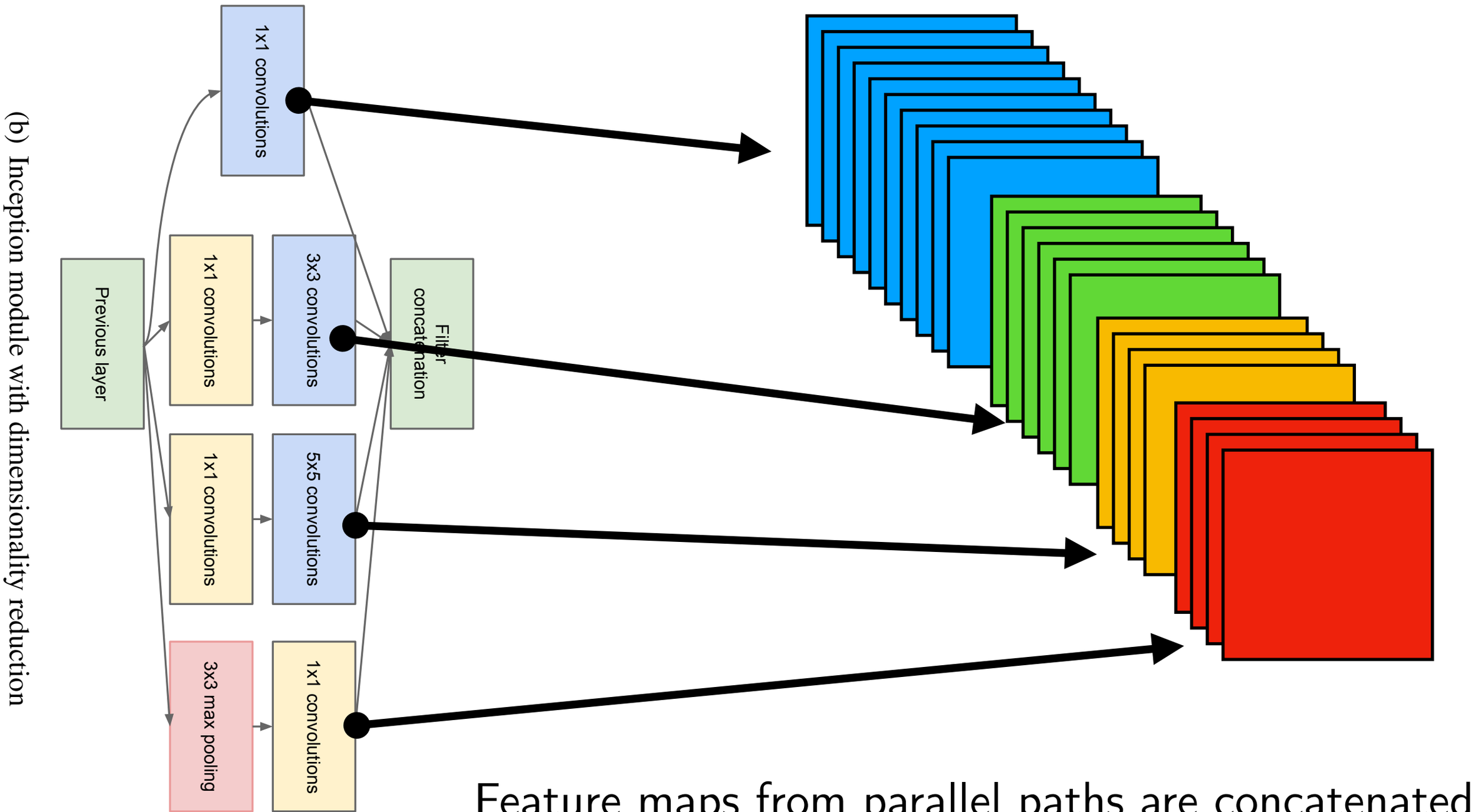
(b) Inception module with dimensionality reduction

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.



Inception v3

Lecture Overview

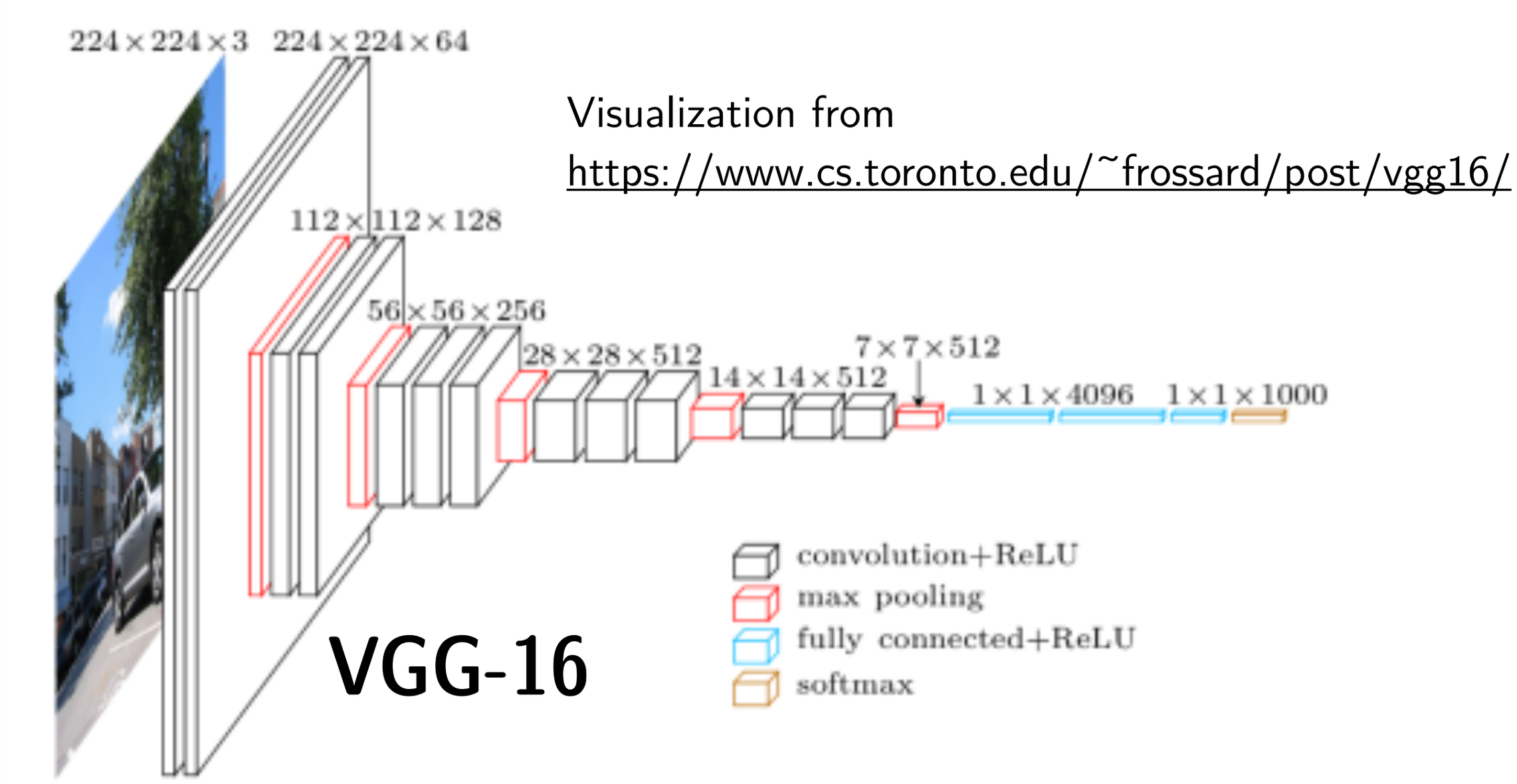
1. Padding (control output size in addition to stride)
2. Spatial Dropout and BatchNorm
3. Considerations for CNNs on GPUs
4. Common Architectures
    A. VGG16 (simple, deep CNN)
    B. ResNet and skip connections
    C. Fully convolutional networks (no fully connected layers)
    D. Inception (parallel 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., pretrained on ImageNet)
  - ✦ Freeze the weights: Only train last layer (or last few layers)
- Related approach: Finetuning, train a pre-trained network on your smaller dataset

Transfer Learning

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

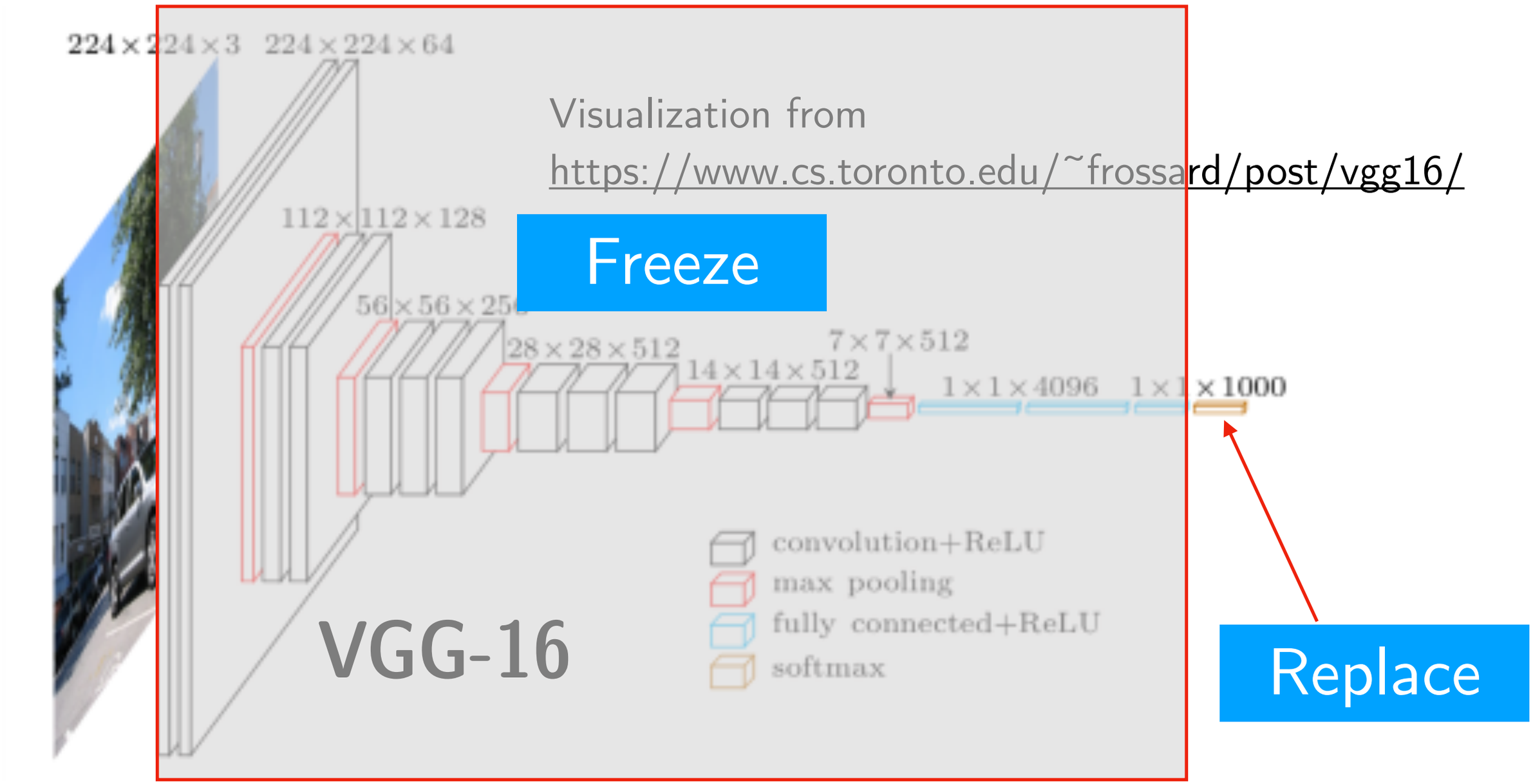


Visualization from
https://www.cs.toronto.edu/~frossard/post/vgg16/

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

Docs > torchvision > torchvision.models

TORCHVISION.MODELS

The models subpackage contains definitions for the following model architectures:

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet

You can construct a model with random weights by calling its constructor:

```
import torchvision.models as models
resnet18 = models.resnet18()
alexnet = models.alexnet()
vgg16 = models.vgg16()
squeezenet = models.squeezenet1_0()
densenet = models.densenet161()
inception = models.inception_v3()
googlenet = models.googlenet()
```

Transfer Learning

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

Docs > torchvision > torchvision.models

TORCHVISION.MODELS

The models subpackage contains definitions for the following model architectures:

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet

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

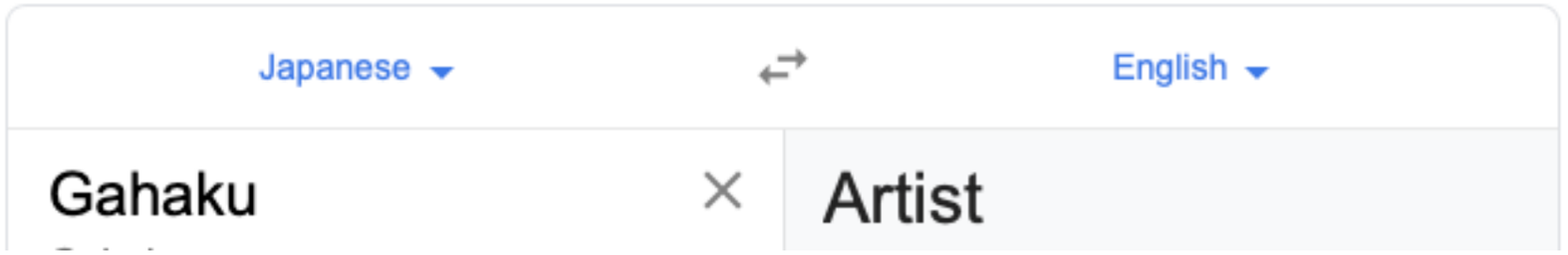
PyTorch implementation: https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/vgg16-transferlearning.ipynb

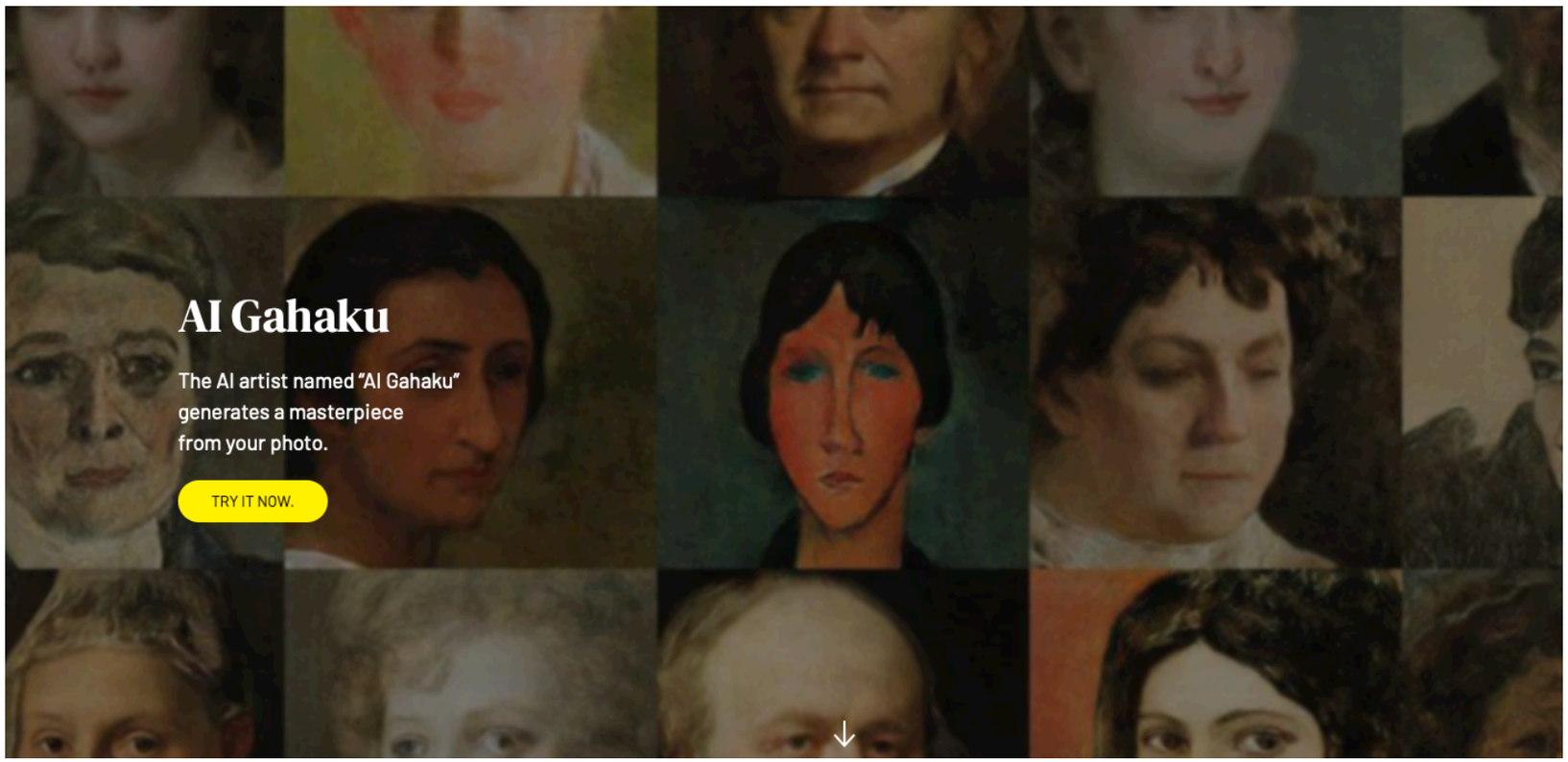
Stuff in the News

Optional Reading Material

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

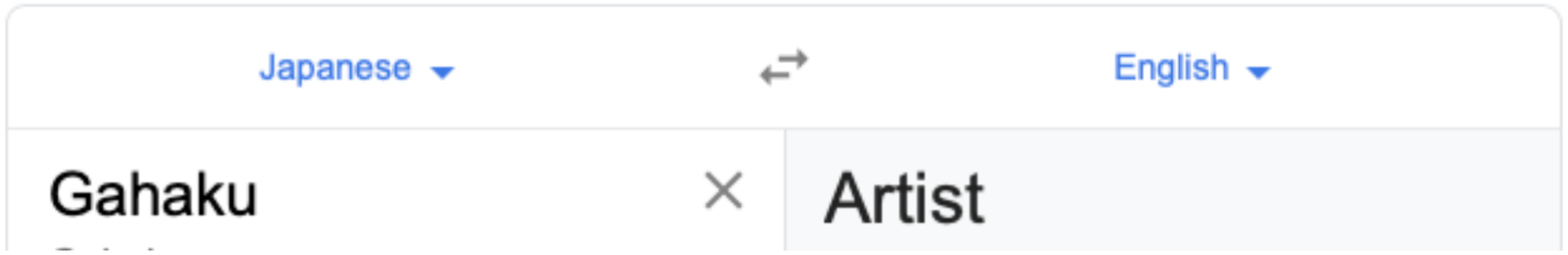
AI Gahaku





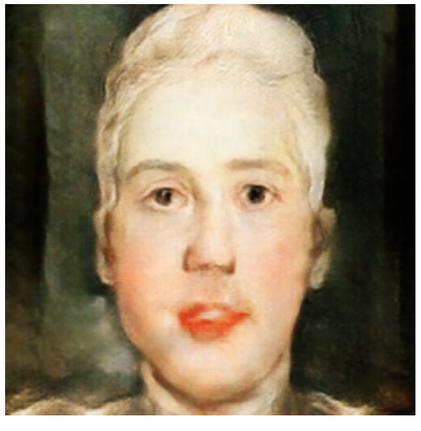
https://ai-art.tokyo/en/

AI Gahaku



The AI artist named "AI Gahaku" generates a masterpiece from your photo.

https://ai-art.tokyo/en/

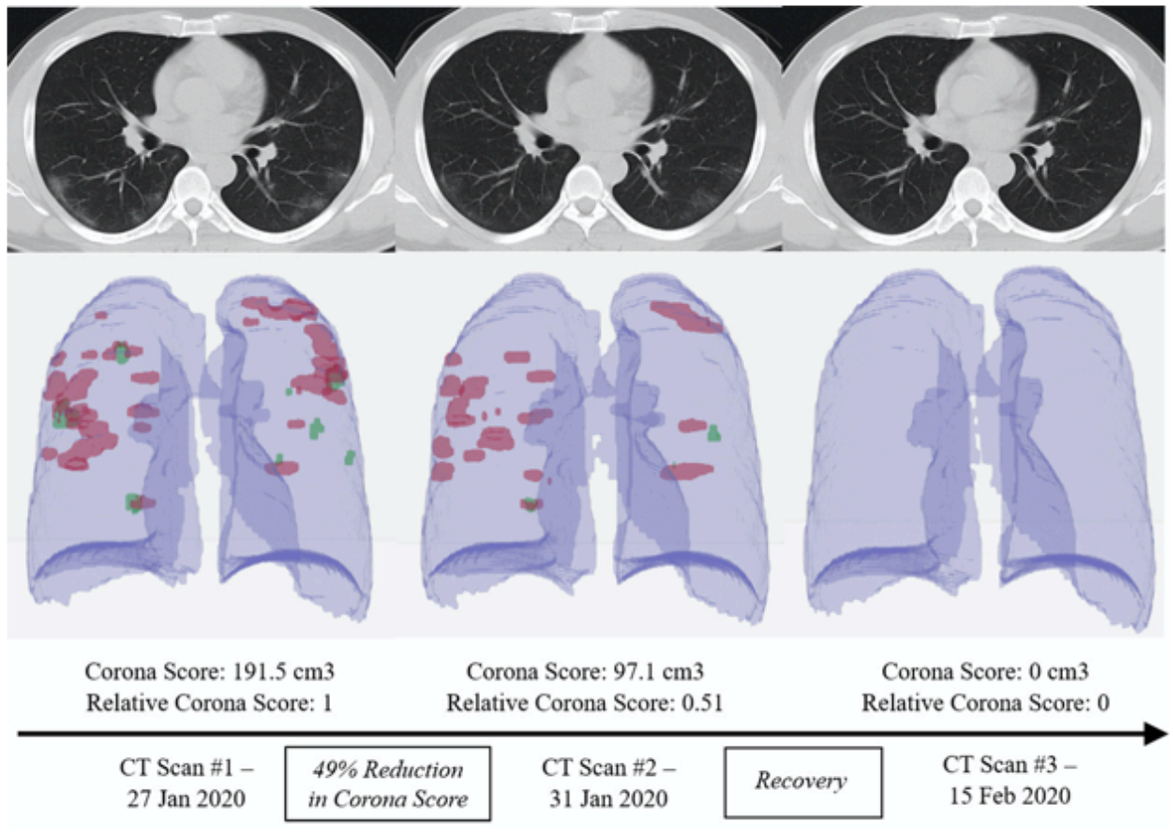


More Coronavirus Datasets

Hospitals Deploy AI Tools to Detect COVID-19 on Chest Scans

Deep learning algorithms can diagnose, triage, and monitor coronavirus cases from lung images. Next, can they predict who will need a ventilator?

By **Megan Scudellari**



https://spectrum.ieee.org/the-human-os/biomedical/imaging/hospitals-deploy-ai-tools-detect-covid19-chest-scans

More Coronavirus Datasets

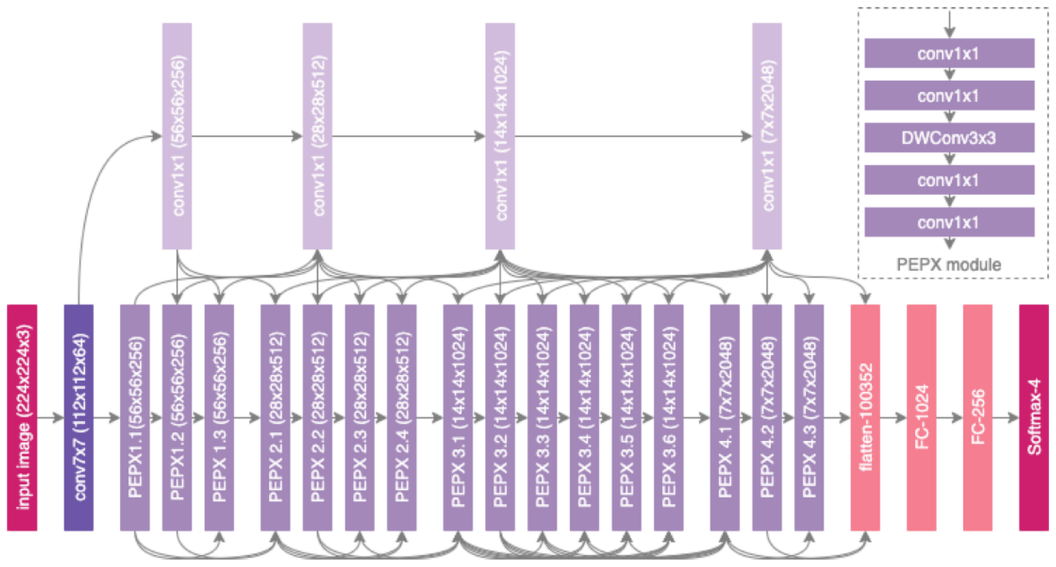
"The **COVIDx** dataset now consists of 16,756 chest X-Rays across 13,645 patients"

https://github.com/lindawangg/COVID-Net

"COVID-Net, a deep convolutional neural network design tailored for the detection of COVID-19 cases from chest radiography images"

Based on a ResNet-like architecture

https://arxiv.org/pdf/2003.09871.pdf



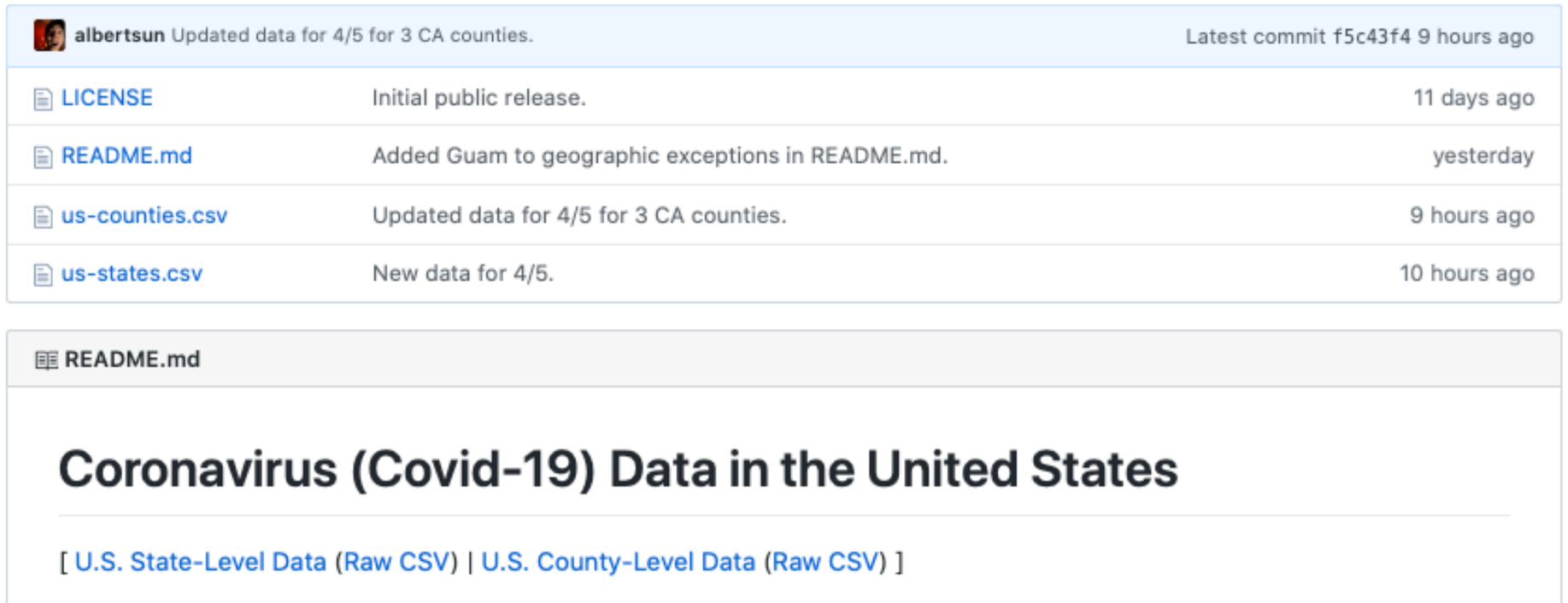
https://medium.com/@sheldon.fernandez/covid-net-larger-dataset-new-models-and-covid-risknet-fd8e7c451c

More Coronavirus Datasets

We're Sharing Coronavirus Case Data for Every U.S. County

https://www.nytimes.com/article/coronavirus-county-data-us.html

Note: you need to open this article in a private browser window to be able to view it.



https://github.com/nytimes/covid-19-data

500'000€ Prize for Compressing Human Knowledge

(widely known as the Hutter Prize)

Compress the 1GB file `enwik9` to less than the current record of about 116MB

News: *The H-Prize went BIG! Why?*
10×*Size*, 10×*Purse*, 10×*Time*, 10×*RAM*, 10×*HDD*.

Who will get here first?

Being able to compress well is closely related to intelligence as explained below. While intelligence is a slippery concept, file sizes are hard numbers. Wikipedia is an extensive snapshot of Human Knowledge. If you can compress the first 1GB of Wikipedia better than your predecessors, your (de)compressor likely has to be smart(er). The intention of this prize is to encourage development of intelligent compressors/programs as a path to AGI.

Losslessly compress the 1GB file `enwik9` to less than 116MB. More precisely:

- Create a Linux or Windows compressor `comp.exe` of size *S1* that compresses `enwik9` to `archive.exe` of size *S2* such that *S:=S1+S2* < *L* := 116'673'681 = previous record.
- If run, `archive.exe` produces (without input from other sources) a 10^9 byte file that is identical to `enwik9`.
- If we can verify your claim, you are eligible for a prize of 500'000€×(1-*S*/*L*). Minimum claim is 5'000€ (1% improvement).
- Restrictions: Must run in ≲100 hours using a single CPU core and <10GB RAM and <100GB HDD on our test machine.

http://prize.hutter1.net