Computer Vision

Image classification II

Tae-Hyun Oh (오태현)
전자전기공학과
POSTECH

Slide by Sungbin Kim (김성빈)
TAs: {Dongmin Choi , Jongha Kim, Juyong Lee, Sungbin Kim} (in alphabetic order)

boostcamp AI Tech

1. Problems with deeper layers

1.1 Going deeper with convolutions

1.2 Hard to optimize

2. CNN architectures for image classification 2

2.1 GoogLeNet

2.2 ResNet

2.3 Beyond ResNet

3. Summary of image classification

3.1 Summary of image classification

3.2 CNN backbones

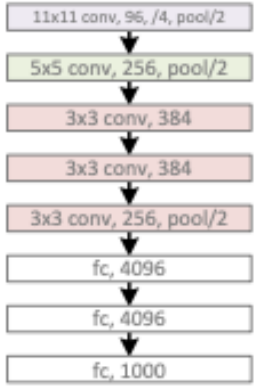
1.

Problems with deeper layers

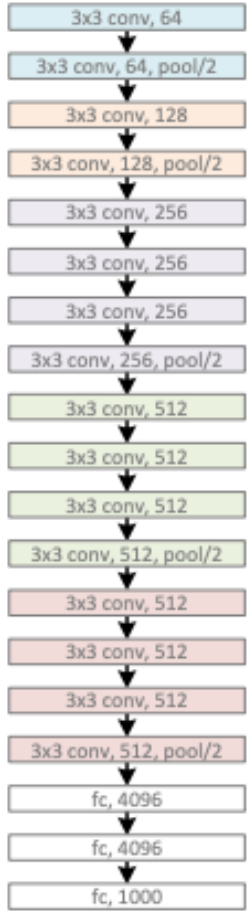
1.1 Going deeper with convolutions

The neural network is getting deeper and wider

- Deeper networks learn more powerful features, because of
  - Larger receptive fields
  - More capacity and non-linearity



Alexnet (8 layers)



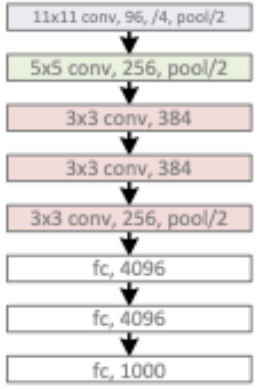
VGGNet (19 layers)

[Deep Residual Learning MSRA @ ILSVRC & COCO 2015 competitions, Kaiming He, MSRA]

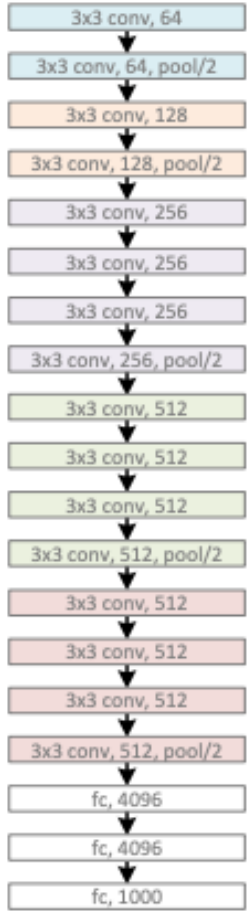
1.1 Going deeper with convolutions

The neural network is getting deeper and wider

- Deeper networks learn more powerful features, because of
  - Larger receptive fields
  - More capacity and non-linearity
- But, getting deeper and deeper always works better?



Alexnet (8 layers)



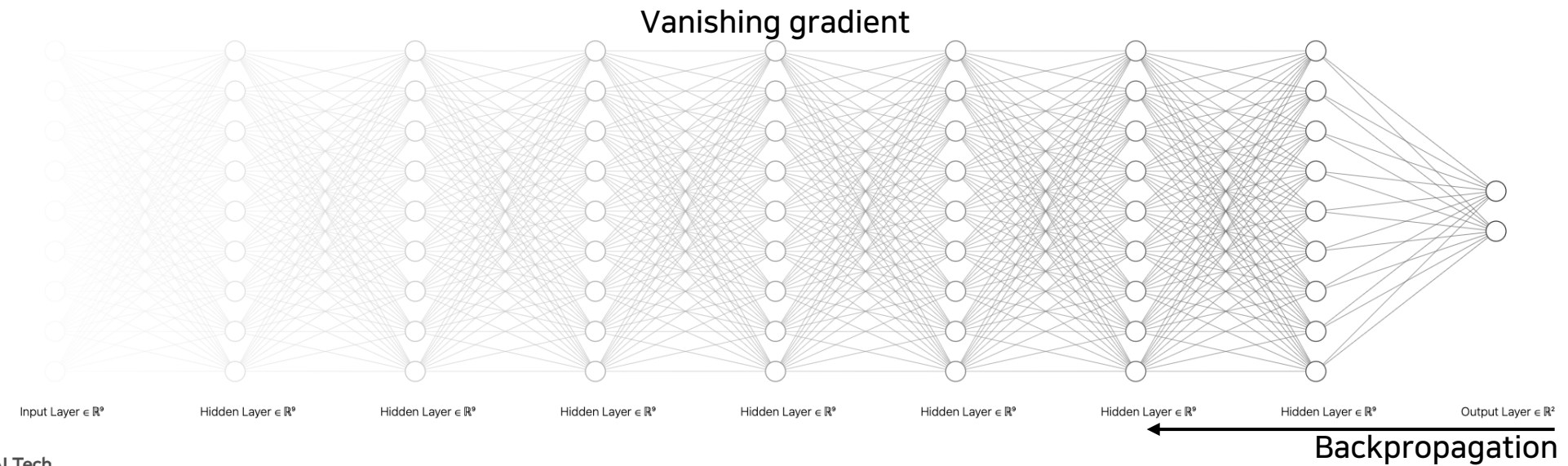
VGGNet (19 layers)

[Deep Residual Learning MSRA @ ILSVRC & COCO 2015 competitions, Kaiming He, MSRA]

1.2 Hard to optimize

Deeper networks are harder to optimize

- Gradient vanishing / exploding

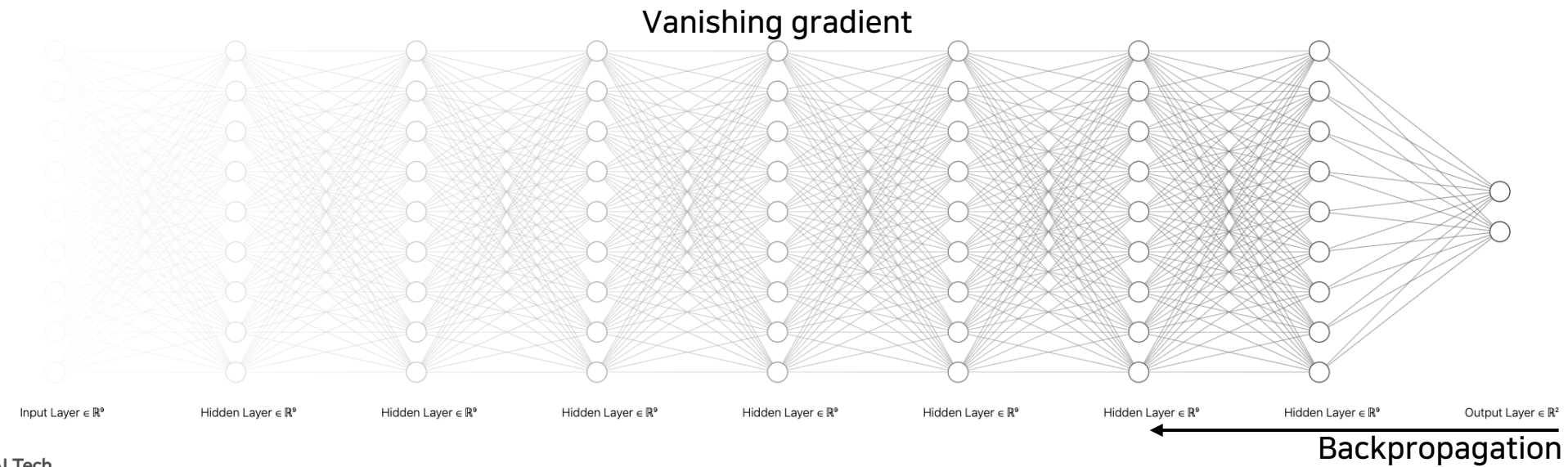


[http://alexlenail.me/NN-SVG/index.html]

1.2 Hard to optimize

Deeper networks are harder to optimize

- Gradient vanishing / exploding
- Computationally complex

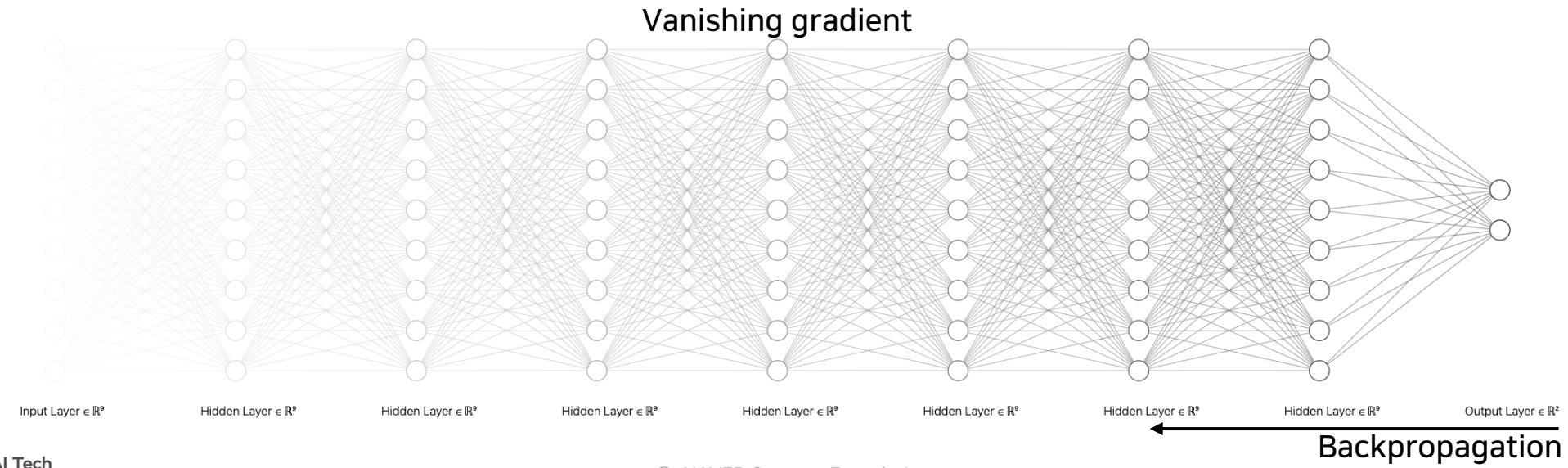


[http://alexlenail.me/NN-SVG/index.html]

1.2 Hard to optimize

Deeper networks are harder to optimize

- Gradient vanishing / exploding
- Computationally complex
- ~~Overfitting problem~~ Degradation problem



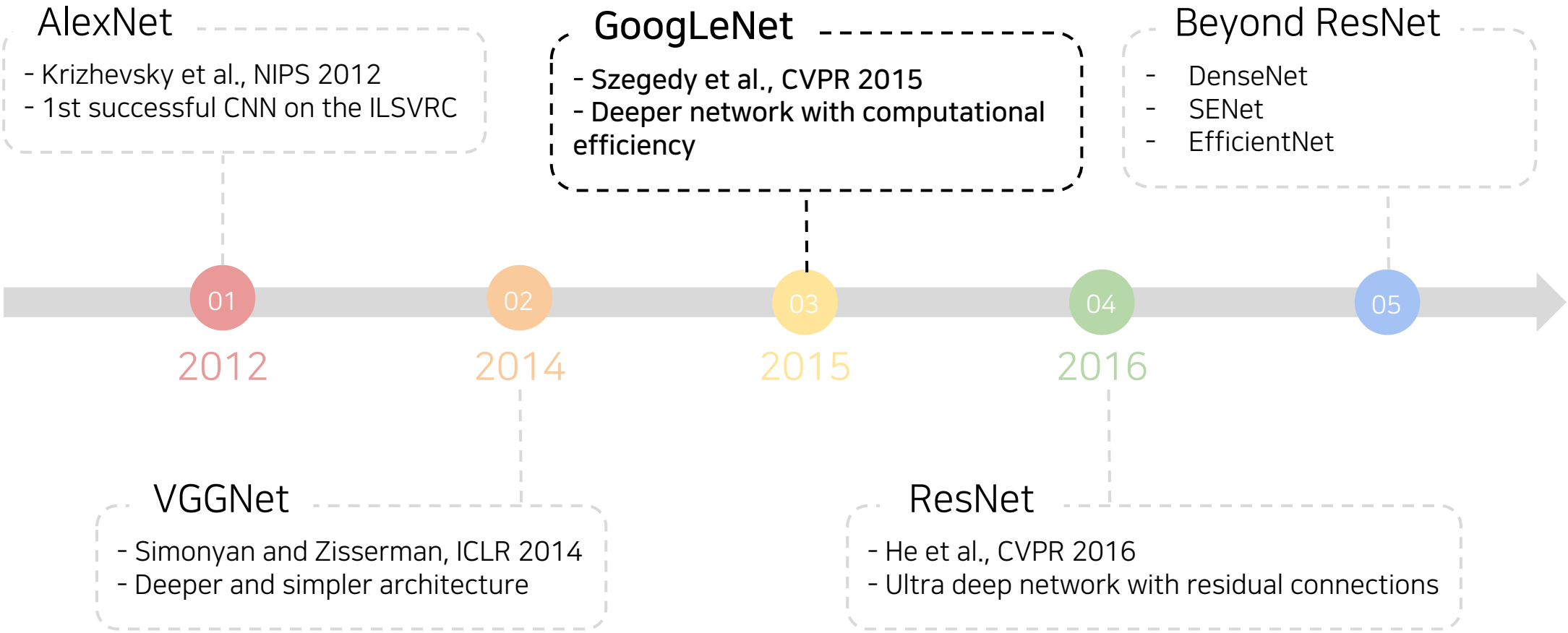
[http://alexlenail.me/NN-SVG/index.html]

2.

CNN architectures for image classification 2

2.1 GoogLeNet

[Szegedy et al., CVPR 2015]

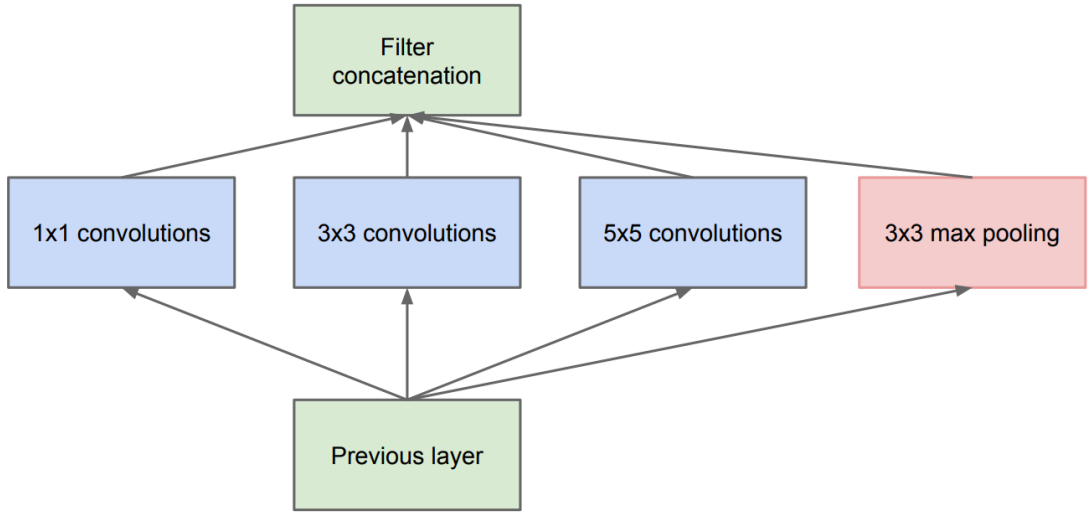


2.1 GoogLeNet

Inception module

[Szegedy et al., CVPR 2015]

- Apply multiple filter operations on input activation from the previous layer:
  - 1x1, 3x3, 5x5 convolution filters
  - 3x3 pooling operation
- Concatenate all filter outputs together along the channel axis



Naïve version

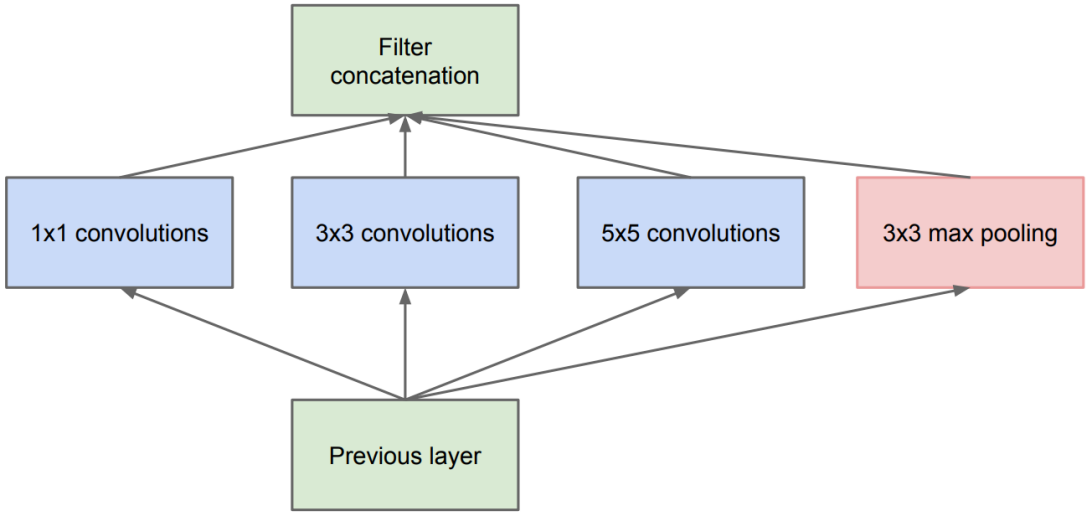
×

2.1 GoogLeNet

Inception module

[Szegedy et al., CVPR 2015]

- The increased network size increases the use of computational resources
- Use 1x1 convolutions!



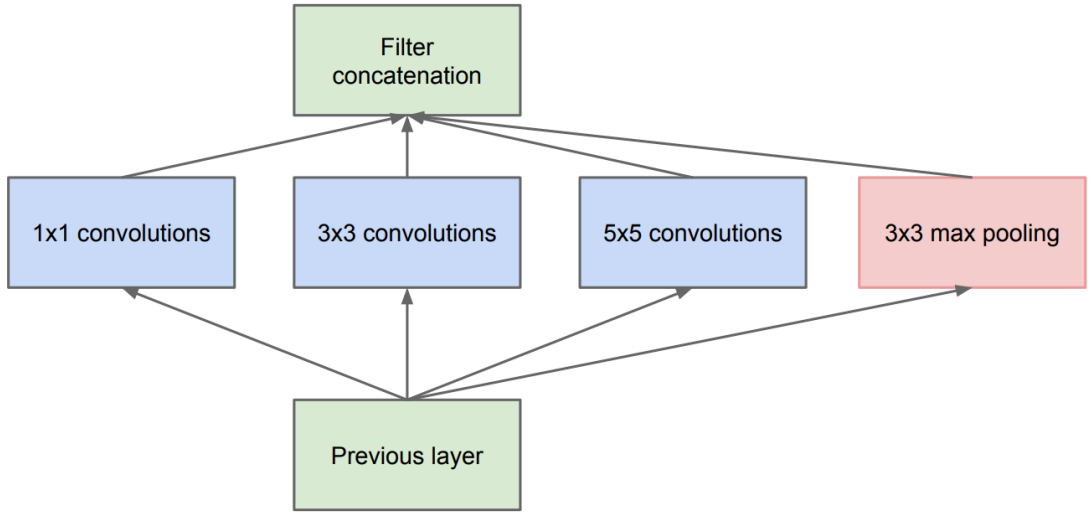
Naïve version

2.1 GoogLeNet

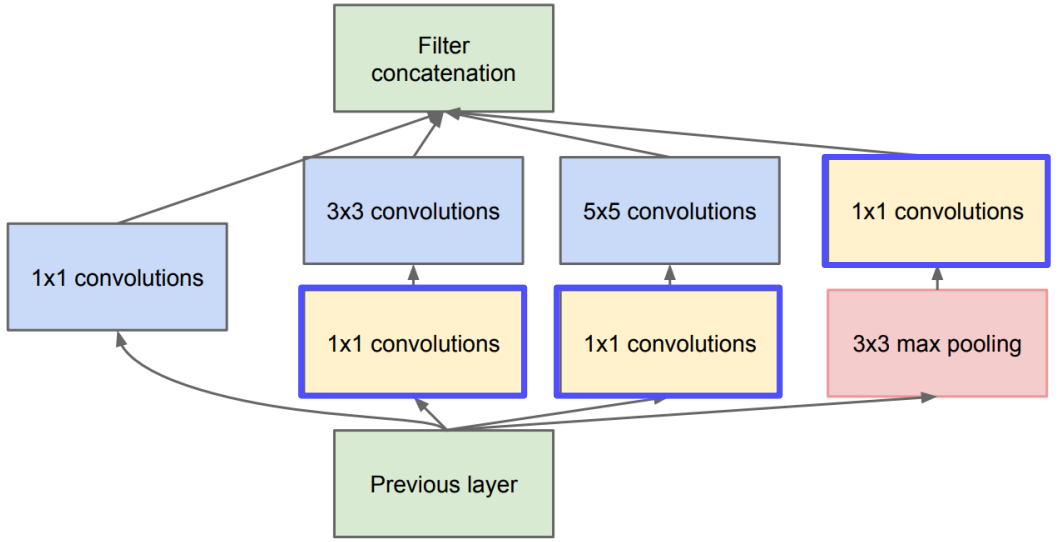
Inception module

[Szegedy et al., CVPR 2015]

- Use 1x1 convolutions as "bottleneck" layers that reduce the number of channels



Naïve version

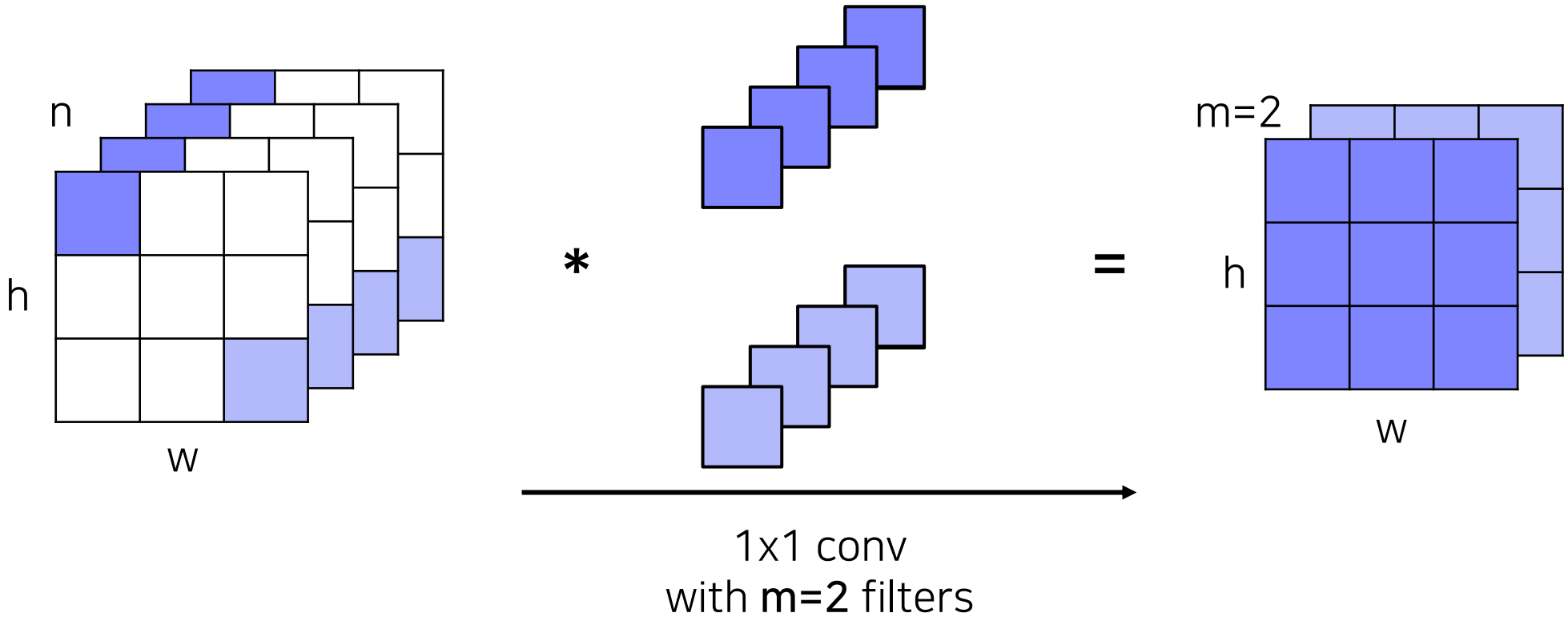


Dimension reduced version

2.1 GoogLeNet

1x1 convolutions

[Szegedy et al., CVPR 2015]

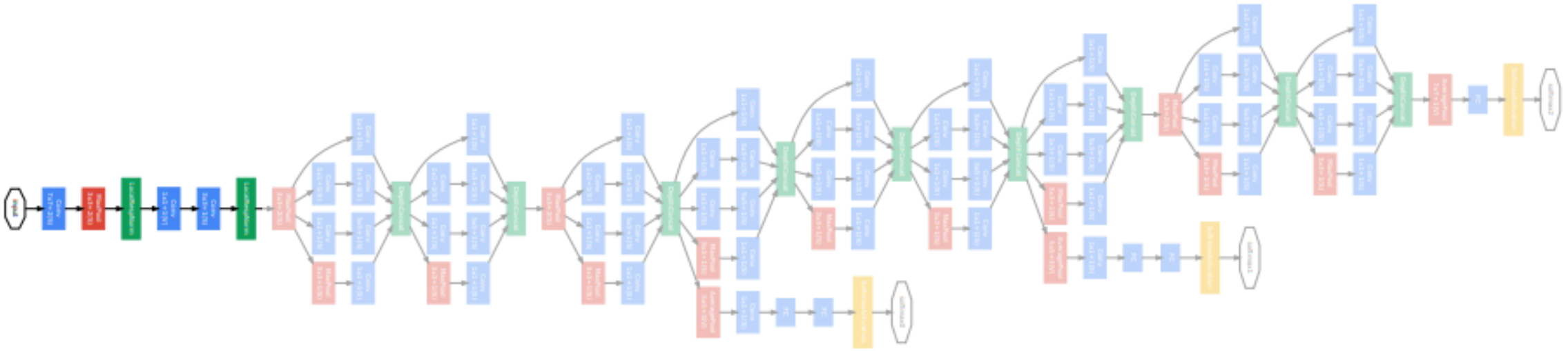


2.1 GoogLeNet

Overall architecture

[Szegedy et al., CVPR 2015]

- Stem network: vanilla convolution networks

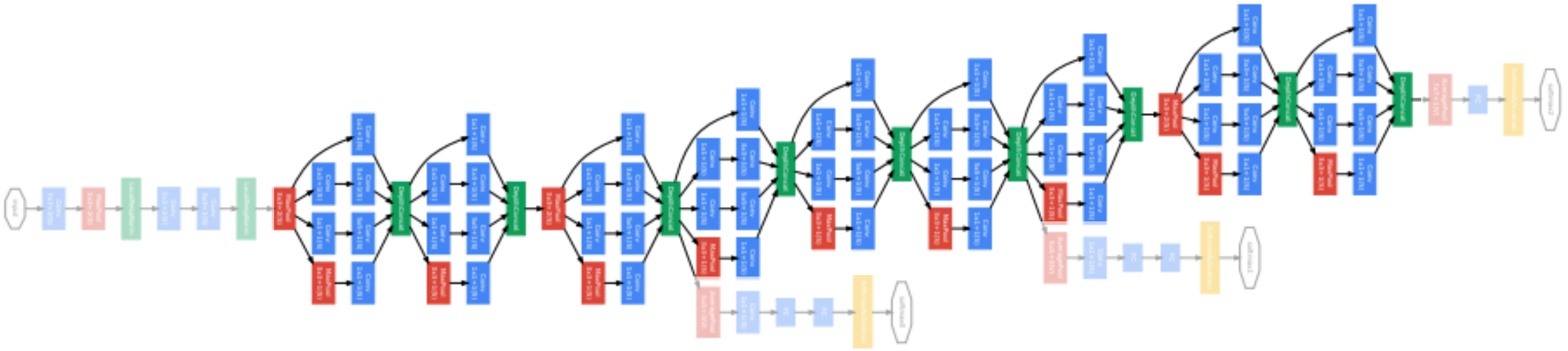


2.1 GoogLeNet

Overall architecture

[Szegedy et al., CVPR 2015]

- Stem network: vanilla convolution networks
- Stacked inception modules

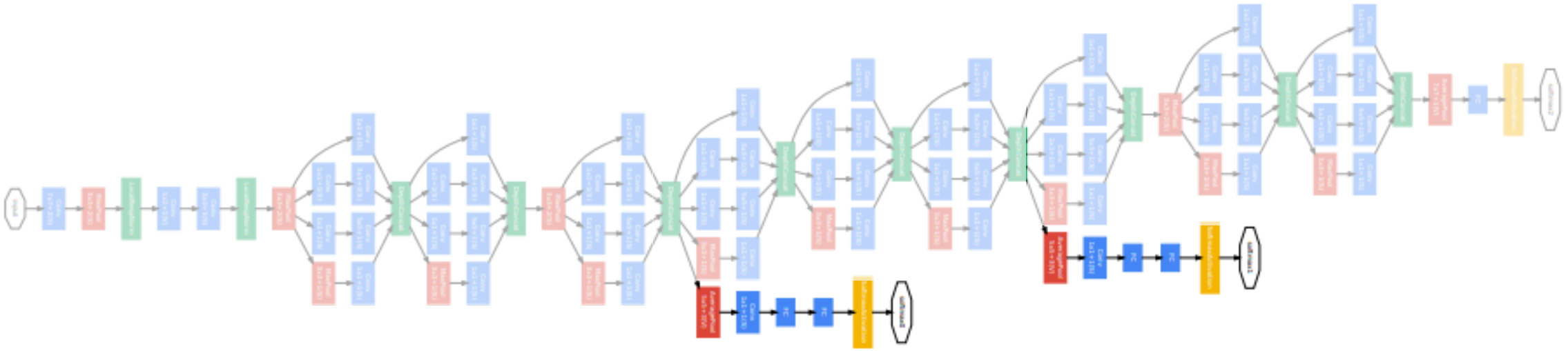


2.1 GoogLeNet

Overall architecture

[Szegedy et al., CVPR 2015]

- Stem network: vanilla convolution networks
- Stacked inception modules
- Auxiliary classifiers

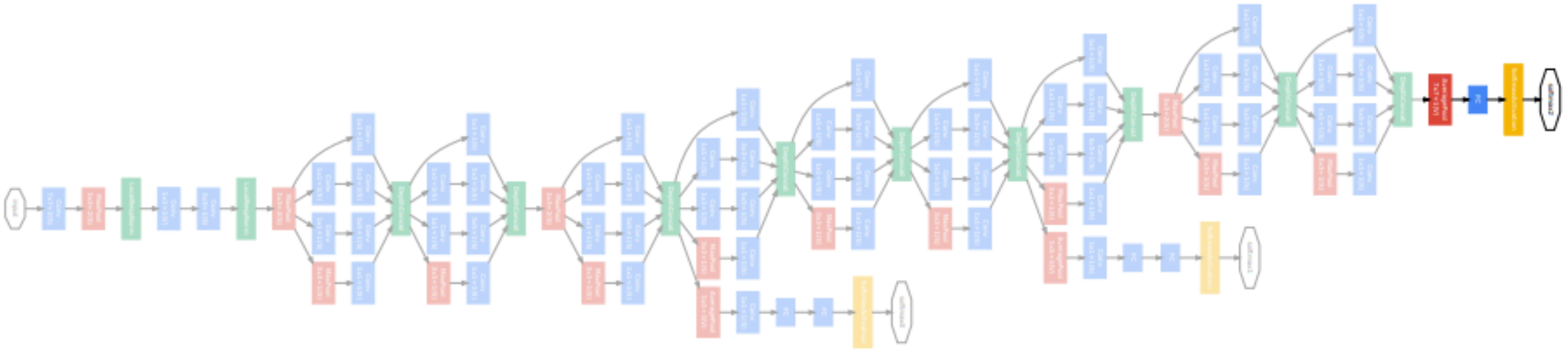


2.1 GoogLeNet

Overall architecture

[Szegedy et al., CVPR 2015]

- Stem network: vanilla convolution networks
- Stacked inception modules
- Auxiliary classifiers
- Classifier ouput (a single FC layer)

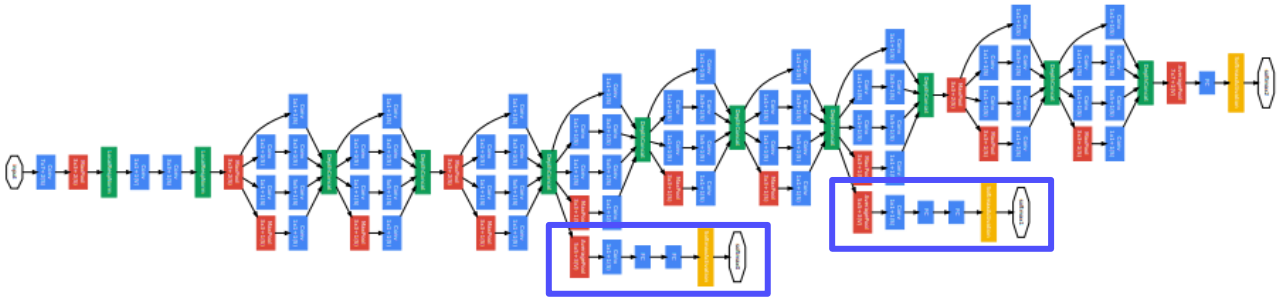


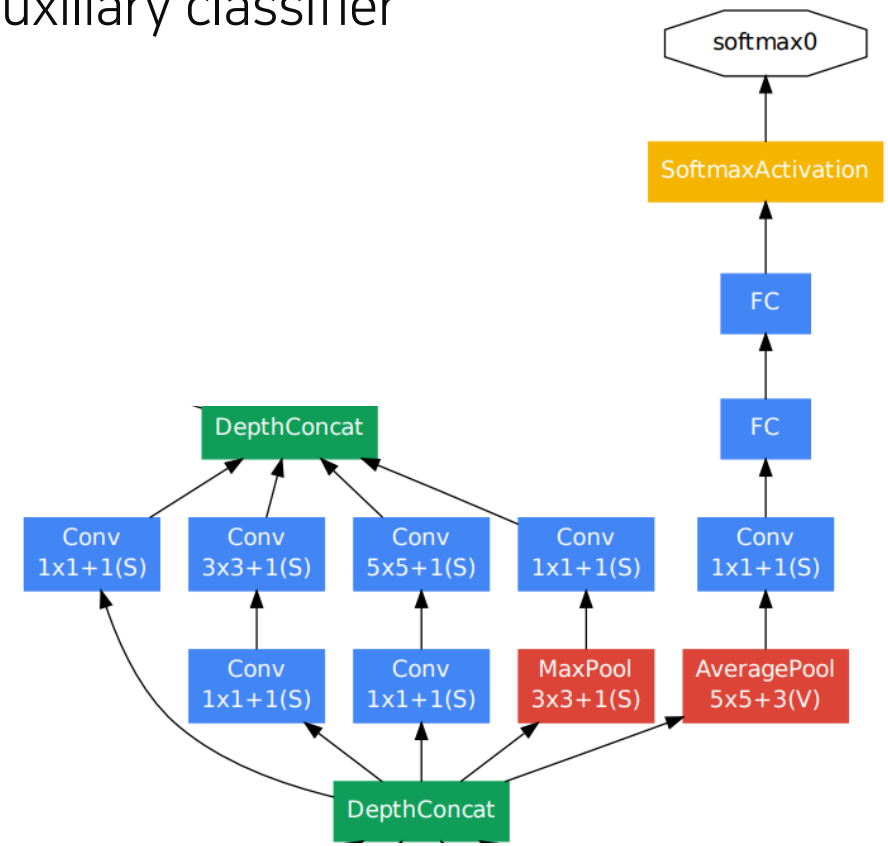
2.1 GoogLeNet

[Szegedy et al., CVPR 2015]

Auxiliary classifier

- The vanishing gradient problem is dealt with by the auxiliary classifier
- Injecting additional gradients into lower layers
- Used only during training, removed at testing time

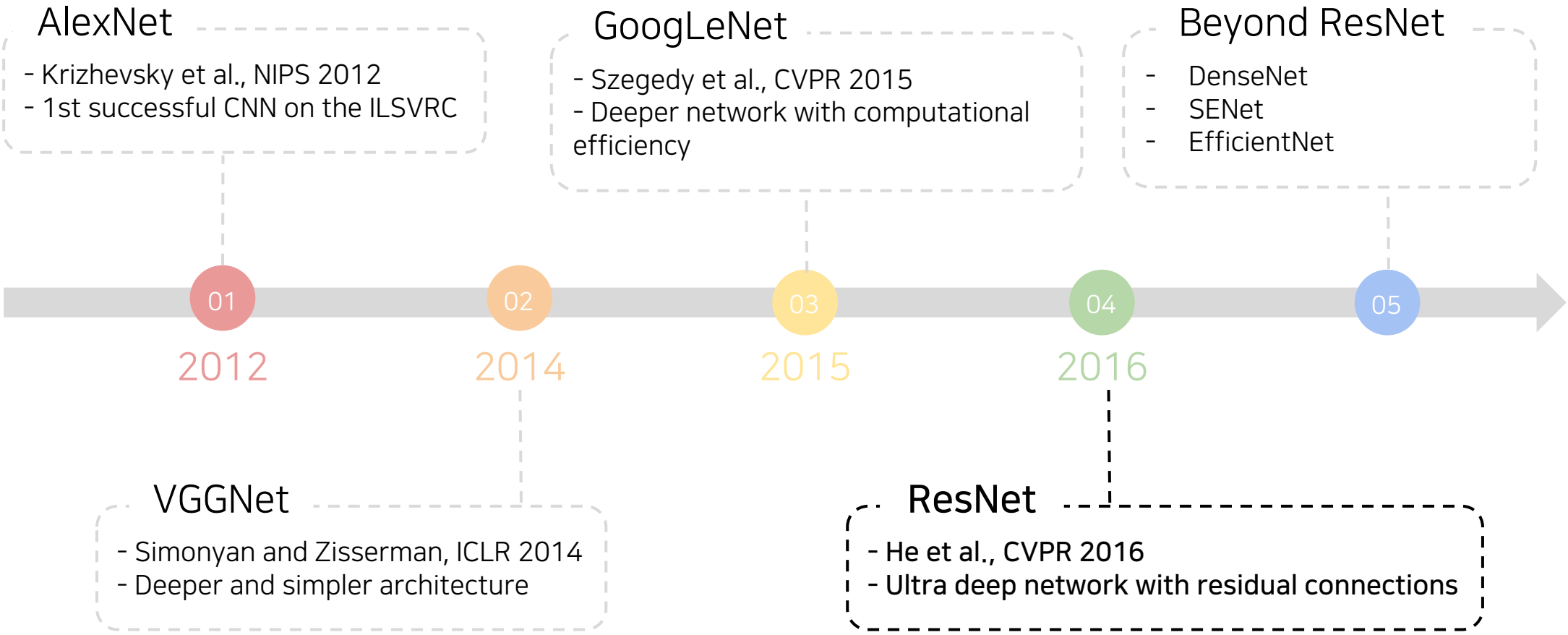




Auxiliary classifier

2.2 ResNet

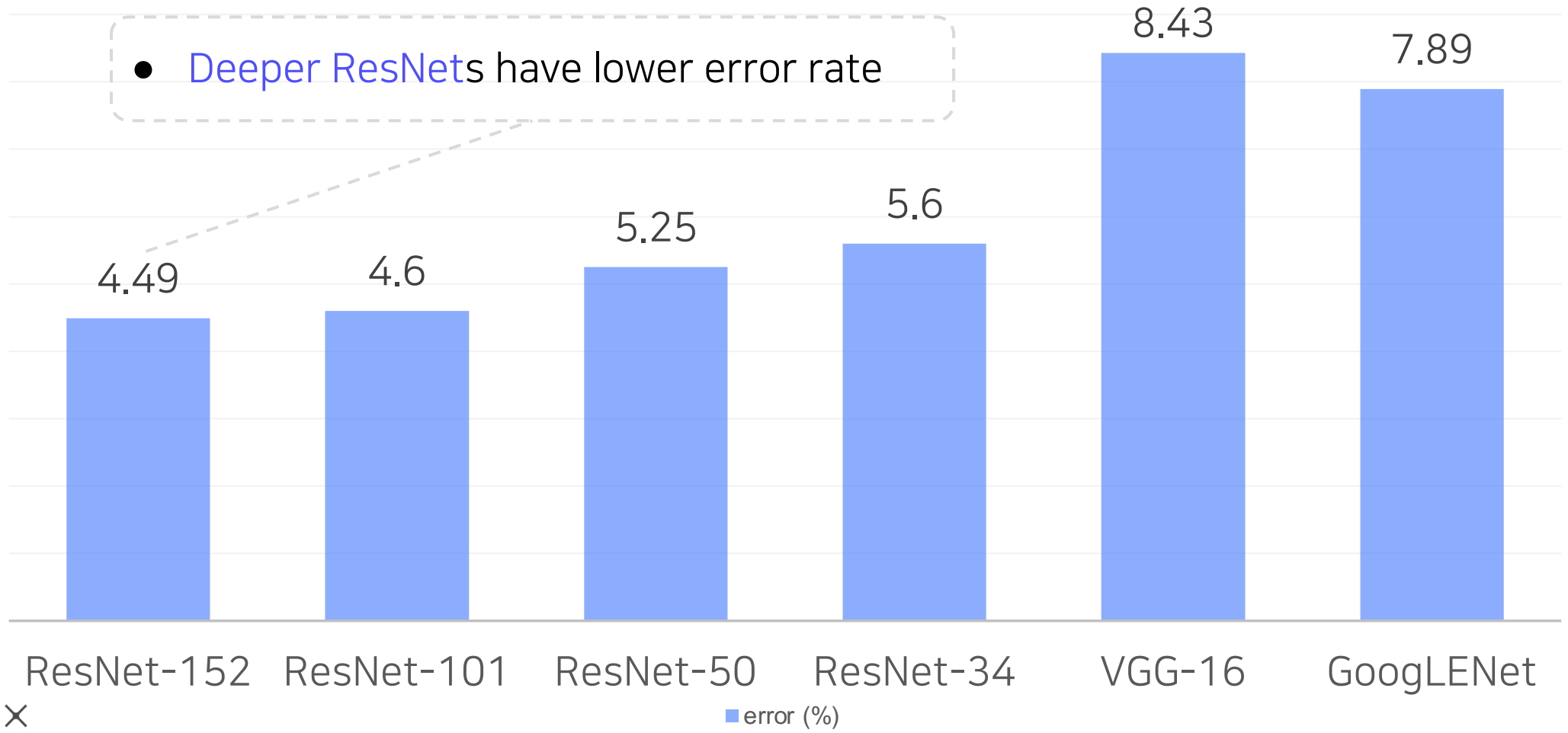
[He et al., CVPR 2015]



2.2 ResNet

Top-5 validation error on the ImageNet

[He et al., CVPR 2016]

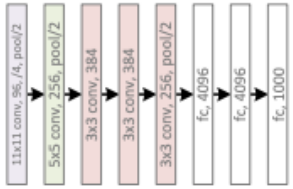


2.2 ResNet

Revolutions of depth

[He et al., CVPR 2016]

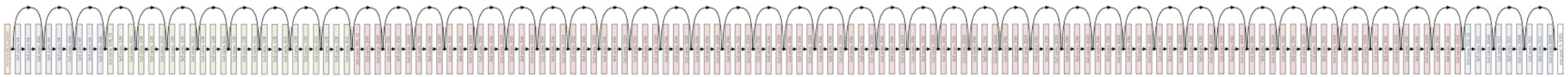
- Building ultra-deeper than any other networks
- What makes it hard to build a very deep architecture?



AlexNet (8 layers)

VGGNet (19 layers)

ResNet (152 layers)

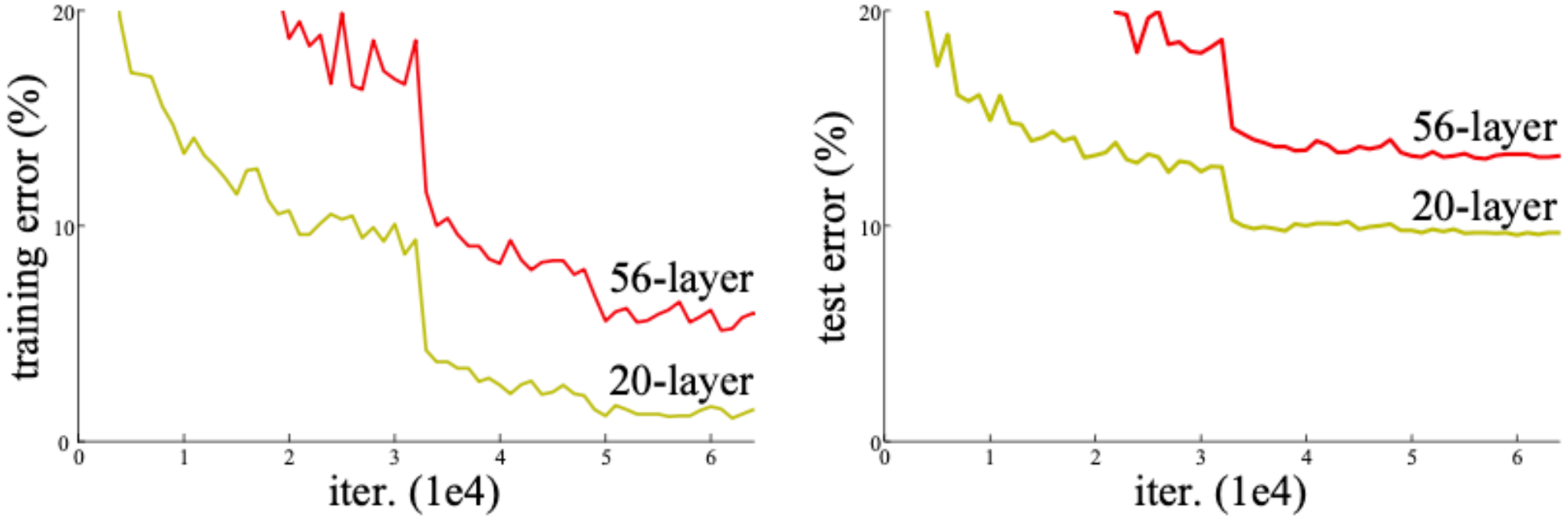


2.2 ResNet

Degradation problem

[He et al., CVPR 2016]

- As the network depth increases, accuracy gets saturated ⇒ **degrade rapidly**

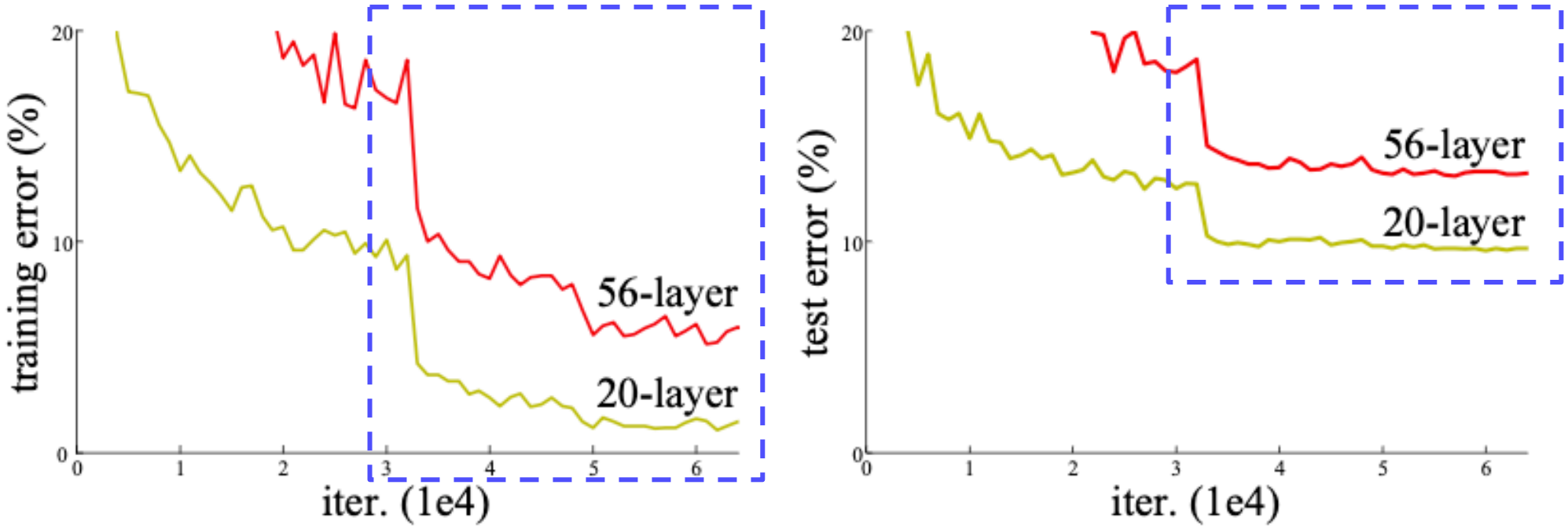


2.2 ResNet

Degradation problem

[He et al., CVPR 2016]

- As the network depth increases, accuracy gets saturated ⇒ **degrade rapidly**
- This is not caused by overfitting. The problem is **optimization**!

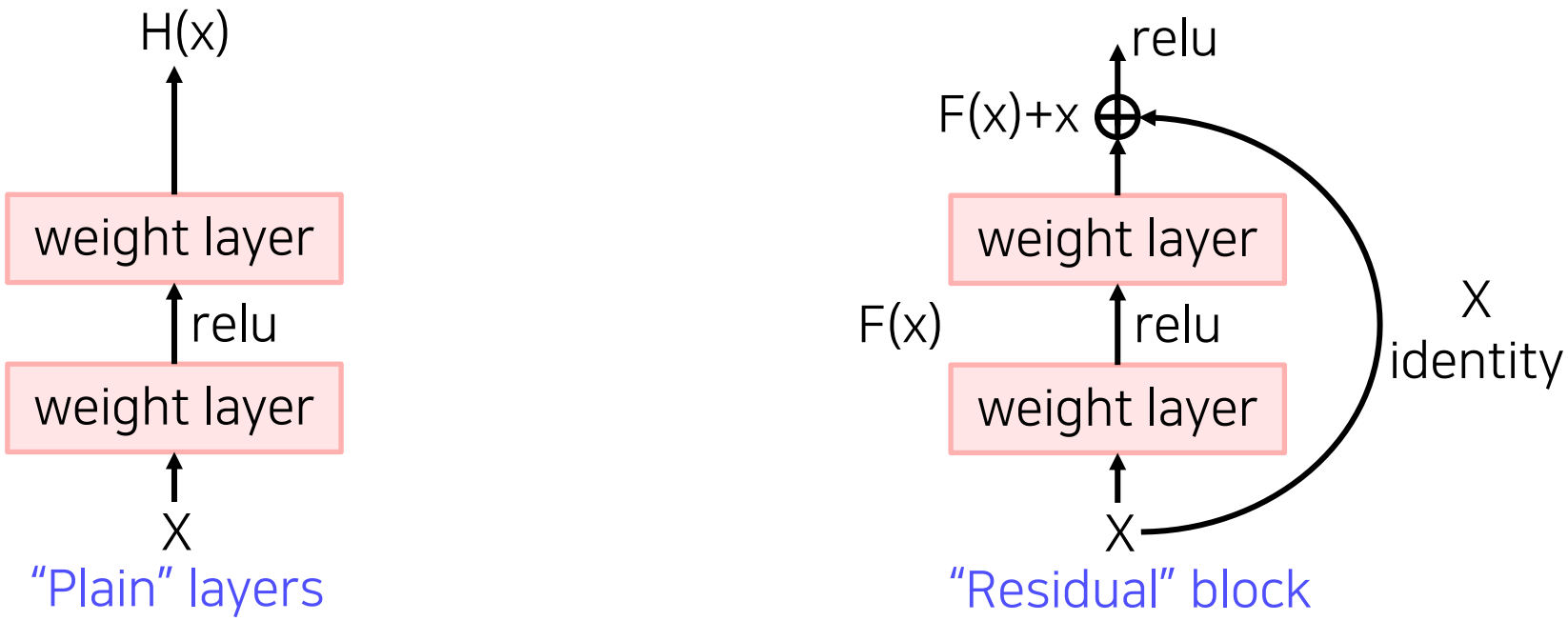


2.2 ResNet

Hypothesis

[He et al., CVPR 2016]

- Plain layer: As the layers get deeper, **it is hard to learn good $\boldsymbol{H(x)}$ directly**

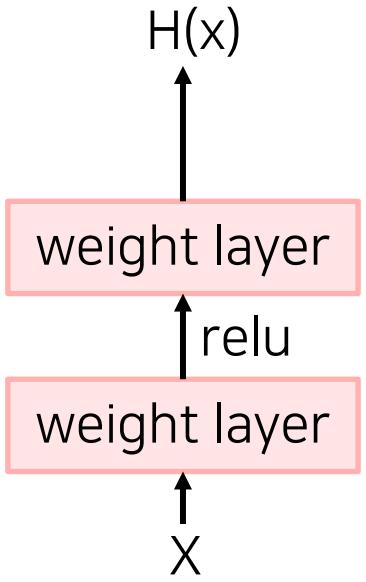


2.2 ResNet

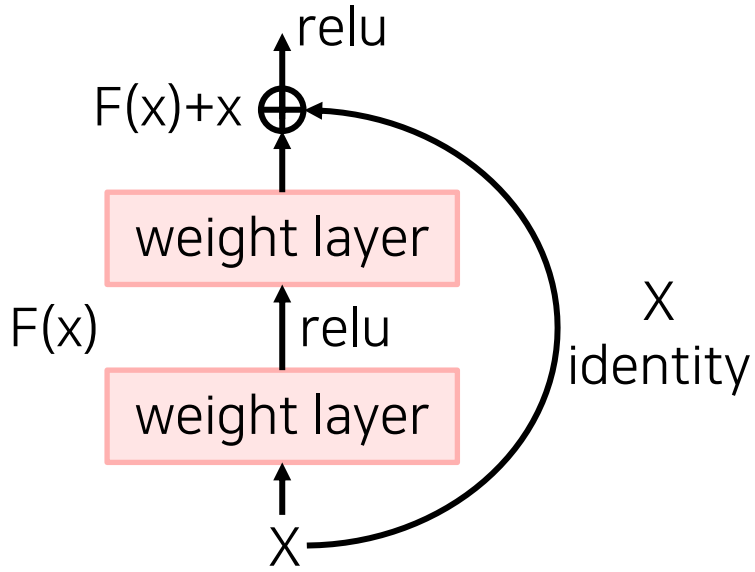
[He et al., CVPR 2016]

Hypothesis

- Plain layer: As the layers get deeper, **it is hard to learn good $\boldsymbol{H(x)}$ directly**
- Residual block: Instead, we learn residual
  - Target function : $H(x) = F(x) + x$
  - Residual function : $F(x) = H(x) - x$



"Plain" layers



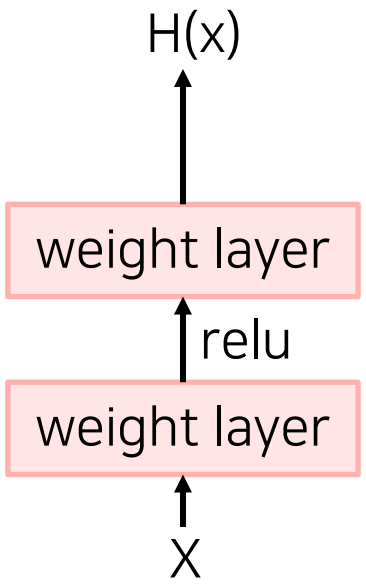
"Residual" block

2.2 ResNet

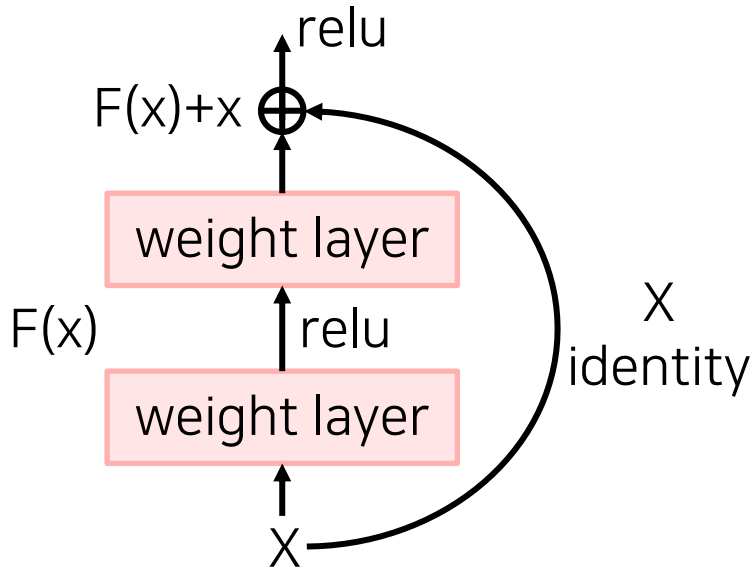
A solution: Shortcut connection

[He et al., CVPR 2016]

- Use layers to fit a residual mapping instead of directly fitting a desired underlying mapping
- The vanishing gradient problem is solved by shortcut connection
- Don't just stack layers up, instead **use shortcut connection**!



"Plain" layers



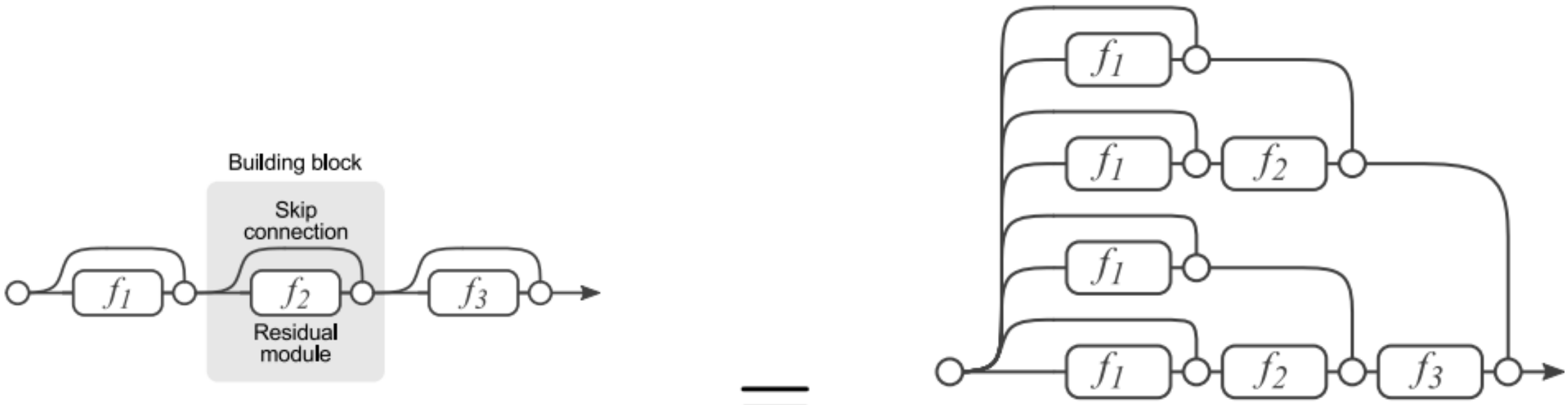
"Residual" block

2.2 ResNet

Analysis of residual connection

[Veit et al., NIPS 2016]

- During training, gradients are mainly from relatively shorter paths
- Residual networks have $O(2^n)$ implicit paths connecting input and output, and adding a block doubles the number of paths



(a) Conventional 3-block residual network

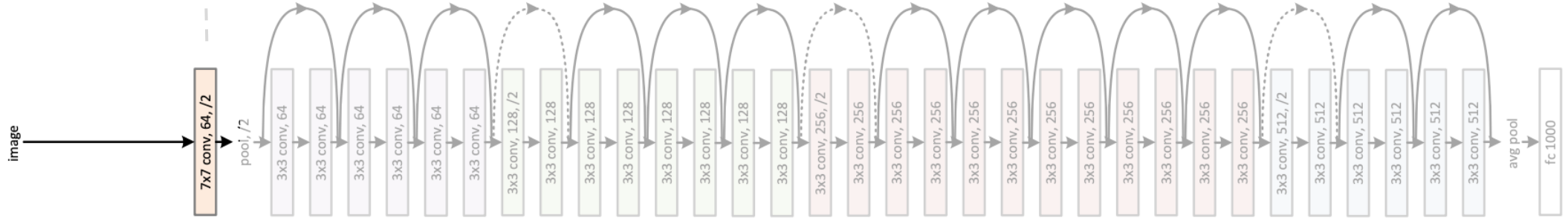
(b) Unraveled view of (a)

2.2 ResNet

Overall architecture

[He et al., CVPR 2016]

- He initialization
- Additional conv layer at the beginning



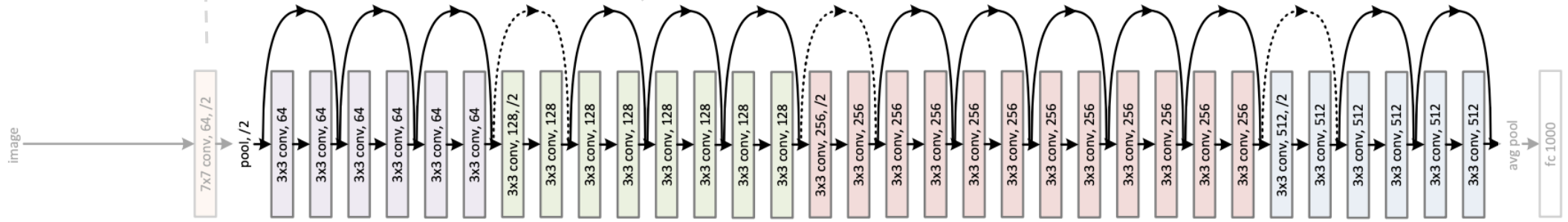
2.2 ResNet

Overall architecture

[He et al., CVPR 2016]

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Batch norm after every conv layer

- He initialization
- Additional conv layer at the beginning



2.2 ResNet

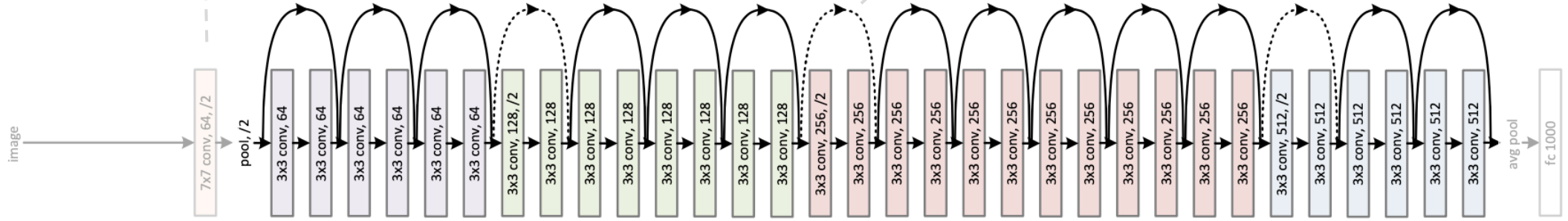
Overall architecture

[He et al., CVPR 2016]

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Batch norm after every conv layer

- Doubling the number of filters and spatially down-sampling by stride 2 instead of spatial pooling

- He initialization
- Additional conv layer at the beginning



2.2 ResNet

Overall architecture

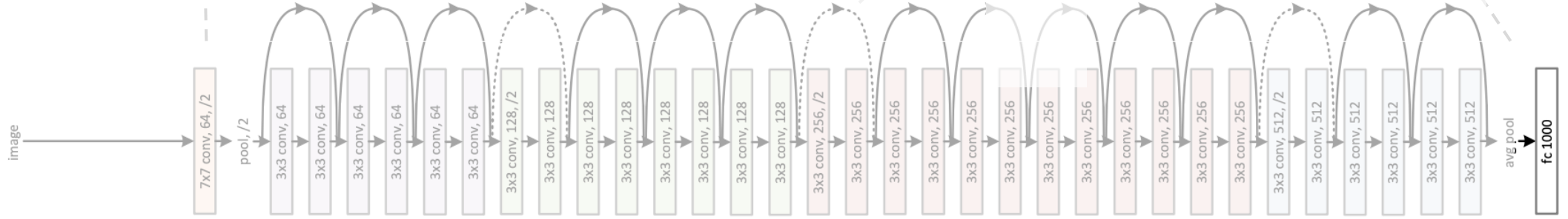
[He et al., CVPR 2016]

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Batch norm after every conv layer

- Doubling the number of filters and spatially down-sampling by stride 2 instead of spatial pooling

- Only a single FC layer for output classes

- He initialization
- Additional conv layer at the beginning



2.2 ResNet

PyTorch code for ResNet

[He et al., CVPR 2016]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

```
return _resnet('resnet18', BasicBlock, [2, 2, 2, 2], pretrained, progress,**kwargs)
```

2.2 ResNet

PyTorch code for ResNet

[He et al., CVPR 2016]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

```
return _resnet('resnet18', BasicBlock, [2, 2, 2, 2], pretrained, progress,**kwargs)
```

2.2 ResNet

PyTorch code for ResNet

[He et al., CVPR 2016]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2				
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2				
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2				
	1×1					
FLOPs		1.8×10^9				

```
def _forward_impl(self, x: Tensor):
    # See note [TorchScript super()]
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.maxpool(x)

    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    x = self.layer4(x)

    x = self.avgpool(x)
    x = torch.flatten(x, 1)
    x = self.fc(x)

    return x
```

2.2 ResNet

PyTorch code for ResNet

[He et al., CVPR 2016]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2				
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2				
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2				
	1×1					
FLOPs		1.8×10^9				

```
def _forward_impl(self, x: Tensor):
    # See note [TorchScript super()]
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.maxpool(x)

    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    x = self.layer4(x)

    x = self.avgpool(x)
    x = torch.flatten(x, 1)
    x = self.fc(x)

    return x
```

```
self.layer1 = self._make_layer(block=2, 64, layers[0])

self.layer2 = self._make_layer(block=2, 128, layers[1], stride=2,
                               dilate=replace_stride_with_dilation[0])

self.layer3 = self._make_layer(block=2, 256, layers[2], stride=2,
                               dilate=replace_stride_with_dilation[1])

self.layer4 = self._make_layer(block=2, 512, layers[3], stride=2,
                               dilate=replace_stride_with_dilation[2])
```

2.2 ResNet

PyTorch code for ResNet

[He et al., CVPR 2016]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1					
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

```python
def _make_layer(self, block: Type[Union[BasicBlock, Bottleneck]], planes: int, blocks: int,
                stride: int = 1, dilate: bool = False) -> nn.Sequential:
    norm_layer = self._norm_layer
    downsample = None
    previous_dilation = self.dilation
    if dilate:
        self.dilation *= stride
        stride = 1
    if stride != 1 or self.inplanes != planes * block.expansion:
        downsample = nn.Sequential(
            conv1x1(self.inplanes, planes * block.expansion, stride),
            norm_layer(planes * block.expansion),
        )

    layers = []
    layers.append(block(self.inplanes, planes, stride, downsample, self.groups,
                        self.base_width, previous_dilation, norm_layer))
    self.inplanes = planes * block.expansion
    for _ in range(1, blocks):
        layers.append(block(self.inplanes, planes, groups=self.groups,
                            base_width=self.base_width, dilation=self.dilation,
                            norm_layer=norm_layer))

    return nn.Sequential(*layers)
```

2.2 ResNet

PyTorch code for ResNet

[He et al., CVPR 2016]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

```
self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
self.fc = nn.Linear(512 * block.expansion, num_classes)
```

1

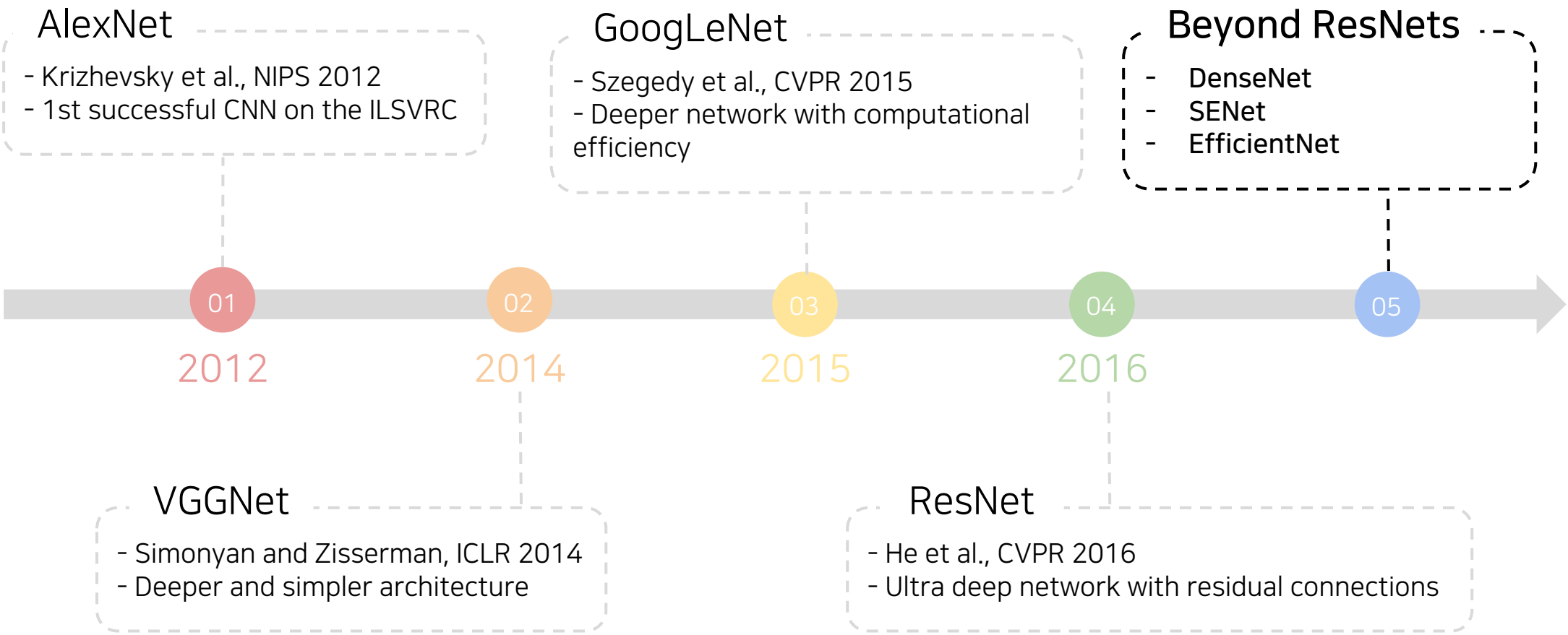
```
def _forward_impl(self, x: Tensor):
    # See note [TorchScript super()]
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.maxpool(x)

    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    x = self.layer4(x)

    x = self.avgpool(x)
    x = torch.flatten(x, 1)
    x = self.fc(x)

    return x
```

2.3 Beyond ResNets

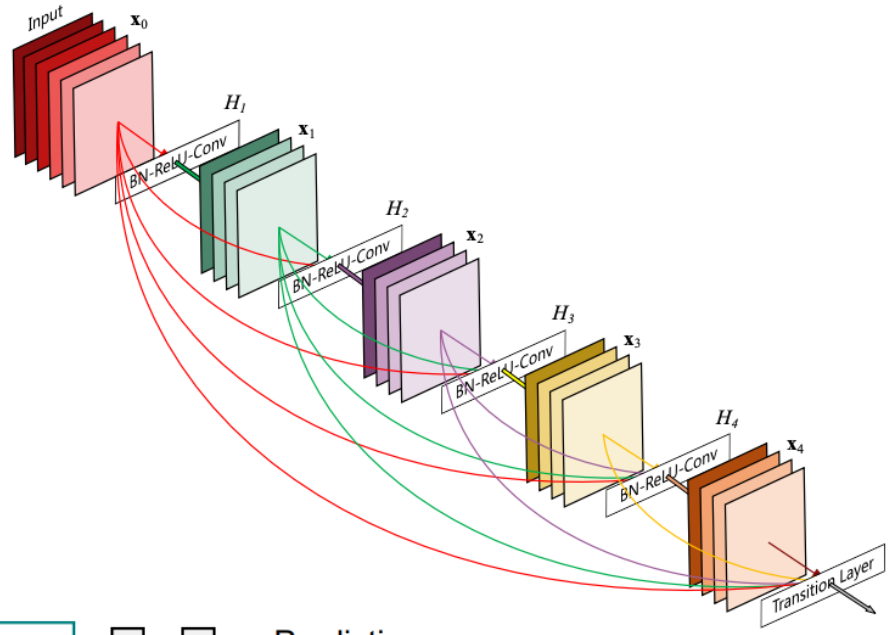


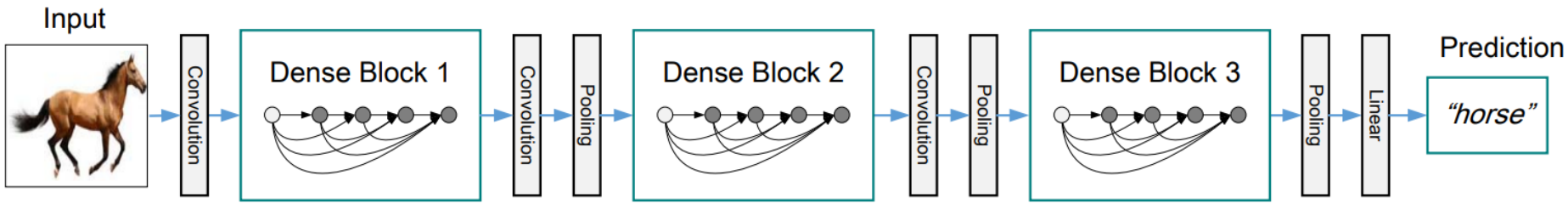
2.3 Beyond ResNets

DenseNet

[Huang et al., CVPR 2017]

- In ResNet, we added the input and the output of the layer element-wisely





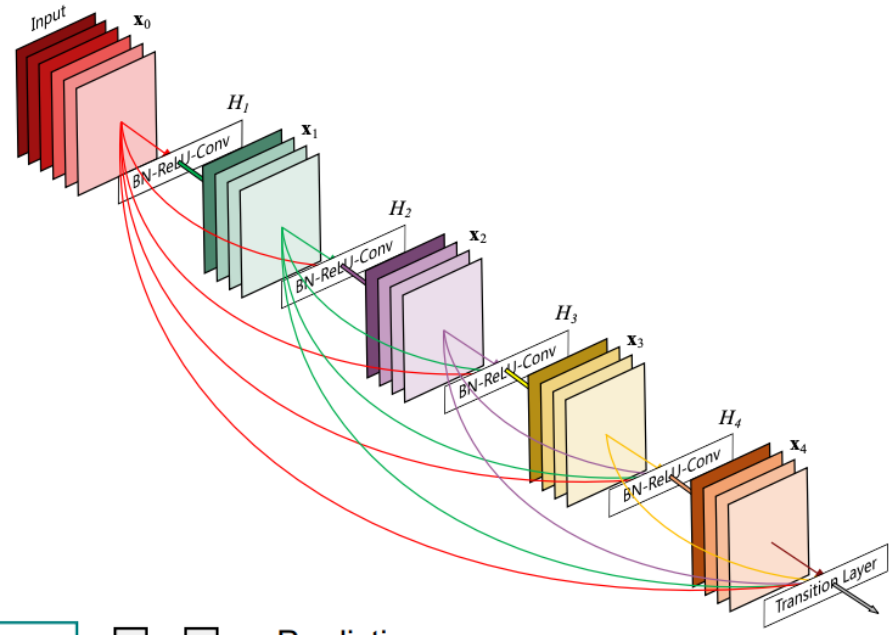
2.3 Beyond ResNets

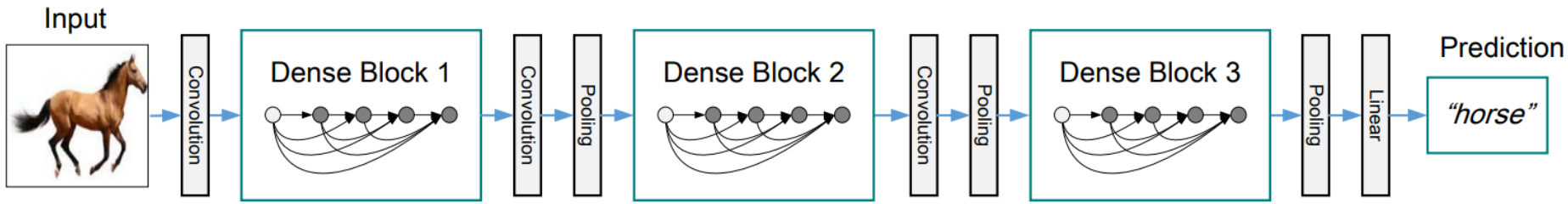
DenseNet

[Huang et al., CVPR 2017]

- In ResNet, we added the input and the output of the layer element-wisely

- In the Dense blocks, every output of each layer is concatenated along the channel axis.
  - Alleviate vanishing gradient problem
  - Strengthen feature propagation
  - Encourage the reuse of features





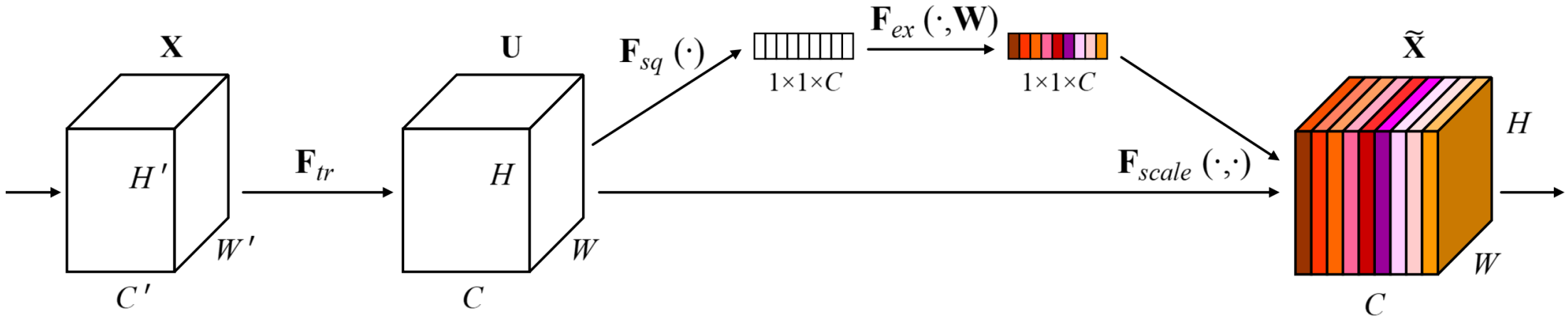
2.3 Beyond ResNets

SENet

[Hu et al., CVPR 2018]

* SE : Squeeze and Excitation

- Attention across channels
- Recalibrates channel-wise responses by modeling interdependencies between channels



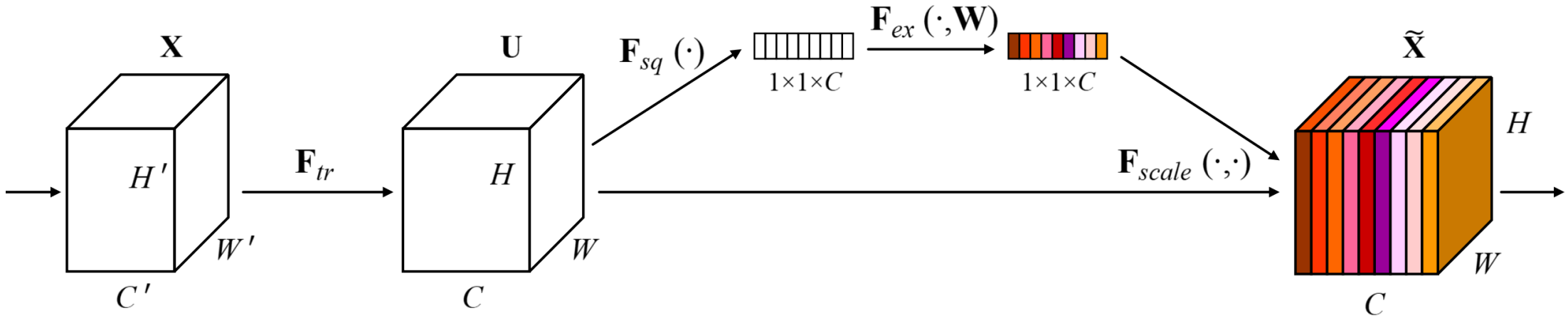
2.3 Beyond ResNets

SENet

[Hu et al., CVPR 2018]

* SE : Squeeze and Excitation

- Attention across channels
- Recalibrates channel-wise responses by modeling interdependencies between channels
- Squeeze and excitation operations
  - Squeeze: capturing distributions of channel-wise responses by global average pooling
  - Excitation: gating channels by channel-wise attention weights obtained by a FC layer

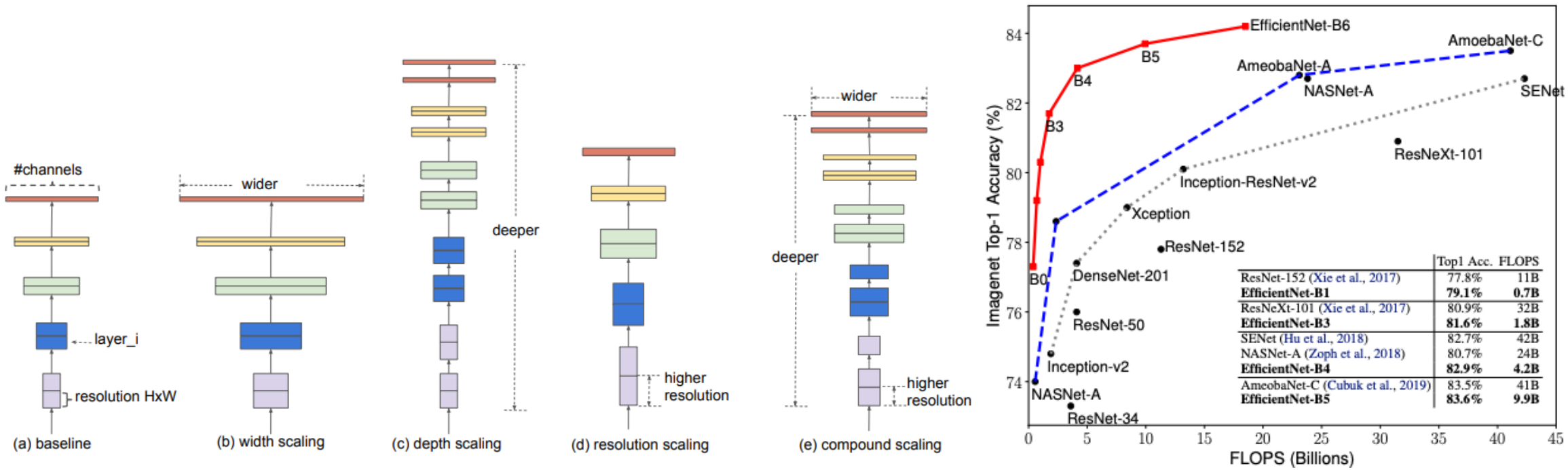


2.3 Beyond ResNets

EfficientNet

[Tan and Le, ICML 2019]

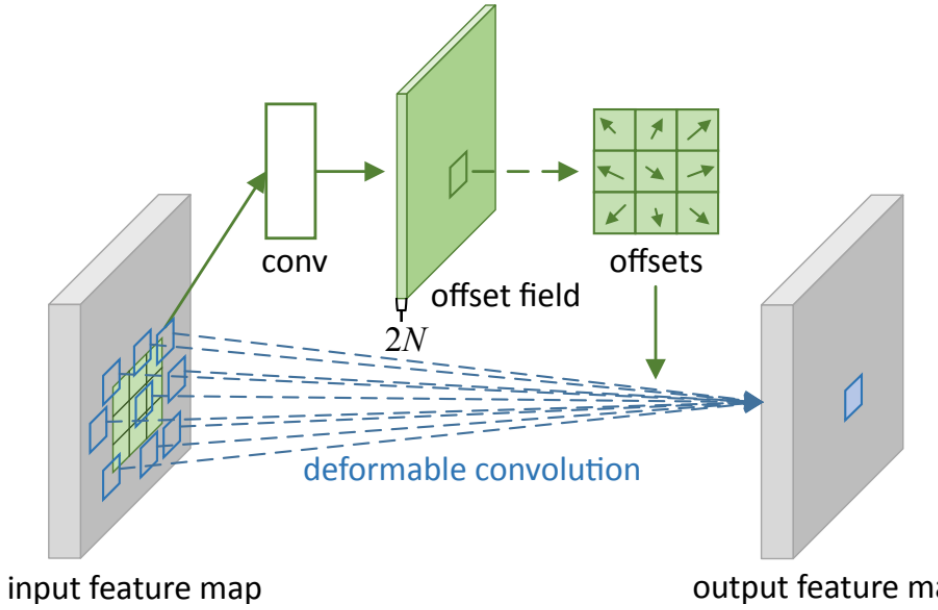
- Building deep, wide, and high resolution networks in an efficient way

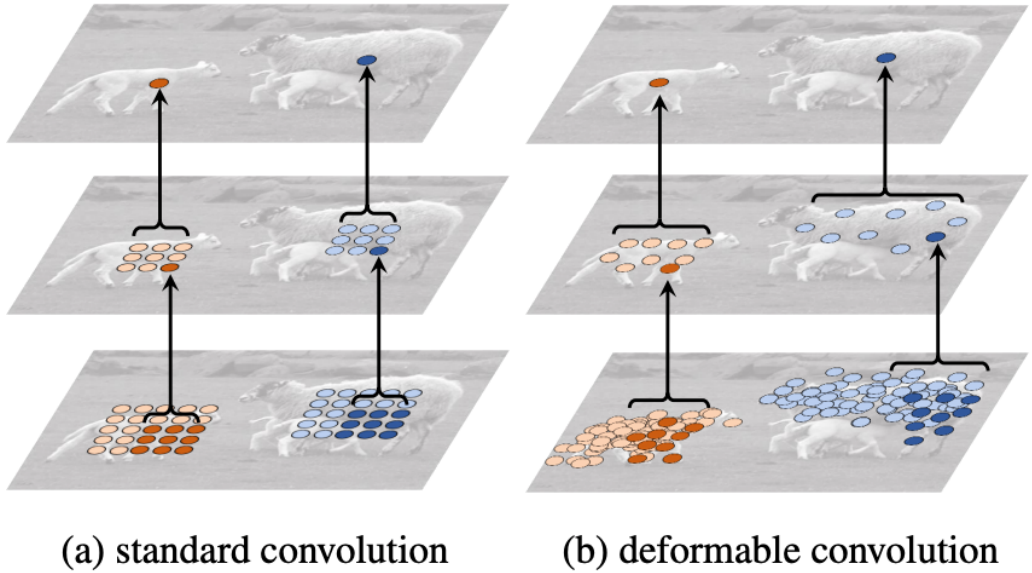


Deformable convolution

[Dai et al., ICCV 2017]

- 2D spatial offset prediction for irregular convolution
- Irregular grid sampling with 2D spatial offsets
- Implemented by standard CNN and grid sampling with 2D offsets





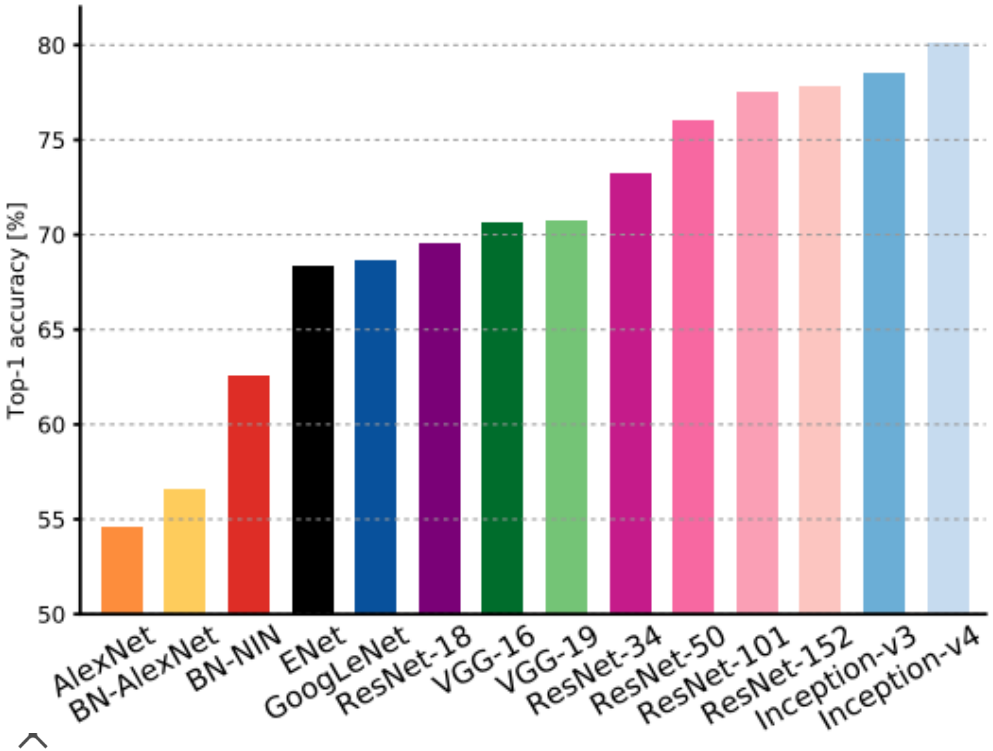
3.

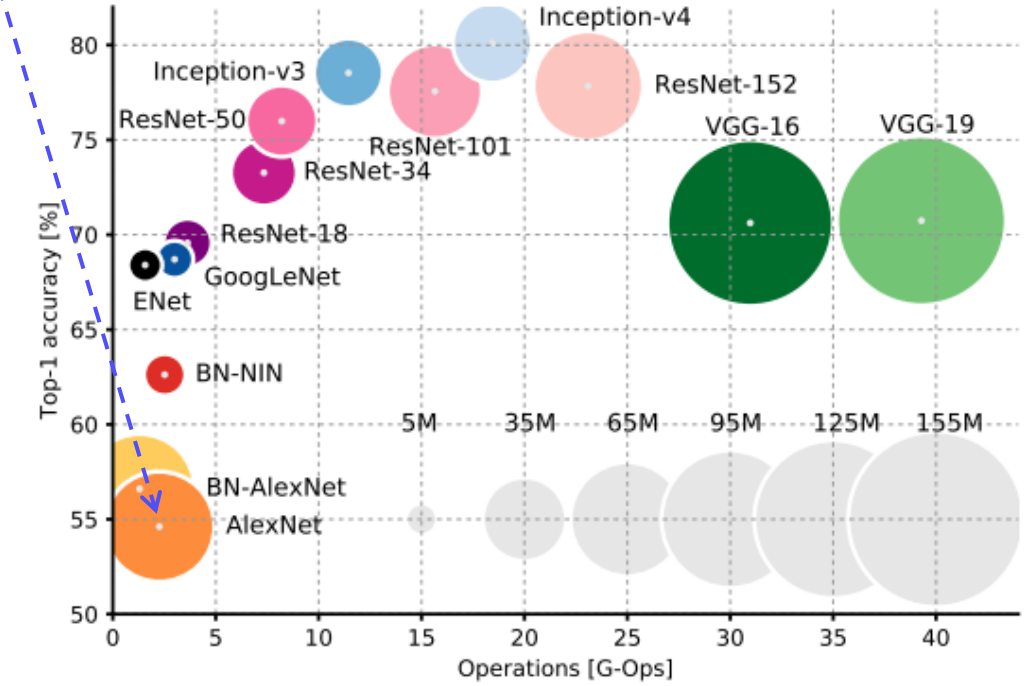
Summary of image classification

3.1 Summary of image classification

[Canziani et al., CVPR 2016]

- AlexNet: simple CNN architecture
- Simple computation, but heavy memory size
- Low accuracy

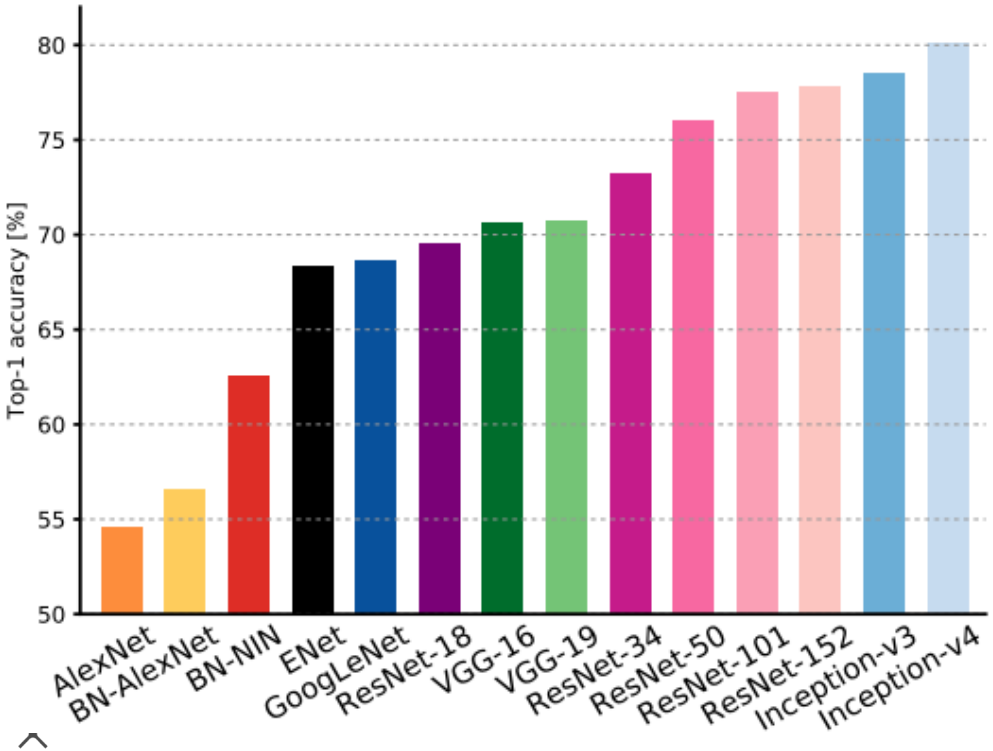


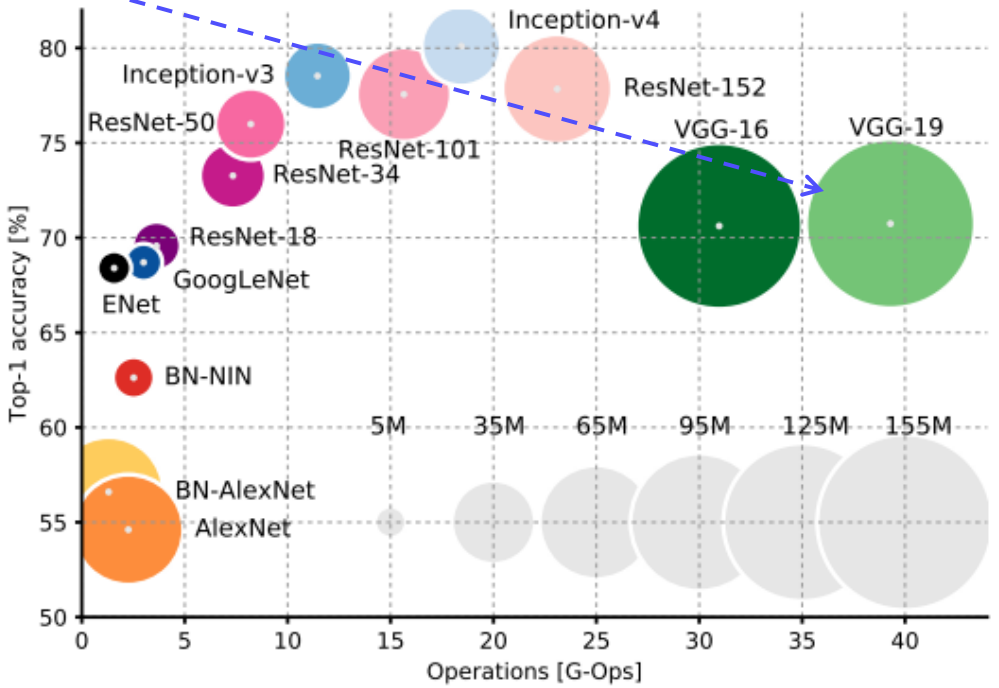


3.1 Summary of image classification

[Canziani et al., CVPR 2016]

- VGGNet: simple with 3x3 convolutions
- Highest memory, the heaviest computation

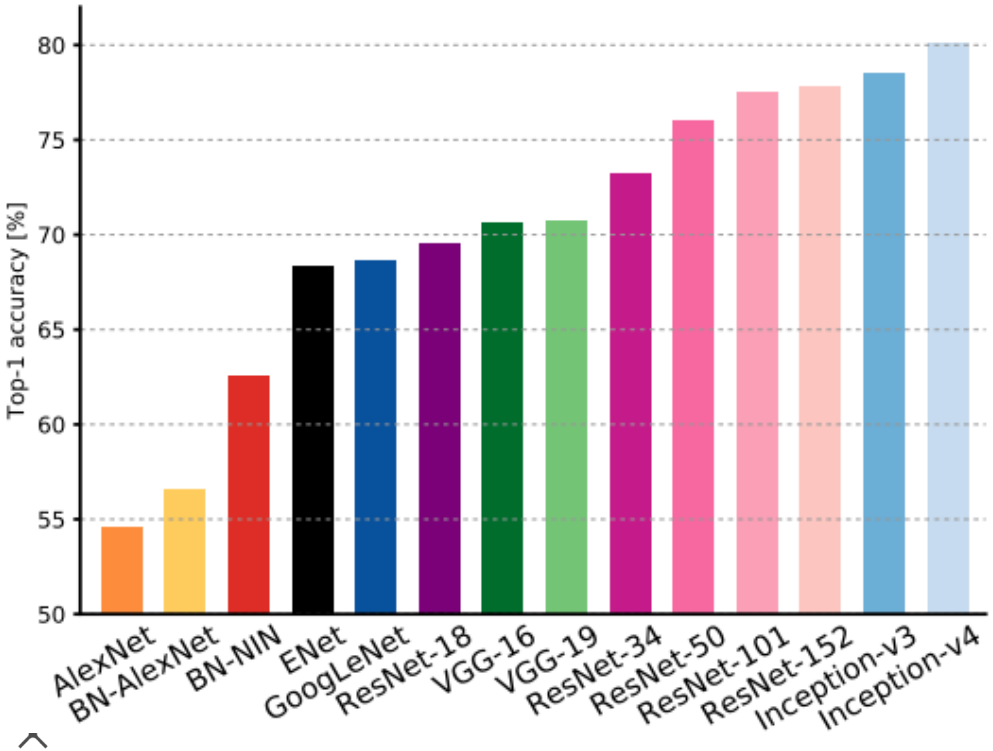


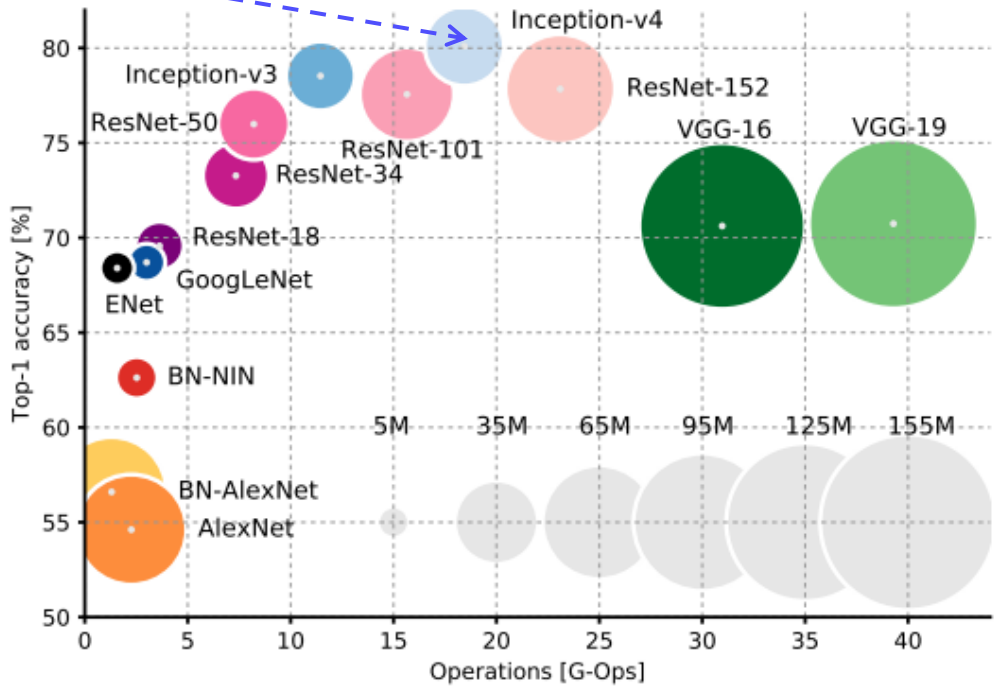


3.1 Summary of image classification

[Canziani et al., CVPR 2016]

- GoogLeNet: inception module and auxiliary classifier

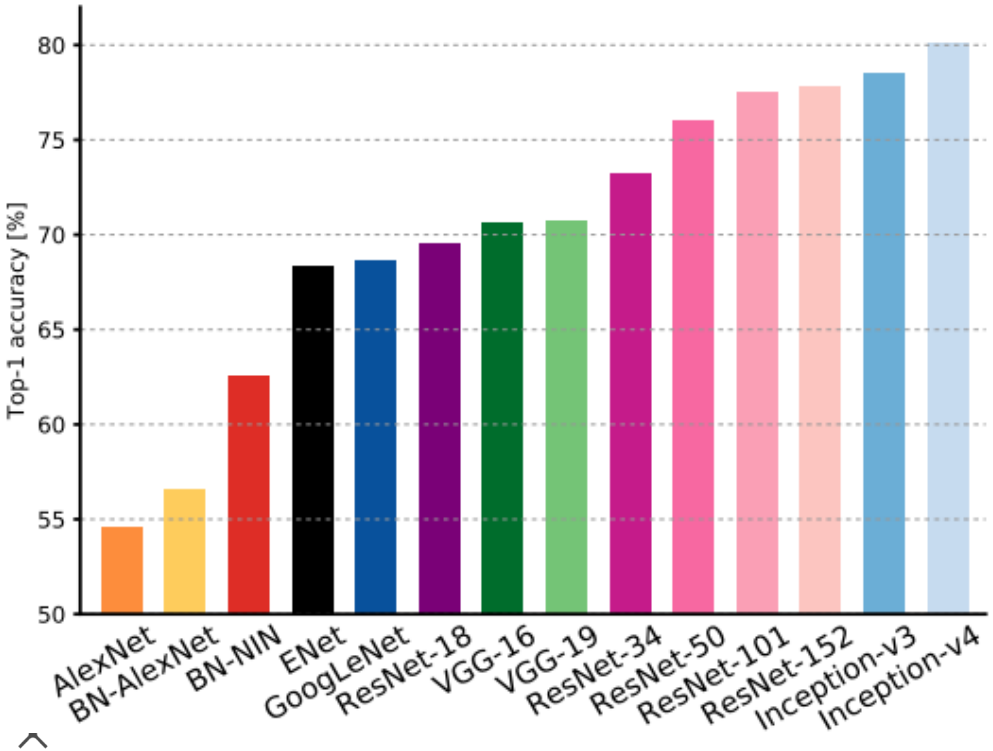


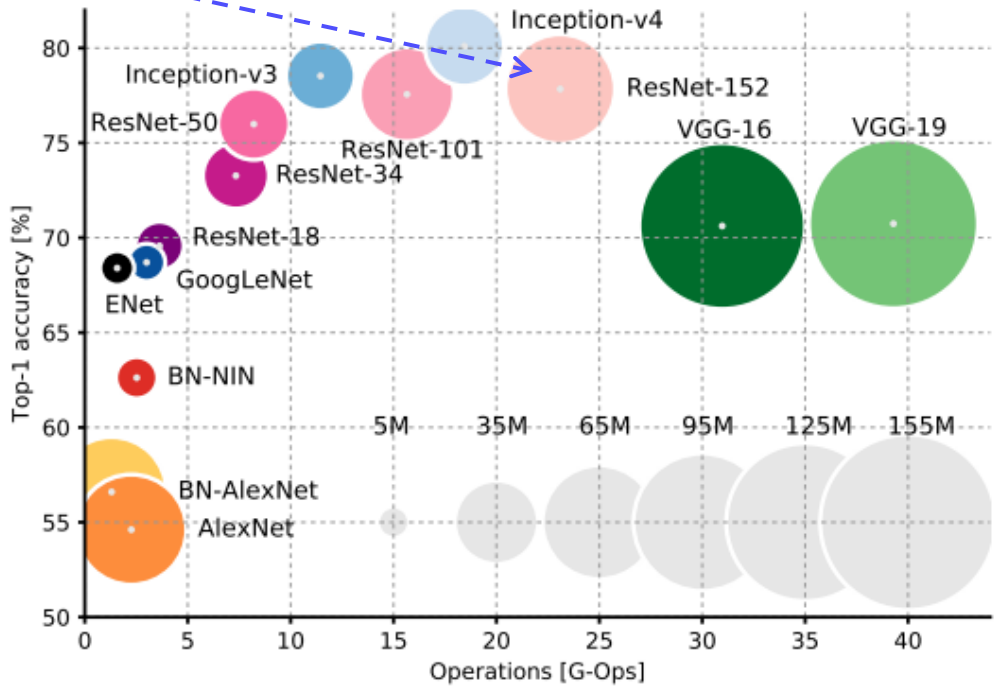


3.1 Summary of image classification

[Canziani et al., CVPR 2016]

- ResNet: deeper layers with residual blocks
- Moderate efficiency (depending on the model)

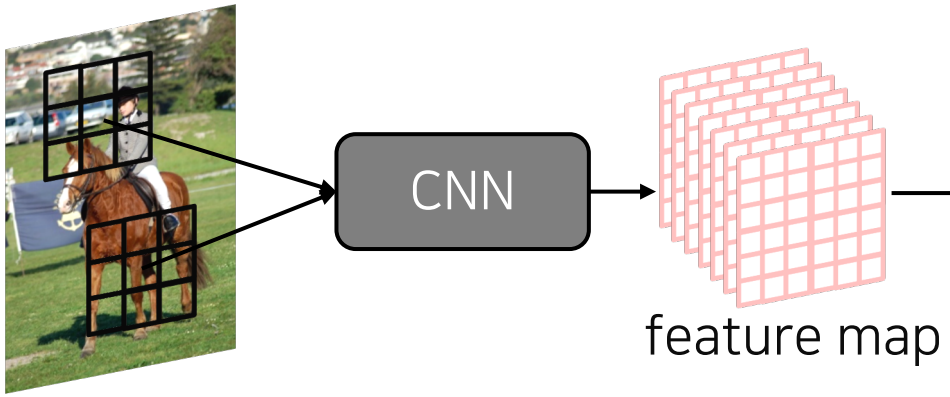


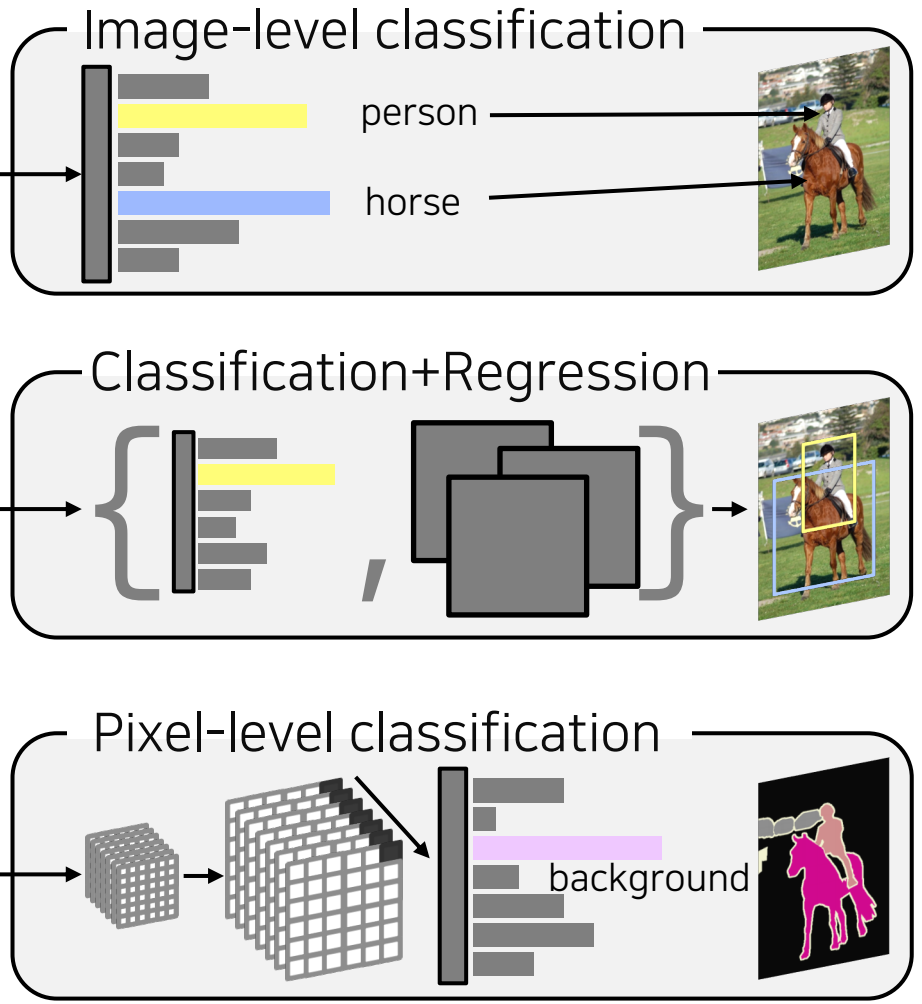


3.2 CNN backbones

Simple but powerful backbone model

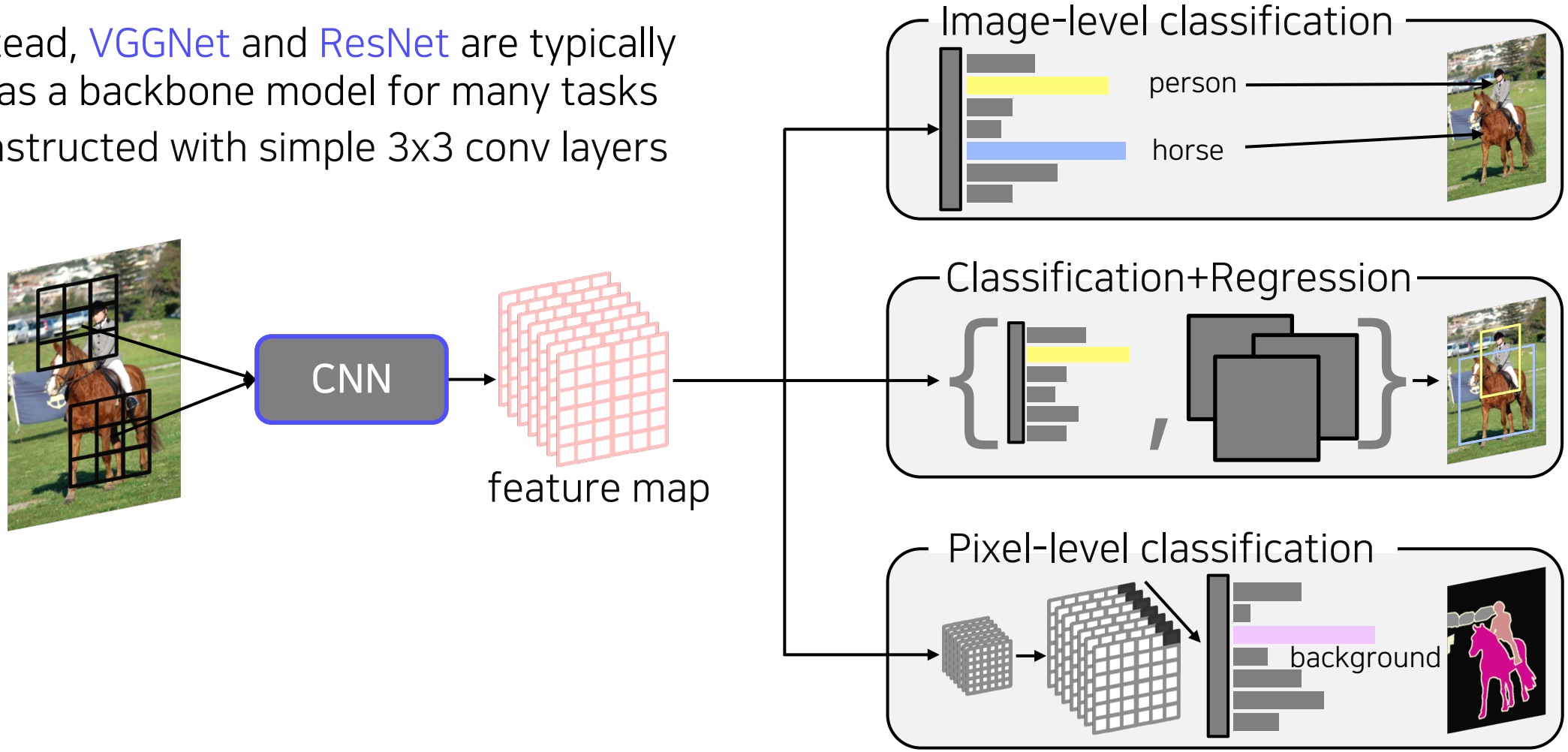
- GoogLeNet is the most efficient CNN model out of {AlexNet, VGG, ResNet}
- But, it is complicated to use





Simple but powerful backbone model

- Instead, VGGNet and ResNet are typically used as a backbone model for many tasks
- Constructed with simple 3x3 conv layers



Reference

2. CNN architectures for image classification 2

- Szegedy et al., Going Deeper with Convolution, CVPR 2015
- He et al., Deep Residual Learning for Image Recognition, CVPR 2015
- Veit et al., Residual Networks Behave Like Ensembles of Relatively Shallow Networks, NIPS 2016
- Huang et al., Densely Connected Convolutional Networks, CVPR 2017
- Hu et al., Squeeze-and-Excitation Networks, CVPR 2018
- Tan and Le, EfficientNet: Rethinking Model Scalinng for Convolutional Neural Networks, ICML 2019
- Dai et al., Deformable Convolutional Networks, ICCV 2017

3. Summary of image classification

- Canziani et al., An Analysis of Deep Neural Network Models for Practical Applications, CVPR 2016

End of Document
Thank You.