KOREATECH
한국기술교육대학교

Modern CNNs

November 2023

http://link.koreatech.ac.kr

[Reference]
https://d2l.ai/index.html

VGG

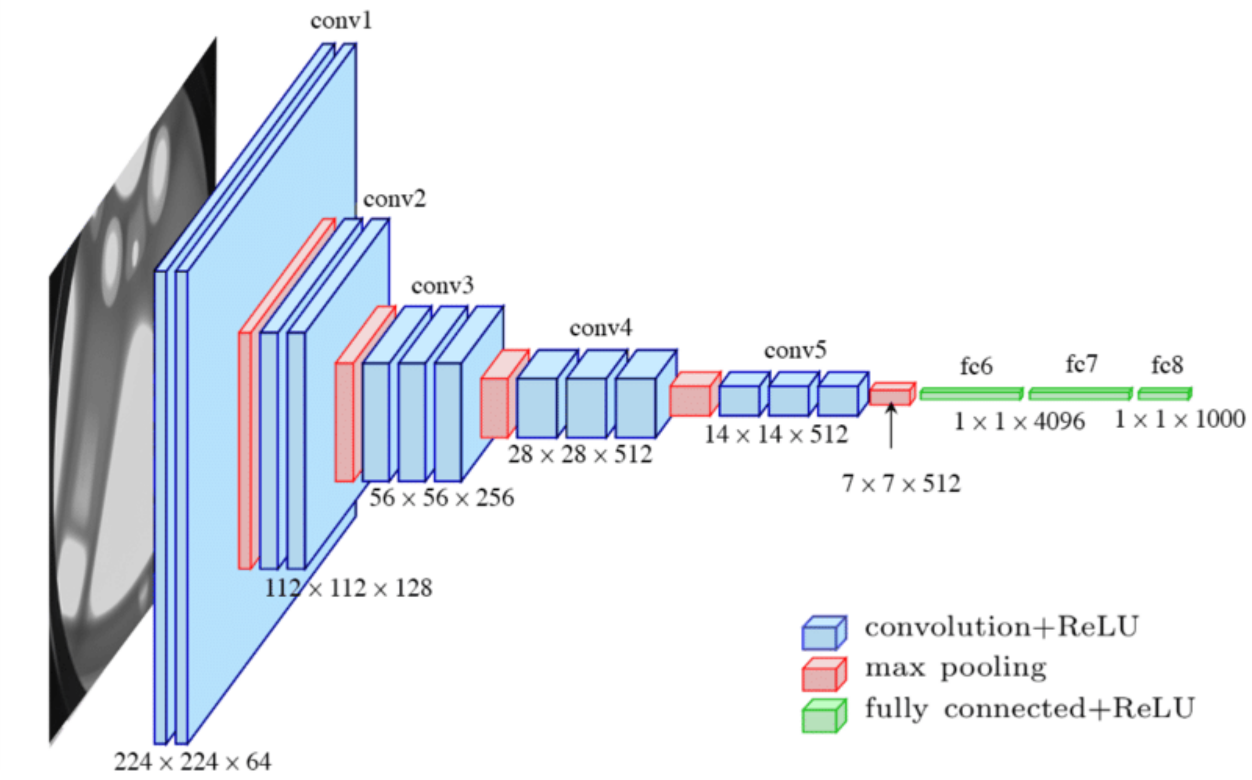
VGG

VGG:
"Visual Geometry Group," which is the name of a research group at the University of Oxford

◈VGG16 (ILSVRC 2014 2nd place, Oxford)

– It makes use of <u>a number of repeating blocks of layers</u>

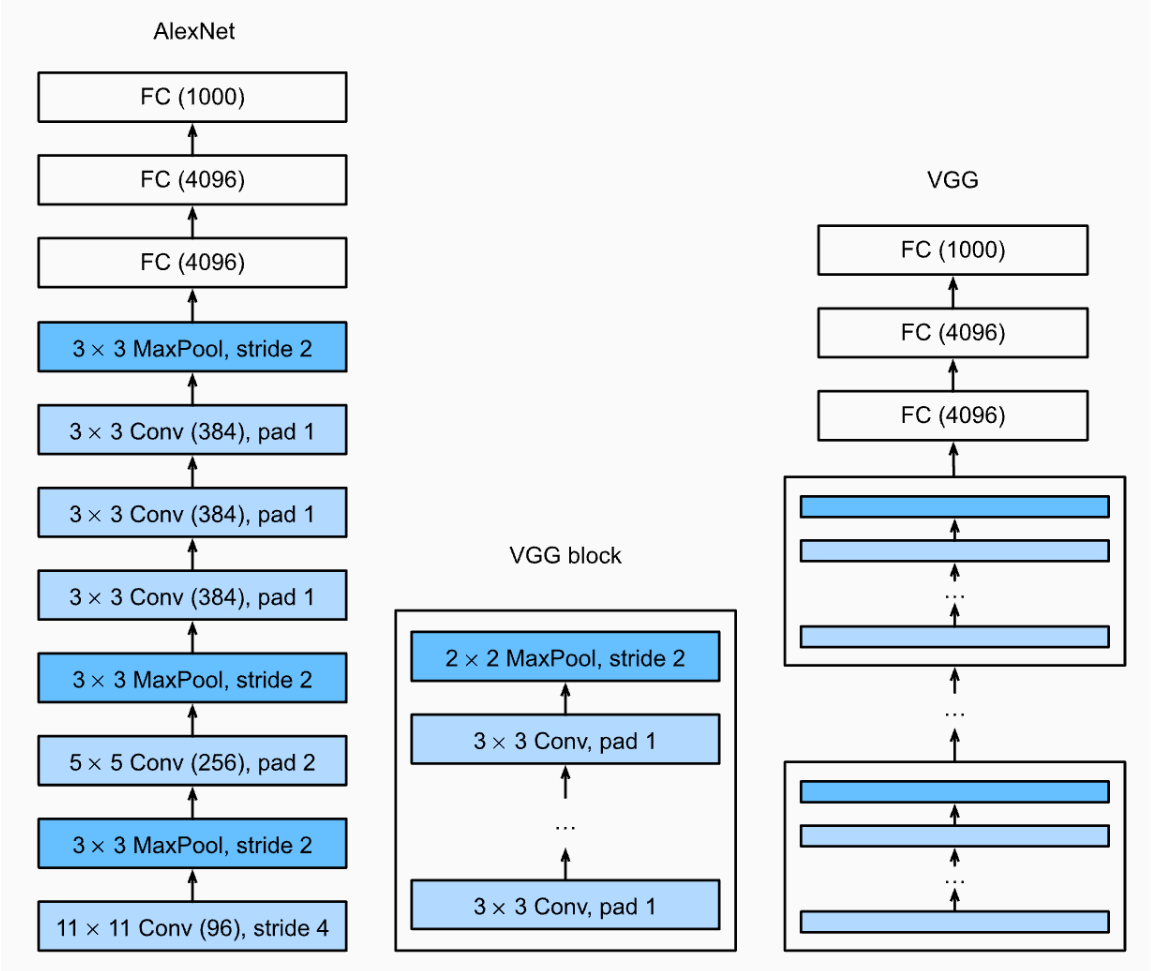
– The whole architecture is
[Input -
Conv1-1 - Conv1-2 - MaxPool -
Conv2-1 - Conv2-2 - MaxPool -
Conv3-1 - Conv3-2 - Conv3-3 - MaxPool -
Conv4-1 - Conv4-2 - Conv4-3 - MaxPool -
Conv5-1 - Conv5-2 - Conv5-3 - MaxPool -
FC6 - FC7 - FC8 (Output)]



– Why VGG16?
  - 13 Conv3x3 Networks
  - 3 FCNs

VGG

◈AlexNet vs. VGG16



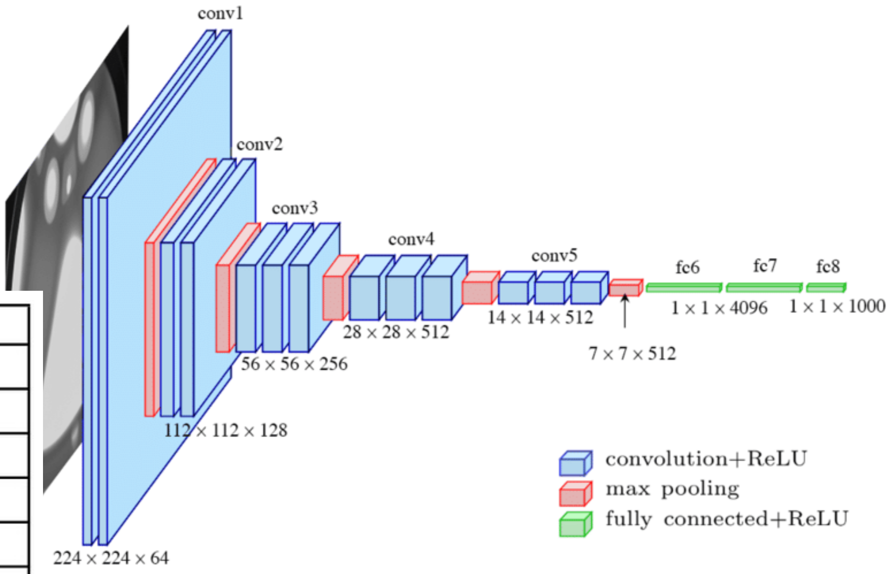
Published as a conference paper at ICLR 2015

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan* **& Andrew Zisserman**+
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

[From AlexNet to VGG]
The key difference is that
VGG consists of blocks of layers,
whereas AlexNet's layers are all designed individually.

VGG



◈VGG16 Architecture Details for ImageNet Dataset

VGG16 - Structural Details													
#	Input Image			output			Layer	Stride	Kernel		in	out	Param
1	224	224	3	224	224	64	conv3-64	1	3	3	3	64	1792
2	224	224	64	224	224	64	conv3064	1	3	3	64	64	36928
	224	224	64	112	112	64	maxpool	2	2	2	64	64	0
3	112	112	64	112	112	128	conv3-128	1	3	3	64	128	73856
4	112	112	128	112	112	128	conv3-128	1	3	3	128	128	147584
	112	112	128	56	56	128	maxpool	2	2	2	128	128	65664
5	56	56	128	56	56	256	conv3-256	1	3	3	128	256	295168
6	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
7	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
	56	56	256	28	28	256	maxpool	2	2	2	256	256	0
8	28	28	256	28	28	512	conv3-512	1	3	3	256	512	1180160
9	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
10	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
	28	28	512	14	14	512	maxpool	2	2	2	512	512	0
11	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
12	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
13	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
	14	14	512	7	7	512	maxpool	2	2	2	512	512	0
14	1	1	25088	1	1	4096	fc		1	1	25088	4096	102764544
15	1	1	4096	1	1	4096	fc		1	1	4096	4096	16781312
16	1	1	4096	1	1	1000	fc		1	1	4096	1000	4097000
Total													138,423,208

[NOTE] $7 \times 7 \times 512 = 25{,}088$

VGG with PyTorch

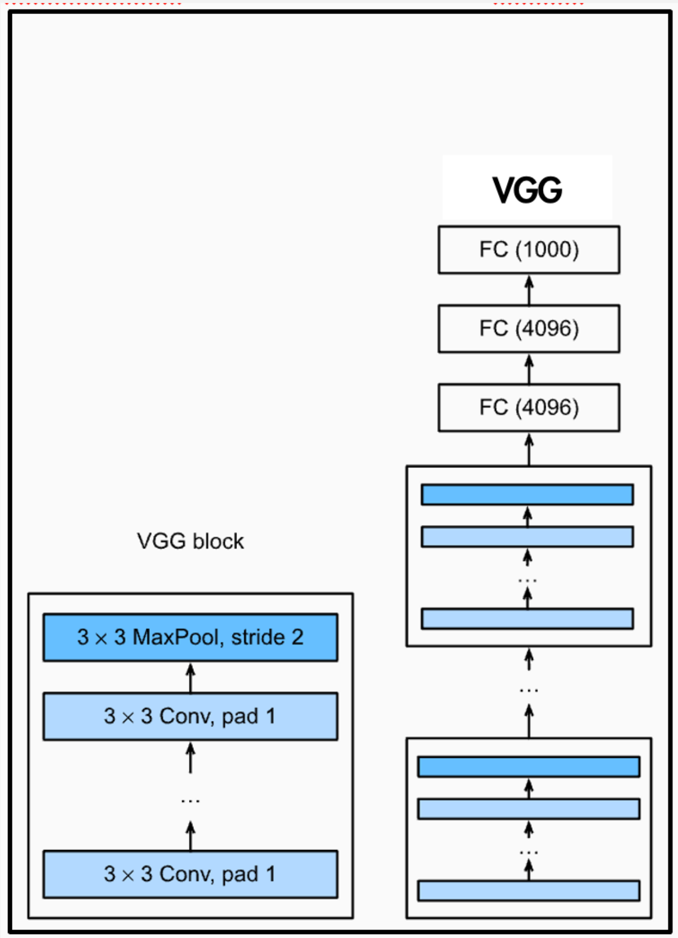
◈VGG16 with Cifar10 Dataset

```
def get_vgg_model():

  def vgg_block(num_conv_layers, out_channels):
    layers = []
    for _ in range(num_conv_layers):
      # When you instantiate a LazyConv2d, it doesn't immediately initialize its weights,
      # because it doesn't yet know the number of input channels.
      # Only when the module processes its first batch of data,
      # it infers the required number of input channels and properly initialize its weights.
      layers.append(
        nn.LazyConv2d(out_channels=out_channels, kernel_size=3, padding=1)
      )
      layers.append(nn.ReLU())

    layers.append(nn.MaxPool2d(kernel_size=2, stride=2))
    block = nn.Sequential(*layers)

    return block
```

VGG with PyTorch

◈VGG16 with Cifar10 Dataset

```
def get_vgg_model():
  ...
  class VGG(nn.Module):
    def __init__(self, block_info, n_output=10):
      super().__init__()
      conv_blocks = []
      for (num_conv_layers, out_channels) in block_info:
        conv_blocks.append(vgg_block(num_conv_layers, out_channels))

      self.model = nn.Sequential(
        *conv_blocks,
        nn.Flatten(),
        nn.LazyLinear(out_features=512),
        nn.ReLU(),
        nn.Dropout(0.5),
        nn.LazyLinear(out_features=512),
        nn.ReLU(),
        nn.Dropout(0.5),
        nn.LazyLinear(n_output)
      )
```



VGG with PyTorch

◈VGG16 with Cifar10 Dataset

```python
def get_vgg_model():
    ...

  class VGG(nn.Module):
    ...

    def forward(self, x):
      x = self.model(x)
      return x

  my_model = VGG(
    block_info=(
      (1, 64), (1, 128), (2, 256),
      (2, 512), (2, 512)
    ),
    n_output=10
  )

  return my_model
```

```
==========================================================================================================================================
Layer (type:depth-idx)                   Kernel Shape              Input Shape               Output Shape              Param #                   Mult-Adds
==========================================================================================================================================
VGG                                      --                        [1, 3, 32, 32]            [1, 10]                   --                        --
├─Sequential: 1-1                        --                        [1, 3, 32, 32]            [1, 10]                   --                        --
│    └─Sequential: 2-1                   --                        [1, 3, 32, 32]            [1, 64, 16, 16]           --                        --
│    │    └─Conv2d: 3-1                  [3, 3]                    [1, 3, 32, 32]            [1, 64, 32, 32]           1,792                     1,835,008
│    │    └─ReLU: 3-2                    --                        [1, 64, 32, 32]           [1, 64, 32, 32]           --                        --
│    │    └─MaxPool2d: 3-3               2                         [1, 64, 32, 32]           [1, 64, 16, 16]           --                        --
│    └─Sequential: 2-2                   --                        [1, 64, 16, 16]           [1, 128, 8, 8]            --                        --
│    │    └─Conv2d: 3-4                  [3, 3]                    [1, 64, 16, 16]           [1, 128, 16, 16]          73,856                    18,907,136
│    │    └─ReLU: 3-5                    --                        [1, 128, 16, 16]          [1, 128, 16, 16]          --                        --
│    │    └─MaxPool2d: 3-6               2                         [1, 128, 16, 16]          [1, 128, 8, 8]            --                        --
│    └─Sequential: 2-3                   --                        [1, 128, 8, 8]            [1, 256, 4, 4]            --                        --
│    │    └─Conv2d: 3-7                  [3, 3]                    [1, 128, 8, 8]            [1, 256, 8, 8]            295,168                   18,890,752
│    │    └─ReLU: 3-8                    --                        [1, 256, 8, 8]            [1, 256, 8, 8]            --                        --
│    │    └─Conv2d: 3-9                  [3, 3]                    [1, 256, 8, 8]            [1, 256, 8, 8]            590,080                   37,765,120
│    │    └─ReLU: 3-10                   --                        [1, 256, 8, 8]            [1, 256, 8, 8]            --                        --
│    │    └─MaxPool2d: 3-11              2                         [1, 256, 8, 8]            [1, 256, 4, 4]            --                        --
│    └─Sequential: 2-4                   --                        [1, 256, 4, 4]            [1, 512, 2, 2]            --                        --
│    │    └─Conv2d: 3-12                 [3, 3]                    [1, 256, 4, 4]            [1, 512, 4, 4]            1,180,160                 18,882,560
│    │    └─ReLU: 3-13                   --                        [1, 512, 4, 4]            [1, 512, 4, 4]            --                        --
│    │    └─Conv2d: 3-14                 [3, 3]                    [1, 512, 4, 4]            [1, 512, 4, 4]            2,359,808                 37,756,928
│    │    └─ReLU: 3-15                   --                        [1, 512, 4, 4]            [1, 512, 4, 4]            --                        --
│    │    └─MaxPool2d: 3-16              2                         [1, 512, 4, 4]            [1, 512, 2, 2]            --                        --
│    └─Sequential: 2-5                   --                        [1, 512, 2, 2]            [1, 512, 1, 1]            --                        --
│    │    └─Conv2d: 3-17                 [3, 3]                    [1, 512, 2, 2]            [1, 512, 2, 2]            2,359,808                 9,439,232
│    │    └─ReLU: 3-18                   --                        [1, 512, 2, 2]            [1, 512, 2, 2]            --                        --
│    │    └─Conv2d: 3-19                 [3, 3]                    [1, 512, 2, 2]            [1, 512, 2, 2]            2,359,808                 9,439,232
│    │    └─ReLU: 3-20                   --                        [1, 512, 2, 2]            [1, 512, 2, 2]            --                        --
│    │    └─MaxPool2d: 3-21              2                         [1, 512, 2, 2]            [1, 512, 1, 1]            --                        --
│    └─Flatten: 2-6                      --                        [1, 512, 1, 1]            [1, 512]                  --                        --
│    └─Linear: 2-7                       --                        [1, 512]                  [1, 512]                  262,656                   262,656
│    └─ReLU: 2-8                         --                        [1, 512]                  [1, 512]                  --                        --
│    └─Dropout: 2-9                      --                        [1, 512]                  [1, 512]                  --                        --
│    └─Linear: 2-10                      --                        [1, 512]                  [1, 512]                  262,656                   262,656
│    └─ReLU: 2-11                        --                        [1, 512]                  [1, 512]                  --                        --
│    └─Dropout: 2-12                     --                        [1, 512]                  [1, 512]                  --                        --
│    └─Linear: 2-13                      --                        [1, 512]                  [1, 10]                   5,130                     5,130
==========================================================================================================================================
Total params: 9,750,922
Trainable params: 9,750,922
Non-trainable params: 0
Total mult-adds (M): 153.45
==========================================================================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 1.22
Params size (MB): 39.00
Estimated Total Size (MB): 40.24
==========================================================================================================================================
```

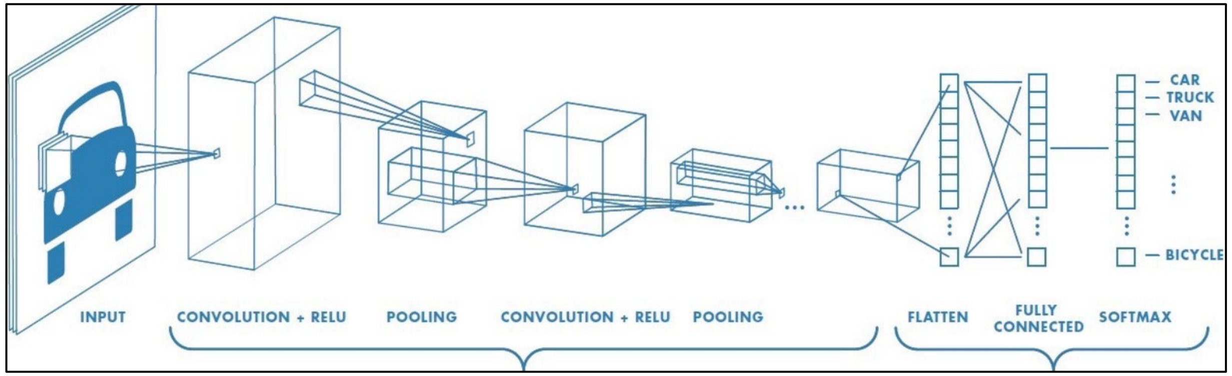
Network in Network (NiN)

NiN

◈A Common Design Pattern of LeNet, AlexNet, and VGG

– extract features exploiting spatial structure via a sequence of convolutions and pooling layers, and then post-process the representations via fully connected layers
  - Compared with LeNet, AlexNet and VGG widen and deepen the sequence

◈Drawbacks of LeNet, AlexNet, and VGG



– 1) the fully connected layers at the end of the architecture require tremendous numbers of parameters

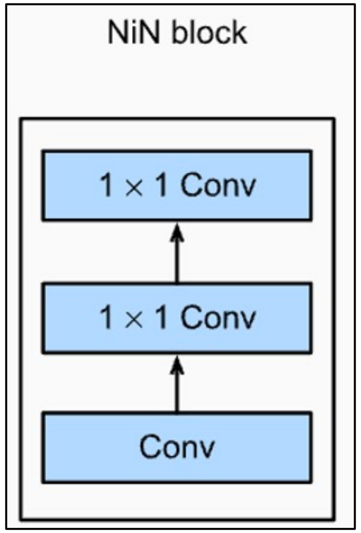
– 2) it is impossible to add fully connected layers earlier in the network to increase the degree of nonlinearity
  - doing so would destroy the spatial structure and require potentially even more memory

NiN

◈ Network in Network [National University of Singapore, 2014]

– 1×1 Convolution
  - To add local nonlinearities across the channel activations

– Global Average Pooling
  - To integrate across all locations in the last representation layer



Network In Network

Min Lin[1,2], Qiang Chen[2], Shuicheng Yan[2]
[1]Graduate School for Integrative Sciences and Engineering
[2]Department of Electronic & Computer Engineering
National University of Singapore, Singapore
{linmin,chenqiang,eleyans}@nus.edu.sg

The term "Network in Network" (NiN) was chosen by the authors to describe their novel architectural innovation in CNNs.
The main idea behind this naming is the introduction of small (FC) neural networks within traditional convolutional layers, essentially embedding a network within the broader network.

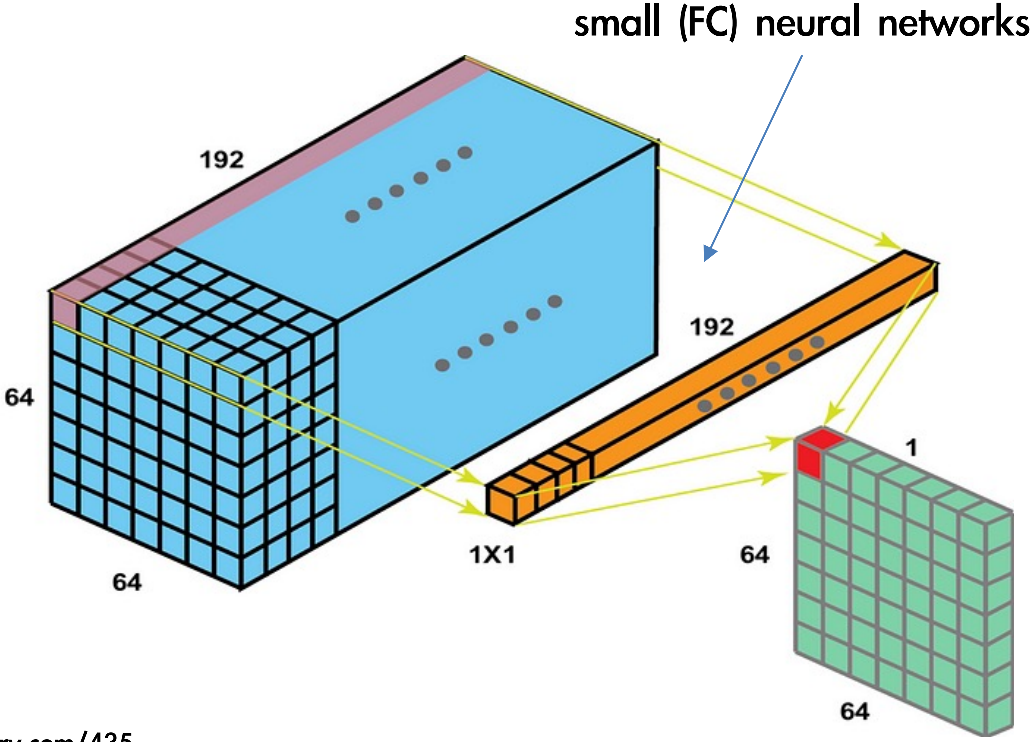
NiN

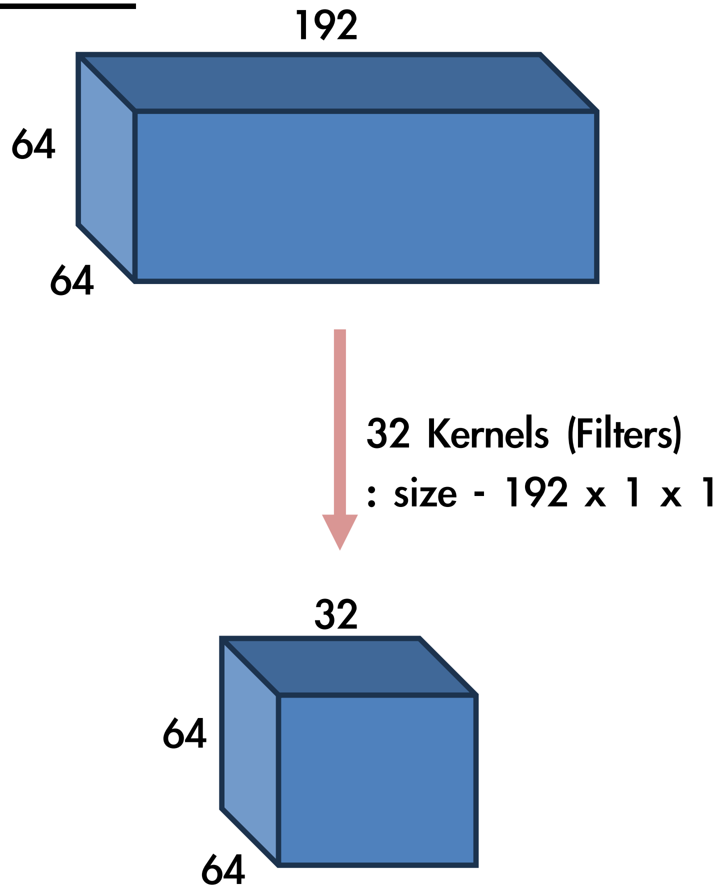
◈1×1 Convolution (Pointwise Convolution)

– Channel-wise feature response (output or activation) recalibration

– <u>Increasing non-linearity without changing spatial dimensions</u>

– <u>Dimensionality reduction</u>





[Source] https://geunuk.tistory.com/435

NiN

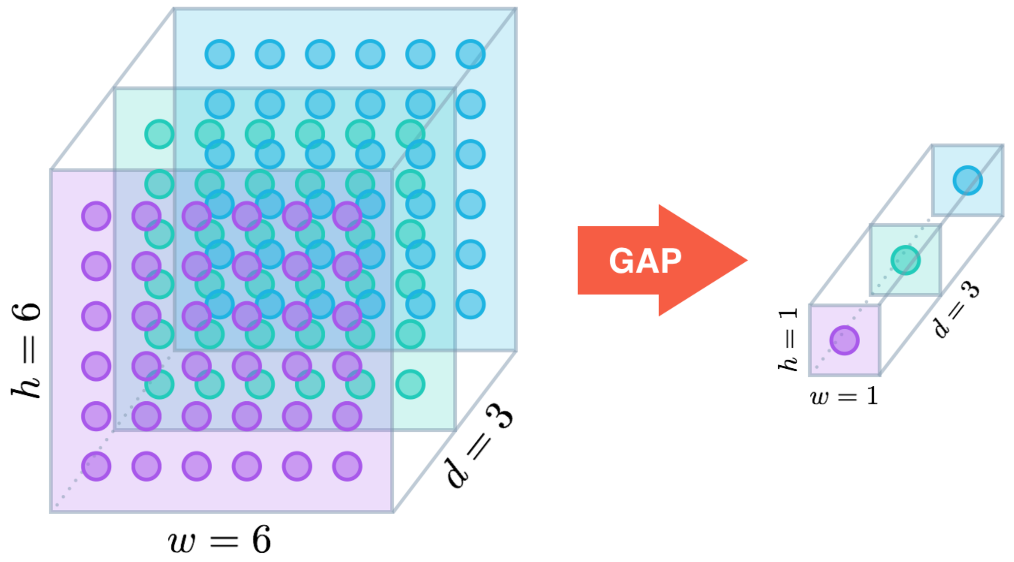
◈Global Average Pooling (GAP)

– Given a feature map (output from some convolutional layer) of shape $C \times H \times W$, GAP computes <u>the average value of each feature map</u> across the full spatial dimensions

- for each channel C, the average of all $H \times W$ values is computed
- This results in a $C \times 1 \times 1$ output

– GAP with classification tasks

- After extracting feature maps using convolutional layers, GAP is applied to obtain a fixed-size vector, which is then passed to a softmax layer for classification

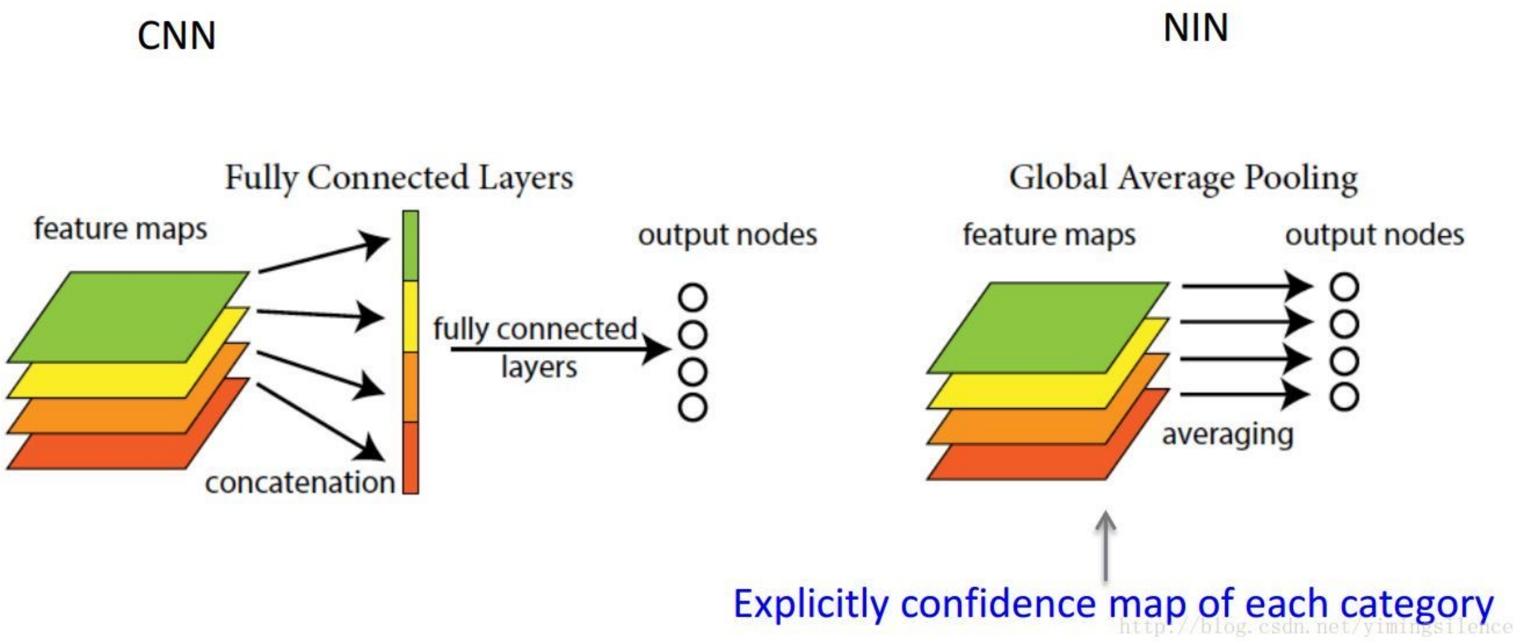


[Source] https://mole-starseeker.tistory.com/66

NiN

◈GAP (NiN) vs. Fully Connected Layers (CNN, VGG, AlexNet, LeNet)

– Spatial invariance level of GAP is higher than the level of fully connected layer
  - By averaging out the spatial information, GAP introduces some level of spatial invariance



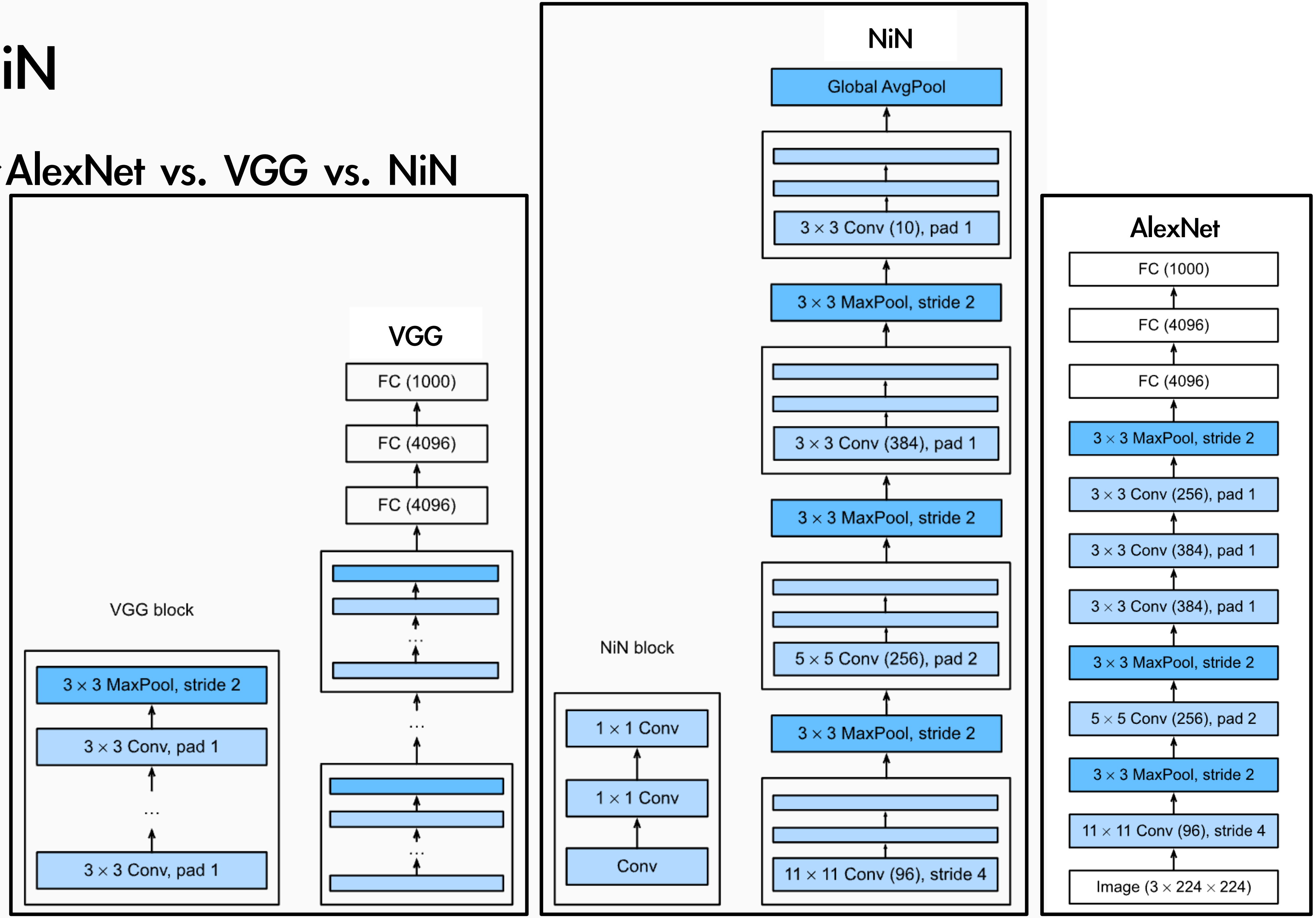
– Less Computational Load

– Reduction in Overfitting
  - GAP reduces the number of parameters in the network when compared to using fully connected layers directly after the convolutional layers
  - Fewer parameters mean less risk of overfitting, especially when there's limited training data

[Source] https://www.plob.org/article/22160.html

NiN

◇ AlexNet vs. VGG vs. NiN



NiN

◈NiN with Cifar10 Dataset

```
def get_nin_model():

  def nin_block(out_channels, kernel_size, strides, padding):
    block = nn.Sequential(
      nn.LazyConv2d(out_channels=out_channels, kernel_size=kernel_size, stride=strides, padding=padding),
      nn.ReLU(),
      nn.LazyConv2d(out_channels=out_channels, kernel_size=1),
      nn.ReLU(),
      nn.LazyConv2d(out_channels=out_channels, kernel_size=1),
      nn.ReLU()
    )

    return block
```

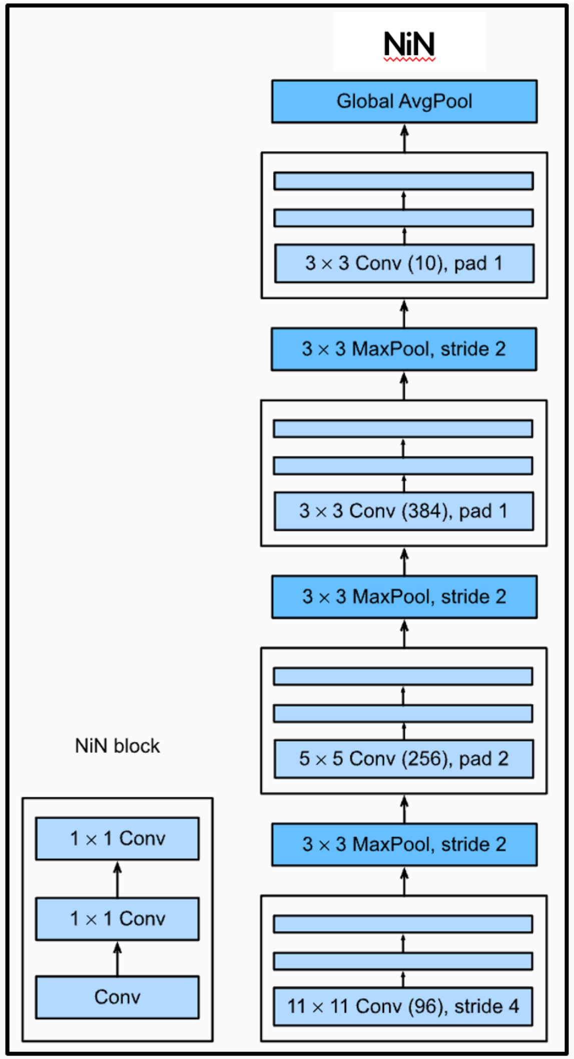
◈NiN with Cifar10 Dataset

NiN

link_dl/_01_code/_09_modern_cnns/_02_nin/a_cifar10_train_nin.py

```
def get_nin_model():
  ...
  class NiN(nn.Module):
    def __init__(self, n_output=10):
      super().__init__()

      self.model = nn.Sequential(
        nin_block(out_channels=96, kernel_size=3, strides=1, padding=1),
        nn.MaxPool2d(kernel_size=3, stride=2),

        nin_block(out_channels=256, kernel_size=3, strides=1, padding=1),
        nn.MaxPool2d(kernel_size=3, stride=2),

        nin_block(out_channels=384, kernel_size=3, strides=1, padding=1),
        nn.MaxPool2d(kernel_size=3, stride=2),
        nn.Dropout(0.5),

        nin_block(out_channels=n_output, kernel_size=3, strides=1, padding=1),
        nn.AdaptiveAvgPool2d(output_size=(1, 1)),
        nn.Flatten()
      )
```

The number of output features is equal to the number of input planes



NiN

◈NiN with Cifar10 Dataset

```python
def get_nin_model():
  ...

  class NiN(nn.Module):
    ...

    def forward(self, x):
      x = self.model(x)
      return x

  my_model = NiN(n_output=10)

  return my_model
```

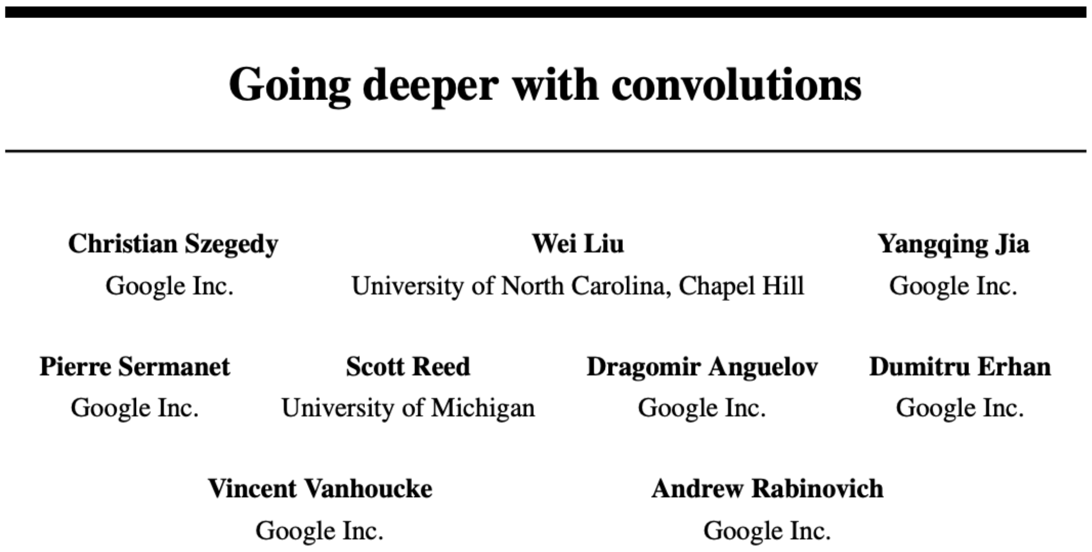
```
==========================================================================================================================================
Layer (type:depth-idx)                   Kernel Shape              Input Shape               Output Shape              Param #                   Mult-Adds
==========================================================================================================================================
NiN                                      --                        [1, 3, 32, 32]            [1, 10]                   --                        --
├─Sequential: 1-1                        --                        [1, 3, 32, 32]            [1, 10]                   --                        --
│    └─Sequential: 2-1                   --                        [1, 3, 32, 32]            [1, 96, 32, 32]           --                        --
│    │    └─Conv2d: 3-1                  [3, 3]                    [1, 3, 32, 32]            [1, 96, 32, 32]           2,688                     2,752,512
│    │    └─ReLU: 3-2                    --                        [1, 96, 32, 32]           [1, 96, 32, 32]           --                        --
│    │    └─Conv2d: 3-3                  [1, 1]                    [1, 96, 32, 32]           [1, 96, 32, 32]           9,312                     9,535,488
│    │    └─ReLU: 3-4                    --                        [1, 96, 32, 32]           [1, 96, 32, 32]           --                        --
│    │    └─Conv2d: 3-5                  [1, 1]                    [1, 96, 32, 32]           [1, 96, 32, 32]           9,312                     9,535,488
│    │    └─ReLU: 3-6                    --                        [1, 96, 32, 32]           [1, 96, 32, 32]           --                        --
│    └─MaxPool2d: 2-2                    3                         [1, 96, 32, 32]           [1, 96, 15, 15]           --                        --
│    └─Sequential: 2-3                   --                        [1, 96, 15, 15]           [1, 256, 15, 15]          --                        --
│    │    └─Conv2d: 3-7                  [3, 3]                    [1, 96, 15, 15]           [1, 256, 15, 15]          221,440                   49,824,000
│    │    └─ReLU: 3-8                    --                        [1, 256, 15, 15]          [1, 256, 15, 15]          --                        --
│    │    └─Conv2d: 3-9                  [1, 1]                    [1, 256, 15, 15]          [1, 256, 15, 15]          65,792                    14,803,200
│    │    └─ReLU: 3-10                   --                        [1, 256, 15, 15]          [1, 256, 15, 15]          --                        --
│    │    └─Conv2d: 3-11                 [1, 1]                    [1, 256, 15, 15]          [1, 256, 15, 15]          65,792                    14,803,200
│    │    └─ReLU: 3-12                   --                        [1, 256, 15, 15]          [1, 256, 15, 15]          --                        --
│    └─MaxPool2d: 2-4                    3                         [1, 256, 15, 15]          [1, 256, 7, 7]            --                        --
│    └─Sequential: 2-5                   --                        [1, 256, 7, 7]            [1, 384, 7, 7]            --                        --
│    │    └─Conv2d: 3-13                 [3, 3]                    [1, 256, 7, 7]            [1, 384, 7, 7]            885,120                   43,370,880
│    │    └─ReLU: 3-14                   --                        [1, 384, 7, 7]            [1, 384, 7, 7]            --                        --
│    │    └─Conv2d: 3-15                 [1, 1]                    [1, 384, 7, 7]            [1, 384, 7, 7]            147,840                   7,244,160
│    │    └─ReLU: 3-16                   --                        [1, 384, 7, 7]            [1, 384, 7, 7]            --                        --
│    │    └─Conv2d: 3-17                 [1, 1]                    [1, 384, 7, 7]            [1, 384, 7, 7]            147,840                   7,244,160
│    │    └─ReLU: 3-18                   --                        [1, 384, 7, 7]            [1, 384, 7, 7]            --                        --
│    └─MaxPool2d: 2-6                    3                         [1, 384, 7, 7]            [1, 384, 3, 3]            --                        --
│    └─Dropout: 2-7                      --                        [1, 384, 3, 3]            [1, 384, 3, 3]            --                        --
│    └─Sequential: 2-8                   --                        [1, 384, 3, 3]            [1, 10, 3, 3]             --                        --
│    │    └─Conv2d: 3-19                 [3, 3]                    [1, 384, 3, 3]            [1, 10, 3, 3]             34,570                    311,130
│    │    └─ReLU: 3-20                   --                        [1, 10, 3, 3]             [1, 10, 3, 3]             --                        --
│    │    └─Conv2d: 3-21                 [1, 1]                    [1, 10, 3, 3]             [1, 10, 3, 3]             110                       990
│    │    └─ReLU: 3-22                   --                        [1, 10, 3, 3]             [1, 10, 3, 3]             --                        --
│    │    └─Conv2d: 3-23                 [1, 1]                    [1, 10, 3, 3]             [1, 10, 3, 3]             110                       990
│    │    └─ReLU: 3-24                   --                        [1, 10, 3, 3]             [1, 10, 3, 3]             --                        --
│    └─AdaptiveAvgPool2d: 2-9            --                        [1, 10, 3, 3]             [1, 10, 1, 1]             --                        --
│    └─Flatten: 2-10                     --                        [1, 10, 1, 1]             [1, 10]                   --                        --
==========================================================================================================================================
Total params: 1,589,926
Trainable params: 1,589,926
Non-trainable params: 0
Total mult-adds (M): 159.43
==========================================================================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 4.20
Params size (MB): 6.36
Estimated Total Size (MB): 10.57
==========================================================================================================================================
```

GoogLeNet (with Inception)

GoogLeNet

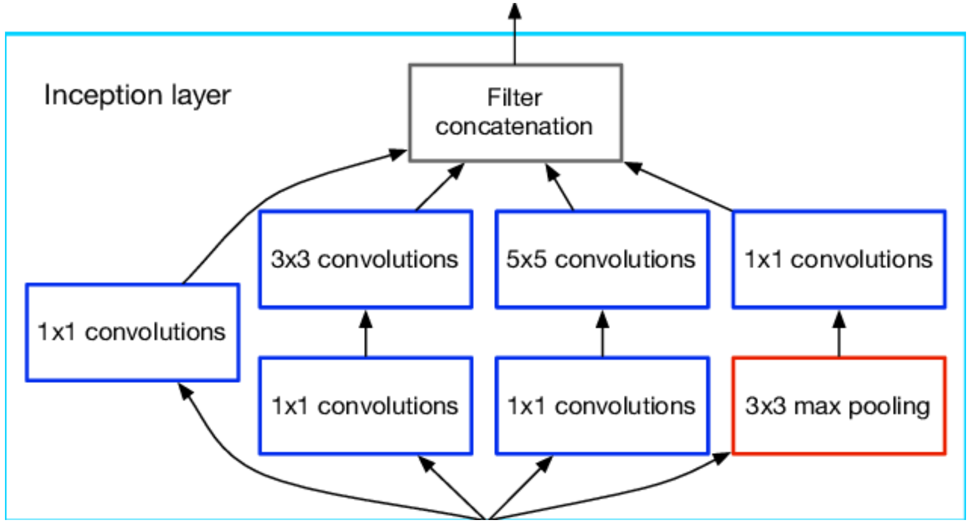
◈GoogLeNet (ILSVRC 2014 1st place, Google)

– Multi-Branch Convolutions
  - it simply concatenated multi-branch convolutions


Going deeper with convolutions

Christian Szegedy Google Inc.
Wei Liu University of North Carolina, Chapel Hill
Yangqing Jia Google Inc.
Pierre Sermanet Google Inc.
Scott Reed University of Michigan
Dragomir Anguelov Google Inc.
Dumitru Erhan Google Inc.
Vincent Vanhoucke Google Inc.
Andrew Rabinovich Google Inc.

– Main module: Inception
  - a novel component that allows the network to choose from different sizes of convolution filters (1x1, 3x3, 5x5) and a 3x3 max pooling at each layer of the network
  - This design enables the network to capture information at various scales and complexities, making it more efficient and powerful for feature extraction



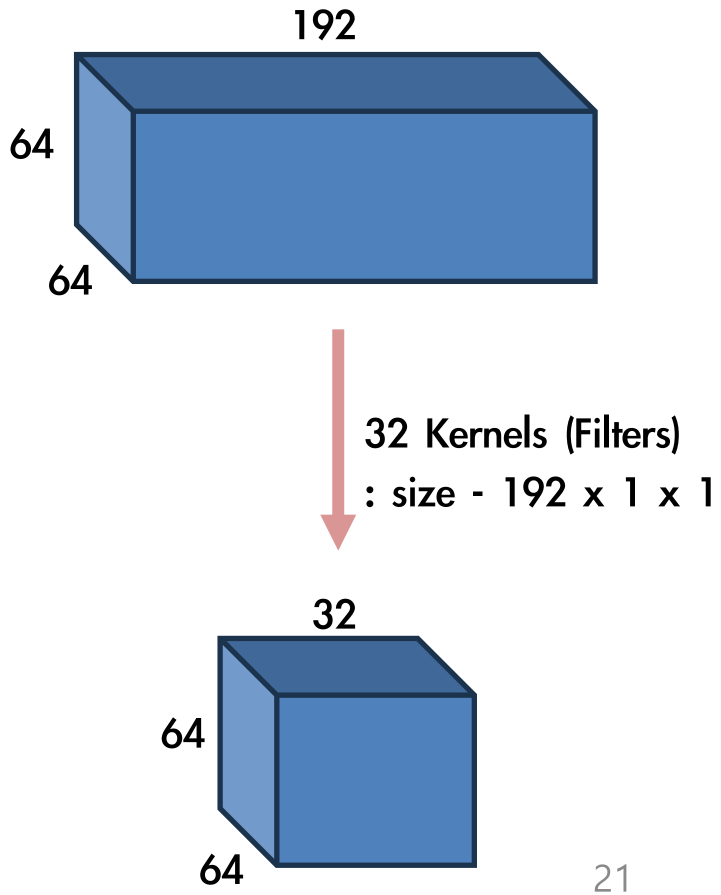
GoogLeNet

◈ 1x1 Convolution (Pointwise Convolution)

– Parameter Efficiency

- 1x1 convolutions can be used to design more parameter-efficient architectures

- By reducing the depth of the feature maps before applying expensive 3x3 or 5x5 convolutions, the total number of parameters in the network can be reduced, leading to faster computation and reduced risk of overfitting



GoogLeNet

◈ 1x1 Convolution (Pointwise Convolution)

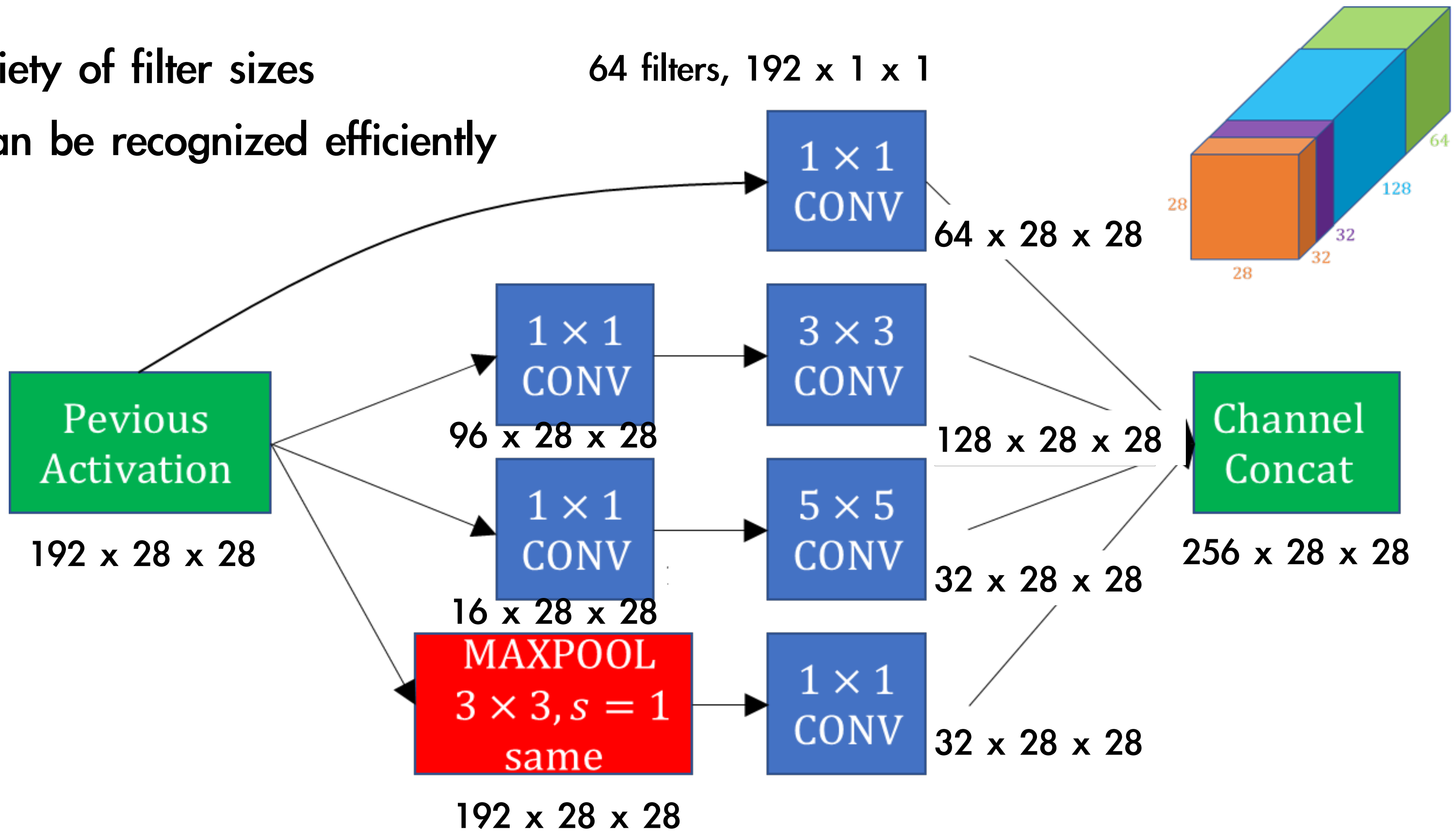
– Parameter Efficiency: Example with a feature map: 480 x 14 x 14

- Method 1
  - ➢ 5x5 conv
    - » 48 filters with 480 x 5 x 5 size (zero padding: 2, slide: 1) → output feature map: 48 x 14 x 14
    - » Number of parameters: (480 x 5 x 5 + 480) x 48 = 599,040
  - ➢ Total number of parameters: 599,040
- Method 2
  - ➢ 1x1 conv
    - » 16 filters with 480 x 1 x 1 size → output feature map: 16 x 14 x 14
    - » Number of parameters: (480 x 1 x 1 + 480) x 16 = 15,360
  - ➢ 5x5 conv
    - » 48 filters with 16 x 5 x 5 size (zero padding: 2, slide: 1) → output feature map: 48 x 14 x 14
    - » Number of parameters: (16 x 5 x 5 + 16) x 48 = 19,968
  - ➢ Total number of parameters: 15,360 + 19,969 = 35,329

GoogLeNet

◈Inception Block

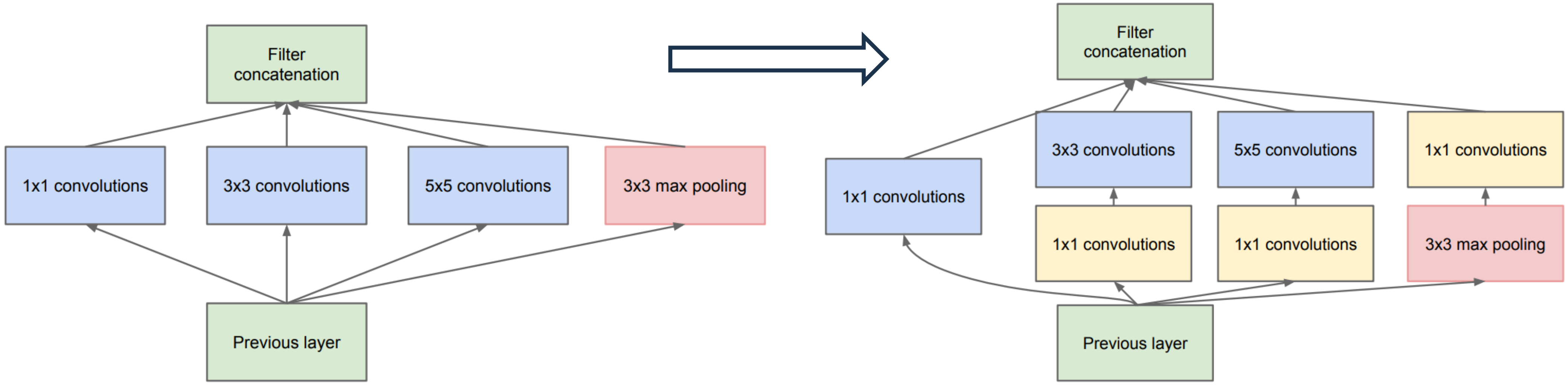
- The terminology stems from the meme "we need to go deeper" from the movie Inception
- four parallel branches
  - They explore the image in a variety of filter sizes
  - The details at different extents can be recognized efficiently by filters of different sizes



GoogLeNet

◈Inception Block

– Use 1x1 convolutions for dimensionality reduction before expensive convolutions



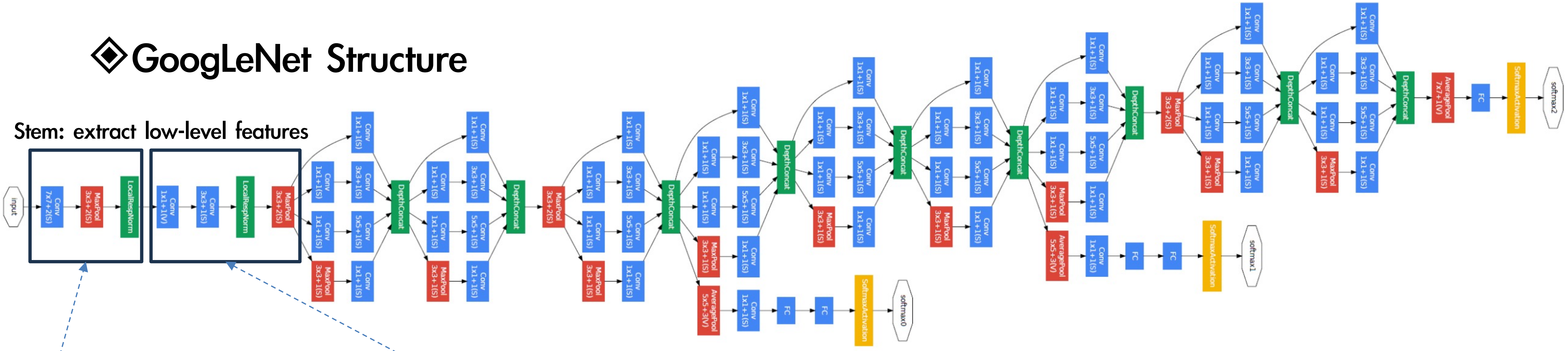
(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2: Inception module

GoogLeNet with PyTorch

◇GoogLeNet Structure

Stem: extract low-level features

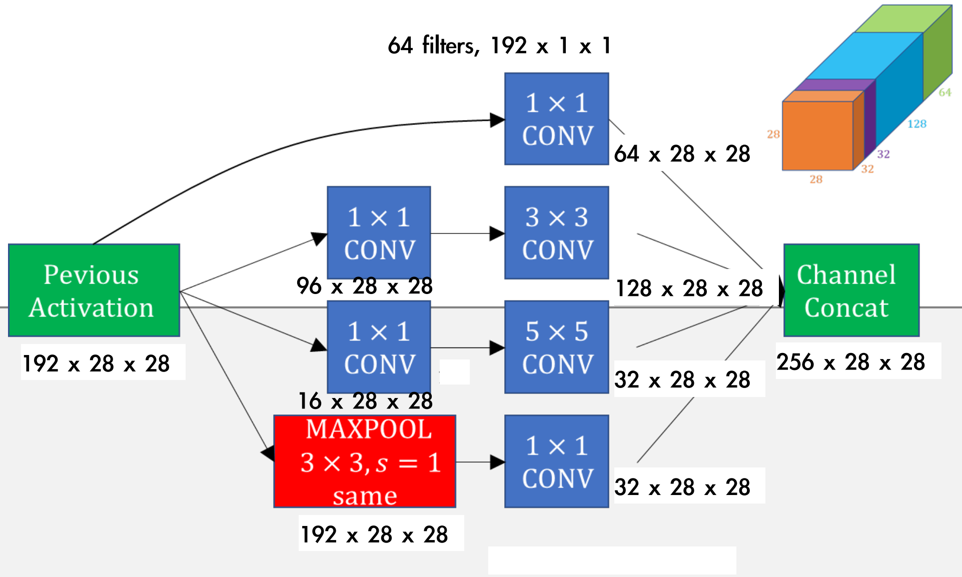


```python
def conv_blk_1(self):
  # LocalRespNorm is ignored
  return nn.Sequential(
    nn.LazyConv2d(
      out_channels=64, kernel_size=7,
      stride=2, padding=3
    ),
    nn.ReLU(),
    nn.MaxPool2d(
      kernel_size=3, stride=2, padding=1
    )
  )
```

```python
def conv_blk_2(self):
  # LocalRespNorm is ignored
  return nn.Sequential(
    nn.LazyConv2d(out_channels=64, kernel_size=1),
    nn.ReLU(),
    nn.LazyConv2d(
      out_channels=192, kernel_size=3, padding=1
    ),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
  )
```

link_dl/_01_code/_09_modern_cnns/_03_googlenet/a_cifar10_train_googlenet.py

GoogLeNet



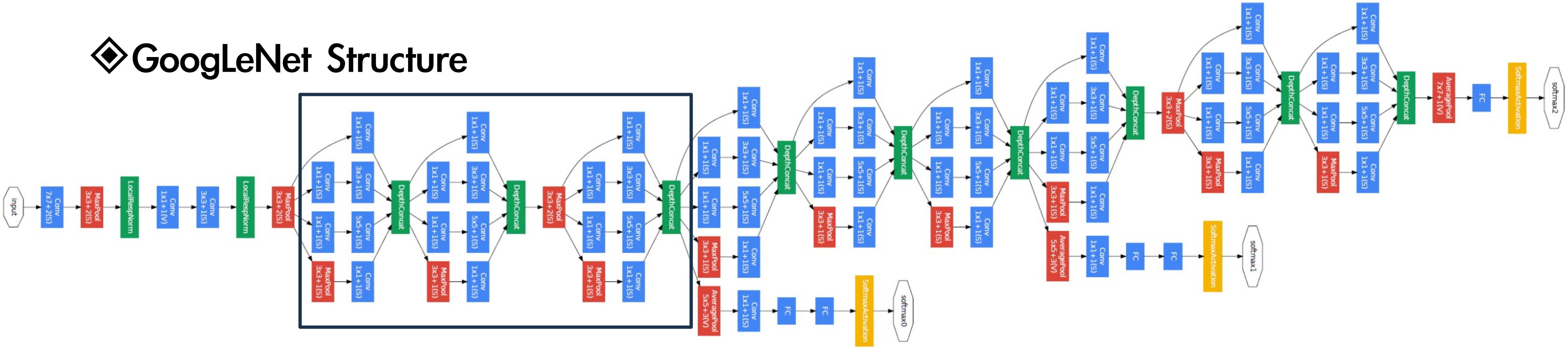
◈ GoogLeNet Structure

```python
class Inception(nn.Module):
  def __init__(self, c1, c2, c3, c4, **kwargs):
    super(Inception, self).__init__(**kwargs)
    # Branch 1
    self.b1_1 = nn.LazyConv2d(out_channels=c1, kernel_size=1)
    # Branch 2
    self.b2_1 = nn.LazyConv2d(out_channels=c2[0], kernel_size=1)
    self.b2_2 = nn.LazyConv2d(out_channels=c2[1], kernel_size=3, padding=1)
    # Branch 3
    self.b3_1 = nn.LazyConv2d(out_channels=c3[0], kernel_size=1)
    self.b3_2 = nn.LazyConv2d(out_channels=c3[1], kernel_size=5, padding=2)
    # Branch 4
    self.b4_1 = nn.MaxPool2d(kernel_size=3, stride=1, padding=1)
    self.b4_2 = nn.LazyConv2d(out_channels=c4, kernel_size=1)

  def forward(self, x):
    b1 = torch.relu(self.b1_1(x))
    b2 = torch.relu(self.b2_2(torch.relu(self.b2_1(x))))
    b3 = torch.relu(self.b3_2(torch.relu(self.b3_1(x))))
    b4 = torch.relu(self.b4_2(self.b4_1(x)))
    return torch.cat((b1, b2, b3, b4), dim=1)
```

link_dl/_01_code/_09_modern_cnns/_03_googlenet/a_cifar10_train_googlenet.py

GoogLeNet

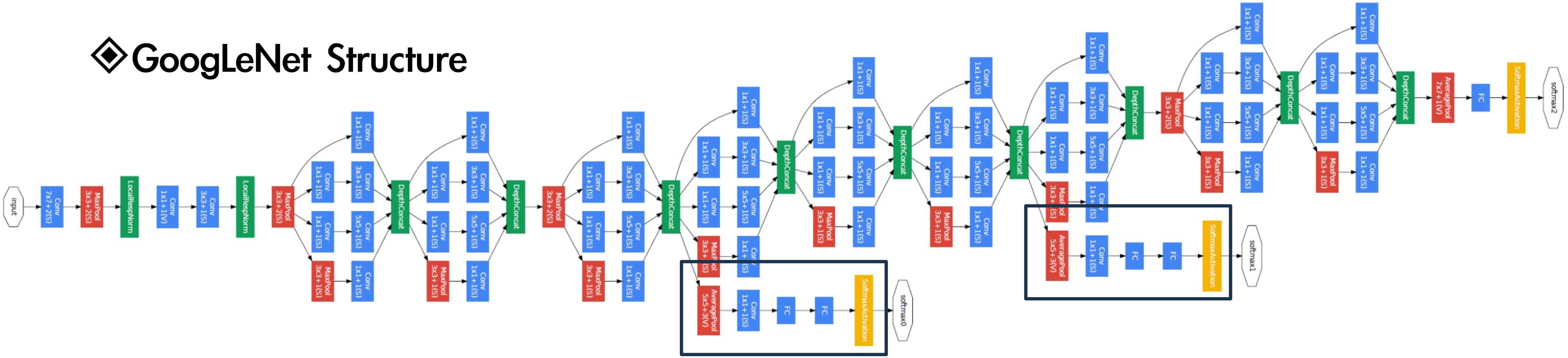
◈GoogLeNet Structure



```python
def inception_blk_1(self):
  return nn.Sequential(
    Inception(c1=64, c2=(96, 128), c3=(16, 32), c4=32),

    Inception(c1=128, c2=(128, 192), c3=(32, 96), c4=64),

    nn.MaxPool2d(kernel_size=3, stride=2, padding=1),

    Inception(c1=192, c2=(96, 208), c3=(16, 48), c4=64),
  )
```

GoogLeNet

◈GoogLeNet Structure



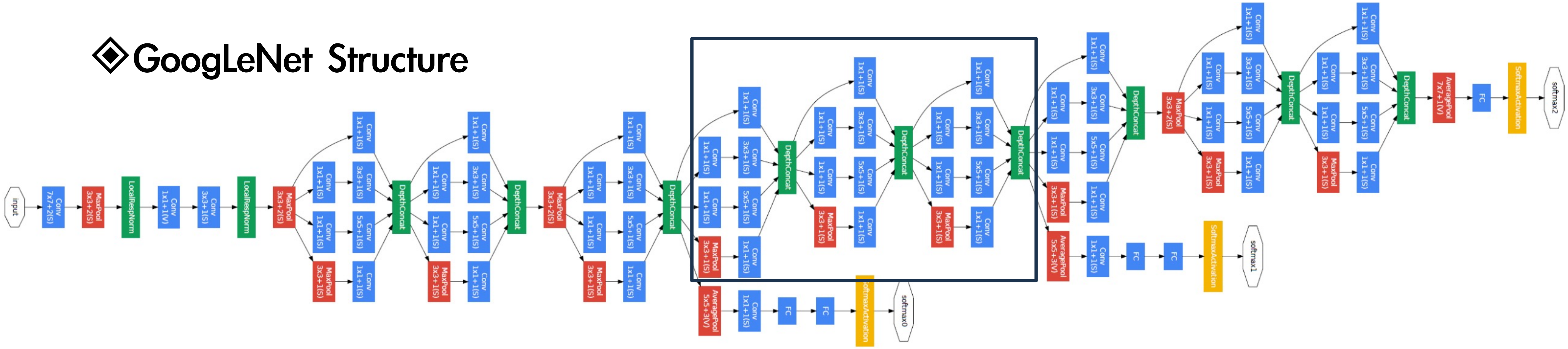
```
class InceptionAux(nn.Module):
  def __init__(self, n_outputs, **kwargs):
    super(InceptionAux, self).__init__(**kwargs)

    self.conv = nn.Sequential(
      nn.AvgPool2d(kernel_size=5, stride=3),
      nn.LazyConv2d(out_channels=128, kernel_size=1),
    )
```

```
class InceptionAux(nn.Module):
 def __init__(self, n_outputs, **kwargs):
    ...

    self.fc = nn.Sequential(
      nn.LazyLinear(out_features=1024),
      nn.ReLU(),
      nn.Dropout(),
      nn.LazyLinear(out_features=n_outputs),
    )
```

link_dl/_01_code/_09_modern_cnns/_03_googlenet/a_cifar10_train_googlenet.py

GoogLeNet

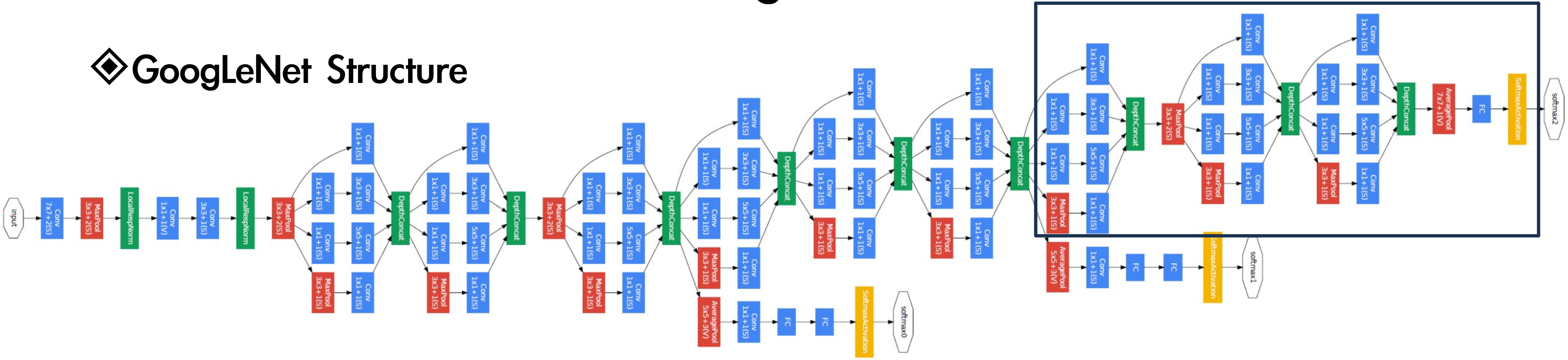
◈GoogLeNet Structure



```
def inception_blk_2(self):
  return nn.Sequential(
    Inception(c1=160, c2=(112, 224), c3=(24, 64), c4=64),

    Inception(c1=128, c2=(128, 256), c3=(24, 64), c4=64),

    Inception(c1=112, c2=(144, 288), c3=(32, 64), c4=64),
  )
```

GoogLeNet

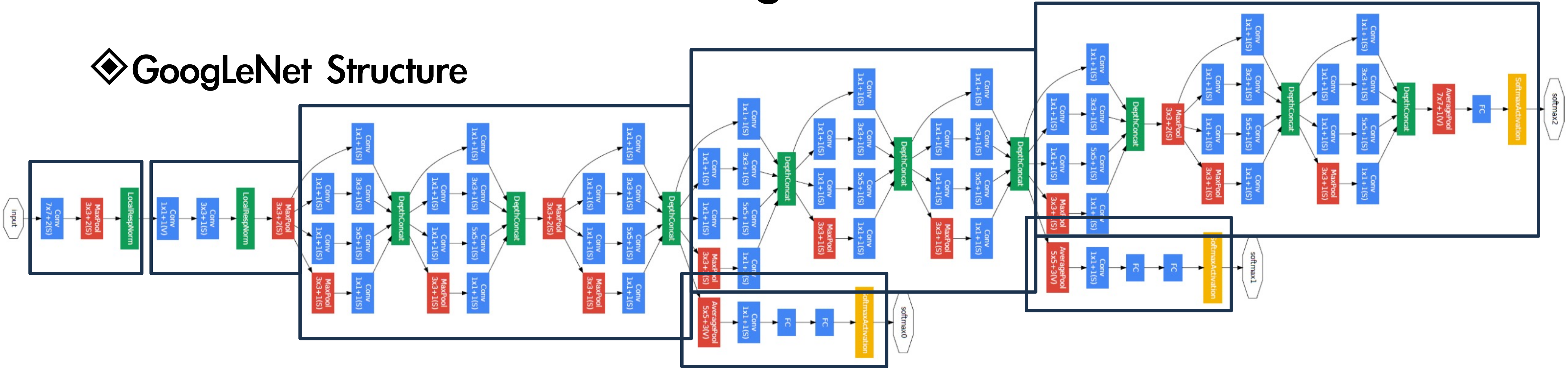
◈GoogLeNet Structure



```
def inception_blk_3(self):
  return nn.Sequential(
    Inception(c1=256, c2=(160, 320), c3=(32, 128), c4=128),
    nn.MaxPool2d(kernel_size=3, stride=2, padding=1),
    Inception(c1=256, c2=(160, 320), c3=(32, 128), c4=128),
    Inception(c1=384, c2=(192, 384), c3=(48, 128), c4=128),
    nn.AdaptiveAvgPool2d((1, 1)),
    nn.Flatten()
  )
```

GoogLeNet

◈GoogLeNet Structure



```python
class GoogleNet(nn.Module):
  def __init__(self, n_outputs=10):
    super(GoogleNet, self).__init__()
    self.conv_block = nn.Sequential(
      self.conv_blk_1(), self.conv_blk_2()
    )
    self.inception_block_1 = self.inception_blk_1()
    self.inception_block_2 = self.inception_blk_2()
    self.inception_block_3 = self.inception_blk_3()
    self.aux_1 = InceptionAux(n_outputs)
    self.aux_2 = InceptionAux(n_outputs)
```

```python
class GoogleNet(nn.Module):
  def forward(self, x):
    x = self.conv_block(x)
    inception_out_1 = self.inception_block_1(x)
    aux_out_1 = self.aux_1(inception_out_1)
    inception_out_2 =\
      self.inception_block_2(inception_out_1)
    aux_out_2 = self.aux_2(inception_out_2)
    inception_out_3 =\
      self.inception_block_3(inception_out_2)
  return inception_out_3, aux_out_1, aux_out_2
```

GoogLeNet Structure

Layer (type:depth-idx)	Kernel Shape	Input Shape	Output Shape	Param #	Mult-Adds
GoogleNet	--	[1, 3, 32, 32]	[1, 1024]	--	--
├─Sequential: 1-1	--	[1, 3, 32, 32]	[1, 192, 4, 4]	--	--
│ └─Sequential: 2-1	--	[1, 3, 32, 32]	[1, 64, 8, 8]	--	--
│ │ └─Conv2d: 3-1	[7, 7]	[1, 3, 32, 32]	[1, 64, 16, 16]	9,472	2,424,832
│ │ └─ReLU: 3-2	--	[1, 64, 16, 16]	[1, 64, 16, 16]	--	--
│ │ └─MaxPool2d: 3-3	3	[1, 64, 16, 16]	[1, 64, 8, 8]	--	--
│ └─Sequential: 2-2	--	[1, 64, 8, 8]	[1, 192, 4, 4]	--	--
│ │ └─Conv2d: 3-4	[1, 1]	[1, 64, 8, 8]	[1, 64, 8, 8]	4,160	266,240
│ │ └─ReLU: 3-5	--	[1, 64, 8, 8]	[1, 64, 8, 8]	--	--
│ │ └─Conv2d: 3-6	[3, 3]	[1, 64, 8, 8]	[1, 192, 8, 8]	110,784	7,090,176
│ │ └─ReLU: 3-7	--	[1, 192, 8, 8]	[1, 192, 8, 8]	--	--
│ │ └─MaxPool2d: 3-8	3	[1, 192, 8, 8]	[1, 192, 4, 4]	--	--
├─Sequential: 1-2	--	[1, 192, 4, 4]	[1, 512, 2, 2]	--	--
│ └─Inception: 2-3	--	[1, 192, 4, 4]	[1, 256, 4, 4]	--	--
│ │ └─Conv2d: 3-9	[1, 1]	[1, 192, 4, 4]	[1, 64, 4, 4]	12,352	197,632
│ │ └─Conv2d: 3-10	[1, 1]	[1, 192, 4, 4]	[1, 96, 4, 4]	18,528	296,448
│ │ └─Conv2d: 3-11	[3, 3]	[1, 96, 4, 4]	[1, 128, 4, 4]	110,720	1,771,520
│ │ └─Conv2d: 3-12	[1, 1]	[1, 192, 4, 4]	[1, 16, 4, 4]	3,088	49,408
│ │ └─Conv2d: 3-13	[5, 5]	[1, 16, 4, 4]	[1, 32, 4, 4]	12,832	205,312
│ │ └─MaxPool2d: 3-14	3	[1, 192, 4, 4]	[1, 192, 4, 4]	--	--
│ │ └─Conv2d: 3-15	[1, 1]	[1, 192, 4, 4]	[1, 32, 4, 4]	6,176	98,816
│ └─Inception: 2-4	--	[1, 256, 4, 4]	[1, 480, 4, 4]	--	--
│ │ └─Conv2d: 3-16	[1, 1]	[1, 256, 4, 4]	[1, 128, 4, 4]	32,896	526,336
│ │ └─Conv2d: 3-17	[1, 1]	[1, 256, 4, 4]	[1, 128, 4, 4]	32,896	526,336
│ │ └─Conv2d: 3-18	[3, 3]	[1, 128, 4, 4]	[1, 192, 4, 4]	221,376	3,542,016
│ │ └─Conv2d: 3-19	[1, 1]	[1, 256, 4, 4]	[1, 32, 4, 4]	8,224	131,584
│ │ └─Conv2d: 3-20	[5, 5]	[1, 32, 4, 4]	[1, 96, 4, 4]	76,896	1,230,336
│ │ └─MaxPool2d: 3-21	3	[1, 256, 4, 4]	[1, 256, 4, 4]	--	--
│ │ └─Conv2d: 3-22	[1, 1]	[1, 256, 4, 4]	[1, 64, 4, 4]	16,448	263,168
│ └─MaxPool2d: 2-5	3	[1, 480, 4, 4]	[1, 480, 2, 2]	--	--
│ └─Inception: 2-6	--	[1, 480, 2, 2]	[1, 512, 2, 2]	--	--
│ │ └─Conv2d: 3-23	[1, 1]	[1, 480, 2, 2]	[1, 192, 2, 2]	92,352	369,408
│ │ └─Conv2d: 3-24	[1, 1]	[1, 480, 2, 2]	[1, 96, 2, 2]	46,176	184,704
│ │ └─Conv2d: 3-25	[3, 3]	[1, 96, 2, 2]	[1, 208, 2, 2]	179,920	719,680
│ │ └─Conv2d: 3-26	[1, 1]	[1, 480, 2, 2]	[1, 16, 2, 2]	7,696	30,784
│ │ └─Conv2d: 3-27	[5, 5]	[1, 16, 2, 2]	[1, 48, 2, 2]	19,248	76,992
│ │ └─MaxPool2d: 3-28	3	[1, 480, 2, 2]	[1, 480, 2, 2]	--	--
│ │ └─Conv2d: 3-29	[1, 1]	[1, 480, 2, 2]	[1, 64, 2, 2]	30,784	123,136
├─InceptionAux: 1-3	--	[1, 512, 2, 2]	[1, 10]	--	--
│ └─Sequential: 2-7	--	[1, 512, 2, 2]	[1, 128, 2, 2]	--	--
│ │ └─Conv2d: 3-30	[1, 1]	[1, 512, 2, 2]	[1, 128, 2, 2]	65,664	262,656
│ └─Sequential: 2-8	--	[1, 512]	[1, 10]	--	--
│ │ └─Linear: 3-31	--	[1, 512]	[1, 1024]	525,312	525,312
│ │ └─ReLU: 3-32	--	[1, 1024]	[1, 1024]	--	--
│ │ └─Dropout: 3-33	--	[1, 1024]	[1, 1024]	--	--
│ │ └─Linear: 3-34	--	[1, 1024]	[1, 10]	10,250	10,250
├─Sequential: 1-4	--	[1, 512, 2, 2]	[1, 528, 2, 2]	--	--
│ └─Inception: 2-9	--	[1, 512, 2, 2]	[1, 512, 2, 2]	--	--
│ │ └─Conv2d: 3-35	[1, 1]	[1, 512, 2, 2]	[1, 160, 2, 2]	82,080	328,320
│ │ └─Conv2d: 3-36	[1, 1]	[1, 512, 2, 2]	[1, 112, 2, 2]	57,456	229,824
│ │ └─Conv2d: 3-37	[3, 3]	[1, 112, 2, 2]	[1, 224, 2, 2]	226,016	904,064
│ │ └─Conv2d: 3-38	[1, 1]	[1, 512, 2, 2]	[1, 24, 2, 2]	12,312	49,248
│ │ └─Conv2d: 3-39	[5, 5]	[1, 24, 2, 2]	[1, 64, 2, 2]	38,464	153,856
│ │ └─MaxPool2d: 3-40	3	[1, 512, 2, 2]	[1, 512, 2, 2]	--	--
│ │ └─Conv2d: 3-41	[1, 1]	[1, 512, 2, 2]	[1, 64, 2, 2]	32,832	131,328
│ └─Inception: 2-10	--	[1, 512, 2, 2]	[1, 512, 2, 2]	--	--
│ │ └─Conv2d: 3-42	[1, 1]	[1, 512, 2, 2]	[1, 128, 2, 2]	65,664	262,656
│ │ └─Conv2d: 3-43	[1, 1]	[1, 512, 2, 2]	[1, 128, 2, 2]	65,664	262,656
│ │ └─Conv2d: 3-44	[3, 3]	[1, 128, 2, 2]	[1, 256, 2, 2]	295,168	1,180,672
│ │ └─Conv2d: 3-45	[1, 1]	[1, 512, 2, 2]	[1, 24, 2, 2]	12,312	49,248
│ │ └─Conv2d: 3-46	[5, 5]	[1, 24, 2, 2]	[1, 64, 2, 2]	38,464	153,856
│ │ └─MaxPool2d: 3-47	3	[1, 512, 2, 2]	[1, 512, 2, 2]	--	--
│ │ └─Conv2d: 3-48	[1, 1]	[1, 512, 2, 2]	[1, 64, 2, 2]	32,832	131,328
│ └─Inception: 2-11	--	[1, 512, 2, 2]	[1, 528, 2, 2]	--	--
│ │ └─Conv2d: 3-49	[1, 1]	[1, 512, 2, 2]	[1, 112, 2, 2]	57,456	229,824
│ │ └─Conv2d: 3-50	[1, 1]	[1, 512, 2, 2]	[1, 144, 2, 2]	73,872	295,488
│ │ └─Conv2d: 3-51	[3, 3]	[1, 144, 2, 2]	[1, 288, 2, 2]	373,536	1,494,144
│ │ └─Conv2d: 3-52	[1, 1]	[1, 512, 2, 2]	[1, 32, 2, 2]	16,416	65,664
│ │ └─Conv2d: 3-53	[5, 5]	[1, 32, 2, 2]	[1, 64, 2, 2]	51,264	205,056
│ │ └─MaxPool2d: 3-54	3	[1, 512, 2, 2]	[1, 512, 2, 2]	--	--
│ │ └─Conv2d: 3-55	[1, 1]	[1, 512, 2, 2]	[1, 64, 2, 2]	32,832	131,328
├─InceptionAux: 1-5	--	[1, 528, 2, 2]	[1, 10]	--	--
│ └─Sequential: 2-12	--	[1, 528, 2, 2]	[1, 128, 2, 2]	--	--
│ │ └─Conv2d: 3-56	[1, 1]	[1, 528, 2, 2]	[1, 128, 2, 2]	67,712	270,848
│ └─Sequential: 2-13	--	[1, 512]	[1, 10]	--	--
│ │ └─Linear: 3-57	--	[1, 512]	[1, 1024]	525,312	525,312
│ │ └─ReLU: 3-58	--	[1, 1024]	[1, 1024]	--	--
│ │ └─Dropout: 3-59	--	[1, 1024]	[1, 1024]	--	--
│ │ └─Linear: 3-60	--	[1, 1024]	[1, 10]	10,250	10,250
├─Sequential: 1-6	--	[1, 528, 2, 2]	[1, 1024]	--	--
│ └─Inception: 2-14	--	[1, 528, 2, 2]	[1, 832, 2, 2]	--	--
│ │ └─Conv2d: 3-61	[1, 1]	[1, 528, 2, 2]	[1, 256, 2, 2]	135,424	541,696
│ │ └─Conv2d: 3-62	[1, 1]	[1, 528, 2, 2]	[1, 160, 2, 2]	84,640	338,560
│ │ └─Conv2d: 3-63	[3, 3]	[1, 160, 2, 2]	[1, 320, 2, 2]	461,120	1,844,480
│ │ └─Conv2d: 3-64	[1, 1]	[1, 528, 2, 2]	[1, 32, 2, 2]	16,928	67,712
│ │ └─Conv2d: 3-65	[5, 5]	[1, 32, 2, 2]	[1, 128, 2, 2]	102,528	410,112
│ │ └─MaxPool2d: 3-66	3	[1, 528, 2, 2]	[1, 528, 2, 2]	--	--
│ │ └─Conv2d: 3-67	[1, 1]	[1, 528, 2, 2]	[1, 128, 2, 2]	67,712	270,848
│ └─MaxPool2d: 2-15	3	[1, 832, 2, 2]	[1, 832, 1, 1]	--	--
│ └─Inception: 2-16	--	[1, 832, 1, 1]	[1, 832, 1, 1]	--	--
│ │ └─Conv2d: 3-68	[1, 1]	[1, 832, 1, 1]	[1, 256, 1, 1]	213,248	213,248
│ │ └─Conv2d: 3-69	[1, 1]	[1, 832, 1, 1]	[1, 160, 1, 1]	133,280	133,280
│ │ └─Conv2d: 3-70	[3, 3]	[1, 160, 1, 1]	[1, 320, 1, 1]	461,120	461,120
│ │ └─Conv2d: 3-71	[1, 1]	[1, 832, 1, 1]	[1, 32, 1, 1]	26,656	26,656
│ │ └─Conv2d: 3-72	[5, 5]	[1, 32, 1, 1]	[1, 128, 1, 1]	102,528	102,528
│ │ └─MaxPool2d: 3-73	3	[1, 832, 1, 1]	[1, 832, 1, 1]	--	--
│ │ └─Conv2d: 3-74	[1, 1]	[1, 832, 1, 1]	[1, 128, 1, 1]	106,624	106,624
│ └─Inception: 2-17	--	[1, 832, 1, 1]	[1, 1024, 1, 1]	--	--
│ │ └─Conv2d: 3-75	[1, 1]	[1, 832, 1, 1]	[1, 384, 1, 1]	319,872	319,872
│ │ └─Conv2d: 3-76	[1, 1]	[1, 832, 1, 1]	[1, 192, 1, 1]	159,936	159,936
│ │ └─Conv2d: 3-77	[3, 3]	[1, 192, 1, 1]	[1, 384, 1, 1]	663,936	663,936
│ │ └─Conv2d: 3-78	[1, 1]	[1, 832, 1, 1]	[1, 48, 1, 1]	39,984	39,984
│ │ └─Conv2d: 3-79	[5, 5]	[1, 48, 1, 1]	[1, 128, 1, 1]	153,728	153,728
│ │ └─MaxPool2d: 3-80	3	[1, 832, 1, 1]	[1, 832, 1, 1]	--	--
│ │ └─Conv2d: 3-81	[1, 1]	[1, 832, 1, 1]	[1, 128, 1, 1]	106,624	106,624
│ └─AdaptiveAvgPool2d: 2-18	--	[1, 1024, 1, 1]	[1, 1024, 1, 1]	--	--
│ └─Flatten: 2-19	--	[1, 1024, 1, 1]	[1, 1024]	--	--

Total params: 7,178,052
Trainable params: 7,178,052
Non-trainable params: 0
Total mult-adds (M): 33.95

Input size (MB): 0.01
Forward/backward pass size (MB): 0.55
Params size (MB): 28.71
Estimated Total Size (MB): 29.28

GoogLeNet

◈GoogLeNet Train (1/2)

```
def do_train(self):
  self.model.train()  # Explained at 'Diverse Techniques' section

  loss_train = 0.0
  num_corrects_train = 0
  num_trained_samples = 0
  num_trains = 0

  for train_batch in self.train_data_loader:
    input_train, target_train = train_batch
    input_train = input_train.to(device=self.device)
    target_train = target_train.to(device=self.device)

    input_train = self.transforms(input_train)

    output_train, output_train_ax_1, output_train_ax_2 = self.model(input_train)
    loss = self.loss_fn(output_train, target_train)
    loss_aux_1 = self.loss_fn(output_train_ax_1, target_train)
    loss_aux_2 = self.loss_fn(output_train_ax_2, target_train)
    loss += 0.3 * (loss_aux_1 + loss_aux_2)
    loss_train += loss.item()
```

GoogLeNet

◈GoogLeNet Train (2/2)

```
def do_train(self):
  ...

  for train_batch in self.train_data_loader:
    ...
    predicted_train = torch.argmax(output_train, dim=1)
    num_corrects_train += torch.sum(torch.eq(predicted_train, target_train)).item()

    num_trained_samples += len(input_train)
    num_trains += 1

    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()

  train_loss = loss_train / num_trains
  train_accuracy = 100.0 * num_corrects_train / num_trained_samples

  return train_loss, train_accuracy
```

GoogLeNet

◈GoogLeNet Validation (1/2)

```
def do_validation(self):
  self.model.eval()   # Explained at 'Diverse Techniques' section
  loss_validation = 0.0
  num_corrects_validation = 0
  num_validated_samples = 0
  num_validations = 0

  with torch.no_grad():
    for validation_batch in self.validation_data_loader:
      input_validation, target_validation = validation_batch
      input_validation = input_validation.to(device=self.device)
      target_validation = target_validation.to(device=self.device)

      input_validation = self.transforms(input_validation)

      output_validation, output_validation_ax_1, output_validation_ax_2 = self.model(input_validation)
      loss_validation = self.loss_fn(output_validation, target_validation)
      loss_validation_aux_1 = self.loss_fn(output_validation_ax_1, target_validation)
      loss_validation_aux_2 = self.loss_fn(output_validation_ax_2, target_validation)
      loss_validation += 0.3 * (loss_validation_aux_1 + loss_validation_aux_2)
      loss_validation += loss_validation.item()
```

GoogLeNet

◈GoogLeNet Validation (2/2)

```
def do_validation(self):
  ...

  with torch.no_grad():
    for validation_batch in self.validation_data_loader:
      ...

      predicted_validation = torch.argmax(output_validation, dim=1)
      num_corrects_validation += torch.sum(torch.eq(predicted_validation, target_validation)).item()

      num_validated_samples += len(input_validation)
      num_validations += 1

  validation_loss = loss_validation / num_validations
  validation_accuracy = 100.0 * num_corrects_validation / num_validated_samples

  return validation_loss, validation_accuracy
```

ResNet

ResNet

◈ Deep Residual Learning (ResNet) [Microsoft, 2015]

– One of the most popular off-the-shelf architectures in computer vision

Deep Residual Learning for Image Recognition

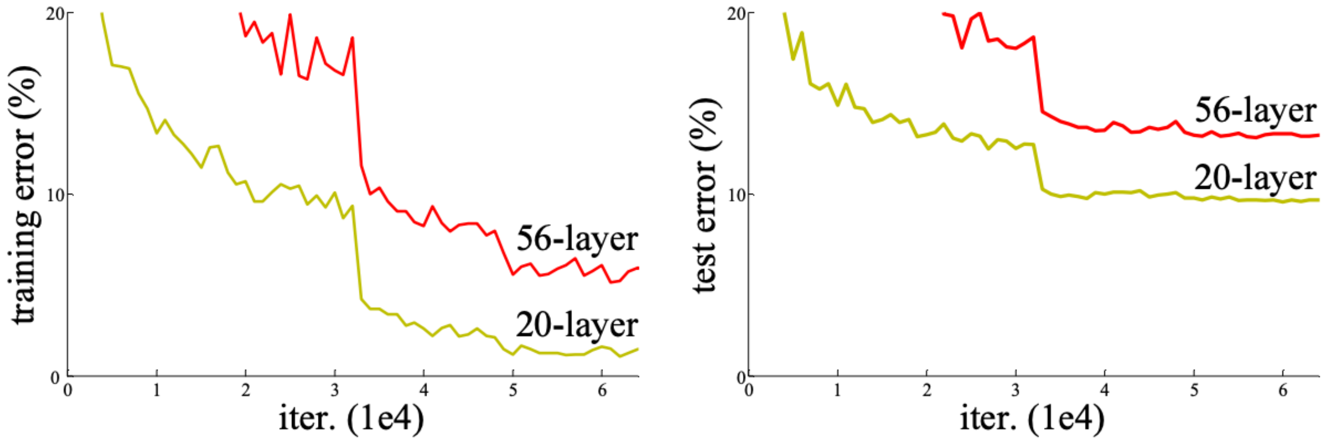
Kaiming He   Xiangyu Zhang   Shaoqing Ren   Jian Sun

Microsoft Research

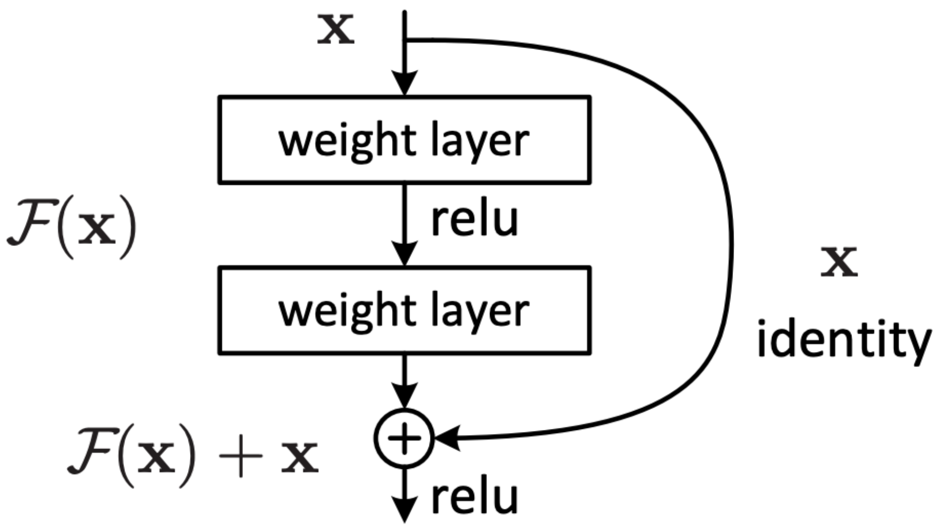
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

– Motivation of ResNet

- Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks
- The deeper network has higher training error, and thus test error



ResNet



◈ <u>Residual Learning</u> is easy to be trained

– Instead of hoping each few stacked layers directly fit a desired underlying mapping, we explicitly let these layers fit a residual mapping

– $H(x)$: Desired underlying mapping

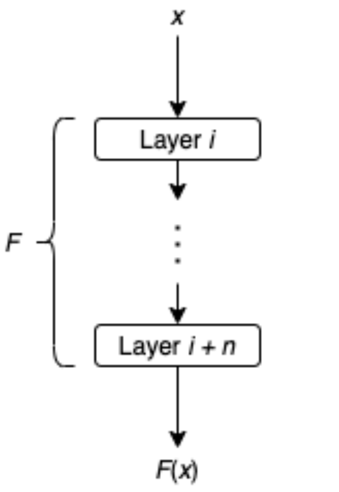
Residual

$$H(x) = x + F(x)$$

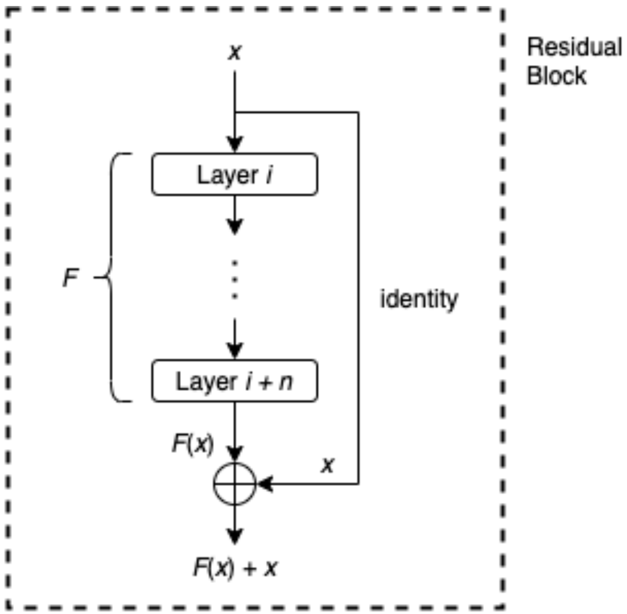
- It is easier to optimize the residual mapping than to optimize the original, unreferenced mapping

$$H(x) = F(x) \qquad \text{vs.} \qquad H(x) = x + F(x)$$

- Why?



Traditional Feedforward without Residual Connection



With Residual Connection

ResNet

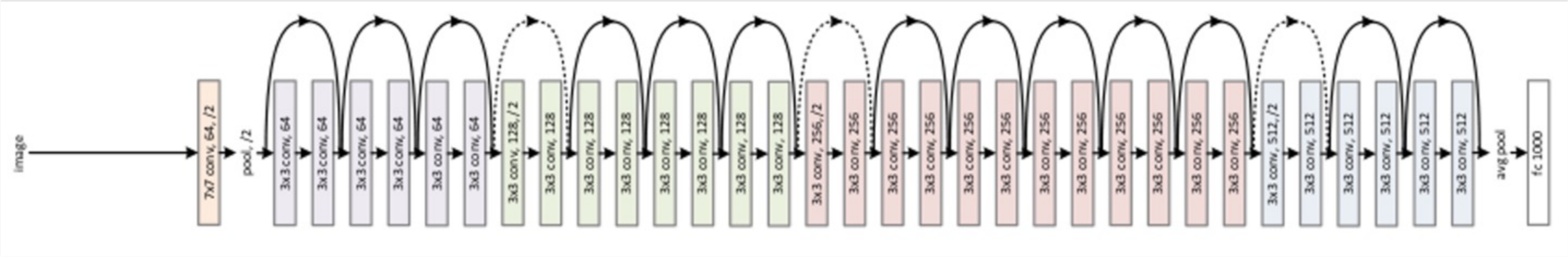
[Our Interest]
$H(x) = F(x)$ vs. $H(x) = x + F(x)$

◈ "Desirable" mapping for ONE layer

– Identity Mapping

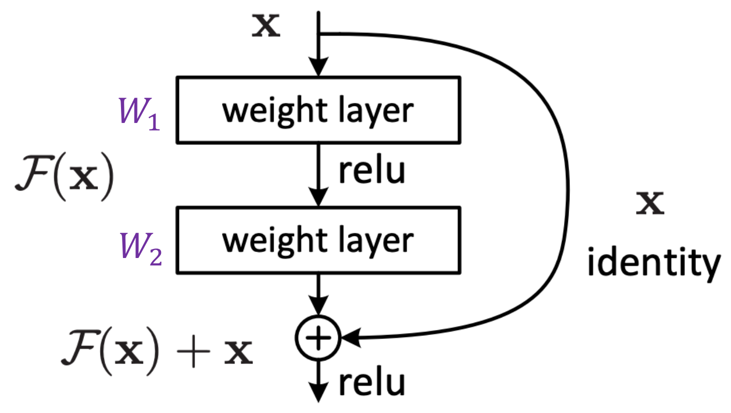
$$H(x) \approx x$$

– The deeper the network, the better it is to change values gradually and very little from the input



- Rather than trying to do everything in one layer, each layer plays its role <u>little by little</u> to achieve the given purpose.

ResNet



◈For the identity mapping, what is easier?

- Let's assume $F(x) \approx \sigma(xW_1)W_2$
- Without skip-connection, we will train W_1 and W_2 to satisfy

$$\underline{H(x) = F(x) = \sigma(xW_1)W_2 \approx x}$$

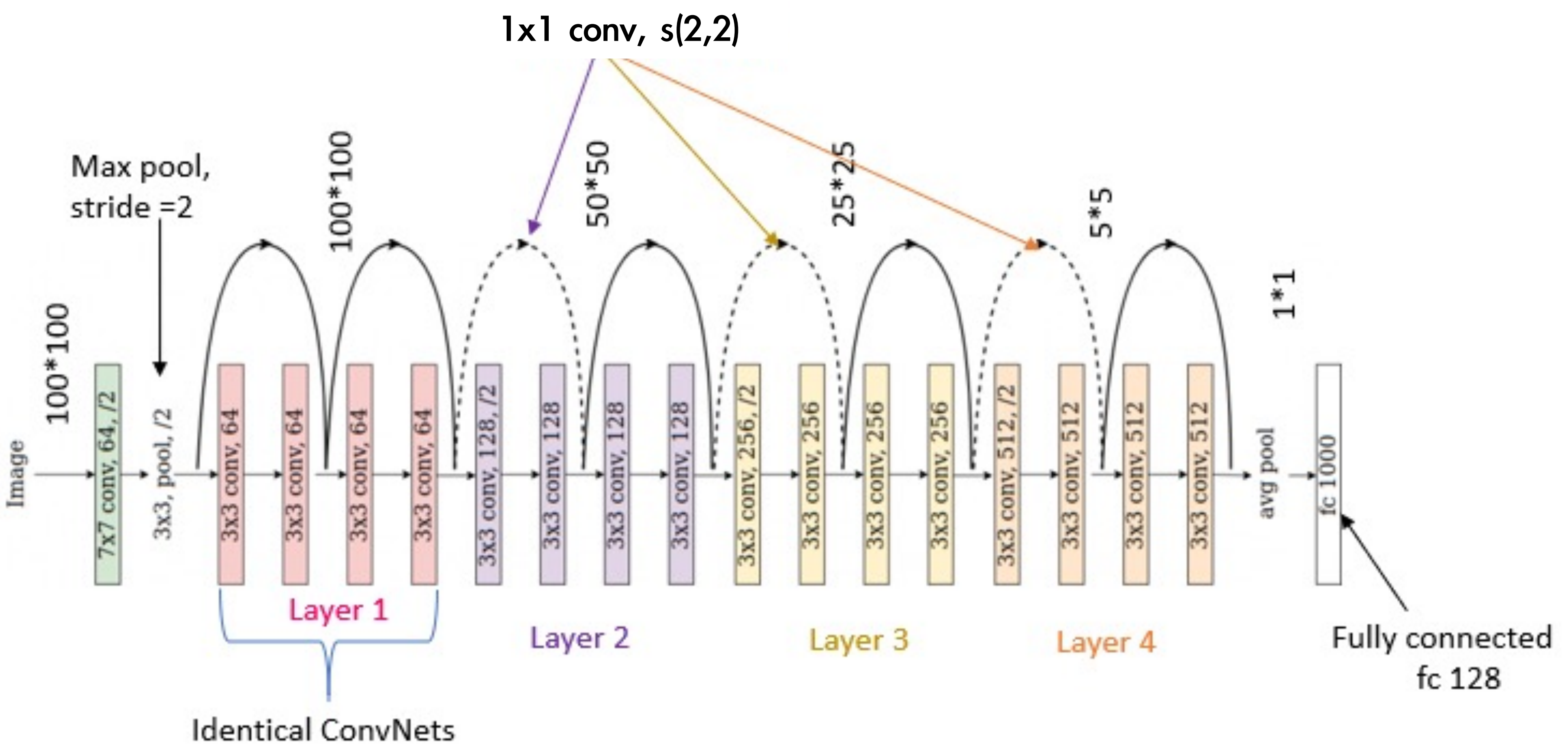
  - If σ is ReLU, $\sigma(xW_1)W_2 = xW_1W_2$ if $xW_1 \geq 0$ → $xW_1W_2 \approx x$
  - Then, we need to train them so that $W_1 \approx I$ and $W_2 \approx I$ (Identity (or Orthogonal) Matrix)
- With a skip-connection, we will train W_1 and W_2 to satisfy

$$\underline{H(x) = x + F(x) = x + \sigma(xW_1)W_2 \approx x}$$

  - If σ is ReLU, $\sigma(xW_1)W_2 = xW_1W_2$ if $xW_1 \geq 0$ → $x + xW_1W_2 \approx x$
  - Then, we need to train them so that $W_1 \approx 0$ and $W_2 \approx 0$ (Zero Matrix)
- It would be easier to <u>push the residual to zero</u> than to fit into Identity (or orthogonal) matrix
  - since weights are usually initialized with samples ~ $N(0, \alpha)$

ResNet with PyTorch

◈ResNet-18 Structure



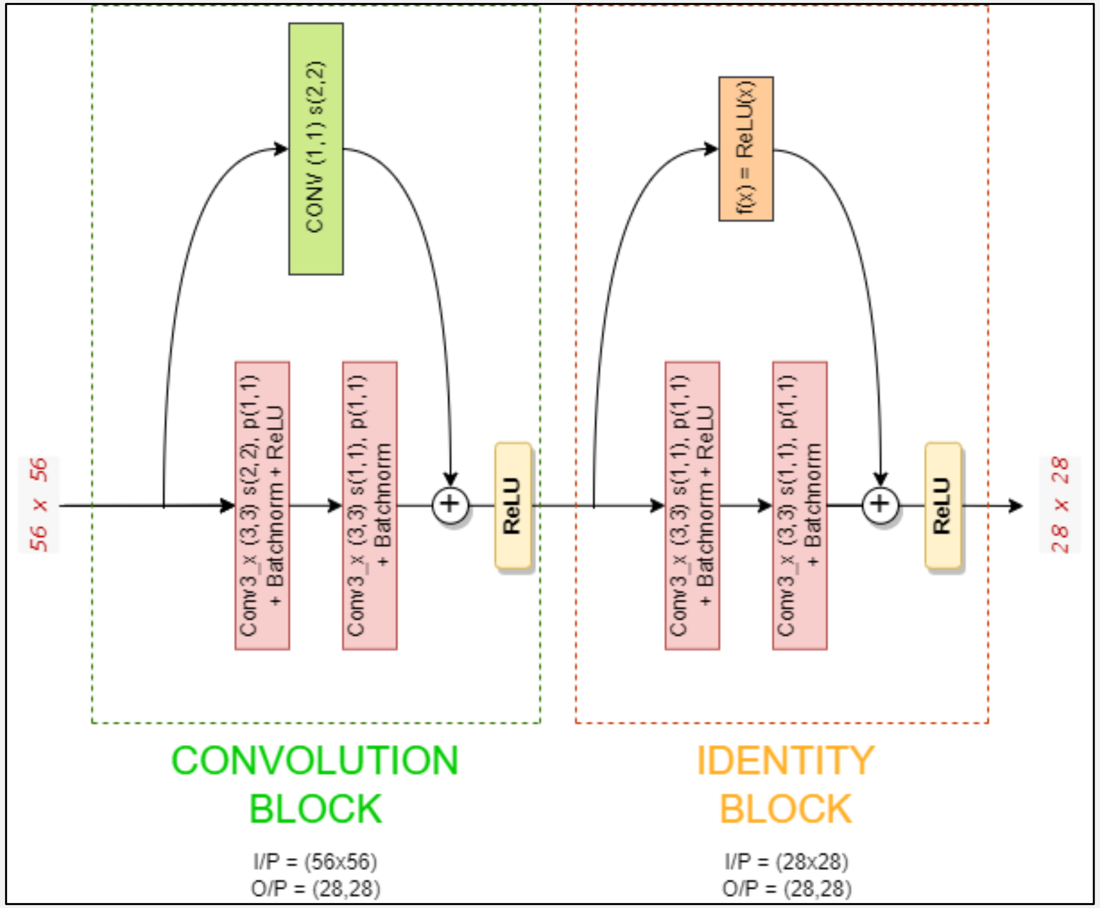
ResNet-18 Architecture

Fruit 360 Input Image size= 100*100 px

ResNet with PyTorch

link_dl/_01_code/_09_modern_cnns/_04_resnet/a_cifar10_train_resnet.py

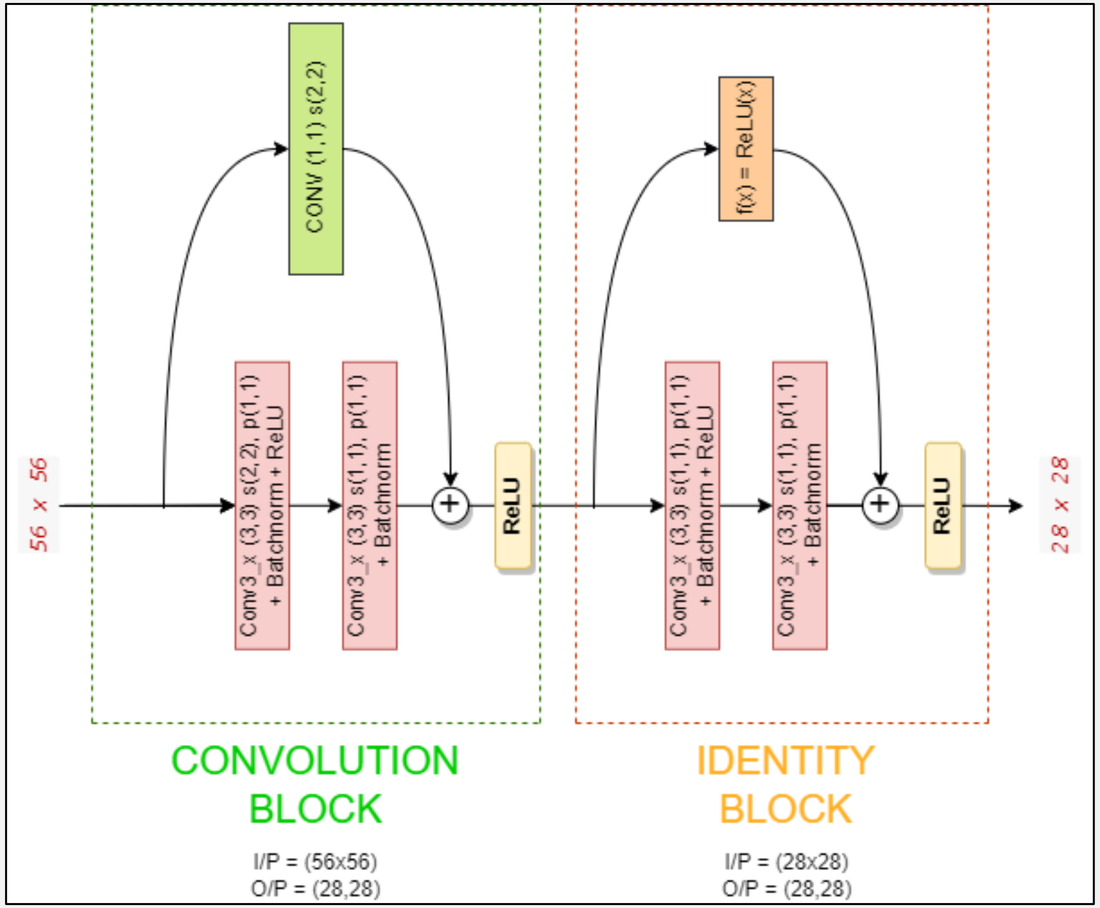
```
class Residual(nn.Module):
  """The Residual block of ResNet models."""
  def __init__(self, num_channels, use_1x1conv=False, strides=1):
      super().__init__()
      self.conv1 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1, stride=strides)
      self.conv2 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1)
      if use_1x1conv:
        self.conv3 = nn.LazyConv2d(
          num_channels, kernel_size=1, stride=strides
        )
      else:
          self.conv3 = None
      self.bn1 = nn.LazyBatchNorm2d()
      self.bn2 = nn.LazyBatchNorm2d()

  def forward(self, X):
      Y = torch.relu(self.bn1(self.conv1(X)))
      Y = self.bn2(self.conv2(Y))
      if self.conv3:
          X = self.conv3(X)
      Y += X
      return torch.relu(Y)
```



link_dl/_01_code/_09_modern_cnns/_04_resnet/a_cifar10_train_resnet.py

ResNet with PyTorch

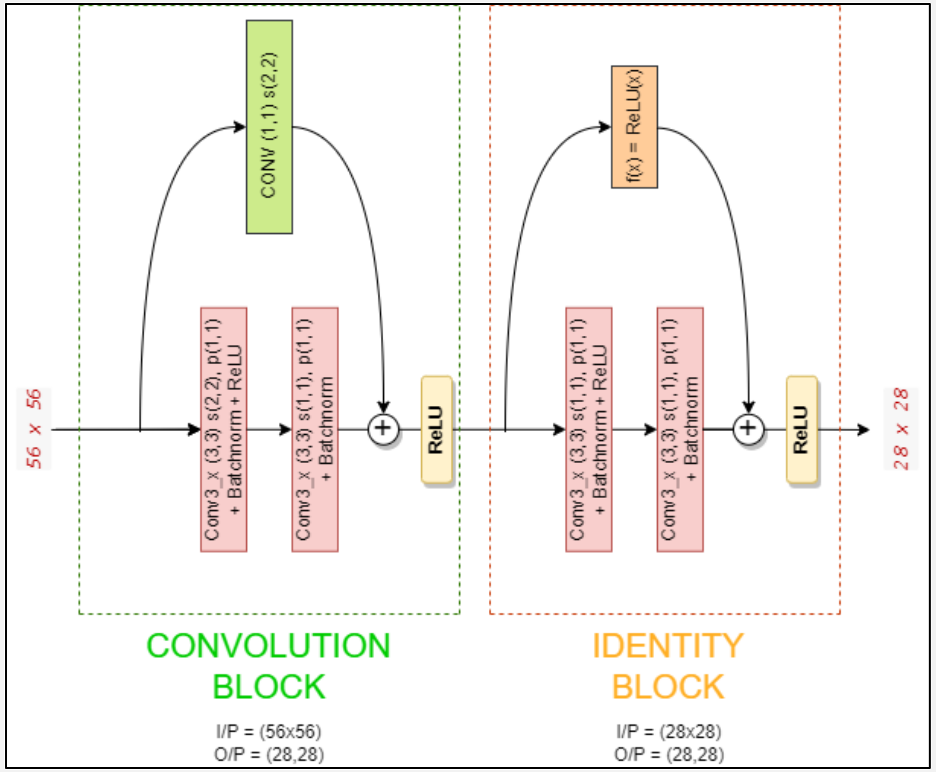
```python
class ResNet(nn.Module):
  def __init__(self, arch, n_outputs=10):
    super(ResNet, self).__init__()
    self.model = nn.Sequential(
      nn.Sequential(
        nn.LazyConv2d(
          out_channels=64, kernel_size=7, stride=2,
          padding=3
        ),
        nn.LazyBatchNorm2d(),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
      )
    )

    for i, b in enumerate(arch):
      self.model.add_module(
        name=f'b{i + 2}’,
        module=self.block(*b, first_block=(i == 0))
      )
```



◈ResNet

ResNet with PyTorch

```
class ResNet(nn.Module):
  def __init__(self, arch, n_outputs=10):
    ...
    self.model.add_module(
      name='last',
      module=nn.Sequential(
        nn.AdaptiveAvgPool2d((1, 1)),
        nn.Flatten(),
        nn.LazyLinear(n_outputs)
      )
    )

  def block(self, num_residuals, num_channels, first_block=False):
    blk = []
    for i in range(num_residuals):
      if i == 0 and not first_block:
        blk.append(Residual(num_channels=num_channels, use_1x1conv=True, strides=2))
      else:
        blk.append(Residual(num_channels=num_channels))
    return nn.Sequential(*blk)
```

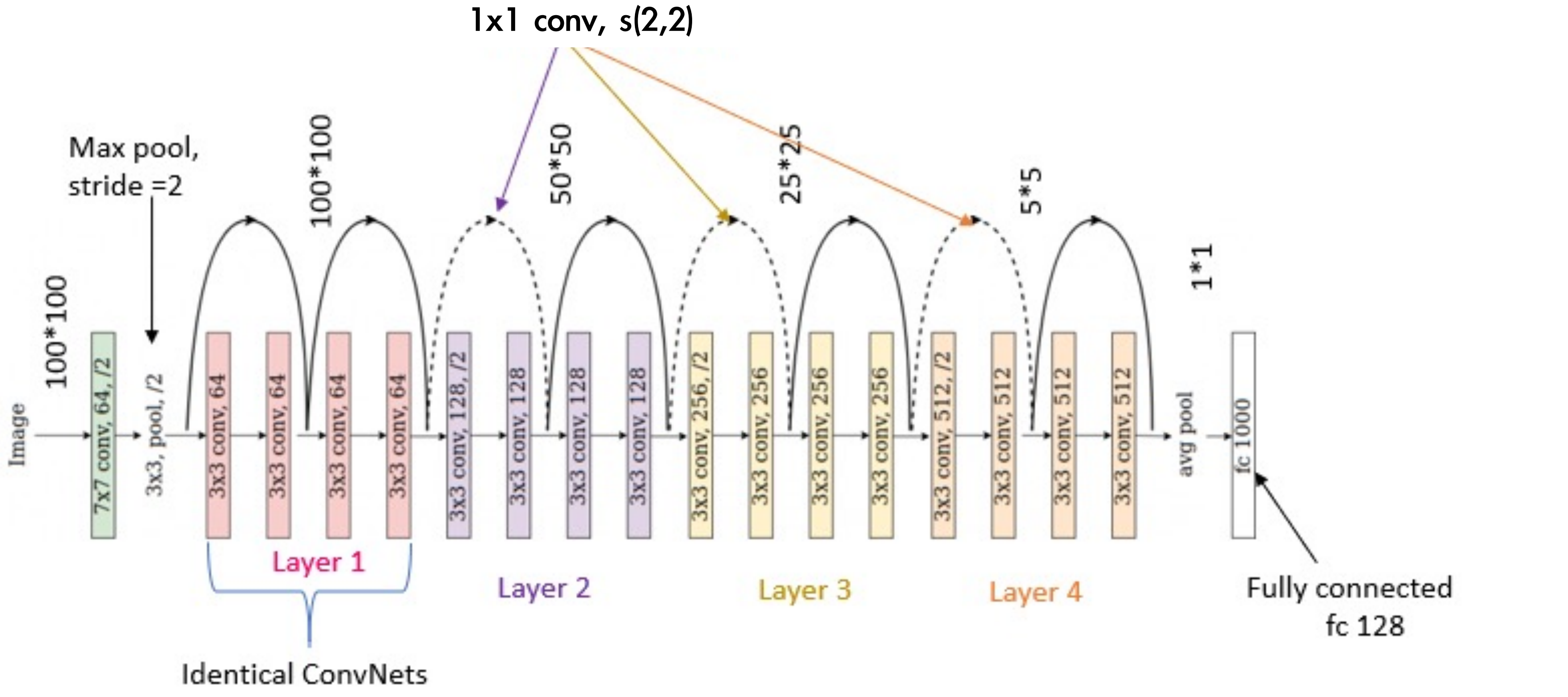


link_dl/_01_code/_09_modern_cnns/_04_resnet/a_cifar10_train_resnet.py

ResNet with PyTorch

◈ResNet

```
class ResNet(nn.Module):
  ...
  def forward(self, x):
    x = self.model(x)
    return x

def get_resnet_model():
  my_model = ResNet(
    arch=(
      (2, 64),
      (2, 128),
      (2, 256),
      (2, 512)
    ),
    n_outputs=10
  )

  return my_model
```



ResNet-18 Architecture

Fruit 360 Input Image size= 100*100 px

Torchvision Package - Overview

- torchvision.datasets
  - MNIST
  - Fashion-MNIST
  - KMNIST
  - EMNIST
  - FakeData
  - COCO
  - LSUN
  - ImageFolder
  - DatasetFolder
  - Imagenet-12
  - CIFAR10
  - CIFAR100
  - STL10
  - SVHN
  - PhotoTour
  - SBU
  - Flickr
  - VOC
  - Cityscapes

Torchvision Package - Overview

- torchvision.models
  - Alexnet
  - VGG
  - ResNet
  - Inceptionv3
  - GoogLeNet
  - SqueezeNet
  - DenseNet

- torchvision.transforms
  - Transforms on PIL Image
  - Transforms on torch.*Tensor
  - Conversion Transforms
  - Generic Transforms
  - Functional Transforms

- torchvision.utils