Institute of Electronics
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

AI Training Course Series

CNN-Based Image Classification

Lecture 4

Architecture Development & Algorithm Refinement for Intelligent Computing

Student: Ting-Yu Chang
Student: Lyu-Ming Ho
Advisor: Juinn-Dar Huang, Ph.D.

July 18, 2024

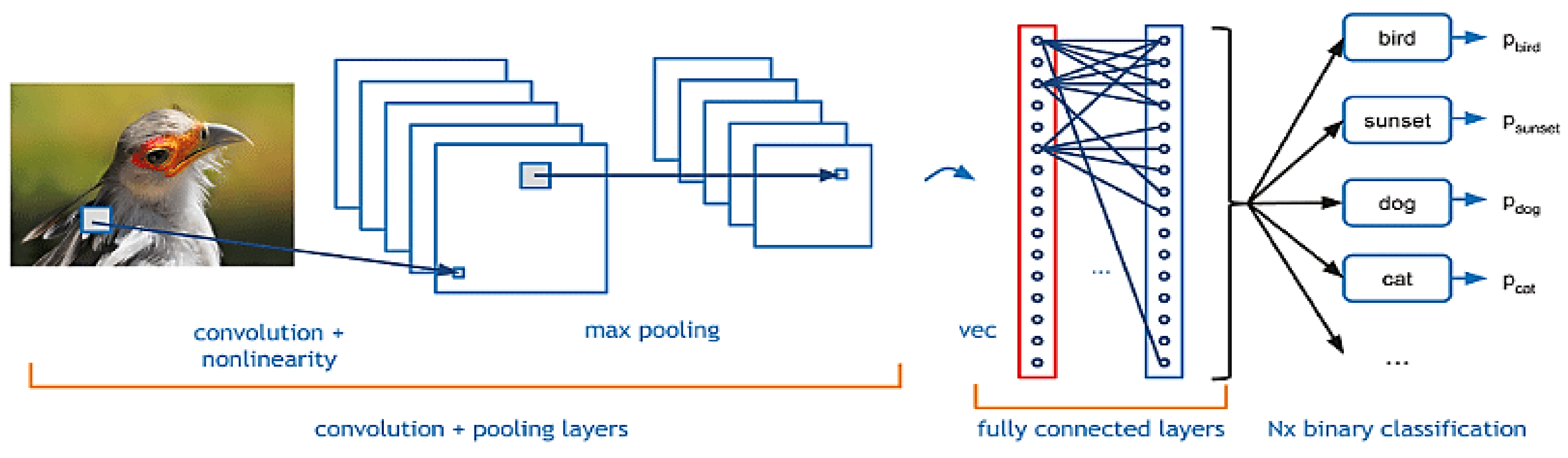
Outline

- Computer vision
- Well-known CNN models
- CNN architecture search
  (Network Architecture Search, NAS)
- Exercise

Computer Vision

Image Classification

- Image Classification
  - Datasets (CIFAR-10, CIFAR-100, ImageNet)
  - Models (VGG, ResNet, HarDNet)



Object Detection

- Object Detection
  - Dataset (PASCAL VOC, Coco)
  - Models (RCNN, SSD, YOLO)

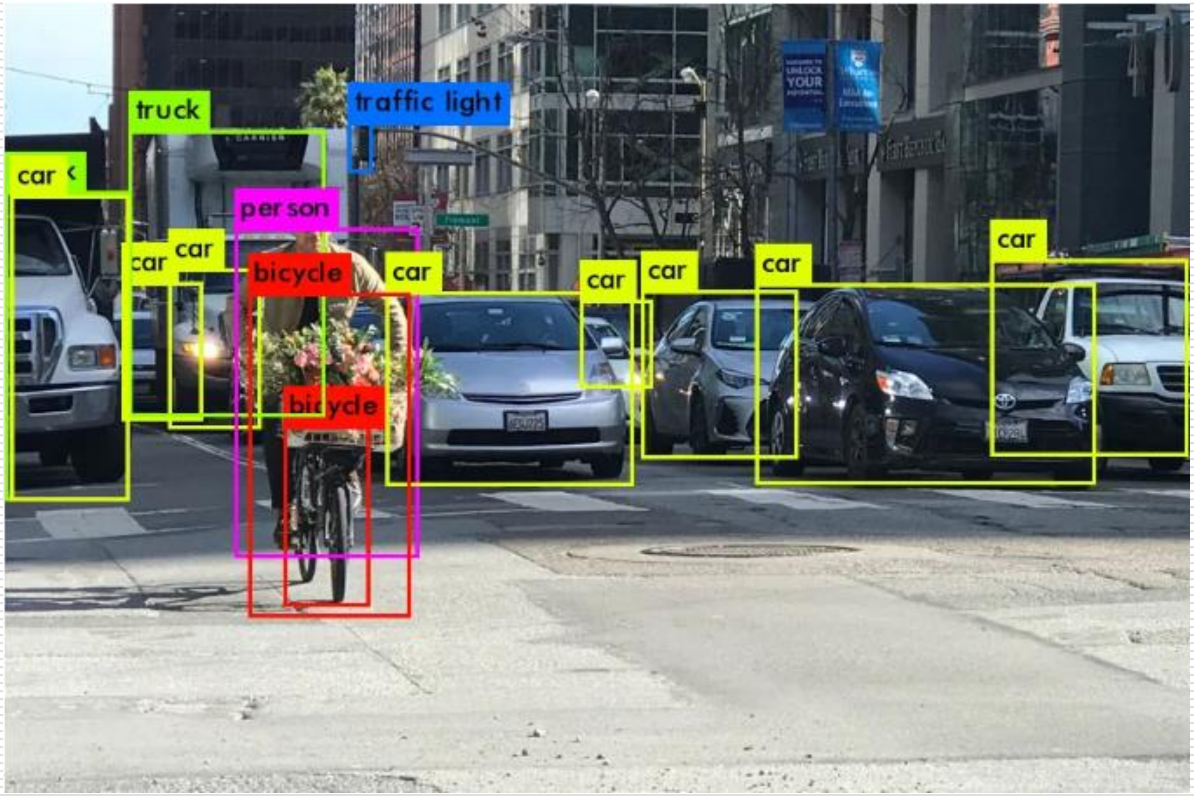


Image Semantic Segmentation

- Image Semantic Segmentation
  - Datasets (Cityscapes, PASCAL VOC)
  - Models (U-Net, FC-HarDNet)

Multi-Object Tracking (MOT)

- Multi-Object Tracking
  - Dataset (MOT17)
  - Models (TrackFormer, FairMOT)

Frame t   Frame t+1

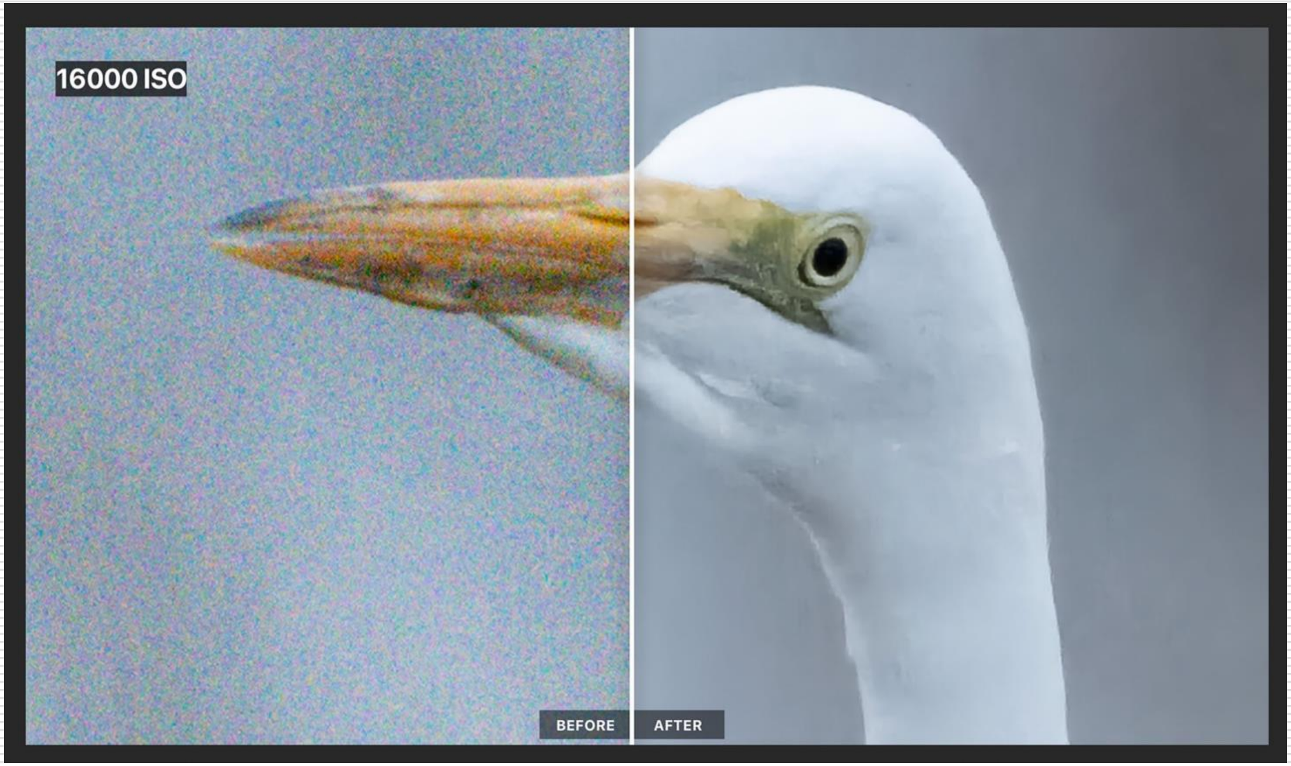
Super Resolution (SR)

- Image Super Resolution
  - Dataset (Set5, Set12)
  - Models (Swin transformer, hybrid-attention transformer)

- Upscale original resolution of images to higher resolution
- Ex. 240 X 240 → 1080 X 1080

Noise Reduction (NR)

- Image Noise Reduction (image denoising)
  - Dataset (SIDD)
  - Models (Restormer, U-former...)



CNN Models

ImageNet

- A large visual database designed for use in visual object recognition software research
  - More than 14 million images with more than 20,000 categories
  - 2010 ~ 2017 ILSVRC

- ILSVRC2012
  - Training set 1,281,167 images
  - Validation set 50,000 images
  - Test set 100,000 images

LeNet

- A convolutional neural network structure proposed by Yann LeCun et al. in 1989
- 5 CONV layers + 2 FC layers
- Handwriting recognition

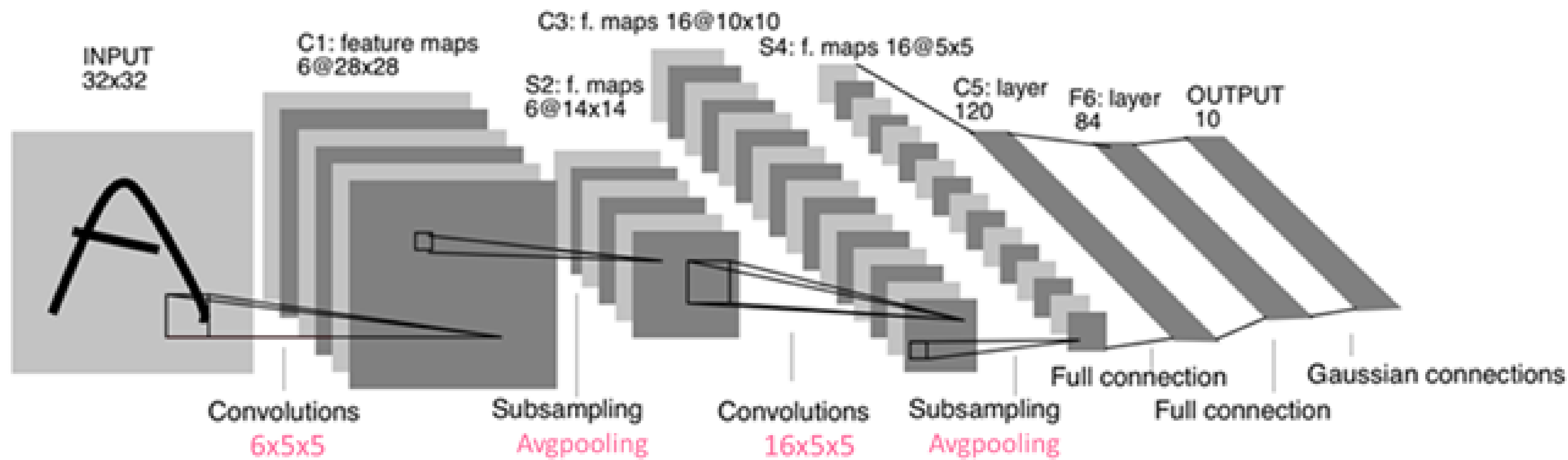
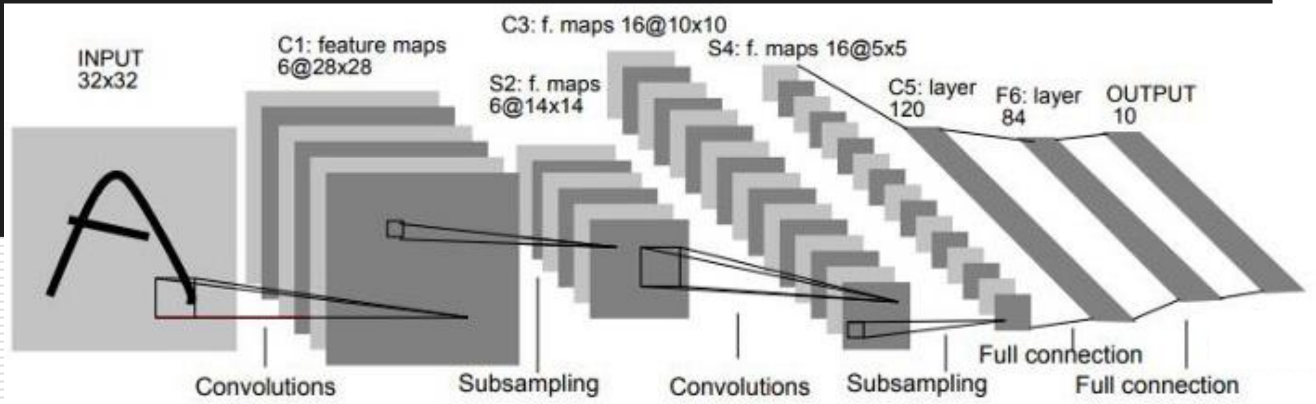


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet – Code

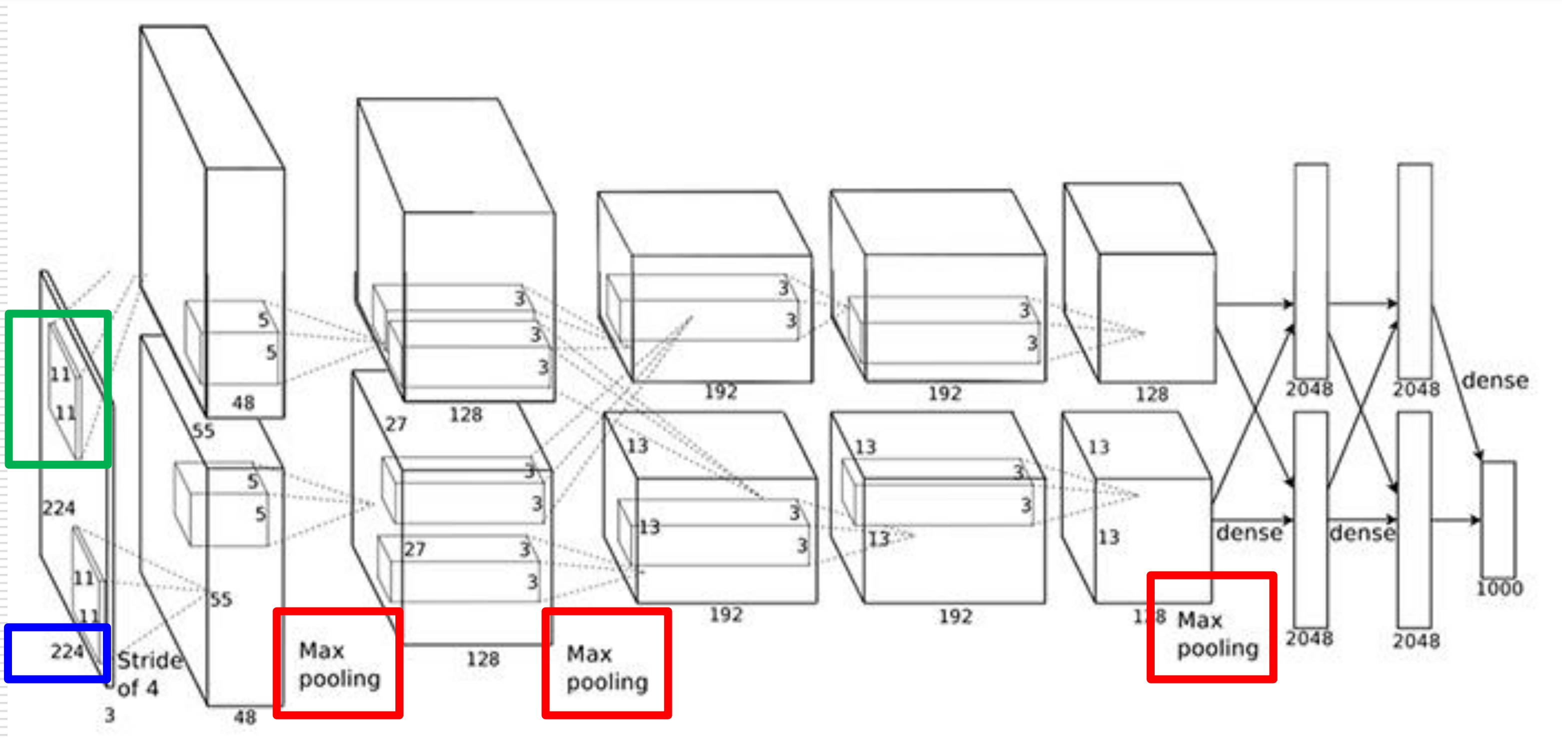
```python
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, padding=2, stride=1)
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)
        self.fc1 = nn.Linear(in_features=16*5*5, out_features=120)
        self.fc2 = nn.Linear(in_features=120, out_features=84)
        self.fc3 = nn.Linear(in_features=84, out_features=10)

    def forward(self, x):
        x = F.sigmoid(self.conv1(x))
        x = F.avg_pool2d(x, kernel_size=2, stride=2)
        x = F.sigmoid(self.conv2(x))
        x = F.avg_pool2d(x, kernel_size=2, stride=2)
        x = torch.flatten(x, 1)
        x = F.sigmoid(self.fc1(x))
        x = F.sigmoid(self.fc2(x))
        x = self.fc3(x)
        return x
```



AlexNet

- Alex Krizhevsky proposed an 8-layer neural network in 2012 and won 1st place in ILSVRC in the same year
  - Top5 : **84.7%** (2nd place: **73.9%**)



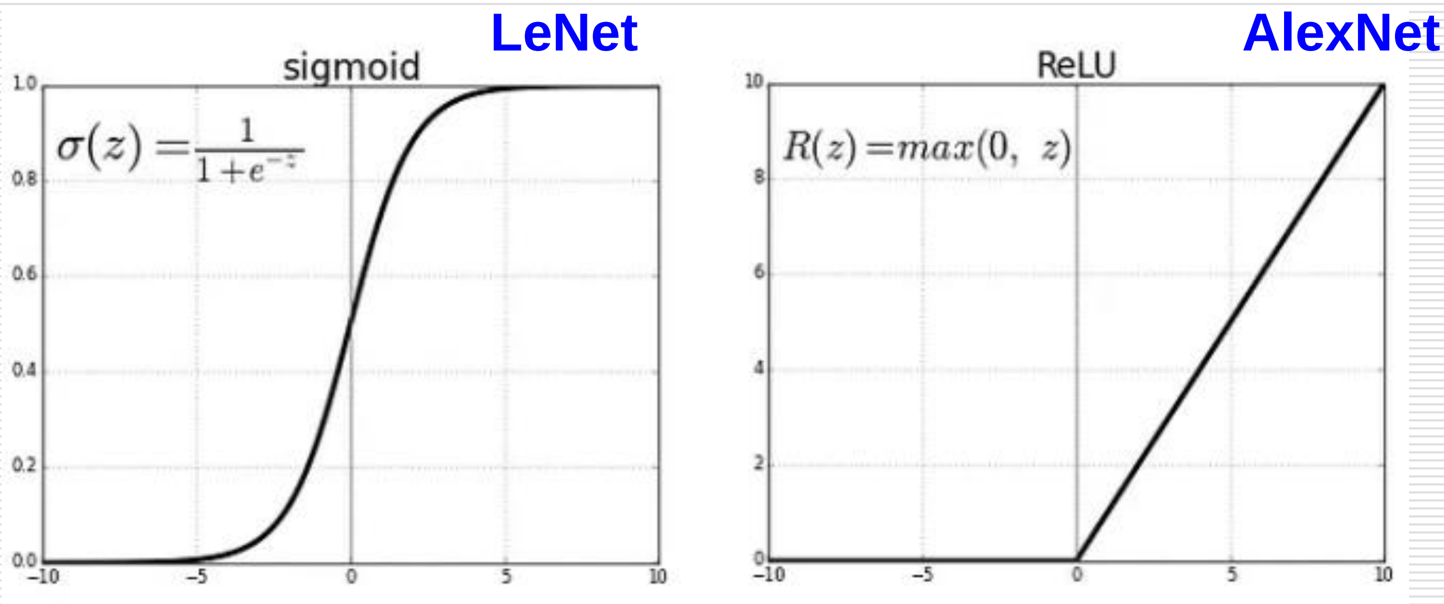
AlexNet - Code

```
class AlexNet(nn.Module):
    def __init__(self, num_classes):
        super(AlexNet, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=11, padding=2, stride=4)
        self.conv2 = nn.Conv2d(in_channels=64, out_channels=192, kernel_size=5, padding=2)
        self.conv3 = nn.Conv2d(in_channels=192, out_channels=384, kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d(in_channels=384, out_channels=256, kernel_size=3, padding=1)
        self.conv5 = nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(in_features=256*6*6, out_features=4096)
        self.fc2 = nn.Linear(in_features=4096, out_features=1024)
        self.fc3 = nn.Linear(in_features=1024, out_features=num_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, kernel_size=3, stride=2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, kernel_size=3, stride=2)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = F.relu(self.conv5(x))
        x = F.max_pool2d(x, kernel_size=3, stride=2)
        x = torch.flatten(x, start_dim=1)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, p=0.5)
        x = F.relu(self.fc2(x))
        x = F.dropout(x, p=0.5)
        x = self.fc3(x)
        return x
```

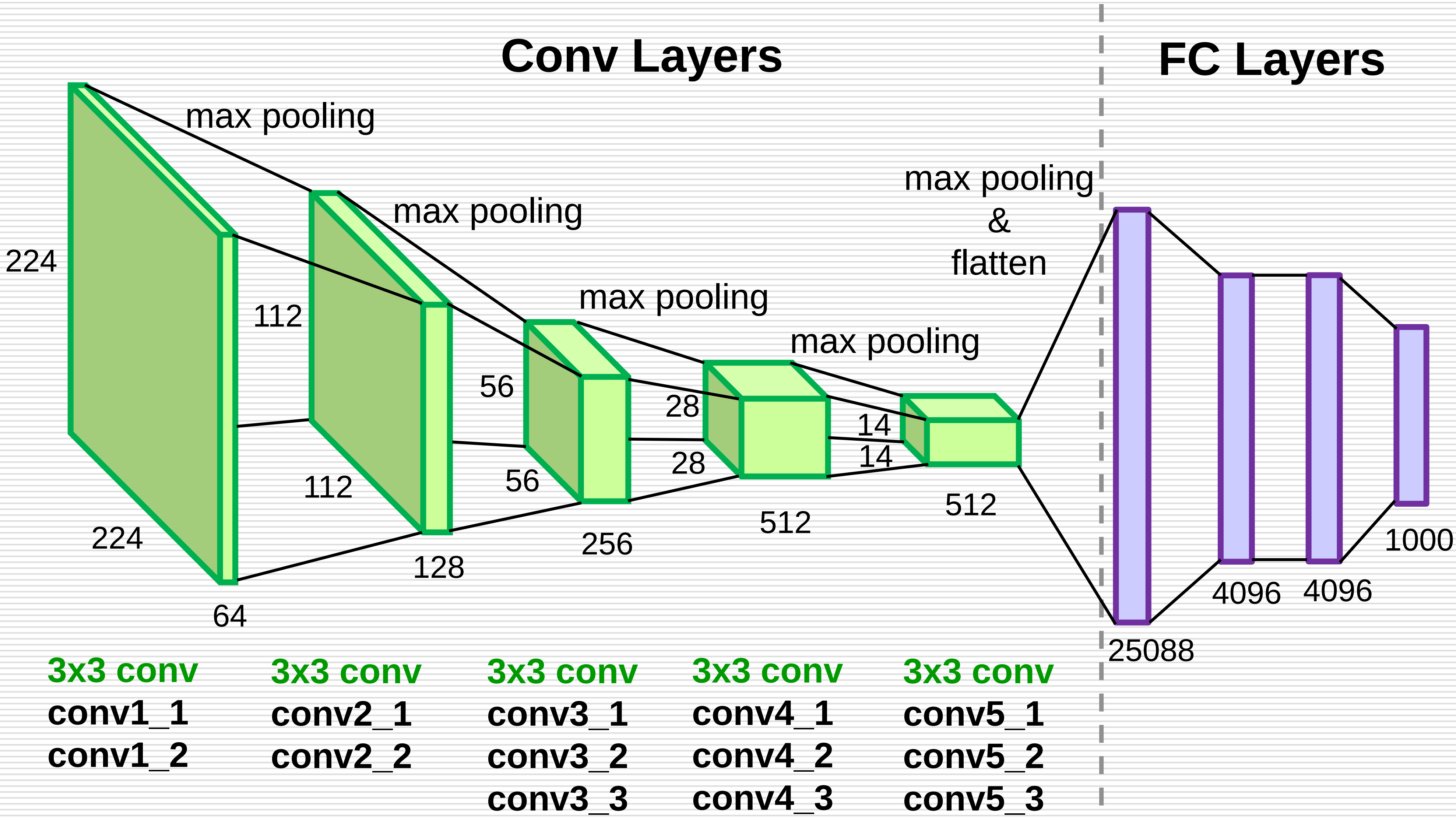
AlexNet

- Data augmentation
- Activation function (ReLU)



- Adopt dropout to avoid model overfitting
- Multiple GPUs in use

VGG-16



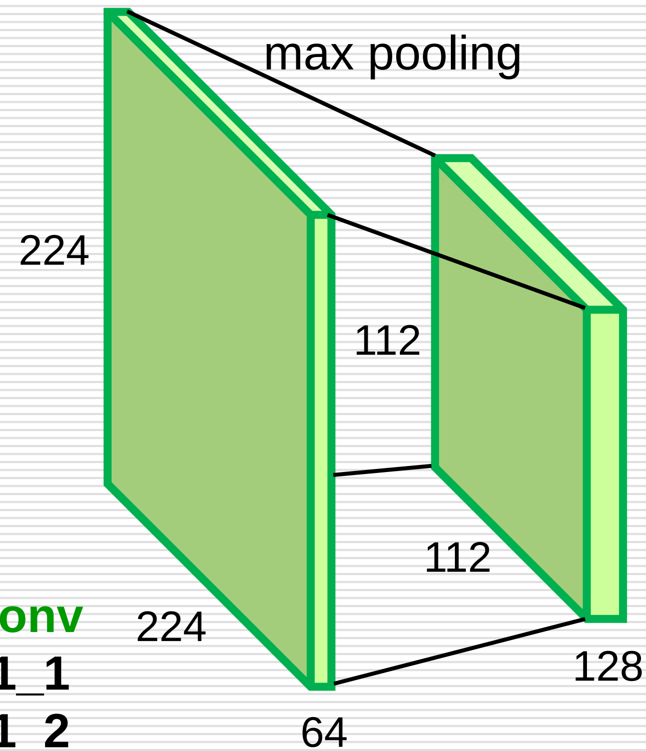
Conv Layers
FC Layers
max pooling
max pooling
max pooling
max pooling
max pooling
&
flatten
224
224
64
112
112
128
56
56
256
28
28
512
14
14
512
25088
4096
4096
1000
3x3 conv
conv1_1
conv1_2
3x3 conv
conv2_1
conv2_2
3x3 conv
conv3_1
conv3_2
conv3_3
3x3 conv
conv4_1
conv4_2
conv4_3
3x3 conv
conv5_1
conv5_2
conv5_3

VGG-16 – Conv1

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
```

Code:

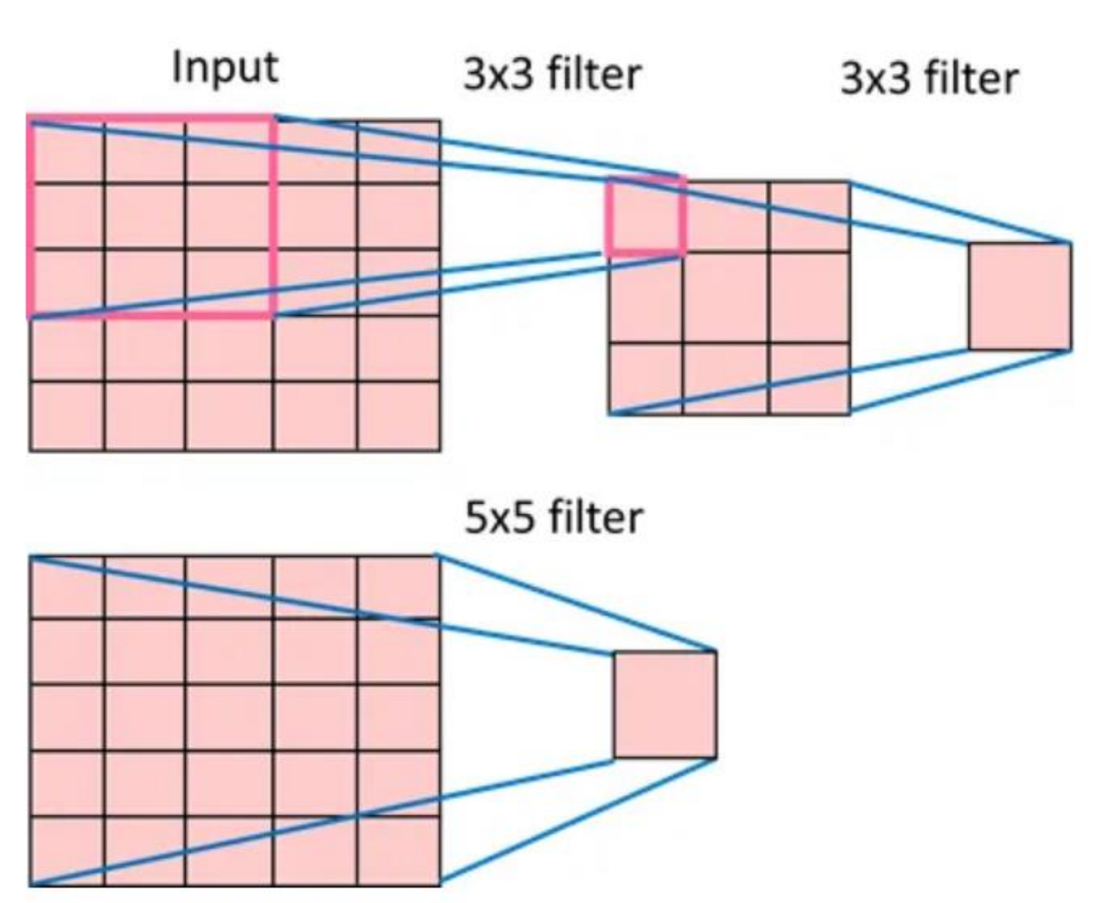
```
self.features = nn.Sequential(
    # conv1
    nn.Conv2d(3, 64, 3, padding=1),
    nn.ReLU(),
    nn.Conv2d(64, 64, 3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(2, stride=2, return_indices=True)
)
```



3x3 conv
conv1_1
conv1_2

VGG-16

- 2 times 3x3conv vs. 1 times 5x5conv
- Same reception field, less # of parameters



# of parameters
= (3*3) * 2 = 18

# of parameters
= (5*5) * 1 = 25

VGG-16

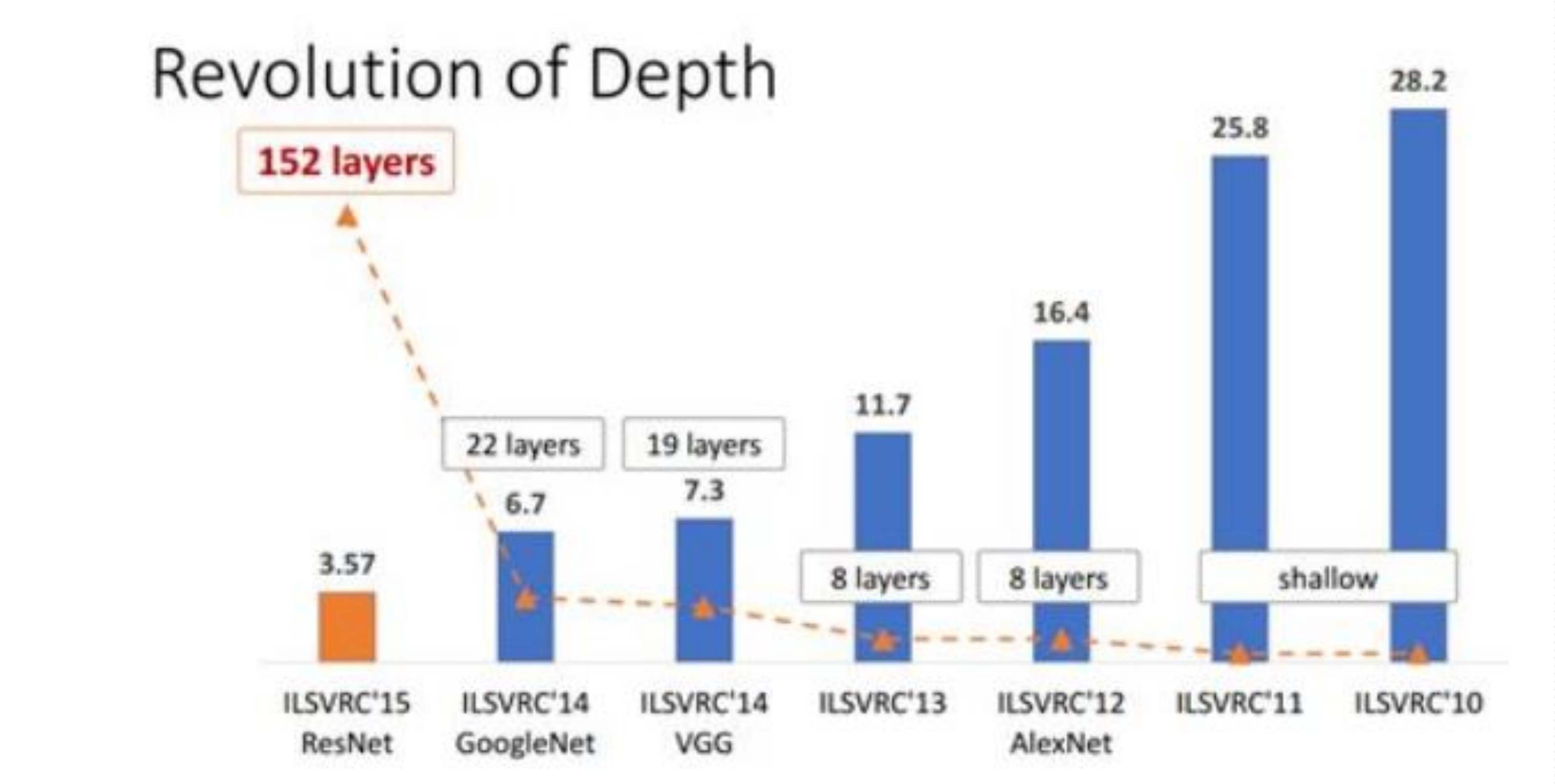
```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace)
    (2): Dropout(p=0.5)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace)
    (5): Dropout(p=0.5)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

#Layer increases ➔ accuracy increases

Q: always true?

Introduction to ResNet

- Reference: https://arxiv.org/abs/1512.03385
- Propose a new convolution architecture to solve the **degradation** problem in deep networks



Problems of Deep Networks

- Degradation

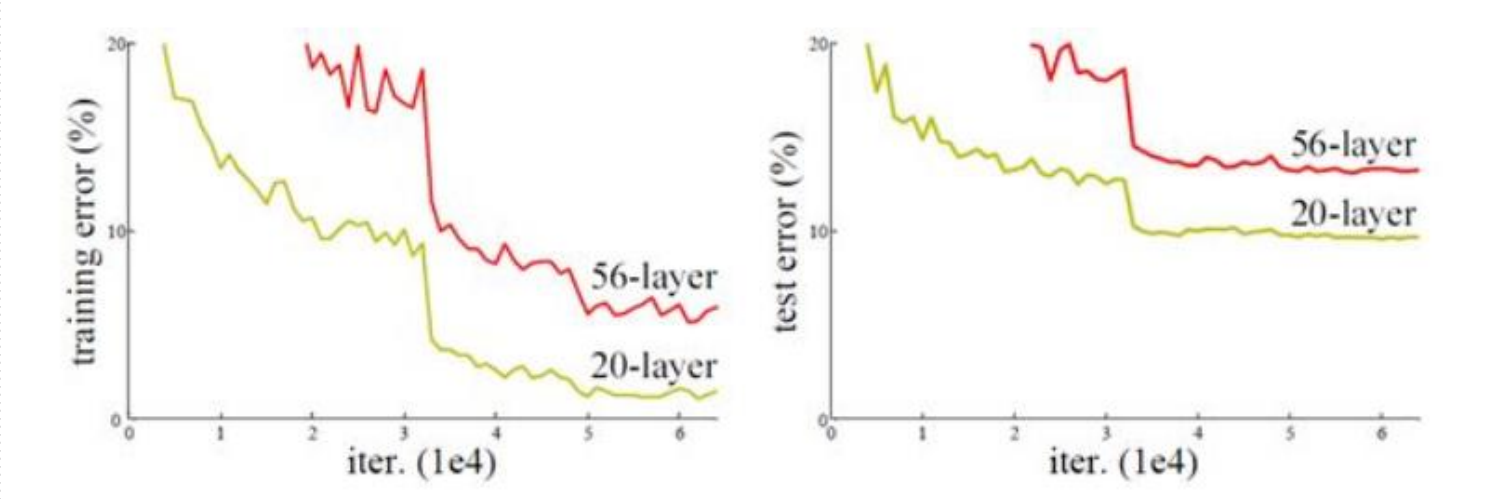
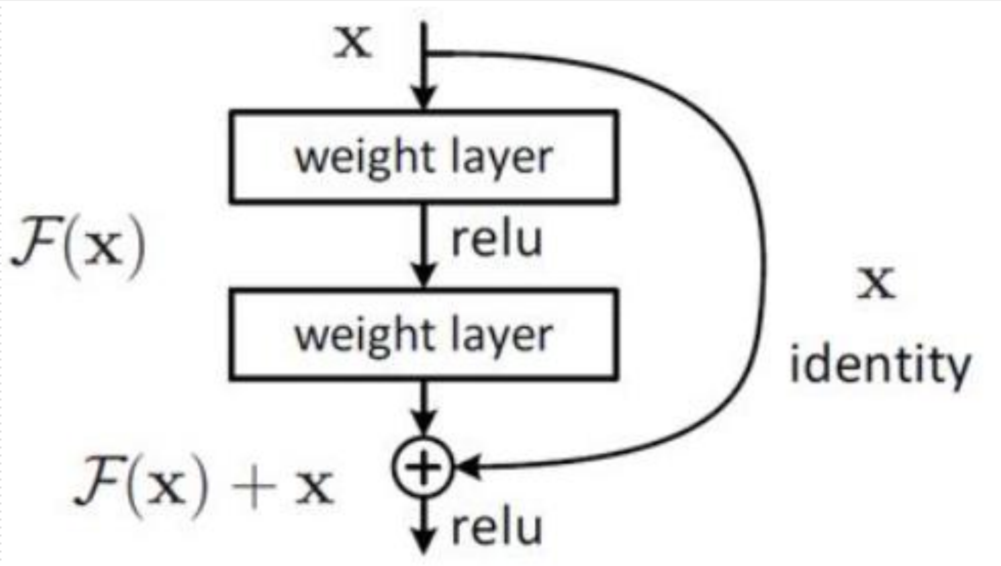


图3 20层与56层网络在CIFAR-10上的误差

- Reason:
  - Deep networks are not easy to train: **gradient vanishing**
  - Efficiency of gradient update

Residual Blocks (1/2)

- H(x) consists of residual part F(x) and identity part x
  - This reformulation is motivated by the counterintuitive phenomena about the degradation problem
  - If identity mappings are optimal, the solvers may simply drive the weights of the multiple nonlinear layers toward zero to approach identity mappings



Residual Blocks (2/2)

- Residual Learning
  - Learn the residual mapping between the input and the output, rather than directly learning the output
- Shortcut Connections
  - Path that directly connects the input to the output
  - Ensures that gradients can be directly backpropagated through these connections, thereby alleviating the vanishing gradient problem
- Stacking Residual Blocks
  - Contain several convolutional layers, activation functions, and regularization layers

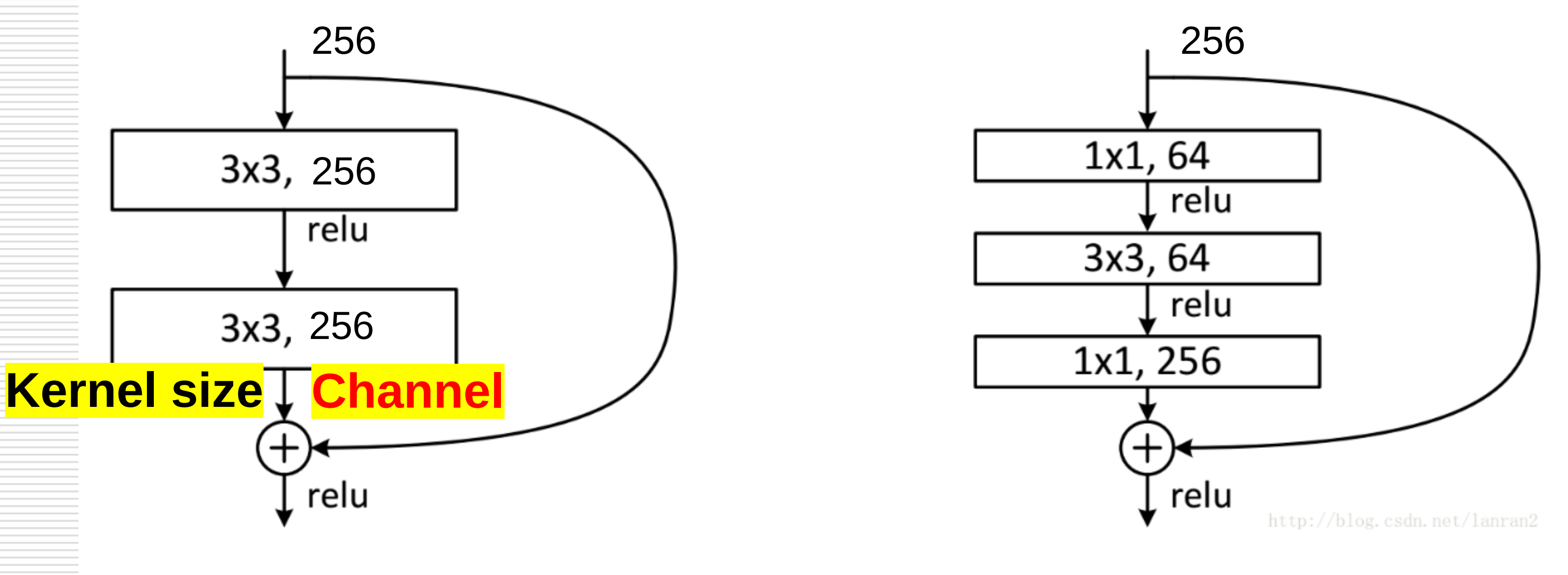
Two Types of Residual Blocks

- The left one is used in ResNet34

  - # of param. = 3*3*256*256*2 = 1179648

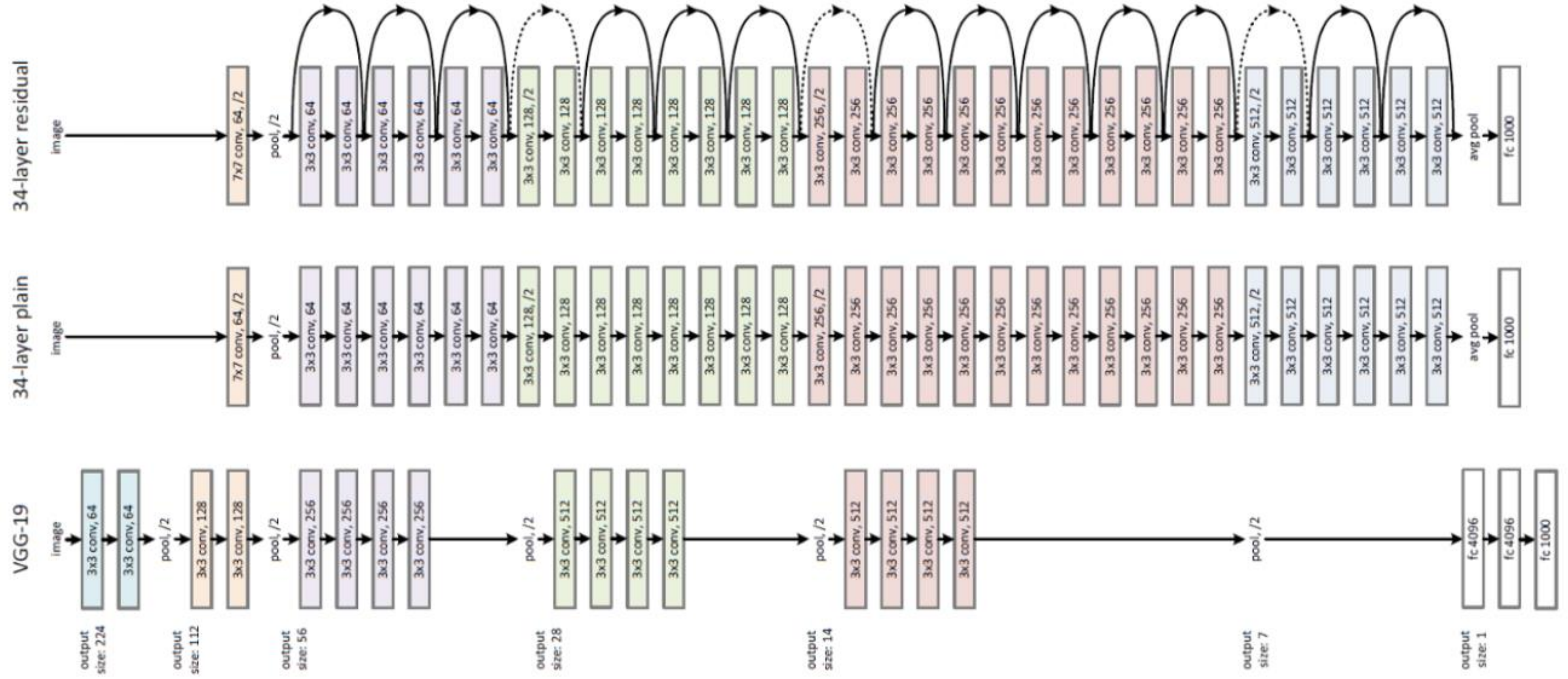
- The right one is also called Bottleneck design which is used in ResNet50/101/152

  - # of param. = 1*1*256*64+3*3*64*64+1*1*64*256 = 69632



Architecture of ResNet34

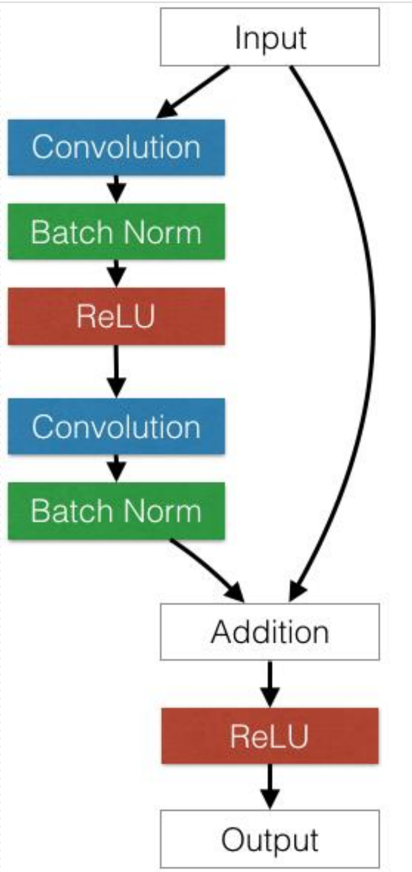
- Transformation of VGG19
  - **shortcut** connection

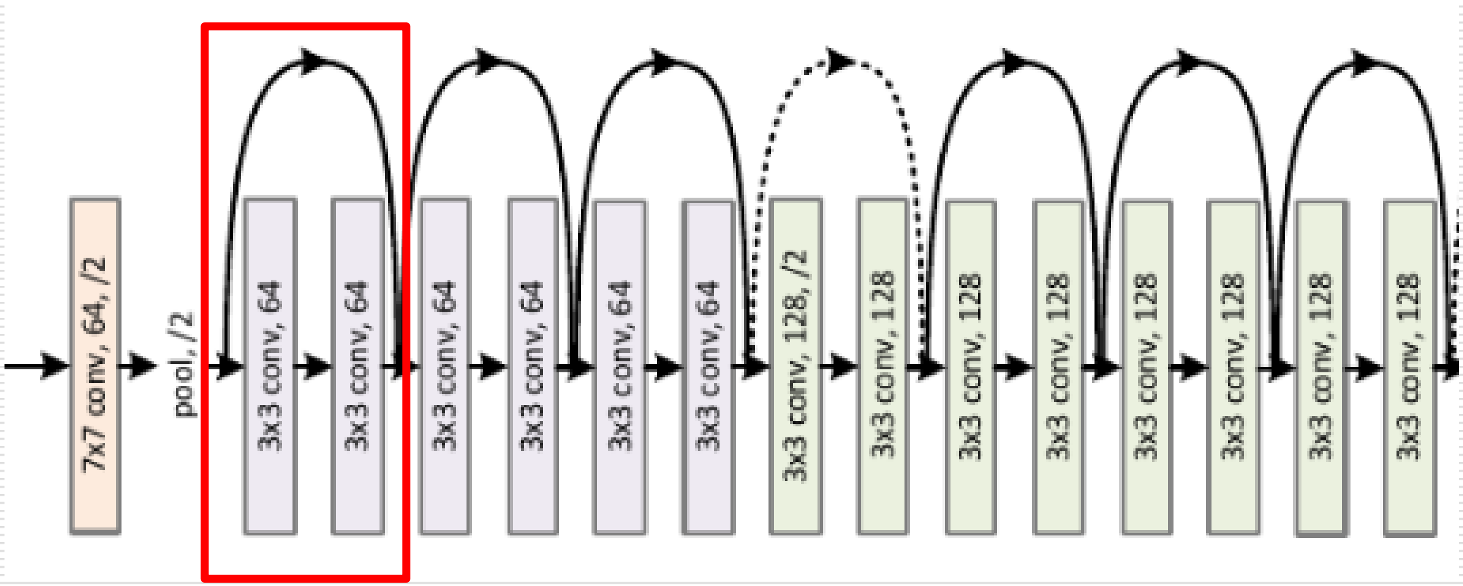


34-layer ResNet with Skip / Shortcut Connection (Top), 34-layer Plain Network (Middle), 19-layer VGG-19 (Bottom)

Architecture of ResNet34

- **Shortcut** connection





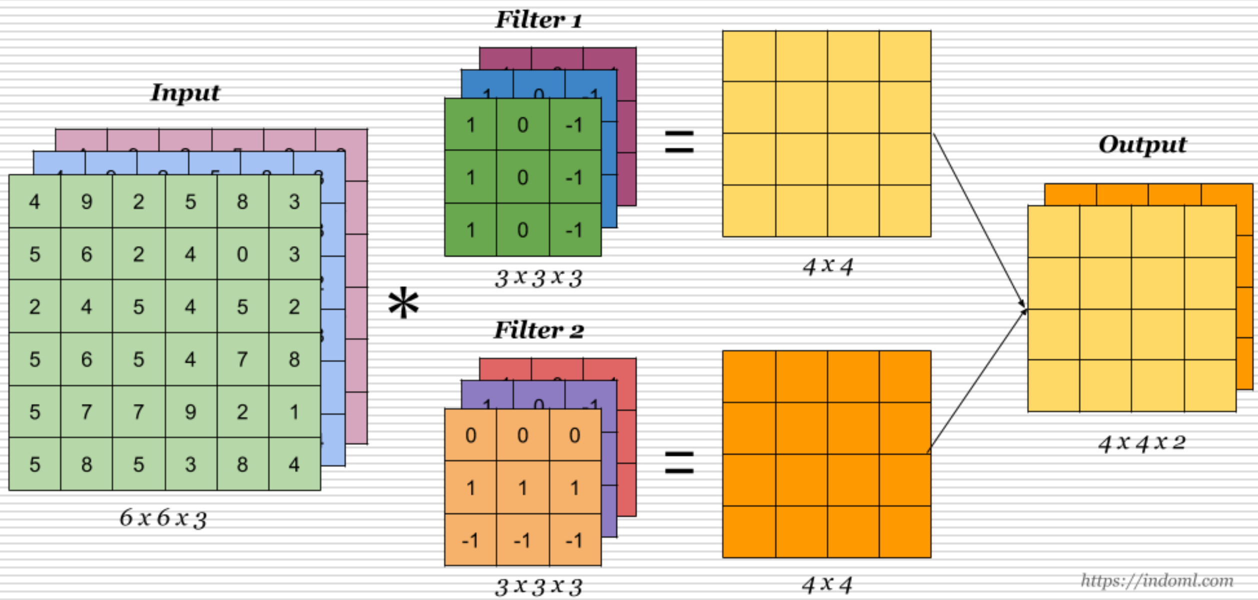
Introduction to MobileNet V1

- Reference: https://arxiv.org/abs/1704.04861
- Google proposed In 2017, in order to use on mobile devices
  - Propose a new convolution operation to reduce #MACs

Figure 1. MobileNet models can be applied to various recognition tasks for efficient on device intelligence.

Convolution Layer

- Conventional convolution layer



- Depthwise separable convolution
  - MobileNet separates a conventional convolution layer into:

    (1) Depthwise convolution

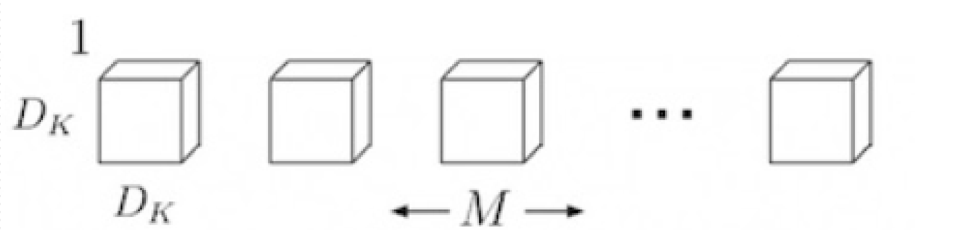
    (2) Pointwise convolution

Recall: Depth-wise Convolution

- Depth-wise (DW) convolution
  - Each channel is independent
  - Each input channel will generate only one output feature map
  - Feature map(fp): # of input channel = # of output channel

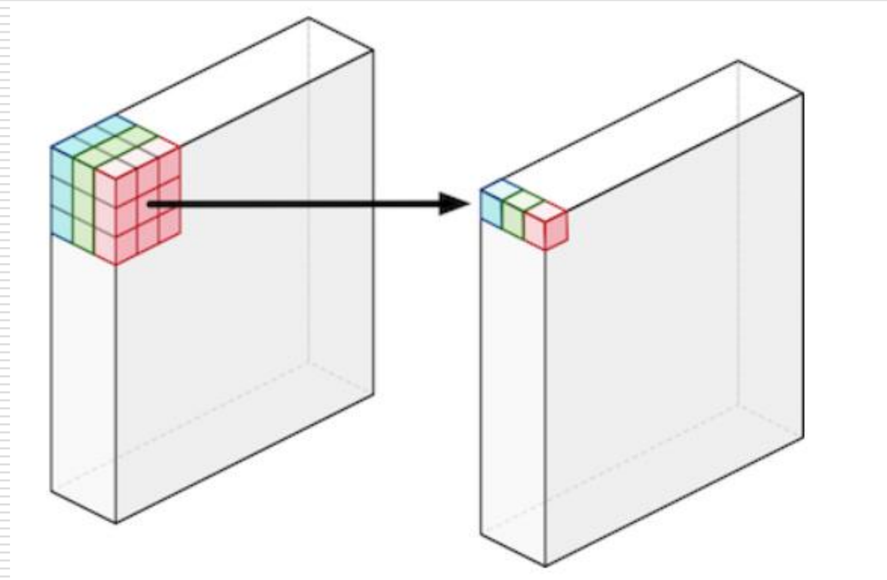
Kernel:

# of kernel’s input channel = 1



kernel size

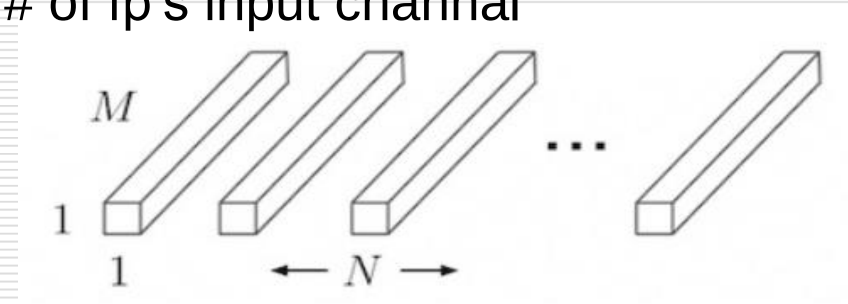
# of fp’s input channal



Pointwise Convolution

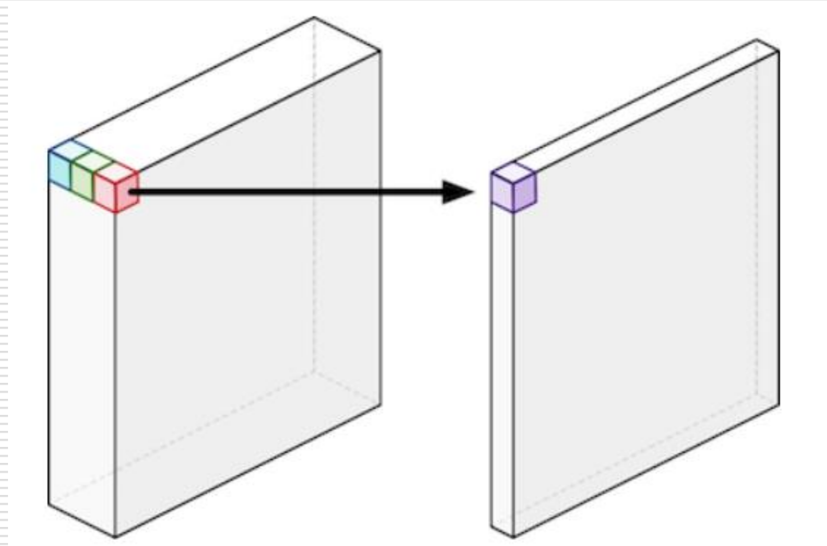
- Pointwise (PW) convolution
  - Similar to conventional convolution layer
  - kernel size: 1*1

# of fp’s input channal



kernel size=1

# of fp’s output channal



Benefits of DW & PW Convolution

- Reducing #MACs in convolution layers

Definition: M-input channel N-output channel

D_k-kernel size D_f-feature map size

➢ Conventional convolution

= D_k x D_k x M x (N x D_f x D_f)

➢ Depthwise convolution

= D_k x D_k x (M x D_f x D_f)

➢ Pointwise convolution

= M x (N x D_f x D_f)

Total = (D_k x D_k + N) x M x D_f x D_f

Depth-Wise Separable Convolution

- Depth-wise separable convolution
  - = depth-wise convolution + point-wise convolution (used to change channel dimension)

- When number of input channel and output channel is huge, the advantage of depth-wise separable in terms of computation cost and parameters is unneglectable.

MobileNet Architecture

- Red frame: conventional conv.
  - Reduce size of feature map by convolution with stride 2
- Blue frame: depthwise conv. + pointwise conv.

- Blue frame with the formula, # of MAC will be:

✓ DW convolution

  - Depth-wise Conv. = 3.6M
  - Pointwise Conv. = 25.7M

✓ If conventional Conv.

  - Convolution = 231.2M

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Performance

- MobileNet

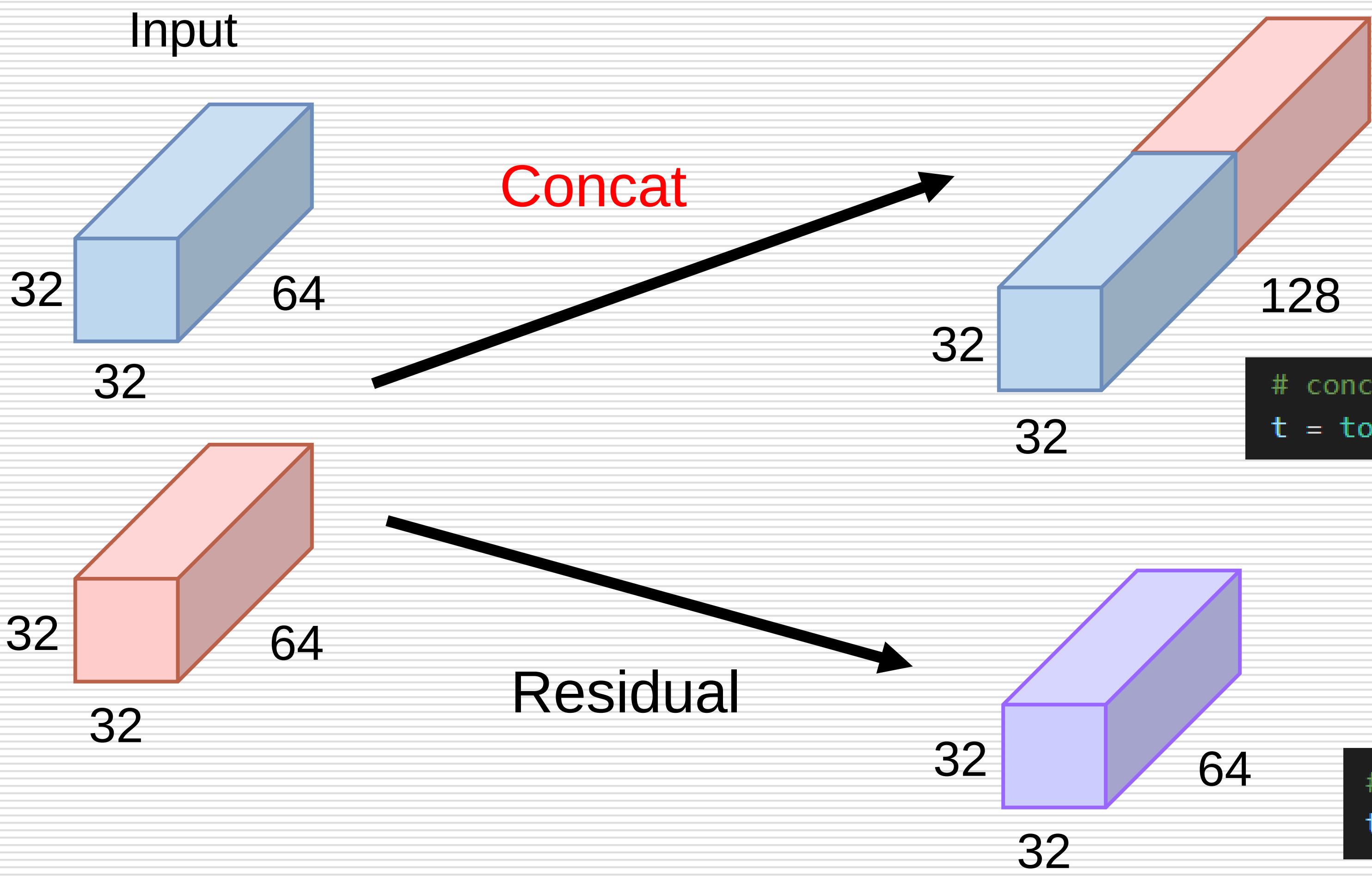
  - DW&PW Convolution vs. Conventional Convolution

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Concatenate

- Assume there are 2 inputs, which size is 32x32 with CH=64

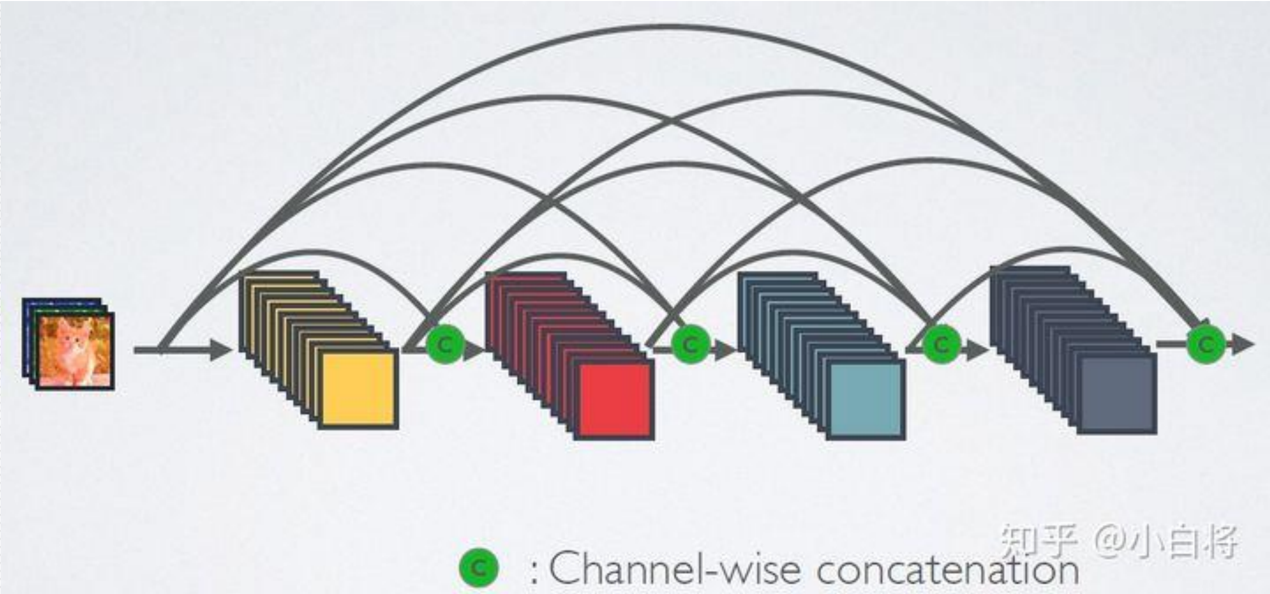



```python
# concat
t = torch.cat((t1, t2), 1)
```

```python
# residual
t = t1 + t2
```

Introduction to DenseNet

- Connect all layers (with matching feature-map sizes) directly with each other
  - Channel-wise concatenation
  - Alleviate the vanishing-gradient problem
  - Strengthen feature propagation & encourage feature reuse



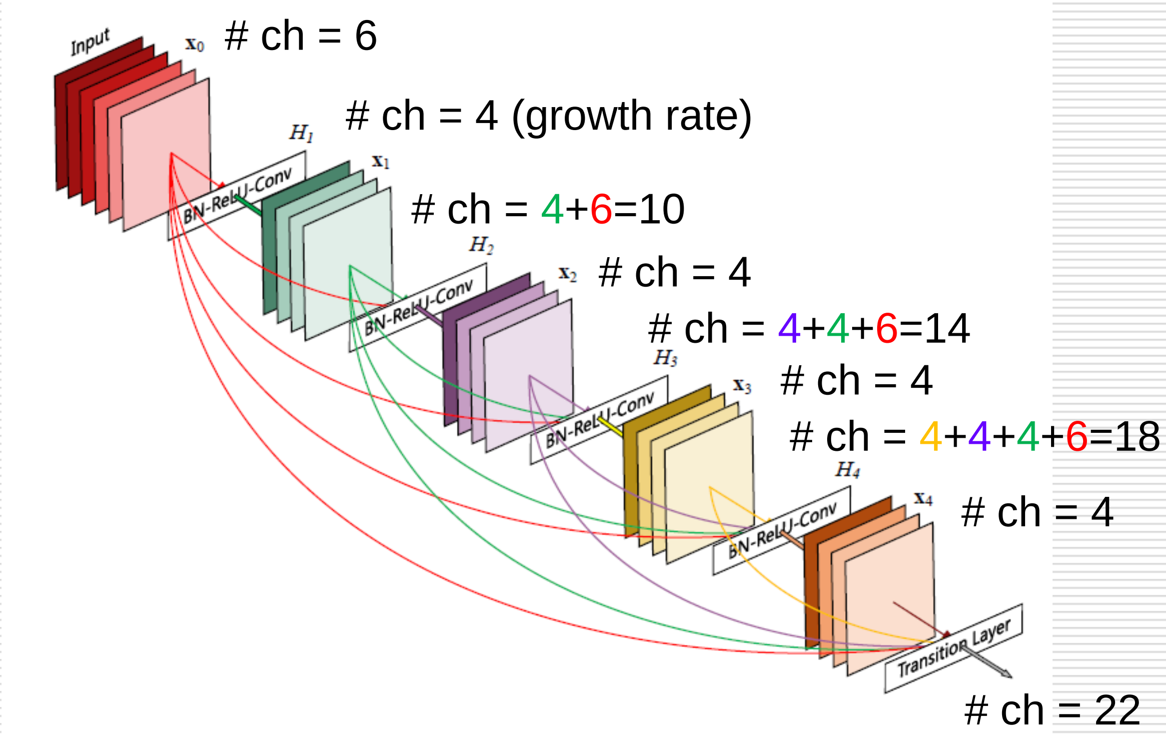
Growth Rate

- The number of output feature maps of a DenseBlock is defined as the growth rate
- Output feature map = Input channel + k*(layer-1)
  - Ex :

    Input channel = 6

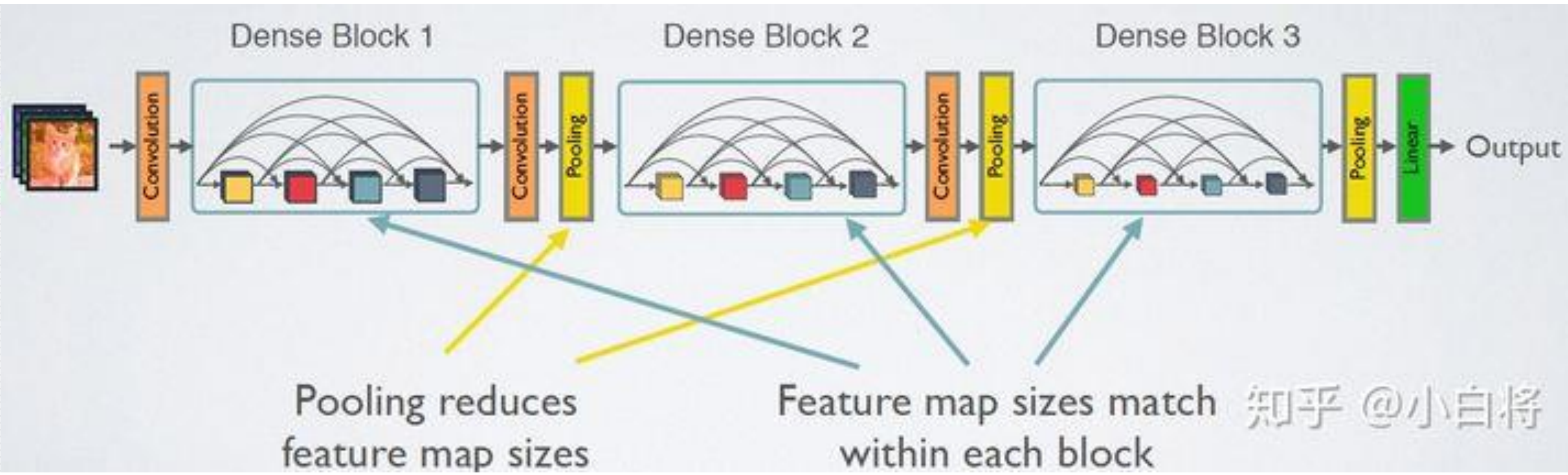
    Growth rate(k) = 4

    Output channel = 22



Transition Layer

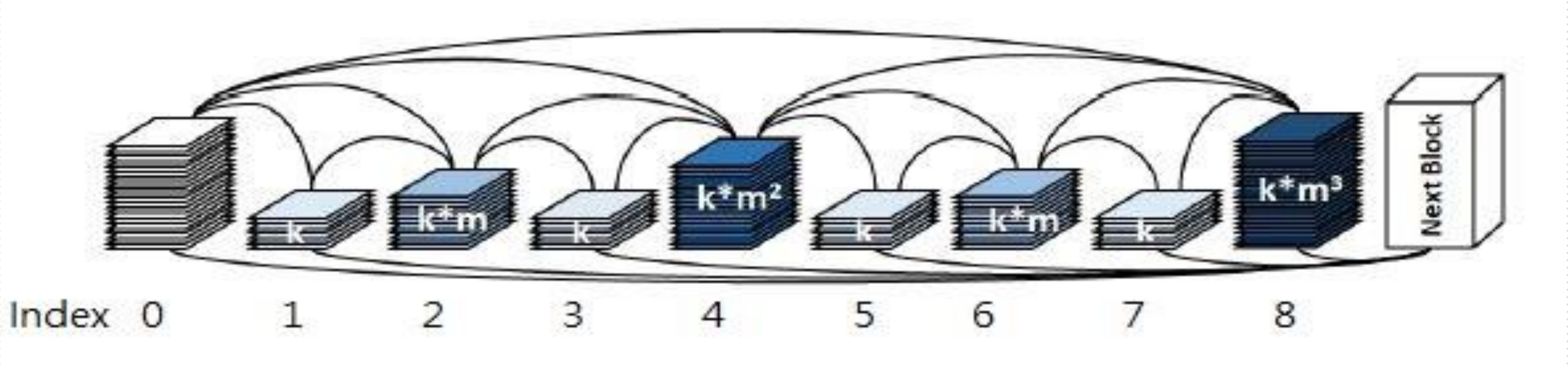
- Dense Block is a group of layers connected to all their previous layers
  - The feature maps of each layer has the same size
- Transition layer is used to connect 2 DenseBlock
  - Down-sample the feature maps with Pooling layer



Introduction to HarDNet (ICCV'19, NTHU)

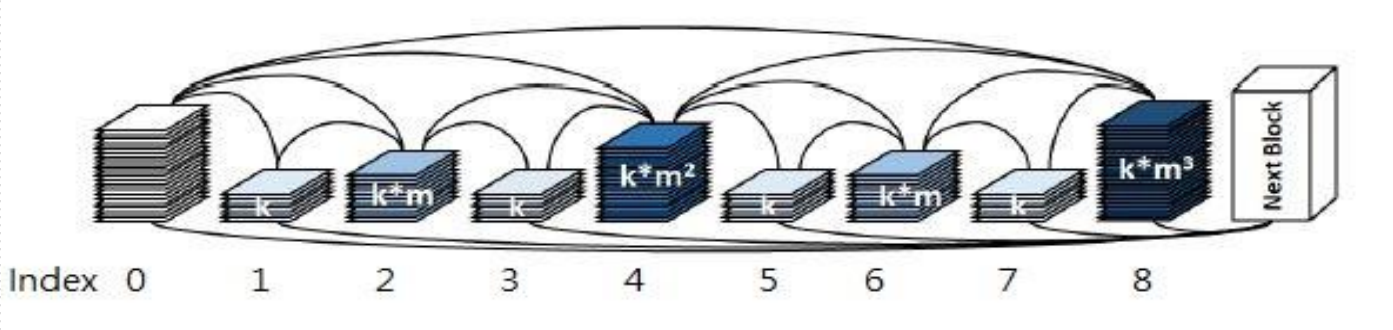
- The goals of most current models:
  - High Accuracy
  - Low Computation (MACs, flops)
- #MACs may not be able to accurately predict the execution time
- Times of accessing the feature maps from memory may be the major factor in execution time
- Analyze how execution time can be reduced by reducing DRAM accesses without accuracy drop

Harmonic DenseNet



- Layers with an index divided by a larger power of two are more influential than those that divided by a smaller power of two -- amplify these key layers by m
- Gradient vanishing problem of back propagation can be solved by concatenating previous layers

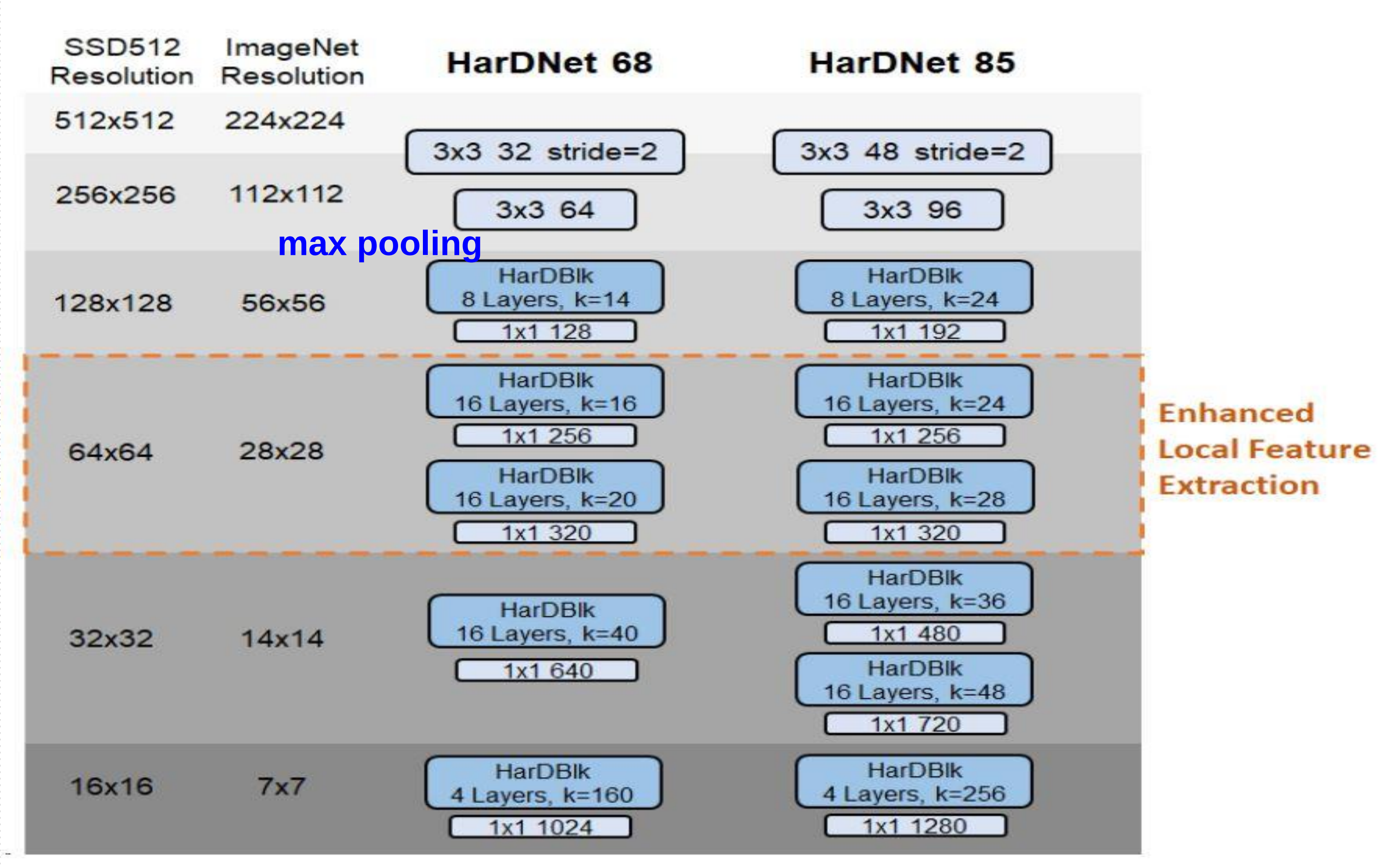
Harmonic DenseNet



- Multiplier m
- Each block has a growth rate k; layers of block should be 2(m) to the power of n
- If block input=100 => k=20, m=2, layer(n)=8

	1	2	3	4	5	6	7	8	Block out
Input channel	100	120	40	160	80	100	40	240	
Output channel	20	40	20	80	20	40	20	160	240

HarDNet Architecture



Conclusion

- Gradient vanishing
  - concat dimention
  - Skip connection (resnet)
  - Activation alternative (sigmod → relu, … etc)
- Reduce MAC(mult-add count) & parameters
  - Depthwise-separable convolution
  - Nn.Conv2(64, 256, 3,3) → nn.Conv2d(64, 64, 3,3,) & nn.Conv2d(64, 256, 1, 1)
- Change feature map size
  - Option1 : convolution stride
  - Option 2: maxpooling, avgpooling(will introduce another computation cost)

Introduction to EfficientNet (1/2)

- Propose a more efficient way to augment the dimensions of the model with a more principled way

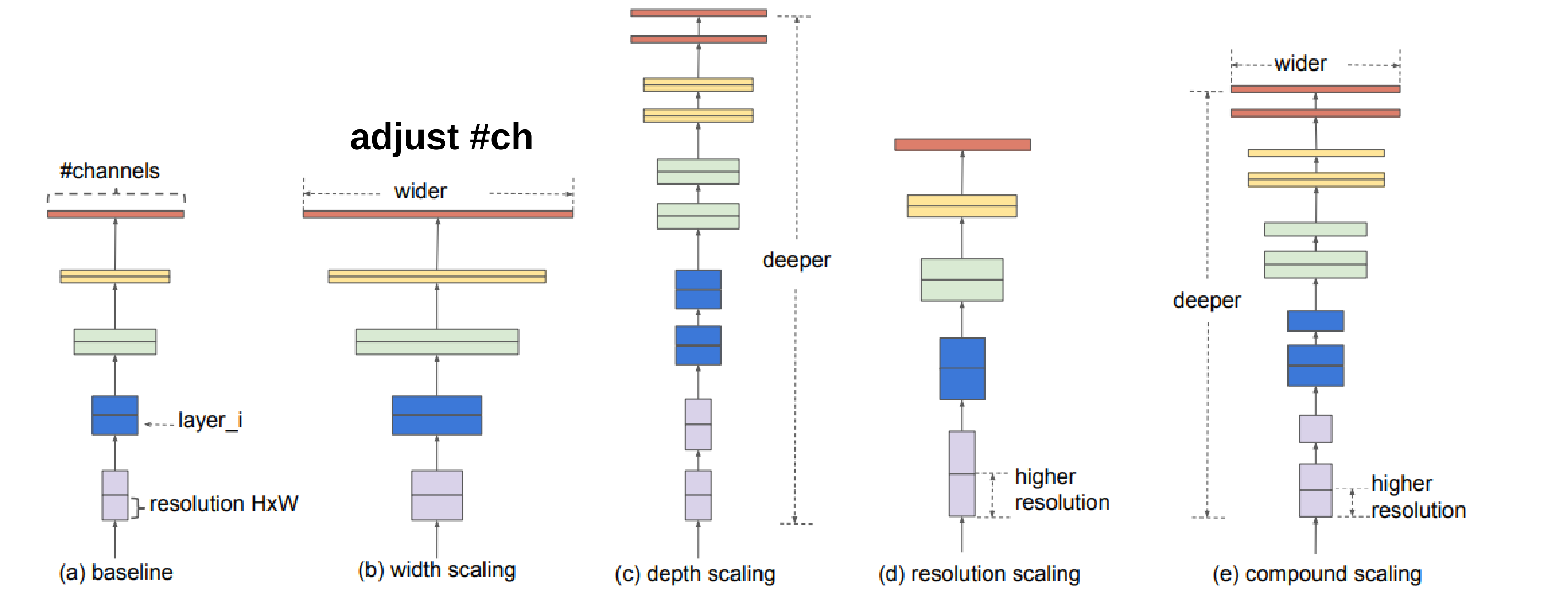


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

Introduction to EfficientNet (2/2)

- EfficientNet-B0 baseline network

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Problem Formulation

- Define a CNN model as the following formula:

$$\mathcal{N} = \bigodot_{i=1\ldots s} \mathcal{F}_i^{L_i}\left(X_{\langle H_i, W_i, C_i \rangle}\right)$$

Fi: i stage

Li: depth of i

- d, w, r represent the magnification of the three dimensions of depth, width, and resolution in the CNN model
- Find the three parameters d, w, r that can have the largest accuracy under the limitations (memory, flops).

$$\begin{aligned} \max_{d,w,r} \quad & Accuracy\big(\mathcal{N}(d,w,r)\big) \\ s.t. \quad & \mathcal{N}(d,w,r) = \bigodot_{i=1\ldots s} \hat{\mathcal{F}}_i^{d\cdot \hat{L}_i}\big(X_{\langle r\cdot \hat{H}_i, r\cdot \hat{W}_i, w\cdot \hat{C}_i\rangle}\big) \\ & \mathrm{Memory}(\mathcal{N}) \leq \mathrm{target\_memory} \\ & \mathrm{FLOPS}(\mathcal{N}) \leq \mathrm{target\_flops} \end{aligned}$$

EfficientNet – Depth

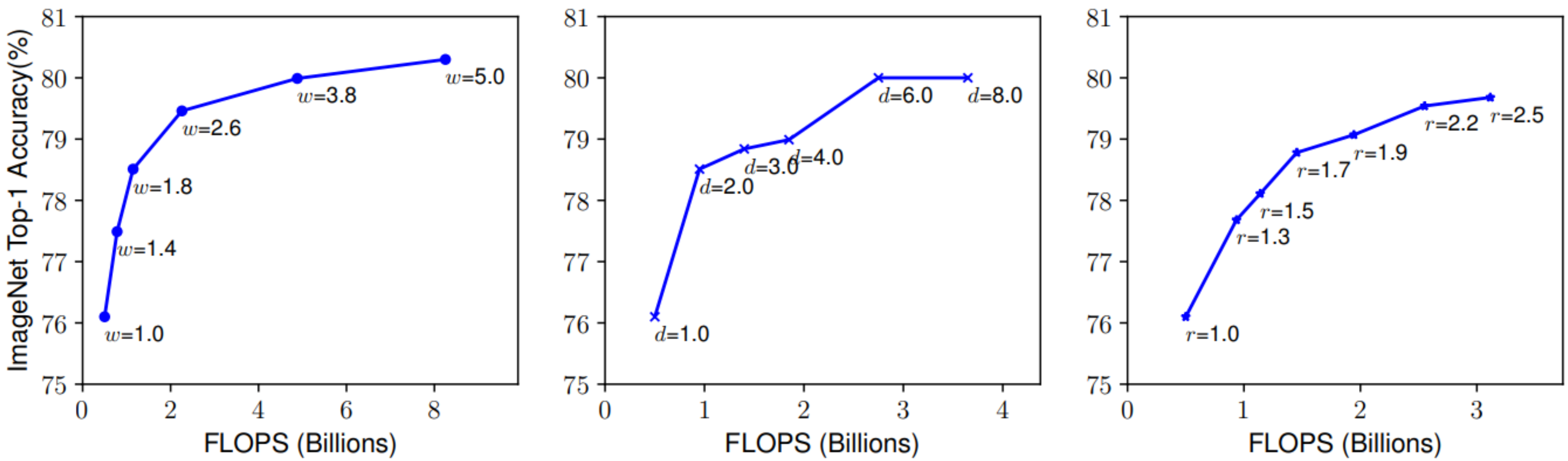
ACC: 77.3%

ACC: 79%
+1.4B FLOPS

EfficientNet – Scaling Dimensions

- Depth (d)
  - Deeper network can capture more rich features
  - Difficult to train due to the vanishing gradient problem
- Width (w)
  - Wider network can capture more fine-grained features
  - Wide but have shallow depth struggle to capture higher-level features (saturation).
- Resolution (r)
  - Capture more fine-grained patterns from higher resolution input images.

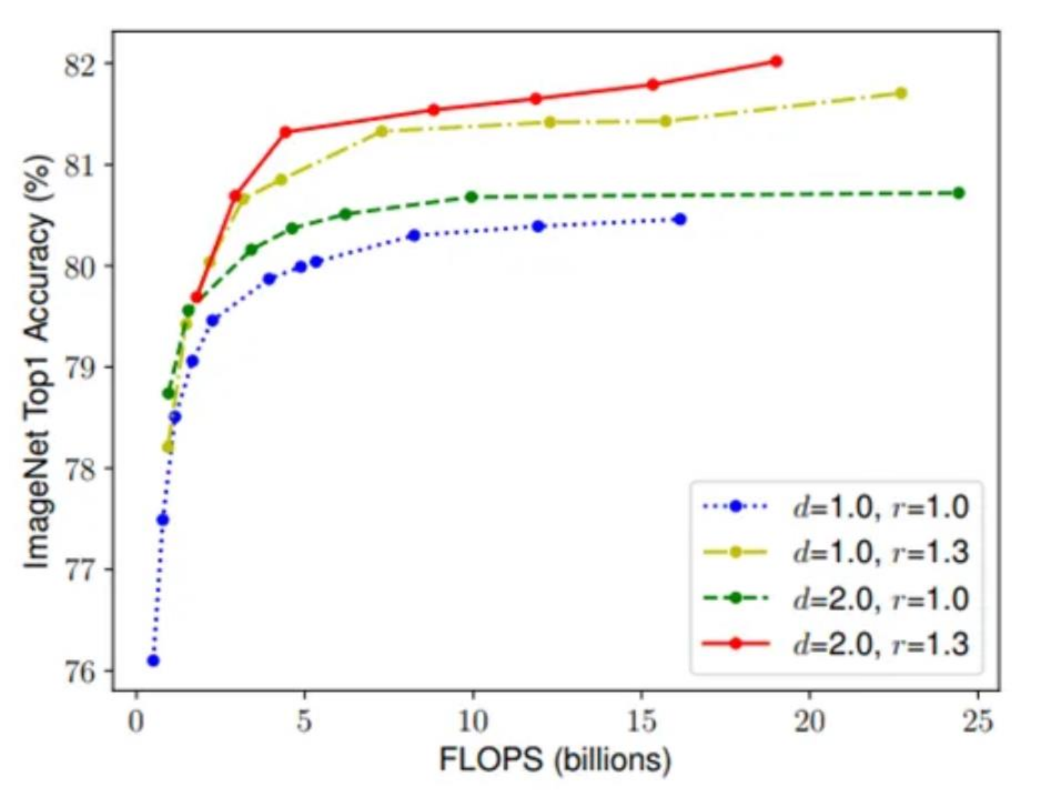
EfficientNet – Compound



Model	FLOPS	Top-1 Acc.
Baseline model (EfficientNet-B0)	0.4B	77.3%
Scale model by depth (d=4)	1.8B	79.0%
Scale model by width (w=2)	1.8B	78.9%
Scale model by resolution (r=2)	1.9B	79.1%
Compound Scale (d=1.4, w=1.2, r=1.3)	**1.8B**	**81.1%**

EfficientNet – Compound Scaling

- From the blue line in the chart (d=1, r=1), it can be seen that the accuracy quickly saturates, whereas the red line (d=2, r=1.3) shows that higher accuracy can be achieved at the same FLOPS.



Contribution EfficientNet

Model	FLOPS	Top-1 Acc.
Baseline MobileNetV1 (Howard et al., 2017)	0.6B	70.6%
Scale MobileNetV1 by width (w=2)	2.2B	74.2%
Scale MobileNetV1 by resolution (r=2)	2.2B	72.7%
compound scale (d=1.4, w=1.2, r=1.3)	**2.3B**	**75.6%**
Baseline MobileNetV2 (Sandler et al., 2018)	0.3B	72.0%
Scale MobileNetV2 by depth (d=4)	1.2B	76.8%
Scale MobileNetV2 by width (w=2)	1.1B	76.4%
Scale MobileNetV2 by resolution (r=2)	1.2B	74.8%
MobileNetV2 compound scale	**1.3B**	**77.4%**
Baseline ResNet-50 (He et al., 2016)	4.1B	76.0%
Scale ResNet-50 by depth (d=4)	16.2B	78.1%
Scale ResNet-50 by width (w=2)	14.7B	77.7%
Scale ResNet-50 by resolution (r=2)	16.4B	77.5%
ResNet-50 compound scale	**16.7B**	**78.8%**

CNN on ImageNet

- CNN-based
  - https://**paperswithcode.com**/sota/image-classification-on-imagenet

Rank	Model	Top 1 Accuracy	Number of params	GFLOPs	energy consumption	Extra Training Data	Paper	Code	Result	Year	Tags
1	**OmniVec** (ViT)	92.4%				×	OmniVec: Learning robust representations with cross modal sharing			2023	
2	**CoCa** (finetuned)	91.0%	2100M			×	CoCa: Contrastive Captioners are Image-Text Foundation Models			2022	ALIGN, Transforr, JFT-3B
3	**Model soups** (BASIC-L)	90.98%	2440M			×	Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time			2022	ALIGN, JFT-3E, Conv+Transforme
4	**Model soups** (ViT-G/14)	90.94%	1843M			×	Model soups: averaging weights of multiple fine-tuned models improves accuracy without			2022	JFT-3B, Transforr

Introduction to Once-for-All

- Efficiently design neural network models on different platforms
  - constraints: latency, energy
- Design models with different sizes separately (human-based, NAS)

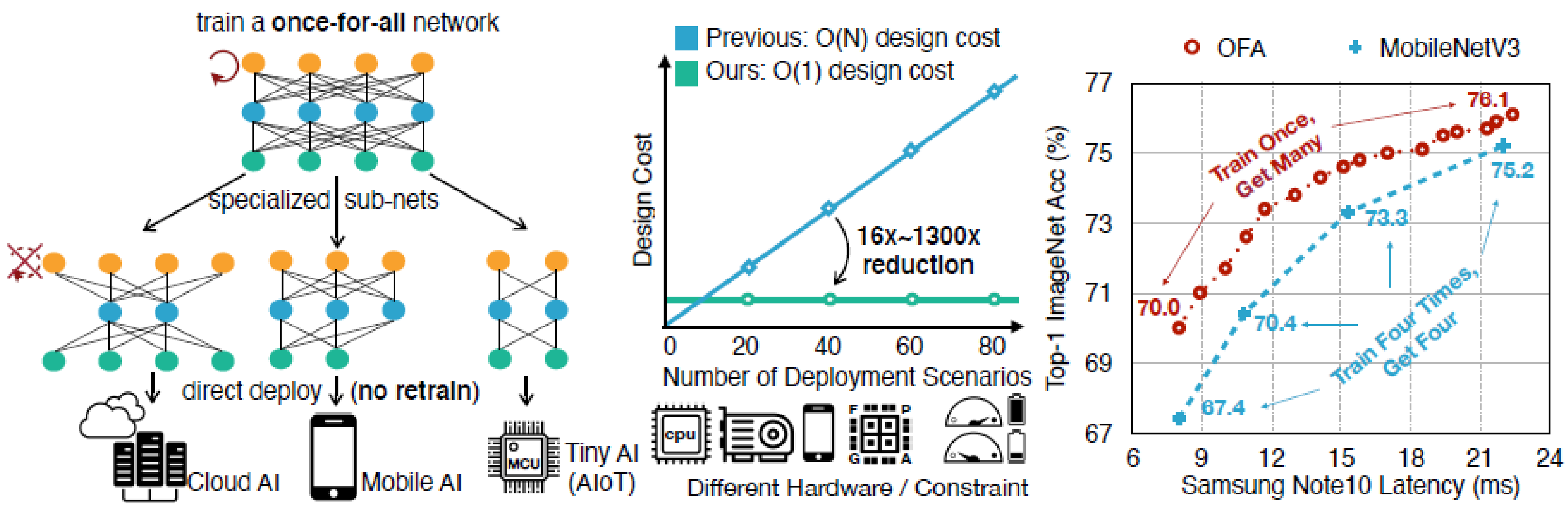
  1. repeat the network design process
  2. retrain the network from scratch

  - linear growth O(N): expensive!!!
- Once-for-all network

  1. Select parts from once-for-all models as new small models
  2. Generate different depths, widths, and kernel sizes without retraining

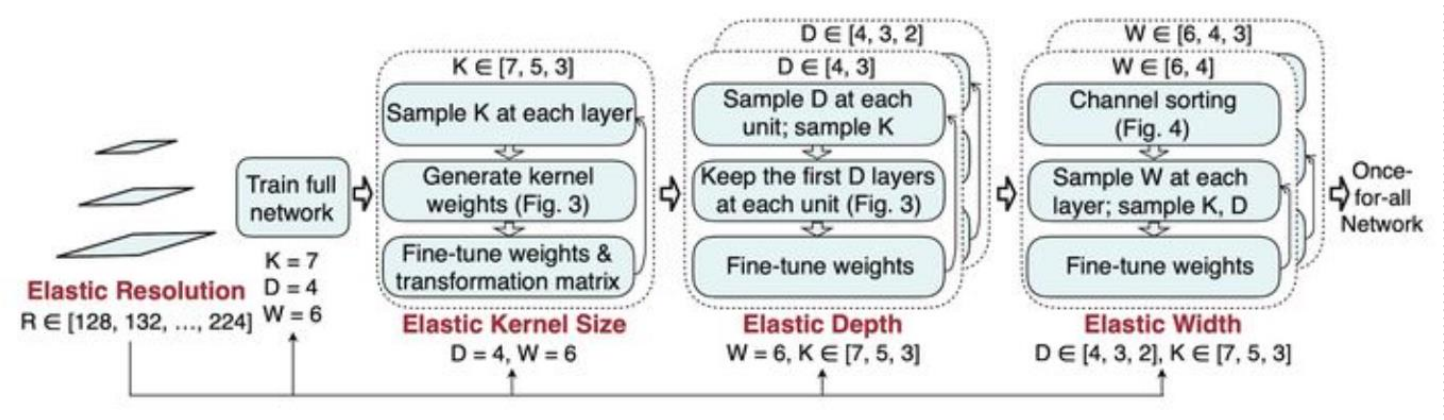
  - constant growth O(1)

Example of Once-for-All Network



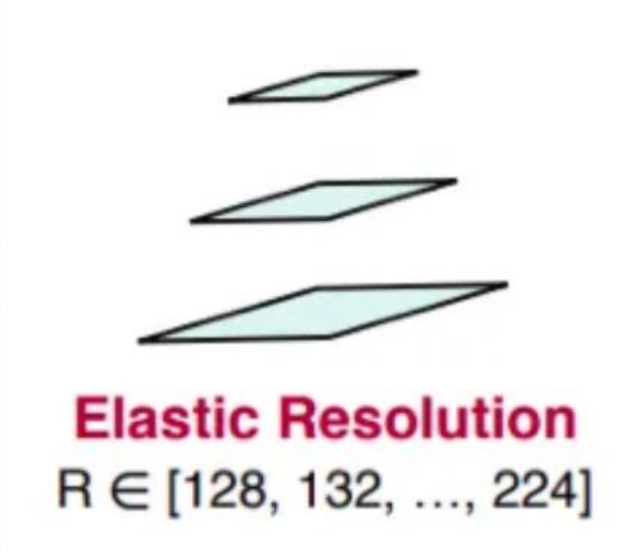
Progressive Shrinking

- It is difficult to train an once-for-all network that can support all sub-networks
- Start by training the entire once-for-all network, and then fine-tune smaller sub-networks
  - Sub-networks can have good initial values by retaining important parameters in the larger model
  - Parameters are sorted to prevent the sub-networks from affecting the performance of the overall network



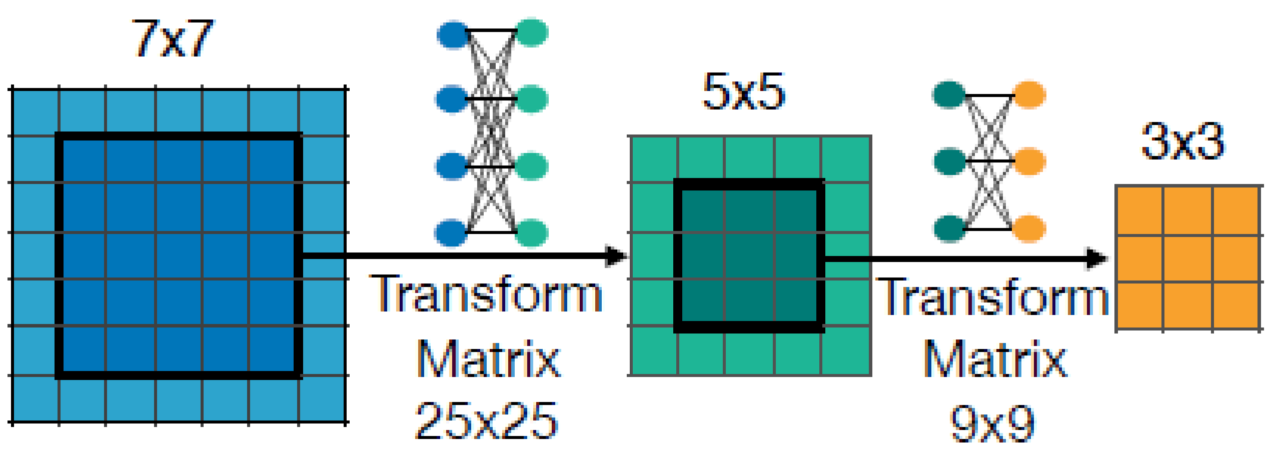
Elastic Resolution

- Model has not seen photos of a certain size during training, its accuracy can significantly decrease
- To support elastic resolution, we randomly scale photos up or down during training



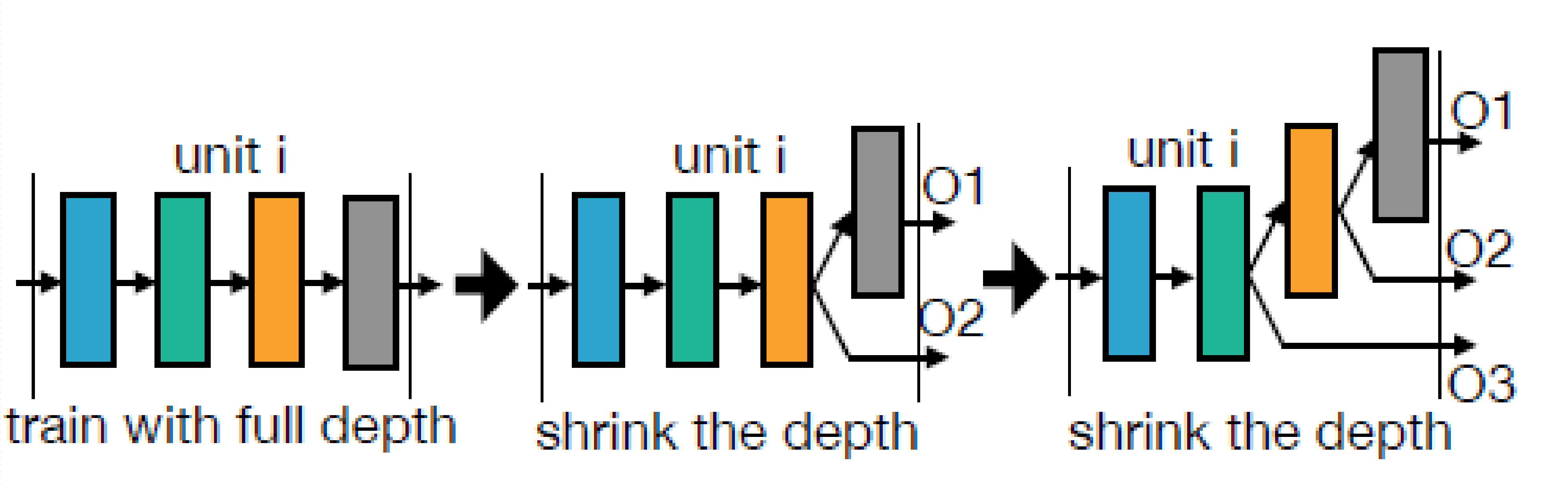
Elastic Kernel Size

- The sub-kernels that close to the center are preserved in different networks
  - Transform matrix (trained)
  - Different sizes and distributions
  - 25x25, 9x9 size MLP



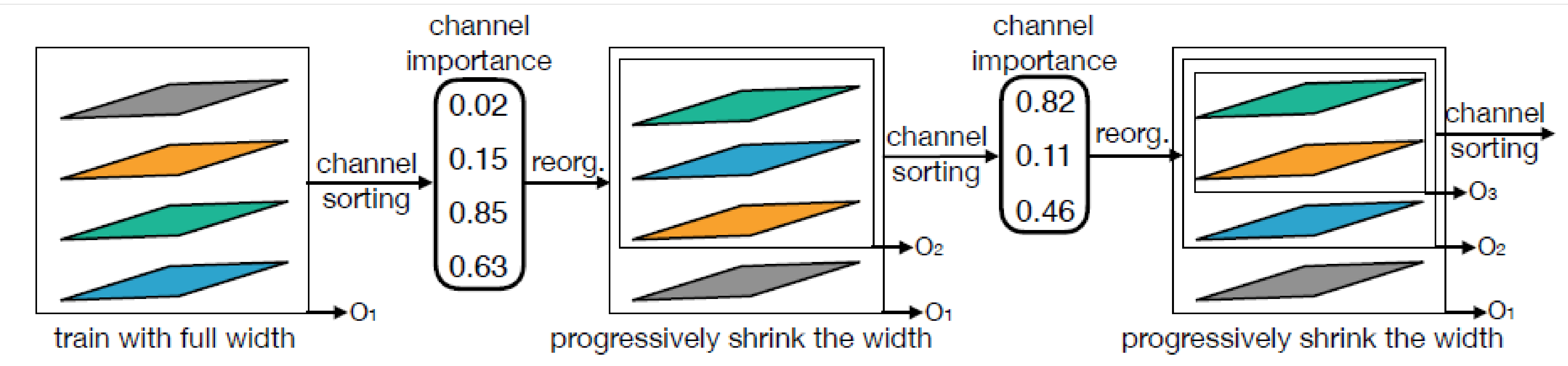
Elastic Depth

- Origin: N layers
  Target: D layers
  - Keep the first D layers, and skip the last N-D layers
  - The previous layers will be shared among models of different sizes



Elastic Width

- Channel sorting operation
  - Sort by importance of different channels (L1 norm of channels weight)
  - Preserve the accuracy of larger sub-networks



Experiments (vs. EfficientNet)

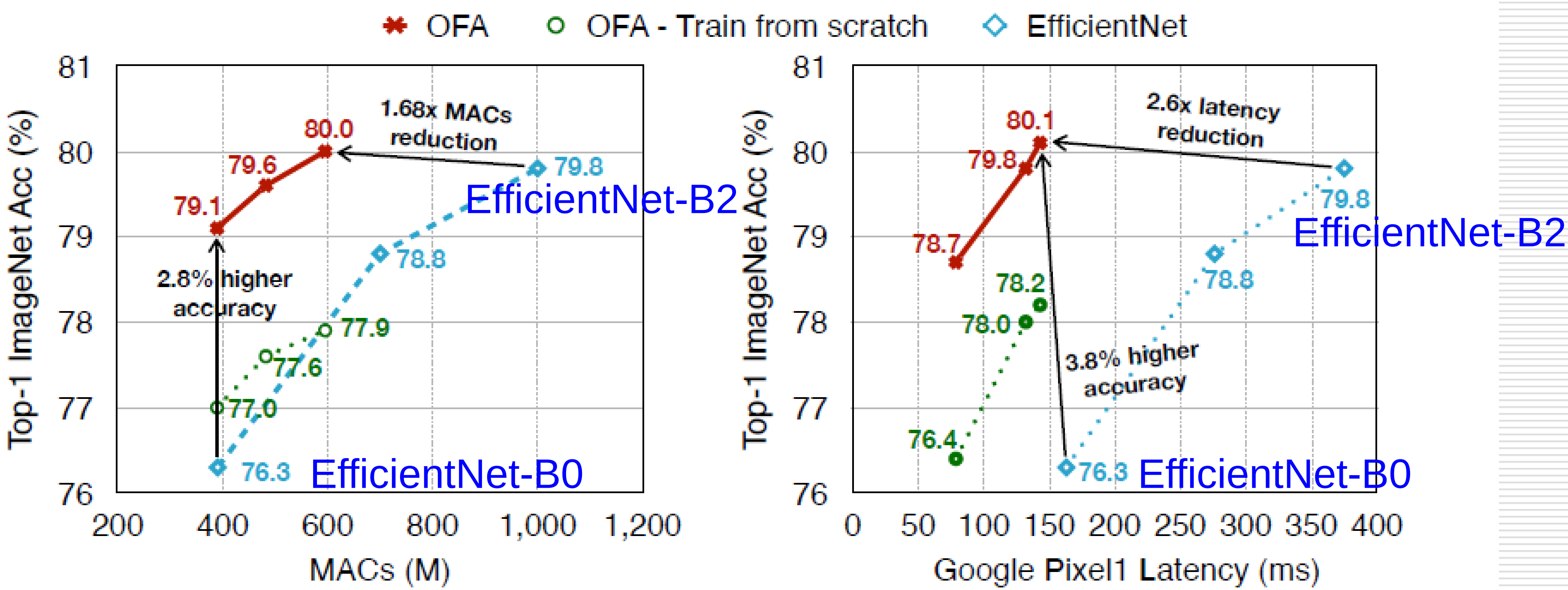


Figure 9: OFA achieves 80.0% top1 accuracy with 595M MACs and 80.1% top1 accuracy with 143ms Pixel1 latency, setting a new SOTA ImageNet top1 accuracy on the mobile setting.

Case Study

Example (HarDNet-39DS) (1/2)

- HarDNet
  - main.py (training API)
  - hardnet.py (model)
  - Confirm that main.py has imported the correct model file

```
=> loading checkpoint 'checkpoint.pth.tar'
=> loaded checkpoint 'checkpoint.pth.tar' (epoch 131)
Parameters= 3529270
Test: [  0/196] Time: 22.214    Loss: 7.5311e-01        Acc@1:  79.30   Acc@5:  95.31
Test: [ 10/196] Time:  0.100    Loss: 9.1498e-01        Acc@1:  76.74   Acc@5:  92.79
Test: [ 20/196] Time:  0.178    Loss: 9.3457e-01        Acc@1:  76.26   Acc@5:  92.56
Test: [ 30/196] Time:  0.090    Loss: 8.9900e-01        Acc@1:  77.37   Acc@5:  92.80
Test: [ 40/196] Time:  0.114    Loss: 9.6639e-01        Acc@1:  75.09   Acc@5:  92.62
Test: [ 50/196] Time:  0.515    Loss: 9.4871e-01        Acc@1:  75.08   Acc@5:  93.05
Test: [ 60/196] Time:  0.100    Loss: 9.6274e-01        Acc@1:  74.71   Acc@5:  93.01
Test: [ 70/196] Time:  0.258    Loss: 9.4381e-01        Acc@1:  75.19   Acc@5:  93.17
Test: [ 80/196] Time:  3.000    Loss: 9.6088e-01        Acc@1:  74.92   Acc@5:  92.94
Test: [ 90/196] Time:  0.217    Loss: 1.0185e+00        Acc@1:  73.83   Acc@5:  92.15
Test: [100/196] Time:  0.100    Loss: 1.0806e+00        Acc@1:  72.56   Acc@5:  91.34
Test: [110/196] Time:  0.188    Loss: 1.1058e+00        Acc@1:  72.09   Acc@5:  90.90
Test: [120/196] Time:  0.101    Loss: 1.1330e+00        Acc@1:  71.66   Acc@5:  90.42
Test: [130/196] Time:  2.417    Loss: 1.1664e+00        Acc@1:  70.88   Acc@5:  90.01
Test: [140/196] Time:  0.095    Loss: 1.1925e+00        Acc@1:  70.37   Acc@5:  89.72
Test: [150/196] Time:  0.183    Loss: 1.2155e+00        Acc@1:  69.90   Acc@5:  89.35
Test: [160/196] Time:  2.494    Loss: 1.2349e+00        Acc@1:  69.56   Acc@5:  89.03
Test: [170/196] Time:  1.606    Loss: 1.2578e+00        Acc@1:  69.03   Acc@5:  88.73
Test: [180/196] Time:  0.170    Loss: 1.2733e+00        Acc@1:  68.76   Acc@5:  88.50
Test: [190/196] Time:  0.100    Loss: 1.2742e+00        Acc@1:  68.67   Acc@5:  88.51
 * Acc@1 68.858 Acc@5 88.600
```

Example (HarDNet-39DS) (2/2)

- Download model from github
  - git clone xxxxxxxxxxxxxxx

```
(base) [M112tychang@adar10 M112tychang]$ git clone https://github.com/PingoLH/Pytorch-HarDNet.git
Cloning into 'Pytorch-HarDNet'...
remote: Enumerating objects: 168, done.
remote: Counting objects: 100% (30/30), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 168 (delta 25), reused 15 (delta 15), pack-reused 138
Receiving objects: 100% (168/168), 197.06 MiB | 16.99 MiB/s, done.
Resolving deltas: 100% (68/68), done.
(base) [M112tychang@adar10 M112tychang]$ ls
```

Argument Parser

```
parser.add_argument('-a', '--arch', metavar='ARCH', default='hardnet39ds',
                    choices=model_names,
                    help='model architecture: ' +
                        ' | '.join(model_names) +
                        ' (default: hardnet39ds)')
parser.add_argument('-b', '--batch-size', default=256, type=int,
                    metavar='N',
                    help='mini-batch size (default: 256), this is the total '
                         'batch size of all GPUs on the current node when '
                         'using Data Parallel or Distributed Data Parallel')
parser.add_argument('-e', '--evaluate', dest='evaluate', action='store_true',
                    help='evaluate model on validation set')
parser.add_argument('--pretrained', dest='pretrained', action='store_true',
                    help='use pre-trained model')
parser.add_argument('--resume', default='', type=str, metavar='PATH',
                    help='path to latest checkpoint (default: none)')
```

Data Loader (1/2)

- Data Pre-processing
  - Resize
  - Center crop
  - Numpy to tensor

```
# Data loading code
traindir = os.path.join(args.data, 'train')
valdir = os.path.join(args.data, 'val')
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])

train_dataset = datasets.ImageFolder(
    traindir,
    transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        normalize,
    ]))

train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=args.batch_size, shuffle=(train_sampler is None),
    num_workers=args.workers, pin_memory=True, sampler=train_sampler)
```

Data Loader (2/2)

- Data Pre-processing
  - Center crop
  - Numpy to tensor

```
val_loader = torch.utils.data.DataLoader(
    datasets.ImageFolder(valdir, transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        normalize,
    ])),
    batch_size=args.batch_size, shuffle=False,
    num_workers=args.workers, pin_memory=True)
```

Criterion and Optimizer

- Loss function
  - Cross entropy
- Optimizer
  - SGD with momentum

```
# define loss function (criterion) and optimizer
criterion = nn.CrossEntropyLoss().cuda(args.gpu)


optimizer = torch.optim.SGD(model.parameters(), args.lr,
                            momentum=args.momentum,
                            nesterov=True,
                            weight_decay=args.weight_decay)
```

Model Creation

```
print("=> creating model '{}'".format(args.arch))
model = HarDNet(depth_wise, arch, pretrained=False)
print(model)
```

```
HarDNet(
  (base): ModuleList(
    (0): ConvLayer(
      (conv): Conv2d(3, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (norm): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU6(inplace=True)
    )
    (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (2): ConvLayer(
      (conv): Conv2d(24, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (norm): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU6(inplace=True)
    )
    (3): DWConvLayer(
      (dwconv): Conv2d(48, 48, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=48, bias=False)
      (norm): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (4): HarDBlock(
      (layers): ModuleList(
        (0): CombConvLayer(
          (layer1): ConvLayer(
            (conv): Conv2d(48, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (norm): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): ReLU6(inplace=True)
```

Module vs. Sequential (1/2)

- nn.Module

```
class HarDBlock(nn.Module):
    def __init__(self, in_channels, growth_rate, grmul, n_layers, keepBase=False, residual_out=False, dwconv=False):
        super().__init__()
        self.keepBase = keepBase
        self.links = []
        layers_ = []
        self.out_channels = 0 # if upsample else in_channels
        for i in range(n_layers):
          outch, inch, link = self.get_link(i+1, in_channels, growth_rate, grmul)
          self.links.append(link)
          use_relu = residual_out
```

- nn.Sequential

```
class ConvLayer(nn.Sequential):
    def __init__(self, in_channels, out_channels, kernel=3, stride=1, dropout=0.1, bias=False):
        super().__init__()

        self.add_module('conv', nn.Conv2d(in_channels, out_channels, kernel_size=kernel,
                                          stride=stride, padding=kernel//2, groups=1, bias=bias))
        self.add_module('norm', nn.BatchNorm2d(out_channels))
        self.add_module('relu', nn.ReLU6(True))
    def forward(self, x):
        return super().forward(x)
```

Module vs. Sequential (2/2)

- nn.Module
  - Add any subclass of nn.Module to the list
  - Define different layers in no order
  - Define the order between layers on your own

- nn.Sequential
  - layers must be executed in order
  - Ensure the output channels of the previous layer are the same as the input channels of the next layer

Model Invocation

- Use the whole model

```
# compute output
output = model(input)
loss = criterion(output, target)
```

- Use one of layers (debug)
  - Not recommended

```
output=model.base[1](output)
output=model.base[2](output)
output=model.base[3](output)
output=model.base[4](output)
output=model.base[5](output)
```

Checkpoint

- Read parameters
  - torch.load()
  - model.load_state_dict()

```
checkpoint = torch.load('checkpoint.pth.tar')

for ele in checkpoint['state_dict']:
    print(ele)
```

```
model.load_state_dict(checkpoint['state_dict'])
optimizer.load_state_dict(checkpoint['optimizer'])
```

- Save parameters
  - torch.save()
  - model.state_dict()

```
state = {
            'epoch'      : epoch + 1,
            'arch'       : args.arch,
            'state_dict': model.state_dict(),
            'best_acc1'  : best_acc1,
            'optimizer'  : optimizer.state_dict(),
        }
torch.save(state, 'checkpoint.pth.tar')
```

Parallelism (1/4)

- Data Parallel
  - Single-process, multi-thread
  - Divide training data into one or more subsets and then distribute them to different computing units for execution
  - Copy the neural network model to different computing units
  - After the calculations are completed, data will be sent back to the main computing unit for updating, and the model will be updated uniformly. And then copied out.
- Model Parallel
  - Multi-process
  - The model is divided into several small models that can be executed in different GPUs independently

Parallelism (2/4)

- DistributedDataParallel
  - multi-process, multi-thread
  - Similar to data parallel, data is divided into different computing units; the model is also copied to the different computing units
  - After the calculations, parameters do not return to the main computing unit for updating. Only the gradients will be passed to each computing unit for updating

Data Parallelism (3/4)

- Without Data Parallel
  - Model

```
HarDNet(
  (base): ModuleList(
    (0): ConvLayer(
      (conv): Conv2d(3, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (norm): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU6(inplace=True)
    )
    (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (2): ConvLayer(
      (conv): Conv2d(24, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (norm): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU6(inplace=True)
    )
    (3): DWConvLayer(
      (dwconv): Conv2d(48, 48, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=48, bias=False)
      (norm): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
```

```
base.0.conv.weight
base.0.norm.weight
base.0.norm.bias
base.0.norm.running_mean
base.0.norm.running_var
base.0.norm.num_batches_tracked
base.2.conv.weight
base.2.norm.weight
base.2.norm.bias
```

Data Parallelism (4/4)

- Data Parallel
  - Model

```
DataParallel(
  (module): HarDNet(
    (base): ModuleList(
      (0): ConvLayer(
        (conv): Conv2d(3, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (norm): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU6(inplace=True)
      )
      (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (2): ConvLayer(
        (conv): Conv2d(24, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU6(inplace=True)
      )
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
```

```
module.base.0.conv.weight
module.base.0.norm.weight
module.base.0.norm.bias
module.base.0.norm.running_mean
module.base.0.norm.running_var
module.base.0.norm.num_batches_tracked
module.base.2.conv.weight
module.base.2.norm.weight
module.base.2.norm.bias
module.base.2.norm.running_mean
```

Exercise

- 1 (50%) : Please explain the pros and cons of “regular convolution” and “depth-wise separable convolution” ?
  - List at least 3 pros and cons respectively
  - What’s the main reason for us to use dw-separable convolution rather than regular convolution sometimes?
- 2 (50%): Please report the parameters of AlexNet by manual calculations. Show the actual “FLOPS / parameters” reported by code. Attached with Screenshot.
- 3 (5%) : With hw4.py, train a CNN-based model without pre-trained weights. (Dataset: MNIST)
  - Please provide some images about your exercise
  - Give a short summary of why you choose the model and how to improve and implement it.
- Please submit your Report as hw4.pdf file.

References (1/3)

- ResNet
  - Paper: https://arxiv.org/pdf/1512.03385.pdf
  - https://github.com/kuangliu/pytorch-cifar/blob/master/models/resnet.py

- MobileNet
  - Paper: https://arxiv.org/pdf/1704.04861.pdf
  - https://github.com/wjc852456/pytorch-mobilenet-v1

- VGG-16
  - Paper: https://arxiv.org/pdf/1409.1556.pdf
  - https://github.com/ashushekar/VGG16

References (2/3)

- DataParallel
  - https://ithelp.ithome.com.tw/articles/10226382
- Module and Sequential
  - https://zhuanlan.zhihu.com/p/64990232
- DenseNet
  - https://zhuanlan.zhihu.com/p/37189203
- HarDNet
  - https://github.com/PingoLH/Pytorch-HarDNet

References (3/3)

- Network In Network
  - https://arxiv.org/pdf/1312.4400.pdf
- GoogLeNet (Inception-V1，2014)
  - https://wmathor.com/usr/uploads/2020/01/3184187721.pdf
- EfficientNet
  - https://arxiv.org/pdf/1905.11946.pdf
- Convolutional Neural Networks (台大李弘毅)
  - https://www.youtube.com/watch?v=OP5HcXJg2Aw&list=PLJV_el3uVTsMhtt7_Y6sgTHGHp1Vb2P2J&index=9

Appendix

Python Modules Import

Packages for Model Structure and FLOPS

Packages torchsummary & thop

- Package torchsummary
  - Installation : pip install torchsummary
  - Function : Show the structure of a PyTorch model
- Package thop
  - Installation : pip install thop
  - Function : Calculate the FLOPS of a PyTorch model

Installing Packages

- Commands

```
chmod +x install_packages.sh
./install_packages.sh
```

info.py

- **Model structure**

```
from torchsummary import summary
summary(model, (1,32,32))
```

- **FLOPS**

```
from thop import profile
flops, params = profile(model, inputs=(torch.randn(1,1,32,32).to(device),))
print("FLOPS: ", flops, " / Params: ", params)
```

Thank you