Deep Learning & Engineering Applications

4. Modern ConvNet Architectures

JIDONG J. YANG

COLLEGE OF ENGINEERING

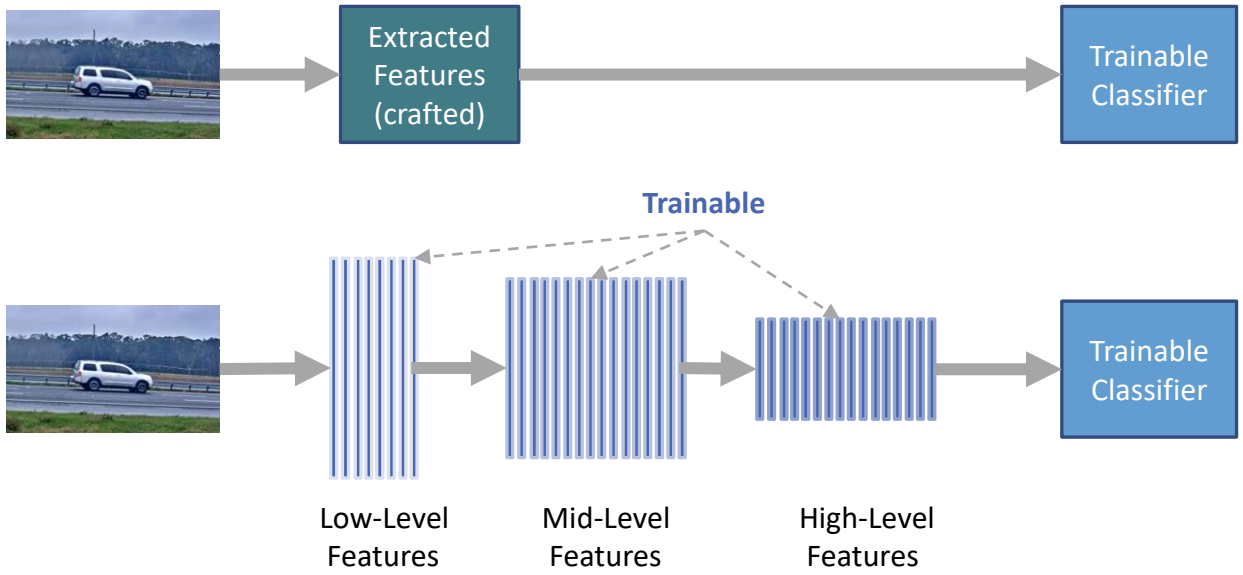
UNIVERSITY OF GEORGIA

Rising of Deep Learning

Before 2012, the handcrafted features/representations are used.
- SIFT [Lowe, 2004], SURF [Bay et al., 2006], HOG [Dalal & Triggs, 2005], etc.

Large DataSet (e.g., ImageNet, 2009)

Graphical processing units (GPUs)



Modern ConvNets

AlexNet [Krizhevsky et al., 2012]

VGG [Simonyan & Zisserman, 2014]

NiN [Lin et al., 2013]

GoogLeNet [Szegedy et al., 2015]

ResNet [He et al., 2016]

DenseNet [Huang et al., 2017]

EfficientNet [Tan and Le. 2019]

AlexNet

Trained on two NVIDIA GTX 580s with 3GB of memory.

Offered empirical evidence that deep CNNs work.

Differences from LeNet:

- Deeper than LeNet
  - 8 layers (5 conv layers + 3 FC layers) vs 5 layers (2 conv layers + 3 FC layers)
- Use ReLU instead of Sigmoid
- Use Max Pooling instead of Avg Pooling

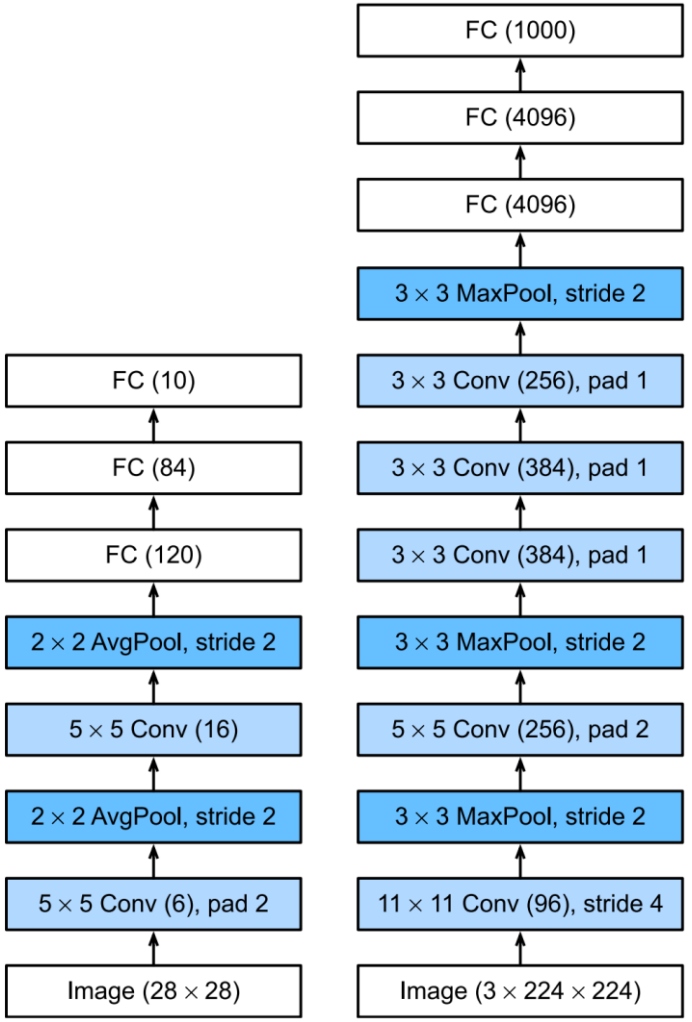


Fig. 7.1.2: From LeNet (left) to AlexNet (right).

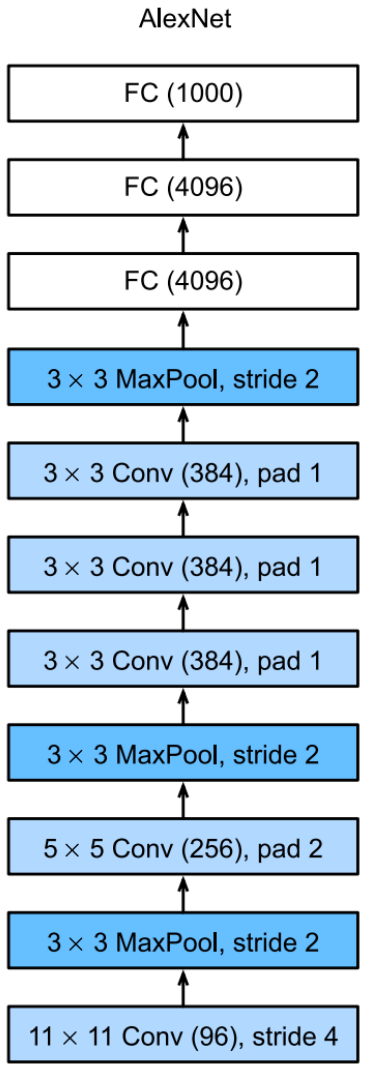
VGG

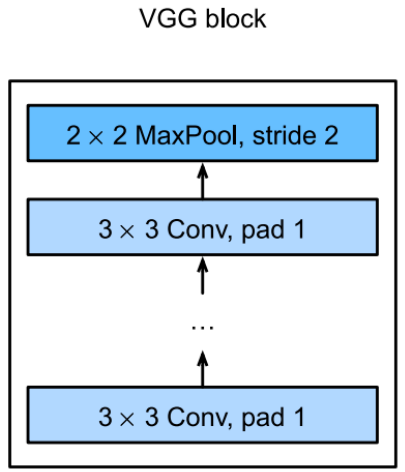
The idea of using blocks first emerged from the Visual Geometry Group (VGG) at Oxford University.

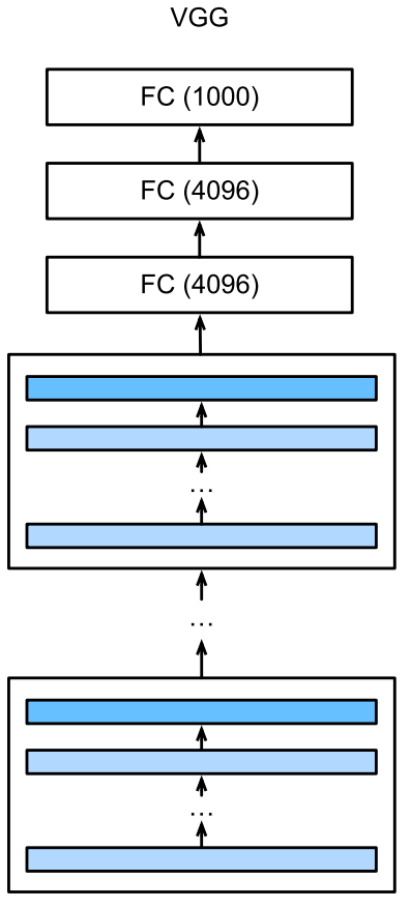
The basic building block of classic CNNs is a sequence of following:

- Conv layers (with padding to maintain the resolution)
- Nonlinearity (e.g., ReLU)
- Pooling (e.g., Max Pooling)

VGG paper [Simonyan & Zisserman, 2014]







VGG-16

Figure credit: https://www.kaggle.com/blurredmachine/vggnet-16-architecture-a-complete-guide

13 conv layers

3 FC layers

3x3 kernel with pad 1

2x2 Max Pooling with stride 2

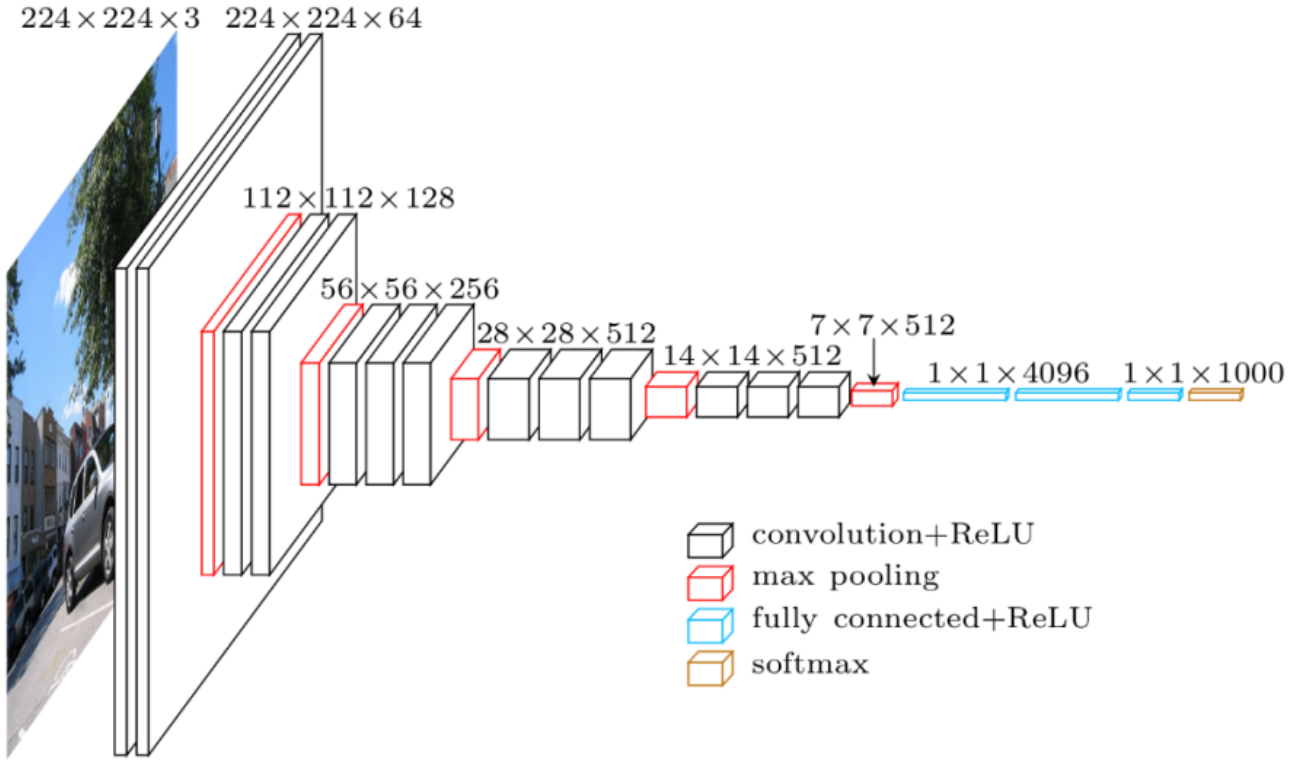


Image credit: https://www.researchgate.net/publication/328966158_A_review_of_deep_learning_in_the_study_of_materials_degradation

VGG-16

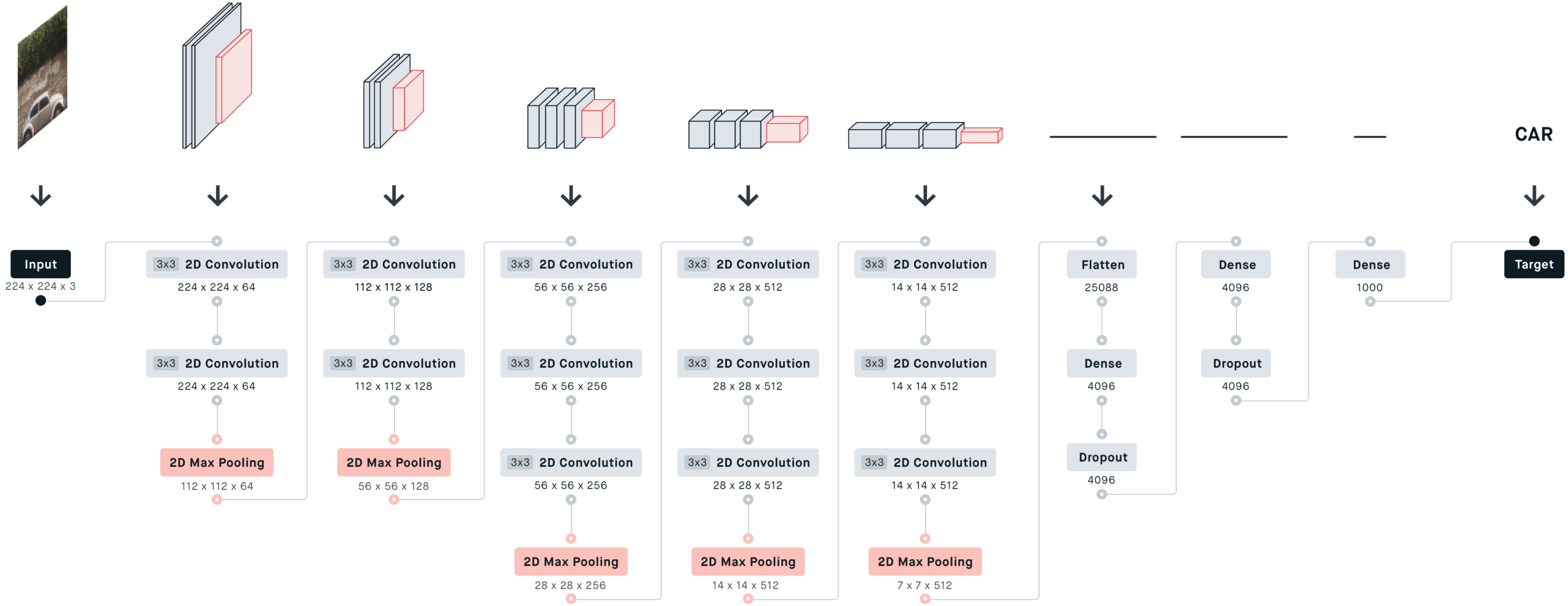


Figure credit: https://www.kaggle.com/blurredmachine/vggnet-16-architecture-a-complete-guide

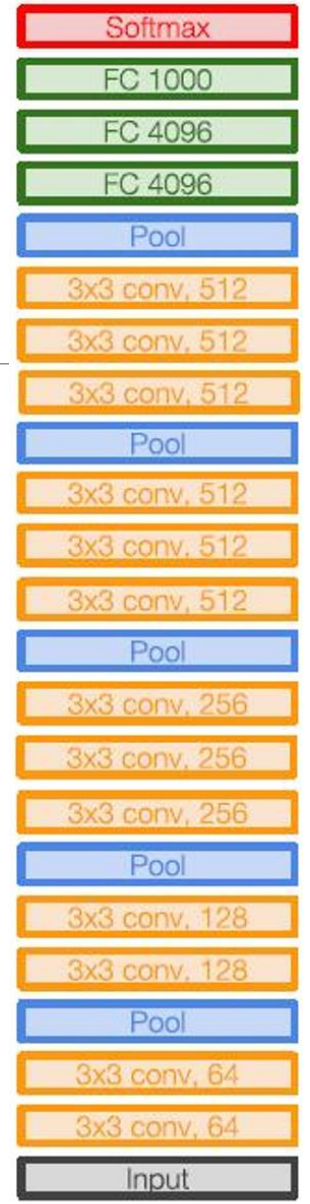
VGG-16

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Source: http://cs231n.stanford.edu/slides/2021/lecture_9.pdf



VGG-16

VGG-19

Additional conv layers were added in the upper blocks.

Adapted from http://cs231n.stanford.edu/slides/2021/lecture_9.pdf

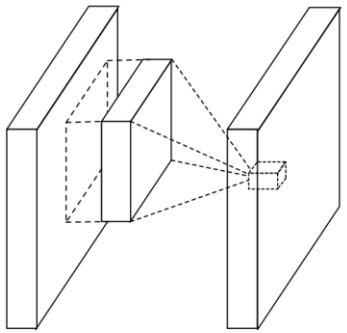
Network in Network (NiN)

Introduce 1×1 convolutional layers, i.e., per-pixel FC layers with ReLU activations (“Mlpconv layers”), referred to as “micro network”.

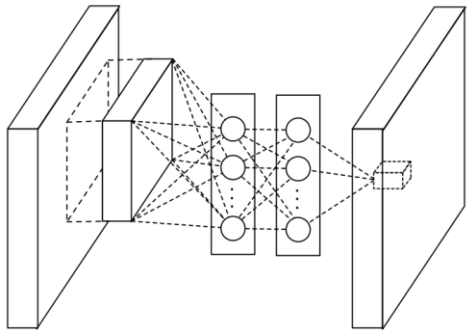
The classification layer uses global average pooling, which is easier to interpret and less prone to overfitting than FC layer.

Enforce correspondences between feature maps and categories. The feature maps can be easily interpreted as categories confidence maps.

Global average pooling sums out the spatial information, thus it is more robust to spatial translations of the input.

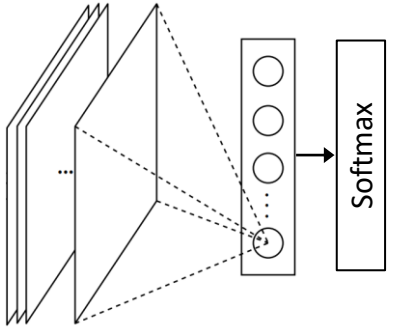


(a) Linear convolution layer

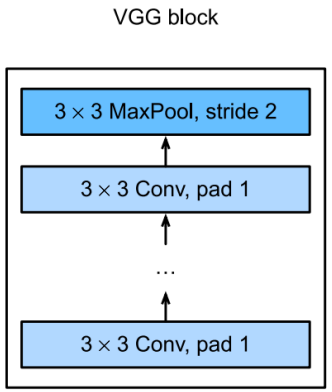


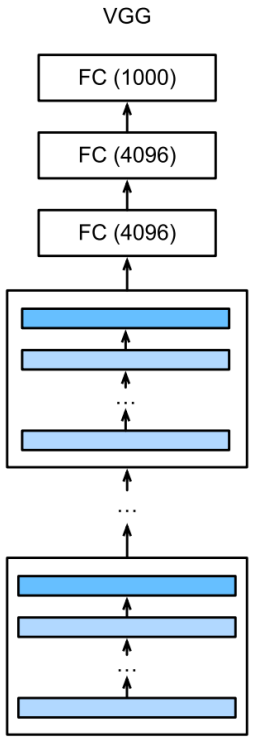
(b) Mlpconv layer

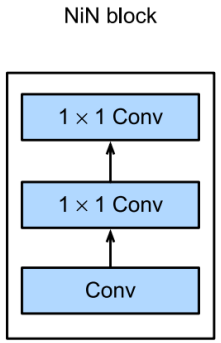
[Lin et al., 2013] https://arxiv.org/abs/1312.4400

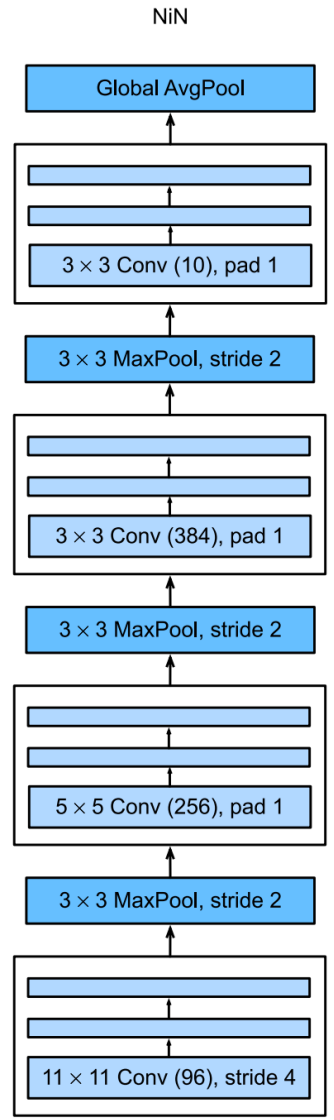


Global AvgPool









GoogLeNet

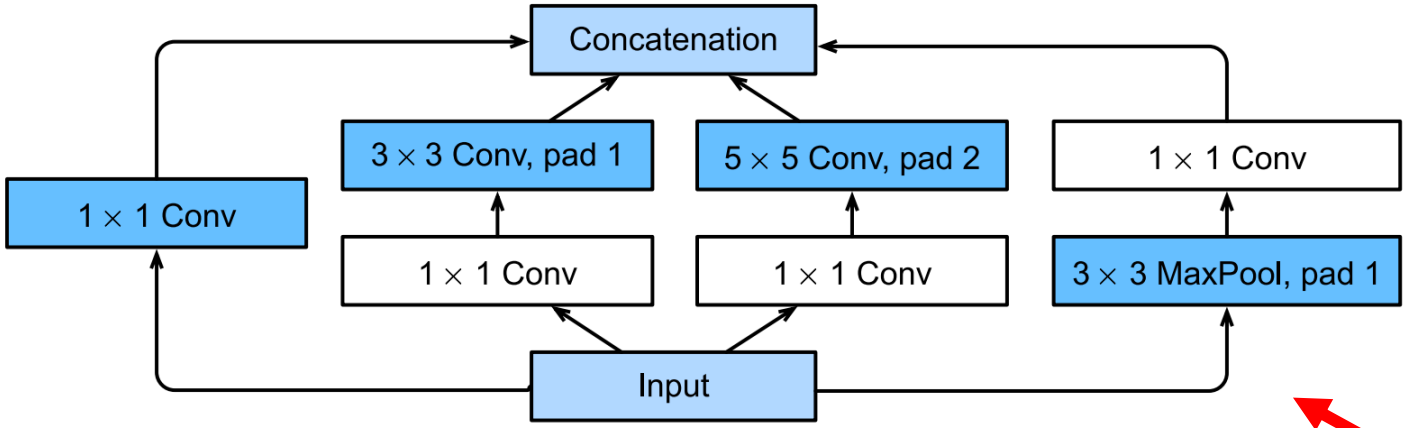
Combine the strengths of NiN and paradigms of repeated blocks.

Introduce Inception block

- Employ a combination of variously-sized kernels (1x1, 3x3, 5x5).
- The output of all four paths have same spatial dimension (w x h).
- The concatenation is along the channel dimension.

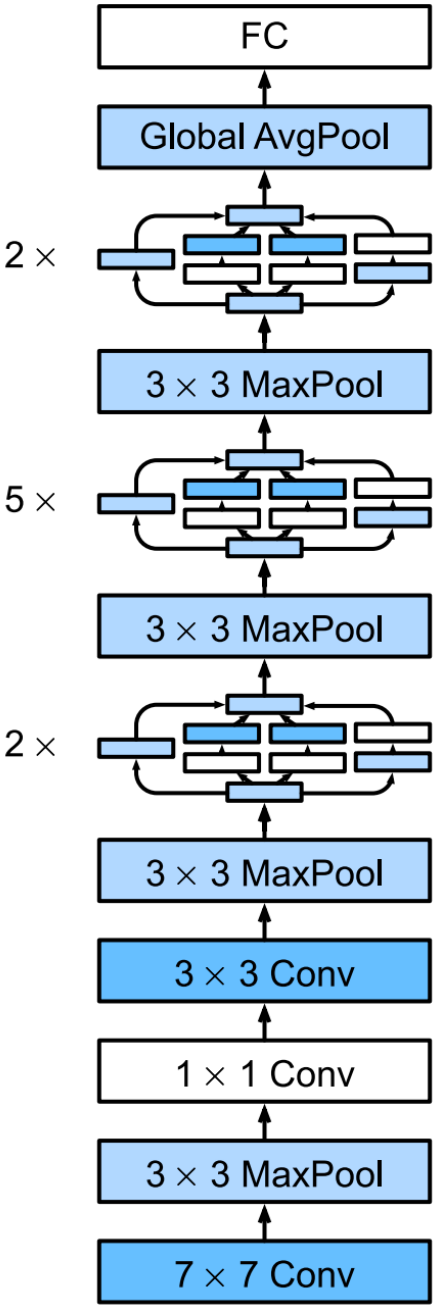
Maximum pooling between inception blocks reduces the dimensionality.

Global average pooling at the end avoids fully-connected layers.



Inception Block

Split-Transform-Merge Strategy



[Szegedy et al., 2014] https://arxiv.org/abs/1409.4842

Batch Normalization (BN)

Before we talk about the next modern network architecture, ResNet, we need to know Batch Normalization (BN).

$$\mathrm{BN}(\mathbf{x}) = \boldsymbol{\gamma} \odot \frac{\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}}}{\hat{\boldsymbol{\sigma}}_{\mathcal{B}}} + \boldsymbol{\beta} \qquad \hat{\boldsymbol{\mu}}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|}\sum_{\mathbf{x}\in\mathcal{B}} \mathbf{x}, \quad \hat{\boldsymbol{\sigma}}^2_{\mathcal{B}} = \frac{1}{|\mathcal{B}|}\sum_{\mathbf{x}\in\mathcal{B}} (\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}})^2 + \epsilon$$

Where, $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are parameters, having the same shape as x. They need to be learned jointly with the other model parameters. (Note: the normalization can be cancelled by the scaling & shifting.)

- BN is typically applied after FC and Conv layers to scale/shift the output.
- BN permits much higher learning rates and make initialization less sensitive.
- BN acts as a regularizer, eliminating the need for dropout in some cases.
- BN behaves differently during training and at test time. Why?

[Ioffe & Szegedy, 2015] https://arxiv.org/pdf/1502.03167.pdf

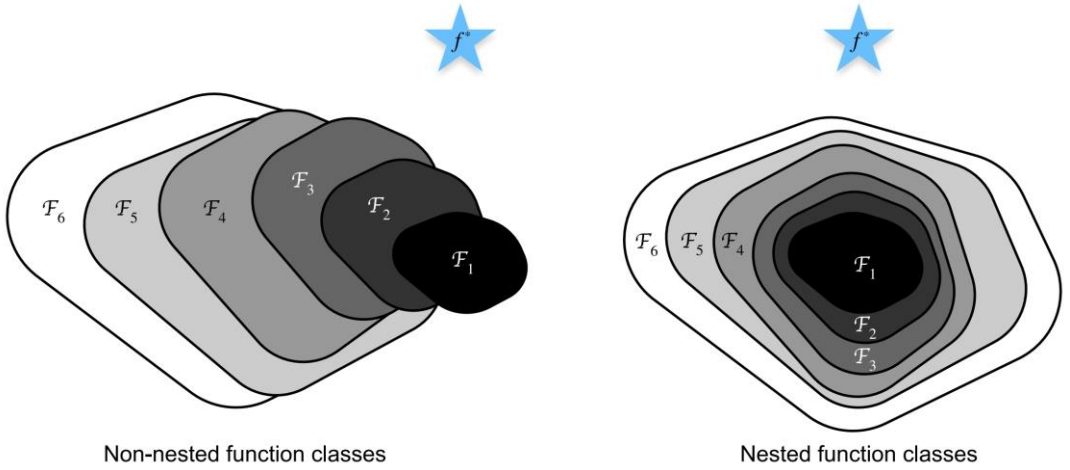
Residual Network (ResNet)

Think about $\mathcal{F}$ as the class of functions that a specific network architecture can reach.

Training a predefined neural network architecture can be thought as searching a function (weights and biases) in the function class.

$$f_{\mathcal{F}}^* \overset{\text{def}}{=} \underset{f}{\operatorname{argmin}} L(\mathbf{X}, \mathbf{y}, f) \text{ subject to } f \in \mathcal{F}$$

Will increasing $\mathcal{F}$ (make a network deeper) always move us closer to the "truth" function f^*?



Increased network depth results in worse performance.

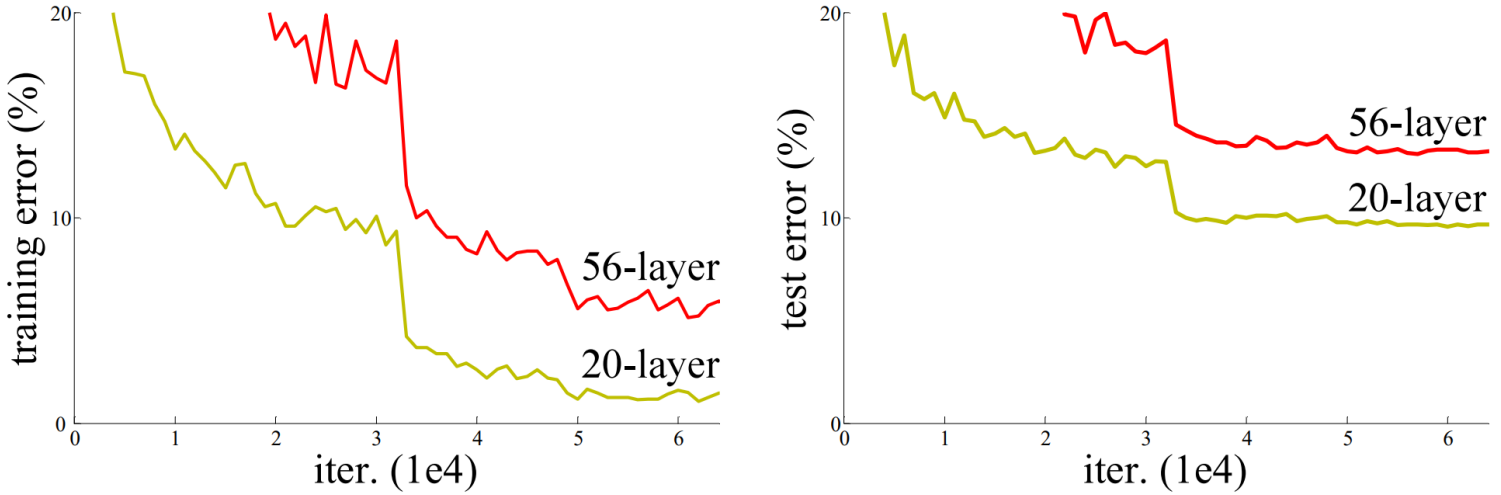


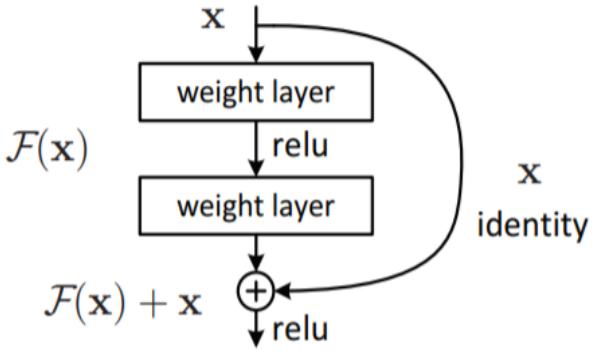
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

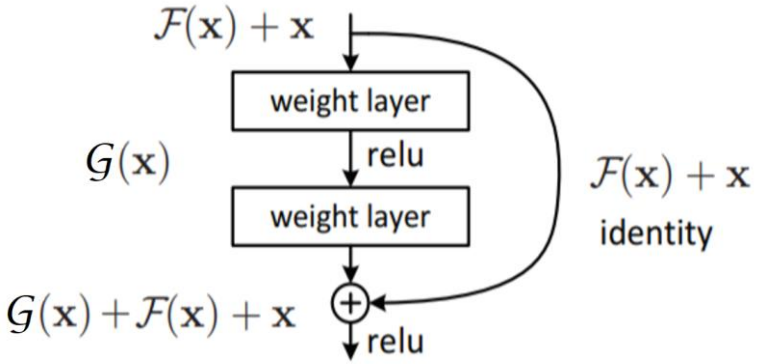
[He et al., 2015] https://arxiv.org/abs/1512.03385

ResNet

Only if larger function classes contain the smaller ones are we guaranteed that increasing them strictly increases the expressive power of the network.

At the heart of the residual network (ResNet) is the idea that every additional layer should more easily contain the identity function as one of its elements.

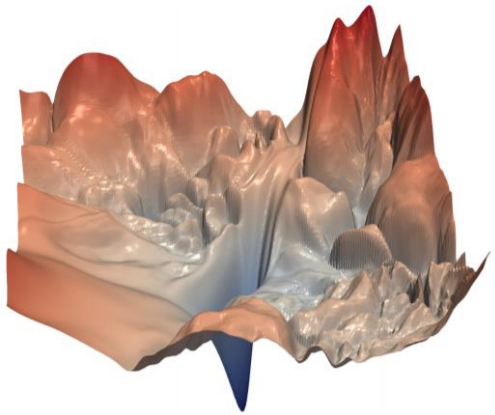




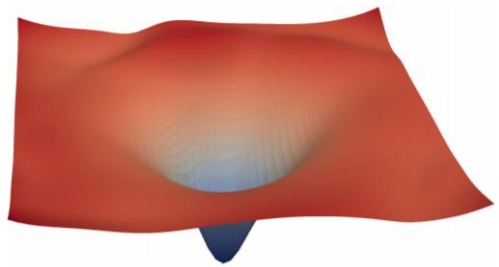
Similar ideas:

- Highway Networks introduced gated shortcut connections [https://arxiv.org/pdf/1507.06228.pdf]
- Long Term Short Memory (LSTM) cell uses a parameterized forget gate that controls how much information will flow to the next time step. [https://dl.acm.org/doi/10.1162/neco.1997.9.8.1735]

ResNet Architectures

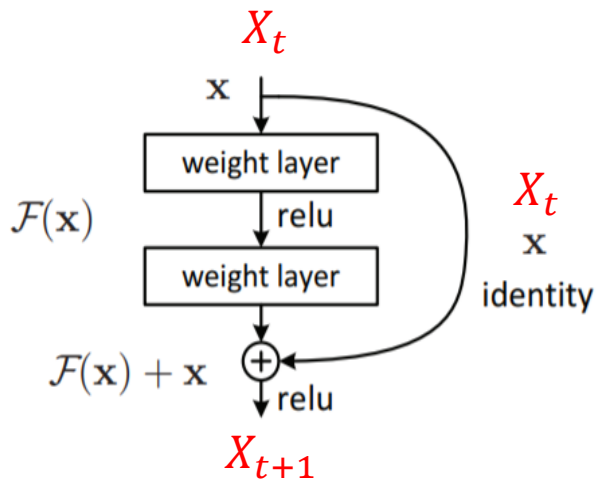


(a) without skip connections



(b) with skip connections

The loss surfaces of ResNet-56 with/without skip connections (Li, et al., 2017)



- Explored over 1,000 layers, which showed no optimization difficulty.
- The 1202-layer ResNet resulted in worse performance than its 110-layer counterpart.

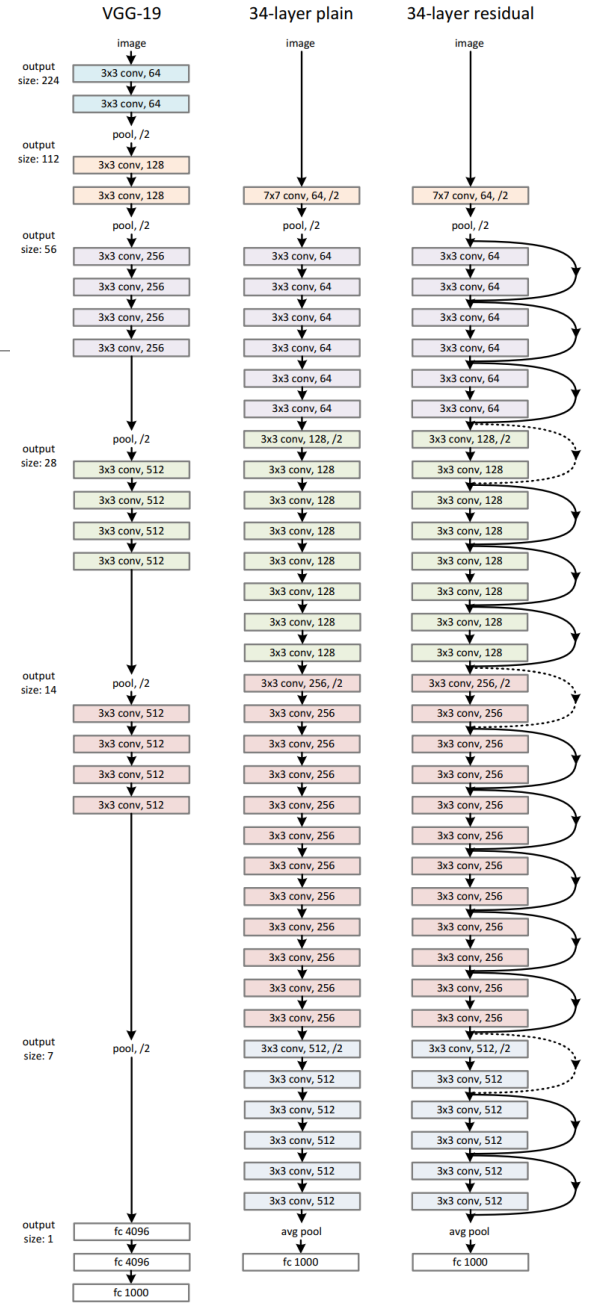
$$X_{t+1} = F(X_t) + X_t$$

$$F(X_t) = X_{t+1} - X_t$$

$$= \frac{X_{t+1} - X_t}{(t+1) - t}$$

Neural ODE

$$= \frac{\Delta X_t}{\Delta t} \approx \frac{dX_t}{dt}$$



[Li et al., 2017]
https://arxiv.org/abs/1712.09913

[He et al., 2015]
https://arxiv.org/abs/1512.03385

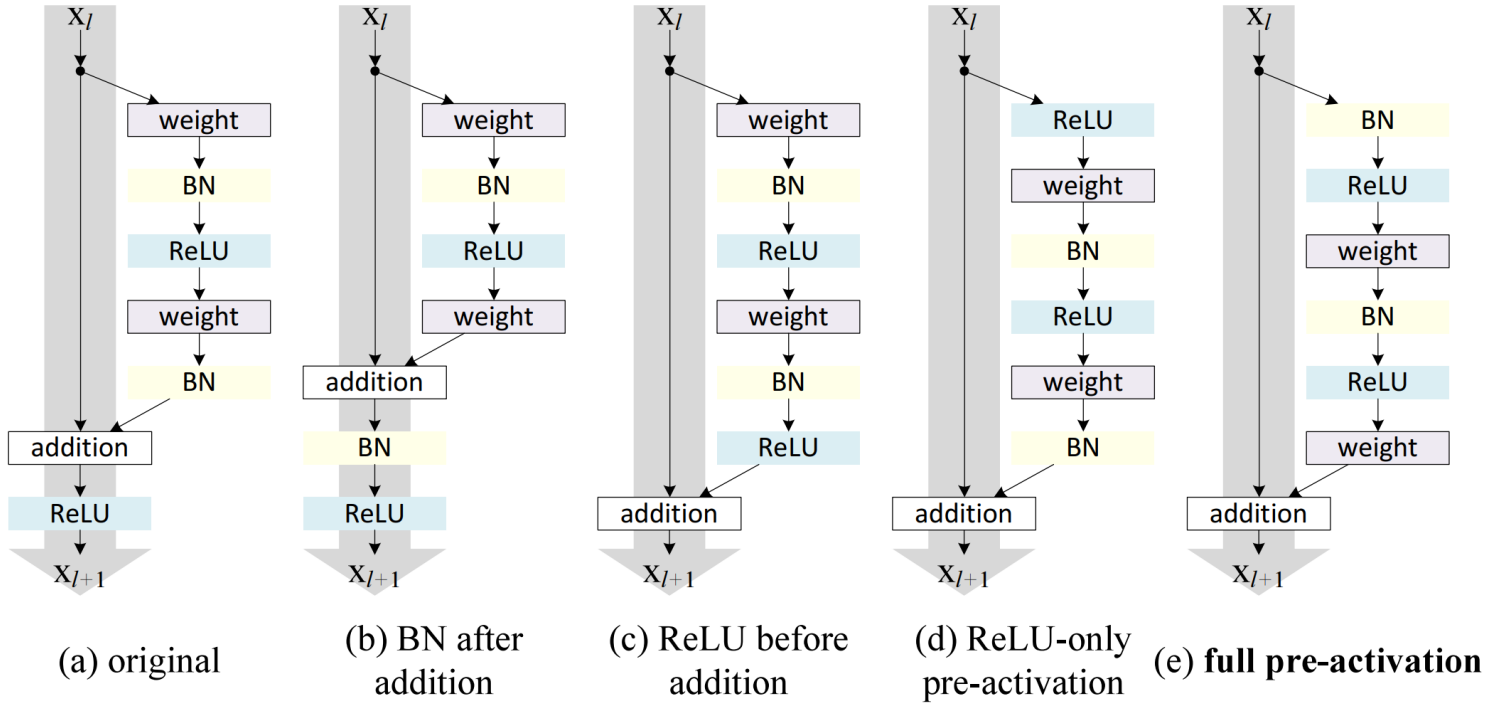
Identity Mapping in Deep ResNet

All these units consist of the same components — only the orders are different.

Pre-activation variant of residual block

The ResNet-1001 resulted in better performance than its shallower counterparts (ResNet-110 and 164).

case	Fig.	ResNet-110	ResNet-164
original Residual Unit [1]	Fig. 4(a)	6.61	5.93
BN after addition	Fig. 4(b)	8.17	6.50
ReLU before addition	Fig. 4(c)	7.84	6.14
ReLU-only pre-activation	Fig. 4(d)	6.71	5.91
full pre-activation	Fig. 4(e)	**6.37**	**5.46**

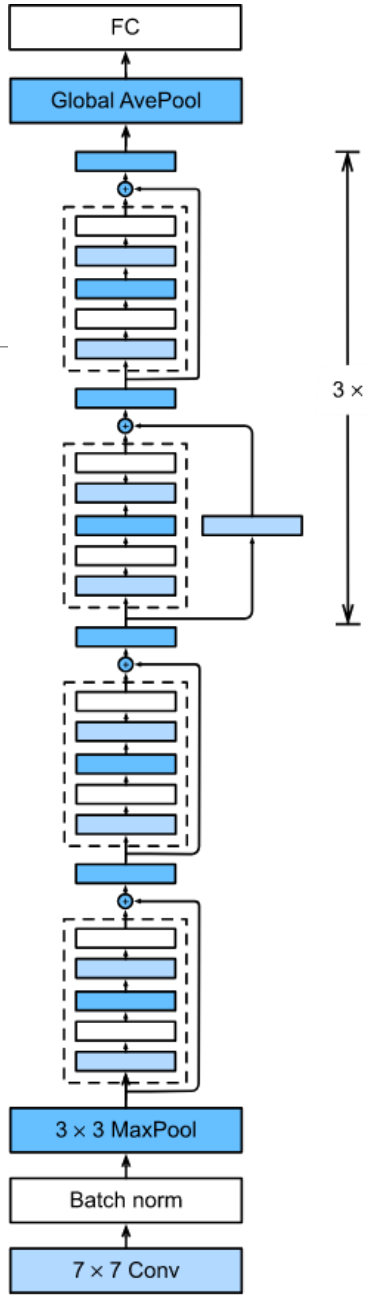


[He et al., 2016] https://arxiv.org/pdf/1603.05027.pdf

Example – ResNet 18

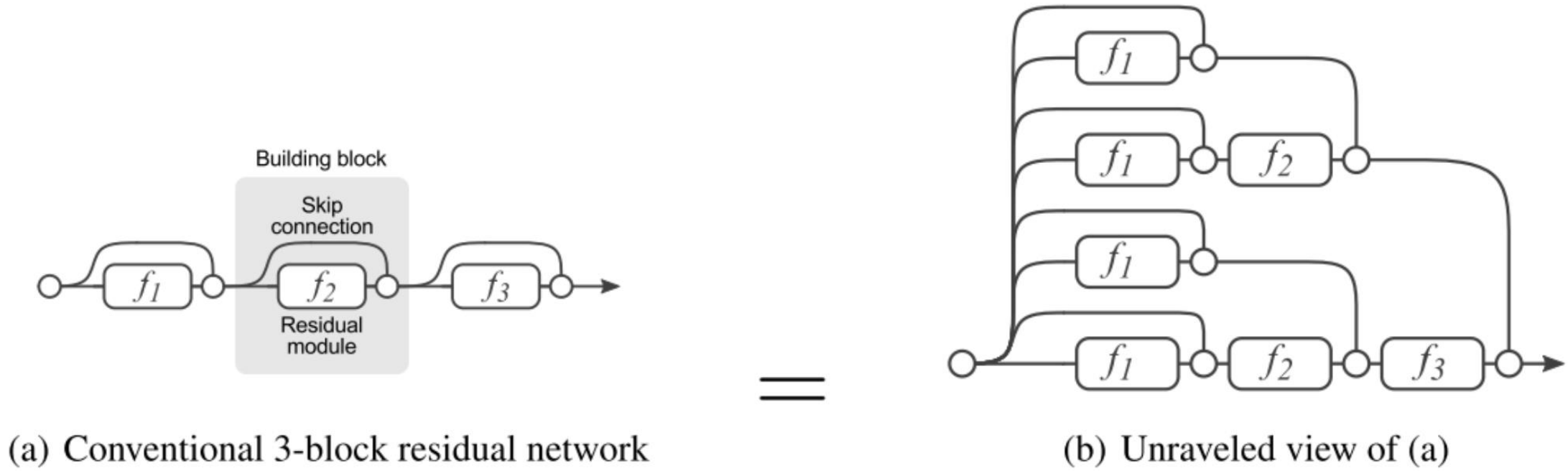
```
b1 = nn.Sequential(nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3),
                   nn.BatchNorm2d(64), nn.ReLU(),
                   nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
b2 = nn.Sequential(*resnet_block(64, 64, 2, first_block=True))
b3 = nn.Sequential(*resnet_block(64, 128, 2))
b4 = nn.Sequential(*resnet_block(128, 256, 2))
b5 = nn.Sequential(*resnet_block(256, 512, 2))

net = nn.Sequential(b1, b2, b3, b4, b5, nn.AdaptiveAvgPool2d((1, 1)),
                    nn.Flatten(), nn.Linear(512, 10))
```



ResNet as an Ensemble of Smaller Networks

Number of paths: 2^n, in this example, $2^3 = 8$



(a) Conventional 3-block residual network

(b) Unraveled view of (a)

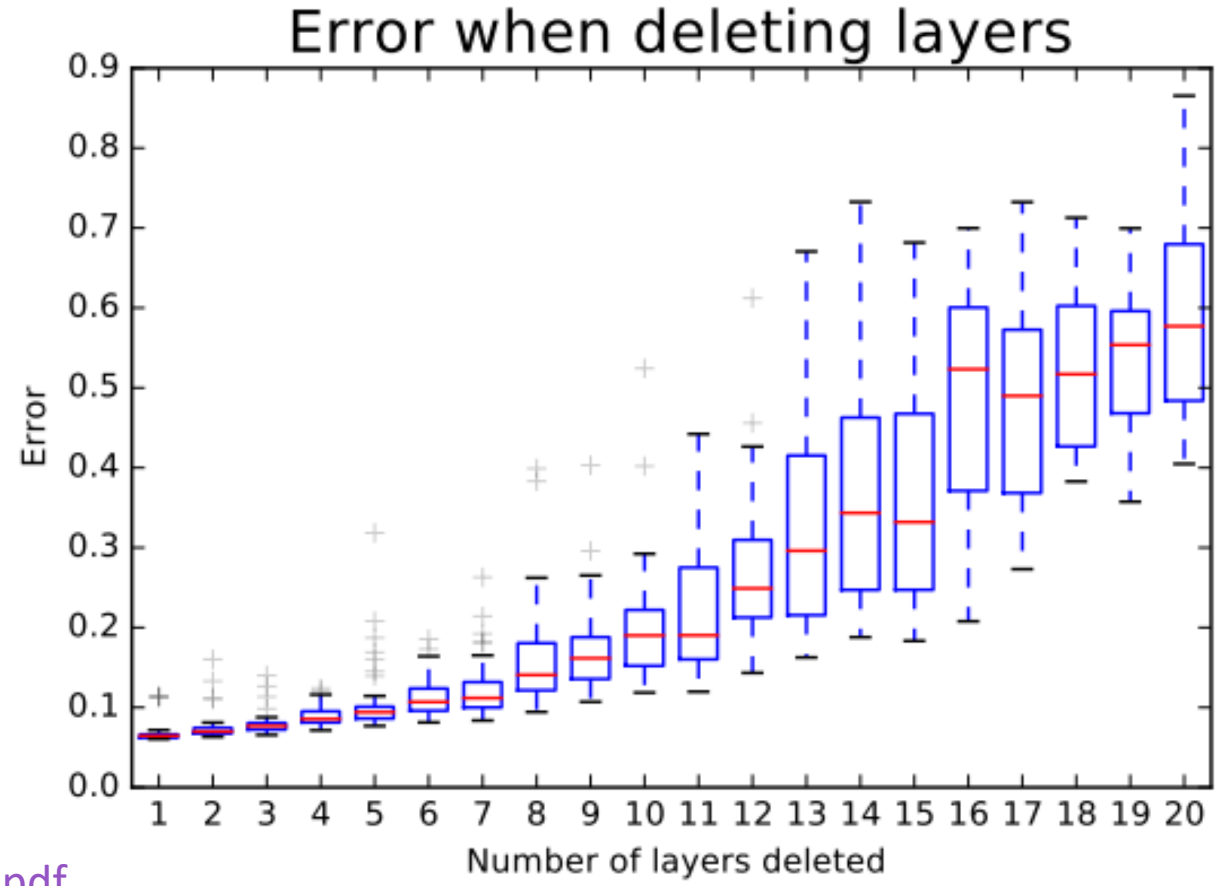
[Veit et al., 2016] https://arxiv.org/pdf/1605.06431.pdf

ResNet as an Ensemble of Smaller Networks

Conducted experiments by deleting different number of layers at test time.

The performance of the network smoothly correlates with the number of deleted layers.

The results suggest that the network indeed behaves like ensemble.



[Veit et al., 2016] https://arxiv.org/pdf/1605.06431.pdf

ResNet as an ensemble of smaller networks

It suggests that ResNet did not solve the vanishing gradients problem for very long paths, and that ResNet actually enables training very deep network by shortening its effective paths.

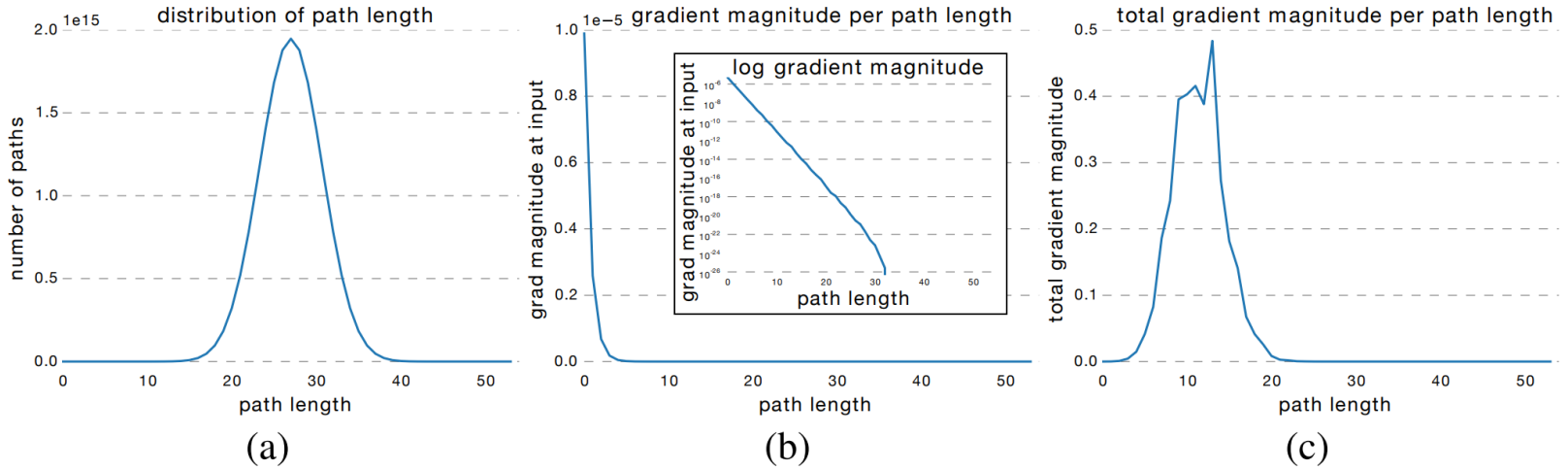


Figure 6: How much gradient do the paths of different lengths contribute in a residual network? To find out, we first show the distribution of all possible path lengths (a). This follows a Binomial distribution. Second, we record how much gradient is induced on the first layer of the network through paths of varying length (b), which appears to decay roughly exponentially with the number of modules the gradient passes through. Finally, we can multiply these two functions (c) to show how much gradient comes from all paths of a certain length. Though there are many paths of medium length, paths longer than $\sim$20 modules are generally too long to contribute noticeable gradient during training. This suggests that the effective paths in residual networks are relatively shallow.

[Veit et al., 2016] https://arxiv.org/pdf/1605.06431.pdf

ResNeXt

A variant of ResNet, inspired by Inception (Split-Transform-Merge).

ResNeXt modules perform a set of transformations, each on a low-dimensional embedding, whose outputs are aggregated by summation.

Reduced solution space (subspace) and computational complexity

Introduced a hyperparameter called cardinality (i.e., the number of paths) to provide a new way of adjusting the model capacity.

Experiments showed that accuracy can be gained more efficiently by increasing the cardinality than by going deeper or wider.

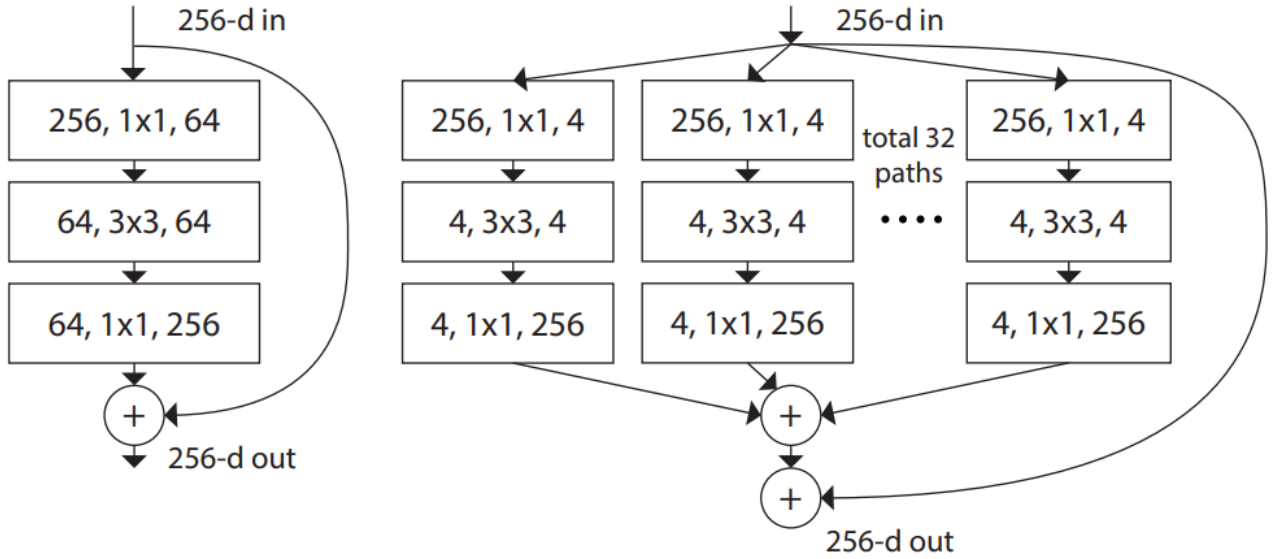
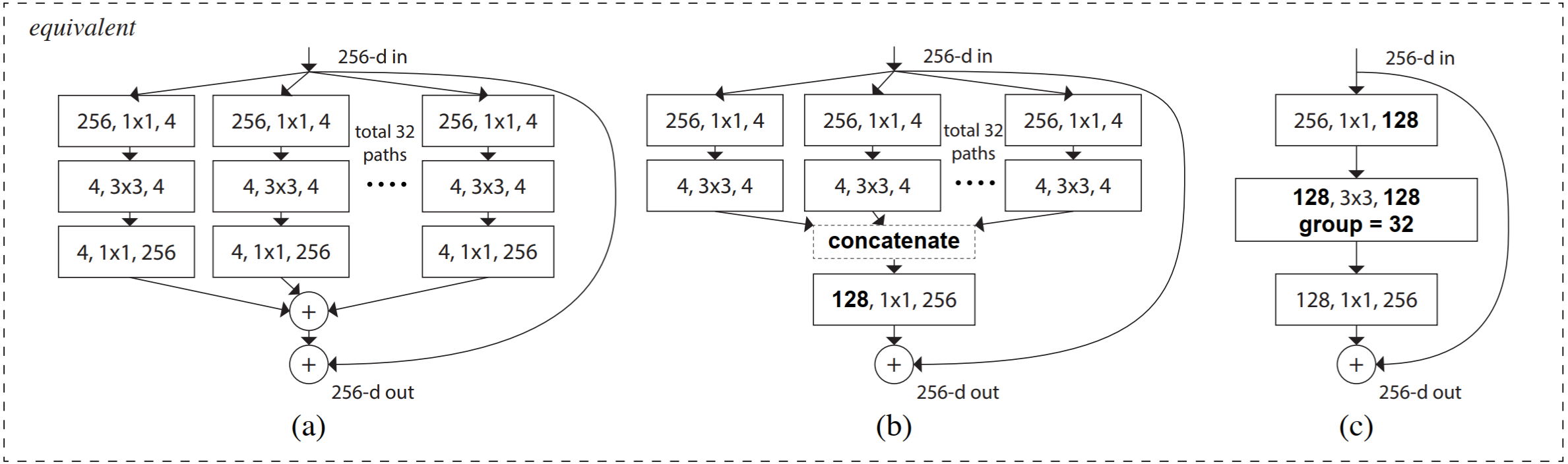


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

[Xie et al., 2016] https://arxiv.org/pdf/1611.05431.pdf

Three Equivalent Forms of ResNeXt



[Xie et al., 2016] https://arxiv.org/pdf/1611.05431.pdf

Dense Convolutional Networks (DenseNet)

Can be thought as a logical extension of ResNet.

Recall Taylor expansion:

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \dots.$$

ResNet decompose f into a simple linear term (x) and a more complex nonlinear one g(x), i.e., ResNet Block.

$$f(\mathbf{x}) = \mathbf{x} + g(\mathbf{x})$$

We could add more terms to the ResNet above, but that won't be well justified as g(x) can be complex and highly expressive as needed.

DenseNet

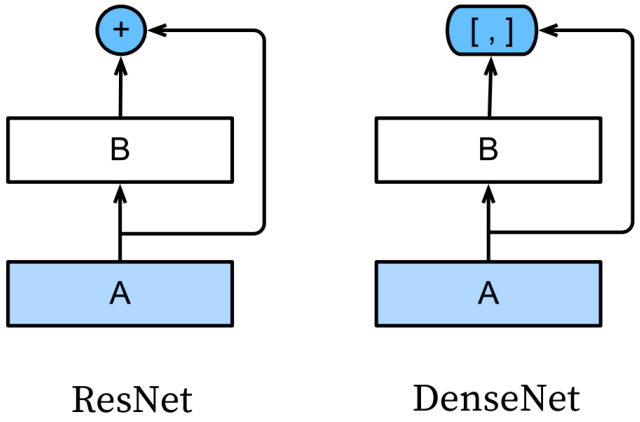
DenseNet provides an alternative solution:

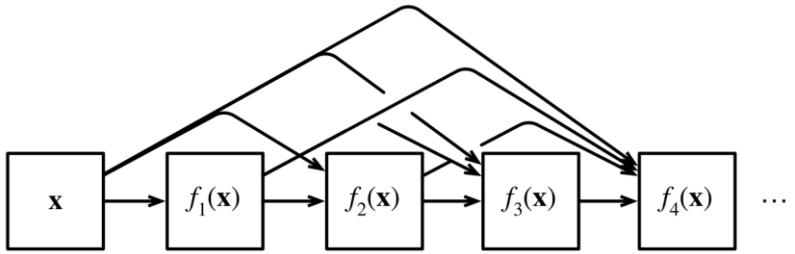
In stead of adding more terms, cumulatively concatenate the outputs from previous layers/blocks.

$$\mathbf{x} \to [\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})]), f_3([\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})])]), \ldots]$$

Use **dense blocks** for modularity (analogous to ResNet Blocks).

Use **transition layers** to control the complexity of the model.





Dense Blocks and Transition Layers

A dense block consists of multiple convolution blocks, each using the same number of output channels.

In the forward propagation, the input and output of each convolution block are concatenated along the channel dimension.

A transition layer reduces the number of channels by using the 1×1 convolutional layer and halves the height and width of the average pooling layer with a stride of 2.

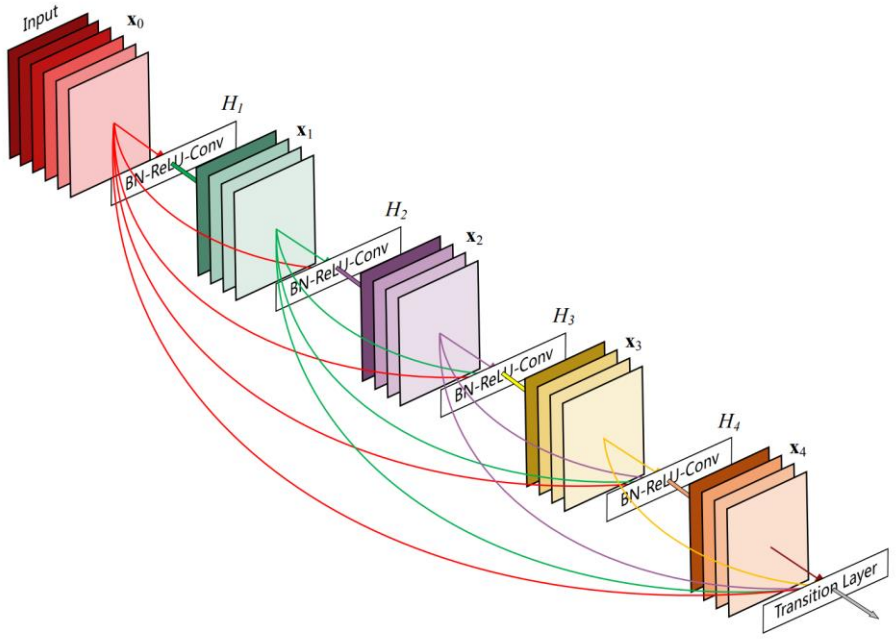


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

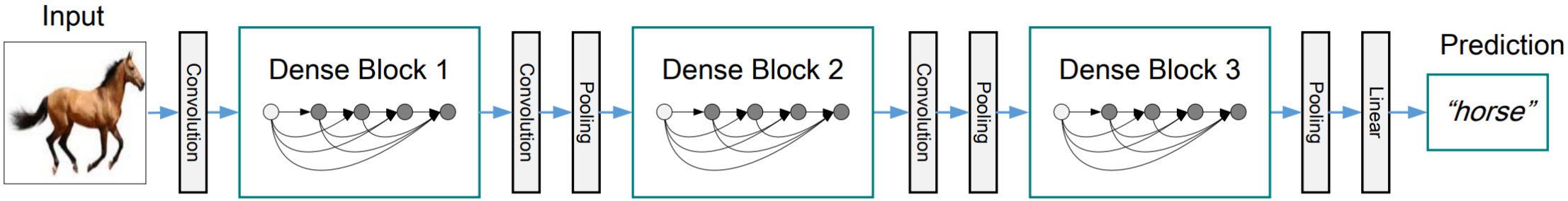


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

[Huang et al., 2017] https://arxiv.org/pdf/1608.06993.pdf

Neural Architecture Search (NAS)

Neural Architecture Search with Reinforcement Learning [Zoph and Le 2017]
https://arxiv.org/pdf/1611.01578.pdf

Learning Transferable Architectures for Scalable Image Recognition [Zoph et al. 2017]
https://arxiv.org/abs/1707.07012

Deep Neural Architecture Search with Deep Graph Bayesian Optimization [Ma et al. 2019]
https://arxiv.org/abs/1905.06159

Graph Hypernetworks for Neural Architecture Search [Zhang et al. 2018]
https://arxiv.org/abs/1810.05749

MnasNet: Platform-Aware Neural Architecture Search for Mobile [Tan et al. 2019]
https://arxiv.org/pdf/1807.11626.pdf

MnasNet: Platform-Aware Neural Architecture Search for Mobile

$$\underset{m}{\text{maximize}} \quad ACC(m) \times \left[\frac{LAT(m)}{T}\right]^w$$

$$w = \begin{cases} \alpha, & \text{if } LAT(m) \le T \\ \beta, & \text{otherwise} \end{cases}$$

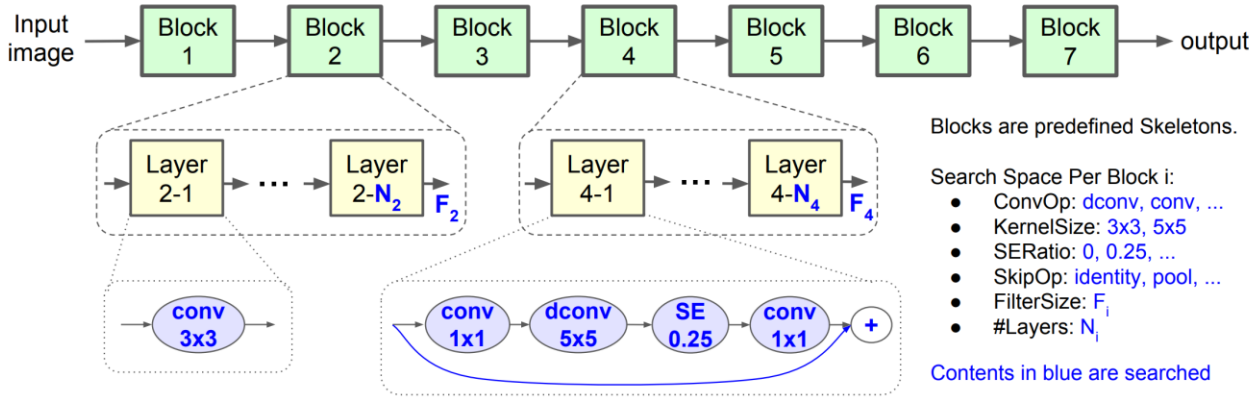


Figure 4: **Factorized Hierarchical Search Space.** Network layers are grouped into a number of predefined skeletons, called blocks, based on their input resolutions and filter sizes. Each block contains a variable number of repeated identical layers where only the first layer has stride 2 if input/output resolutions are different but all other layers have stride 1. For each block, we search for the operations and connections for a single layer and the number of layers N, then the same layer is repeated N times (e.g., Layer 4-1 to 4-N_4 are the same). Layers from different blocks (e.g., Layer 2-1 and 4-1) can be different.

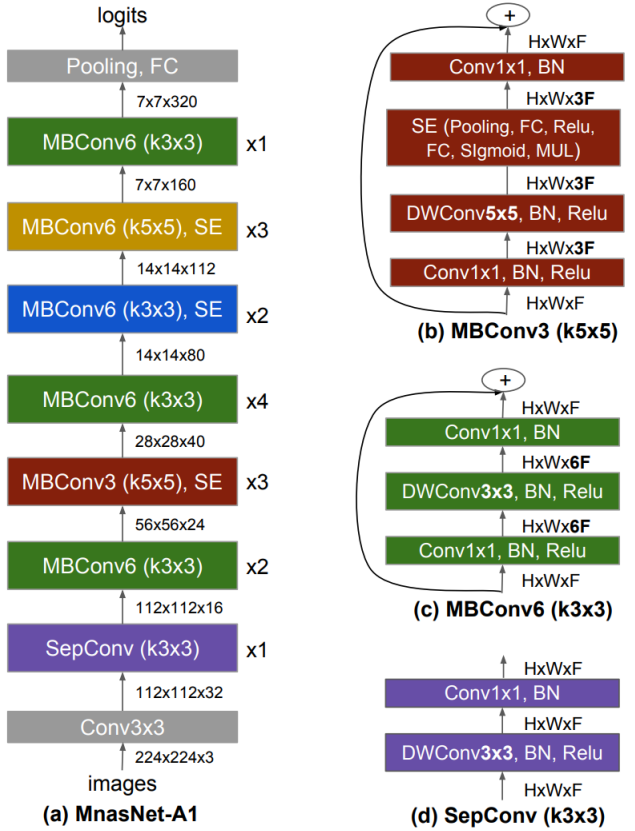


Figure 7: **MnasNet-A1 Architecture** – (a) is a representative model selected from Table 1; (b) - (d) are a few corresponding layer structures. *MBConv* denotes mobile inverted bottleneck conv, *DWConv* denotes depthwise conv, k3x3/k5x5 denotes kernel size, *BN* is batch norm, HxWxF denotes tensor shape (height, width, depth), and $\times 1/2/3/4$ denotes the number of repeated layers within the block.

[Tan et al. 2019] https://arxiv.org/abs/1807.11626

EfficientNet: Smart Compound Scaling

$$\max_{d,w,r} \quad Accuracy(\mathcal{N}(d,w,r))$$

$$s.t. \quad \mathcal{N}(d,w,r) = \bigodot_{i=1\ldots s} \hat{\mathcal{F}}_i^{d\cdot\hat{L}_i}\left(X_{\langle r\cdot\hat{H}_i, r\cdot\hat{W}_i, w\cdot\hat{C}_i\rangle}\right)$$

$$\text{Memory}(\mathcal{N}) \leq \text{target\_memory}$$

$$\text{FLOPS}(\mathcal{N}) \leq \text{target\_flops}$$

Sometimes, smart heuristic is better than NAS.

ConvNet N can be represented by a list of composed layers:

$$\mathcal{N} = \mathcal{F}_k \odot \ldots \odot \mathcal{F}_2 \odot \mathcal{F}_1(X_1) = \bigodot_{j=1\ldots k} \mathcal{F}_j(X_1)$$

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

$\mathcal{F}_i^{L_i}$ denotes layer F_i is repeated L_i times in stage i

$\langle H_i, W_i, C_i\rangle$ denotes the shape of input tensor X of layer i

Scale up using smart heuristic rules.

$$\text{depth: } d = \alpha^\phi$$

$$\text{width: } w = \beta^\phi$$

$$\text{resolution: } r = \gamma^\phi$$

$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

w, d, r are coefficients for scaling network width, depth, and resolution

$\hat{\mathcal{F}}_i, \hat{L}_i, \hat{H}_i, \hat{W}_i, \hat{C}_i$ are predefined parameters in baseline network

Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i\rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

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

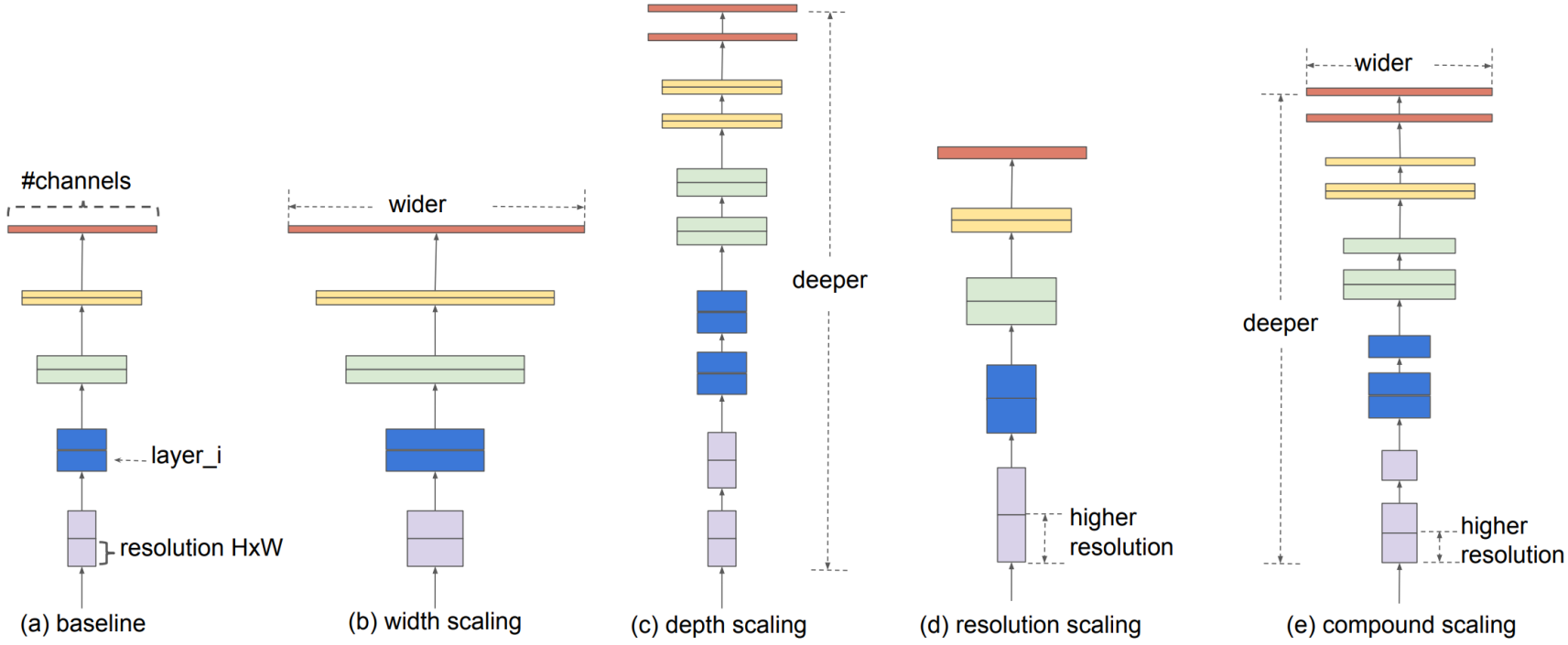
[Tan and Le. 2019] https://arxiv.org/pdf/1905.11946.pdf

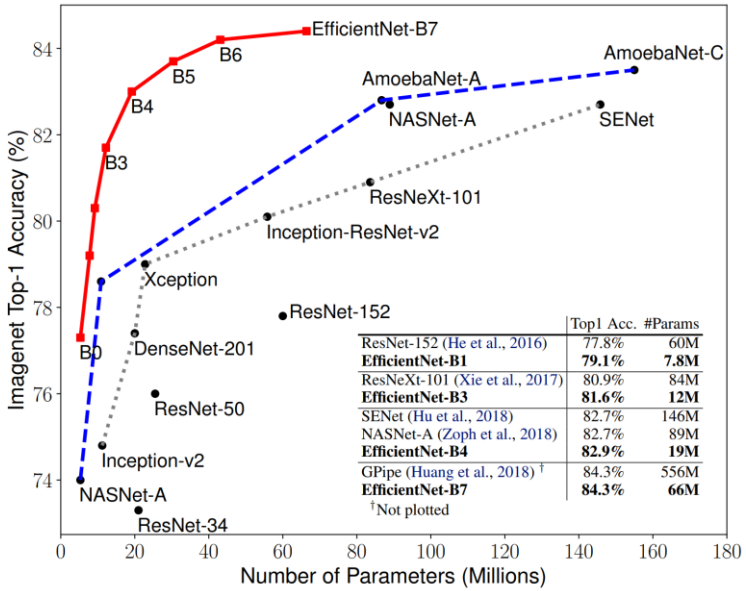
EfficientNet: Smart Compound Scaling

Increase network capacity by scaling width, depth, and resolution, while balancing accuracy and efficiency.

Search for optimal set of compound scaling factors subject to a compute budget.

Scale up using smart heuristic rules.



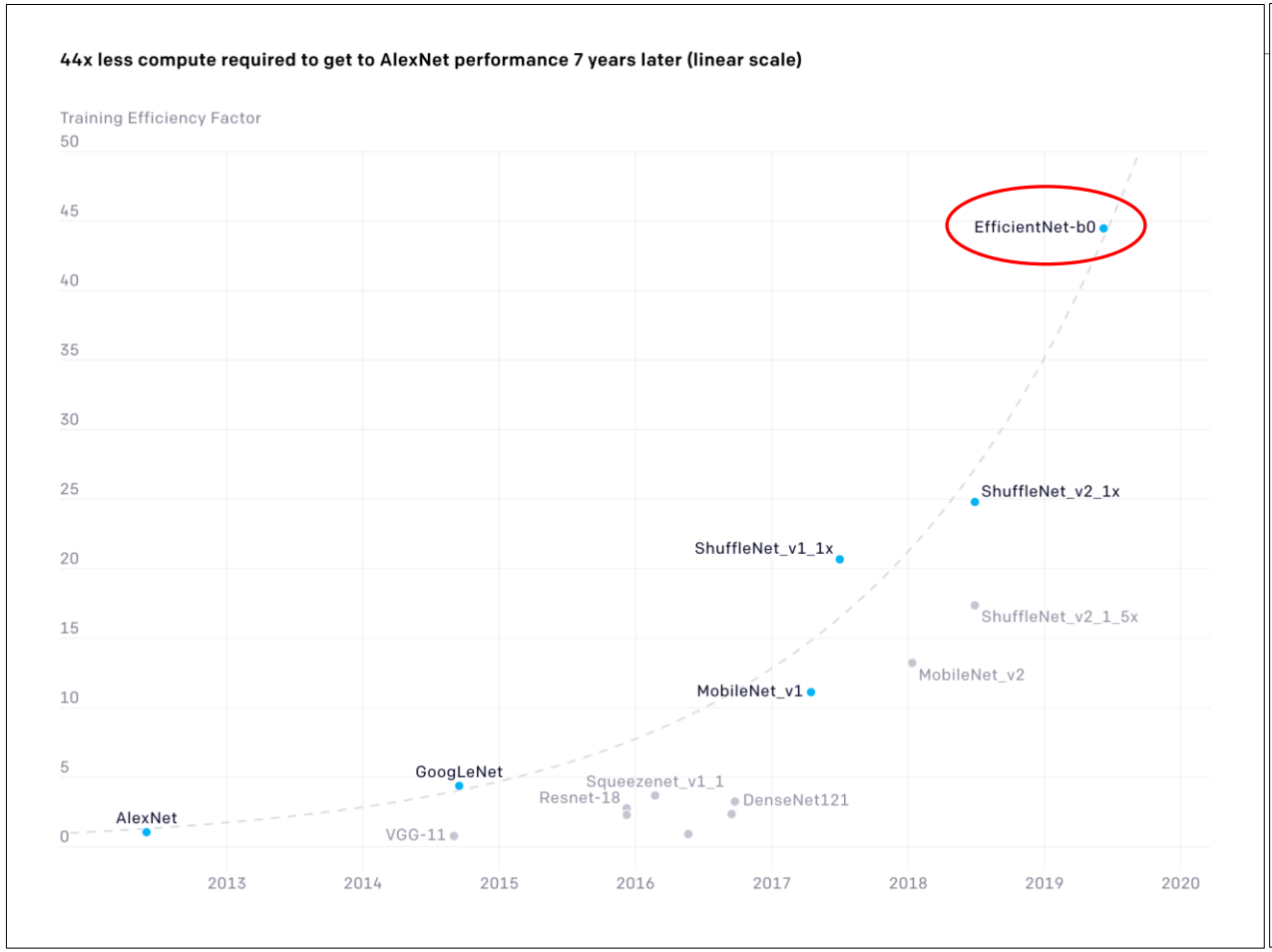


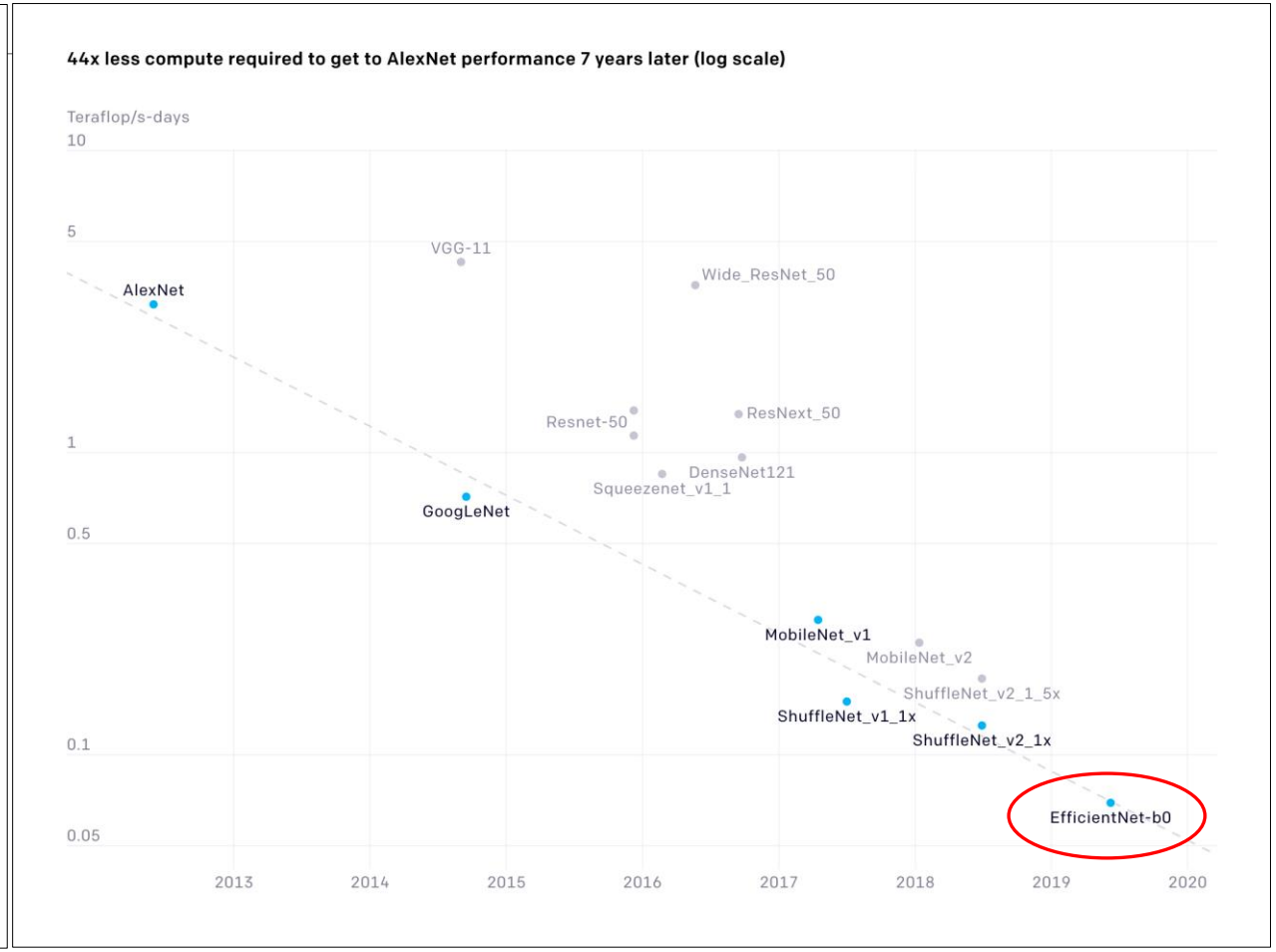
	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

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.

Efficiency





https://openai.com/blog/ai-and-efficiency/