Image Classification

Junkun Yuan
Zhejiang University
yuanjk@zju.edu.cn

1. Introduction

Image classification aims to classify the given image into the corresponding classes.

We summarize the famous models with key phrases (the timeline of these models is shown in Fig. 1):

- **LeNet:** back-propagation, convolution. Sec. 2.
- **AlexNet:** GPU, ReLU, local response normalization, overlapping max-pooling, dropout. Sec. 3.
- **VGG:** deep, 3×3 convolution filters. Sec. 4.
- **Inception:** group conv with different filters. Sec. 5.
  - **GoogleNet (InceptionV1):** inception module.
  - **InceptionV2:** replances 5×5 convolution with two 3×3 convolutions, factorize them into $1 \times n$ and $n \times 1$ convolutions.
  - **InceptionV3:** improved version of InceptionV2.
  - **InceptionV4:** improved version of Inception V3.
  - **Inception-ResNet:** introduce skip connection.
- **ResNet:** add short connection in each block. Sec. 6.
- **ResNeXt:** group conv with the same topology. Sec. 7.
- **DenseNet:** maximum information flow, connect all layers directly with each other. Sec. 8.
- **MobileNet:** high accuracy, low latency. Sec. 9
  - **MobileNet:** depth separable convolution, width and resolution multipliers, small, low latency.
  - **MobileNetV2:** expansion layer, inverted residuals, and linear bottleneck.
  - **MobileNetV3:** NAS, h-swish nonlinearities, redesign last stage, segmentation decoder.
- **ShuffleNet:** extremely computation-efficient. Sec. 10.
  - **ShuffleNet:** pointwise group convolution, shuffle channels for cross talk between groups.
  - **ShuffleNetV2:** equal channel width of input and output, fewer group convolutions, fewer network fragmentation, fewer element-wise operations.
- **SENet:** learn relationship between channels. Sec. 11.
- **EfficientNet:** efficient scaling methods, NAS. Sec. 12.
  - **EfficientNet:** balance deep, width, resolution.
  - **EfficientNetV2:** training-aware NAS, MBConv structure, progressive learning.
- **RegNet:** combine manual design and NAS. Sec. 13.

2. LeNet

LeNet [8] applied back-propagation to a real-world problem in recognizing hand-written digits. The network is directly fed with images, rather than feature vectors, demonstrating the ability of back-propagation networks to deal with low-level information. (From Sec. 1 of [8])

The "weight sharing" technique [11] makes all units in a plane (feature map) share the same set of weights, thereby detecting the same feature at different locations. It can be interpreted as imposing equality constraints among the connection strengths. It not only greatly reduces free parameters in network but also can express information about the geometry and topology of the task. (From Sec. 3.2 of [8])

See LeNet architecture in Fig. 2. The input of the network is a 16 by 16 normalized image. The output is composed of 10 units (one per class) and uses place coding. H1 comprises 768 units (8 by 8 times 12 groups), 19,968 connections (768 times 26 (25 parameters plus 1 bias)), but only 1068 free parameters (768 biases plus 25 times 12 feature kernels) since many connections share the same weight. LeNet has 1256 units, 64,660 connections, and 9760 independent parameters. (From Sec. 3.1 and Sec. 3.3 of [8])

The nonlinear function used at each node was tangent, and was sigmoid for the output. The cost function was MSE. The weights were initialized with uniform distribution. The weights were updated with SGD and a special version of Newton's algorithm. (From Sec. 4 of [8])

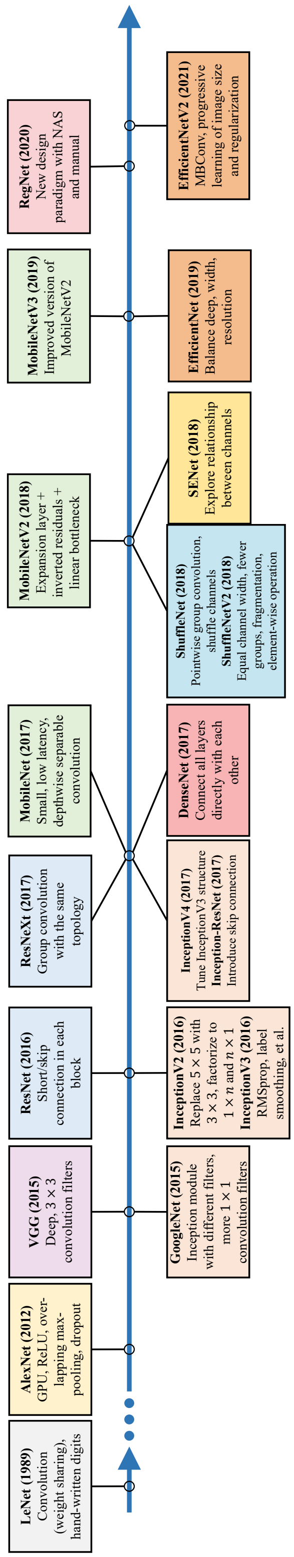


Figure 1. **Timeline**. Use the time of publication.

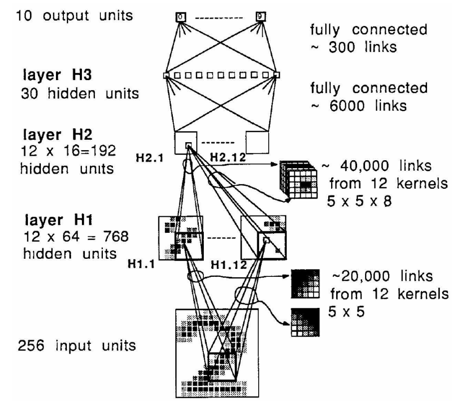


Figure 2. **LeNet** network architecture. (From Fig. 3 of [8])

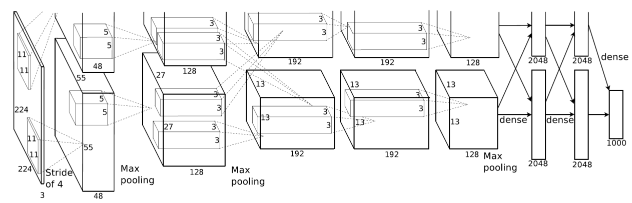


Figure 3. **AlexNet** network architecture. (From Fig. 2 of [7])

3. AlexNet

AlexNet [7] (Fig. 3) gives GPU implementation of CNNs. Instead of using tanh activation function, AlexNet adopts ReLU for faster training. It places local response normalization after ReLU in certain layers, and uses overlapping max-pooling. To reduce overfitting, it uses two augmentation strategies: extracting random 224×224 patches from original images and altering RGB channels, and uses dropout in FC layers. (From Sec. 3 and Sec. 4 of [7])

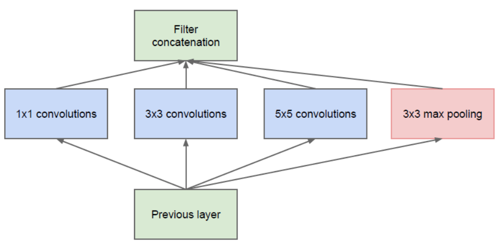
It uses SGD with momentum of 0.9 and weight decay 0.0005. The weights are initialized with Gaussian distribution $\mathcal{N}(0, 0.01)$. The biases are initialized with constant 1 (the 2-nd, 4-th, 5-th layer) or 0 (elsewhere). The learning rate is divided by 10 when the validation error rate stopped improving. The learning rate is initialized at 0.01 and reduced 3 times. AlexNet is trained 90 cycles on NVIDIA GTX 580 3GB GPUs over 5 to 6 days. (From Sec. 5 of [7])

4. VGG

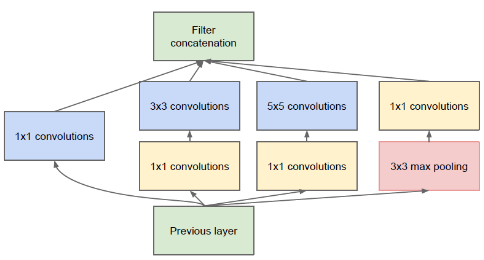
VGG [13] (Fig. 4) raises the importance of depth in ConvNet. It borrows ReLU but abandons local response normalization from AlexNet [7]. The width starts from 64 and increases by a factor of 2 after each max-pooling layer, to 512. Max-pooling is performed with 2×2 pixel window and stride 2. It uses 3×3 convolution filters in all layers. Because (1) a stack of three 3×3 layers with three ReLU instead of a single 7×7 layer with a single ReLU (they have

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 4. **VGG** network architecture. (From Table 1 of [13])



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

Figure 5. **Inception module**. (From Fig. 2 of [15])

the same receptive filed) to make the decision function more discriminative, and (2) few parameters $3(3^2C^2) = 27C^2$ vs. $7^2C^2 = 49C^2$. (From Sec. 2 and Sec. 3 of [13])

5. Inception

GoogleNet [15] (or **InceptionV1**) (Fig. 6) exploits the inception module (Fig. 5) with parallel convolutions containing different filters to make the network wider.

InceptionV2 [16] (Fig. 8) exploits the three kinds of

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figure 6. **GoogleNet** structure. (From Table 1 of [15])

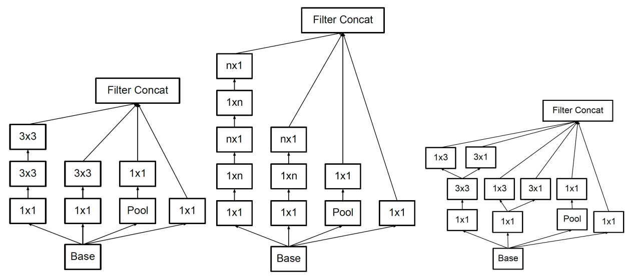


Figure 7. **InceptionV2** modules. (From Fig.5, 6, 7 of [16])

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Figure 8. **InceptionV2** network structure. (From Table 1 of [16])

blocks in Fig. 7, which replaces 5×5 convolution with two 3×3 convolutions, and factorizes them into $1 \times n$ and $n \times 1$.

InceptionV3 [16] is based on InceptionV2, but further uses RMSProp optimizer, factorized 7×7 convolution, BN in the auxillary classifiers, and label smoothing.

InceptionV4 [14] (Fig. 9) tunes InceptionV3 structure.

Inception-ResNet [14] (Fig. 10) introduces the skip connection into the Inception structure.

6. ResNet

It is hard to train a deeper networks because of the notorious problem of vanishing/exploding gradients. While normalized initialization and intermediate normalization layers have largely addressed this problem, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated and then degrades rapidly. It is not caused by overfitting, and adding more layers to a deep

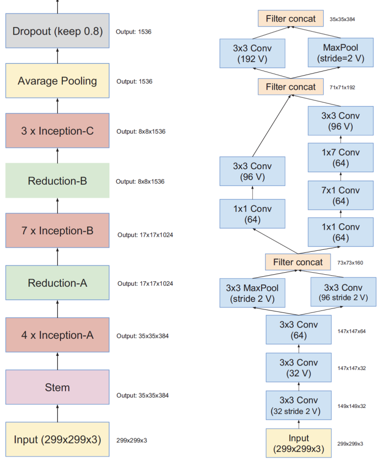


Figure 9. **InceptionV4** network structure (left) and its detailed composition (right). (From Fig. 2 of [14])

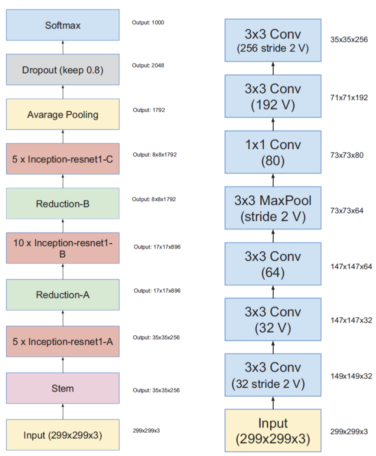


Figure 10. **Inception-ResNET** network structure (left) and its detailed composition (right). (From Fig. 6 of [14])

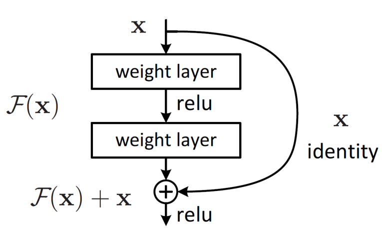


Figure 11. **Residual learning**: building block. (From Fig. 2 of [2])

model leads to higher training error. (From Sec. 1 of [2])

The **idea** is clear: if the added layers can be constructed as identity mappings, a deeper model should have training error no greater than its shallow counterpart. The degradation problem suggests that the solvers might have difficulties in approximating identity mappings by multiple nonlinear layers. With the residual learning reformulation, if identity mappings are optimal, the solvers may simply drive the weights of the multiple nonlinear layers toward zero to approach identity mappings. (From Sec. 3.1 of [2])

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

Figure 12. **ResNet** architectures. (From Table 1 of [2])

stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 128; 3×3, 128, C=32; 1×1, 256] ×3
conv3	28×28	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 256; 3×3, 256, C=32; 1×1, 512] ×4
conv4	14×14	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 512; 3×3, 512, C=32; 1×1, 1024] ×6
conv5	7×7	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 1024; 3×3, 1024, C=32; 1×1, 2048] ×3
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		$\mathbf{25.5}\times10^6$	$\mathbf{25.0}\times10^6$
FLOPs		$\mathbf{4.1}\times10^9$	$\mathbf{4.2}\times10^9$

Figure 13. **ResNeXt** network architecture. (From Table 1 of [20])

ResNet [2] (Fig. 12) uses deep residual learning framework with "short connections" (Fig. 11). That is:

$$y = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}. \tag{1}$$

The operation $\mathcal{F} + \mathbf{x}$ is performed by a shortcut connection and element-wise addition. It introduces neither extra parameter nor computation complexity. (From Sec. 3.1 of [2])

7. ResNeXt

Different from Inception, **ResNeXt** [20] (Fig. 13) makes all the paths share the same topology to aggregate a set of transformations and make the network structure simple and highly modularized. The number of paths is a structure parameter "cardinality"/"groups". (From Sec. 1 of [20])

8. DenseNet

To ensure maximum information flow between layers, **DenseNet** [6] (Fig. 14) connect all layer directly with each

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Figure 14. **DenseNet** architecture. (From Table 1 of [6])

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	$1 \times 1 \times 1000$

Figure 15. **MobileNet** architectures. (From Table 1 of [4])

other. The difference between DenseNet and ResNet:

$$\begin{aligned} \text{ResNet: } \mathrm{x}_\ell &= H_\ell(\mathrm{x}_{\ell-1}) + \mathrm{x}_{\ell-1} \\ \text{DenseNet: } \mathrm{x}_\ell &= H_\ell([\mathrm{x}_0, \mathrm{x}_1, ..., \mathrm{x}_{\ell-1}]), \end{aligned} \tag{2}$$

where $[\cdot]$ is concatenation. (From Sec. 1 and Sec. 3 of [6])

9. MobileNet

In order to build very small, low latency models to carry out recognition task in a timely fashion on a computationally limited platform, **MobileNet** [4] (Fig. 15) uses depth separable convolution with a depthwise convolution for filtering and a 1×1 pointwise convolution for combining. Let a feature map with height/width D_F be fed into a convolution layer with kernel size D_K, input channel M, output channel N. We have convolution computation cost:

$$\frac{\text{separable}}{\text{standard}} = \frac{D_K^2 M D_F^2 + M N D_F^2}{D_K^2 M N D_F^2} = \frac{1}{N} + \frac{1}{D_K^2} \tag{3}$$

It adopts a width multiplier $\alpha \in (0, 1]$ and resolution multiplier $\rho \in (0, 1]$ to further reduce computation cost by α^2 and ρ^2, respectively. (From Sec. 1 and Sec. 3 of [4])

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 16. **MobileNetV2** architectures. (From Table 2 of [12])

Input	Operator	exp size	$\#out$	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

Input	Operator	exp size	$\#out$	SE	NL	s
$224^2 \times 3$	conv2d, 3x3	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	✓	RE	2
$56^2 \times 16$	bneck, 3x3	72	24	-	RE	2
$28^2 \times 24$	bneck, 3x3	88	24	-	RE	1
$28^2 \times 24$	bneck, 5x5	96	40	✓	HS	2
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	120	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	144	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	288	96	✓	HS	2
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	conv2d, 1x1	-	576	✓	HS	1
$7^2 \times 576$	pool, 7x7	-	-	-	-	1
$1^2 \times 576$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

Figure 17. **MobileNetV3** architectures. Left: MobileNetV3-Large. Right: MobileV3-Small. (From Table 1 and Table 2 of [3])

However, the ReLU collapses some channels and losses information when reducing channels. Instead of directly using depthwise separable convolution in the MobileNet:

$$\xrightarrow{D_F D_F M} 3 \times 3, \sigma \xrightarrow{D_F D_F M} 1 \times 1, \sigma \xrightarrow{D_F D_F N}, \tag{4}$$

MobileNetV2 [12] (Fig. 16) uses bottleneck residual block with expansion layer, inverted residuals, linear bottleneck:

$$\underbrace{\xrightarrow{D_F D_F M} 1 \times 1, \sigma \xrightarrow{D_F D_F tM} 3 \times 3, \sigma \xrightarrow{D_F D_F tM} 1 \times 1 \xrightarrow{D_F D_F N}}_{\text{short cut connection}}, \tag{5}$$

where σ is $ReLU6$, t is the expansion factor. Compared with Eq. (3), its computation cost is (from Sec. 3 of [12]):

$$\underbrace{{D_F}^2 t M^2}_{\text{expansion layer}} + \underbrace{{D_F}^2 M t 3^2}_{\text{convolution}} + \underbrace{{D_F}^2 M t N}_{\text{linear bottleneck}}. \tag{6}$$

MobileNetV3 [3] (Fig. 17) introduces (1) complementary search techniques with platform-aware NAS (blockwise search) and NetAdapt (layerwise search), (2) new efficient versions of nonlinearities (h-swish), (3) new efficient network design (redesign last stage), (4) a new efficient segmentation decoder. (From Sec. 1, Sec. 4, and Sec. 5 of [3])

10. ShuffleNet

ShuffleNet [21] (Fig. 18) proposes pointwise group convolution to reduce computation cost, and solves the problem

Layer	Output size	KSize	Stride	Repeat	Output channels (g groups)				
					$g=1$	$g=2$	$g=3$	$g=4$	$g=8$
Image	224×224				3	3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24	24
MaxPool	56×56	3×3	2						
Stage2	28×28		2	1	144	200	240	272	384
	28×28		1	3	144	200	240	272	384
Stage3	14×14		2	1	288	400	480	544	768
	14×14		1	7	288	400	480	544	768
Stage4	7×7		2	1	576	800	960	1088	1536
	7×7		1	3	576	800	960	1088	1536
GlobalPool	1×1	7×7							
FC					1000	1000	1000	1000	1000
Complexity					143M	140M	137M	133M	137M

Figure 18. **ShuffleNet** architecture. (From Table 1 of [21])

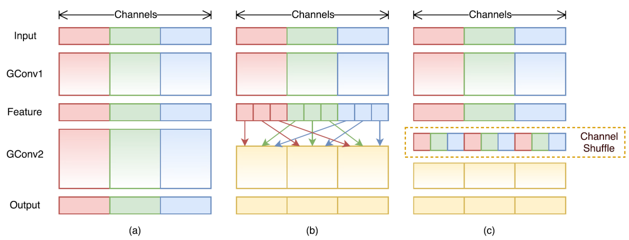


Figure 19. **Motivation of ShuffleNet**. a) two stacked convolution with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; b) input and output channels are fully related when GConv2 takes data from different groups after GConv1; c) an equivalent implementation to b) using channel shuffle. (From Fig. 1 of [21])

Layer	Output size	KSize	Stride	Repeat	Output channels			
					0.5×	1×	1.5×	2×
Image	224×224				3	3	3	3
Conv1 MaxPool	112×112 56×56	3×3 3×3	2 2	1	24	24	24	24
Stage2	28×28 28×28		2 1	1 3	48	116	176	244
Stage3	14×14 14×14		2 1	1 7	96	232	352	488
Stage4	7×7 7×7		2 1	1 3	192	464	704	976
Conv5	7×7	1×1	1	1	1024	1024	1024	2048
GlobalPool	1×1	7×7						
FC					1000	1000	1000	1000
FLOPs					41M	146M	299M	591M
# of Weights					1.4M	2.3M	3.5M	7.4M

Figure 20. **ShuffleNetV2** architecture. (From Table 5 of [9])

of cross talk between groups in group convolutions by shuffling channels as shown in Fig. 19. (From Sec. 3 of [21])

ShuffleNetV2 [9] (Fig. 20) is designed by considering direct metric of speed instead of indirect metric of FLOPs. Because the models with the same FLOPs but different memory access cost (MAC), degree of parallelism, and platform could have different speed. It gives practical guidelines for efficient network design: (1) The 1×1 convolution with input channel c_1, output channel c_2, height h, and width w have $\text{FLOPs}(B) = hwc_1c_2$ and $\text{MAC} = hw(c_1+c_2)+c_1c_2 \le 2\sqrt{hwB}+\frac{B}{hw}$. Thus, equal channel width minimizes MAC. (2) The 1×1 group convolutions with groups g have $\text{FLOPs}(B) = hwc_1c_2/g$ and $\text{MAC} = hw(c_1+c_2)+c_1c_2/g = hwc_1+\frac{Bg}{c_1}+\frac{B}{hw}$. Given the fixed input shape $c_1\times h\times w$ and the computation cost B, excessive group convolution increases MAC. (3) Network fragmentation reduces degree of parallelism. (4) Element-wise operations are non-negligible.

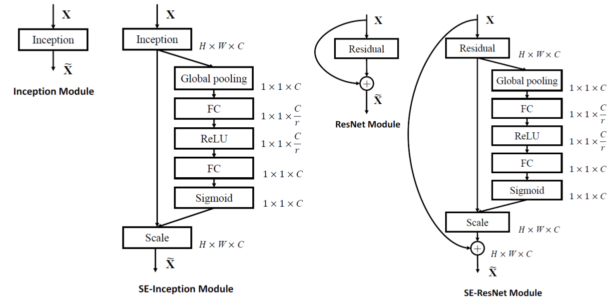


Figure 21. **SE-Inception** and **SE-ResNet** modules. (From Fig. 2 and Fig. 3 of [5])

Output size	ResNet-50	SE-ResNet-50	SE-ResNeXt-50 (32 × 4d)
112 × 112	conv, 7 × 7, 64, stride 2		
56 × 56	max pool, 3 × 3, stride 2		
	$\begin{bmatrix}\text{conv}, 1\times1, 64\\ \text{conv}, 3\times3, 64\\ \text{conv}, 1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}\text{conv}, 1\times1, 64\\ \text{conv}, 3\times3, 64\\ \text{conv}, 1\times1, 256\\ fc, [16, 256]\end{bmatrix}\times3$	$\begin{bmatrix}\text{conv}, 1\times1, 128 & \\ \text{conv}, 3\times3, 128 & C=32\\ \text{conv}, 1\times1, 256 & \\ fc, [16, 256] & \end{bmatrix}\times3$
28 × 28	$\begin{bmatrix}\text{conv}, 1\times1, 128\\ \text{conv}, 3\times3, 128\\ \text{conv}, 1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}\text{conv}, 1\times1, 128\\ \text{conv}, 3\times3, 128\\ \text{conv}, 1\times1, 512\\ fc, [32, 512]\end{bmatrix}\times4$	$\begin{bmatrix}\text{conv}, 1\times1, 256 & \\ \text{conv}, 3\times3, 256 & C=32\\ \text{conv}, 1\times1, 512 & \\ fc, [32, 512] & \end{bmatrix}\times4$
14 × 14	$\begin{bmatrix}\text{conv}, 1\times1, 256\\ \text{conv}, 3\times3, 256\\ \text{conv}, 1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}\text{conv}, 1\times1, 256\\ \text{conv}, 3\times3, 256\\ \text{conv}, 1\times1, 1024\\ fc, [64, 1024]\end{bmatrix}\times6$	$\begin{bmatrix}\text{conv}, 1\times1, 512 & \\ \text{conv}, 3\times3, 512 & C=32\\ \text{conv}, 1\times1, 1024 & \\ fc, [64, 1024] & \end{bmatrix}\times6$
7×7	$\begin{bmatrix}\text{conv}, 1\times1, 512\\ \text{conv}, 3\times3, 512\\ \text{conv}, 1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}\text{conv}, 1\times1, 512\\ \text{conv}, 3\times3, 512\\ \text{conv}, 1\times1, 2048\\ fc, [128, 2048]\end{bmatrix}\times3$	$\begin{bmatrix}\text{conv}, 1\times1, 1024 & \\ \text{conv}, 3\times3, 1024 & C=32\\ \text{conv}, 1\times1, 2048 & \\ fc, [128, 2048] & \end{bmatrix}\times3$
1 × 1	global average pool, 1000-d fc, softmax		

Figure 22. **SENet** structure. (From Table 1 of [5])

11. SENet

SENet [5] (Fig. 22) is proposed to explore the relationship between channels. It uses a squeeze operation to average the spatial dimensions and get a channel descriptor. It then uses a excitation operation with two FC layers to learn per-channel modulation weights that applied to the feature maps. See SE modules in Fig. 21. (From Sec. 1 of [5])

12. EfficientNet

The most common way to scale up ConvNets is by their depth, width, and image size (resolution). But, arbitrary scaling requires tedious manual tuning and still often yields sub-optimal accuracy and efficiency. (From Sec. 1 of [18])

The authors found: (1) scaling up any dimension of network width, depth, resolution improves accuracy, but accuracy gain diminishes for bigger models, and (2) it is crucial to balance them. Thus, **EfficientNet** used a compound scal-

Stage	Operator	Stride	#Channels	#Layers
0	Conv3x3	2	24	1
1	Fused-MBConv1, k3x3	1	24	2
2	Fused-MBConv4, k3x3	2	48	4
3	Fused-MBConv4, k3x3	2	64	4
4	MBConv4, k3x3, SE0.25	2	128	6
5	MBConv6, k3x3, SE0.25	1	160	9
6	MBConv6, k3x3, SE0.25	2	256	15
7	Conv1x1 & Pooling & FC	-	1280	1

Figure 23. **EfficientNetV2** architecture. (From Table 4 of [19])

ing method with a coefficient ϕ to uniformly scales network:

$$\text{depth: } d = \alpha^\phi, \quad \text{width: } w = \beta^\phi, \quad \text{resolution: } r = \gamma^\phi,$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2, \quad \alpha >= 1, \quad \beta >= 1, \quad \gamma >= 1, \tag{7}$$

ϕ is a user-specific coefficient that controls how many more resources are available for model scaling, while α, β, γ specify how to assign these extra resources to network width, depth, and resolution respectively. For any new ϕ, the total FLOPS will approximately increase by 2^ϕ. The baseline network EfficientNet-B0 is developed by leveraging a multi-objective NAS [17] that optimizes both accuracy and FLOPS. Starting from the baseline network, the authors applied the compound scaling method to scale it up with two steps: (1) fix $\phi = 1$, assuming twice more resources available, and do a small grid search of α, β, γ via Eq. (7), and (2) fix α, β, and γ as constants and scale up baseline network with different ϕ using Eq. (7), to obtain EfficientNet-B1 to B7. (From Sec. 3 and Sec. 4 of [18])

EfficientNetV2 [19] (Fig. 23) is optimized with training-aware NAS and model scaling with MBConv structure [1], and is further sped up with progressive learning by jointly increasing image size and regularization (Dropout, RandAugment, Mixup) during training. (From Sec. 7 of [19])

13. RegNet

Manual network design like LeNet [8], AlexNet [7], VGG [13], and ResNet [2] that demonstrate the importance of convolution, network and data size, depth, and residuals in network design principles, respectively. Meanwhile, despite the effectiveness of NAS, its search outcome is a single network instance in a specific setting. (From Sec. 1 of [10])

RegNet [10] is proposed under a new network design paradigm that combines the advantages of manual design and NAS. It progressively design simplified versions of an initial, relatively unconstrained, design space while maintaining or improving its quality. (From Sec. 1 of [10])

Specifically, assume there are four stages of the main body of the designed network, and each stage i consists of a sequence of identical blocks with four parameters we need to design: the number of blocks d_i, block width w_i, bottleneck ratio b_i, and group width g_i. The initial network design space is called AnyNetA. Then, the authors found that let $b_i = b$ and $g_i = g$ would not change the results and named them AnyNetB and AnyNetC, respectively. Next, the authors that found design spaces with $w_{i+1} >= w_i$ and $d_{i+1} >= d_i$ are good and name them AnyNetD and AnyNetE, respectively. Finally, the authors chose best 20 models from AnyNetE and fit them using piecewise constant functions. Now, we can specify a network structure via 6 parameters of the obtained functions: d, w_0, w_a, w_m, b, and g. Design space and RegNet structures are shown in Fig. 24 and Fig. 25, respectively. (From Sec. 3 of [10])

	restriction	dim.	combinations	total
AnyNetX$_A$	none	16	$(16{\cdot}128{\cdot}3{\cdot}6)^4$	$\sim 1.8{\cdot}10^{18}$
AnyNetX$_B$	+ $b_{i+1} = b_i$	13	$(16{\cdot}128{\cdot}6)^4{\cdot}3$	$\sim 6.8{\cdot}10^{16}$
AnyNetX$_C$	+ $g_{i+1} = g_i$	10	$(16{\cdot}128)^4{\cdot}3{\cdot}6$	$\sim 3.2{\cdot}10^{14}$
AnyNetX$_D$	+ $w_{i+1} \geq w_i$	10	$(16{\cdot}128)^4{\cdot}3{\cdot}6/(4!)$	$\sim 1.3{\cdot}10^{13}$
AnyNetX$_E$	+ $d_{i+1} \geq d_i$	10	$(16{\cdot}128)^4{\cdot}3{\cdot}6/(4!)^2$	$\sim 5.5{\cdot}10^{11}$
RegNet	quantized linear	6	$\sim 64^4{\cdot}6{\cdot}3$	$\sim 3.0{\cdot}10^8$

Figure 24. Design space of **RegNet**. (From Table 1 of [10])

References

[1] Suyog Gupta and Mingxing Tan. Efficientnet-edgetpu: Creating accelerator-optimized neural networks with automl. *Google AI Blog*, 2:1, 2019. 7

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 4, 7

[3] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019. 5

[4] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 5

[5] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 6

[6] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 4, 5

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012. 2, 7

[8] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 1, 2, 7

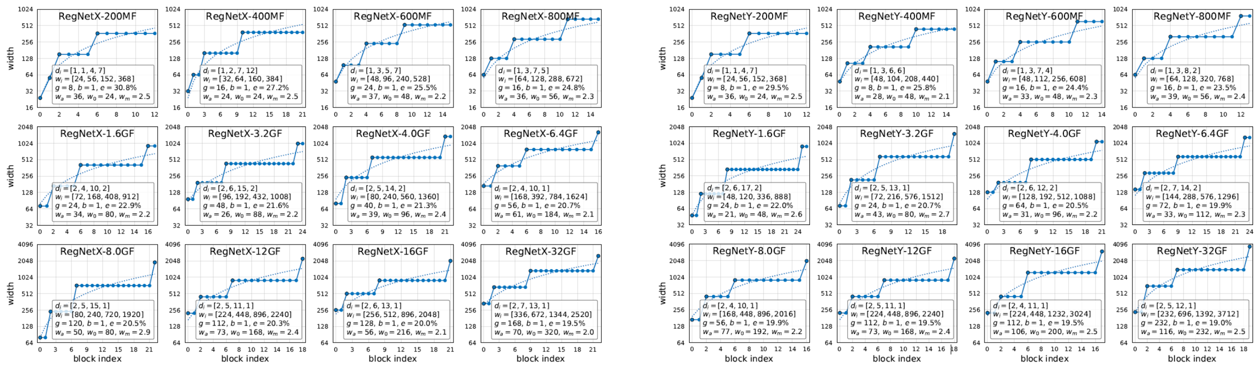


Figure 25. The structures of Top **RegNetX** (left) and **RegNetY** with SE (right) models. (From Figure 11 of [10])

[9] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018. 6

[10] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020. 7, 8

[11] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985. 1

[12] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 5

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2, 3, 7

[14] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017. 3, 4

[15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 3

[16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016. 3

[17] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019. 7

[18] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019. 6, 7

[19] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International Conference on Machine Learning*, pages 10096–10106. PMLR, 2021. 7

[20] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 4

[21] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018. 5, 6