



The Vision Architecture Zoo

Architectures | Reference cards from LeNet to ConvNeXt, ViT, DeiT and Swin, detectors (Faster R-CNN, YOLO, DETR) and segmenters (U-Net, Mask R-CNN, SAM): year, parameters, key idea, what each replaced, receptive fields and U-Net shapes computed

#Vision #CNN #ViT #Detection #Segmentation

Fourteen years of vision backbones

Vision architectures 1998-2026

In one sentence
●○○○

Practitioner
●●○○

Expert
●●●○

Under the hood
●●●●

Networks that turn pixels into labels, boxes or masks; each famous one fixed a weakness of the previous one.

Three eras: plain CNNs get deeper (LeNet → VGG), then smarter blocks make depth cheap (Inception, ResNet, MobileNet), then transformers treat patches as tokens (ViT, Swin) while CNNs modernise (ConvNeXt).

- one backbone, many heads: classify, detect, segment

The trade-off moved from accuracy per parameter to accuracy per FLOP and per millisecond on target hardware.

- inductive bias (CNN) helps small data
- global attention (ViT) wins with large pretraining

In 2026 the reused parts are self-supervised ViT encoders (DINOv2-style, SigLIP-style) feeding task heads and multimodal LLMs, and promptable segmenters (SAM line) used zero-shot.

- CNNs dominate the edge and real-time detection

Analogy Like camera lenses: the sensor (pixels) stays; each generation changes the optics (blocks) to see more of the scene for less glass.

The vision zoo at four depths

Prerequisites and scope

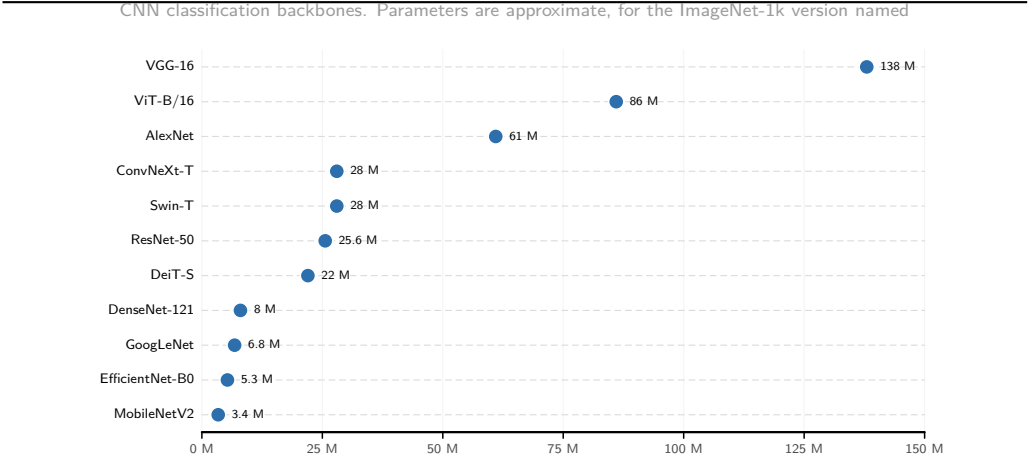
Convolution, pooling, receptive field and the CNN line are taught in deep-learning sheet 06; ViT patch embedding in practice and vision tasks with their metrics (IoU, NMS, mAP) are perception-and-generation 01-02. This sheet is the reference card set and the lineage.

The landmark vision architectures by year of publication (first paper or arXiv release)

Convolutional reference cards			
Model (year)	Params	Key idea	Replaced because
LeNet-5 (1998)	60 k	conv-pool-conv-pool-FC on 32×32 digits	no GPUs or big data then
AlexNet (2012)	61 M	ReLU, dropout, augmentation, 2 GPUs, 8 layers	large 11×11 kernels wasteful
VGG-16 (2014)	138 M	only 3×3 convs, 16 weight layers	huge FC layers, slow
GoogLeNet / Inception v1 (2014)	6.8 M	parallel 1×1/3×3/5×5 branches, 1×1 bottlenecks, 22 layers	hand-designed, hard to scale
ResNet-50 (2015)	25.6 M	residual x + F(x); 1×1-3×3-1×1 bottleneck; 152 layers trainable	still the baseline
DenseNet-121 (2016)	8 M	each layer gets all earlier maps (concatenation)	memory-hungry activations
MobileNetV1/V2 (2017-18)	4.2 M / 3.4 M	depthwise separable; V2 inverted residual, linear bottleneck	superseded by NAS designs

Tadata | tadata.fr | contact@tadata.codes

EfficientNet-B0 (2019)	5.3 M	NAS block (MBConv + squeeze-excitation), compound scaling to B7 (66 M)	slow on GPUs (depthwise)
ConvNeXt-T (2022)	28 M	ResNet modernised: 7×7 depthwise, inverted bottleneck, LayerNorm, GELU, 4×4 patchify stem	used today



Parameters in millions (approx.) of the cards on this page. VGG's 138 M come mostly from its fully connected layers; accuracy since 2015 has grown with far fewer parameters

Layer	Weights	Example
Standard conv	$k^2 \cdot C_{in} \cdot C_{out}$	$3 \times 3, 256 \rightarrow 256: 589,824$
Depthwise separable	$k^2 \cdot C_{in} + C_{in} \cdot C_{out}$	$2,304 + 65,536 = 67,840$ (8.7× fewer)
ResNet bottleneck	$C \cdot \frac{C}{4} + 9 \cdot (\frac{C}{4})^2 + \frac{C}{4} \cdot C$	$C = 256: 69,632$ vs $1,179,648$ for two 3×3 convs

where k = kernel side (pixels); C_{in} , C_{out} = input and output channels (count); C = channels at the block input and output (inner width $C/4$) (count)
 Two numbers that explain MobileNet and ResNet: the cost of a convolution layer

♀ Which CNN in 2026

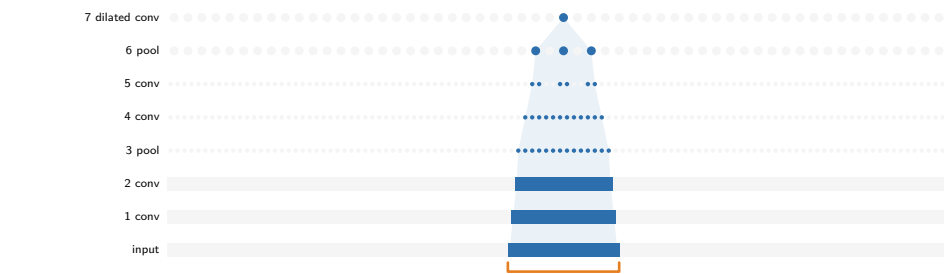
Transfer-learning baseline: **ResNet-50** or **ConvNeXt-T** (well supported, fast on GPUs). Phone or microcontroller: **MobileNetV3** or an EfficientNet-Lite variant. Avoid VGG except as a perceptual-loss feature extractor, where its plain 3×3 features are still the convention.

©
 Receptive fields, computed

l	layer	k	s	d	p	k_eff	out n	jump j	RF r	start
0	input						224	1	1	0
1	conv	3	1	1	1	3	224	1	3	0
2	conv	3	1	1	1	3	224	1	5	0
3	pool	2	2	1	0	2	112	2	6	0.5
4	conv	3	1	1	1	3	112	2	10	0.5
5	conv	3	1	1	1	3	112	2	14	0.5
6	pool	2	2	1	0	2	56	4	16	1.5
7	dilated conv	3	1	2	2	5	56	4	32	1.5

A unit of the last layer sees 32 of 224 pixels (receptive field 32); neighbouring units are 4 pixels apart; output 56 units per side.

Unit 28 of layer 7 traced back (pixels from 0)



receptive field $r = 32$: pixels 98 to 129 around centre 114; 32 pixels actually used

$$k_{\text{eff}} = d \cdot (k - 1) + 1$$

$$n_l = \text{floor}((n_{l-1} + 2p - k_{\text{eff}}) / s) + 1$$

$$j_l = j_{l-1} \cdot s_l, r_l = r_{l-1} + (k_{\text{eff}} - 1) \cdot j_{l-1}$$

$$\text{start}_l = \text{start}_{l-1} + ((k_{\text{eff}} - 1) / 2 - p) \cdot j_{l-1}$$

$$n_0 = \text{input size}, j_0 = r_0 = 1, \text{start}_0 = 0$$

where
k, s, d, p
 kernel, stride, dilation, padding of layer l

k_eff
 span of the dilated kernel in its input (pixels of layer l - 1)

n_l
 units per side after layer l

j_l
 jump: input pixels between neighbouring units of layer l

r_l
 receptive field: input pixels one unit of layer l sees, per side

start_l
 input position of the centre of unit 0 of layer l

centre of unit u
 $\text{start}_l + u \cdot j_l$

floor
 round down

A VGG-style stack: two 3×3 convs, pool, two 3×3 convs, pool, then one dilated 3×3 conv. The renderer computes receptive field, jump (input pixels between neighbouring outputs) and output size per layer

PROBLEM

How many input pixels wide is the receptive field after conv3, conv3, pool2, conv3, conv3, pool2?

GIVEN

- recurrences: $r_l = r_{l-1} + (k_l - 1) \times j_{l-1}$, $j_l = j_{l-1} \times s_l$
- r = receptive field (pixels), j = jump (input pixels between adjacent outputs), k = kernel, s = stride; start $r_0 = 1$, $j_0 = 1$

1
conv3, conv3 (s = 1)

$r = 1 + 2 = 3$, then $3 + 2 = 5$; $j = 1$

Why: two 3×3 convs see 5×5, like one 5×5 with fewer weights

2
pool2 (s = 2)

$r = 5 + 1 \times 1 = 6$; $j = 2$

Why: stride doubles every later step

3
conv3, conv3

$r = 6 + 2 \times 2 = 10$, then $10 + 4 = 14$; $j = 2$

Why: each 3×3 now adds 2 × j pixels

4
pool2 (s = 2)

$r = 14 + 1 \times 2 = 16$; $j = 4$

Why: jump reaches 4

ANSWER

16 × 16 input pixels after six layers

CHECK

✓ A dilated 3×3 conv (dilation 2) next would add $(2 \cdot (3-1)) \times 4 = 16$ pixels: $r = 32$, with no extra downsampling

Receptive field by hand for the first six layers of the stack above

CNNs	Transformers
Grow it with depth, stride, pooling and dilation	Every token attends to all others: global from layer 1
Effective field is smaller than theoretical (centre-weighted)	Swin restores locality with windows, then grows by merging patches
Cost linear in pixels	Global attention cost grows with (number of patches) ²

🔧 Vision transformers			
Model (year)	Params	Key idea	Trade-off
ViT-B/16 (2020)	86 M	16×16 patches + class token + learned positions, plain encoder	needs huge pretraining (JFT-300M) to beat CNNs
DeiT-S / DeiT-B (2020)	22 M / 86 M	same ViT trained on ImageNet only: strong augmentation, distillation token	recipe, not architecture, closed the data gap
Swin-T (2021)	28 M	windowed attention, shifted every other layer, patch merging → 4-stage pyramid	linear cost in image size; dense-prediction friendly
ConvNeXt-T (2022)	28 M	pure CNN built by copying Swin's design choices	matched Swin at equal FLOPs

Transformer-era backbones (ImageNet-1k classification versions)

Quantity	Formula	ViT-B/16 at 224 px
Patches	$N = (H/P) \cdot (W/P)$	$14 \times 14 = 196$ (+1 class token = 197)
Patch embedding	$P^2 \cdot C \cdot D + D$	$16 \cdot 16 \cdot 3 \cdot 768 + 768 = 590,592$

where H, W = image height and width (pixels); P = patch side (pixels); C = image channels (3 for RGB) (count); D = token width (model dimension) (count)

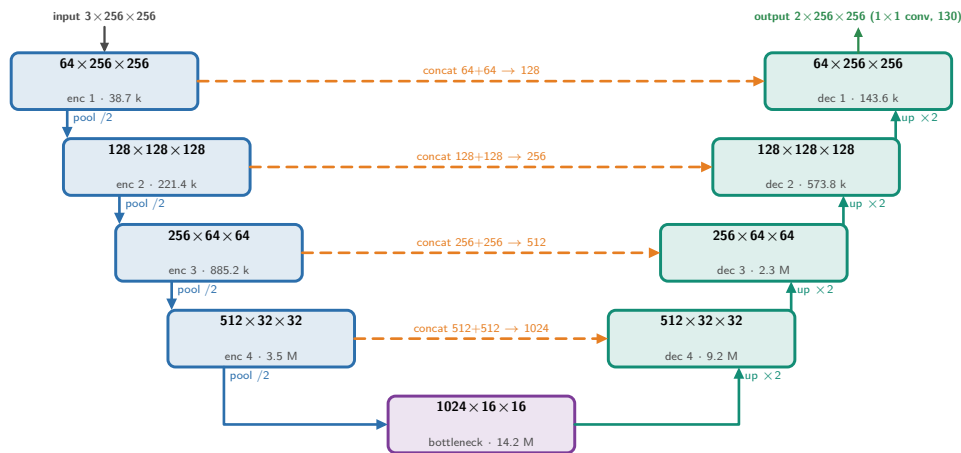
Tokens and patch-embedding weights of a ViT

🗨 Interview trap: resolution

Doubling the input side of a ViT/16 from 224 to 448 px quadruples the tokens (196 → 784) and multiplies global-attention cost by about 16. Swin and CNNs grow linearly. That is why detection and segmentation backbones are hierarchical (Swin, ConvNeXt, ViTDet adds windows).

🔍 Detection and segmentation			
Model (year)	Task	Key idea	Still used for
Faster R-CNN (2015)	detection, 2-stage	region proposal network shares the backbone; RoI pooling then classify	accuracy-first baselines
YOLO v1 (2016) → YOLO11 (2024)	detection, 1-stage	grid predicts boxes and classes in one pass; later anchors, FPN necks, anchor-free heads; YOLOv10 drops NMS	real-time, edge
RetinaNet (2017)	detection, 1-stage	focal loss down-weights easy background	dense detectors
DETR (2020)	detection, set	CNN + transformer encoder-decoder, object queries, Hungarian loss, no anchors or NMS	basis of Deformable DETR, RT-DETR, DINO-DETR
U-Net (2015)	semantic seg.	symmetric encoder-decoder, skips concatenated at each scale	medical images, diffusion denoisers
Mask R-CNN (2017) SAM (2023)	instance seg. promptable seg.	Faster R-CNN + mask head per box, RoIAlign heavy ViT image encoder run once + light prompt encoder and mask decoder	instance masks, keypoints zero-shot masks, labelling tools

Detector and segmenter cards



U-Net (skip connections) $\cdot$ bottleneck $1024 \times 16 \times 16 = 262,144$ values ($\times 1.33$ more than the input) $\cdot$ total 31.0 M parameters

Stage	Operations	Output C $\times$ H $\times$ W	Params
enc 1	$2 \times \text{conv } 3 \times 3, \text{ pool } /2$	$64 \times 256 \times 256$	38,720
enc 2	$2 \times \text{conv } 3 \times 3, \text{ pool } /2$	$128 \times 128 \times 128$	221,440
enc 3	$2 \times \text{conv } 3 \times 3, \text{ pool } /2$	$256 \times 64 \times 64$	885,248
enc 4	$2 \times \text{conv } 3 \times 3, \text{ pool } /2$	$512 \times 32 \times 32$	3,539,968
bottleneck	$2 \times \text{conv } 3 \times 3$	$1024 \times 16 \times 16$	14,157,824
dec 4	up-conv $2 \times 2 \times 2$, concat $512+512$, $2 \times \text{conv } 3 \times 3$	$512 \times 32 \times 32$	9,176,576
dec 3	up-conv $2 \times 2 \times 2$, concat $256+256$, $2 \times \text{conv } 3 \times 3$	$256 \times 64 \times 64$	2,294,528
dec 2	up-conv $2 \times 2 \times 2$, concat $128+128$, $2 \times \text{conv } 3 \times 3$	$128 \times 128 \times 128$	573,824
dec 1	up-conv $2 \times 2 \times 2$, concat $64+64$, $2 \times \text{conv } 3 \times 3$	$64 \times 256 \times 256$	143,552
head	conv 1×1	$2 \times 256 \times 256$	130
Total			31,031,810

down: $H \rightarrow H/2, W \rightarrow W/2$ (stride 2); up: $H \rightarrow 2H$ (2×2 transposed conv, stride 2)

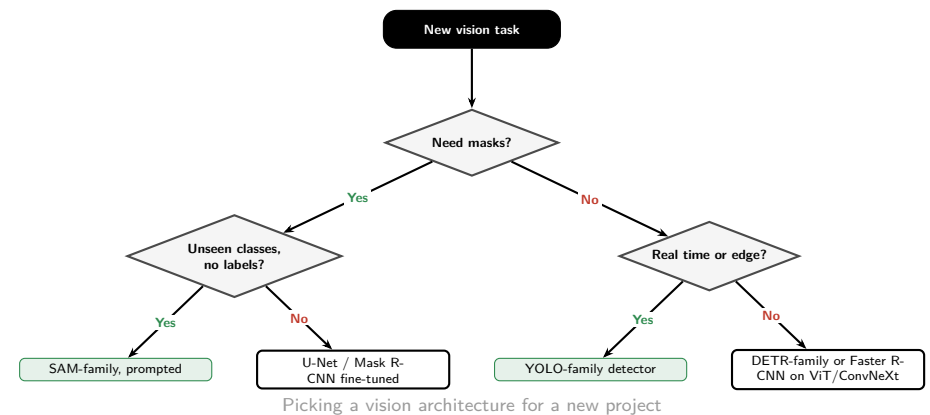
conv params $= k^2 \cdot C_{\text{in}} \cdot C_{\text{out}} + C_{\text{out}}$; up-conv $= 2^2 \cdot C_{\text{in}} \cdot C_{\text{out}} + C_{\text{out}}$

skip concat: $C = C_{\text{up}} + C_{\text{skip}}$ (decoder sees fine detail from the encoder)

where

H, W height and width (pixels) $+ C_{\text{out}}$ one bias per output channel
 $C_{\text{in}}, C_{\text{out}}$ input and output channels $C_{\text{up}}, C_{\text{skip}}$ channels from the up-conv and from the encoder level
 k kernel size (3, same padding keeps $H \times W$)

A U-Net with five encoder stages ($64 \rightarrow 1,024$ channels) on a $3 \times 256 \times 256$ image, two output classes. The renderer computes each stage's tensor shape, the skips and parameter estimates



★ Myths, recap and self-test

MYTH

✗ ViT beat CNNs because attention is better

✗ Deeper plain networks are always better

✗ Fewer parameters means faster

✗ SAM labels what it segments

FACT

✓ It needed far more pretraining data to do so

Trained on ImageNet-1k alone the original ViT trailed ResNets; DeiT closed the gap with augmentation and distillation, ConvNeXt with a modern recipe.

✓ Without residuals, 56 layers trained worse than 20

ResNet's paper showed higher training error for deeper plain nets: an optimisation problem, not over-fitting.

✓ Depthwise convolutions are memory-bound on GPUs

EfficientNet-B0 has 5 times fewer parameters than ResNet-50 but is not 5 times faster on a GPU.

✓ The original SAM returns masks, not class names

Class names come from a detector or text-capable model; SAM 3 (2025) adds concept prompts.

Vision-architecture beliefs to correct

FPN (feature pyramid network) — A top-down path with lateral connections that gives every backbone stage strong semantics for multi-scale detection (2017).

NMS (non-maximum suppression) — Post-processing that removes overlapping duplicate boxes, keeping the highest-scoring one.

Key takeaways

01

Depth became cheap through blocks

Inception bottlenecks, residuals, depthwise separable

02

ViT trades inductive bias for data

DeiT and ConvNeXt showed the recipe matters

03

Hierarchy wins dense tasks

Swin, ConvNeXt, FPN; plain ViT needs windows

04

Detection: two-stage $\rightarrow$ one-stage $\rightarrow$ set prediction

Faster R-CNN $\rightarrow$ YOLO $\rightarrow$ DETR

05

Segmentation: U-Net, Mask R-CNN, then promptable SAM

zero-shot masks since 2023

Q1 Why did ResNet allow 152 layers?

A Residual connections $y = x + F(x)$ give gradients an identity path, so deeper networks optimise at least as well as shallower ones.

Q2 Params of AlexNet, VGG-16, ResNet-50 (approx.)?

A About 61 M, 138 M and 25.6 M.

Q3 What did GoogLeNet's 1×1 convolutions do?

A Reduce channels before expensive 3×3 and 5×5 convs (bottlenecks), giving 22 layers with about 6.8 M parameters.

Q4 Weight saving of a depthwise separable 3×3 conv, $256 \rightarrow 256$ channels?

A 67,840 vs 589,824 weights: about $8.7 \times$ fewer.

Q5 ViT-B/16 tokens at 224×224 ?

A 196 patches + 1 class token = 197.

Q6 What does Swin change in ViT?

A Attention inside local windows shifted every other layer, plus patch merging into a 4-stage pyramid: linear cost in image size.

Q7 How does DETR avoid NMS?

A A fixed set of object queries is matched one-to-one to ground-truth boxes with the Hungarian algorithm, so training penalises duplicates.

Q8 What do U-Net skips carry?

A Encoder feature maps concatenated into the decoder at the same resolution, restoring spatial detail.

Q9 Receptive field after conv3, conv3, pool2 (stride 2)?

A 6 pixels with a jump of 2 ($r = 1 + 2 + 2 + 1$).

Q10 What is ConvNeXt?

A A 2022 pure CNN obtained by modernising ResNet with Swin's choices (7×7 depthwise, inverted bottleneck, LayerNorm, GELU, patchify stem); it matched Swin at equal compute.