PHASE 3
Computer Vision

Convolutions, CNN architectures, transfer learning, detection, segmentation, and ViTs

Phase goal

Understand how neural networks see: the convolution operation, the architectural lineage from LeNet to ResNet to ViT, and the practical craft of transfer learning. By the end you can fine-tune a pretrained model and serve it behind an API.

Estimated time: 4–6 weeks | **Track:** Comprehensive ML & Deep Learning Curriculum

A self-paced, depth-first path from mathematical foundations to production deep learning.

Contents

1 Why images need special architectures

An image is a grid of pixels with strong *spatial structure*: nearby pixels are correlated, and patterns (edges, textures, objects) can appear anywhere. A plain MLP ignores this — it would need a separate weight for every pixel-position pair (a $224 \times 224 \times 3$ image flattened is ∼150,000 inputs) and would not generalize across translations. Convolutional networks bake in two priors that match images: **locality** and **translation equivariance**, with massive **weight sharing**.

Key idea

A convolution slides a small set of learnable weights (a *filter*) across the image, computing the same dot product at every location. This gives (1) far fewer parameters, (2) the ability to detect a pattern wherever it occurs, and (3) a hierarchy: early layers learn edges, later layers learn textures, then parts, then objects.

2 The convolution operation

2.1 Filters, stride, padding

A filter (kernel) of size $k \times k$ is convolved with the input: at each position, multiply the overlapping pixels by the filter weights and sum. **Stride** s is the step size of the slide (larger stride ⇒ smaller output). **Padding** adds a border of zeros so the output can keep the input's spatial size ("same" padding).

Definition

For an input of size W, kernel k, padding p, stride s, the output spatial size is

$$W_{\text{out}} = \left\lfloor \frac{W - k + 2p}{s} \right\rfloor + 1.$$

A convolutional layer with C_{in} input channels and C_{out} filters has $k \cdot k \cdot C_{\text{in}} \cdot C_{\text{out}}$ weights — independent of the image size.

Counting parameters

A 3×3 conv from 64 to 128 channels has $3 \cdot 3 \cdot 64 \cdot 128 = 73{,}728$ weights (plus 128 biases) — regardless of whether the feature map is 56×56 or 7×7. Compare to a fully-connected layer between two $56 \times 56 \times 64$ maps: over 4×10^{10} weights. Weight sharing is the difference between trainable and impossible.

2.2 Pooling and receptive field

Pooling (max or average) downsamples feature maps, adding translation invariance and reducing computation. The **receptive field** is the region of the input that influences one output unit; it grows with depth, so deep layers "see" large parts of the image. Modern nets increasingly replace pooling with strided convolutions.

Intuition

Think of a CNN as a funnel: spatial resolution shrinks ($224 \to 112 \to 56 \to \dots$) while channel depth grows ($3 \to 64 \to 128 \to \dots$). The network trades "where" for "what" — progressively discarding precise location while building richer semantic features.

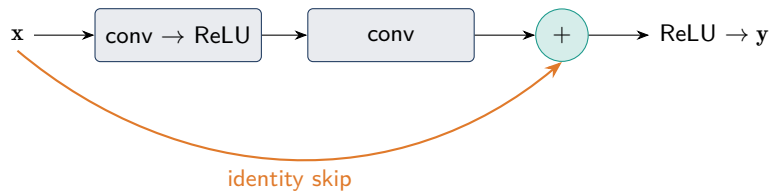
3 CNN architectures: a lineage

Model	Year	Key contribution
LeNet-5	1998	first practical CNN (digit recognition)
AlexNet	2012	ReLU, dropout, GPUs; ignited the deep-learning era
VGG	2014	simplicity: stacks of 3×3 convs; very deep
GoogLeNet/Inception	2014	multi-scale "inception" blocks; 1×1 convs
ResNet	2015	**skip connections** enable 100+ layer training
EfficientNet	2019	principled *compound scaling* of depth/width/resolution

3.1 ResNet and the skip connection

The central problem before ResNet: stacking more layers made networks *harder* to optimize (degradation), not just prone to overfitting. The fix is the **residual block**, which learns a residual $F(\mathbf{x})$ added to the input:

$$\mathbf{y} = F(\mathbf{x}) + \mathbf{x}.$$



Key idea

Skip connections give gradients a "highway" straight back to early layers, defeating the vanishing-gradient problem and making very deep networks trainable. This idea — add the input back — recurs everywhere: it is also at the heart of the Transformer (Phase 4) and diffusion U-Nets (Phase 5). If you remember one architectural trick, remember the residual connection.

3.2 1×1 convolutions and EfficientNet scaling

A 1×1 conv mixes channels without touching spatial dimensions — a cheap way to change depth and add nonlinearity (the "bottleneck" in ResNet). EfficientNet's lesson is that depth, width, and input resolution should be scaled *together* by a single compound coefficient, yielding better accuracy per FLOP.

4 Transfer learning & fine-tuning

You will almost never train a vision model from scratch. Instead, start from weights pretrained on a large dataset (ImageNet) and adapt them — this works because early features (edges, textures) are universal.

Definition

Feature extraction: freeze the pretrained backbone, replace and train only the final classifier head. **Fine-tuning:** unfreeze some or all backbone layers and continue training at a *small* learning rate so you refine rather than destroy the learned features.

Intuition

Rule of thumb by data size and similarity to ImageNet: *little data, similar domain* → freeze backbone, train head; *lots of data* → fine-tune more layers; *very different domain* (e.g. medical, satellite) → fine-tune deeply or train longer. Always use a much smaller LR for pretrained layers than for the new head ("discriminative learning rates").

```python
import torch, torch.nn as nn
from torchvision import models

model = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V2)
for p in model.parameters():           # 1) freeze backbone
    p.requires_grad = False
model.fc = nn.Linear(model.fc.in_features, num_classes)   # 2) new head

opt = torch.optim.AdamW(model.fc.parameters(), lr=1e-3)    # train head only
# Later: unfreeze last block(s) and fine-tune at lr=1e-5 for a few epochs.
```

Common pitfall

Use the *same* preprocessing (resize, normalization mean/std) that the pretrained model expects — mismatched normalization silently wrecks accuracy. And when fine-tuning, a too-large learning rate causes "catastrophic forgetting": the pretrained features are overwritten before they can help.

5 Beyond classification

5.1 Object detection

Detection predicts *what* and *where* (bounding boxes + classes). Two families:

- **Two-stage** (Faster R-CNN): propose regions, then classify/refine them — accurate, slower.
- **One-stage** (YOLO, SSD, RetinaNet): predict boxes and classes in a single pass — fast, real-time. YOLO is the practical default for most applications.

Key concepts: anchor boxes, Intersection-over-Union (IoU), non-maximum suppression (NMS), and mean Average Precision (mAP) as the evaluation metric.

5.2 Image segmentation

Segmentation labels *every pixel*. **Semantic** segmentation classes each pixel; **instance** segmentation also separates object instances. **U-Net** (encoder–decoder with skip connections between matching resolutions) is the workhorse, especially in medical imaging; **Mask R-CNN** extends Faster R-CNN with a mask head. Note U-Net's skip connections recover the spatial detail lost during downsampling — and reappear in diffusion models.

6 Vision Transformers (ViT)

A ViT splits an image into fixed patches (e.g. 16×16), linearly embeds each patch as a token, adds positional embeddings, and feeds the sequence to a standard Transformer encoder (Phase 4). With enough data (or strong pretraining), ViTs match or beat CNNs.

Intuition

CNNs bake in locality and translation equivariance, so they are data-efficient. ViTs have weaker built-in priors but more flexibility, so they shine with large-scale pretraining and can model long-range relationships from layer one. In practice: CNNs (or hybrids like ConvNeXt) for smaller datasets; ViTs when you can leverage big pretrained checkpoints. Both are commodities on Hugging Face / timm today.

7 Practical skills

7.1 Data augmentation

Label-preserving transforms multiply your effective data and are the cheapest accuracy boost in vision: random crops, flips, rotations, color jitter, and stronger schemes (RandAugment, Mixup, CutMix). Use `torchvision.transforms` or **albumentations** (fast, rich, great for detection/segmentation where boxes/masks must transform too).

Common pitfall

Augment the *training* set only; validation/test must see clean, deterministic preprocessing. And keep augmentations realistic — vertically flipping street-sign images or digits teaches the model wrong invariances.

7.2 Efficient data pipelines

Use `Dataset` + `DataLoader` with multiple `num_workers`, pinned memory, and sensible batch sizes. Data loading — not the GPU — is often the bottleneck; cache decoded images, use efficient formats, and profile. On GPU, enable mixed precision (`torch.cuda.amp`) for a large speed/memory win.

8 Checkpoint project

Fine-tune and deploy an image classifier

Goal: fine-tune a pretrained ResNet (or ViT) on a custom dataset and serve it behind a simple API — playing directly to your backend strengths.

1. **Data:** pick or assemble a 3–10 class image dataset; build a `DataLoader` with train-time augmentation and correct normalization.
2. **Train:** feature-extract first (train the head), then fine-tune the last block(s) at a small LR. Track loss/accuracy in Weights & Biases or TensorBoard.
3. **Evaluate:** confusion matrix, per-class accuracy, and a look at the worst misclassifications (error analysis matters more than the single accuracy number).
4. **Interpret:** Grad-CAM heatmaps to confirm the model attends to the object, not the background (a classic "clever Hans" check).
5. **Deploy:** wrap the model in a FastAPI endpoint that accepts an uploaded image and returns top-k predictions; containerize it (foreshadowing Phase 7).

Definition of done: a repo with training script, an evaluation report with the confusion

matrix and Grad-CAM examples, and a running `/predict` endpoint with a sample `curl` command.

Phase 3

- **Stanford CS231n** — the definitive vision course (lecture videos + assignments, free online).
- **Hands-On Machine Learning** (Géron) — the CNN chapters.
- **fast.ai** — Practical Deep Learning, vision lessons (excellent transfer-learning craft).
- Papers (read abstracts + figures at least): *ResNet*, *EfficientNet*, *An Image is Worth 16×16 Words* (ViT).
- Libraries: **torchvision**, **timm** (huge model zoo), **albumentations**.

9 Phase 3 self-check

- Compute conv output sizes and parameter counts from memory.
- Explain why weight sharing and skip connections matter.
- Decide between feature extraction and fine-tuning given a dataset's size and domain.
- Contrast one-stage vs. two-stage detection and explain IoU/NMS/mAP.
- Explain when you would reach for a ViT vs. a CNN.
- Fine-tune a pretrained model and serve it behind an endpoint.