African Institute for Mathematical Sciences — Cameroon
Structured Master's, Data Science Stream · Computer Vision

Deeper, Smarter, Reusable

Week 2 · Day 2: CNN Architectures, What They Look At, and Transfer Learning

Yusuf Brima, Ph.D.

Introduction to Computer Vision · AIMS Cameroon, 2026–2027

Today

1. **A decade of architectures**: LeNet, AlexNet, VGG, Inception
2. **Residual networks**: how to train 150 layers
3. **Efficient networks**: accuracy per operation
4. **What does the network look at?** Receptive fields and Grad-CAM
5. **Transfer learning**: reusing ImageNet knowledge, tested on real bean-leaf disease photos from Uganda

By the end of today you should be able to

- explain the key idea each landmark architecture added;
- explain why residual connections make deep networks trainable;
- compute the savings of 1×1 and depthwise convolutions;
- produce and interpret a Grad-CAM map;
- choose between feature extraction and fine-tuning for a new problem.

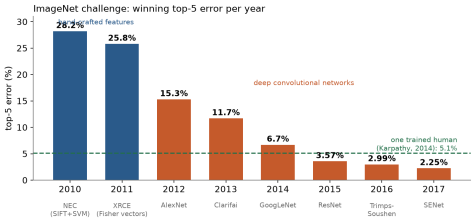
Where we are

Yesterday

- A CNN: conv → ReLU → pool, then a classifier head.
- Cross-entropy, backpropagation, SGD/Adam, batch norm, regularisation.
- Our 2-layer CNN reached about 88% on Fashion-MNIST.

Today

- Real photos need much bigger networks. How should we *arrange* the layers?
- And when we have only a few hundred labelled images, as in most African applications, how do we still get a good model?



Each bar after 2012 is a new architecture (Week 1, Day 1).

Part 1

A decade of architectures

The benchmark that shaped a decade: ImageNet

- The ImageNet challenge (2010–2017): 1.28 million training images, 50 000 validation images, 1000 classes, from "goldfish" to "volcano".
- Every architecture today was first judged by its ImageNet accuracy; its trained weights are what we transfer later today.

Top-1 and top-5 accuracy

Top-1: the highest-scoring class is correct. **Top-5**: the correct class is among the five highest.

One prediction

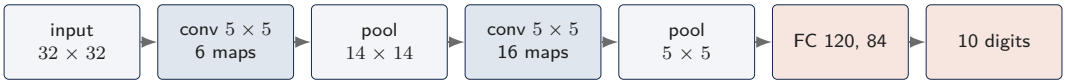
A photo of a Persian cat. Top five predictions: lynx, Persian cat, tabby, tiger cat, Egyptian cat.

Top-1: wrong ("lynx"). Top-5: right.

Intuition

Top-5 forgives near-misses between very similar classes (dozens of dog breeds, five kinds of domestic cat). Modern papers report top-1, which is stricter.

LeNet-5 (1998): the template



- Yann LeCun and colleagues at Bell Labs; about 60 000 parameters.
- Read millions of handwritten cheques in US banks in the late 1990s.
- Already the full recipe: convolution, subsampling, fully connected head, backpropagation.

Intuition

Yesterday's network is essentially LeNet. The ideas were there in 1998; what was missing for natural photos was *data* and *compute* (Week 1, Day 1).

AlexNet (2012): scale it up

- 5 conv layers + 3 fully connected layers; about **61 million** parameters, 1000 ImageNet classes.
- First layer: 11×11 kernels with stride 4 on 224×224 images.
- Trained for about a week on two consumer GPUs.

What made it work

- **ReLU** instead of tanh: trained several times faster.
- **Dropout** in the FC layers.
- **Data augmentation**: random crops, flips, colour changes.
- **GPUs** and 1.2 million labelled images.

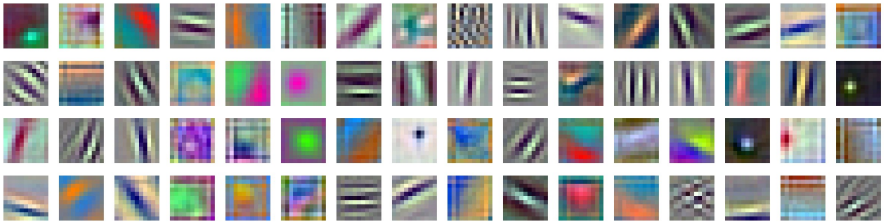
Where the parameters are

The first FC layer maps $6 \times 6 \times 256 = 9216$ values to 4096 neurons: $9216 \times 4096 \approx 37.7$ million weights, over 60% of the whole network.

Intuition

Every ingredient of yesterday's lecture appears here. AlexNet was less an invention than a demonstration that the recipe scales.

What AlexNet's first layer learned



The 64 real $11 \times 11 \times 3$ first-layer filters of AlexNet (torchvision weights), each shown as a small colour image.

Intuition

Oriented edges and stripes at many angles and frequencies (Gabor-like, Week 1, Day 1), and colour blobs. The same building blocks as the ResNet filters we saw on Week 1, Day 5, from a different network trained years earlier.

VGG (2014): small filters, deep stacks

- Use **only** 3×3 **convolutions**, in stacks of 2–3, then max pool. Double the channels after each pool: 64, 128, 256, 512, 512.
- 16 or 19 weight layers; 138 million parameters.
- Simple and uniform: still a favourite feature extractor.

Why stack 3×3?

Three 3×3 layers see a 7×7 region, like one 7×7 layer.

With C channels in and out:
one 7×7: $49C^2$ weights;
three 3×3: $3 \times 9C^2 = 27C^2$ weights.

45% fewer weights, and three ReLUs instead of one.

Intuition

Depth buys expressiveness cheaply: each extra layer can combine the previous layer's patterns into more complex ones. But VGG-19 was close to the limit of what plain stacks could train.

The hidden costs of VGG

Where VGG-16's parameters are

The first FC layer maps $7 \times 7 \times 512 = 25\,088$ values to 4096:
$25\,088 \times 4096 \approx 102.8$ million weights, three quarters of VGG-16's 138 million.

Memory for activations

First two layers: $224 \times 224 \times 64 \approx 3.2$M numbers each. All layers: about 24M numbers, roughly 93 MB per image forward (CS231n estimate), all kept for backprop.

Global average pooling

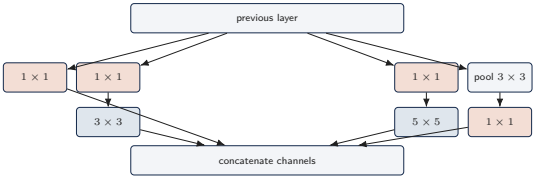
Replace flatten + huge FC layers by averaging each final feature map to a single number:

$$7 \times 7 \times 512 \;\rightarrow\; 512$$

Then one small FC layer: $512 \times 1000 \approx 0.5$ million weights instead of over 100 million.

Intuition: GAP asks "was this part present anywhere?", and works for any input size.

GoogLeNet / Inception (2014): look at several scales at once



- Objects appear at many sizes: let each block try 1×1, 3×3 and 5×5 filters *in parallel* and let training decide.
- Orange 1×1 convolutions shrink the channels first, keeping the cost down.
- 22 layers, only about 7 million parameters: 20 times fewer than VGG.

The 1×1 bottleneck

5×5 conv, 256 $\to$ 64 channels: $5^2 \cdot 256 \cdot 64 = 409\,600$ weights.
First 1×1 to 32 channels: $256 \cdot 32 + 5^2 \cdot 32 \cdot 64 = 59\,392$. Seven times cheaper.

The degradation problem: deeper was worse

- If 20 layers work, 56 should work at least as well: the extra 36 could simply learn to copy their input.
- In practice, He et al. (2015) found a plain 56-layer network had **higher training error** than a 20-layer one on CIFAR-10.
- Not overfitting (that would show only on test data): the deeper network was simply *harder to optimise*.

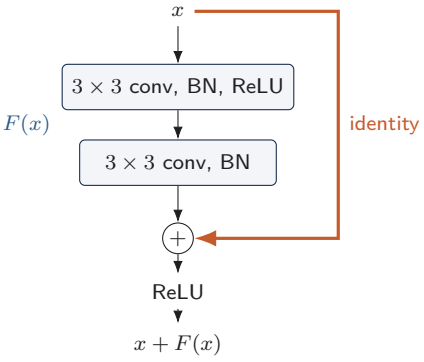
Intuition

Learning "do nothing" (the identity) through a stack of convolutions and ReLUs is surprisingly hard: the weights must be set very precisely. And the gradient reaching the early layers fades as it passes back through many layers.

The ResNet idea

Do not ask a block to learn the whole output $H(x)$; ask it to learn only the *change* $F(x) = H(x) - x$, and add the input back: output $= x + F(x)$.

The residual block

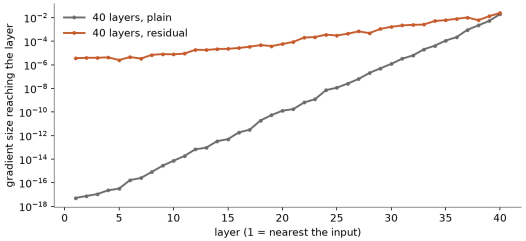


- The orange **shortcut** carries x around the block unchanged.
- If the block has nothing useful to add, it learns $F(x) \approx 0$: easy, just shrink the weights.
- In the backward pass, the gradient flows straight through the "+": every layer gets a direct line to the loss.

Intuition

Editing a draft is easier than writing from scratch. Each residual block receives a draft (x) and only suggests edits ($F(x)$).

Residual connections keep gradients alive



Two 40-layer CNNs at initialisation, one batch of Fashion-MNIST. We measure the size of the gradient reaching each layer. Plain: the gradient at layer 1 is 5.2×10^{-18}, at layer 40 2.0×10^{-2}. Residual: 3.7×10^{-6} and 2.5×10^{-2}.

Intuition

In the plain network the first layers receive essentially nothing: they will never change. With shortcuts the signal still fades with depth (our blocks also rescale by $1/\sqrt{2}$), but it reaches layer 1 about a trillion times stronger: enough to learn.

ResNet in practice

- ResNet-18/34: stacks of the basic block; ResNet-50/101/152: the **bottleneck** block.
- Four stages; each halves the resolution and doubles the channels: $56 \to 28 \to 14 \to 7$.
- Global average pooling, then one FC layer: no huge FC layers.
- ResNet-152 won ImageNet 2015 with 3.57% top-5 error.

The bottleneck block, 256 channels

1×1 to 64, 3×3 at 64, 1×1 back to 256:
$256 \cdot 64 + 9 \cdot 64 \cdot 64 + 64 \cdot 256 \approx 69{,}600$ weights.

Two 3×3 convs at 256 channels:
$2 \times 9 \cdot 256^2 \approx 1.18$ million: 17 times more.

Practitioner's Note

ResNet-50 (25.6M parameters) is still the default backbone in countless papers and products, ten years later.

ResNet-50, stage by stage

Stage	Output size	Blocks	Typical content
input	$224 \times 224 \times 3$		
stem: conv $7 \times 7/2$, pool/2	$56 \times 56 \times 64$		fast early downsampling
layer1	$56 \times 56 \times 256$	3 bottlenecks	edges, colours, simple textures
layer2	$28 \times 28 \times 512$	4	textures, small parts
layer3	$14 \times 14 \times 1024$	6	parts and their combinations
layer4	$7 \times 7 \times 2048$	3	object-level patterns
average pool, FC	$2048 \to 1000$		class scores

Intuition

Resolution halves and channels double at each stage: the network trades *where* for *what*. Detection and segmentation (Days 3–4) tap these stages at different resolutions.

Another view: a residual network is many networks at once

- Each residual block offers two routes: through the block, or around it.
- With n blocks, an input can take 2^n different paths to the output, of every length from 0 to n blocks.
- Veit et al. (2016): deleting one block from a trained ResNet barely hurts it; deleting one layer from VGG destroys it.

How many paths?

ResNet-50 has 16 bottleneck blocks: $2^{16} = 65\,536$ paths. Most of the gradient travels along the *short* ones.

Intuition

A ResNet behaves like a large ensemble of shallower networks that share weights: the benefits of depth without most of its training difficulty.

Squeeze-and-excitation (2017): which channels matter here?

1. **Squeeze**: global-average-pool each channel to one number: "how active is this detector in this image?"
2. **Excite**: a tiny two-layer network turns those numbers into a weight between 0 and 1 per channel.
3. **Scale**: multiply each channel by its weight.

Added to ResNet blocks, SENet won the last ImageNet challenge (2017) with 2.25% top-5 error.

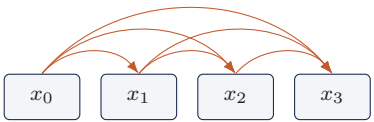
Intuition

The network turns some detectors up and others down *depending on the image*: on a leaf photo, spot detectors; on a street, wheel detectors. A first form of *attention*, the topic of Day 5.

Cost

256 channels reduced to 16 and back: $256 \cdot 16 + 16 \cdot 256 \approx 8200$ weights per block.

DenseNet (2017): reuse everything



- Each layer receives the feature maps of *all* earlier layers in its block (concatenated, not added).
- Each layer adds only a few new channels (the "growth rate").
- Very parameter-efficient: DenseNet-121 has 8 million parameters.

Intuition

ResNet: "here is the draft, suggest edits". DenseNet: "here are all previous drafts, add a new note". Both give gradients short paths back to every layer.

Efficient networks: depthwise separable convolution

- A normal convolution does two jobs at once: look at a *neighbourhood* and *mix channels*.
- **MobileNet** (2017) splits them:
  1. **depthwise**: one 3×3 filter per channel, no mixing;
  2. **pointwise**: a 1×1 conv to mix channels.

3×3, 256 $\to$ 256 channels

Standard: $9 \cdot 256 \cdot 256 = 589\,824$ weights.
Depthwise + pointwise: $9 \cdot 256 + 256 \cdot 256 = 67\,840$.
8.7 times fewer weights and multiplications.

- **MobileNetV3** (2019): designed by automated search for phones.
- **EfficientNet** (2019): scale depth, width and resolution *together* by fixed ratios.
- **ConvNeXt** (2022): a ResNet modernised with ideas from transformers (large 7×7 depthwise kernels, LayerNorm, GELU); matches vision transformers (Day 5).

Practitioner's Note

Separable convolution is Week 1's separable filter idea (Day 3), applied to channels instead of rows and columns.

EfficientNet: scale depth, width and resolution together

- A network improves with more layers (depth), more channels (width) or more pixels (resolution). Each alone saturates quickly.
- Tan & Le (2019): scale all three together by fixed ratios found by a small search.

Compound scaling

depth $\times 1.2^{\phi}$, width $\times 1.1^{\phi}$, resolution $\times 1.15^{\phi}$,

chosen so that $1.2 \times 1.1^2 \times 1.15^2 \approx 2$: each step of ϕ roughly doubles the computation.

One step up from B0

$\phi = 1$: about 20% more layers, 10% more channels, images about 15% larger ($224 \rightarrow 258$ pixels): about twice the multiply-adds.

Intuition

A bigger image has more detail, which needs more layers to see larger patterns and more channels to describe them. The three dimensions are not independent.

Parameters, operations and speed are different things

- **Parameters**: memory to store the model; overfitting risk.
- **Operations (FLOPs)**: arithmetic per image.
- **Latency**: what the user waits for, on *their* hardware.

From our measurements on a laptop CPU: VGG-16 1093 ms per image; MobileNetV3 155 ms.

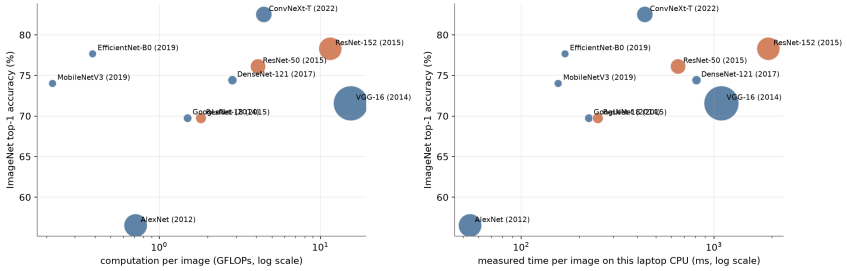
Intuition

Depthwise convolutions have few FLOPs but use hardware less efficiently than big dense convolutions, so their real speed-up is often smaller than the FLOP count suggests. Moving data in memory, not arithmetic, is often the bottleneck.

Practitioner's Note

Always time the model on the device it will run on (phone, Raspberry Pi, Colab GPU), with the real input size.

Ten architectures, compared



Left: accuracy vs. computation (torchvision model cards). **Right**: the same accuracy vs. time we measured on a laptop CPU (one image). VGG-16 needs 1093 ms per image; MobileNetV3 155 ms, with higher accuracy.

Intuition

Progress after 2015 is mostly *efficiency*: AlexNet scored 56.5% with 61M parameters; EfficientNet-B0 scores 77.7% with 5.3M. Architecture design is about where to spend computation.

What each architecture added

Architecture	Year	Key idea	Why it mattered
LeNet-5	1998	conv + pool + FC, trained by back-prop	the template
AlexNet	2012	ReLU, dropout, augmentation, GPUs	deep learning works on real photos
VGG	2014	only 3×3 convs, depth 16–19	depth is cheap if filters are small
GoogLeNet	2014	parallel scales, 1×1 bottlenecks	$20\times$ fewer parameters than VGG
ResNet	2015	residual connections	150+ layers train reliably
DenseNet, SENet	2017	feature reuse; channel attention	efficiency; image-dependent weighting
MobileNet, EfficientNet	2017–19	depthwise convs; compound scaling	accuracy per operation; phones
ConvNeXt	2022	ResNet modernised with transformer ideas	CNNs competitive with ViTs (Day 5)

Intuition: read the right-hand column: most advances made networks *easier to train* or *cheaper to run*, rather than adding a new operation.

Choosing an architecture

Situation	Good starting point	Why
Default, server or Colab GPU	ResNet-50, ConvNeXt-T	well understood, strong, many pre-trained weights
Phone, offline, low power	MobileNetV3, EfficientNet-B0	few operations, fast on CPU
Very small dataset	ResNet-18/50, frozen features	less to learn, less to overfit
Highest accuracy, lots of data	ConvNeXt, ViT (Day 5)	scale with data and compute
Detection / segmentation	ResNet or ConvNeXt backbone	spatial feature maps at several scales

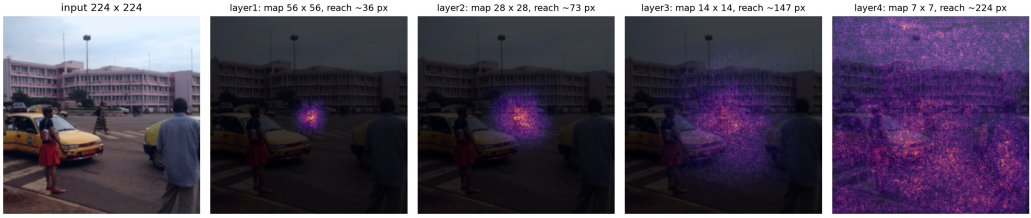
Practitioner's Note

Do not design an architecture for a new problem. Start from a well-tested one with pre-trained weights, and spend your effort on data and evaluation.

Part 2

What does a network look at?

Receptive fields grow with depth



Bdx, CC BY-SA 4.0

We picked the centre unit of each ResNet-18 stage and asked: which input pixels can change its value? (The size of the gradient with respect to the input.) Measured reach: about 36, 73, 147 and 224 pixels.

Intuition

Early units see a few pixels (an edge); late units see most of the image (a whole taxi). Notice the influence is strongest at the centre and fades smoothly: the *effective* receptive field is Gaussian-shaped.

Grad-CAM: where is the evidence?

1. Run the image forward; pick a class c.
2. Backpropagate the class score s_c to the last conv layer's feature maps A^k (e.g. $7 \times 7 \times 2048$).
3. Weight each map by its average gradient:

$$\alpha_k = \text{mean}_{i,j} \frac{\partial s_c}{\partial A^k_{ij}}$$

4. Heat map $= \text{ReLU}\left(\sum_k \alpha_k A^k\right)$, upsampled onto the image.

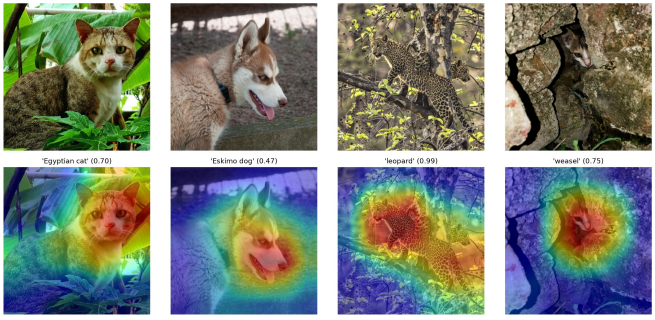
Intuition

Each feature map is a "part detector". α_k says how much that part pushes the score for class c up. The heat map adds the detectors' outputs, each weighted by its importance: *where* the evidence for c lies.

Practitioner's Note

Grad-CAM (Selvaraju et al., 2017) needs no retraining and works on any CNN. Use it to check *why* a model is right, not only *whether*.

Grad-CAM on real photographs

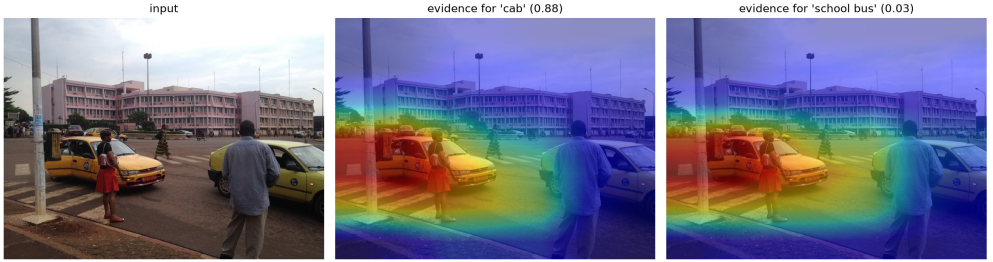


Photos: RyanDannyChacko; MrPanyGoff; Davidvraju; Wikimedia user Rohit Sav (CC BY-SA)

ResNet-50 (ImageNet). The cat behind leaves: evidence on the face. The husky: on the face and ears. The leopard: on its spotted body among the branches.

And the cat behind the rock, called "weasel" on Week 1, Day 1: the evidence is the small head peering out of the crack. Grad-CAM shows the network was looking at the right place and still drew the wrong conclusion.

One image, two classes



BY-SA 4.0

Bdx, CC

The same Yaoundé street photo, explained for the network's top two ImageNet classes ('cab' and 'school bus'). Each class draws its evidence from a different part of the scene.

Intuition

A classifier assumes one label per image, but real scenes contain many objects. Grad-CAM reveals this tension, and motivates detection (Day 3) and segmentation (Day 4).

Explanation methods, and how they can mislead

A small family

- **Occlusion** (Week 1, Day 1): hide patches, watch the score. Slow but intuitive.
- **Gradient saliency**: $|\partial s_c / \partial \text{pixel}|$. Fast but noisy.
- **Grad-CAM**: class-specific, coarse (7×7, upsampled).
- **Integrated gradients**, **SHAP**: attributions with theoretical guarantees.

Sanity checks

Adebayo et al. (2018) found that some saliency methods produce almost the same picture even when the network's weights are *randomised*: they show image edges, not the model's reasoning. Check that your explanation changes when the model changes.

Intuition

An explanation is a hypothesis about the model, not proof. Test it: occlude the highlighted region; does the prediction really change?

Part 3

Transfer learning

The problem: we rarely have a million labelled images

- ResNet-50 has 25 million weights and was trained on 1.28 million labelled images.
- A plant pathologist in Uganda or Cameroon may have a few hundred labelled leaf photos.
- Training a big network from scratch on a few hundred images: it memorises them (yesterday's overfitting).

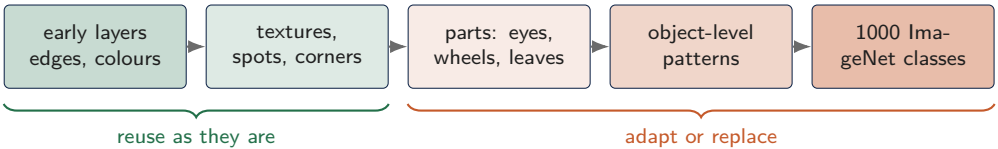
Intuition

But the early layers of an ImageNet network detect edges, textures, spots and shapes, which are useful for *any* photograph, leaves included. Why learn them again?

Transfer learning

Start from a network trained on a large dataset (the *source*) and adapt it to your problem (the *target*).

Layers go from general to specific



- Yosinski et al. (2014): first-layer features transfer almost perfectly between tasks; the last layers are specific to the source classes.
- The last layer (1000 ImageNet classes) is always replaced by a new one for your classes.

Intuition

Like a doctor retraining as a vet: anatomy, physiology and clinical reasoning transfer; only the species-specific knowledge must be learned anew.

Three ways to transfer

1. Feature extraction

Freeze the whole network; use it to turn each image into a feature vector; train a simple classifier (logistic regression) on top.

Cheap; works with very little data; cannot adapt the features.

2. Fine-tune the top

Replace the head; unfreeze the last block(s); train with a small learning rate.

Adapts high-level features to the new domain; moderate data and compute.

3. Fine-tune everything

Unfreeze all layers; start from the pre-trained weights; small learning rate.

Best with more data or a very different domain (X-rays, satellite bands).

Practitioner's Note

Rule of thumb: little data and a similar domain $\rightarrow$ option 1. More data or a different domain $\rightarrow$ options 2–3. Always compare with option 1 as a baseline: it takes minutes.

What ImageNet does not contain

- ImageNet's photos were collected from the web, mostly from North America and Europe.
- DeVries et al. (2019) photographed the same household objects (soap, stoves, toothbrushes) in homes around the world: commercial recognisers were roughly 10–20% less accurate on images from low-income households, many in Africa and Asia.
- Plant diseases, parasites, X-rays and local vehicle types barely appear at all.

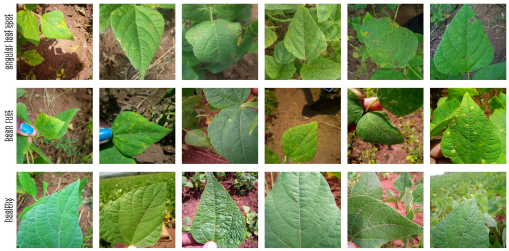
Intuition

Transfer learning brings ImageNet's *general visual vocabulary* (edges, textures, shapes), which travels well. It also brings its *blind spots*, which do not. Your own data must fill them.

Practitioner's Note

Fine-tune and test on images from where the system will be used, taken with the devices people there actually use.

A real case study: bean diseases in Uganda

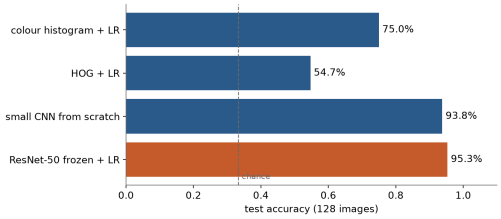


The **iBean** dataset (Makerere AI Lab, Uganda): leaf photos taken with phones in farmers' fields, labelled by experts from the National Crops Resources Research Institute. 1034 training, 133 validation, 128 test images.

Intuition

Beans are a staple across East and Central Africa, Cameroon included. Angular leaf spot and bean rust can destroy much of a harvest; a phone that recognises them early helps a farmer act in time.

Four approaches, one test set



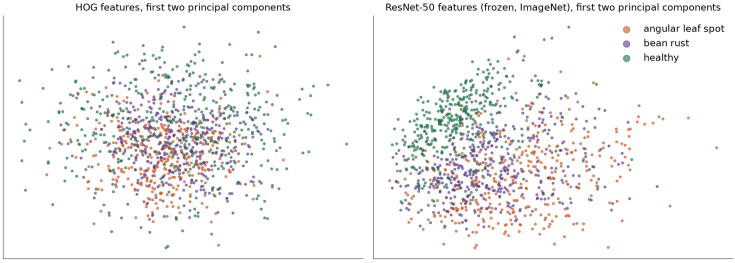
Same 1034 training and 128 test images for all.

- **Colour histograms** (Week 1, Day 2): 75.0%.
- **HOG** (Week 1, Day 4): 54.7%.
- **Small CNN from scratch** (4 conv layers, 13 min): 93.8%.
- **Frozen ResNet-50 + logistic regression**: 95.3%, no network training.

Intuition

With about 1000 images of three visually distinct classes, a small CNN trained from scratch for 13 minutes gets close. Frozen ImageNet features, which never saw a bean, do slightly better *with no network training at all*, and win by far when labels are scarce (next slides). Note too that colour beats HOG: the diseases are mainly colour patterns.

Why: what the features see



Every bean image projected onto its first two principal components. In HOG space the three classes overlap almost completely. In ResNet-50 feature space they already form separate clouds, before any training on beans.

Intuition

A good representation makes the classifier's job easy (Week 1, Day 5). ImageNet taught the network to describe textures, spots and colour patterns well enough that a straight line separates disease from health.

Feature extraction in code

```python
import torchvision, torch
from sklearn.linear_model import LogisticRegression
w = torchvision.models.ResNet50_Weights.
    IMAGENET1K_V1
net = torchvision.models.resnet50(weights=w).eval()
net.fc = torch.nn.Identity()          # remove the
    ImageNet head
prep = w.transforms()

@torch.no_grad()
def features(images):                 # list of PIL
     images
    return net(torch.stack([prep(im) for im in
     images])).numpy()

F_train, F_test = features(train_imgs), features(
    test_imgs)
clf = LogisticRegression(max_iter=4000).fit(F_train
    , y_train)
print(clf.score(F_test, y_test))      # 2048
    features per image
```

- The network runs once per image; training the classifier then takes seconds.
- Cache the features to disk: try many classifiers and settings instantly.
- This is how the bean results on the previous slides were produced.

Where does the frozen model still go wrong?

Confusion matrix on the 128 test leaves

true / predicted	spot	rust	healthy
angular leaf spot	41	2	0
bean rust	4	39	0
healthy	0	0	42

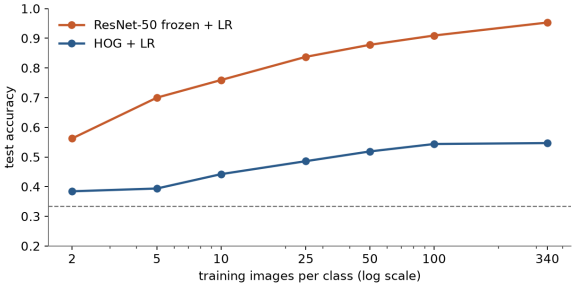
Intuition

Read the off-diagonal cells: which pairs of classes does the model confuse? For crop diseases, the two diseases often look alike early on, as they do to non-experts.

Practitioner's Note

Which error is worse depends on what the farmer does next. If both diseases get the same treatment, confusing them matters little; calling a diseased leaf healthy matters a lot.

How many labelled images do you need?



Accuracy on the 128 test images as we give each method more training images per class (averaged over 5 random draws).

- With **10 images per class**: frozen ResNet-50 75.9%; HOG 44.2%.
- With **50 per class**: 87.8% vs. 51.9%.

Practitioner's Note

For a new crop disease, 30–50 well-chosen, well-labelled photos per class plus a pre-trained network give a usable first model. Spend the next effort on labels from *other* fields, seasons and phones, not on a fancier architecture.

Learning from a handful of examples

Two labelled leaves per class

Six training images in total.

Frozen ResNet-50 + logistic regression: 56.2% on the test set (average of 5 random draws).
HOG + logistic regression: 38.4%.
Chance: 33%.

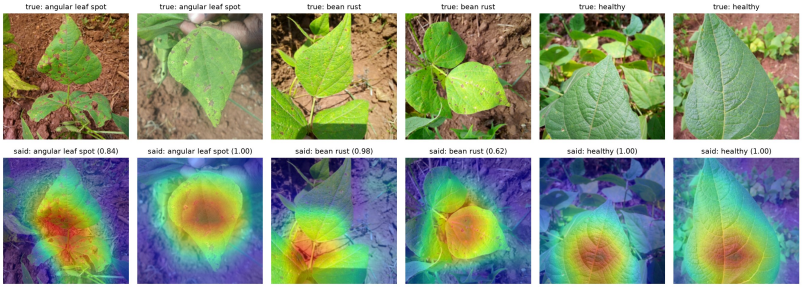
Intuition

With good features, a handful of examples already points the classifier in roughly the right direction. With poor features, six examples teach almost nothing.

Practitioner's Note

This is the idea behind **few-shot learning**, and behind zero-shot models like CLIP (Week 1, Day 1), which need no labelled examples at all. For deployment, though, collect more: two examples cannot represent field conditions.

Fine-tuning: adapting the features



ResNet-18, ImageNet weights, new 3-class head; only the last block and the head trained, 3 epochs at 224×224 (5 minutes on CPU). Test accuracy after each epoch: 92.2%, 93.0%, 95.3%. Grad-CAM shows the evidence on the lesions.

Intuition

The evidence sits on the leaf itself, not on the soil or the hand holding it: the check you want before trusting a model in a field. Grad-CAM is coarse (7×7), so it shows the region, not individual spots.

Transfer learning in a few lines of PyTorch

```
import torchvision, torch.nn as nn
w = torchvision.models.ResNet18_Weights.
    IMAGENET1K_V1
model = torchvision.models.resnet18(weights=w)
model.fc = nn.Linear(512, 3)          # new head: 3
    bean classes

for name, p in model.named_parameters():   # freeze
     all but
    p.requires_grad = name.startswith(("layer4", "
    fc"))

preprocess = w.transforms()       # resize, crop,
    ImageNet mean/std
opt = torch.optim.Adam(
    [p for p in model.parameters() if p.
    requires_grad], lr=3e-4)
# ... then the usual training loop from Day 1
```

- **Same preprocessing** as the pre-trained model: its input size and normalisation statistics (Week 1, Day 2).
- **Smaller learning rate** than from scratch: we only want to nudge good weights.
- Keep frozen layers' batch norm in eval mode.

Learning rates when fine-tuning

- Pre-trained weights are already good: big steps destroy them ("catastrophic forgetting").
- Typical: about 10 times smaller than from scratch, e.g. 10^{-4} instead of 10^{-3} with Adam.
- **Discriminative learning rates**: smallest for early layers, larger for later layers, largest for the new head.
- **Gradual unfreezing**: train the head first, then unfreeze one stage at a time.

A common schedule

Head: 10^{-3}. layer4: 10^{-4}. layer3: 3×10^{-5}. layers 1–2: frozen.

Once the head has converged, unfreeze layer3 for a few more epochs.

Intuition

Early layers hold knowledge that transfers almost unchanged; later layers must adapt. The learning rates encode how much you trust each.

When your images are not RGB

- X-rays and microscopy: 1 channel. Satellites: 4–13 bands (Week 1, Day 2). Pre-trained first layers expect 3.
- **Grayscale**: repeat the channel three times, or sum the first-layer weights over R, G, B.
- **Multispectral**: a new first layer with more input channels; copy the RGB weights; initialise the extra bands (e.g. near-infrared) from their average.

Sentinel-2 into ResNet-50

Pre-trained conv1: 64 filters of $7 \times 7 \times 3$. New conv1: 64 filters of $7 \times 7 \times 13$. Copy the weights for red, green and blue; set the other ten bands to the mean of the three; rescale all by $3/13$ to keep the output scale similar.

Practitioner's Note

For very different domains, models pre-trained on that domain (medical, satellite) often beat ImageNet weights. Search for them first.

Where to find pre-trained models

- **torchvision**: classic CNNs and ViTs, detection and segmentation models; documented accuracy.
- **timm**: over a thousand image models with consistent interfaces and benchmarks.
- **Hugging Face Hub**: transformers, CLIP, DINO, SAM, captioning models.
- **Ultralytics**: YOLO for detection, segmentation and pose (Days 3–4).

Check before you use

- **Licence**: some weights forbid commercial use.
- **Training data**: which population and devices?
- **Preprocessing**: input size, normalisation, channel order.
- **Reported accuracy**: on which test set?

A transfer-learning project in six steps

1. **Define the decision**: who uses the output, and which errors are costly?
2. **Collect and split data by source**: put different fields, clinics, days or phones in the test set.
3. **Baseline**: frozen features + logistic regression. Minutes of work; a number to beat.
4. **Fine-tune**: unfreeze the last stage with a small learning rate and label-preserving augmentation.
5. **Inspect**: confusion matrix, worst mistakes, Grad-CAM on right *and* wrong predictions.
6. **Stress-test**: new location, new device, poor lighting. Report what breaks.

Intuition

This is also a good outline for your capstone report: decision, data, baseline, model, inspection, limitations.

Transfer learning: practical advice

Do

- Start with the frozen-feature baseline; it takes minutes.
- Augment with transformations that preserve the label: flips and rotations for leaves; *not* flips for text.
- Hold out whole fields, farms, hospitals or days for testing.
- Look at Grad-CAM and at the mistakes.

Watch out for

- **Domain shift**: ImageNet photos vs. X-rays, microscopy or multispectral images. Transfer still helps, but fine-tune deeper.
- **Shortcuts**: a background, a watermark or one photographer's style that correlates with the label (Week 1, Day 5).
- **Class imbalance**: report per-class accuracy, not only the average.

Intuition: transfer learning moves the hard part from "train a network" to "collect and check the right data", where local knowledge matters most.

Part 4

Wrapping up

Key takeaways

1. **AlexNet**: scale + ReLU + dropout + GPUs. **VGG**: deep stacks of 3×3. **Inception**: parallel scales, 1×1 bottlenecks.
2. **ResNet**: learn the change, $x + F(x)$; gradients reach every layer; 150+ layers train.
3. **Efficient nets**: depthwise separable convolutions; accuracy per operation.
4. Receptive fields grow with depth; **Grad-CAM** shows where the evidence lies.
5. **Transfer learning**: reuse ImageNet features; frozen features + logistic regression is a strong baseline; fine-tune when you have more data or a different domain.
6. On real Ugandan bean leaves: 95.3% with a frozen ResNet vs. 54.7% with HOG.

Before the next lecture

Practise (tutorial with the tutors)

- Load three pre-trained architectures; count their parameters; time them on your machine.
- Reproduce the bean experiment: frozen features, then fine-tuning.
- Run Grad-CAM on your own photos; find one where the model is right for the wrong reason.

Read

- He et al. (2016), *Deep residual learning for image recognition*
- Zhang et al., *Dive into Deep Learning*, Ch. 8 and §14.2 (fine-tuning)

Next lecture: object detection

- From "what" to "what and where"
- Boxes, IoU, non-maximum suppression
- R-CNN, Fast and Faster R-CNN
- YOLO and single-shot detectors
- Measuring detectors honestly: precision, recall, mAP

Credits

iBean dataset: Makerere AI Lab and NaCRRI, Uganda (MIT licence). Pre-trained weights and model-card numbers: torchvision.
Fashion-MNIST: Zalando Research (MIT licence). All experiments were run for this lecture with `Week2/tools/make_week2_figures.py`.

Photographs from Wikimedia Commons:
`yaounde_taxis`: Bdx, CC BY-SA 4.0
`cat_leaves`: RyanDannyChacko, CC BY-SA 4.0
`husky`: MrPanyGoff, CC BY-SA 3.0
`leopard_tree`: Davidvraju, CC BY-SA 4.0
`cat_rock`: Wikimedia user Rohit Sav, CC BY-SA 4.0