Neural Networks 2 - Convolutional Neural Networks and Transfer Learning

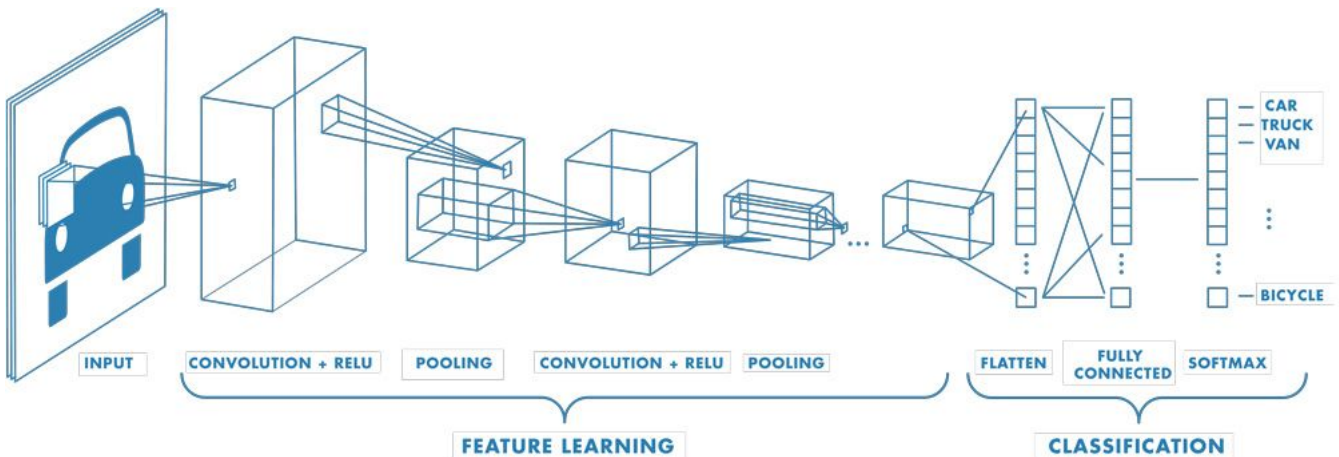
18NES2 - Week 7, Winter semester 2025/26

Zuzana Petříčková

November 11, 2025

What We Covered Last Time

- **Convolutional Neural Networks (CNNs)**
  - A specialized type of deep neural network for processing image data
  - Efficient feature extraction using convolutional layers (filters)



Source:

https://matlabacademy.mathworks.com/details/deep-learning-onramp/deeplearning

What We Covered Last Time

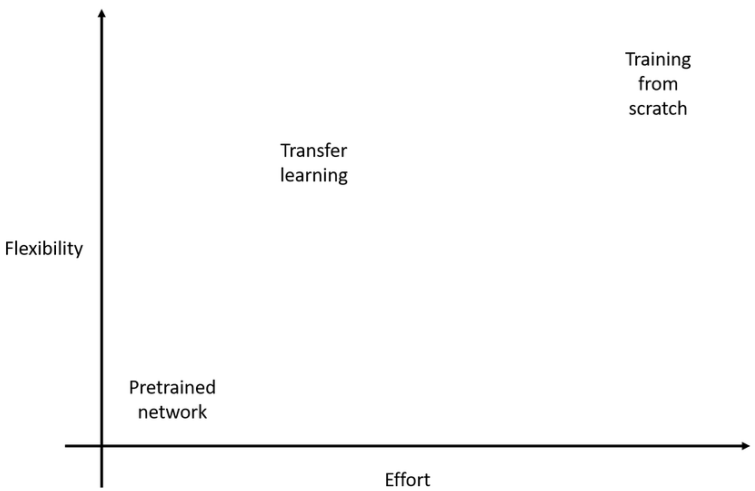
- Motivating example: bird species classification
- Convolution operation – intuition, purpose, and parameters
- CNN architecture (layers, filters, pooling)
- Classic CNN bipyramidal architecture and the Fashion MNIST example
- Training a CNN from scratch and the Cats and Dogs example
  - Image preprocessing
  - Efficient data loading using data loaders
  - Visualization of filters and feature maps
  - Regularization: Data Augmentation, Dropout, Batch Normalization
- 4th homework assignment

This Week

1. Ways to Build and Train a CNN
   - Training the model from scratch
   - Using a Pretrained Model
   - From Pretrained Models to Transfer Learning
   - Transfer Learning
   - Fine-tuning
   - Limitations of Transfer Learning
2. CNN architectures and design patterns
   - Evolution of CNN Architectures
   - Practical Examples
3. Graded Homework

Ways to Build and Train a Convolutional Neural Network

- Training from scratch
- Using a pretrained model
- Transfer learning
- Fine-tuning a pretrained model



Source:

https://matlabacademy.mathworks.com/details/deep-learning-onramp/deeplearning

Last week: Training The Model from Scratch

Example: Training a model from scratch on a small dataset (Cats vs Dogs)
cats_dogs_from_scratch.ipynb

- Dataset: **Kaggle Cats vs Dogs** (25000 images, 500 MB compressed)
- Two classes: **Cat** and **Dog**
- We use a smaller subset of 4000 images (balanced)
- Suitable for demonstrating CNN from scratch, overfitting on small data, and regularization/augmentation

Last week: Training The Model from Scratch

Task:

- Build a simple bipyramidal CNN from scratch
- Train on the small dataset split; monitor training and validation curves

Observations

- **Baseline (no regularization):** test accuracy around **70%**; the model **overfits quickly** (validation accuracy peaks early; validation loss rises after a few epochs).
- Compared to Fashion-MNIST, the model needs more inductive bias and variability in the data (natural images, backgrounds, poses).
- Saliency often highlights facial regions (eyes, muzzle), but can be distracted by background clutter.

How to improve generalization?

Last week: Training The Model from Scratch

Convolutional Neural Networks and Generalization

- Compared to MLPs, CNNs typically learn more slowly, especially on CPUs
- Convolutional neural networks tend to suffer more from overfitting
- Overfitting is a major issue when only a small dataset is available (hundreds or a few thousand samples)
- How can generalization be improved?
  - **Standard regularization**: early stopping, dropout, normalization (for deep models)
  - **Data augmentation**: expanding the dataset *on the fly*
  - **Transfer learning**

Last week: Training The Model from Scratch

cats_dogs_from_scratch.ipynb

- The model without regularization generalized very poorly.

With augmentation and regularization

- **Accuracy improves from about 0.65-0.7 to about 0.8-0.85 on the test set.**
- **Overfitting is delayed and weaker**: training and validation curves stay closer for longer, the validation peak occurs later.

Next Step

- How about using a pretrained model or transfer learning?

Using a Pretrained Model

What if we could use an existing model trained on a large dataset?

Pretrained models:

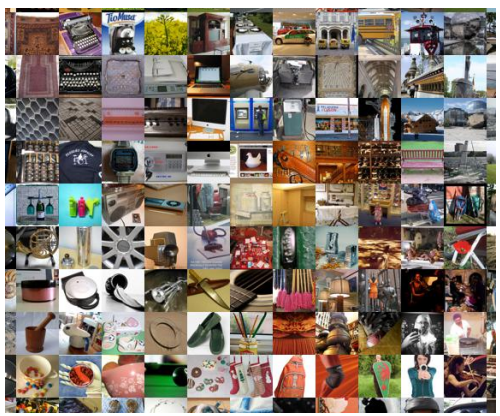
- **https://keras.io/api/applications/** – classic interface for pretrained CNNs (VGG16, MobileNet, ResNet, Xception, EfficientNet)
- **https://keras.io/keras_hub/** – unified access to hundreds of pretrained models (vision, text, multimodal, generative, etc.)
- Model weights can be randomly initialized or pretrained (typically on **ImageNet**)

pretrained_model.ipynb

- A practical 18NES1 example of using a pretrained CNN (Keras Applications)

ImageNet Dataset

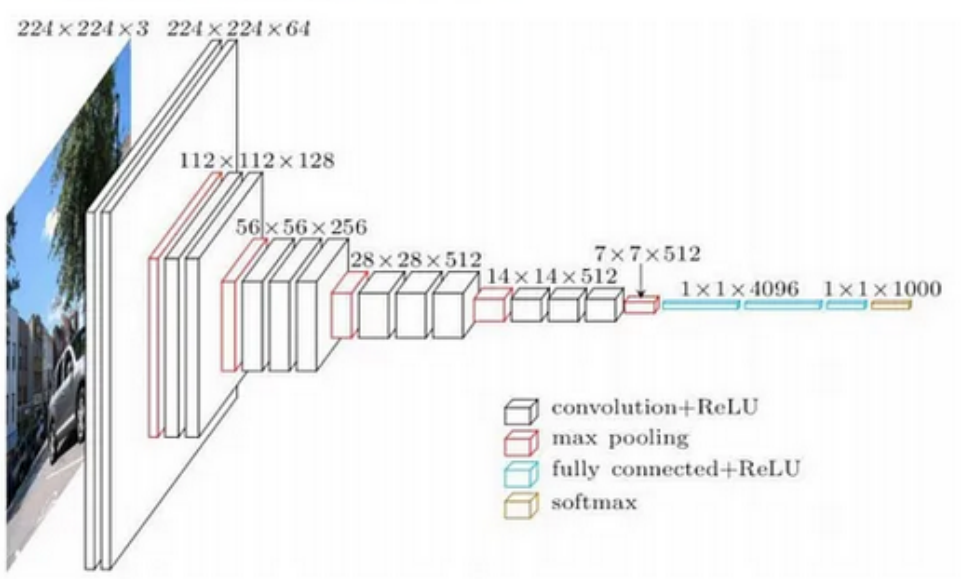
- Contains over **16 million color images** across **20,000 categories**
- Created as part of the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC, 2010–2017)
- This challenge triggered the breakthrough of **CNNs** in image recognition
- ImageNet became the standard benchmark dataset for model comparison (replacing MNIST)
- Most pretrained models use the **ImageNet-1K subset**: about **1.2 million images** from **1,000 categories**



Example of a Pretrained Model: VGGNet

pretrained_model.ipynb

- Karen Simonyan and Andrew Zisserman, 2014 – a family of models (e.g., VGG16, VGG19)
- Classic pyramidal architecture, relatively shallow (16 or 19 layers)



Source: https://medium.com/nerd-for-tech/vgg-16-easiest-explanation-12453b599526

Using a Pretrained Model

- Input images need to be resized to the expected dimensions and typically converted to RGB

- Required input size depends on the specific model (but is usually quite small, around 200×200)
- Images are typically rescaled and optionally cropped

From Pretrained Models to Transfer Learning

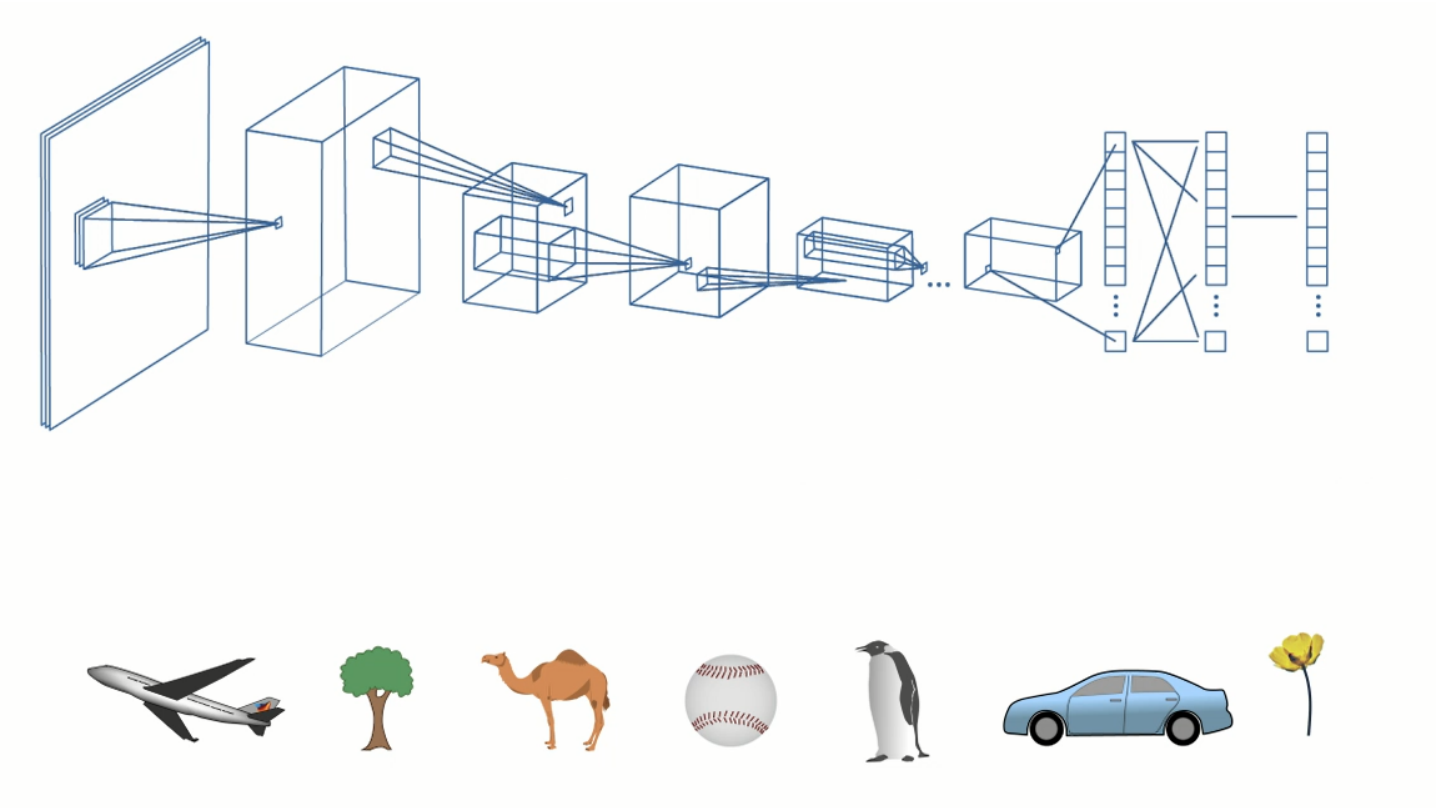
Using a pretrained model is a great starting point — but it usually won't be that simple.

- Although ImageNet-1K contains 1,000 classes, the classification accuracy on your custom dataset may still be low.
- See our example (18NES1): **pretrained_model.ipynb**

So how exactly do we adapt the model?

From a Pretrained Model to Transfer Learning

- Our pretrained model performs classification into 1,000 classes:

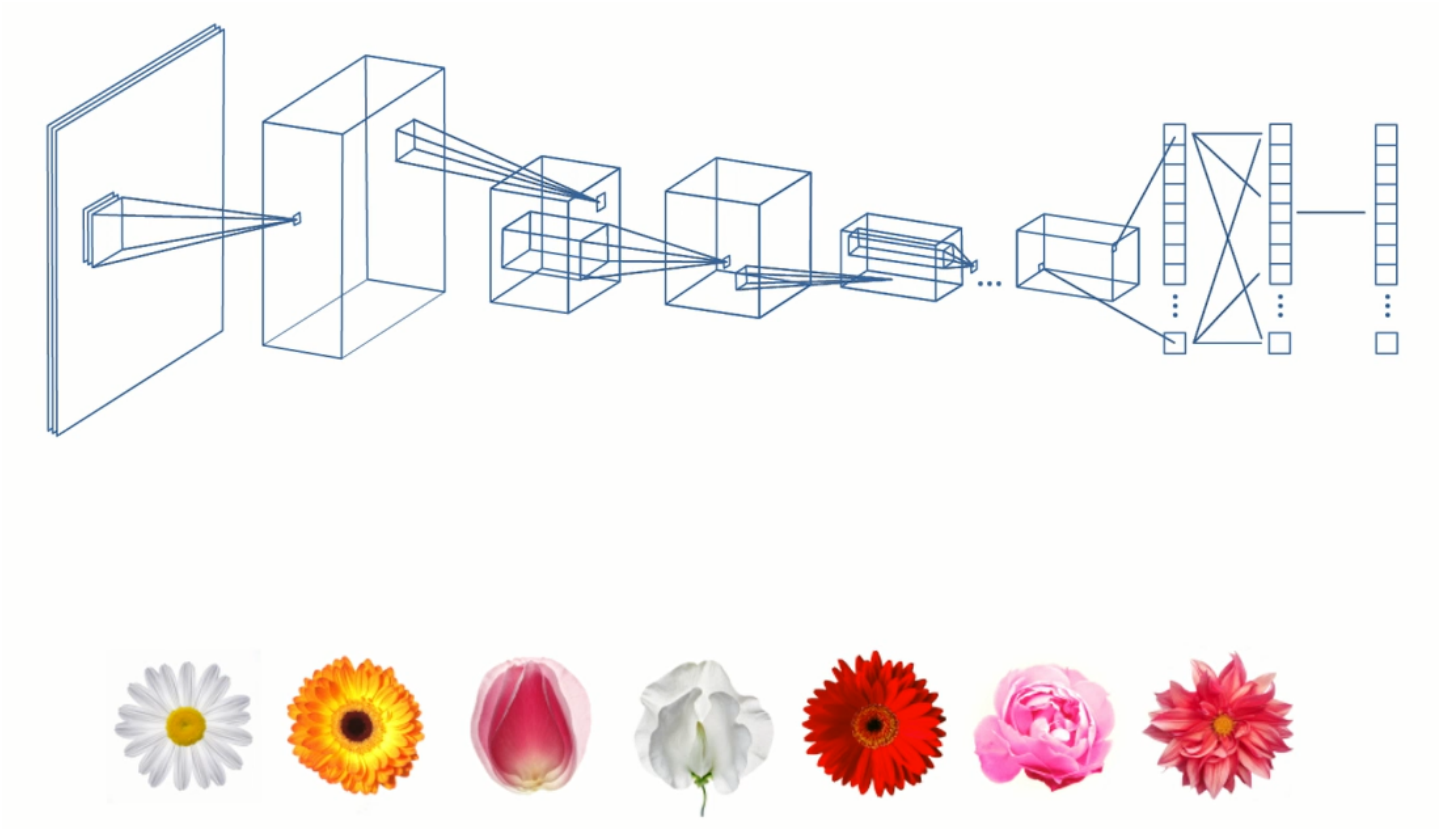


Source:

https://matlabacademy.mathworks.com/details/deep-learning-onramp/deeplearning

From a Pretrained Model to Transfer Learning

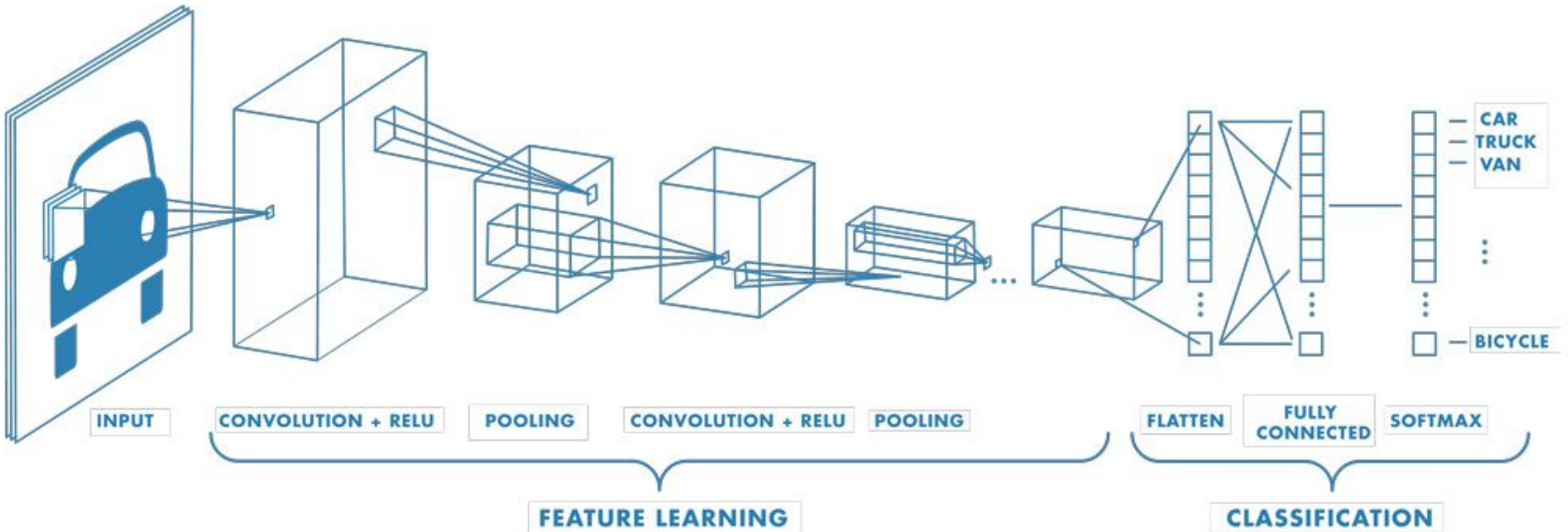
- But we need to classify into a different set of classes:



Source:

https://matlabacademy.mathworks.com/details/deep-learning-onramp/deeplearning

Recap: CNN Architecture



Components of a Convolutional Neural Network

- Convolutional base – extracts hierarchical features
- Flattening layer – converts data to a numeric vector
- Fully connected neural network for classification – **classification head**

Source:

https://matlabacademy.mathworks.com/details/deep-learning-onramp/deeplearning

From Pretrained Model to Transfer Learning

How exactly? First idea:

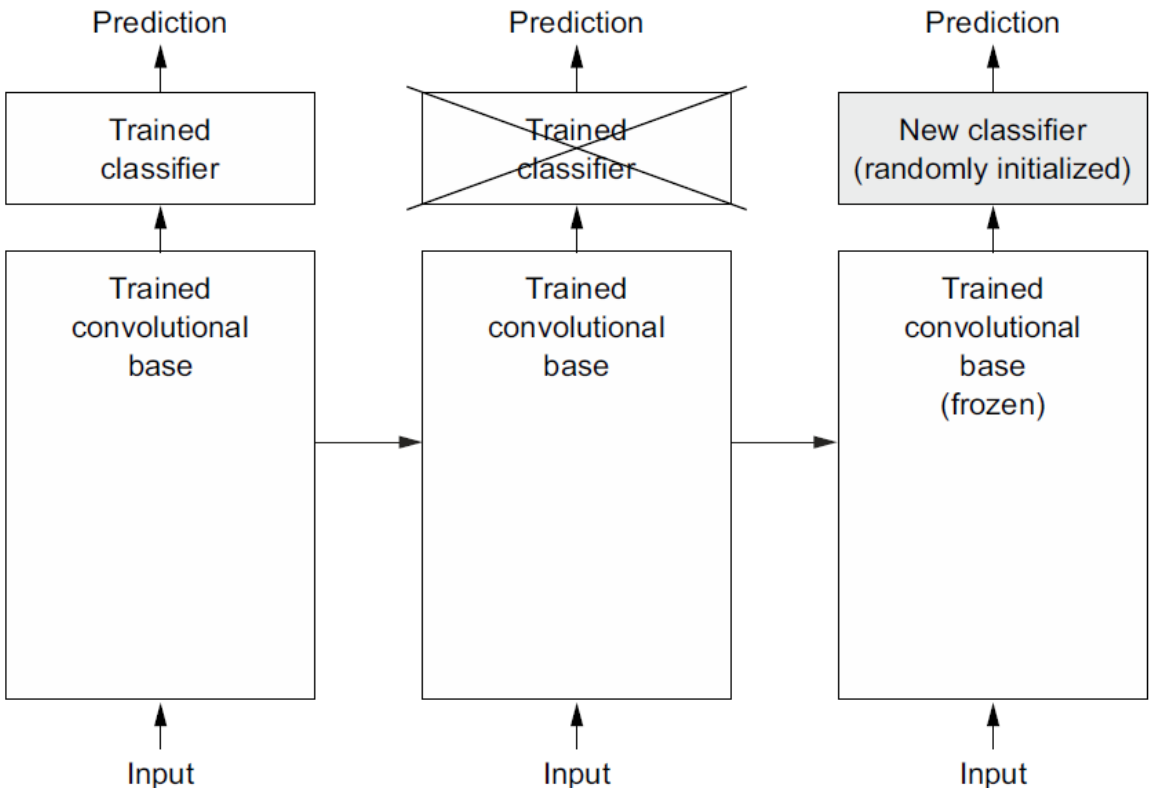
- Take a network pretrained on ImageNet
- Remove its classification head
- Use it to extract features from the new data $\rightarrow$ create a new training set
- Build a new fully connected (MLP) neural network and train it on the extracted features

This approach is efficient, but often impractical:

- It's static – hard to apply built-in data augmentation tools

Transfer Learning

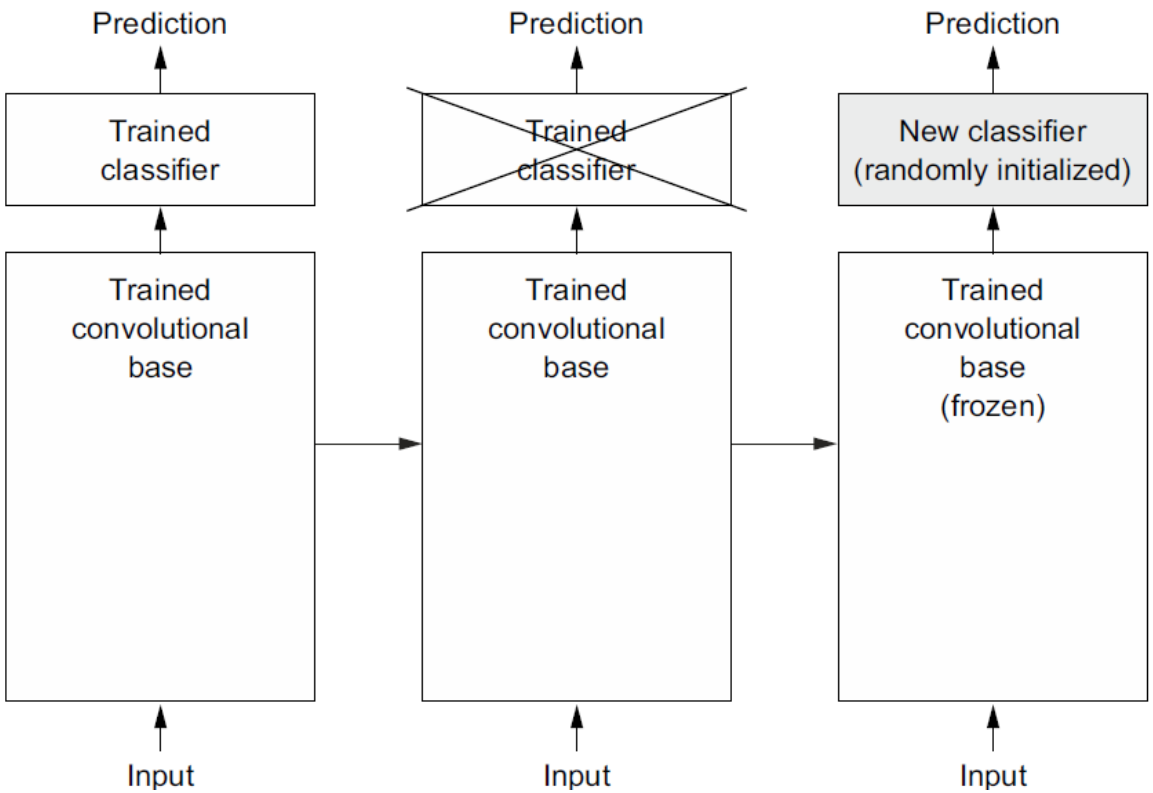
- Take a network pretrained on ImageNet
- Replace its classification head (or just its top part) with a new one (randomly initialized)



F. Chollet: Deep Learning with Python, Fig. 8.2

Transfer Learning

- Train the new classification head on your new data (keep the earlier layers frozen)



F. Chollet: Deep Learning with Python, Fig. 8.2

Transfer Learning

Practical Notes

- Typically, a smaller learning rate is chosen compared to training a network from scratch
- Regularization is recommended (dropout, data augmentation)

Example: Cats and Dogs – Continued

- **cats_dogs_transfer.ipynb**

Three Variants

1. Using a pretrained convolutional base for efficient feature extraction
2. Full transfer learning (combined with data augmentation and dropout)
3. Additional fine-tuning

Example: Cats and Dogs – Continued

- **cats_dogs_transfer.ipynb**

Observations:

- **Feature extraction only:** Training is fast and achieves very high test accuracy (around 98%), since the **Cats vs Dogs** dataset is visually similar to classes in **ImageNet**. However, the model still tends to overfit early.
- **Full transfer learning:** Training is slower because the convolutional base is included in the model, but data augmentation and dropout help to **delay and reduce overfitting**.
- Compared to training from scratch, both approaches provide a **significant boost in performance and generalization**.

Transfer Learning and Fine-Tuning

- A model trained via transfer learning can be further improved with fine-tuning:
  1. First, apply transfer learning (freezing weights outside the new classifier head)
  2. Then, unfreeze the deepest part of the convolutional base and continue training along with the new layers

Practical Notes

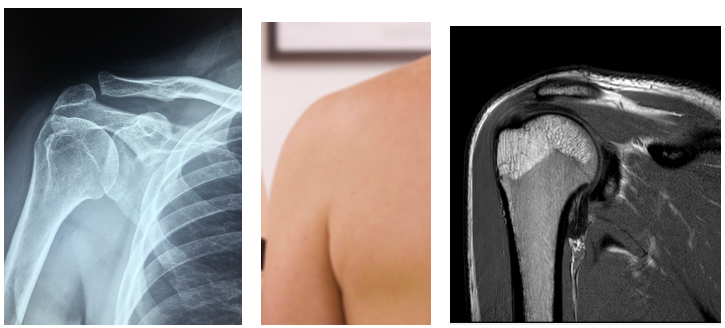
- Proceed carefully: start with a very small learning rate — ideally smaller than the final learning rate used in the pretrained model
- Regularization can also be applied

cats_dogs_transfer.ipynb

Limitations of Transfer Learning

- Transfer learning is suitable when training a model on data similar to the source domain (general model $\rightarrow$ more specific model)
- It cannot be applied if input layers need to be changed:
  - Different number of channels (e.g., X-ray images)
  - Different level of abstraction
  - Completely different types of data
- Sometimes, training a model from scratch is unavoidable

Example: Shoulder imaging extended with X-ray and MRI scans



Source: `https://radiopaedia.org/cases/normal-shoulder-mri`

CNN architectures and design patterns

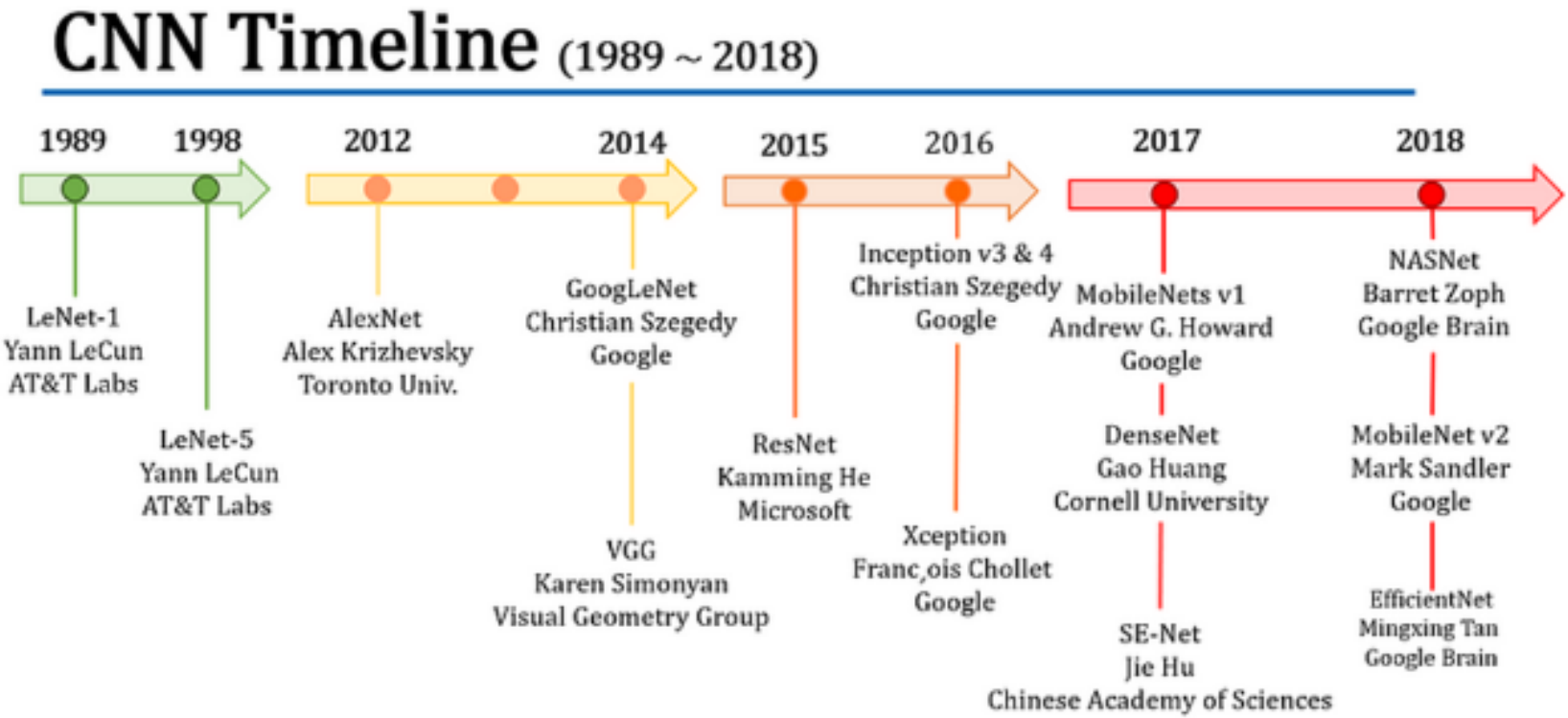
From Basic CNNs to Modern Architectures

- So far, we have built simple (bipyramidal) CNNs and used pretrained ones (e.g., VGG16, Xception).
- The enhanced CNN architectures share common **design patterns** — repeated building blocks that make them efficient and powerful.
- Understanding these patterns helps you:
  - design better models for your own tasks,
  - fine-tune pretrained architectures more effectively,
  - and understand why modern CNNs work so well.

In this part:

- Overview of key CNN architectures
- Typical architectural patterns (residuals, stacking, bottlenecks, separable convolutions)
- Construction of a small custom model inspired by modern architectures

Evolution of CNN Architectures



Source of the image:

https://miro.medium.com/v2/resize:fit:640/format:webp/0*1SDlsJ7snNv_deec.png

Evolution of CNN Architectures

Key Milestones

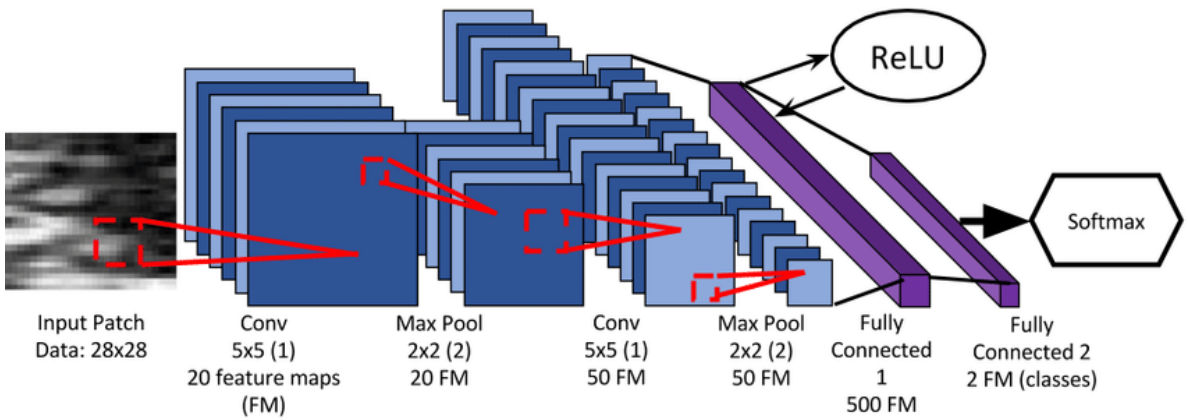
- **Early CNNs (LeNet, 1998)** — simple, shallow models
- **Deep CNNs (AlexNet, VGG, 2012–2014)** — bipyramidal architecture, increased depth, use of GPUs, ReLU activation, data augmentation.
- **Efficient and scalable designs (GoogLeNet, ResNet, 2014–2016)** — inception modules and skip connections enable very deep networks.
- **Lightweight and optimized models (Xception, MobileNet, EfficientNet, 2017-2019)** — separable convolutions, architectural scaling, and automated design (NAS).
- **Hybrid and Transformer-based models (ViT, ConvNeXt, 2020+)** — combine convolution and self-attention for superior global context understanding.

CNN Evolution - Early CNNs

- The earliest architectures were wide and shallow, often with a deeper fully connected head, following a basic design scheme

LeNet-5

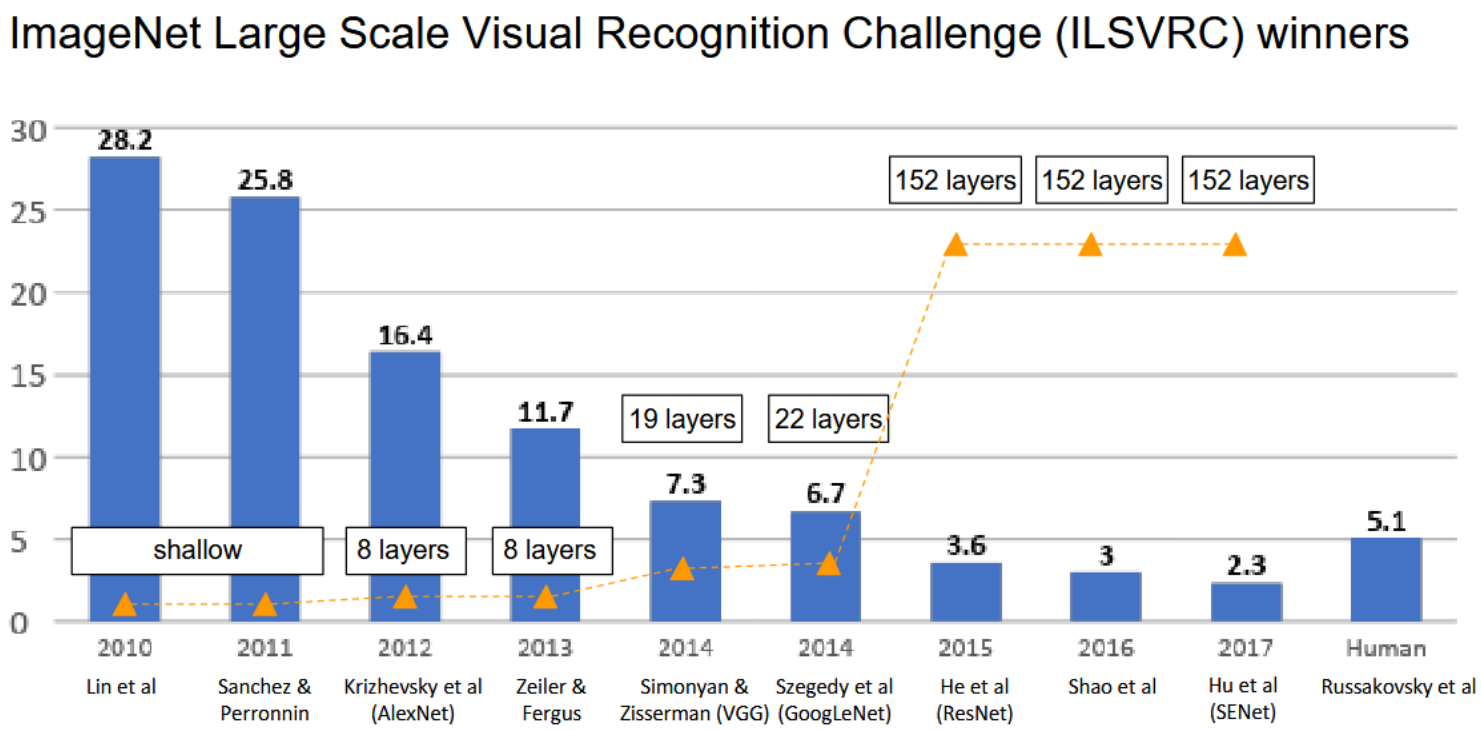
- One of the original architectures (Yann LeCun, 1998), relatively simple, trained on MNIST



Source of the image: M. H. Yap et al., "Automated Breast Ultrasound Lesions Detection Using Convolutional Neural Networks," IEEE Journal of Biomedical and Health Informatics, vol. 22, 2018.

CNN Evolution - From Shallow to Deep CNNs

- The ILSVRC competition: a key contest using the ImageNet dataset

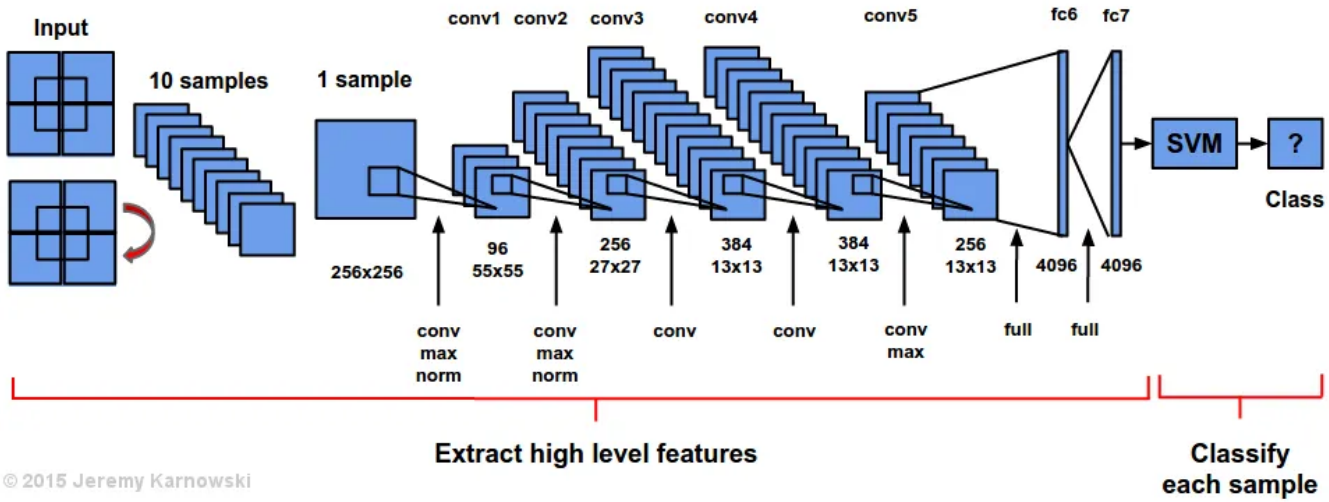


Source of the image: https://cs231n.stanford.edu/slides/2024/lecture_6_part_1.pdf

CNN Evolution - From Shallow to Deep CNNs

AlexNet

- The first CNN to win the ILSVRC competition (2012) on ImageNet — achieving 84.7% top-5 accuracy
- Significantly more complex: 8 layers, 61 million parameters, training took 5–6 days on 2 GPUs
- New elements: ReLU activation, data augmentation, dropout



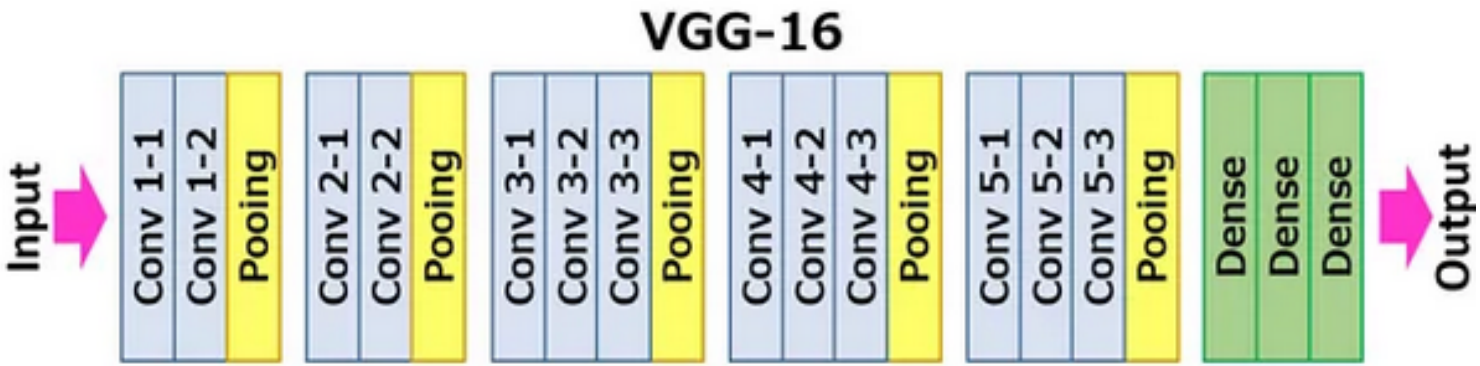
Source of the image:

https://medium.com/@jkarnows/alexnet-visualization-35577e5dcd1a

CNN Evolution - From Shallow to Deep CNNs

VGGNet

- Participant of ILSVRC 2014, a family of models (e.g., VGG16 – 16 layers, VGG19 – 19 layers) achieving 92.7% top-5 accuracy
- Pyramid-shaped sequential architecture, small convolutional filters (3×3)



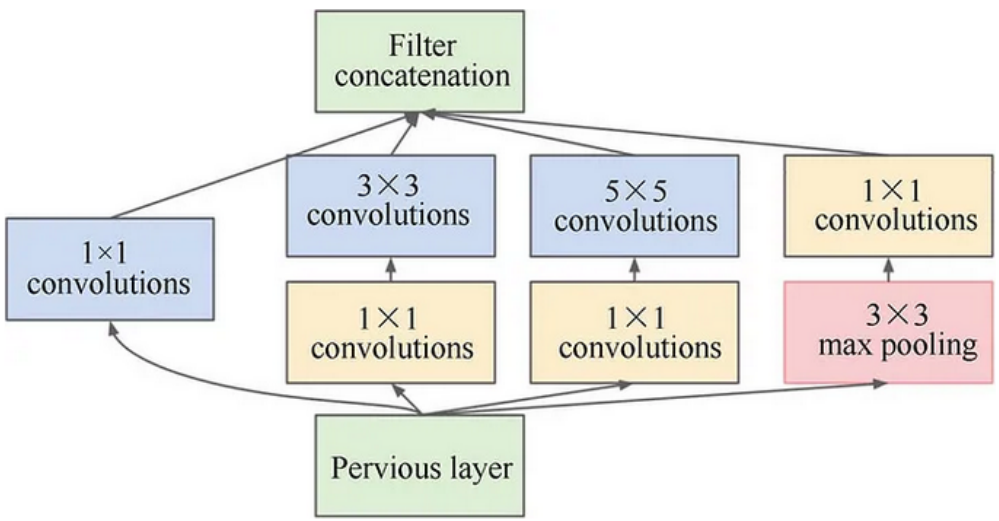
Source of the image:

https://medium.com/nerd-for-tech/vgg-16-easiest-explanation-12453b599526

CNN Evolution - Efficient and scalable designs

GoogLeNet (Inception v1), 2014

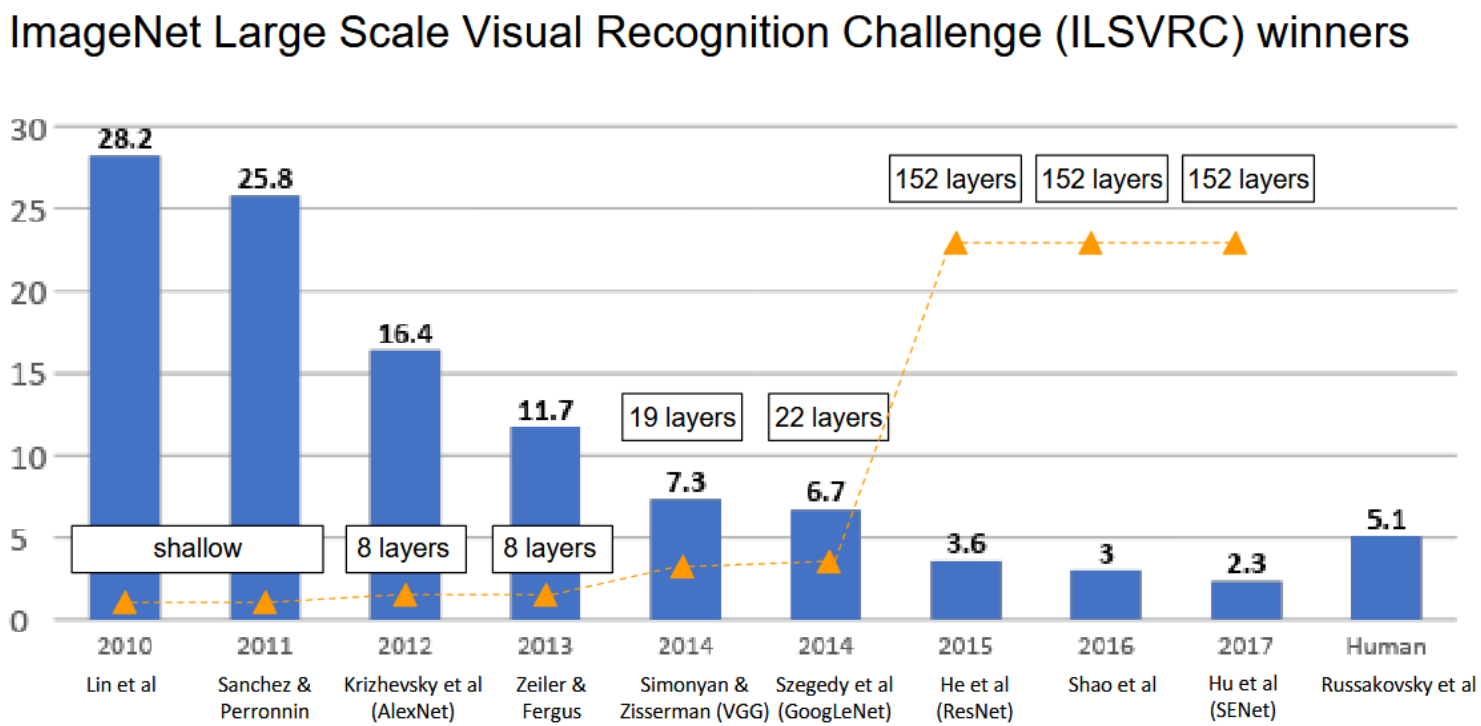
- Winner of ILSVRC 2014, 22 layers — 93.33% top-5 accuracy
- Revolutionary element: **Inception blocks** — feature extraction at multiple scales, parallelism, improved efficiency



The illustration is from the original paper[1]

CNN Evolution - Going deeper

- After 2014, ILSVRC competition: significantly deeper and narrower architectures (reducing redundancy: depthwise separable convolutions), modular design

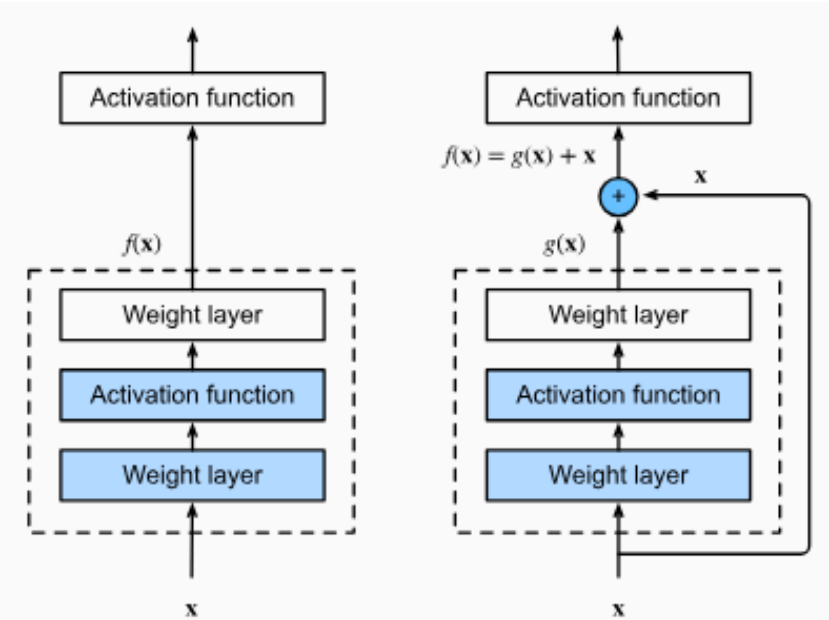


Source: https://cs231n.stanford.edu/slides/2024/lecture_6_part_1.pdf

CNN Evolution - Efficient and scalable designs

ResNet, 2015

- Winner of ILSVRC 2015, 152 layers — 96.4% top-5 accuracy
- Revolutionary element: **skip connections** — solve the vanishing gradient problem, enable extremely deep and narrow architecture
- Residual blocks

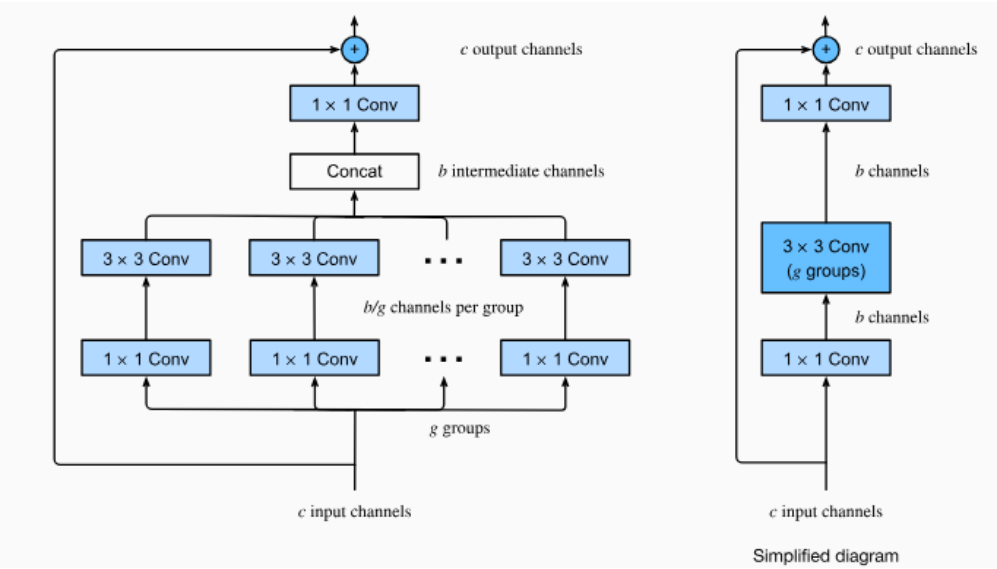


Source: A. Zhang et al.: Dive into Deep Learning, Figure 8.6.2

CNN Evolution - Efficient and scalable designs

ResNet, 2015

- Another new element: **Bottleneck residual blocks** — reduce number of channels to reduce computation
- Structure: 1×1 convolution (reduction) → 3×3 convolution → 1×1 convolution (expansion).
- This design enables networks with hundreds of layers.

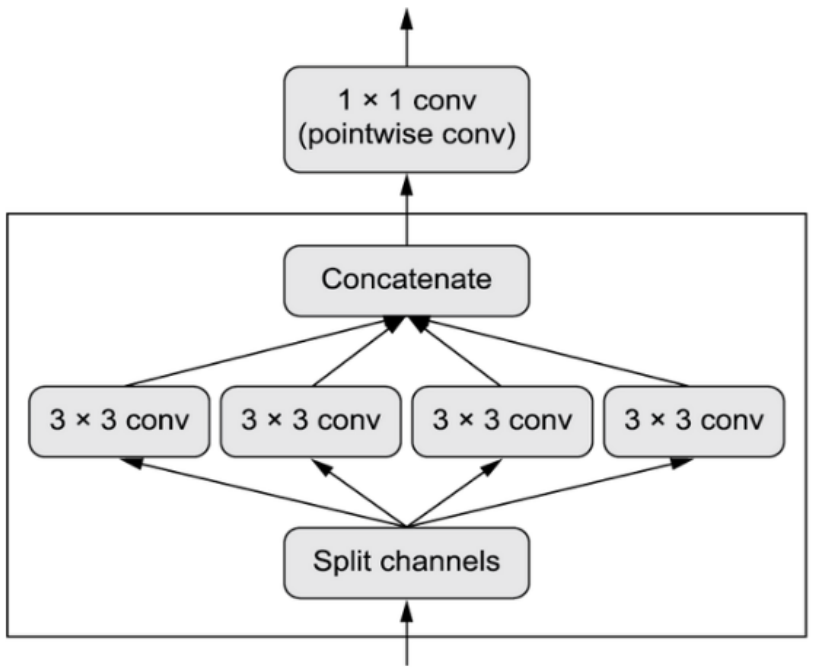


Source: A. Zhang et al.: Dive into Deep Learning, Figure 8.6.5

CNN Evolution - Efficient and scalable designs

Depthwise Separable Convolutions (Xception, 2017)

- Another breakthrough to improve efficiency and slim down layers
- Spatial convolutions applied independently per channel, followed by channel mixing via pointwise convolution



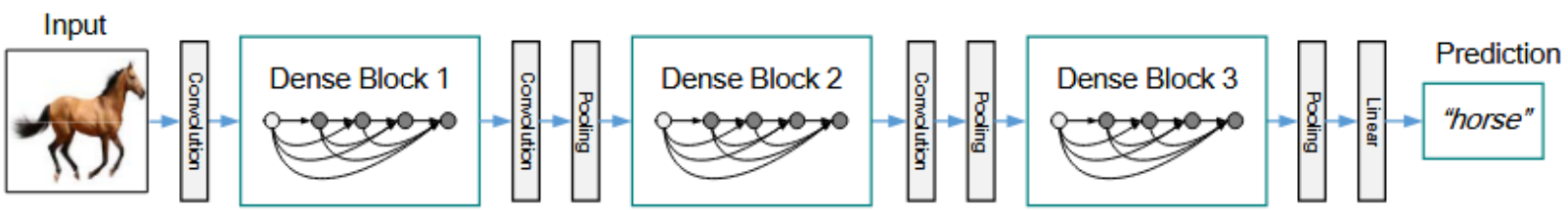
Source: F. Chollet, "Deep Learning with Python," Figure 9.10

CNN Evolution – Derived and Scalable Designs

Architectures building upon earlier CNN ideas:

- **Inception v2, v3 (2015–2016)** – deeper, factorized convolutions for efficiency.
- **ResNet variants (2015+)** – deeper residual blocks, improved skip connections and bottlenecks.
- **DenseNet (2016)** – each layer receives inputs from all previous layers.

Goal: improve training stability and information flow in very deep networks.



Source: Huang G. et al: Densely Connected Convolutional Networks, 2016

CNN Evolution – Efficient Architectures and NAS

Lightweight CNNs:

- **MobileNet (2017)**, **SqueezeNet (2016)** – small, low-latency models for mobile and embedded devices.
- **EfficientNet (2019)** – compound scaling of depth, width, and resolution for optimal performance–efficiency tradeoff.

Neural Architecture Search (NAS):

- **NASNet (2017)**, **AmoebaNet (2018)** – model architectures discovered automatically using reinforcement learning or evolutionary algorithms.
- **Goal:** find optimal architectures for given resource constraints.

In Keras:

- https://keras.io/api/applications/
- https://keras.io/keras_hub/

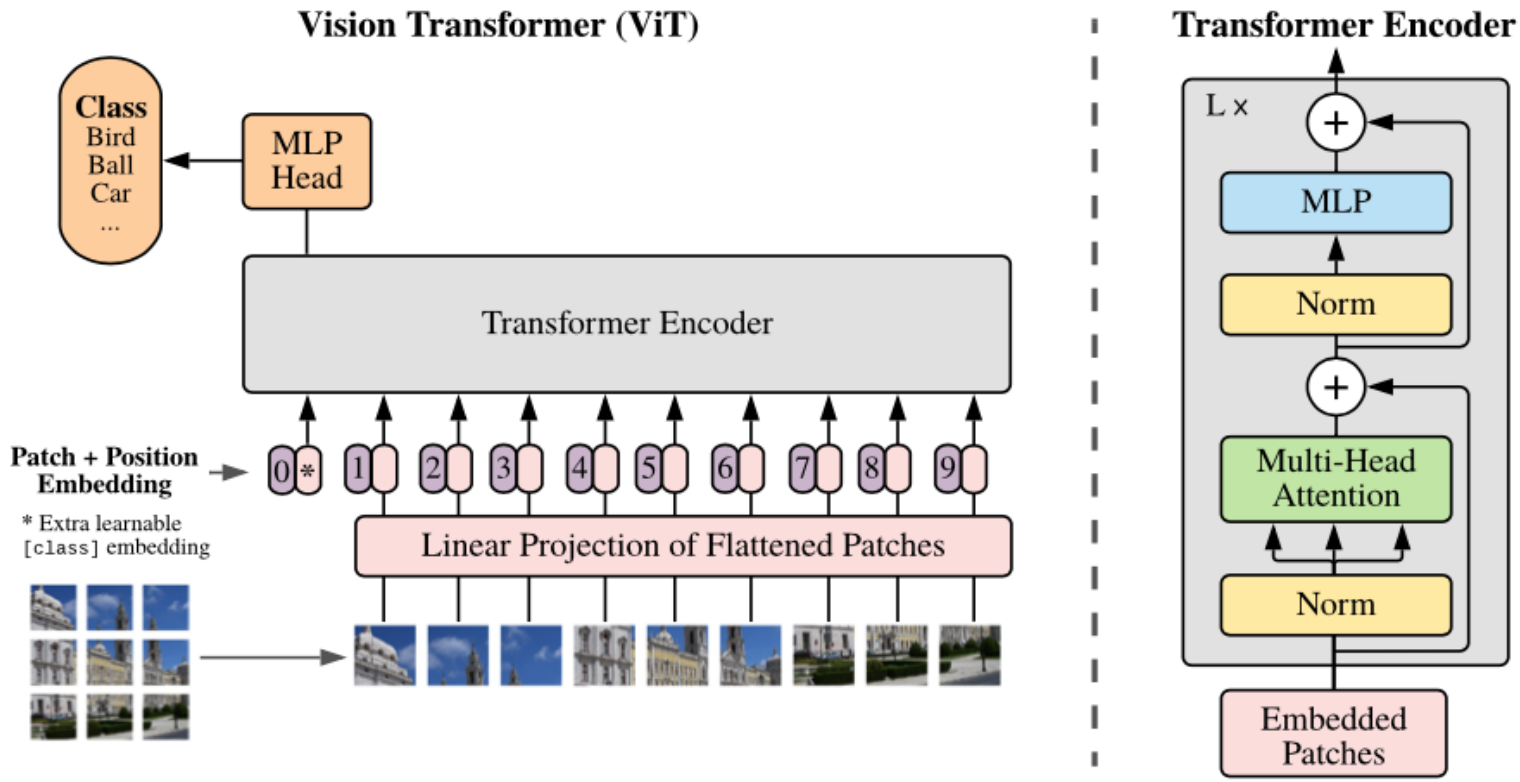
CNN Evolution – Hybrid and Transformer-Based Models

Recent trend (2020+): combining convolution with attention mechanisms.

- **Vision Transformer (ViT, 2020)** – applies the transformer encoder directly to image patches.
- **ConvNeXt (2022)** – pure CNN redesigned with transformer-inspired components (LayerNorm, large kernels).
- **Swin Transformer (2021)** – hierarchical attention windows for scalable vision tasks.
- **Hybrid models** – combine CNN feature extractors with transformer layers for global context.

Goal: combine local spatial understanding (CNNs) with global contextual reasoning (Transformers).

CNN Evolution – Hybrid and Transformer-Based Models



Source:

https://github.com/google-research/vision_transformer/blob/main/vit_figure.png

Evolution of CNN Architectures: Summary

- **Early CNNs** (e.g., LeNet-5, 1998): shallow and wide, simple fully connected parts
- **AlexNet** (2012): deeper networks, use of GPUs, ReLU activation, data augmentation, dropout
- **VGGNet** (2014): deeper and structured (pyramidal) networks, small convolutional filters (3×3)
- **GoogLeNet / Inception** (2014): multi-scale feature extraction, parallelism, improved efficiency
- **ResNet** (2015): extremely deep networks enabled by skip connections (residual learning)
- **Xception, MobileNet, EfficientNet** (2017–2019): model slimming via depthwise separable convolutions and architecture optimization
- **NASNet (2017) and beyond**: automated design of optimal architectures (neural architecture search)
- **ViT, ConvNeXt, Swin (2020+)** : attention and hybrid CNN–Transformer designs $\rightarrow$ *global context, new state-of-the-art*

Practical example: CNN Architecture Patterns

Common design patterns in modern CNNs:

- **Residual connections** — add shortcut links to ease training of deep networks (ResNet family).
- **Bottleneck blocks** — compress and expand feature maps to reduce computation (ResNet, Inception).
- **Depthwise separable convolutions** — factorize convolutions to make models lighter (Xception, MobileNet).
- **Inception-style modules** — parallel multi-scale feature extraction.
- **Batch Normalization and Activation ordering** — stabilize and speed up training.

cnn_architecture_patterns.ipynb

- includes residual blocks, bottlenecks, separable convolutions, and a small Xception-like model.

more details: next week

5th Graded Homework: Transfer Learning and Advanced CNN Architectures

Goal: Compare different CNN approaches on a subset of the **CIFAR-10** dataset (reuse the same dataset as in the 4th homework).

Dataset:

- Reuse the dataset from the previous assignment
- Use a smaller subset of the data (e.g., 2000 training, 1000 validation, 1000 test samples).
- You can select 3–5 classes (e.g., *airplane*, *car*, *bird*, *cat*, *dog*) or use all 10 classes if resources allow.
- Each image is 32×32 px, RGB.
- The dataset can be easily loaded using `tf.keras.datasets.cifar10`.
- Example preprocessing of the dataset: **CIFAR10_load_data.ipynb**

5th Graded Homework: Suggested Workflow

Steps:

- **Model 1:** Simple CNN trained from scratch (reuse your best model from previous homework).
- **Model 2:** Xception-like or residual-block CNN (see the lecture notebook).
- **Model 3:** Transfer learning using a pretrained base (e.g., Xception, MobileNet, ResNet) .

You can start from the example notebooks shown in the lab:
CIFAR10_load_data.ipynb cats_dogs_from_scratch.ipynb cats_dogs_transfer.ipynb cats_dogs_small_Xception.ipynb cnn_architecture_patterns.ipynb

5th Graded Homework: Requirements

Pipeline: load → preprocess → build → train → evaluate

Include in your notebook:

- Training and validation curves (loss + accuracy).
- Final test accuracy and confusion matrix.
- Comparison of all three models (bipyramidal CNN, custom CNN, pretrained CNN).
- Short summary of your observations:
  - Which model generalizes best and why?
  - How does training time differ?

Optional:

- Visualize misclassified images.
- Try more model variants.

5th Graded Homework: Submission

Submission

- Submit the notebook by **Nov 17, 2025**.
- **Consultation required by Nov 21, 2025** to receive points (short discussion after lab or individually).
- **Points: 1–3**
  - +1 point if you did not submit the previous homework (for the baseline CNN).
  - +1 point for comparing the three main models (baseline CNN, custom CNN, pretrained CNN).
  - +1 point for including and discussing **additional variants**, e.g., fine-tuning, using multiple pretrained bases, or comparing both **Xception- and ResNet-like** custom architectures.