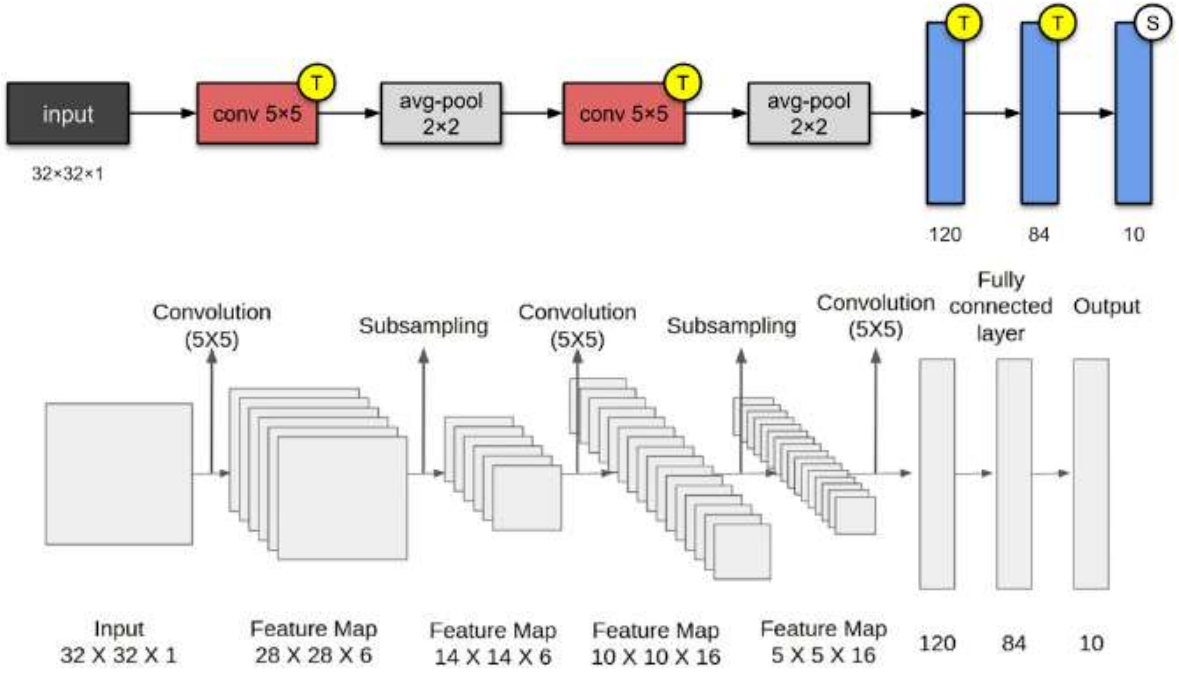
Different CNN Architectures and Transfer Learning

Sourav Karmakar

souravkarmakar29@gmail.com

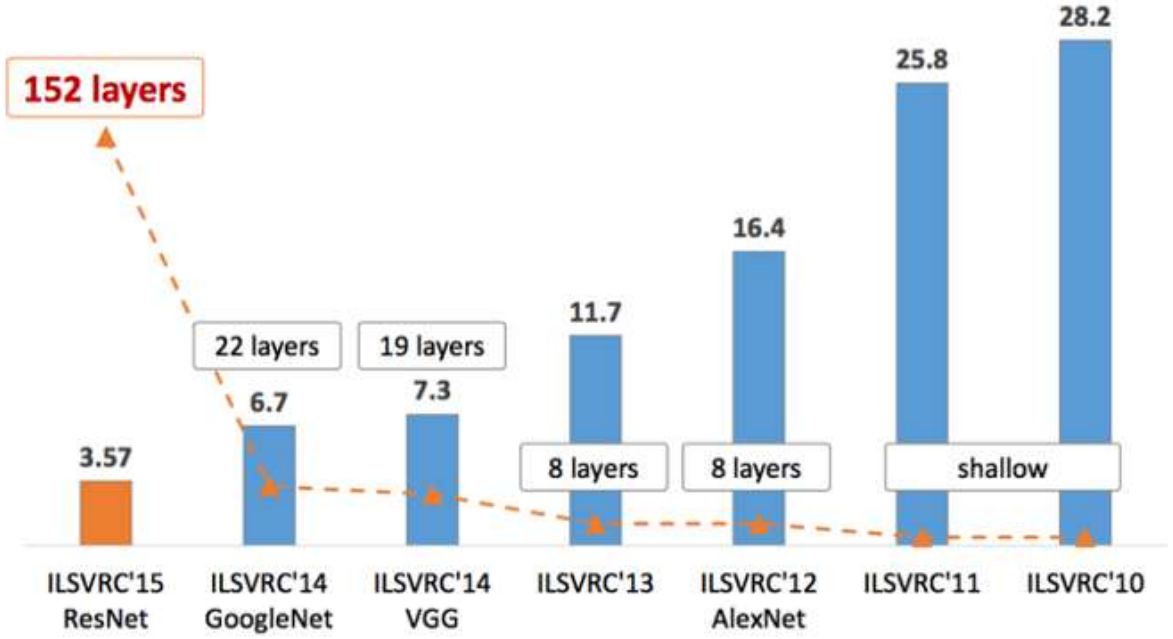
LeNet-5 (1998)



- LeNet-5 is one of the simplest architectures. It has 2 convolutional and 3 fully-connected layers (hence "5" — it is very common for the names of neural networks to be derived from the number of *convolutional* and *fully connected* layers that they have). The average-pooling layer as we know it now was called a *sub-sampling layer* and it had trainable weights (which isn't the current practice of designing CNNs nowadays).
- Very efficient in recognizing handwritten digits.

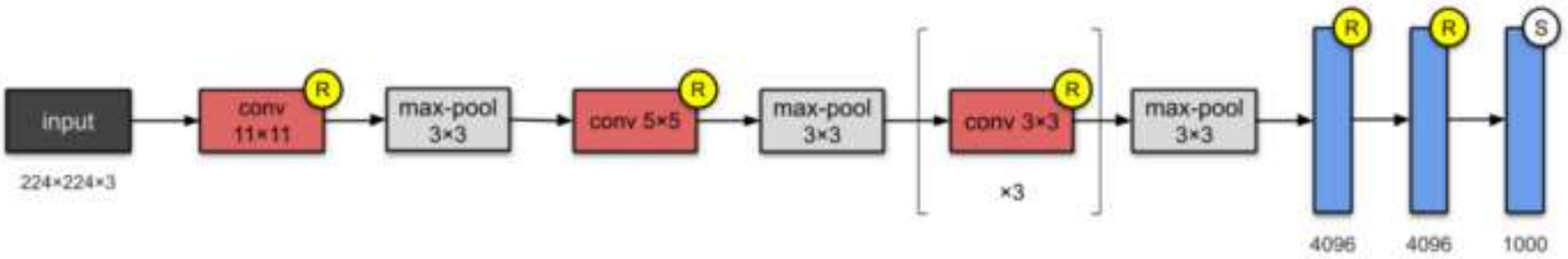
IMAGENET Challenge

- ImageNet Large Scale Visual Recognition Challenge (ILSVRC) was an annual computer vision competition developed upon a subset of a publicly available computer vision dataset called **ImageNet**. As such, the tasks and even the challenge itself is often referred to as the **ImageNet Competition**.
- The ImageNet competitions were hosted from 2010 to 2016 and very interesting and useful CNN architectures were developed by industry and academia for this competition. Those architectures are still widely used.



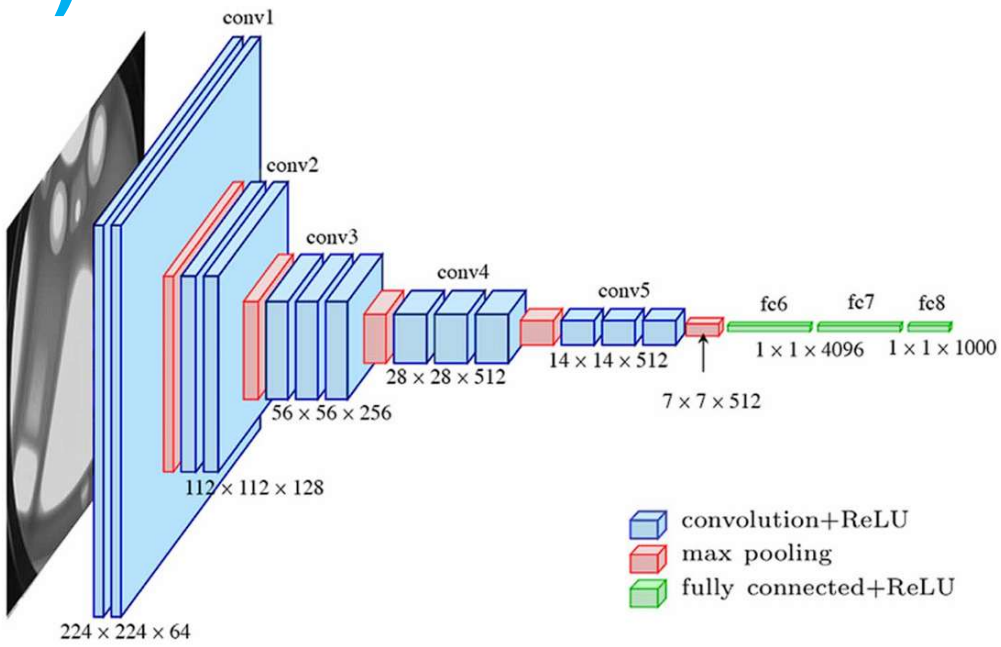
The numbers on the bar denotes top-5 error rates in ILSVRC

AlexNet (2012)



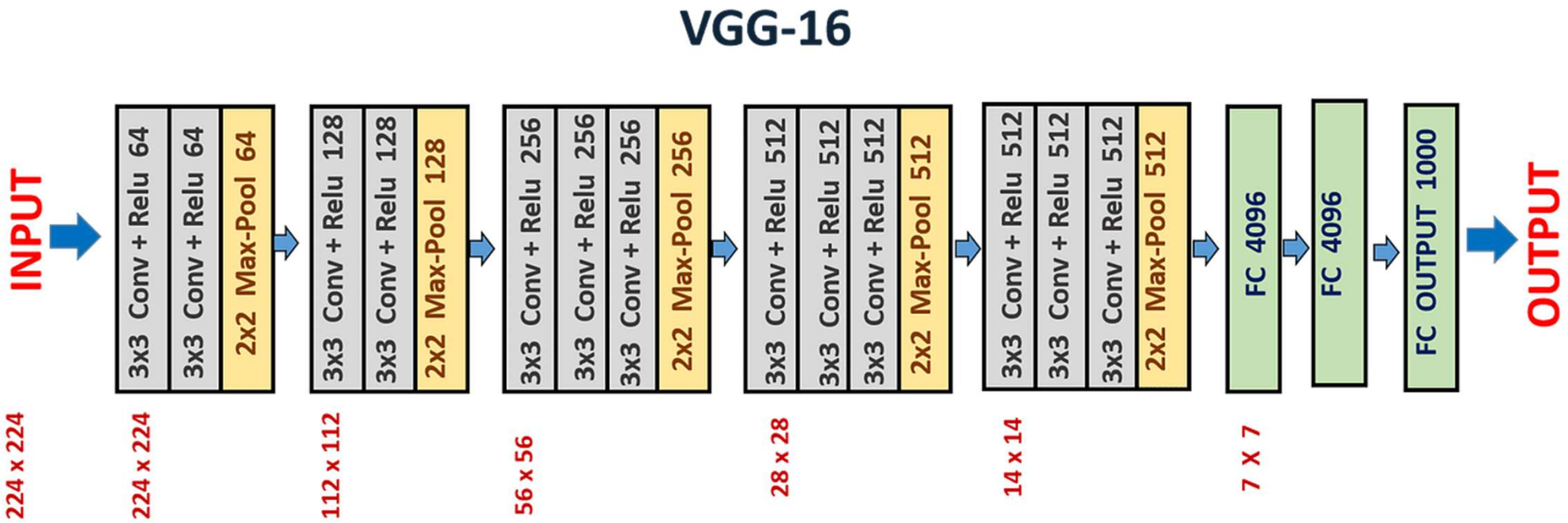
- With **60M parameters**, AlexNet has 8 layers — 5 convolutional and 3 fully connected. AlexNet just stacked a few more layers onto LeNet-5. At the point of publication, the authors pointed out that their architecture was "one of the largest convolutional neural networks to date on the subsets of ImageNet."
- They were the first to implement Rectified Linear Units (ReLUs) as activation functions.
- They also implemented Dropout for handling overfitting in large Neural Networks.
- Winner of ILSVRC – 2012.

VGG-16 (2014)



- This was developed at Visual Geometry Group (VGG) of Oxford University. VGG-16 has 13 convolutional and 3 fully-connected layers, carrying with them the ReLU tradition from AlexNet.
- It consists of 138M parameters and takes about 500MB storage Space.
- It is used as backbone architecture for feature extraction for various Computer Vision related tasks.
- Runners up for ILSVRC-2014

VGG-16 (2014)



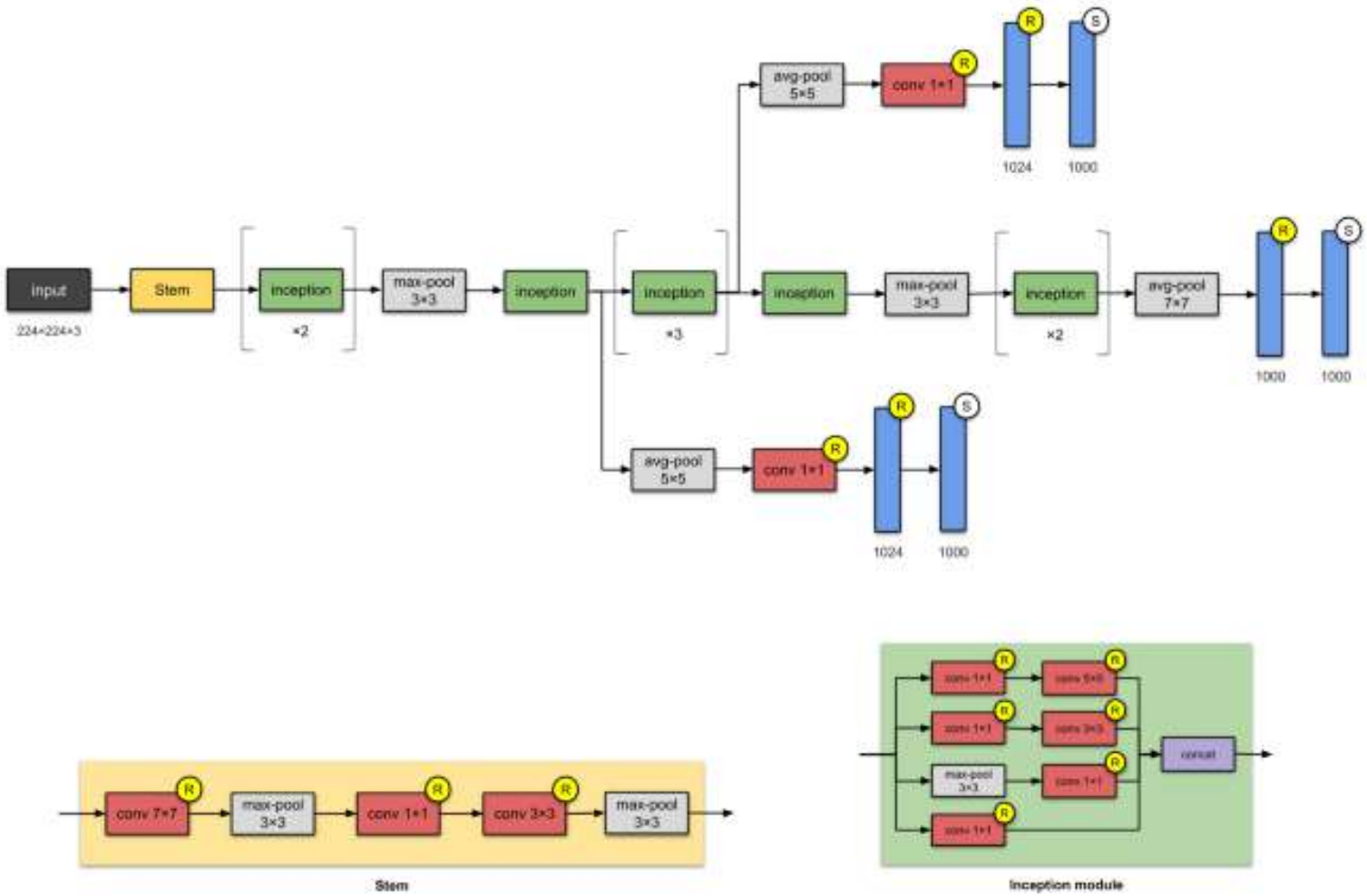
- **Advantages:**

Very simple modular architecture. 3 × 3 convolutions with stride = 1. “Same” padding and 2 × 2 max pooling

- **Disadvantages:**

Very large number of parameters. Making it very sluggish to train from scratch.

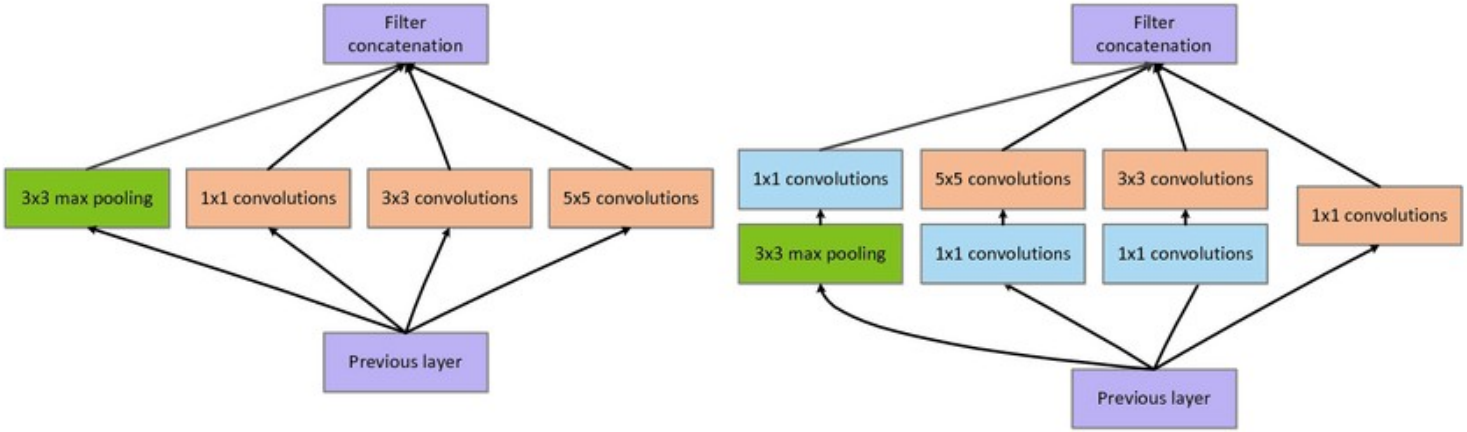
GoogLeNet or Inception-v1 (2014)



GoogLeNet or Inception-v1 (2014)

- This 22-layer architecture with **6.8M** parameters is called the Inception-v1. Here, the **Network In Network** approach is heavily used, as mentioned in the paper. The Network In Network is implemented via ***Inception modules***. Each module presents 3 ideas:

  1. Having **parallel towers** of convolutions with different filters, followed by concatenation, captures different features at 1×1, 3×3 and 5×5, thereby 'concatenating' them.
  2. 1×1 convolutions are used for dimensionality reduction to remove computational bottlenecks.
  3. Due to the activation function from 1×1 convolution, its addition also adds nonlinearity.



(a) Inception module, initial form

(b) Inception module with dimension reductions

GoogLeNet or Inception-v1 (2014)

What is a 1 × 1 Convolution?

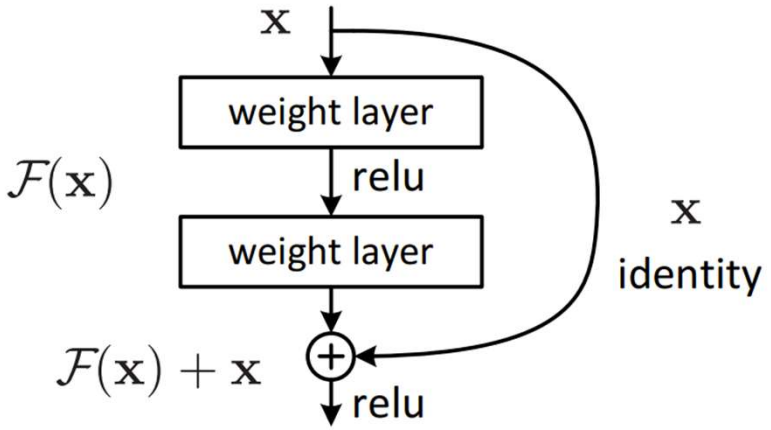
- Normally, convolutional filters are larger (e.g., 3×3, 5×5) and slide across the height & width of an image.
- A **1×1 filter** slides across the input feature map but only looks at **one pixel at a time (spatially)**.
- Even though it covers just **1 pixel spatially**, it spans **all the input channels**.
  - Example: if input feature map = 32 × 32 × 256, then a 1×1 conv filter has size 1 × 1 × 256.
  - If you use, say, 64 such filters, the output becomes 32 × 32 × 64.
- So, it doesn't reduce spatial size — instead, it changes the **depth (number of channels)**.

Benefits of 1 × 1 Convolution?

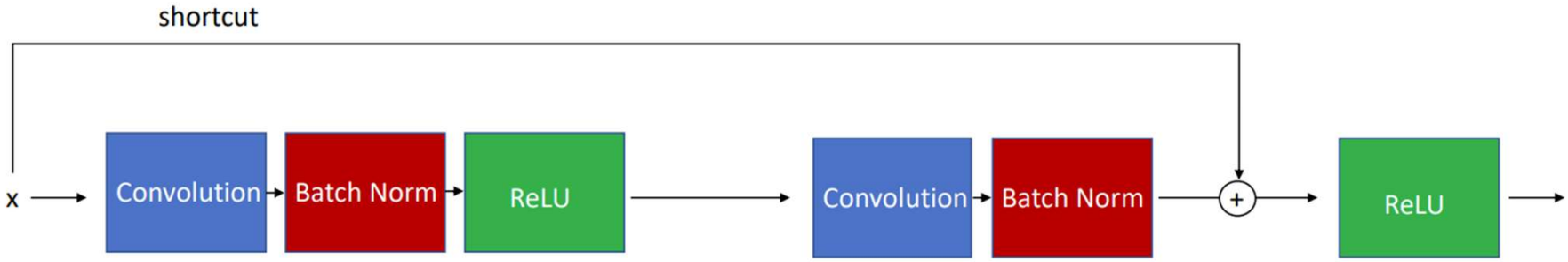
- **Parameter Efficiency:**
  - Suppose you apply a 5×5 filter on input with 256 channels and 32 such filters are used.
    - Total number of parameters for this operation: $5 \times 5 \times 256 \times 32 = 204{,}800$
  - But if you first use 1×1 conv to reduce the channels from 256 to 64 and then apply 5×5 convolution:
    - 1×1 conv parameters: $1 \times 1 \times 256 \times 64 = 16{,}384$
    - 5×5 conv parameters: $5 \times 5 \times 64 \times 32 = 51{,}200$
    - Total number of parameters: 16,284 + 51,200 = 67,484. Which is much less compared to direct 5×5 convolution
- By controlling dimensions, we avoid exploding parameter count, making deeper models feasible.
- Adds more non-linear transformations without affecting spatial resolution.

ResNet and Skip Connections (2016)

With their simple trick of allowing skip connections (the possibility to learn identity functions and skip layers that are not useful), **ResNets** allow us to implement very, very deep architectures.

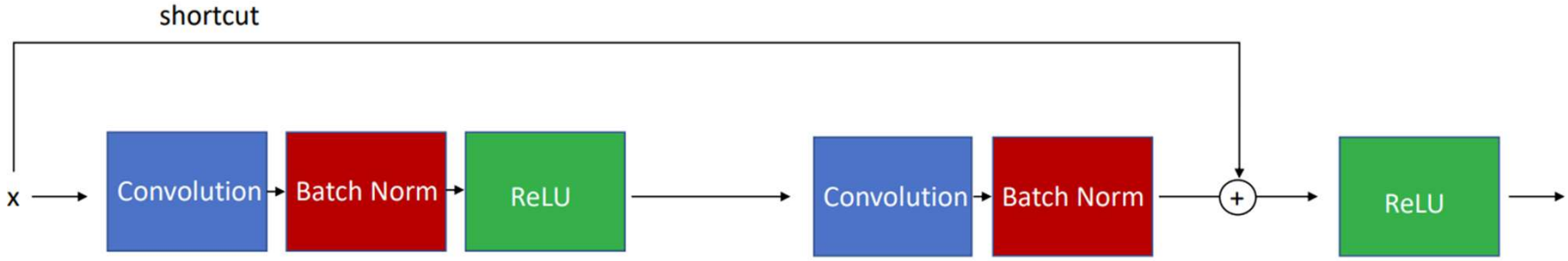


Residual Learning: a building block



In general: $a^{(l+2)} = \sigma\left(z^{(l+2)} + a^{(l)}\right)$

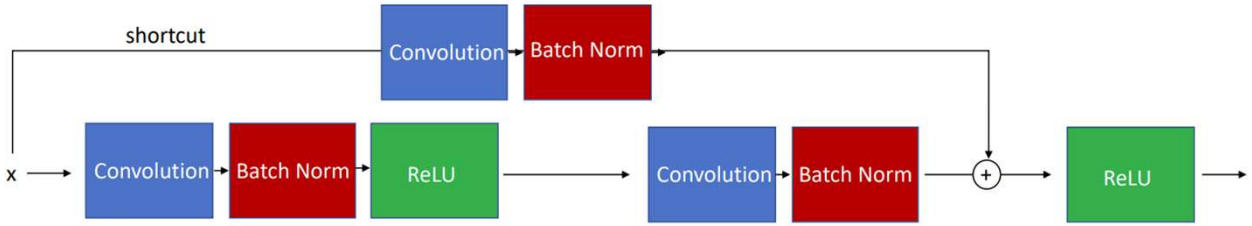
ResNet and Skip Connections (2016)



$$a^{(l+2)} = \sigma\left(z^{(l+2)} + a^{(l)}\right)$$
$$\Rightarrow a^{(l+2)} = \sigma\left(a^{(l+1)}\, W^{(l+2)} + b^{(l+2)} + a^{(l)}\right)$$

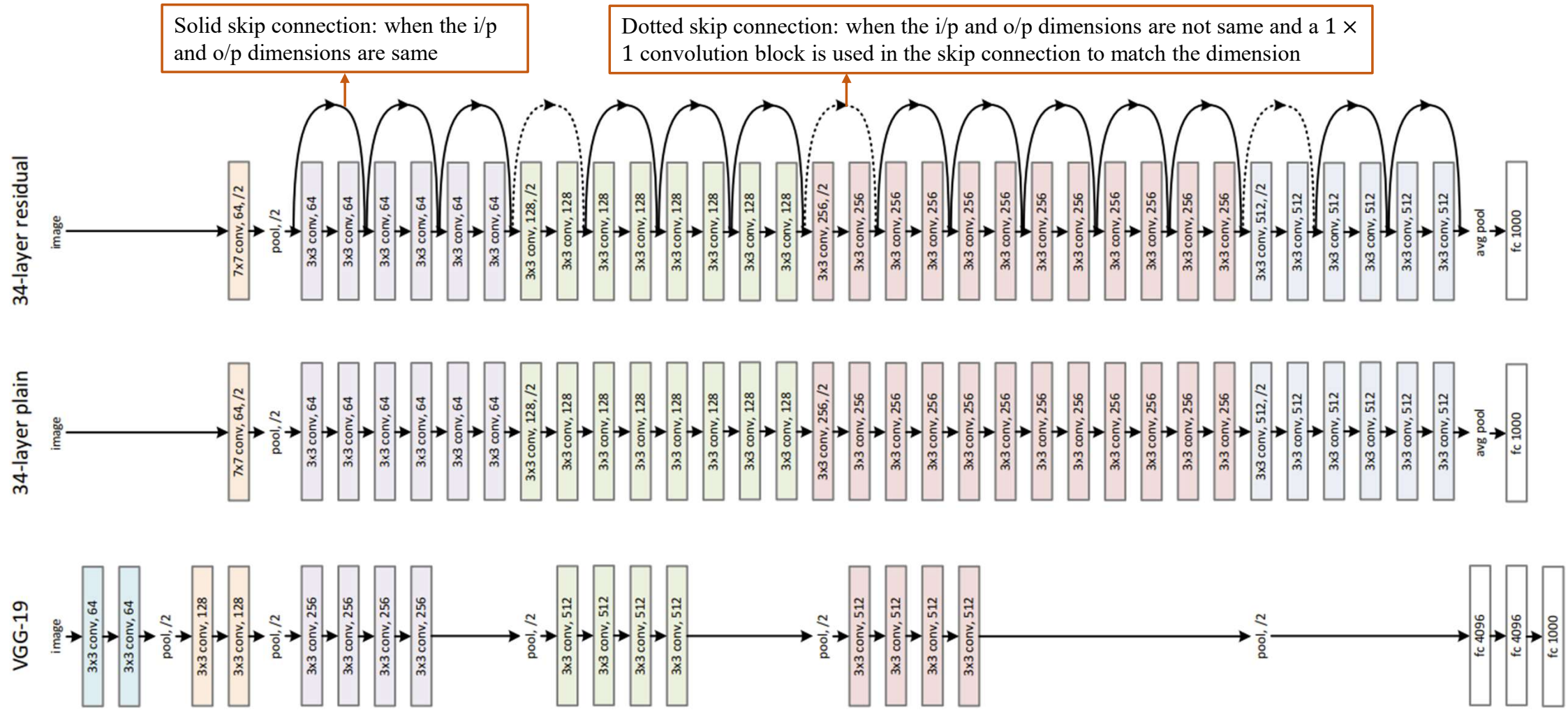
For a particular skip connection, if the learnt weights and bias are zero then

$a^{(l+2)} = \sigma\left(a^{(l)}\right) = a^{(l)}$ [due to ReLU activation]. Then, the network learns the identity function in that skip connection.

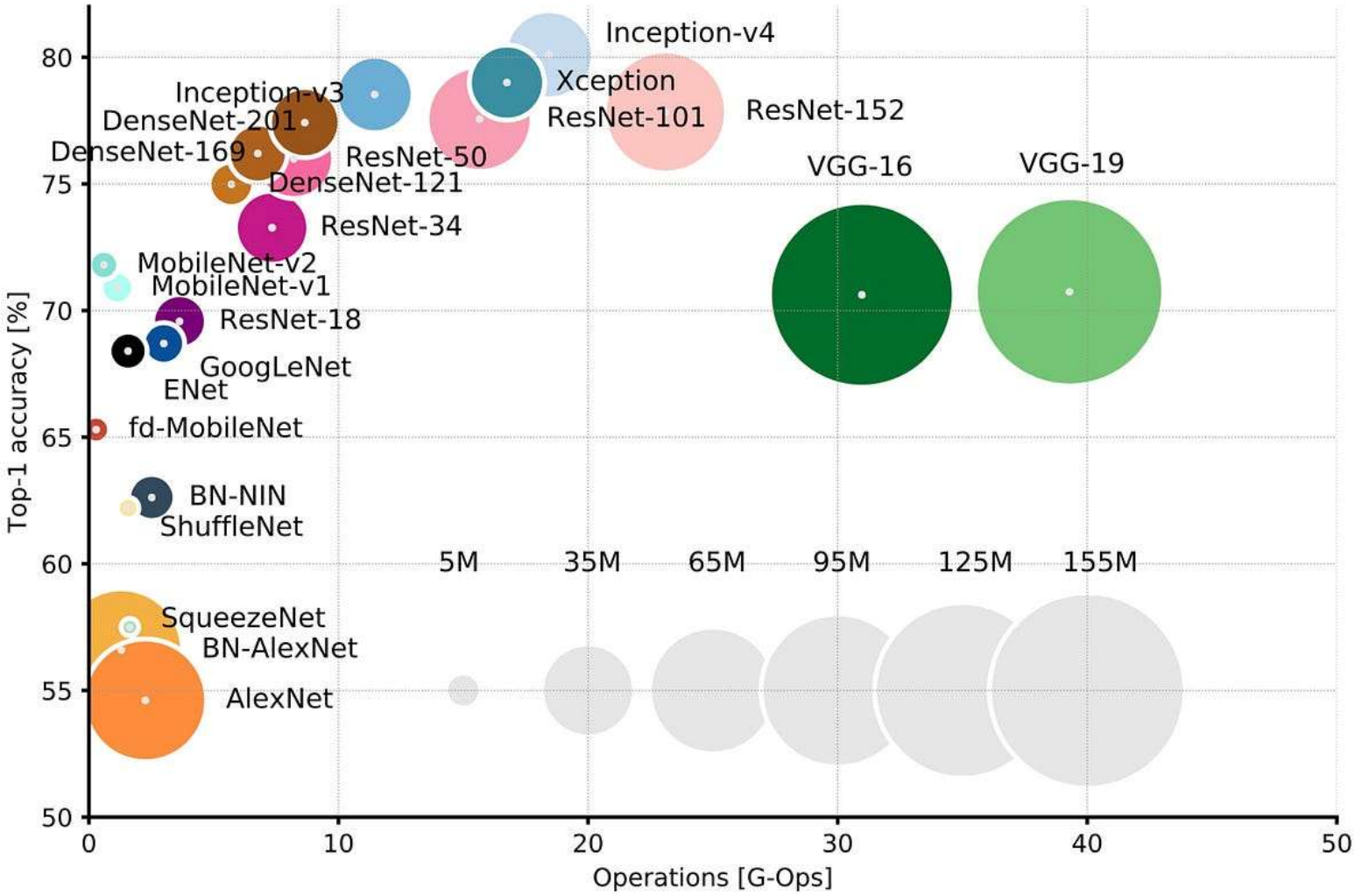


Alternative residual blocks with convolution in skip connections, such that the input passed via the shortcut is resized to the dimensions of the main path's output. The convolution in the shortcut path is a 1×1 convolution.

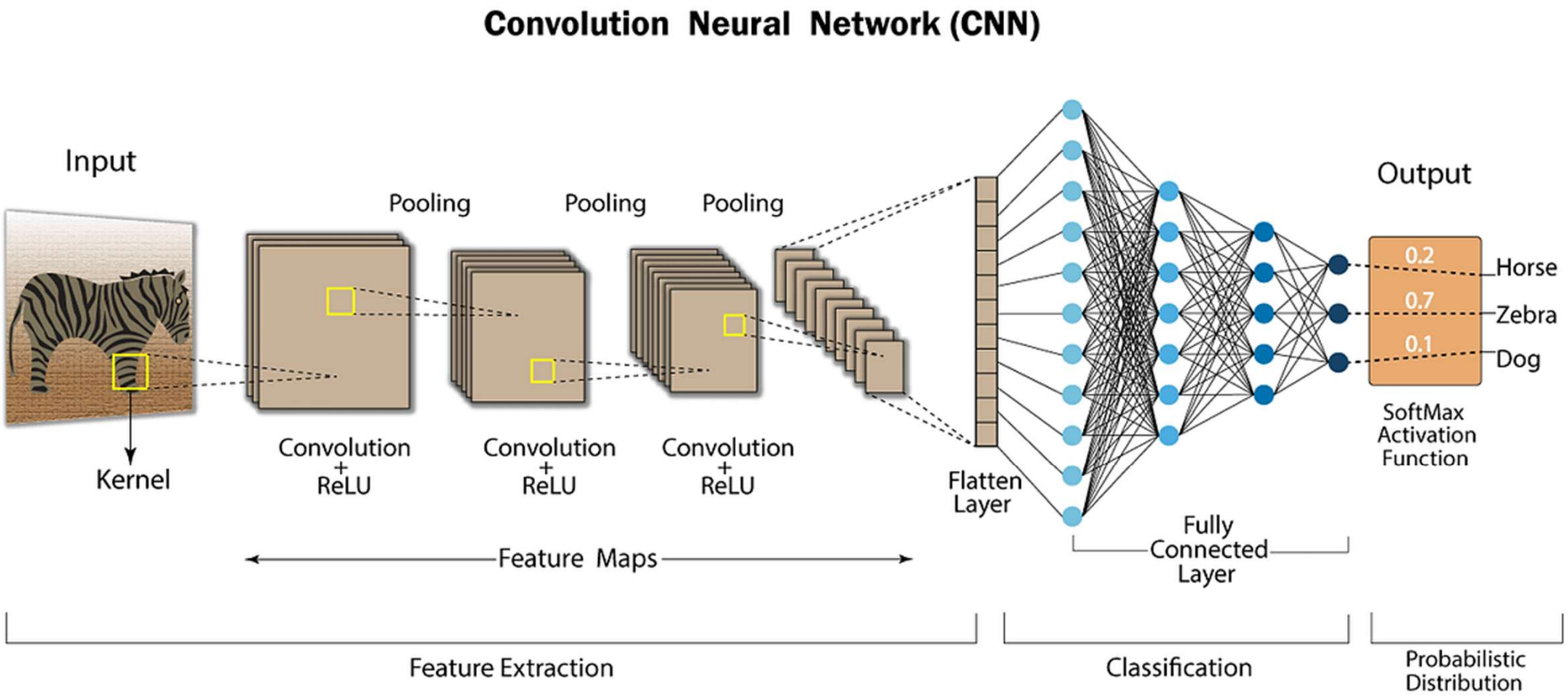
ResNet and Skip Connections (2016)



Performance vs Size comparison



Revisiting general CNN architecture



Revisiting general CNN architecture

Convolutional Neural Networks (CNNs) extract features from an image in a **hierarchical manner**, meaning that each successive convolutional layer builds upon the features learned by the layer before it. [*Reference*]
The features can be grouped into three main types based on the depth of the layer that extracts them:

1. Low-level features (Early Layers)

These are the most fundamental, localized visual elements. The filters in the first few convolutional layers are tuned to detect simple, universal patterns, much like the first stages of human vision. The features extracted by these early layers:

- Edges: Straight, curved, or diagonal lines (e.g. the boundary between a light and dark region)
- Gradients: Changes in pixel intensity or color
- Corners: Intersection of two edges
- Simple Textures: Repeating patterns like stripes, grids, or tiny dots.

2. Mid-level features (Intermediate Layers)

Layers deeper into the network combine the simple low-level features to form more complex, localized shapes. These are often recognizable as parts of an object.

- Object parts: recognizable components like eyes, noses, wheels, doors, wings etc.
- Complex Textures: More intricate, non-uniform surface patterns like fur, skin, fabric weave etc.
- Contours/Shapes: Complete, closed-shapes formed by combining edges and corners.

3. High-level features (Intermediate Layers)

The deepest convolutional layers, before the final classification layers, combine the mid-level features to form a highly abstract and semantic representation of the entire image content.

Transfer Learning

Key idea:

- Feature extraction layers are in-general very useful for detecting high, mid, and low-level features. The goal is to leverage a model's existing, general knowledge (features) without altering the model's core learned structure.
- Transfer Learning involves using a pre-trained model as a **fixed feature extractor**.
- Use a pre-trained model (e.g. VGG16 pretrained on ImageNet dataset),
  - **Freeze the weights** of the feature extractor part of the pre-trained model. This is called **frozen base**. The weights of these layers are not updated during training.
  - A new small output layer (called the "classification head") is added on top the frozen base and trained on the new, specific task data. The number of nodes in the final output layer is same as the number of classes in the new dataset.

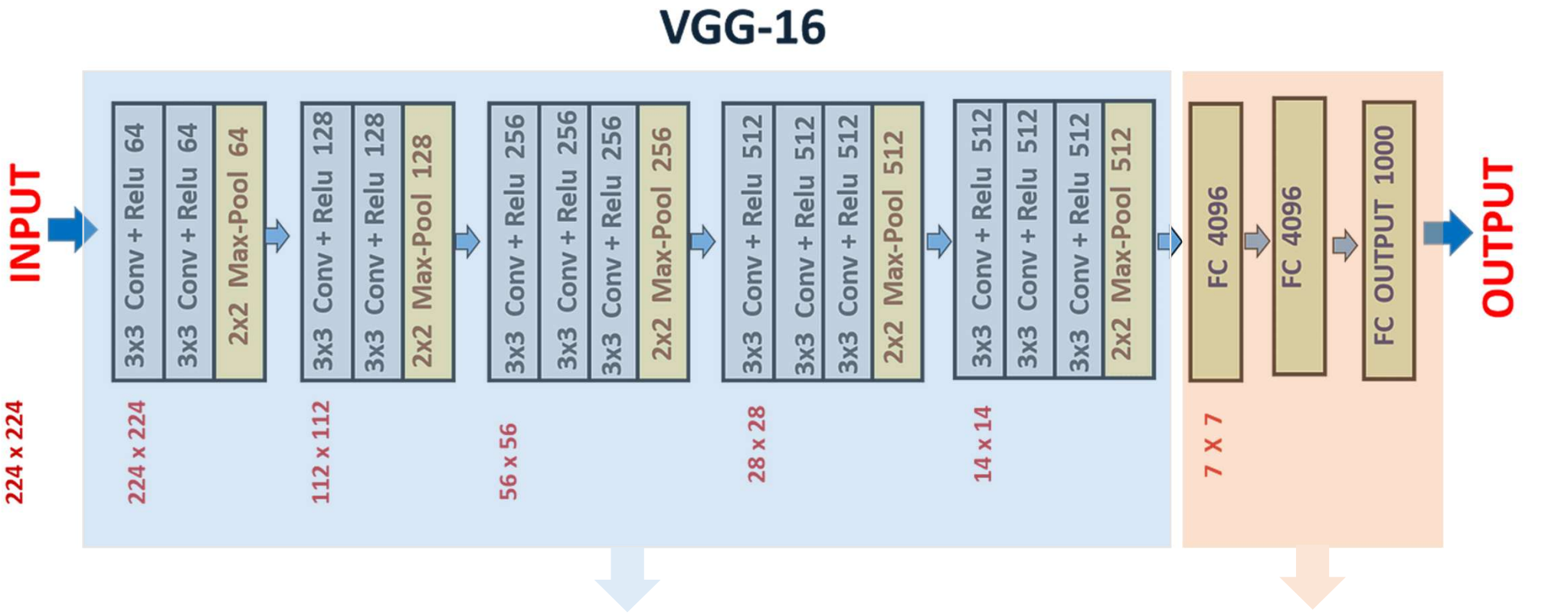
When is the transfer learning useful:

Usually, domain specific datasets don't have a large number of images. When we have a small dataset, then training a CNN from scratch doesn't make sense. In that case, we use transfer learning to achieve good performance on the small dataset.

Benefits of transfer learning:

- Can obtain state-of-the-art performance on small domain specific datasets.
- Can be trained fast as the back-propagation only occurs through the small, new top layers (classification head)
- It's low cost as it doesn't require long GPU hours to train the models.

Transfer Learning



Freeze the weights of the feature extractor part of the model

Replace the classification head

Transfer Learning

Why is it called transfer learning:

- The original pre-trained CNN was trained to classify some images.
- We are using the feature extractor parameters learnt by the base pre-trained model unaltered. i.e. we are using the same feature extractor to extract the features of our new domain specific datasets.
- By freezing the feature extractor part of the model, we are **transferring the knowledge** of classifying one set of images to classify another set of images.
- That's why it is called transfer learning.

Fine Tuning : another related concept

Key idea:

- While the terms are often used interchangeably, **Fine-Tuning is a specific, advanced method of Transfer Learning**.
- Fine tuning is the process of taking a pre-trained model and retraining some (or all) of its existing weights on the new, specific dataset.
- The goal is to adapt the pre-trained model's *general* features to better capture the *specific* features and nuances of the new target domain/task.
- **A subset of the pre-trained layers (usually the later ones) are unfrozen** and their weights are adjusted along with the new top layer. In "full fine-tuning," all layers are unfrozen.
- Requires a **moderate to large dataset** to justify retraining a large number of parameters without overfitting and corrupting the generalized knowledge.
- **Higher** cost and **slower** than pure transfer learning because backpropagation must flow through more (or all) layers.

Sometimes we often use fine-tuning in practice and call it transfer learning. Because, fine-tuning is more advanced method of transfer learning.

Thank You