Convolutional Neural Network (Recap)

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

KAUST Academy
King Abdullah University of Science and Technology

June 3, 2025

Table of Contents

Table of Contents (cont.)

Motivation

- **Transformative Impact:** Convolutional Neural Networks (CNNs) have revolutionized computer vision and deep learning.
- **Wide Applications:** Power tasks such as image classification, object detection, facial recognition, and medical image analysis.
- **Automatic Feature Learning:** CNNs learn hierarchical features directly from raw pixel data, reducing the need for manual feature engineering.
- **Demystifying CNNs:** This presentation breaks down core components and traces the evolution of key architectures.
- **Practical Relevance:** Understanding CNNs enables better design, optimization, and application to real-world problems.

Learning Outcomes

By the end of this presentation, you will be able to:

- **Define CNNs:** Understand what a Convolutional Neural Network is and its role in deep learning.
- **Explain Key Components:**
  - Convolutional layers
  - Activation functions
  - Pooling layers
  - Padding and strides
  - Batch normalization
  - Dropout regularization
  - Flattening and fully connected layers
- **Differentiate Training Techniques:** Compare loss functions and optimization methods used in CNNs.

Learning Outcomes (cont.)

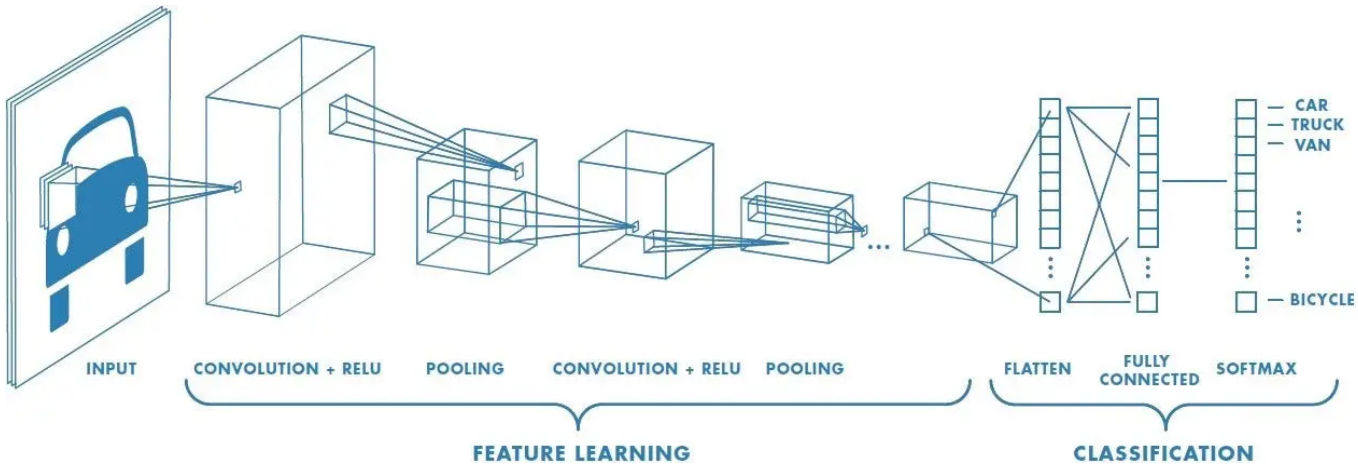
- **Identify Major Architectures:** Recognize and contrast LeNet, AlexNet, VGG, InceptionNet, ResNet, and MobileNet.
- **Understand Transfer Learning:** Grasp the concept and advantages of transfer learning in CNNs.
- **Evaluate Performance:** Use common metrics to assess CNN effectiveness.
- **Explore Further:** Discover advanced topics and resources for continued learning in the CNN field.

CNN - Definition

- Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for processing structured grid data, such as images.
- They are particularly effective for tasks like image classification, object detection, and segmentation.
- CNNs leverage the spatial structure of images by using convolutional layers to automatically learn hierarchical features.
- The architecture typically consists of convolutional layers, activation functions, pooling layers, and fully connected layers.
- CNNs are known for their ability to capture local patterns and translate them into higher-level representations.

What is a CNN?

A CNN is a deep network of neurons with learnable filters that perform convolution operations on inputs, usually images, to extract hierarchical features.

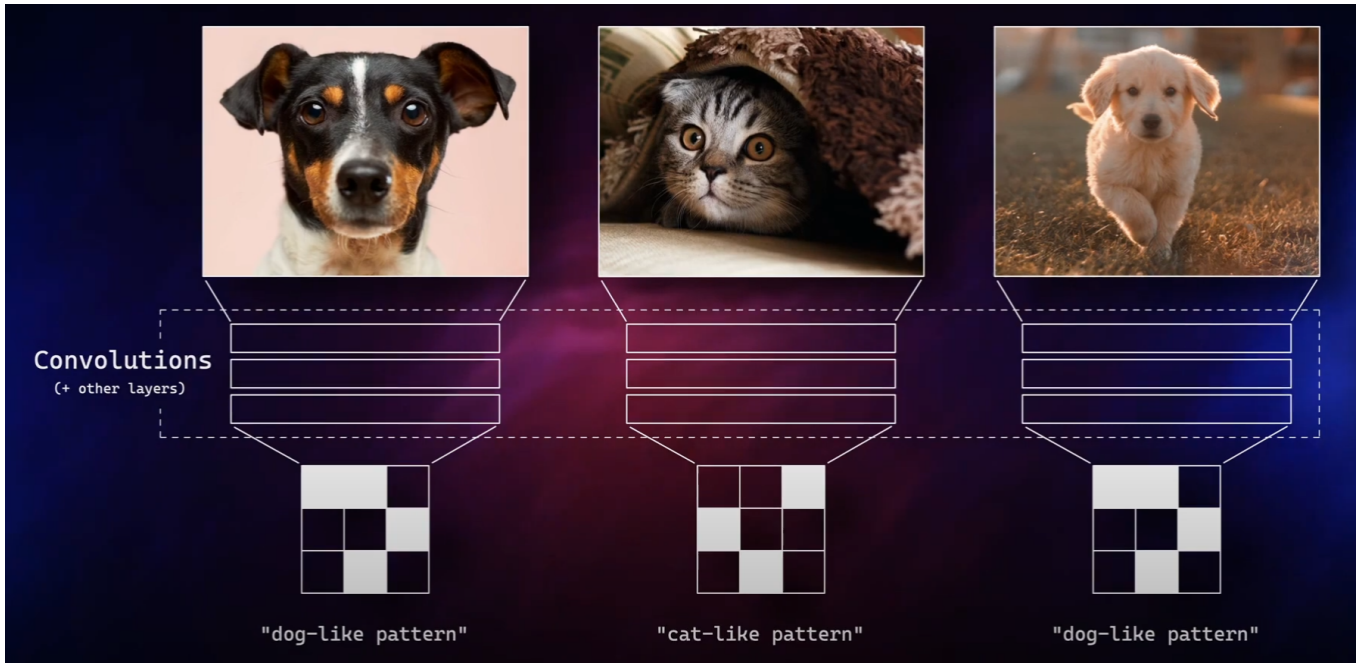


Why use CNNs?

Parameter sharing and sparse connectivity reduce number of parameters and improve spatial feature extraction.

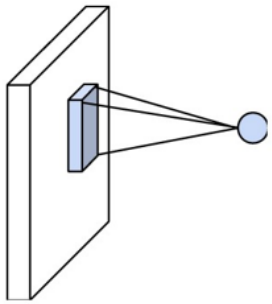
What makes a Convolutional Neural Network?

Characterised by "Convolutional Layer" – they are able to detect "abstract features" and "almost ideas within the image"

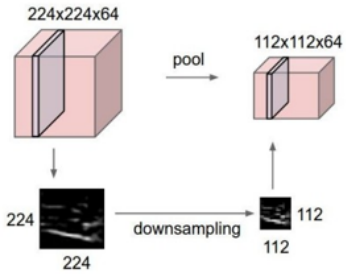


Components of a CNN

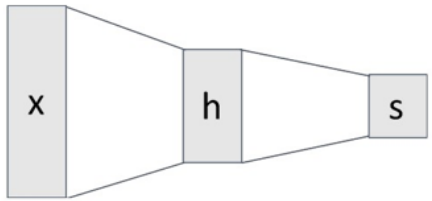
Convolution Layers



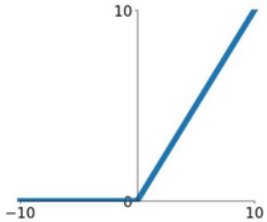
Pooling Layers



Fully-Connected Layers



Activation Function



Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

CNN - Convolutional Layer

أكاديمية كاوست
KAUST ACADEMY

Operation:

- Element-wise multiply filter with image patch and sum $\rightarrow$ feature map.

Hyperparameters:

- kernel size
- number of filters
- stride
- padding

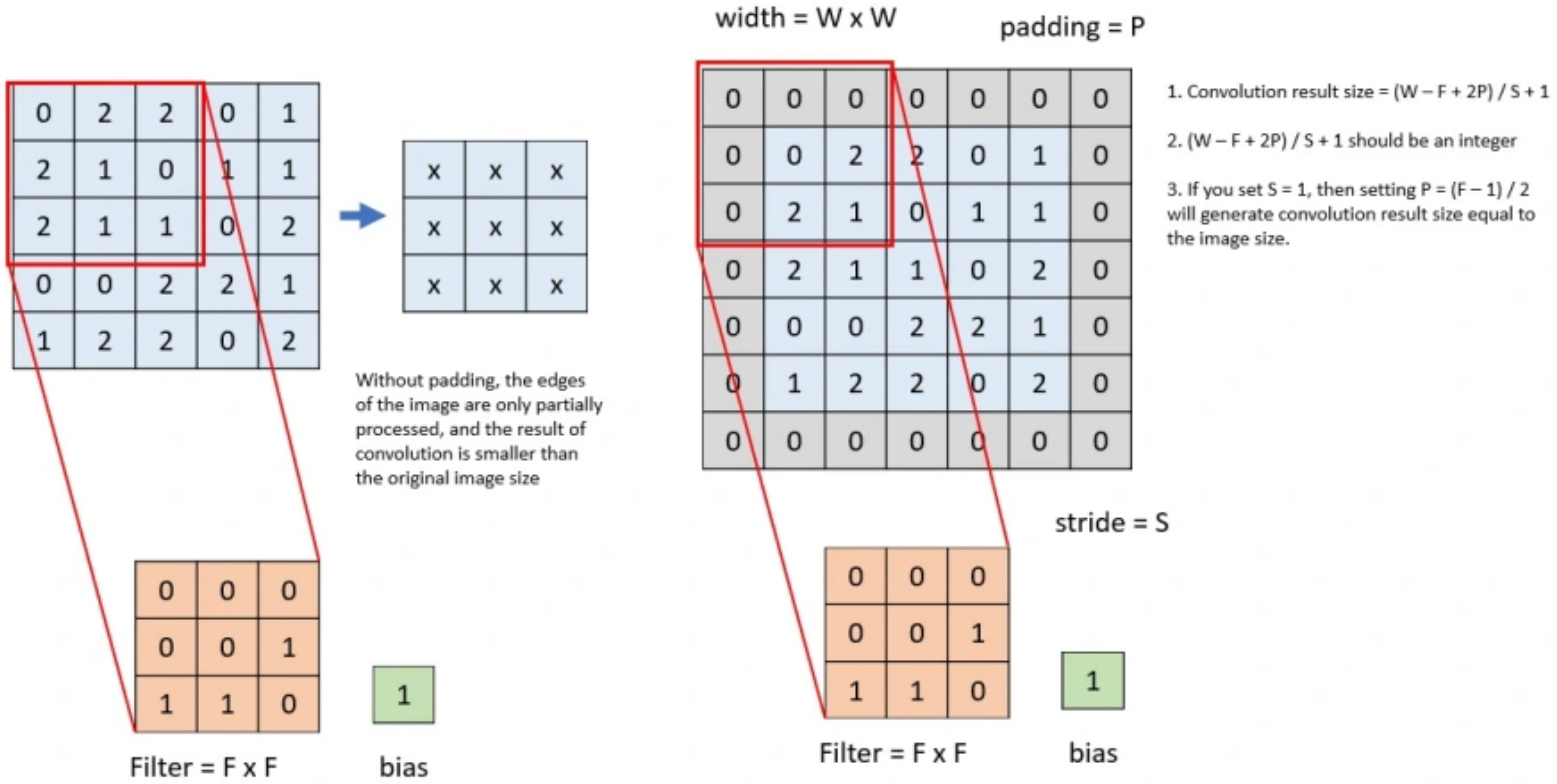
Listing 1: Code snippet (PyTorch)

```
import torch.nn as nn

conv = nn.Conv2d(in_channels=3, out_channels=16, kernel_size
    =3, stride=1, padding=1)

output = conv(input_tensor)  # input_tensor: [batch_size, 3,
     H, W]
```

CNN - Convolutional Layer

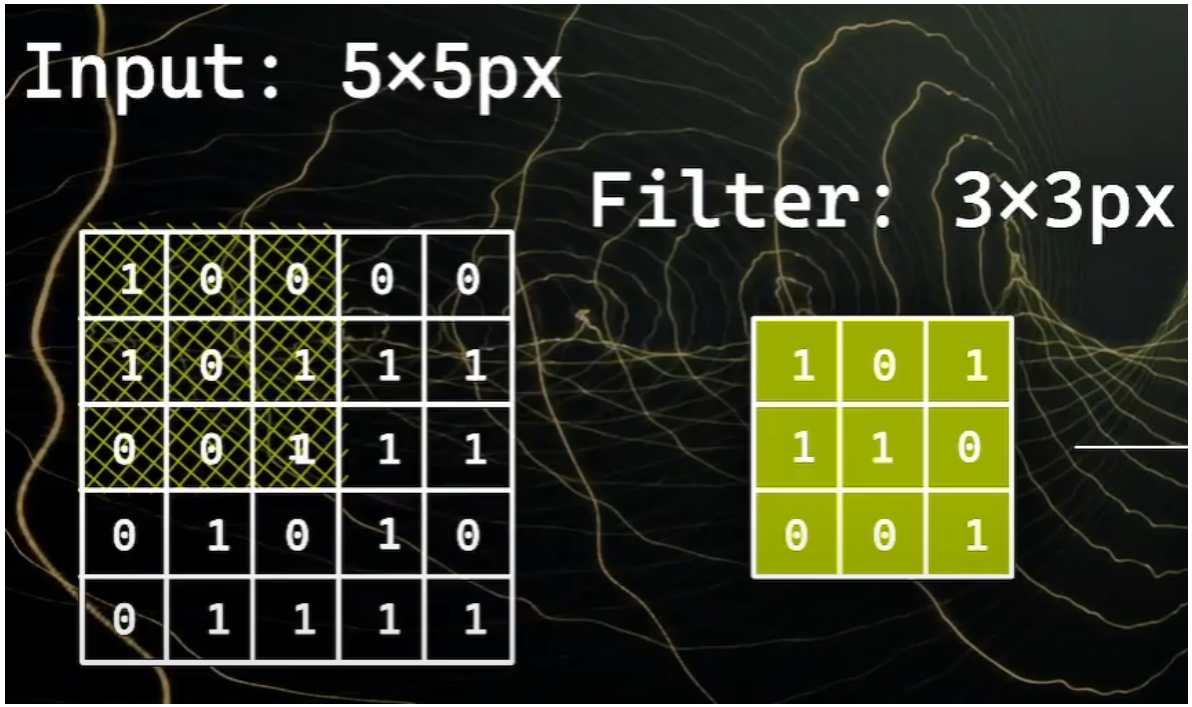


Interactive demo: cs231n.github.io/convolutional-networks

Quick Exercise (5 mins)

Let's find out what this can give us:

- Padding $= 0$
- Stride $= 1$



Note: Once you traverse entire image/matrix it will give you a matrix calls Feature Map or Activation Map.

CNN - Activation Functions

Role:

- Introduce non-linearity so multiple conv layers can learn complex mappings.

Common:

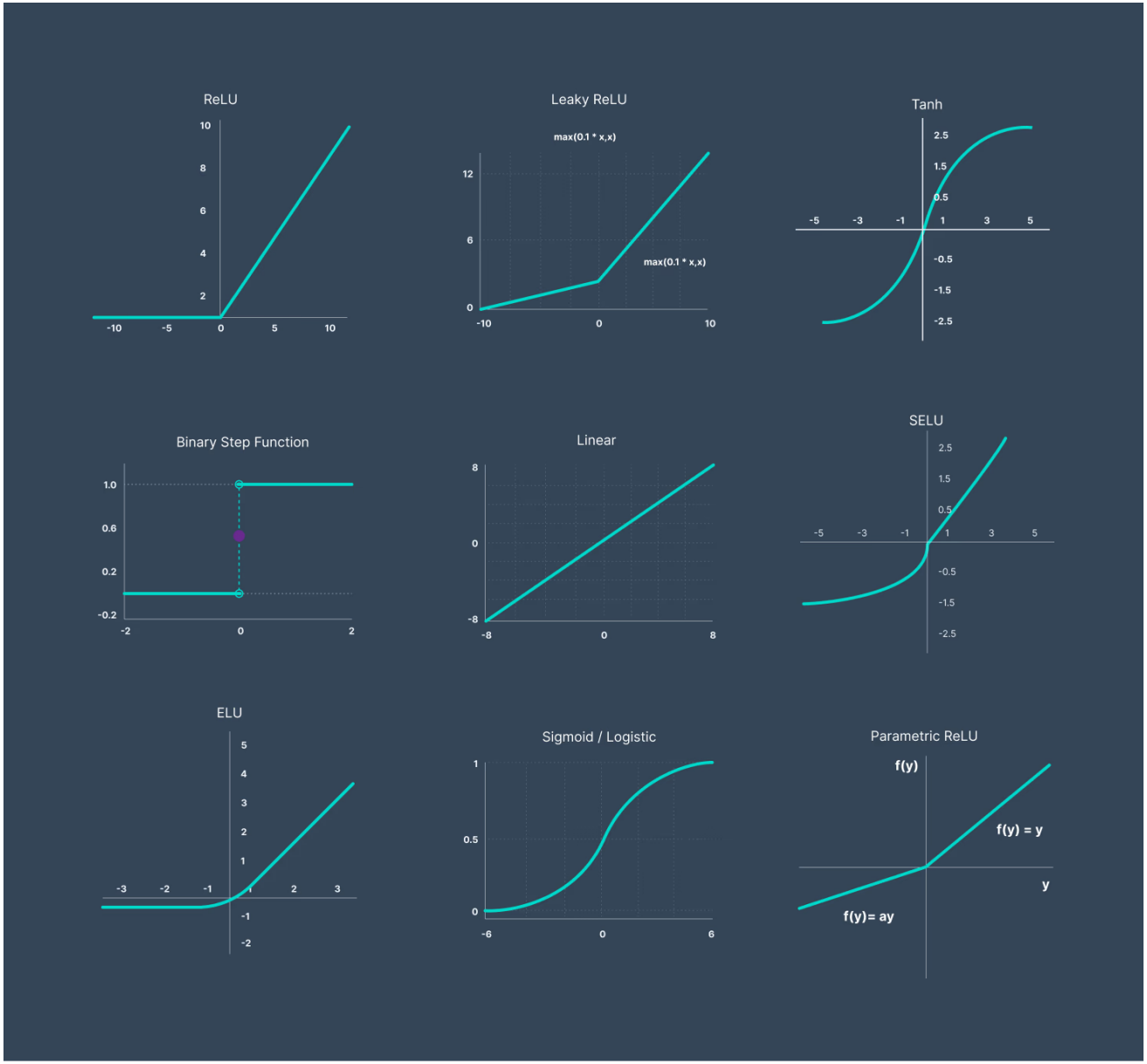
- ReLU (Rectified Linear Unit)
- Leaky ReLU
- Sigmoid
- Tanh

Listing 2: Code snippet (PyTorch)

```
import torch.nn.functional as F

x = conv(input_tensor)
x = F.relu(x)                  # ReLU
x = F.leaky_relu(x, 0.1)       # Leaky ReLU
```

CNN - Activation Functions



CNN - Pooling Layers

Purpose:

- Downsample feature maps, reduce spatial dims and parameters, add invariance.

Types:

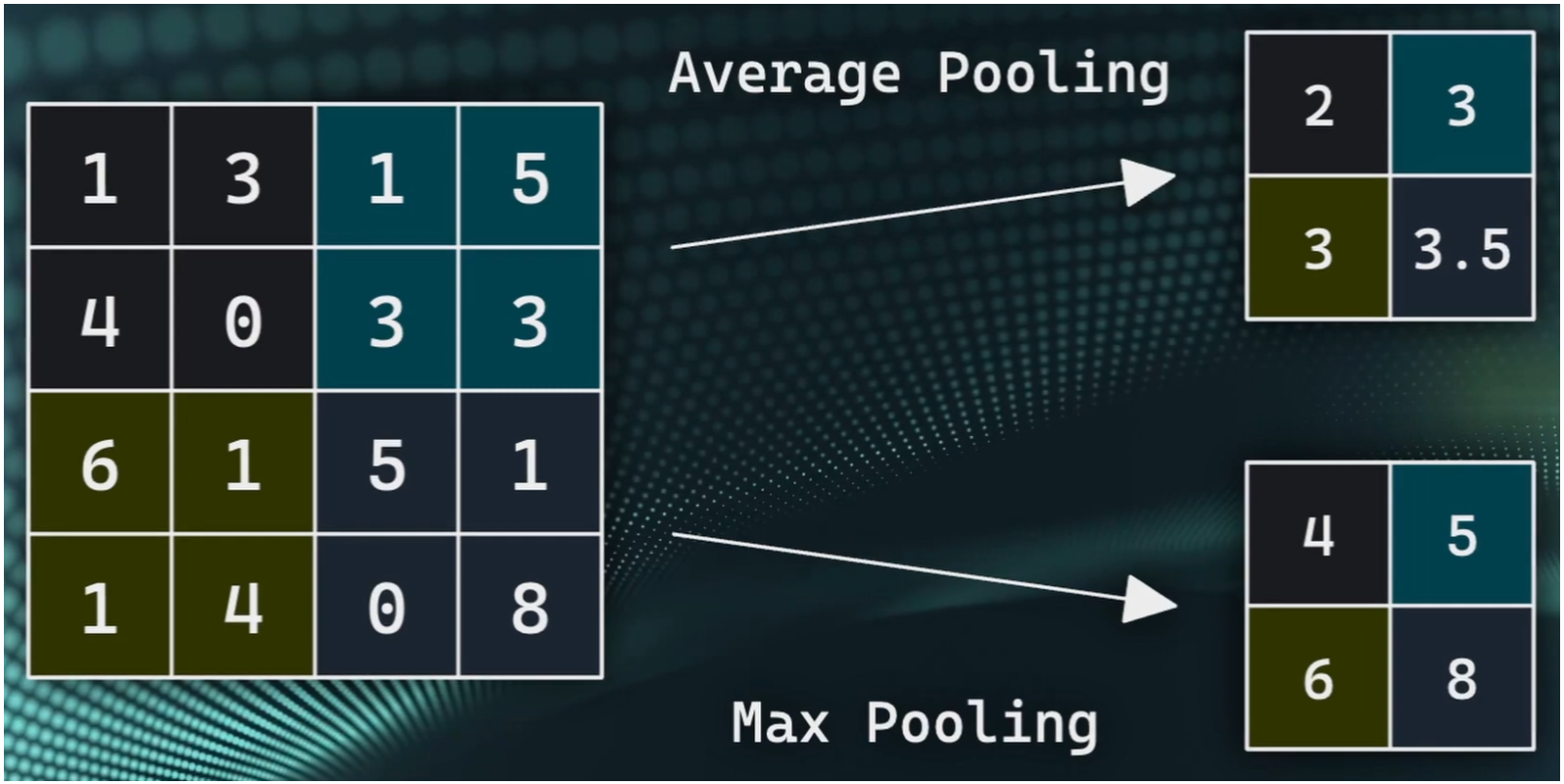
- Max Pooling
- Average Pooling
- Global Average Pooling
- Global Max Pooling

Listing 3: Code snippet (PyTorch)

```
import torch.nn as nn

pool = nn.MaxPool2d(kernel_size=2, stride=2)
pooled = pool(x)  # halves H and W
```

CNN - Pooling Layers



CNN - Padding Strides

أكاديمية كاوست
KAUST ACADEMY

Padding:

- "same" preserves spatial size by adding zeros around input;
- "valid" reduces spatial size by not adding any padding.
- Padding is used to control the spatial size of the output feature map.
- Padding is added to the input image before applying the convolution operation.

Strides:

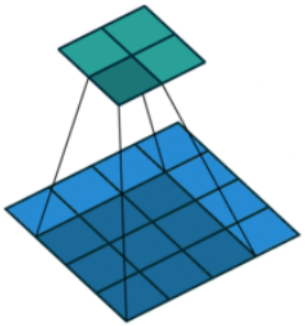
- Strides control how much the filter moves across the input image.
- Strides can be set independently for height and width.
- Strides are used to control the spatial size of the output feature map.
- Strides are set in the convolutional layer.

Listing 4: Code snippet (PyTorch)

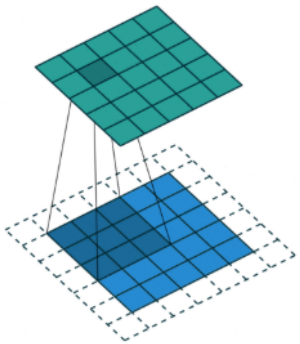
```
import torch.nn as nn

conv_valid = nn.Conv2d(3, 16, 3, stride=2, padding=0) #
        Valid    : no padding (padding=0)
conv_same  = nn.Conv2d(3, 16, 3, stride=1, padding=1) #
        Same     : preserve size
```

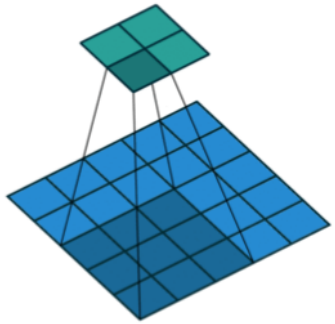
CNN - Padding Strides



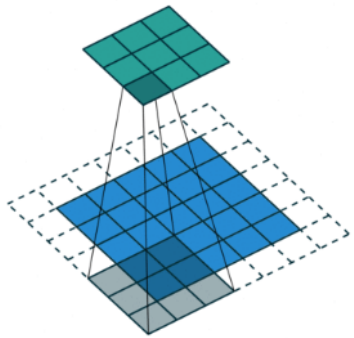
padding = 0, stride = 1



padding = 1, stride = 1



padding = 0, stride = 2



padding = 1, stride = 2

CNN - Normalization (Batch Norm)

What:

- Normalize layer inputs over mini-batch, then scale shift.

Benefits:

- Faster training,
- Allows higher learning rates,
- Reduces sensitivity to initialization,
- Acts as a regularizer.

Listing 5: Code snippet (PyTorch)

```
import torch.nn as nn

bn = nn.BatchNorm2d(16)
x = bn(x)
```

Note:

- BatchNorm is typically used after convolutional layers and before activation functions.
- BatchNorm normalizes activations per-batch, then scales/shifts them.

CNN - Normalization (Batch Norm)

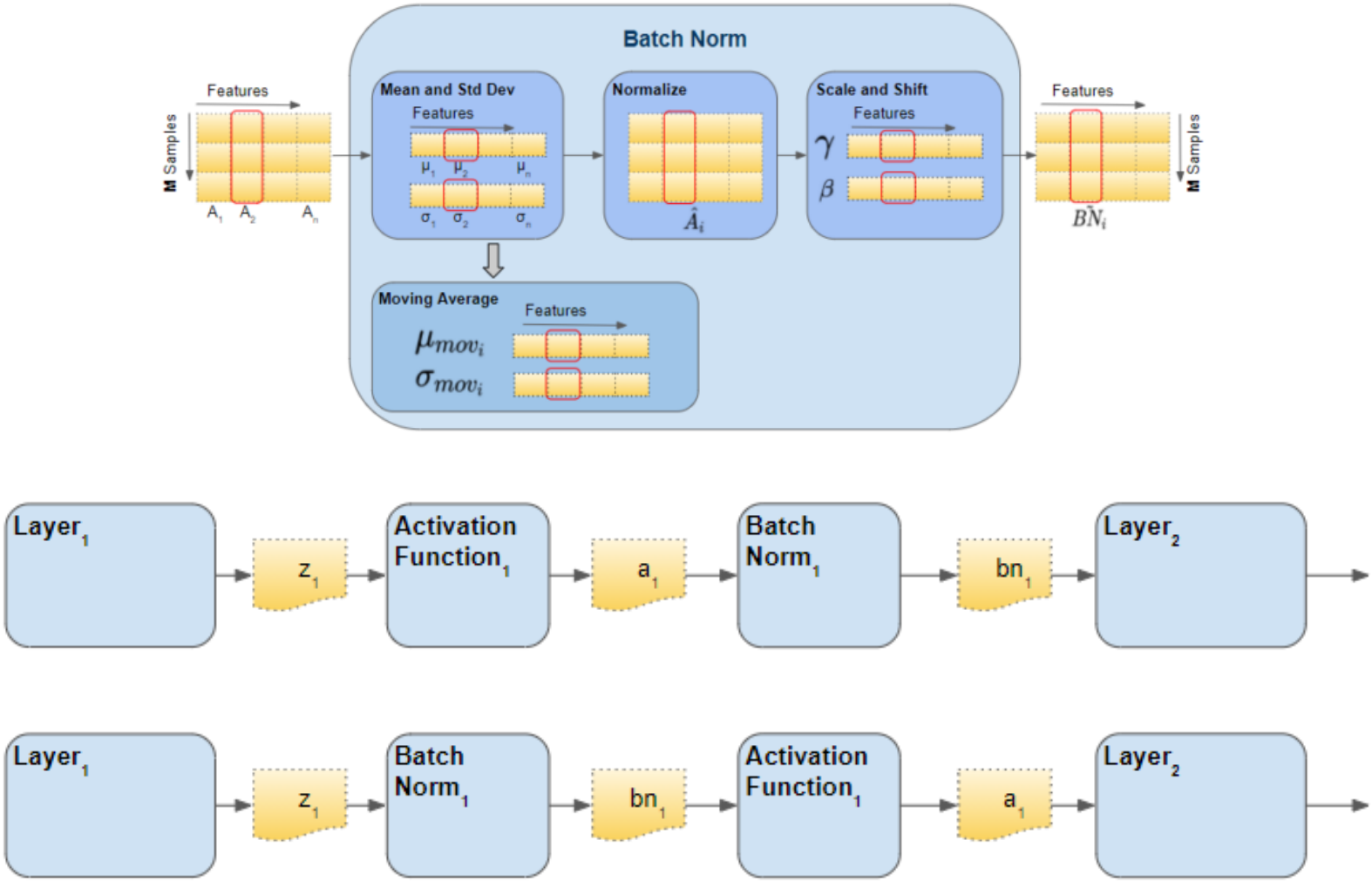


Figure 2: Order of placement of Batch Norm layer

More on Batch Norm from Towards Data Science

CNN - Regularization (Dropout)

What:

- Randomly drop units with probability p during training to prevent co-adaptation.

Where:

- Often after FC layers, sometimes after conv layers.

Effect on images:

- Regularization does not alter the input images but modifies the training process to improve generalization.

Listing 6: Code snippet (PyTorch)

```
import torch.nn as nn

drop = nn.Dropout(p=0.5)
x = drop(x)
```

CNN - Flattening Fully Connected Layers

أكاديمية كاوست
KAUST ACADEMY

Flatten:

- Flattening is the process of converting a multi-dimensional tensor into a one-dimensional tensor.
- Flattening is typically done before passing the data to fully connected layers.

FC Layer:

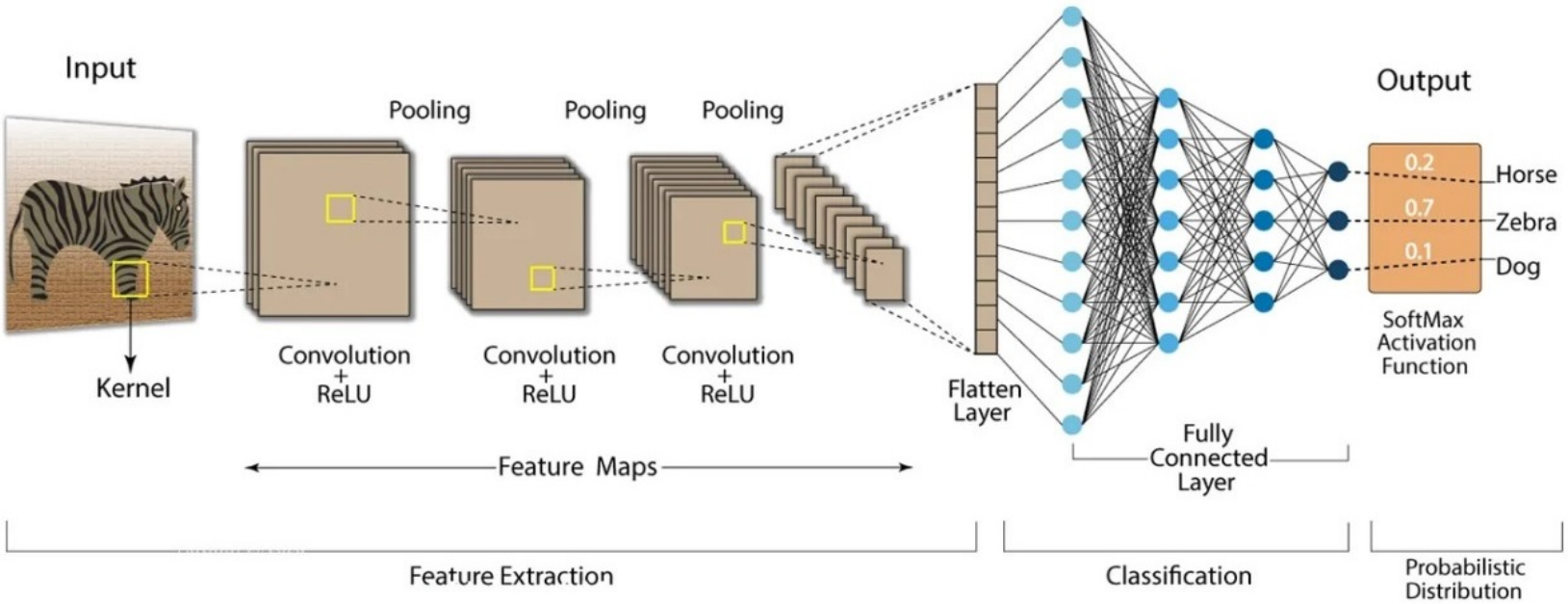
- Used to learn complex relationships between features.
- Typically used at the end of the network to make predictions.
- Take the flattened output from the previous layer and produce a vector of class scores.
- Are often followed by activation functions like ReLU or softmax.

Listing 7: Code snippet (PyTorch)

```
import torch.nn as nn
import torch

x = torch.flatten(x, 1)  # preserve batch dim
fc = nn.Linear(in_features=16*8*8, out_features=10)
out = fc(x)
```

CNN - Flattening Fully Connected Layers



CNN - Loss Functions Optimizers

Loss:

- Loss functions measure how well the model's predictions match the true labels.
- Common loss functions include:
  - Cross-Entropy Loss (for classification)
  - Mean Squared Error (for regression)
- The choice of loss function depends on the task at hand.

Optimizers:

- Optimizers update the model's parameters based on the gradients computed during backpropagation.
- Common optimizers include:
  - Stochastic Gradient Descent (SGD)
  - Adam
  - RMSprop
- The choice of optimizer can significantly affect the training speed and convergence.

Listing 8: Code snippet (PyTorch)

```
import torch.nn as nn
import torch

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
loss = criterion(preds, labels)
loss.backward()
optimizer.step()
```

Most Notable CNN-based Architectures

Over time, researchers built advanced CNN architectures to improve performance and efficiency. These architectures introduced key innovations:

- **LeNet [LeCun et al., 1998]**: The first CNN architecture, designed for handwritten digit recognition.
- **AlexNet [Krizhevsky et al. 2012]**: The first CNN to achieve breakthrough performance on image classification.
- **VGGNet [Simonyan and Zisserman, 2014]**: Used very deep networks (up to 19 layers).
- **InceptionNet (GoogLeNet) [Szegedy et al., 2014]**: Used multiple filter sizes per layer (Inception modules).
- **ResNet [He et al., 2015]**: Introduced skip connections for training very deep networks.
- **EfficientNet [Tan and Le, 2019]**: Found a scaling method that simultaneously scales a CNN's depth, width, and resolution optimally using a single scaling coefficient.
-
- **MobileNet [Howard et al., 2017]**: Designed for mobile and embedded vision applications, using depthwise separable convolutions.

LeNet

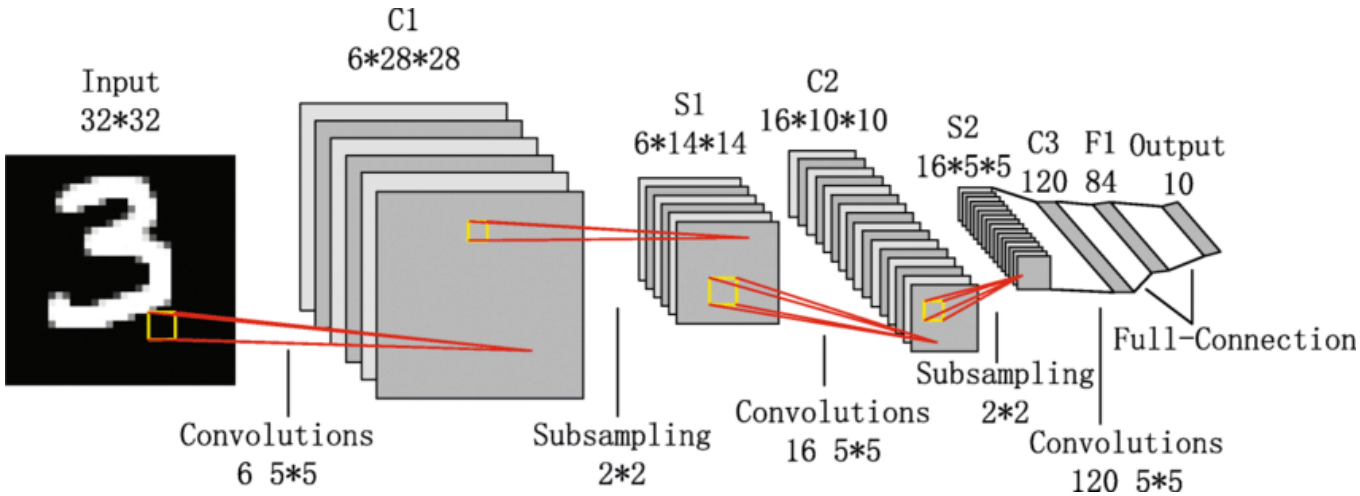
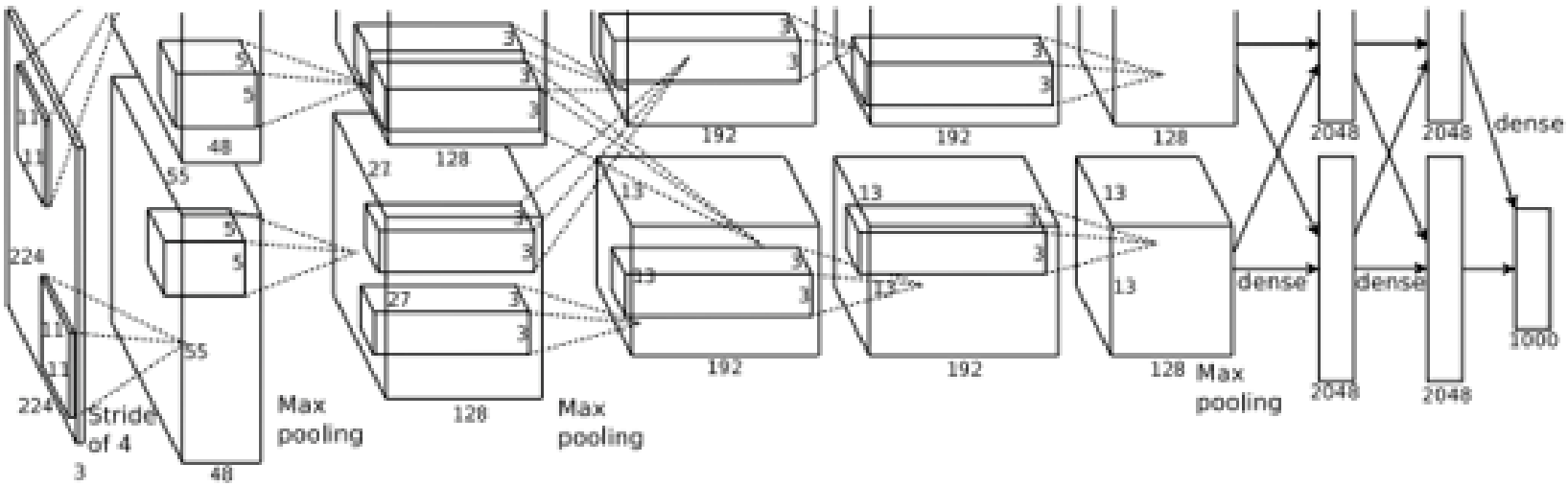


Figure 3: LeNet-5 architecture.

- LeNet-5 is a pioneering convolutional neural network (CNN) architecture developed by Yann LeCun and his collaborators in the late 1980s and early 1990s.
- It was primarily designed for handwritten digit recognition, specifically for the MNIST dataset.
- The architecture consists of two convolutional layers, followed by subsampling (pooling) layers, and then fully connected layers.
- It introduced key concepts such as convolutional layers, pooling layers, and the use of activation functions like sigmoid

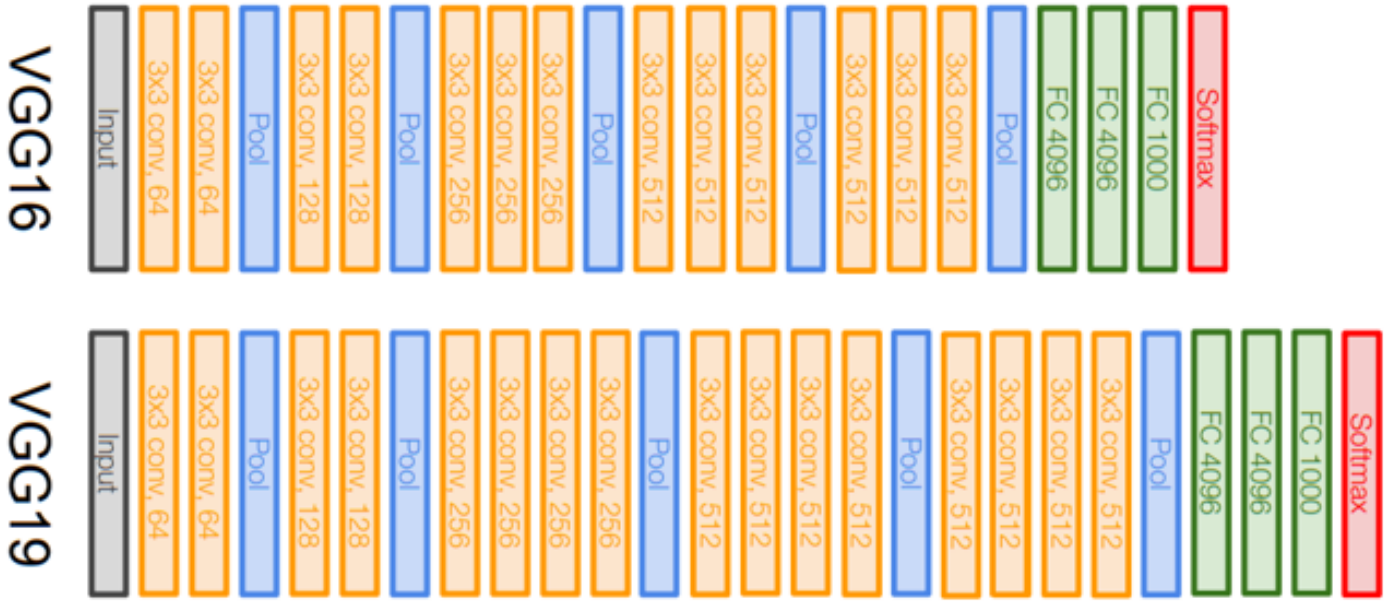
AlexNet

- First big improvement in image classification.
- Made use of CNN, pooling, dropout, ReLU and training on GPUs.
- 5 convolutional layers, followed by max-pooling layers; with three fully connected layers at the end



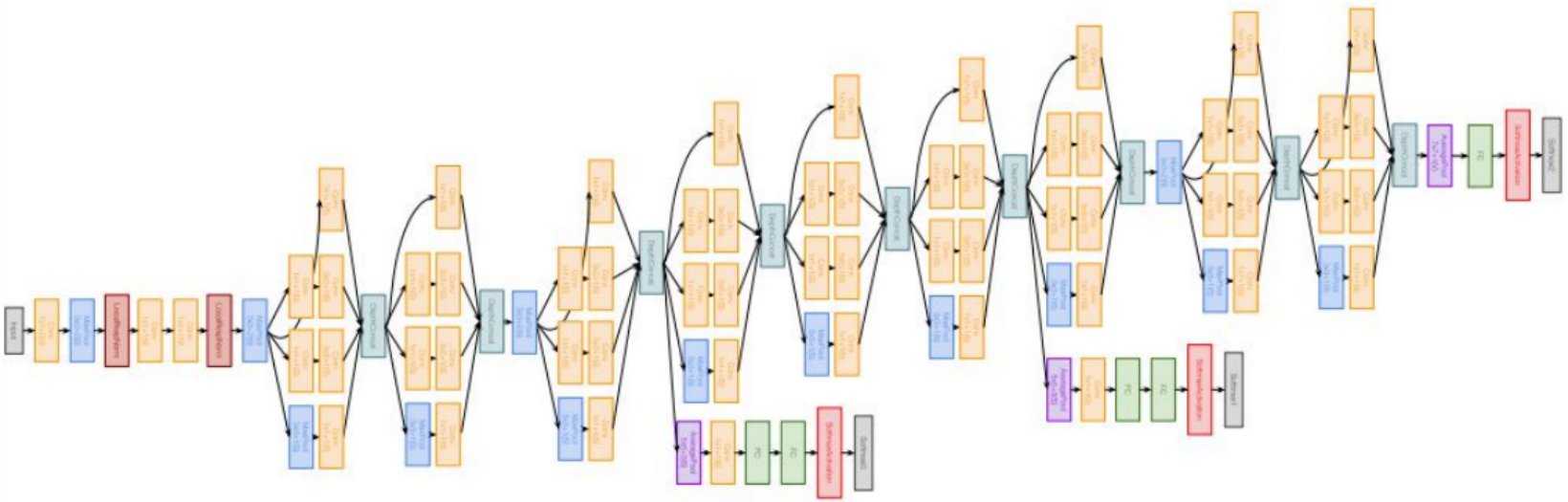
VGGNet

- **Improvement over AlexNet:** Uses a deeper network with small filters instead of a shallow network with larger filters.
- A stack of 3x3 conv layers (vs. 11x11 conv) has same receptive field, more non-linearities, fewer parameters and deeper network representation.
- Two variants: VGG16 or VGG19 conv layers plus 3 FC layers.



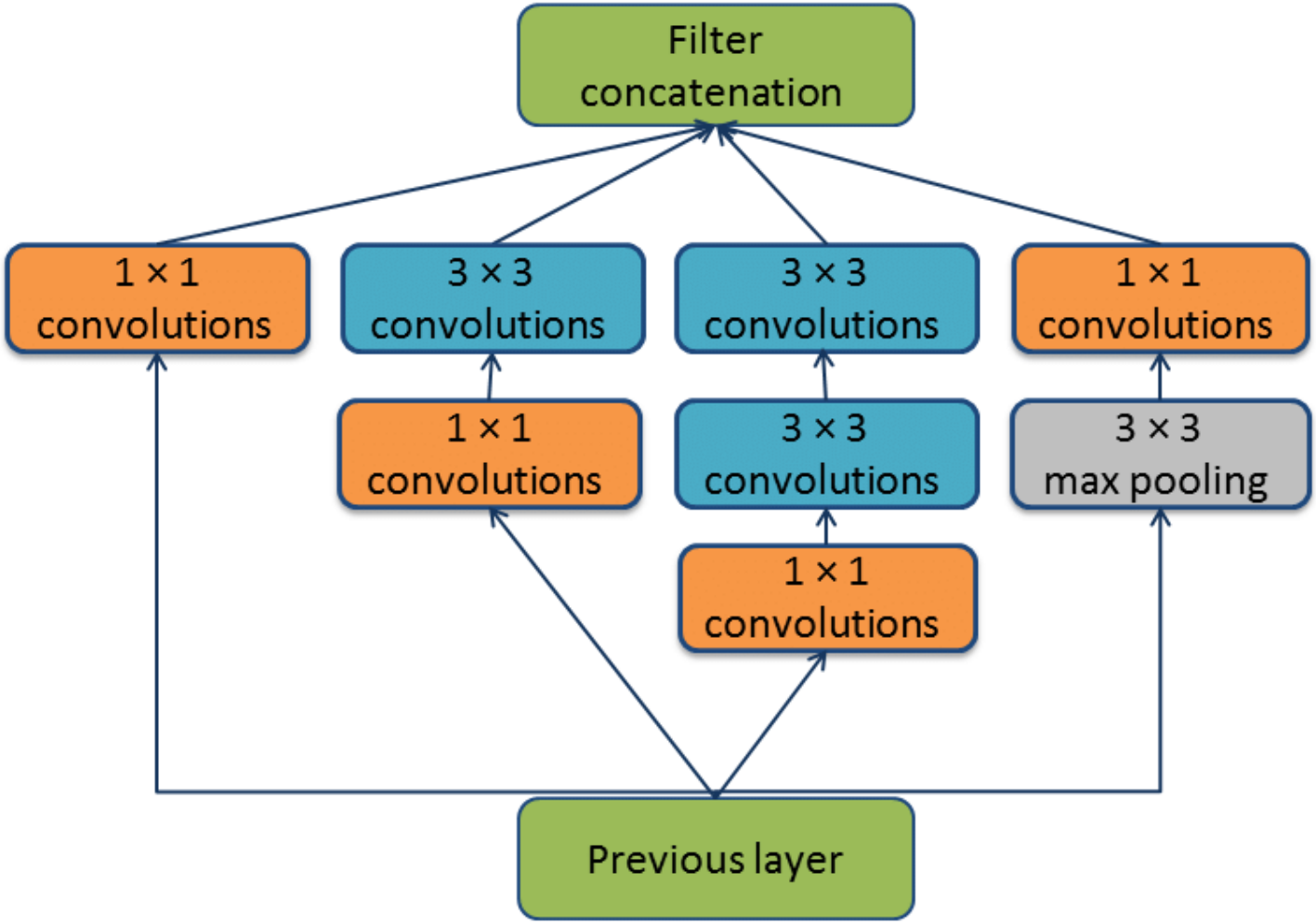
InceptionNet

- Going Deep: 22 layers
- Only 5 million parameters! (12× fewer than AlexNet, 27× fewer than VGGNet).
- Introduced efficient "Inception module"
- Introduced "bottleneck" layers that use 1x1 convolutions to reduce the number of channels and mix information across them.



InceptionNet (cont.)

- **Inception module:** Uses multiple filter sizes (1×1, 3×3, 5×5), in parallel, to capture different features, then combines their outputs.

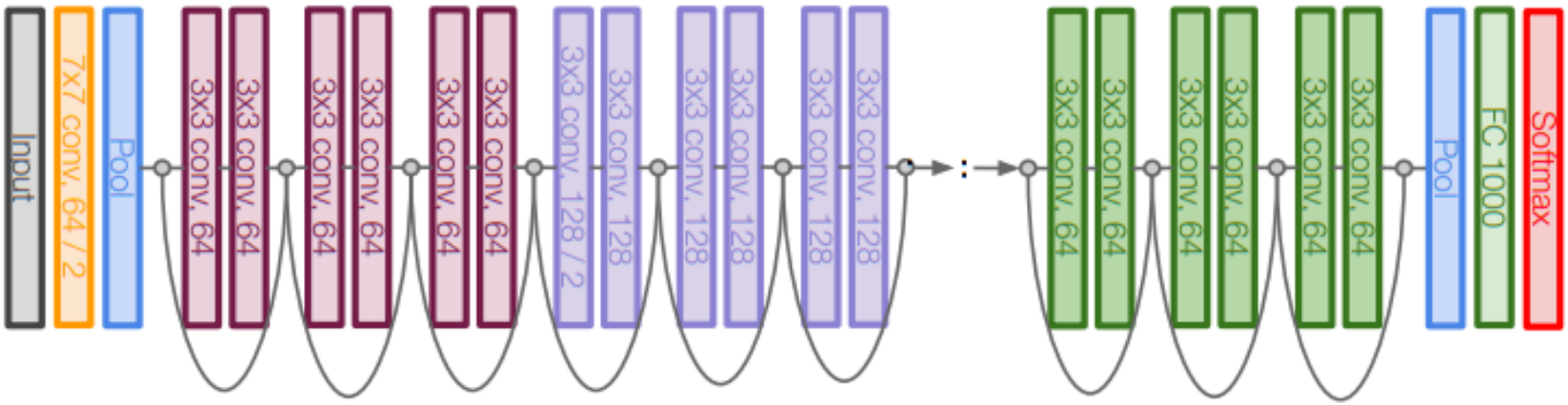


ResNet: Motivation

- **Problem:** Making networks deeper does not always improve accuracy.
- **Why?** In very deep networks, gradients become extremely small as they move backward through layers, making learning slow or stopping it altogether (**vanishing gradient problem**).
- **Solution:** Residual Network (ResNet) introduces **skip connections (residuals)**, allowing information to flow more easily.

ResNet

- Very deep networks using residual connections
- 152-layer model for ImageNet
- Stacked Residual Blocks
- **Residual:** A shortcut connection that helps the network pass information through layers more easily.



EfficientNet: Motivation

- **Problem:** To improve accuracy, we can:
  - Increase the number of layers (**depth**)
  - Increase the number of neurons in each layer (**width**)
  - Use higher-resolution images (**resolution**)

  But finding the right balance of these three was largely based on trial and error.
- **Solution:** EfficientNet introduced a **compound scaling** method—a mathematical formula to systematically find the optimal balance among **depth**, **width**, and **resolution**.

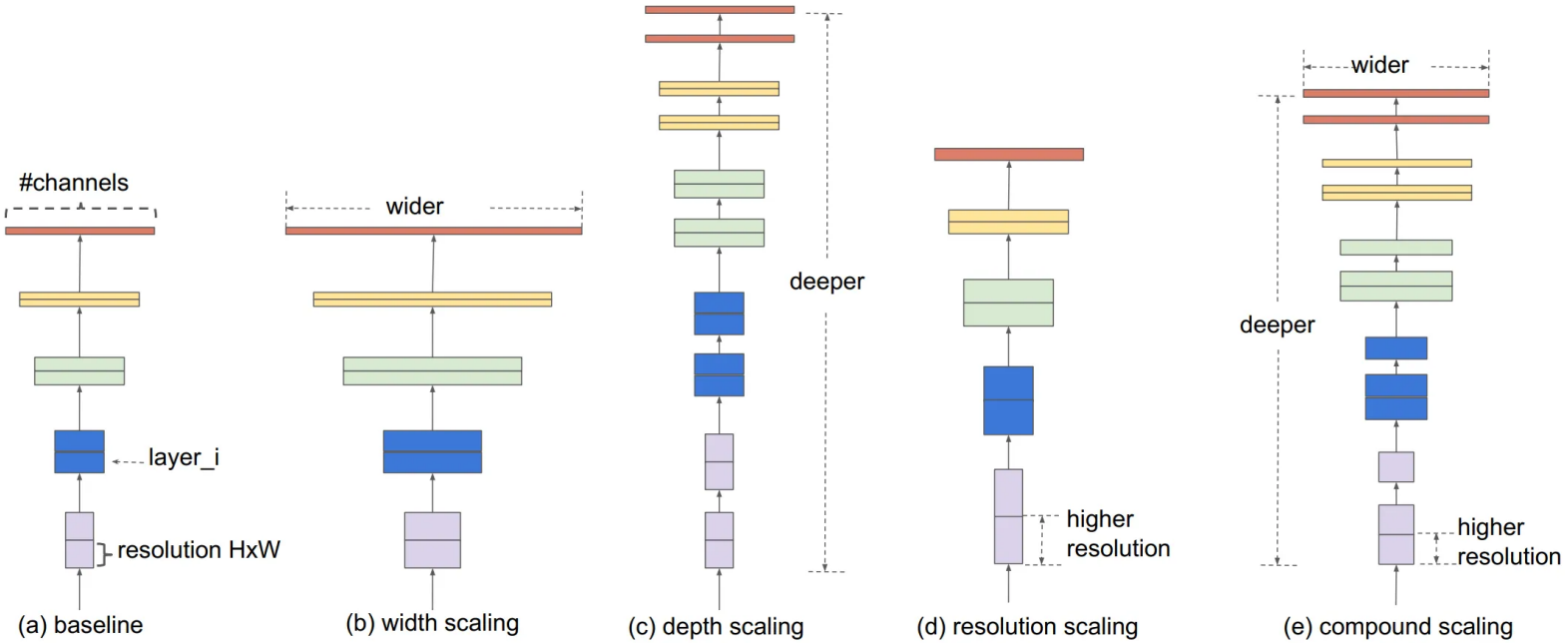


Figure 4: EfficientNet Scaling.

- **Compound Scaling** Uses a single **scaling coefficient** (ϕ) to control:
  - **Network Depth** (α^{ϕ}) $\rightarrow$ More layers
  - **Network Width** (β^{ϕ}) $\rightarrow$ More channels per layer
  - **Input Resolution** (γ^{ϕ}) $\rightarrow$ Larger input images
- The goal: find α, β, γ that balance accuracy & efficiency, then scale up optimally by increasing the global coefficient ϕ.

- EfficientNet optimizes depth, width, and resolution using this constraint:

$$\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

- **Why this equation?**
  - Increasing **depth** (α) increases FLOPs **linearly**.
  - Increasing **width** (β) increases FLOPs **quadratically** (β^2).
  - Increasing **resolution** (γ) increases FLOPs **quadratically** (γ^2).
- To double total FLOPs, the three factors must be balanced together.

EfficientNet

- The authors of EfficientNet searched for the best scaling factors on a small baseline model.
- They found:

$$\alpha = 1.2, \quad \beta = 1.1, \quad \gamma = 1.15$$

EfficientNet Scaling: B0 to B7

- The EfficientNet family (B0 to B7) is generated using:

$$\text{Depth} = 1.2^{\phi}, \quad \text{Width} = 1.1^{\phi}, \quad \text{Resolution} = 1.15^{\phi}$$

Model	ϕ	**Depth** (α^{ϕ})	**Width** (β^{ϕ})
B0	0	$1.2^0 = 1.0$	$1.1^0 = 1.0$
B1	1	$1.2^1 = 1.2$	$1.1^1 = 1.1$
B2	2	$1.2^2 = 1.44$	$1.1^2 = 1.21$
B3	3	$1.2^3 = 1.73$	$1.1^3 = 1.33$
B4	4	$1.2^4 = 2.07$	$1.1^4 = 1.46$
B5	5	$1.2^5 = 2.49$	$1.1^5 = 1.61$
B6	6	$1.2^6 = 2.99$	$1.1^6 = 1.77$
B7	7	$1.2^7 = 3.58$	$1.1^7 = 1.94$

Table 1: Scaling EfficientNet from B0 to B7

- We multiply these scaling factors by the baseline EfficientNet-B0 values and round to the nearest integer to get the new depth, width, and resolution for each model.

EfficientNet

- EfficientNet models achieve state-of-the-art accuracy with significantly fewer parameters and FLOPs.

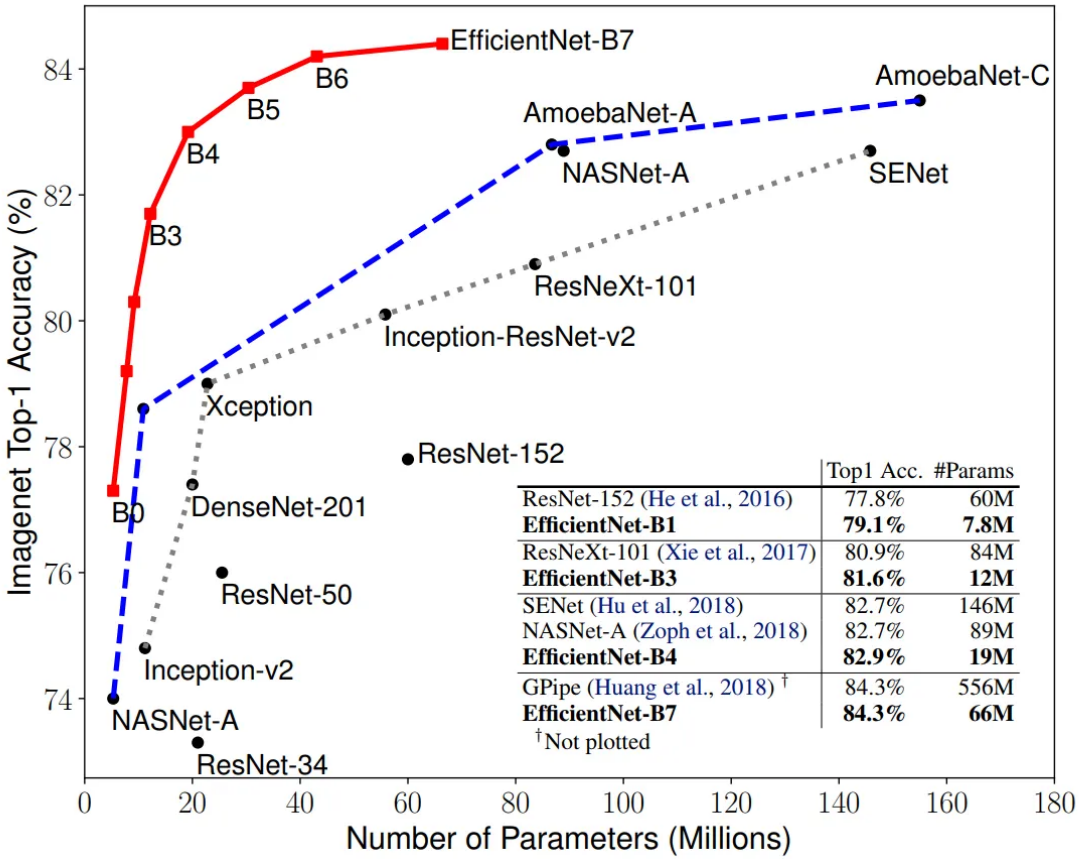
EfficientNet

- EfficientNet models achieve state-of-the-art accuracy with significantly fewer parameters and FLOPs.
- EfficientNet-B7 reaches 84.4% Top-1 and 97.3% Top-5 accuracy on ImageNet.

EfficientNet

- EfficientNet models achieve state-of-the-art accuracy with significantly fewer parameters and FLOPs.
- EfficientNet-B7 reaches 84.4% Top-1 and 97.3% Top-5 accuracy on ImageNet.
- More efficient than previous CNN models—8.4x smaller and 6.1x faster than competitors.

EfficientNet Performance



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

Figure 5: EfficientNet Performance on Imagenet.

Motivation for MobileNets

- **Why MobileNets?**
  - Small-sized models are crucial for mobile and embedded devices.
  - MobileNets reduce computational cost and memory usage while maintaining good accuracy.
- **Key Idea:**
  - Use **depthwise-separable convolutions** to significantly reduce computation compared to standard convolutions.

Computational Cost of Convolutions

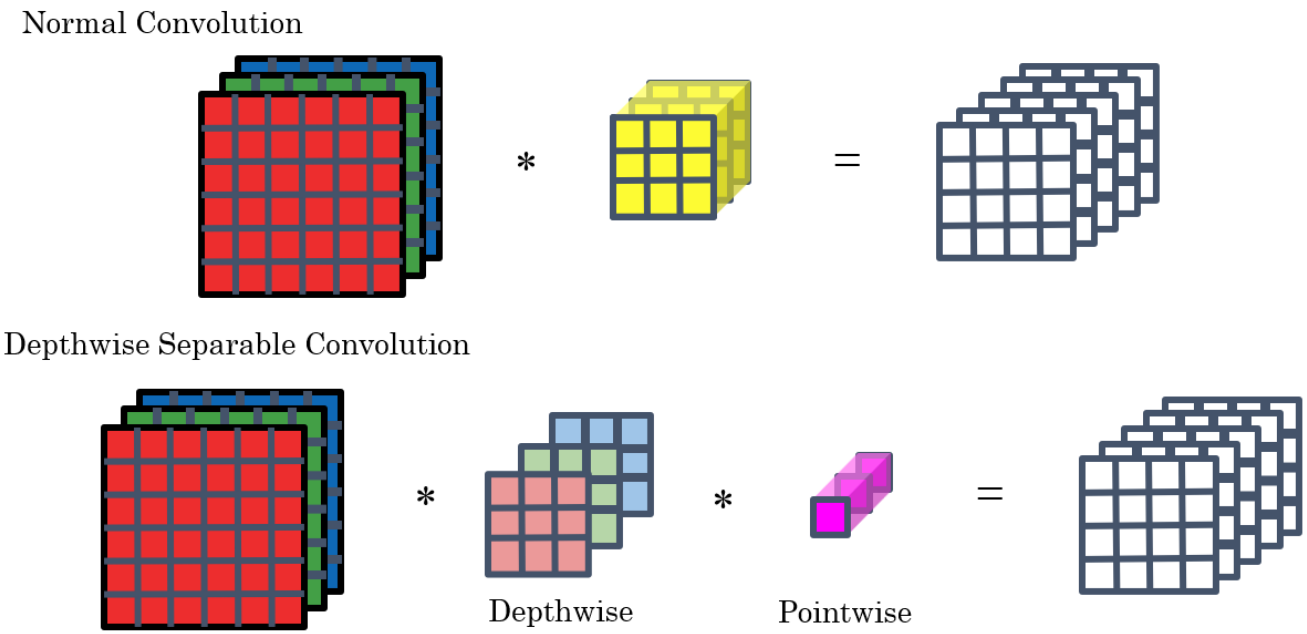
- **Computational cost of standard convolution:**

$$\text{Cost} = \#\text{ filter params} \times \#\text{ filter positions} \times \#\text{ filters}$$

- Filters operate on all input channels, increasing computation significantly.

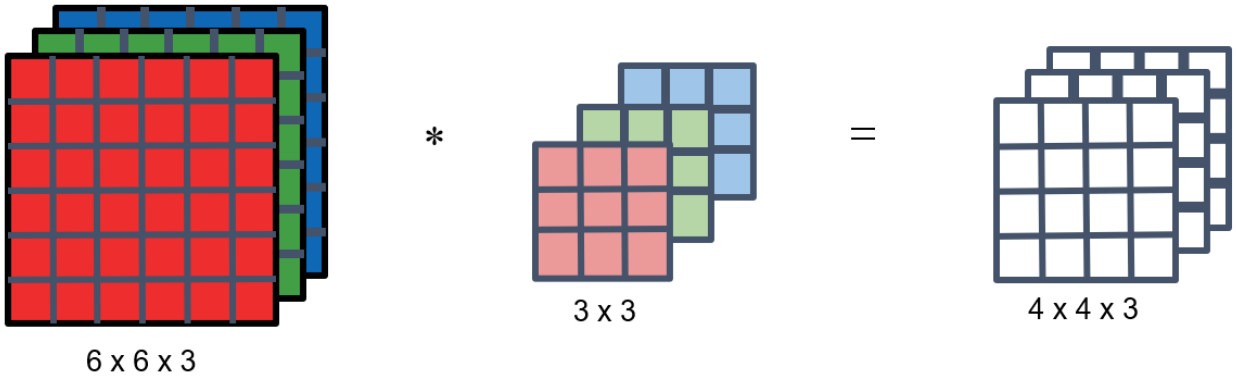
Depthwise-Separable Convolutions

- Split standard convolution into two steps:
  - **Depthwise Convolution:** Applies a single filter per input channel.
  - **Pointwise Convolution:** Combines outputs from depthwise convolution.
- **Key Benefit:** Reduces computational cost significantly compared to standard convolution.



Depthwise Convolution

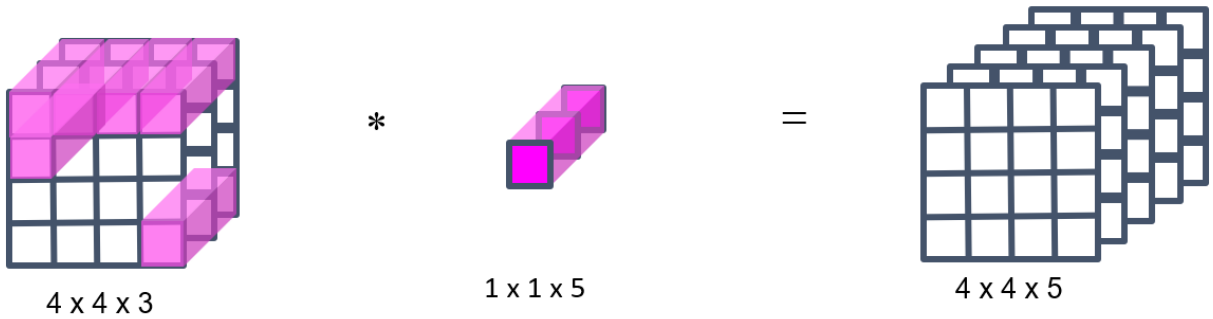
- Operates on each input channel separately.



Computational cost = #filter params x # filter positions x # of filters

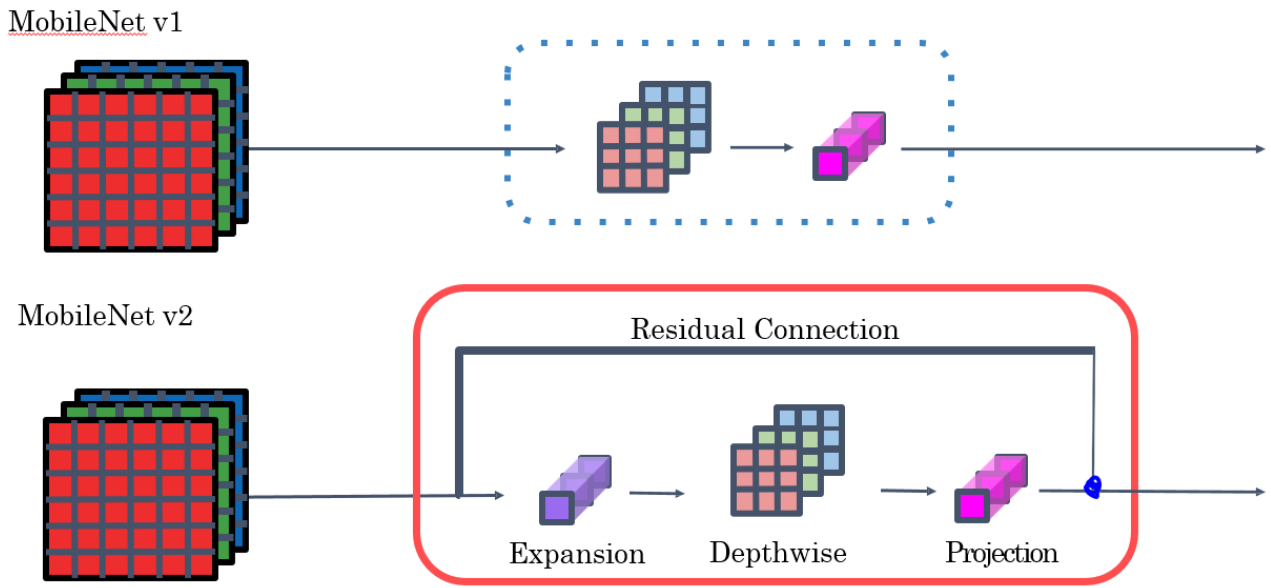
Pointwise Convolution

- Combines outputs from depthwise convolution using 1×1 convolutions (mixes channels).



Computational cost = #filter params x # filter positions x # of filters

- **MobileNet v2:**
  - Adds **residual connections**.
  - Introduces:
    - **Expansion step:** Expands input dimensions before depthwise convolution.
    - **Projection step:** Reduces dimensions after processing.



Concept:

- A technique where a pre-trained model is used as the starting point for a new task.
- It leverages the knowledge gained from a previous task to improve performance on a new, often related task.
- This is particularly useful when the new task has limited data.
- Commonly used in computer vision tasks, where models like VGG, ResNet, and Inception are pre-trained on large datasets like ImageNet.
- The pre-trained model can be fine-tuned on the new dataset by:
  - Freezing some layers and training others.
  - Replacing the final classification layer with a new one specific to the new task.
- Significantly reduce training time and improve model performance.

Use cases:

- Image classification
- Object detection
- Semantic segmentation
- Natural language processing

Listing 9: Code snippet (PyTorch)

```
import torch.nn as nn
from torchvision.models import resnet50

base = resnet50(pretrained=True)
for param in base.parameters(): param.requires_grad = False
base.fc = nn.Linear(base.fc.in_features, num_classes)
```

Accuracy:

- The ratio of correctly predicted instances to the total instances.
- It is a common metric for classification tasks.
- Formula:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- Where:
  - TP: True Positives
  - TN: True Negatives
  - FP: False Positives
  - FN: False Negatives
- Limitations:
  - It can be misleading in imbalanced datasets.
  - It does not provide information about the types of errors made.

Precision:

- The ratio of correctly predicted positive instances to the total predicted positive instances.
- Formula:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- It indicates how many of the predicted positive instances were actually positive.
- High precision means fewer false positives.
- Limitations:
  - It does not consider false negatives.
  - It can be misleading in imbalanced datasets.

Recall:

- The ratio of correctly predicted positive instances to the total actual positive instances.
- Formula:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- It indicates how many of the actual positive instances were predicted correctly.
- High recall means fewer false negatives.
- Limitations:
  - It does not consider false positives.
  - It can be misleading in imbalanced datasets.

F1 Score:

- The harmonic mean of precision and recall.
- Formula:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- It provides a balance between precision and recall.
- Useful for imbalanced datasets where one class is more important than the other.
- Limitations:
  - It can be misleading if the precision and recall are very different.
  - It does not provide information about the types of errors made.

CNN - Evaluation Metrics (cont.)

Confusion Matrix:

- A table that summarizes the performance of a classification model.
- It shows the true positive, true negative, false positive, and false negative counts.
- It provides a detailed breakdown of the model's performance.
- Useful for understanding the types of errors made by the model.
- Can be used to calculate other metrics like accuracy, precision, recall, and F1 score.
- Example:

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

- Where:
  - TP: True Positives
  - TN: True Negatives
  - FP: False Positives
  - FN: False Negatives

CNN - Evaluation Metrics (cont.)

mAP:

- mAP is calculated by averaging the average precision (AP) across all classes.
- AP is calculated by plotting the precision-recall curve and computing the area under the curve.
- mAP is particularly useful for evaluating models on datasets with multiple classes.
- It provides a comprehensive measure of the model's performance across all classes.
- Example:

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i$$

- Where:
  - N is the number of classes.
  - AP_i is the average precision for class i.
- mAP can be calculated at different IoU thresholds (e.g., 0.5, 0.75) to evaluate the model's performance at different levels of overlap.

CNN - Evaluation Metrics (cont.)

Listing 10: Code snippet (PyTorch)

```
from torcheval.metrics.functional import
    multiclass_precision, multiclass_recall,
    multiclass_f1_score

p = multiclass_precision(preds, labels)
r = multiclass_recall(preds, labels)
f1= multiclass_f1_score(preds, labels)
```

CNN - Further Reading

Books:

- **Deep Learning** by Ian Goodfellow, Yoshua Bengio, and Aaron Courville
- **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow** by Aurélien Géron
- **Pattern Recognition and Machine Learning** by Christopher Bishop
- **Deep Learning for Computer Vision with Python** by Adrian Rosebrock

CNN - Further Reading (cont.)

Online Courses:

- **Deep Learning Specialization** by Andrew Ng (Coursera)
- **Convolutional Neural Networks for Visual Recognition** by Fei-Fei Li (Stanford University)
- **Practical Deep Learning for Coders** by Jeremy Howard (fast.ai)
- **Deep Learning with PyTorch** by Facebook AI Research

CNN - Further Reading (cont.)

Research Papers:

- **ImageNet Classification with Deep Convolutional Neural Networks** by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton
- **Very Deep Convolutional Networks for Large-Scale Image Recognition** by K. Simonyan and A. Zisserman
- **Deep Residual Learning for Image Recognition** by Kaiming He et al.
- **Densely Connected Convolutional Networks** by Gao Huang et al.

CNN - Further Reading (cont.)

Websites and Blogs:

- **Towards Data Science** (Medium)
- **Distill.pub** (Research and visualization)
- **Kaggle** (Datasets and competitions)
- **Papers with Code** (Research papers with code implementations)

CNN - Further Reading (cont.)

YouTube Channels:

- **3Blue1Brown** (Mathematics and deep learning)
- **Two Minute Papers** (Research summaries)
- **Sentdex** (Python programming and machine learning)
- **StatQuest with Josh Starmer** (Statistics and machine learning)

CNN - Further Reading (cont.)

GitHub Repositories:

- **TensorFlow Models** (TensorFlow implementations)
- **PyTorch Examples** (PyTorch implementations)
- **fastai** (High-level library for deep learning)
- **Keras** (High-level neural networks API)

CNN - Further Reading (cont.)

Conferences:

- **NeurIPS** (Neural Information Processing Systems)
- **CVPR** (Computer Vision and Pattern Recognition)
- **ICML** (International Conference on Machine Learning)
- **ICLR** (International Conference on Learning Representations)

Credits

Dr. Prashant Aparajeya

Computer Vision Scientist — Director(AISimply Ltd)

p.aparajeya@aisimply.uk

This project benefited from external collaboration, and we acknowledge their contribution with gratitude.