Computer Vision

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

KAUST Academy
King Abdullah University of Science and Technology

February 17, 2025

Table of Contents

Learning Outcomes

- Understand the basic concepts of Computer Vision and its real-world applications.
- Describe how images are represented and processed in a computer.
- Explain the fundamental building blocks of Convolutional Neural Networks (CNNs).
- Differentiate between popular CNN architectures (AlexNet, VGG, InceptionNet, ResNet, EfficientNet, and MobileNet) and their key innovations.

Computer Vision

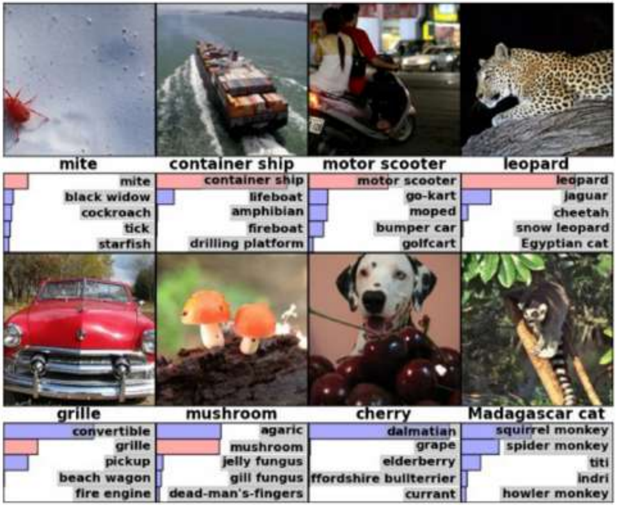
Building artificial systems that process, perceive, and reason about visual data

Computer Vision is Everywhere

[illegible]

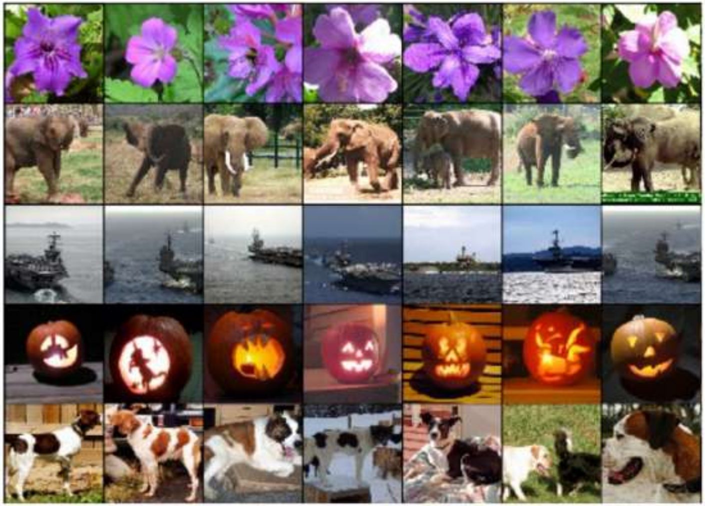
Some Applications

Image Classification



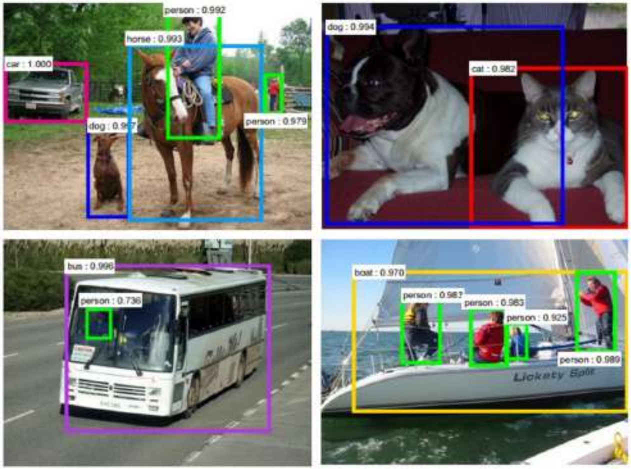
Some Applications (cont.)

Image Retrieval



Some Applications (cont.)

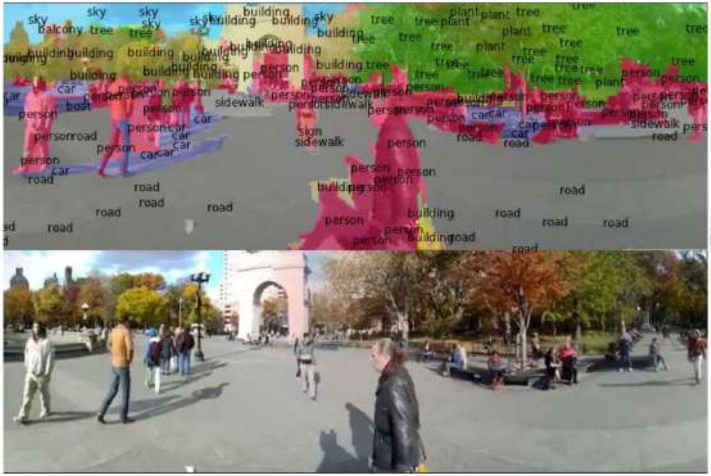
Object Detection



Ren, He, Girshick, and Sun, 2015

Some Applications (cont.)

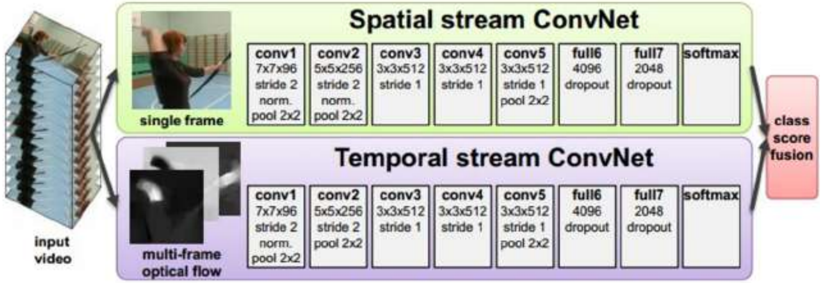
Image Segmentation



Fabaret et al, 2012

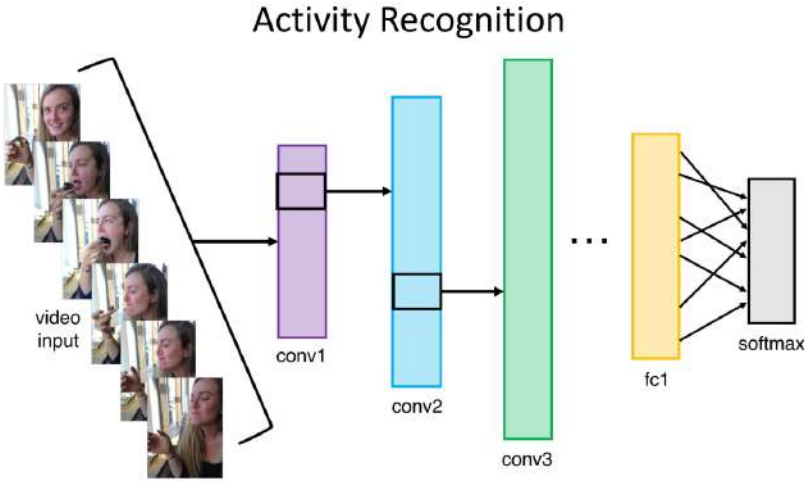
Some Applications (cont.)

Video Classification



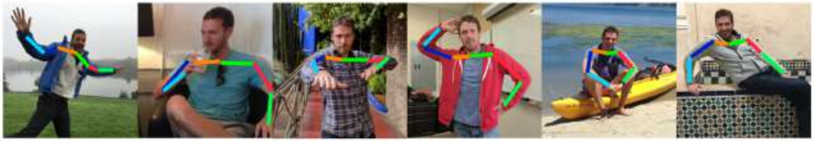
Simonyan et al, 2014

Some Applications (cont.)

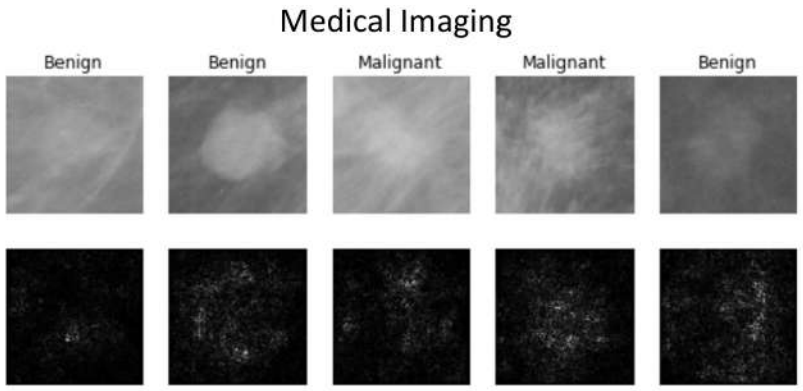


Some Applications (cont.)

Pose Recognition (Toshev and Szegedy, 2014)



Some Applications (cont.)



Some Applications (cont.)

Image Captioning

Vinyals et al, 2015

Karpathy and Fei-Fei, 2015

A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

A woman is holding a cat in her hand

A man riding a wave on top of a surfboard

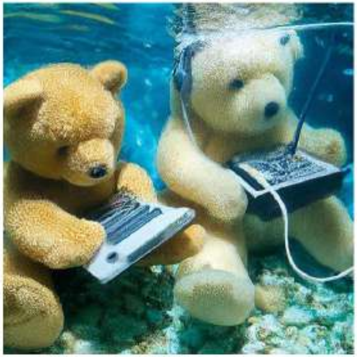
A cat sitting on a suitcase on the floor

A woman standing on a beach holding a surfboard

[illegible]

Some Applications (cont.)

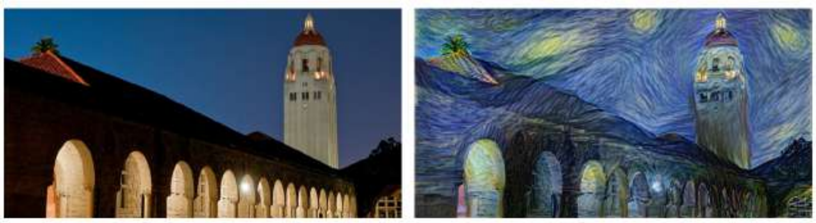
Image Generation



"Teddy bears working on new AI research underwater with 1990s technology"

DALL-E 2

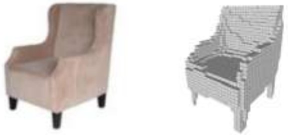
Some Applications (cont.)



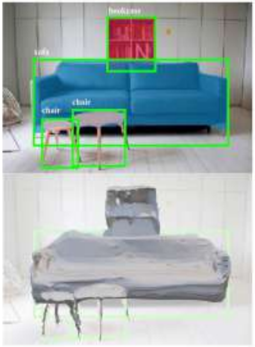
Style Transfer

Some Applications (cont.)

3D Vision


Choy et al., 3D-R2N2: Recurrent Reconstruction Neural Network (2016)

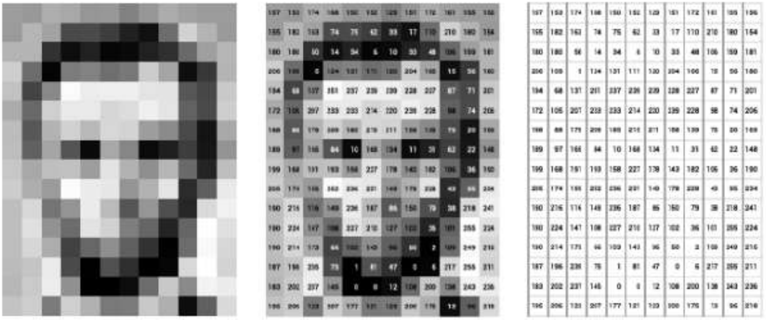
Zhou et al., 3D Shape Generation and Completion through Point-Voxel Diffusion (2021)



Gkioxari et al., "Mesh R-CNN", ICCV 2019

How to represent an image?

- Images are represented as Matrices with elements in $[0, 255]$
- Grayscale images have one channel.

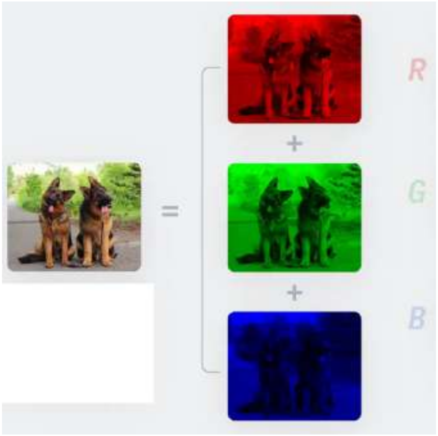


157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

[0] https://www.v7labs.com/blog/image-recognition-guide

How to represent an image?

- RGB images have 3 channels.



How can we build vision models?

- So far, we've worked with tabular data, where each sample consists of structured features. We processed this data using Fully Connected Neural Networks.

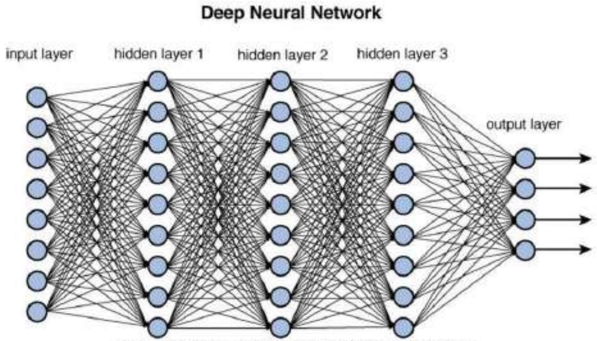


Figure 12.2 Deep network architecture with multiple layers.

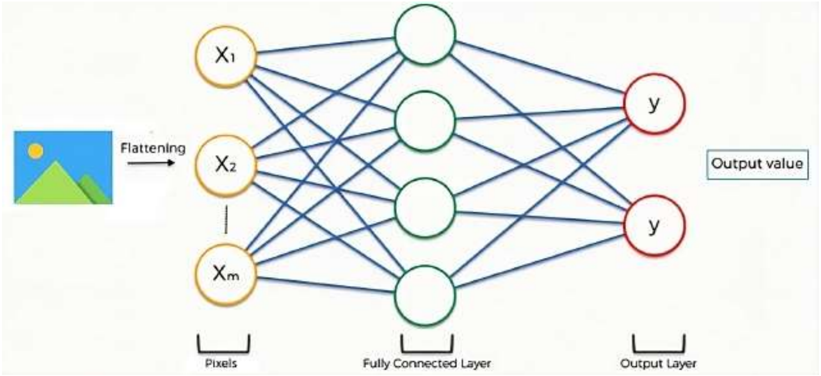
- But can we apply the same approach to images?

How can we build vision models?

- Let’s try.

How can we build vision models?

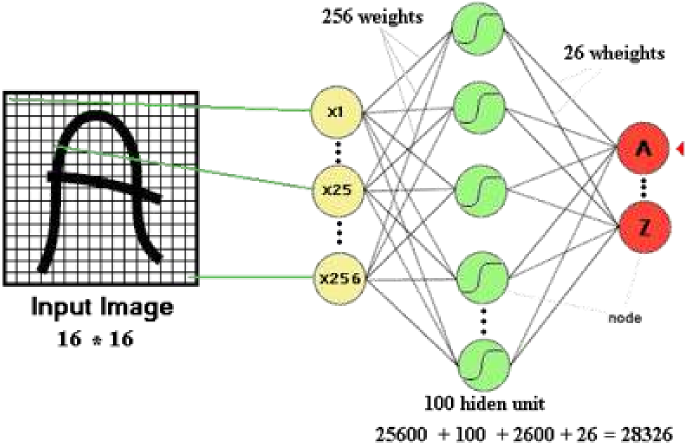
- Let's consider each pixel as a feature.



- This should work! But...

Drawbacks of Fully-Connected Neural Networks

- The number of trainable parameters becomes extremely large



Drawbacks of Fully-Connected Neural Networks (cont.)

- Little or no invariance to shifting, scaling, and other forms of distortion
- In other words, the Fully connected NN cannot recognize that the two images (original and shifted) represent the same object (letter "A").

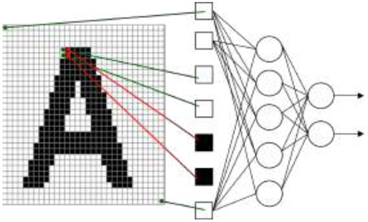


Figure 2: Original "A" character.

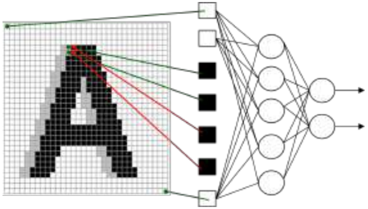
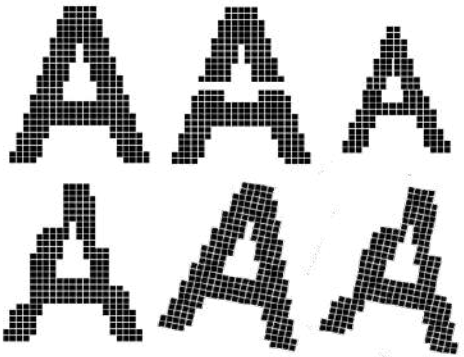


Figure 3: Shifted "A" character.

Drawbacks of Fully-Connected Neural Networks (cont.)

- The topology of the input data is completely ignored (it treats pixels as independent features rather than structured patterns.)
- For a 32×32 image, we have
  - Black and white patterns: $2^{32*32} = 2^{1024}$
  - Grayscale patterns: $256^{32*32} = 256^{1024}$



What do we need?

- We need a model that can:
  - **Find Patterns in Images**: Recognize small features like edges and shapes in different parts of the image.
  - **Work Efficiently**: Avoid looking at every pixel separately by focusing on groups of pixels together.
  - **Handle Changes**: Still recognize the same object even if it moves, rotates, or looks slightly different.

What do we need?

- We need a model that can:
  - **Find Patterns in Images**: Recognize small features like edges and shapes in different parts of the image.
  - **Work Efficiently**: Avoid looking at every pixel separately by focusing on groups of pixels together.
  - **Handle Changes**: Still recognize the same object even if it moves, rotates, or looks slightly different.
  - We need **Convolutional Neural Networks (CNNs)!**

Convolutional Neural Networks (CNNs)

- CNNs are neural networks designed for image processing and consist of two main parts:
  - **Feature Extractor**: Automatically learns patterns (features) such as edges, textures, and shapes from the image.
  - **Classifier**: The extracted features are flattened into a vector and passed to a standard neural network (like the ones we used before) for classification.

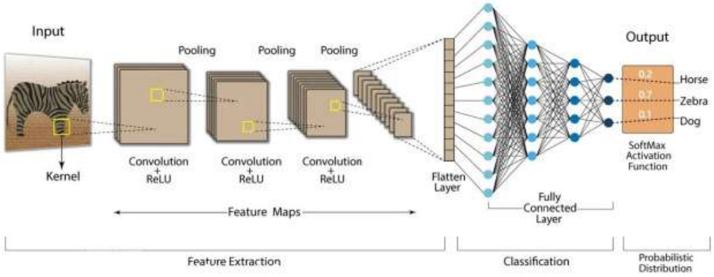


Figure 4: Convolutional Neural Network

Convolutional Neural Networks (CNNs)

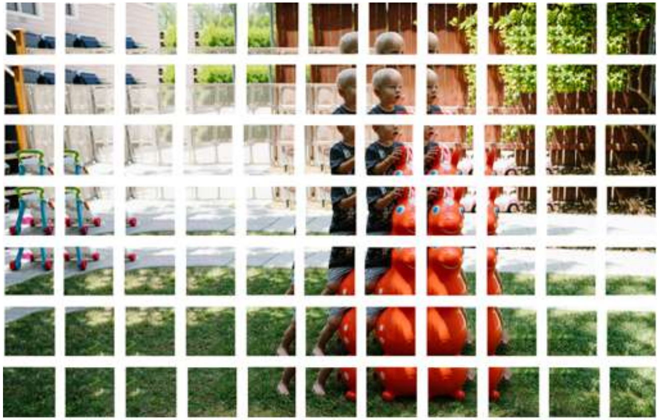
- The feature extractor consists of three essential components:
  - **Convolution Layers**: Detect local patterns such as edges, textures, and shapes by applying filters to the image.
  - **Activation Function (ReLU)**: Adds non-linearity.
  - **Pooling Layers**: Reduce the spatial size of extracted features.
- Let's start with the convolution layer.

How Convolution Works?

- Let's consider this image.

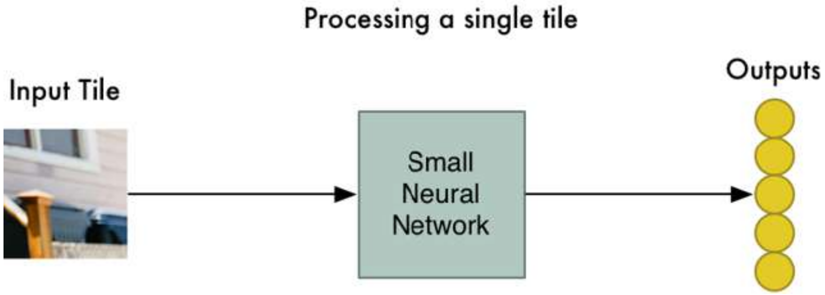
How Convolution Works? (cont.)

1. The image is divided into small overlapping tiles (regions).



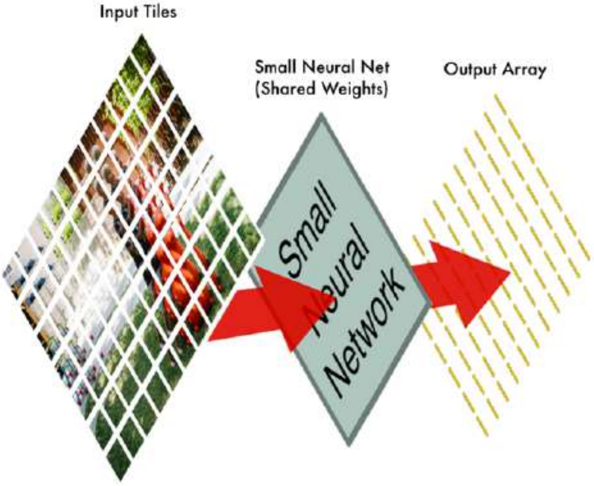
How Convolution Works? (cont.)

2. We process each tile using the weights matrix of a small neural network. We call this matrix a **kernel** or **filter**.



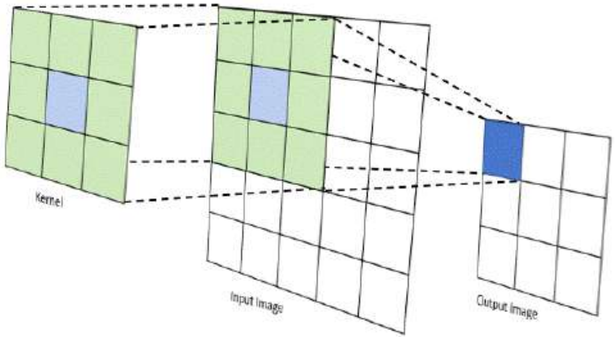
How Convolution Works? (cont.)

3. Finally, the filter slides across the image (with the same weights) and processes all the tiles, creating an output **feature map**.



How Convolution Works? (cont.)

- What we did in the last step is called the **Convolution Operation**.
- We convolved the kernel with the image, which means sliding the kernel over the entire image and computing the dot product between the kernel and small regions of the image at each step.



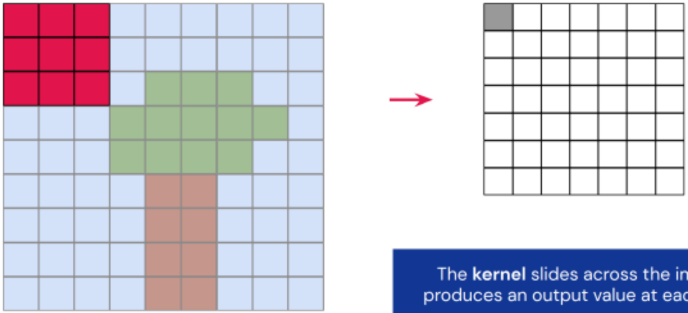
How Convolution Works? (cont.)

- This can be represented mathematically by:

$$z = W * x_{i,j} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} W_{ab} X_{(i+a)(j+b)}$$

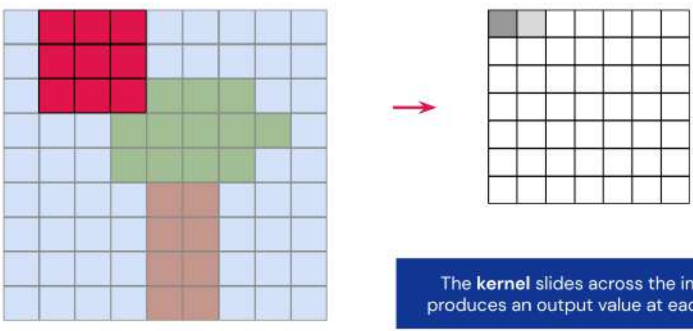
- z: Output of the convolution at (i, j).
- W: Filter (kernel) matrix of size $m \times n$.
- $x_{i,j}$: Input value at position (i, j).
- W_{ab}: Weight of the filter at (a, b).
- $X_{(i+a)(j+b)}$: Input value in the small region at $(i + a, j + b)$.

How Convolution Works? (cont.)



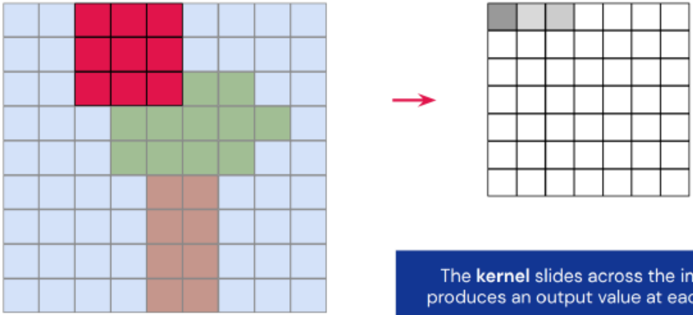
The **kernel** slides across the image and produces an output value at each position

How Convolution Works? (cont.)



The **kernel** slides across the image and produces an output value at each position

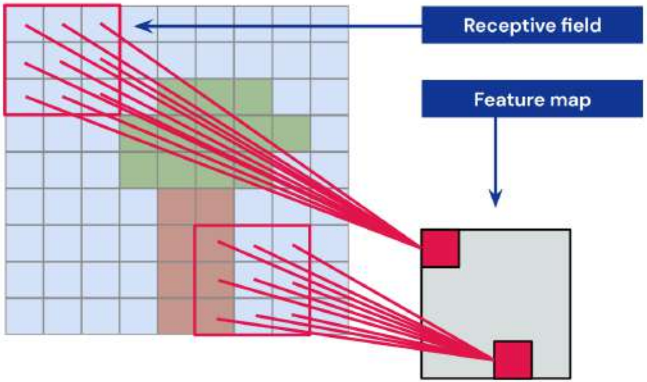
How Convolution Works? (cont.)



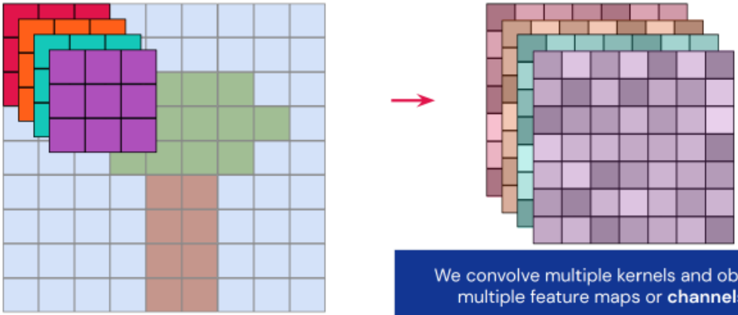
The **kernel** slides across the image and produces an output value at each position

How Convolution Works? (cont.)

- **Receptive Field**: The region of the input image that the kernel operates on at each step.
- **Feature Map**: The features extracted from the input image.



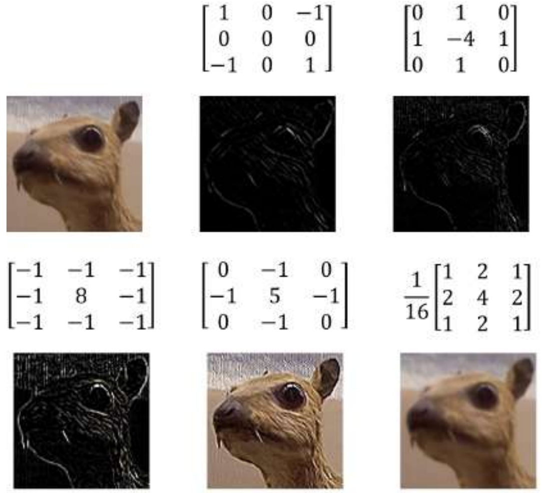
How Convolution Works? (cont.)



We convolve multiple kernels and obtain multiple feature maps or **channels**

How Convolution Works? (cont.)

- **Filters Example**: Different filters detect patterns like edges, textures, or smoothness, producing corresponding feature maps.

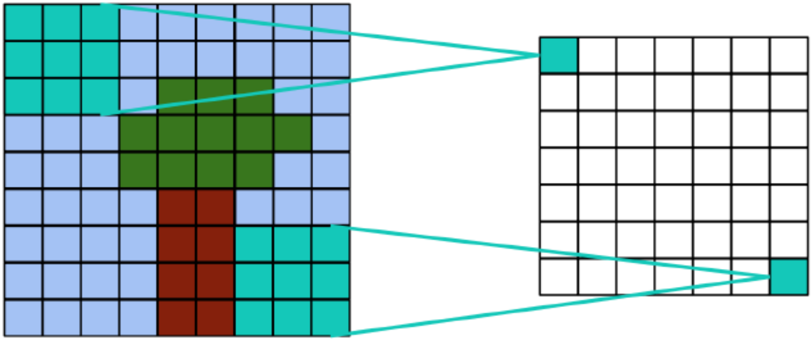


$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

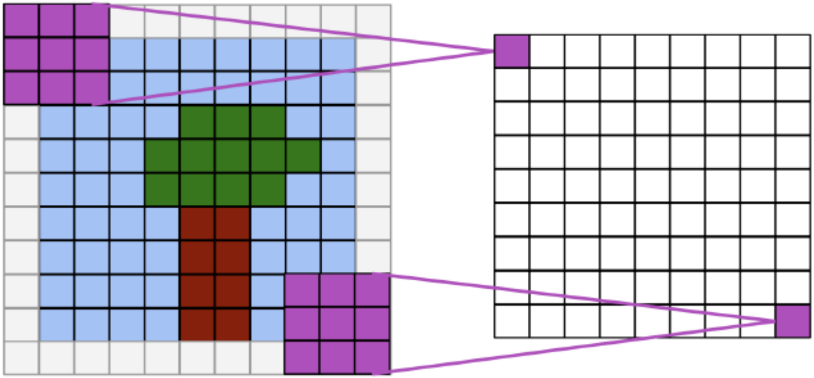
Controlling the Convolution Process

- Applying Convolution as such reduces the size of the borders.
- Sometimes this is not desirable.



Controlling the Convolution Process (cont.)

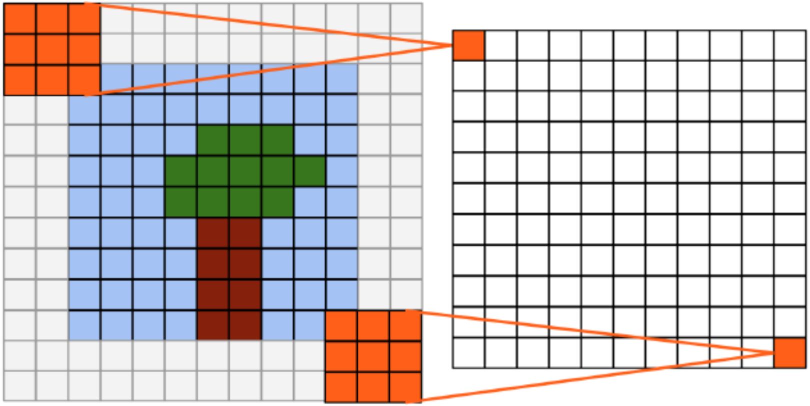
- Solution: We can use **Padding**, which pads the border with zeros.
- it has two types:
   1. **Same Convolution**: Padding is added so the output size equals the input size.



Controlling the Convolution Process (cont.)

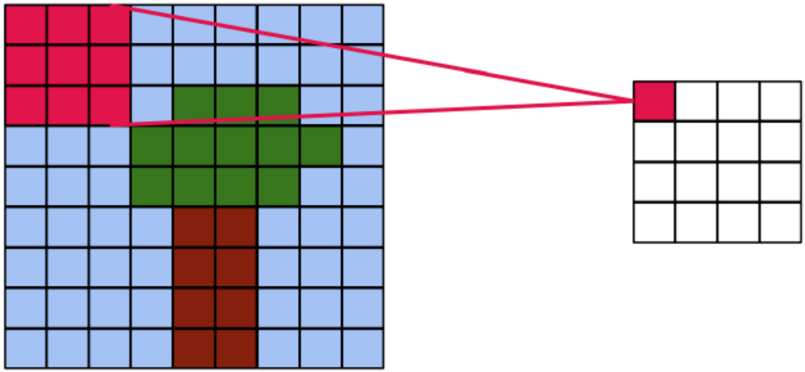
2. **Full Convolution**: Padding is added so the kernel covers the entire input, including edges.

   output size = input size + kernel size - 1



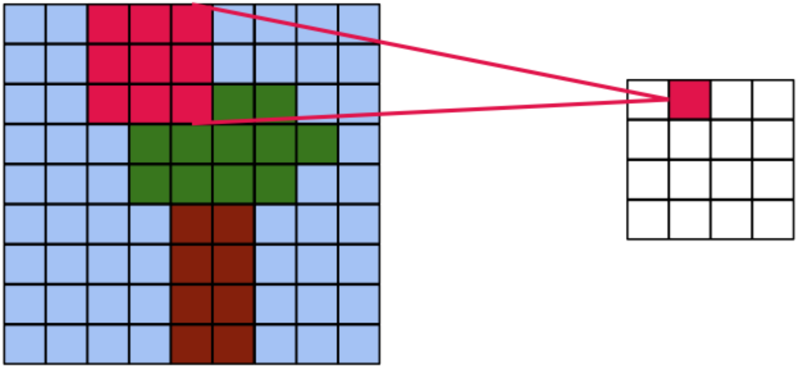
Controlling the Convolution Process (cont.)

- **Strided Convolution**: Kernel slides along the image with a step > 1
- Larger strides reduce output size.



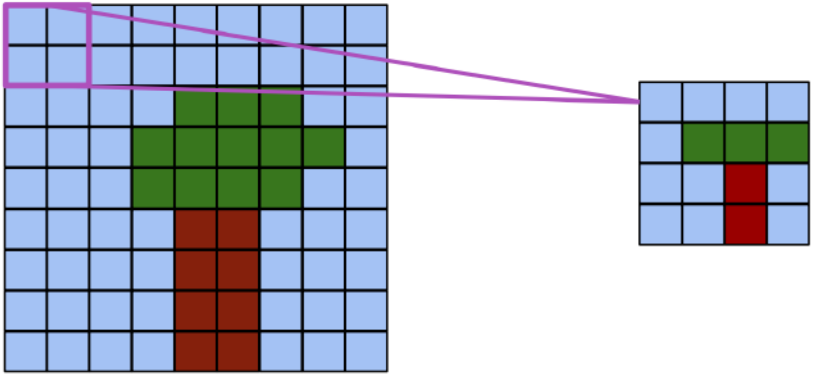
Controlling the Convolution Process (cont.)

- **Strided Convolution**: Kernel slides along the image with a step > 1
- Larger strides reduce output size.

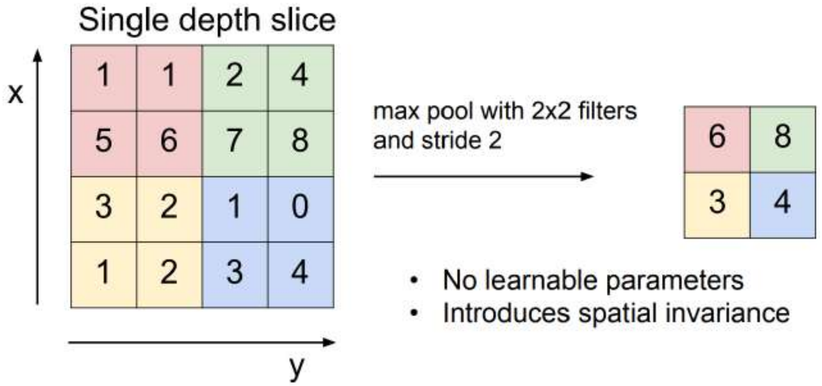


Controlling the Convolution Process (cont.)

- **Pooling**: Compute mean or max over small windows to reduce resolution and extract more general features.



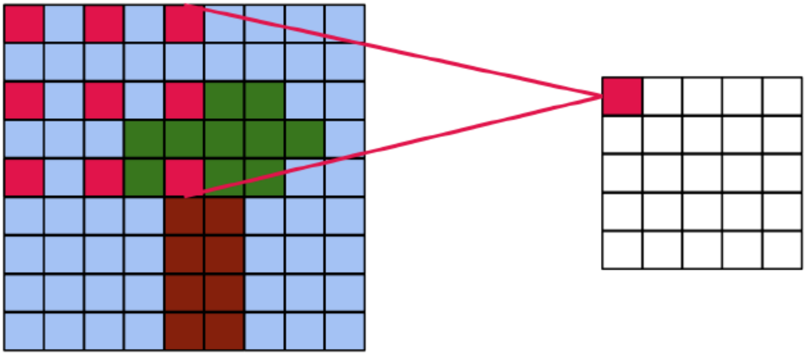
Controlling the Convolution Process (cont.)



- No learnable parameters
- Introduces spatial invariance

Controlling the Convolution Process (cont.)

- **Dilated Convolution**: Kernel is spread out, step > 1 between kernel elements.
- It expands the receptive field without increasing the number of parameters, which makes it efficient.



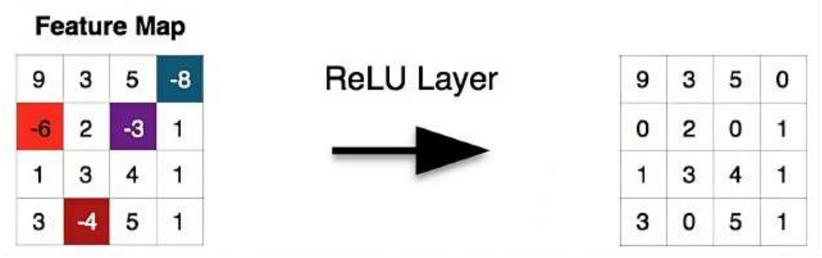
CNN Output size

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

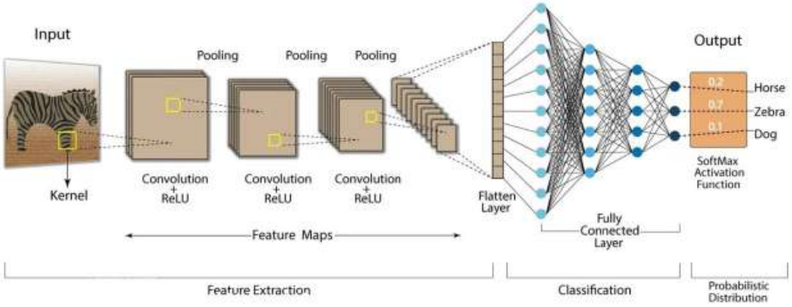
n_{in}	: number of input features
n_{out}	: number of output features
k	: convolution kernel size
p	: convolution padding size
s	: convolution stride size

Activation

- Just like Fully-Connected Neural Networks, we can apply an activation over convolutional layer outputs
- It helps break linearity
- For example, Rectified Linear Unit (ReLU): $\sigma(x) = max(0, x)$

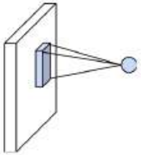


Convolutional Neural Networks

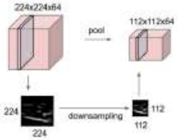


Components of a CNN

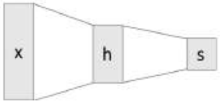
Convolution Layers



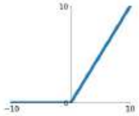
Pooling Layers



Fully-Connected Layers



Activation Function



Normalization

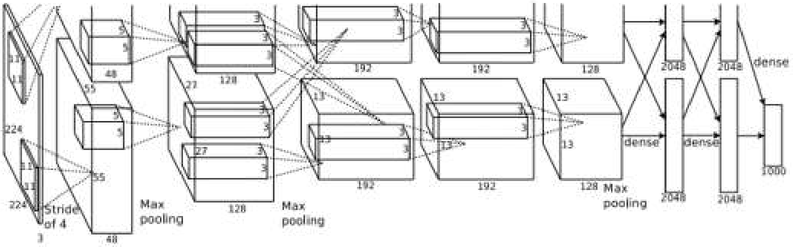
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Most Notable CNN-based Architectures

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

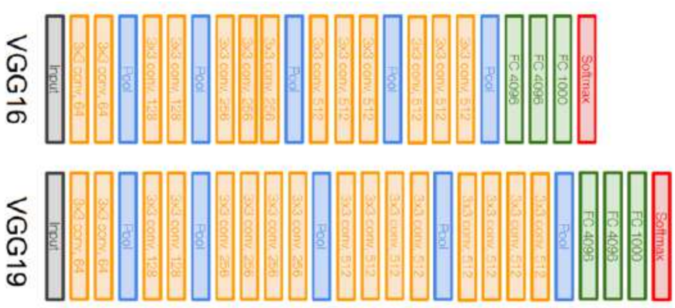
AlexNet

- 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



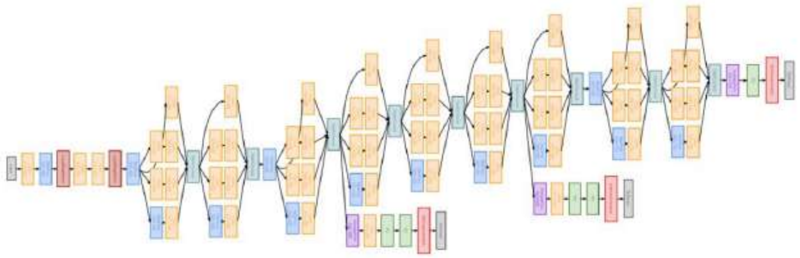
VGGNet

- **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.



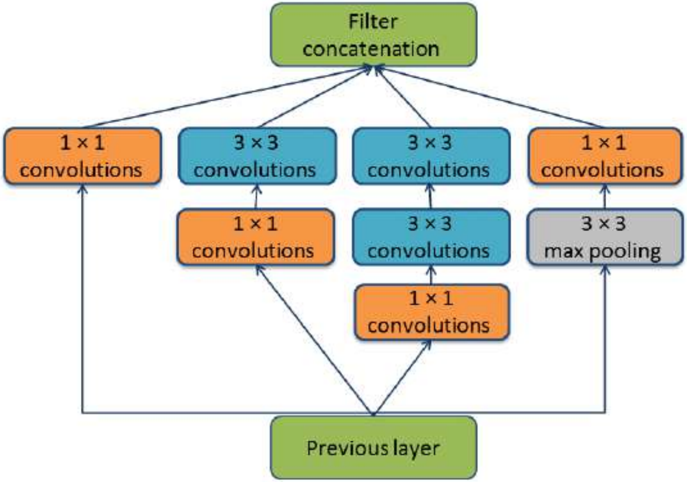
InceptionNet

- 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.

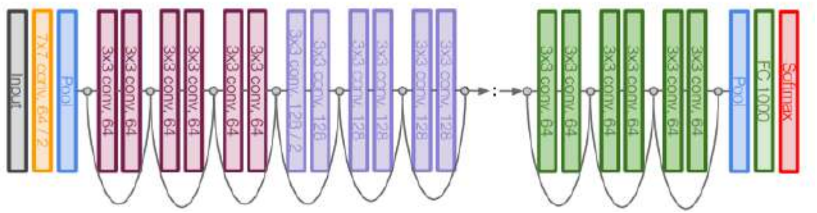


ResNet: Motivation

- **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.

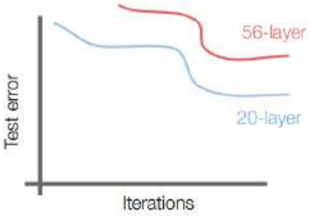


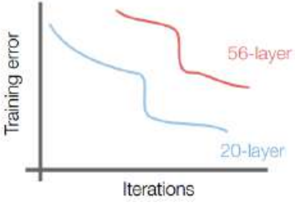
ResNet

- What happens when we continue stacking deeper layers on a ”plain” convolutional neural network?

ResNet

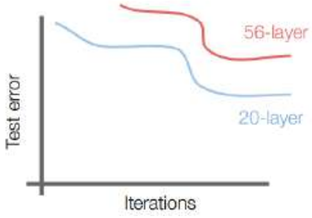
- What happens when we continue stacking deeper layers on a "plain" convolutional neural network?

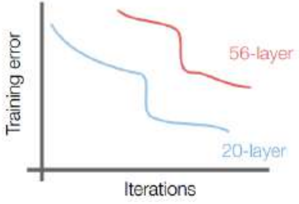




ResNet

- What happens when we continue stacking deeper layers on a "plain" convolutional neural network?

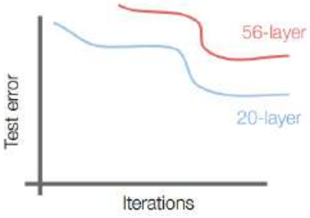


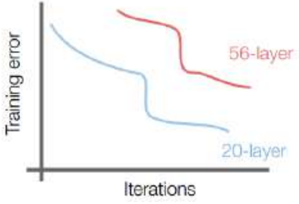


- 56-layer model performs worse on both test and training error

ResNet

- What happens when we continue stacking deeper layers on a ”plain” convolutional neural network?





- 56-layer model performs worse on both test and training error
- The deeper model performs worse, but it’s not caused by overfitting!

ResNet

- **Fact:** Deep models have more representation power (more parameters) than shallower models.

ResNet

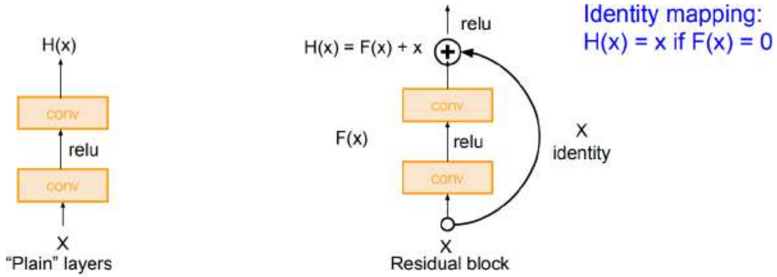
- **Fact:** Deep models have more representation power (more parameters) than shallower models.
- **Hypothesis:** The problem is an optimization problem, deeper models are harder to optimize

ResNet

- **Fact:** Deep models have more representation power (more parameters) than shallower models.
- **Hypothesis:** The problem is an optimization problem, deeper models are harder to optimize
- **Solution:** Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

ResNet

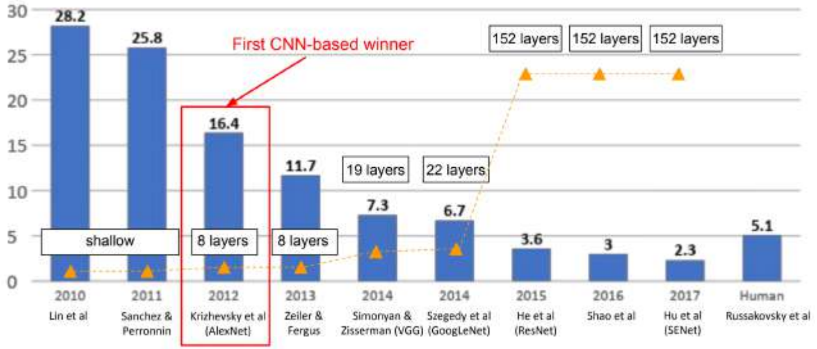
- **Fact:** Deep models have more representation power (more parameters) than shallower models.
- **Hypothesis:** The problem is an optimization problem, deeper models are harder to optimize
- **Solution:** Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



ImageNet

- The most extensive data for Image Classification
- 3 RGB channels from 0 to 255
- 14,197,122 images
- 1000 classes

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



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**.

EfficientNet

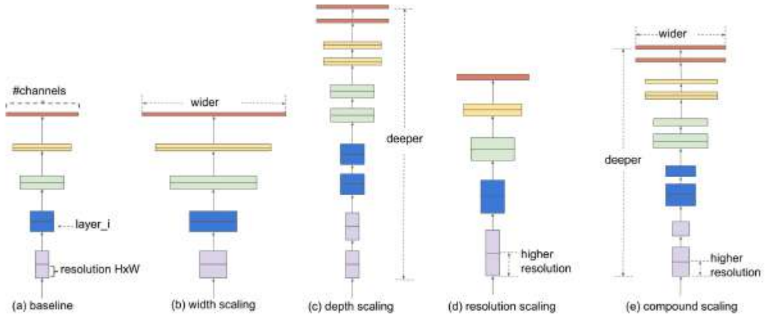


Figure 5: EfficientNet Scaling.

EfficientNet

- **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

- 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.

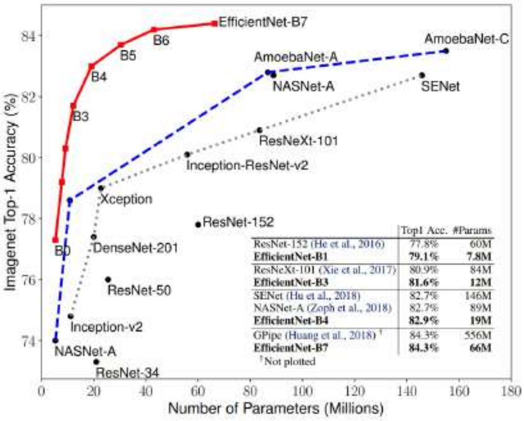
EfficientNet

- 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 6: EfficientNet Performance on Imagenet.

Motivation for MobileNets

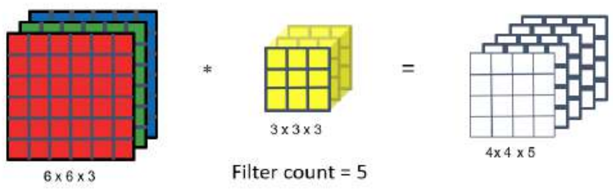
- **Why MobileNets?**
  - Small-sized models are crucial for mobile and embedded devices.
  - MobileNets reduce computational cost and memory usage while maintaining good accuracy.
- **Key Idea:**
  - Use **depthwise-separable convolutions** to significantly reduce computation compared to standard convolutions.

Computational Cost of Convolutions

- **Computational cost of standard convolution:**

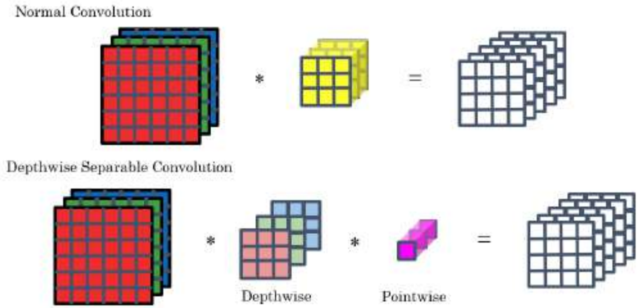
$$\text{Cost} = \#\ \text{filter params} \times \#\ \text{filter positions} \times \#\ \text{filters}$$

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



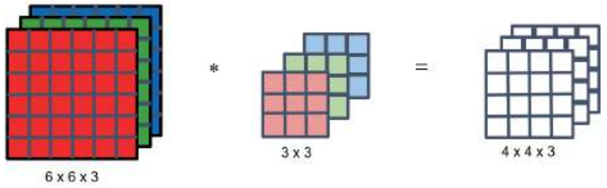
Depthwise-Separable Convolutions

- 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

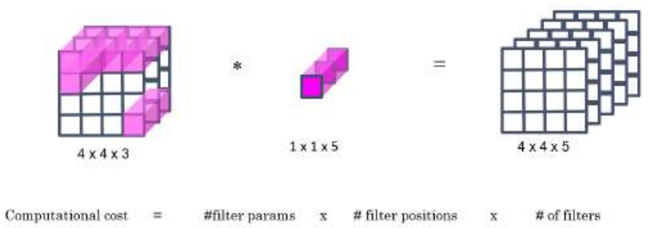
- Operates on each input channel separately.



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

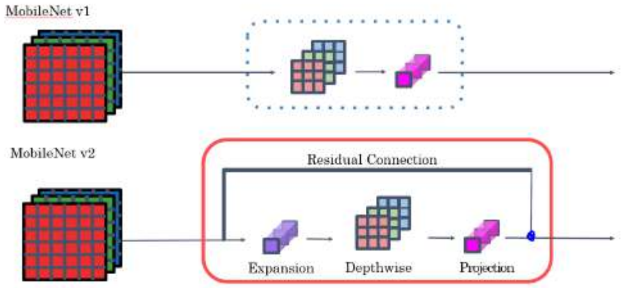
Pointwise Convolution

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


4 x 4 x 3
*
1 x 1 x 5
=
4 x 4 x 5
Computational cost = #filter params x # filter positions x # of filters

MobileNet Architectures

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



References

These slides have been adapted from

- Fei-Fei Li, Yunzhu Li & Ruohan Gao, Stanford CS231n: Deep Learning for Computer Vision
- Assaf Shocher, Shai Bagon, Meirav Galun & Tali Dekel, WAIC DL4CV Deep Learning for Computer Vision: Fundamentals and Applications
- Justin Johnson, UMich EECS 498.008/598.008: Deep Learning for Computer Vision
- Sander Dieleman, Deepmind: Deep Learning Lecture Series 2020