Convolutional Neural Networks

Filip and Ari

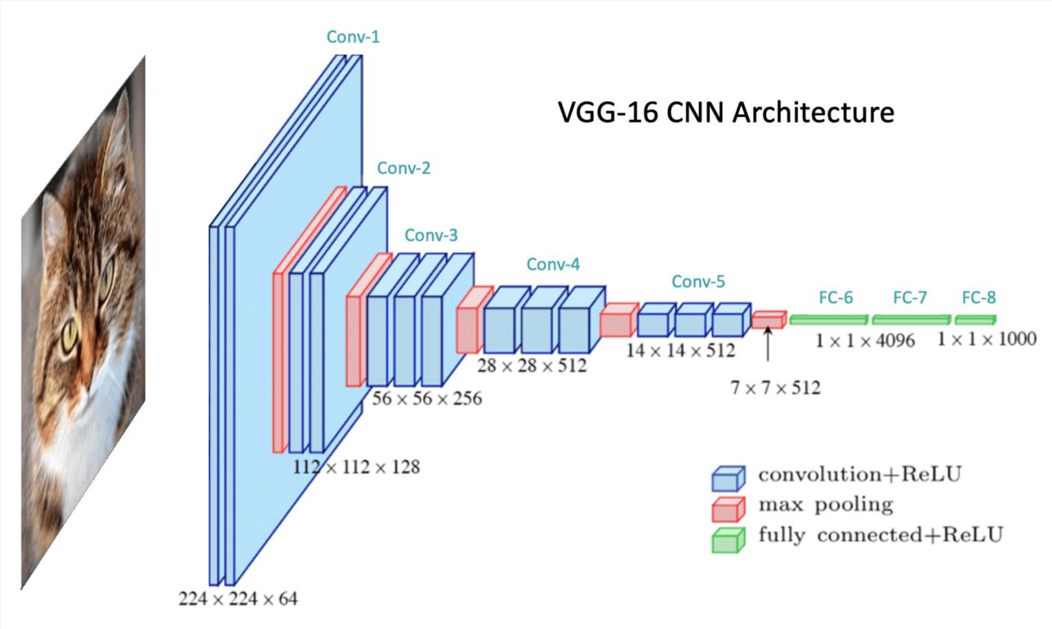
Common Applications

Computer Vision

- Image Classification
- Object Detection
- Face Recognition

Other Domains

- Image generation
- Signal Processing
- Medical Imaging



Why CNNs? Problem with images

Traditional Neural Networks (like MLPs)

Treat input as flat data - too many parameters for images (every pixel)

Lose spatial relationships

Like reading a book letter by letter vs understanding sentences

Traditional approaches to image recognition involved labor-intensive feature engineering

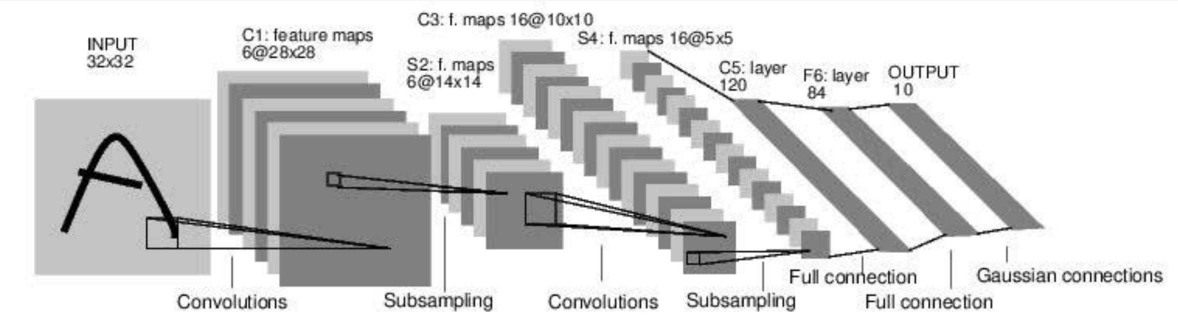
CNNs

Preserve spatial structure/location of features

Detect local patterns/things next to each other

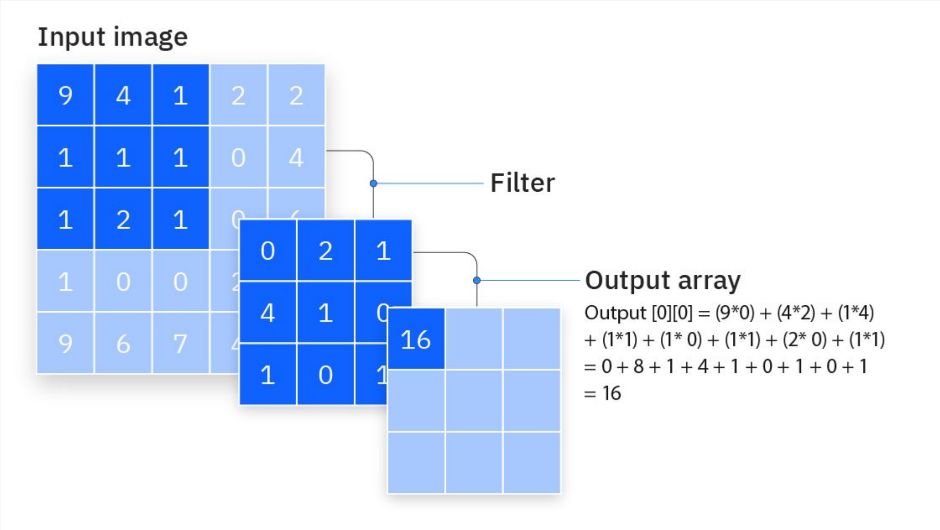
Parameter efficient / not every pixel is an input into the NN

CNNs learn features automatically from raw pixel data.

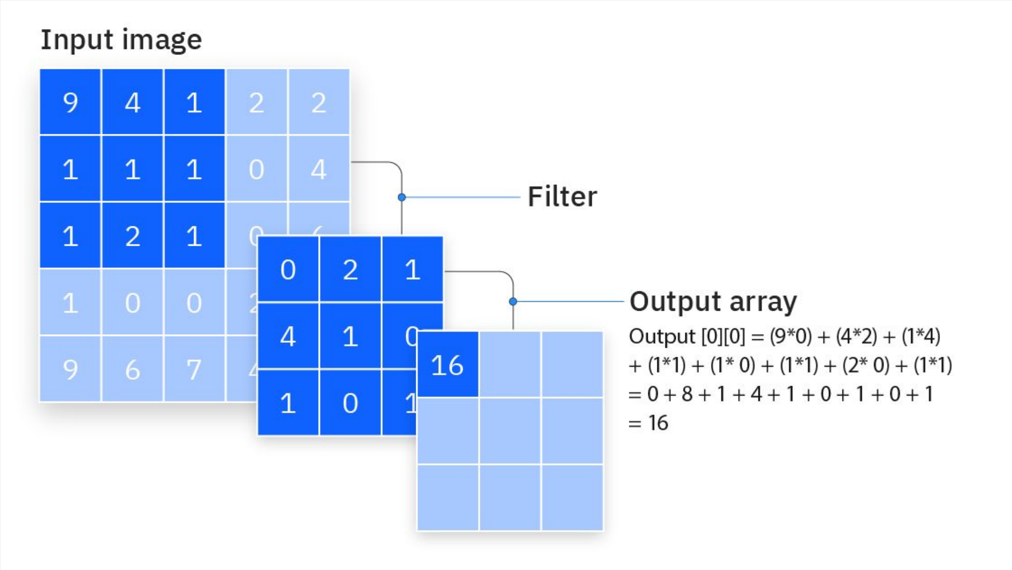


Core Components 1: Convolutional Layer

Convolution kernel/filter: A small matrix (e.g., 3×3) that slides over the image.
Operation: At each position, multiply the filter values with the pixel values, sum them up. High values mean the pattern was detected!
Result: A “feature map” highlighting specific features (edges, textures, etc.).



How Convolutions Work

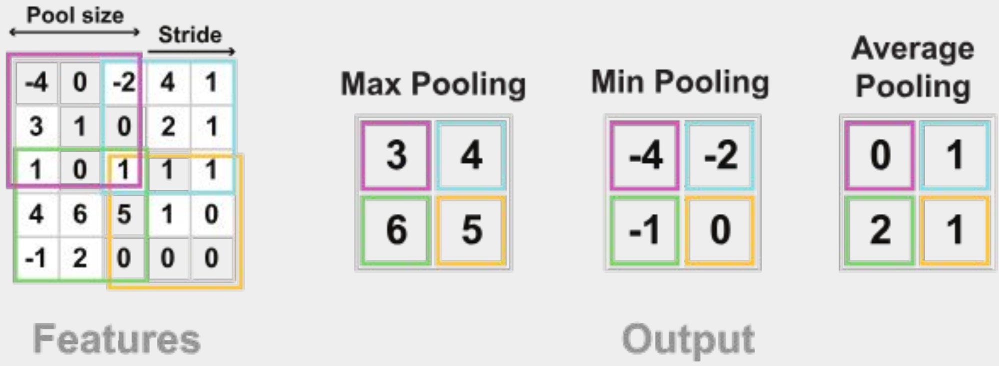


1. Small filter (kernel) moves across image
2. Element-wise multiplication and sum
3. Creates feature map highlighting patterns

```
Image Region     Filter        Result
[ 1 1 1 ]      [ 1  0 ]      [ 4 ]
[ 1 1 1 ]  x [ 0 -1 ]  =
[ 1 1 1 ]
```

Core Components 2: Pooling Layer

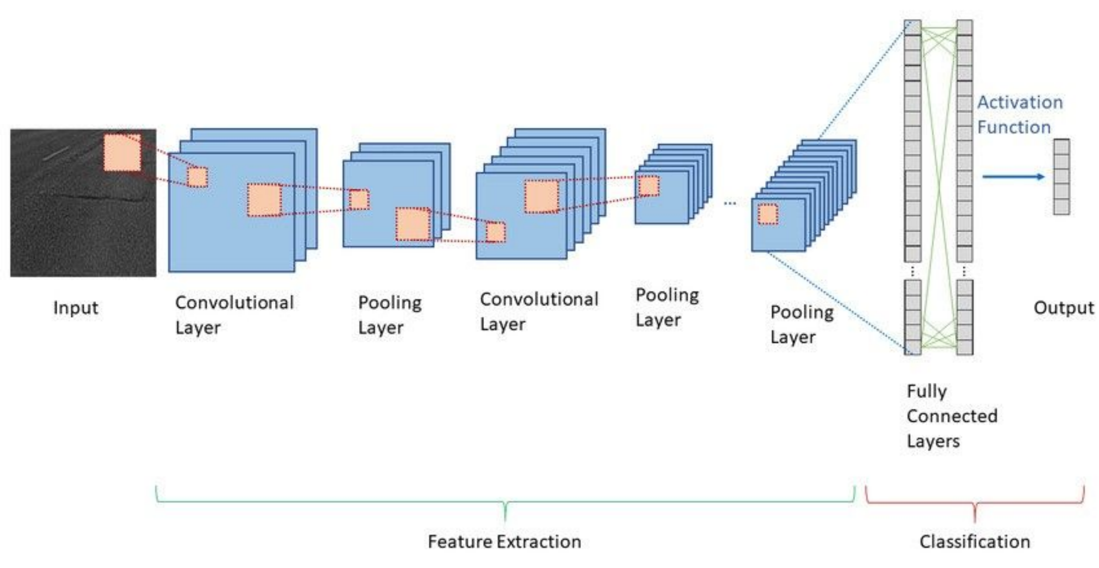
Purpose: Reduce dimension to decrease computational load and capture the most prominent features in each region.
Max Pooling (most common/strongest signal): Outputs the maximum value in each window (e.g., 2×2).
Average Pooling (less popular/blurring signals): Outputs the average of values in the window.
Adds translational invariance: small shifts in the image do not drastically change the feature maps.



Building a full CNN

Typical Layers:

a. **Convolution**: Detect low-level features (edges).
b. **Pooling**: Compress and generalize.
c. **Repeat**: Next conv layers detect higher-level features (eyes, wheels).
d. **Flatten**: Convert 2D feature maps to 1D for classification.
e. **Fully Connected Layers**: Final decision (e.g., “cat” vs. “dog”).



Visualization Tool:

https://adamharley.com/nn_vis/cnn/2d.html

Explainer Videos:

https://www.youtube.com/watch?v=pj9-rr1wDhM

https://www.youtube.com/watch?v=aircAruvnKk&t=43s

Business Application

Chess Predict!

In Class Exercise

Exercise 1: Simple Edge Detection

Here's a 4x4 input image. We'll use a 2x2 edge detection kernel.

Input Image:

```
1   1   0   0
1   1   0   0
0   0   1   1
0   0   1   1
```

Kernel (edge detection):

```
1   -1
```

```
-1   1
```

Solution:

0	0	0
0	2	0
0	0	0

Exercise 2: Vertical Line Detection

Input Image:

```
0   1   0   1
0   1   0   1
0   1   0   1
0   1   0   1
```

Kernel:

```
1   -1
1   -1
```

Solution:

-2	2	-2
-2	2	-2
-2	2	-2

Exercise 3: Max Pooling

Given the following 4x4 feature map, apply max pooling with a 2x2 window and stride of 2:

3	7	2	4
1	5	8	2
4	2	9	3
6	1	3	4

Solution:

7	8
6	9

Exercise 4: Average Pooling

Using the same 4x4 feature map, now apply average pooling with a 2x2 window and stride of 2.

```
3   7   2   4
1   5   8   2
4   2   9   3
6   1   3   4
```

Solution:

4	4
3.25	4.75

Challenge Problem

Let's combine convolution and pooling!

1. First, apply this 2x2 kernel to the input image:

```
2 0
0 2
```

Input Image:

```
1 2 3 4
2 3 4 1
3 4 1 2
4 1 2 3
```

2. Then apply max pooling (2x2, stride 2) to your result.

Solution:

12 12

12 12