Topic 6a

Convolutional Neural Network (Basics)

CSE465: Pattern Recognition and Neural Network
Sec: 3
Faculty: Silvia Ahmed (SvA)
Summer 2025

Topics

1. What is a Convolutional Neural Network (CNN)?
2. Basic intuition.
3. Visual Cortex vs CNN
4. Operations:
    1. Convolution
    2. Padding
    3. Stride
    4. Pooling
5. CNN Architecture

What is CNN?

- Convolutional Neural Network, also known as convnet, or CNNs, are a special kind of neural network for processing data that has a known grid-like topology like time series data (1D) or images (2D).

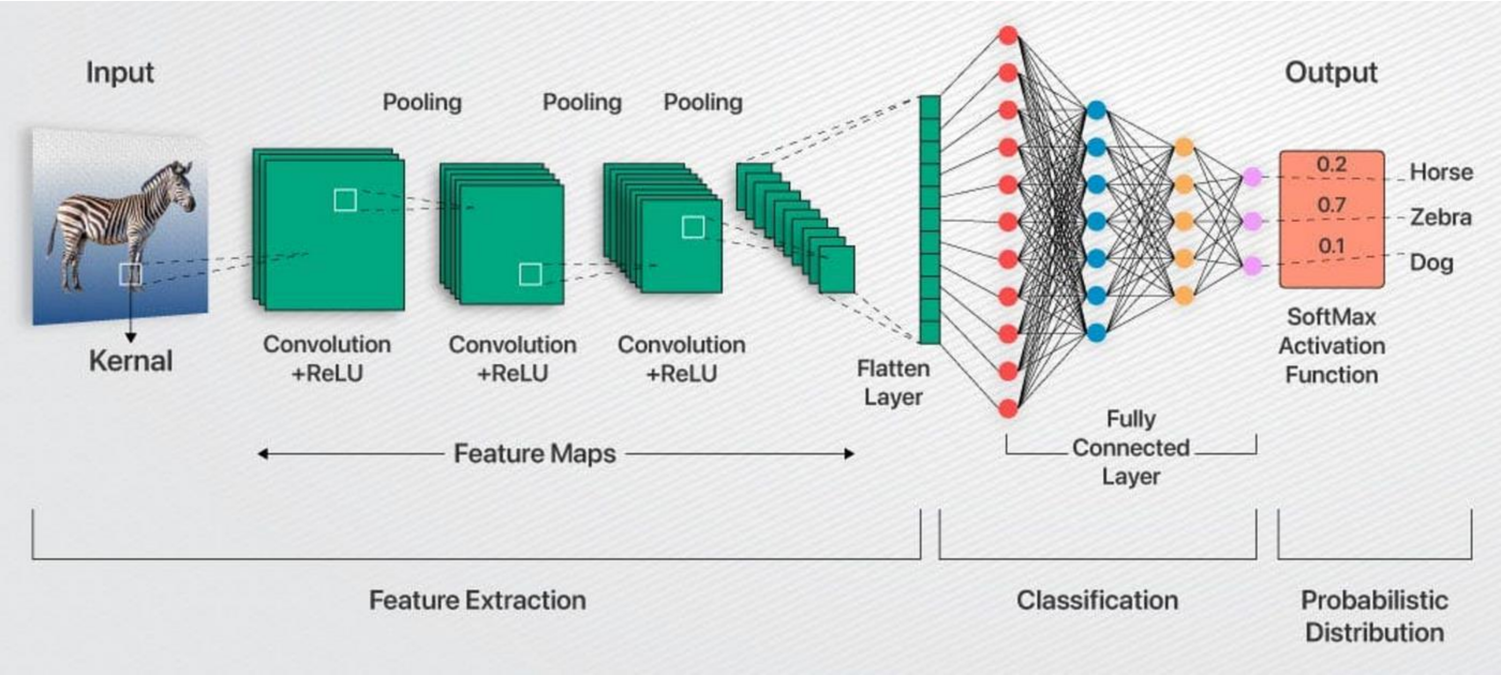
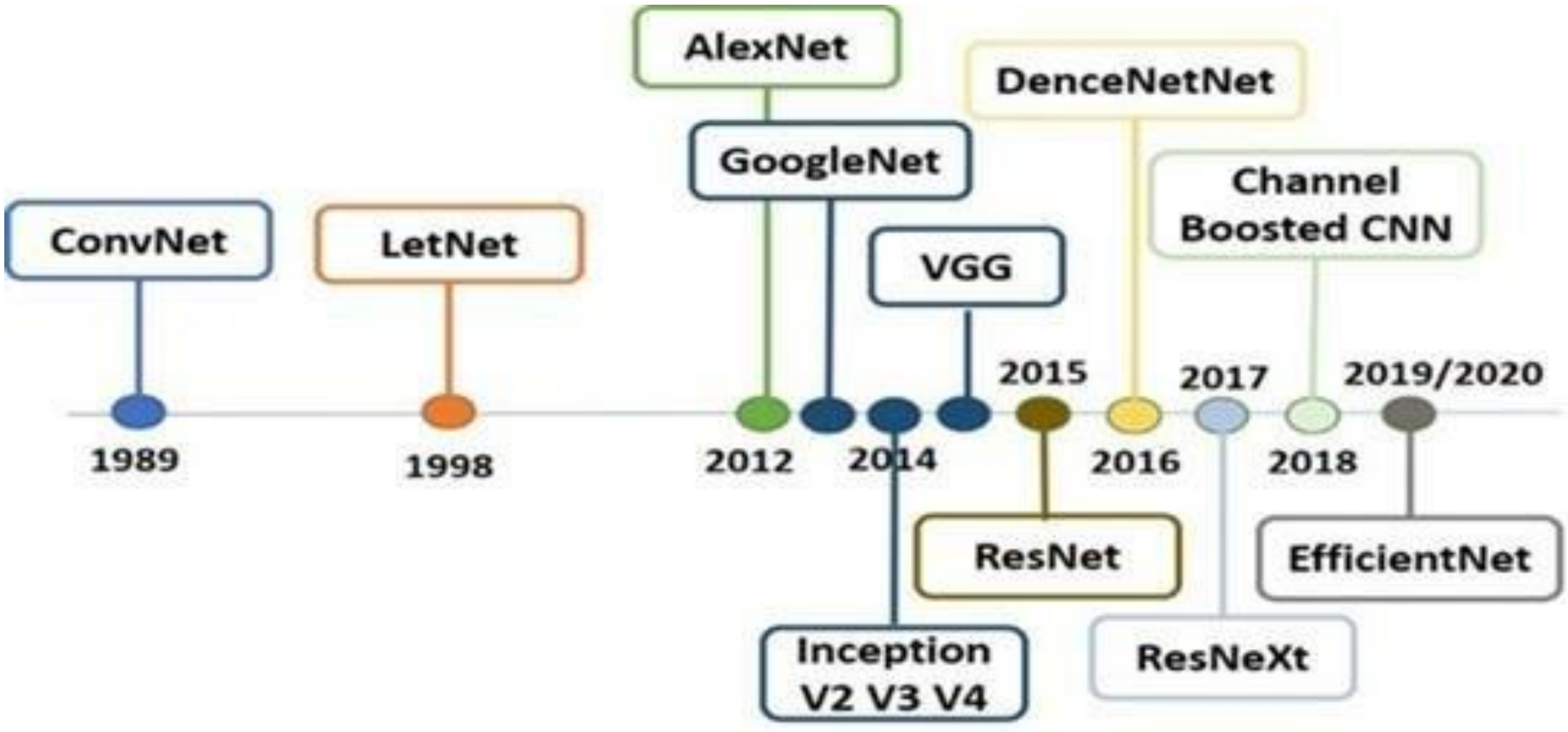


Figure: Basic structure of a CNN [1]

Layers in CNN

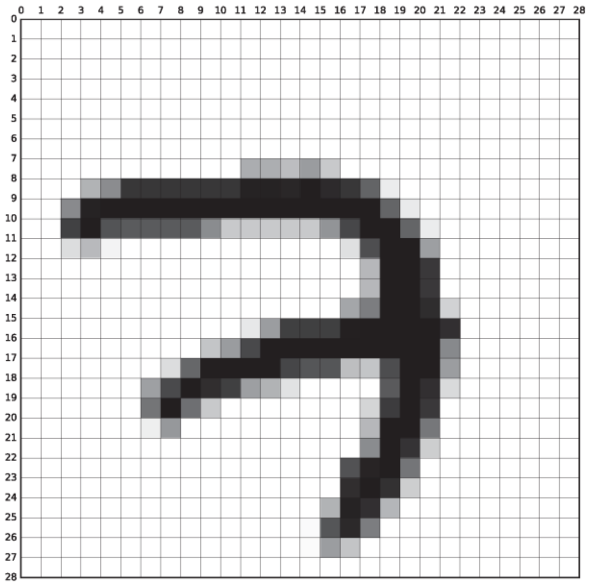
- A special layer for “convolution” operation.
- If there’s even a single “convolution” layer in a neural network, then that becomes a CNN.
- 3 types of layers:
    - Convolution layer
    - Pooling layer
    - Fully-connected (FC) layer

Brief history



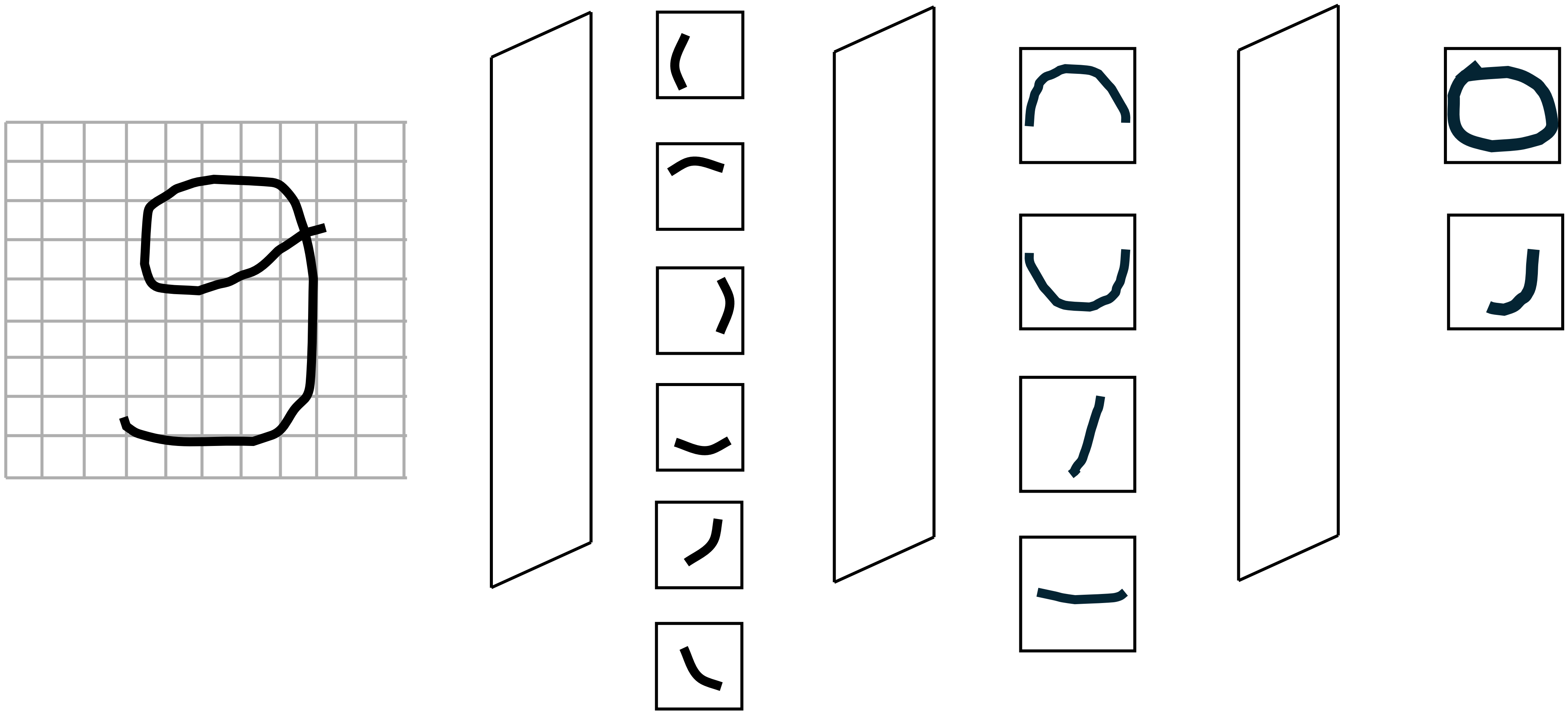
Limitations of ANN

- High computational cost
- Overfitting
- Loss of important info (e.g., spatial arrangement of pixels)

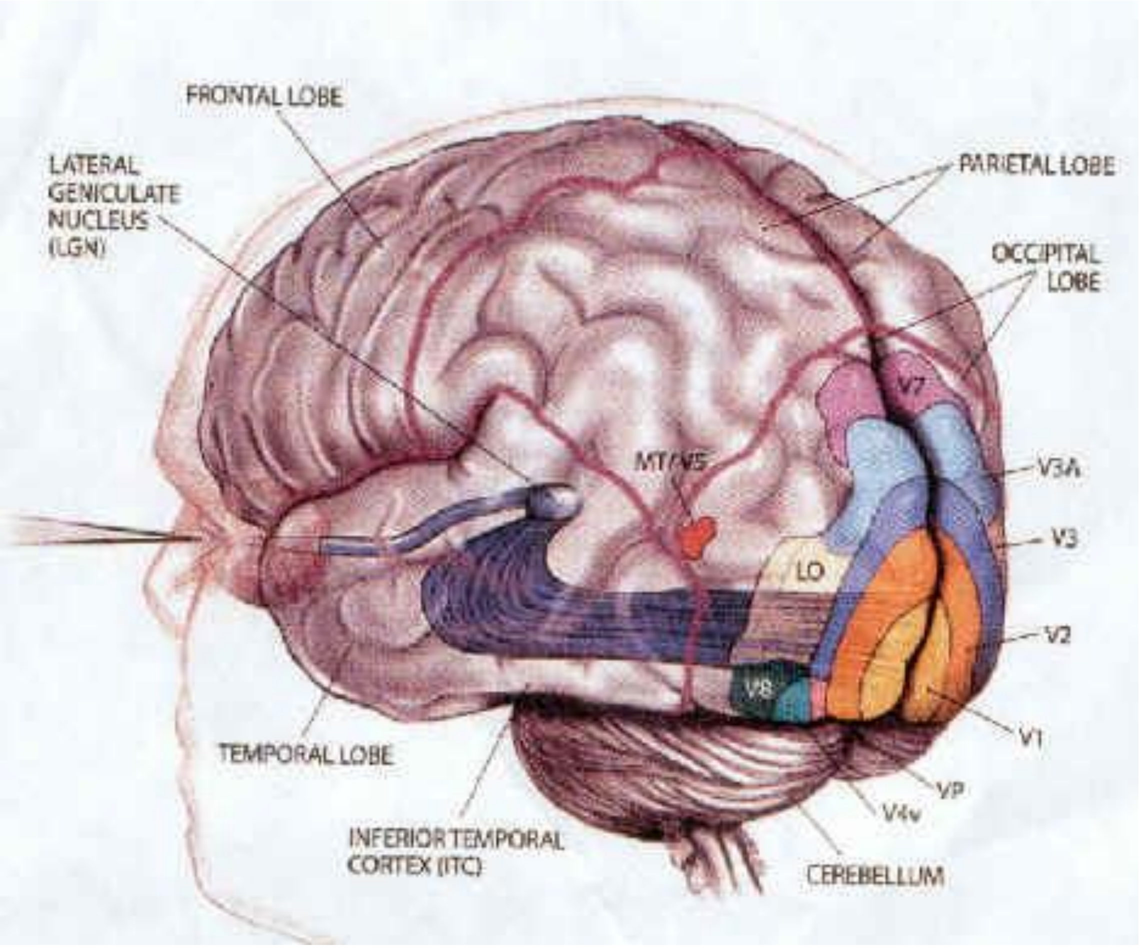


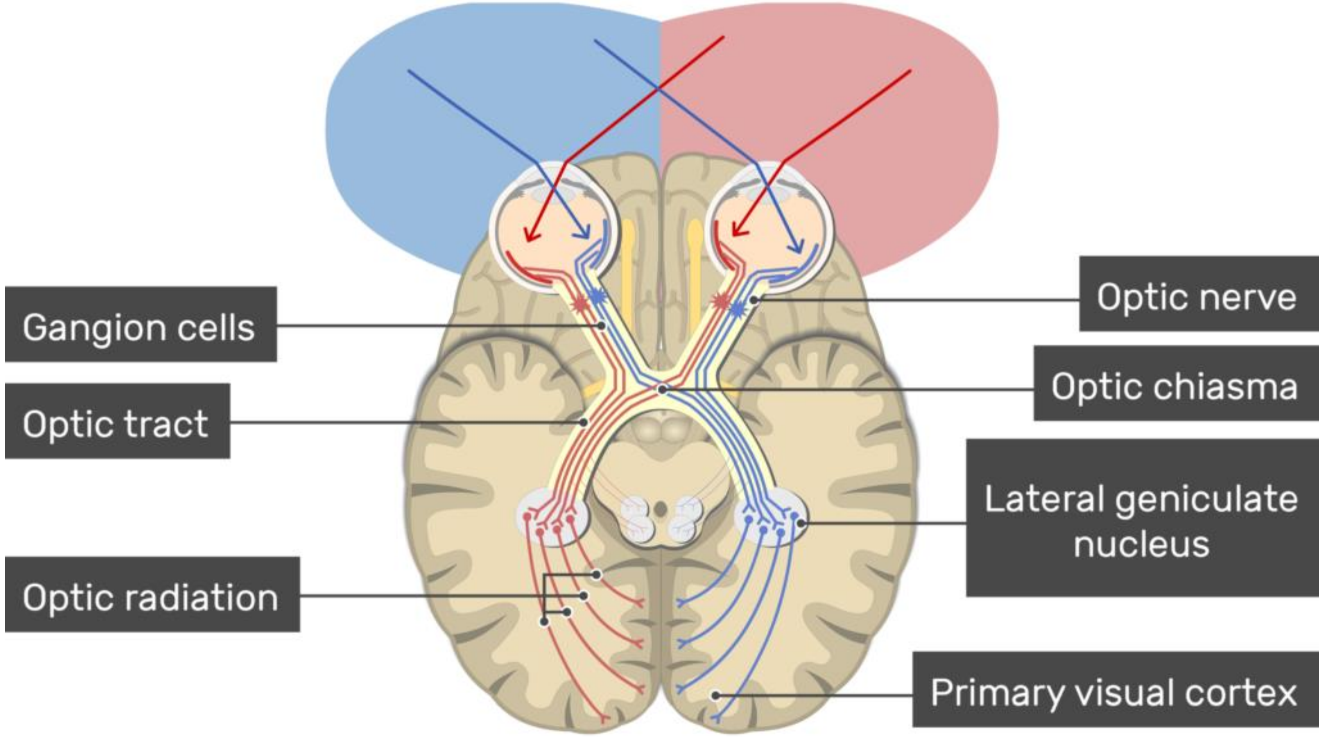
- 28x28 pixels grayscale MNIST image
- To feed into ANN, this must be flatten out to make 784 input nodes
- So layer 1 with 100 neurons will require learning parameters of 78400, and so on.
- Images with higher dimension will make this number exponentially bigger.
- Learning this huge no of parameters will take a lot of time which will eventually end up in overfitting.
- Flattening of the 2D structure looses spatial information as well.

Intuition

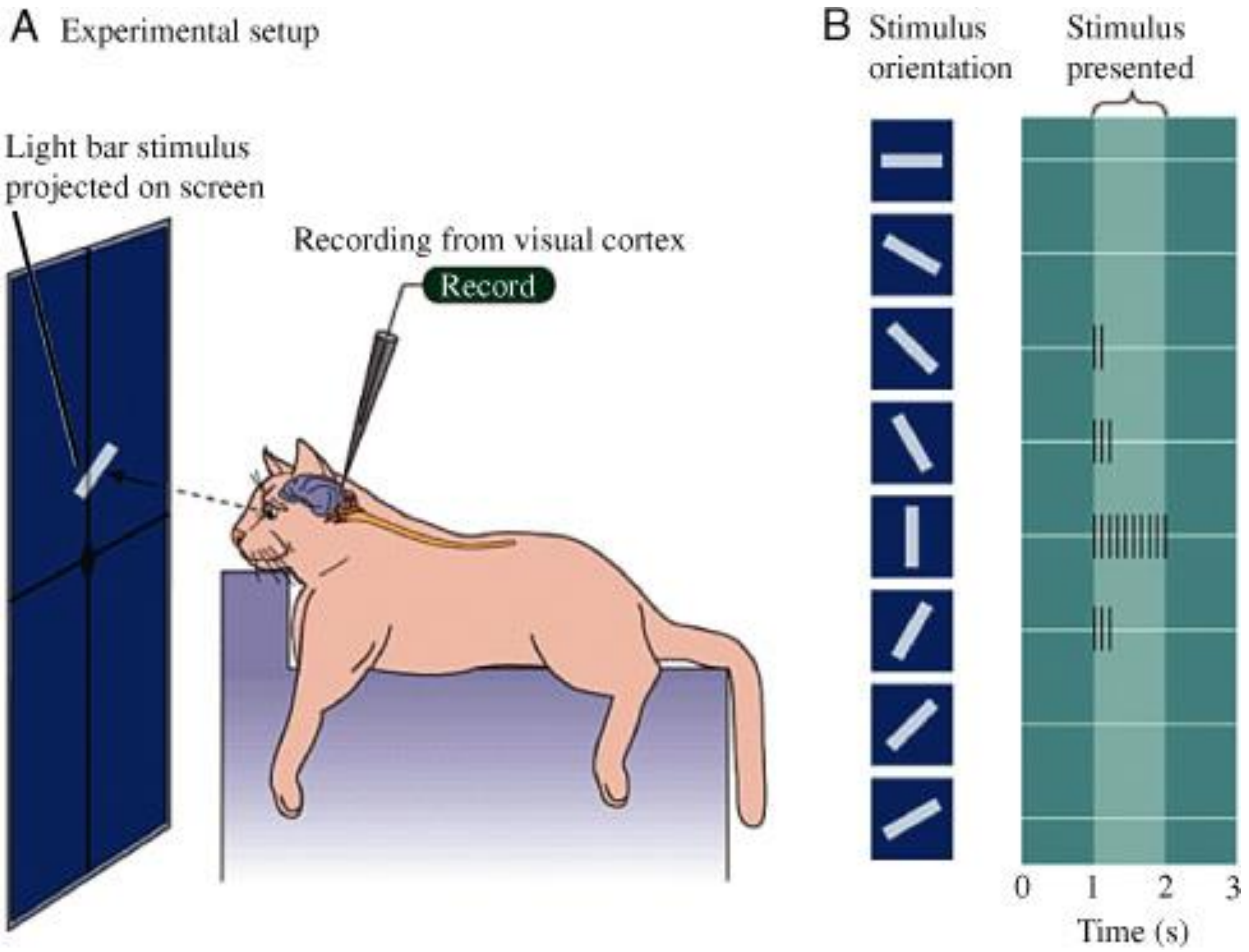


Visual Cortex



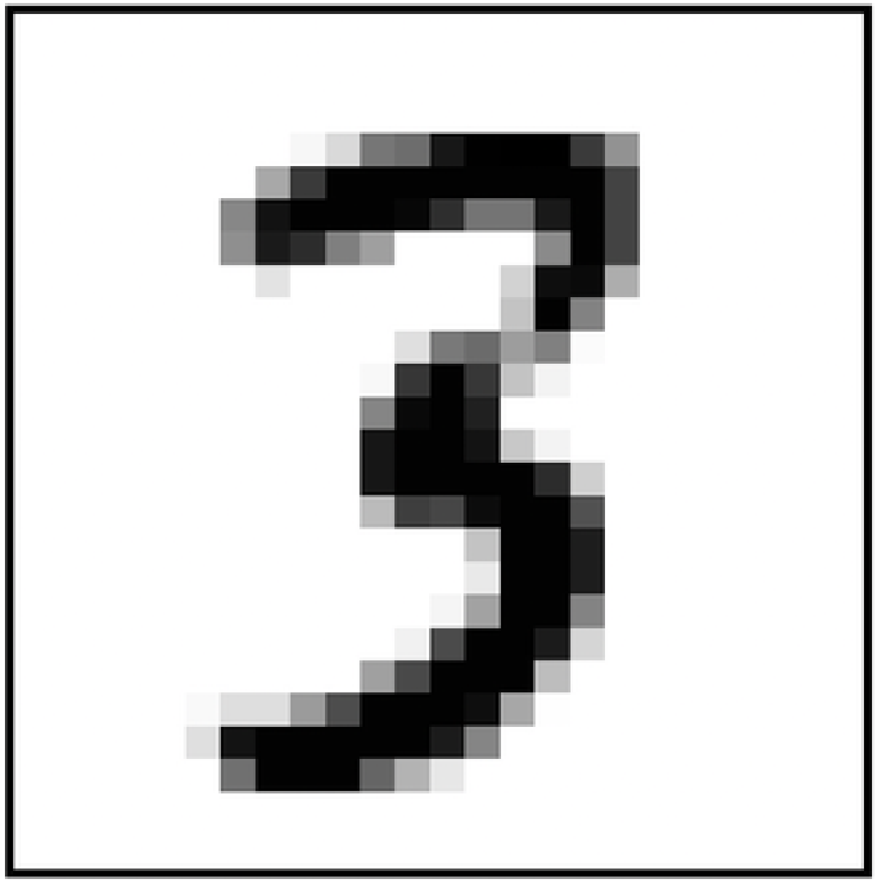


Hubel and Wiesel Redux



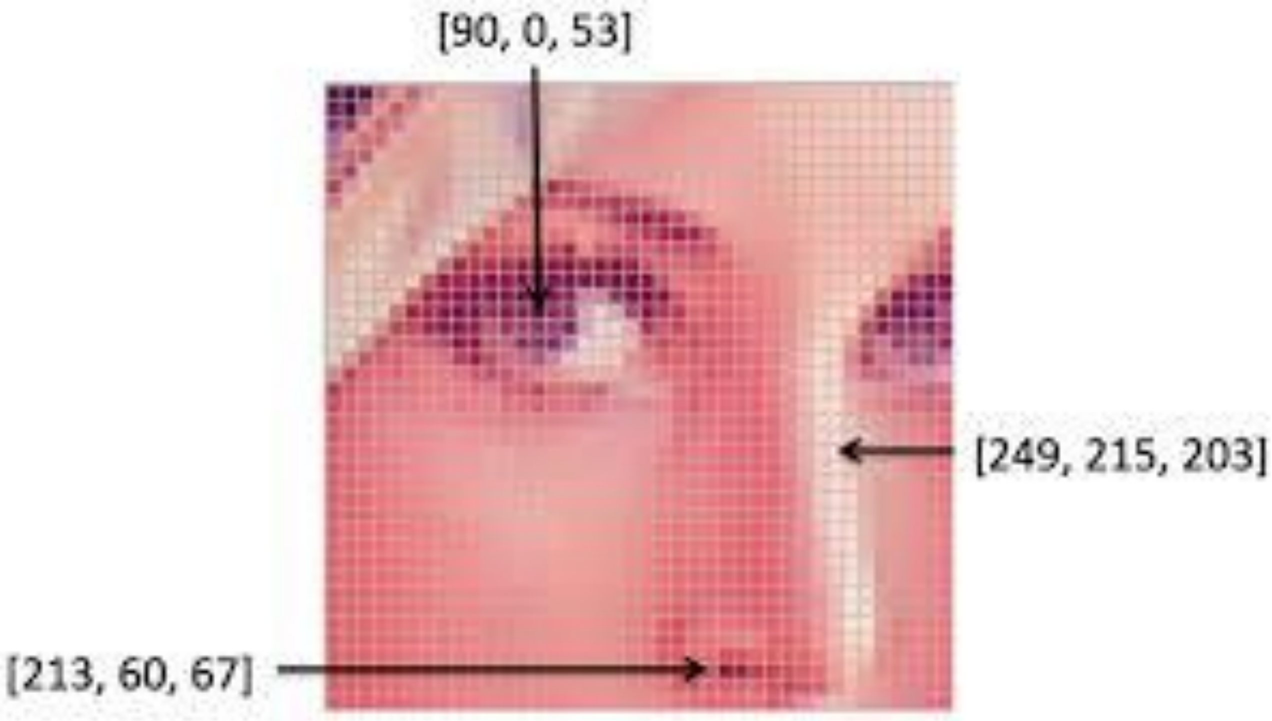
- Their original findings, showing that neurons in V1 detect simple edge-like patterns, while later layers respond to increasingly complex features, have been largely validated by modern neuroscience.
- Hubel and Wiesel categorized neurons into **simple** (edge detectors) and **complex** (more spatially invariant feature detectors).

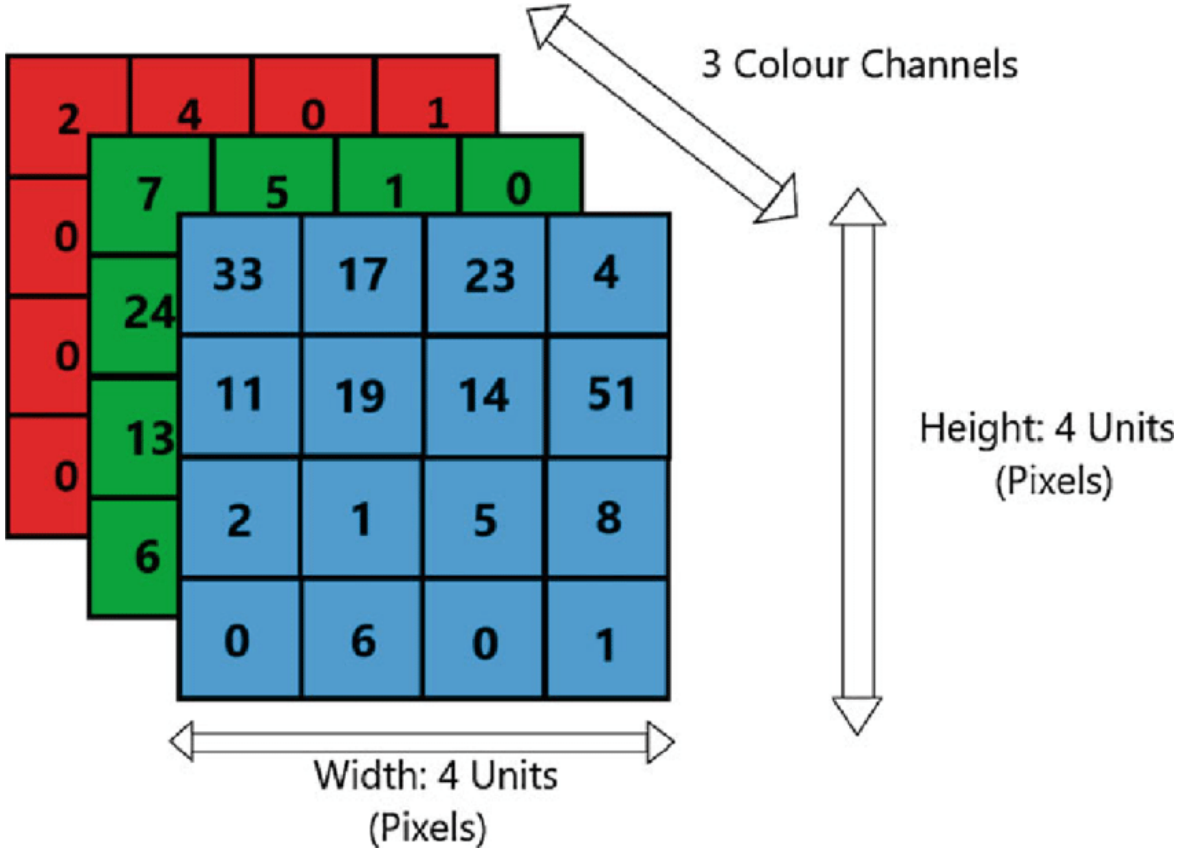
Basics of Images (Grayscale images)



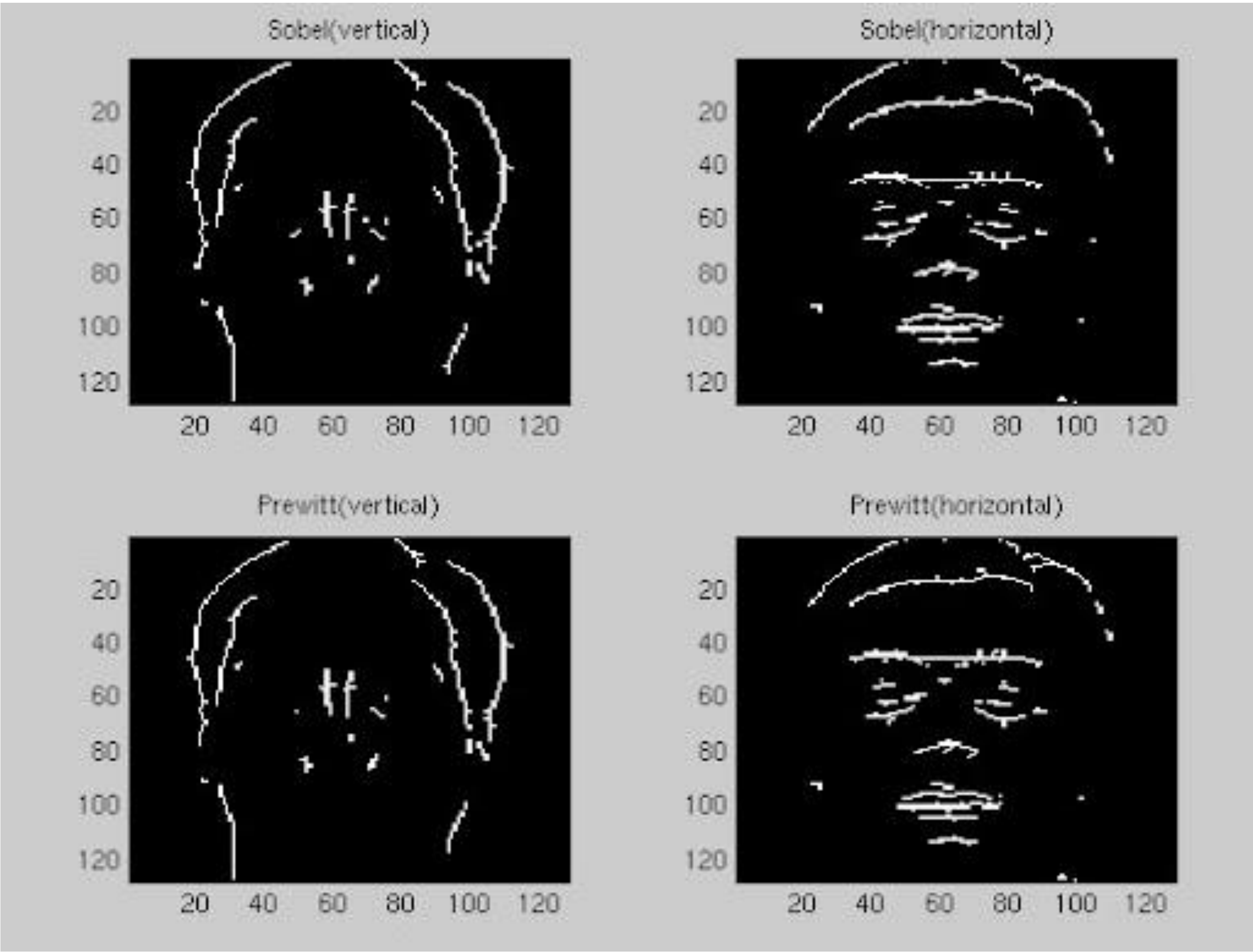
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	8	38	137	146	232	254	255	255	197	109	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	87	197	253	253	253	253	253	253	253	253	188	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	120	237	253	253	253	248	209	139	139	230	253	188	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	112	229	210	128	96	0	0	0	0	117	253	188	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	28	0	0	0	0	0	0	45	241	245	82	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	253	125	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	32	134	148	98	127	4	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	7	201	253	200	59	12	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	122	246	253	223	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	233	253	253	236	55	11	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	233	253	253	253	253	210	48	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	68	193	185	243	253	253	173	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59	253	253	226	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22	253	253	226	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	93	253	253	123	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	14	180	253	253	228	42	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	95	181	253	253	253	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	6	33	33	100	178	253	253	253	241	88	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	32	236	253	253	253	253	253	228	122	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	143	253	253	253	154	76	24	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Basics of Images (Color images)





Edge Detection (Convolution operation)



Edges are changed intencity

Horizontal Edge Detection

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

Image

*

-1	-1	-1
0	0	0
1	1	1

Filter/Kernel

=

Feature Map

Horizontal Edge Detection (calculation)

0	0	0
0	0	0
0	0	0

Itemwise multiplication then add

-1	-1	-1
0	0	0
1	1	1

$$= 0(-1) + 0(-1) + 0(-1) + 0*0 + 0*0 + 0*0 + 0*1 + 0*1 + 0*1 = 0$$

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

*

-1	-1	-1
0	0	0
1	1	1

=

0			

Horizontal Edge Detection (calculation)

0	0	0
0	0	0
0	0	0

Itemwise multiplication then add

-1	-1	-1
0	0	0
1	1	1

$= 0(-1) + 0(-1) + 0(-1) + 0*0 + 0*0 + 0*0 + 0*1 + 0*1 + 0*1 = 0$

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

*

-1	-1	-1
0	0	0
1	1	1

=

0	0		

Horizontal Edge Detection (calculation)

0	0	0
0	0	0
0	0	0

Itemwise multiplication then add

-1	-1	-1
0	0	0
1	1	1

$= 0(-1) + 0(-1) + 0(-1) + 0*0 + 0*0 + 0*0 + 0*1 + 0*1 + 0*1 = 0$

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

*

-1	-1	-1
0	0	0
1	1	1

=

0	0	0	

Horizontal Edge Detection (calculation)

0	0	0
0	0	0
0	0	0

Itemwise multiplication then add

-1	-1	-1
0	0	0
1	1	1

$= 0(-1) + 0(-1) + 0(-1) + 0*0 + 0*0 + 0*0 + 0*1 + 0*1 + 0*1 = 0$

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

$*$

-1	-1	-1
0	0	0
1	1	1

$=$

0	0	0	0

Horizontal Edge Detection (calculation)

0	0	0
0	0	0
0	0	0

Itemwise multiplication then add

-1	-1	-1
0	0	0
1	1	1

$$= 0(-1) + 0(-1) + 0(-1) + 0*0 + 0*0 + 0*0 + 255*1 + 255*1 + 255*1 = 765 \approx 255$$

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

*

-1	-1	-1
0	0	0
1	1	1

=

0	0	0	0
255			

Horizontal Edge Detection (calculation)

0	0	0
0	0	0
0	0	0

Itemwise multiplication then add

-1	-1	-1
0	0	0
1	1	1

$$= 0(\text{-}1) + 0(\text{-}1) + 0(\text{-}1) + 0*0 + 0*0 + 0*0 + 255*1 + 255*1 + 255*1 = 765 \approx 255$$

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

$*$

-1	-1	-1
0	0	0
1	1	1

$=$

0	0	0	0
255	255		

Horizontal Edge Detection (calculation)

0	0	0
0	0	0
0	0	0

Itemwise multiplication then add

-1	-1	-1
0	0	0
1	1	1

$$= 0(-1) + 0(-1) + 0(-1) + 0*0 + 0*0 + 0*0 + 255*1 + 255*1 + 255*1 = 765 \approx 255$$

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

*

-1	-1	-1
0	0	0
1	1	1

=

0	0	0	0
255	255	255	

Horizontal Edge Detection (calculation)

0	0	0
0	0	0
0	0	0

Itemwise multiplication then add

-1	-1	-1
0	0	0
1	1	1

$$= 0(-1) + 0(-1) + 0(-1) + 0*0 + 0*0 + 0*0 + 255*1 + 255*1 + 255*1 = 765 \approx 255$$

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

*

-1	-1	-1
0	0	0
1	1	1

=

0	0	0	0
255	255	255	255

Horizontal Edge Detection (calculation)

0	0	0
0	0	0
0	0	0

Itemwise multiplication then add

-1	-1	-1
0	0	0
1	1	1

$$= 255(-1) + 255(-1) + 255(-1) + 255*0 + 255*0 + 255*0 + 255*1 + 255*1 + 255*1 = 0$$

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

*

-1	-1	-1
0	0	0
1	1	1

=

0	0	0	0
255	255	255	255
255	255	255	255
0	0	0	

Horizontal Edge Detection (calculation)

0	0	0
0	0	0
0	0	0

Itemwise multiplication then add

-1	-1	-1
0	0	0
1	1	1

$= 255(-1) + 255(-1) + 255(-1) + 255*0 + 255*0 + 255*0 + 255*1 + 255*1 + 255*1 = 0$

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

*

-1	-1	-1
0	0	0
1	1	1

=

0	0	0	0
255	255	255	255
255	255	255	255
0	0	0	0

The filter values are traineable parameters. So no matter which values we initialize the filter with, through Back propagation, they become optimized.

Live Demonstration

- https://deeplizard.com/resource/pavq7noze2
- Red: Positive activation
- Blue: Opposite edge detection
- For example, if we chose left-edge filter, then red means left edges are detected. On the other hand, blue means opposite (right-edge) has been detected.

Size of feature map

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

n x n

*

-1	-1	-1
0	0	0
1	1	1

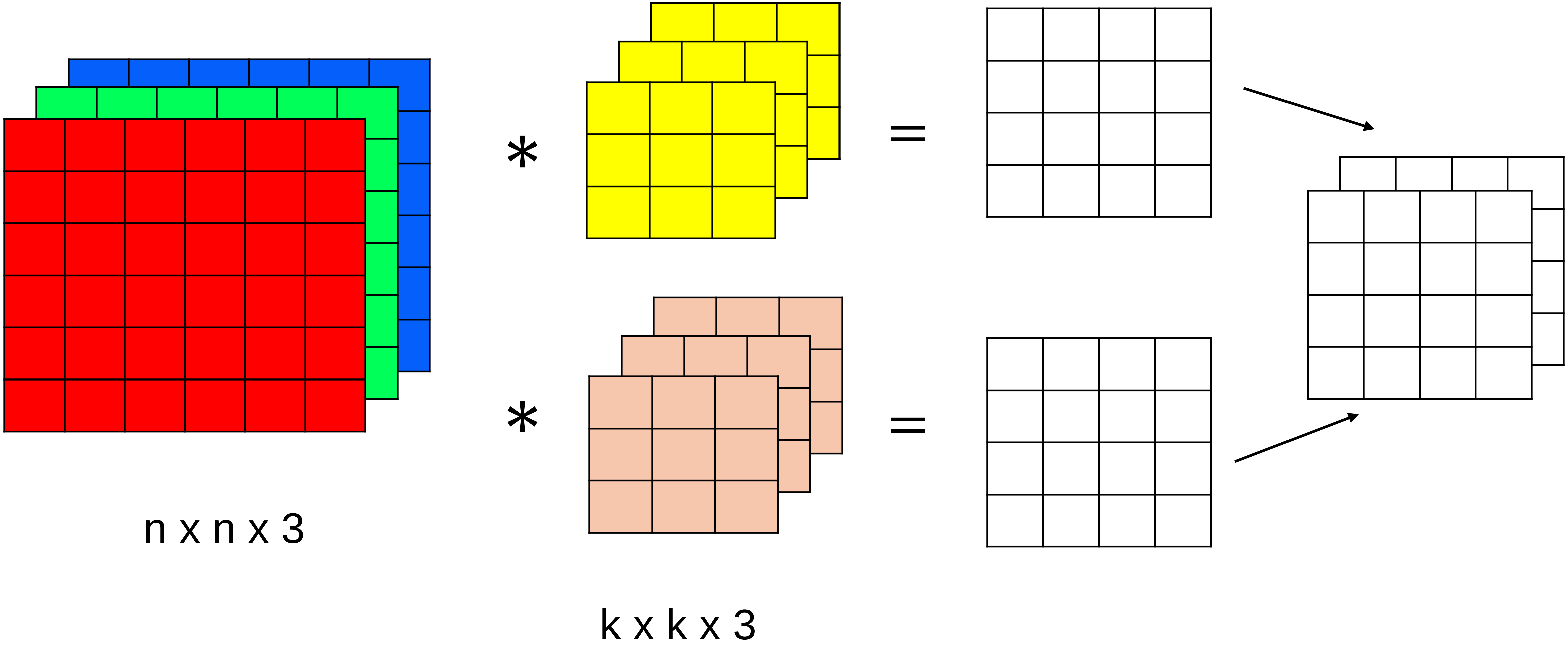
k x k

=

0	0	0	0
255	255	255	255
255	255	255	255
0	0	0	0

(n – k + 1) x (n – k + 1)

Working with RGB images



Issues with Convolution

- Reduction in Spatial Dimensions:
  - the output size of the feature map decreases after each convolutional layer due to the kernel sliding over the input image.
  - If we use a **k × k** filter on an **n × n** input with a stride of 1, the output size is reduced to **(n − k + 1) × (n − k + 1)**.
  - This means that as we go deeper into the network, the feature maps keep shrinking, leading to information loss.
  - Example: For a **28 × 28** input with a **3 × 3** filter and no padding:

$$\text{Output size} = 28 - 3 + 1 = 26$$

  - Each layer further reduces the size, which can lead to vanishing spatial information in deep networks.

Issues with Convolution (contd.)

- Loss of Edge Information:
  - **edge pixels** are used fewer times compared to central pixels during convolution, leading to **biased feature extraction**.
  - Padding ensures that even edge and corner features contribute equally in the learning process.
- No Control Over Output Size

- **Solution: Add extra pixels** (usually zeros) around the input to maintain or control the spatial dimensions.

Padding

- Padding refers to **adding extra pixels** (usually zeros) around the input image before applying convolution operations.
- It helps control the spatial size of feature maps and enhances model performance.
- Benefits:
  - Maintains Spatial Dimensions
  - Prevents Loss of Edge Information
  - Allows for Control Over Feature Map Size
  - Improves Performance in Deep CNNs

Types of Padding in CNNs

1. Valid Padding ("No Padding"):

- **Definition:** No extra pixels are added, meaning the kernel applies only to the original image.
- **Effect:** The feature map shrinks after each convolution.
- **Formula:**

$$Output\ size = (n - k + 1) \times (n - k + 1)$$

- **Example:**
  - **Input:** 28 x 28, **Filter:** 3 x 3, No Padding
  - **Output:** (28 – 3 + 1) x (28 – 3 + 1) = 26 x 26
- When to use?
  - When reducing spatial size is acceptable (e.g., classification tasks).
  - When deeper layers apply **global average pooling** (e.g., ResNet, MobileNet).

Types of Padding in CNNs (contd.)

2. Same Padding (Zero-Padding):

- **Definition:** Padding is added to ensure that the **output size is the same as the input size**.
- **Formula for padding size:**

$$P = \frac{(k-1)}{2} \text{ (for odd-sized kernels)}$$

- **Example:**
  - **Input:** 28 x 28, **Filter:** 3 x 3, **Padding:** $\frac{3-1}{2} = 1$ pixel
  - **Output:** 28 x 28 (unchanged)
- Advantages:
  - Keeps feature map size constant, simplifying architecture design.
  - Useful for deep networks like **VGG, ResNet**.

Types of Padding in CNNs (contd.)

3. Full Padding:

- **Definition:** Maximum padding is applied so that every pixel gets covered by the kernel the same number of times.
- **Formula for padding size,** P = k - 1
- **Formula for Output Size:**

$$Output\ size = (n + 2P - k + 1)\ \times\ (n + 2P - k + 1)$$

- **Example:**
  - **Input:** 28 x 28, Fi**lter:** 3 x 3, **Full padding:** 2 pixels
  - **Output:** (28 + 2x2 – 3 + 1) or 30 x 30
- **When to use?**
  - If we want **larger feature maps** than the input size.
  - Used in **certain styles of generative models (GANs, autoencoders).**

Types of Padding Techniques

1. Zero Padding (Most Common)
   - Adds zeros around the image.
   - Simple and widely used.

0	0	0	0	0	0
0	1	2	3	4	0
0	5	6	7	8	0
0	9	10	11	12	0
0	13	14	15	16	0
0	0	0	0	0	0

2. Replication Padding
   - Duplicates edge values to preserve texture.
   - Used in **image super-resolution**.

1	1	2	3	4	4
1	1	2	3	4	4
5	5	6	7	8	8
9	9	10	11	12	12
13	13	14	15	16	16
13	13	14	15	16	16

Types of Padding Techniques

3. Reflection Padding

- Mirrors pixels at the edge..
- Reduces border artifacts in **image processing**.

6	5	6	7	8	7
2	1	2	3	4	3
6	5	6	7	8	7
10	9	10	11	12	11
14	13	14	15	16	15
10	9	10	11	12	11

2. Circular Padding

- Wraps the image around itself.
- Used in **periodic signal processing**.

16	13	14	15	16	13
4	1	2	3	4	1
8	5	6	7	8	5
12	9	10	11	12	9
16	13	14	15	16	13
4	1	2	3	4	1

Stride

- **Stride** is the step size by which the convolutional filter (kernel) moves across the input image during convolution.
- It determines:
  - How much the receptive field moves at each step
  - How much the output shrinks compared to the input
  - How much computational efficiency is improved
- **S = 1** → The filter moves **one pixel** at a time → **Dense feature extraction**
- **S = 2** → The filter moves **two pixels** at a time → **Downsampling occurs**
- **S > 2** → The filter moves **more than two pixels** at a time → **Aggressive Downsampling occurs**

Formulation of Padding and Strides

- For a **stride S** and **padding P**, the **output size** of a convolutional layer is given by:

$$Output\ width = \left\lfloor \frac{(Input\ width + 2P - k)}{S} \right\rfloor + 1$$

$$Output\ height = \left\lfloor \frac{(Input\ height + 2P - k)}{S} \right\rfloor + 1$$

Where:

- k = kernel/filter size
- S = Stride
- P = Padding
- Input width/height = Original Image size

Further Issues with Convolution

1. Memory issues: Example, For a **224 × 224 RGB image**, assuming:
    - **64 feature maps** in a layer,
    - **32-bit float representation (4 bytes per value)**,

    the memory required for one layer is:

    $$224 \times 224 \times 64 \times 4 \text{ bytes} = 12.8 \text{ MB}$$

    - For **10 layers**, the memory usage becomes **128 MB per image!**
    - With **batch processing**, memory usage increases further

Solution: Pooling

2. Translation variance:
    - CNNs are **not fully translation-invariant**, meaning:
        - If an object shifts slightly in an image, the CNN might classify it differently.
        - Small shifts cause different activations, affecting feature maps.
        - Example: Digit Recognition (MNIST Dataset): If a "5" is shifted **one pixel to the right**, the CNN may **misclassify it as "3"** due to different activations.

Pooling (layer)

- Pooling is a **downsampling operation** used in CNNs to reduce the spatial dimensions of feature maps while preserving important information.
- It helps in:
  - **Reducing computational cost** by shrinking the feature map size
  - **Improving translation invariance** (i.e., detecting patterns regardless of their exact location)
  - **Preventing overfitting** by reducing unnecessary details
- How Pooling Works?
  - Pooling operates on small regions (typically **2×2**) of the feature map and summarizes them using a specific function

Types of Pooling

1. Max Pooling (Most Common)

- Takes the **largest value** in the pooling window.
- Preserves the **strongest features** (e.g., edges, textures).
- Commonly used in deep CNN architectures (e.g., VGG, ResNet).

1	3	2	4
5	6	8	7
9	10	12	11
13	14	16	15

2 x 2 max pooling,
Stride = 1

6	8
14	16

- Max pooling reduces noise and keeps dominant features
- Commonly used after convolutional layers to reduce dimensions

Types of Pooling (contd.)

2. Average Pooling:

- Computes the **mean value** in the pooling region.
- Retains **global structure** but loses some sharp details.
- Used in **shallow networks or specific tasks like regression**.

1	3	2	4
5	6	8	7
9	10	12	11
13	14	16	15

2 x 2 avg pooling,
Stride = 1

3.75	5.25
11.25	13.75

- **Blurs sharp edges** but **keeps overall distribution**
- Used in classification tasks like ImageNet models (AlexNet, ResNet)
- Better for smooth feature extraction

Types of Pooling (contd.)

3. Global Pooling (Global Average Pooling):

- Reduces the entire feature map to a **single value per channel**.
- Used in architectures like **Google's Inception** and **ResNet**.
- Helps replace **fully connected (FC) layers**, reducing parameters.

1	3	2	4
5	6	8	7
9	10	12	11
13	14	16	15

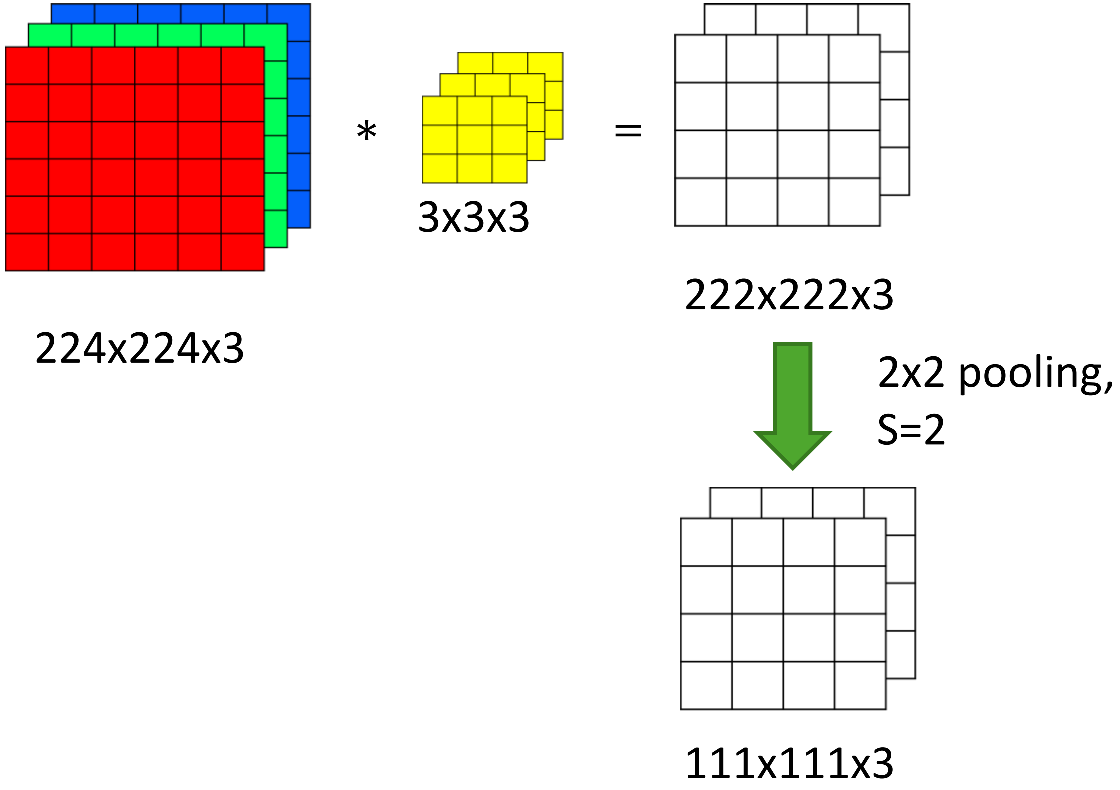
8.5

(Single value per channel)

- Used before the final classification layer in **ResNet**
- Eliminates need for large FC layers, reducing parameters
- Prevents overfitting in deep networks

Benefits of Pooling

- Reduced size:



- Translation invariance:

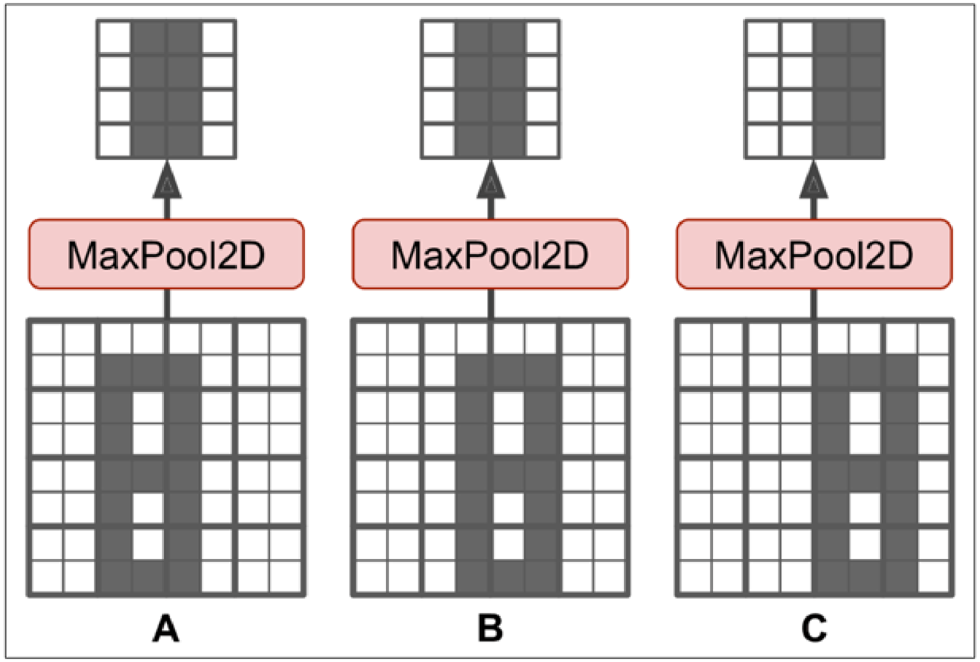
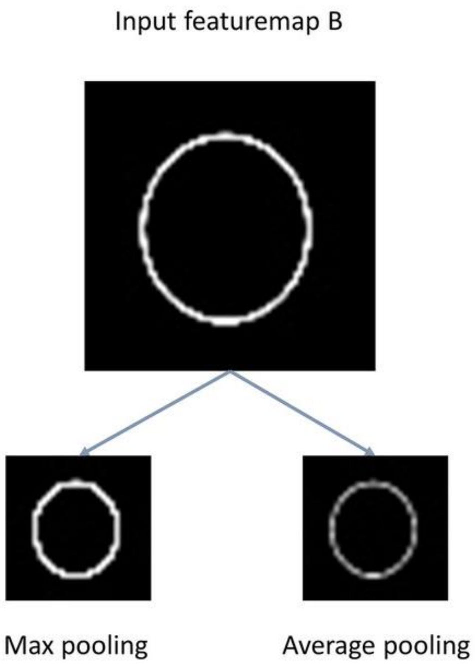


Figure 14-9. Invariance to small translations

Benefits of Pooling (contd.)

- Enhanced Features:Benefits of Pooling
  - Only in case of Max pooling



- No need of training

Stride & Pooling

- Pooling layers typically use **stride = pool size** to ensure:
  - **Non-overlapping** receptive fields (e.g., **2×2 pool, stride = 2**)
  - Downsampling without overlap
  - Smaller, more efficient feature maps

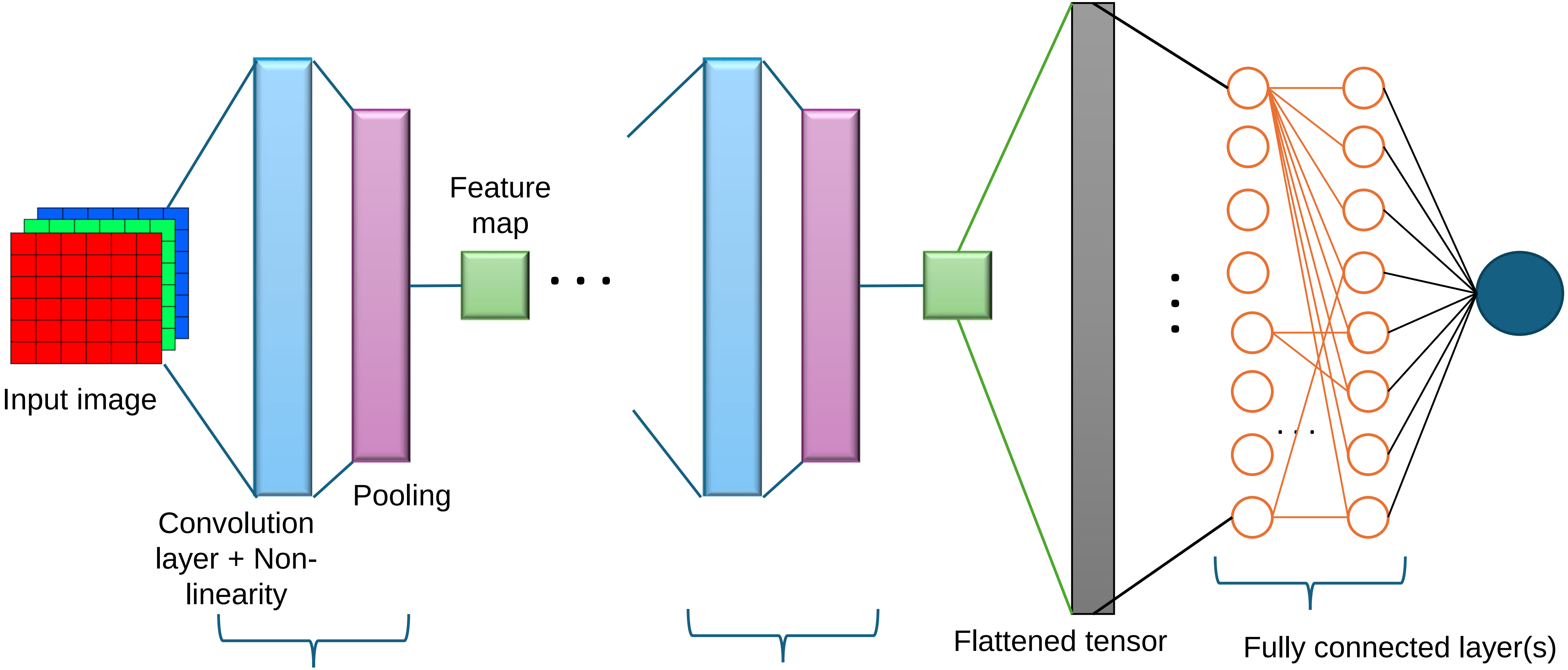
Pooling vs Convolution: Key Differences

Feature	Convolution	Pooling
Purpose	Extracts features (edges, textures)	Reduces feature map size
Operation	Learns from data (weights)	Fixed function (max/avg)
Effect	Preserves information	Removes redundant information
Computational Cost	High	Low

When NOT to Use Pooling

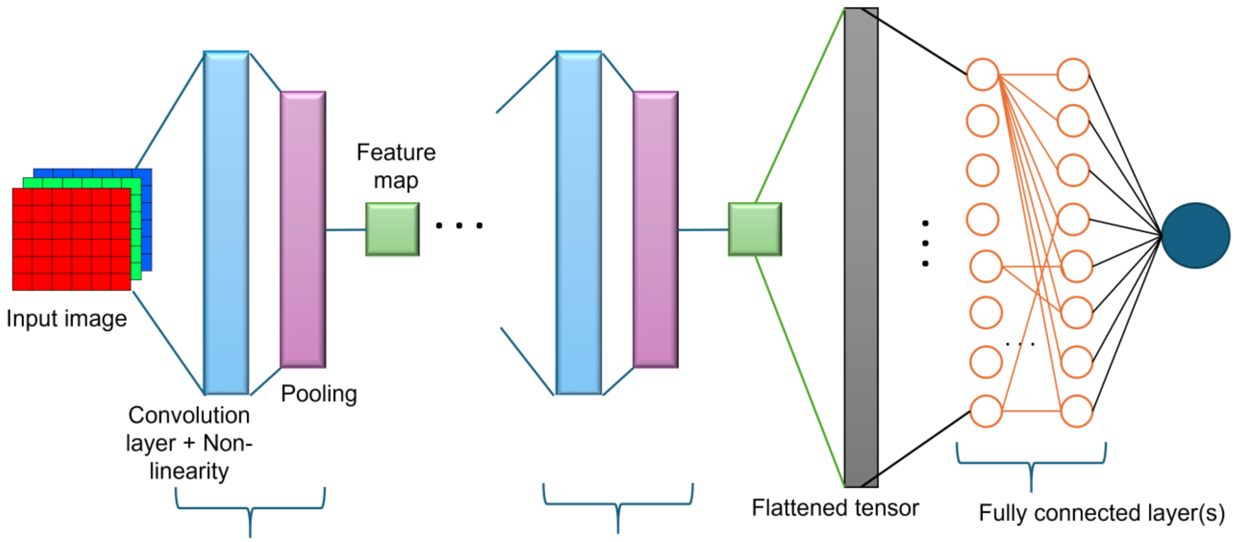
- **If spatial relationships are important** (e.g., segmentation tasks)
- **If information loss is harmful** (e.g., GANs, super-resolution models)
- **If using Strided Convolution as an alternative** (e.g., ResNet, MobileNet)

Basic CNN architecture

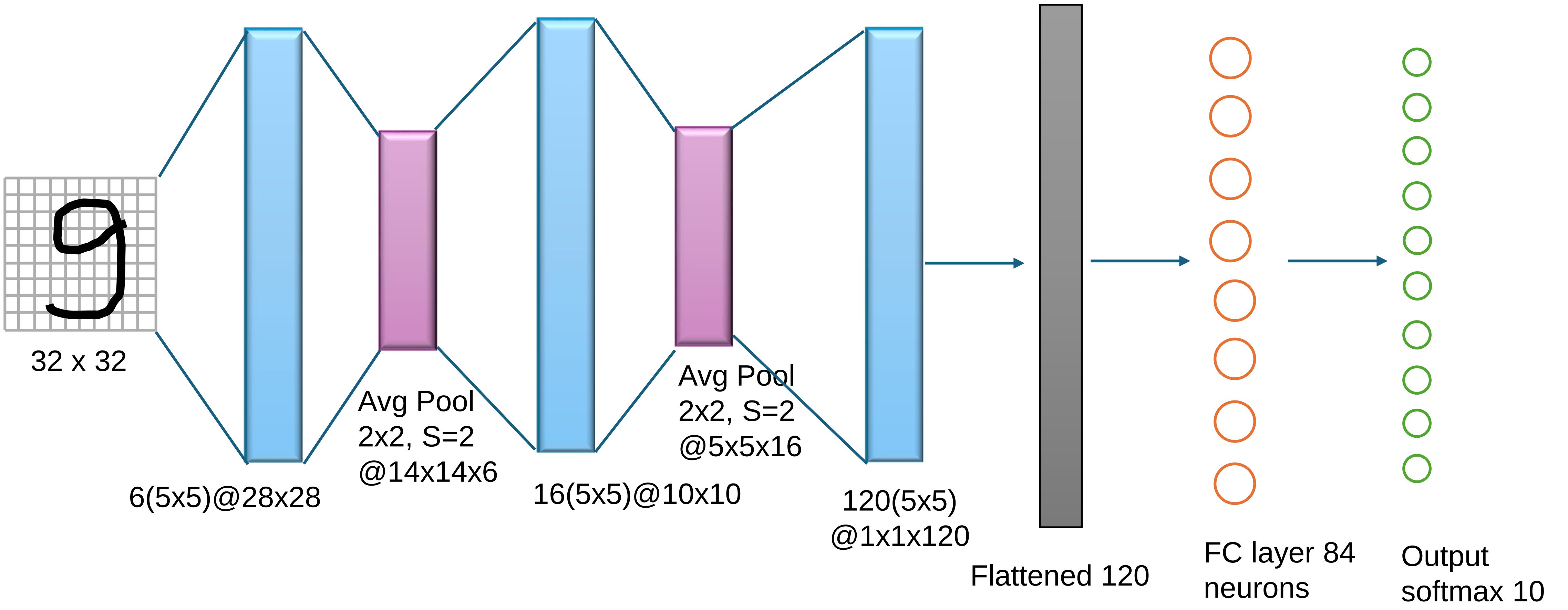


Difference in CNN Architecture

- Number of Convolution layer
- Number of filters/kernels
- Stride
- Pooling
- Number of Fully Connected (FC) nodes
- Number of FC layers
- Activation functions
- Dropouts
- Batch norm



Example: LeNet-5



Source: [3]

References

[1] https://ravjot03.medium.com/decoding-cnns-a-beginners-guide-to-convolutional-neural-networks-and-their-applications-1a8806cbf536

[2] “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow”, 2nd Edition

[3] https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/

[4] Youtube playlist: 100 Days of Deep Learning by CampusX