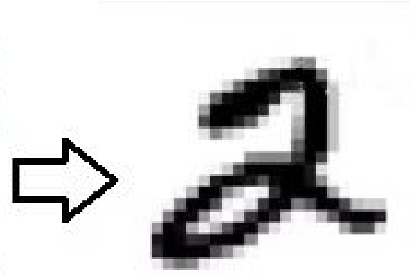
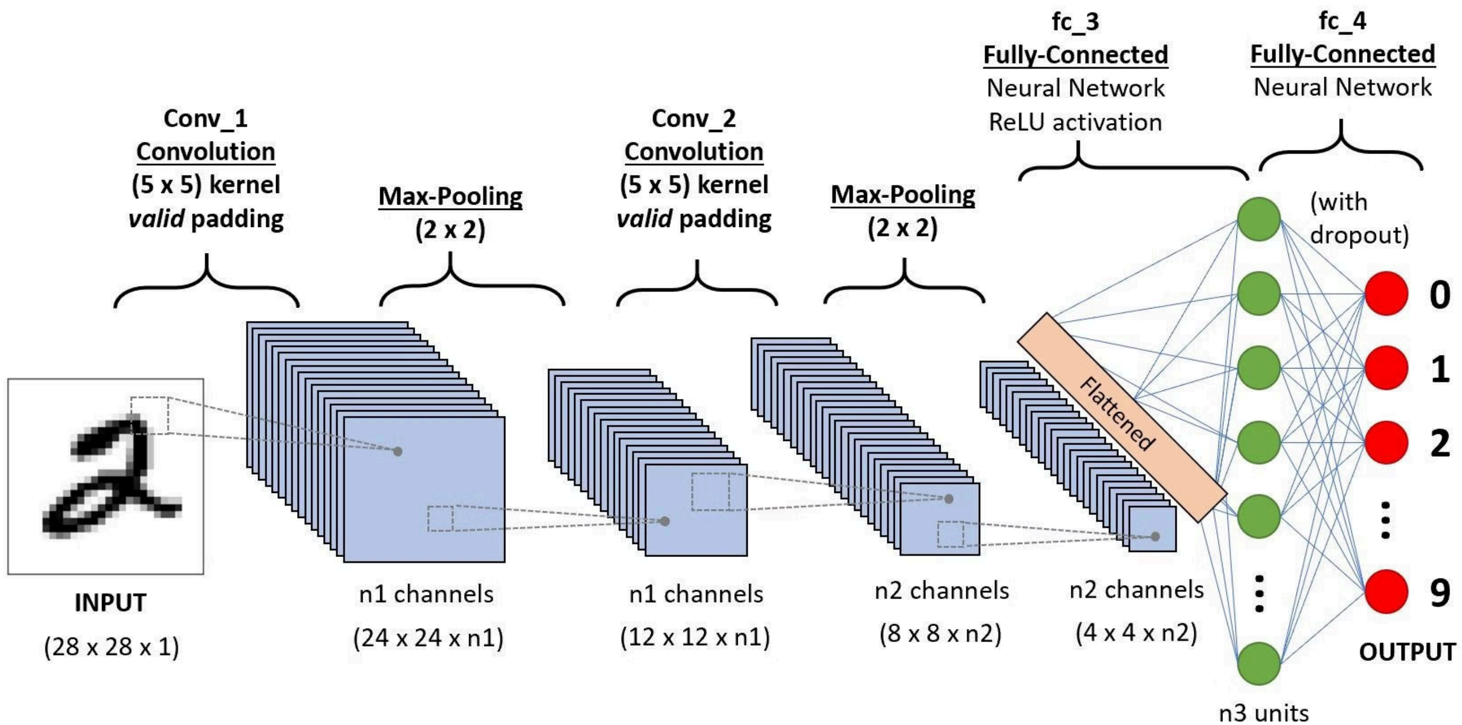
Introduction to Convolutional Neural Networks

A convolutional neural network (CNN) is a type of artificial intelligence (AI) algorithm that is most commonly used for analyzing visual imagery. It is inspired by the organization and functionality of the visual cortex in the human brain and is highly efficient for image recognition and classification tasks.

The Architecture of a CNN

```
[[ 69, 130,  51, ...,  55,  13, 117],
 [155,  71, 101, ..., 197,  21, 104],
 [ 24,  79,  38, ...,  64,  47,  48],
 ...,
 [ 15, 121, 138, ..., 215, 185, 164],
 [ 68,  12, 174, ..., 102, 250, 197],
 [ 96, 129, 207, ..., 249,   3, 165]]
```





Convolutional Layers and Their Purpose

1. **Feature Extraction**

   Convolutional layers are responsible for extracting features from input images through the use of learnable filters that slide across the input and detect patterns and structures.

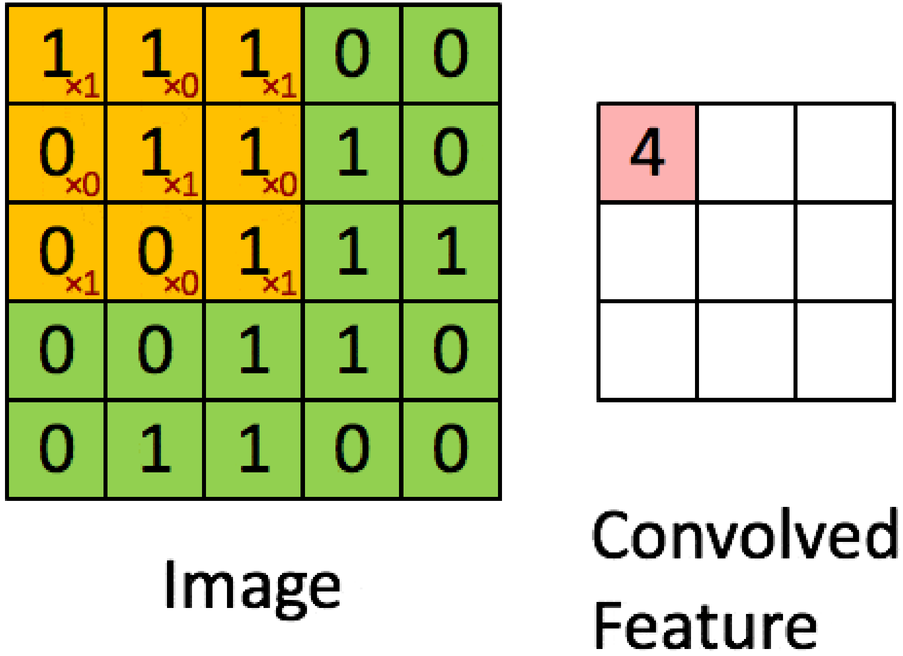
2. **Parameter Sharing**

   By using shared weights, convolutional layers reduce the number of parameters, making it possible to apply the same feature detector across the entire input, improving efficiency.

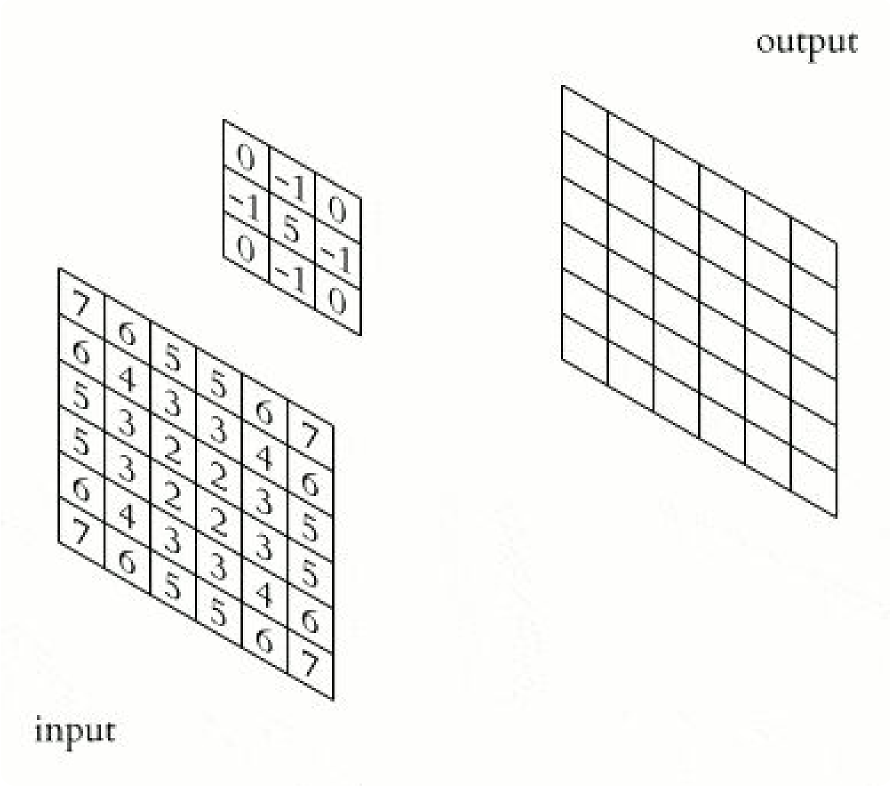
3. **Spacial Hierarchies**

   These layers help in creating a hierarchical representation of the input data, capturing high-level features by combining low-level features, enabling deeper insights into the data.

Convolution layer

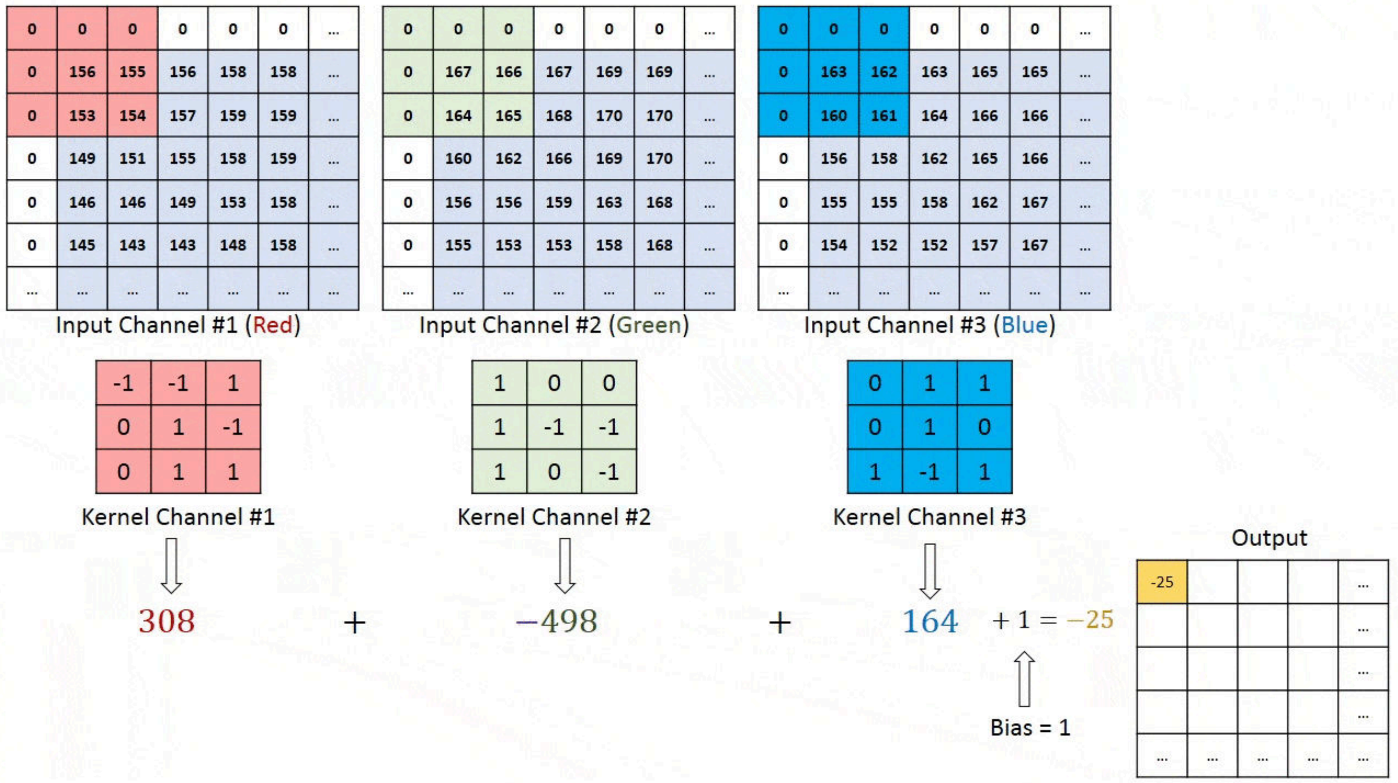


Convolution for 1D array



Made with Gamma

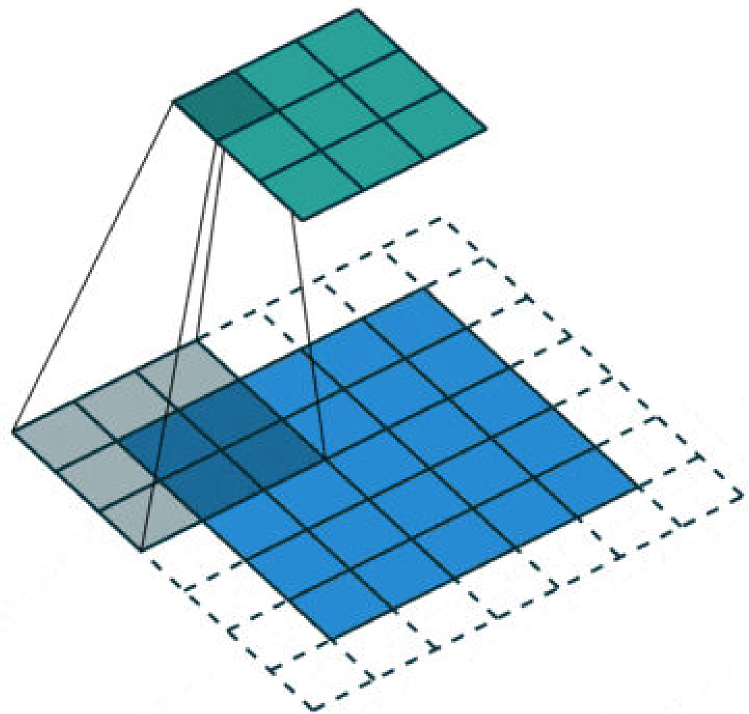
Convolution for 3D array(RGB image)



Made with Gamma

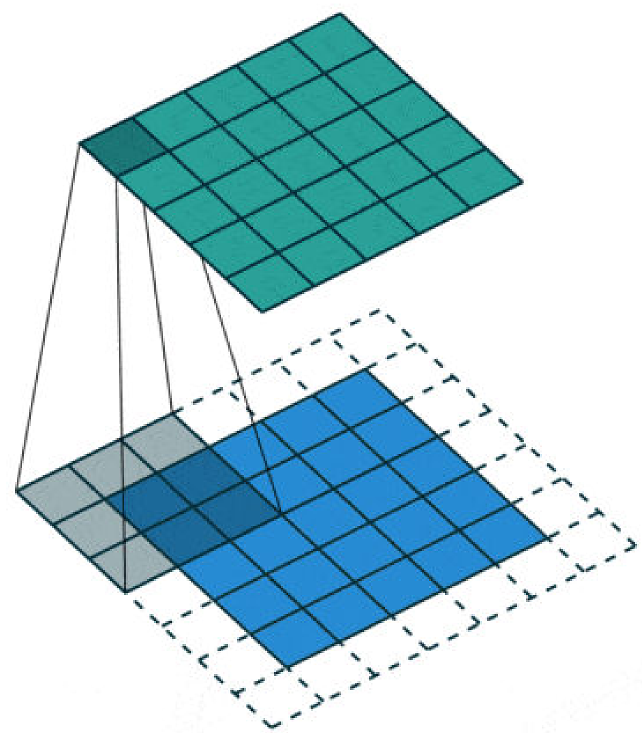
Padding and stride

with padding = 1 and stride = 2



Padding and Stride

with padding = 1 and stride = 1



```
nn.Conv2d(in_channels=16, out_channels=33, kernerl=(3,3), stride=2, padding= 1)
```

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Filters/kernels in CNN

3x3

Size

The most common filter size used in CNNs is 3x3, which captures local dependencies within the input, aiding in feature extraction.

5x5

Diversity

Filters come in various sizes, such as 5x5, providing diversity in feature extraction, allowing for the detection of more complex patterns in the data.

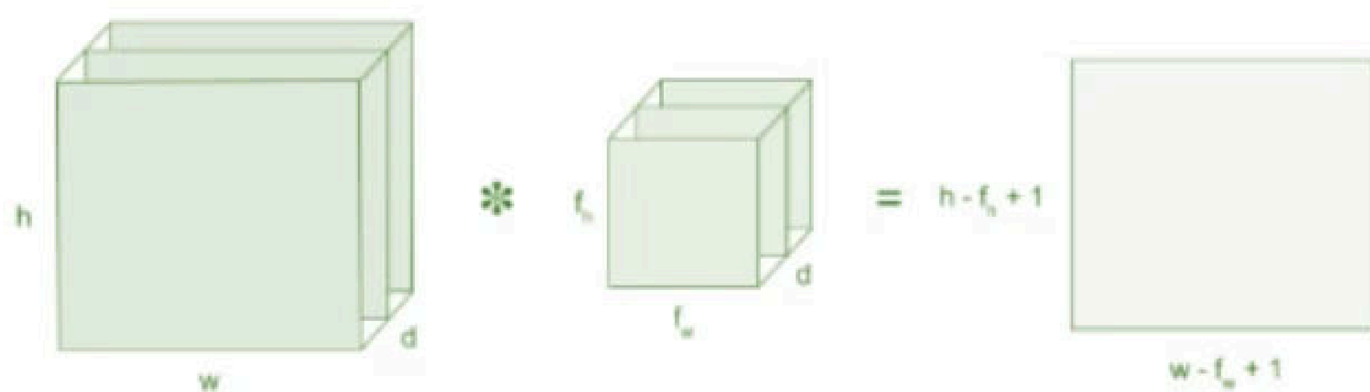
7x7

Specialization

7x7 filters are used to capture larger patterns within the input, specializing in recognizing more global features and structures.

Feature map is the output of the convolution operation, and its dimension can be calculated with this operation

- An image matrix (volume) of dimension **(h x w x d)**
- A filter **(f_h x f_w x d)**
- Outputs a volume dimension **(h - f_h + 1) x (w - f_w + 1) x 1**



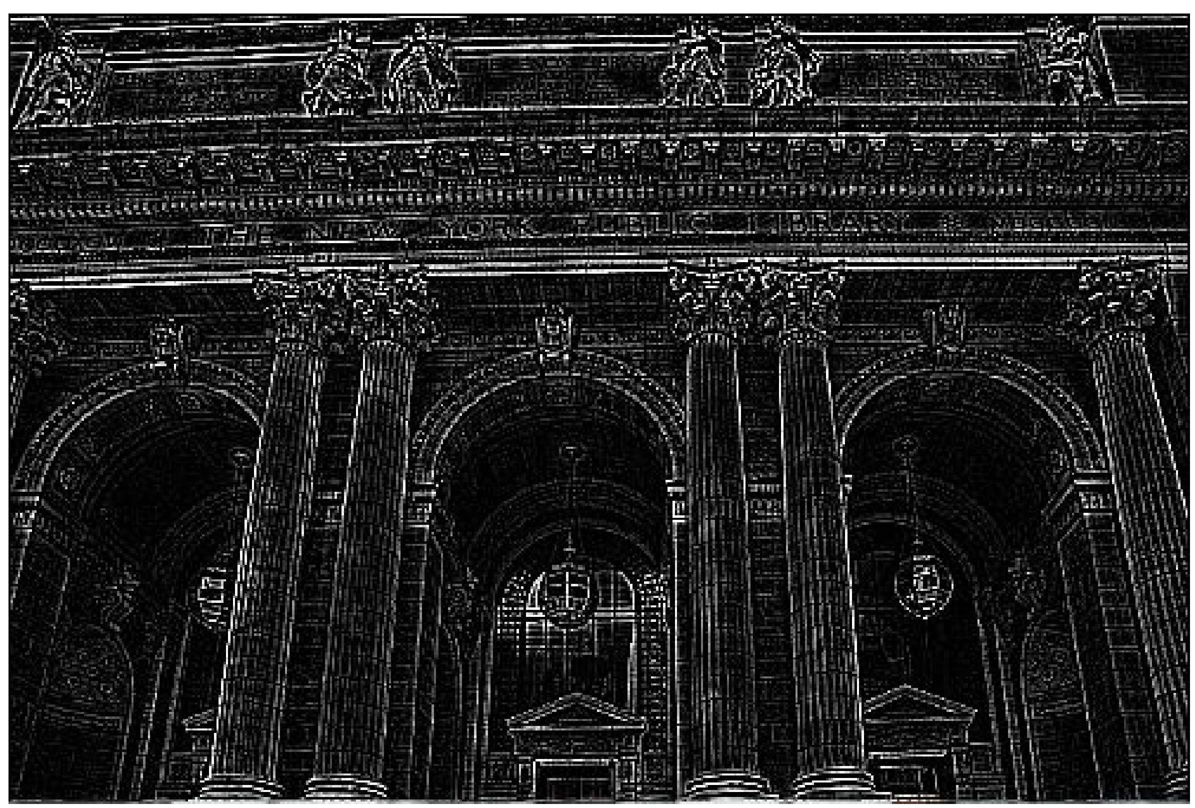
Applying convolution with different kernels

blur kernel

0,0625	0,125	0,0625
0,125	0,25	0,125
0,0625	0,125	0,0625

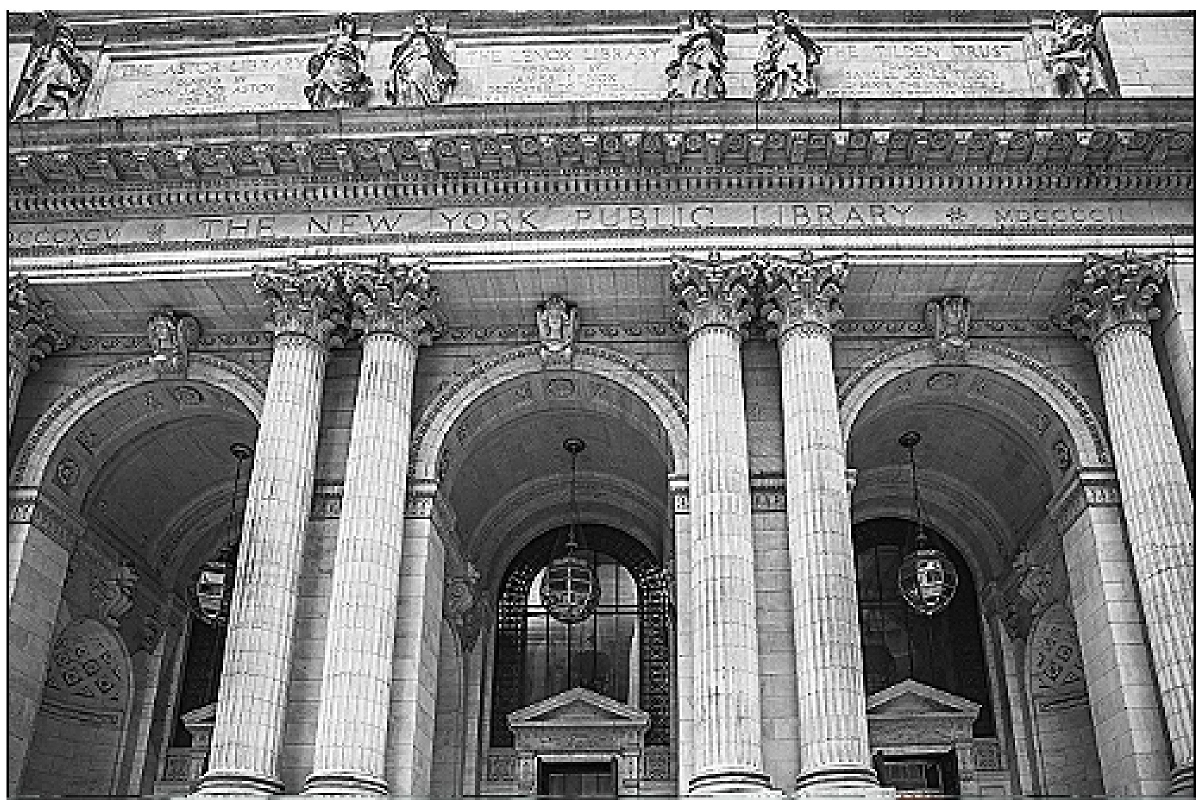
Outline kernel

-1	-1	-1
-1	8	-1
-1	-1	-1



Sharpen kernel

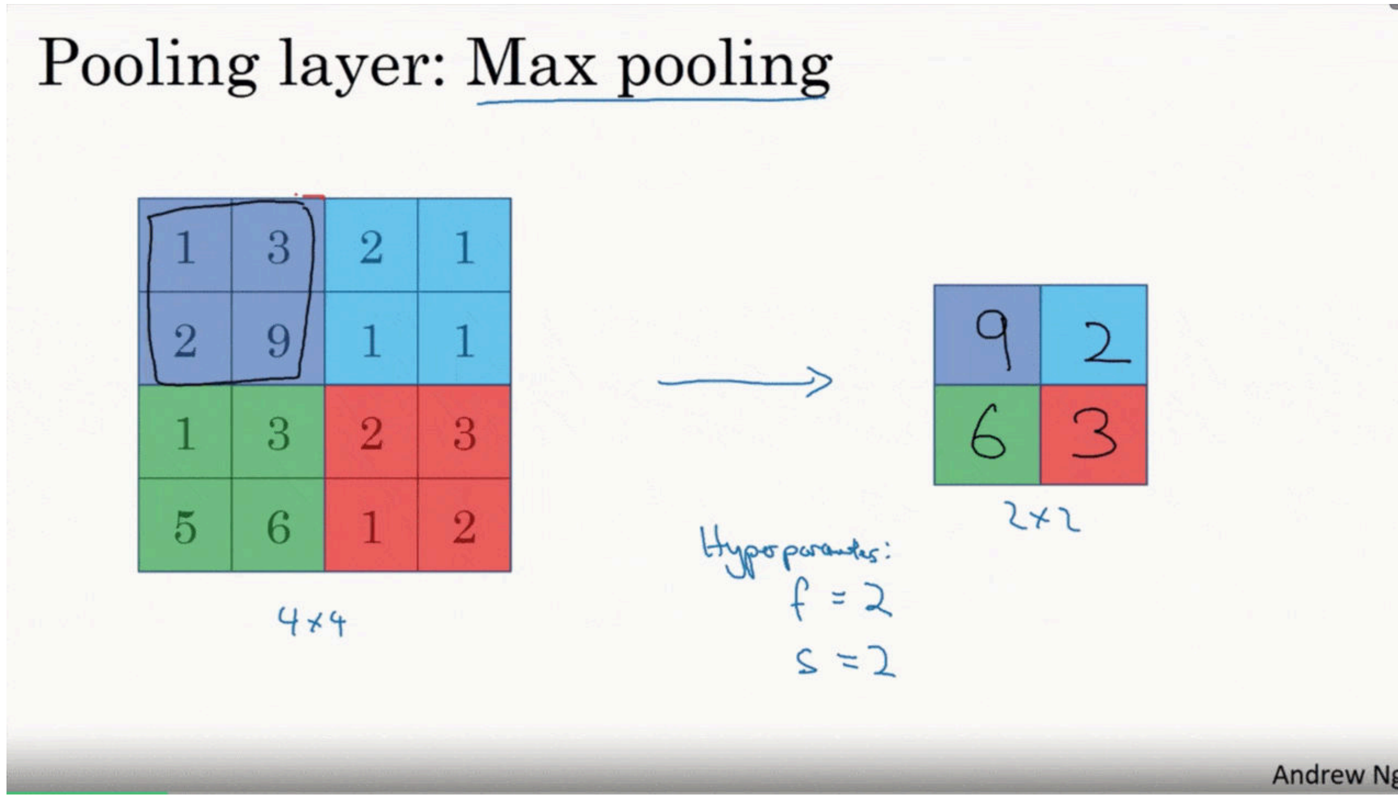
0	-1	0
-1	5	-1
0	-1	0



Pooling

Pooling layers reduce the spatial dimensions of the input, which helps in decreasing the computational load and maintaining the main features.

Max pooling



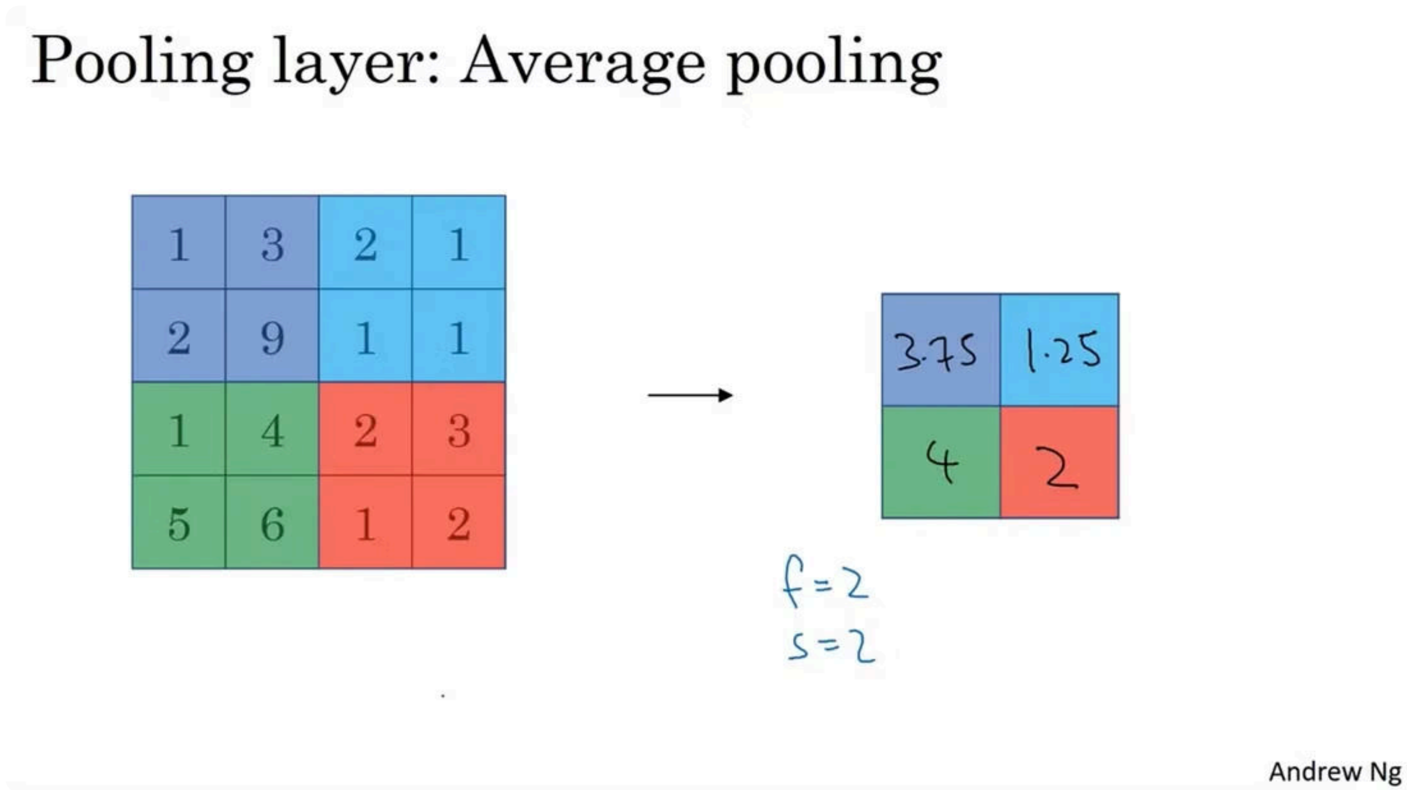

```
nn.MaxPool2d(2, stride=2)
```

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Average pooling




```
nn.AvgPool2d(2, stride=2)
```

Shape:

- Input: (N, C, H_{in}, W_{in}) or (C, H_{in}, W_{in}).
- Output: (N, C, H_{out}, W_{out}) or (C, H_{out}, W_{out}), where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{kernel\_size}[0]}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{kernel\_size}[1]}{\text{stride}[1]} + 1 \right\rfloor$$

Unpooling layer

Nearest neighbor

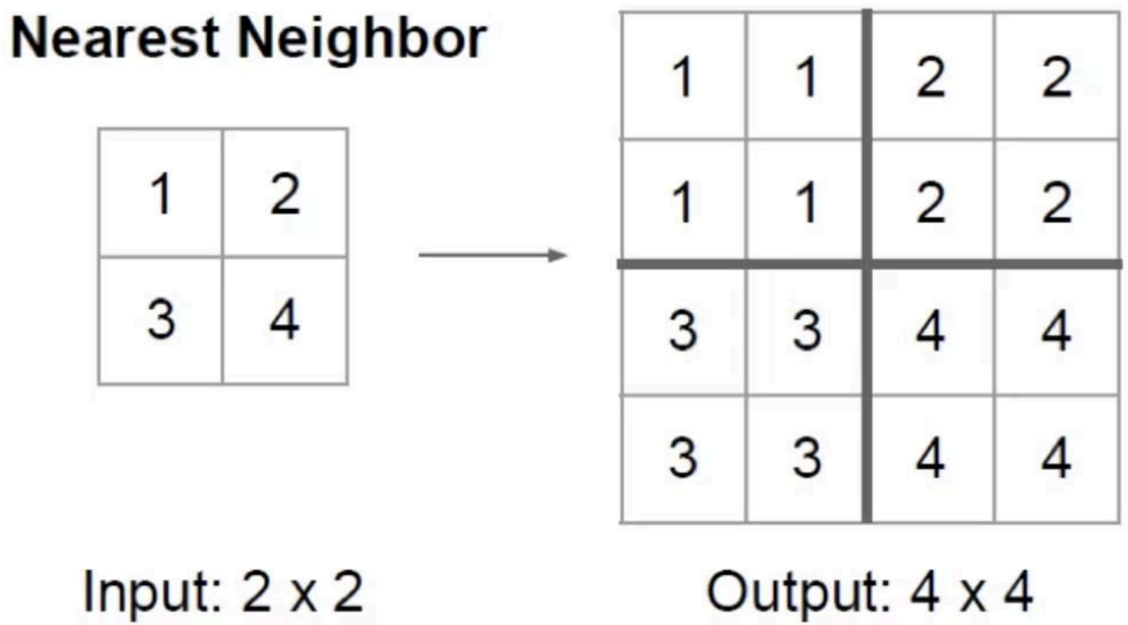
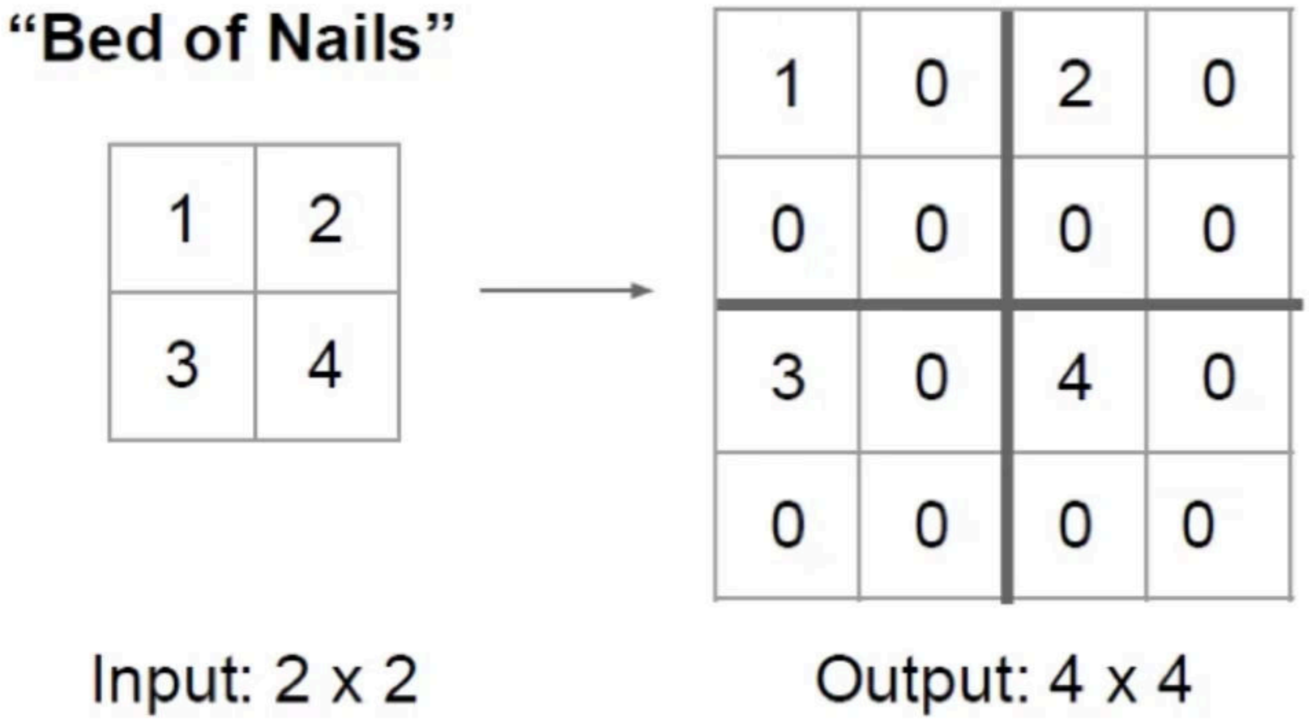
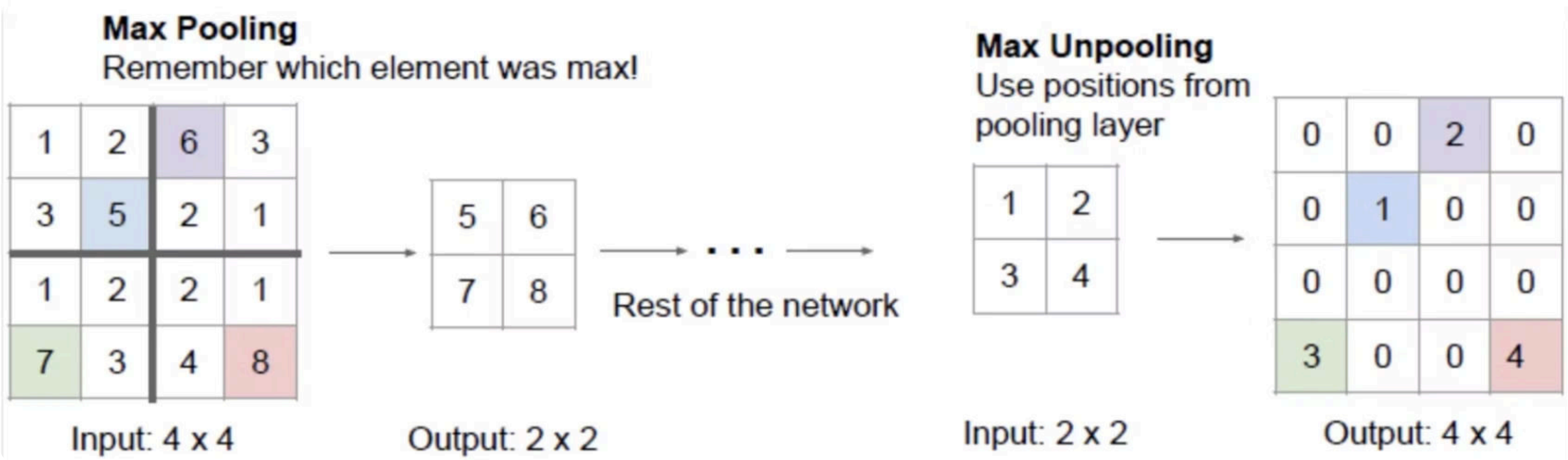


Figure 3. Illustration of Nearest-Neighbor, from [1, 7]

Bed of nails



Max unpooling




```
nn.MaxUnpool2d(2, stride=2)
```

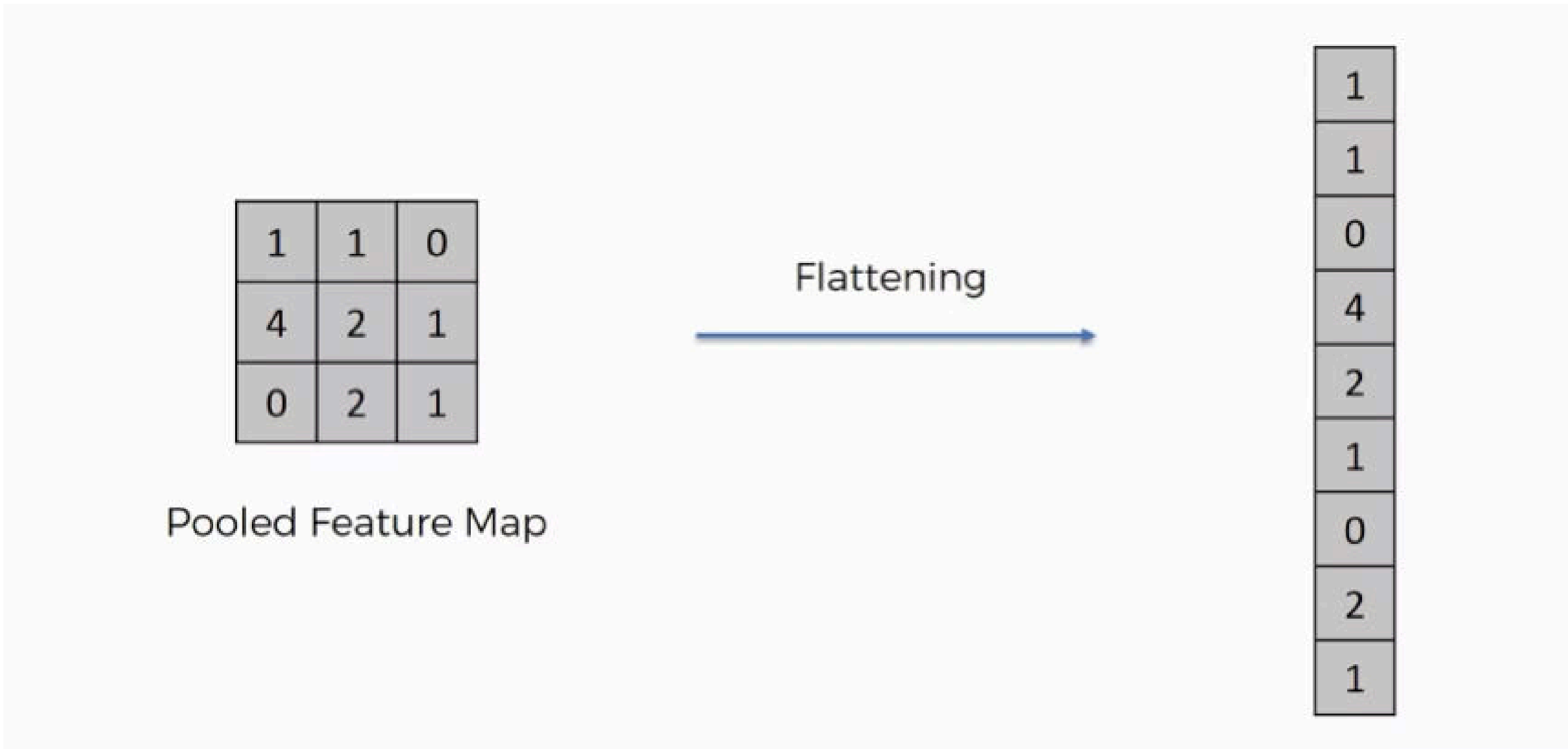
Shape:

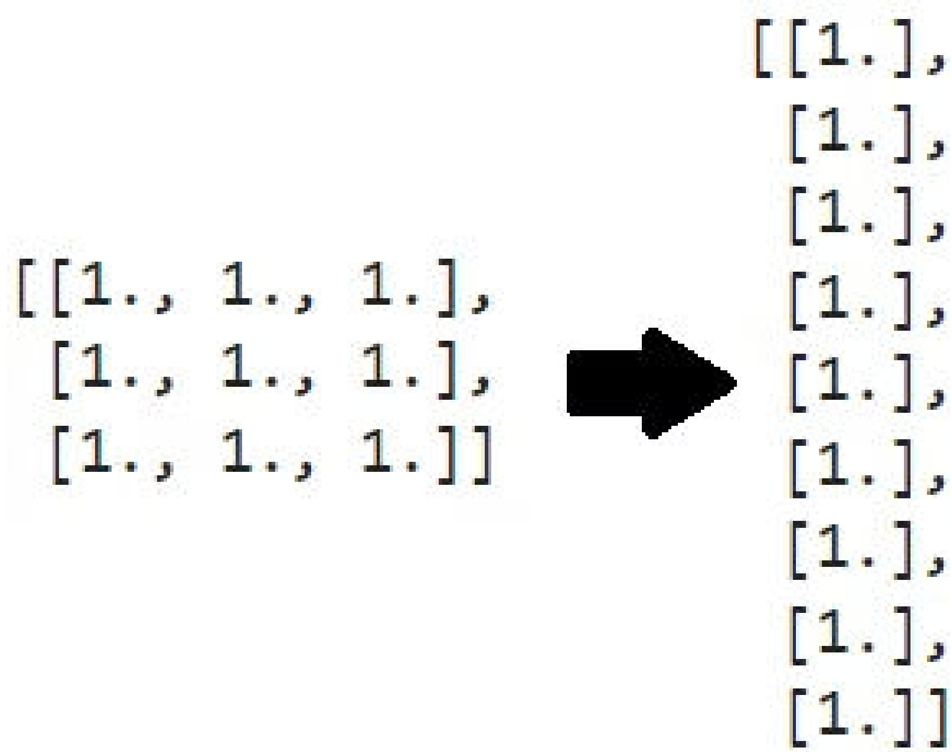
- Input: (N, C, H_{in}, W_{in}) or (C, H_{in}, W_{in}).
- Output: (N, C, H_{out}, W_{out}) or (C, H_{out}, W_{out}), where

$$H_{out} = (H_{in} - 1) \times \text{stride}[0] - 2 \times \text{padding}[0] + \text{kernel\_size}[0]$$

$$W_{out} = (W_{in} - 1) \times \text{stride}[1] - 2 \times \text{padding}[1] + \text{kernel\_size}[1]$$

Flatten






```
torch.flatten(CNN)
```

Hyperparameter/Layer type	What does it do?
Input image(s)	Target images you'd like to discover patterns in
Input layer	Takes in target images and preprocesses them for further layers
Convolution layer	Extracts/learns the most important features from target imagaes
Hidden activation	Adds non-linearity to learned features (non-straight lines)
Pooling layer	Reduces the dimensionality of learned image features
Fully connected layer	Further refines learned features from convolution layers
Output layer	Takes learned features and outputs them in shape of target labels
Output activation	Adds non-linearities to output layer

Activation Functions Used in CNNs

1

Rectified Linear Unit (ReLU)

One of the most commonly used activation functions, ReLU effectively mitigates the vanishing gradient problem and speeds up training.

2

Sigmoid

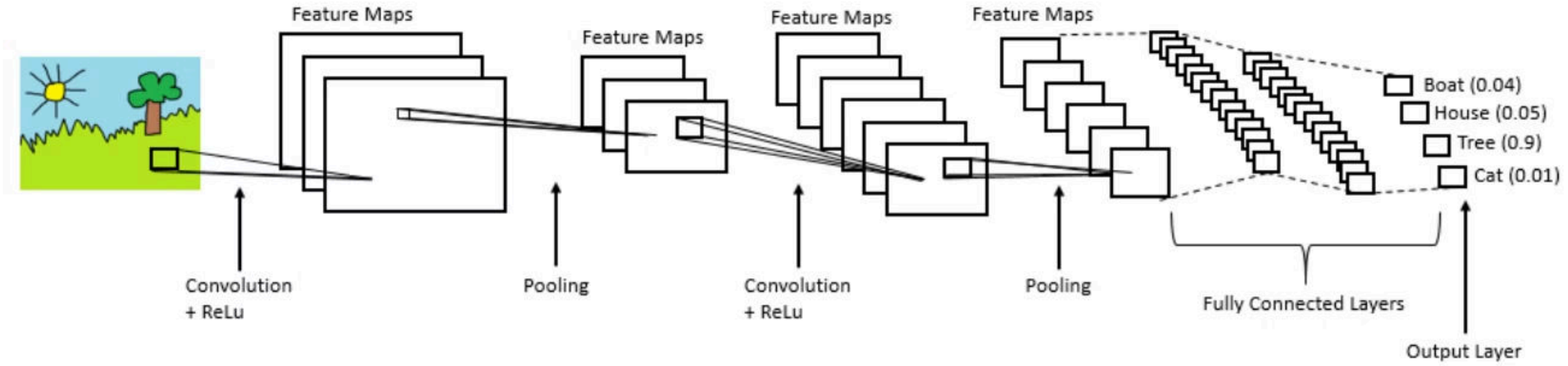
Although less commonly used in CNNs, the sigmoid function is used in binary classification tasks as it squashes input to a range of 0 to 1.

3

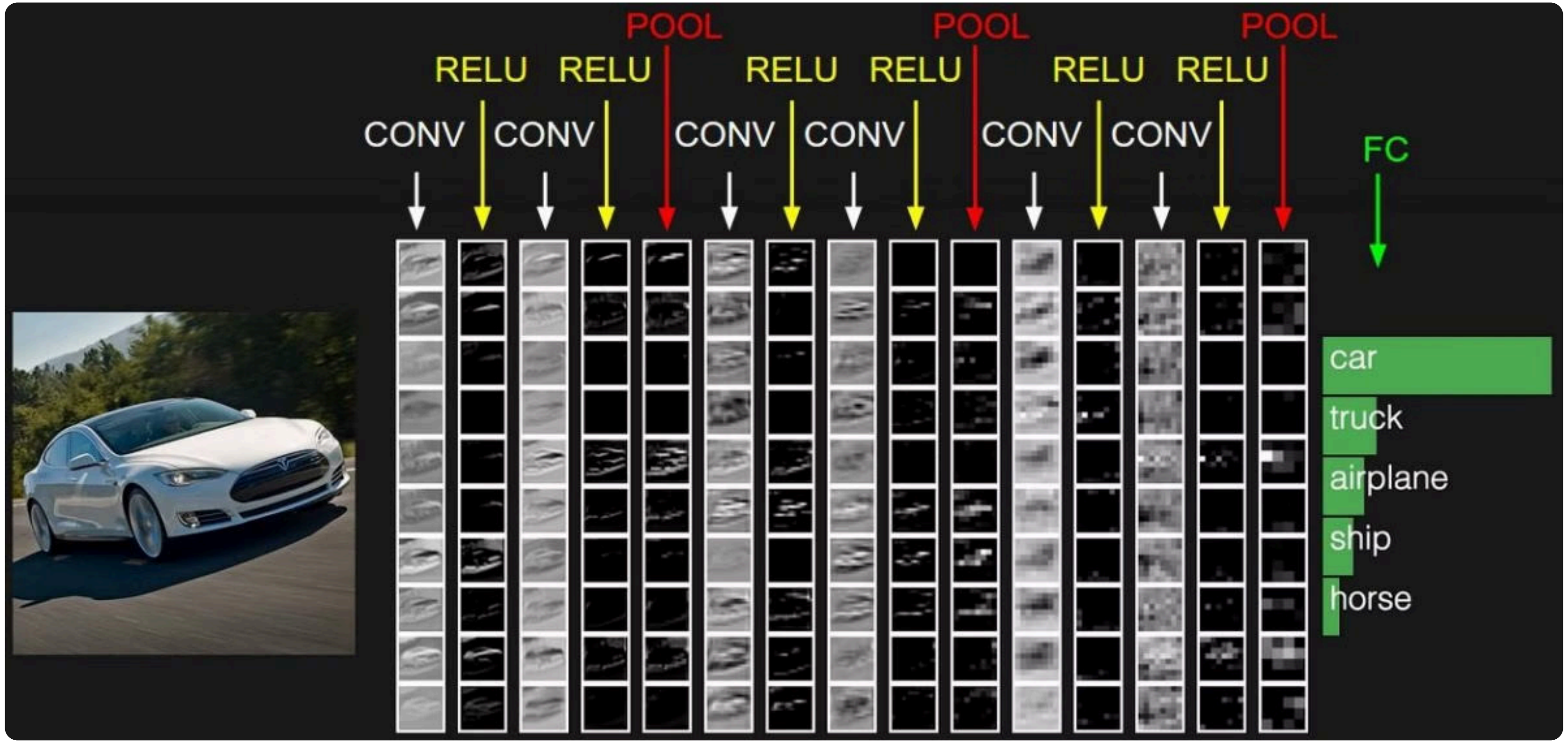
Tanh

Tanh is another activation function that maps input to a range of -1 to 1, allowing for better training in deeper networks.

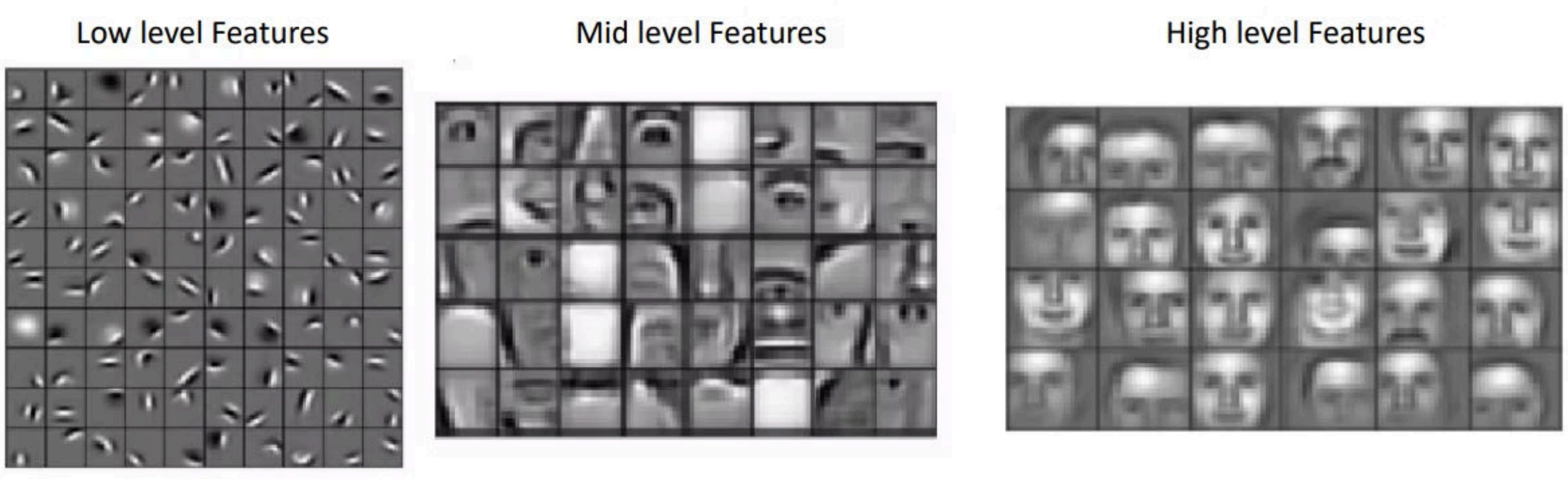
Example convolutional neural network



Example convolutional neural network



The features get more complex as we go deep in the network



Training a CNN

1. **Data Preprocessing**

   Training data is preprocessed to standardize, normalize, and augment it to ensure the network trains on high-quality data.

2. **Optimization Techniques**

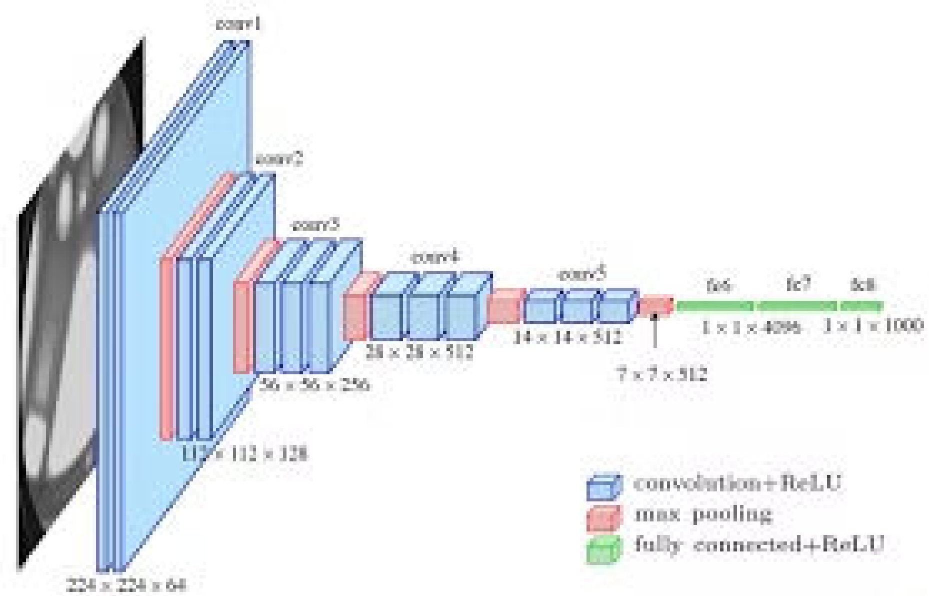
   Various optimization techniques such as gradient descent, learning rate schedules, and momentum are applied to improve the training process.

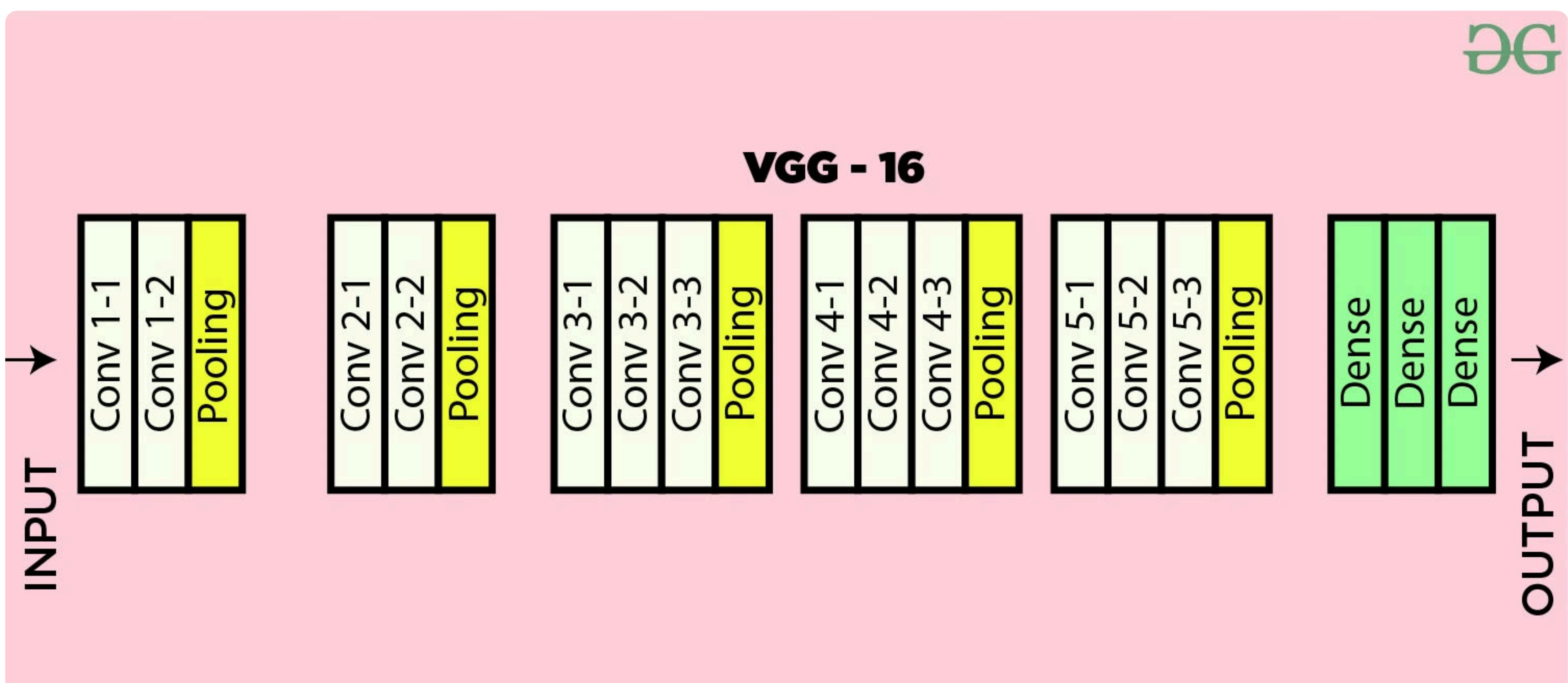
3. **Model Evaluation**

   Trained models need to be rigorously evaluated using metrics like accuracy, precision, recall, and F1 score to ensure their effectiveness.

Some CNN algorithms

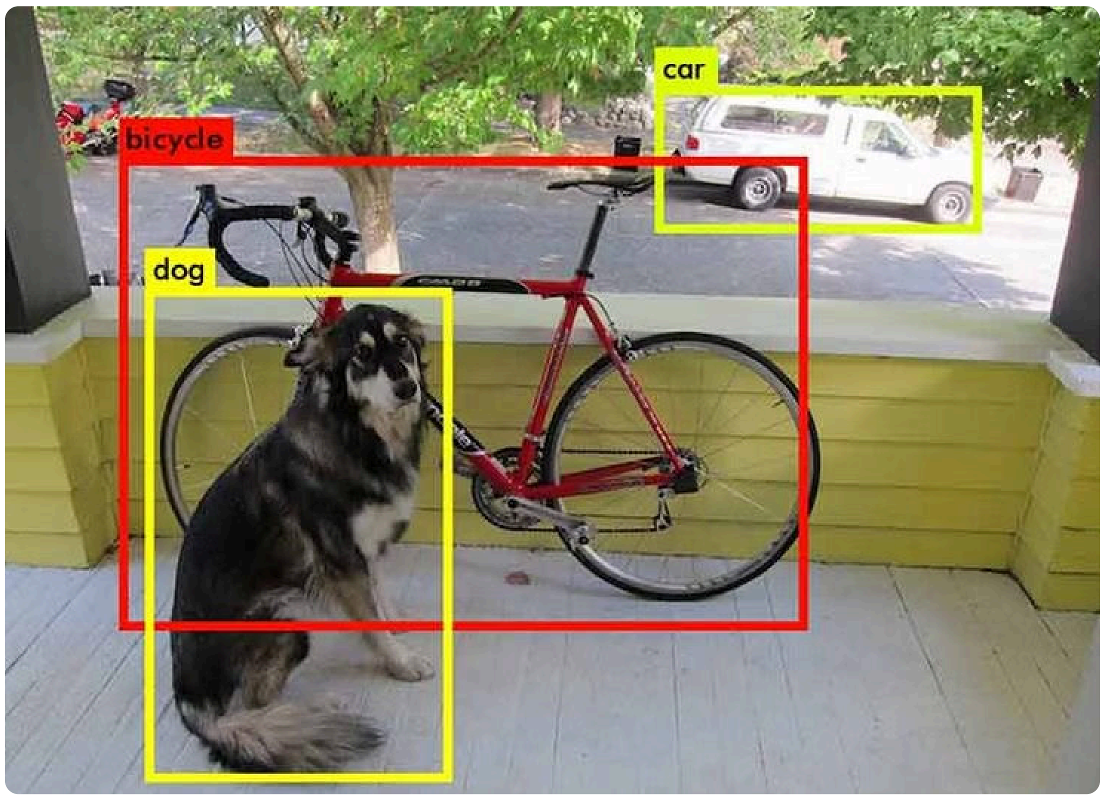
Classification : VGG16

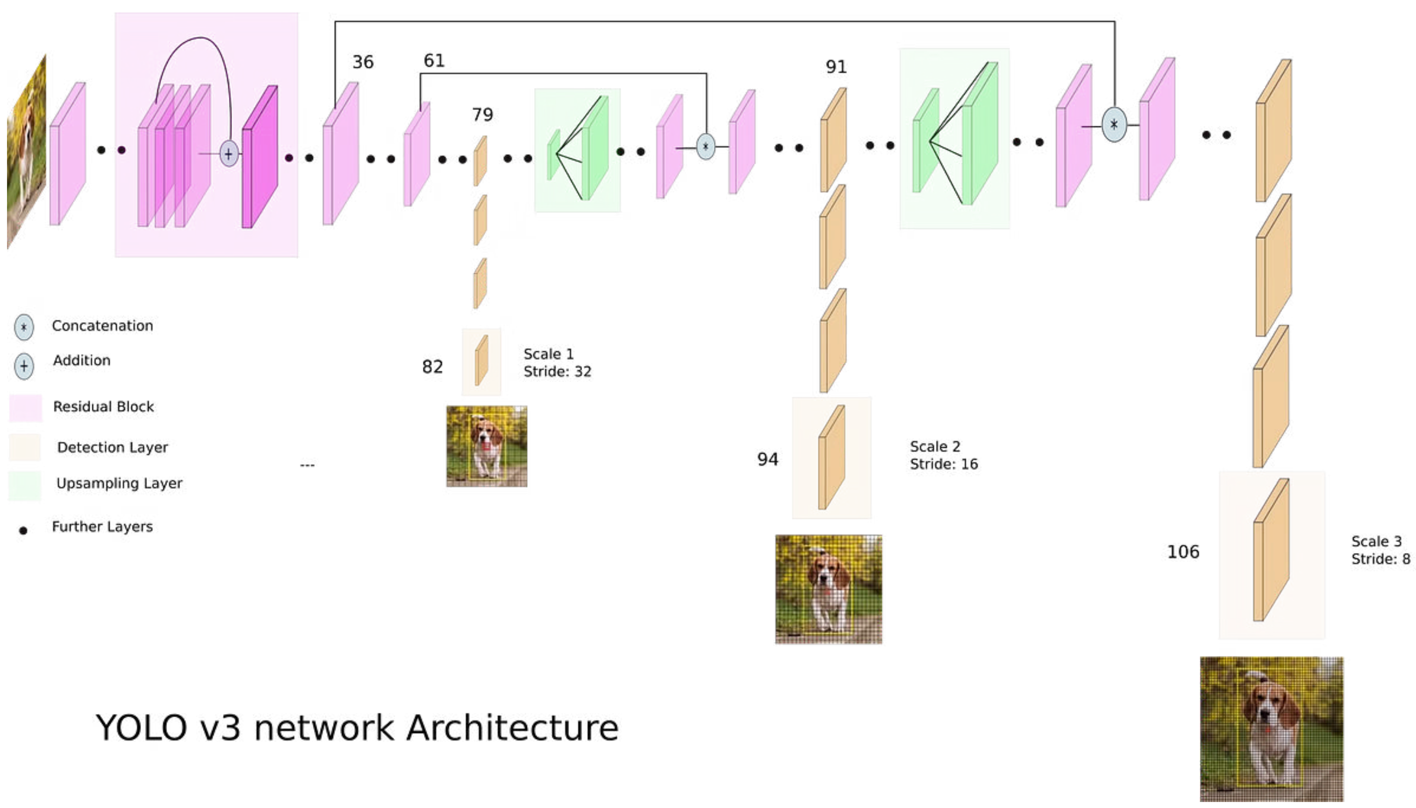




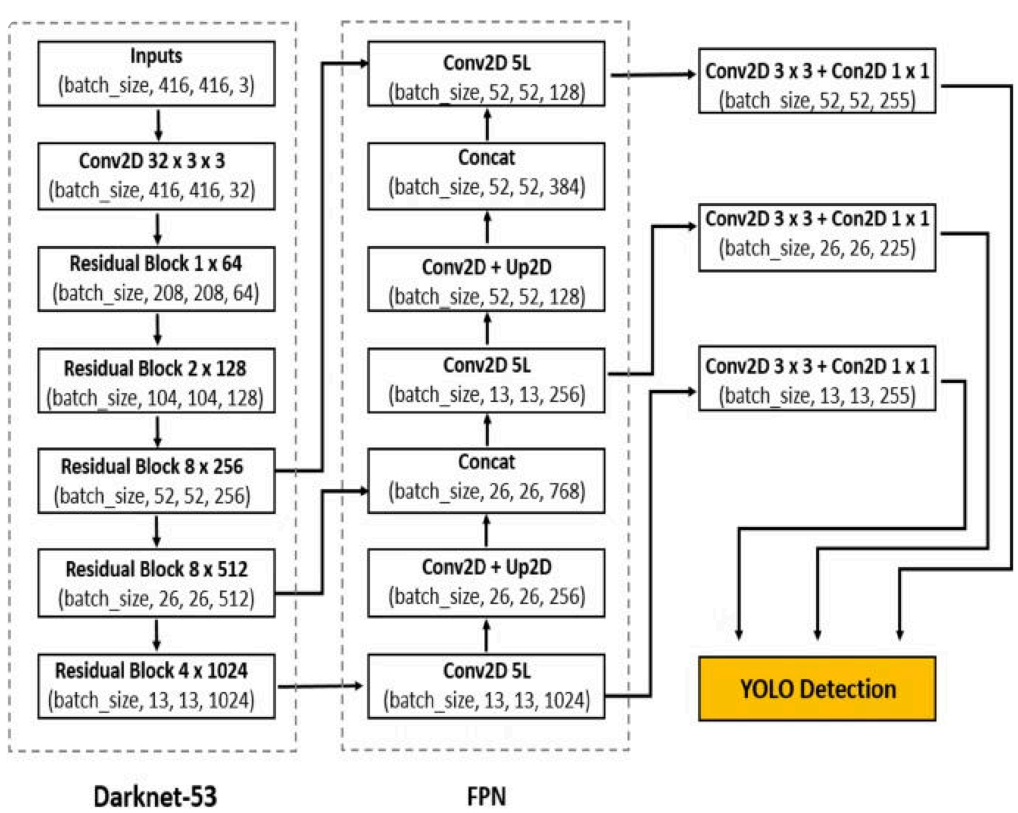
Object Detection :

Yolov3 :





YOLO v3 network Architecture



Segmentation

Unet