CONVOLUTIONAL NEURAL NETWORKS

By,
Roshan Badrinath
(1NT15CS140)

BRIEF INTRODUCTION TO NEURAL NETWORKS

- Neural networks are a computational model that resembles a human brain in which there are many small units working in parallel with no centralized control unit. Smallest unit in a neural network is called as a perceptron.
- A neural network architecture consists of a certain number of layers. Each layer consists of a certain number of neurons. Each neuron is connected to several other neurons.
- The algorithm used for training a neural network is called Backpropogation.
- The process of training a neural network is done by readjusting the weights and biases.
- An image of height 32 pixels and width 32 pixels with 3 channels (RGB) will require 3072 weights per neuron in the first hidden layer.
- A normal image of a resolution of 300 x 300 would require 270,000 weights per neuron in the first hidden layer.
- This will create a huge number of parameters to train for the complete neural network.

CONVOLUTIONAL NEURAL NETWORKS (CNN)

- Convolution neural network's are designed to take images as their input. Different types of hidden layers are used to train efficiently.
- Neurons in a CNN can be arranged in a three-dimensional structure as width (Image width in pixels), height (Image height in pixels) and depth (RGB channels).
- A CNN consists of three major groups of layers:
  a. Input layer
  b. Feature-extraction layers
  c. Classification layers
- The two main types of hidden layers are:
  a. Convolution layers with ReLU activation function
  b. Pooling layers

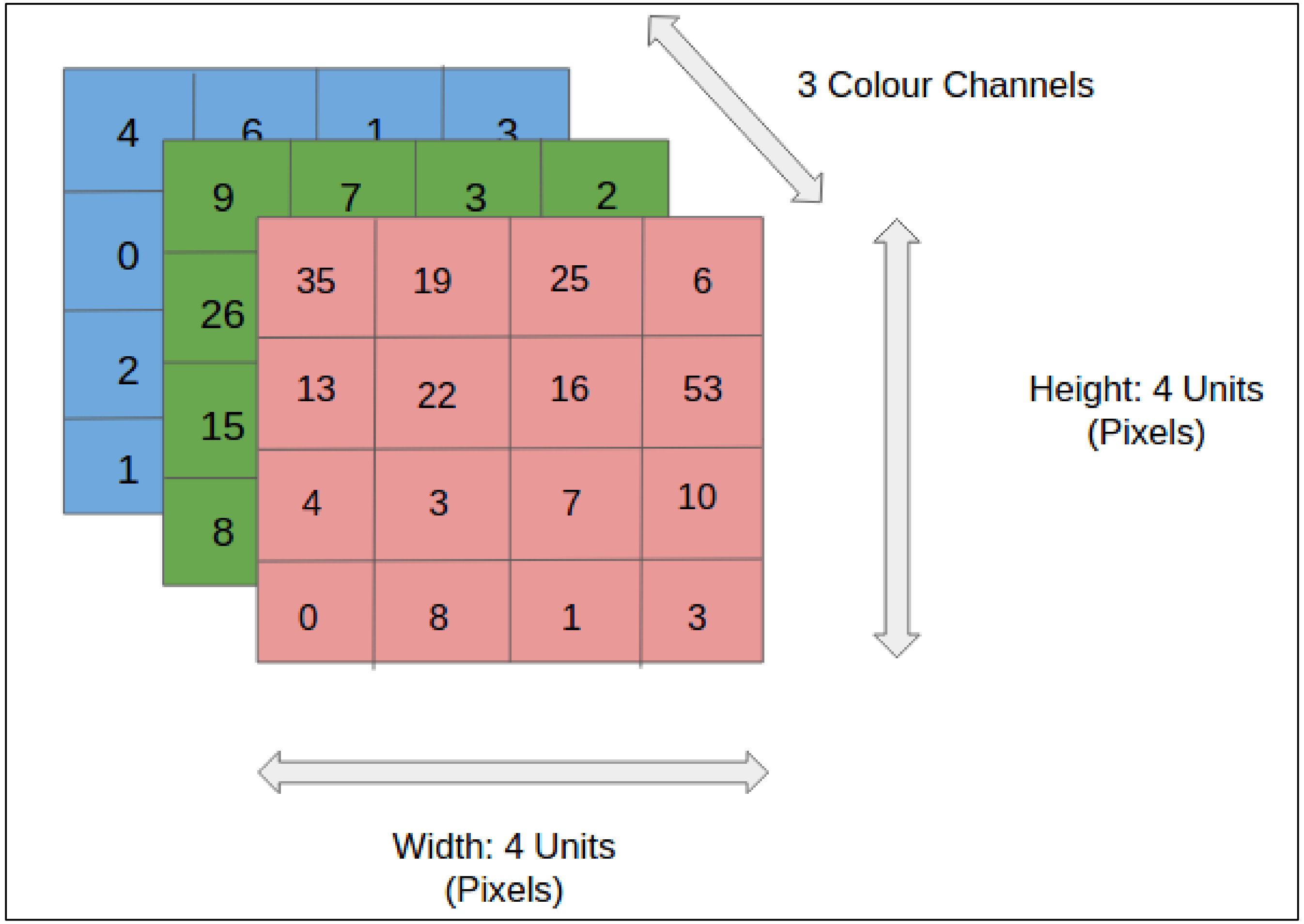


Fig 1: 3-D input given to a CNN

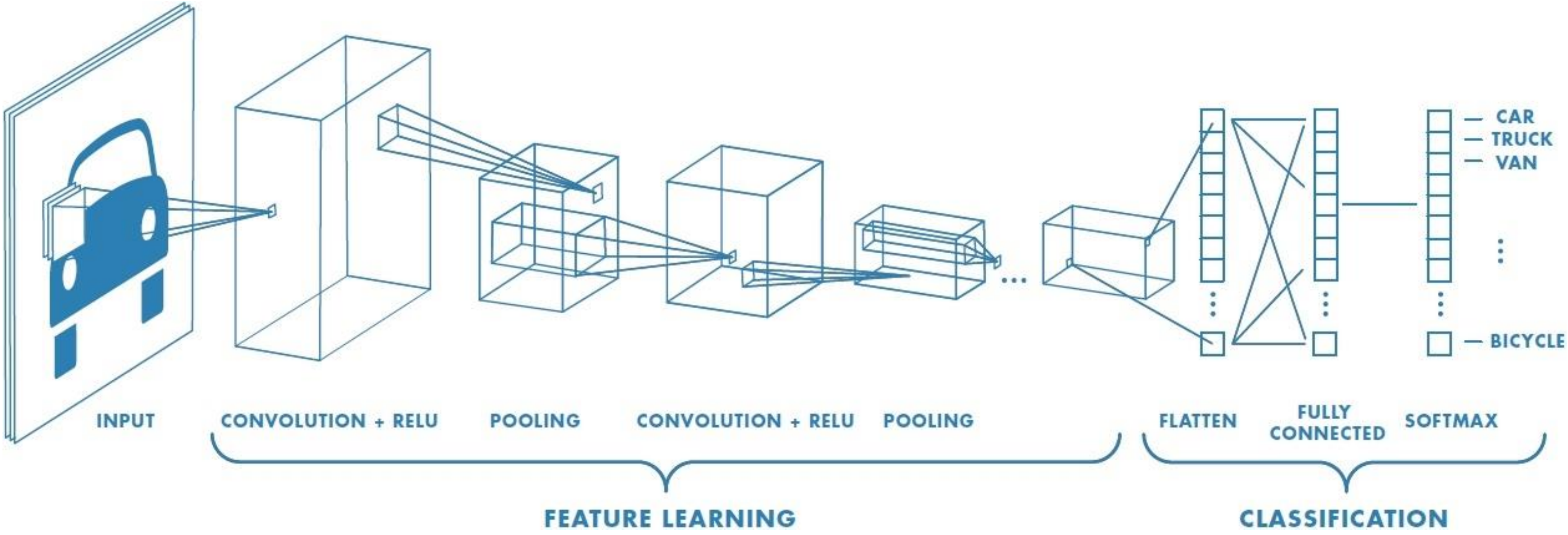


Fig 2: Major groups of layers present in CNN

CONVOLUTION LAYER

- Convolution is a mathematical operation of merging two sets of information.
- In the convolution layer, a neuron is **locally connected** to a patch of neurons in the previous layer. The matrix formed by these neurons is called as a **receptive field**.
- The locally connected weights are called as **filters** or **kernels**.
- This layer will compute a **dot product** between all **receptive fields** and the **kernel**. Hence the **parameters are shared.**
- The size of kernel is very much less than the size of the input. Hence there are less parameters to be learned.
- Each kernel can be associated with one bias value.
- The output of the layer results in a **feature map**, also called as **activation map**.
- When a filter "activates", it means that the filter lets information pass through it.

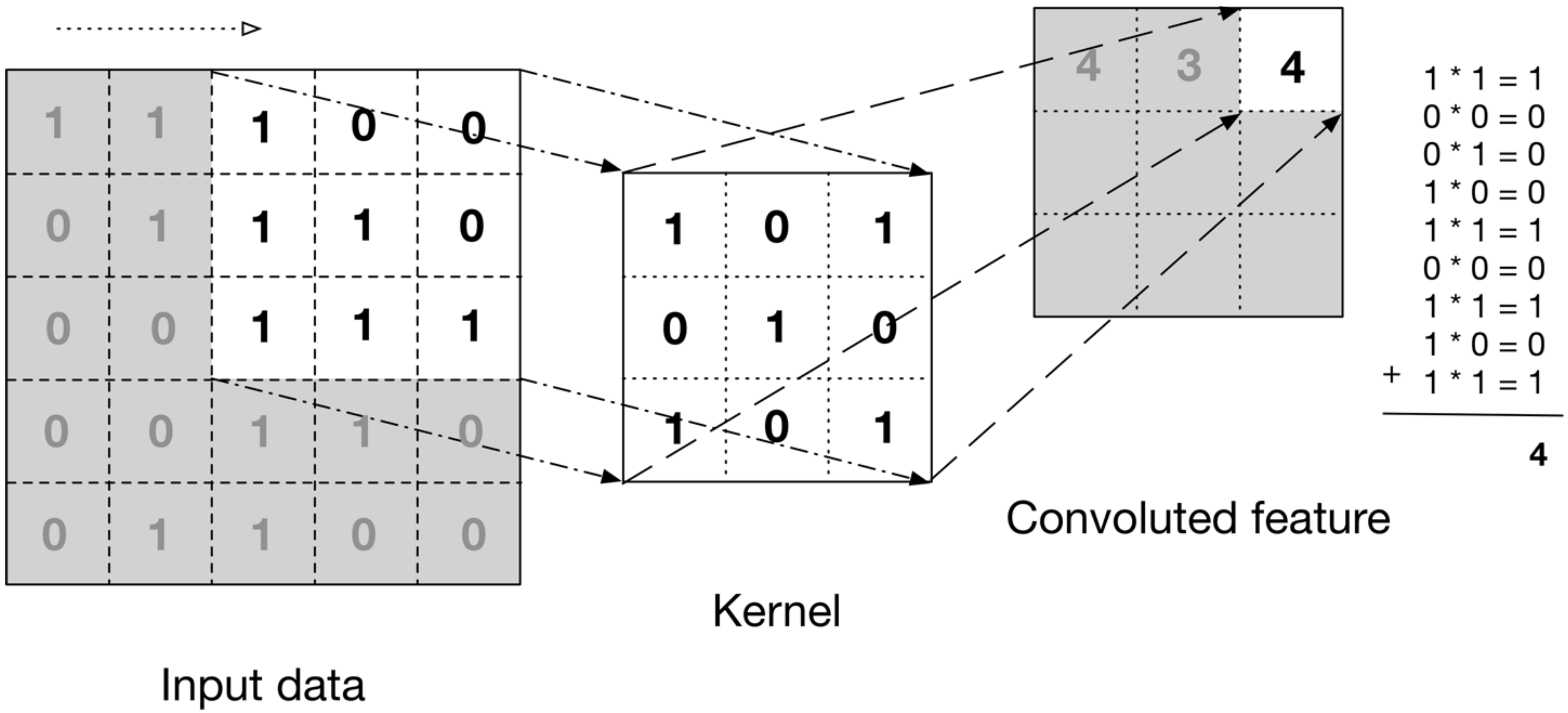
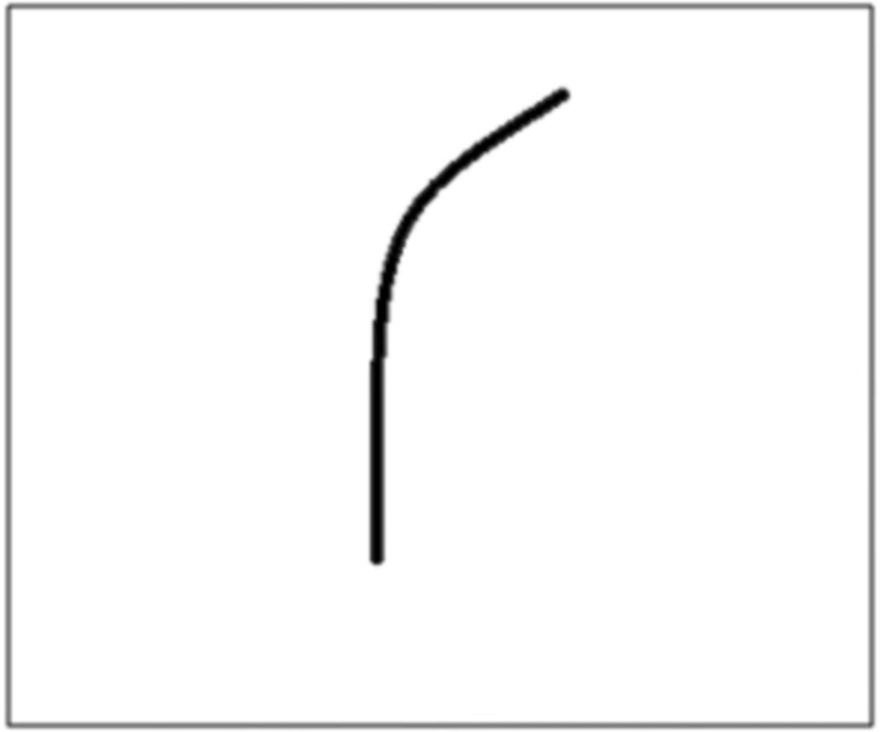


Fig 3: Convolution operation using kernels/filters.

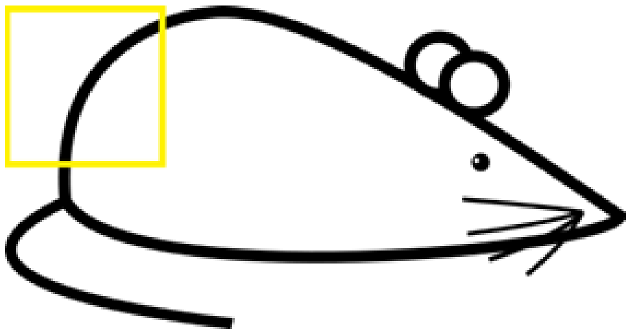
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

Fig 4: Example of a trained filter or a feature learned



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

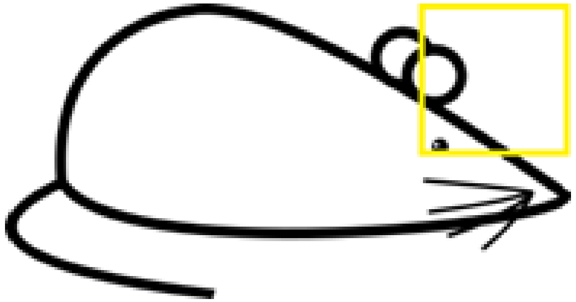
*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Fig 5: Example of a filter allowing the information to pass as the dot product is a large value.



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

Fig 6: Example of a filter not allowing the information to pass as the dot product is zero.

RECTIFIED LINEAR UNIT (ReLU)

- Rectified Linear Unit is a non-linear activation function which is mostly used in Convolutional neural networks.

$$ReLU(x) = \max(0, x)$$

- Unlike other non-linear functions such as *tanh(x)* and *sigmoid(x)*, *ReLU(x)* increases the non linear properties of the overall network without affecting the receptive fields of the convolution layer.
- ReLU, compared to other functions, trains the neural network faster without affecting the accuracy.
- The final output after Convolution Layer + ReLU is as follows:

$$O = ReLU\left(\sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} x_{ij} + b \right)$$

Where "I" and "j" are row and column indices respectively of receptive field input "x" and kernel input "w". Bias associated with the kernel is "b".

POOLING LAYER

- Pooling layer is generally inserted between successive convolution layers.
- Pooling layer is used after convolution layer to progressively reduce the special size of the data.
- By reducing the special size, polling layer helps control the problem of overfitting.
- Pooling layers use filters to perform the downsampling process on the input volume.
- MaxPooling is the most common pooling operation. With a 2×2 filter size, the MaxPooling operation is taking the largest of four numbers in the filter area.
- If a MaxPooling filter of size 2 x 2 is applied to an input of size 32 x 32 x 3, an output of size 16 x 16 x 3 is obtained.
- The depth of the input is not affected.

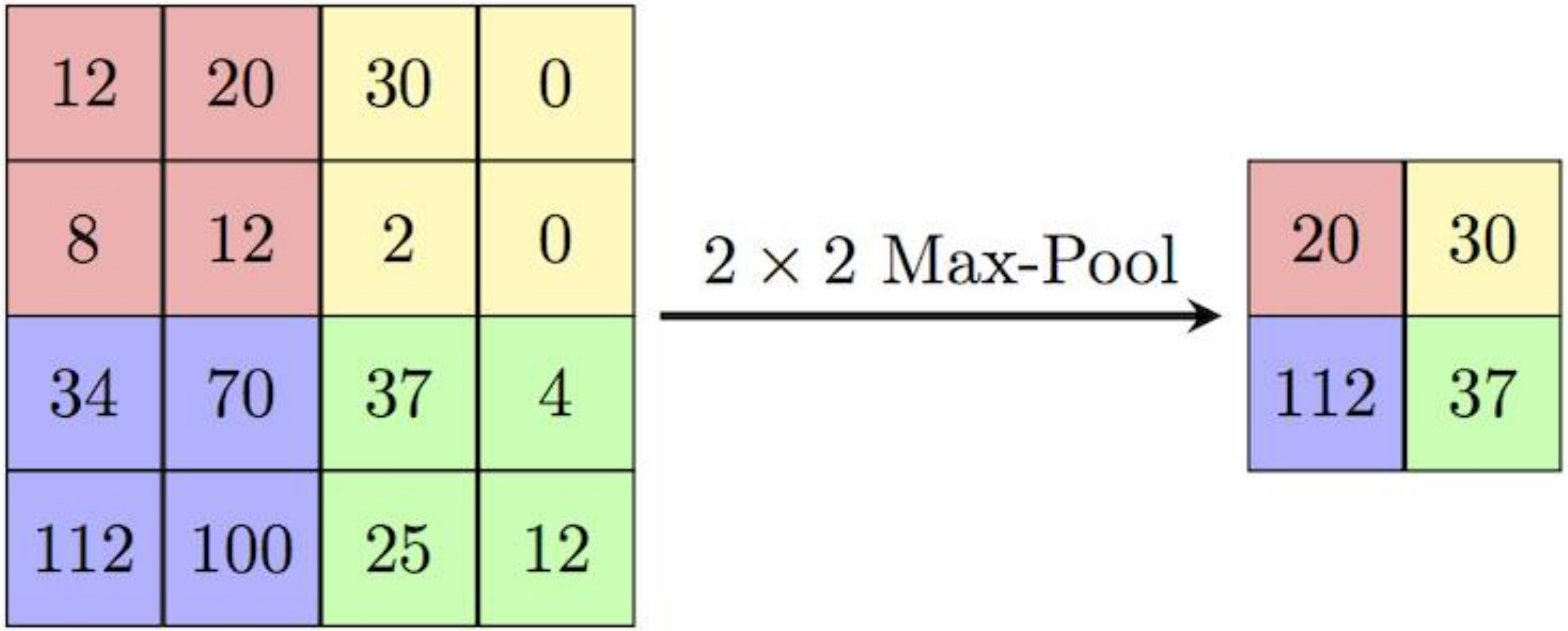


Fig 7: MaxPooling operation with filter of size 2 x 2.

HYPERPARAMETERS

In convolution layer

1. Number of filters/kernels
2. Size of the filter/kernel
3. **Stride:** The amount by which the filter/kernel shifts is called as stride.
4. **Padding:** The process of adding zeroes to input volume around the border to control the special size of the output volume is called as padding. **Zero-padding** is used to obtain an output volume of same size as the input volume.

In pooling layer:

1. Size of the filter/kernel

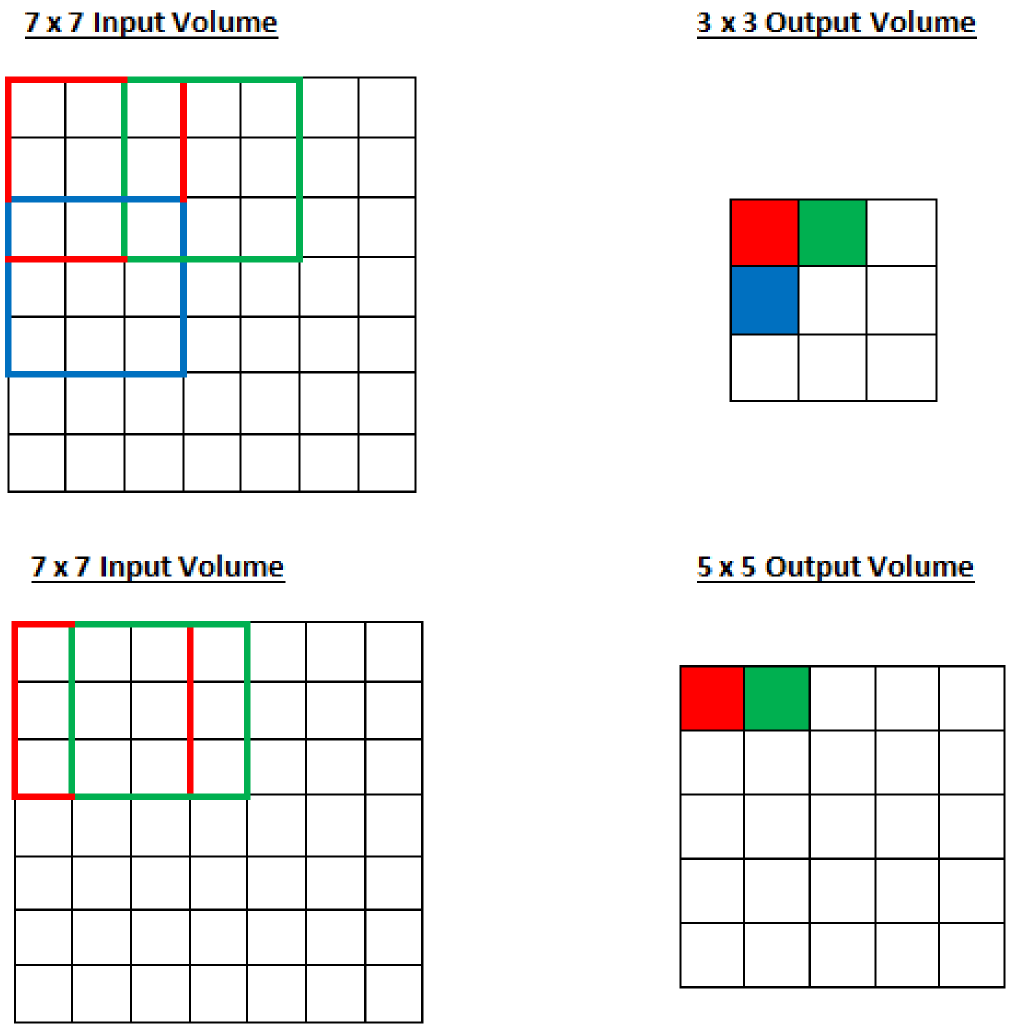


Fig 8: (above) Stride is equal to 2. (below) Stride is equal to 1.

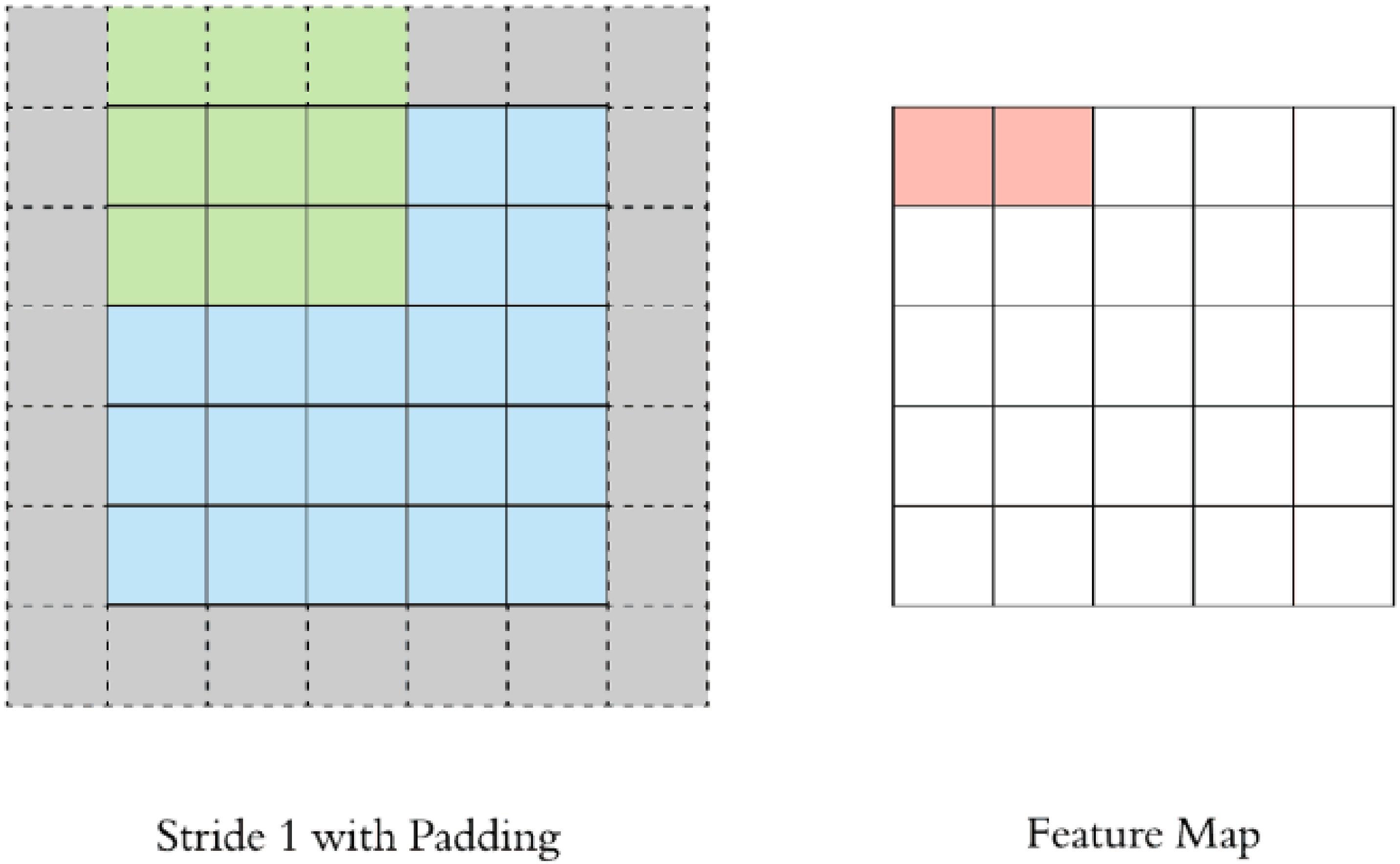


Fig 9: Example of zero-padding for stride equals1.

SUMMARY INFORMATION

- Let the input volume be of size $W1 \times H1 \times D1$ and the output volume be of size $W2 \times H2 \times D2$. Let the hyperparameters be as follows:
  - Number of filters K
  - Spatial extent F (Equal height and width)
  - Stride S
  - Padding P
- For Convolution layer:
  - $W2 = (W1 - F + 2P)/S + 1$
  - $H2 = (H1 - F + 2P)/S + 1$
  - $D2 = K$
- For Pooling layer:
  - $W2 = (W1 - F)/S + 1$
  - $H2 = (H1 - F)/S + 1$
  - $D2 = K$

FULLY-CONNECTED LAYERS

- After high level features are extracted from convolution layers and pooling layers, the feature maps are flattened and are given to a series of fully connected layers.
- These fully connected layers are meant for classification.
- The final/output layer produces a 1-D vector of size N, where N is the number of classes. **Softmax** activation function is generally used with this layer.
- The final output vector consists of probability values.

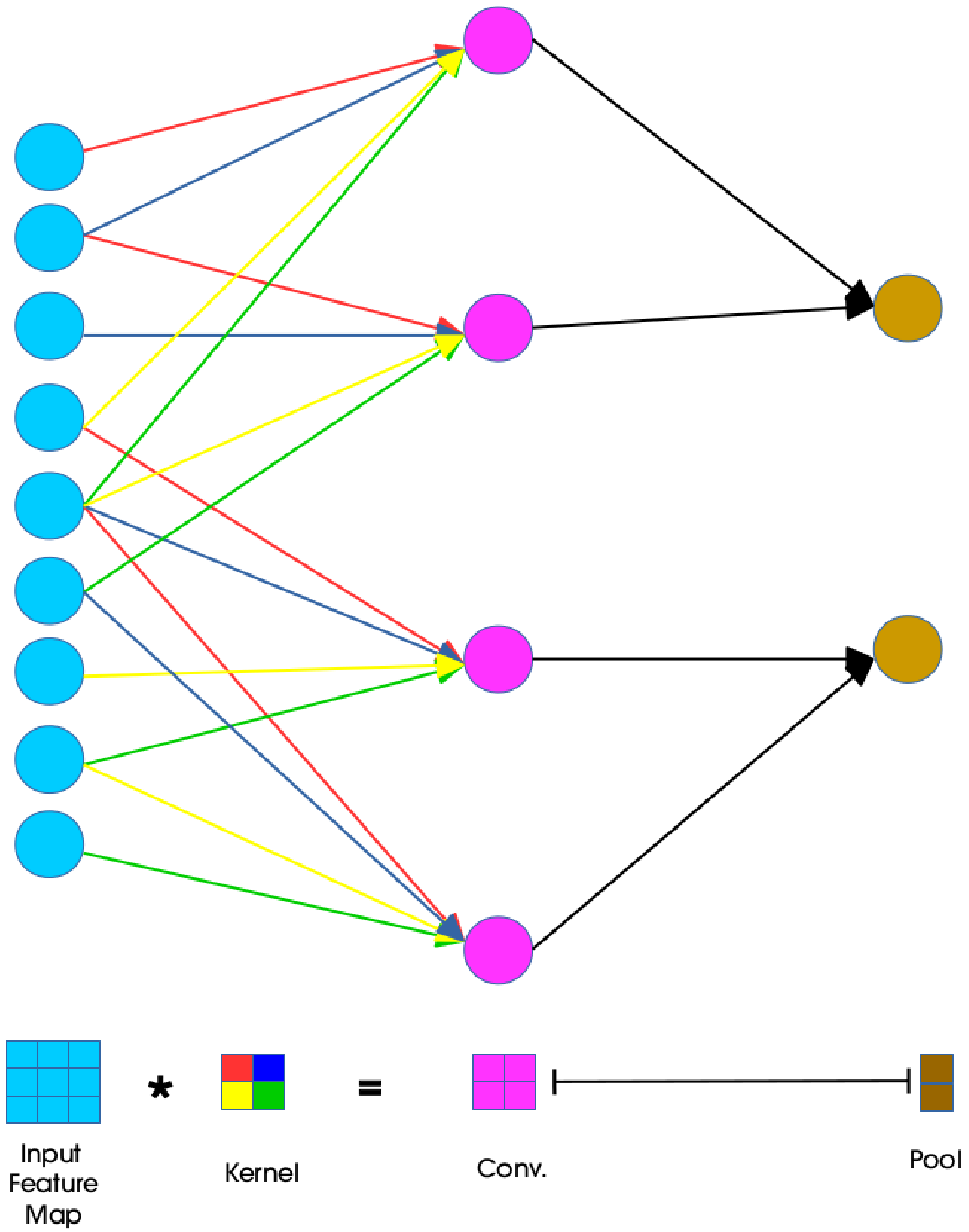


Fig 10: Local connectivity of CNN

$$
\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}
\quad
\begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix}
\quad
\begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix}
$$

$$\partial h_{ij} \text{ represents } \frac{\partial L}{\partial h_{ij}}$$

$$\partial w_{ij} \text{ represents } \frac{\partial L}{\partial w_{ij}}$$

$$
\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}
\quad
\begin{bmatrix}
\begin{matrix} X_{11}\partial h_{11}+ \\ X_{12}\partial h_{12}+ \\ X_{21}\partial h_{21}+ \\ X_{22}\partial h_{22} \end{matrix} &
\begin{matrix} X_{12}\partial h_{11}+ \\ X_{13}\partial h_{12}+ \\ X_{22}\partial h_{21}+ \\ X_{23}\partial h_{22} \end{matrix} \\
\begin{matrix} X_{21}\partial h_{11}+ \\ X_{22}\partial h_{12}+ \\ X_{31}\partial h_{21}+ \\ X_{32}\partial h_{22} \end{matrix} &
\begin{matrix} X_{22}\partial h_{11}+ \\ X_{23}\partial h_{12}+ \\ X_{32}\partial h_{21}+ \\ X_{33}\partial h_{22} \end{matrix}
\end{bmatrix}
\quad
\begin{bmatrix} \partial h_{11} & \partial h_{12} \\ \partial h_{21} & \partial h_{22} \end{bmatrix}
$$

$$\partial W_{11} = X_{11}\partial h_{11} + X_{12}\partial h_{12} + X_{21}\partial h_{21} + X_{22}\partial h_{22}$$

$$\partial W_{12} = X_{12}\partial h_{11} + X_{13}\partial h_{12} + X_{22}\partial h_{21} + X_{23}\partial h_{22}$$

$$\partial W_{21} = X_{21}\partial h_{11} + X_{22}\partial h_{12} + X_{31}\partial h_{21} + X_{32}\partial h_{22}$$

$$\partial W_{22} = X_{22}\partial h_{11} + X_{23}\partial h_{12} + X_{32}\partial h_{21} + X_{33}\partial h_{22}$$

Fig 11: Backpropogation in CNN

```python
import keras
from keras.models import Sequential
from keras.layers import Activation
from keras.layers.core import Dense, Flatten
from keras.layers.convolutional import *
from keras.layers.pooling import *
```

```python
model = Sequential([
        Conv2D(2, kernel_size=(3, 3), input_shape=(20,20,3),activation='relu', padding='same'),
        Conv2D(3, kernel_size=(3, 3), activation='relu', padding='same'),
        Flatten(),
        Dense(2, activation='softmax'),
    ])
```

```python
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_20 (Conv2D)           (None, 20, 20, 2)         56
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 20, 20, 3)         57
_________________________________________________________________
flatten_9 (Flatten)          (None, 1200)              0
_________________________________________________________________
dense_14 (Dense)             (None, 2)                 2402
=================================================================
Total params: 2,515
Trainable params: 2,515
Non-trainable params: 0
```

Fig 12: Building a CNN using Keras

THANK YOU!