Introduction to Convolutional Neural Network (CNN)

Resource(s):

Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courvillellet, *a*n MIT Press book.

An Introduction to Convolutional Neural Networks, Alessandro Giusti Dalle Molle Institute for Artificial Intelligence, Lugano, Switzerland

https://en.wikipedia.org/wiki/Convolutional_neural_network.

Convolutional Neural Networks

- **Convolutional Networks** or **Convolutional Neural Networks** (**CNNs**) (LeCun, 1989), are a specialized kind of neural network for processing data in the form of a **grid topology**
  - For example, an **image** can be thought of as a **2-D grid of pixels**.
- **CNNs** have been tremendously successful in practical applications (especially in image recognition).
  - **CNN** stands on ***neuro-scientific principles*** influenced by **deep learning methodology**.

CNN an Introduction

- A **CNN** is a **deep learning algorithm** that can take in an **input image**, assign **kernel** (**learnable weights** and **biases**) to various aspects/objects in the **image**, and differentiate one from the other.
- The ***pre-processing*** of input data (called **convolution**) is required in a **CNN** that generates an *affine transformation* of data, which is not common in other classification algorithms.

Convolutional Neural Networks

*https://en.wikipedia.org/wiki/Convolutional_neural_network

- There are at least <u>**four** layered **concepts** </u>used to understand **Convolutional Neural Networks**:
  1. **Convolution** layer**,**
  2. **Rectified Linear Unit** (**ReLu**) layer (*activation function* of the convolution layer),
  3. **Pooling** layer, and
  4. A **Fully Connected** Layer.
- A sample structures of CNN are shown in **Figure 7.1(a)** and **(b)** and **(c)**.

Structure of a CNN

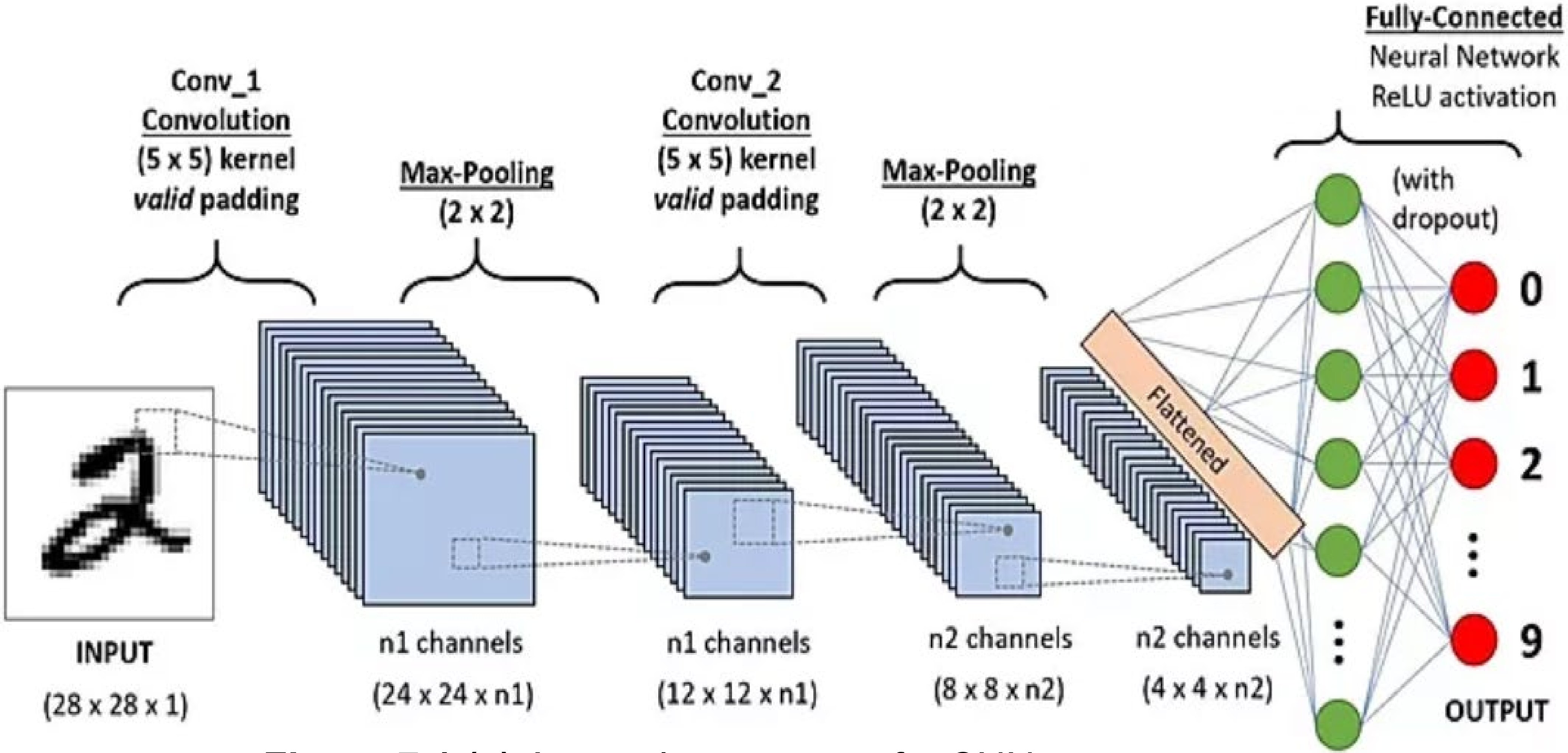


Figure 7.1 (a) A sample structure of a CNN.

Structure of a CNN

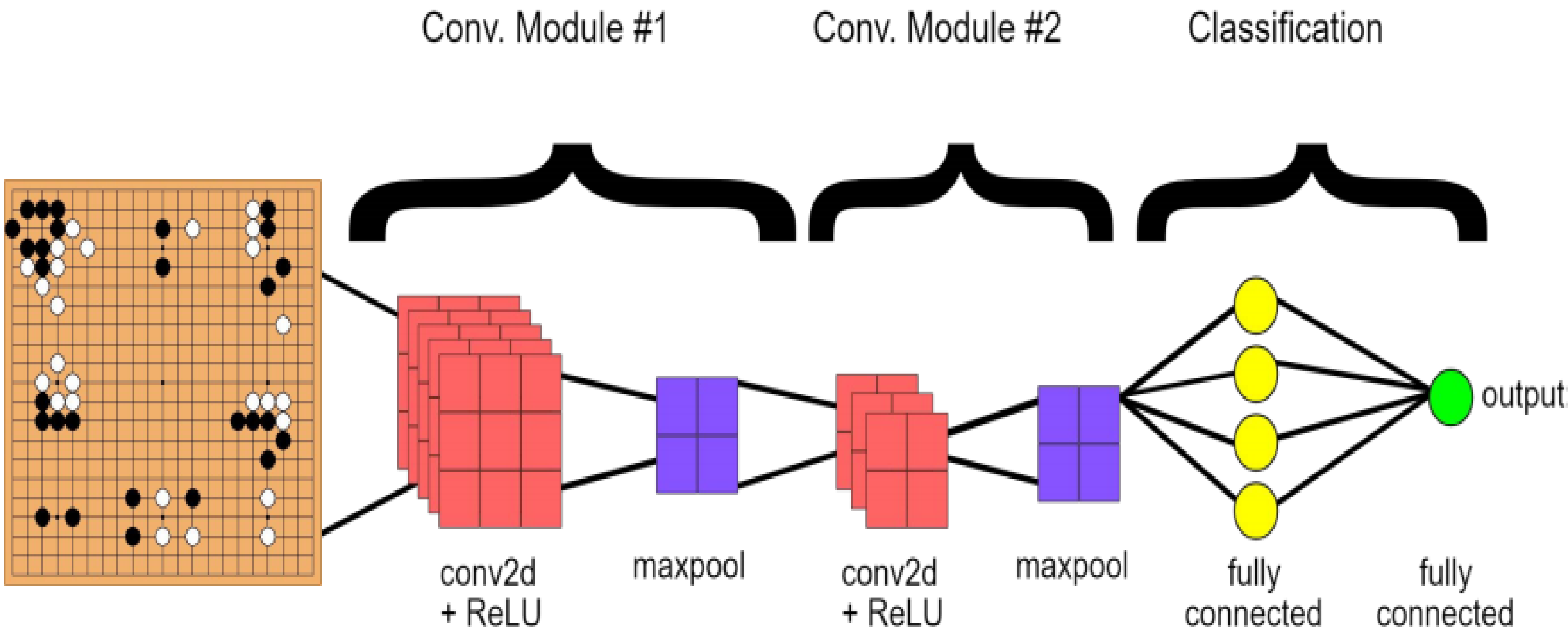


Figure 7.1 (b) A sample structure of a CNN.

Structure of a CNN

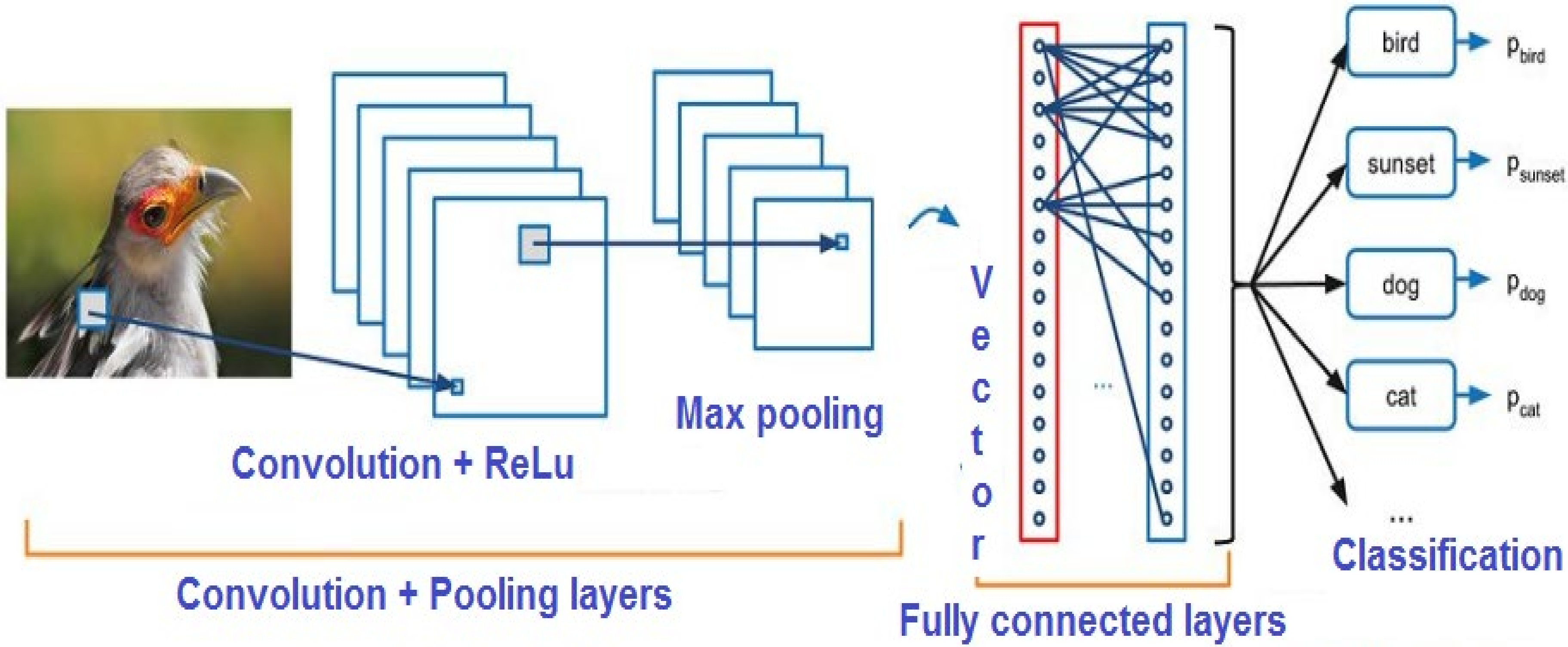


Figure 7.1 (c) A sample structure of a CNN.

Convolutional Neural Networks

*https://en.wikipedia.org/wiki/Convolutional_neural_network

- **CNN**s are neural networks that use **convolution** instead of general **matrix multiplication** in at least one of their layers.
- A **CNN** consists of an **input layer**, an **output layer, and** many **hidden layers**.
- The **hidden layers** of a **CNN** typically consist of a series of **convolutional layers**.

Convolutional Neural Networks

*https://en.wikipedia.org/wiki/Convolutional_neural_network

- The **ReLu** layer is subsequently followed by additional ***convolutions*** such as **pooling layers**, **fully connected layers,** and **normalization layers**, referred to as **hidden layers** because their **inputs** and **outputs** are masked by the **activation function** and **final convolution**
  - Though the layers are colloquially referred to as **convolutions,** this is only by convention.
  - In fact, they all have a different operation paradigm.

Convolutional Neural Networks

*https://en.wikipedia.org/wiki/Convolutional_neural_network

- When programming a **CNN**, the **input** is a **tensor** (a *grid* or *array of image-pixel matrix*) with a **shape:**

  Shape of total Input tensor = ***Number of images*** × ***image_width*** × ***image_height*** × , ***image_depth*** (*color depth indicates no. of bits of a color pixe*l).

- The **convolutional layer** generates an abstraction of the **input image** called a **feature map.**
  – The ***shape*** of a **total feature map** = ***Number of images*** × ***feature map width*** × ***feature map height*** × ***feature map color channels*** (for color images)

Convolutional Neural Networks

*https://en.wikipedia.org/wiki/Convolutional_neural_network

- A **convolutional layer** within the **CNN** should have the following attributes:
  - **Convolutional kernels** (called **hyperparameters**) are defined by a ***width*** and ***height***.
  - <u>The **depth** of the **convolution filter** (or **kernel**) must be equal to the depth of the **feature map.**</u>
    - ***Feature map is the output of the convolutional layer***

Convolutional Neural Networks

*https://en.wikipedia.org/wiki/Convolutional_neural_network

- A **convolutional layer** ***convolves*** the **input image** and passes its result (*feature map*) to the **next layer**
- Each **convolutional layer** processes data using its **receptive field** to generate a ***feature map*** of the input
- The ***features*** of the images are then learned by a **Multilayer Perceptron** (MLP) to ***classify image data,*** called the **fully connected layer** of the CNN
  - Applying **image data** directly to this layer is impractical.
  - Because many neurons would be necessary, even in a shallow architecture, due to the large pixel size associated with images.
-

Convolutional Neural Networks

*https://en.wikipedia.org/wiki/Convolutional_neural_network

- For instance, assume a **non-convolutional neural model** with only **fully connected layers,** with a small gray image of size **100 x 100 x 1, which** needs **10,000** (**100 x 100**) ***weights for each neuron*** in the **second layer. It is impractical!**
- The **convolution operation** solves this huge parameter (*weight*) problem by reducing the number of ***parameters***, allowing the **network to be deeper with fewer parameters**.
  - For instance, tiling regions of size **5 x 5**, each with the same **shared weights**, requires only **25 parameters (weights)**.
- In this way, **CNN** resolves the **vanishing** or **exploding gradients problem** in a traditional **MLP** using the **backpropagation** learning.

Convolutional Neural Networks

- **Vanishing gradients problem**:
  - The problem of ***vanishing gradient*** describes that the ***weights and biases do not update during the training (backpropagation), so the neural network fails to learn the data***, and it leads to slow convergence.
  - *The performance of the neural network will decrease as a result*.
- **Exploding gradients problem**:
  - ***Exploding gradients*** is a problem **where significant error gradients accumulate during training, resulting in anomalous weight/bias updating** (means, *weights/bias become excessively large*, affecting the convergence badly)**.**

Convolutional Neural Networks

- In a **CNN**, finally, after several **convolutional** and **pooling** layers, the final reasoning is done via its **fully connected layer**.
- **Neurons** in a **fully connected layer** connect to all activations in the previous layers.

Convolutional Neural Networks

- The best **CNN architectures** have consistently been composed of various layered building blocks.
- In **CNN,** a **filter** (also called a **kernel**) is a set of **learnable weights** that are learned using the **backpropagation algorithm**
  - A **filter (kernel)** is represented by a **vector of weights** that **convolves** the **input tensor**.

What is “Convolution” in a CNN?

- The name “**convolutional neural network**” indicates that the network employs a mathematical operation called ***convolution***
  - The **convolution** is a specialized kind of ***linear operation***.
- **CNNs** are **neural networks** that use **convolution** instead of general **matrix multiplication (input-weight)** in at least one of their layers.
  - **The convolution operation used in a CNN *does not precisely support the pure mathematical convolution.***

The Convolution Operation

- The **convolution** operation between two values is typically denoted with an asterisk as below:

$$s(t) = (x * w)(t)$$

  - where ***x* is referred to as the input, and *w*** is the **convolution kernel** or **filter** (a 2D array of weights), and ***t*** is the **time factor**.
  - The **output** of the **convolution operation** is called the **feature map**.
  - **Negative values** from a **convolution** operation in a CNN are converted to **0** by the **ReLu** function.

The Convolution Operation

- In **CNN**, the **input *x*** is usually a **multidimensional array of data (pixels)**, and the ***w*** is the *set of weights* (called a **kernel** or a **filter**).
  - Each input and kernel array element must be explicitly stored separately.
- **Figure 6.1** shows an example of a **convolution** operation on a **2-D tensor input** with a **stride value of 1**.

The Convolution Operation

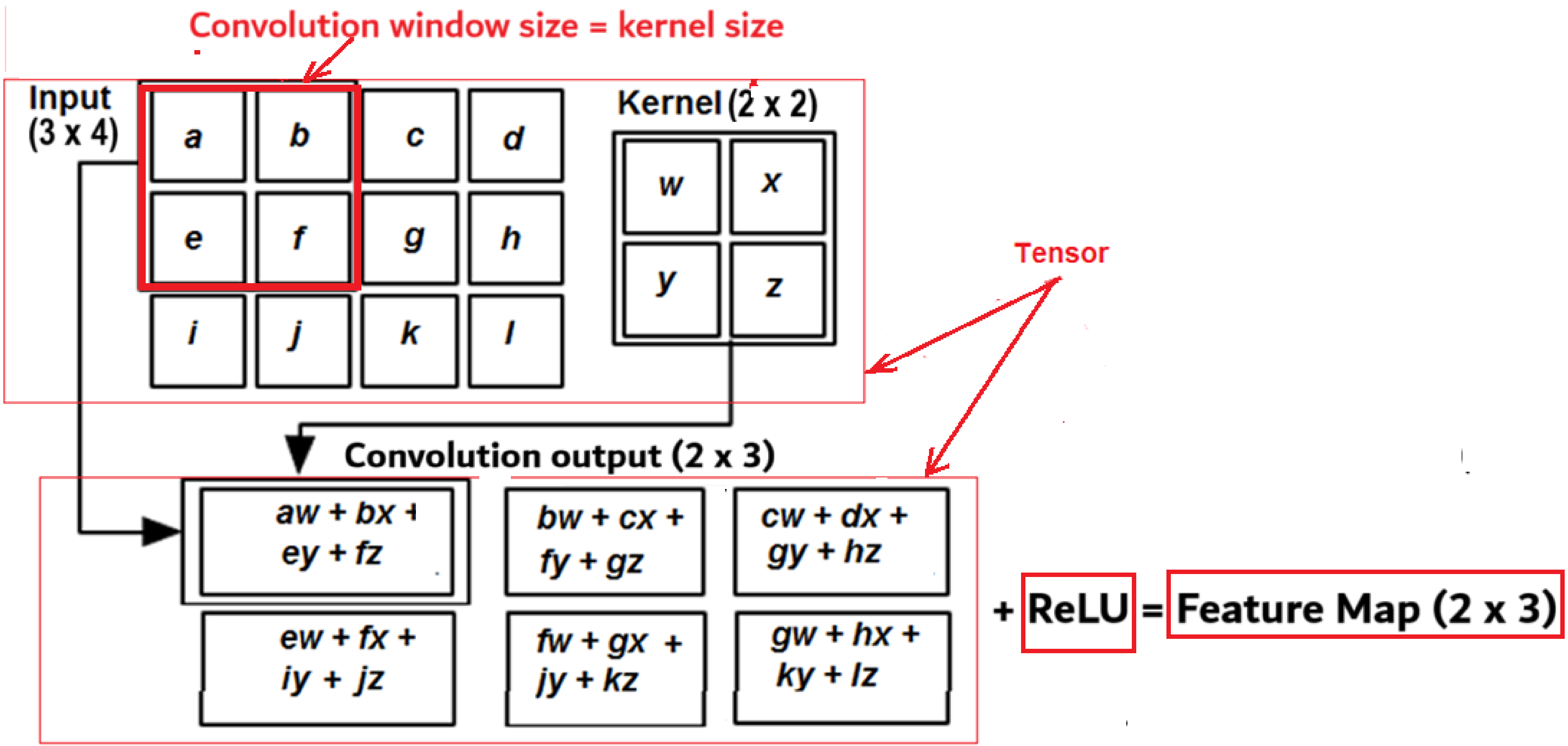
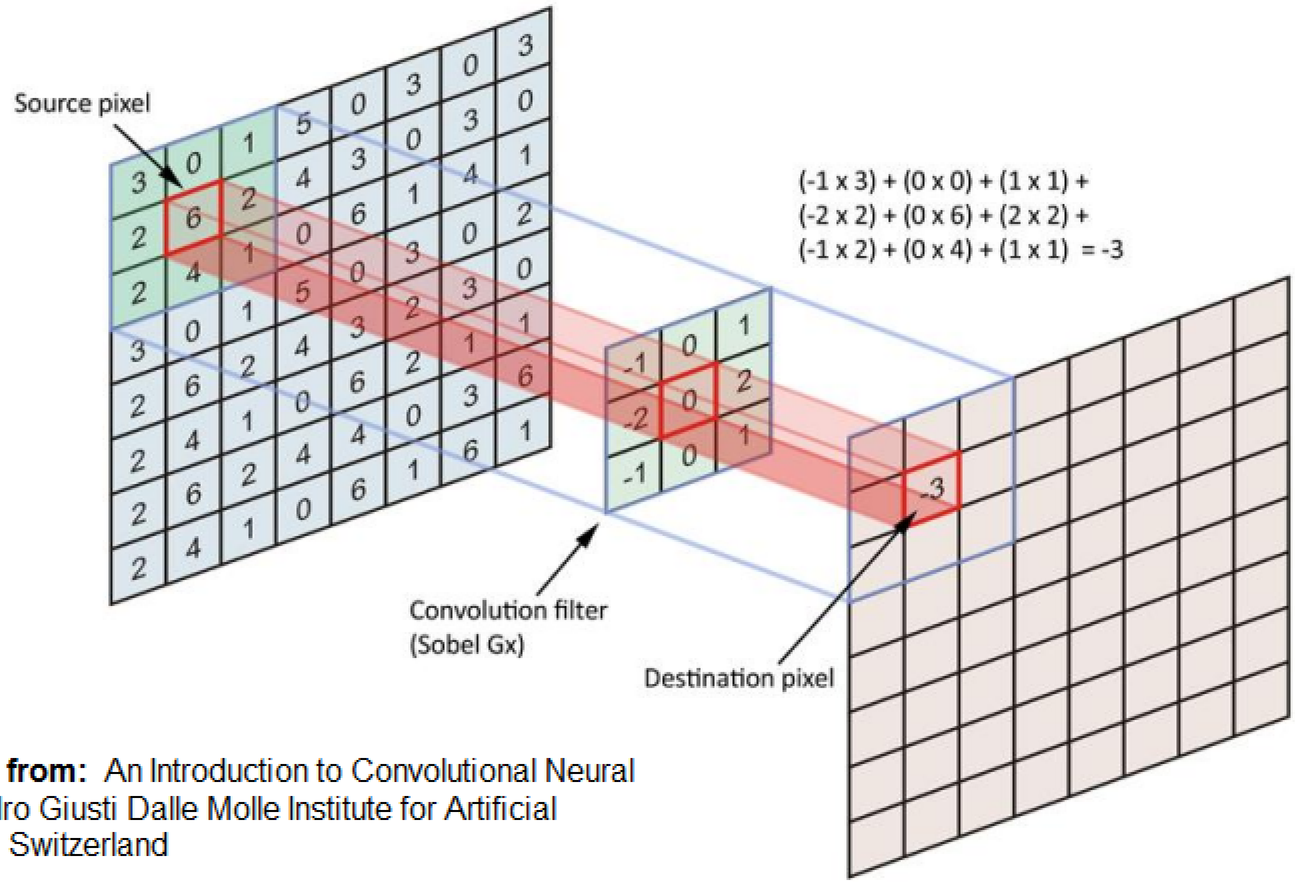


Figure 6.1: A example of 2D convolution without kernel flipping (stride = 1)

The Convolution Operation

- **Figure 6.1** shows an example of **2-D convolution <u>without kernel flipping</u> (**which means that *the same kernel is involved with a stride* = 1).
- In **Figure 6.1**, the **output** is restricted to only positions where the **kernel** lies entirely within the image, called a "<u>**valid**" **convolution**</u>
  - ***Means no extra bits are added to the image during the convolution (the process of adding extra bits to the image is called <u>zero padding</u>)***
  - The boxes with arrows indicate how the **upper-left** element of the **output tensor** is formed by applying the kernel to the corresponding **upper-left** region of the **input tensor**.

The Convolution Operation



This slide is taken from: An Introduction to Convolutional Neural Networks, Alessandro Giusti Dalle Molle Institute for Artificial Intelligence Lugano, Switzerland

The Convolution Operation

0	0	0	0	0	0
0	105	102	100	97	96
0	103	99	103	101	102
0	101	98	104	102	100

Image Matrix

Kernel Matrix

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

$$0*0+0*-1+0*0$$
$$+0*-1+105*5+102*-1$$
$$+0*0+103*-1+99*0 \quad = 320$$

320					

Output Matrix

Here, a **3x3 filter (kernel)** performs a **convolution (stride = 1)** over a **zero-padded input image** to produce a **feature map (**the application of **ReLu** is implicit). This convolution result is **not a valid one** because of the **zero-padding**.

Convolution: Motivation

- **Convolution** leverages **three** critical ideas that can help improve a **machine-learning** system:
  1. **Sparse interactions**
  2. **Parameter sharing** and
  3. **Equivariant representations**
- Moreover, **convolution** provides a means for working with **variable-sized inputs**.

Convolution: Motivation

- Traditional neural network layers use **matrix multiplication** by a matrix of parameters with a separate parameter describing the interaction between each **input unit** and each **output unit**
  - **This means that every output unit interacts with every input unit.**
- But **convolutional networks** have **sparse interactions** (also referred to as **sparse connectivity** or **sparse weights**)
  - This is accomplished by making the **kernel smaller** than the **input image** (see **Figure 6.1**).

Parameter sharing

- There are **two kernel parameters**: **weights** and **biases**.
- The **total number of parameters** is the **sum of all weights** and **biases.**
- **Parameter sharing** refers *to using the same parameter for more than one function in a network model*.
- In a traditional neural net, each element of the **weight matrix** is used exactly once when computing the **output** of a layer.
  - ***It is multiplied by one input element and then never revisited***.

Stages of a CNN Layer

- A typical layer of a **convolutional neural network** consists of **three stages** (see **Figure 6.7**).
  - In the **first stage**, the layer performs several **convolutions** in parallel to produce a set of **linear activations**.
  - In the **second stage**, each **linear activation** (convolution result) is run through a ***nonlinear function***, such as the **Rectified Linear Unit (ReLu)** activation function. This stage is sometimes called the **detector stage**
  - In the **third stage**, we use a **pooling function** to further modify the layer's output.
  - https://en.wikipedia.org/wiki/Affine_transformation

Typical CNN Layers

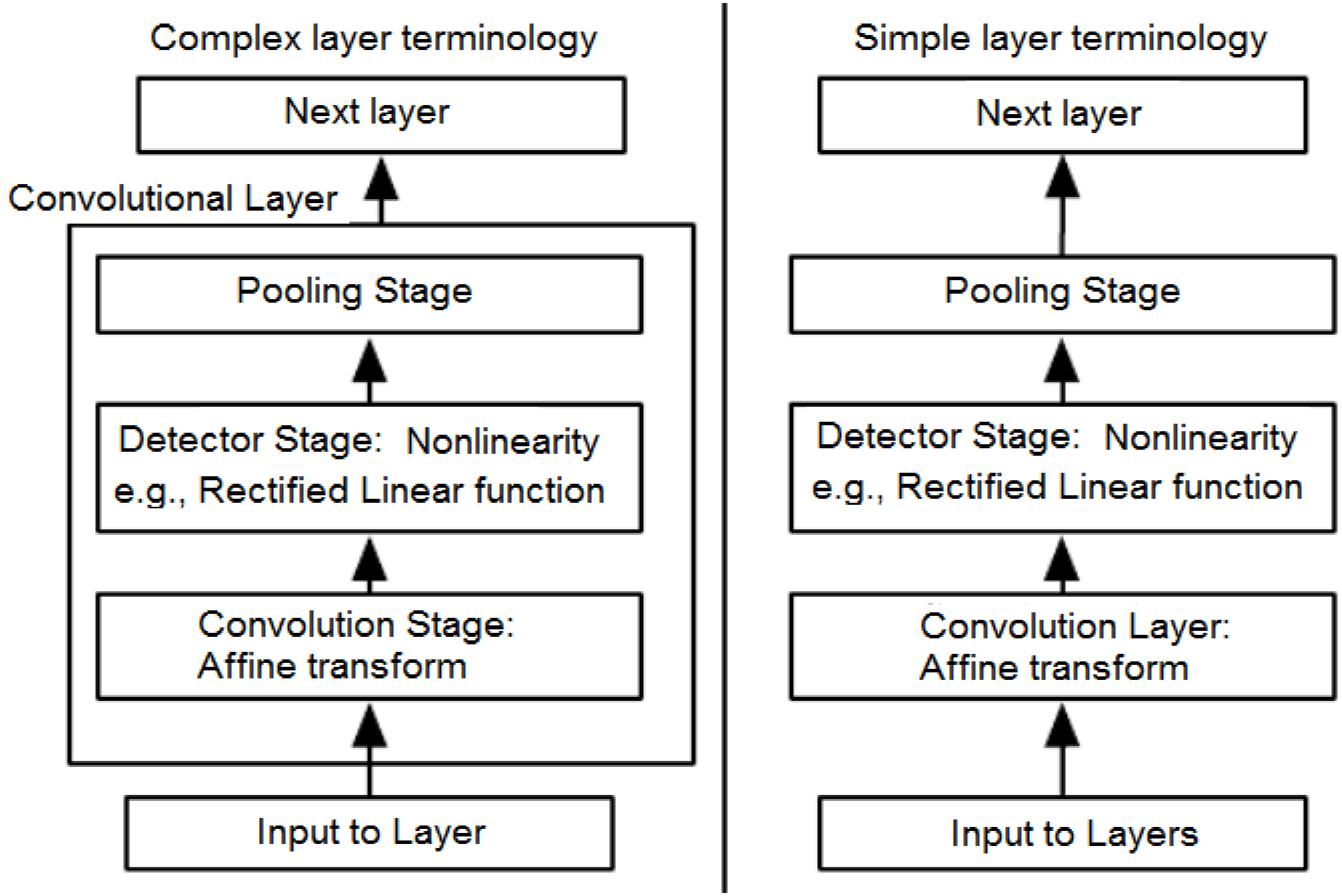
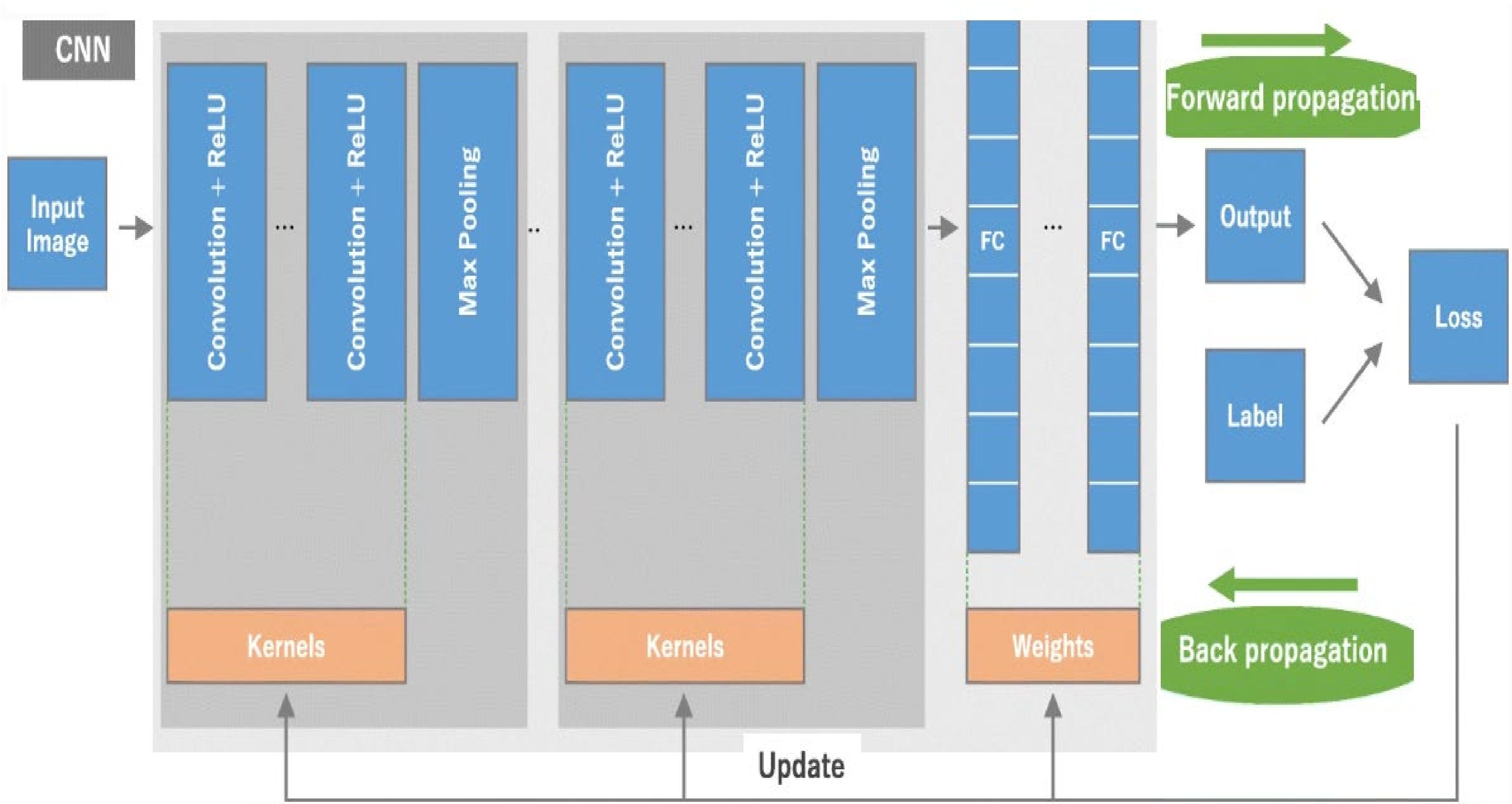


Figure 6.7: The components of a typical CNN layer.

An **affine transformation** is a linear mapping method that preserves points, straight lines, and planes of the data structure.

Typical CNN Layers



Typical CNN Layers

- In **Figure 6.7** shows two commonly used sets of terminology for describing **CNN layers**.
- The CNN is viewed as a small number of relatively **complex layers**, each having many "***stages***."
- In the **simple layer terminology**, a CNN is viewed as **a larger number of simple layers**; *not every "**layer**" has parameters.*
- A **CNN** can ***successfully capture the Spatial and Temporal dependencies*** in an image by applying relevant **filters (kernels)**.
- The **CNN architecture** performs a better fit to the **image dataset** due to the **reduction in the number of parameters** involved and the <u>**reusability of weights**</u>.
  - In other words, ***the network can be trained to understand the image's features with minimal weights*.**

-

Input Image

- **Figure 7.2** shows a **4 x 4 x 3 input image (color)** separated by its **color** planes—**Red**, **Green**, and **Blue (3)**.
  - There are several such color spaces in which images exist—***Grayscale***, ***RGB***, ***HSV***, ***CMYK***, etc.
- From the **RGB** image (**Figure 7.2),** you can imagine how **computationally intensive it is** to reach image dimensions.
  - ***The role of a CNN is to reduce the complexities of images into a form that is easier to process without losing features critical for a good prediction.***
  - This is important when designing a **CNN architecture** that is good at learning features and **scalable to massive datasets**.

Input Image

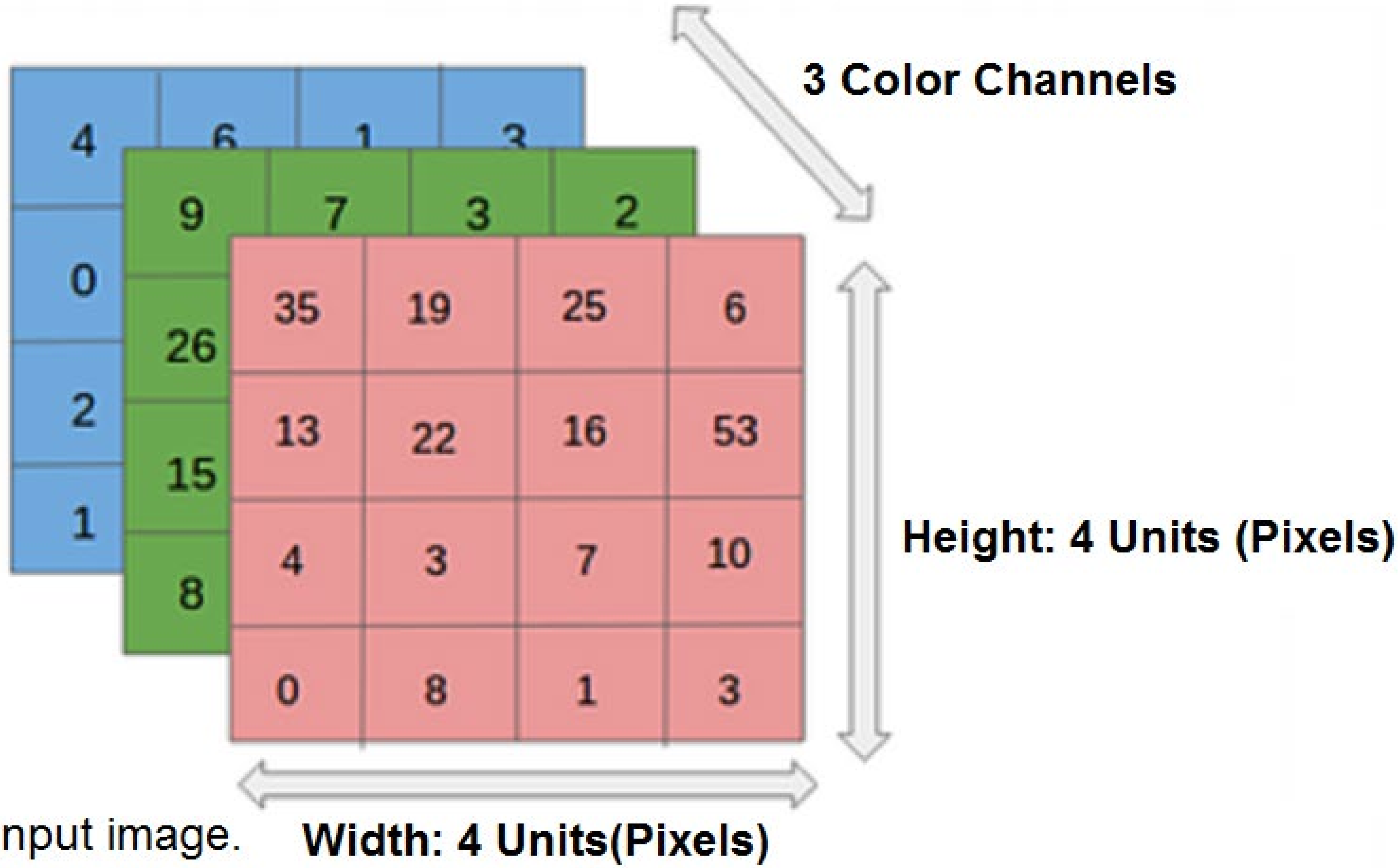


Figure 7.2: A **4x4x3 RGB** input image.

Convolution on Images

- **Figure 7.3** shows the extraction of a **3x3x1 feature map** (**convolved feature** without **zero padding**) result from a **gray image** with size **5 x 5 x 1** (height = **5**, width = **5**, and color = **1**).
- The **kernel/Filter, *K*,** with size **3x3x1** (where **1** indicates that the image is a *gray image*, and only ***one kernel is involved***). The convolution **flip stride** is **1**.
- **Figure 7.4** shows convolution on a **4x4x1 image** with zero-padding and a **3x3x1 kernel**.
- **Figure 7.6** shows a convolution operation with **a stride of 2** (with a **valid Zero Padding**).

Convolution on a Gray Image

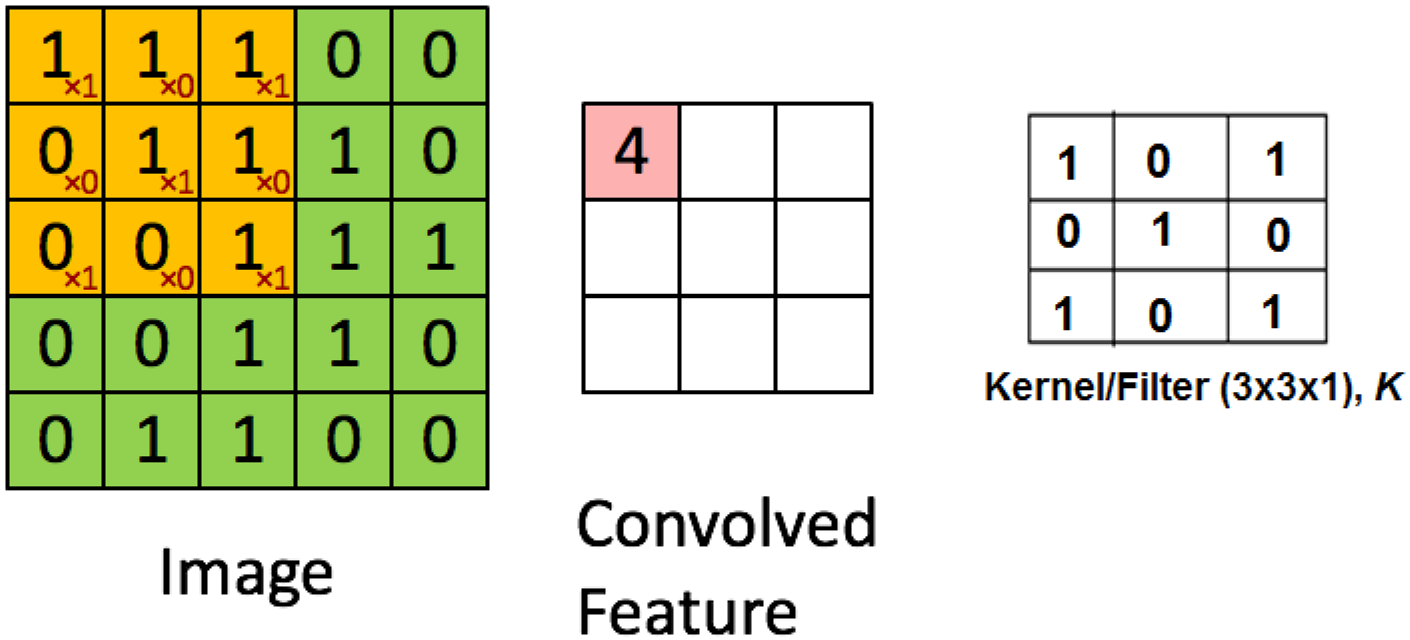


Figure 7.3: Convoluting a **5x5x1 image** with a **3x3x1 kernel** to get a **3x3x1 convolved feature** (feature map).

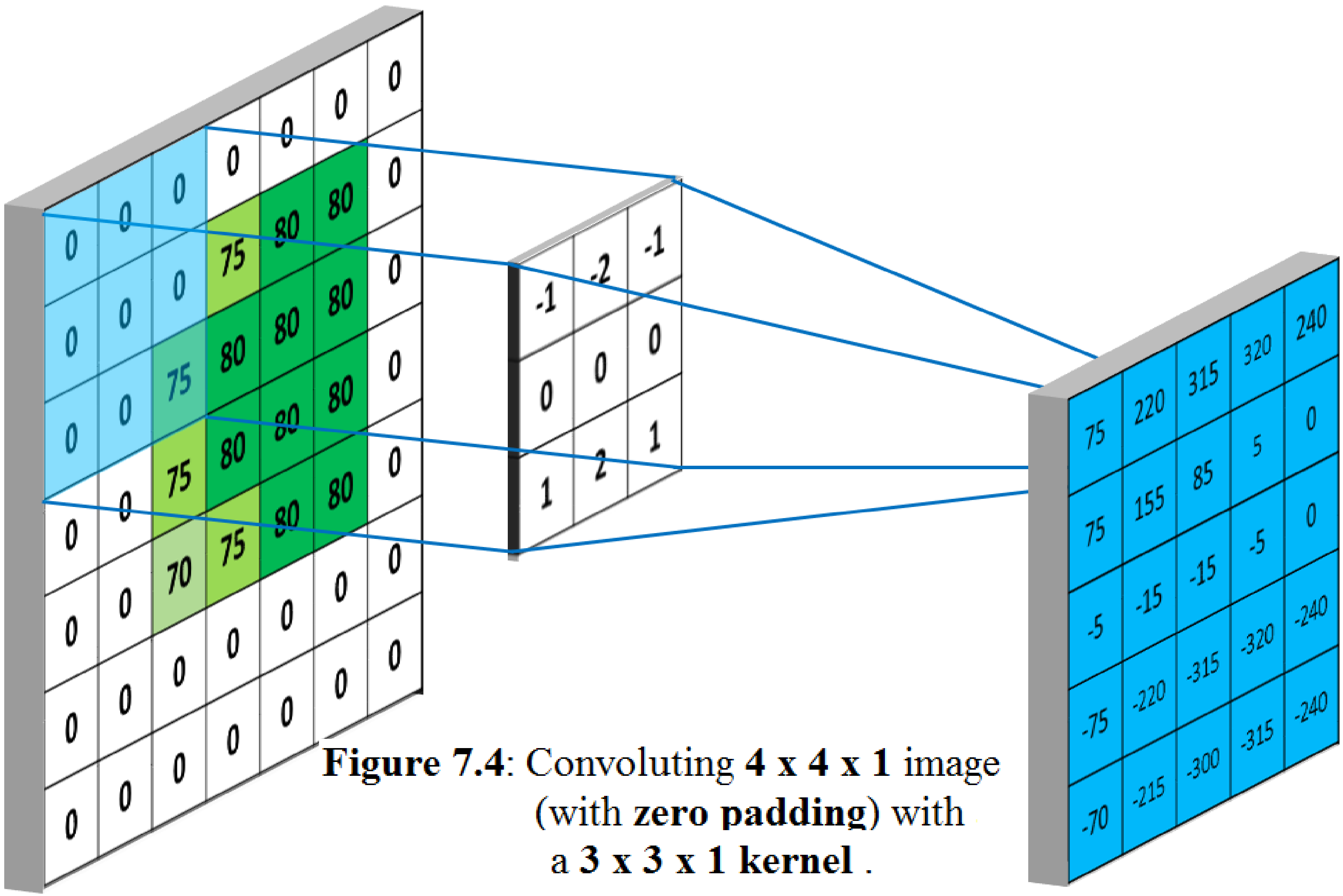


Figure 7.4: Convoluting **4 x 4 x 1** image (with **zero padding**) with a **3 x 3 x 1 kernel** .

https://mlnotebook.github.io/post/CNN1/

Convolution with Stride Length = 2

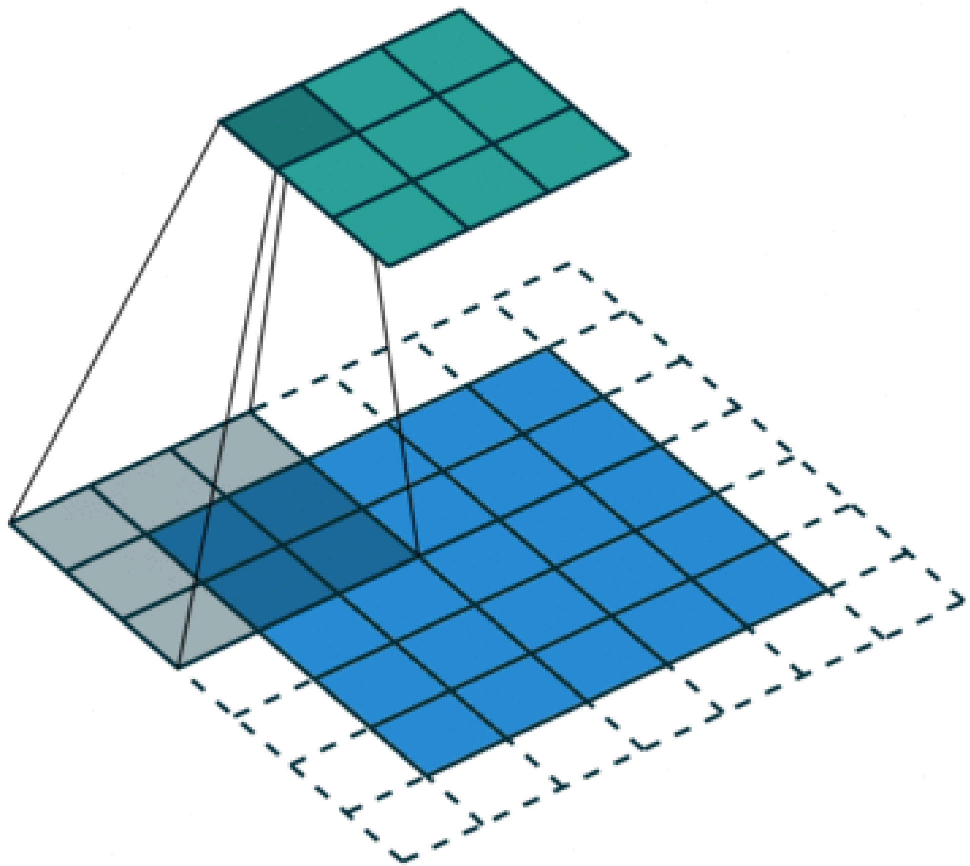


Figure 7.6: A **Convolution Operation** with **Stride Length = 2**

Convolution on an RGB image

- The complete convolutions on the **input image** of size **5x5x3, with a kernel size 3x3x3 (stride value = 1)** shown in **Figure 7.5**.
  - **Figure 7.5** shows how a complete convolution happens on an input image with **three input channels** (e.g., **RGB**).
  - The **kernel** has the same depth (**depth = 3**) as the **input image channels**.
  - Here, the **convolution** is performed between **image channels** (**I** =**1**, **2**, **3**) and the **kernel channels (k = 1, 2, 3),** with the **bias** value of **1** to give us a squashed one-depth channel of **Convoluted output Feature**.
-

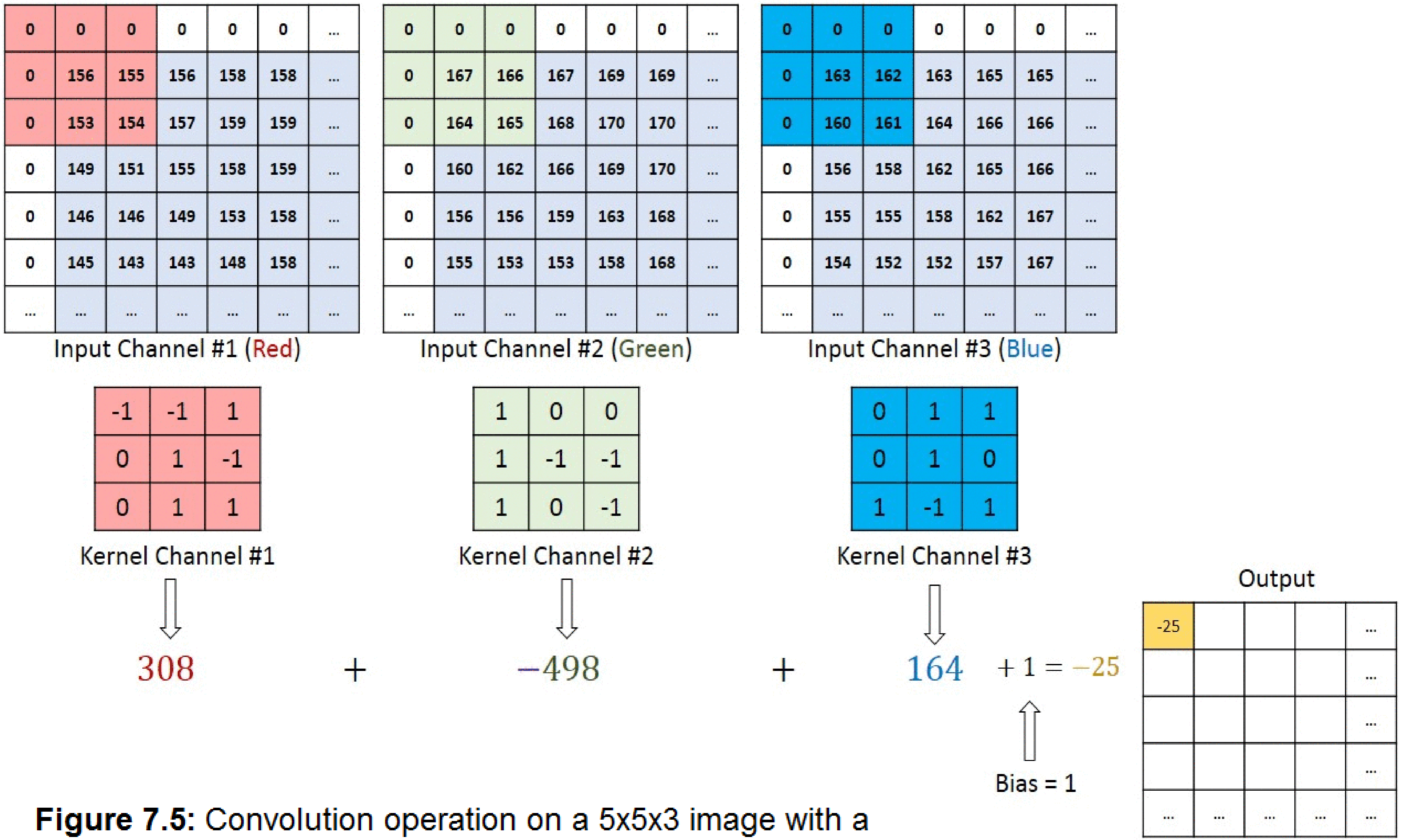


Figure 7.5: Convolution operation on a 5x5x3 image with a 3x3x3 Kernel.

Objective of Convolution Operation

- The main objective of the **convolution** operation is to **extract the high-level features**, such as **edges, color**, etc., from the **input image**.
- A **CNN** need not be limited to only one **convolutional Layer** (**ConvLayer**):
  - The **first ConvLayer** captures the **Low-Level features** such as ***edges***, ***color***, ***gradient orientation***, etc.
  - With the application of further **ConLayers,** capture **High-Level features,** which provide a wholesome understanding of images in the dataset.

Objective of Convolution Operation

- There are **two types of results from a convolution** operation:
  - The **first one** in which the **convolved feature** is reduced in dimensionality compared to the **input**,
  - and the **second one** is in which the **dimensionality** is either **increased** or **remains** the same.
- This is done by applying **Valid Zero Padding** in the case of the former, or the **Same Padding** in the latter case.
  - In **valid zero padding**, the size of the convoluted feature is the same as the original image

Convolution with Valid Zero-padding

- **Valid Zero Padding:** Converting the **5x5x1 image** into a **7x7x1 image** by applying **zero padding** and then performing the convolution operation with the **kernel size 3x3x1** over it, the size of the resulting **convolved matrix** turns out to be **5x5x1** with a **stride of 1** (see **Figure 7.7**).
  - Hence, it is also called the **same Padding** (because the **5x5x1 feature map is** generated from the **zero-padded** image of size **5x5x1** (after the zero padding, the image size is **7x7x1**))
  - Thus, **"same"** results in padding the **input** such that the **output** has the **same length as the original input**.

The size of the feature map is 5x5x1 (same as the original image 5x5x1)

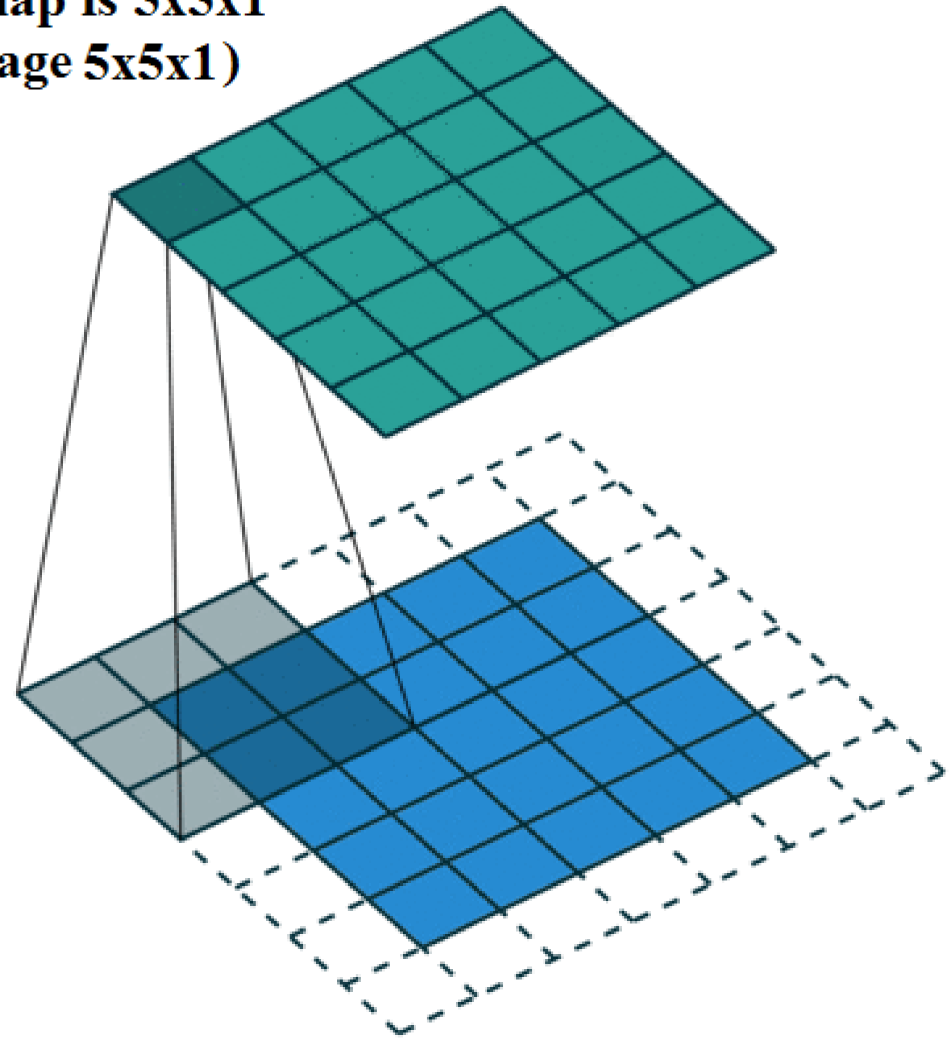


Figure 7.7: SAME padding: 5x5x1 image is padded with 0s to create a 7x7x1 image.

Summary of Convolution

2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3^{3}	4^{4}	6^{4}
3	2	4	1	9^{1}	8^{0}	3^{2}
0	1	3	9	2^{-1}	1^{0}	4^{3}

Input size = 7x7

*

3	4	4
1	0	2
-1	0	3

Kernel (filter) = 3x3
Stride = 2

=

91	100	83
69	91	127
44	72	74

Feature map = 3x3

If ***n* x *n*** image, ***f* x *f*** filter, padding ***p***, and stride ***s***, then size of feature map is

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

Pooling Layer

- Similar to the **Convolutional Layer**, the **Pooling Layer** is responsible for **reducing the spatial size of the Convolved Feature of an input image.**
  - A **3x3 pooling** over a **5x5 convolved feature** is shown in **Figure 7.8** with a **stride of 1**.
- The **pooling operation decreases the computational power required to process the data through the matrix dimensionality reduction**.
- Furthermore, **pooling helps extract dominant features** that are rotational and positional invariant, thus enhancing the model's training process.

Pooling Layer

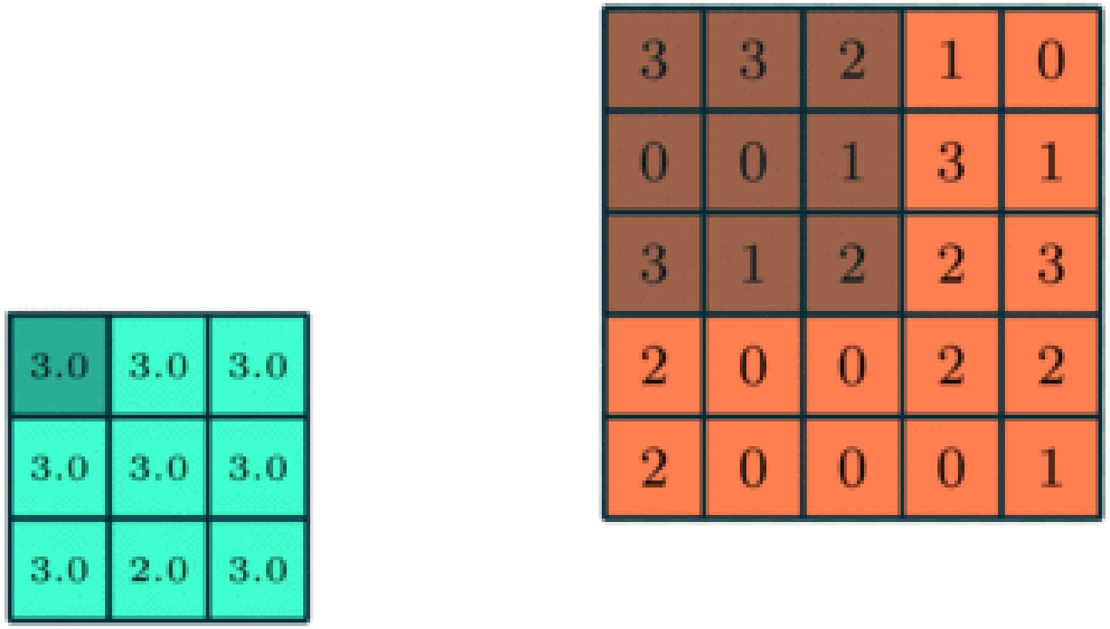


Figure 7.8: A 3x3 pooling over 5x5 convolved feature.

Pooling Layer

- There are **two types of Pooling** with a **stride of 2** (see **Figure 7.9**):
  - **Max Pooling** returns the maximum value from the portion of the image covered by the kernel.
  - **Average Pooling** returns the average of all the values from the portion of the image covered by the kernel.
- **Max Pooling** also performs as a **Noise Suppressant**.
  - It discards the noisy activations altogether and performs de-noising and dimensionality reduction.
- **Average Pooling** performs **dimensionality reduction** as a noise-suppressing mechanism.
  - Hence, we can say that **Max Pooling performs much better than Average Pooling**.

Pooling Layer

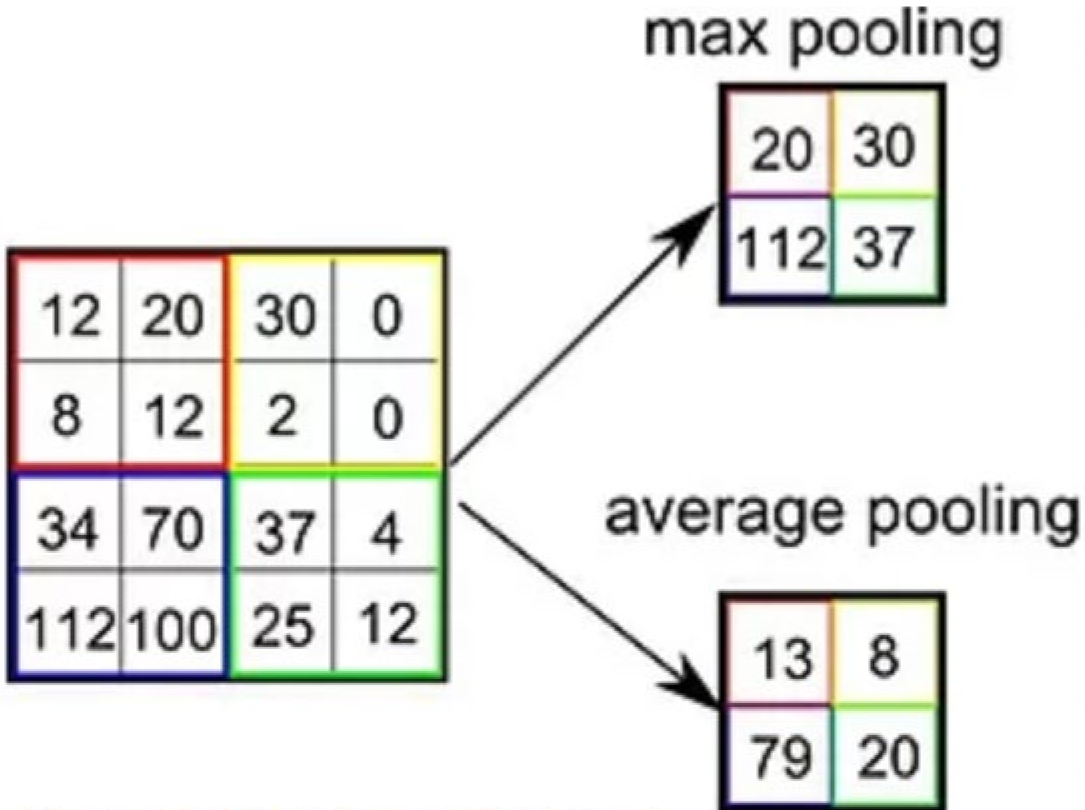


Figure 7.9: Types of **Pooling**.

Pooling Layer

- The **Convolutional Layer** and the **Pooling Layer**, together, *depending on the complexities in the images*, *the number of such layers may be increased for capturing from* ***low-level*** *details to higher or even further* at the cost of **more computational power**.
  - After going through the above process, we have successfully enabled the model to understand the **features**.
- The **output** of the **Pooling Layer** is **flattened (vectorized)** for the **fully connected layer** for generating the **final output** of the CNN.

Fully Connected Layer

- Adding a **Fully-Connected layer (FC layer)** is a cheap way of learning **non-linear combinations** of the **high-level features** as represented by the **output** of the **ConvLayer (convolution + ReLU + pooling)** .
- Now that we have converted our **input image** into a suitable form for the **MLP** (fully connected layer) of the CNN.
- **Flatten** the output of the **final ConvLayer** of the CNN into a **column vector** form (see **Figure 7.10**).
- The **flattened output** is fed to an **MLP,** and the **backpropagation algorithm** is applied to every iteration of its training.
  - Over a series of epochs, the model can distinguish between dominating and certain **low-level features** of the image dataset and classify them using the **Softmax Classification** technique (see **Figure 7.11**).

Vectorization (flatten) for the Fully Connected Layer

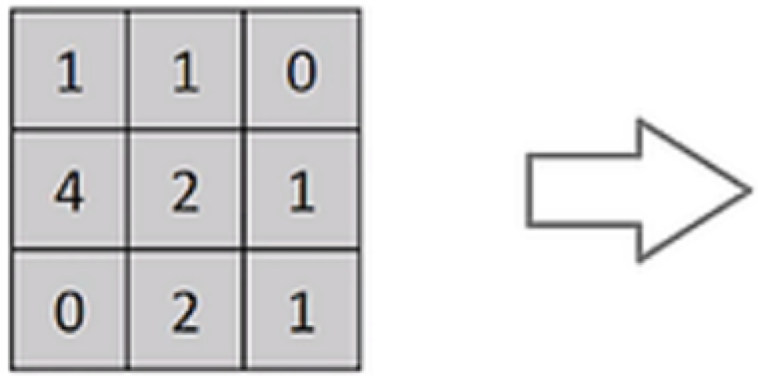


Figure 7.10: Flattening of a **3x3 image feature** into a **9x1 vector.**

Classification by Fully Connected Layer

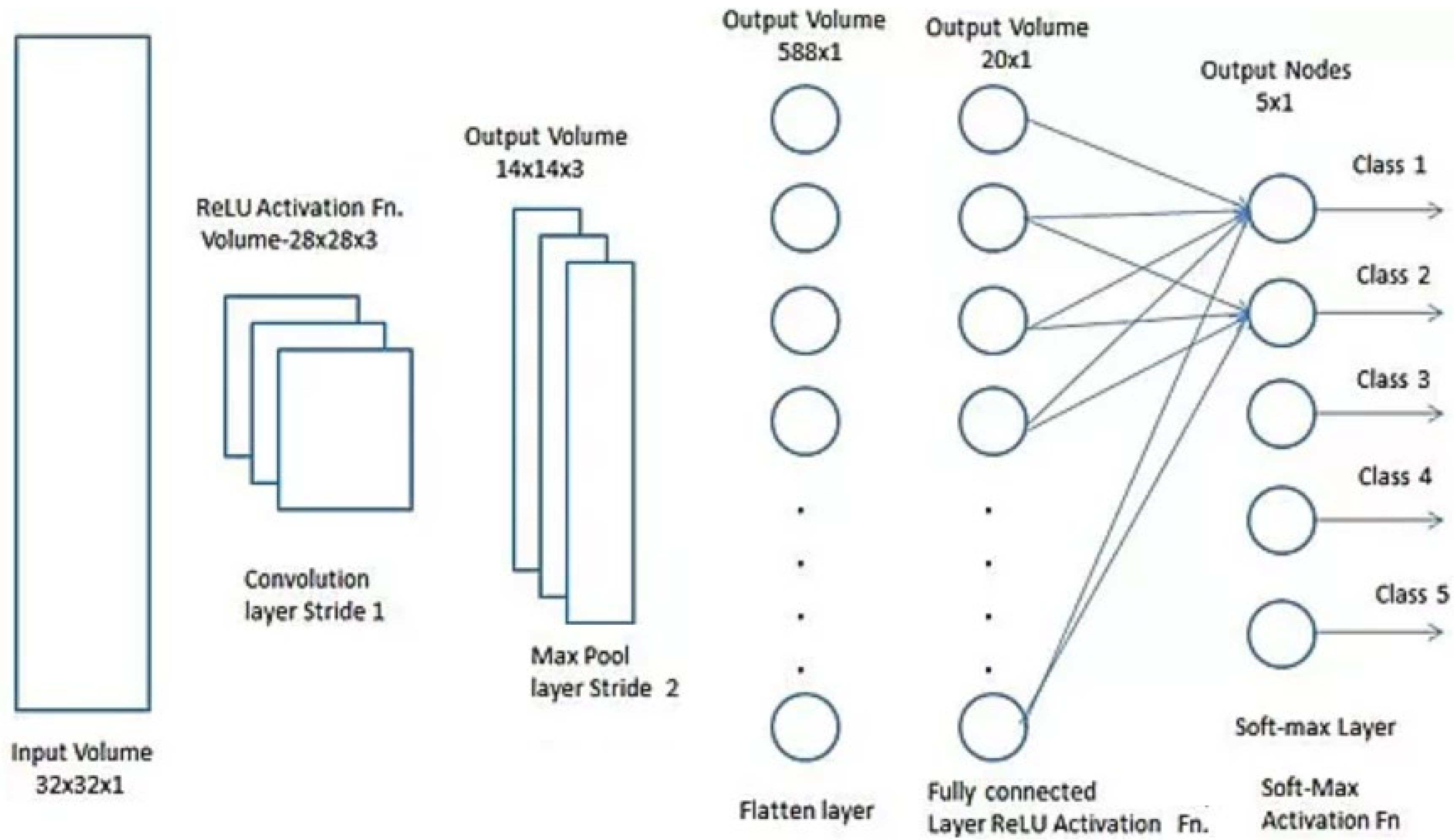


Figure 7.11: The complete operation structutre of a CNN.

The Various CNN Architectures

- There are various architectures of **CNNs** available for various applications listed below:

  LeNet

  AlexNet

  VGGNet

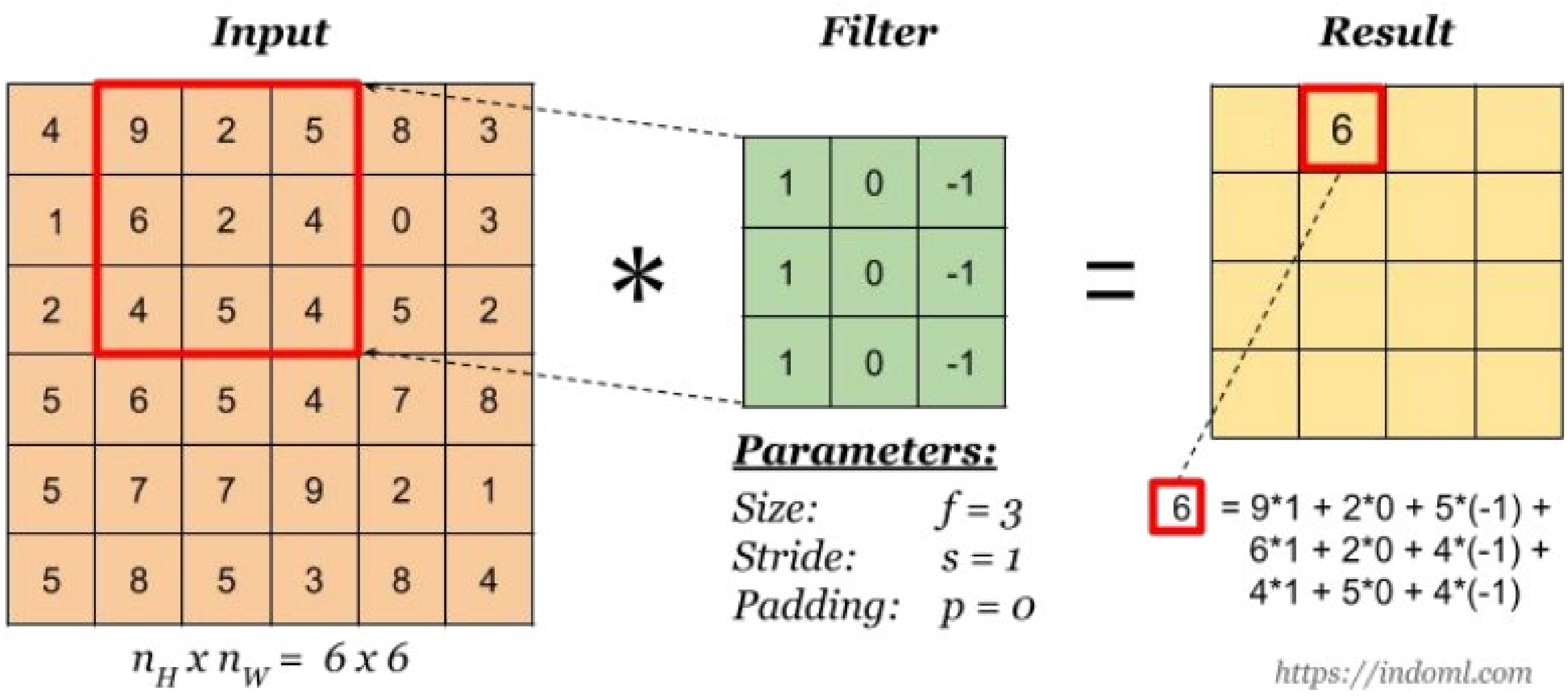
  GoogLeNet

  ResNet

  ZFNet

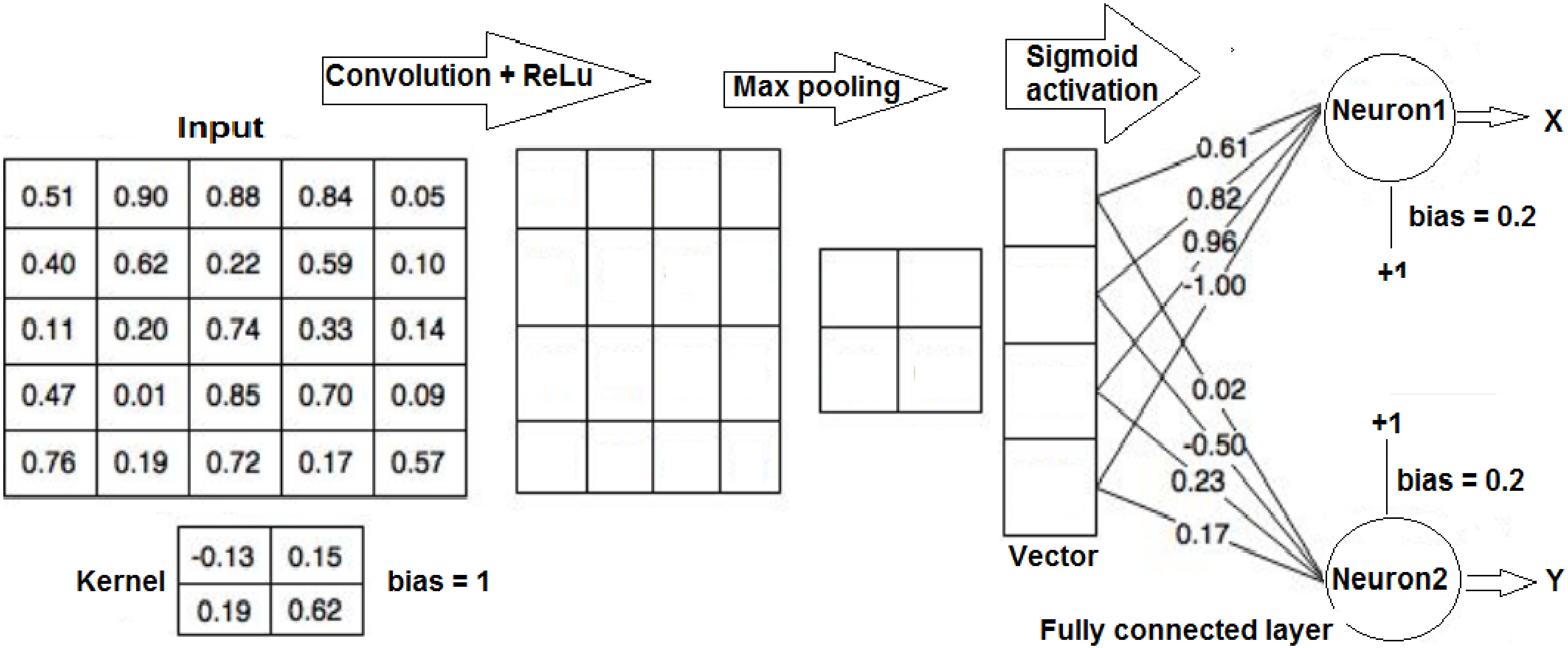
Exercises

1. Complete the convolution operation



2. Repeat the above exercise with **stride = 1** and **zero padding.**

Exercise: Find X and Y



Convolution Stride = 1

How Does Training Happen in a CNN?

- https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add
- We know that **kernels/filters,** also known as **feature identifiers**, are used to identify specific features.
- But how are the kernels initialized with the **specific weights,** or **do the filters know what values to have?**
- The **CNN training process** is known as **backpropagation**, which is further separated into **four distinct sections** or **processes**:
  - **Forward Pass**
  - **Loss Function**
  - **Backward Pass**
  - **Weight Update**

How Does Training Happen in a CNN?

- **The Forward Pass:**
  - In the first epoch or iteration of the training, the **initial kernels** of the first **convLayer** are initialized with **random values**.
- **The Loss Function:**
  - Calculate the loss function, the **Mean Squared Error** (**MSE**)**.** The objective is to minimize the loss, an optimization problem in calculus. It involves trying to adjust the weights to reduce the loss.

    Error = (target output – calculated output)

How Does Training Happen in a CNN?

- **The Backward Pass:**
  - It involves determining which weights contributed most to the loss and finding ways to adjust them, so the loss decreases. It is computed using **dL/dW,** where **L** is the loss, and **W** is the weight of the corresponding kernel.
- **<u>The weight update:</u>**

New_weight = old_weight + learningrate * Gradient of Weight

CNN Training

- The CNN is a **supervised learning-based** algorithm, and its training process includes two stages of propagation;
  - **forward propagation** or **forward pass** and
  - **back propagation** or **backward pass**

Calculating Loss function in CNN

https://stats.stackexchange.com/questions/432896/what-is-the-loss-function-used-for-cnn

- In most cases, CNNs use a **cross-entropy** or **cross-entropy loss** on the **one-hot encoded** data label to calculate the **loss value** of its input during training. For example, for a **single image,** the **cross-entropy loss** looks like this:

$$-\sum_{c=1}^{M}(y_c \cdot \log \hat{y}_c)$$

- **One-hot encoding is a representation of categorical variables as binary vectors of the input class**.
- Where ***M*** is the number of **image class *c***, and $\hat{Y}_c$ is the **model's prediction** for that class. It is the ***softmax*** output for the images by the CNN. The **image labels** are **one-hot encoded, and y is a vector of ones** and **zeroes.** Only **one** will be added for the sum, hence $y_c = 1$.

One-hot Encoding

https://deepai.org/machine-learning-glossary-and-terms/one-hot-encoding

- **What is One-Hot Encoding?**
  - **One-hot encoding** is used in machine learning to quantify **categorical data**. ***One-hot encoding produces a vector with a length equal to the number of categories in the data set***. If **1000 images** are in the dataset, then the **one-hot encoded** label of each image will be a vector of size **1000 x 1**
    - For example, [**0,0,0,1,0**] would be a valid **one-hot encoded** form of an image dataset with **five classes** of images.
    - It indicates that the classification object is located at position **4** (or **3** in array indexing) of the **encoded vector**.
      - Contrastingly, [~~**0,1,0,1,0**],and [**1,1,1,1,1**]~~ are **invalid one-hot encoding forms**.

One-hot Encoding

https://deepai.org/machine-learning-glossary-and-terms/one-hot-encoding

- Consider the problem of classifying a person into one of **four classes**: [***male***, ***female***, ***gender-neutral***, ***other***].
  - For every person we encounter, we want to be able to represent them as a **one-hot encoding** in relation to **four categories**.
  - Let's say while walking down the street, we encounter:
    - **4** people who identify as **female**,
    - **3** people who identify **as male**,
    - **1** person who identifies as **gender-neutral**, and
    - **2** people who identify as **something other than the other three categories**.

One-hot-encoded array of
[***male, female, gender-neutral, other***]:

```
[0,1,0,0] // female
[0,1,0,0]
[0,1,0,0]
[0,1,0,0]
[1,0,0,0] // male
[1,0,0,0]
[1,0,0,0]
[0,0,1,0] // gender-neutral
[0,0,0,1] // other
[0,0,0,1]
```

One-hot Encoding

https://en.wikipedia.org/wiki/One-hot

- Example of **one-hot encoded** results of the input dataset with **three colour classes**:

Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

Softmax Function

https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

- When working on a Machine Learning or a Deep Learning Problem, ***loss/cost functions*** are used to optimize the model during training.
- ***The objective is to minimize the loss function; the lower the loss, the better the model.***
- **Cross-entropy loss** is the most important **cost function**.
- The **cross-entropy** is based on the ***Softmax function*** (the *softmax function is not an activation function*).
- The ***softmax function*** is always applied to the last layer (output layer) of the MLP layer (fully connected layer) of the CNN.
- To understand the ***softmax function***, consider a classification task based on the **4 class image data** below (where an image is classified as a ***dog***, ***cat***, ***horse***, or ***cheetah***).

Softmax Function

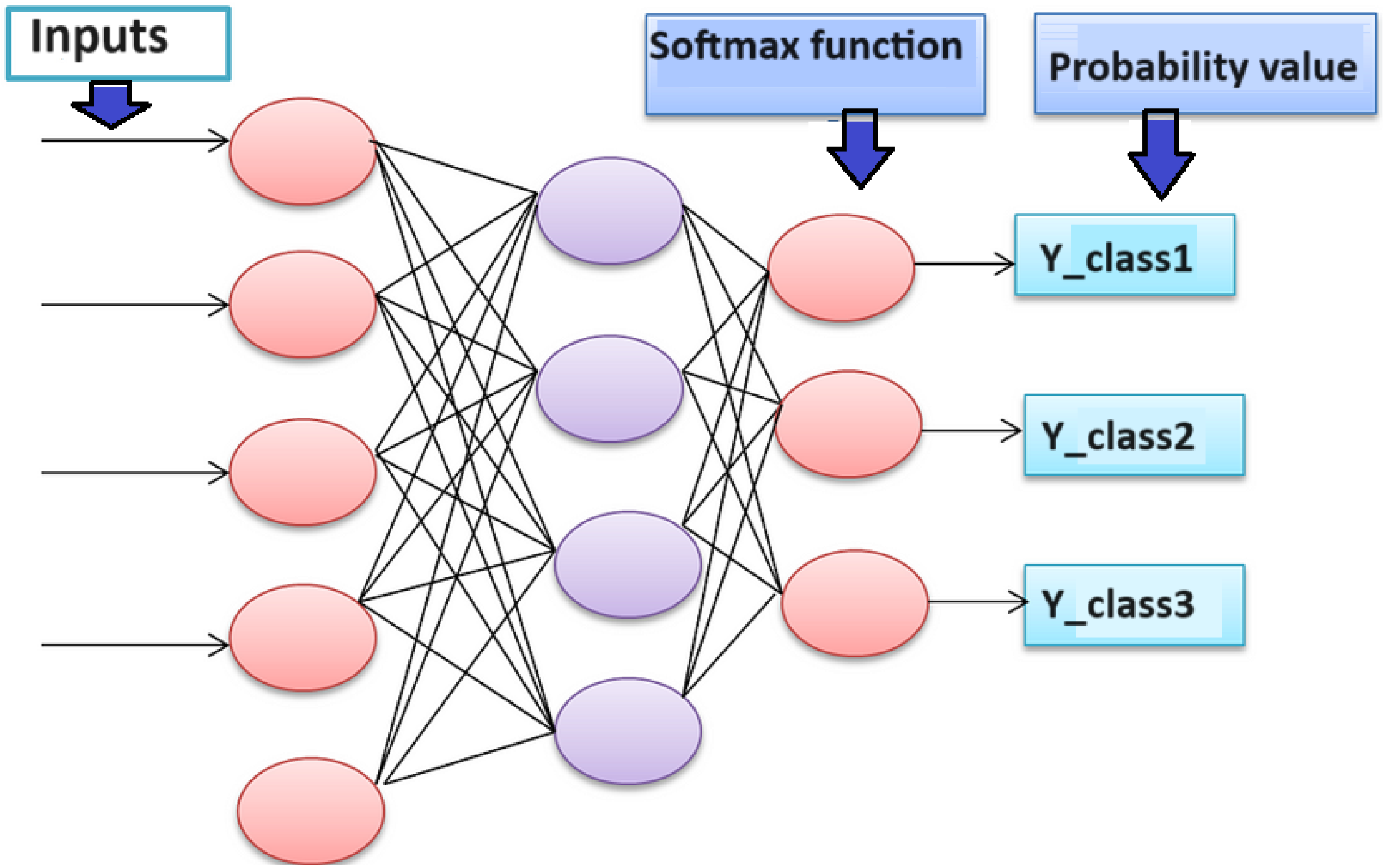


Diagram source: https://medium.com/@ibtedaazeem/loss-functions-in-deep-learning-e4bd353ea08a

Softmax Function

https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

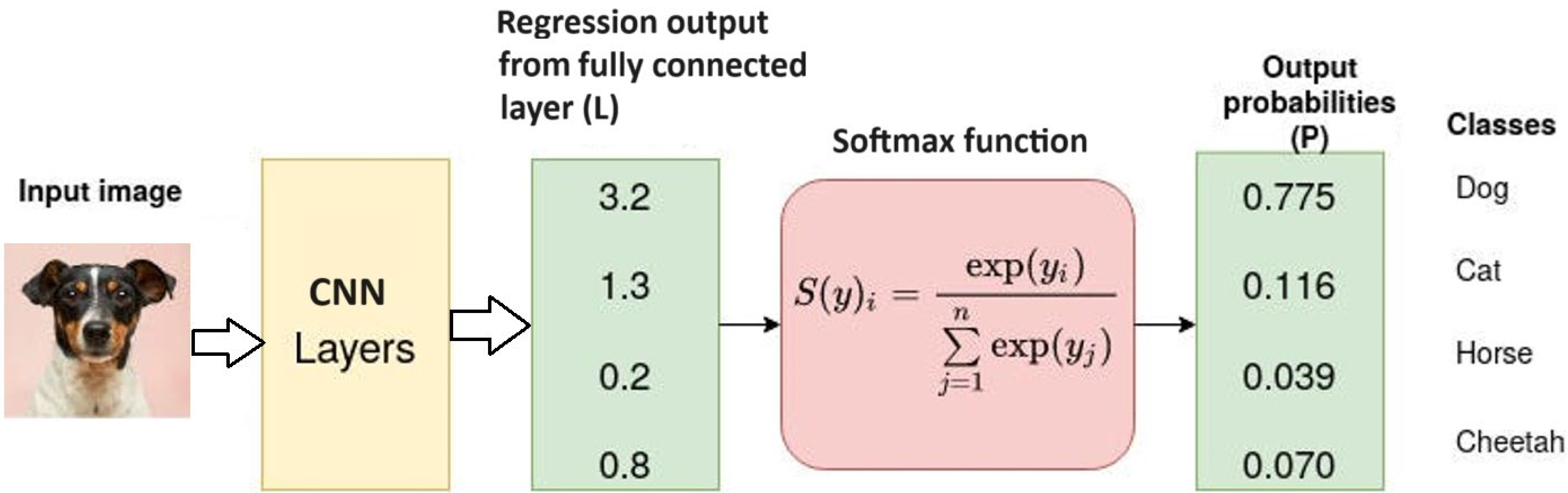


Diagram source: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

$e^{3.2}/(e^{3.2}+e^{1.3}+e^{0.2}+e^{0.8}) = 2.7183^{3.2}/(2.7183^{3.2}+2.7183^{1.3}+2.7183^{0.2}+2.7183^{0.8}) = \mathbf{0.775}$

$e^{1.3}/(e^{3.2}+e^{1.3}+e^{0.2}+e^{0.8}) = 2.7183^{1.3}/(2.7183^{3.2}+2.7183^{1.3}+2.7183^{0.2}+2.7183^{0.8}) = \mathbf{0.116}$

$e^{0.2}/(e^{3.2}+e^{1.3}+e^{0.2}+e^{0.8}) = 2.7183^{0.2}/(2.7183^{3.2}+2.7183^{1.3}+2.7183^{0.2}+2.7183^{0.8}) = \mathbf{0.039}$

$e^{0.8}/(e^{3.2}+e^{1.3}+e^{0.2}+e^{0.8}) = 2.7183^{0.8}/(2.7183^{3.2}+2.7183^{1.3}+2.7183^{0.2}+2.7183^{0.8}) = \mathbf{0.070}$

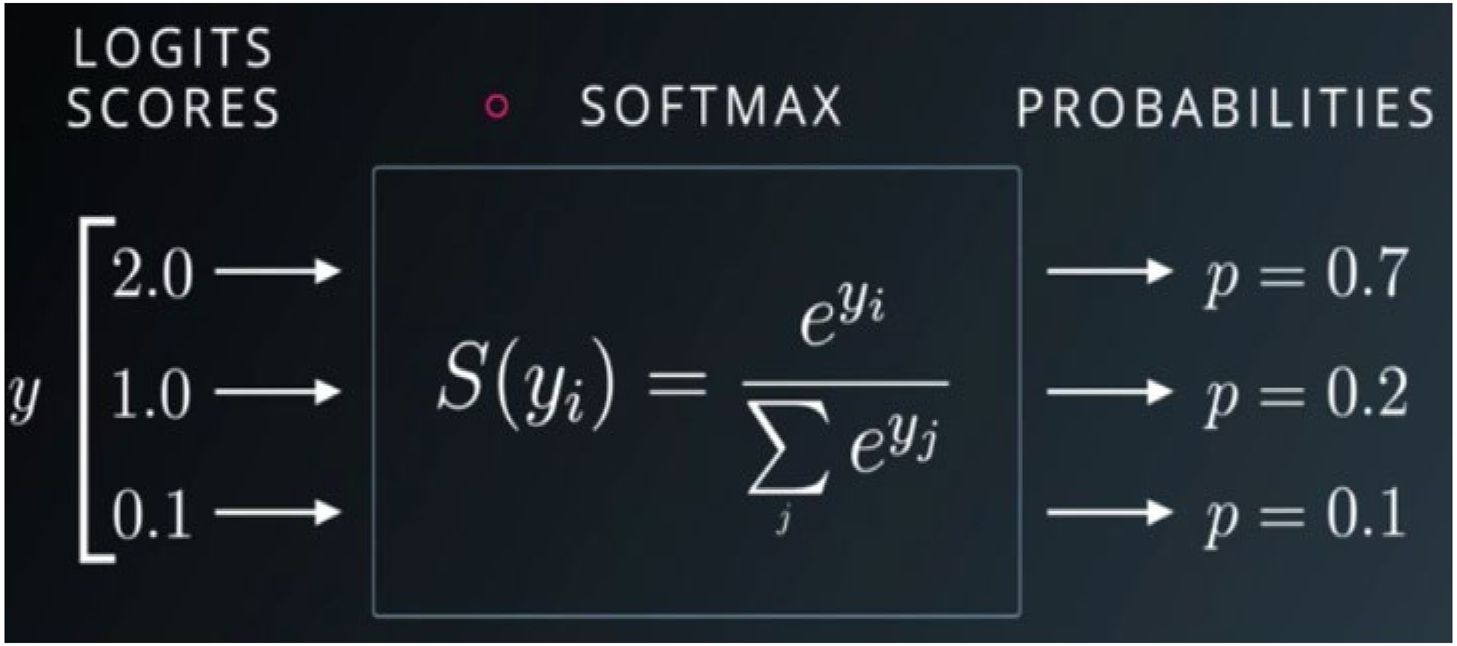
Softmax Function

https://www.mikulskibartosz.name/understanding-the-softmax-activation-function/

- In ML, the **softmax function** acts as the **output function of the last layer of the NN model** (if the network has ***n*** layers, the ***n*****th** layer is the **softmax** function layer).
- The **softmax function** is an **arg max** function: *it does not return the largest value from the input, but it indicates the position of the input value in terms of their probabilities.*
- The **softmax** function generates a **high probability** for a **high value.**
- The sum of all the **probabilities** is equal to **1**.
- **Softmax Function Usage:** Used in multiple classification logistic regression model.
- The **softmax** acts as the **probability** of the data class, and the following example shows how it works:

Softmax Function

https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d



- The **softmax function** works in the following way: Given a **vector** of numbers (the results from the **(n-1)th** layer). It returns the **probability** of the largest value being the ***i***th element of the **vector** if we have an input: ***y*** = [**2.0, 1.0, 0.1**] {**e = 2.7183**}
  - The **softmax f**unction generates **probabilities:** [**0.7, 0.2, 0.1**] **{ sum of these is 1}**
  - It indicates that the first value of ***y*** (**2.0**) has more probability (**0.7**)

Calculating Cross Entropy

https://datascience.stackexchange.com/questions/20296/cross-entropy-loss-explanation

- Suppose a neural net model is designed for **classification**, and its **last layer** is a **dense layer** with the **SoftMax function**. Assume that an **input dataset of five classes** needs to be trained. Suppose the **output label** of the first data (in **one-hot encoded** vector) is [**1 0 0 0 0**] and suppose its **predictions by the softmax function** are [**0.1 0.52 0.3 0.08 0.27**].
- The following **cross-entropy loss** formula takes in **two distributions**: ***p*(*x*)**, **the actual distribution from a one-hot-encode vector**, and ***q*(*x*)**, the **estimated distribution from softmax function** (where ***x*** is the elements of ***p*** and ***q***):

$$\boxed{H(p,q) = -\sum_{\forall x} p(x)\log(q(x))}$$

where **log** = $\mathbf{log_2}$ or $\mathbf{log_e}$

- For an NN model, the **cross-entropy loss** calculation is independent of the following:
  – ***What kind of layer was used?***
  – ***What kind of activation function was used?***

Calculating Cross Entropy

https://datascience.stackexchange.com/questions/20296/cross-entropy-loss-explanation

- From the **one-hot-encoded data label**, [**1 0 0 0 0**] and its **softmax estimation**, [**0.1 0.52 0.3 0.08 0.27**], the **cross-entropy loss *L*** is calculated as:

= $-(1\times\log_2(0.1)$ = $-1\times\log_2(0.1)$ = -1(**log (0.1)/log 2) = -1(-1/0.30103) = <u>3.322</u>**

OR The **entropy loss** result using **natural log (ln** or **log$_e$):**

= $-(1\times\log_e(0.1)$ = $-(1\times\ln(0.1)$ = **<u>2.303</u>**

- In **cross-entropy loss calculation,** when using **log$_2$**, *the unit of entropy is in **bits***, whereas with **natural log** (**ln**, or **log$_e$**), *the unit of entropy is in **nats***.

Calculating Cross Entropy

https://datascience.stackexchange.com/questions/20296/cross-entropy-loss-explanation

- From the **one-hot-encoded data label**, [**1 0 0 0 0**] and its **softmax estimation**, [**0.1 0.52 0.3 0.08 0.27**], the **cross-entropy loss *L*** of each class is calculated as:

Class1 [**1 0 0 0 0**] = $-1 \times \log_2(0.1)$ = **3.322** (very high loss)

Class2 [**0 1 0 0 0**] = $-1 \times \log_2(0.52)$ = **0.9434** (low loss)

Class3 [**0 0 1 0 0**] = $-1 \times \log_2(0.3)$ = **1.685** (high loss)

Class4 [**0 0 0 1 0**] = $-1 \times \log_2(0.08)$ = **3.644** (very high loss)

Class5 [**0 0 0 0 1**] = $-1 \times \log_2(0.27)$ = **1.889** (high loss)

- The training will continue until it reaches a **categorical cross-entropy loss** or **cost function (*J*) is zero or defined.**

Calculating Cross-Entropy Loss

https://programmathically.com/an-introduction-to-neural-network-loss-functions/

- For the ***cost function*** or **categorical cross-entropy loss**, we need to calculate the ***cross-entropy loss of all the individual training examples of the dataset.***
- ***Its function is similar to the SOSE calculation in MLP***
- A **categorical cross-entropy loss** or **cost function (*J*)** based on **multiclass log loss** for a data set of row size ***N*** might look like this:

$$\boxed{J = -\frac{1}{N}\left(\sum_{i=1}^{N} \mathbf{y_i} \cdot \log(\hat{\mathbf{y}}_\mathbf{i})\right)}$$

From the previous case, the **value *J*:**

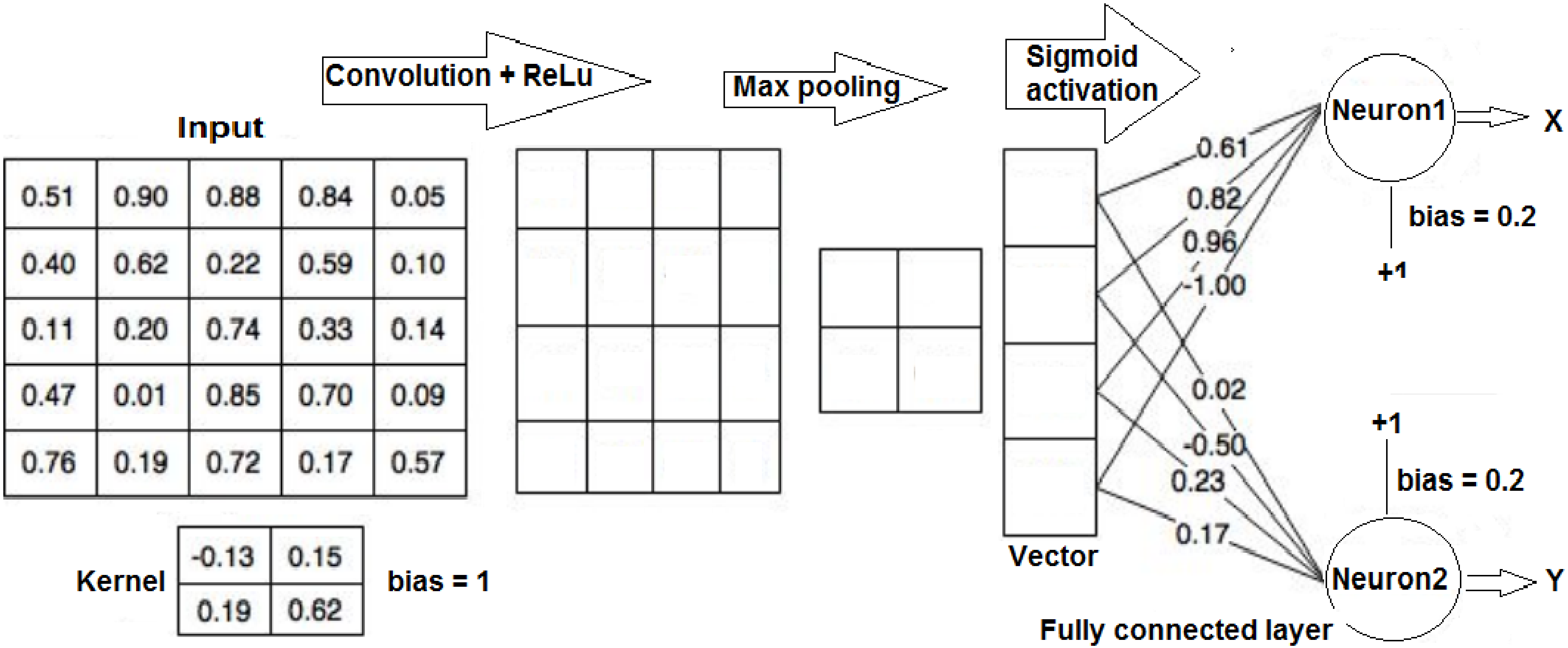
= (3.322 + 0.9434 + 1.685 + 3.644 + 1.889)/5 = 2.2967.

The training will be stopped when *j* = 0 or **near 0.**

Estimating Error with Loss

- In CNN, the ***kernel weights*** and ***bias*** are updated based on the **cross entropy loss** of each image, as explained in the previous slides.
- During the **backpropagation** process, CNN tries to match the **cross-entropy loss** value of each **input image** to its label value, a **one-hot encoded** value.
- It means that if the **cross-entropy loss** value is not equivalent to the image's label (**one-hot encoded** value), the kernel weights and bias are updated via backpropagation.
- The **color images** are converted into **gray images** to reduce the computation complexity of the training.

Exercise: Find X, Y, loss of X, and loss of Y



Input

0.51	0.90	0.88	0.84	0.05
0.40	0.62	0.22	0.59	0.10
0.11	0.20	0.74	0.33	0.14
0.47	0.01	0.85	0.70	0.09
0.76	0.19	0.72	0.17	0.57

Kernel

-0.13	0.15
0.19	0.62

Convolution Stride = 1

Deep Learning Libraries

mxnet

TensorFlow

theano

scikit learn

Caffe2

Keras

PYTORCH

Homework

Can you do this? (5 classes of input images) Show your simulation results.