Unit -3

Convolutional Neural Network (CNN)

Artificial Neural Network

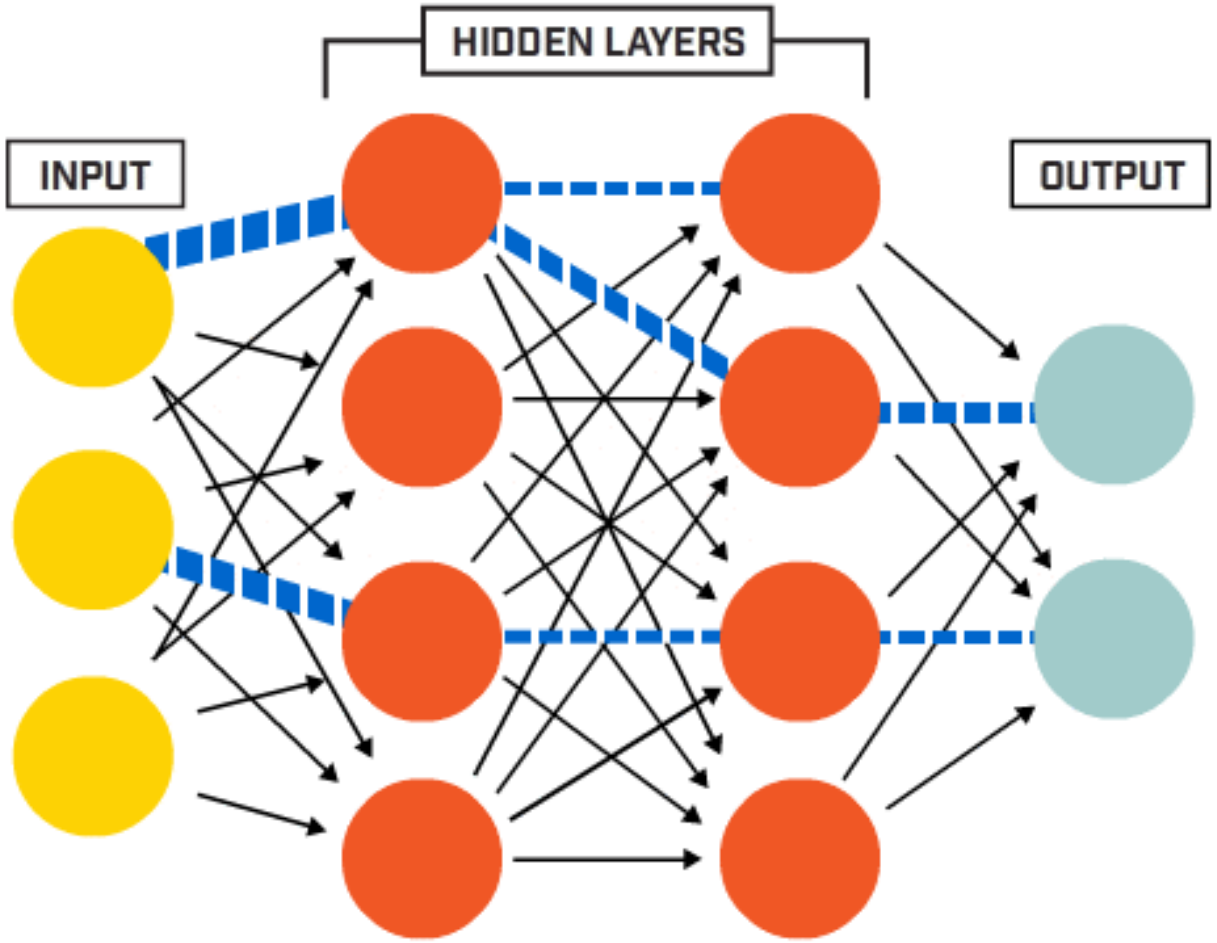
to

Evolution of Deep Learning – Deep Neural Network

Prepared By: Dr. Anil B. Gavade

Study from Previous Unit -2,

Artificial Neural Network (ANN) – Architecture



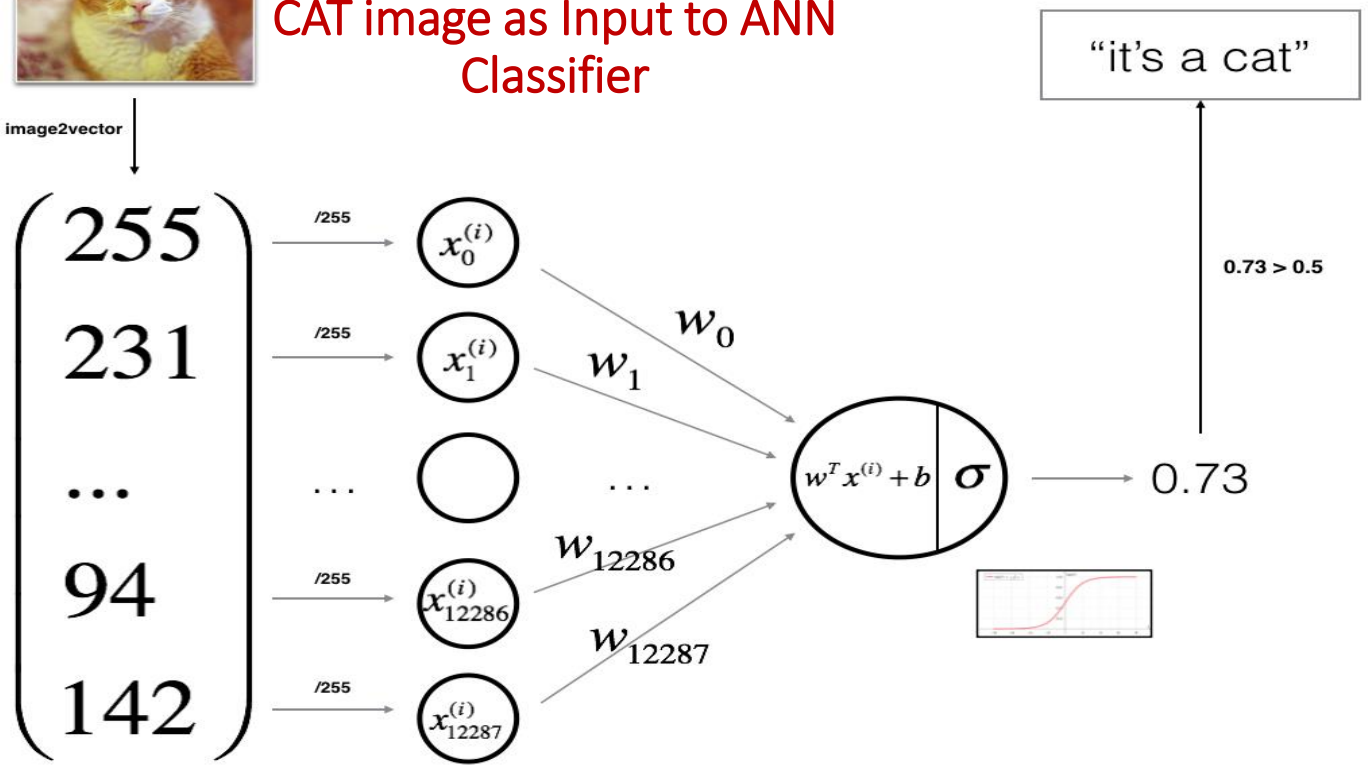
How ANN deals with Image Input?

Convert all columns to Rows and feed to ANN as inputs

1	1	0
4	2	1
0	2	1

1
1
0
4
2
1
0
2
1

CAT image as Input to ANN Classifier



For one example $x^{(i)}$:

$$z^{(i)} = w^T x^{(i)} + b$$
$$\hat{y}^{(i)} = a^{(i)} = sigmoid(z^{(i)})$$
$$\mathcal{L}(a^{(i)}, y^{(i)}) = -y^{(i)} \log(a^{(i)}) - (1 - y^{(i)}) \log(1 - a^{(i)})$$

The cost is then computed by summing over all training examples:

$$J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y^{(i)})$$

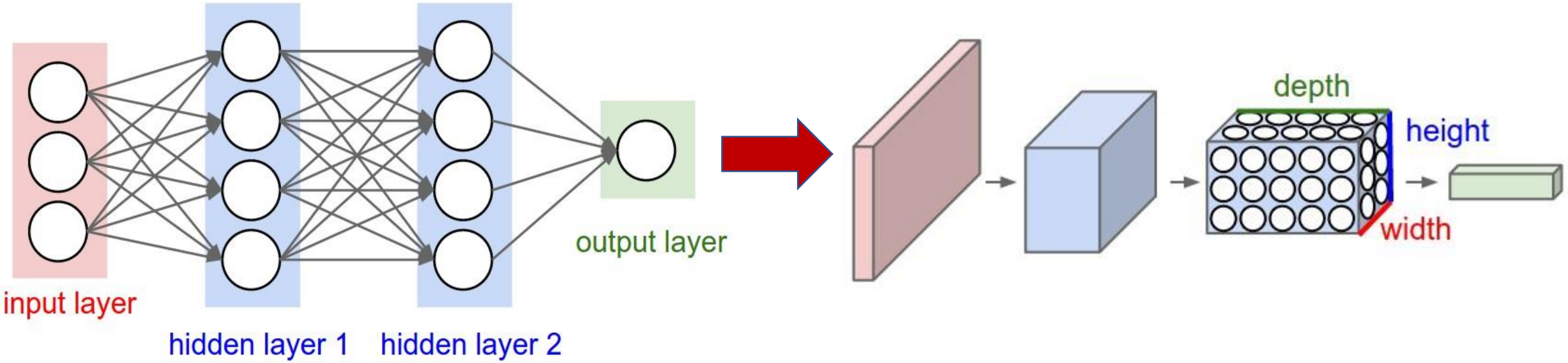
Introduction

Convolutional Neural Networks (CNNs), also known as ConvNets, are a class of deep neural networks that have proven very effective in areas such as image recognition and computer vision. They were designed to mimic the human visual system by leveraging the spatial hierarchy of features in input data.

Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are designed for image data, addressing the scalability issues of regular neural networks. Unlike fully connected layers, CNN layers have neurons arranged in 3D, with local connectivity, reducing the number of parameters. This enables effective feature extraction and hierarchical learning, making CNNs particularly successful in tasks like image recognition, where they transform input images through convolutional and pooling layers to output class scores.

From ANN to CNN:



Left: A regular 3-layer Neural Network. **Right**: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Figure Credit: CS231n: Deep Learning for Computer Vision
http://cs231n.stanford.edu/schedule.html

Convolutional Neural Networks (CNNs) :

Layers used to build ConvNets

Convolutional Layers:

These layers apply convolutional operations to the input data using learnable filters, capturing local patterns and features within the input.

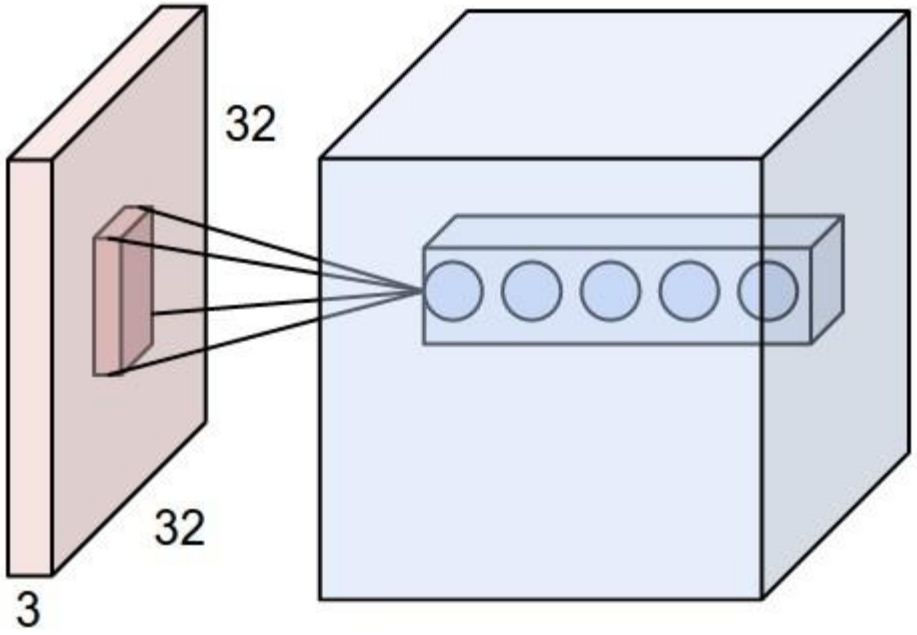
Pooling Layers:

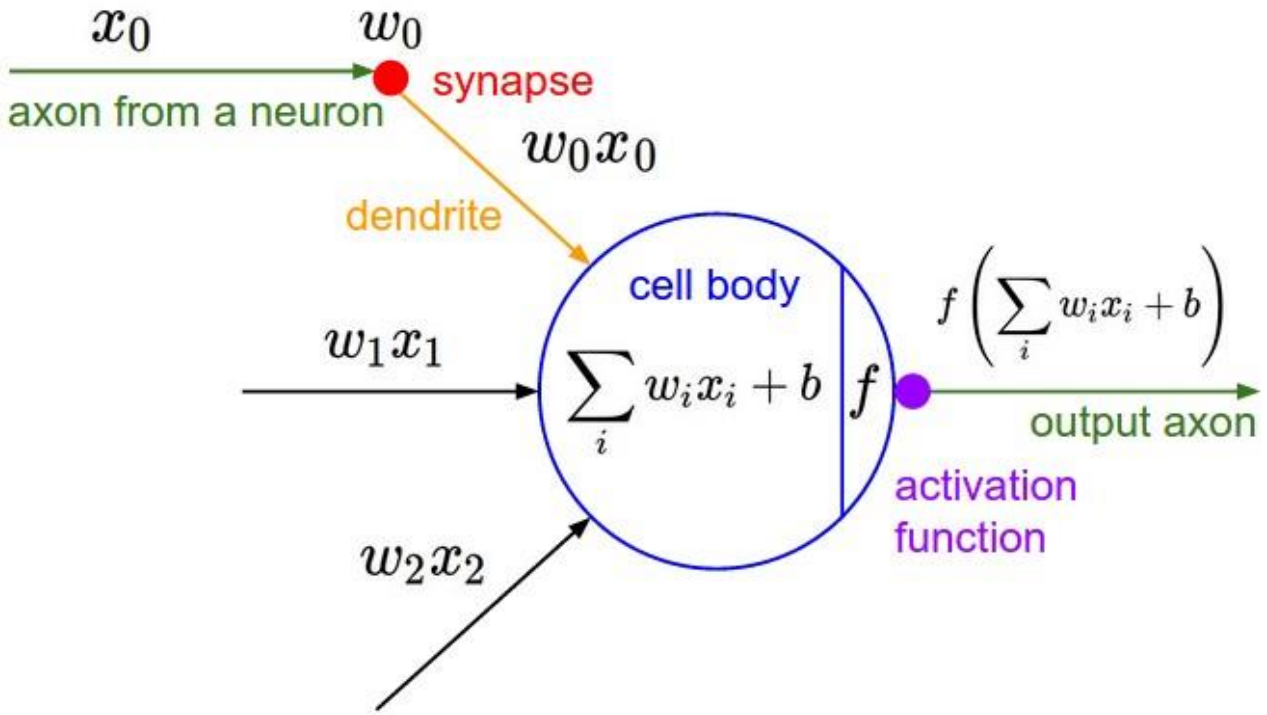
Pooling layers reduce the spatial dimensions of the input volume by selecting the maximum (max pooling) or average (average pooling) values from local regions. This helps in retaining important features while downsampling the data.

Fully Connected Layers:

The fully connected layers are often placed at the end of the CNN architecture. They connect every neuron in one layer to every neuron in the next layer, providing high-level reasoning and making the final predictions.

Convolutional Layer





Convolutional Layer

- **Core Building Block:**
  - The Convolutional Layer is the central component of Convolutional Neural Networks (CNNs) and performs the majority of computational tasks.
- **Learnable Filters:**
  - Parameters of the Conv Layer consist of small, learnable filters that slide across the input volume during the forward pass, capturing visual features like edges or colors.
- **Activation Maps:**
  - Convolution operations generate 2D activation maps for each filter, representing responses to specific visual features, and these maps are stacked along the depth dimension to create the output volume.
- **Local Connectivity:**
  - Neurons in the Conv Layer connect only to local regions of the input volume, determined by the receptive field (filter size), with connectivity extending fully along the depth axis.
- **Example Receptive Field:**
  - In practical terms, for an input volume of size [32x32x3], a 5x5 receptive field results in each neuron having 75 weights, capturing information from a [5x5x3] region in the input volume. This local connectivity allows effective feature learning from high-dimensional inputs like images.

Example Architecture: Overview. We will go into more details below, but a simple ConvNet for **CIFAR-10** classification could have the architecture [INPUT - CONV - RELU - POOL - FC].

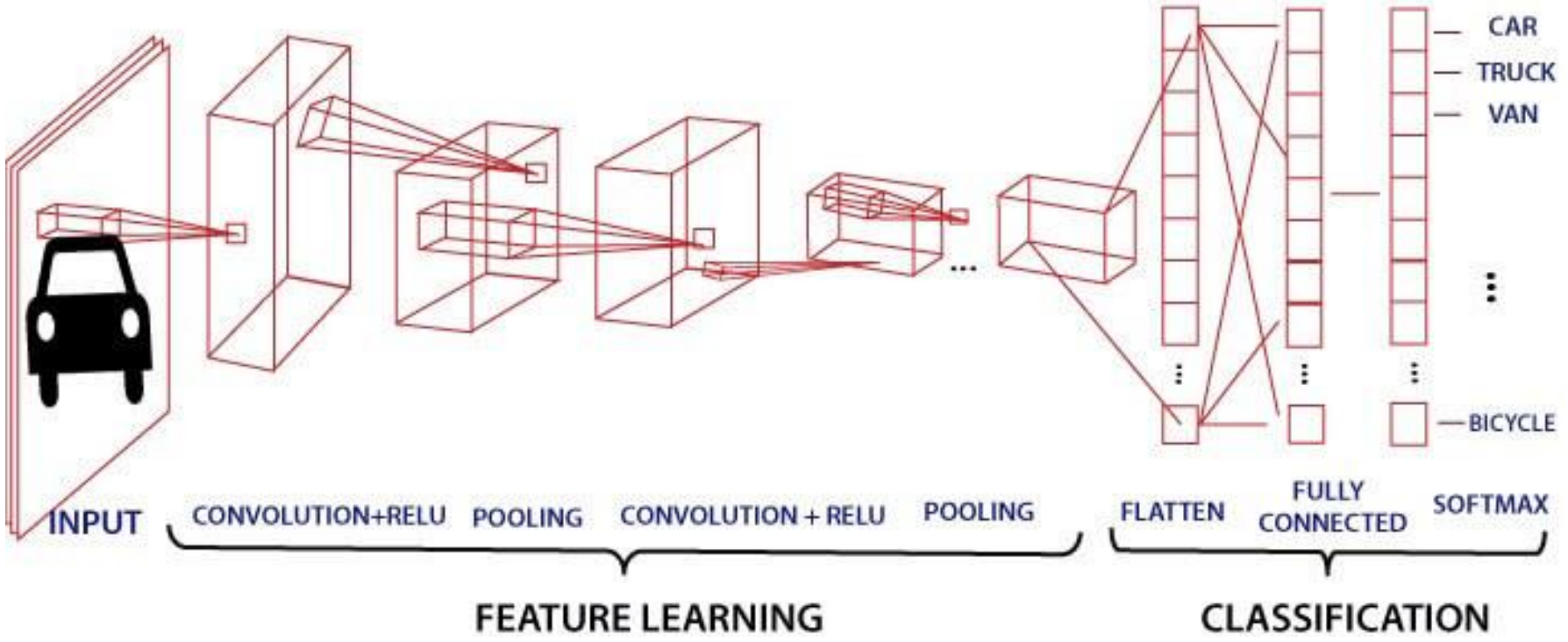
- **INPUT [32x32x3]** will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- **CONV layer** will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
- **RELU** layer will apply an elementwise activation function, such as the ***max*(0,*x*)** thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).

- **POOL layer** will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- **FC (i.e. fully-connected)** layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

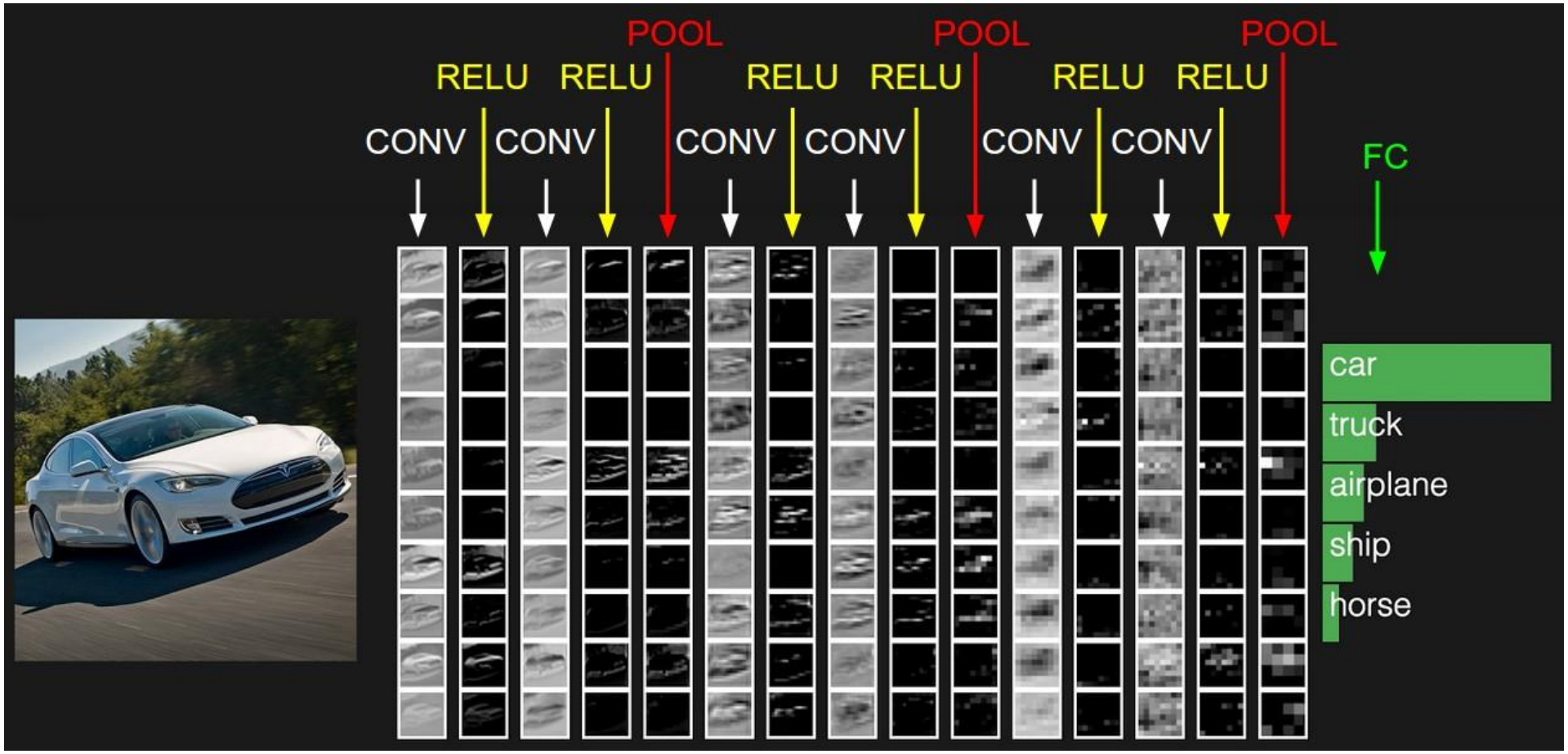
In summary:

- A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume (e.g. holding the class scores)
- There are a few distinct types of Layers (e.g. CONV/FC/RELU/POOL are by far the most popular)
- Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function
- Each Layer may or may not have parameters (e.g. CONV/FC do, RELU/POOL don't)
- Each Layer may or may not have additional hyperparameters (e.g. CONV/FC/POOL do, RELU doesn't)

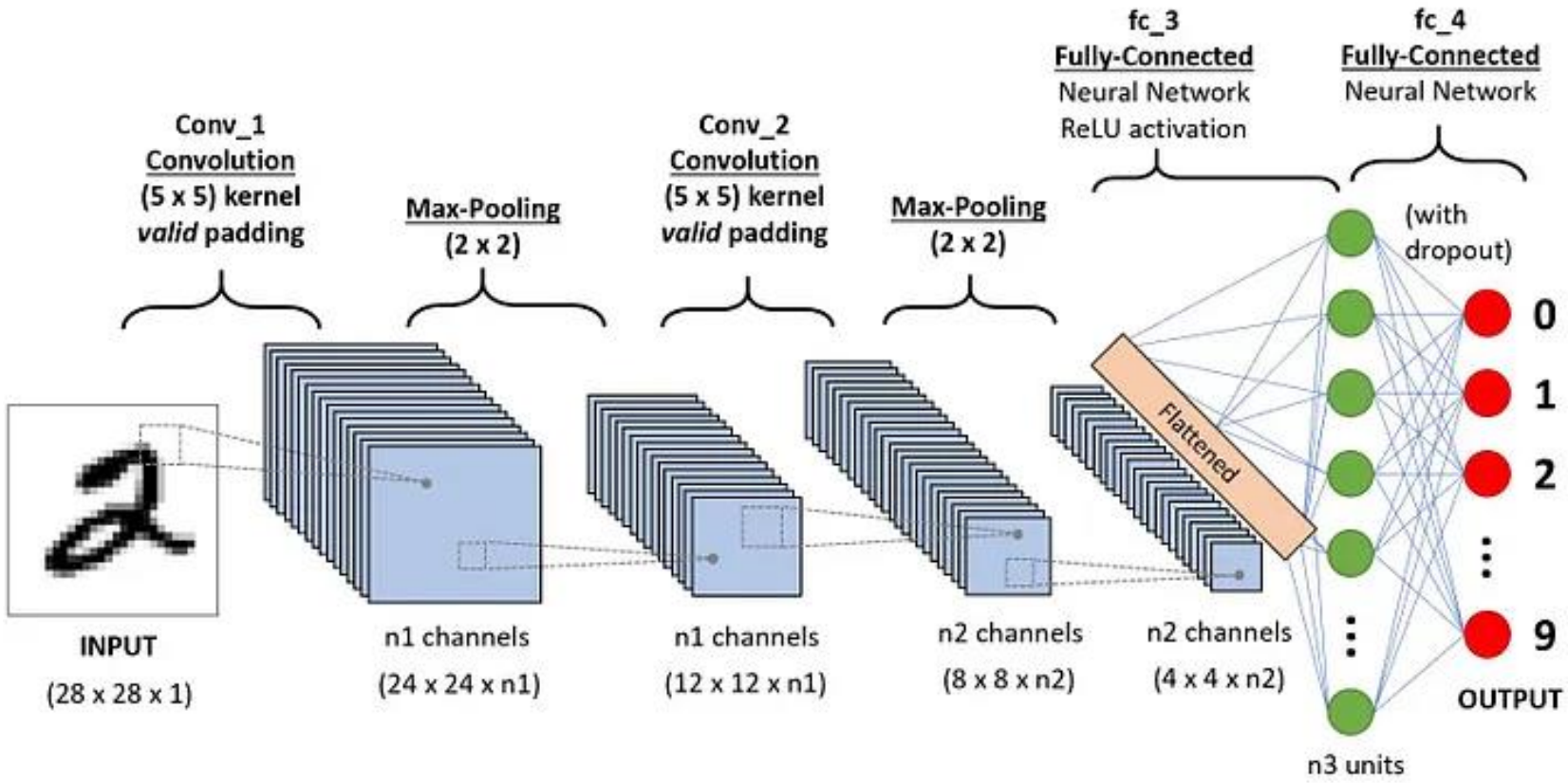
How CNN works: Example



How CNN works – Example



How CNN works – Example with MNIST digit datasets



How CNN works Convolution Layers

The Convolution Layers are the initial layers to pull out features from the image. It maintains the relationship between pixels by learning features using a small input data sequence. It is a mathematical term that takes two inputs, an image matrix and a kernel or filter. The result is calculated by:

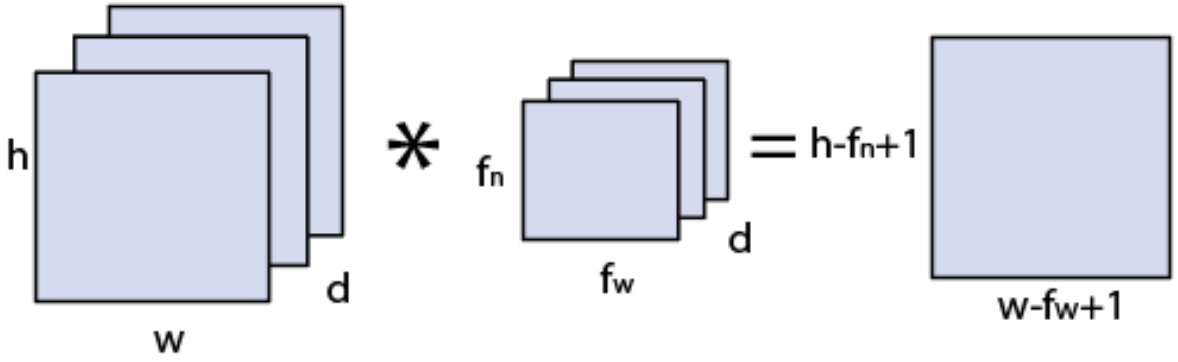


Image matrix multiplies kernl or filter matrix

In the above image,

The image matrix is h x w x d

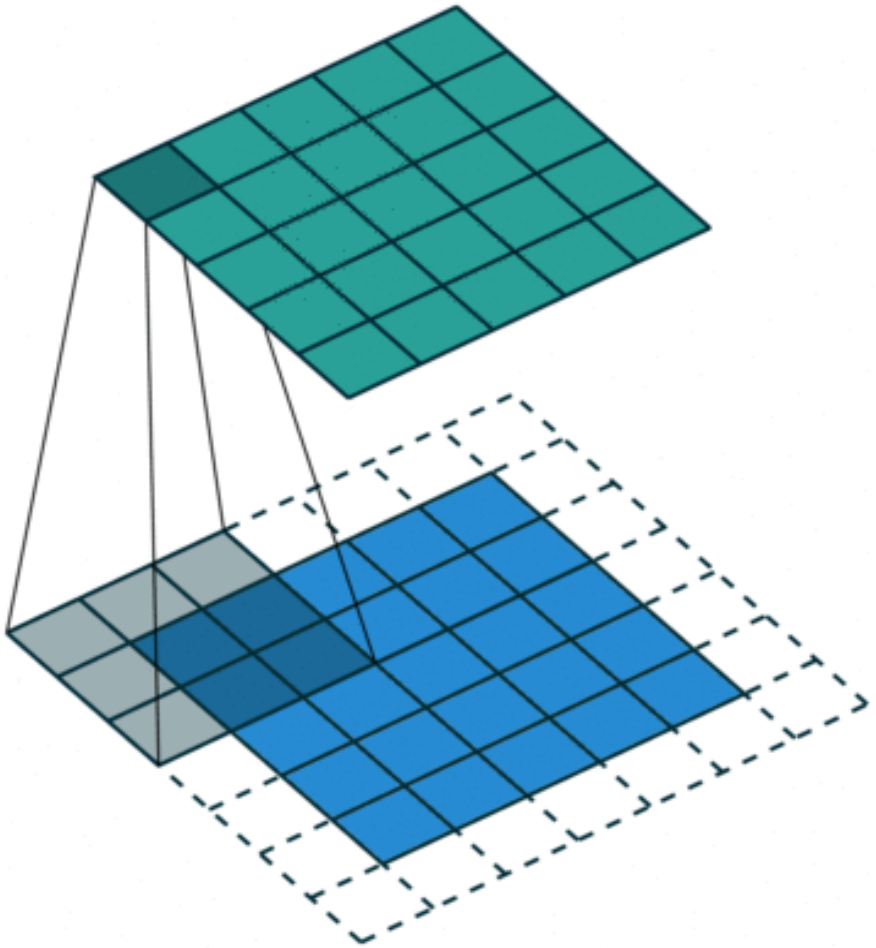
The dimensions of the filter are f_h x f_w x d

The output is calculated as (h- f_h +1)(w- f_w+1) x 1

Convolutions

- **Kernel Size**: The kernel size defines the field of view of the convolution. A common choice for 2D is 3 — that is 3x3 pixels.
- **Stride**: The stride defines the step size of the kernel when traversing the image. While its default is usually 1, we can use a stride of 2 for down sampling an image similar to MaxPooling.
- **Padding**: The padding defines how the border of a sample is handled. A (half) padded convolution will keep the spatial output dimensions equal to the input, whereas unpadded convolutions will crop away some of the borders if the kernel is larger than 1.
- **Input & Output Channels**: A convolutional layer takes a certain number of input channels (I) and calculates a specific number of output channels (O). The needed parameters for such a layer can be calculated by I*O*K, where K equals the number of values in the kernel.

Convolutions of 2 dimension data:



2D convolution using a kernel

$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

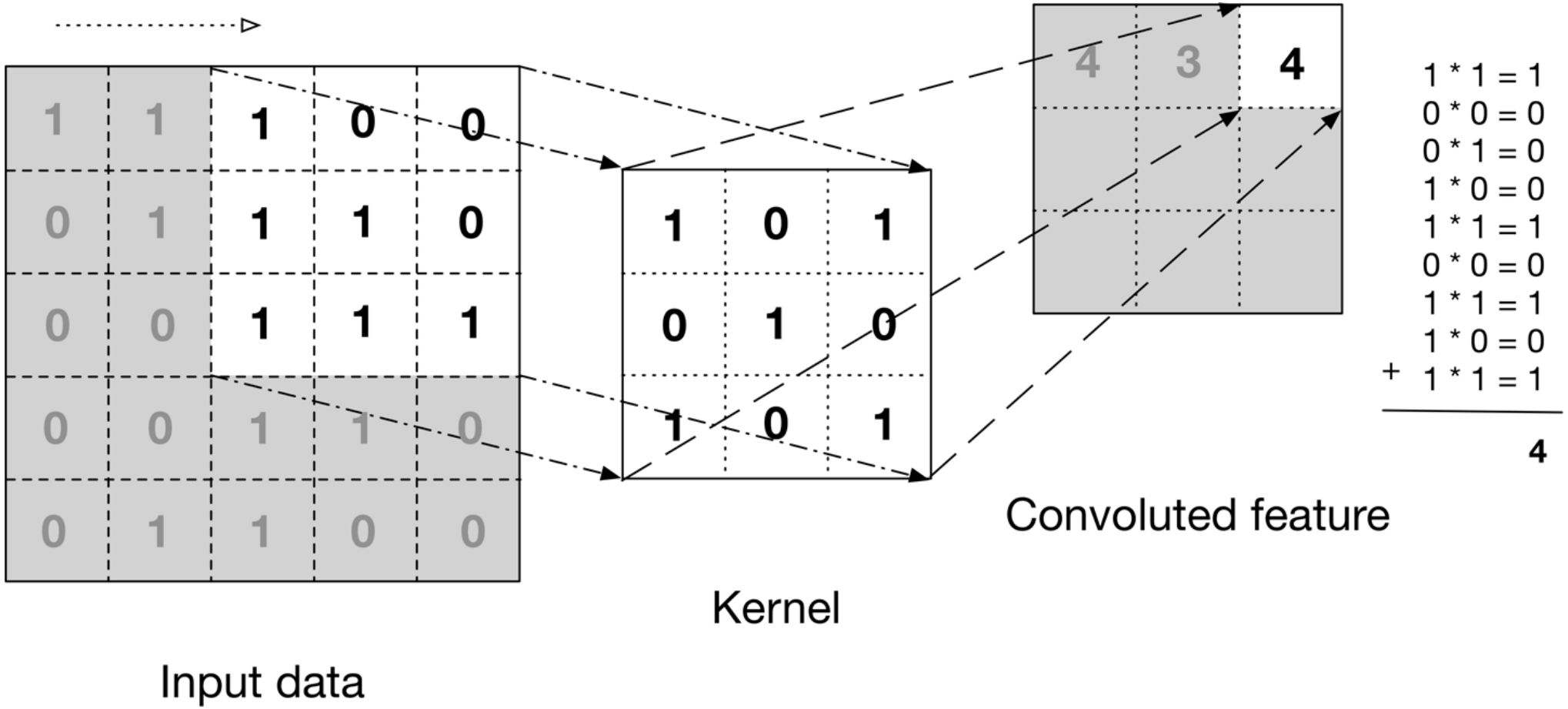
Image

4		

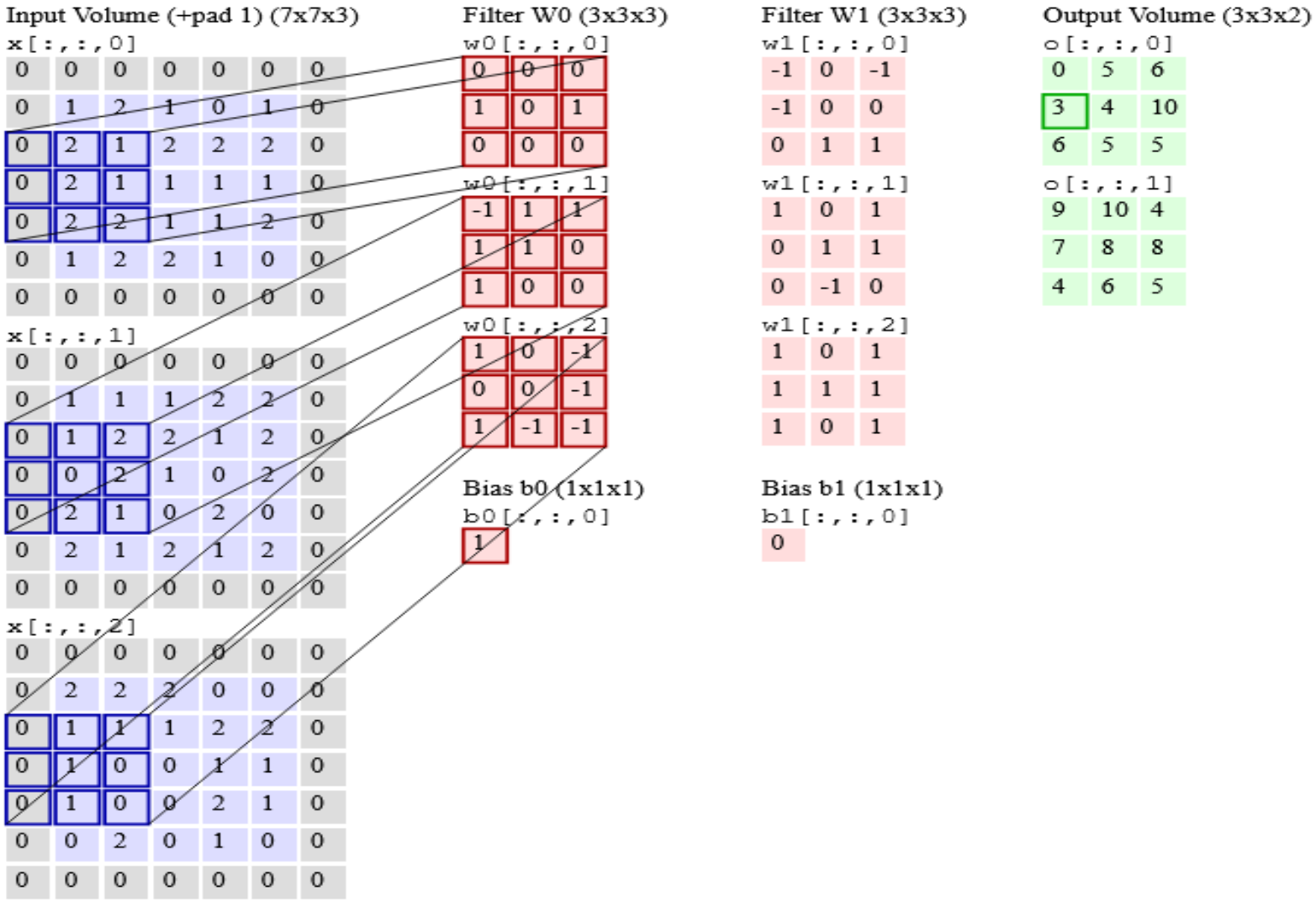
Convolved Feature

Convolution operation with Kernel 3X3

Convolutions of 2 dimension data:



Let’s consider an image of size 5X5 size and 3 filters (since each filter will be used for each color channel: RGB) of 3X3 size.



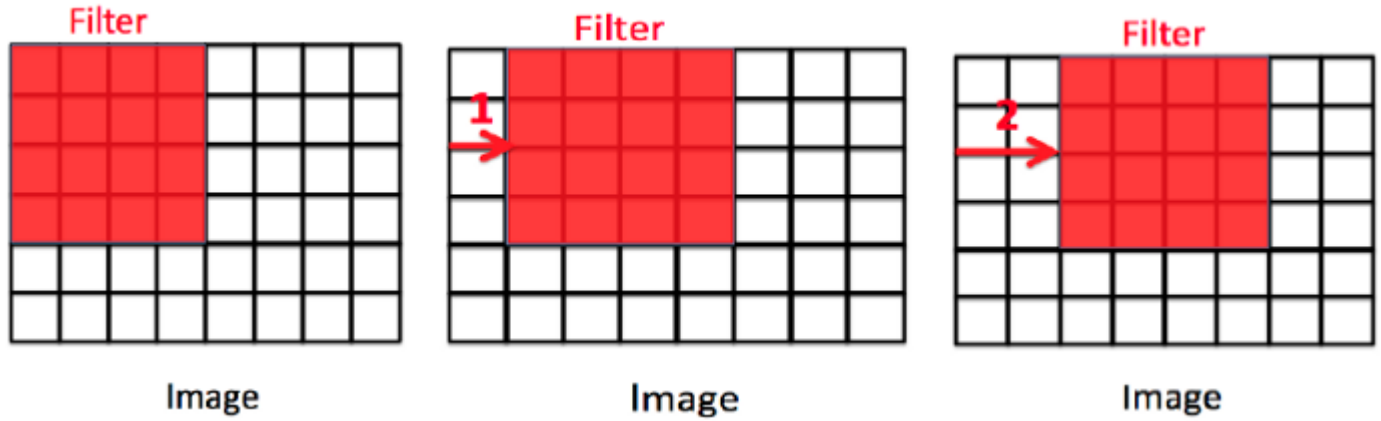
Then the convolution of 5 x 5 image matrix multiplies with 3 x 3 filter matrix which is called **“Feature Map”**. We apply the dot product to the scaler value and then move the filter by the stride over the entire image.

Sometimes filter does not fit perfectly fit the input image. Then there is a need to pad the image with zeros as shown below. This is called **padding**

Next, we need to reduce the size of images, if they are too large. **Pooling layers** section would reduce the number of parameters when the images are too large

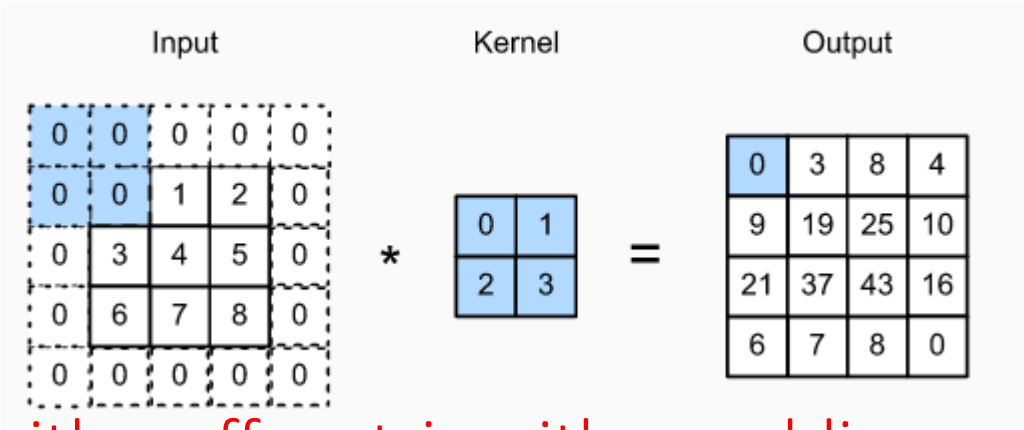
Strides

When the array is created, the pixels are shifted over to the input matrix. The number of pixels turning to the input matrix is known as the strides. When the number of strides is 1, we move the filters to 1 pixel at a time. Similarly, when the number of strides is 2, we carry the filters to 2 pixels, and so on. They are essential because they control the convolution of the filter against the input, i.e., Strides are responsible for regulating the features that could be missed while flattening the image. They denote the number of steps we are moving in each convolution. The following figure shows how the convolution would work.



Padding:

- The padding plays a vital role in creating CNN. After the convolution operation, the original size of the image is shrunk. Also, in the image classification task, there are multiple convolution layers after which our original image is shrunk after every step, which we don't want.
- Secondly, when the kernel moves over the original image, it passes through the middle layer more times than the edge layers, due to which there occurs an overlap.
- To overcome this problem, a new concept was introduced named padding. It is an additional layer that can add to the borders of an image while preserving the size of the original picture. For example:

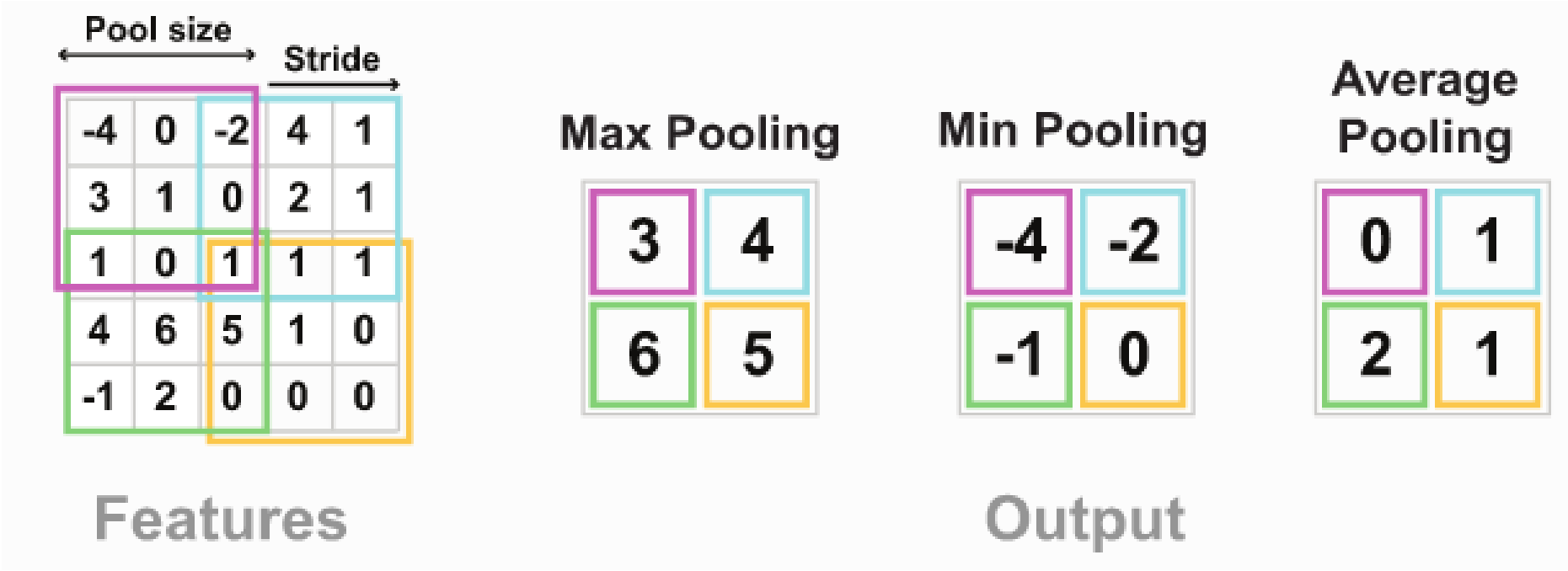


So, if an n x n matrix is convolved with an ff matrix with a padding p, then the size of the output image will be:
(n+2p-f+1) x (n+2p-f+1)

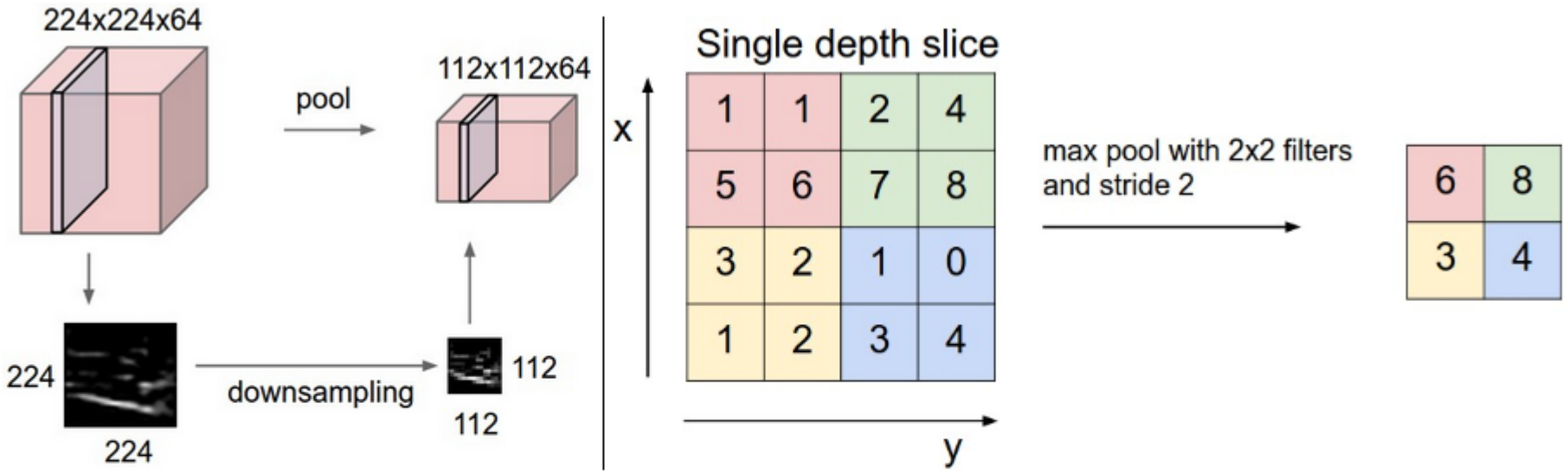
Pooling:

The pooling layer is another building block of a CNN and plays a vital role in pre-processing an image. In the pre-process, the image size shrinks by reducing the number of parameters if the image is too large. When the picture is shrunk, the pixel density is also reduced, the downscaled image is obtained from the previous layers. Basically, its function is to progressively reduce the spatial size of the image to reduce the network complexity and computational cost. Spatial pooling is also known as downsampling or subsampling that reduces the dimensionality of each map but retains the essential features. A rectified linear activation function, or ReLU, is applied to each value in the feature map. Relu is a simple and effective nonlinearity that does not change the values in the feature map but is present because later subsequent pooling layers are added. Pooling is added after the nonlinearity is applied to the feature maps. There are three types of spatial pooling:

Pooling (CNN)

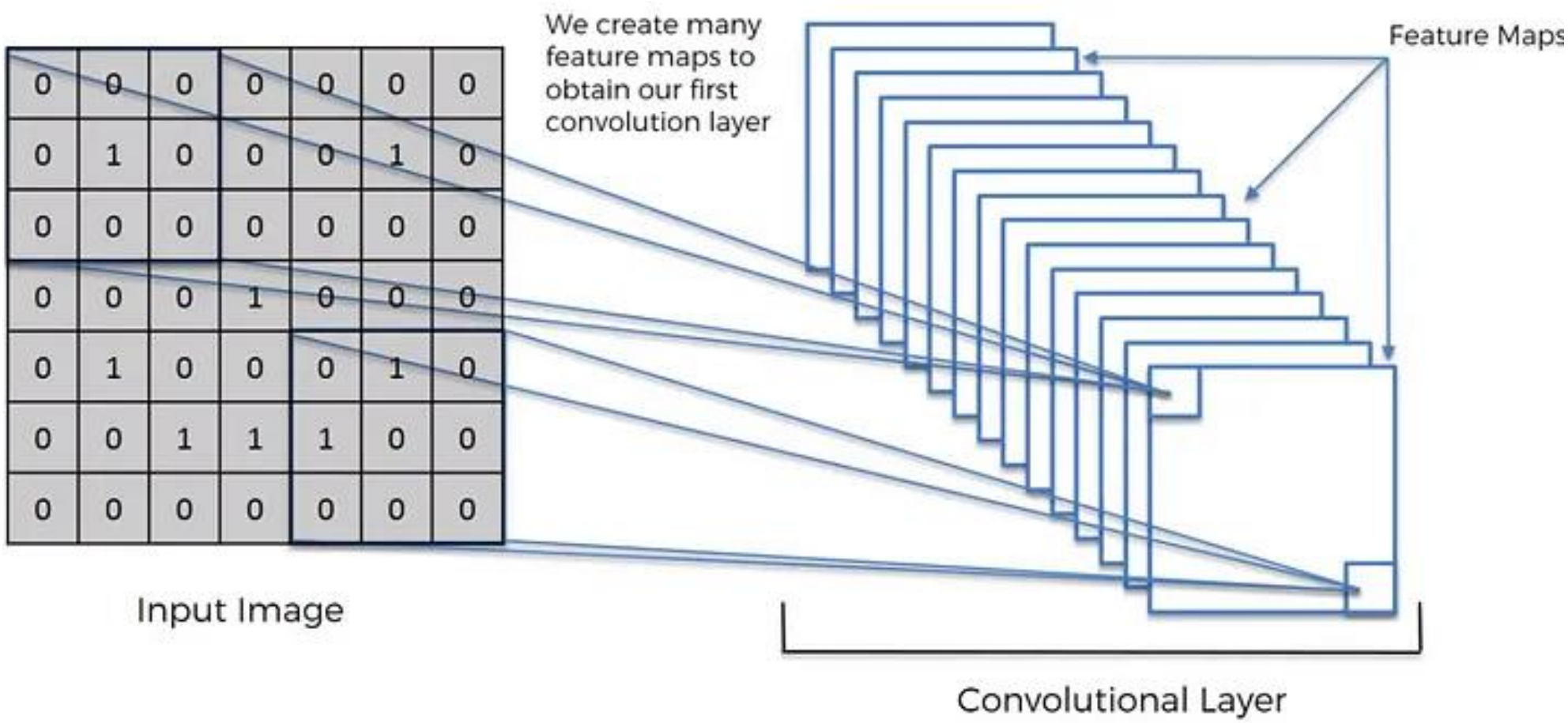

Pool size
Stride
-4 0 -2 4 1
3 1 0 2 1
1 0 1 1 1
4 6 5 1 0
-1 2 0 0 0
Features
Max Pooling
3 4
6 5
Min Pooling
-4 -2
-1 0
Average Pooling
0 1
2 1
Output

General pooling. In addition to max pooling, the pooling units can also perform other functions, such as *average pooling* or even *L2-norm pooling*. Average pooling was often used historically but has recently fallen out of favor compared to the max pooling operation, which has been shown to work better in practice.

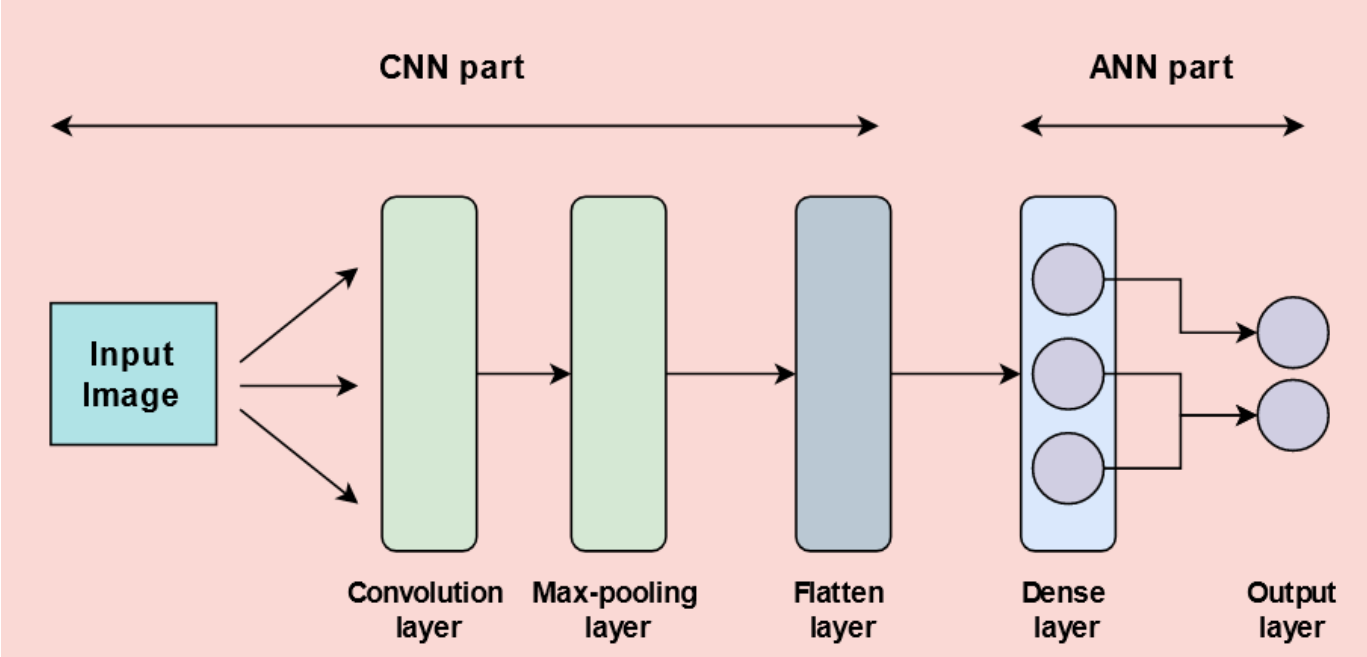


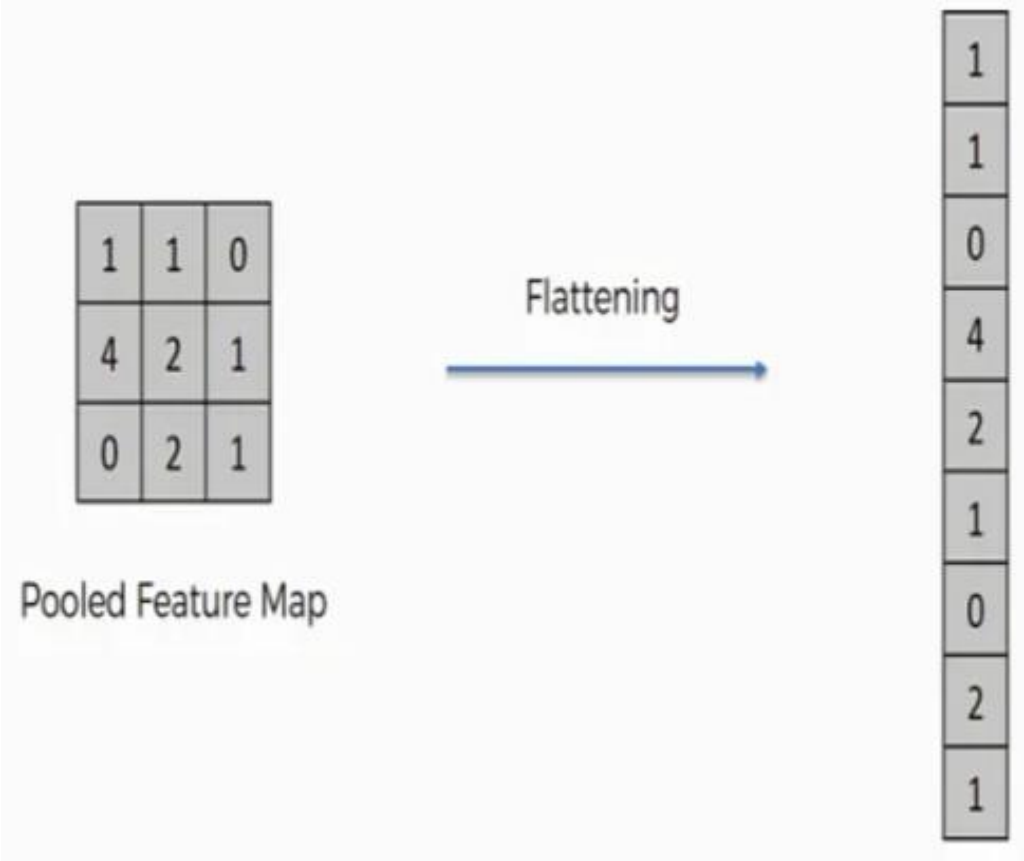
Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

Feature Map after Convolution



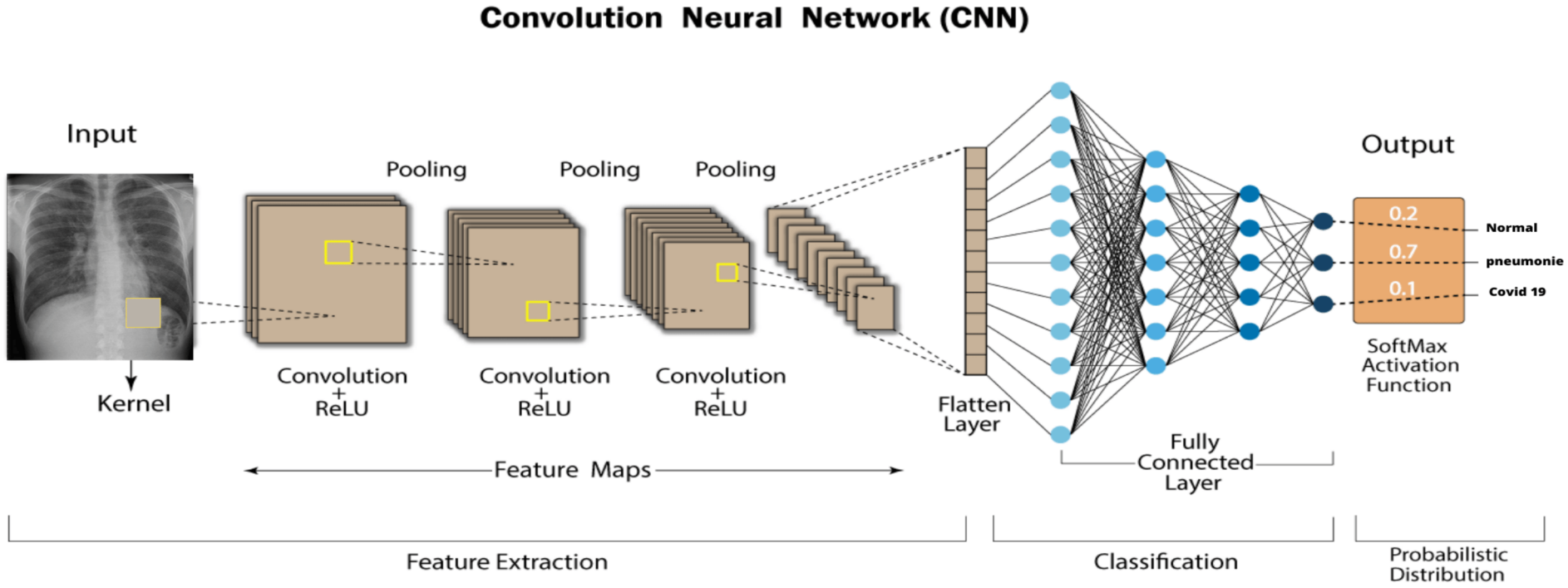
Flattening Layer in CNN





The Flatten layer converts the multi-dimensional feature maps received from the pooling layer into a one-dimensional array. This transformation is necessary for dense layers, which require one-dimensional input for further processing.

Fully Connected Layer

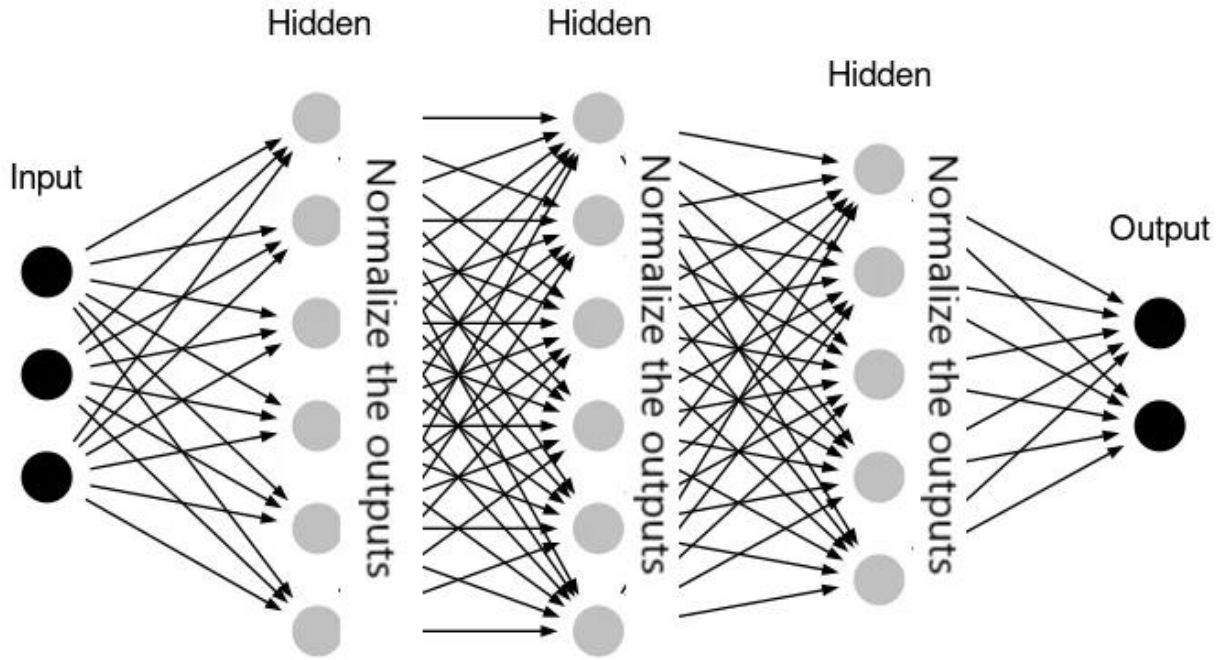


A Fully Connected (FC) layer, also known as a dense layer, links each neuron from the previous layer to every neuron in the current layer, forming complete connections. Typically located at the end of a neural network, FC layers produce final output predictions.

Key Features

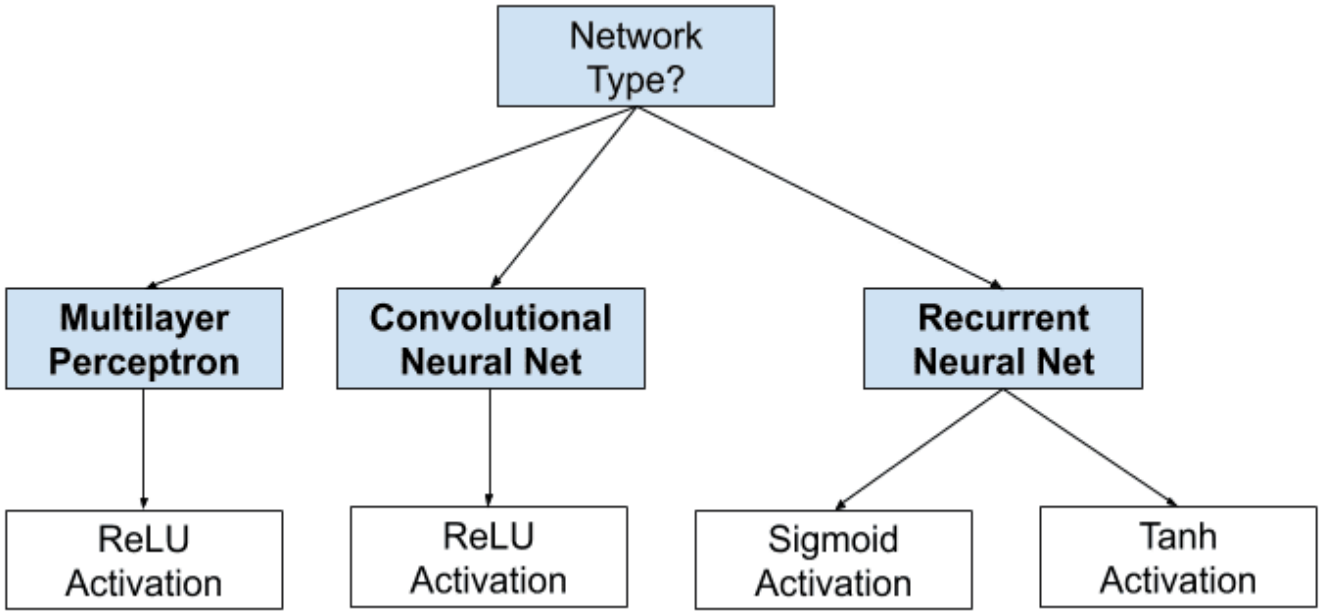
- In CNNs, FC layers often come after the convolutional and pooling layers. They are used to flatten the 2D spatial structure of the data into a 1D vector and process this data for tasks like classification.
- The weights and biases in FC layers are learned during the training process, making them adapt to the specific problem at hand.
- The number of neurons in the final FC layer usually matches the number of output classes in a classification problem. For instance, for a 10-class digit classification problem, there would be 10 neurons in the final FC layer, each outputting a score for one of the classes.

Batch Normalization for Fully Connected Layer



- **Batch Normalization (BN)** is a technique introduced to normalize the activations in the network, which can lead to faster training and improved performance. It addresses the problem of internal covariate shift by normalizing the layer's inputs across each mini-batch.
- For a Fully Connected layer, batch normalization would be applied to the output of the FC layer and before the activation function.

Activation Function and Output Layer

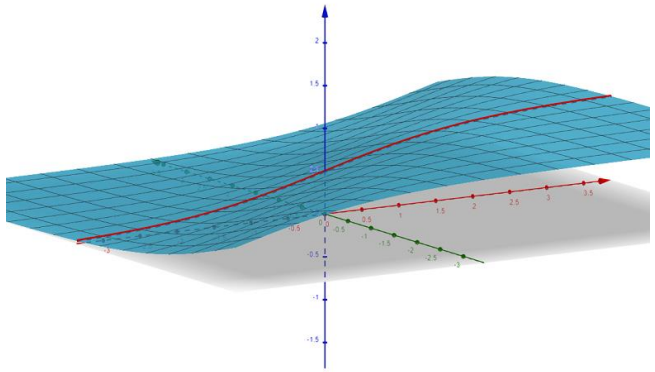


Activation Function: An activation function, like ReLU (Rectified Linear Unit), introduces non-linearity to a neural network, enabling it to learn complex patterns by determining the output of each neuron based on the weighted sum of its inputs.

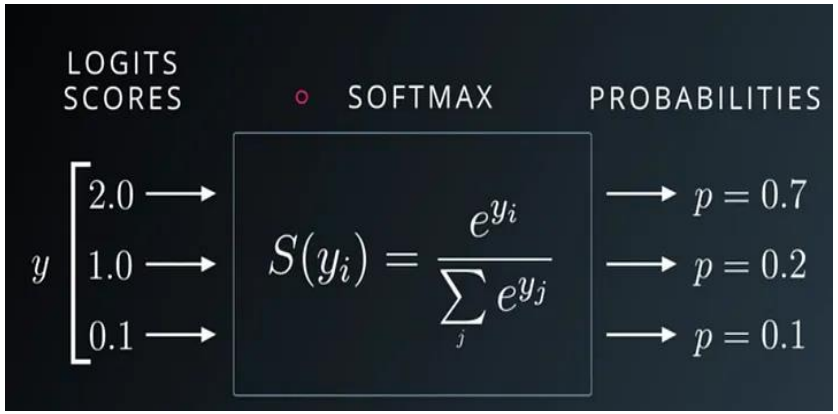
Output Layer: The output layer is responsible for producing the final predictions of the neural network. It often uses specific activation functions (e.g., softmax for classification) to generate output values corresponding to different classes or categories in a given task.

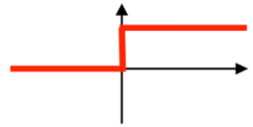
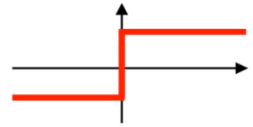
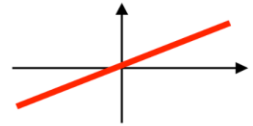
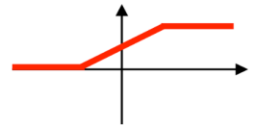
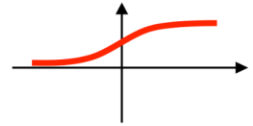
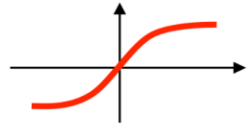
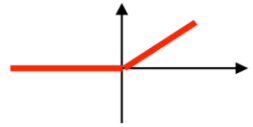
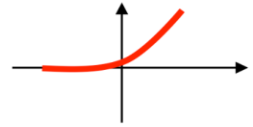
Activation Function

Example:
Softmax Activation Function

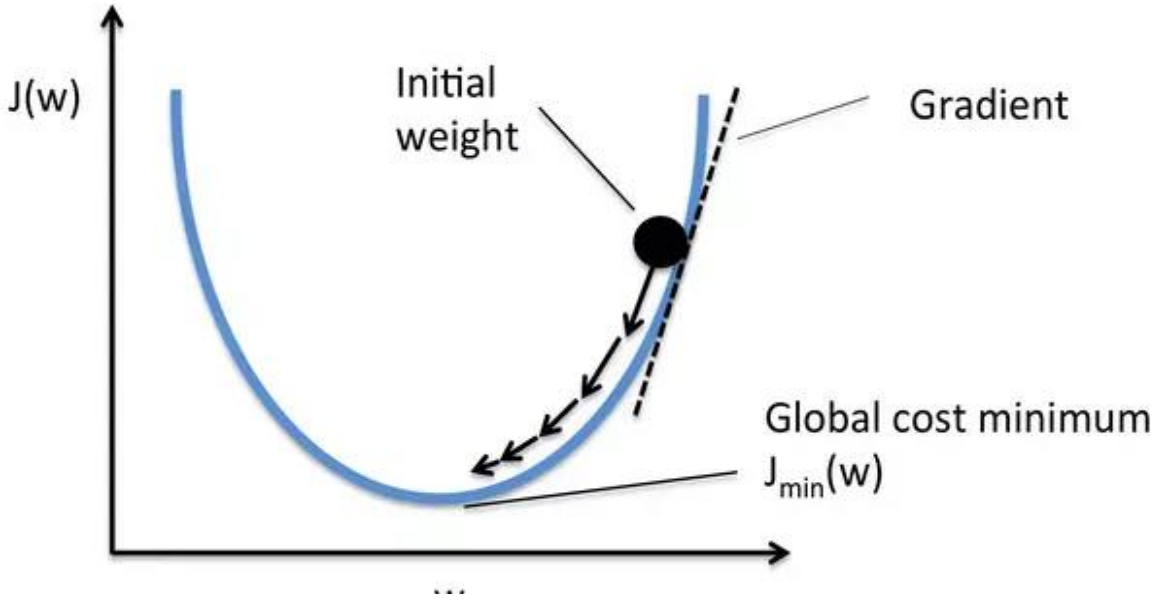


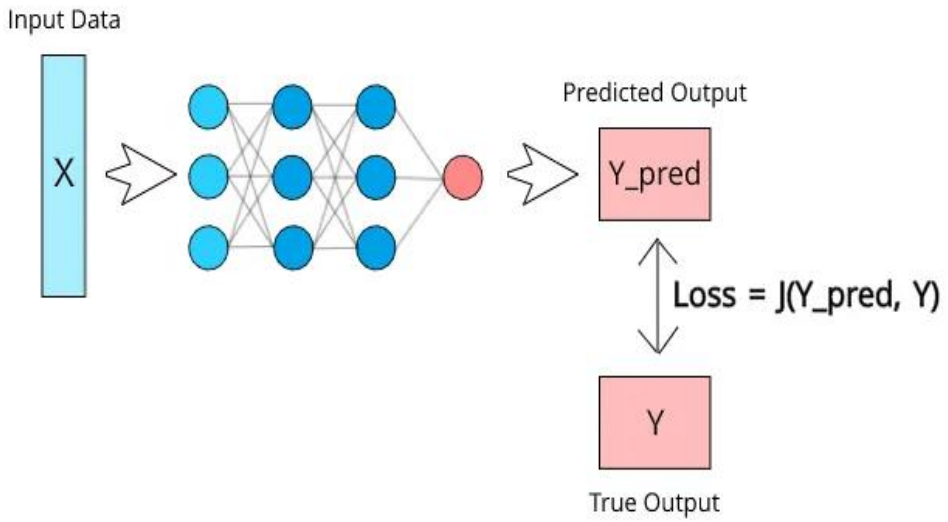
$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$



Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

Loss Function and Optimization Algorithm





Loss Function: A loss function measures the difference between the predicted values and the actual values in the training data. It quantifies the model's performance, guiding the optimization process by providing a signal for adjusting the model's parameters to minimize the error.

Optimization Algorithm: An optimization algorithm, such as Stochastic Gradient Descent (SGD) or Adam, is employed to iteratively update the model's parameters during training. The algorithm aims to minimize the loss function, finding the optimal configuration that best fits the training data and generalizes well to unseen examples.

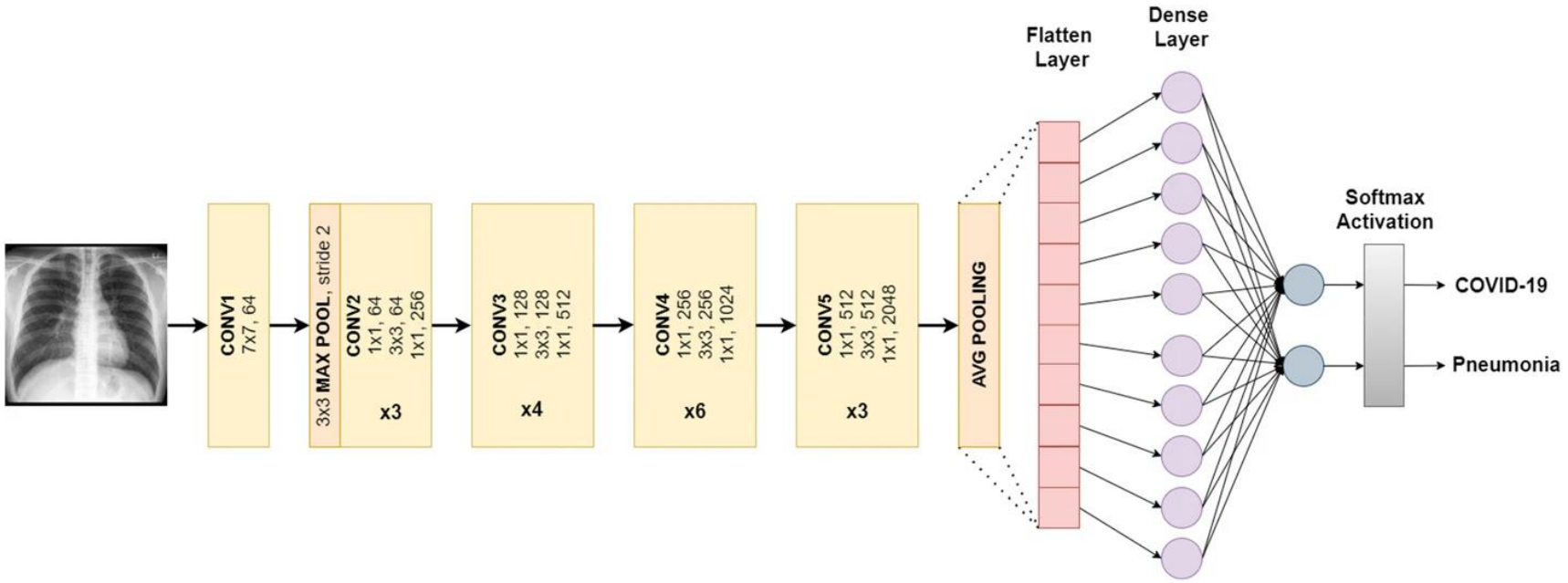
Training, Validation and Testing

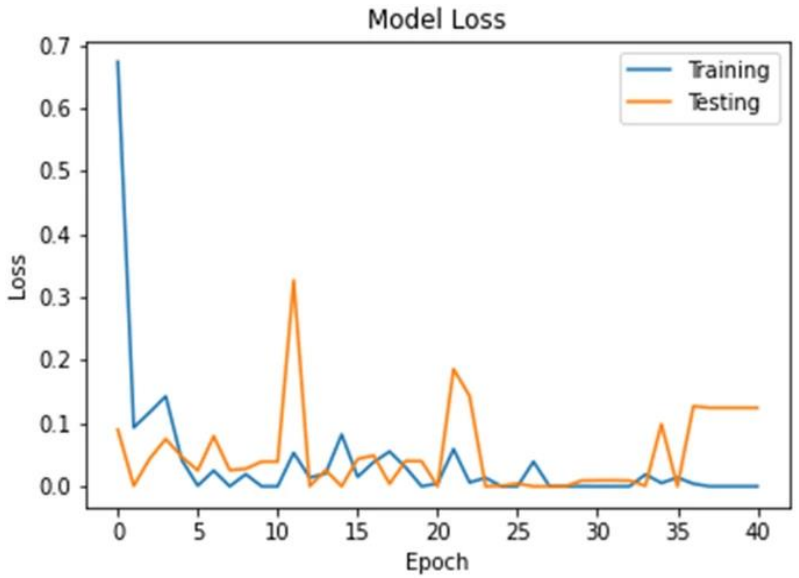
Training: During the training phase of a neural network, the model learns from labeled training data. The optimization algorithm adjusts the model's parameters based on the computed loss, progressively improving its ability to make accurate predictions.

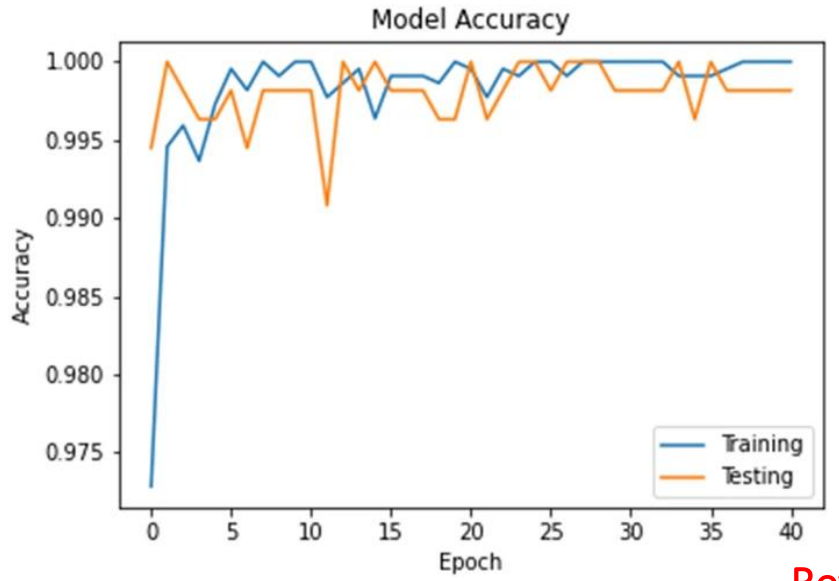
Validation: Validation is a separate dataset used to assess the model's performance during training. It helps prevent overfitting by evaluating the model on data it hasn't seen before. The validation results guide decisions like adjusting hyperparameters or stopping training to achieve better generalization.

Testing: The testing phase involves evaluating the final, trained model on a completely unseen dataset to assess its ability to generalize to new, unseen examples. Testing provides a reliable measure of the model's overall performance and its suitability for real-world applications.

Detection of COVID-19 on chest X-ray - CNN Image Classification







Classification parameters: *where* TP, TN, FP, and FN stand for true positive, true negative, false positive, and false negative.

$$\text{Accuracy} : \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} : \frac{TP}{TP + FP}$$

$$\text{Recall} : \frac{TP}{TP + FN}$$

$$\text{F1} - \text{score} : \frac{2 \times (Recall \times Precision)}{(Recall + Precision)}$$

Ref: https://doi.org/10.1186/s13634-021-00755-1

Q&A?
Thank you