Hands on Machine Learning
Unit-2 (Part) and Unit-3

Raghavendra.M.J
Department of ECE, PESU.

TEXT BOOK AND REFERENCES

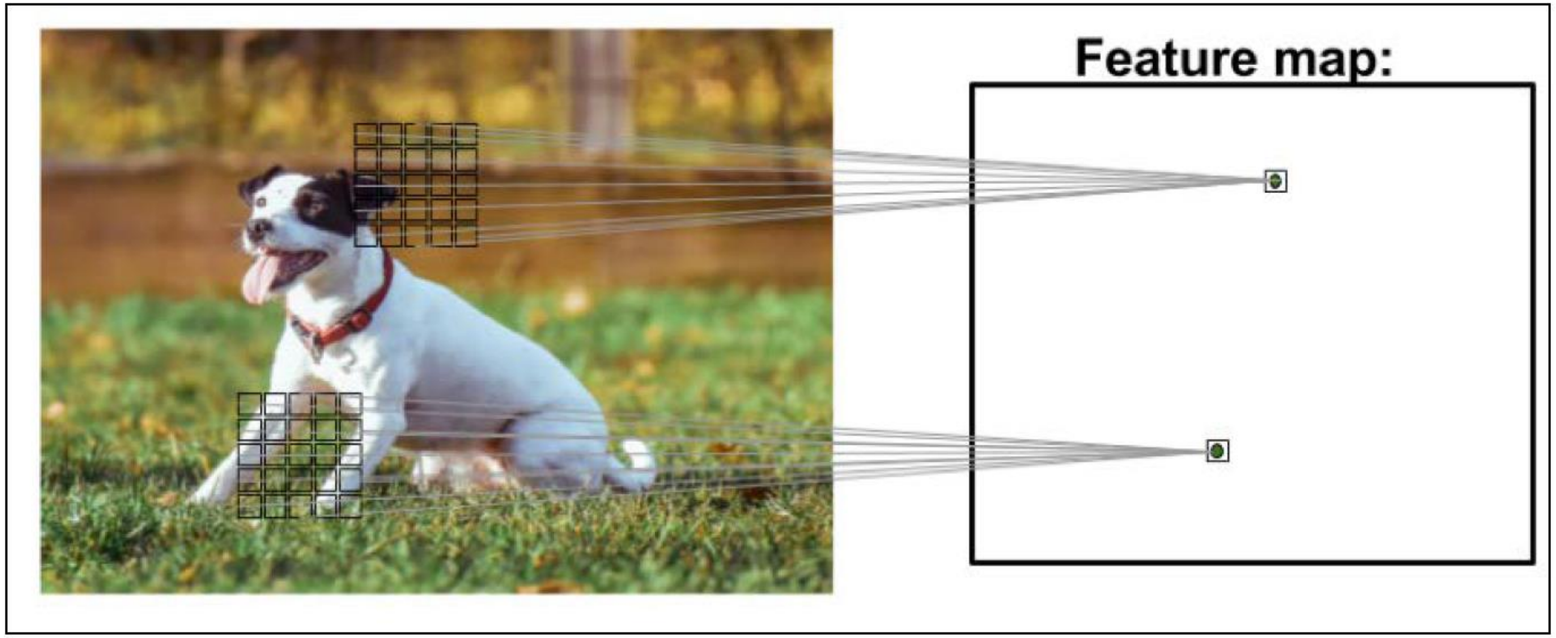
TEXT BOOK:

Book Type	Author & Title	Edition	Publisher	Year
Textbook 1	“Hands on Machine learning with scikit learn and scientific Python Toolkits”, Tarek Amr	1st	Packt	2020
Textbook-2	“Python Machine Learning”, Sebastian Raschka, Vahid Mirjalli	3rd	Packt	2019

Convolution Neural Networks Basics- The building blocks of CNNs

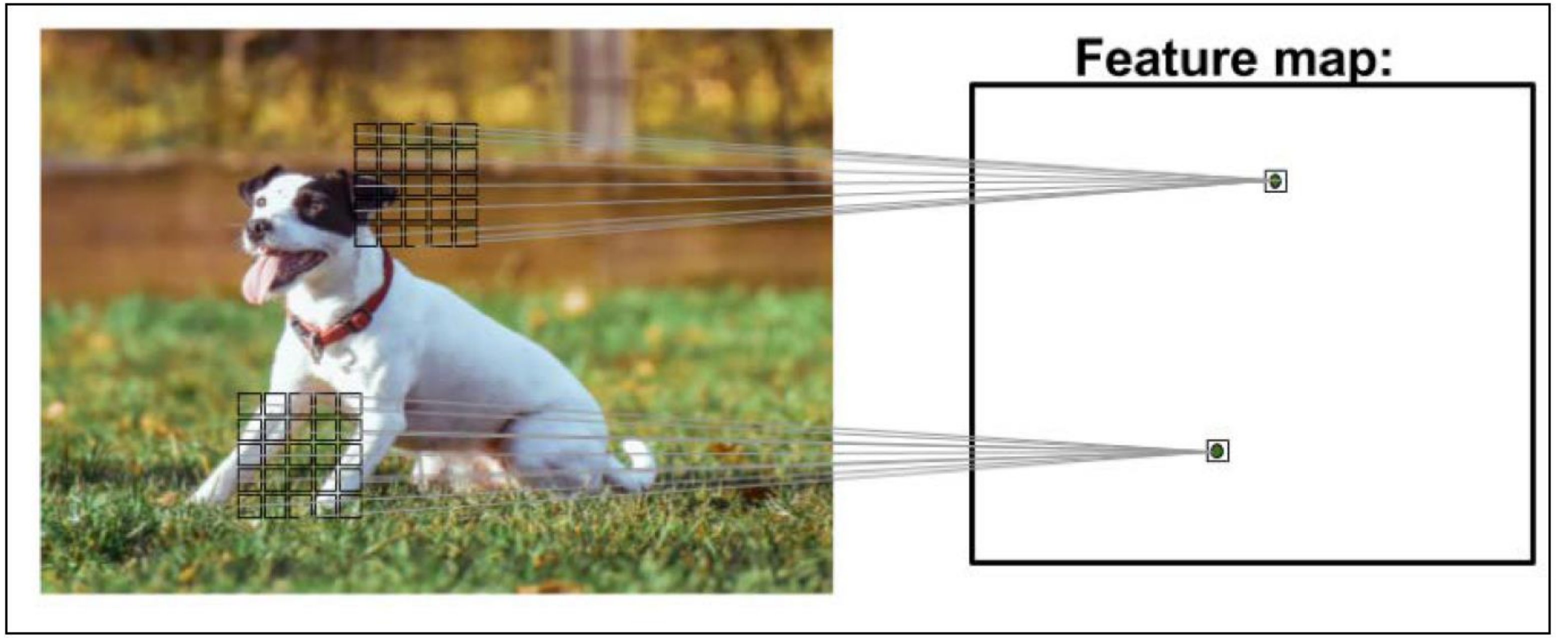
- CNNs are developed in 1990
- Outstanding Performance in Image Classification
- It led to lot of development in Computer Vision and Machine Learning
- Why Convolution layers are treated as feature extraction layers?

Convolution Neural Networks Basics- The building blocks of CNNs



- CNN's are able to learn the features automatically from the raw data which is very useful for a particular task
- CNN layers are extract low-level features from the raw data. Then later layers use these features to predict the output
- Feature Hierarchy: Feature hierarchy formed by combining the low level features in a layer-wise fashion to form high level features
- For example edges and blobs in the image are low level features , when we combine these features, it may lead to high level features

Convolution Neural Networks Basics- The building blocks of CNNs



- **Feature maps: CNN computes feature maps from an input image, where each element comes from a local patch of pixels in the input image**
- **The local patch of pixels is referred to as the local receptive field.**

Convolution Neural Networks Basics- The building blocks of CNNs

- CNNs will usually perform well on image-related tasks and it is due to following two reasons
- Sparse Connectivity : A single element in the feature map is connected to a small patch of pixels.
- Parameter-sharing: The same weights are used for different patches of the input image.

Convolution Neural Networks Basics- The building blocks of CNNs

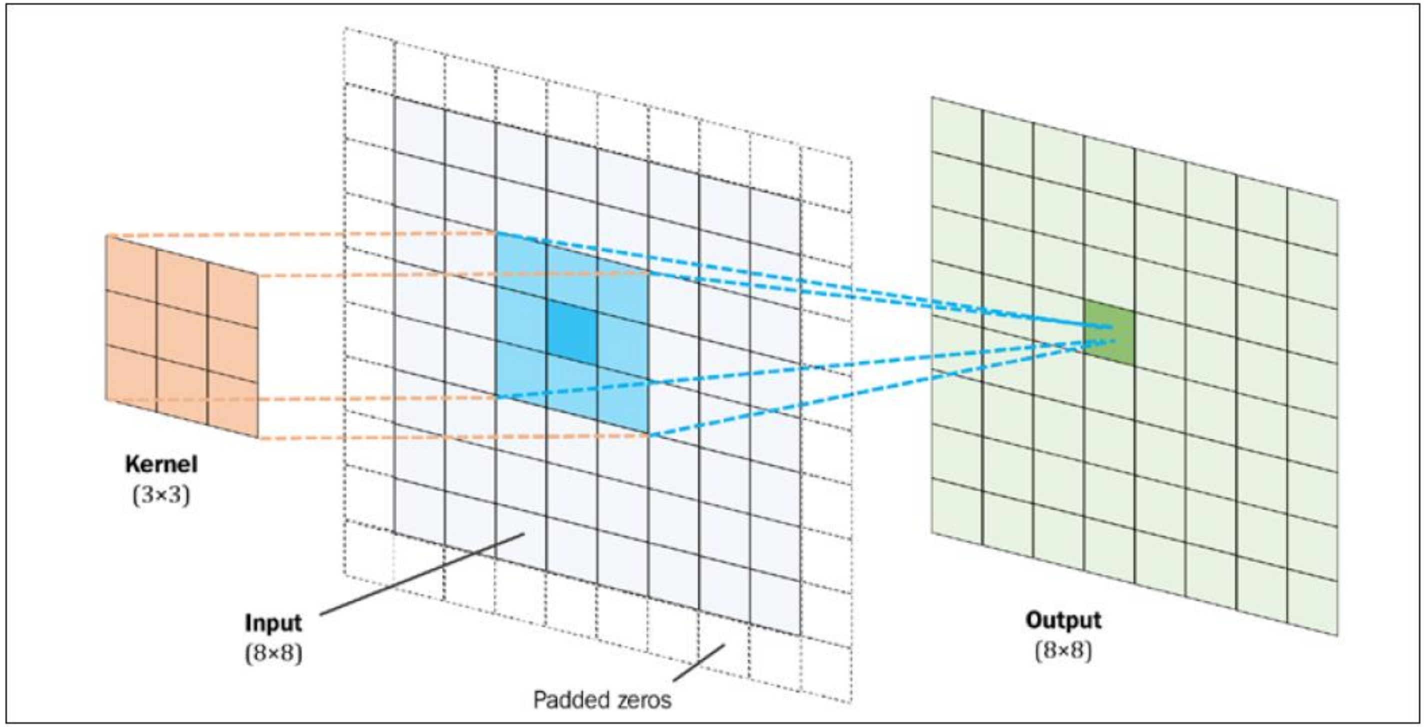
- Typically, CNN's consists of several Convolutional and subsampling layers that are followed by one or more Fully connected layers at the end.

- The Fully connected layers essentially an MLP , where input "i" is connected to every output j with weight W_{ij}

- Subsampling layers are commonly known as Pooling layers.

- Pooling layers do not have any learnable parameters, for instance there are no weights and no bias units.

- Both Convolutional and fully connected layers have weights and biases.



Kernel
(3×3)
Input
(8×8)
Padded zeros
Output
(8×8)

Convolution Neural Networks Basics- Performing discrete convolution

$$y = x * w \rightarrow y[i] = \sum_{k=-\infty}^{\infty} x[i-k]w[k]$$

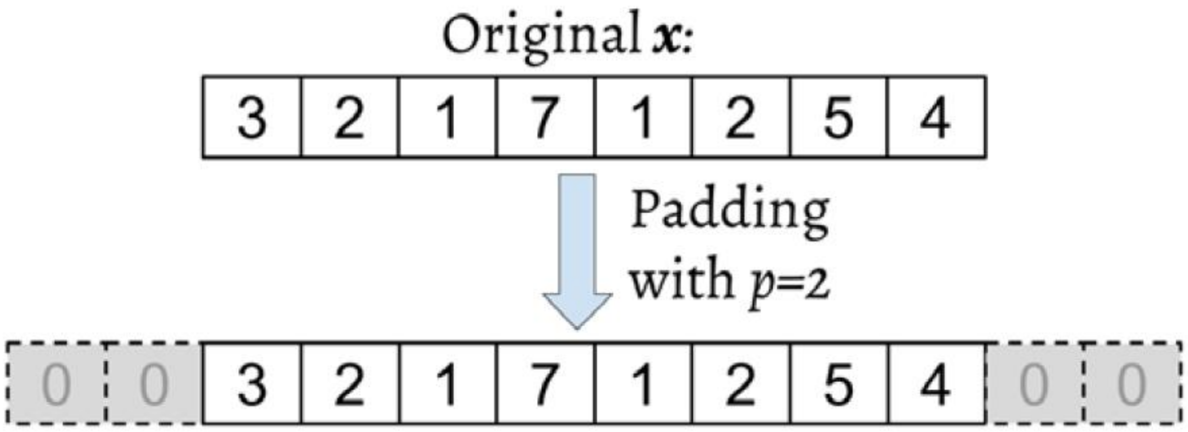
- Where x= input and w= filter or kernel
- Since the summation runs from -∞ to + ∞ , to make the summation to run from 0 to N-1 , the above formula is modified as follows

$$y = x * w \rightarrow y[i] = \sum_{k=0}^{k=m-1} x^p[i+m-k]w[k]$$

- x = original input of n-elements, w= filter of m-elements
- x^p= Padded vector of size n + 2p

Convolution Neural Networks Basics- Performing discrete convolution

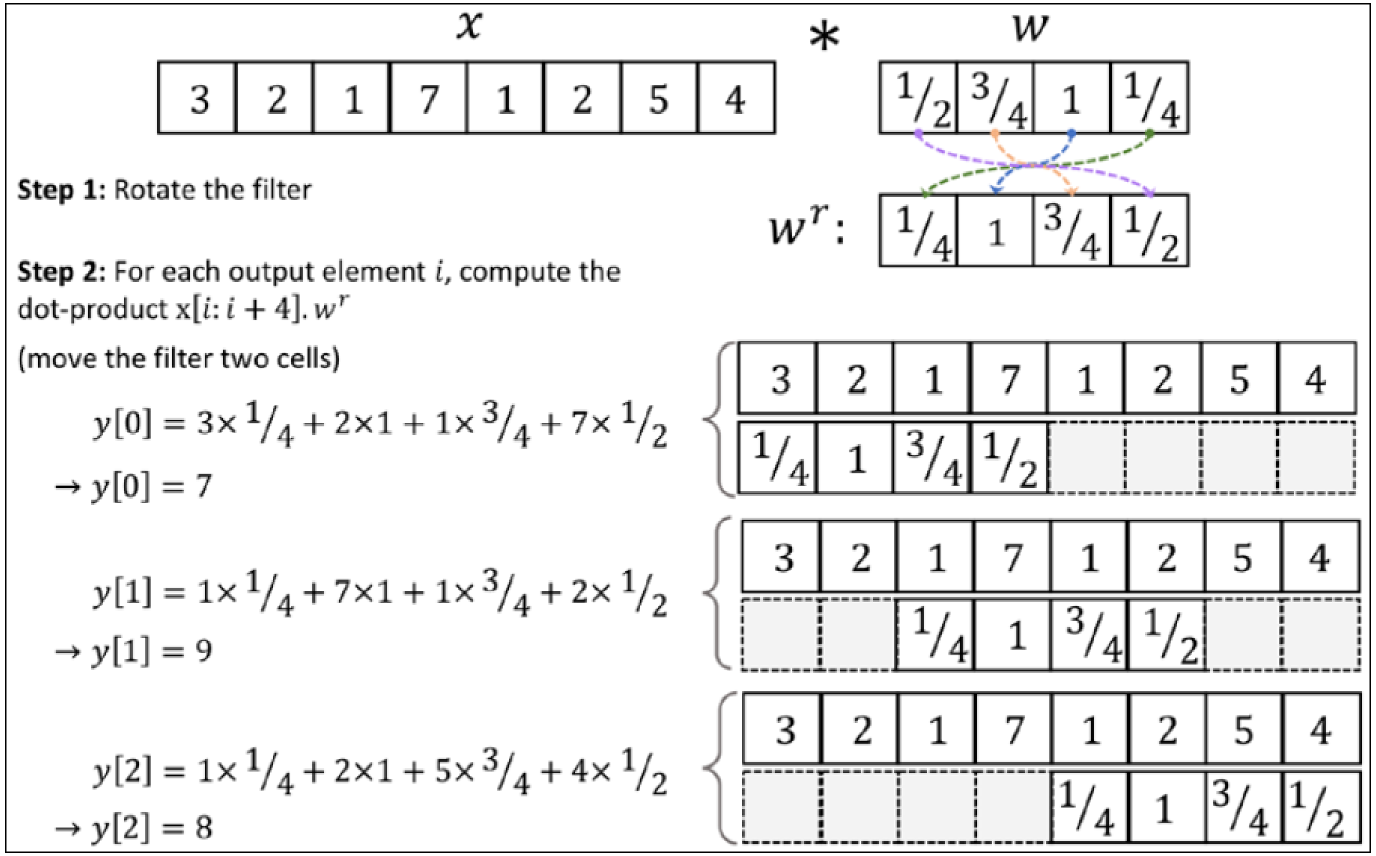
- x = original input of n-elements, w= filter of m-elements
- x^p= Padded vector of size $n + 2p$



Convolution Neural Networks Basics- Performing discrete convolution

$\boldsymbol{x} = [3\ 2\ 1\ 7\ 1\ 2\ 5\ 4]$ $\quad\quad$ $\boldsymbol{w} = \left[\frac{1}{2}\ \frac{3}{4}\ 1\ \frac{1}{4}\right]$

- x^p= Padded vector of size n + 2p with p= 0 x^p =[3 2 1 7 1 2 5 4]
- The Shift "s"is a Hyperparameter of convolution here s = 2 and it is known as stride. Stride is a positive number it should be smaller than the size of input vector.



Convolution Neural Networks Basics- Determining the size of the convolution output

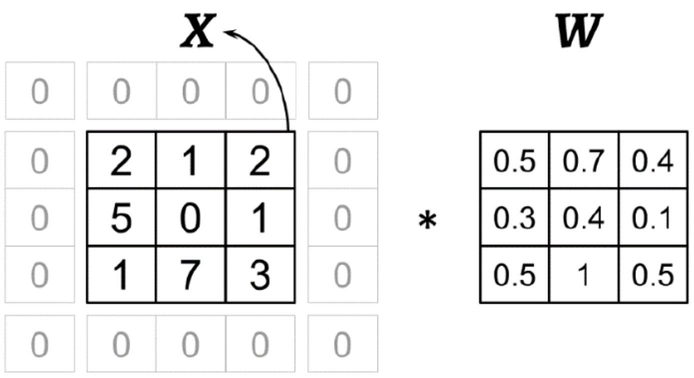
- $y = x*w$
- The output size will be
- $0 = \left\lfloor \frac{n+2p-m}{s} \right\rfloor + 1$ where $\lfloor \quad \rfloor = Floor\ operation, for\ Ex. \lfloor 1.77 \rfloor = 1$
- n= Input vector size
- m= size of the filter
- p=padding
- s=stride
- Ex-1 $0 = \left\lfloor \frac{10+2\times2-5}{1} \right\rfloor + 1 = 10$
- Ex-2 $0 = \left\lfloor \frac{10+2\times2-3}{2} \right\rfloor + 1 = 6$

Convolution Neural Networks Basics- Performing discrete convolution

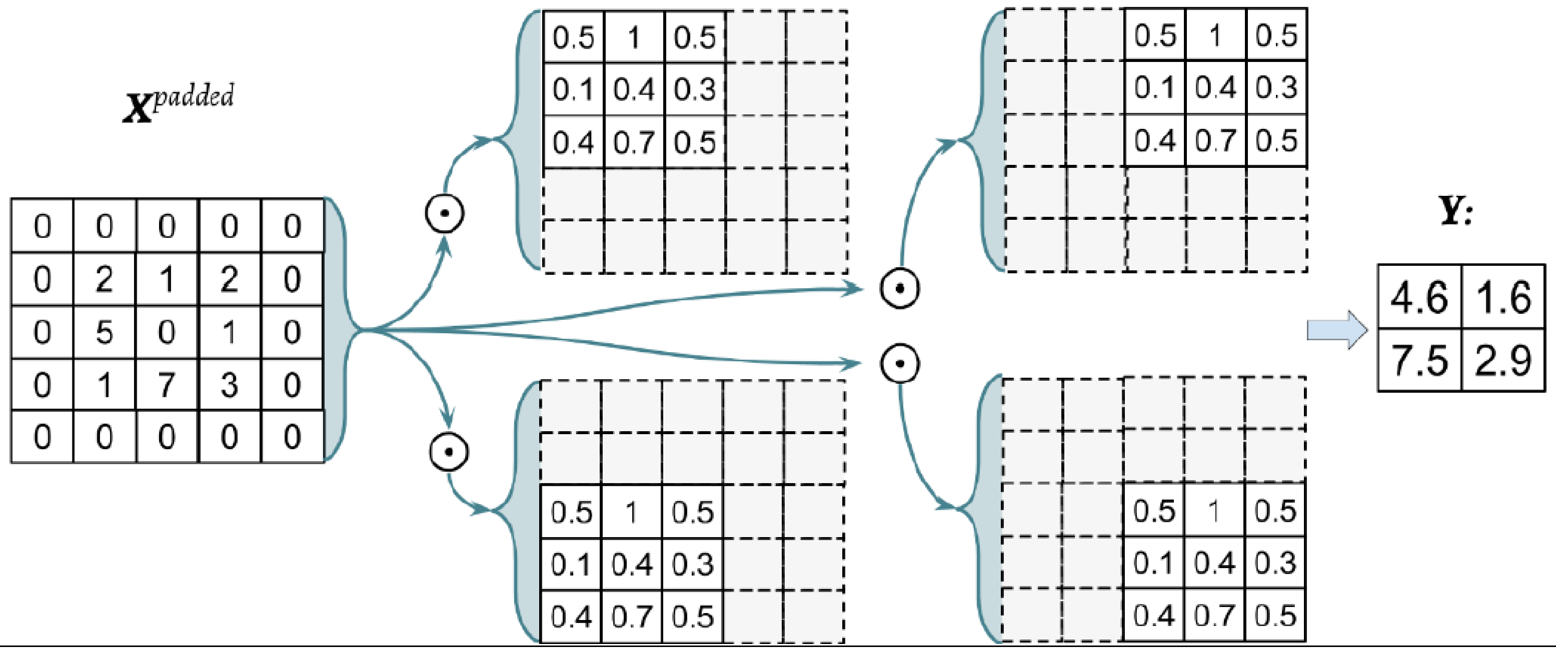
$$\boldsymbol{Y} = \boldsymbol{X} * \boldsymbol{W} \rightarrow \quad Y[i,j] = \sum_{k_1=-\infty}^{+\infty} \sum_{k_2=-\infty}^{+\infty} X[i-k_1, j-k_2]\, W[k_1, k_2]$$

- **$X_{n1 \times n2}$ = Input matrix**
- **$W_{m1 \times m2}$ = Filter matrix**
- **m1<= n1 and m2 <= n2**

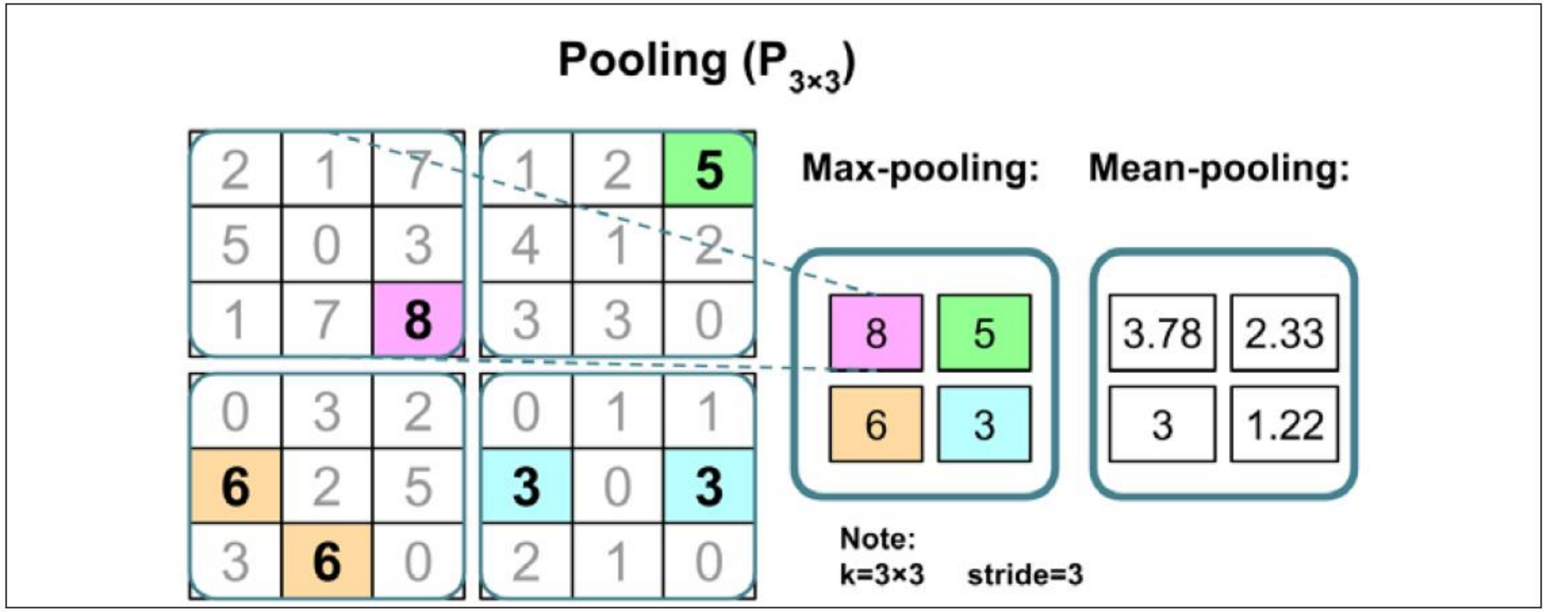
Convolution Neural Networks Basics- Performing discrete convolution



$$W^r = \begin{bmatrix} 0.5 & 1 & 0.5 \\ 0.1 & 0.4 & 0.3 \\ 0.4 & 0.7 & 0.5 \end{bmatrix}$$



Convolution Neural Networks Basics- Subsampling Layers



- Sampling is typically applied in two forms namely Max-pooling and Mean-pooling(Average Pooling)
- Pooling layer is usually denoted by $P_{n1\times n2}$
- Here n1×n2 refer to size of the neighborhood . We refer to such a neighborhood as Pooling size

Convolution Neural Networks Basics- Performing discrete convolution

$$
\left.\begin{array}{l}
X_1 = \begin{bmatrix} 10 & 255 & 125 & 0 & 170 & 100 \\ 70 & 255 & 105 & 25 & 25 & 70 \\ 255 & 0 & 150 & 0 & 10 & 10 \\ 0 & 255 & 10 & 10 & 150 & 20 \\ 70 & 15 & 200 & 100 & 95 & 0 \\ 35 & 25 & 100 & 20 & 0 & 60 \end{bmatrix} \\[2em]
X_2 = \begin{bmatrix} 100 & 100 & 100 & 50 & 100 & 50 \\ 95 & 255 & 100 & 125 & 125 & 170 \\ 80 & 40 & 10 & 10 & 125 & 150 \\ 255 & 30 & 150 & 20 & 120 & 125 \\ 30 & 30 & 150 & 100 & 70 & 70 \\ 70 & 30 & 100 & 200 & 70 & 95 \end{bmatrix}
\end{array}\right\} \xrightarrow{\text{max pooling } P_{2\times 2}} \begin{bmatrix} 255 & 125 & 170 \\ 255 & 150 & 150 \\ 70 & 200 & 95 \end{bmatrix}
$$

Putting everything together – implementing a CNN

- <u>Neural Network(NN)</u> : z = Wx + b , Where W= Weight Matrix b= bias x=input

For example, if the input is an image of size 224 x 224 then x is a column vector representing 224x224 elements it is denoted as $\mathbb{R}^{n\times 1}$ matrix

- <u>Convolutional Neural Network (CNN)</u>: Z = W*X + b , Where W= Weight Matrix b= bias X = Input Matrix

For example, if the input is an image of size 224 x 224 then X is a matrix representing the pixels in a height x width i.e 224 x 224

- In both cases, the pre-activations are passed to an activation function to obtain the activation of a hidden unit A = ϕ (Z) where ϕ is an Activation Function. Further we have Pooling layers in CNN

Working with multiple input or color channels

- An input to a convolutional layer may contain one or more 2D arrays or matrices with dimensions N_1 x N_2 (for example, the image height and width in pixels)
- These N_1 x N_2 are called Channels.
- Conventional implementations of convolutional layers expect a rank-3 tensor representation as an input, for example a three-dimensional array $X_{N_1 \times N_2 \times C_{in}}$ where C_{in} *is the number of input channels*
- If the image is colored and uses the RGB color mode, then C_{in} = 3 (for the red, green, and blue color channels in RGB). However, if the image is in grayscale, then we have C_{in} = 1, because there is only one channel with the grayscale pixel intensity values

➢ **Now that you are familiar with the structure of input data, the next question is, how can we incorporate multiple input channels in the convolution operation that we discussed in the previous sections? The answer is very simple: we perform the convolution operation for each channel separately and then add the results together using the matrix summation.**

➢ **The convolution associated with each channel (c) has its own kernel matrix as *W[:,:,c]***

➢ ***The total pre-activation result is computed in the following formula***

Given an example $\boldsymbol{X}_{n_1 \times n_2 \times C_{in}}$, a kernel matrix $\boldsymbol{W}_{m_1 \times m_2 \times C_{in}}$, and bias value b

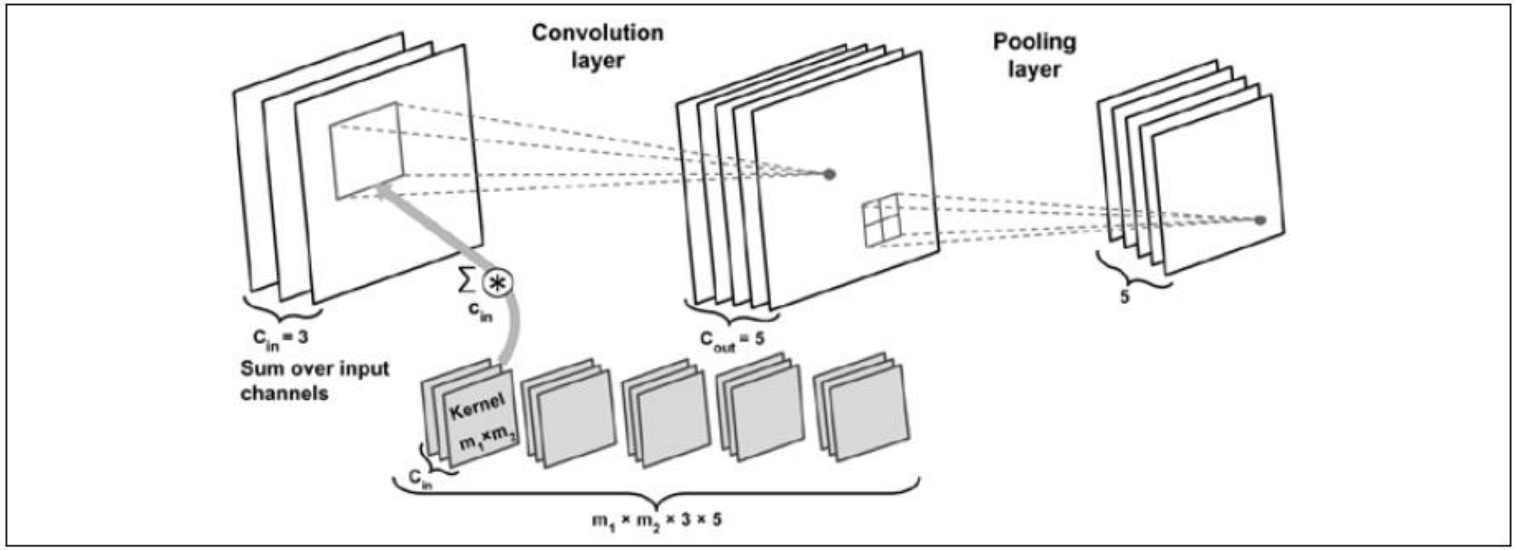
$$\Longrightarrow \begin{cases} \boldsymbol{Z}^{Conv} = \sum_{c=1}^{C_{in}} \boldsymbol{W}[:,:,c] * \boldsymbol{X}[:,:,c] \\ \text{Pre-activation:} \quad \boldsymbol{Z} = \boldsymbol{Z}^{Conv} + b_C \\ \text{Feature map:} \quad \boldsymbol{A} = \phi(\boldsymbol{Z}) \end{cases}$$

- The final result, A, is a **feature map**. Usually, a convolutional layer of a CNN has more than one feature map. If we use multiple feature maps, the **kernel tensor** becomes four-dimensional: $width \times height \times C_{in} \times C_{out}$
Here $width \times height$ *is the kernel size* C_{in} *is the number of input channels, and* C_{out} *is is the number of output feature maps*

- So, now let's include the number of output feature maps in the preceding formula and update it, as follows

Given an example $\boldsymbol{X}_{n_1 \times n_2 \times C_{in}}$,
a kernel matrix $\boldsymbol{W}_{m_1 \times m_2 \times C_{in} \times C_{out}}$,
and bias vector $\boldsymbol{b}_{C_{out}}$

$$\Longrightarrow \begin{cases} \boldsymbol{Z}^{Conv}[:,:,k] = \sum_{c=1}^{C_{in}} \boldsymbol{W}[:,:,c,k] * \boldsymbol{X}[:,:,c] \\ \boldsymbol{Z}[:,:,k] = \boldsymbol{Z}^{Conv}[:,:,k] + b[k] \\ \boldsymbol{A}[:,:,k] = \phi(\boldsymbol{Z}[:,:,k]) \end{cases}$$



- *In this example, there are three input channels. The kernel tensor is four-dimensional.*
- *Each kernel matrix is denoted as* $m_1 \times m_2$ and there are three of them, one for each input channel. Furthermore, there are five such kernels, accounting for five output feature maps. Finally, there is a pooling layer for subsampling the feature maps.
-

- **How many trainable parameters exist in this example?**

- **To illustrate the advantages of convolution, parameter sharing, and sparse connectivity, let's work through an example. The convolutional layer in the network shown in the preceding figure is a four-dimensional tensor. So, there are**

$$m_1 \times m_2 \times 3 \times 5$$

parameters associated with the kernel. Furthermore, there is a bias vector for each output feature map of the convolutional layer. Thus, the size of the bias vector is 5.
Pooling layers do not have any (trainable) parameters; therefore, it can be written as

$$m_1 \times m_2 \times 3 \times 5 + 5$$

- **If the input tensor is of size $n_1 \times n_2 \times 3$, assuming that the convolution is performed with the same-padding mode, then the output feature maps would be of size $n_1 \times n_2 \times 5$**

- Note that if we use a fully connected layer instead of a convolutional layer, this number will be much larger. In the case of a fully connected layer, the number of parameters for the weight matrix to reach the same number of output units would have been as follows:

$$(n_1 \times n_2 \times 3) \times (n_1 \times n_2 \times 5)$$

- In addition, the size of the bias vector is $(n_1 \times n_2 \times 5)$ (one bias element for each output unit). Given that $m_1 < n_1$ and $m_2 < n_2$ we can see that the difference in the number of trainable parameters is significant

Regularizing an NN with dropout

- **Over fitting** :The network will memorize the training data and do extremely well on the training dataset while achieving a poor performance on the held-out test dataset.

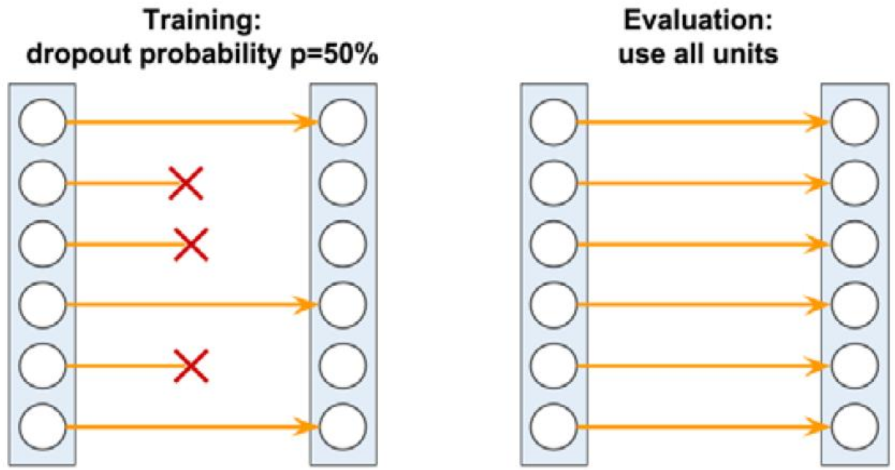
- Then, to prevent overfitting, we can apply one or multiple regularization schemes to achieve a good generalization performance on new data, such as the held-out test dataset

- **L1 and L2 regularization**, can prevent or reduce the effect of overfitting by adding a penalty to the loss that results in shrinking the weight parameters during training

- **L2** is the more common choice.

Dropout

➢ Dropout has emerged as a popular technique for regularizing (deep) NNs to avoid overfitting.



Loss functions for classification

Loss function	Usage
`BinaryCrossentropy`	Binary classification
`CategoricalCrossentropy`	Multiclass classification
`Sparse CategoricalCrossentropy`	Multiclass classification

Courtesy :"*Python Machine Learning*", Sebastian Raschka, Vahid Mirjalli, 3rd Edition, Packt 2019

THANK YOU

Raghavendra.M.J
Department of ECE
raghavendramj@pes.edu