DEEP LEARNING FOR SIGNAL & IMAGE PROCESSING

Dr. Mithun Kumar Kar

School of Artificial Intelligence

Amrita Vishwa Vidyapeetham Coimbatore

What Is Deep Learning?

- Deep learning is another name for a multilayer artificial neural network or multilayer perceptron.
- By adding more layers and more units within a layer, a deep network can represent functions of increasing complexity.
- Deep learning neural networks are able to automatically learn arbitrary complex mappings from inputs to outputs and support multiple inputs and outputs.

What Is Deep Learning?

- Deep Learning: a class of machine learning techniques, where many layers of information processing stages in hierarchical supervised architectures are exploited for unsupervised feature learning and for pattern analysis/classification.
- Deep learning is the name we often use for "deep neural networks" composed of several layers.

Definitions

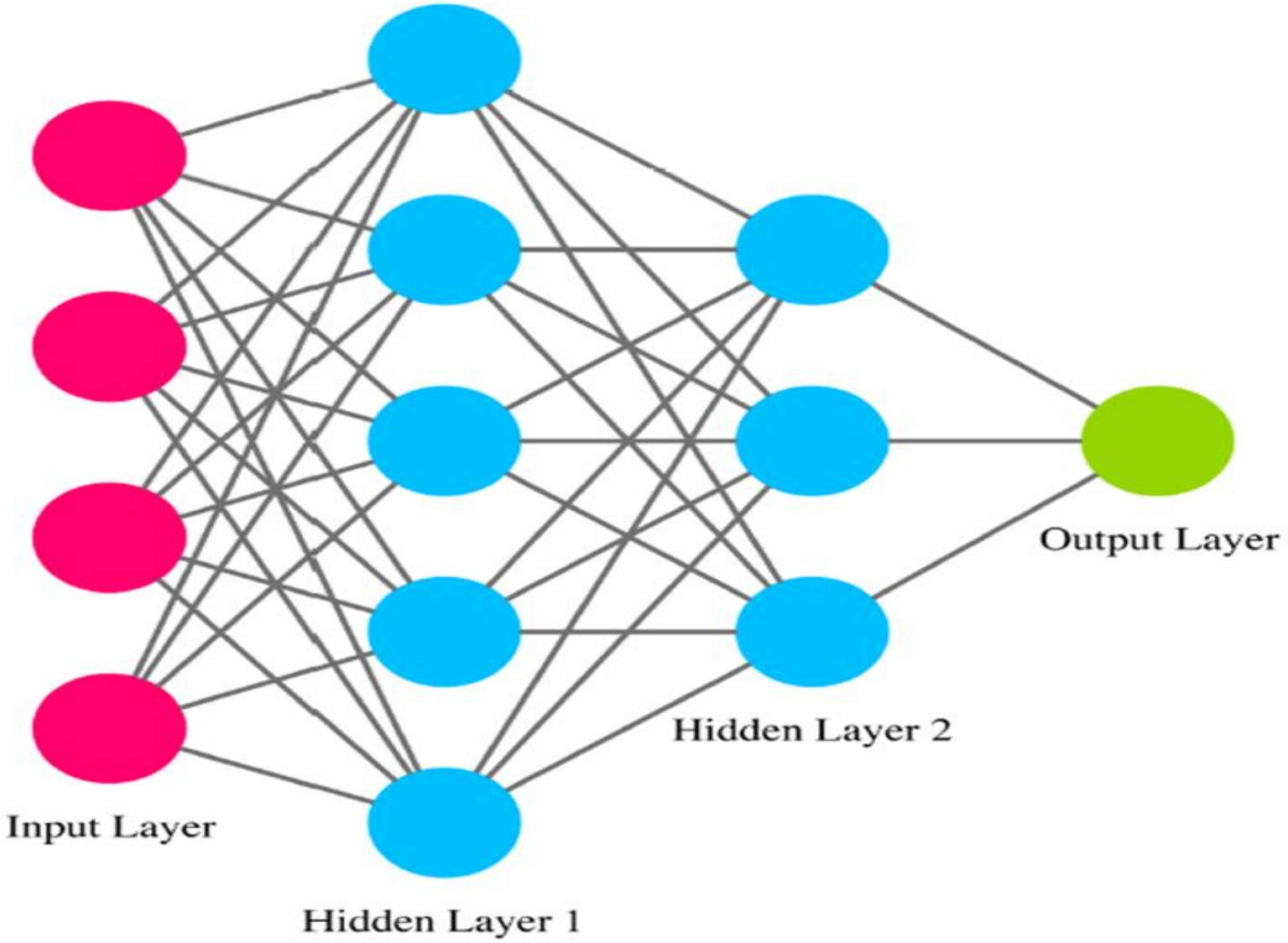
- Definition 1: A class of machine learning techniques that exploit many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification.

- Definition 2: A sub-field within machine learning that is based on algorithms for learning multiple levels of representation in order to model complex relationships among data.
- Higher-level features and concepts are thus defined in terms of lower-level ones, and such a hierarchy of features is called a deep architecture.

MLP

- A multilayer perceptron consists of at least three types of layers: input layer, hidden layers, and output layer.
- Input layer: The first layer of a neural network is called the input layer. This layer takes the input from the external source. The inputs to this layer are the features.
- Hidden layer: The layers of neurons between the input and output layers are called hidden layers.

MLP



MLP

- Output layer: The final layer of the neural network is the output layer. The output layer gets its input from the last hidden layer.
- For regression problems when the network has to predict a continuous value, such as the closing price of stocks, the output node has only one neuron.
- For classification problems when the network has to predict one of many classes, the output layer has as many neurons as the number of all possible classes.

Types of Deep Learning Network Models

- Convolutional Neural Networks (CNNs).
- Multilayer Perceptrons (MLP).
- Recurrent Neural Networks (RNNs).

Types of Computer Vision Problems

- Object detection
- Classification
- Regression
- Segmentation
- Prediction

Convolutional Neural Network (CNN)

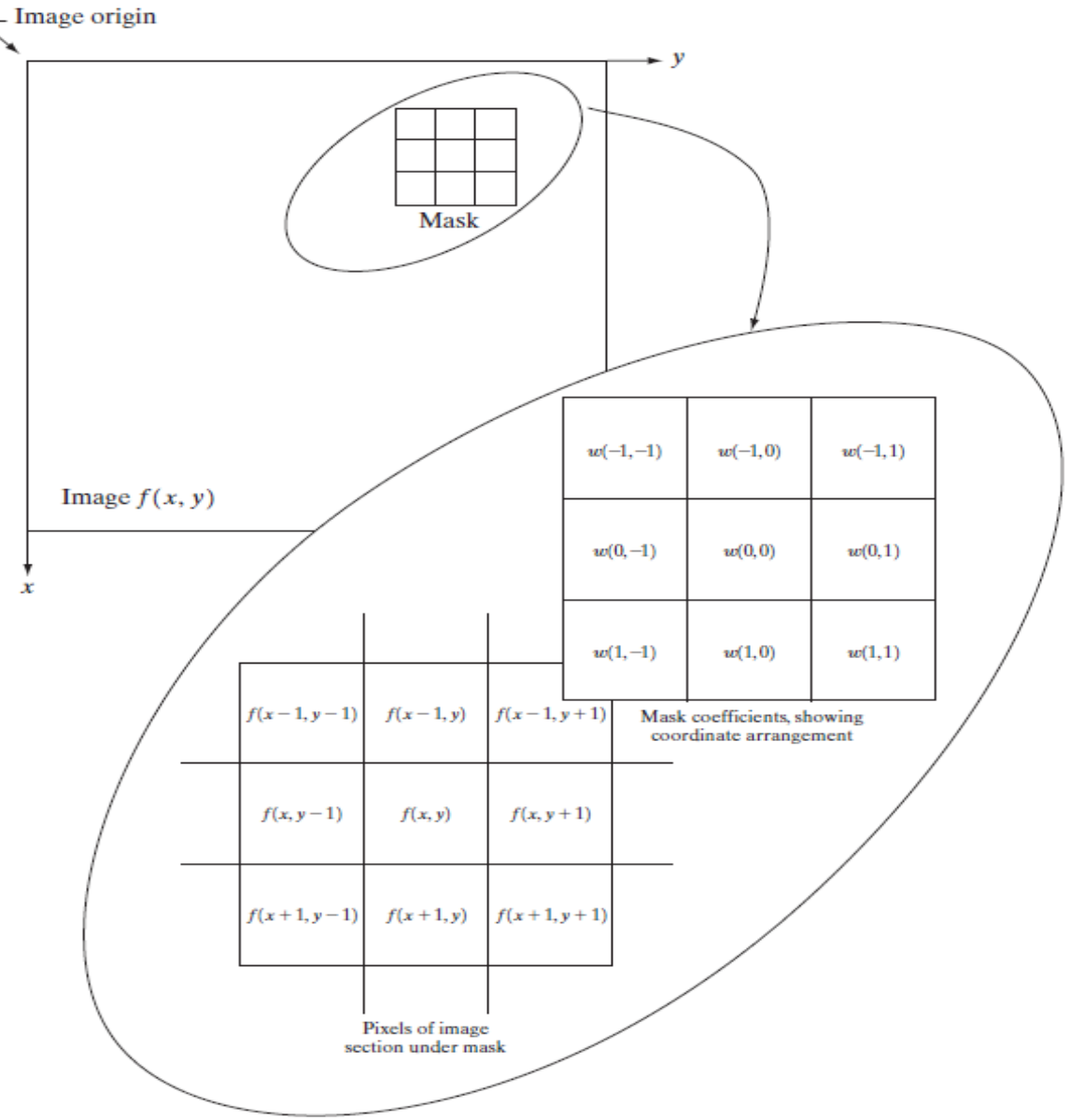
- Convolutional neural networks (CNN) are artificial neural network architectures consist of several convolutional layers that allow spatial convolution of input image with different filter kernels to compute various feature maps.
- In CNN, filters or kernels acting as weights slide across the total input image during convolution to create the next layer, and this new layer is called a feature map.
- The same filters can be used to repeat this operation to create new layers of feature maps

Convolutional Neural Network (CNN)

- The input and output feature maps have different dimensions depending on the dimension of image channels, i.e. 1 or 3 respectively for gray scale and color images.
- Multiple filters can be applied across a set of input image slices where each filter will generate a distinctive output slice.
- These slices highlight the features detected by the filters.
- At each layer, the input image is convolved with a set of kernels or masks with added biases to generate a new feature map.

Convolution in an Image

- **Spatial Filtering**.



Convolution in an Image

- In general, linear filtering of an image f of size M*N with a filter mask of size m*n is given by the expression:

$$g(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x + s, y + t)$$

where, $a = (m - 1)/2$ and $b = (n - 1)/2$. To generate a complete filtered image this equation must be applied for $x = 0, 1\ 2, \ldots, M - 1$ and $y = 0, 1, 2, \ldots, N - 1$.

Smoothing Spatial Filters

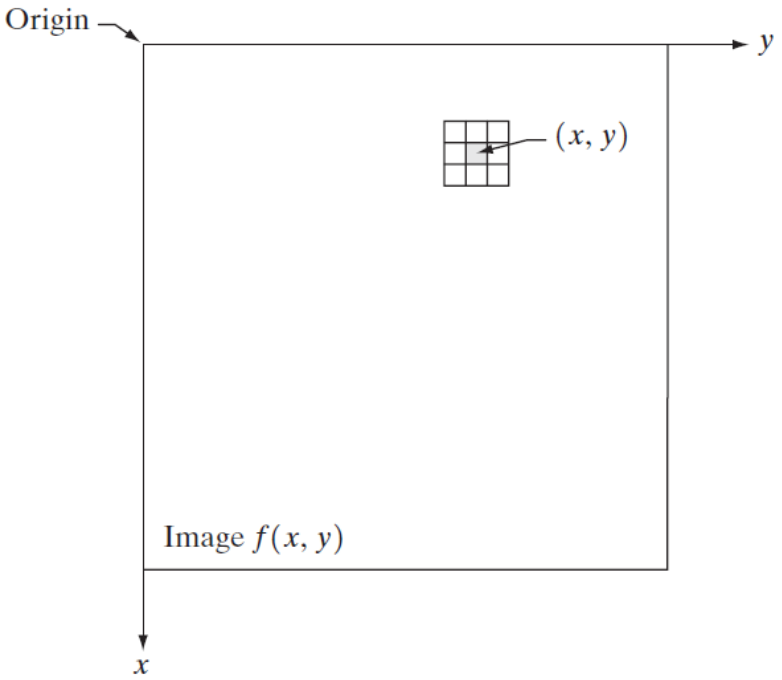
- Two examples of averaging filters.

$\frac{1}{9} \times$

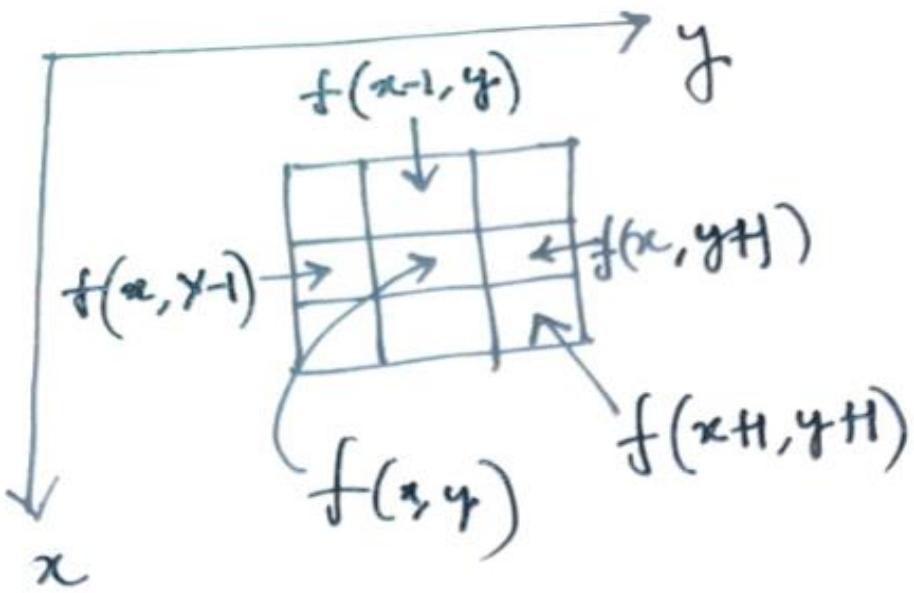
1	1	1
1	1	1
1	1	1

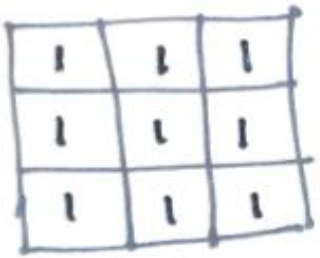
$\frac{1}{16} \times$

1	2	1
2	4	2
1	2	1



Smoothing Spatial Filters





$$\text{Averaging} = P(x,y) = f(x,y) + f(x-1,y-1) + f(x-1,y) + f(x-1,y+1) + \cdots + f(x+1,y+1)$$

$P = fW$ in matrix form.

Sharpening Spatial Filters

- The derivatives of a digital function are defined in terms of differences.
- A basic definition of the first-order derivative of a one-dimensional function f(x) is the difference

$$\frac{\partial f}{\partial x} = f(x+1) - f(x).$$

- Similarly, we define a second-order derivative as the difference

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x).$$

Sharpening Spatial Filters

- It can be shown that the simplest second derivative operator is the **Laplacian**, which, for a function (image) f(x, y) of two variables, is

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}.$$

- Now

$$\frac{\partial^2 f}{\partial^2 x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

$$\frac{\partial^2 f}{\partial^2 y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

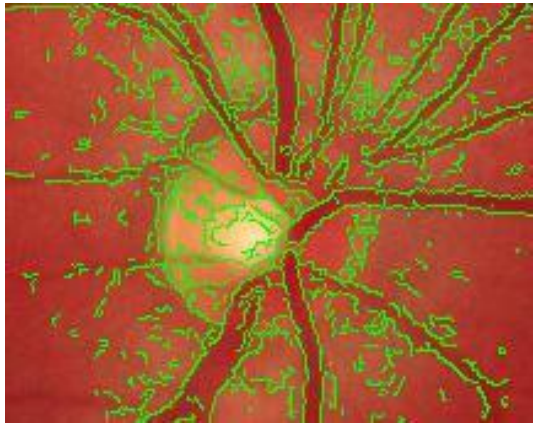
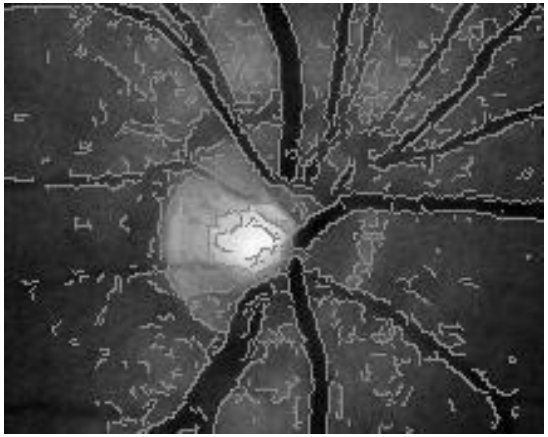
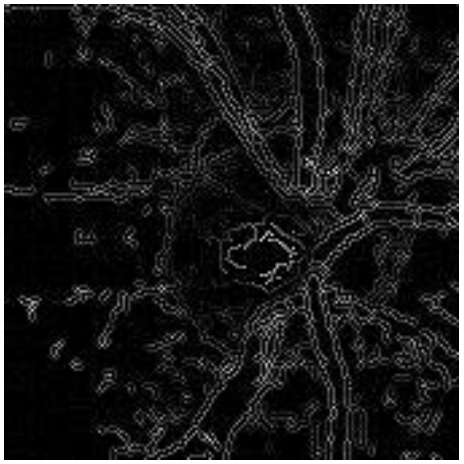
$$\nabla^2 f = [f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1)] - 4f(x, y).$$

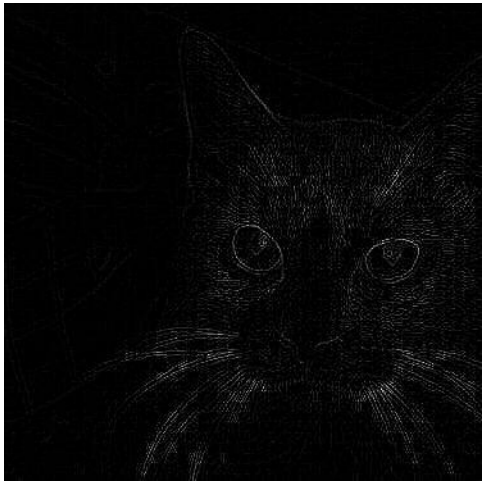
Sharpening Spatial Filters

- Now if we multiply this kernel with f(x,y) we got the same result. This equation can be implemented using the mask shown below.

0	1	0
1	−4	1
0	1	0

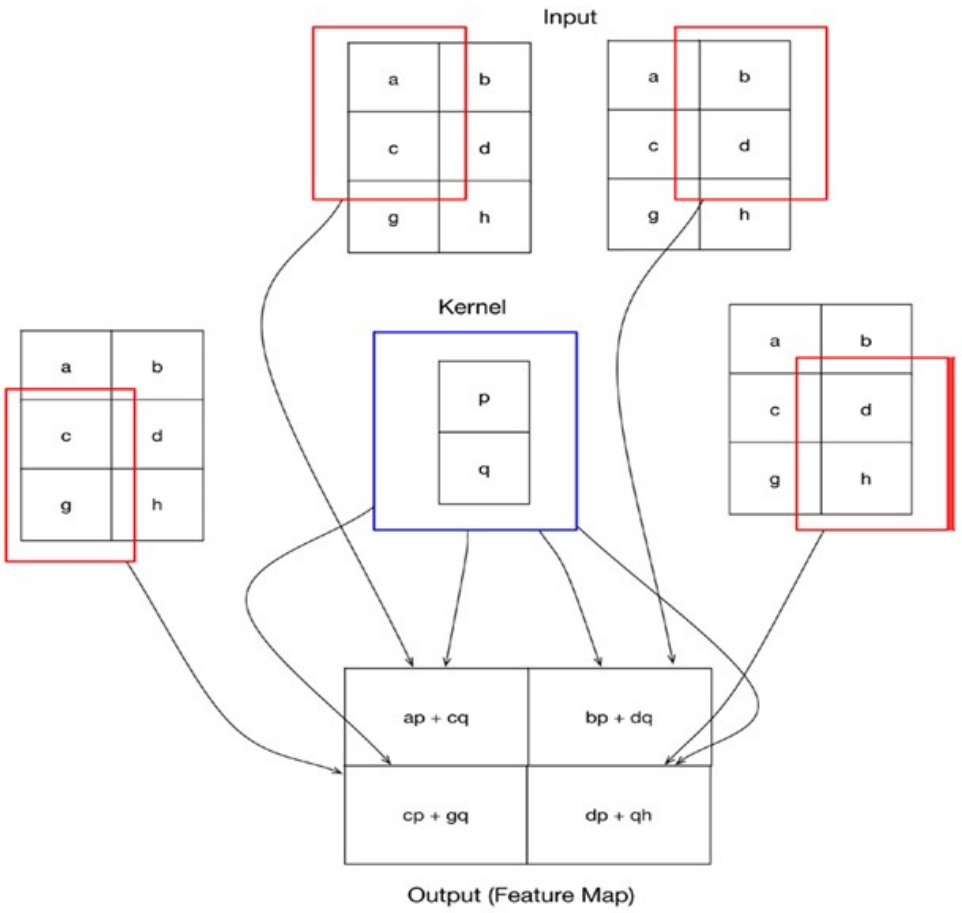
Implementation

CNN

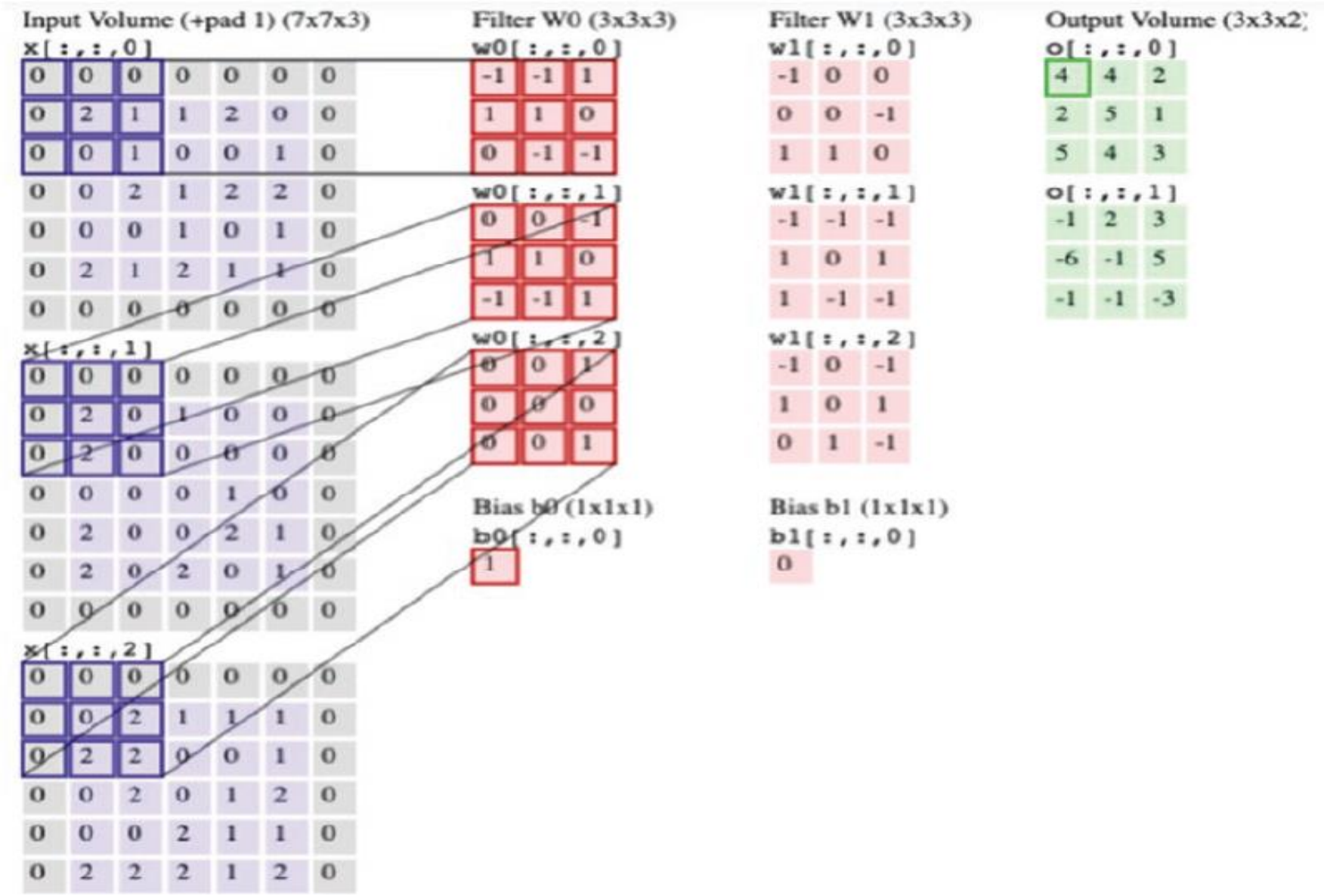
- The shape of the image after convolution is changed wrt the size of kernel used, shifting parameters and pooling.
- **General operations:**
- Convolution layer
- Pooling layer
- Batch normalization layer
- Fully connected layer



Components of Convolution Neural Networks

- Input layer
- Convolution layer
- Activation function for CNNs
- Pooling layer
- Batch normalization layer
- Fully connected layers

Convolutional Neural Network (CNN) Layer



Convolutional Neural Network (CNN) Layer

- **weight connections**: Each input feature to a neuron is multiplied by a weight.
- Think of weight as the contribution or significance of an input feature.
- The higher the weight, the more the contribution of the feature.
- The training objective of a neural network is to calculate the most optimized weights for each input feature for each connection to neurons of each layer

Convolutional Neural Network (CNN) Layer

- In a CNN, only the size of the filters is specified; the weights are initialized to arbitrary values before the start of training. The weights of the filters are learned through the CNN training process.
- Some of the terms with which one should be familiar while defining the convolution layer
- **Filter size**
- **Stride:** The stride determines the number of pixels to move in each spatial direction while performing convolution.

Convolutional Neural Network (CNN) Layer

- **Padding:** Padding is an approach that appends zeroes to the boundary of an image to control the size of the output of convolution.
- The convolved output image length L′ along a specific spatial dimension is given by

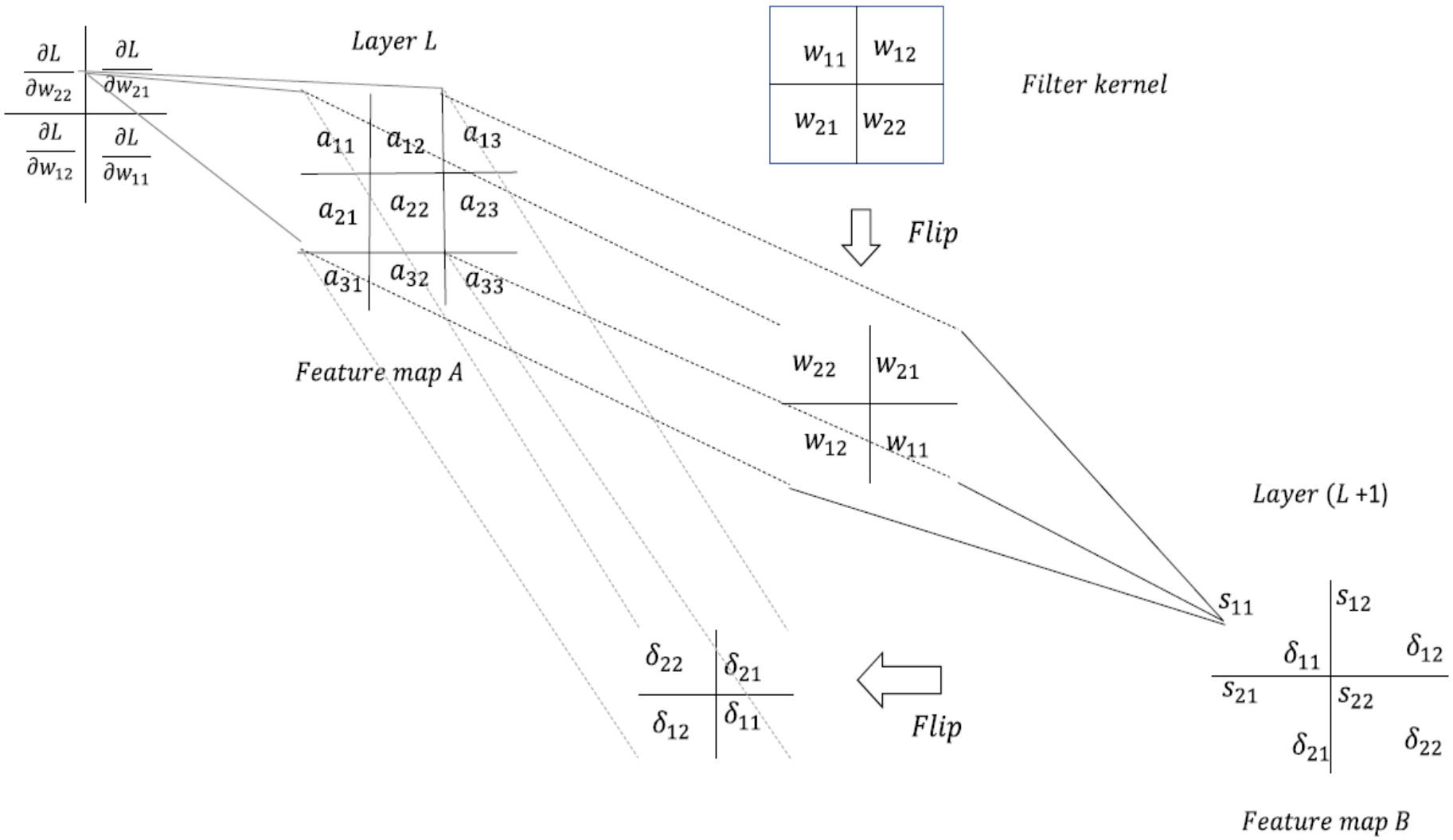
$$L' = \frac{L - K + 2P}{S} + 1$$

- Where L is the length of the input image in a specific dimension.
- K is the length of the kernel/filter in a specific dimension.

Convolutional Neural Network (CNN) Layer

- P-- Zeroes padded along a dimension in either end
- S-- Stride of the convolution
- **In pytorch this is done by**
- CLASS **torch.nn.Conv2d**(*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None*)

Backpropagation Through the Convolutional Layer



Backpropagation Through the Convolutional Layer

- In generalized way

$$s_{ij} = \sum_{n=1}^{2}\sum_{m=1}^{2} w_{(3-m)(3-n)} * a_{(i-1+m)(j-1+n)}$$

- Now, let the gradient of the cost function *L* with respect to the net input *s*ij be denoted by

$$\frac{\partial L}{\partial s_{ij}} = \delta_{ij}$$

- Let's compute the gradient of the cost function with respect to the weight w22

Backpropagation through the Convolutional Layer

- .

$$\frac{\partial \mathrm{L}}{\partial w_{22}} = \sum_{j=1}^{2}\sum_{i=1}^{2} \frac{\partial \mathrm{L}}{\partial s_{ij}} \frac{\partial s_{ij}}{\partial w_{22}} = \sum_{j=1}^{2}\sum_{i=1}^{2} \delta_{ij} \frac{\partial s_{ij}}{\partial w_{22}}$$

$$\frac{\partial s_{11}}{\partial w_{22}} = a_{11},\ \frac{\partial s_{12}}{\partial w_{22}} = a_{12},\ \frac{\partial s_{13}}{\partial w_{22}} = a_{21},\ \frac{\partial s_{14}}{\partial w_{22}} = a_{22}$$

$$\frac{\partial \mathrm{L}}{\partial w_{22}} = \delta_{11} * a_{11} + \delta_{12} * a_{12} + \delta_{21} * a_{21} + \delta_{22} * a_{22}$$

$$\frac{\partial \mathrm{L}}{\partial w_{21}} = \delta_{11} * a_{12} + \delta_{12} * a_{13} + \delta_{21} * a_{22} + \delta_{22} * a_{23}$$

Backpropagation through the Convolutional Layer

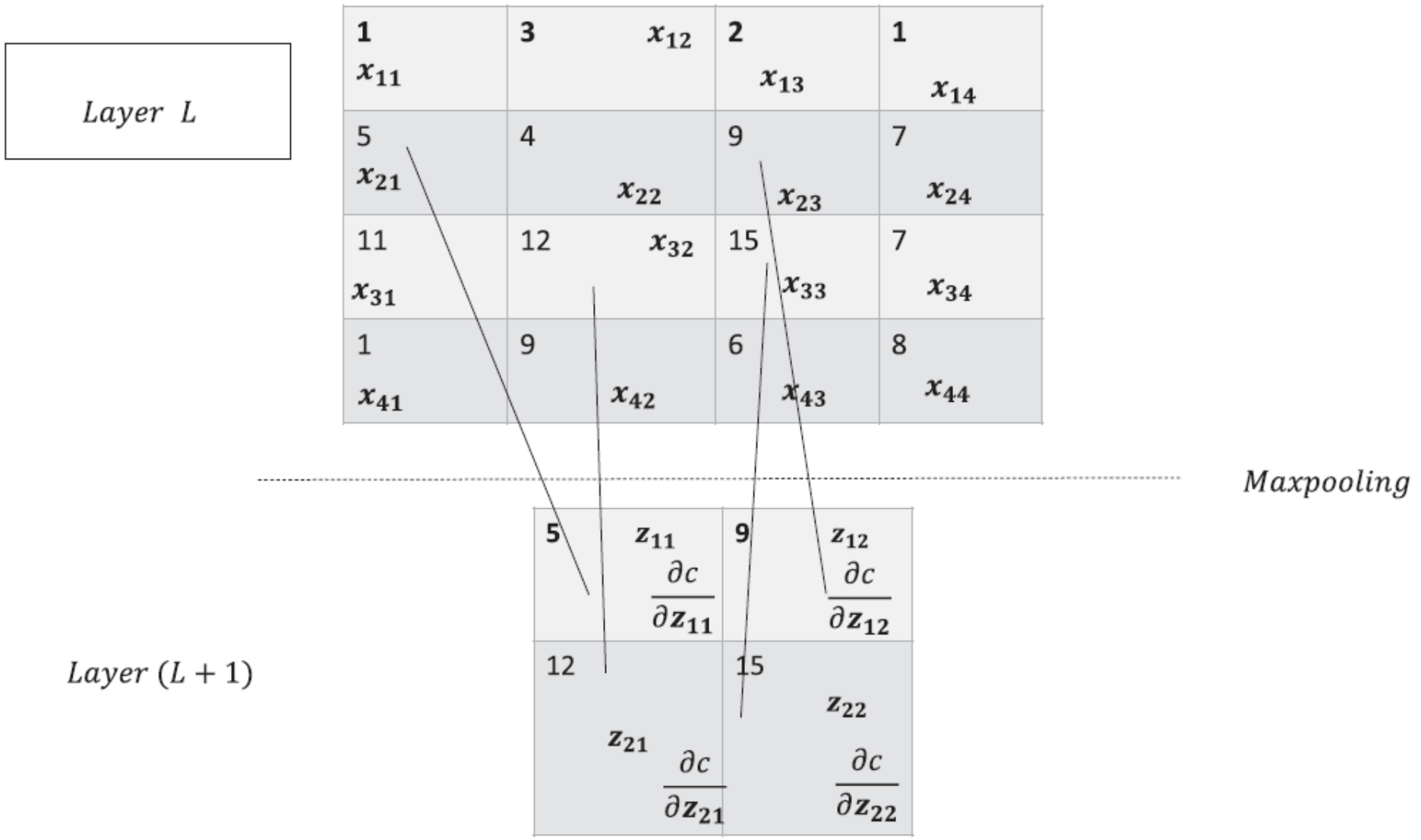
- Generalizing we get

$$\frac{\partial \mathrm{L}}{\partial w_{ij}} = \sum_{n=1}^{2}\sum_{m=1}^{2} \delta_{mn} * a_{(i-1+m)(j-1+n)}$$

$$\begin{bmatrix} \frac{\partial \mathrm{L}}{\partial w_{22}} & \frac{\partial \mathrm{L}}{\partial w_{21}} \\ \frac{\partial \mathrm{L}}{\partial w_{12}} & \frac{\partial \mathrm{L}}{\partial w_{11}} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} (\mathrm{x}) \begin{bmatrix} \delta_{11} & \delta_{12} \\ \delta_{21} & \delta_{22} \end{bmatrix}$$

- In terms of the layers, one can say the flip of the gradient matrix turns out to be a cross-correlation of the gradient at the (L +1) layer with the outputs of the feature map at layer L.

Backpropagation Through the Pooling Layers



Average pooling

Layer L

1 x_{11}	**3** x_{12}	**2** x_{13}	**1** x_{14}
5 x_{21}	4 x_{22}	9 x_{23}	7 x_{24}
11 x_{31}	12 x_{32}	15 x_{33}	7 x_{34}
1 x_{41}	9 x_{42}	6 x_{43}	8 x_{44}

Average *pooling*

Layer $(L+1)$

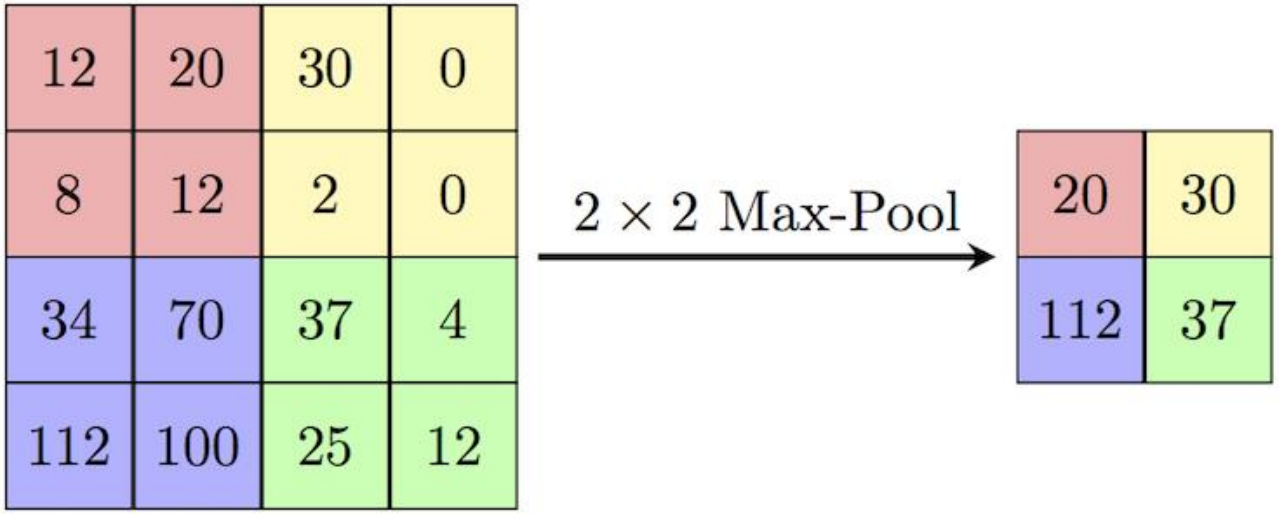
3.25 z_{11} $\frac{\partial c}{\partial z_{11}}$	**4.75** z_{12} $\frac{\partial c}{\partial z_{12}}$
8.25 z_{21} $\frac{\partial c}{\partial z_{21}}$	**9** z_{22} $\frac{\partial c}{\partial z_{22}}$

Activation function for CNNs

- Linear Activation Function
- Rectified Linear Unit
- Leaky ReLU
- Sigmoid
- Tanh
- Softmax

Pooling layer

- **Max Pooling** is a pooling operation that calculates the maximum value for patches of a feature map, and uses it to create a down sampled (pooled) feature map. It is usually used after a convolutional layer.



Batch normalization layer

- **Batch normalization** is used to reduce internal covariate shift and for faster convergence.
- The batch normalized activation is given

$$a_i = \frac{a_i - a_{mean}}{\sqrt{\sigma_b^2 + c}}$$

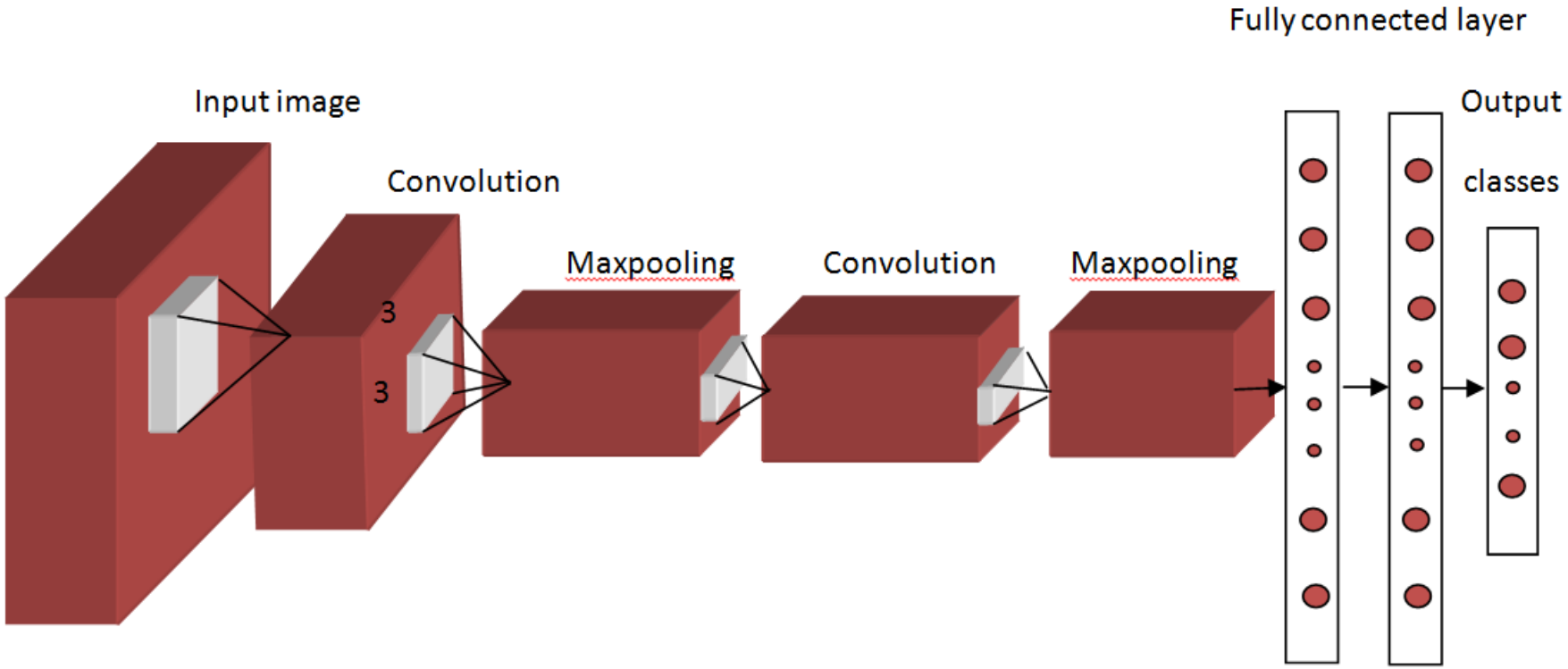
- where a_{mean} represents mini batch mean, σ_b^2 is the mini batch variance and c is a numeric constant used for numerical stability.

Fully connected layer

- **Fully connected layer**: Here each neuron in one layer is connected to other neurons in the subsequent layer.
- The fully connected layer integrates the various features extracted in the previous convolutional and pooling layers and maps them to specific classes or outcomes.
- Limiting the number of fully connected layers balances computational efficiency and generalization ability with the capability to learn complex patterns.

CNN

- CNN as classifier: CNN is used to classify the image data in to different classes.

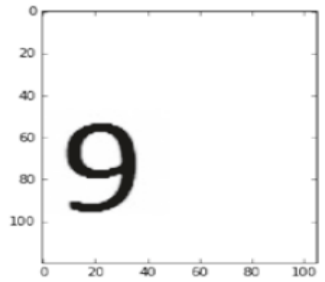


Weight Sharing Through Convolution and Its Advantages

- Weight sharing through convolution greatly reduces the number of parameters in the convolutional neural network.
- In cases of convolution, as in this scenario, we just need to learn the weights for the specific filter kernel. Since the filter size is relatively small with respect to the image, the number of weights is reduced significantly.
- The convolution operation provides translational equivariance.

Weight Sharing Through Convolution and Its Advantages

- if a feature A in an input produces a specific feature B in the output, then even if feature A is translated around in the image, feature B would continue to be generated at different locations of the output.



$*$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$=$

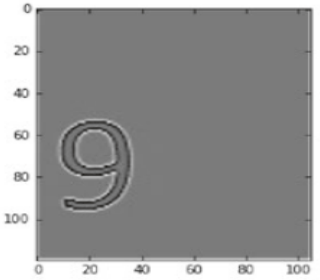
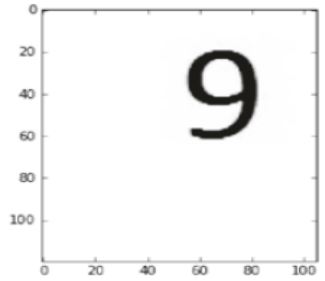


Image (A) with digit 9 *Filter kernel* *Convolved output image (C)*



$*$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$=$

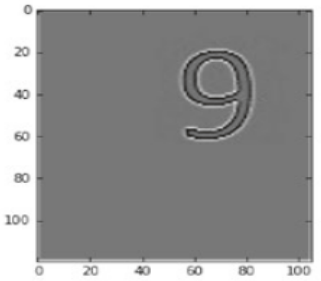


Image (B) with digit 9 translated *Filter kernel* *Convolved output image (D)*

Weight Sharing Through Convolution and Its Advantages

- Translation Invariance Due to Pooling:

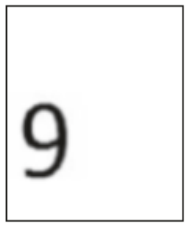


Image A

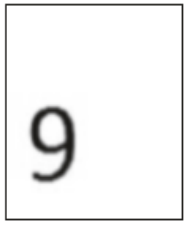


Image B

Convolution with same filter H

0	**0**	**0**	**0**
100	40	0	0
80	20	0	0
0	0	0	0

P

0	**0**	**0**	**0**
50	100	40	0
20	80	20	0
0	0	0	0

P′

Output Feature maps

Max Pooling with 2 × 2 receptive field

100	**0**
80	0

M

100	**40**
80	20

M′

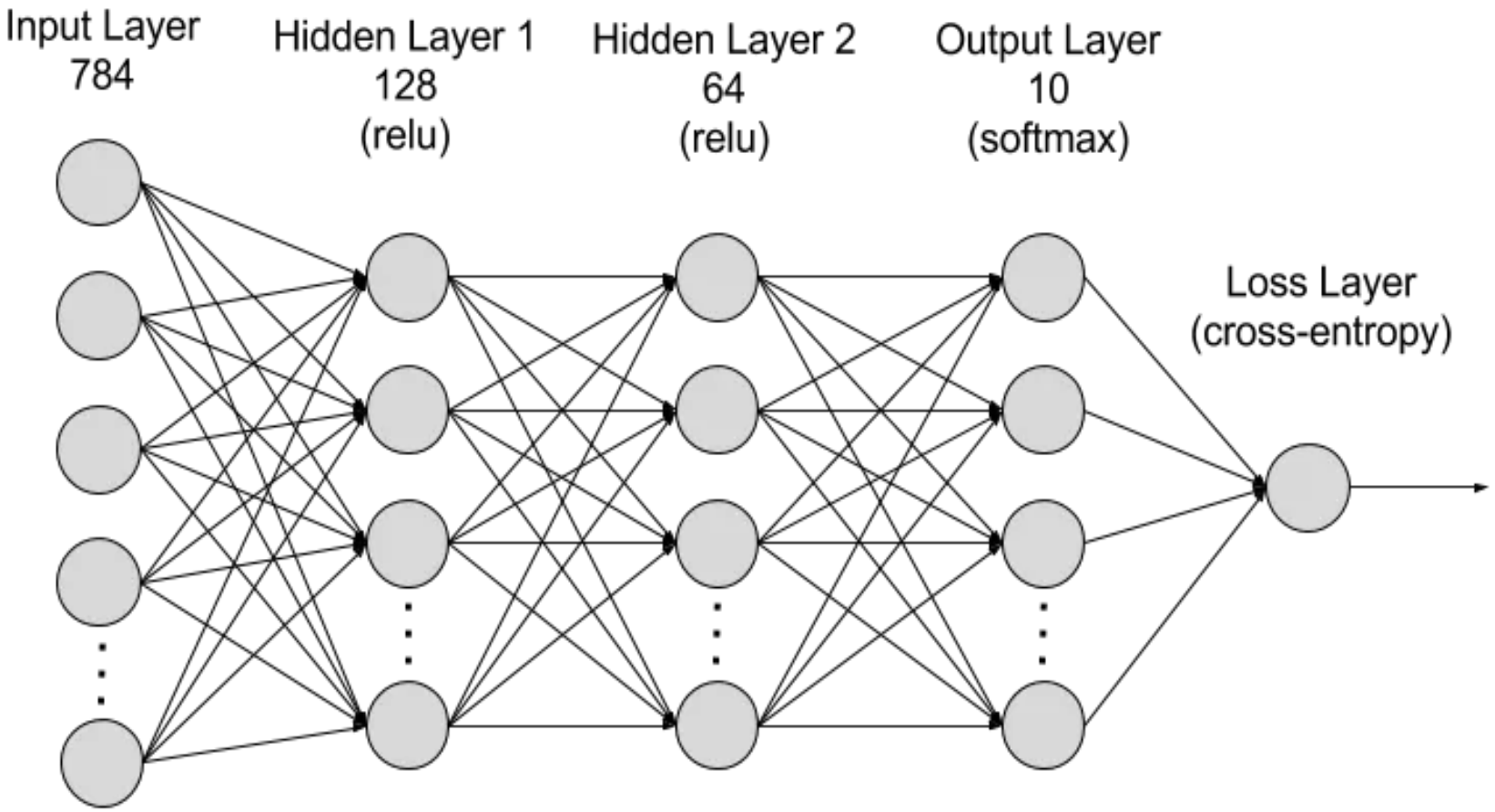
Output after Max Pooling

A classification problem using CNN in pytorch

- Load and normalize the train and test data.
- Define the Convolutional Neural Network (CNN).
- Define the loss function and optimizer.
- Train the model on the train data.
- Validate the model using validation data.
- Test the model on the test data

Digit classification(MNIST database)

Digit classification model



Digit classification

```
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,), (0.5,)),
                                ])
# defining the training and testing set
trainset = datasets.MNIST('./data', download=True, train=True, transform=transform)
testset = datasets.MNIST('./', download=True, train=False, transform=transform)

# defining trainloader and testloader
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=True)

# shape of training data
dataiter = iter(trainloader)
images, labels = dataiter.next()

print(images.shape)
print(labels.shape)
# visualizing the training images
plt.imshow(images[0].numpy().squeeze(), cmap='gray')
# shape of validation data
dataiter = iter(testloader)
images, labels = dataiter.next()
print(images.shape)
print(labels.shape)
```

Digit classification

```
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()

    self.cnn_layers = nn.Sequential (
      nn.Conv2d(1, 4, kernel_size=3, stride=1, padding=1),
      nn.BatchNorm2d(4),
      nn.ReLU(inplace=True),
      nn.MaxPool2d(kernel_size=2, stride=2),
      # Defining another 2D convolution layer
      nn.Conv2d(4, 4, kernel_size=3, stride=1, padding=1),
      nn.BatchNorm2d(4),
      nn.ReLU(inplace=True),
      nn.MaxPool2d(kernel_size=2, stride=2),
    )

    self.linear_layers = nn.Sequential(
      nn.Linear(4 * 7 * 7, 10)
    )

  # Defining the forward pass
  def forward(self, x):
    x = self.cnn_layers(x)
    x = x.view(x.size(0), -1)
    x = self.linear_layers(x)
    return x
# defining the model
model = Net()
```

Digit classification

```
optimizer = optim.Adam(model.parameters(), lr=0.01)
# defining the loss function
criterion = nn.CrossEntropyLoss()
# checking if GPU is available
if torch.cuda.is_available():
    model = model.cuda()
    criterion = criterion.cuda()

print(model)
for i in range(10):
    running_loss = 0
    for images, labels in trainloader:

        if torch.cuda.is_available():
          images = images.cuda()
          labels = labels.cuda()

        # Training pass
        optimizer.zero_grad()

        output = model(images)
        loss = criterion(output, labels)

        #This is where the model learns by backpropagating
        loss.backward()

        #And optimizes its weights here
        optimizer.step()

        running_loss += loss.item()
    else:
        print("Epoch {} - Training loss: {}".format(i+1, running_loss/len(trainloader)))
```

Digit classification

```
correct_count, all_count = 0, 0
for images,labels in testloader:
  for i in range(len(labels)):
    if torch.cuda.is_available():
       images = images.cuda()
       labels = labels.cuda()
    img = images[i].view(1, 1, 28, 28)
    with torch.no_grad():
       logps = model(img)


    ps = torch.exp(logps)
    probab = list(ps.cpu()[0])
    pred_label = probab.index(max(probab))
    true_label = labels.cpu()[i]
    if(true_label == pred_label):
      correct_count += 1
    all_count += 1

print("Number Of Images Tested =", all_count)
print("\nModel Accuracy =", (correct_count/all_count))
```