Introduction to Convolutional Neural Networks (CNNs)

Farhad Maleki

Agenda

- Introduction to Convolutional Neural Networks
- Convolution Operation
- Activation Functions
- Pooling Layers
- Batch Normalization
- Regularization Techniques
- Transfer Learning
- Advanced Architectures
- Real-world Applications

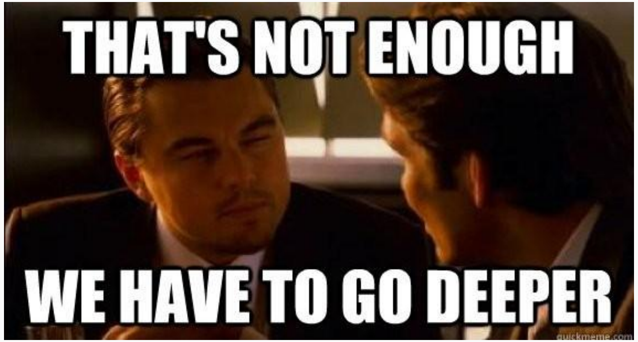
Introduction to Convolutional Neural Networks

- CNNs are designed to process grid-like data, such as images
- They exploit the **spatial structure** of images using **local connectivity** and **parameter sharing**
- CNNs are capable of **hierarchical feature learning**

Why not Using MLP for Images?

Why CNNs are Commonly Used in Computer Vision

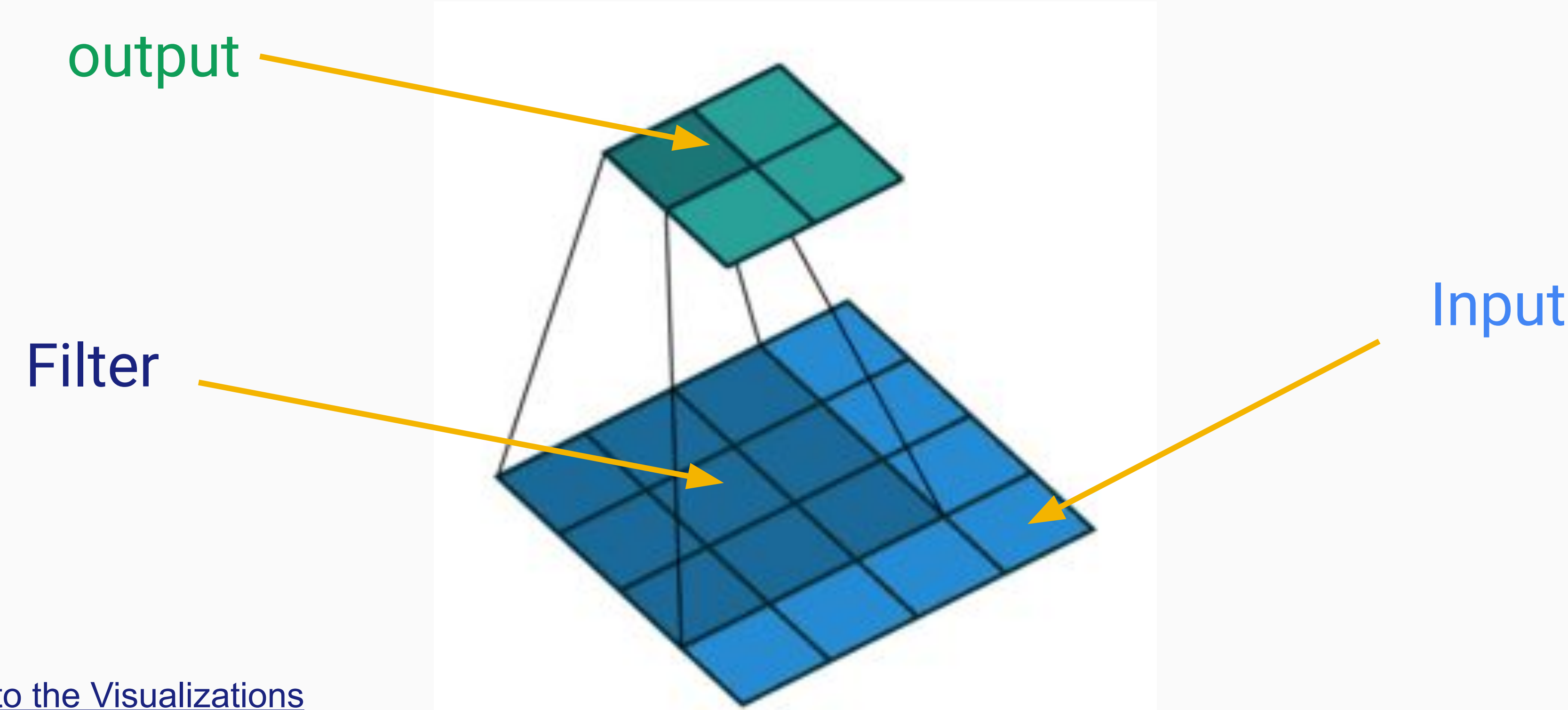
- Scalability: High dimension of visual data
- Local connectivity
- Parameter sharing
- Hierarchical feature learning
- Translation invariance



Convolution Operation

- Convolution involves sliding a filter over the input image
- Computes the element-wise product between the filter and the image patch
- Outputs a feature map or activation map

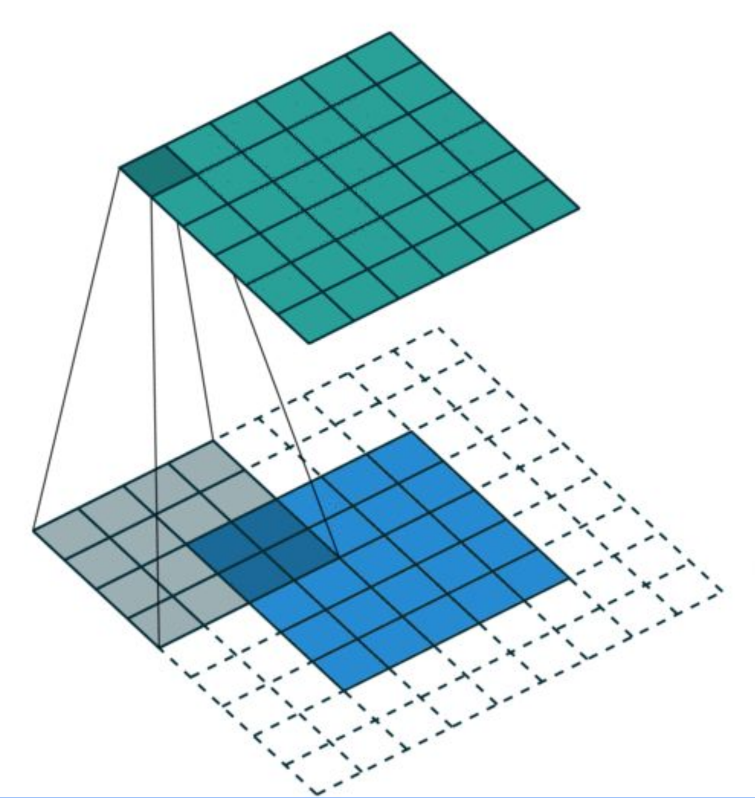
Convolution Operation



[Link to the Visualizations]

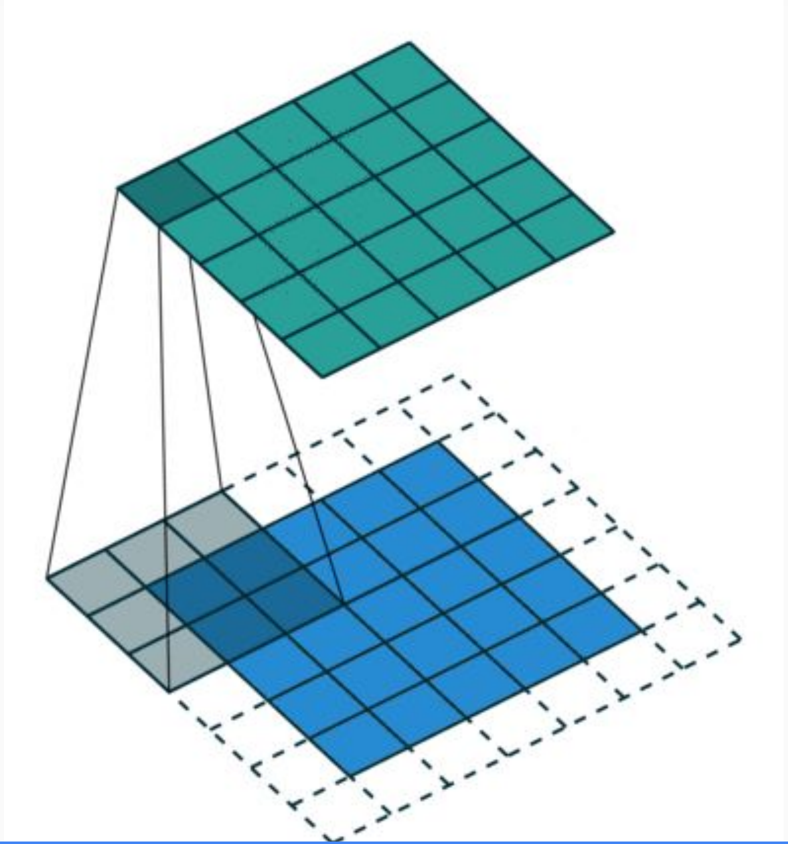
`nn.Conv2d(in_channels=1, out_channels=1, kernel_size=3)`

Convolution with Padding



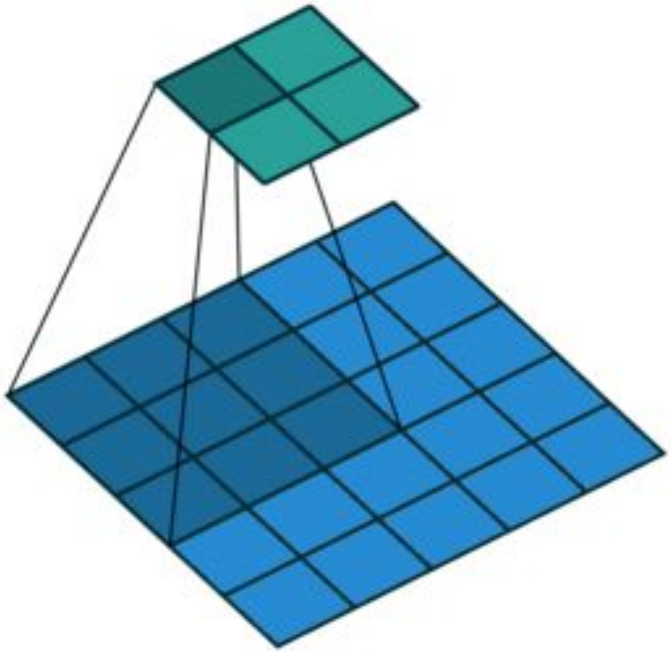
nn.**Conv2d**(**in_channels**=1, **out_channels**=1, **kernel_size**=4, **padding**=2)

Convolution Half Padding (Same Padding)



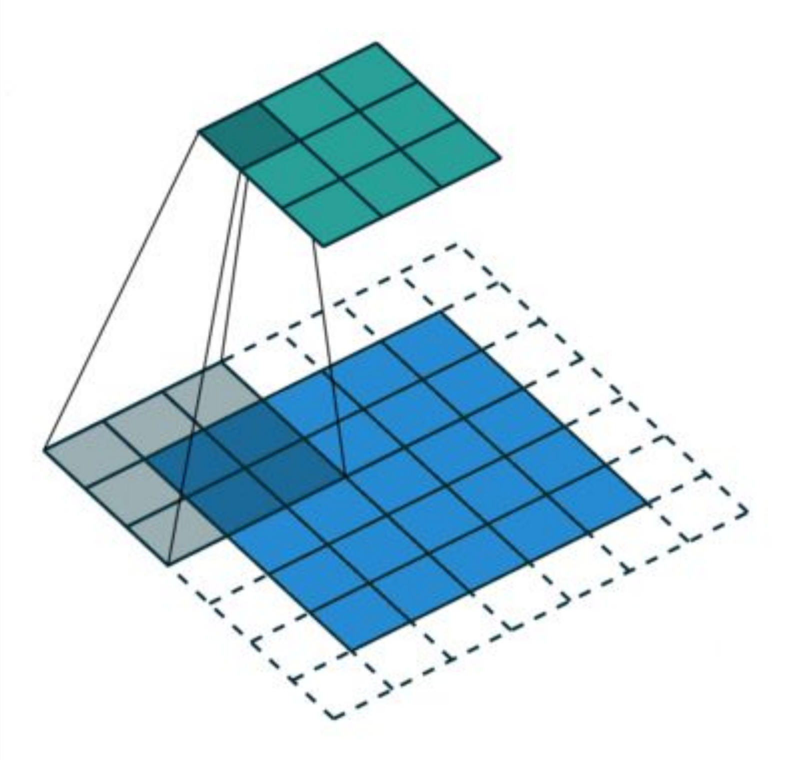
nn.**Conv2d**(**in_channels**=1, **out_channels**=1, **kernel_size**=3, **padding**=1)

Convolution with Stride-No Padding



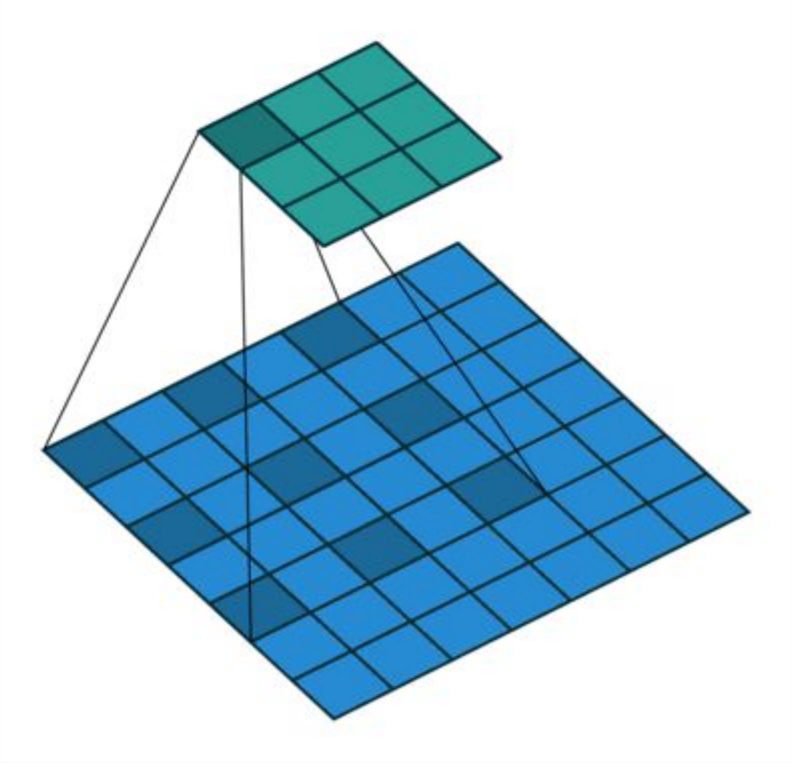
nn.**Conv2d**(**in_channels**=1, **out_channels**=1, **kernel_size**=3, **stride**=2, **padding**=0)

Convolution with Stride and Padding



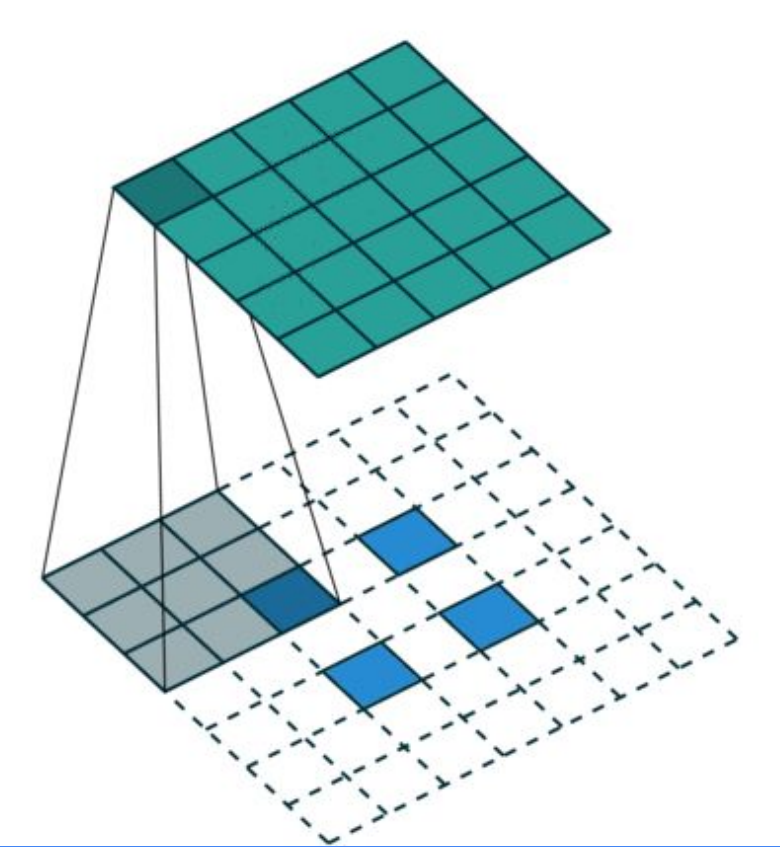
nn.**Conv2d**(**in_channels**=1, **out_channels**=1, **kernel_size**=3, **stride**=2, **padding**=1)

Dilated Convolution



nn.**Conv2d**(**in_channels**=1, **out_channels**=1, **kernel_size**=3, **stride**=1, **padding**=0, **dilation**=2)

Transposed convolution



nn.**ConvTranspose2d**(**in_channels**=1, **out_channels**=1, **kernel_size**=3, **stride**=1, **padding=**2, **dilation**=2)

Activation Functions

- Introduce non-linearity into the model
- Common activation functions: ReLU, sigmoid, and tanh
- ReLU is computationally efficient and helps mitigate the vanishing gradient problem

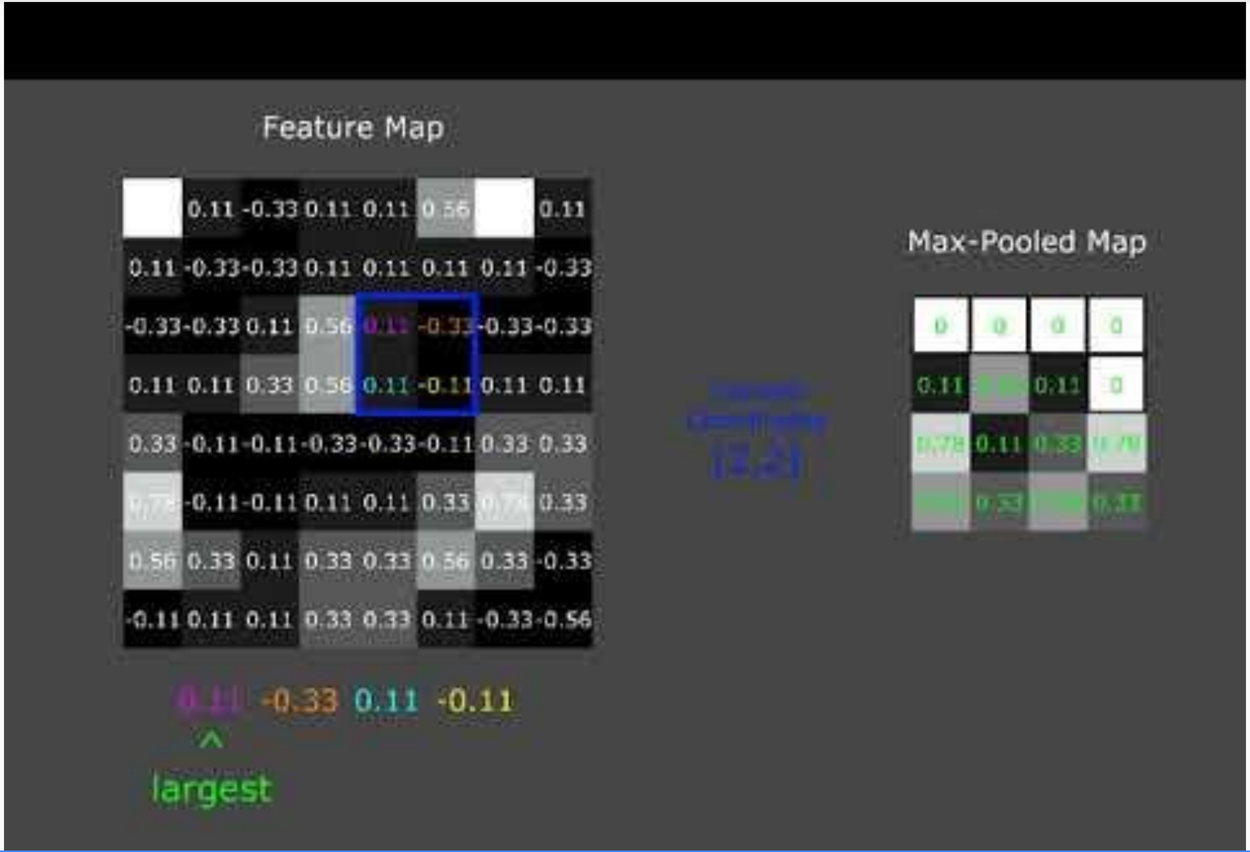
Pooling Layer

- To reduce the spatial dimensions of the feature maps
- To reduce the number of parameters in the network and control overfitting
- Used to downsample spatial dimensions and reduce computational complexity
- Increases translation invariance
- Common types: max pooling and average pooling

nn.**MaxPool2d**(**kernel_size**)

nn.**AvgPool2d**(**kernel_size**)

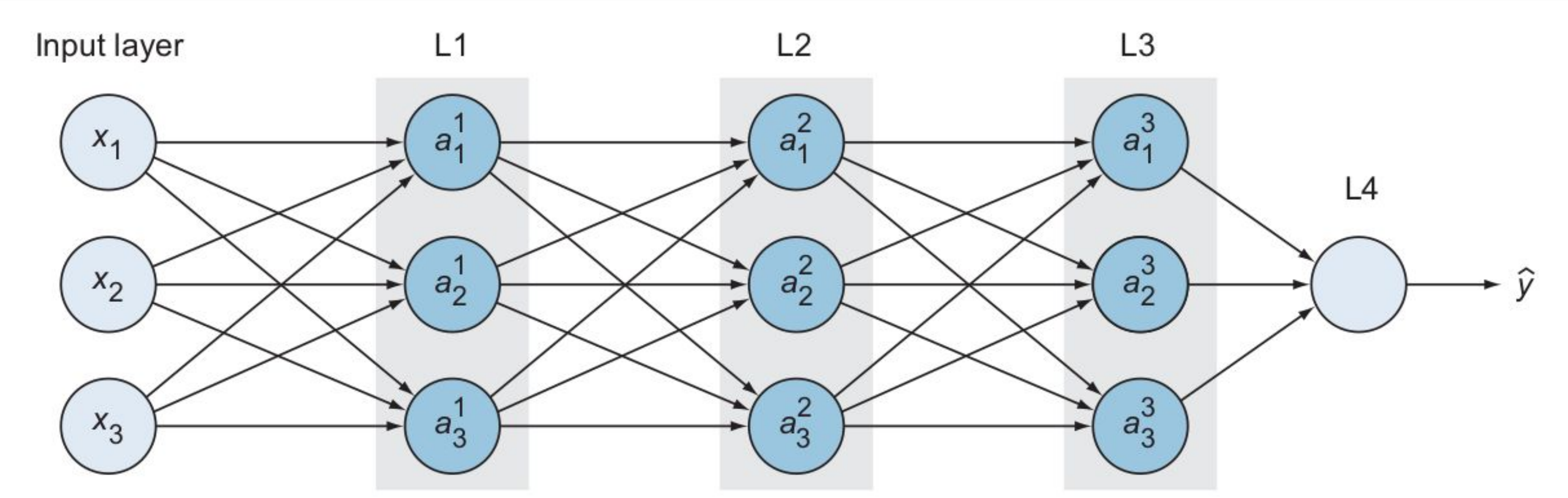
Max-Pooling



[Video Link]

nn.**MaxPool2d**(**kernel_size**=2)

The covariate shift problem



Batch normalization

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$ Mini-batch mean

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2$$ Mini-batch variance

$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

$$y_i \leftarrow \gamma X_i + \beta$$

Batch Normalization

- Improves training by normalizing input distribution of each layer
- Mitigates the internal covariate shift problem
- Improves training speed, generalization, and stability

nn.**BatchNorm2d**(**num_features**)