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

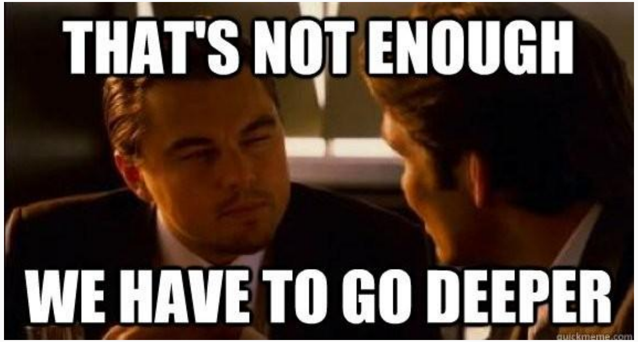
Introduction to Convolutional Neural Networks

- 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

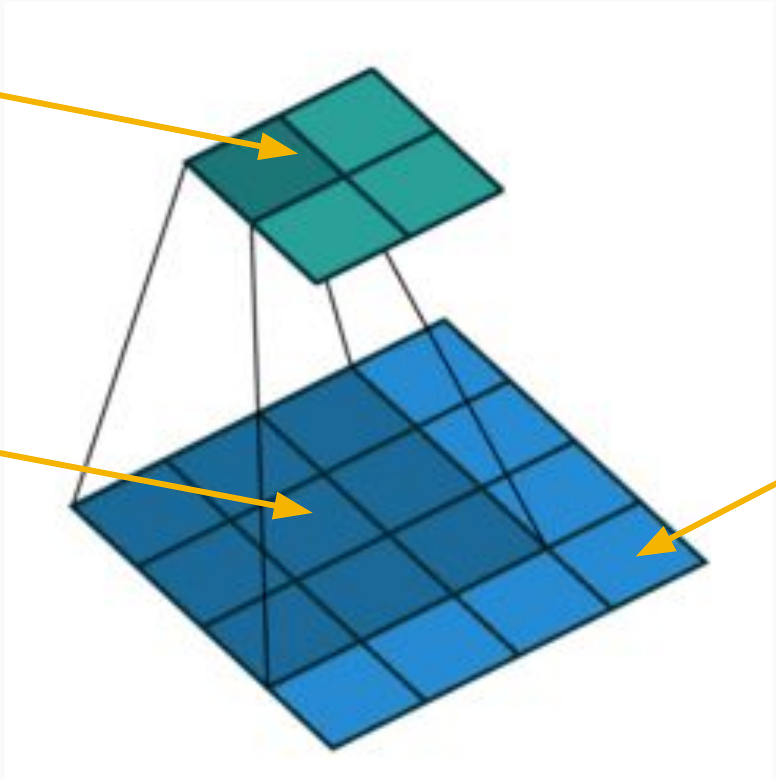
- Scalability: High dimension of visual data
- Local connectivity
- Parameter sharing
- Hierarchical feature learning
- Translation invariance



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

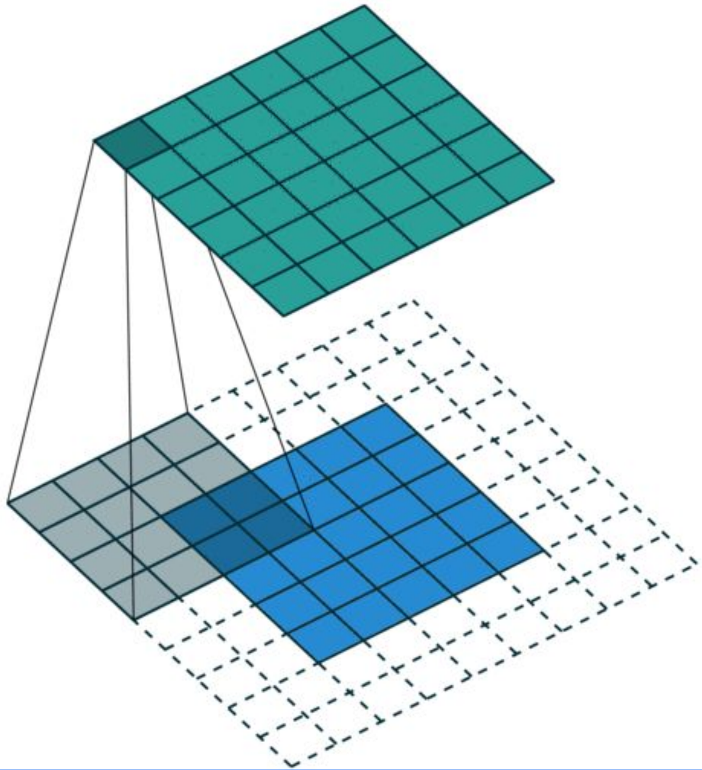
Convolution Operation



[Link to the Visualizations]

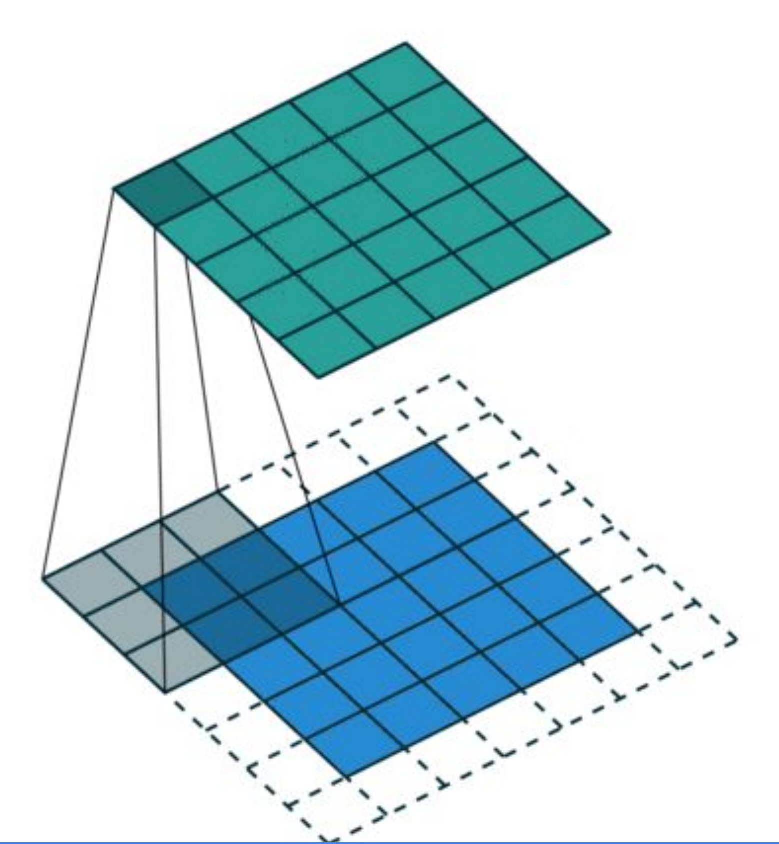
nn.**Conv2d**(**in_channels**=1, **out_channels**=1, **kernel_size**=3)

Convolution with Padding



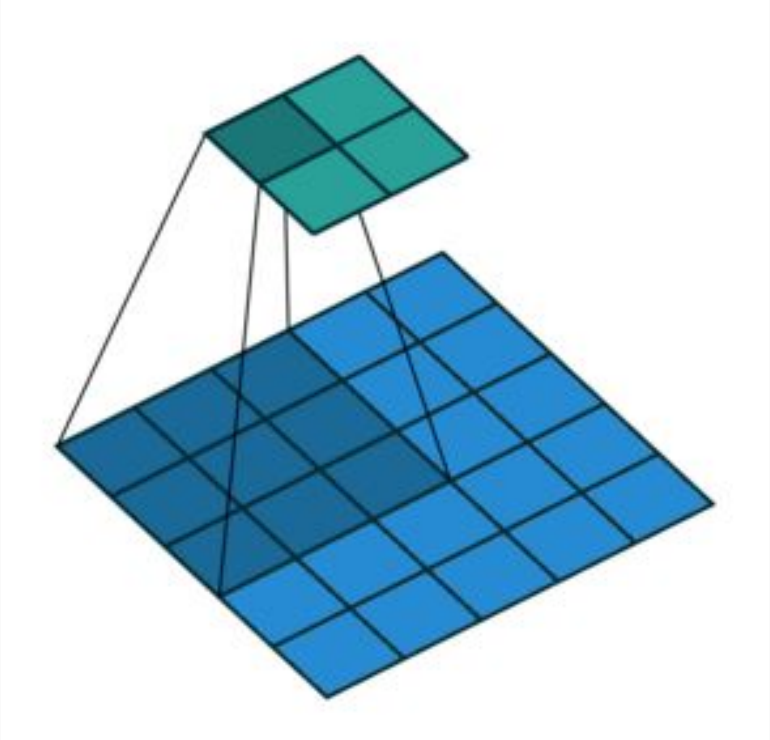
nn.**Conv2d**(**in_channels**=1, **out_channels**=1, **kernel_size**=4, **padding**=2)

Convolution Half Padding (Same Padding)



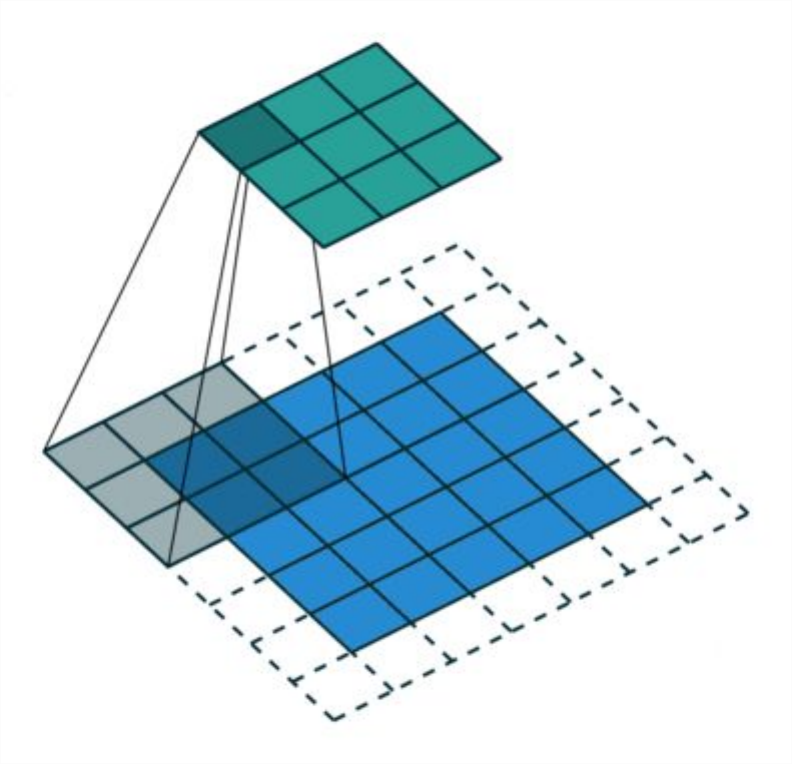
nn.**Conv2d**(**in_channels**=1, **out_channels**=1, **kernel_size**=3, **padding**=1)

Convolution with Stride-No Padding



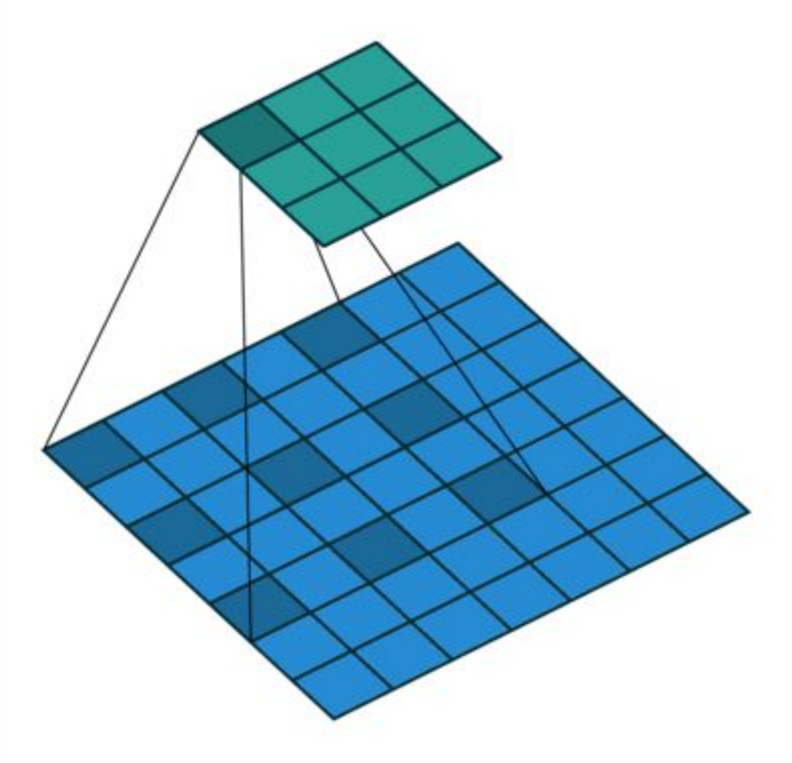
nn.**Conv2d**(**in_channels**=1, **out_channels**=1, **kernel_size**=3, **stride**=2, **padding**=0)

Convolution with Stride and Padding



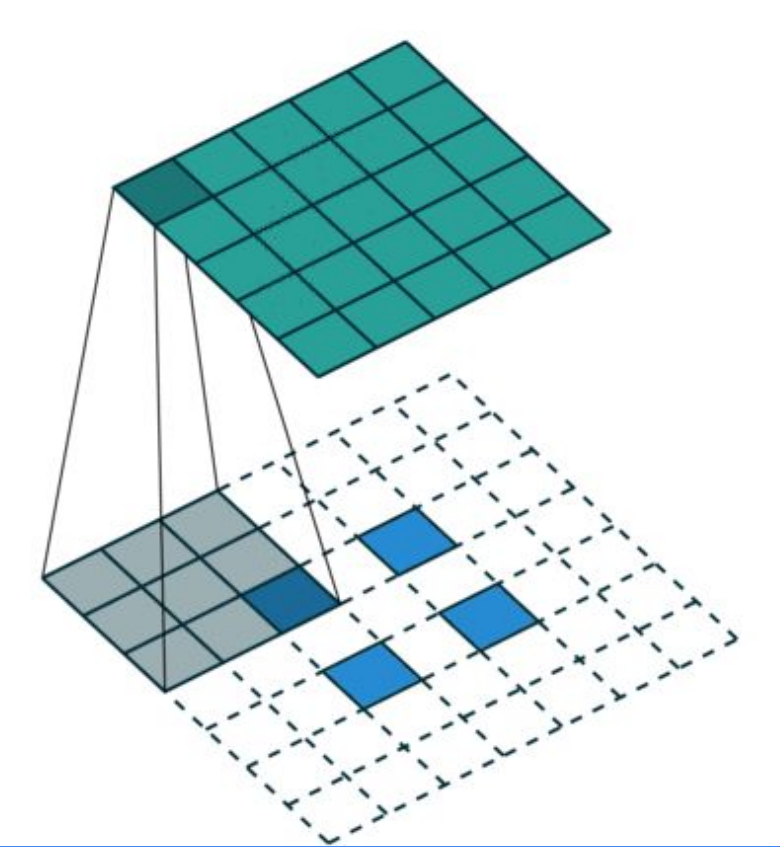
nn.**Conv2d**(**in_channels**=1, **out_channels**=1, **kernel_size**=3, **stride**=2, **padding**=1)

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

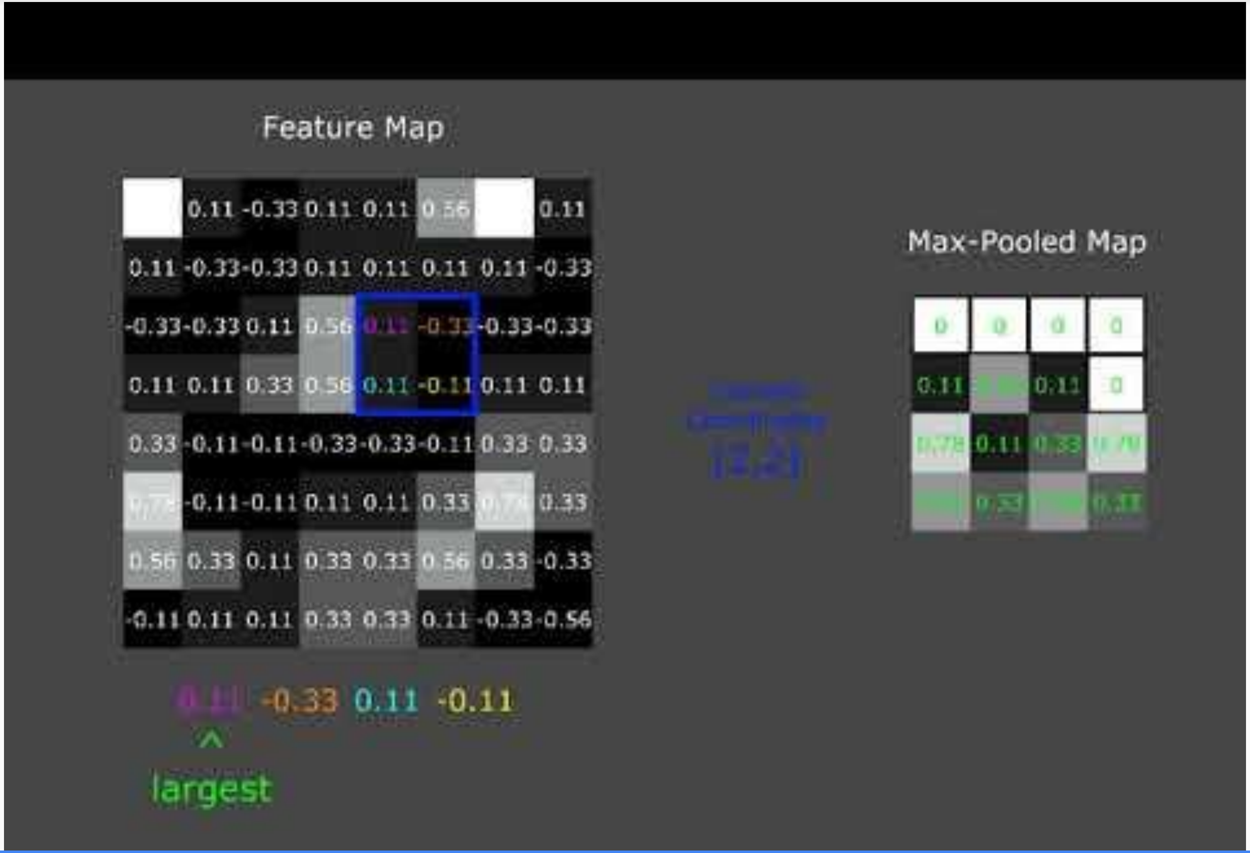
- Introduce non-linearity into the model
- Common activation functions: ReLU, sigmoid, and tanh
- ReLU is computationally efficient and helps mitigate the vanishing gradient problem

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**)

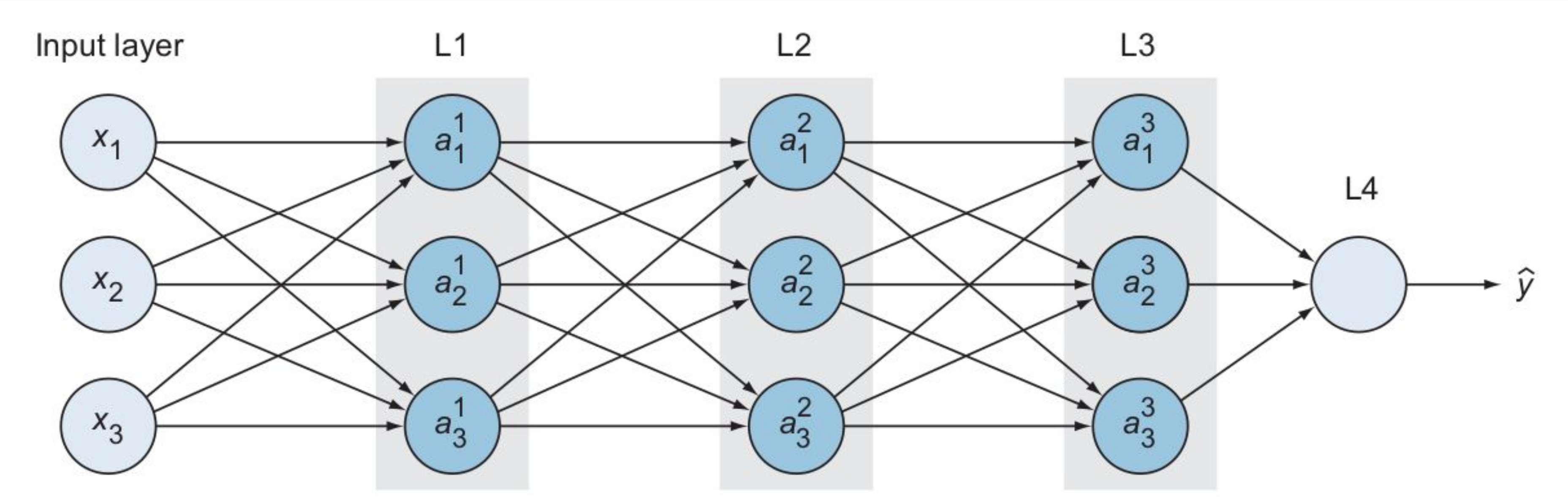
Max-Pooling



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

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

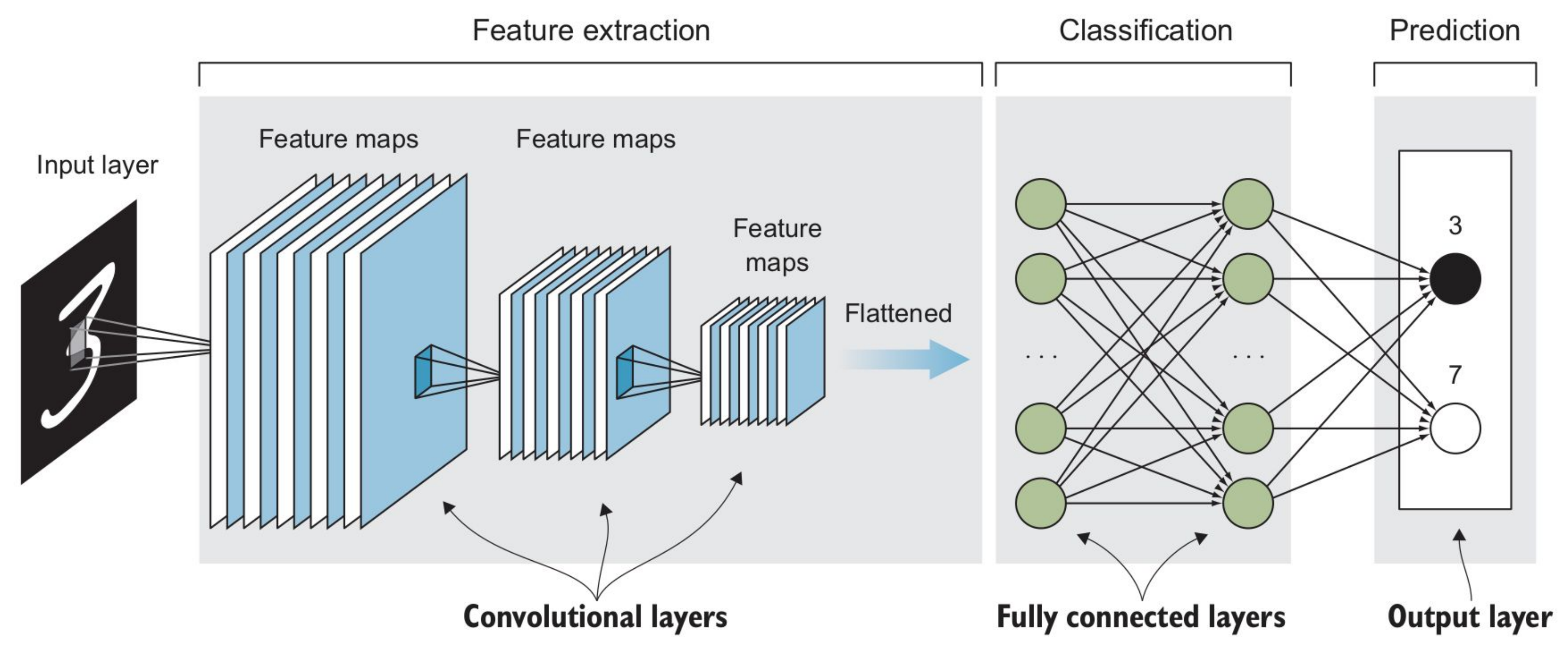
$$y_i \leftarrow \gamma X_i + \beta$$

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**)

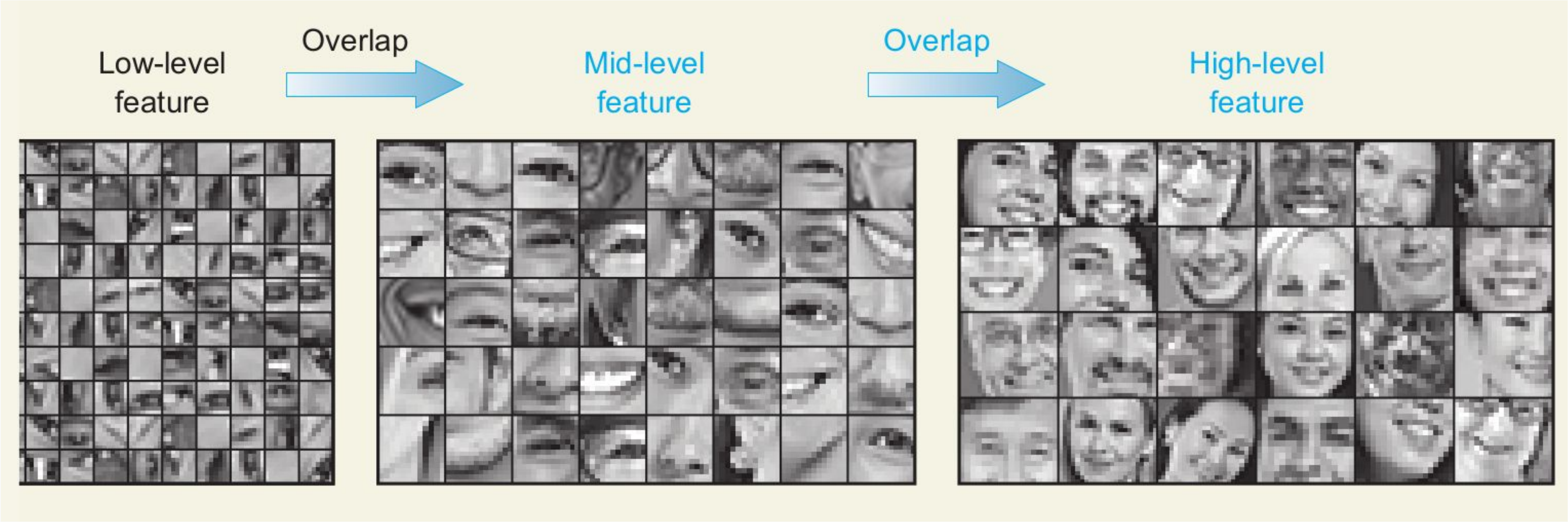
A typical CNN architecture for image classification



A filter make a feature map

In a CNN a feature map is the output of one filter applied to the previous layer

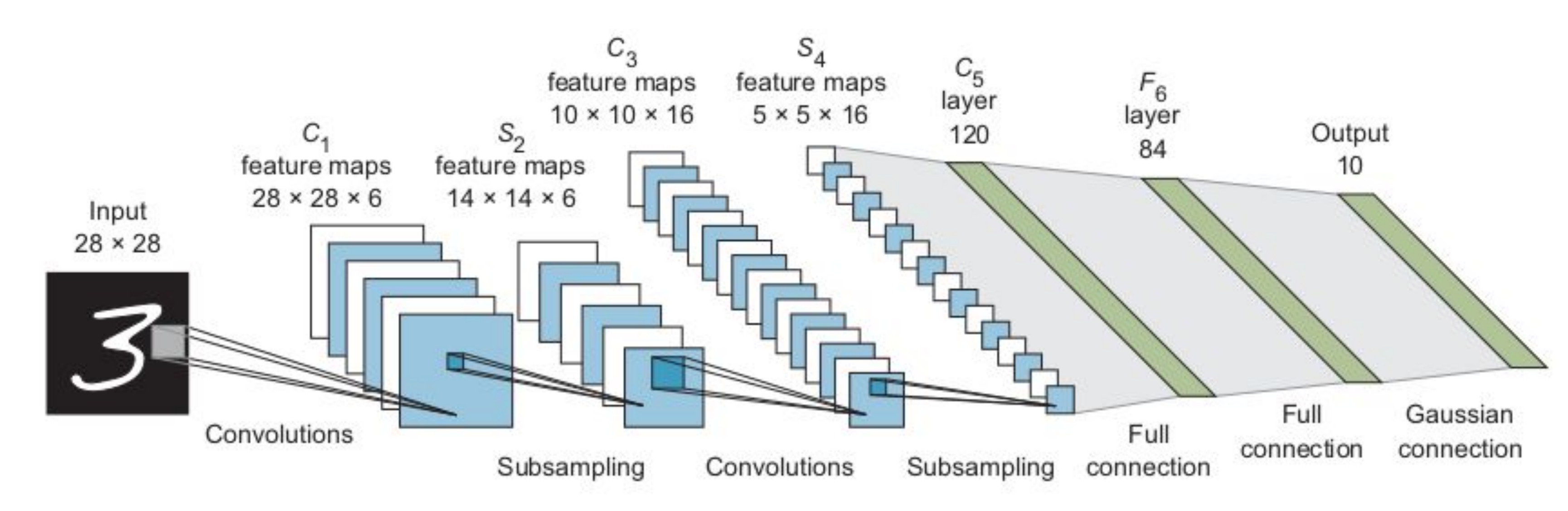
CNNs: Learning complex feature from simple features



Model Training

Coding Part

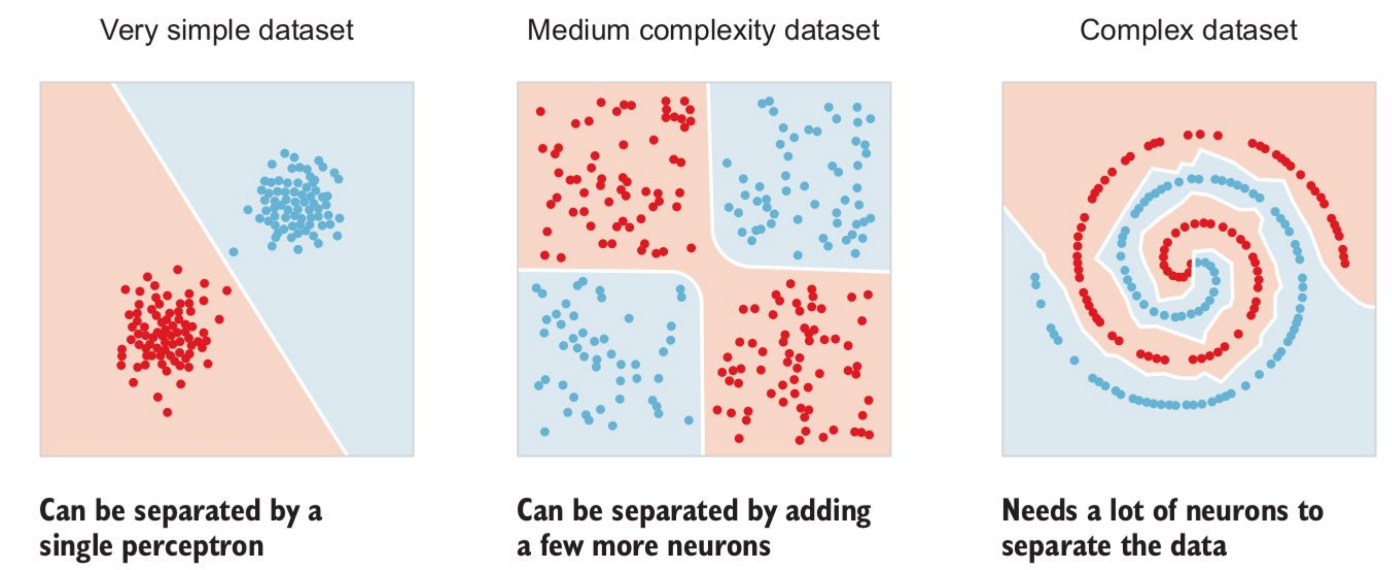
LeNet architecture



Tanh as the activation function!

Neural network hyperparameters: Network architecture

Network architecture
– Number of hidden layers (network depth)
– Number of neurons in each layer (layer width)
– Activation type

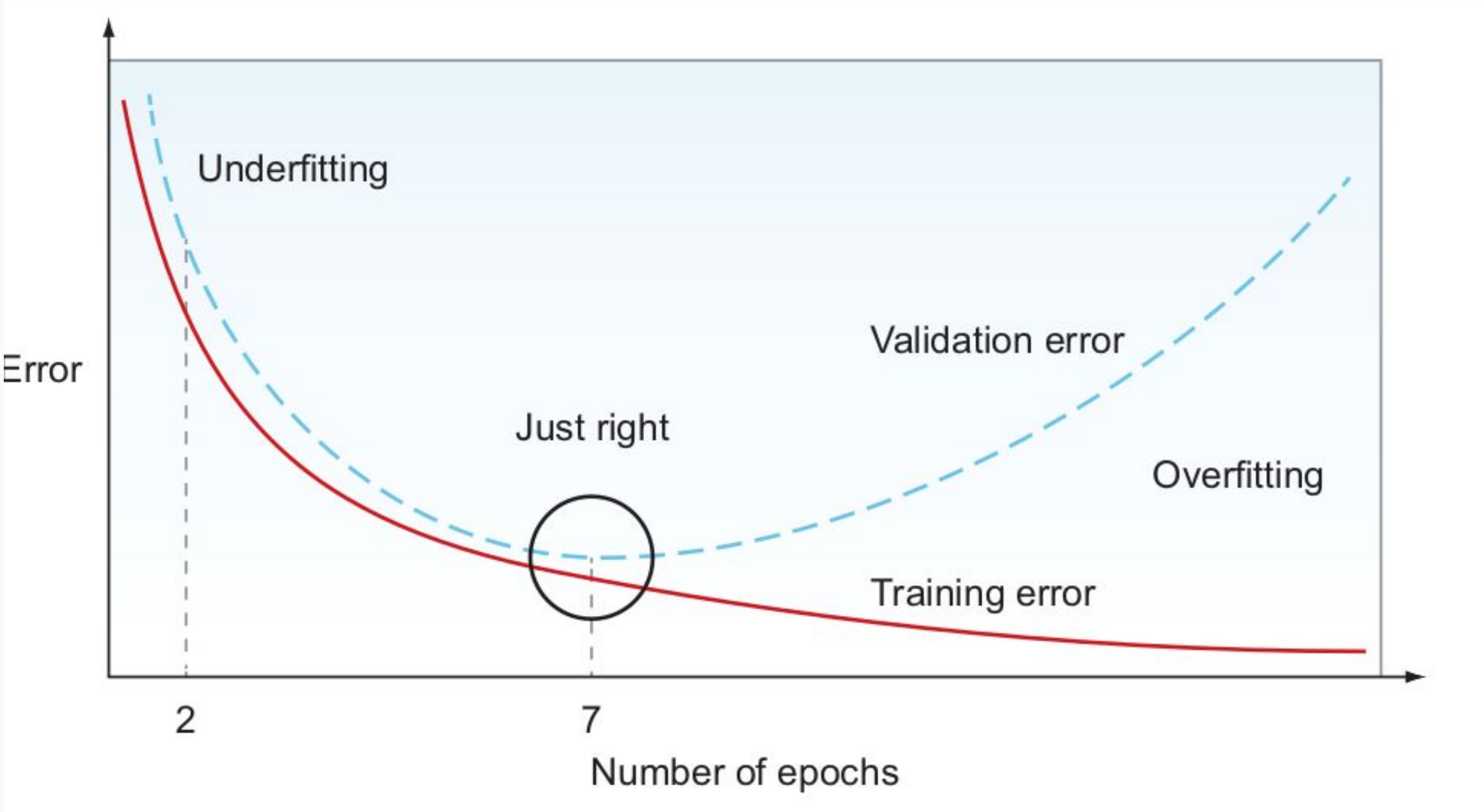


Neural network hyperparameters: Learning and optimization

- Learning rate and decay schedule
- Mini-batch size
- Optimization algorithms
- Number of training iterations or epochs (and early stopping criteria)

> *The learning rate is the single most important hyperparameter, and one should always make sure that it has been tuned. If there is only time to optimize one hyperparameter, then this is the hyperparameter that is worth tuning.*
>
> —Yoshua Bengio

How long to continue the training procedure?



Regularization Techniques

- The importance of regularization in deep learning
- Overfitting and generalization
- Regularization techniques overview
- Common techniques:
    - L1 regularization
    - L2 regularization
    - Dropout
    - Data augmentation

L1 Regularizations

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$$

```
# Add the L1 regularization term to the loss
l1_norm = sum(torch.abs(param) for param in model.parameters())
loss += l1_lambda * l1_norm
```

L2 Regularizations

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

```
# Add the L1 regularization term to the loss
l1_norm = sum(torch.norm(param, 1) for param in model.parameters())
loss += l1_lambda * l1_norm
```

or

```
optimizer = optim.SGD(model.parameters(), lr=0.1, weight_decay=0.001, momentum=0.9)
```

Resources

- https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md
- Elgendy, M. (2020). Deep learning for vision systems. Simon and Schuster.