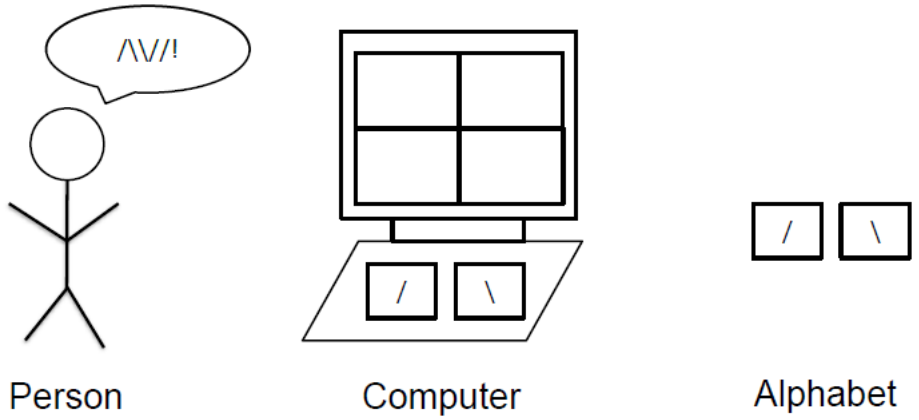
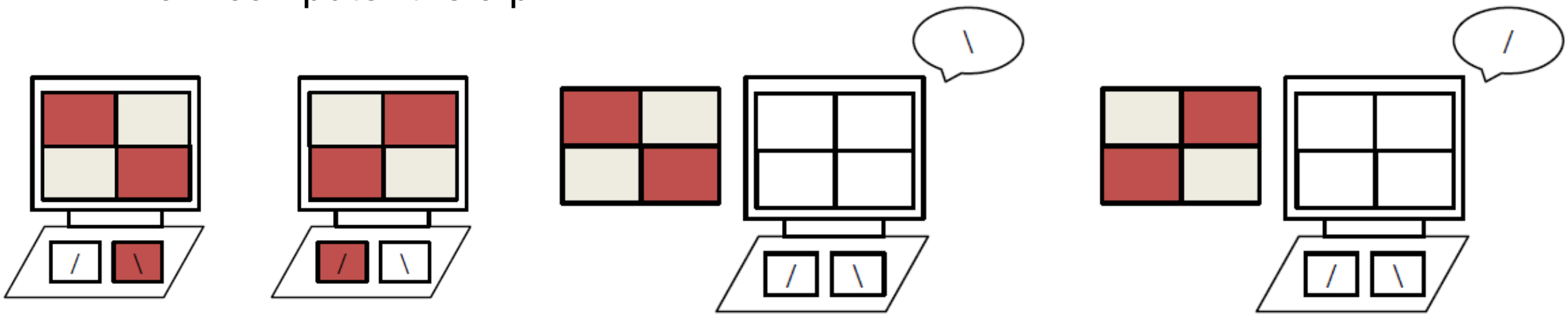
6.3 A First Look at Convolution Neural Network (1)

- Image a simple world

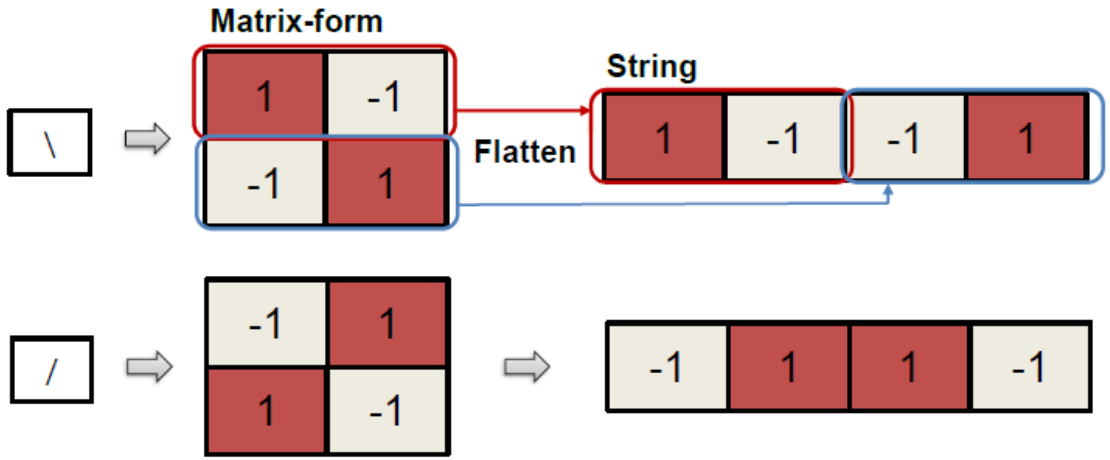


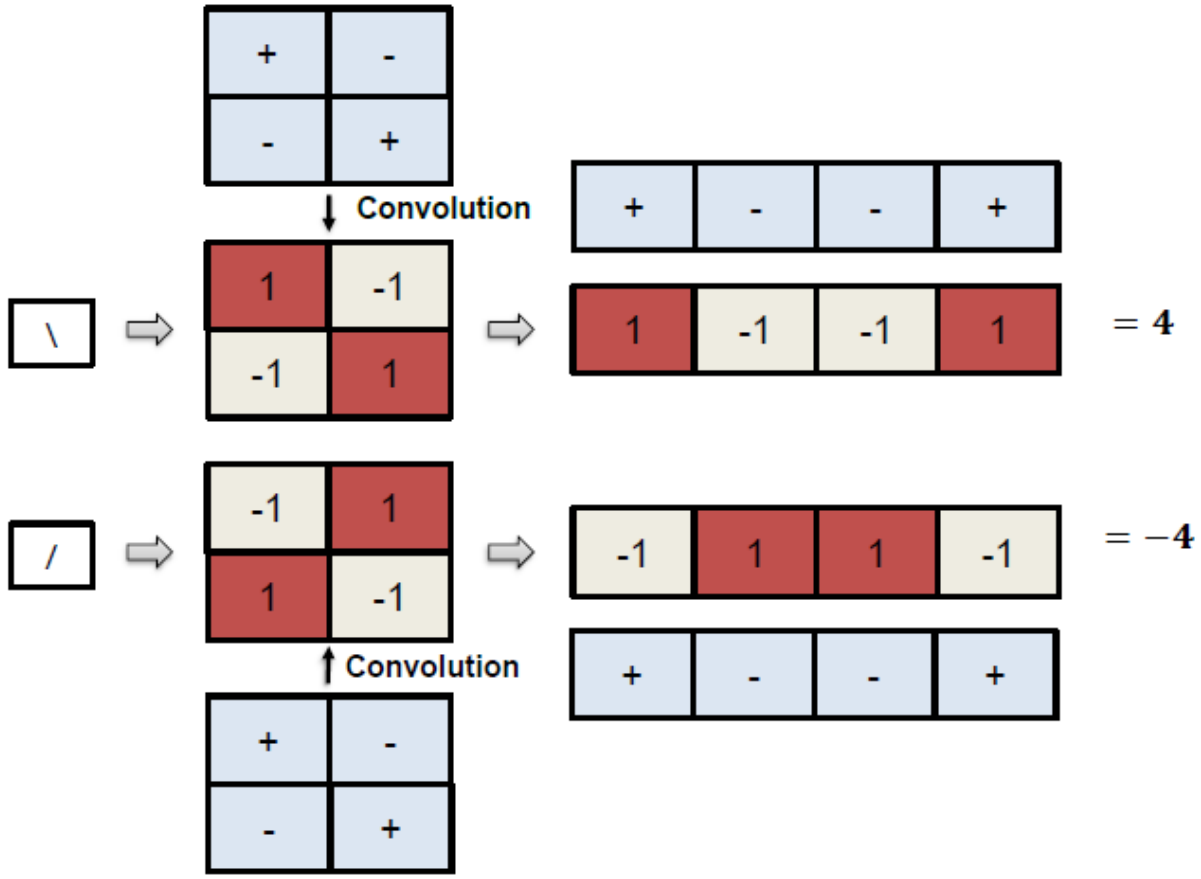
- Train computer the alphabet



A First Look at Convolution Neural Network (2)

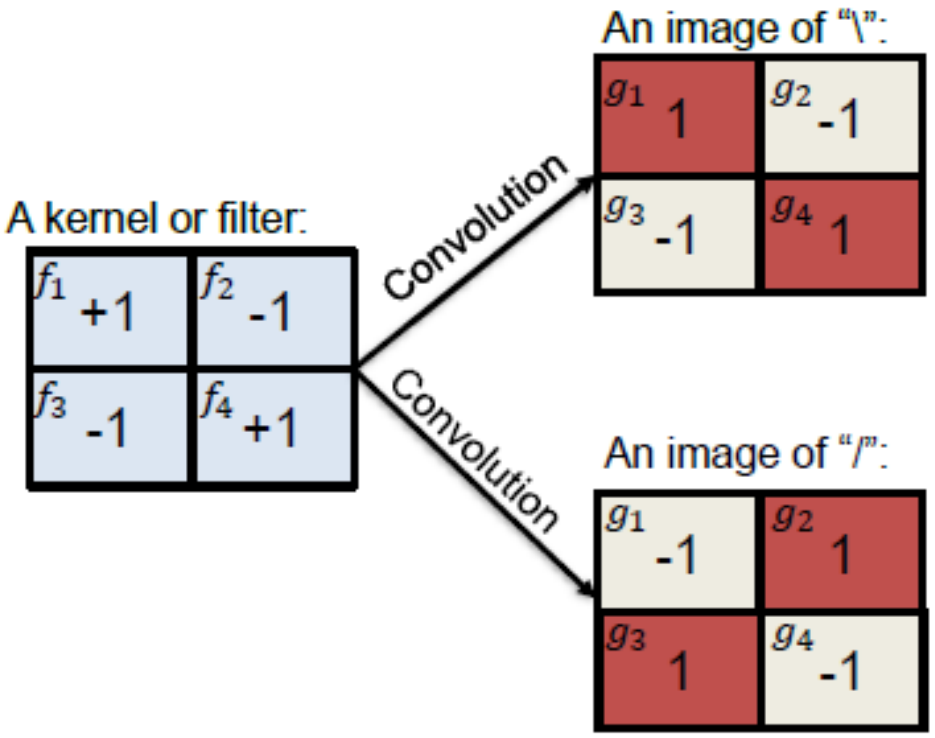
- How the data store in the computer
- How can we define a right operation? (addition?)





A First Look at Convolution Neural Network (3)

- Convolution is a formal mathematical operation, just as multiplication and addition/integration.
- Convolution is a way to pass a filter over the image to find key features.



Result:

$$f_1 \times g_1 + f_2 \times g_2 + f_3 \times g_3 + f_4 \times g_4$$
$$= 1 \times 1 + (-1) \times (-1) + (-1) \times (-1) + 1 \times 1$$
$$= 4$$

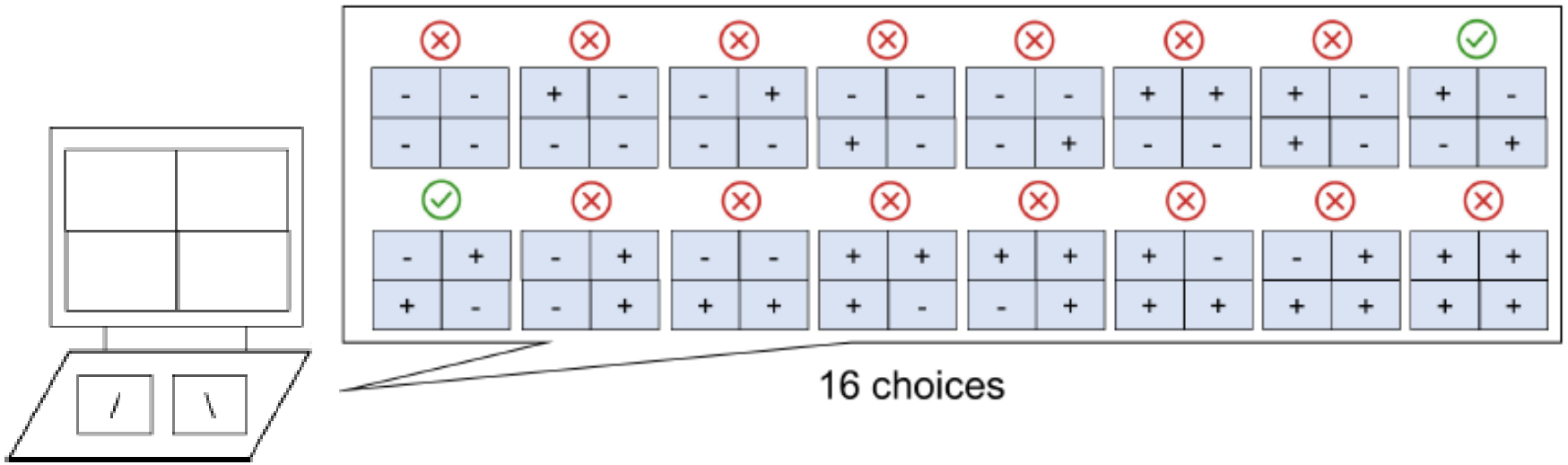
Result is positive, backward slash: "\"

$$f_1 \times g_1 + f_2 \times g_2 + f_3 \times g_3 + f_4 \times g_4$$
$$= 1 \times (-1) + (-1) \times 1 + (-1) \times 1 + 1 \times (-1)$$
$$= -4$$

Result is negative, forward slash: "/"

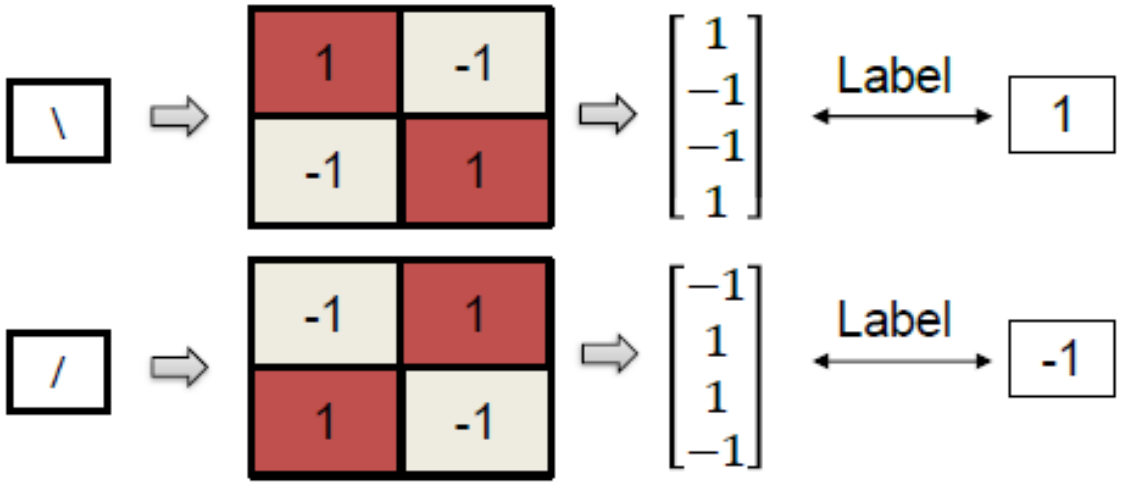
Kernels to be tested

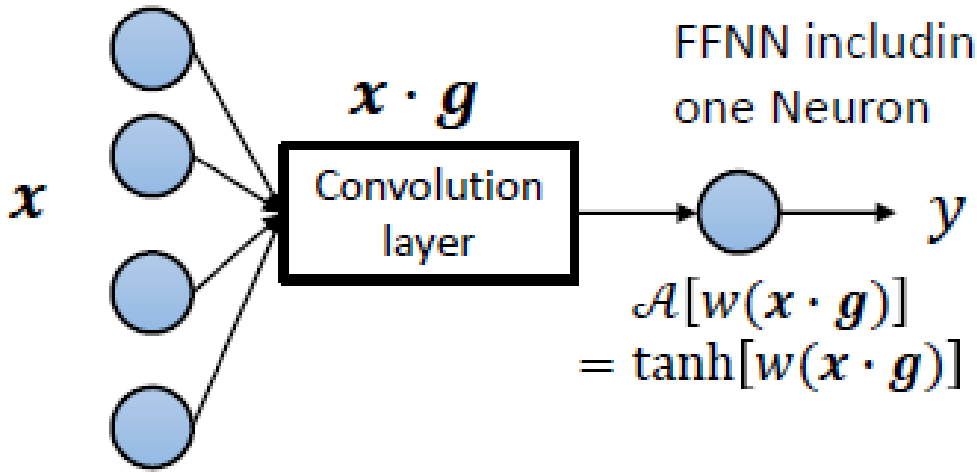
- We have a chance to check all the possible filters only if we assume the values in a filter are either 1 or +1.
  - Select good filters by checking all the possibilities
- If we consider a filter with real numbers, a more efficient approach is required.
- That is why CNN get into the picture!



Mini CNN is used to figure out a good filter

Dataset for training (Two labeled images):





FFNN including only one Neuron

Five unknow parameters:

g_1, g_2, g_3, g_4, w

Cost/objective function:

$(y_{1*} - y_1)^2 + (y_{2*} - y_2)^2$

Python code for Mini-CNN

Import library

```
import numpy as np
```

Generate data

```
image1=np.array([1, -1,-1,1])
image2=np.array([-1, 1,1,-1])
X=np.vstack((image1,image2))
Y=np.array([1,-1])
#Y[1]
```

Define CNN output and objective function

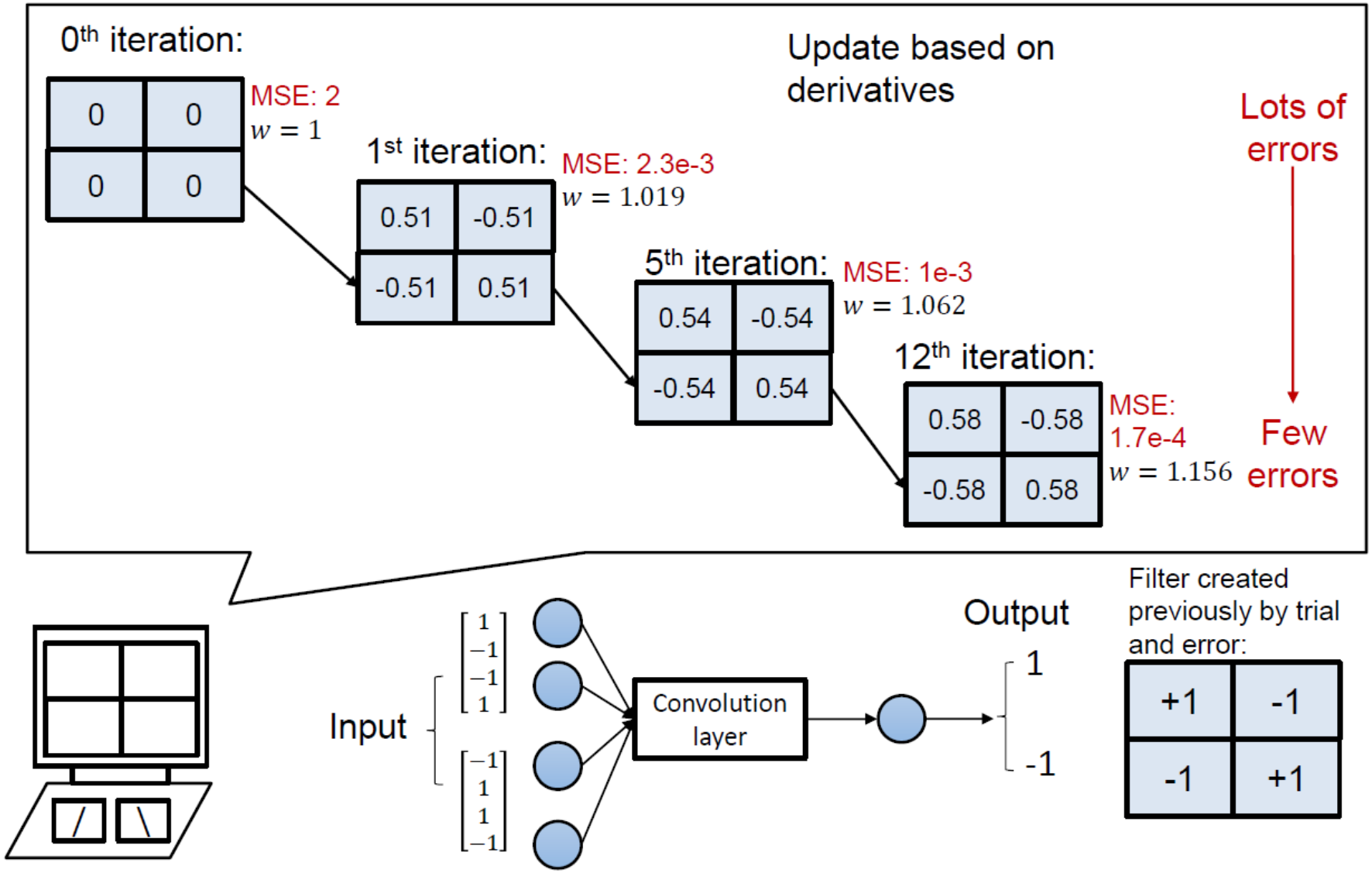
```
def calc_y(g):
# calculate y
    x=g[0]*(g[1]*X[:,0]+g[2]*X[:,1]+g[3]*X[:,2]+g[4]*X[:,3])
    y=np.tanh(x)
    return y

# define objective
def objective(g):
    # calculate y
    y = calc_y(g)
    # calculate objective
    obj = 0.0
    for i in range(len(Y)):
        obj = obj + ((y[i]-Y[i]))**2
    # return result
    return obj
```

Minimize objective

```
niter=12
g=np.array([1,0,0,0,0])
cObjective = 'Objective: ' + str(objective(g))
print(cObjective)
for i in range(niter):
    solution = minimize(objective, g, options={'maxiter':1})
    g=solution.x
    print('coeffi g=',g)
    y_recover=calc_y(g)
    print('y_recover=',y_recover)
    cObjective = 'Objective: ' + str(objective(g))
    print(cObjective)
```

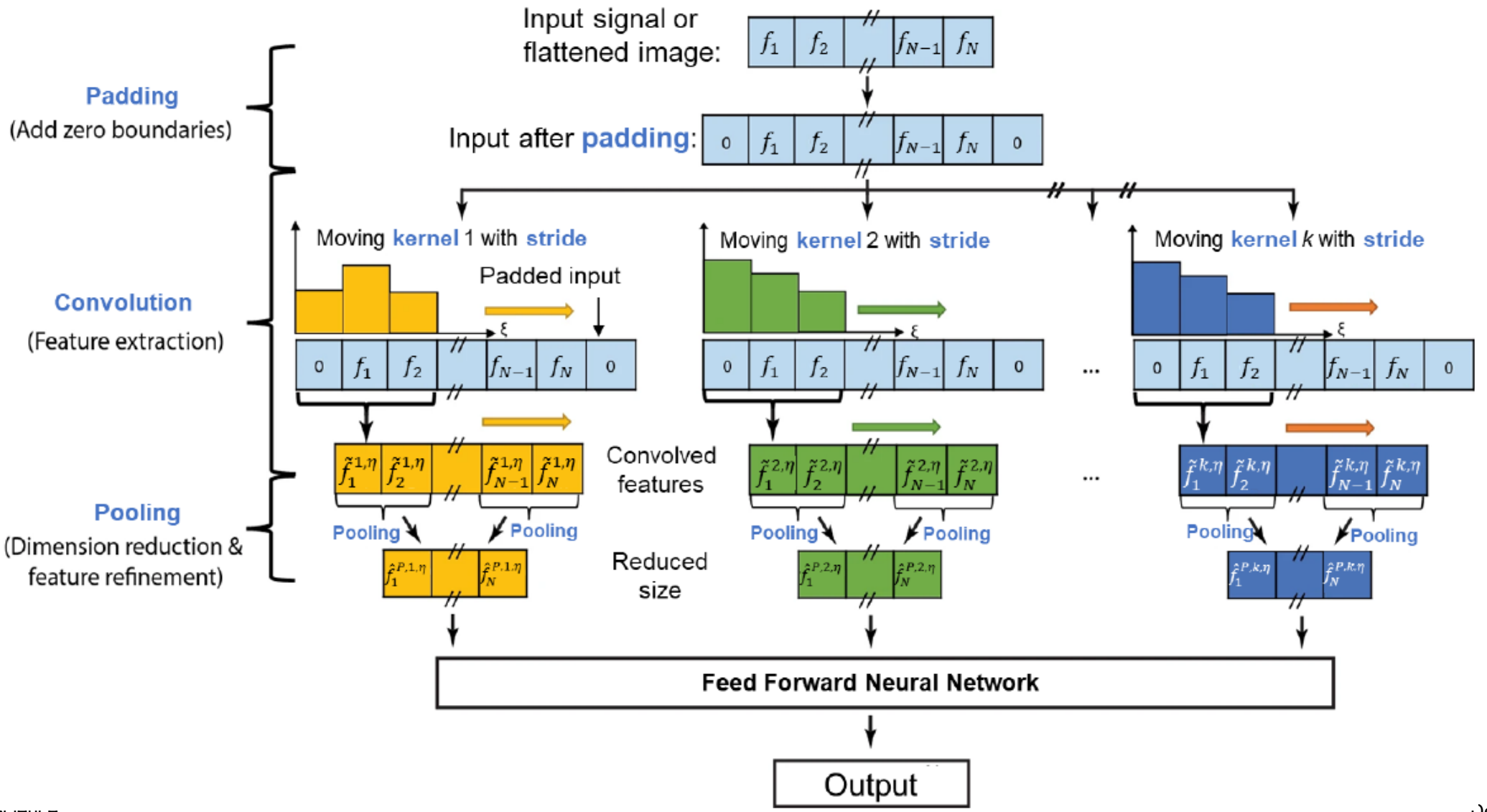
Using gradient descent to update filter



Building Blocks in CNN

- Image classification problem: logistic regression and support vector machine
  - Pixel values as features: 36 x 36 image → 1296 features
  - Lost a lot of spatial interactions between pixels, very time-consuming
  - Highly depends on the knowledge and experience of the domain experts
- CNN (1990's)
  - Use information from adjacent pixels to down-sample the image into features by convolution and pooling
  - Use prediction layers (e.g., a FFNN) to predict the target values
  - Building blocks of input, convolution, padding, stride, pooling, FFNN, and output
    - Input layer such as a signals (1D) or an image (2D)
    - Padded input can be obtained by adding zeros around the margin of the signal or image
    - Multiple convolution operations will be done using several moving kernels to extract features from the padded input
    - Dimensions of the convolved features can be reduced using pooling layers
    - Those reduced features then are used as input for a FFNN to calculate the output of the CNN

An illustrative structure of CNN including several building blocks and concepts

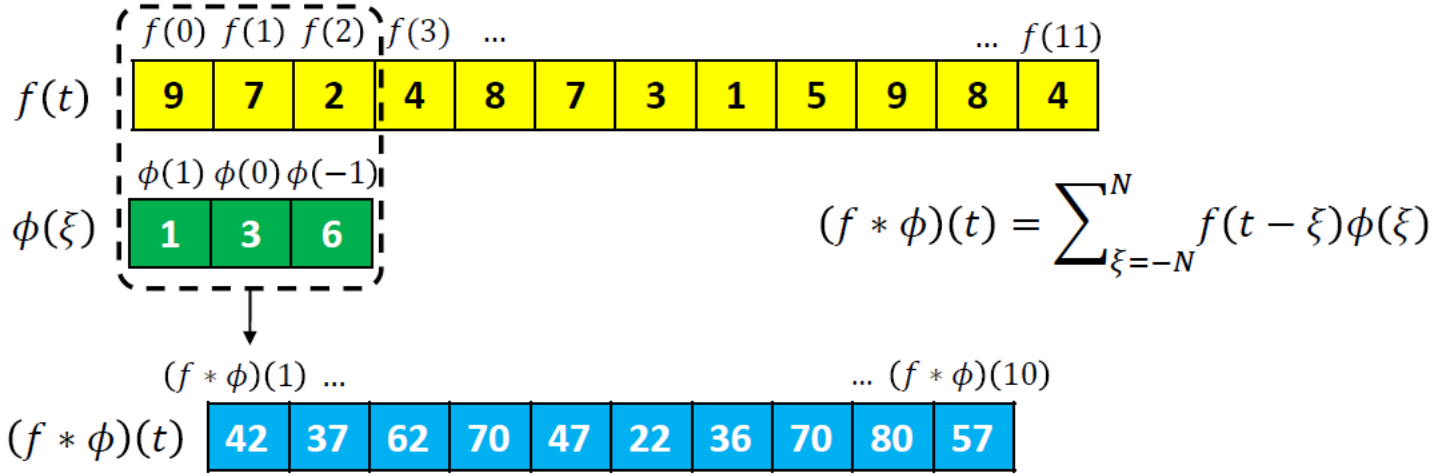


Terminology used in CNN

Convolution	A mathematical operation that does the integral of the product of functions(signals), with one of the signals slides. It can extract features from the input signals
Kernel (filter)	A function used to extract important features
Padding	A technique to simply add zeros around the margin of the signal or image to increase its dimension. Padding allows to emphasize the border values and in order lose less information
Stride	The steps of sliding the kernel during convolution. The kernel move by different stride values is designed to extract different kinds of features. The amount of stride chosen affects the size of the feature extracted
Pooling	An operation that takes maximum or average of the region from the input overlapped by a sliding kernel. The pooling layer helps reduce the spatial size of the convolved features by providing an abstracted representation of them
Fully connected layers	A FFNN in which layer nodes are connected to every node in the next layer. The fully connected layers help learn non-linear combinations of the features outputted by the convolutional layers

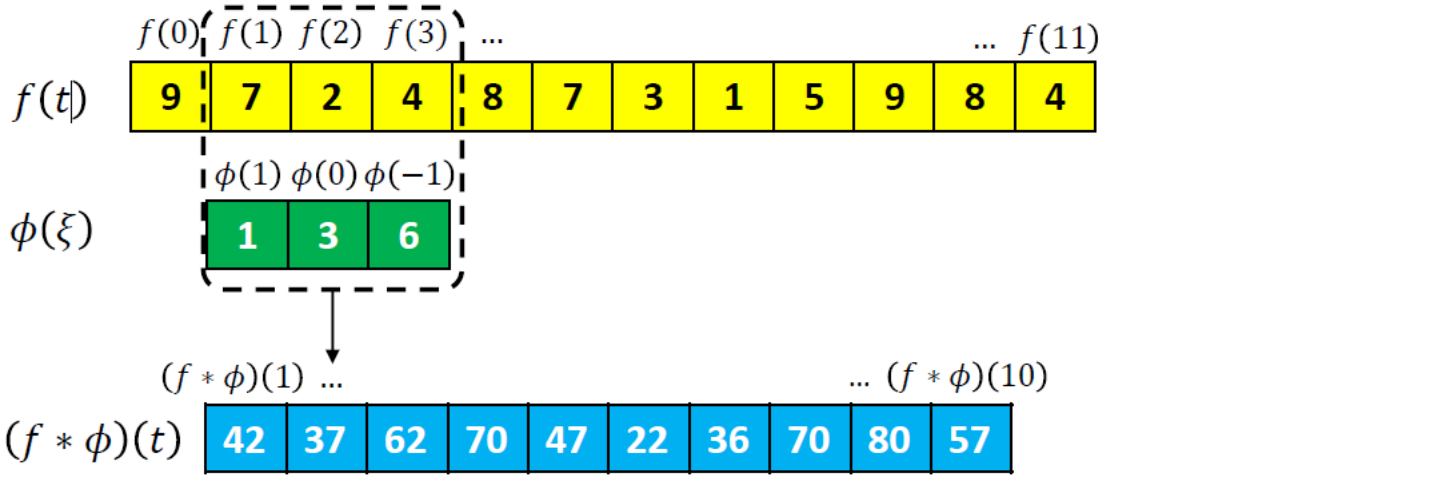
Convolution

Convolution operation step 1:



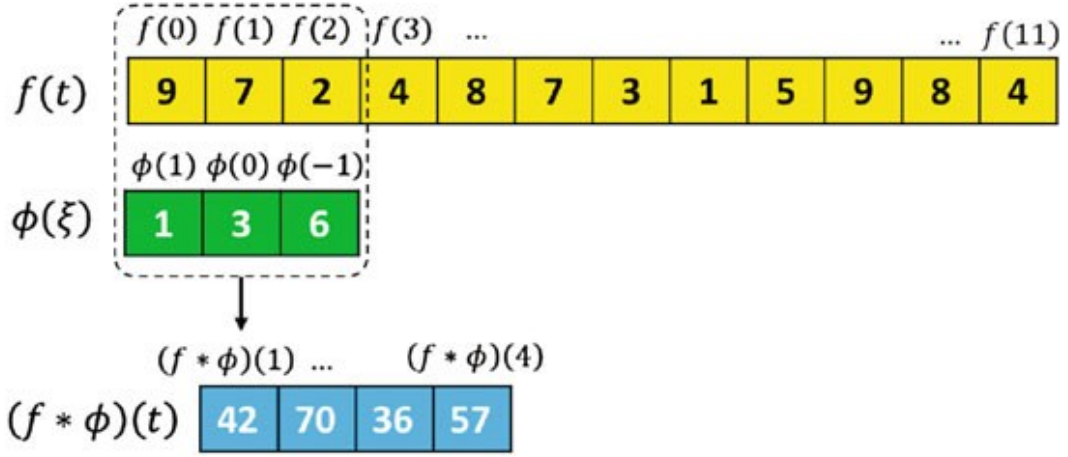
$$(f * \phi)(t) = \sum_{\xi=-N}^{N} f(t-\xi)\phi(\xi)$$

Convolution operation step 2:

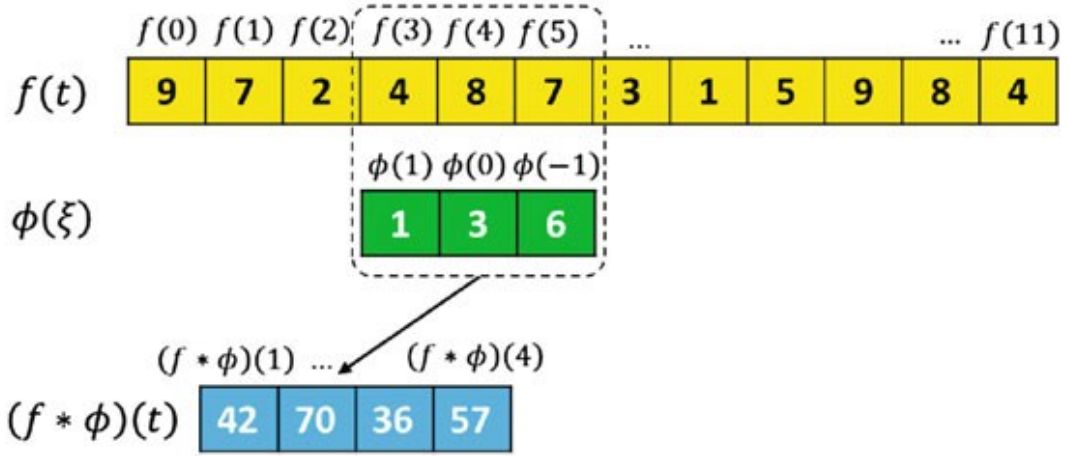


Stride

Convolution operation step 1 with a stride value of 3:



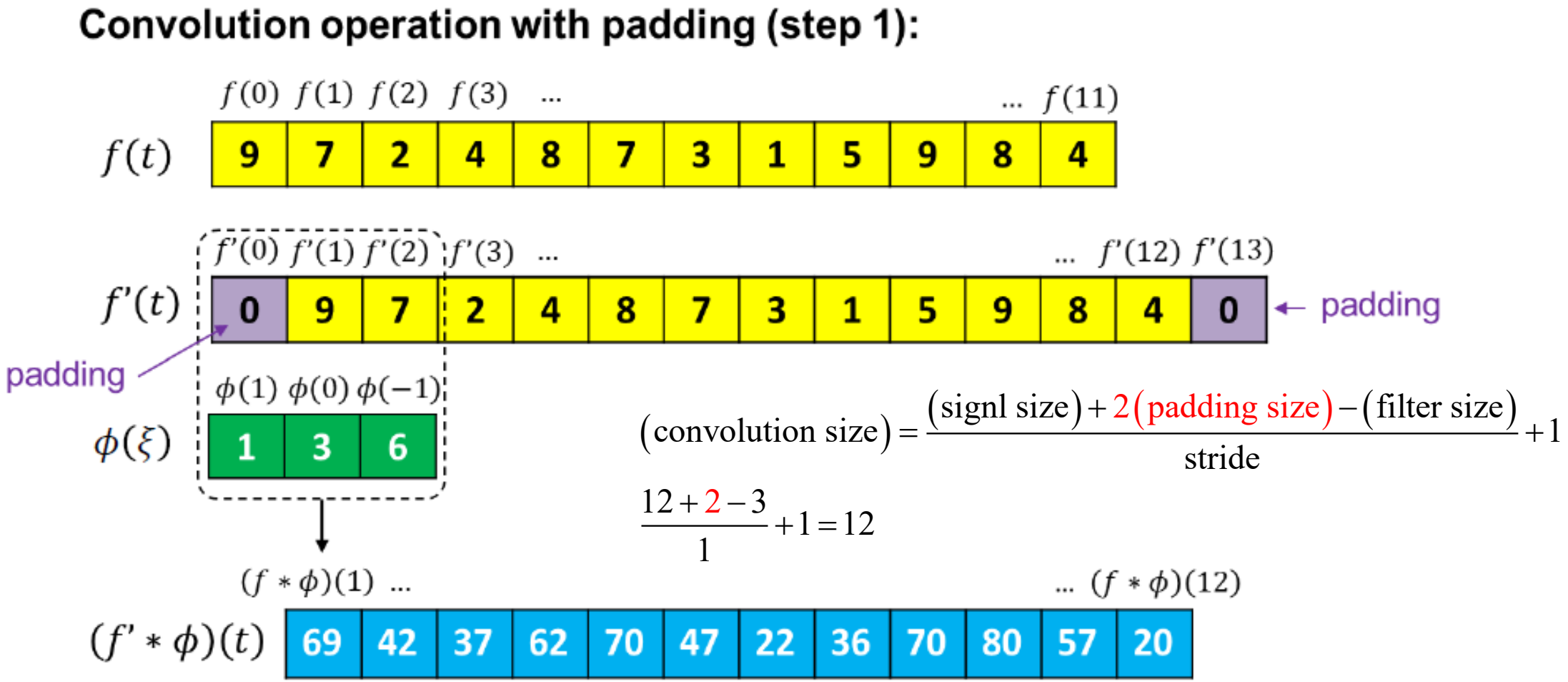
Convolution operation step 2 with a stride value of 3:



$$(\text{convolution size}) = \frac{(\text{signl size}) - (\text{filter size})}{\text{stride}} + 1$$

$$\begin{cases} \frac{12-3}{1} + 1 = 10 \\ \frac{12-3}{3} + 1 = 4 \end{cases}$$

Add padding to make dimensions consistent



An example showing max and average pooling layers

- Reduce the spatial size of the convolved features
- Reduce overfitting by providing low-dimensional representations
  - Max pooling helps reduce noise by ignoring noisy small values in the input data

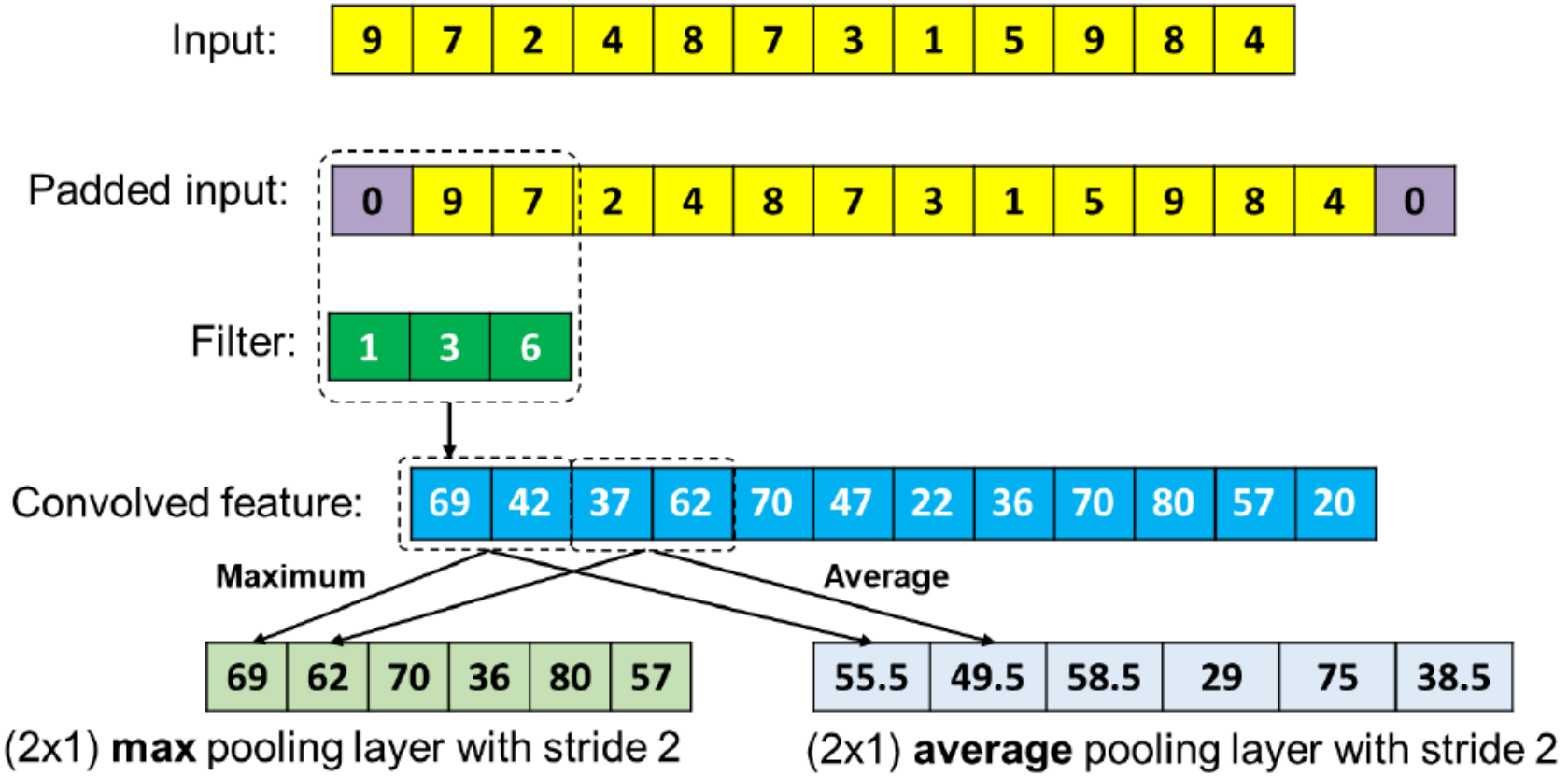
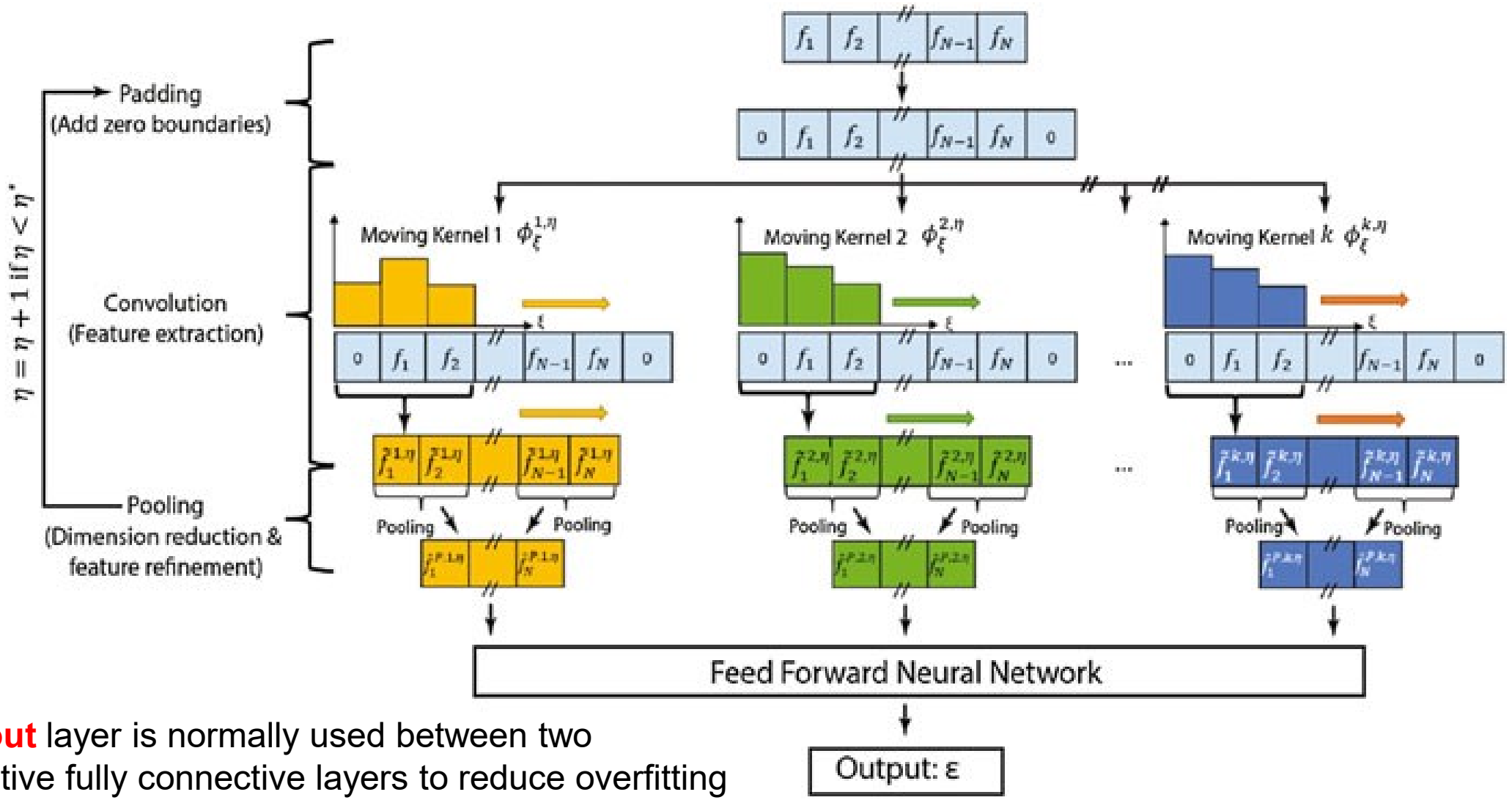


Illustration of one-dimensional CNN



A **dropout** layer is normally used between two consecutive fully connective layers to reduce overfitting

General Notations

$$\tilde{f}_x^{\kappa,\eta} = \sum_{\xi=-(L_{conv}-1)/2}^{(L_{conv}-1)/2} \phi_\xi^{\kappa,\eta} f_{x+\xi}^{padded,\eta} + b^{\kappa,\eta} \leftarrow \text{discrete convolution operator}$$

$$\begin{cases} f_{x+\xi}^{padded,\eta} : \text{ padded input} \\ x : \text{ counting index for a location within the kernel} \\ \phi_\xi^{\kappa,\eta} : \ \kappa\text{-th kernel function} \\ b^{\kappa,\eta} : \text{ bias for convolution process } \left(\eta = 1,\ldots,N_{conv}\right) \\ L : \text{ size of the kernel function} \end{cases}$$

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{\varepsilon}^i + \boldsymbol{\varepsilon}^{*i}\right)^2$$

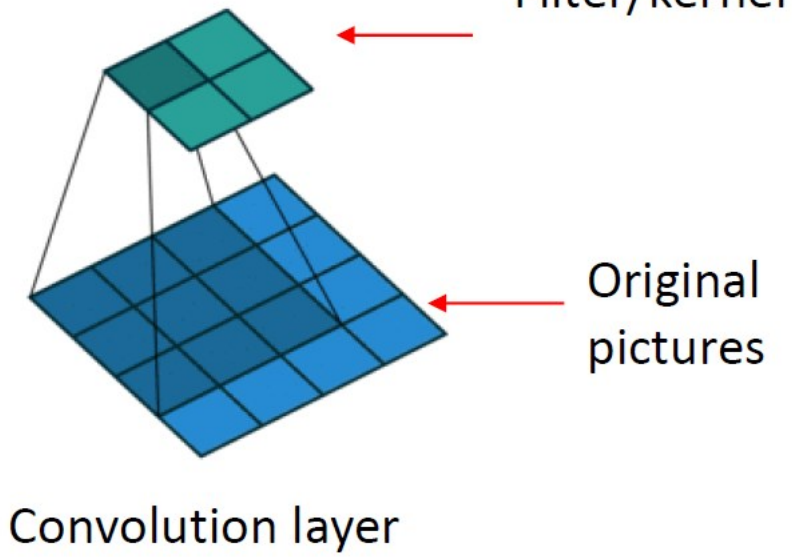
$$\begin{cases} N : \text{ number of data points in the training set} \\ \boldsymbol{\varepsilon}^i : \text{ CNN output of the } i\text{-th data point} \\ \boldsymbol{\varepsilon}^{*i} : \text{ labeled output of the } i\text{-th data point} \end{cases}$$

$$\hat{f}_\alpha^{P,\kappa,\eta} = \max\left(\tilde{f}_\xi^{\kappa,\eta},\ \xi \in \left[\left(\alpha-1\right)L_{pooling}+1,\ \alpha L_{pooling}\right]\right) \leftarrow \text{output value after the max pooling}$$

$$\begin{cases} \alpha : \text{ counting index for location within output after pooling } \left(\alpha = 1,\ldots,N_{pooling}^{\eta}\right) \\ N_{pooling}^{\eta} : \text{ size of the output after pooling for a loop iteration } \eta \\ L_{pooling} : \text{ length of the pooling window} \end{cases}$$

2D Definition of Convolution

- 2D digitalized convolution are widely used in the convolutional neural network.



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input

1	0	1
0	1	0
1	0	1

Filter / Kernel

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

Input x Filter

4		

Feature Map

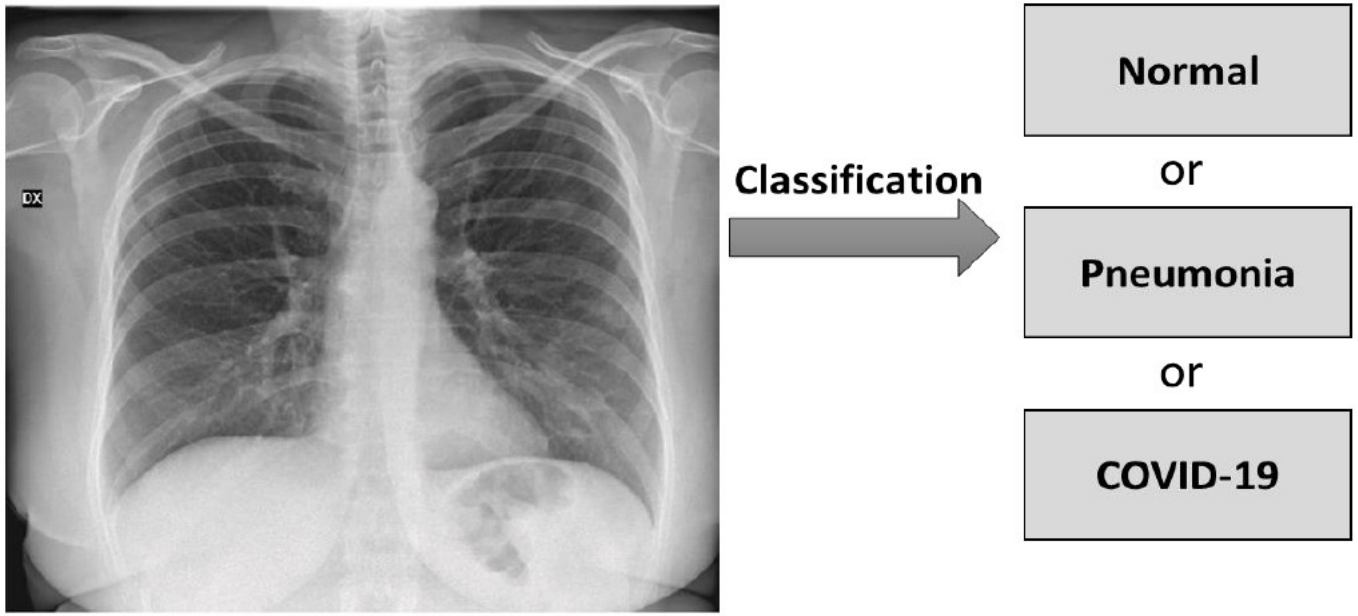
Application: COVID-19 detection from chest x-ray images of patients

- Coronavirus 2019 (COVID-19) became a pandemic caused a disastrous impact on the world.
- It is very important to accurately detect the positive cases at early stage to treat patients and prevent the further spread of the pandemic.
- Chest X-ray imaging has critical roles in early diagnosis and treatment of COVID19.
- Automated toolkits for COVID-19 diagnosis based on radiology imaging techniques such as X-ray imaging can overcome the issue of a lack of physician in remote villages and other underdeveloped regions.

https://news.uchicago.edu/story/researchers-develop-ai-help-diagnose-understand-covid-19-lung-images

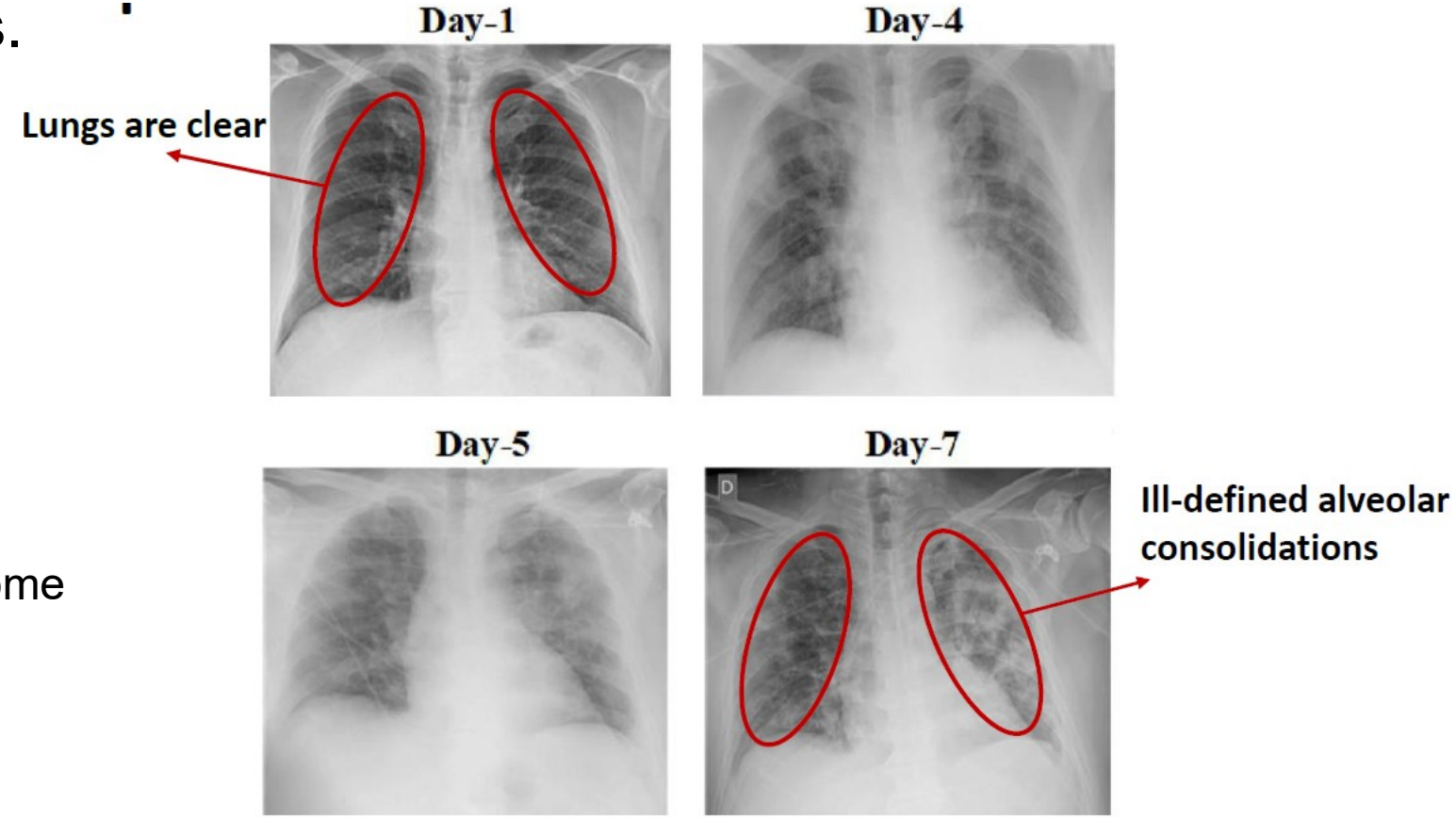
Classification Example

- It is a classification problem where the outputs are discrete (e.g., yes or no) instead of continuous values.
- Whether a patient has been infected with COVID-19 based on their chest X-ray image?



Chest x ray images of a 50-year-old COVID-19 patient over a week

- AI-assisted automatic diagnosis can automatically extract important features (such as alveolar consolidations) from images to accurate diagnosis for clinicians.



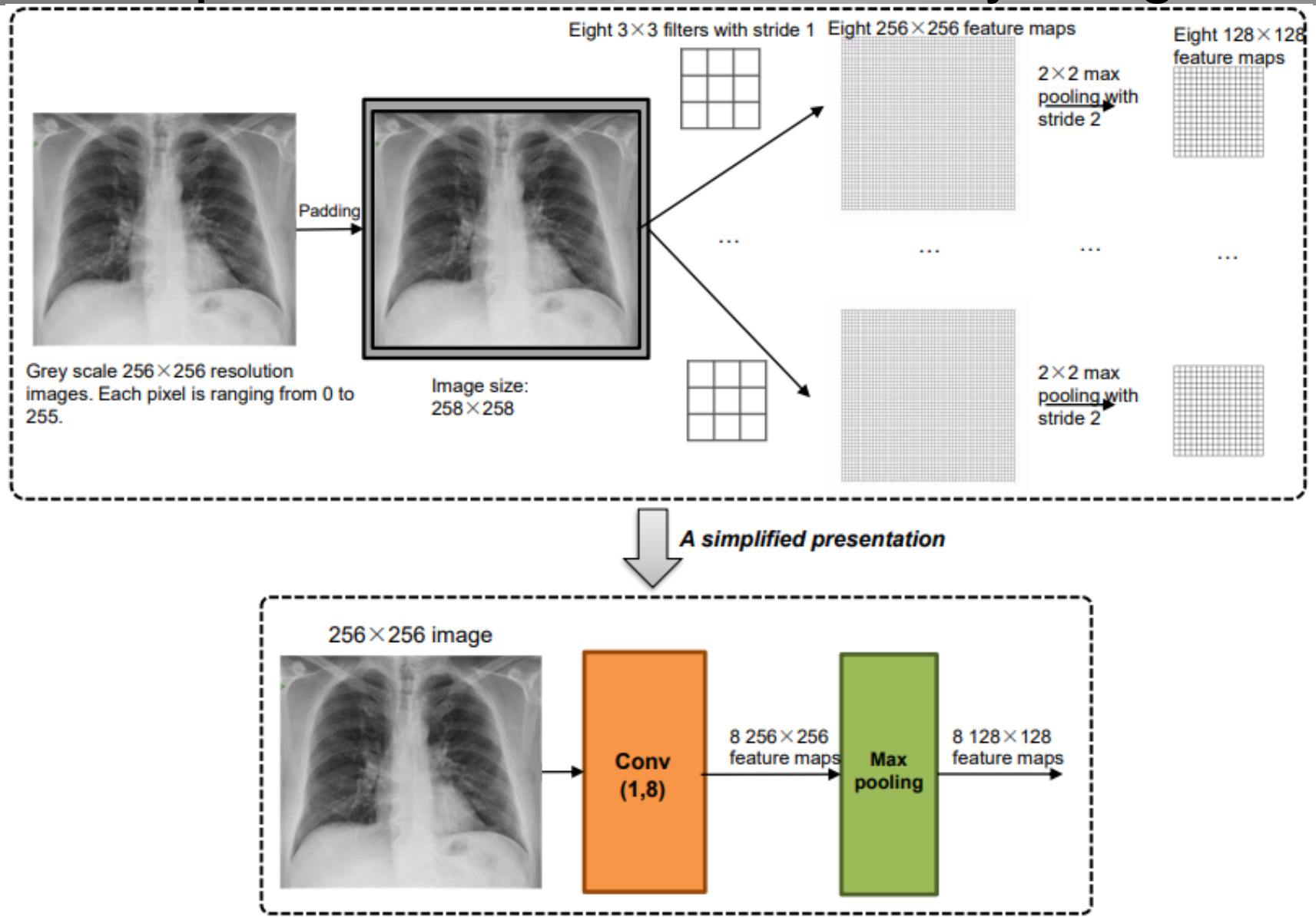
Acute Respiratory Distress Syndrome

https://radiopaedia.org/cases/covid-19-pneumonia-evolution-over-a-week-1

Data Collection

- X-ray database
  - COVID-19 X-ray image database was generated and collected by Cohen JP [22]
    - https://github.com/ieee8023/COVID-chestxray-dataset
  - Another chest X-ray image database was provided by Wang et al. [23]
    - https://nihcc.app.box.com/v/ChestXray-NIHCC
    - https://paperswithcode.com/dataset/chestx-ray8
- Images
  - 125 X-ray images of COVID-19 patients (43 female, 82 male, average age of approximately 55 years)
  - 500 X-ray images pneumonia patients
  - 500 X-ray images of patients with no-findings
- Training (900), Validation (225: 28 COVID-19 cases, 88 pneumonia cases, and 109 no-finding cases)
- Five-fold cross-validation is used to evaluate the model performance

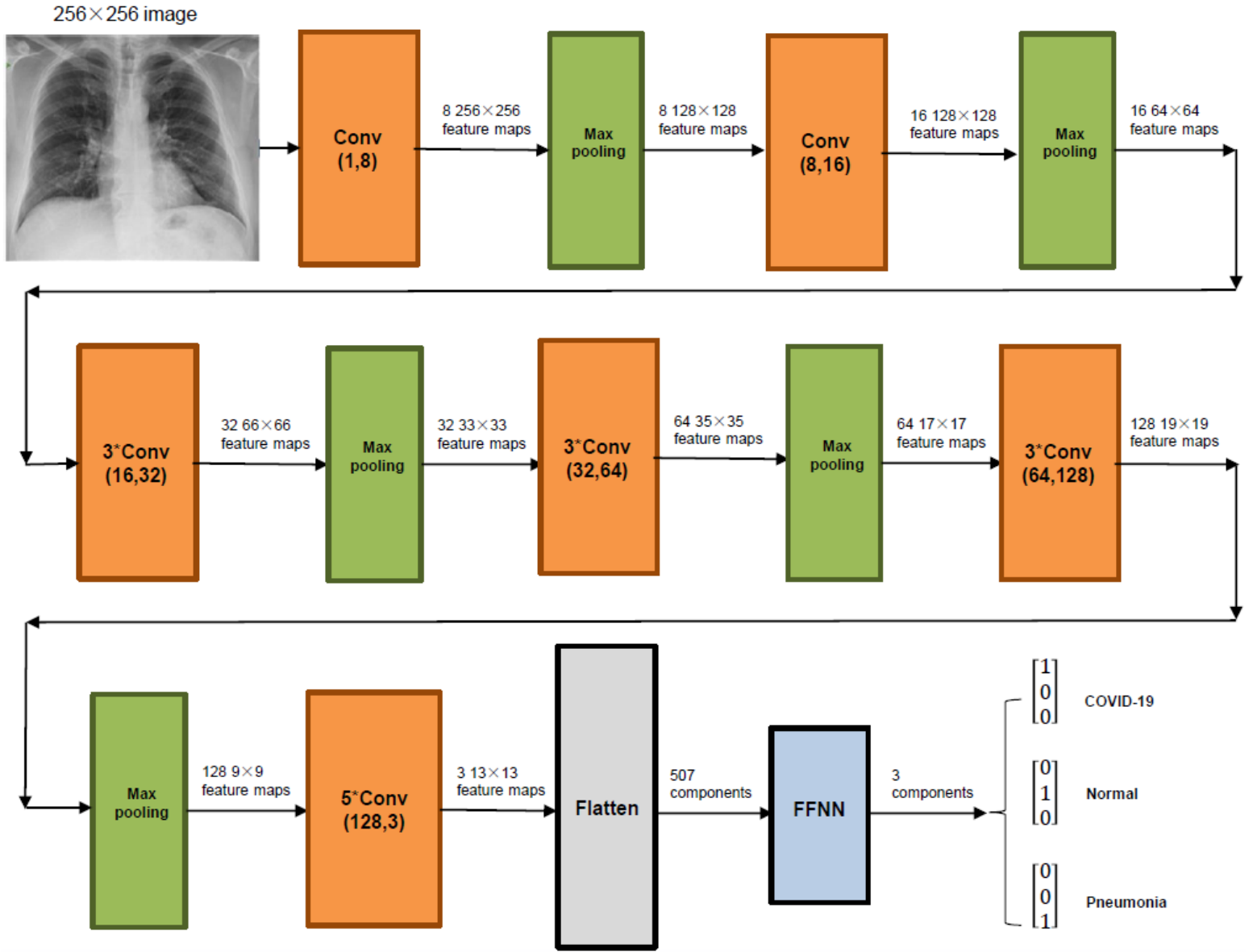
Automated detection of COVID-19 cases using deep neural networks with X ray images



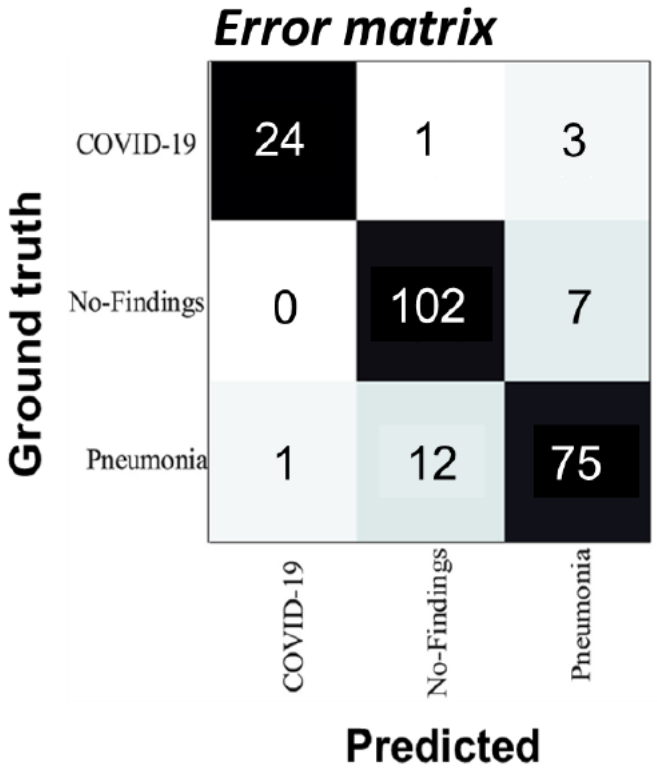
Architecture of the CNN model (1)

- Darknet-19 model [25]
- Leaky rectified linear unit (leaky ReLU)
- 17 convolutional layers and 5 max pooling layers with different filter numbers, sizes, and stride values
  - Padding: 256 x 256 resolution → 258 x 258
  - Eight 3 x 3 filters with stride 1 → eight 256 x 256 feature maps
    - Allows different features from the input image to be extracted
    - The values in the filters will be obtained during the data training process
  - Eight 2 x 2 max pooling operators with stride 2 --> size of feature maps can be reduced to 128 x 128
- 1,164,434 parameters, Adam optimizer, learning rate: $3x10^{-3}$

Architecture of the CNN model (2)

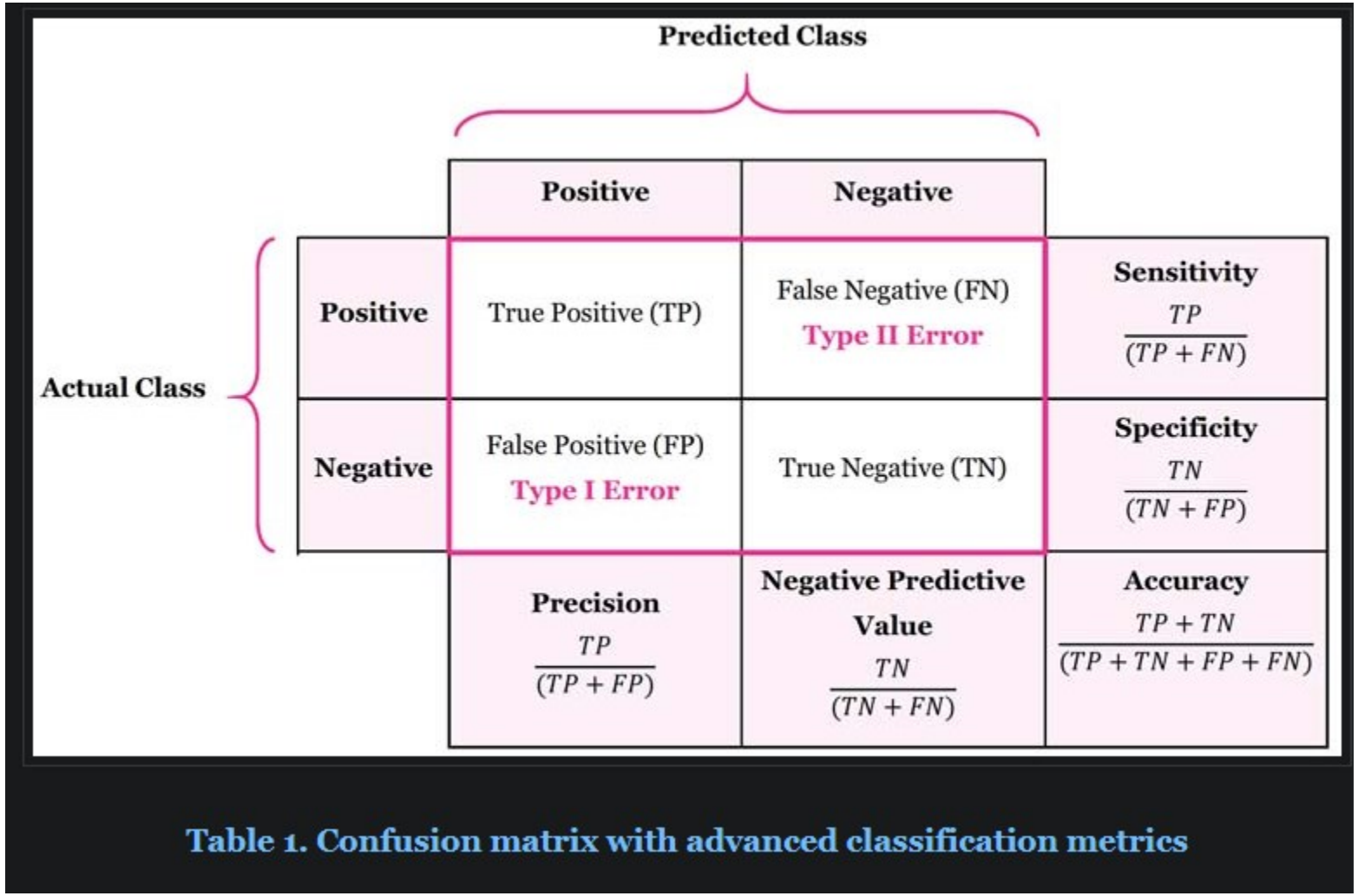


Error matrix allows visualization of the performance of the deep neural networks



- In total 225 images in test set are used to present the performance of the model
- Classification accuracy for COVID-19 is 24/28= 85.7%
- Classification accuracy for Normal is 102/109= 94.6%
- Classification accuracy for Pneumonia is 75/88= 85.2%
- The patients can seek a second opinion by the deep learning system, which significantly reduces waiting time and alleviates clinicians' workload.

Confusion Matrix(오차행렬)



		Predicted Class: Positive	Predicted Class: Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

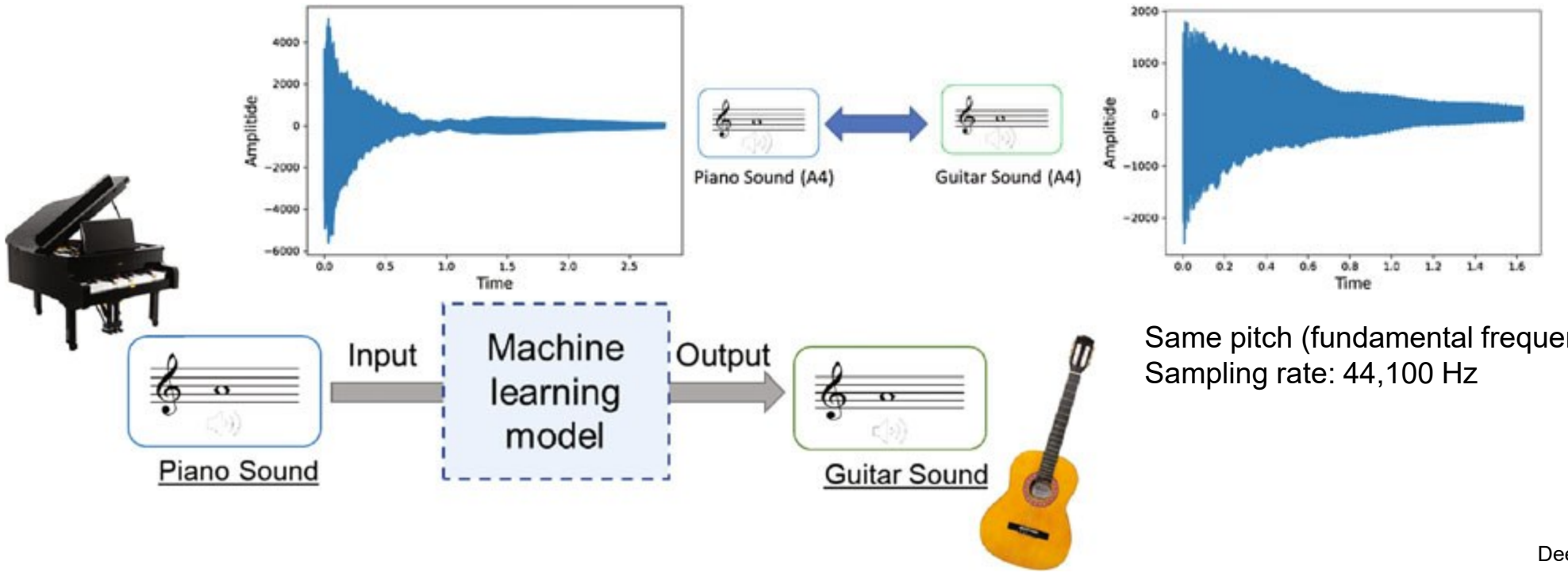
Table 1. Confusion matrix with advanced classification metrics

- Accuracy(정확도)
- Precision(정밀도)
- Recall(재현도), Sensitivity Rate, True Positive Rate
- F1 Score

$$\text{F1 Score} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

6.4 Musical Instrument Sound Conversion

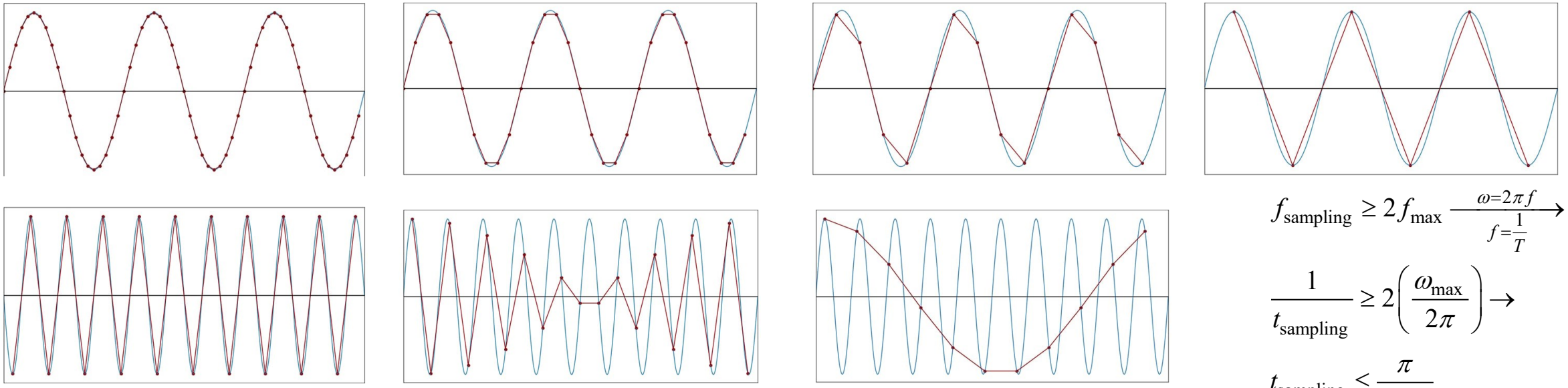
- Use mechanistic data science to convert piano sounds to guitar sounds
- Dataset: eight pairs of notes (notes A4, A5, B5, C5, C6, D5, E5, G5) performed by a piano and by a guitar (apronus.com)
  - Recorded duration: (piano) 2.8 s (guitar) 1.6 s
  - Dimension: (piano) 120,000 (44.1 kHz x 2.8 s), (guitar) 72,000 (44.1 kHz x 1.6 s)



Same pitch (fundamental frequency)
Sampling rate: 44,100 Hz

Nyquist-Shannon Theorem

- If a system uniformly samples an analog signal at a rate that exceeds the signal's highest frequency by at least a factor of two, the original analog signal can be perfectly recovered from the discrete values produced by sampling.
- The human hearing range is 20~20,000 Hz. Therefore, a sufficient sampling rate for sound file should be 40,000 Hz
  - A common sampling time interval in music is 44,100Hz. It is higher than 40,000Hz

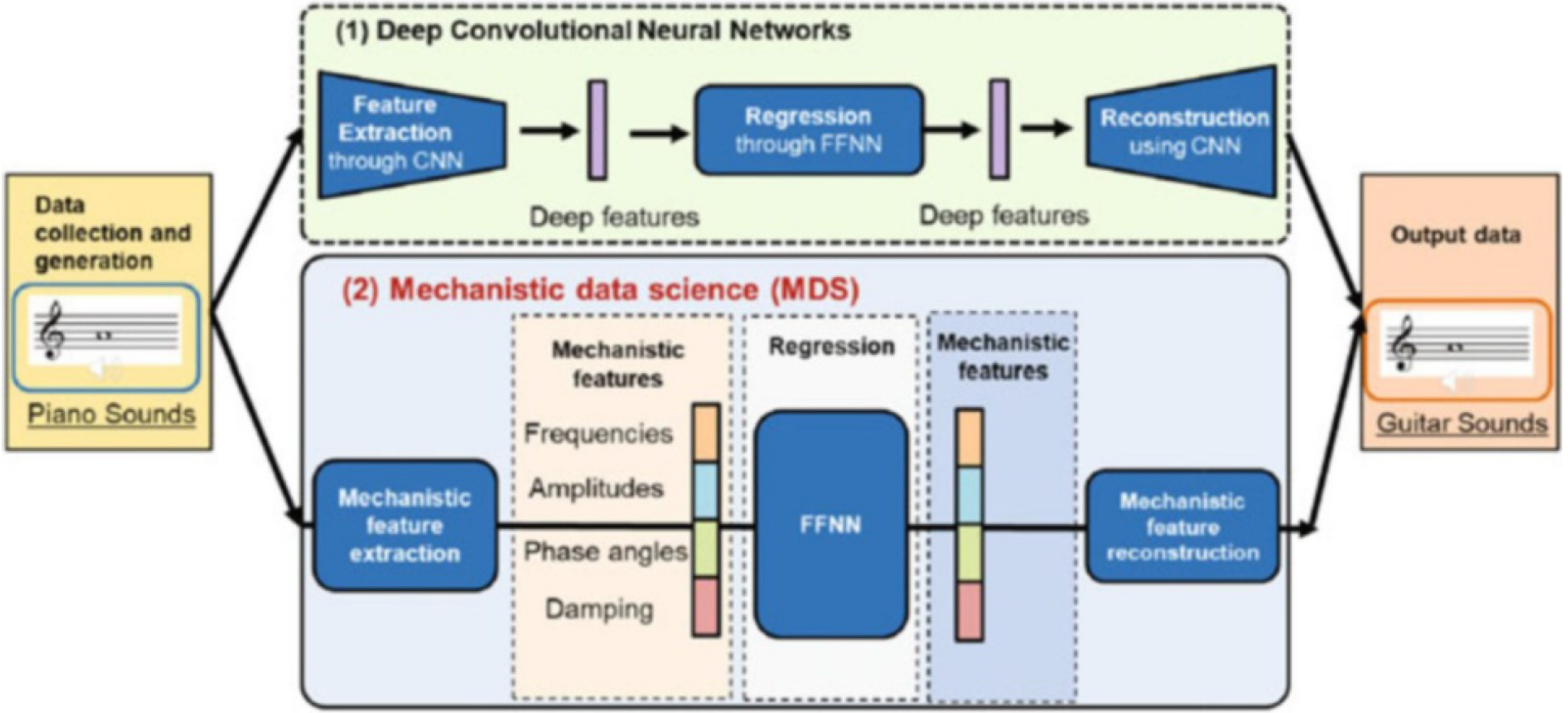


$$f_{\text{sampling}} \geq 2 f_{\max} \xrightarrow[f=\frac{1}{T}]{\omega=2\pi f}$$

$$\frac{1}{t_{\text{sampling}}} \geq 2\left(\frac{\omega_{\max}}{2\pi}\right) \rightarrow$$

$$t_{\text{sampling}} \leq \frac{\pi}{\omega_{\max}}$$

Three Solutions



(3) PCA + NN → Ch.7

Pure CNN Analysis

- Input and output are high-dimensional time-amplitude curves
- Drawback of this strategy is the high number of trainable parameters involved in the CNN and FFNN structures
- For some applications where the amount of data is small or the quality of data is low, the performance of the CNN is limited

Mechanistic Data Science Analysis

- Instead of extracting deep features by CNN, mechanistic data science extracts mechanistic features based on the underlying scientific principles
- Each signal (high-dimensional sound curve) can be simplified to a set of sine functions, where the mechanistic features are the frequencies, damping coefficients, amplitudes, and phase angles
  - set of mechanistic features with a physical meaning
- Short Time Fourier Transform (STFT) and a regression to fit a mechanistic model, such as a spring-mass-damper model
- Hyperparameters involved in the model can be significantly decreased from thousands to dozens in this manner, which reduces the amount of training data required

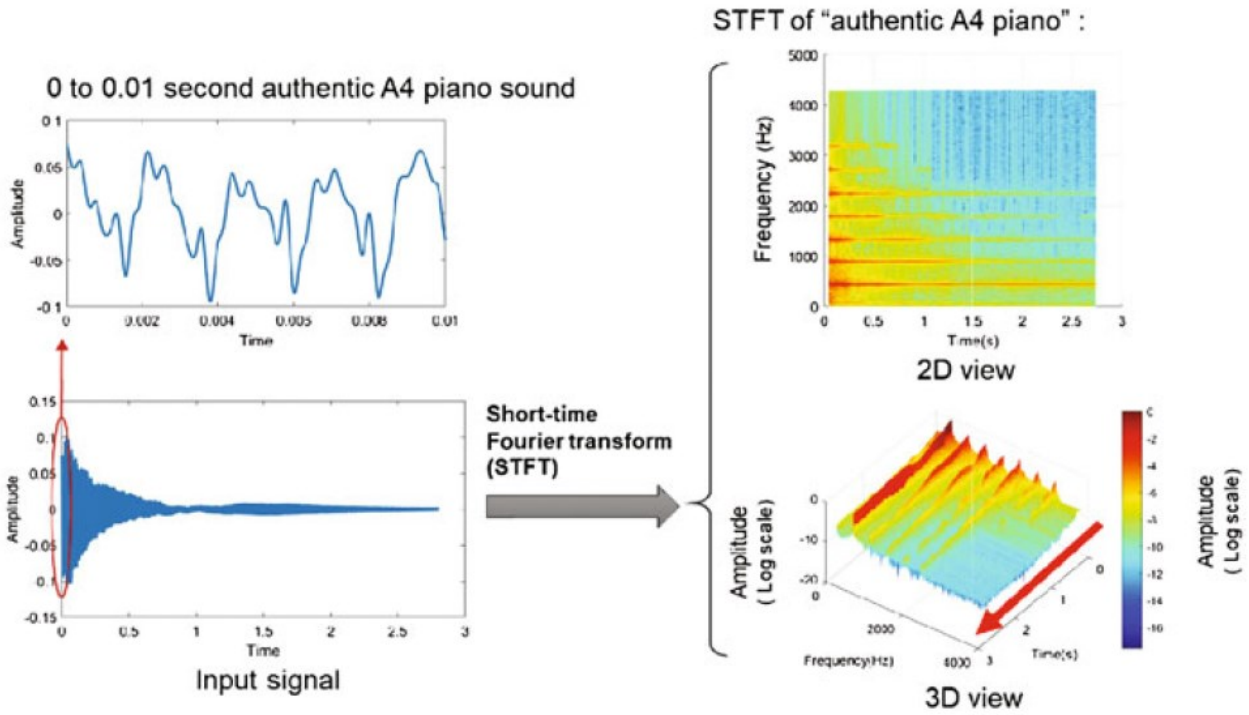


Fig. 6.38 An A4 piano sound signal and its STFT result (2D and 3D)

Table 6.6 Optimal coefficients to represent the authentic A4 piano sound

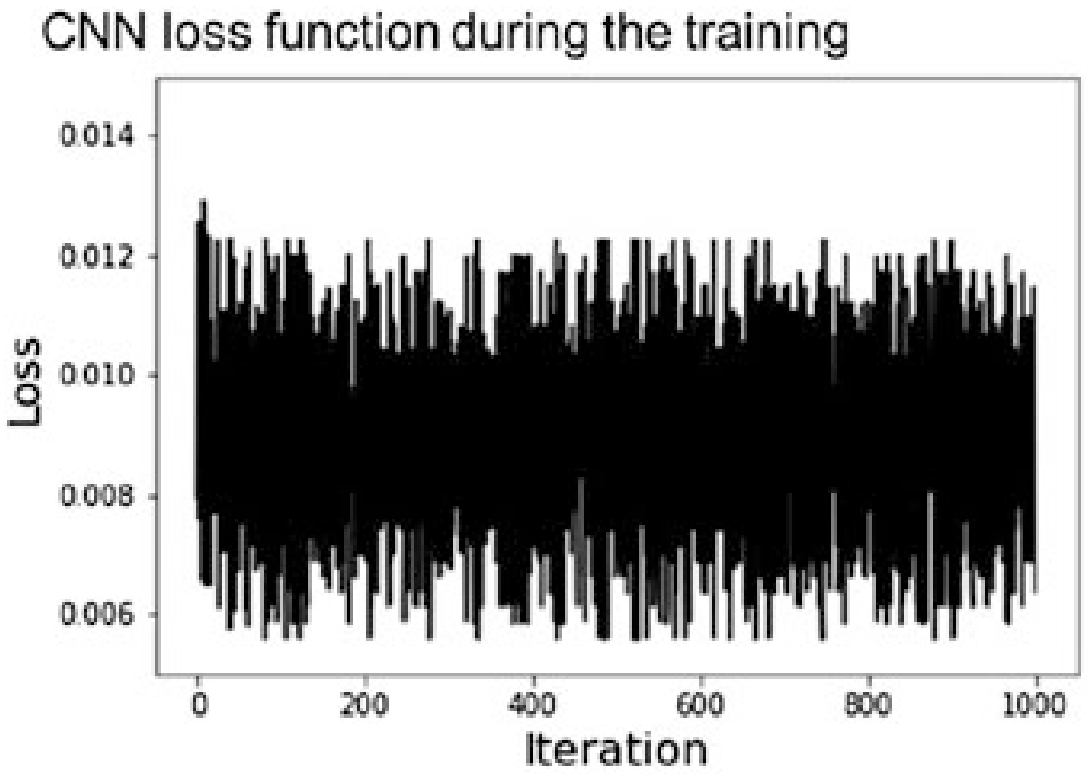
Type	Frequency (Hz)	Initial amplitudes	Damping coefficients	Phase angle (rad)
Fundamental	4.410E+02	1.034E-01	3.309E+00	6.954E-01
Harmonics	8.820E+02	1.119E-02	1.844E+00	7.202E-01
	1.323E+03	6.285E-03	5.052E+00	3.469E-01
	1.764E+03	7.715E-04	2.484E+00	5.170E-01
	2.205E+03	1.455E-03	8.602E+00	5.567E-01
	2.646E+03	5.130E-04	1.198E+01	1.565E-01
	3.087E+03	1.899E-04	8.108E+00	5.621E-01
	3.528E+03	3.891E-05	3.282E+00	6.948E-01

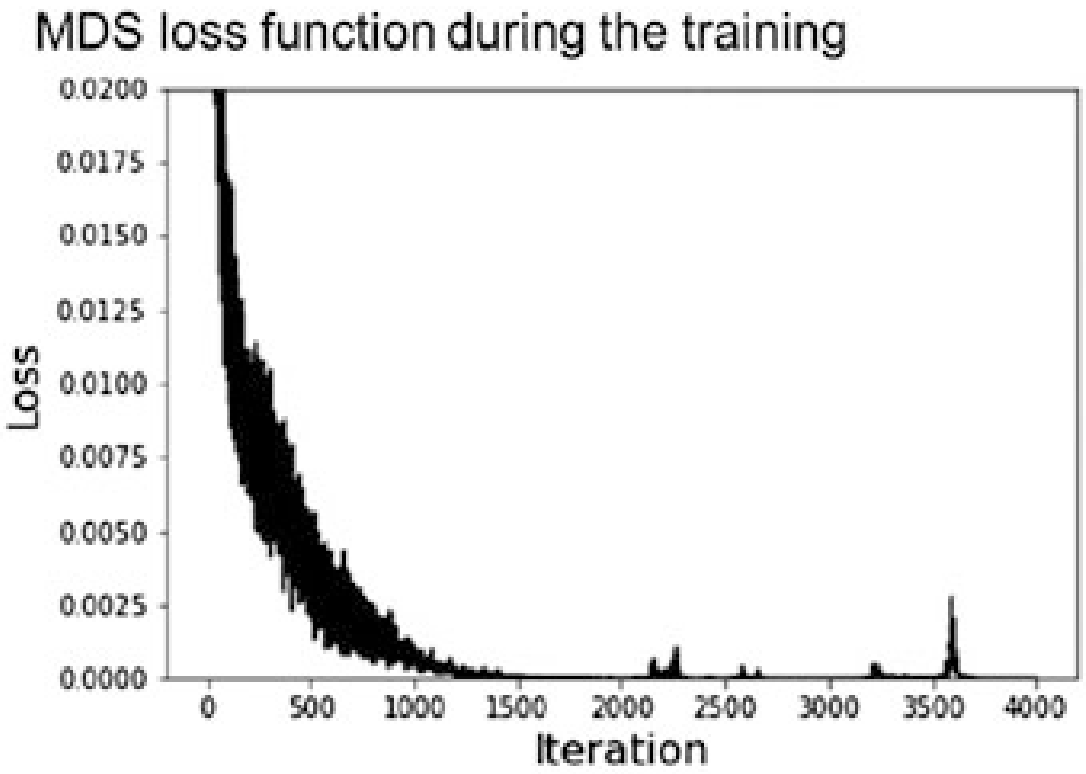
Table 6.7 Optimal coefficients to represent the authentic A4 guitar sound

Type	Frequency (Hz)	Initial amplitudes	Damping coefficients	Phase angle (rad)
Fundamental	4.400E+02	1.649E-02	1.287E+00	9.798E-01
Harmonics	8.800E+02	8.022E-03	1.865E+00	2.848E-01
	1.320E+03	2.551E-03	2.176E+00	5.950E-01
	1.760E+03	5.454E-03	1.100E+00	9.622E-01
	2.200E+03	5.523E-03	3.346E+00	1.858E-01
	2.640E+03	6.742E-03	2.504E+00	1.930E-01
	3.080E+03	7.643E-04	1.666E+00	3.416E-01
	3.520E+03	9.748E-04	2.609E+00	9.329E-01

CNN vs. MDS

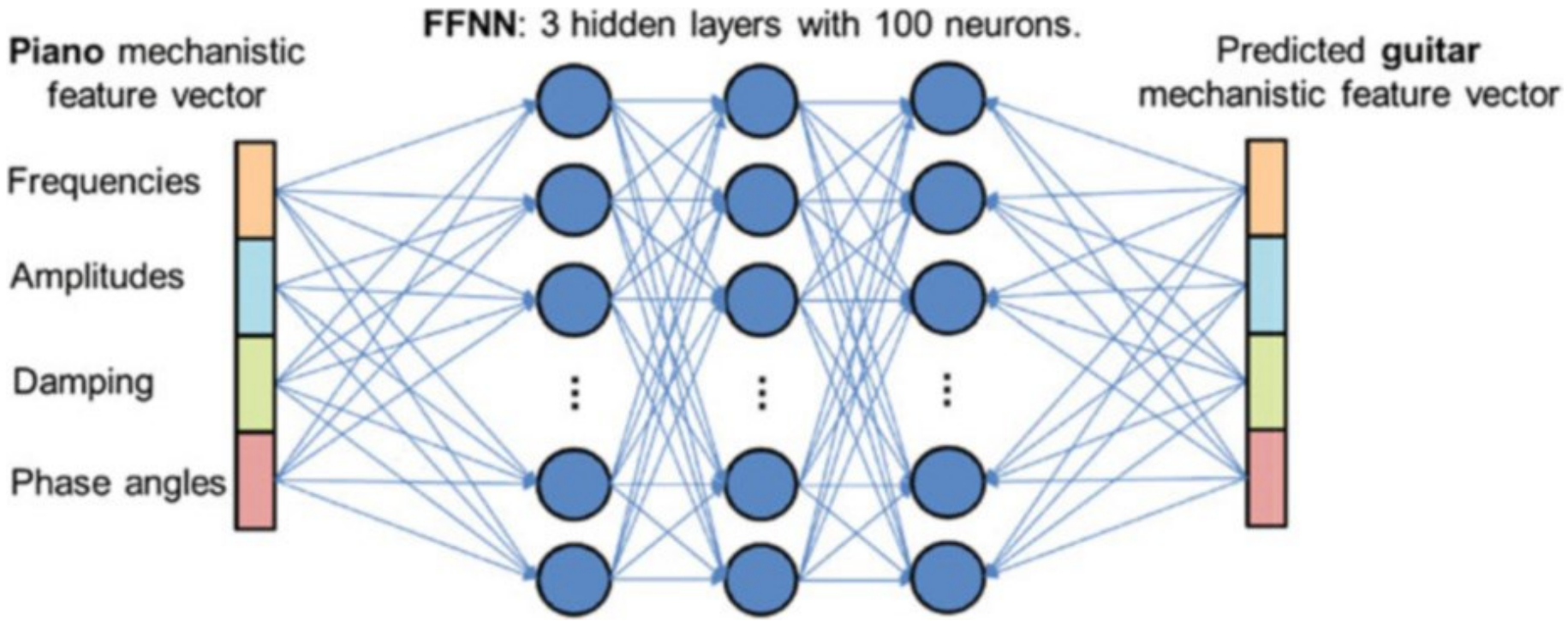
- The loss function of the CNN is volatile and does not converged to zero
- Training a CNN model requires the number of data points to be larger than the dimension of input signal or image
  - In this case, the dimension of input is a million



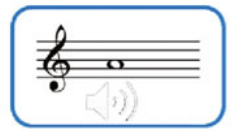


FFNN structure with reduced mechanistic features

- Dataset: 4(features) x 8(sets) = 32 for one note
- Activation function: tanh
- Loss function: MSE

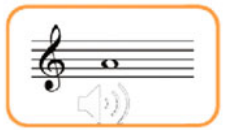


Results of reconstructing a single guitar key A4 from a piano key as input

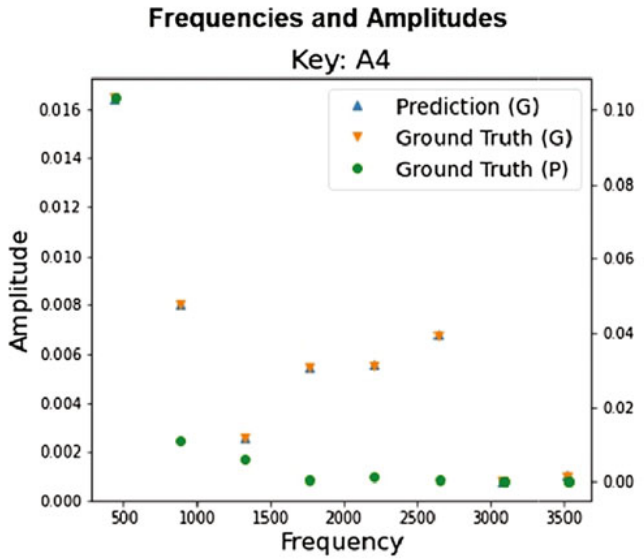


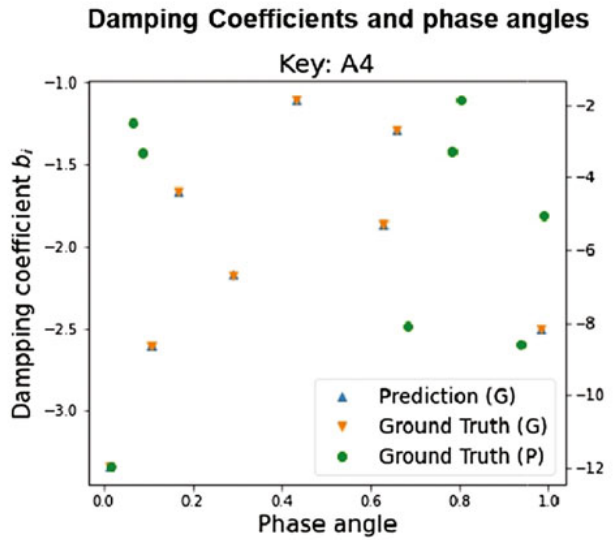
Authentic
Piano Sound (A4)

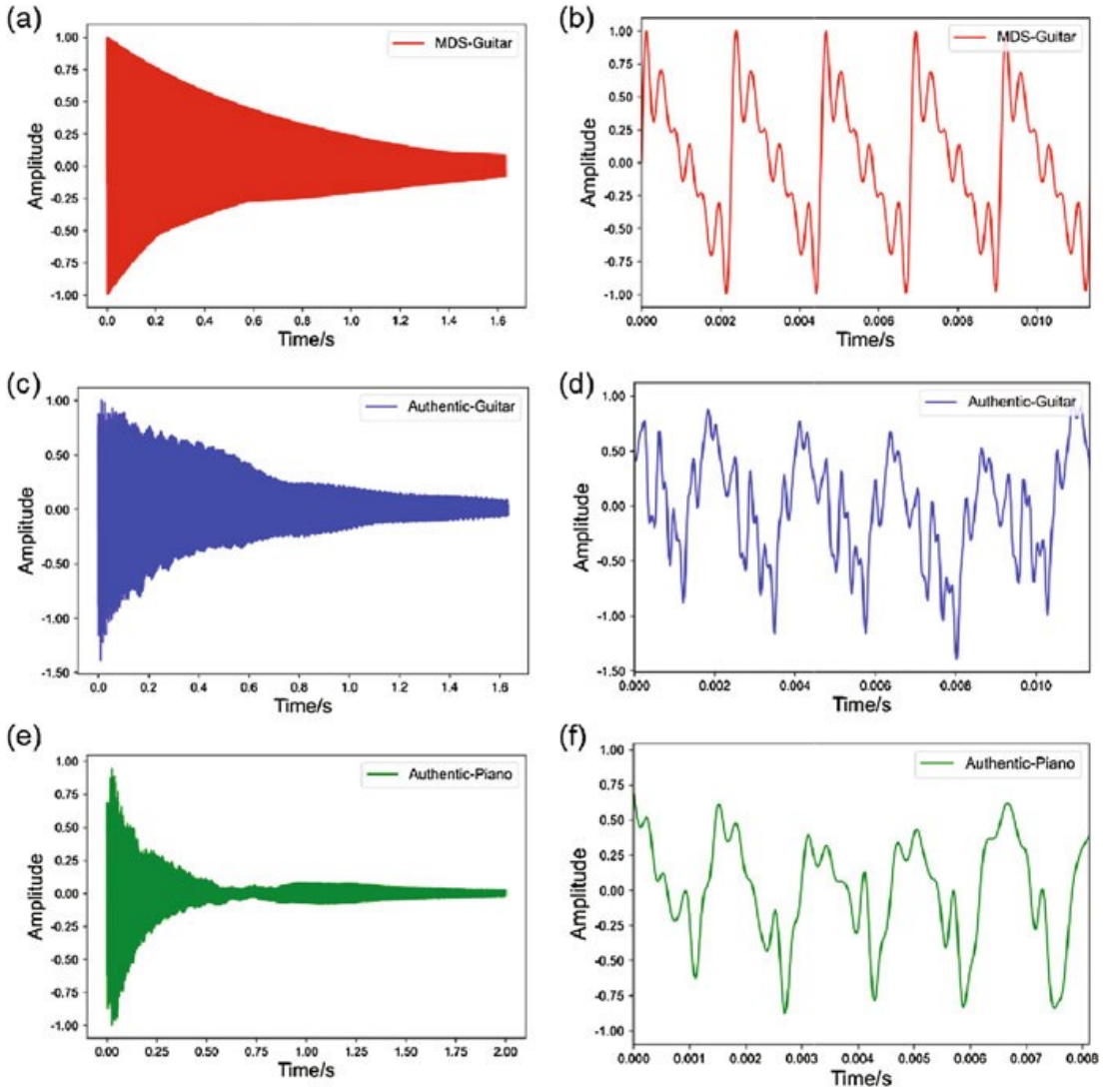
Authentic
Guitar Sound (A4)



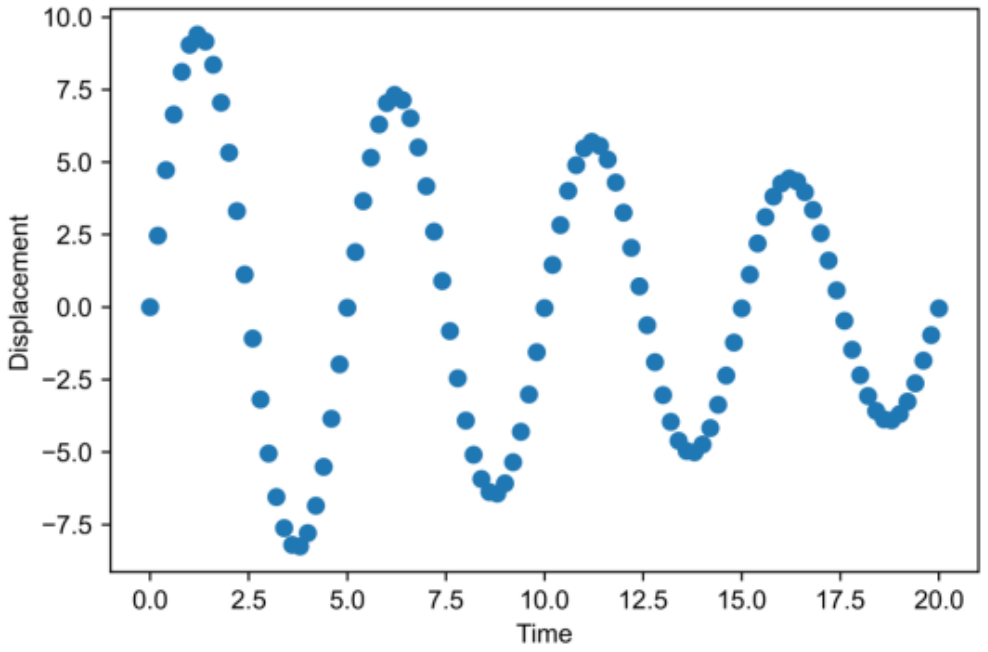
MDS Generated
Guitar Sound (A4)







Use PyTorch to fit spring-mass data (1)



Train a FFNN

Use PyTorch to fit spring-mass data (2)

```
%matplotlib inline
import numpy as np
import torch
import torch.optim as optim
import torch.nn as nn
import matplotlib.pyplot as plt
import math

torch.set_printoptions(edgeitems=2, linewidth=75)
```

Import necessary libraries

```
t=np.linspace(0, 20, 101)
z =10*np.sin(2*3.14/5*t)*np.exp(-5e-2*t)

z = torch.FloatTensor(z).unsqueeze(1) # <1>

t = torch.FloatTensor(t).unsqueeze(1) # <1>
```

Create data

```
from matplotlib import pyplot as plt

t_range = torch.arange(0., 21).unsqueeze(1)

fig = plt.figure(dpi=600)
plt.xlabel("Time")
plt.ylabel("Displacement")
plt.plot(t.numpy(), z.numpy(), 'o')
```

Plot data

Use PyTorch to fit spring-mass data (2)

```
n_samples = t.shape[0]
n_val = int(0.2 * n_samples)
print(n_val)
shuffled_indices = torch.randperm(n_samples)

train_indices = shuffled_indices[0:n_samples-n_val]
val_indices = shuffled_indices[n_samples-n_val:n_samples]

print(train_indices, val_indices)
print(shuffled_indices)
```

Separate data into 80% training data and 20% test data

```
t_train = t[train_indices]
z_train = z[train_indices]

t_val = t[val_indices]
z_val = z[val_indices]

t_n_train = 0.1 * t_train
t_n_val = 0.1 * t_val
```

Use PyTorch to fit spring-mass data (3)

```
def training_loop(n_epochs, optimizer, model, loss_fn, t_train, t_val,
                  z_train, z_val):
    for epoch in range(1, n_epochs + 1):
        z_p_train = model(t_train) # <1>
        loss_train = loss_fn(z_p_train, z_train)

        z_p_val = model(t_val) # <1>
        loss_val = loss_fn(z_p_val, z_val)

        optimizer.zero_grad()
        loss_train.backward() # <2>
        optimizer.step()

        if epoch == 1 or epoch % 1000 == 0:
            print(f"Epoch {epoch}, Training loss {loss_train.item():.4f},"
                  f" Validation loss {loss_val.item():.4f}")
```

Define training process

```
def loss_fn(z_p, z):
    squared_diffs = (z_p - z)**2
    return squared_diffs.mean()
```

Define loss function

Use PyTorch to fit spring-mass data (4)

```
from collections import OrderedDict
neuron_count = 100
seq_model = nn.Sequential(OrderedDict([
    ('hidden_linear', nn.Linear(1, neuron_count )),
    ('hidden_activation', nn.Tanh()),
    ('output_linear', nn.Linear(neuron_count , 1))
]))

seq_model
```

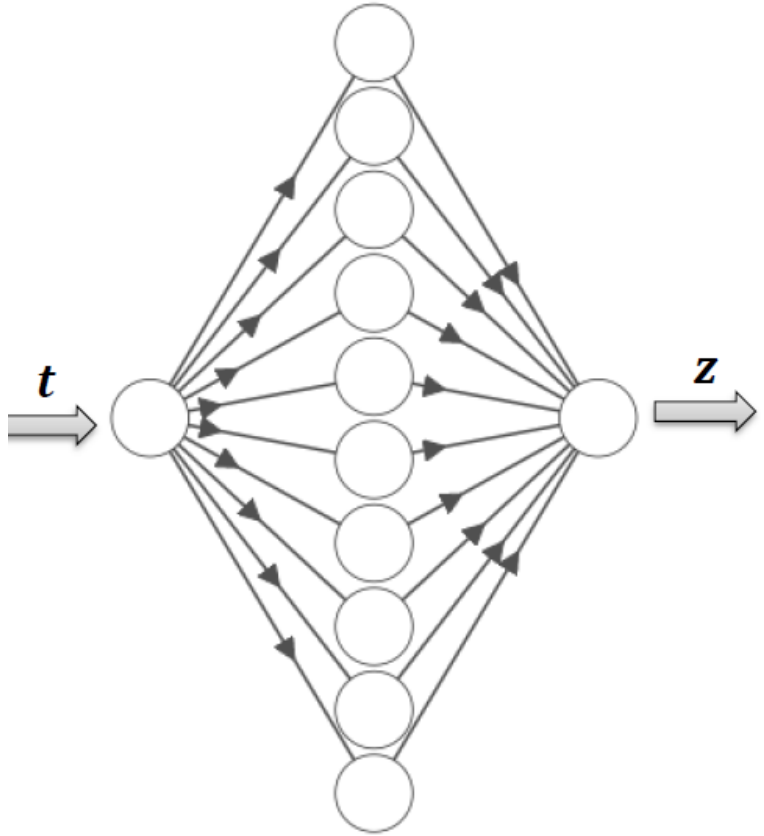
```
optimizer = optim.SGD(seq_model.parameters(), lr=1e-2)

training_loop(
    n_epochs = 10000,
    optimizer = optimizer,
    model = seq_model,
    loss_fn = nn.MSELoss(),
    t_train = t_n_train,
    t_val = t_n_val,
    z_train = z_train,
    z_val = z_val)

print('output', seq_model(t_n_val))
print('answer', z_val)
```

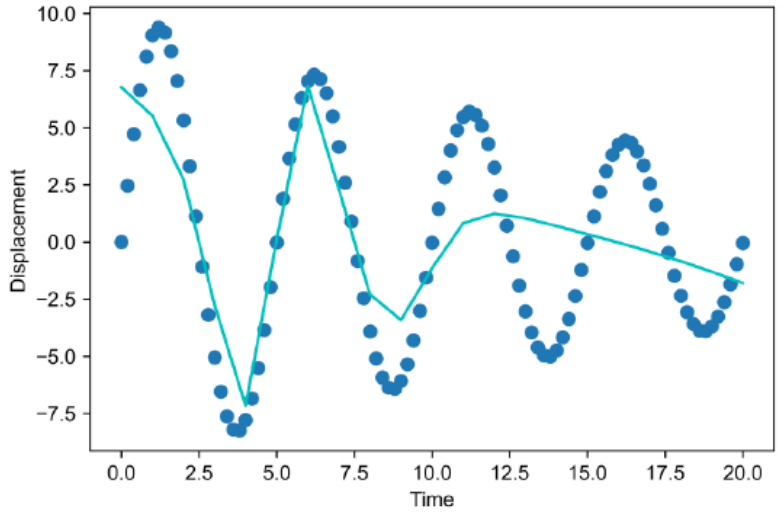
Define network structure

1 layer with 100 neurons

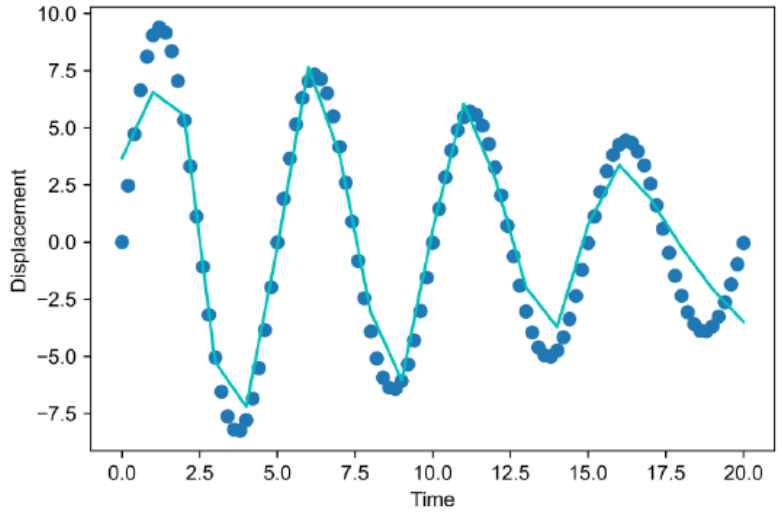


Use PyTorch to fit spring-mass data (5)

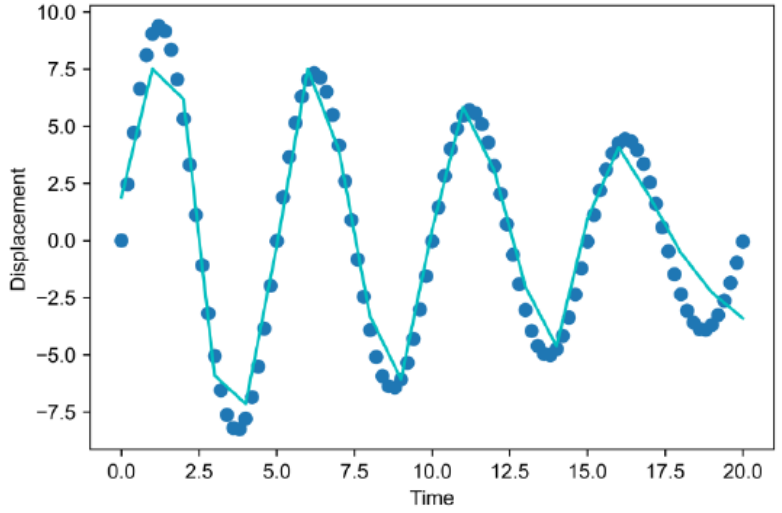
Epoch 10K



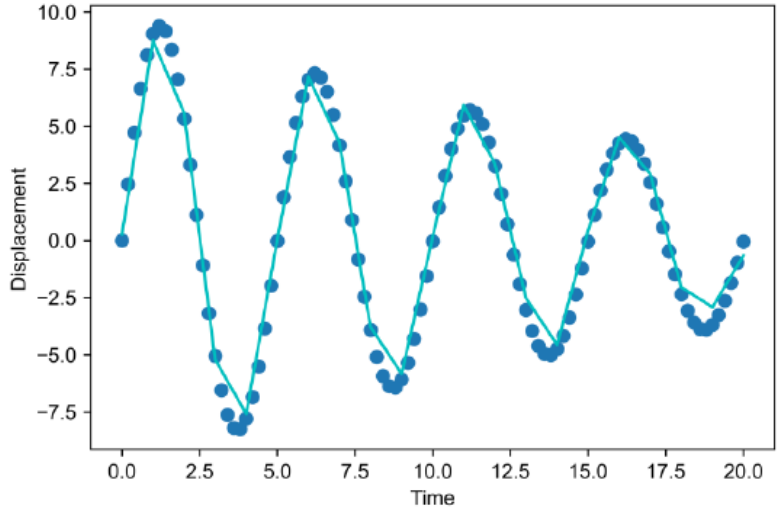
Epoch 20K



Epoch 30K

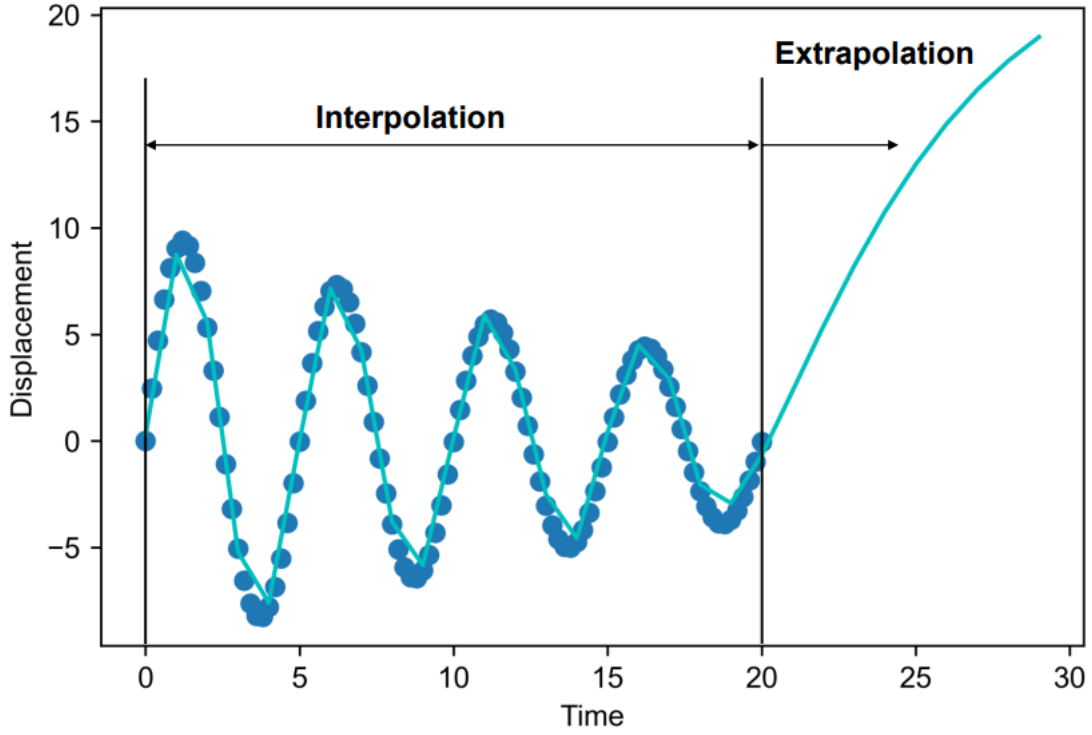


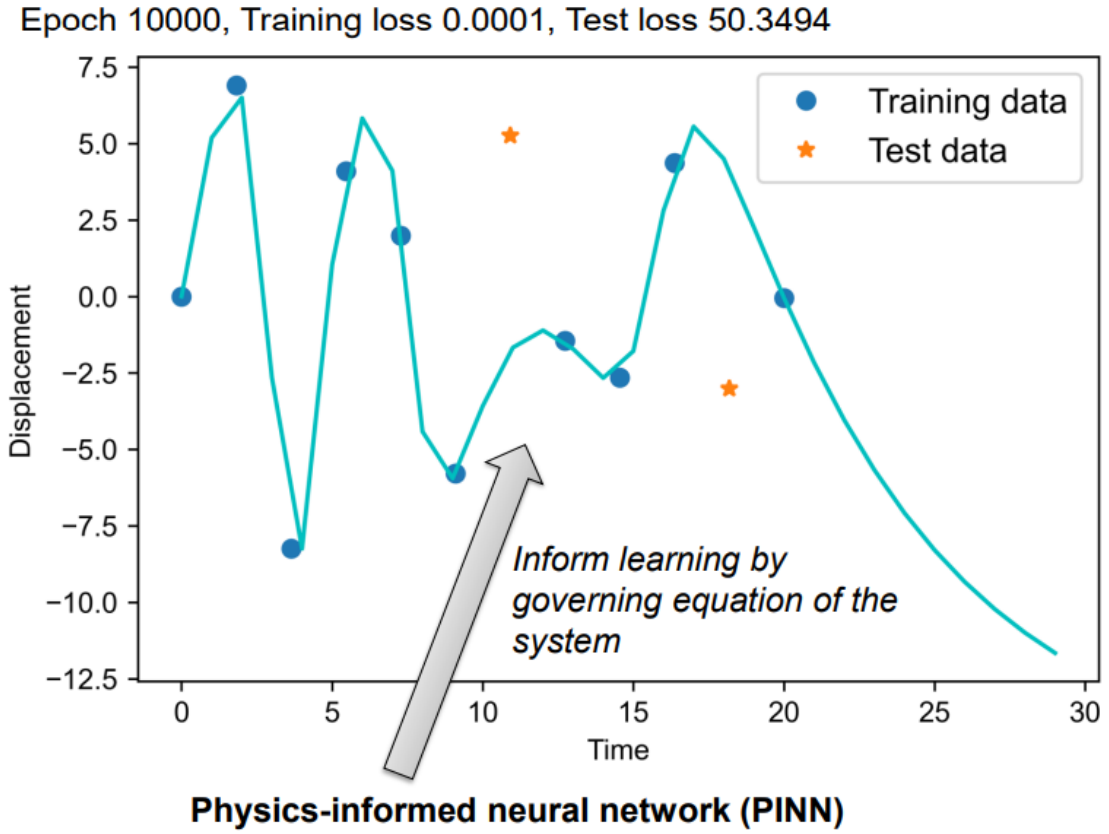
Epoch 100K



Limitations of NN

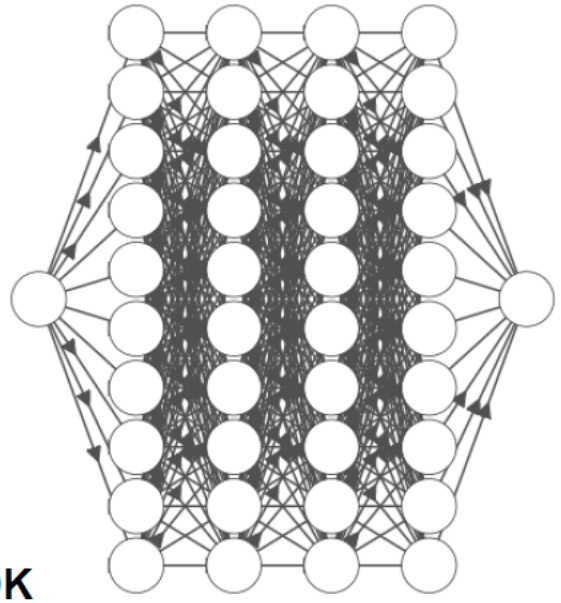
- Good at interpolation, not extrapolation
- May be overfit if there are not enough data





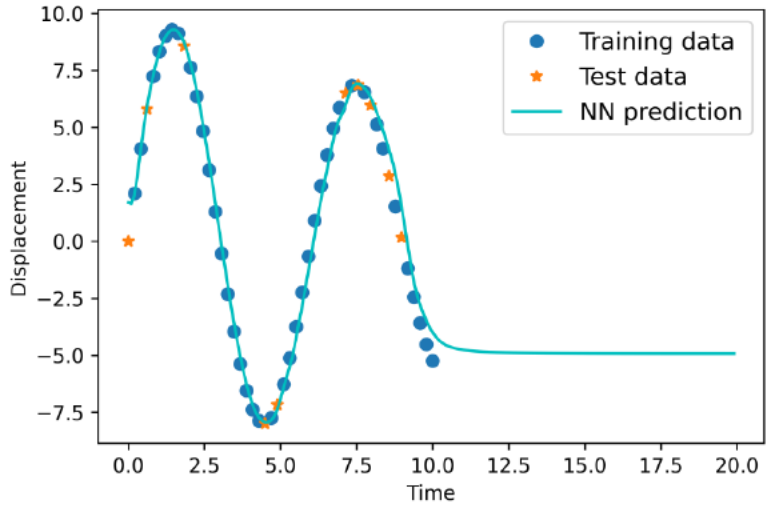
Physics-informed neural network (PINN)

NN:



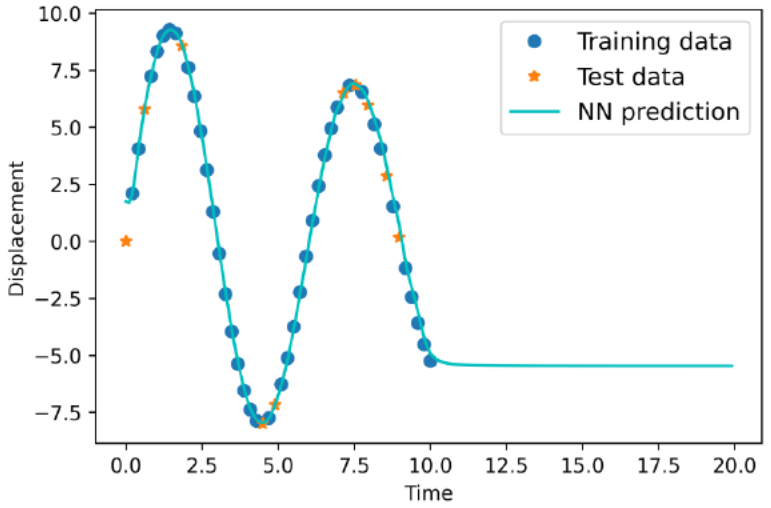
4 hidden layer
(100 neurons per layer)

Epoch 10K



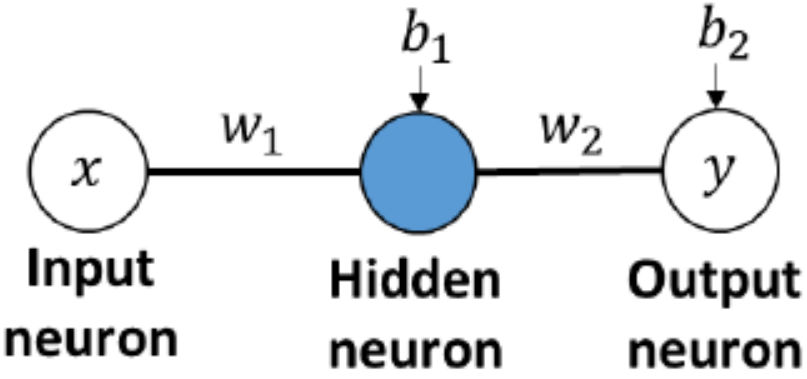
Epoch 20K

Training loss MSE 0.0365
Validation loss MSE 0.5312



Automatic Differentiation

- A trained NN model is differentiable



$$y = \mathcal{A}(w_2 \mathcal{A}(w_1 x + b_1) + b_2)$$

$$A(\ ) = \tanh(\ )$$

$$y = \tanh\left(w_2 \tanh\left(w_1 x + b_1\right) + b_2\right)$$

$$\frac{d}{dx}\left(\tanh(x)\right) = 1 - \tanh^2(x)$$

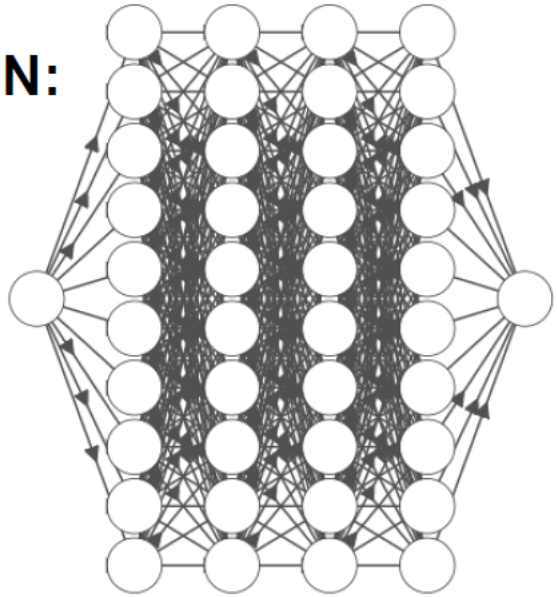
$$\frac{dy}{dx} = \left[1 - \tanh^2\left(w_2 \tanh\left(w_1 x + b_1\right) + b_2\right)\right] \frac{d}{dx}\left[w_2 \tanh\left(w_1 x + b_1\right) + b_2\right]$$

$$= \left[1 - \tanh^2\left(w_2 \tanh\left(w_1 x + b_1\right) + b_2\right)\right] w_2 \left[1 - \tanh^2\left(w_1 x + b_1\right)\right] w_1$$

- PyTorch for automatic differentiation

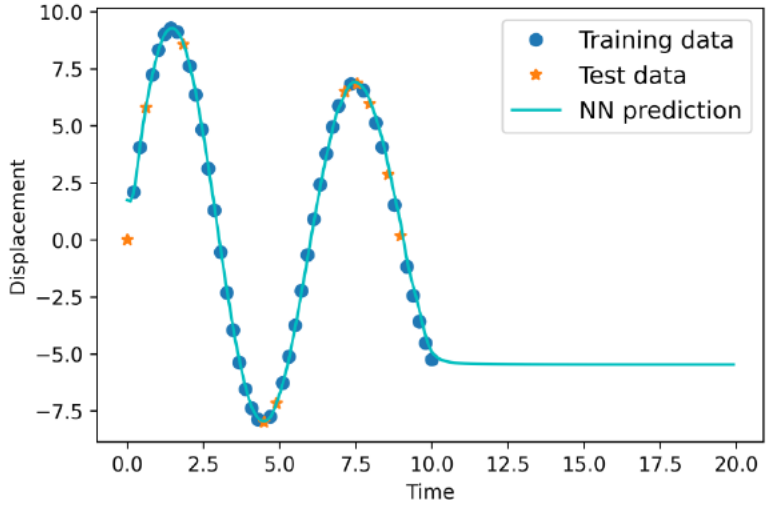
```
torch.autograd.grad(outputs, inputs, grad_outputs=None, retain_graph=None,
create_graph=False, only_inputs=True, allow_unused=False) [SOURCE]
```

NN:

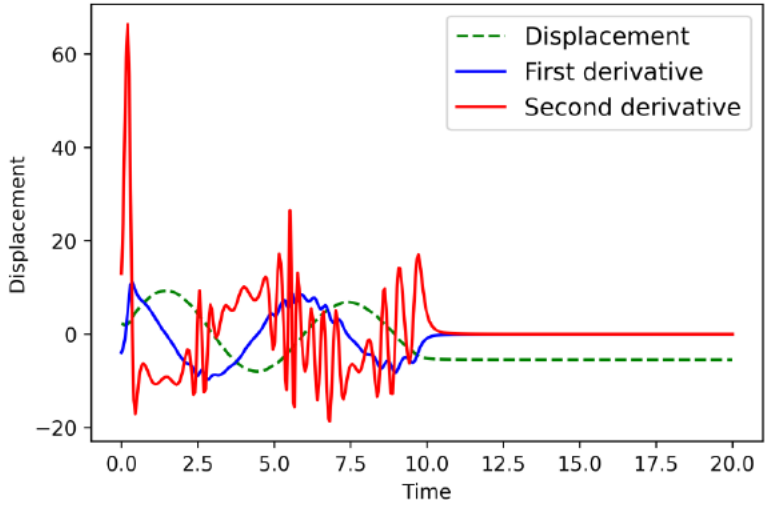


4 hidden layer
(100 neurons per layer)

Epoch 20K



Derivatives are inaccurate



Customized loss function by adding governing equation

data: $D = \left\{ \left(t_i, z_i \right) \middle| i = 1, 2, \ldots, n \right\}$

[Neural Network]

Loss function $= \frac{1}{n} \sum_{i=1}^{n} \left(z_{pi} - z_i \right)^2$: Mean Squared Error (MSE)

```
def loss_fn(z_p, z):
    squared_diffs = (z_p - z)**2
    return squared_diffs.mean()
```

[Physics-Informed Neural Network]

governing equation of spring-mass system: $\ddot{z} + \frac{c}{m} \dot{z} + \frac{k}{m} z = 0 \rightarrow \ddot{z} + 0.1\dot{z} + 1.1z = 0$

Loss function $= \underbrace{\frac{1}{n} \sum_{i=1}^{n} \left(z_{pi} - z_i \right)^2}_{\text{MSE}} + \underbrace{\frac{1}{n_1} \sum_{i=1}^{n_1} \left(\ddot{z}_{pi} + 0.1\dot{z}_{pi} + 1.1z_{pi} \right)^2}_{\text{equation error}}$

n : number of data points

n_1 : number of sampling points for derivative calculation

Loss Function for PINN: Code

```python
def loss_fn1(z_p, z):
    squared_diffs = (z_p - z)**2
    return squared_diffs.mean()

def loss_fn2(t_new):

    dzdt = torch.autograd.grad(outputs=seq_model(t_new),
    inputs=t_new,
    grad_outputs=torch.ones_like(seq_model(t_new)),
    retain_graph=True,
    create_graph=True)[0]
    dzdtt= torch.autograd.grad(outputs=dzdt,
    inputs=t_new,
    grad_outputs=torch.ones_like(dzdt),
    retain_graph=True,
    create_graph=True)[0]
    z=seq_model(t_new)
    f=abs(dzdtt+1.1*z+0.1*dzdt)

    #print(f.mean())
    return f.mean()
```

```python
t=np.linspace(0, 10, 50)
z =10*np.sin(2*3.14/6*t)*np.exp(-5e-2*t)
```

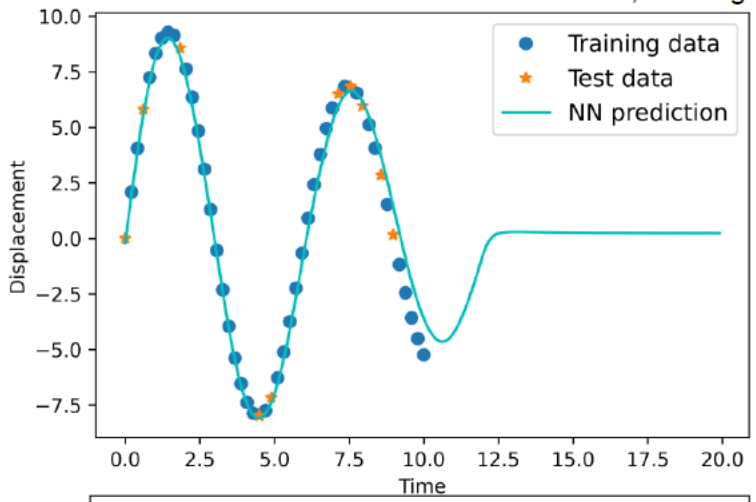
```python
t_new=np.linspace(0, 20, 1000)
```

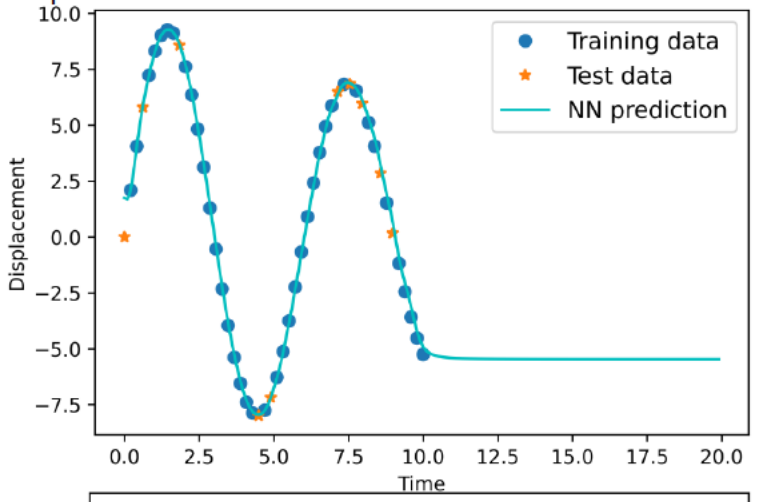
PINN vs. NN

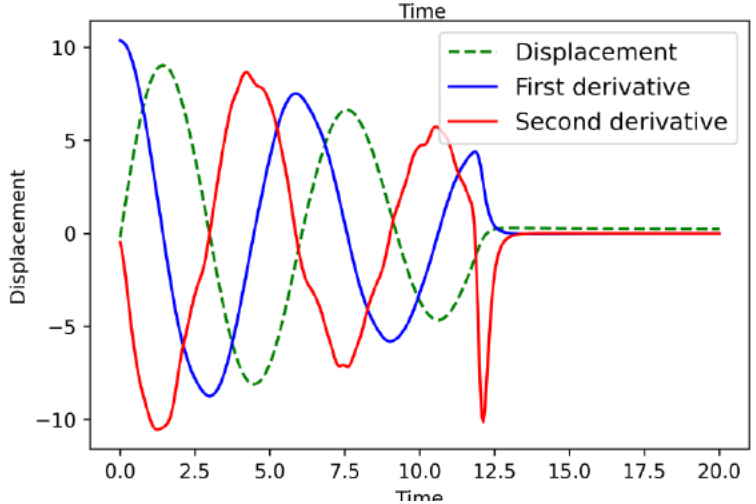
4 hidden layer (100 neurons per layer)

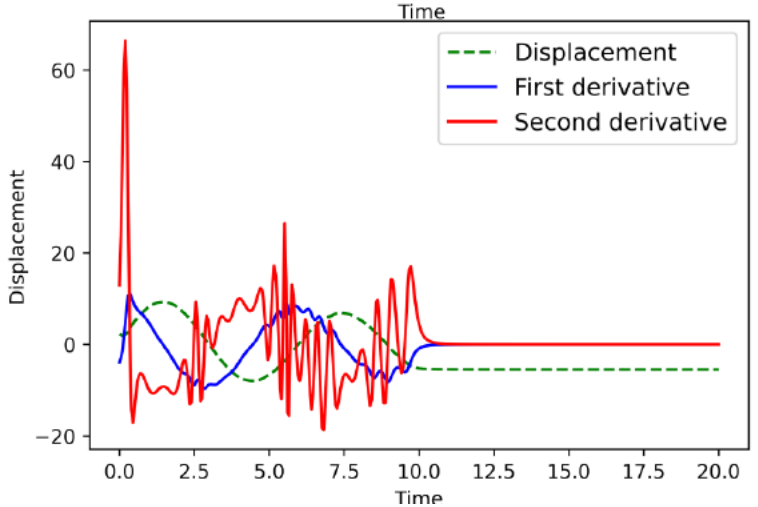
Epoch 20K Training loss MSE 0.4600, Training loss Eqn 0.6014
Validation loss MSE 0.9487, Training loss Eqn 0.6014

NN:

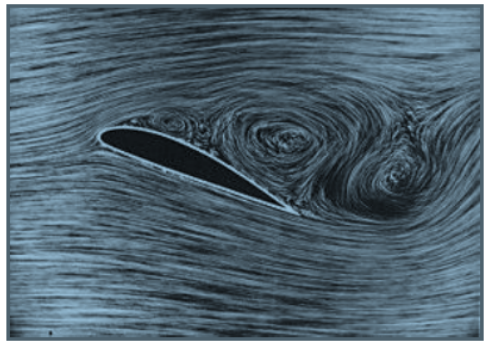


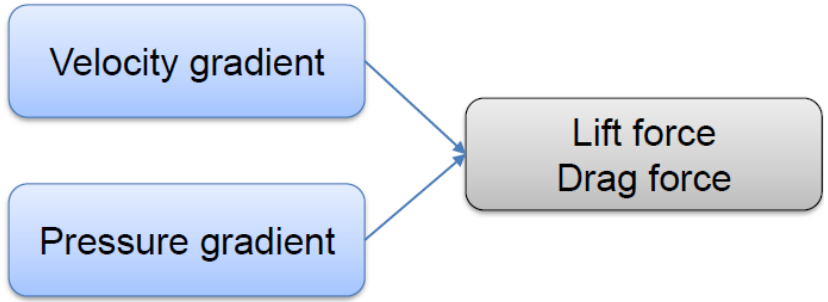




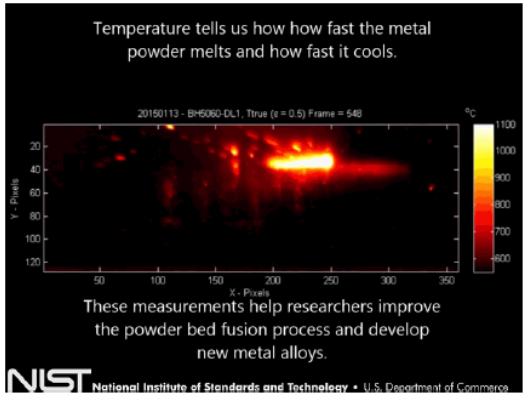


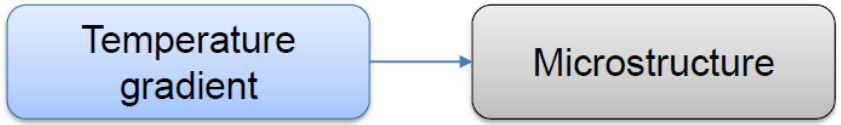
- Derivative/Gradient accuracy is critical in many engineering applications
- Data-driven airfoil design



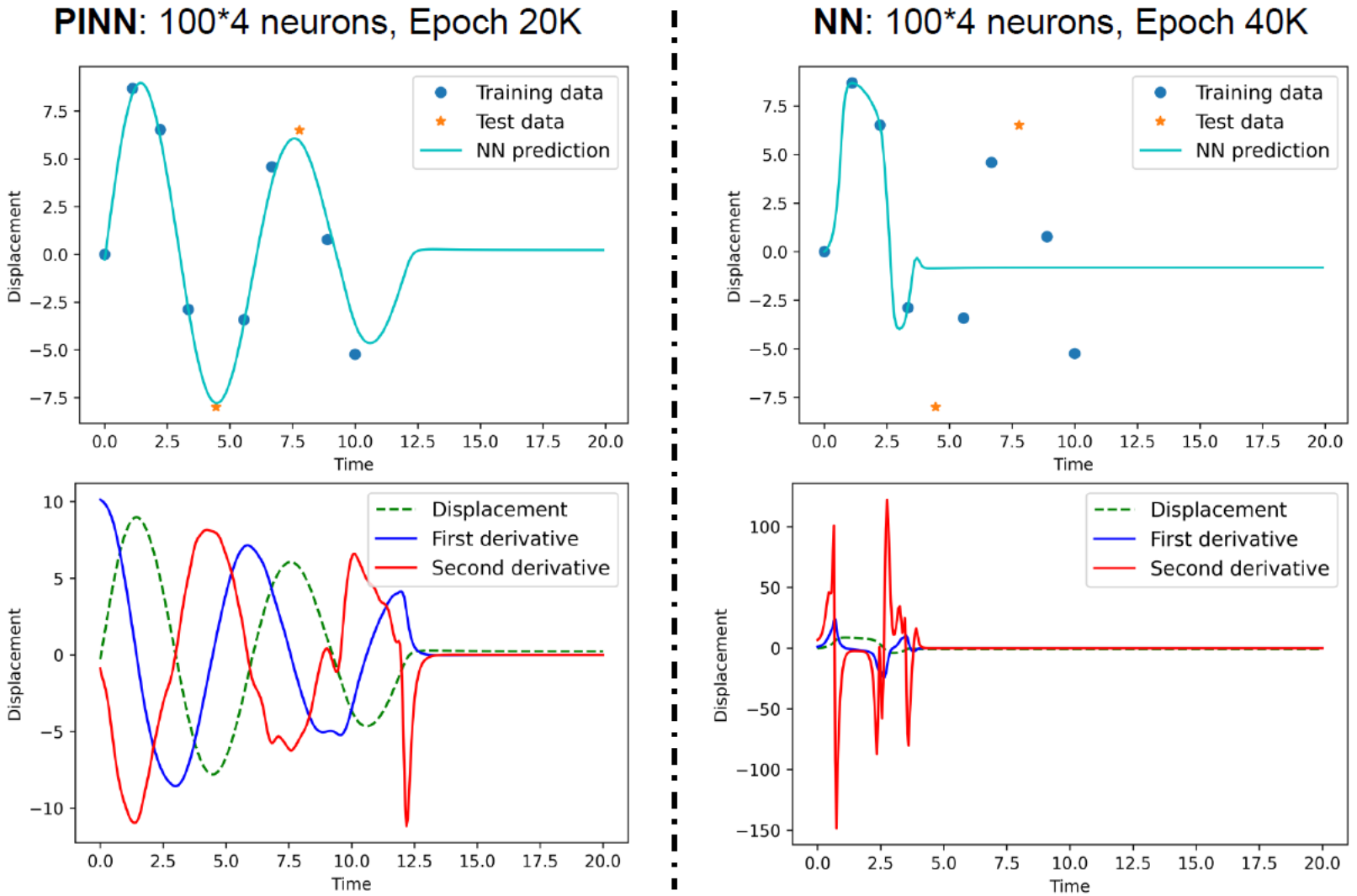


- Pressure-structure linkage in additive manufacturing



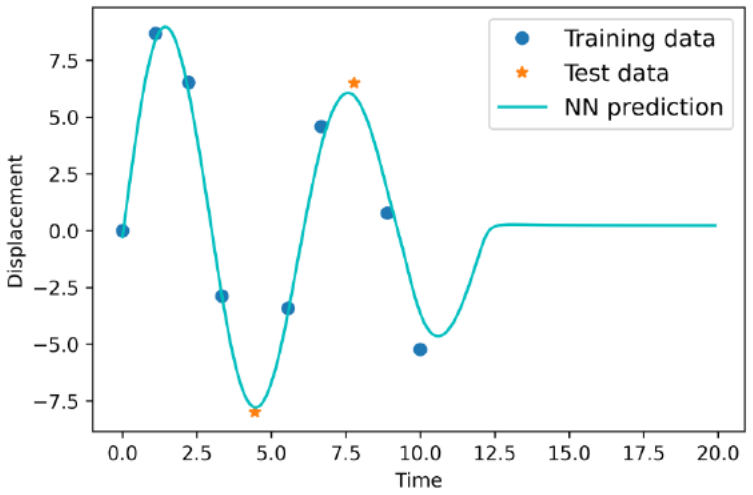


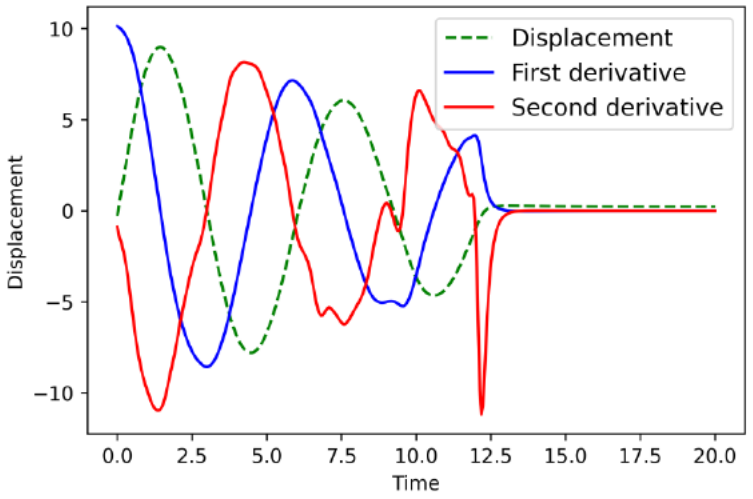
Small Dataset: PINN vs. NN (1)



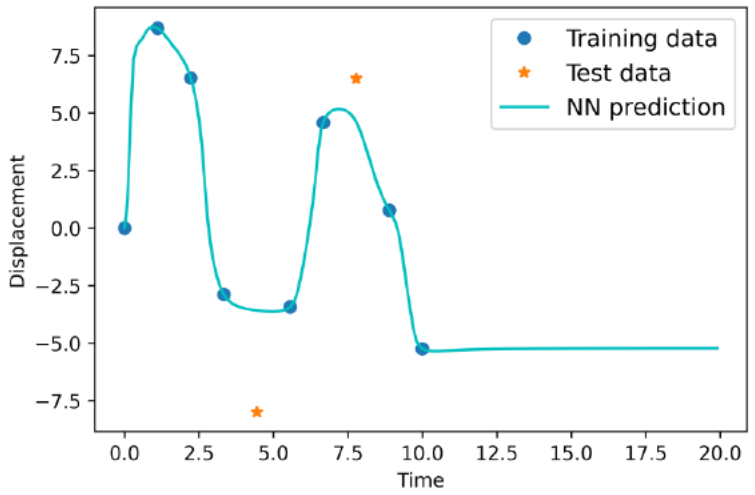
Small Dataset: PINN vs. NN (2)

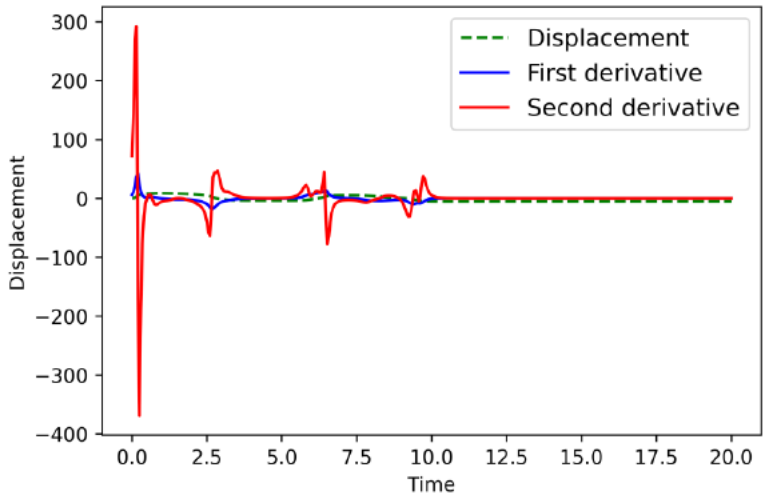
PINN: 100*4 neurons, Epoch 20K





NN: 500*4 neurons, Epoch 10K





Summary

- Adding prior governing equation into NN loss function
  - Can improve prediction accuracy, especially for derivatives
  - Can avoid (to some extent) overfit
  - Works well for small data sets
  - Might improve extrapolation capability
- Physics informed neural networks (PINN)
  - M. Raissi, P. Perdikaris and G.E. Karniadakis . "Physics informed deep learning (Part I): Data driven solutions of nonlinear partial differential equations." arXiv preprint arXiv:1711.10561 (2017)
- Physics guided neural networks (PGNN)
  - A. Karpatne, et al. "Physics guided neural networks (PGNN): An application in lake temperature modeling." arXiv preprint arXiv:1710.11431 2 (2017)
- Physics Aware Neural Networks (PANN)
  - A.S. Zamzam and N.D. Sidiropoulos, IEEE Transactions on Power Systems 35.6 (2020): 4347 4356.
- Mechanistic neural networks (MNN): Embed prior physical knowledge into machine learning
  - S.L Brunton, "Applying Machine Learning to Study Fluid Mechanics.“ arXiv preprint arXiv:2110.02083 (2021)