Application: Diamond Price Regression

4Cs:

Diamond Dataset

- Kaggle (Datasets/Diamonds)
  - 53,940 diamonds with 10 features

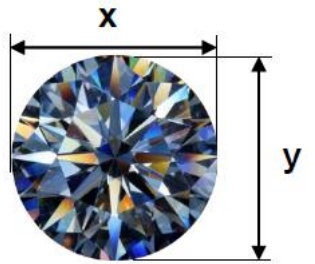
	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58	334	4.2	4.23	2.63
5	0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
6	0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48
7	0.24	Very Good	I	VVS1	62.3	57	336	3.95	3.98	2.47
8	0.26	Very Good	H	SI1	61.9	55	337	4.07	4.11	2.53
9	0.22	Fair	E	VS2	65.1	61	337	3.87	3.78	2.49
10	0.23	Very Good	H	VS1	59.4	61	338	4	4.05	2.39

Price: ($326--$18,823)
Carat: (0.2--5.01)
Cut: (Fair, Good, Very Good, Premium, Ideal)
Color: (J (worst) to D (best))
Clarity: (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))
Size in x direction in mm (0--10.74)
Size in y direction in mm (0--58.9)
Size in z direction in mm (0--31.8)
Depth: **z** / mean(x, y) = 2 * **z** / (x + y) (43--79) (%)
Table: width of top of diamond relative to widest point (43--95) (%)

Color (D, E, F, G, H, I, J) → (1, 2, 3, 4, 5, 6, 7)
D: colorless ~ Z: light yellow or brown

Cut Rating	Numerical value
Premium	1
Ideal	2
Very Good	3
Good	4
Fair	5

Clarity Rating	Numerical value
IF—Internally Flawless	1
VVS1,2—Very, Very Slightly Included 1,2	2
VS1,2—Very Slightly Included 1,2	3
SI1,2—Slightly Included 1,2	4
I1—Included 1	5



Feature Engineering

- Remove missing values
- Convert categorical label
  - Cut: Excellent(0), Very good(1), Good(2), Fair(3), Poor(4)
- Normalization/standardization

$$z = \frac{x - \mu}{\sigma}$$

$$\begin{cases} \mu: \text{ mean (average)} \\ \sigma: \text{ standard deviation from the mean} \\ z: \text{ transformed data} \end{cases}$$

Cut:
Datapoints: 53884
Mean: 2.55
Std: 1.03
Original range: [0, 4]
Normalized range: [-2.48, 1.41]

Price:
Datapoints: 53884
Mean: 3924.82
Std: 3979.89
Original range: [326, 18823]
Normalized range: [-0.9, 3.47]

Dimension Reduction

- Reformulating data using less features
- Separate between relevant and irrelevant features
  - Correlation matrix
- Select importance features (from 9 features)
  - Four important features: carat, clarity, color, length (y)
  - Modeling approach: Random Forest
- Find a new set of linear/nonlinear transformed variables
  - Principal component analysis (PCA)

carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31

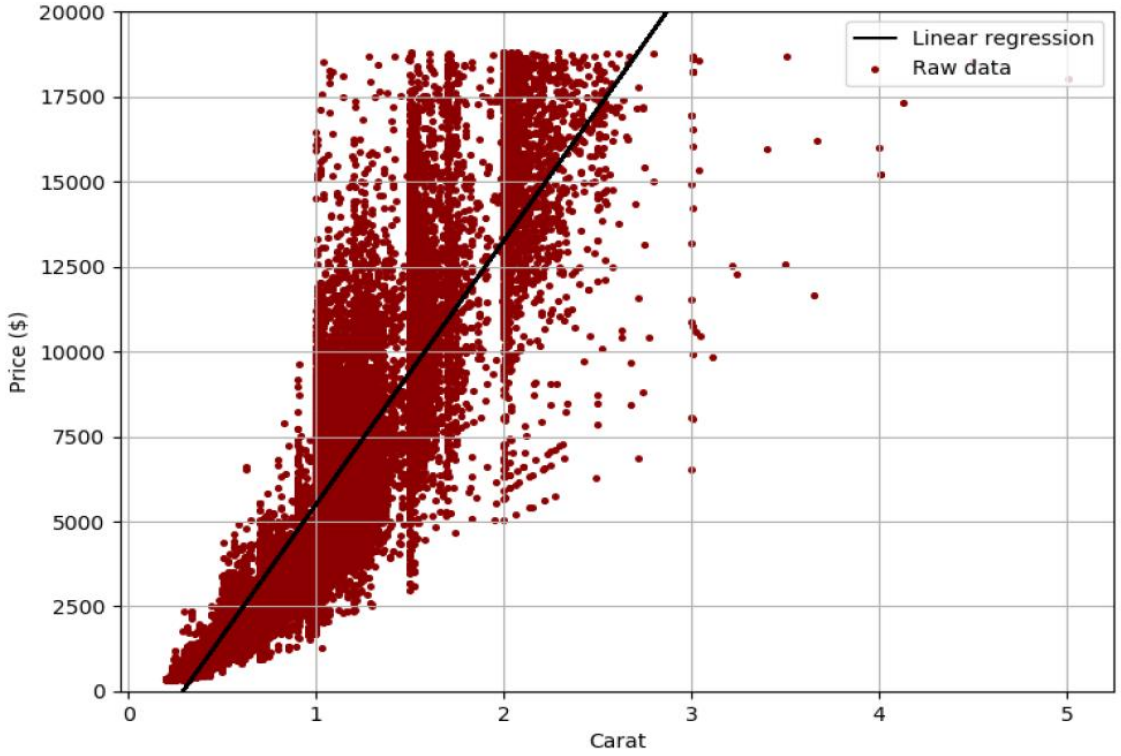
Reduced Order Model

- Model order reduction is a technique for reducing the computational complexity of mathematical models

- Single variable linear regression (Reduced order model)
- Multivariable linear regression (Reduced order model)
- Neural network model

Linear Regression

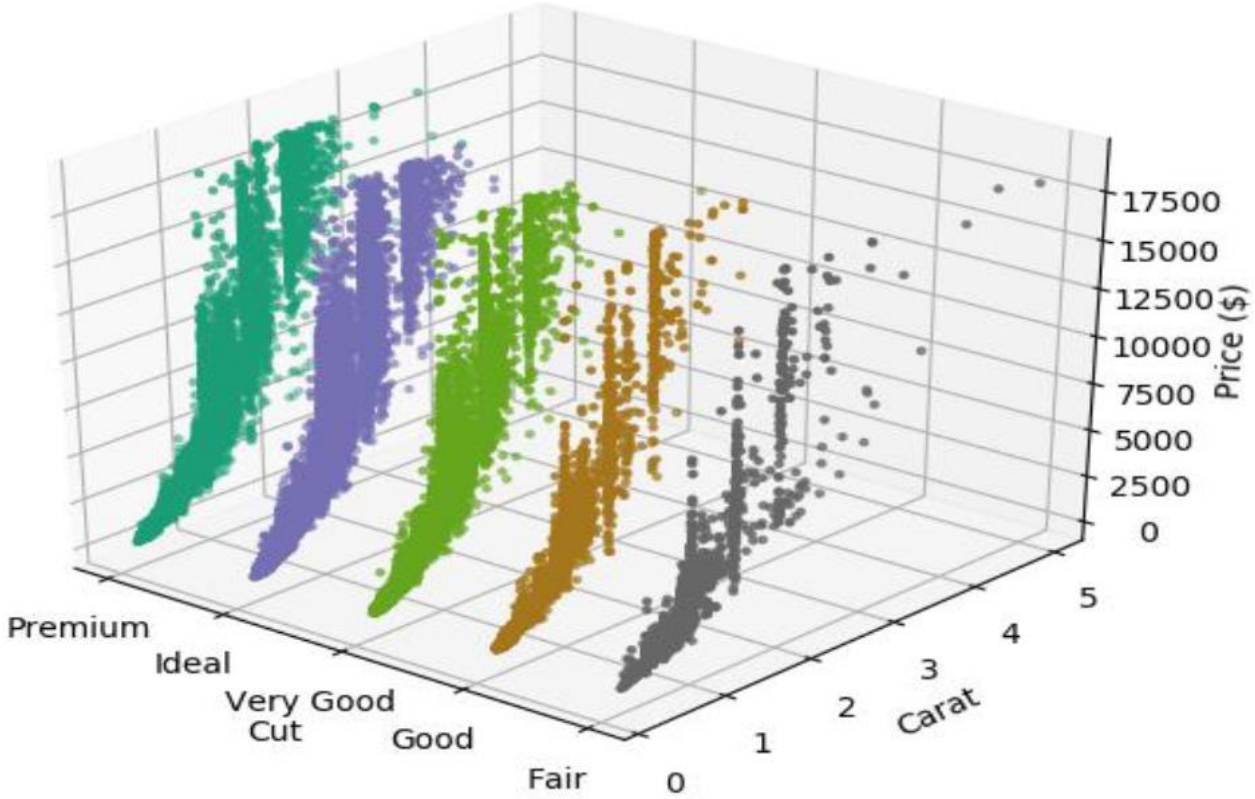
- Single variable/feature:



$$Price = \beta_1 Carat + \beta_2$$

$$minize \sum_{k=1}^{n} |\beta_1 Carat_k + \beta_2 - Price_k|^2$$

- Two variables/features:



Multivariable Linear Regression

- Split dataset for training set (70%) and testing set (30%)
- Training step: $(\text{Price}) = \beta_1(\text{Carat}) + \beta_2(\text{Cut}) + \beta_3(\text{Color}) + \cdots + \beta_9(\text{z}) + \beta_{10}$

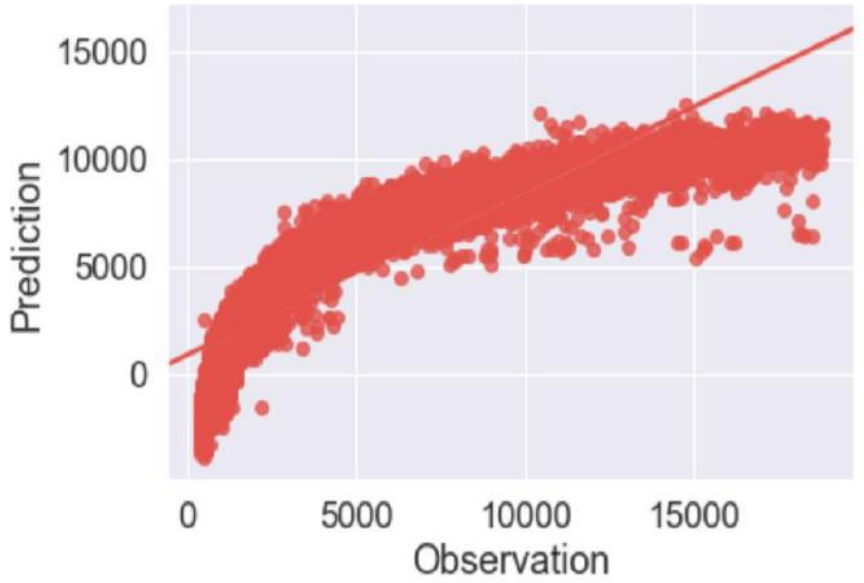
get $\begin{bmatrix}\beta_1 & \beta_2 & \cdots & \beta_9 & \beta_{10}\end{bmatrix}$

$$= \begin{bmatrix}\beta_1 & \beta_2 & \cdots & \beta_9 & \beta_{10}\end{bmatrix} \begin{bmatrix}\text{Carat}\\ \text{Cut}\\ \vdots\\ \text{z}\\ 1\end{bmatrix}$$

- Test step:

$$\begin{bmatrix}\beta_1 & \beta_2 & \cdots & \beta_9 & \beta_{10}\end{bmatrix} \begin{bmatrix}\text{Carat}\\ \text{Cut}\\ \vdots\\ \text{z}\\ 1\end{bmatrix} = (\text{Price}^*)$$

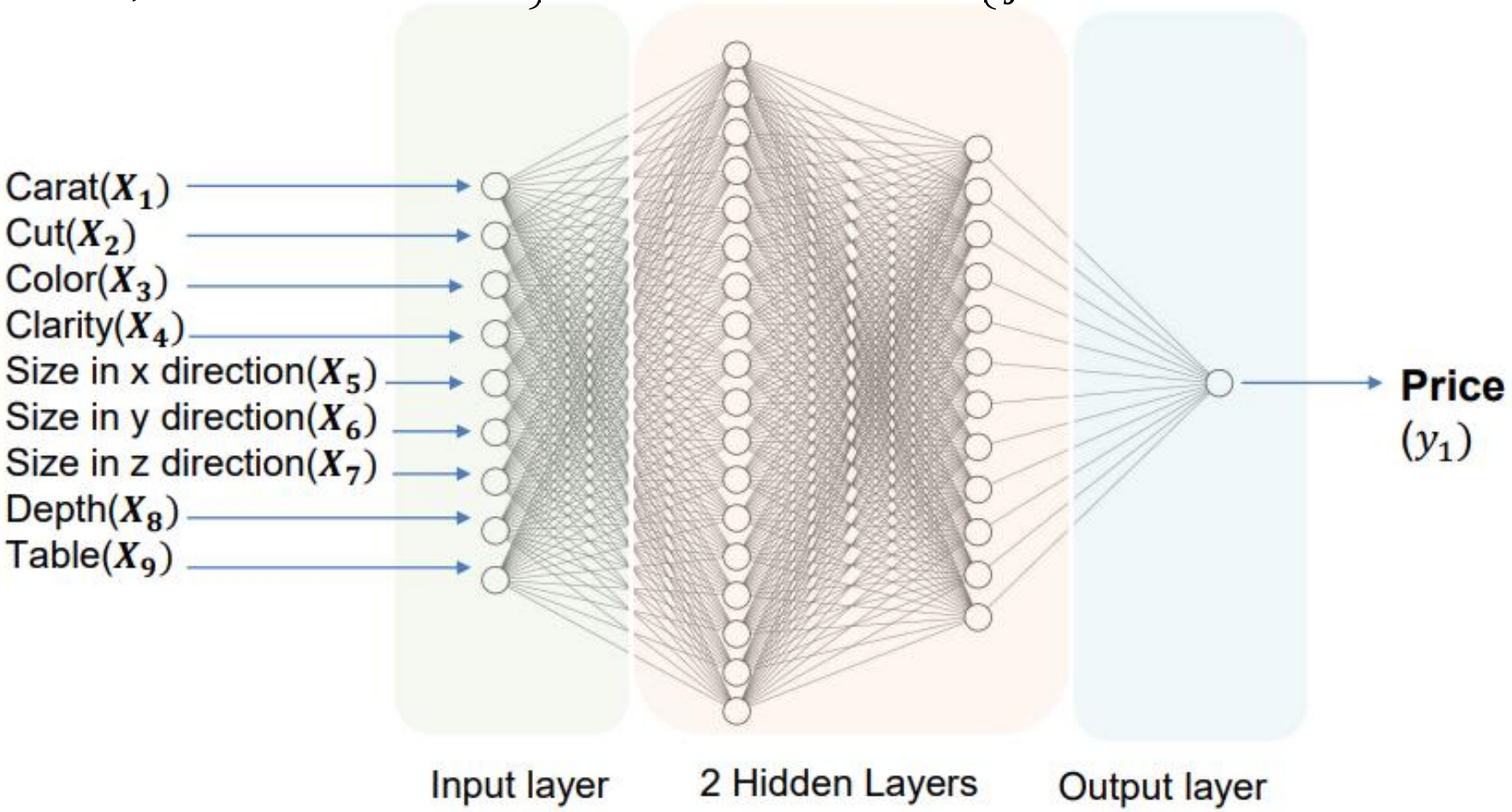
compare (Price^*) and (Price)



	Variable	Coeff
6	x	1118.274677
7	y	1014.538292
0	carat	860.428808
8	z	639.088871
3	clarity	531.345714
1	cut	94.376440
4	depth	-54.461730
5	table	-140.119885
2	color	-221.212498

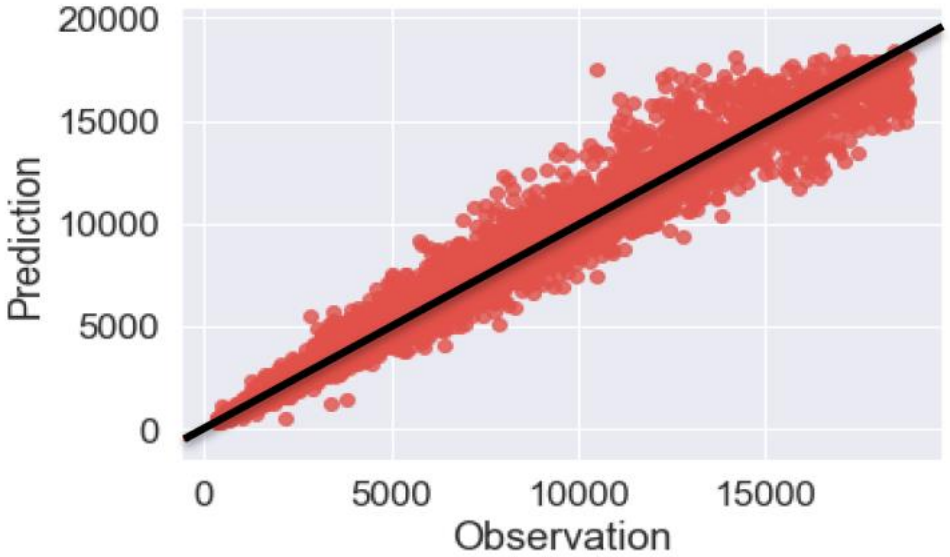
Neural Network Architecture

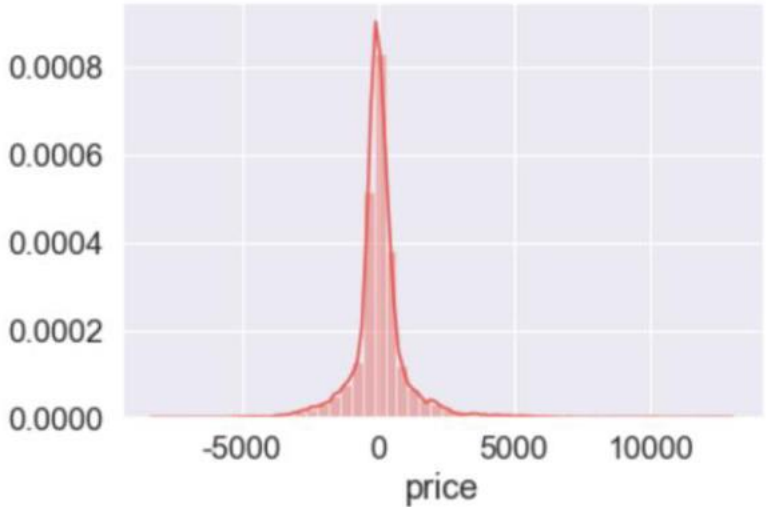
$$\left.\begin{array}{l} 9 \text{ input variables } \left(\mathbf{x}_i^N\right) \\ i = 1,\ldots 9 \text{ and } N = 53940 \end{array}\right\} \xrightarrow{\text{Find the unknown function}} \left\{\begin{array}{l} 1 \text{ output variables } \left(\mathbf{y}_j^N\right) \\ j = 1 \text{ and } N = 53940 \end{array}\right.$$



Network Training and Prediction

- Loss: MSE(mean squared error)
- Optimizer: Adam
- Architecture:
- Metrics
  - Mean absolute error: 615.97
  - Mean squared error: 1139999.82
  - R-squared: 0.93





System & Design: Prediction of Unseen data

	carat	cut	color	clarity	depth	table	x	y	z
0	4.0	1	0	0	65.5	59	10.74	10.54	6.98
1	2.0	1	0	0	65.5	59	10.74	10.54	6.98
2	0.5	2	2	1	61.4	56	4.33	4.37	2.67
3	0.3	0	1	2	59.7	58	4.13	4.14	2.47

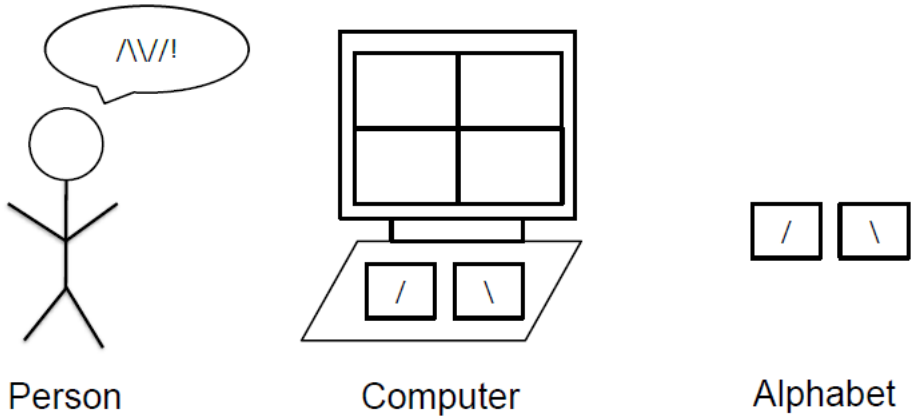
```
Prediction: [12148.8  9818.9  1573.8   503.2]
```

System & Design: Limitations of the model

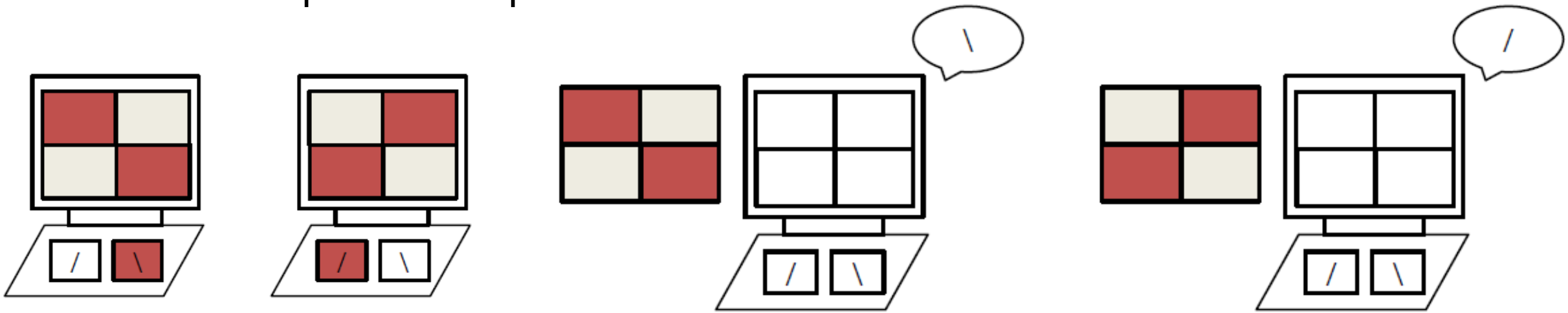
- Dataset is updating which depends on the market.
- Confidence level of the prediction (Uncertainty?)
- Unknown features
  - Jewelry brand
  - SPARKLE
  - Preference of customer
  - …
- Bargaining

A First Look at Convolution Neural Network (1)

- Image a simple world

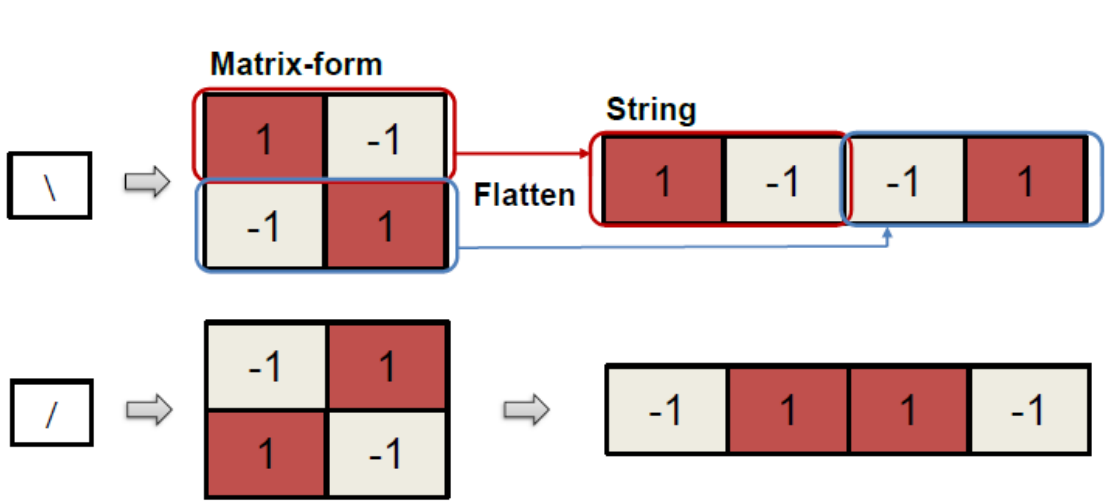


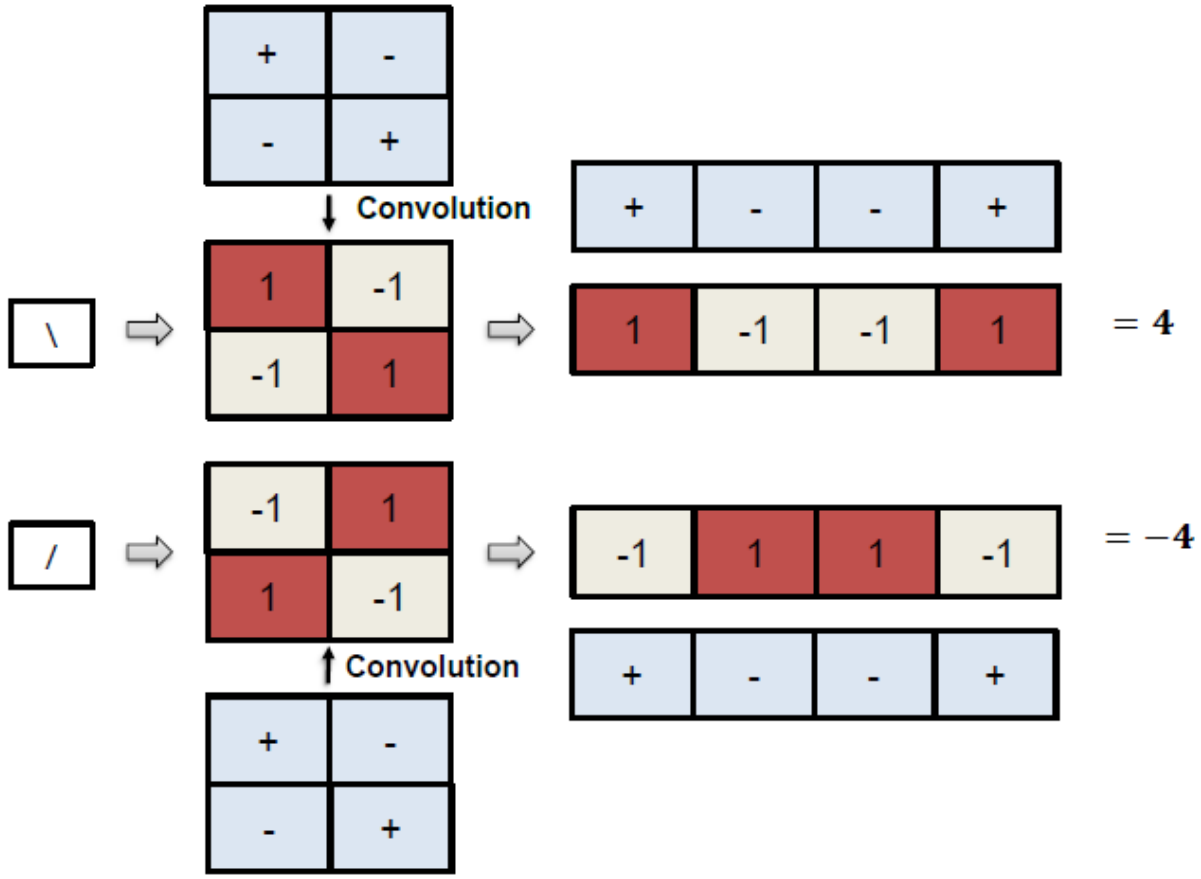
- Train computer the alphabet



A First Look at Convolution Neural Network (2)

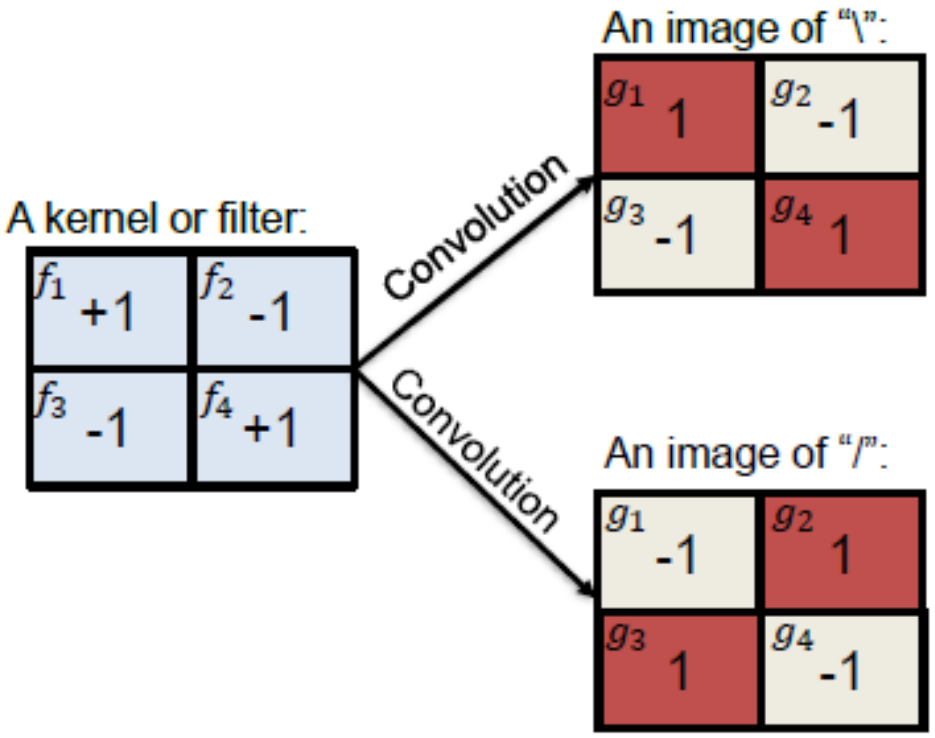
- How the data store in the computer
- How can we define a right operation? (addition?)





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:

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

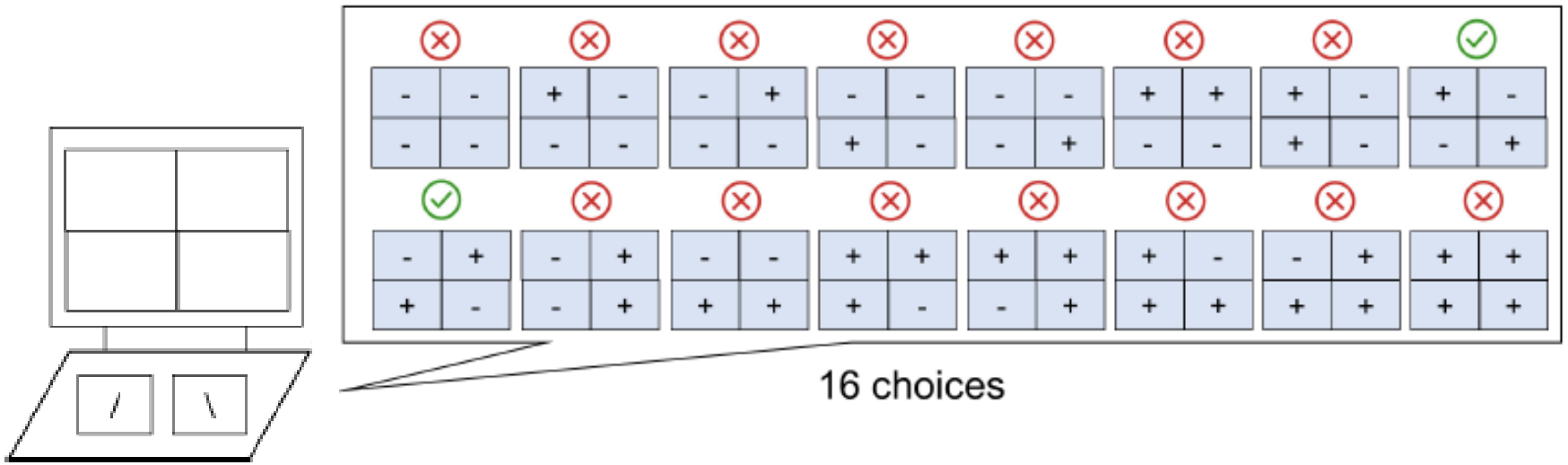
Result is positive, backward slash: "\"

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

Result is negative, forward slash: "/"

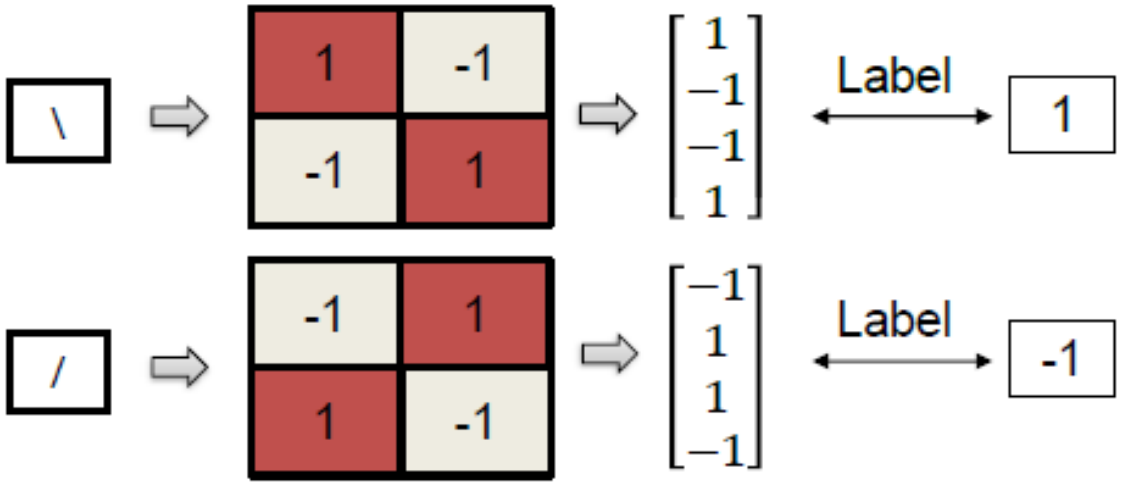
Kernels to be tested

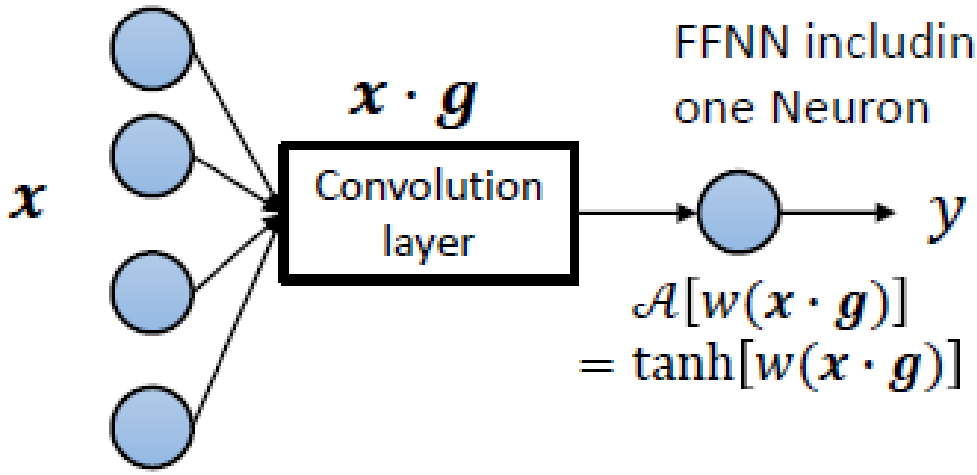
- We have a chance to check all the possible filters only if we assume the values in a filter are either 1 or +1.
  - select good filters by checking all the possibilities
- If we consider a filter with real numbers, a more efficient approach is required.
- That is why CNN get into the picture!



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

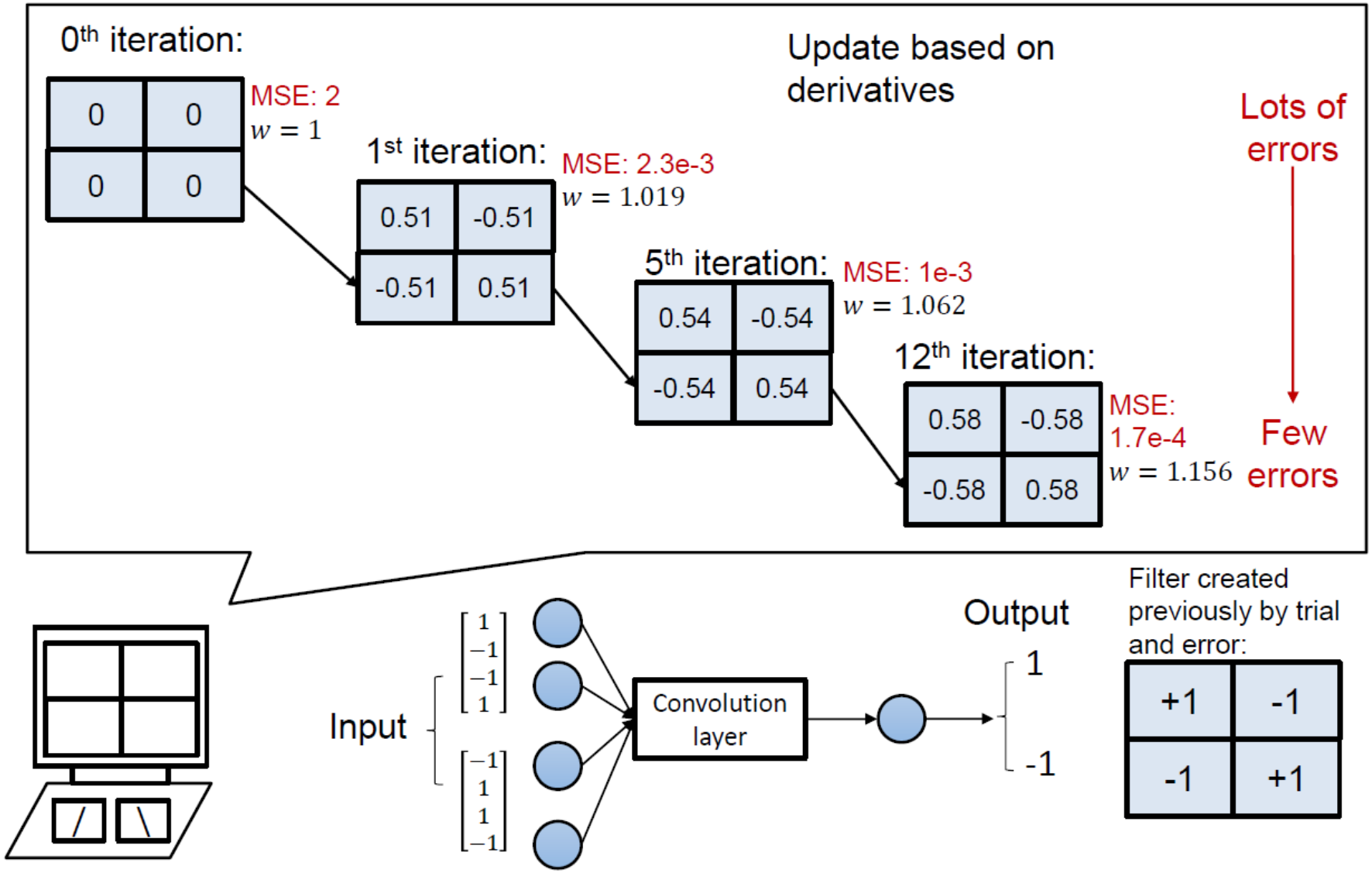
```
def calc_y(g):
# calculate y
    x=g[0]*(g[1]*X[:,0]+g[2]*X[:,1]+g[3]*X[:,2]+g[4]*X[:,3])
    y=np.tanh(x)
    return y

# 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)
```

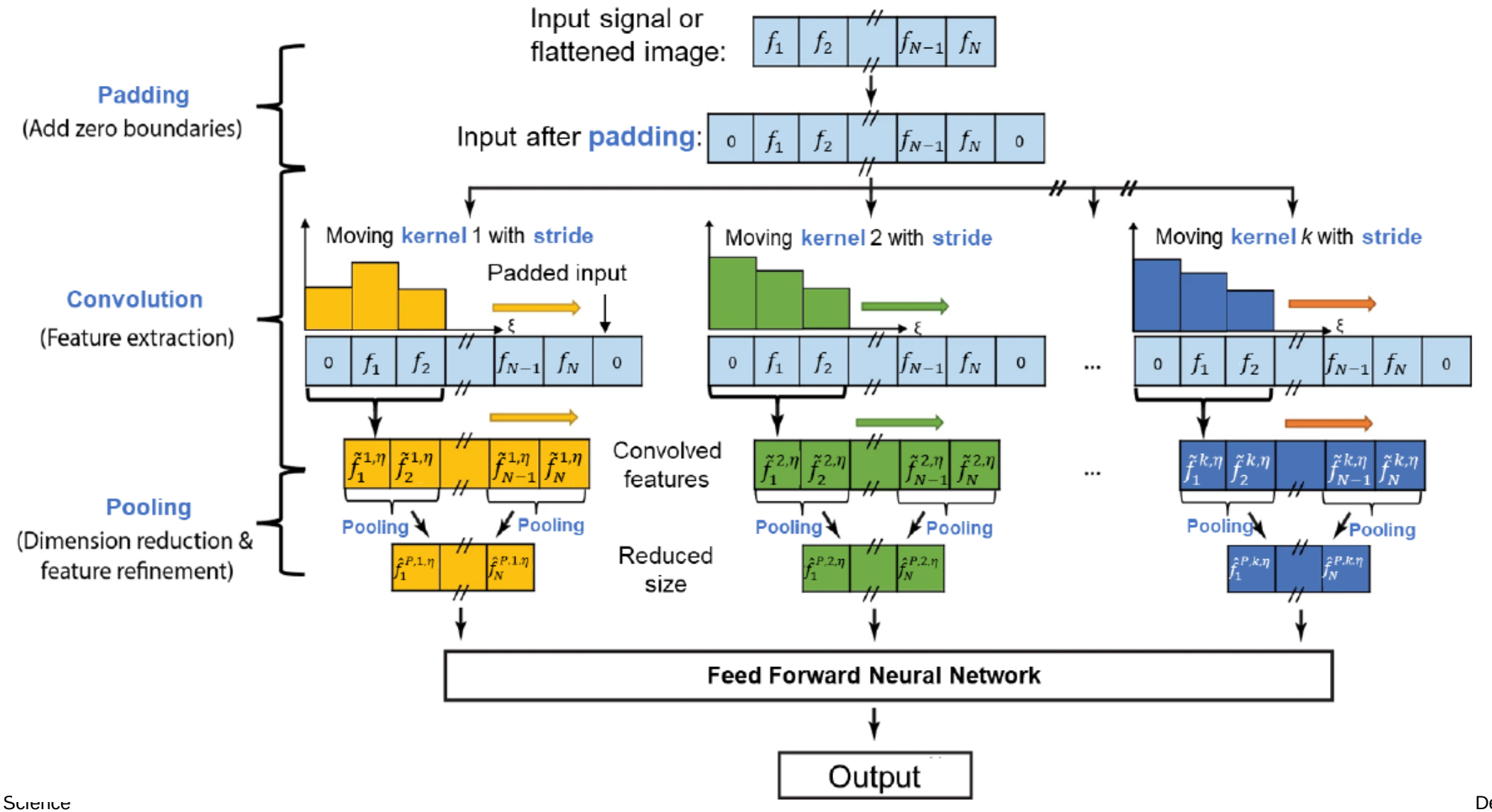
Using gradient descent to update filter



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

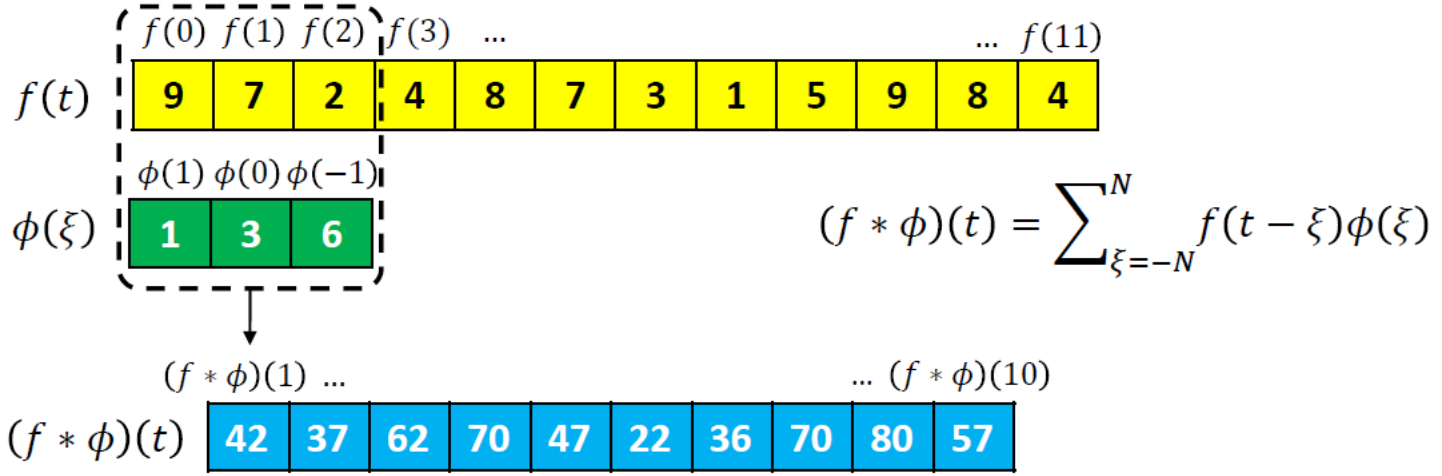


Terminology used in CNN

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

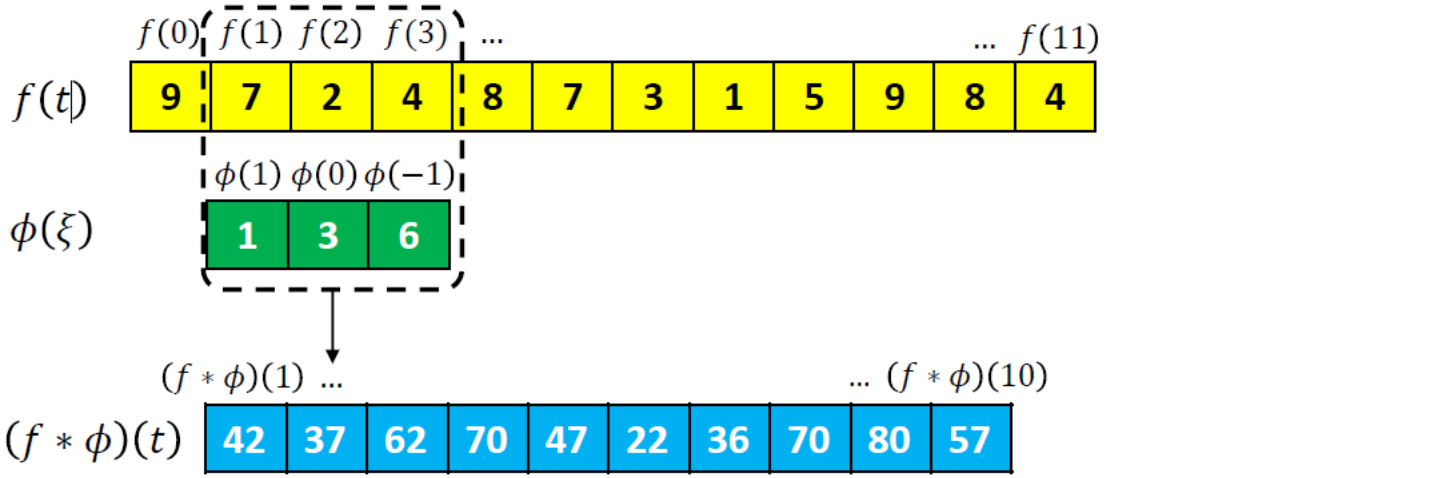
Convolution

Convolution operation step 1:



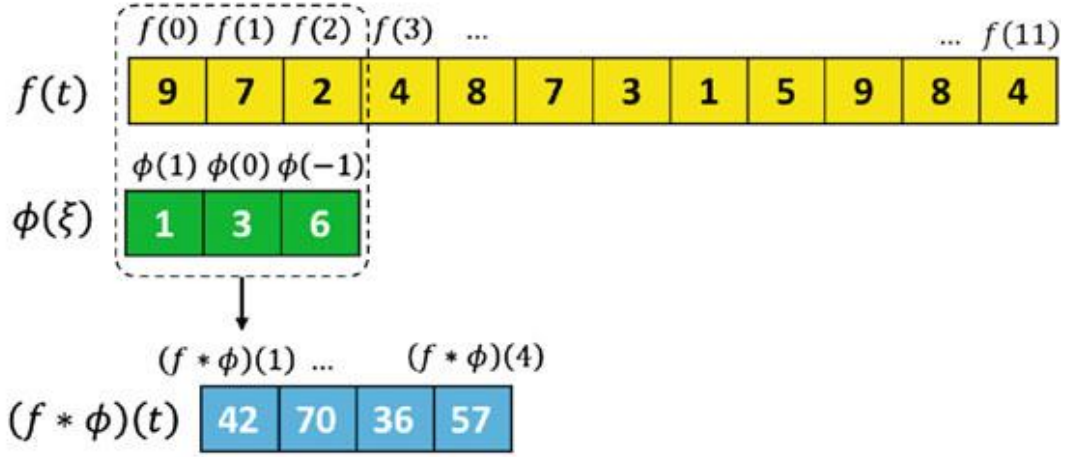
$$(f * \phi)(t) = \sum_{\xi=-N}^{N} f(t-\xi)\phi(\xi)$$

Convolution operation step 2:

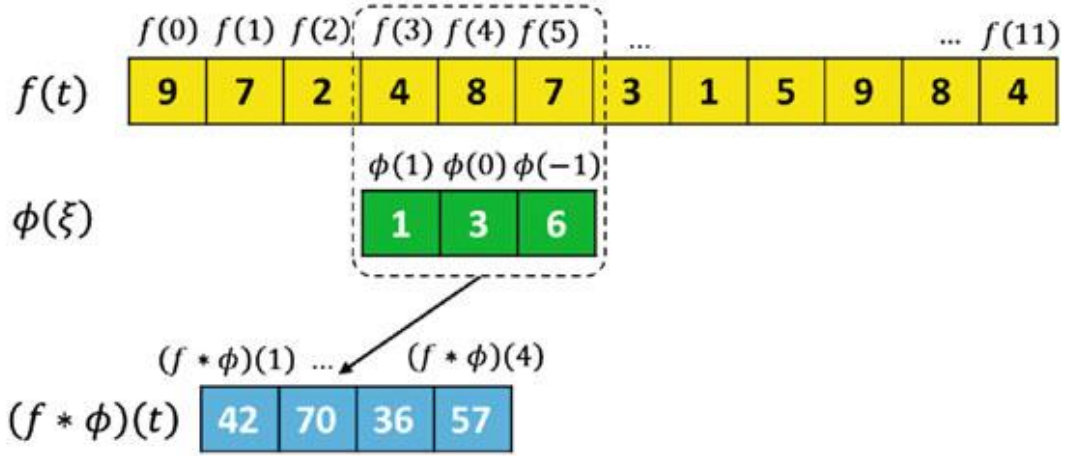


Stride

Convolution operation step 1 with a stride value of 3:



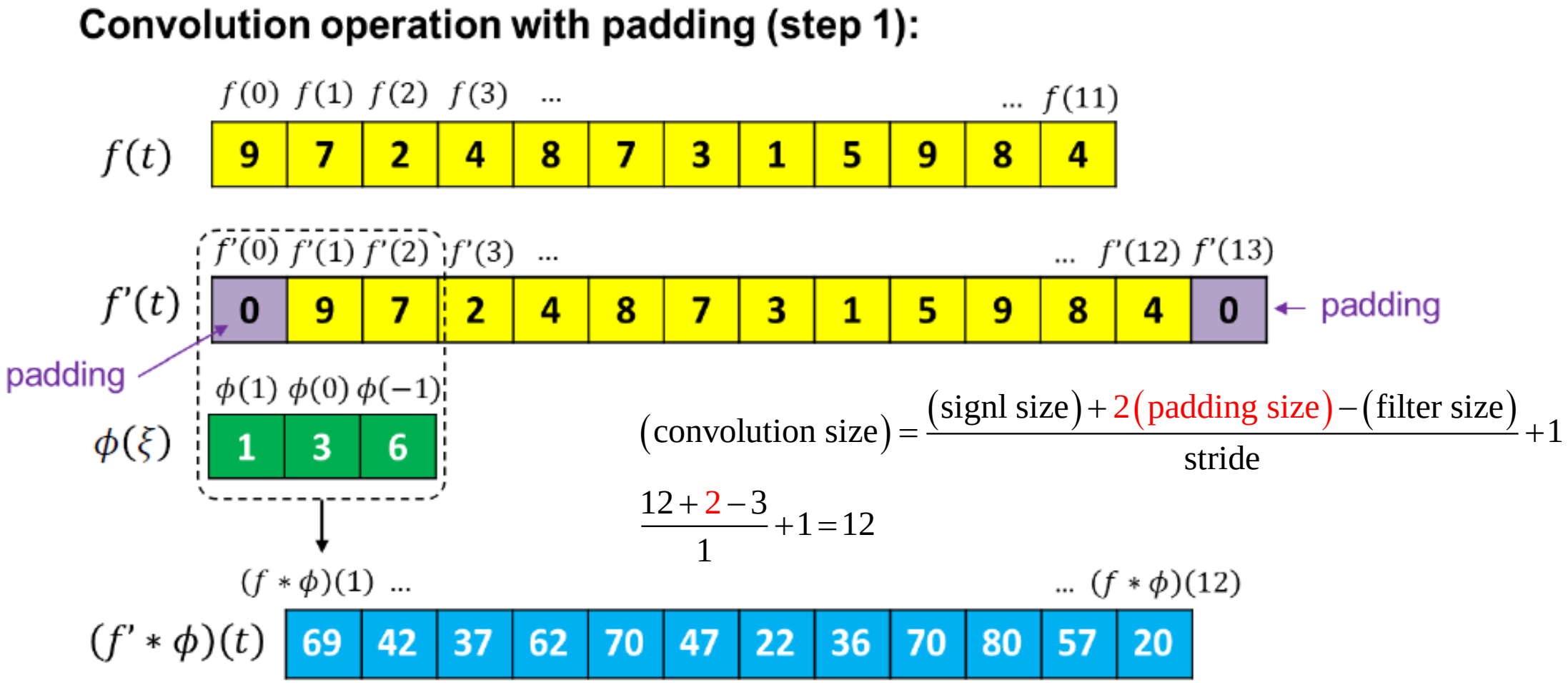
Convolution operation step 2 with a stride value of 3:



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

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

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

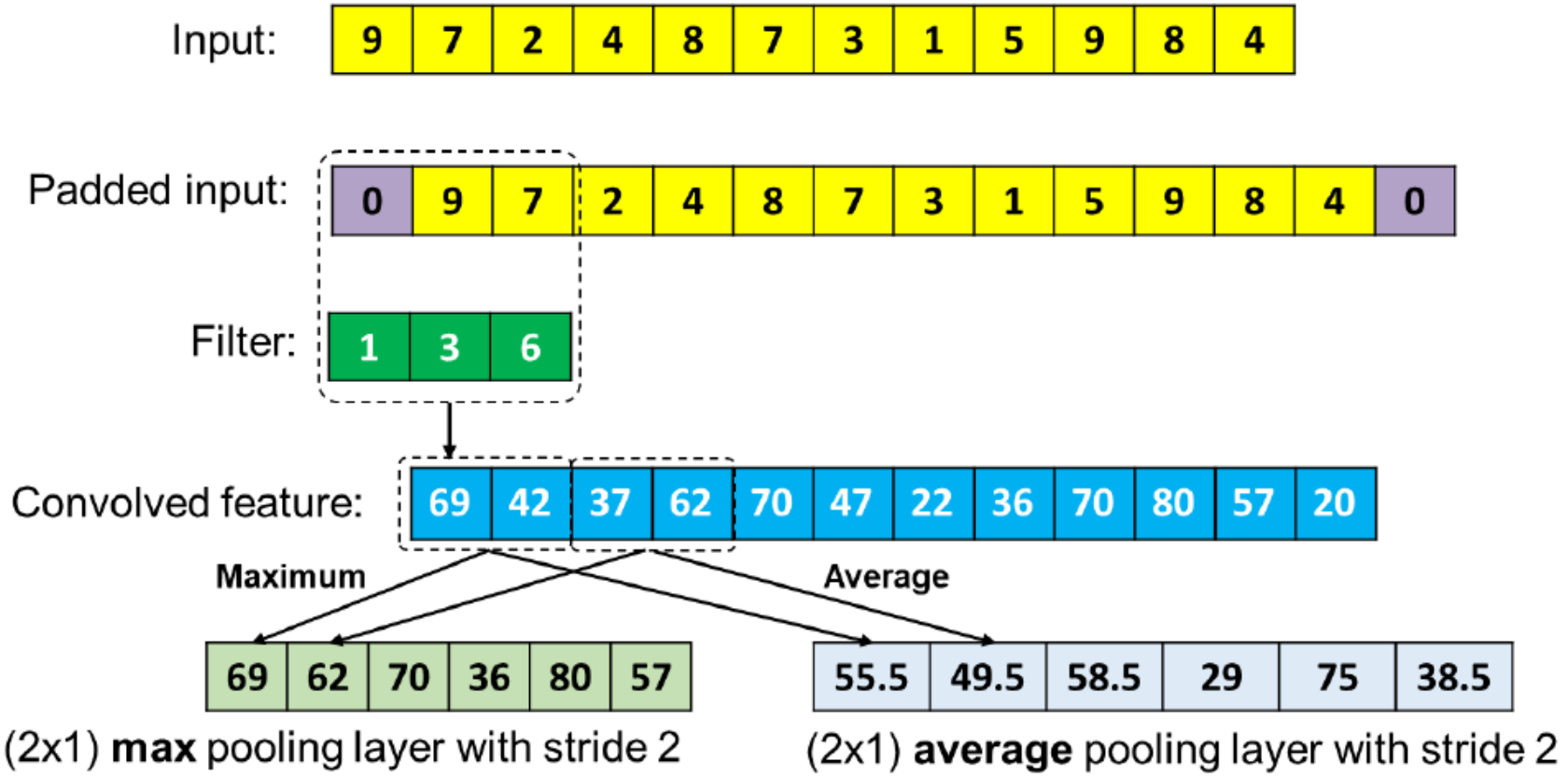
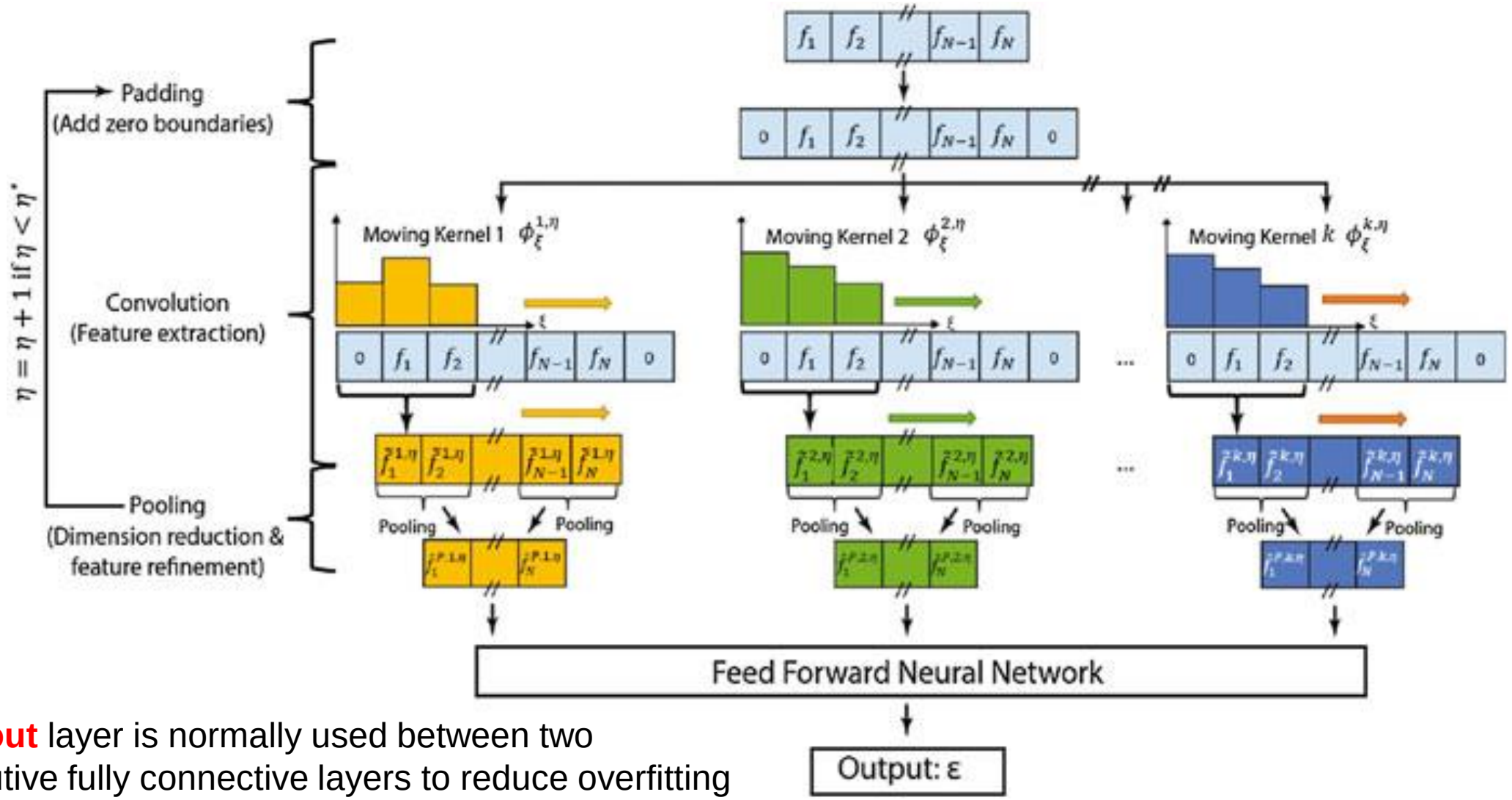


Illustration of one-dimensional CNN



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

General Notations

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

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

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

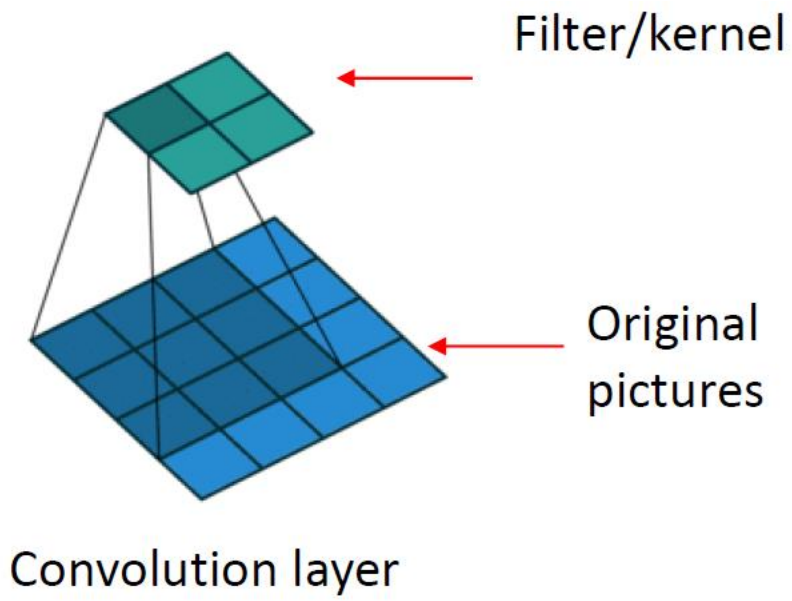
$$\begin{cases} N : \text{ number of data points in the training set} \\ \boldsymbol{\varepsilon}^i : \text{ CNN output of the } i\text{-th data point} \\ \boldsymbol{\varepsilon}^{*i} : \text{ labeled output of the } i\text{-th data point} \end{cases}$$

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

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

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

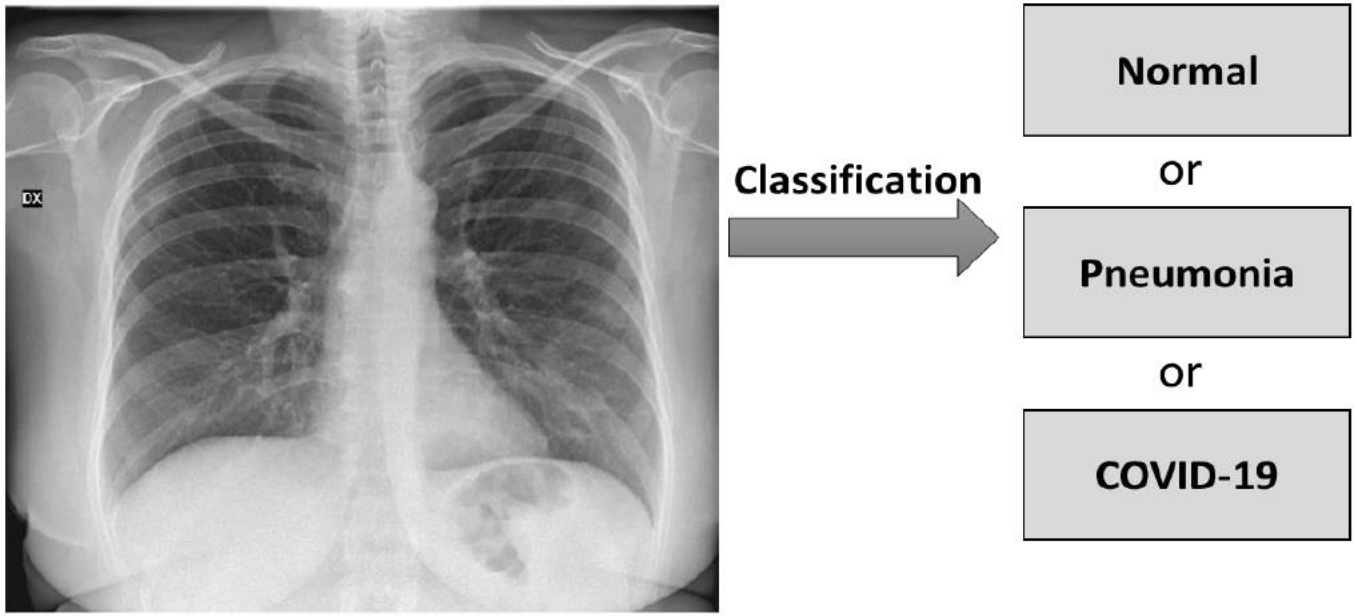
Application: COVID-19 detection from chest x-ray images of patients

- 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

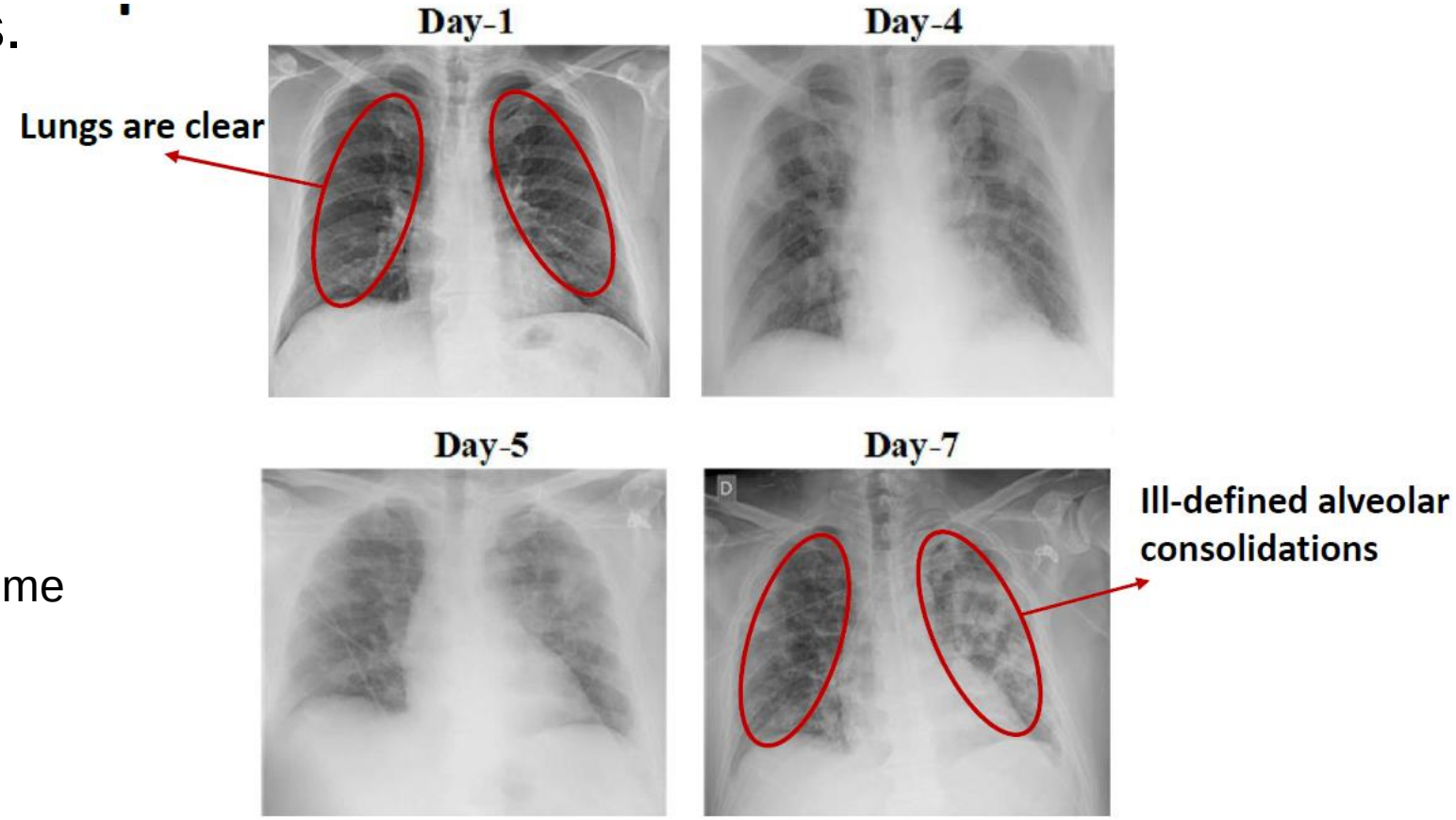
Classification example

- 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.



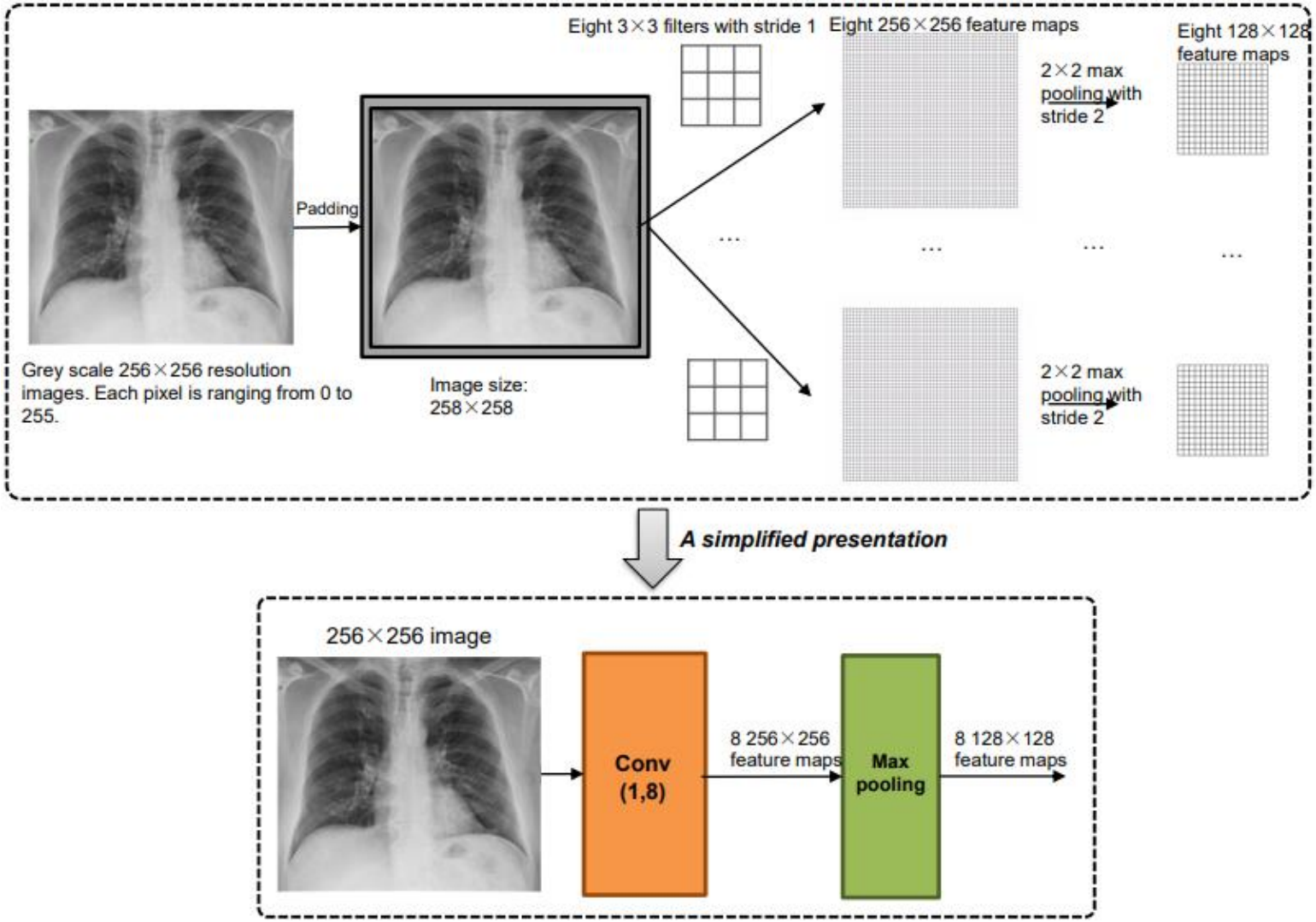
Acute Respiratory Distress Syndrome

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

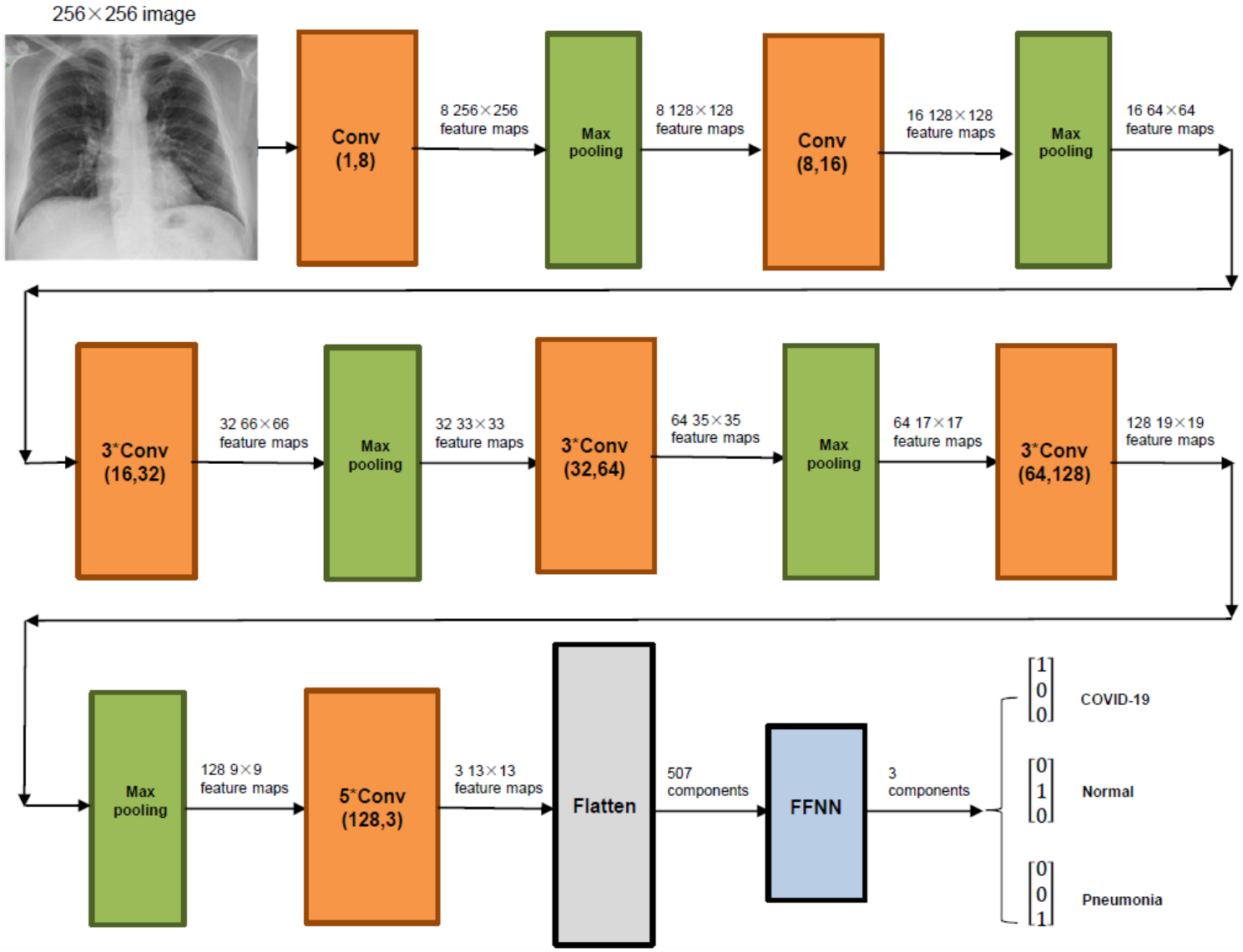
Automated detection of COVID-19 cases using deep neural networks with X ray images



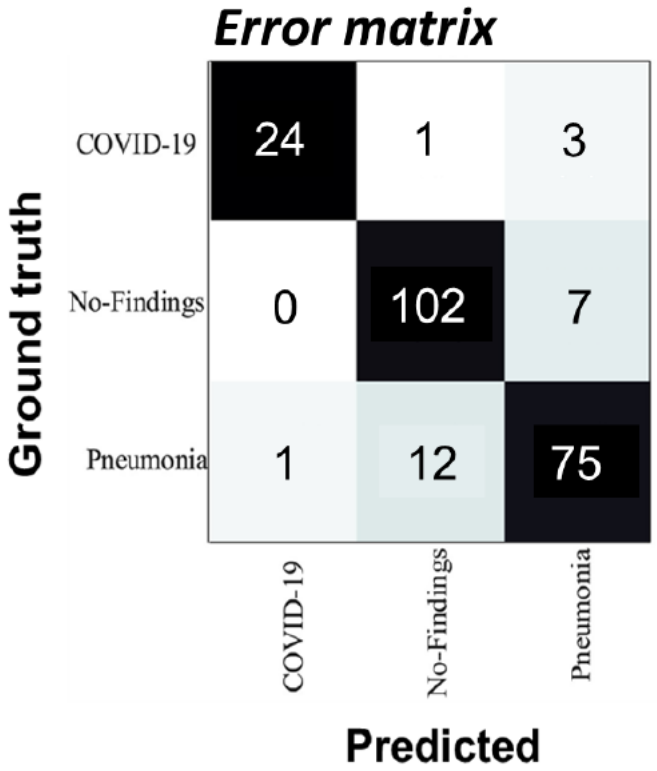
Architecture of the CNN model (1)

- 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.