Deep Learning Fundamentals

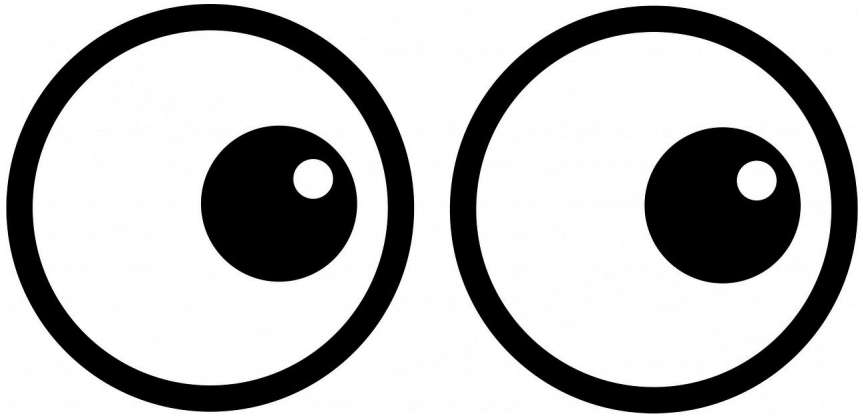
Takuma Kimura

Chapter 2
Convolutional Neural Network

1. Computer Vision

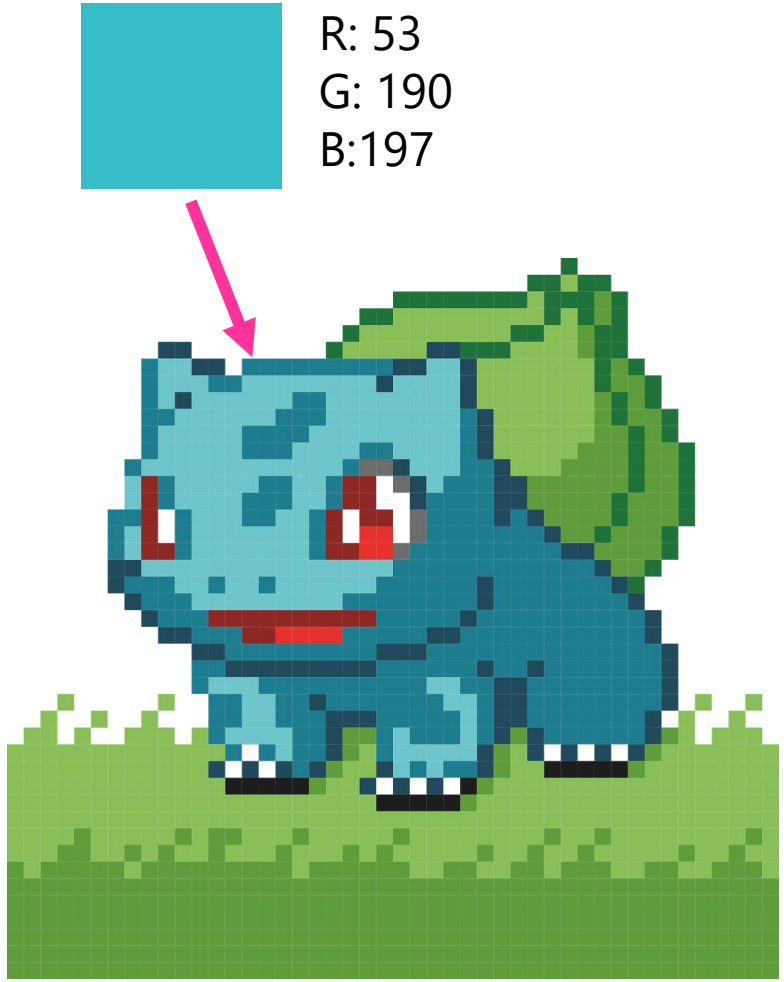
What is Computer Vision

- A scientific field that explores how to reproduce our visual functions by computers.
- Computers extract information and obtain high-level understanding from images and videos as human visual functions do.
- Practical fields
  - SNS
  - Self-driving cars.



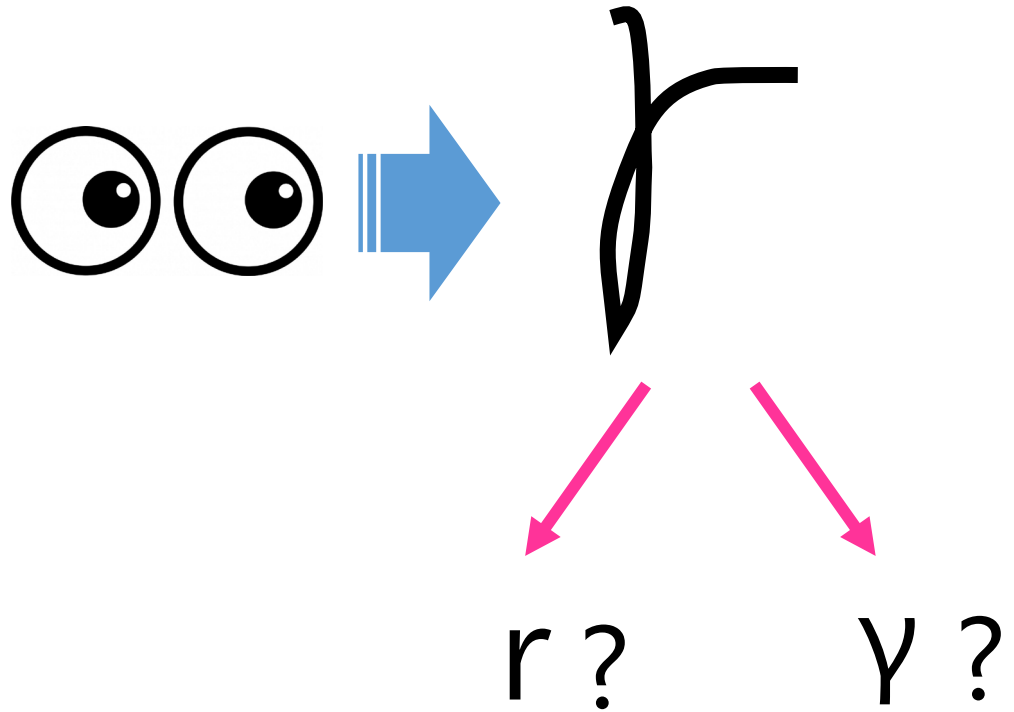
How Do Computers Understand Images?

- Computers cannot understand images in the same way humans do.
- Computers can only get insight from numerical data.
- We need to convert image data into numerical data using pixels.



Computer Vision Problems

- Character recognition
- Image classification
- Object detection
- Facial recognition
- Motion analysis
- Image generation　　etc.



Character Recognition

- Computers recognize each character and number.

- Computers identify what numbers and sentences are written.

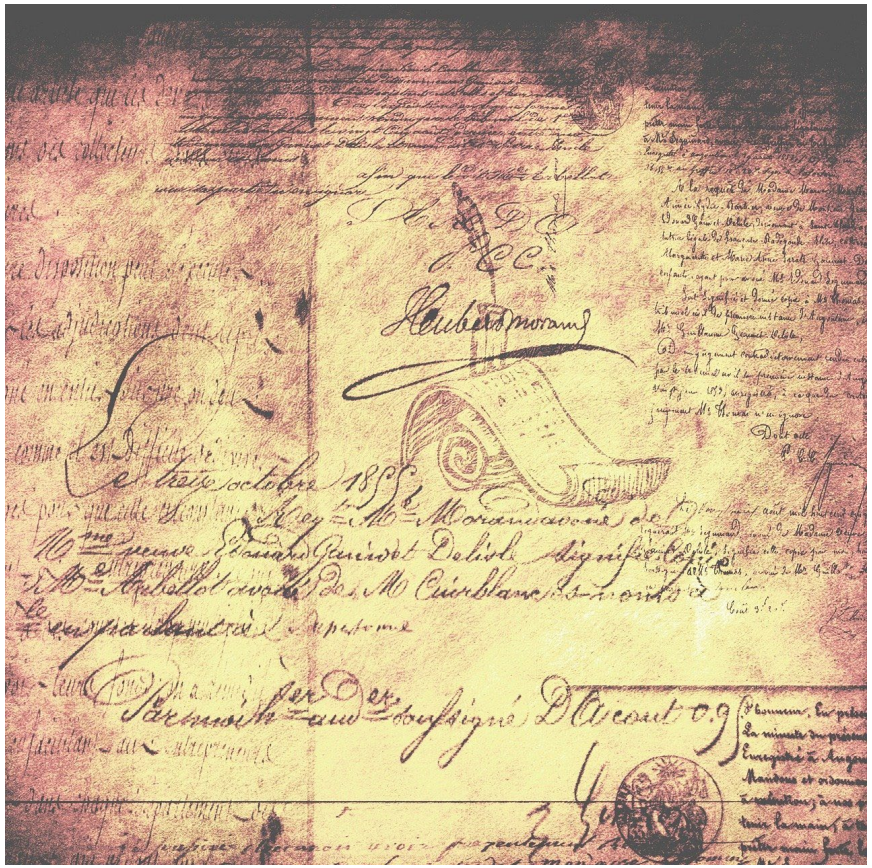
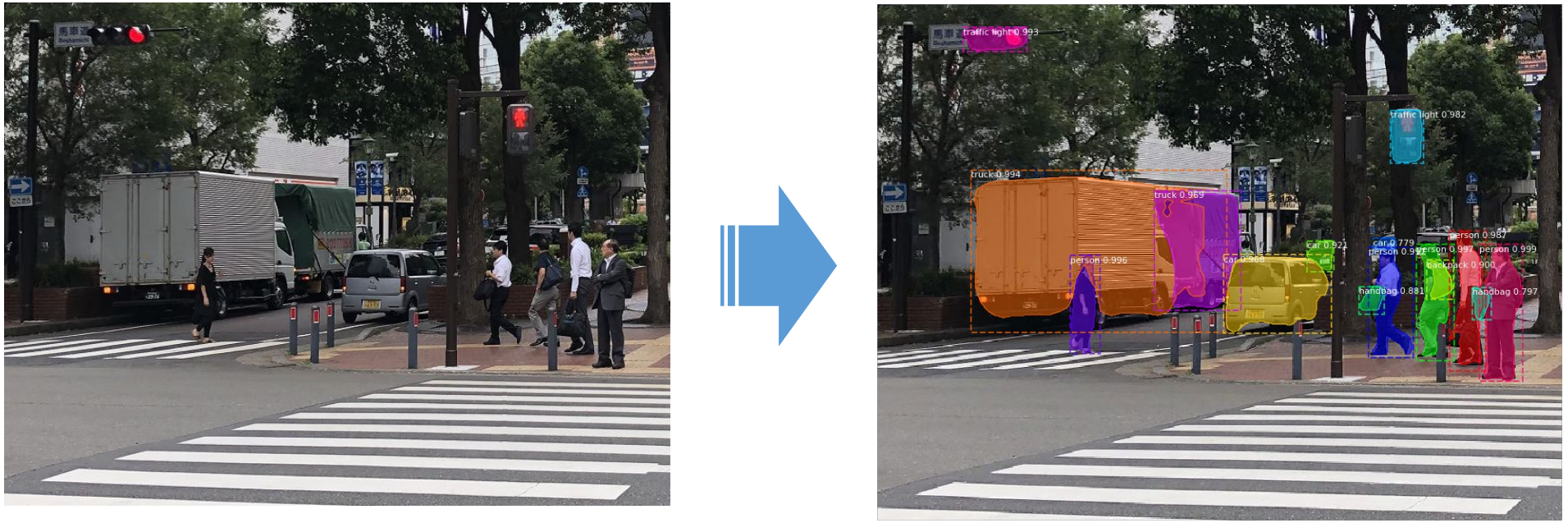


Image Classification

- Computers classify the images and label each data.

Object Detection

- Computers identify objects in an image or video data.



2. Image Data

RGB Color Model

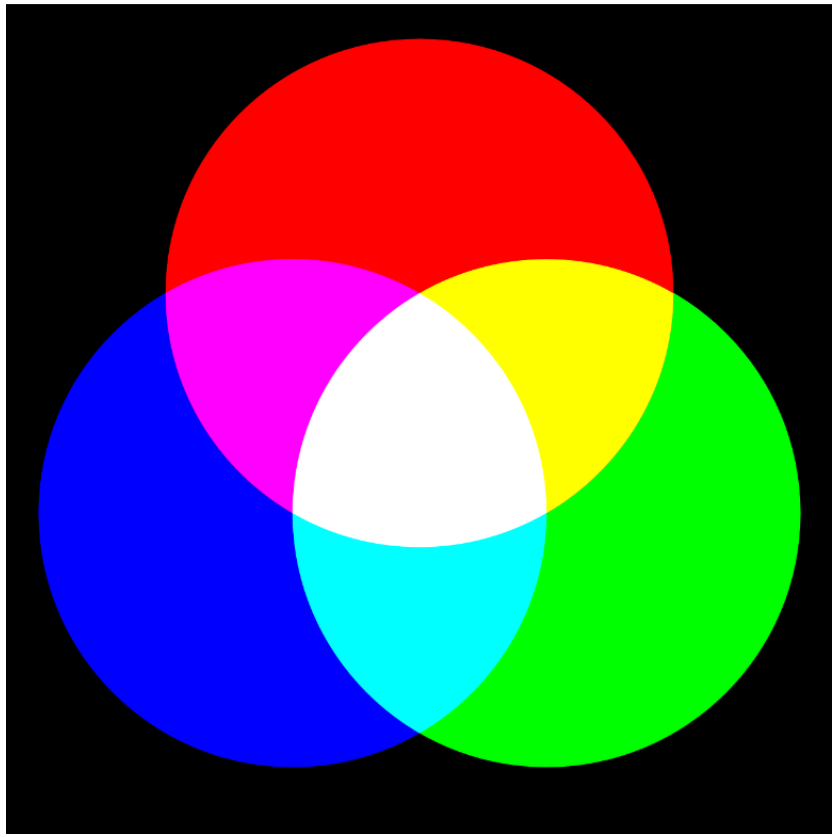
- RGB color model

  Three primary colors:

  Red

  Green

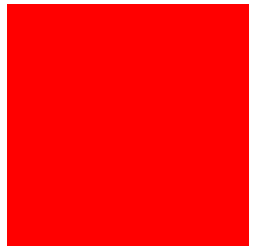
  Blue



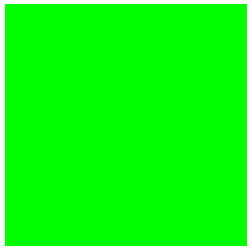
- Each primary color has a pixel value: 0-255.

Examples: RGB Color Model

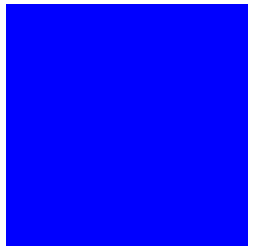
R: 255
G: 0
B: 0



R: 0
G: 255
B: 0



R: 0
G: 0
B: 255

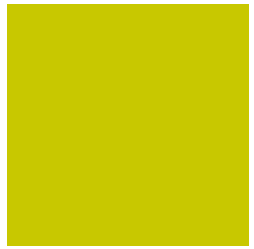


R: 200
G: 0
B: 200

R: 100
G: 0
B: 100

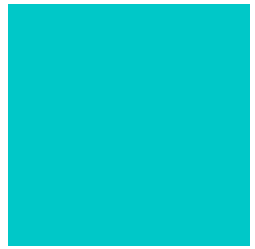
Examples: RGB Color Model (Continued)

R: 200
G: 200
B: 0



R: 100
G: 100
B: 0

R: 0
G: 200
B: 200



R: 0
G: 100
B: 100

R: 200
G: 200
B: 100

R: 100
G: 100
B: 200

R: 100
G: 200
B: 200

R: 200
G: 100
B: 100

Grayscale Image

- A grayscale image express the shading of black and white.

- It contains information of brightness only.

  It does not contain color information.

RGB and Grayscale

R: 0	R: 35	R: 70	R: 105	R: 140	R: 175	R: 220	R: 255
G: 0	G: 35	G: 70	G: 105	G: 140	G: 175	G: 220	G: 255
B: 0	B: 35	B: 70	B: 105	B: 140	B: 175	B: 220	B: 255

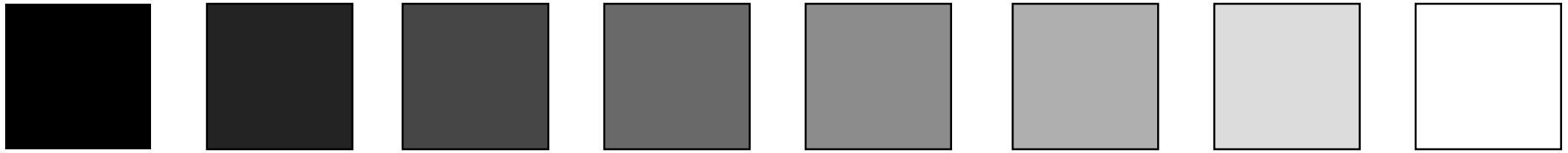


Image Recognition

3. What is CNN?

Convolutional Neural Network

- CNN consists of neurons with weights.

  CNN's learning process is similar to an artificial neural network.

- First, the weights are set randomly.

  Each neuron receives inputs and compute the dot product.

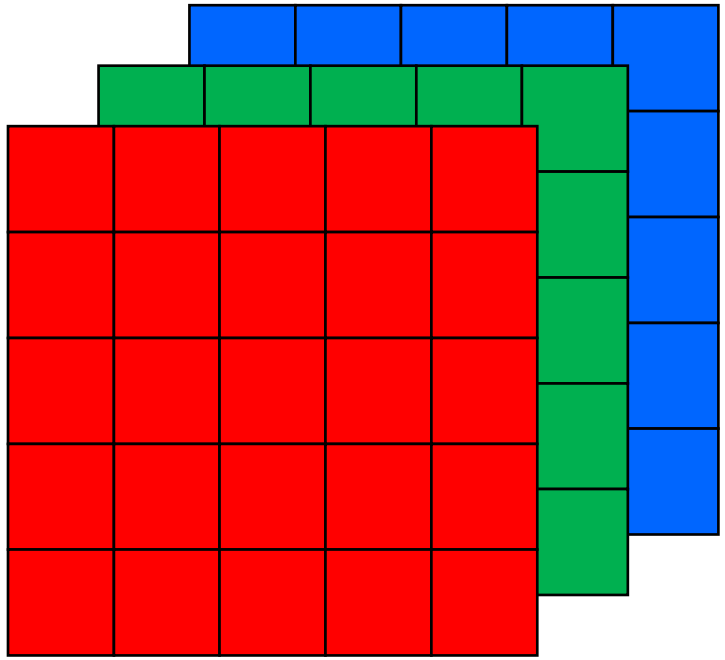
- A CNN model computes loss function using the output and the training data.

  The model learns the weights from the training data.

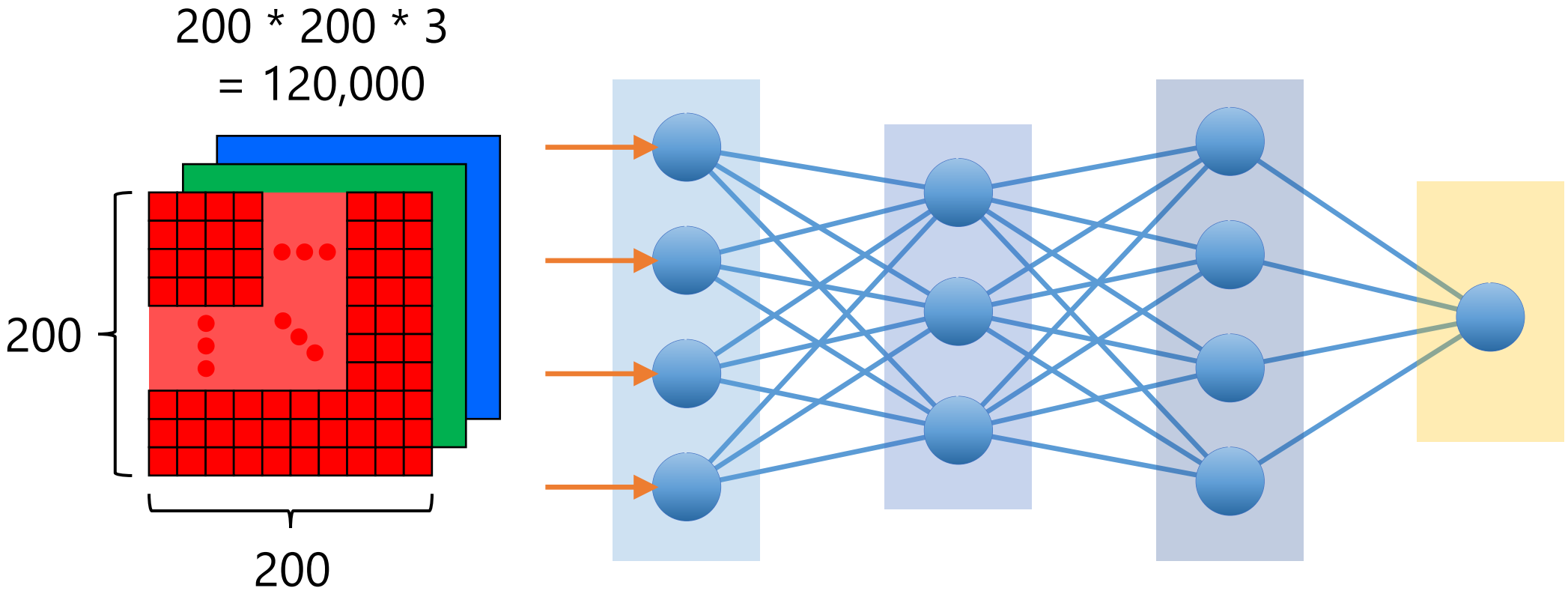
Weakness of ANN (1)

$5 * 5 * \underline{\mathbf{3}} = 75$

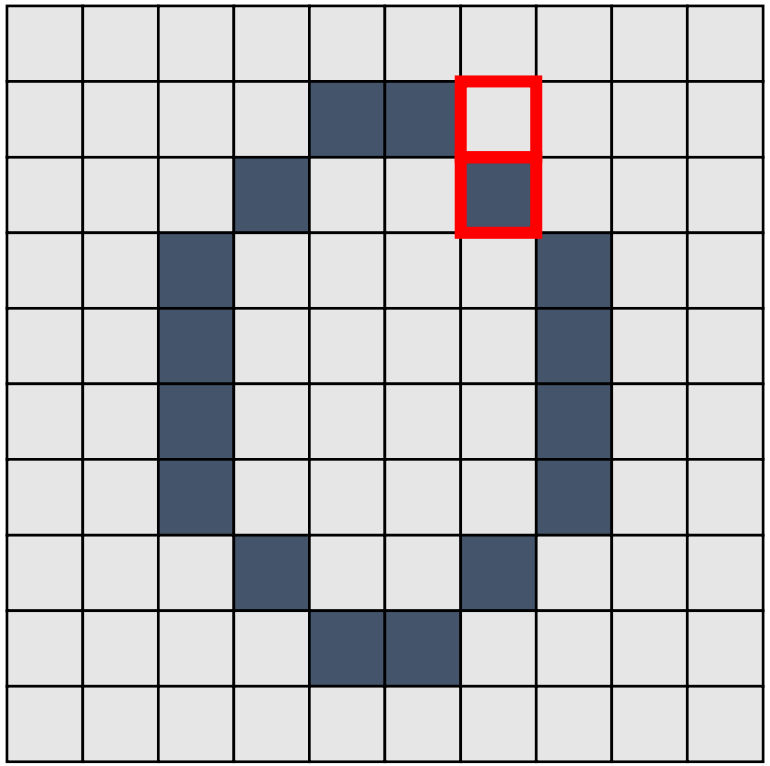
Red
Green
Blue

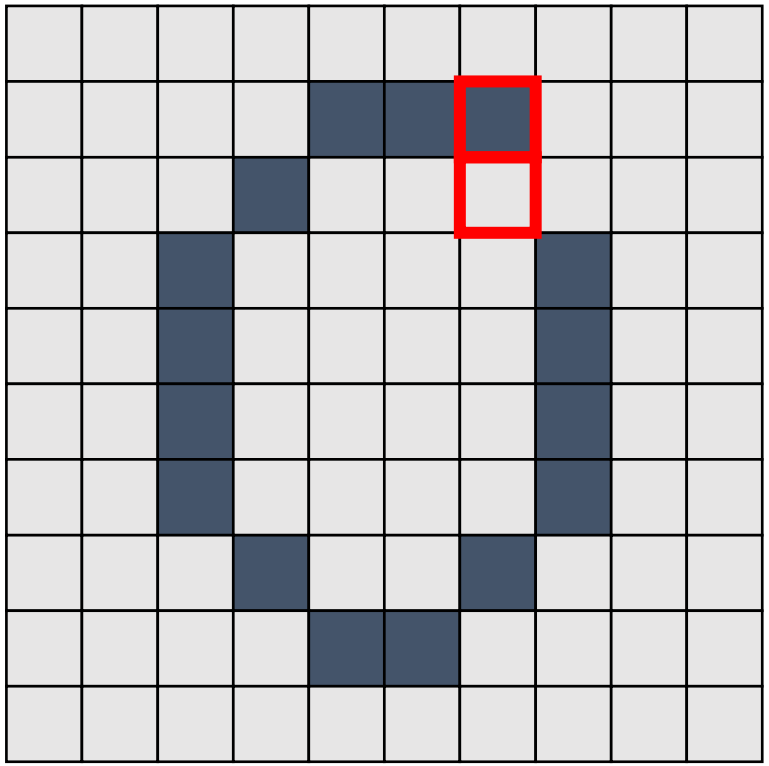


Weakness of ANN (1)

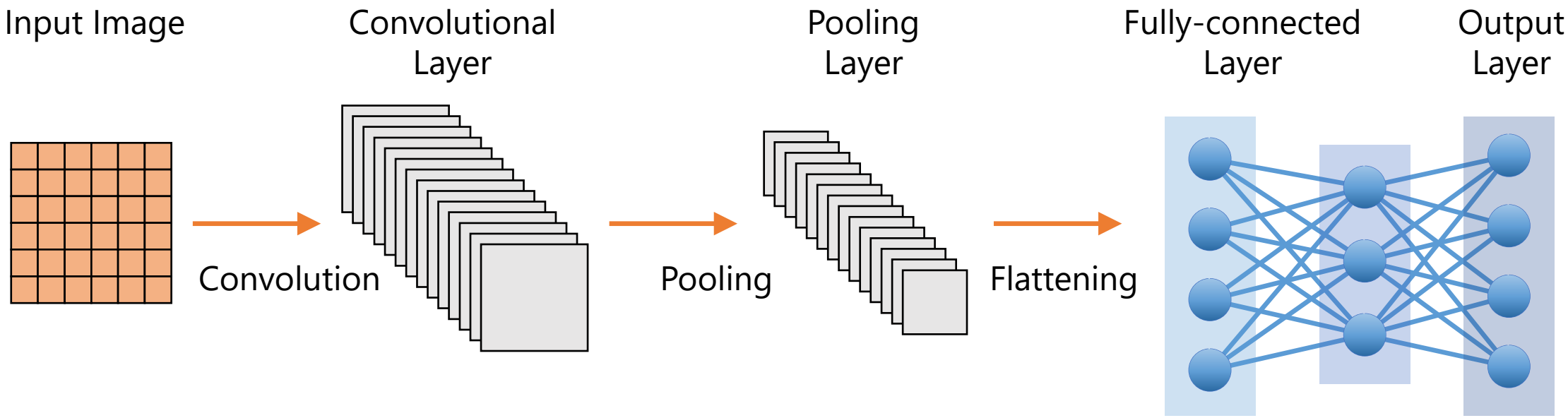


Weakness of ANN (2)



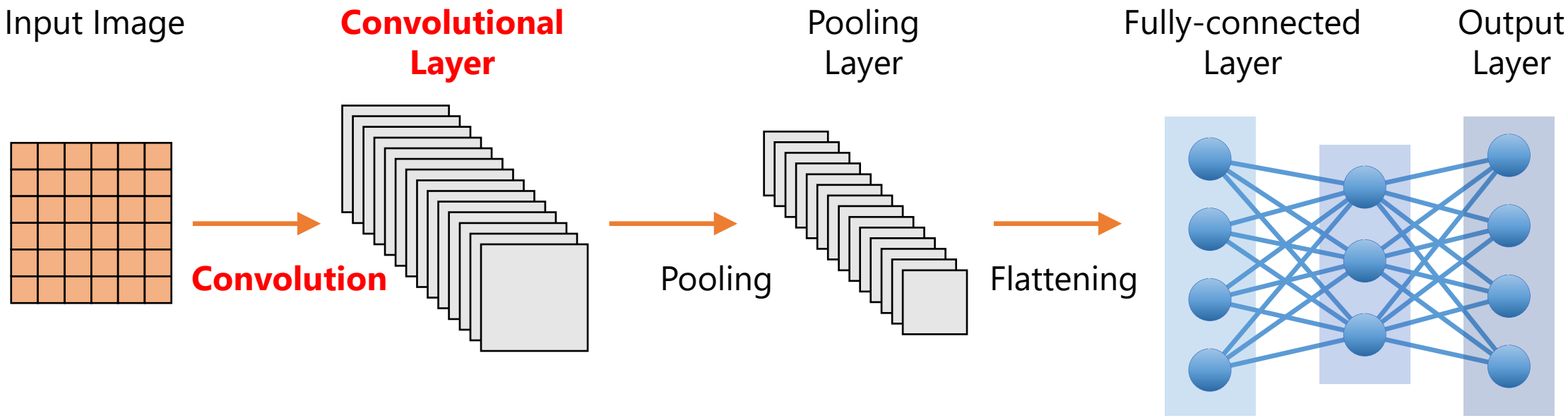


CNN Architecture



4. Convolutional Layer

CNN Architecture



What is Convolution?

Input Image

Information

1	0	0
0	1	0
0	0	1

Filter (Kernel)

Filter

- A set of weights with a shape of (usually) square or rectangle.

0	1	0
0	1	0
0	1	0

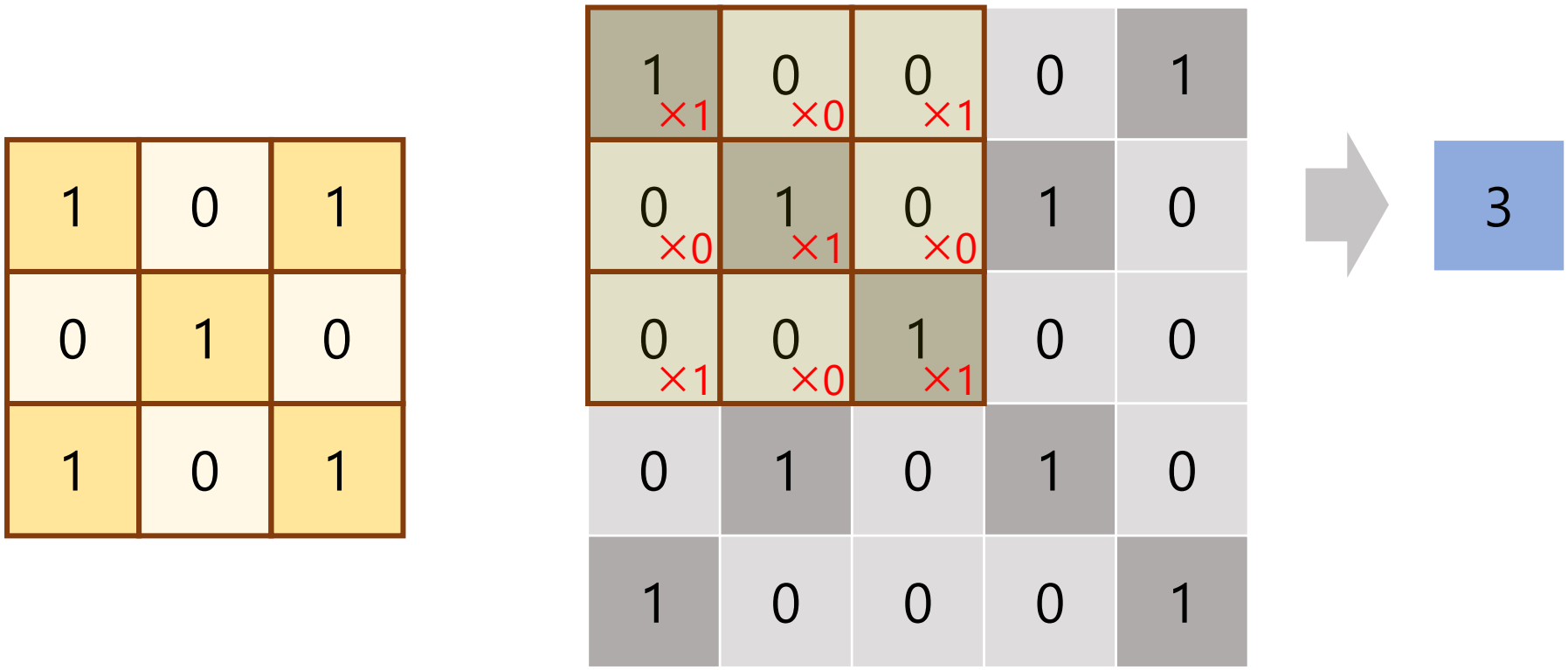
Vertical Line

0	0	0
1	1	1
0	0	0

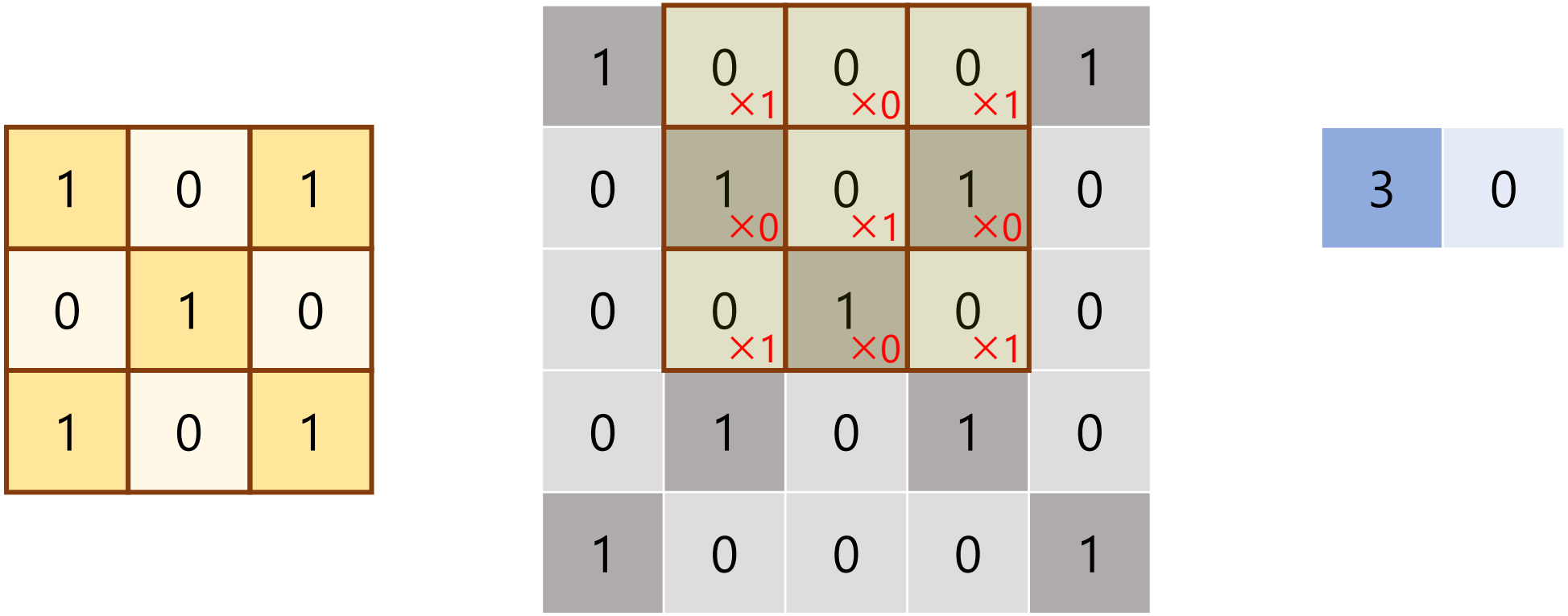
Horizontal Line

We can use a filter to measure the similarity between the image and the filter.

Example: Convolution



Example: Convolution



Example: Convolution

1	0	1
0	1	0
1	0	1

1	0	0	0	1
0	1	0	1	0
0	0	1 ×1	0 ×0	0 ×1
0	1	0 ×0	1 ×1	0 ×0
1	0	0 ×1	0 ×0	1 ×1

Feature map

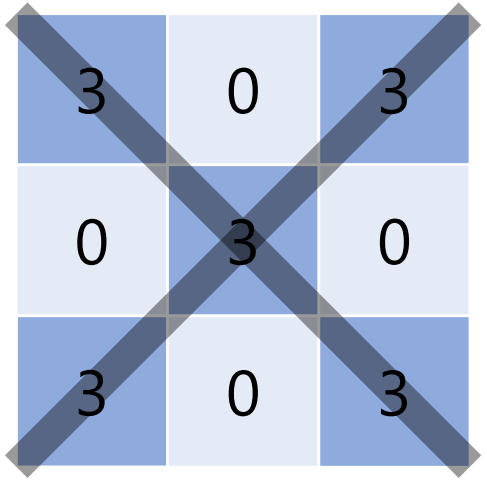
3	0	3
0	3	0
3	0	3

Example: Convolution

1	0	1
0	1	0
1	0	1

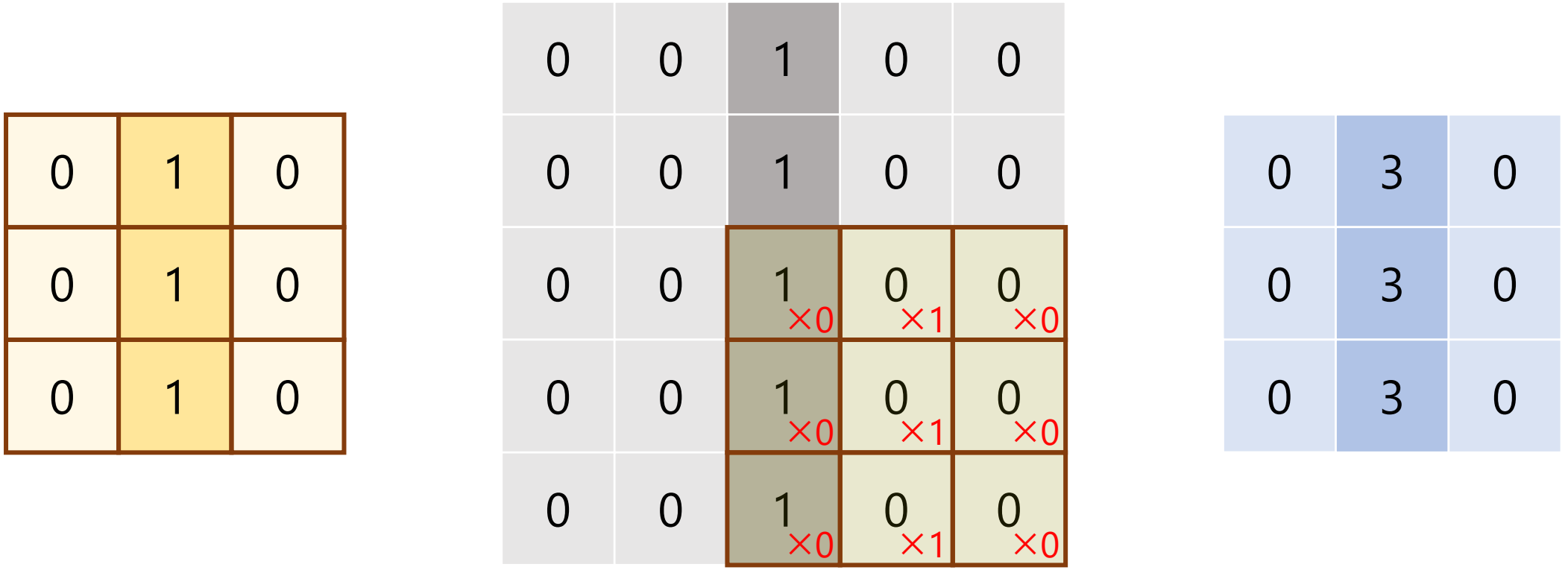
1	0	0	0	1
0	1	0	1	0
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1

Feature map



Example 2: Convolution

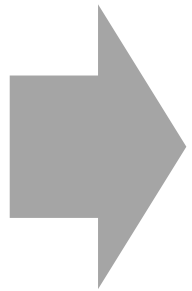
Example 3: Convolution



Locality

- Neighboring pixels are likely to be similar in their values,

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



0	3	0
0	3	0
0	3	0

Filter Size

0	1	0
0	1	0
0	1	0

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

0	3	0
0	3	0
0	3	0

Filter Size

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

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

4	4
4	4

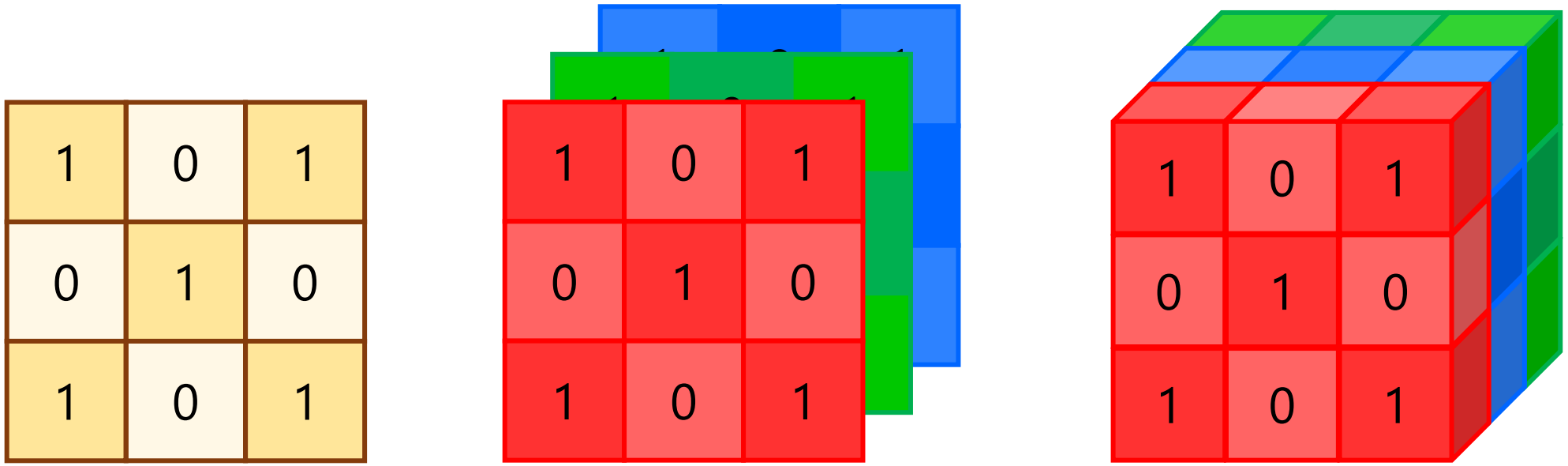
Filter Size

1	0
1	0

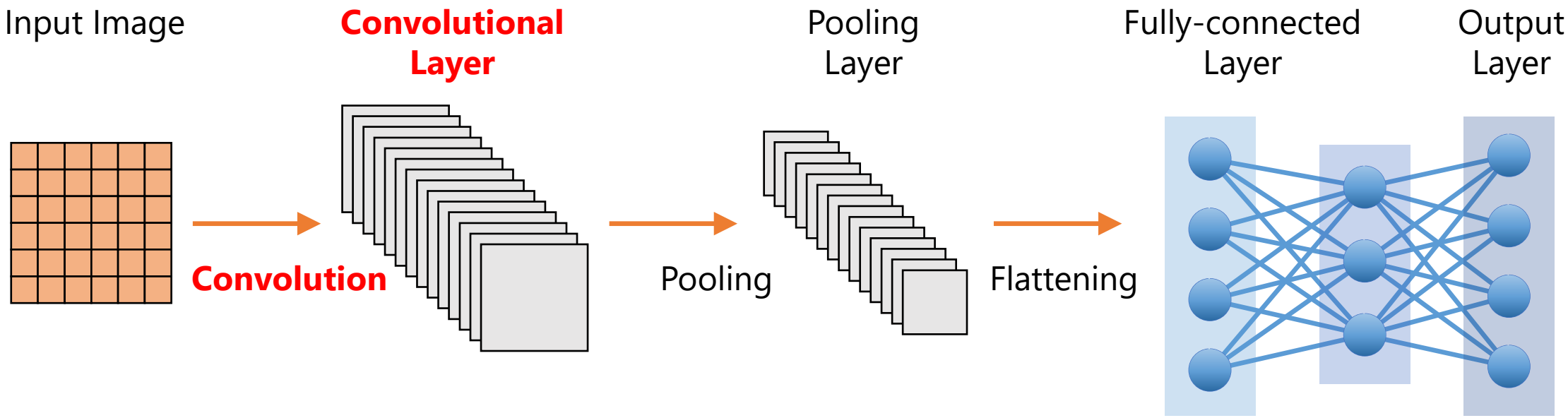
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0

0	0	2	0
0	0	2	0
0	0	2	0
0	0	2	0

Filter for Colored Image

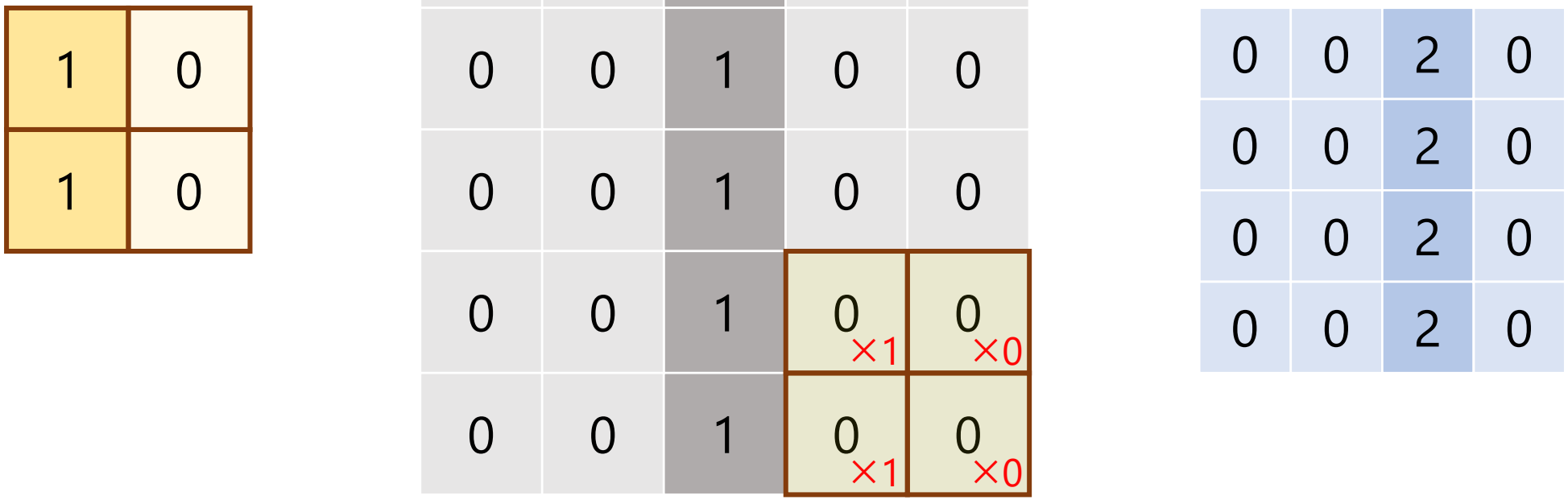


Use Many Filters

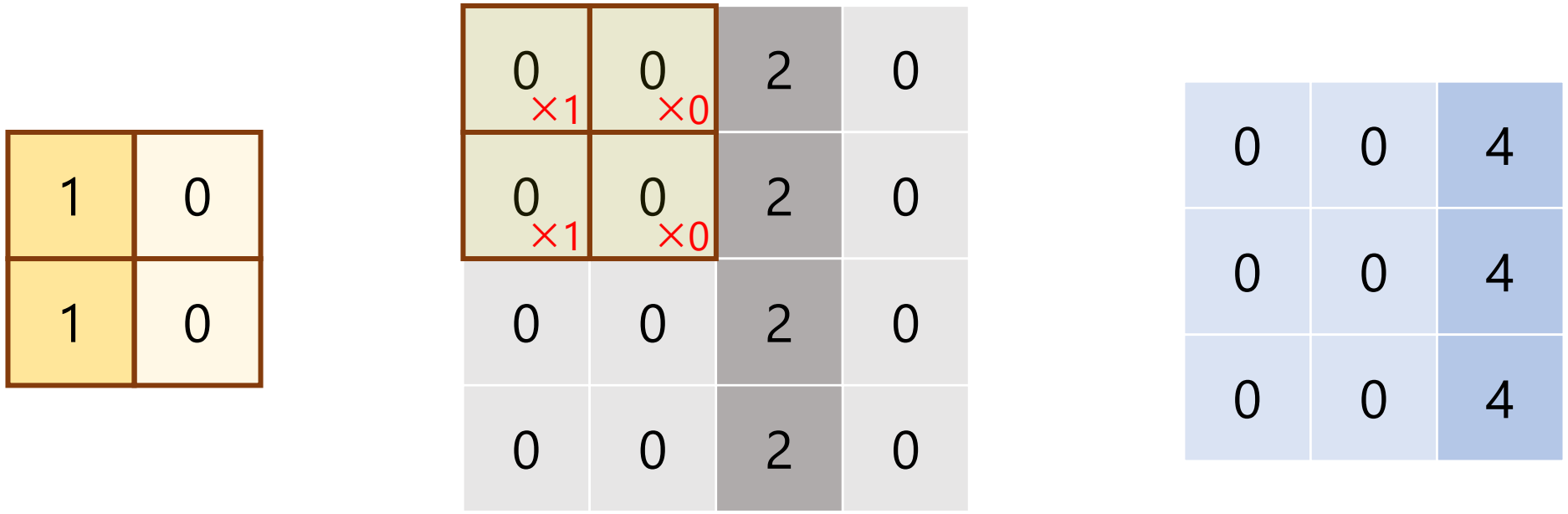


5. Padding

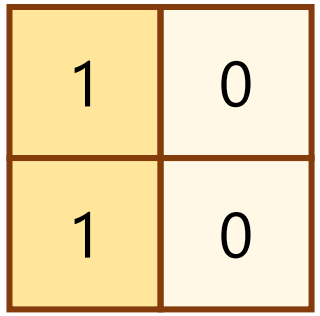
CNN Problem 1

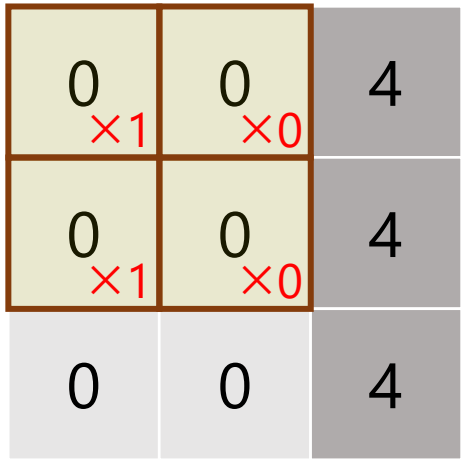


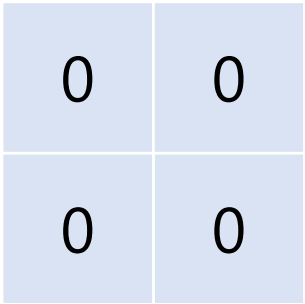
CNN Problem 1



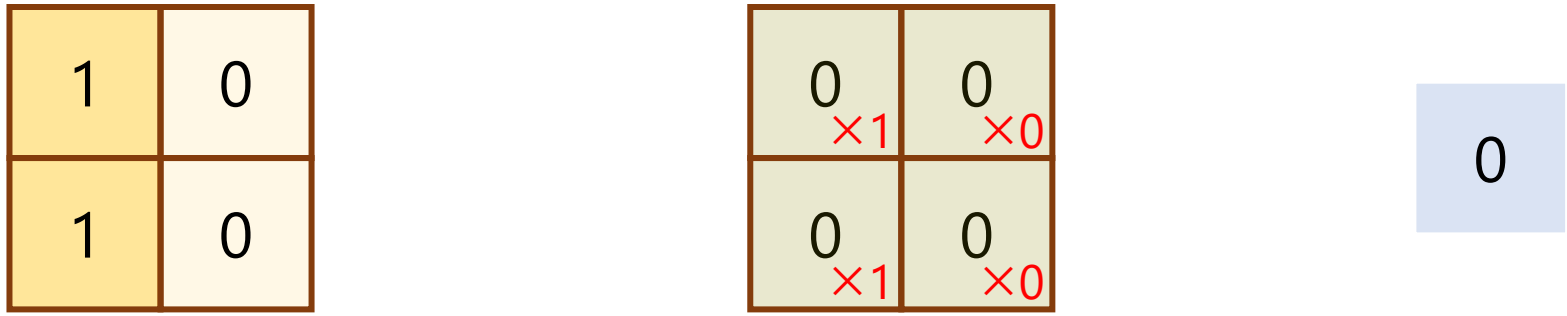
CNN Problem 1



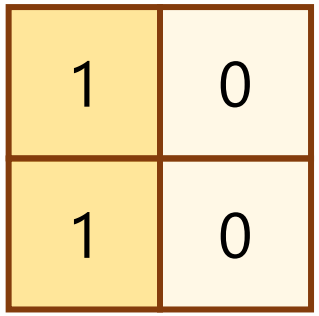




CNN Problem 1



CNN Problem 2

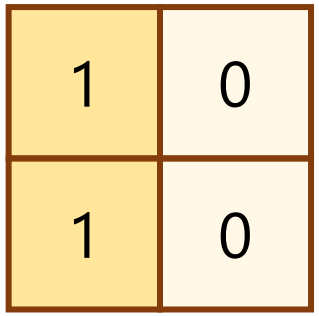


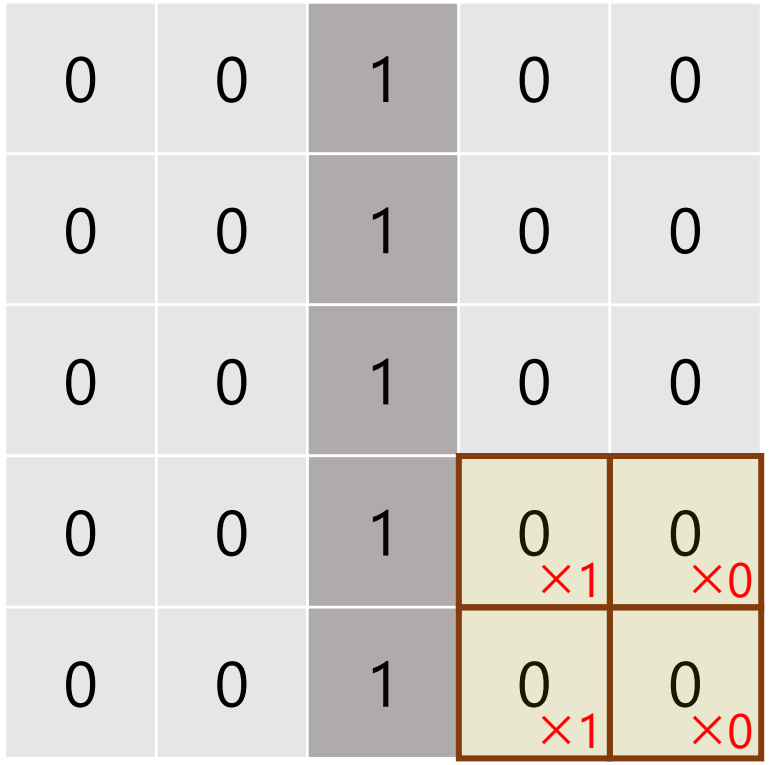
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0

How many times?

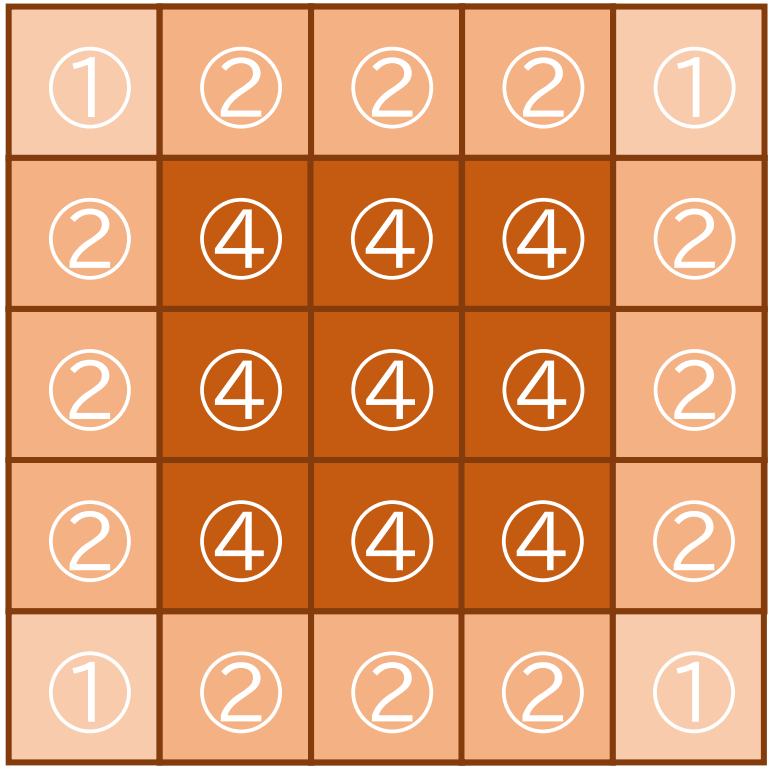
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0

CNN Problem 2



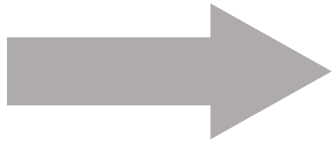


How many times?



Padding

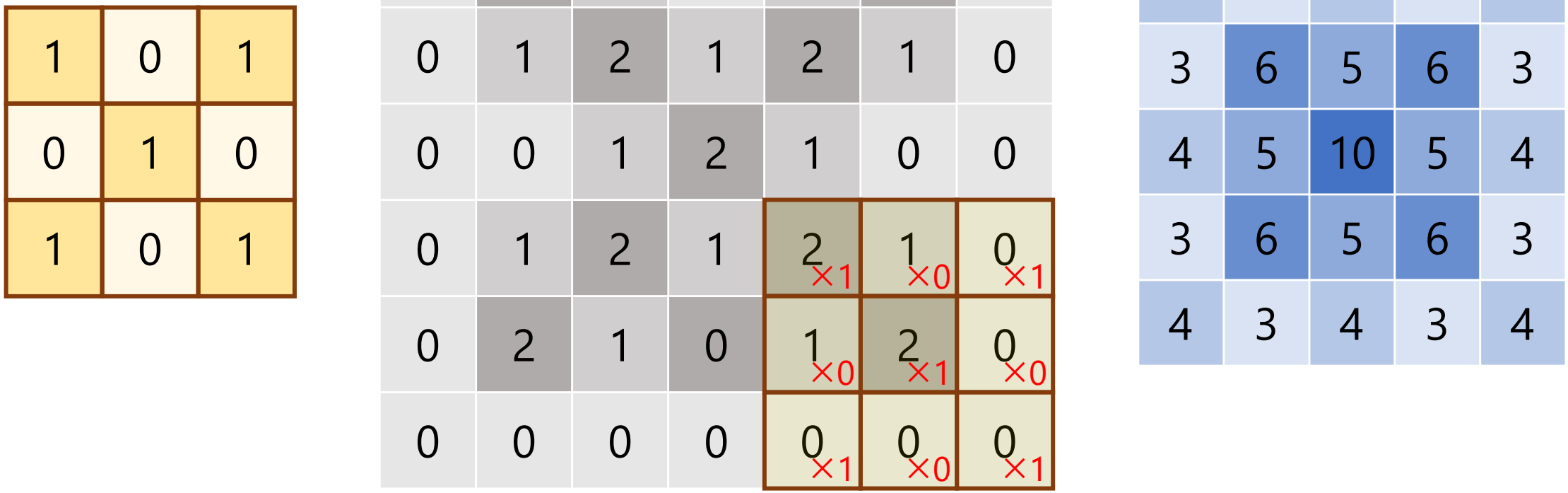
2	1	0	1	2
1	2	1	2	1
0	1	2	1	0
1	2	1	2	1
2	1	0	1	2



Padding

0	0	0	0	0	0	0
0	2	1	0	1	2	0
0	1	2	1	2	1	0
0	0	1	2	1	0	0
0	1	2	1	2	1	0
0	2	1	0	1	2	0
0	0	0	0	0	0	0

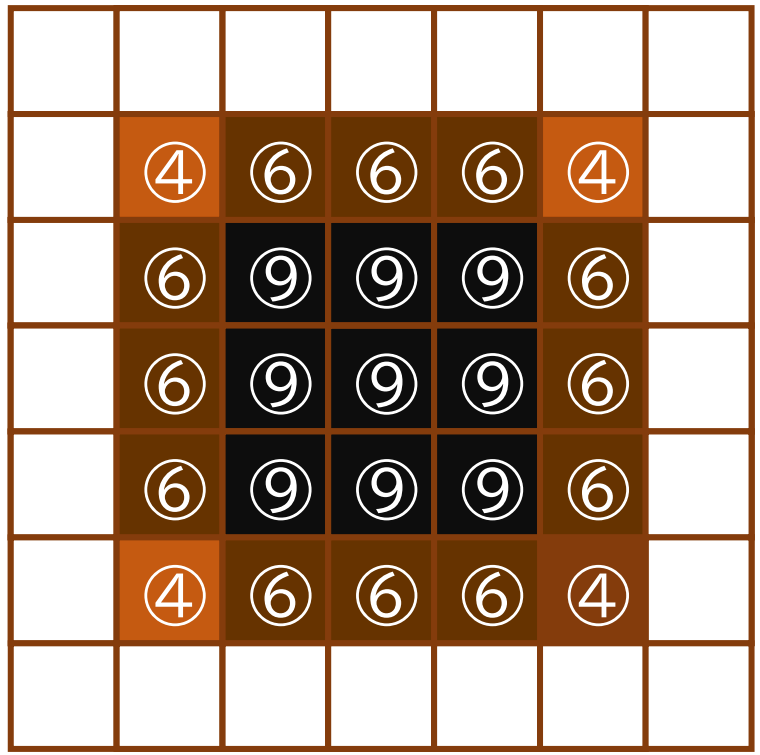
Padding Effect 1



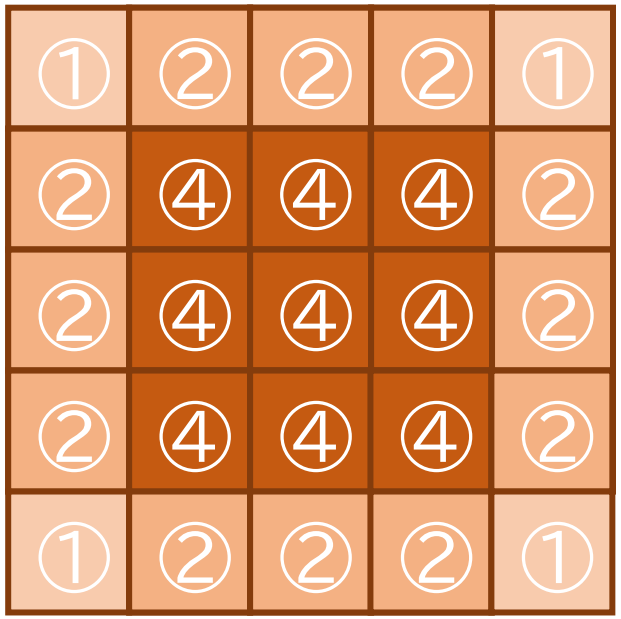
Padding Effect 2

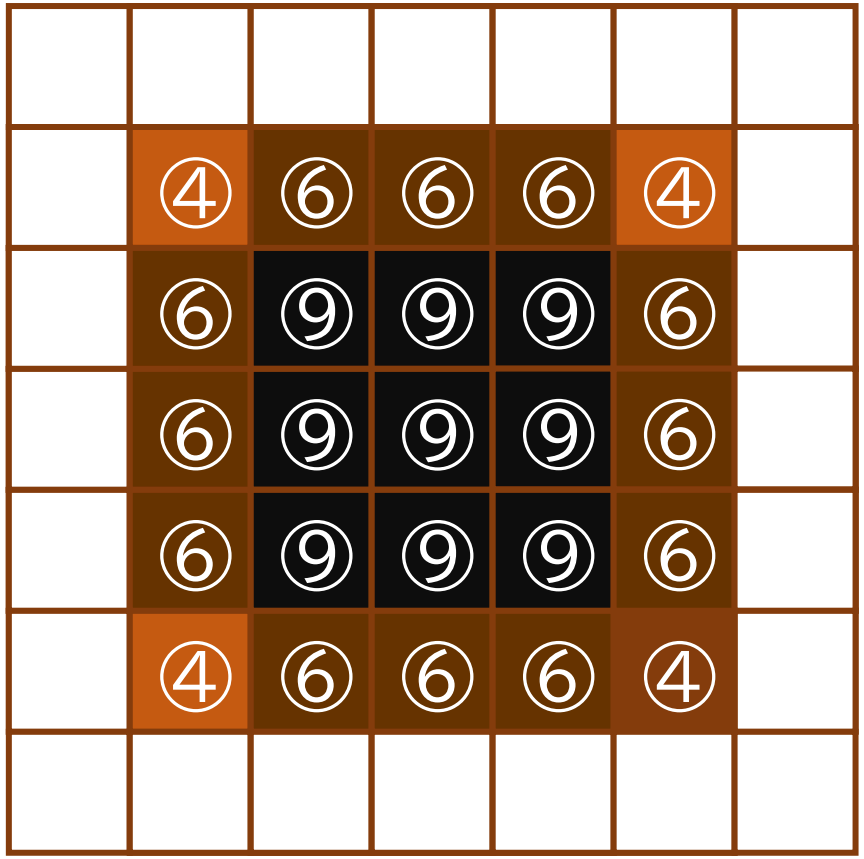
1	0	1
0	1	0
1	0	1

0 ×1	0 ×0	0 ×1	0	0	0	0
0 ×0	2 ×1	1 ×0	0	1	2	0
0 ×1	1 ×0	2 ×1	1	2	1	0
0	0	1	2	1	0	0
0	1	2	1	2	1	0
0	2	1	0	1	2	0
0	0	0	0	0	0	0



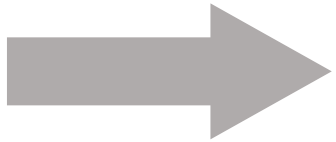
Padding Effect 2





Drawback of Zero Padding

2	1	0	1	2
1	2	1	2	1
0	1	2	1	0
1	2	1	2	1
2	1	0	1	2



Padding

0	0	0	0	0	0	0
0	2	1	0	1	2	0
0	1	2	1	2	1	0
0	0	1	2	1	0	0
0	1	2	1	2	1	0
0	2	1	0	1	2	0
0	0	0	0	0	0	0

Stride

- Stride is the number of pixels that a filter shifts for one move.

Stride = 1

1	0	1
0	1	0
1	0	1

0 ×1	0 ×0	0 ×1	0	0	0	0
0 ×0	2 ×1	1 ×0	0	1	2	0
0 ×1	1 ×0	2 ×1	1	2	1	0
0	0	1	2	1	0	0
0	1	2	1	2	1	0
0	2	1	0	1	2	0
0	0	0	0	0	0	0

Stride

- Stride is the number of pixels that a filter shifts for one move.

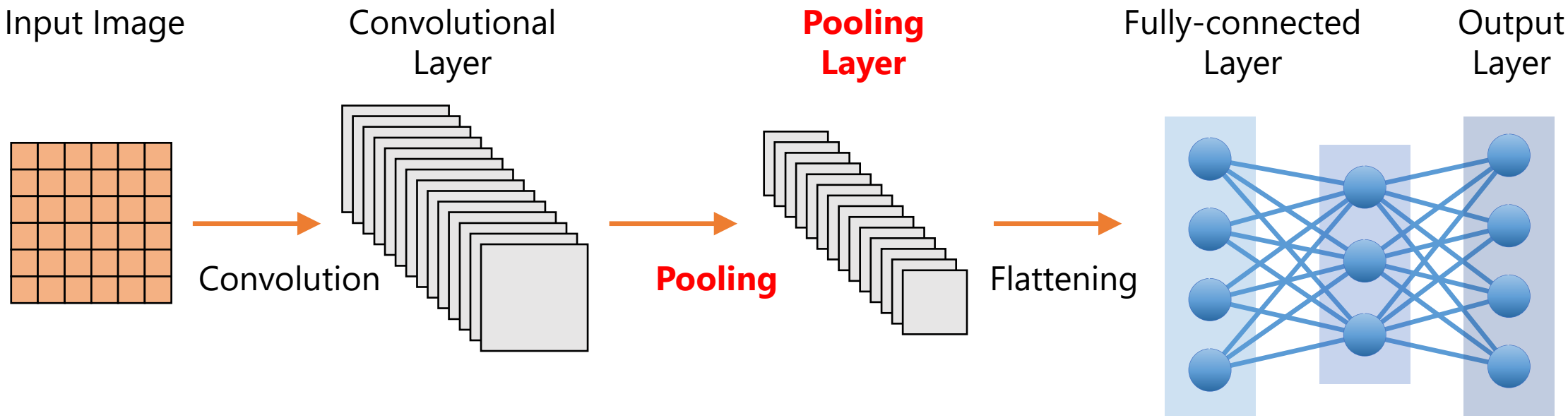
Stride = 2

1	0	1
0	1	0
1	0	1

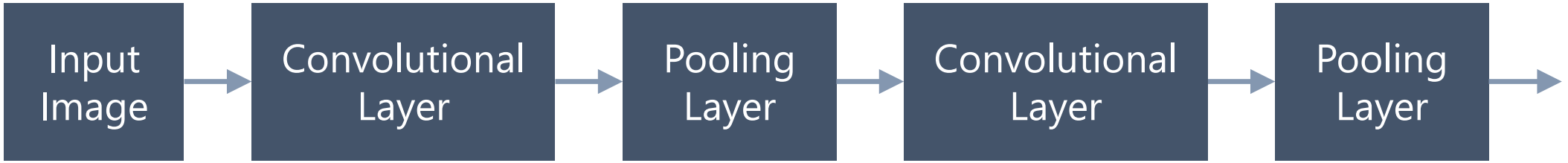
0 ×1	0 ×0	0 ×1	0	0	0	0
0 ×0	2 ×1	1 ×0	0	1	2	0
0 ×1	1 ×0	2 ×1	1	2	1	0
0	0	1	2	1	0	0
0	1	2	1	2	1	0
0	2	1	0	1	2	0
0	0	0	0	0	0	0

6. Pooling

CNN Architecture

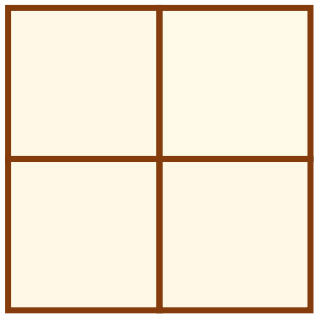


CNN Architecture



Max Pooling

- Pooling by maximum value

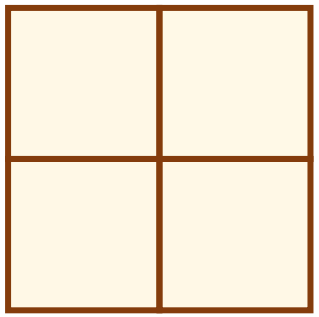


5	2	4	1
3	2	2	9
4	1	1	3
1	2	0	0

5

Max Pooling

- Pooling by maximum value

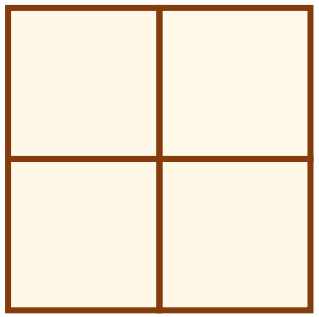


5	2	4	1
3	2	2	9
4	1	1	3
1	2	0	0

5	9
4	3

Average Pooling

- Pooling by average value



5	2	4	1
3	2	2	9
4	1	1	3
1	2	0	0

$$\frac{5+2+3+2}{4} \qquad \frac{4+1+2+9}{4}$$

3	4
2	1

$$\frac{4+1+1+2}{4} \qquad \frac{1+3+0+0}{4}$$

Example: Pooling

0	0	0	0	0	0	0	0	0	0	0	0
0	4	4	4	4	4	4	4	4	4	4	0
0	4	8	8	8	8	8	8	8	8	4	0
0	4	8	4	4	4	4	4	4	8	4	0
0	4	8	4	0	0	0	0	4	8	4	0
0	4	8	4	0	0	0	0	4	8	4	0
0	4	8	4	0	0	0	0	4	8	4	0
0	4	8	4	0	0	0	0	4	8	4	0
0	4	8	4	0	0	0	0	4	8	4	0
0	4	8	4	0	0	0	0	4	8	4	0
0	4	4	4	0	0	0	0	4	4	4	0
0	0	0	0	0	0	0	0	0	0	0	0

Max Pooling

4	4	4	4	4	4
4	8	8	8	8	4
4	8	0	0	8	4
4	8	0	0	8	4
4	8	0	0	9	4
4	4	0	0	4	4

Average Pooling

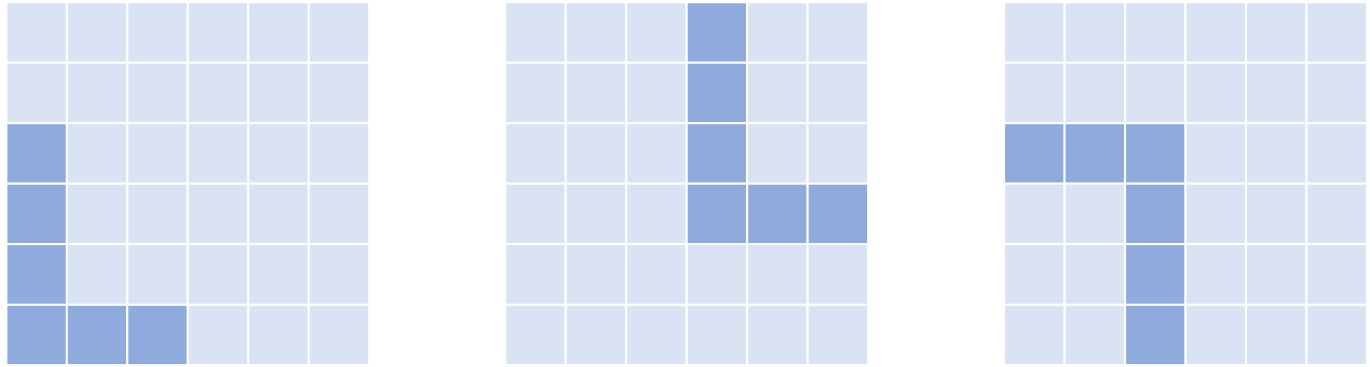
1	2	2	2	2	1
2	7	6	6	7	2
2	6	0	0	6	2
2	6	0	0	6	2
2	6	0	0	6	2
1	2	0	0	2	1

Benefits of Pooling

- Reduce information while keeping important features.

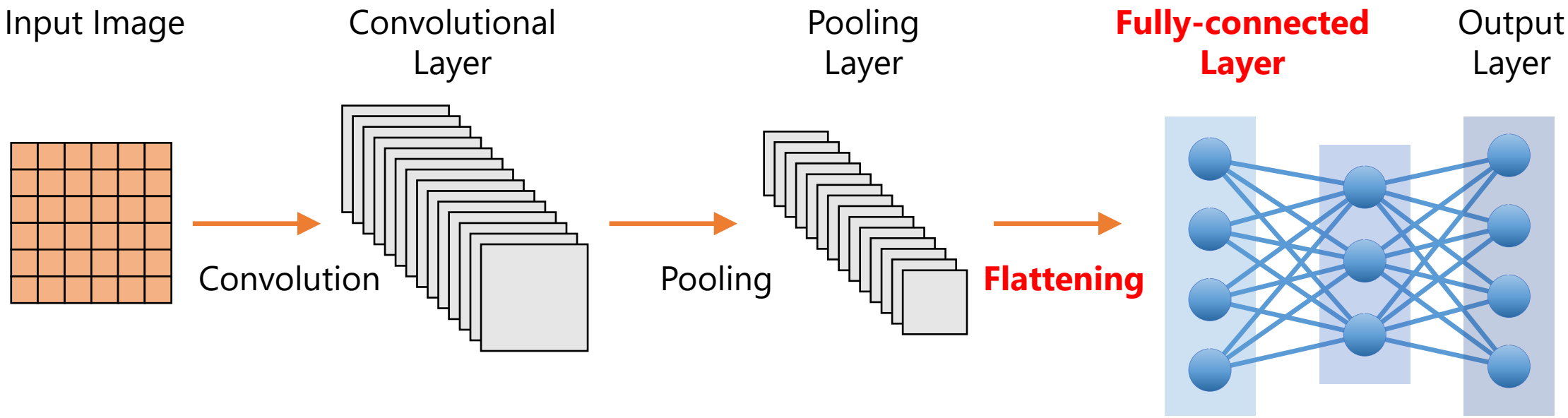
  Make the computation more efficient

- Reduction in sensitivity for location information.



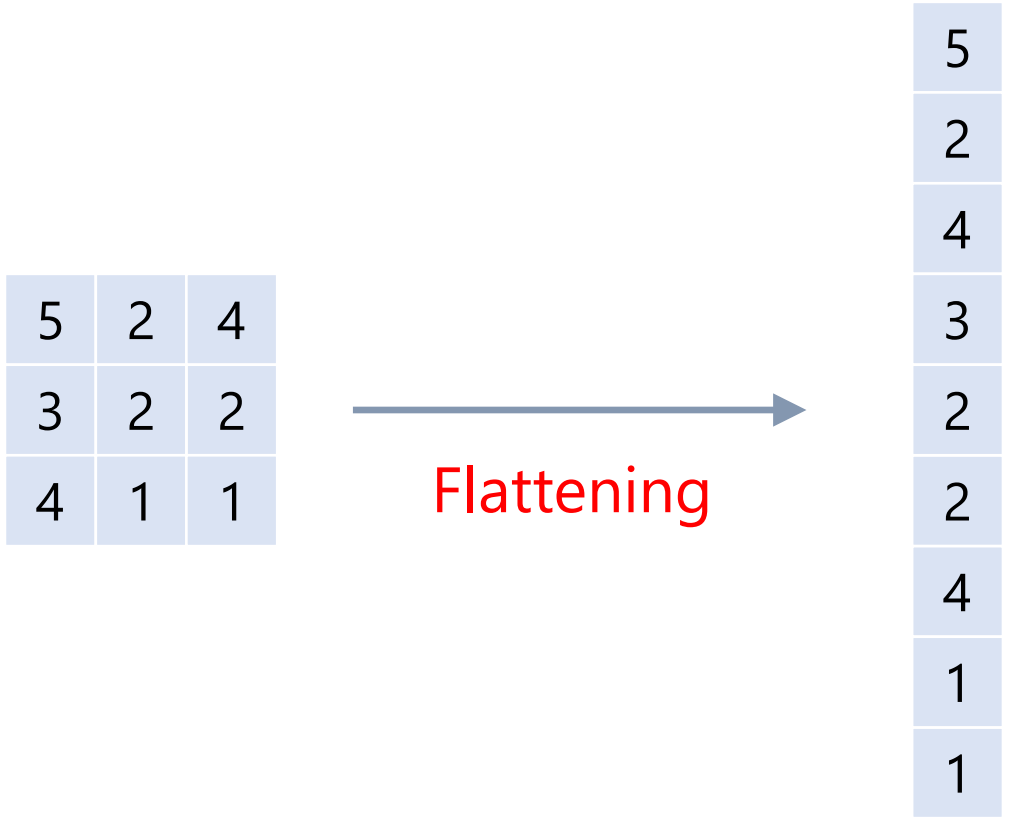
7. Fully-Connected Layer

CNN Architecture

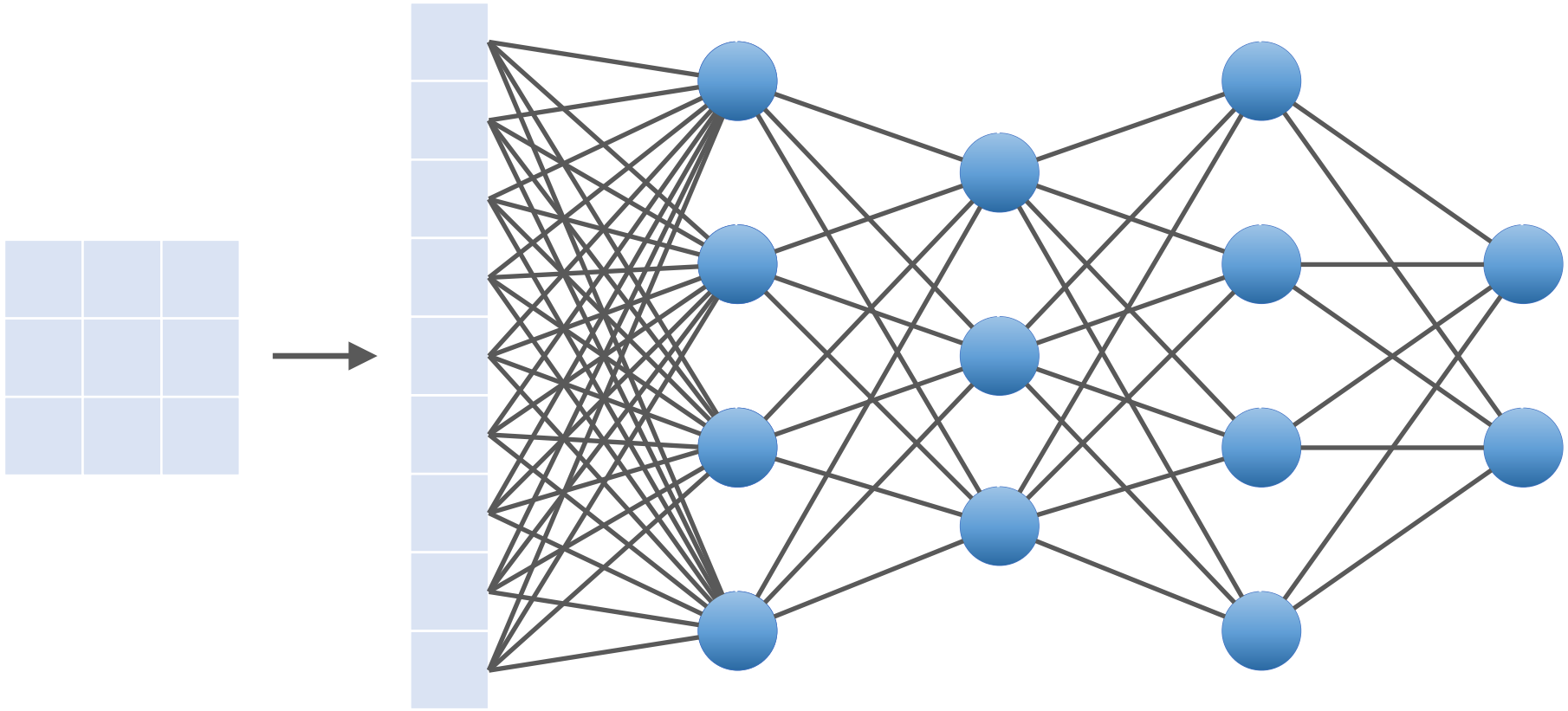


Flattening

- Transform feature maps into vectors.

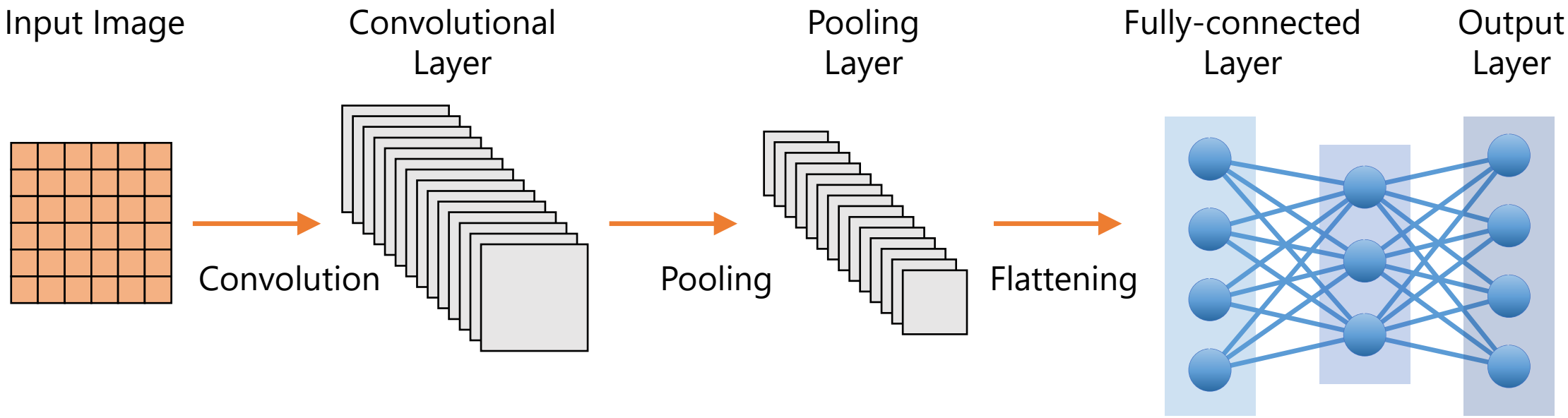


Fully-Connected Layer

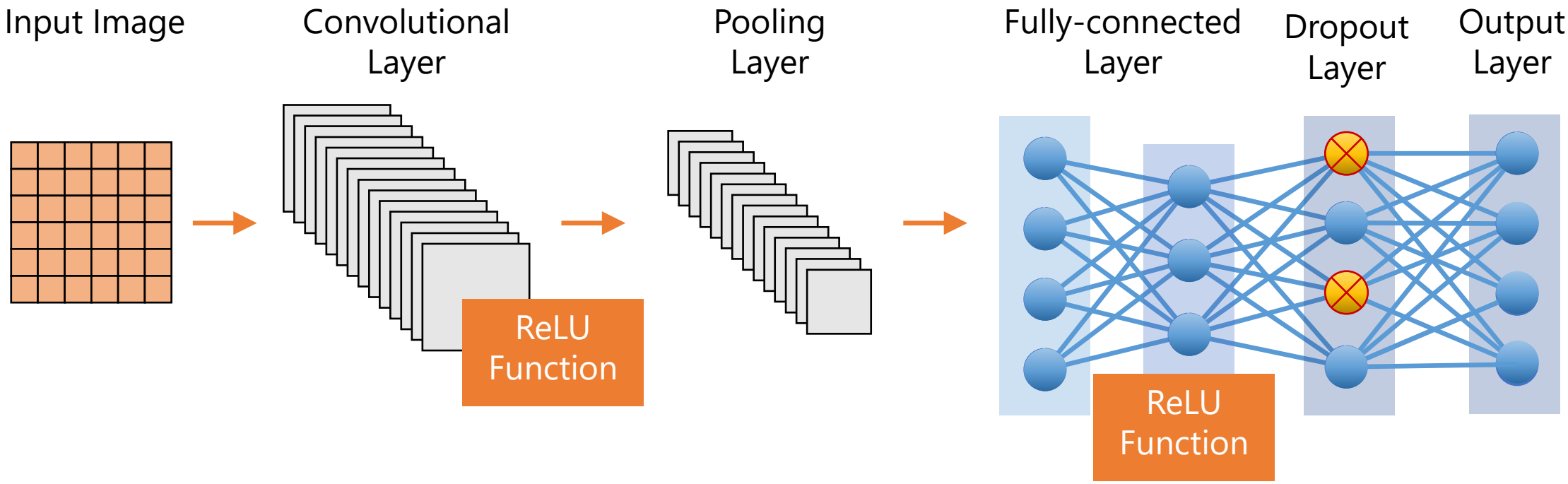


8. CNN Training Overview

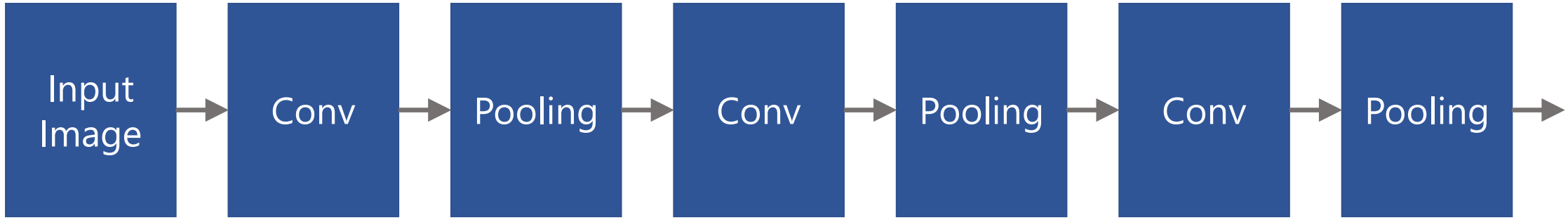
CNN Architecture



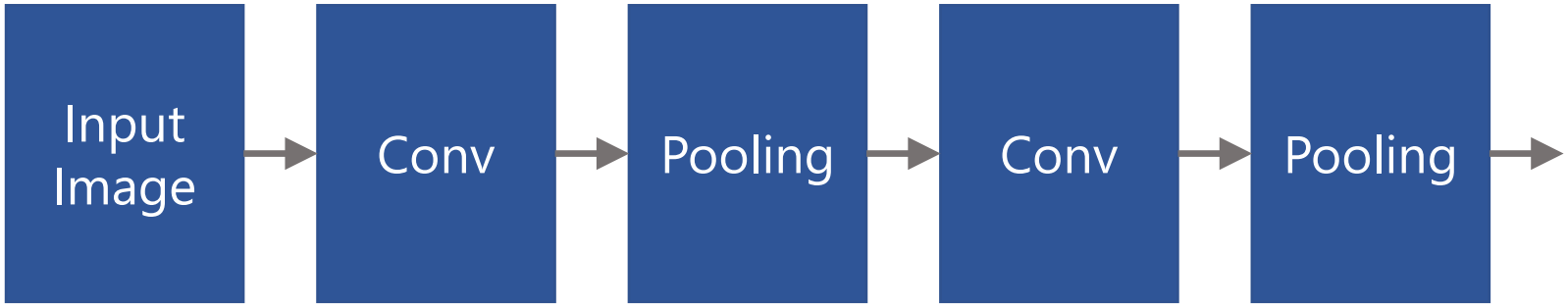
Activation Function and Dropout Layer



How Many Layers?



How Many Layers?



9. Image Data Augmentation

Image Data Augmentation

- A technique to increase the amount of data by creating transformed versions of images in the training data.

  e.g., shifting, flipping, rotating, and zooming, etc.

- Image data augmentation can be used to

  - Improve the model's prediction performance.
  - Prevent overfitting.

Improve Model Performance

- For image data, deep learning models need to learn many parameters to make predictions.

  Such learning needs an enormous amount of data.

- Image data augmentation is useful to increase the amount of data.

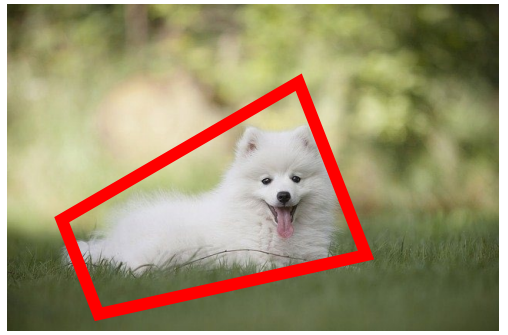
Improve Model Performance

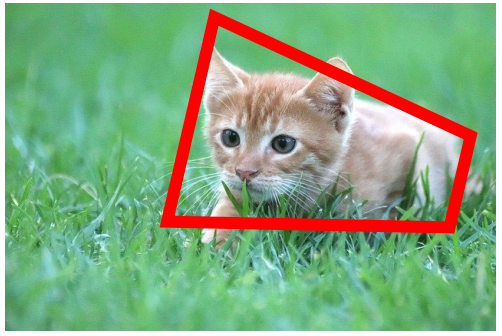
- For image data, deep learning models need to learn many parameters to make predictions.

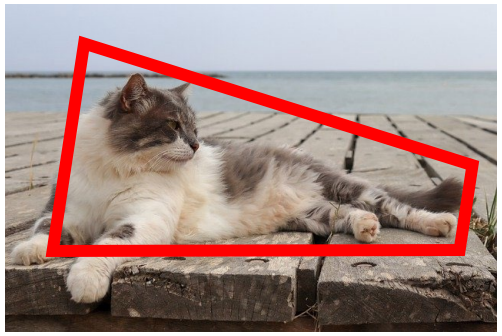
  Such learning needs an enormous amount of data.

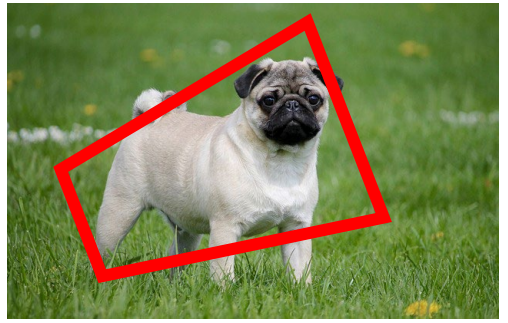
- Image data augmentation is useful to increase the amount of data.

Prevent Overfitting









Prevent Overfitting

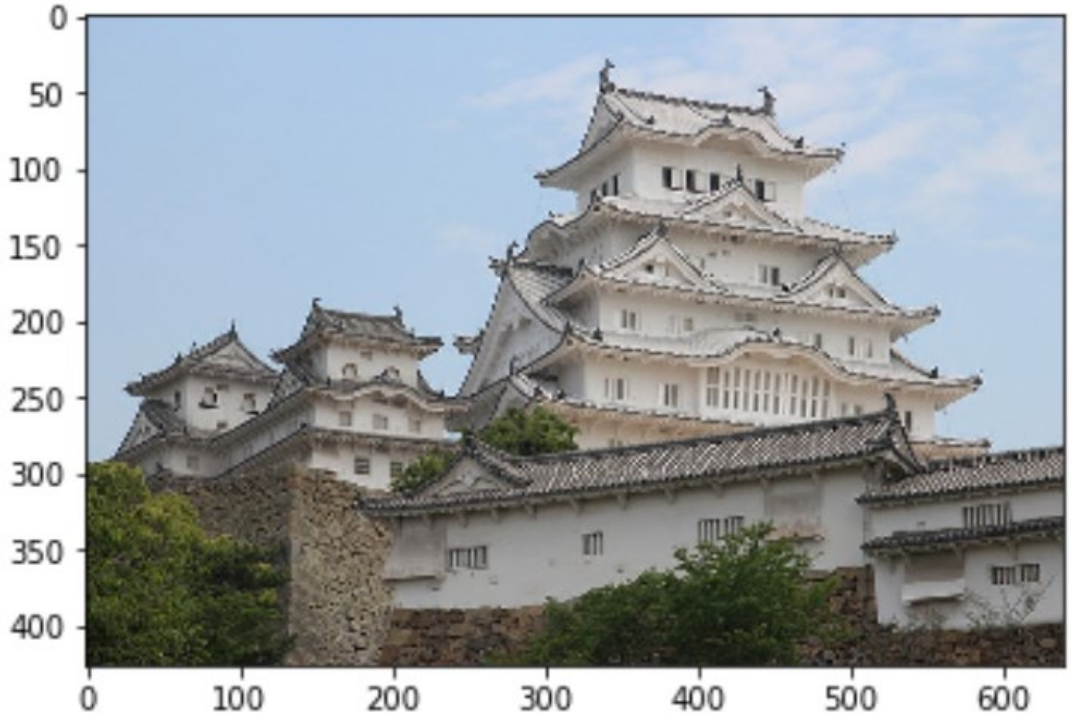
Prevent Overfitting

Image Data Augmentation by Python

```
# Import Libraries
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from numpy import expand_dims
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
%matplotlib inline
```

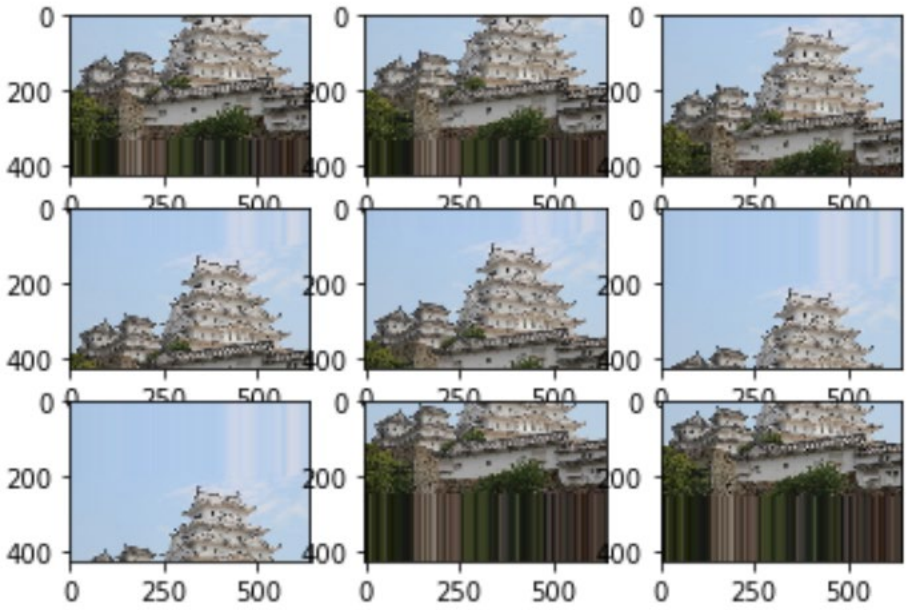
Load and Process the Image

```
# Load and show the sample image
img = load_img('japanese_castle.jpg')
plt.imshow(img)

# Convert the image to numpy array
img_data = img_to_array(img)

# Expand dimension to four dimensions:
batch size, height, width, channel
img_data = expand_dims(img_data, axis=0)
```



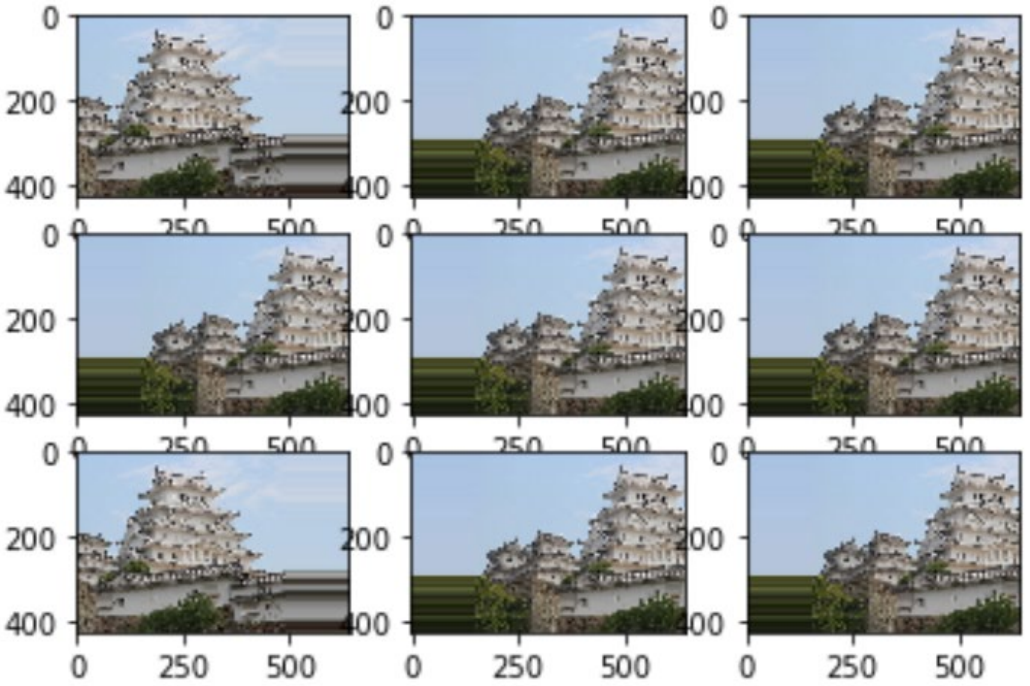
Vertical Shift

```
# Create image data augmentation generator
v_shift_datagen = ImageDataGenerator(height_shift_range=0.5)

# Generate images
# Set batch size to 1 because we use 1 image here.
v_shift_imgs = v_shift_datagen.flow(img_data, batch_size=1)

# Plot 9 augmented images
for i in range(9):
    # define subplot
    plt.subplot(330 + 1 + i)
    # generate batch of images
    batch = v_shift_imgs.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    plt.imshow(image)
```



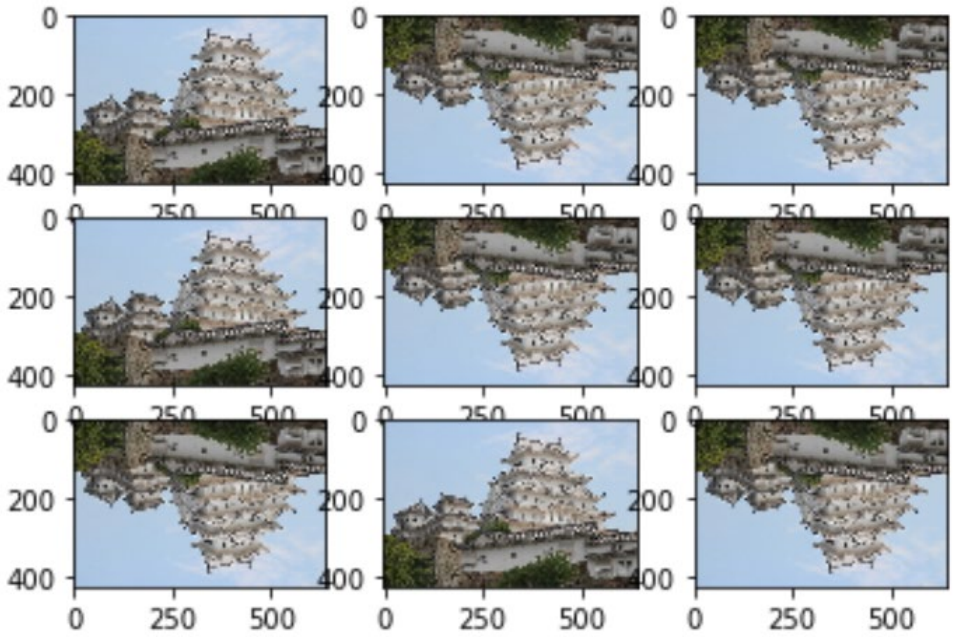
Horizontal Shift

```
# Create image data augmentation generator
h_shift_datagen = ImageDataGenerator(width_shift_range=[-150,150])

# Generate images
h_shift_imgs = h_shift_datagen.flow(img_data, batch_size=1)

# Plot 9 augmented images
for i in range(9):
    # define subplot
    plt.subplot(330 + 1 + i)
    # generate batch of images
    batch = h_shift_imgs.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    plt.imshow(image)
```



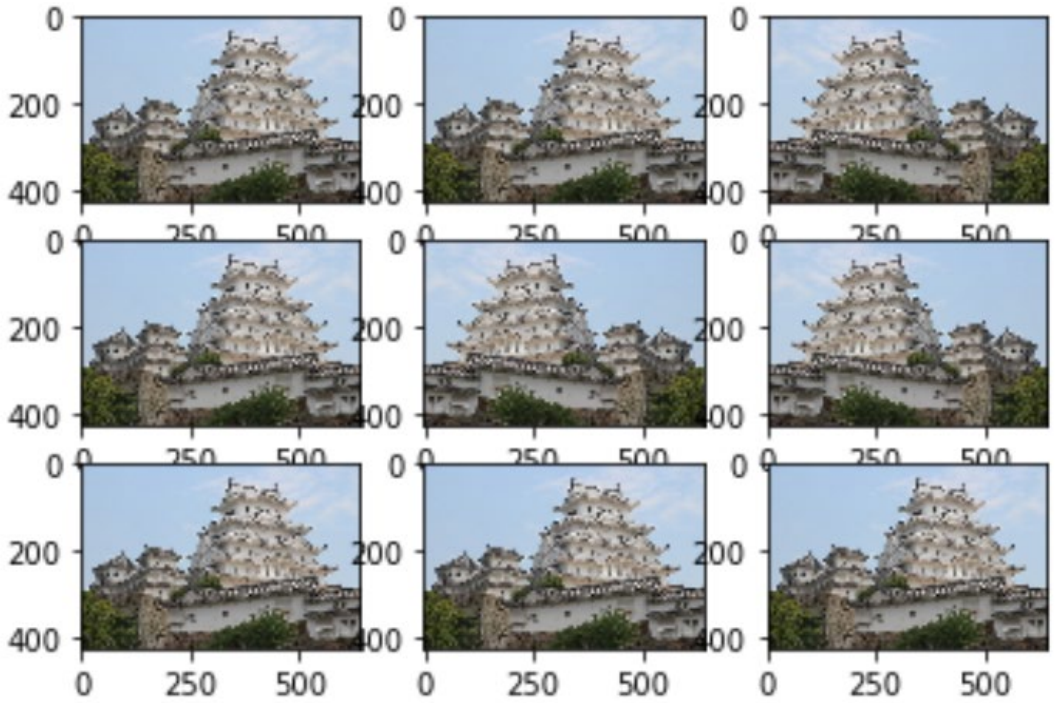
Vertical Flip

```
# Create image data augmentation generator
v_flip_datagen = ImageDataGenerator(vertical_flip=True)

# Generate images
v_flip_imgs = v_flip_datagen.flow(img_data, batch_size=1)

# Plot 9 augmented images
for i in range(9):
    # define subplot
    plt.subplot(330 + 1 + i)
    # generate batch of images
    batch = v_flip_imgs.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    plt.imshow(image)
```



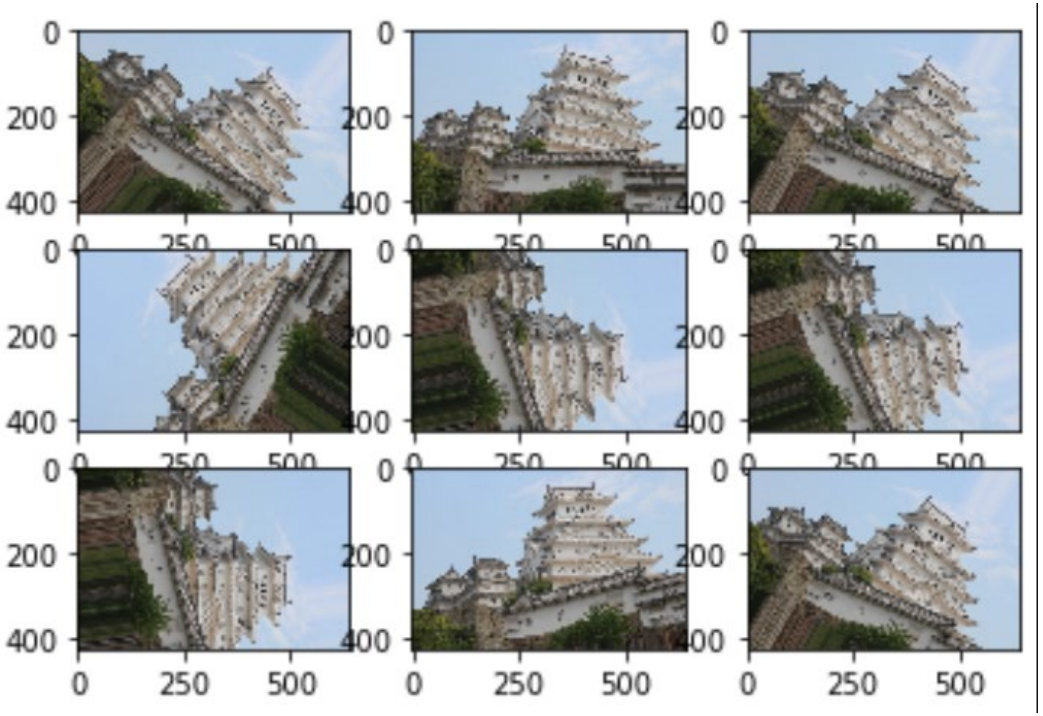
Horizontal Flip

```
# Create image data augmentation generator
h_flip_datagen = ImageDataGenerator(horizontal_flip=True)

# Generate images
h_flip_imgs = h_flip_datagen.flow(img_data, batch_size=1)

# Plot 9 augmented images
for i in range(9):
    # define subplot
    plt.subplot(330 + 1 + i)
    # generate batch of images
    batch = h_flip_imgs.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    plt.imshow(image)
```



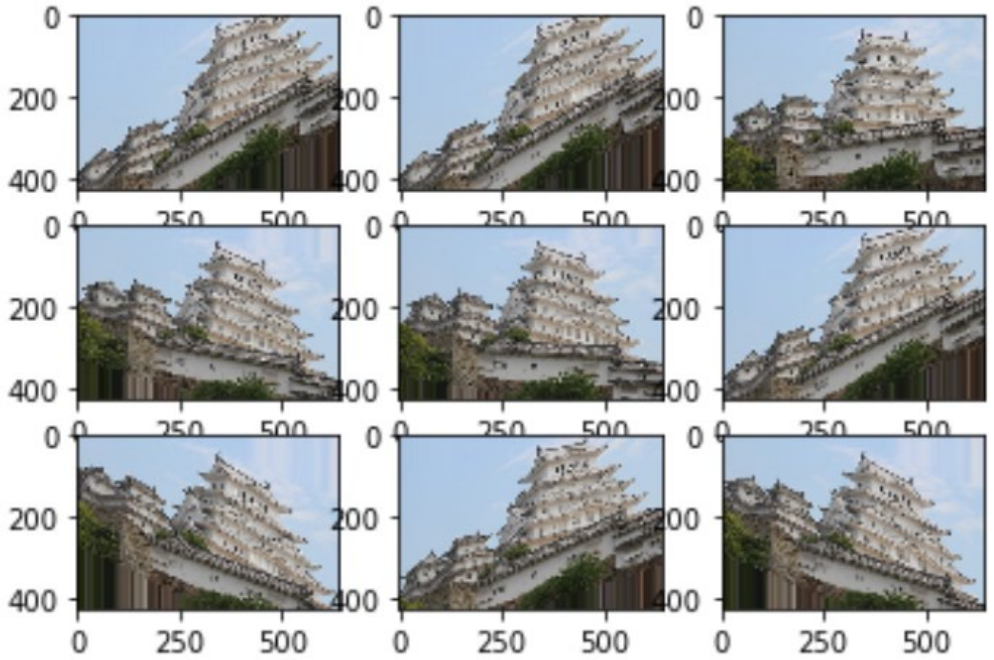
Rotation

```
# Create image data augmentation generator
rot_datagen = ImageDataGenerator(rotation_range=90)

# Generate images
rot_imgs = rot_datagen.flow(img_data, batch_size=1)

# Plot 9 augmented images
for i in range(9):
    # define subplot
    plt.subplot(330 + 1 + i)
    # generate batch of images
    batch = rot_imgs.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    plt.imshow(image)
```



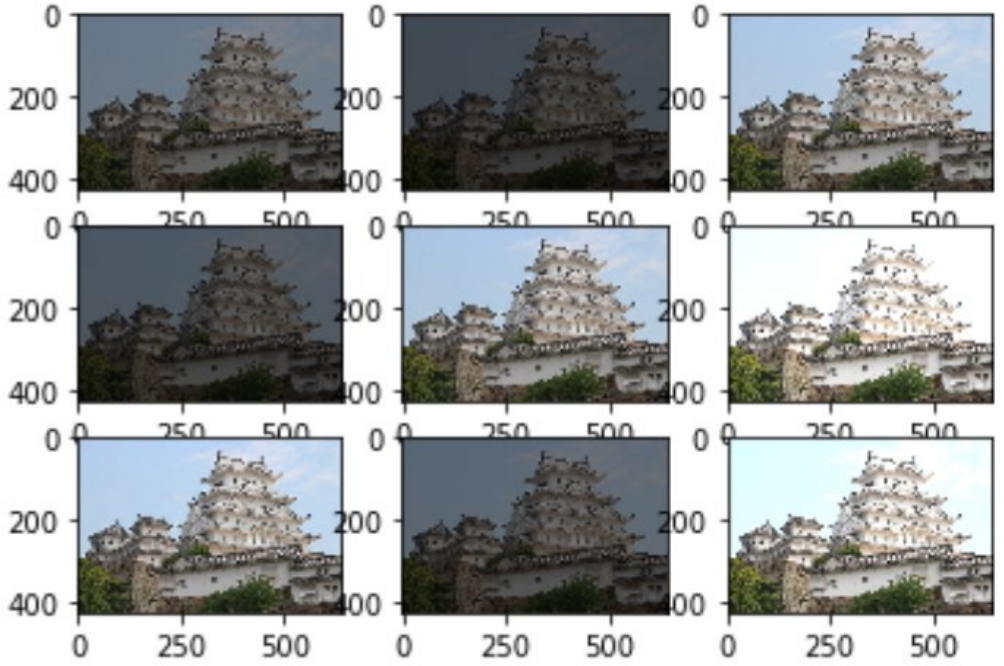
Shear

```
# Create image data augmentation generator
shear_datagen = ImageDataGenerator(shear_range=30)

# Generate images
shear_imgs = shear_datagen.flow(img_data, batch_size=1)

# Plot 9 augmented images
for i in range(9):
    # define subplot
    plt.subplot(330 + 1 + i)
    # generate batch of images
    batch = shear_imgs.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    plt.imshow(image)
```



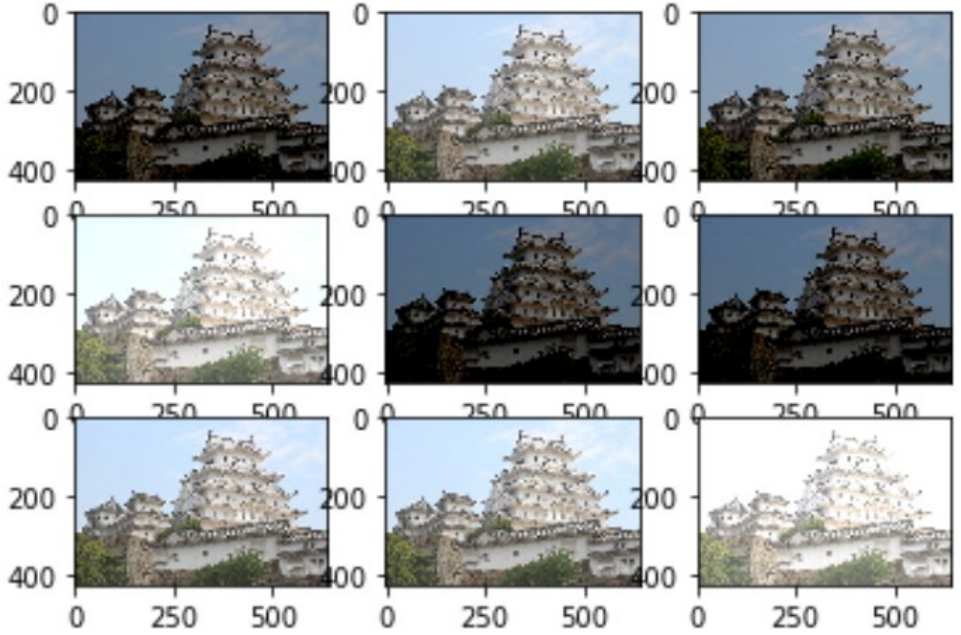
Random Brightness

```
# Create image data augmentation generator
# Range 0.0 = black, 1.0 = No change
bright_datagen = ImageDataGenerator(brightness_range=[0.2, 1.5])

# Generate images
bright_imgs = bright_datagen.flow(img_data, batch_size=1)

# Plot 9 augmented images
for i in range(9):
    # define subplot
    plt.subplot(330 + 1 + i)
    # generate batch of images
    batch = bright_imgs.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    plt.imshow(image)
```



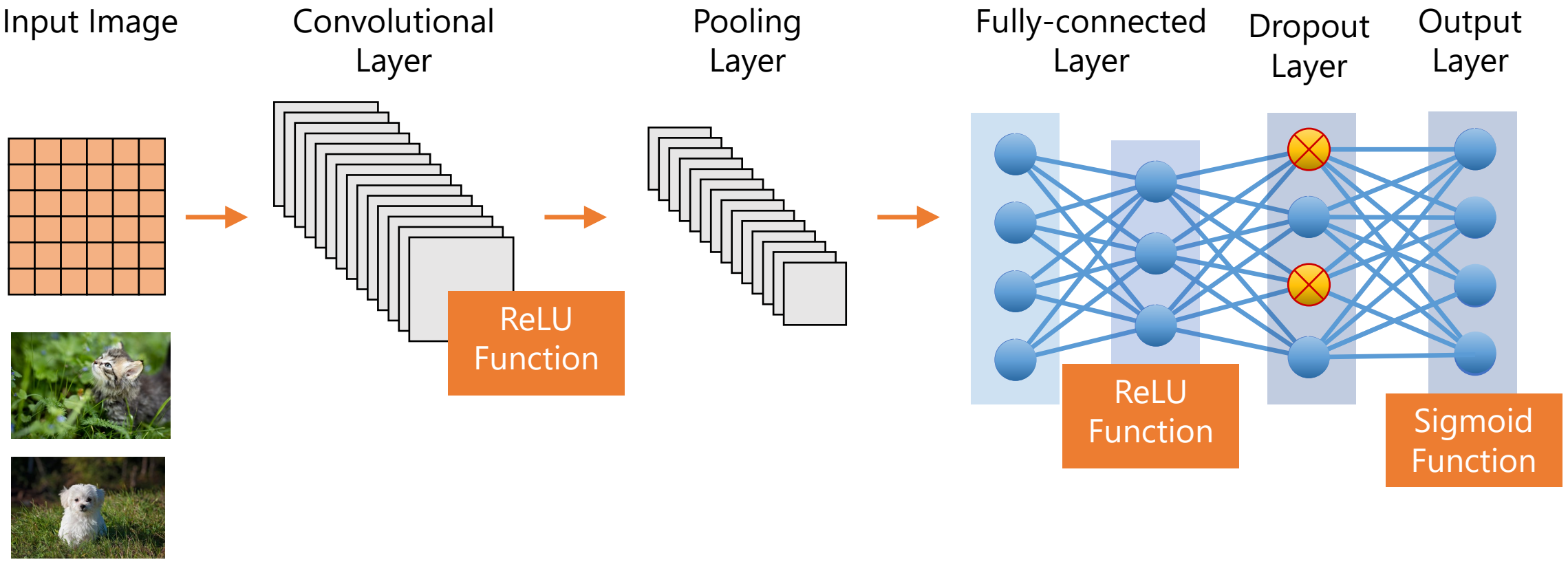
Channel Shift

```
# Create image data augmentation generator
# Randomly shift pixel values -100 to 100
ch_shift_datagen = ImageDataGenerator(channel_shift_range=100)

# Generate images
ch_shift_imgs = ch_shift_datagen.flow(img_data, batch_size=1)

# Plot 9 augmented images
for i in range(9):
    # define subplot
    plt.subplot(330 + 1 + i)
    # generate batch of images
    batch = ch_shift_imgs.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    plt.imshow(image)
```



10. Binary Image Classification with Python

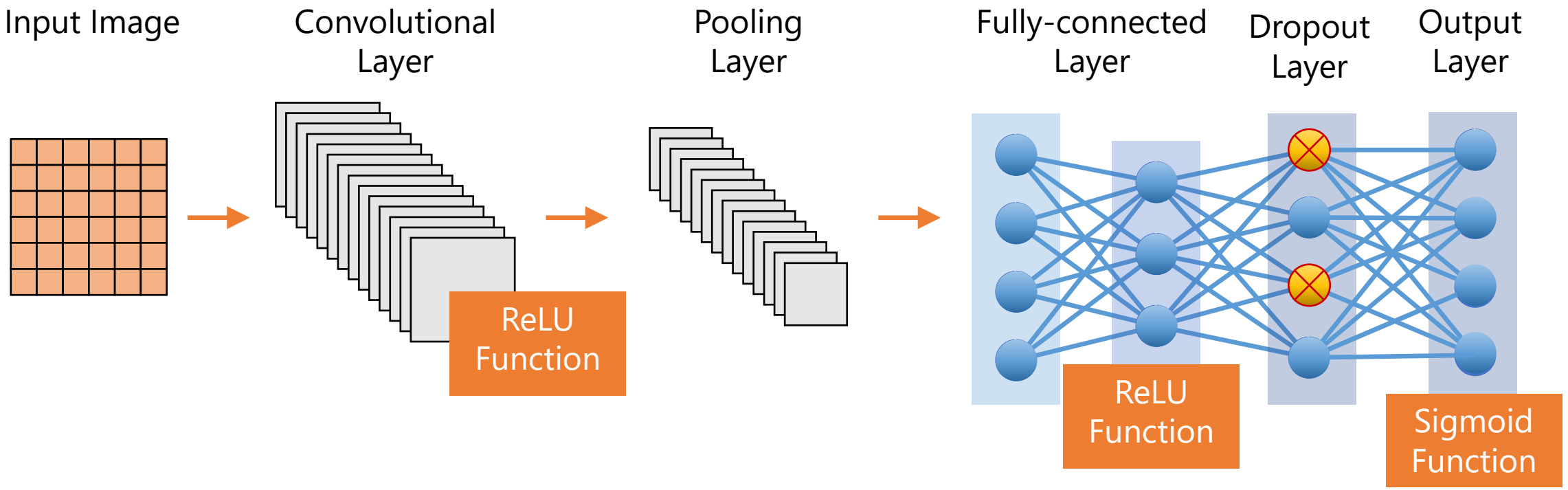
Let's Build CNN Model for Binary Image Classification



Binary Classification Problem

- Classify the input data into two classes.

CNN for Binary Classification



Import Libraries

```
# Import libraries
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
%matplotlib inline

from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Dropout, Flatten
```

Prepare Dataset

```
# Upload dataset
train_data_dir = "C:/Users/username/cats_or_dogs/train"
test_data_dir = "C:/Users/username/cats_or_dogs/test“

# Create data generator
datagen = ImageDataGenerator(rescale=1.0/255.0)

# Prepare iterators
train_it = datagen.flow_from_directory(train_data_dir,
                                       class_mode='binary', batch_size=32, target_size=(200, 200))
test_it = datagen.flow_from_directory(test_data_dir,
                                      class_mode='binary', batch_size=32, target_size=(200, 200))
```

```
Found 3000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

Prepare Dataset (Continued)

```
# Check the class indices
print(train_it.class_indices)
```

```
{'cats': 0, 'dogs': 1}
```

Modeling

```
# Define CNN model
model = Sequential()
model.add(Conv2D(32, input_shape=(200,200,3),
                 kernel_size=(3,3), padding='same',
                 activation='relu',
                 kernel_initializer='he_uniform'))
model.add(MaxPooling2D(2))
#model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
#model.add(Dropout(0.25))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
                      loss='binary_crossentropy',
                      metrics=['accuracy'])
```

Modelling (Continued)

 # Show the model summary
model.summary()

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 200, 200, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 100, 100, 32)      0
_________________________________________________________________
flatten_1 (Flatten)          (None, 320000)            0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               40960128
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 40,961,153
Trainable params: 40,961,153
Non-trainable params: 0
_________________________________________________________________
```

Model Training

```
# Fit CNN model
history = model.fit(train_it, steps_per_epoch=len(train_it),
                            validation_data=test_it,
                            validation_steps=len(test_it),
                            epochs=20, verbose=1)
```

```
----------------------------------------------------------------
Epoch 1/20
94/94 [==============================] - 71s 753ms/step - loss: 11.3531 - accuracy:
0.5420 - val_loss: 1.3144 - val_accuracy: 0.5400
Epoch 2/20
94/94 [==============================] - 69s 735ms/step - loss: 0.9587 - accuracy:
0.6250 - val_loss: 0.5173 - val_accuracy: 0.5410
Epoch 3/20
94/94 [==============================] - 70s 746ms/step - loss: 0.7616 - accuracy:
0.6977 - val_loss: 1.2337 - val_accuracy: 0.6630
 . . . . . .
```

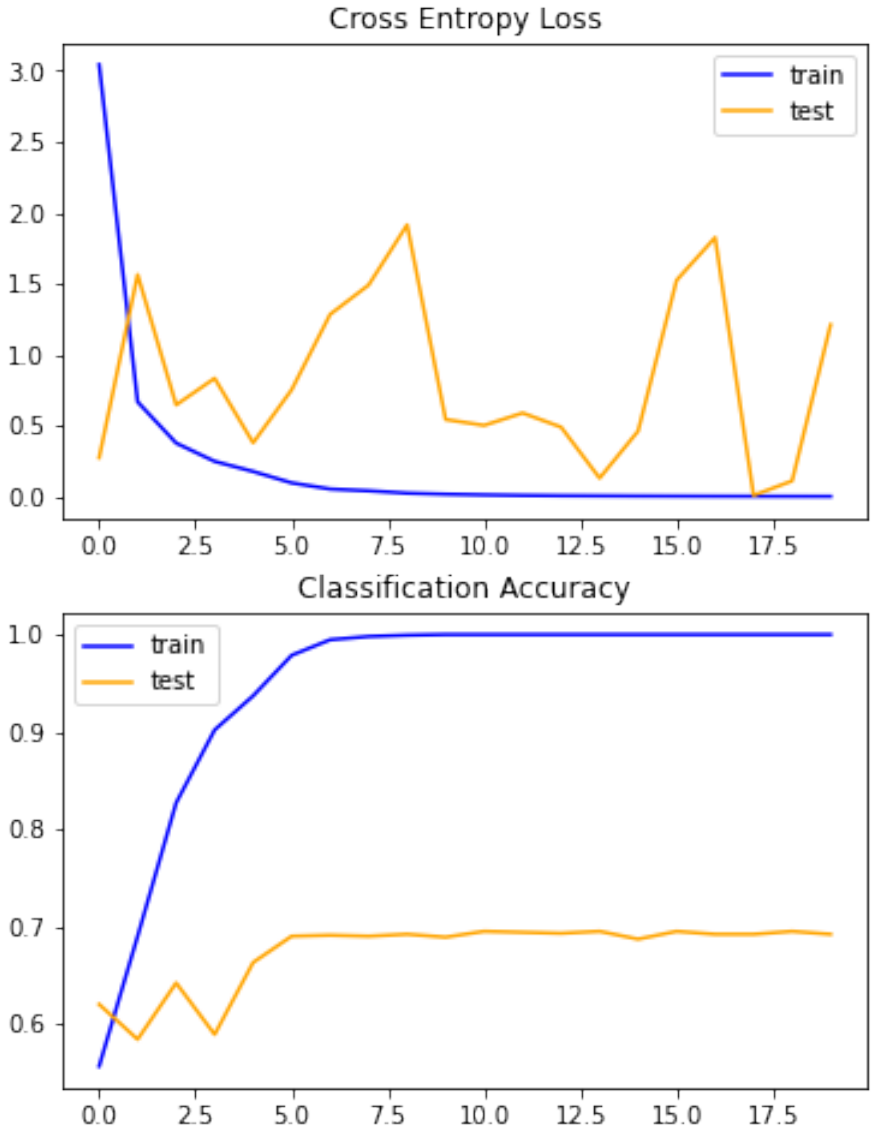
Model Evaluation

```
# Evaluate the model
_, acc = model.evaluate(test_it, steps=len(test_it), verbose=0)
print('%.3f' % acc)
```

```
0.679
```

Visualize Model Performance

```
# Plot the history of cross entropy loss
plt.figure(figsize=(6,8))
plt.subplot(211)
plt.title('Cross Entropy Loss')
plt.plot(history.history['loss'],
        color='blue', label='train')
plt.plot(history.history['val_loss'],
        color='orange', label='test')
plt.legend()

# Plot the history of accuracy
plt.subplot(212)
plt.title('Classification Accuracy')
plt.plot(history.history['accuracy'],
        color='blue', label='train')
plt.plot(history.history['val_accuracy'],
        color='orange', label='test')
plt.legend()
```

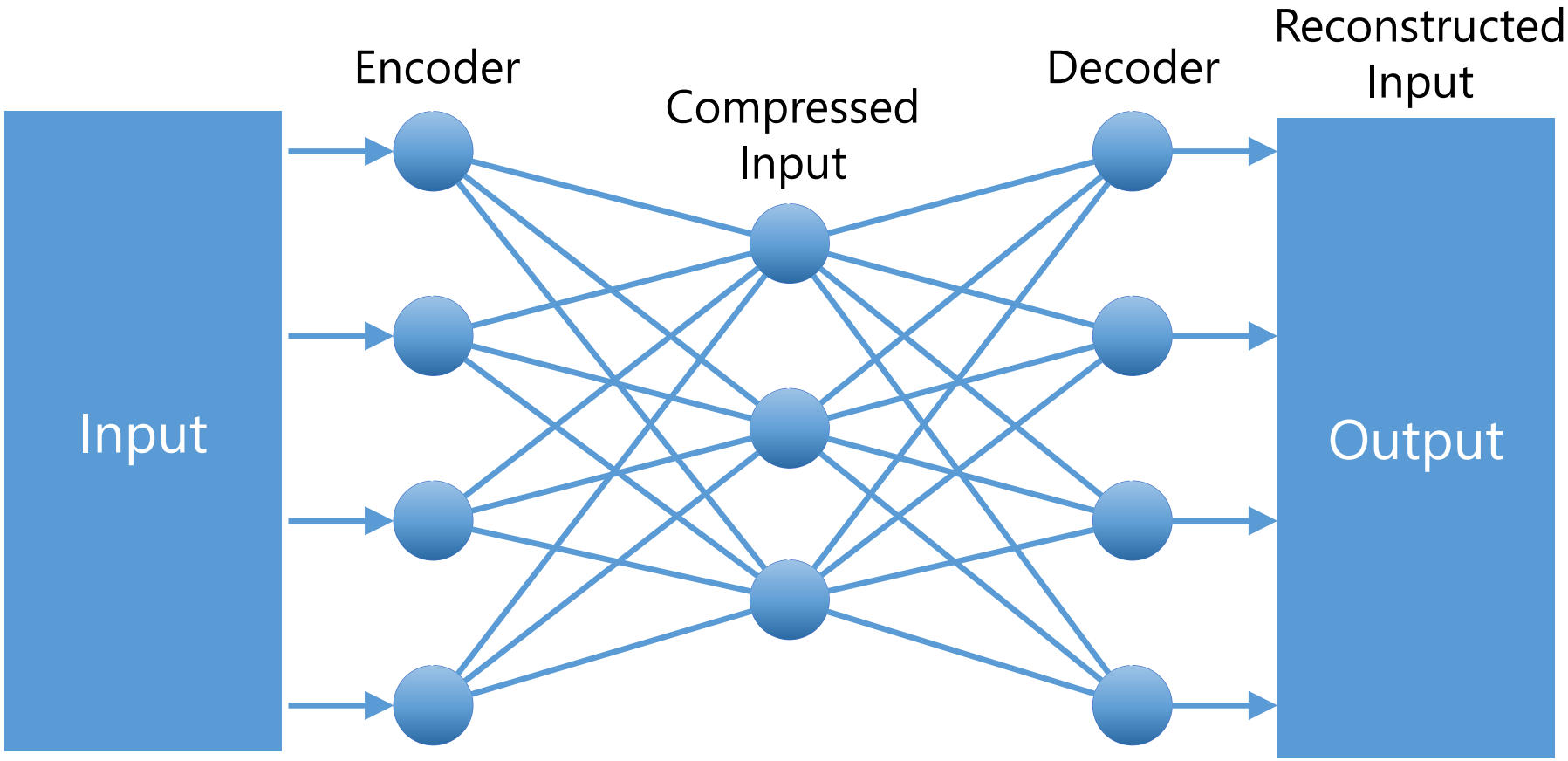


11. Autoencoder

What is Autoencoder?

- Its core is dimensionality reduction.
- Extracts some more important information from the data while discarding less important ones.
- Compress the input data and reconstruct it later.

Structure of Autoencoder



Benefits of Using Autoencoder

- DNN often shows lower performance than simpler models.
- It is due to:
  - Overfitting
  - Vanishing gradient problems
- Autoencoder can prevent these problems

Benefits of Using Autoencoder (Continued)

- Prevent overfitting

  In general, a too complex model falls into overfitting.

  Autoencoder reduces the model complexity by dimensionality reduction.

Benefits of Using Autoencoder (Continued)

- Prevent overfitting

  In general, a too complex model falls into overfitting.

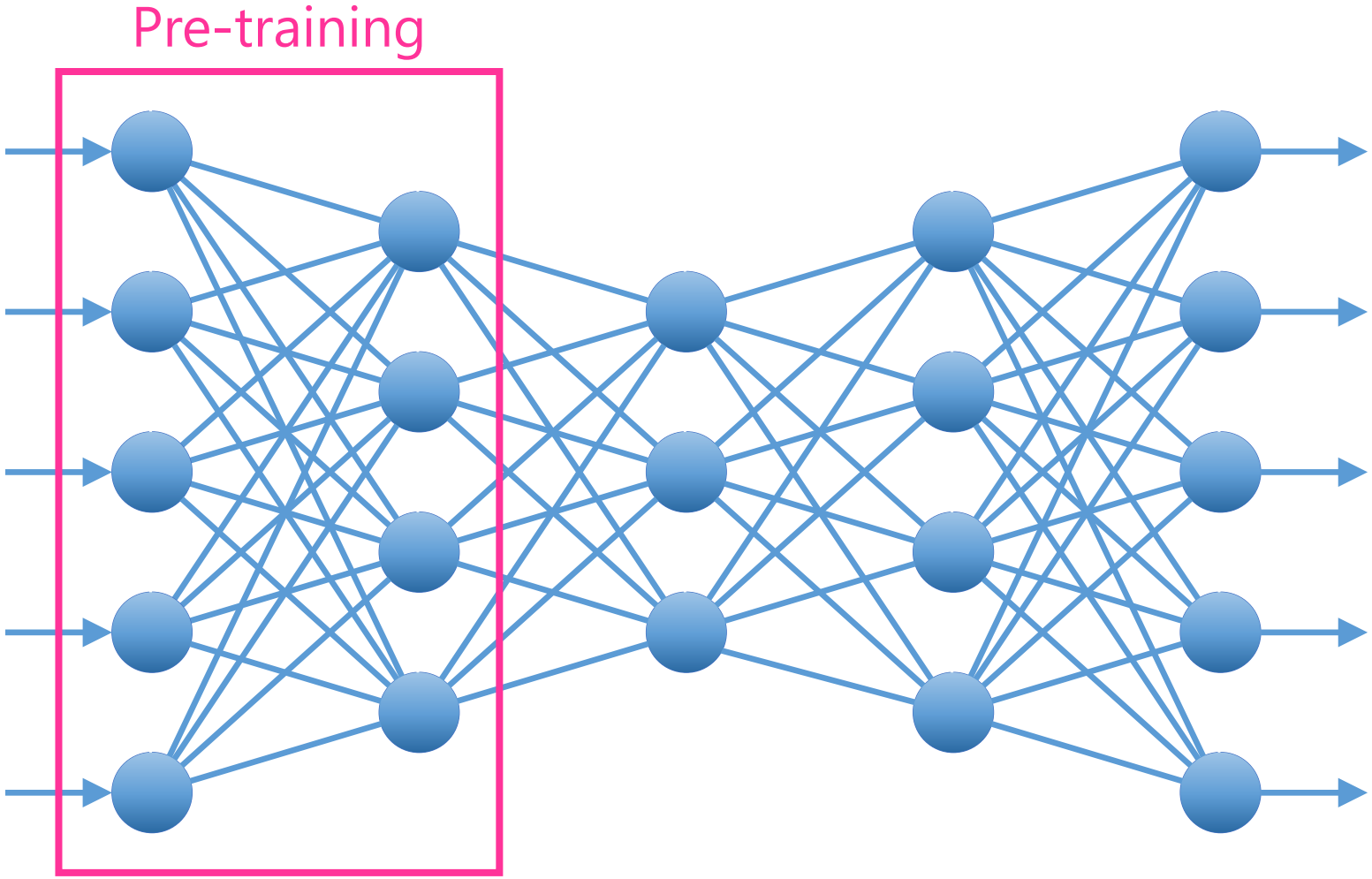
  Autoencoder reduces the model complexity by dimensionality reduction.

- Prevent vanishing gradient problem.

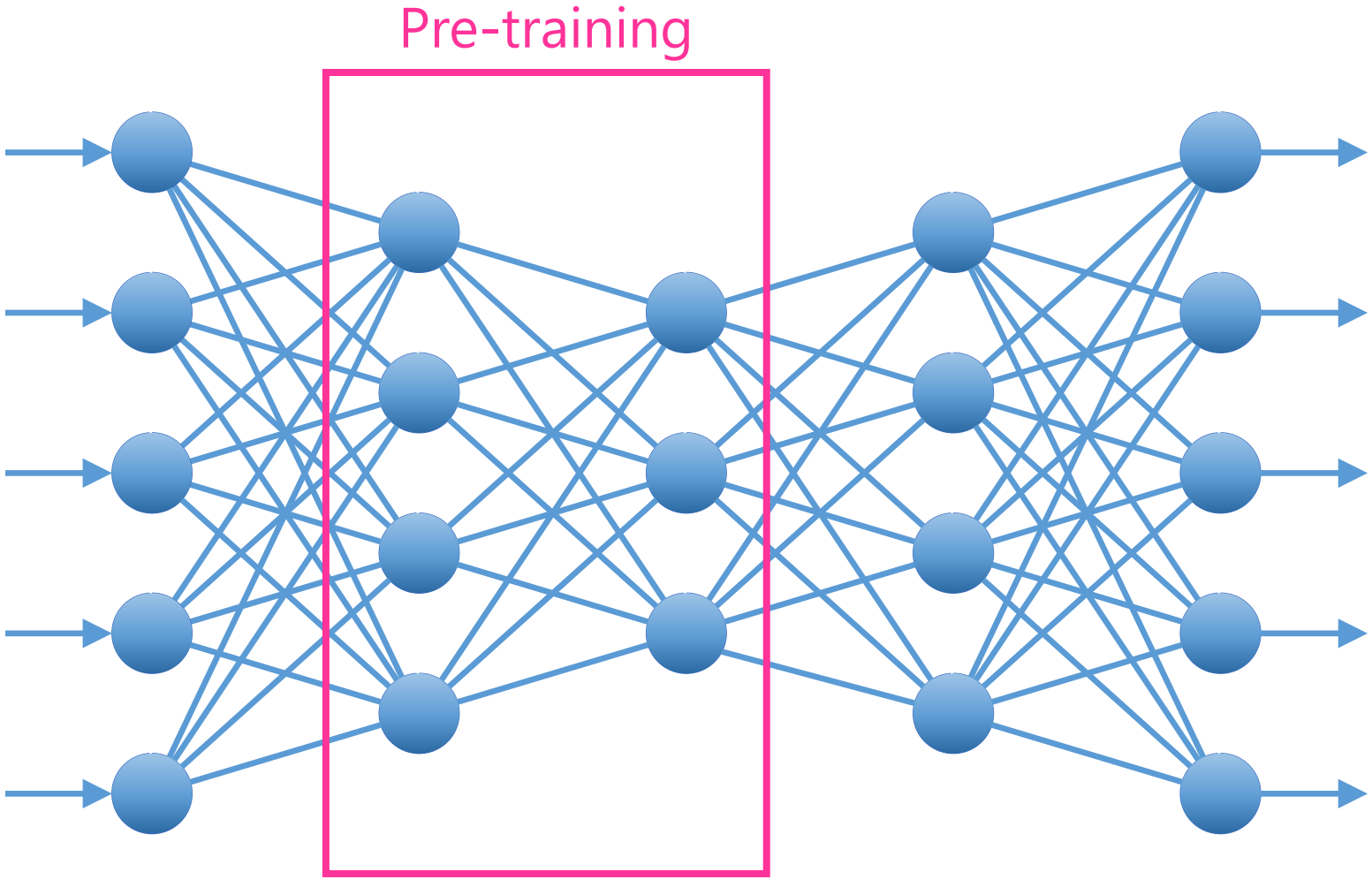
  Backpropagation can result in a vanishing gradient problem when the model has many hidden layers.

  Autoencoder addresses this problem by initializing parameters properly.

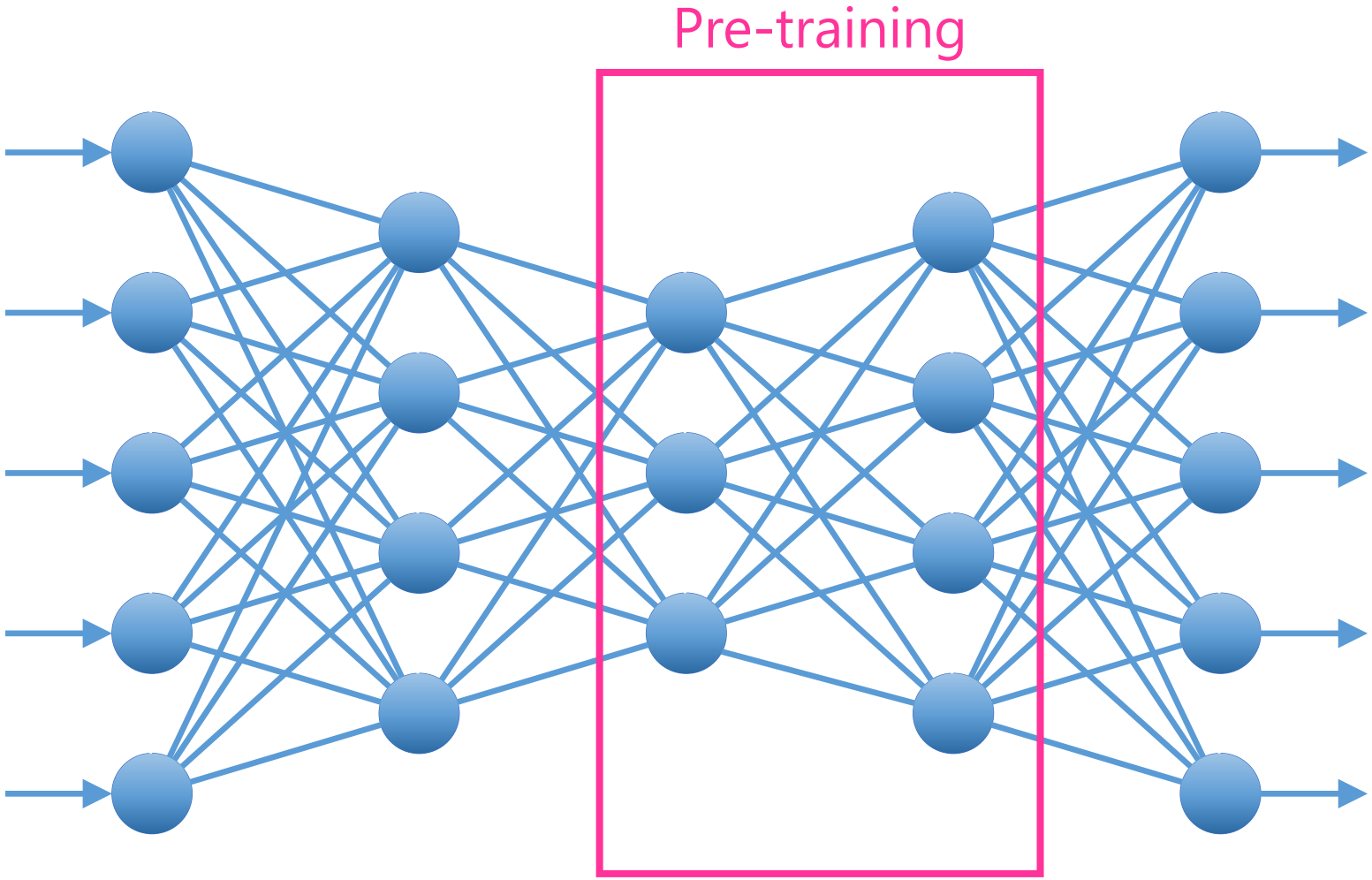
Stacked Autoencoder



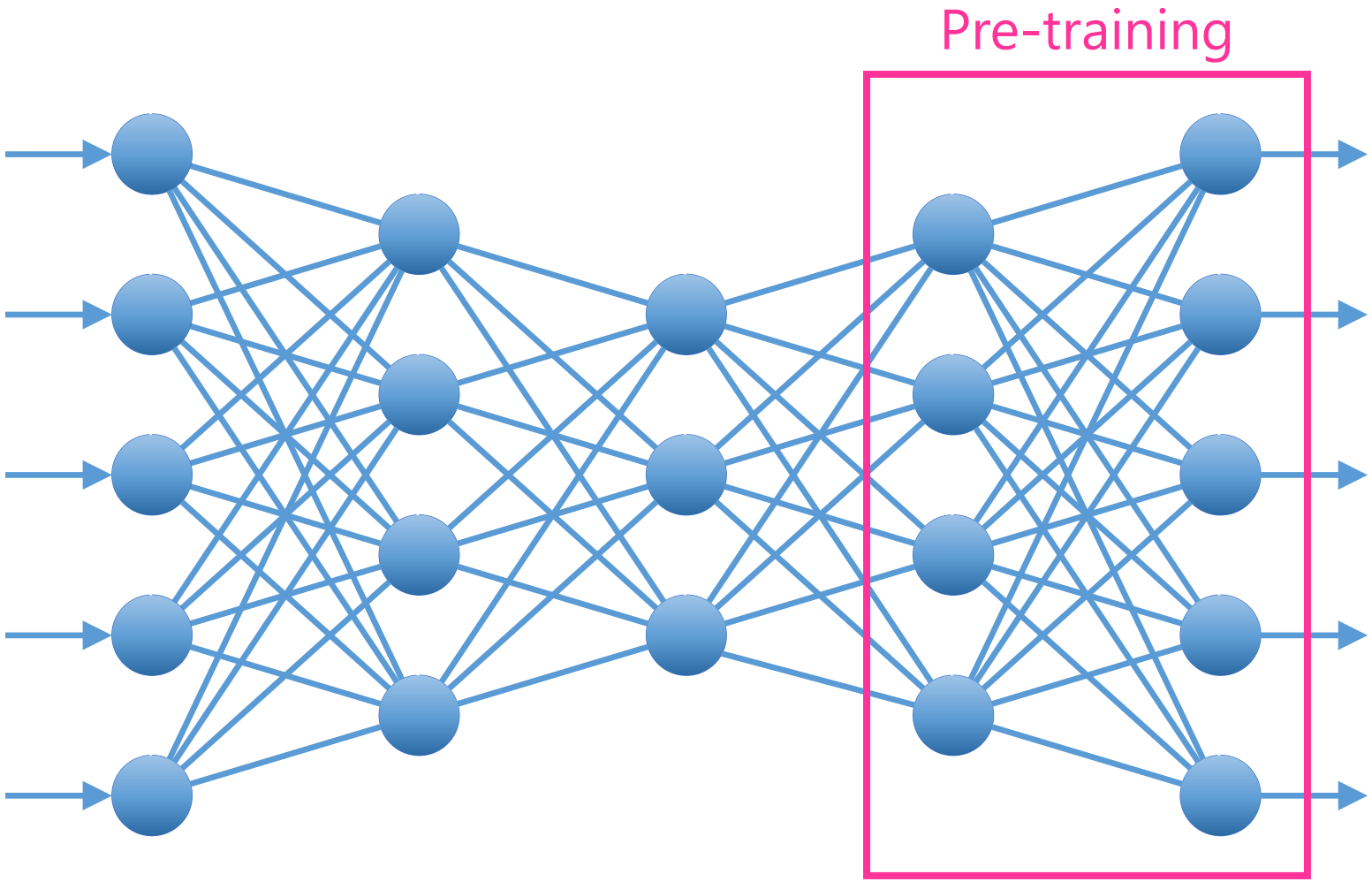
Stacked Autoencoder



Stacked Autoencoder

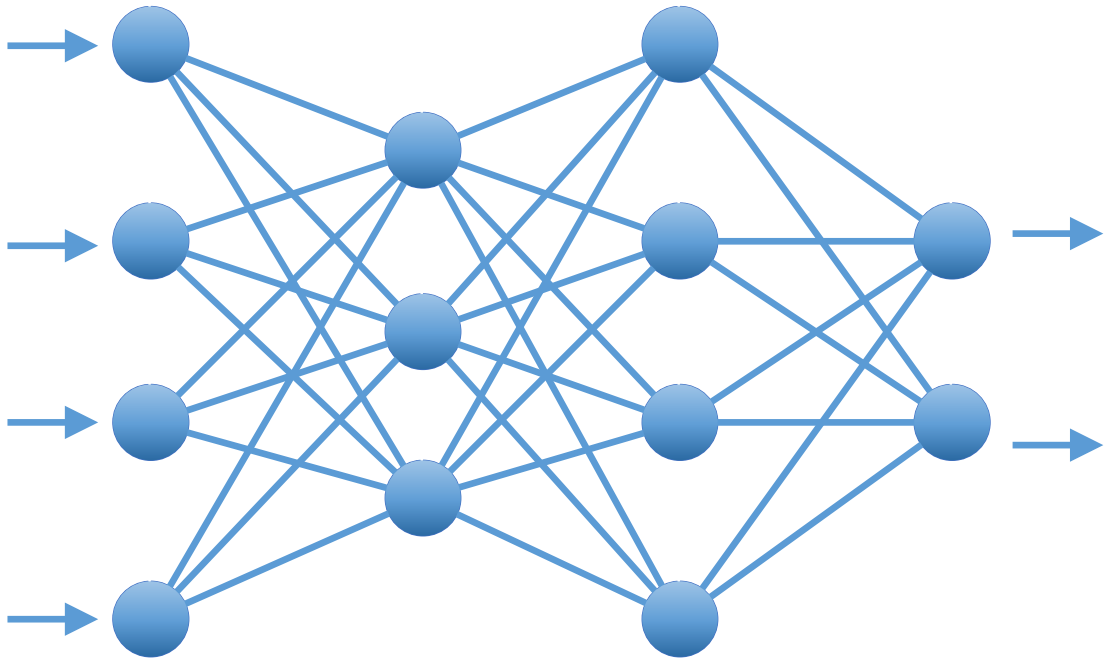


Stacked Autoencoder



Vanishing Gradient Problem

- Gradients often get smaller and smaller as the increase in the number of hidden layers.
- In DNNs with many hidden layers, the gradients can become almost 0, and thus, they can hardly learn from the data.
- It depends on initial parameter value.



$$f'(x) = \frac{e^{-x}}{(1+e^{-x})^2}$$

12. LeNet

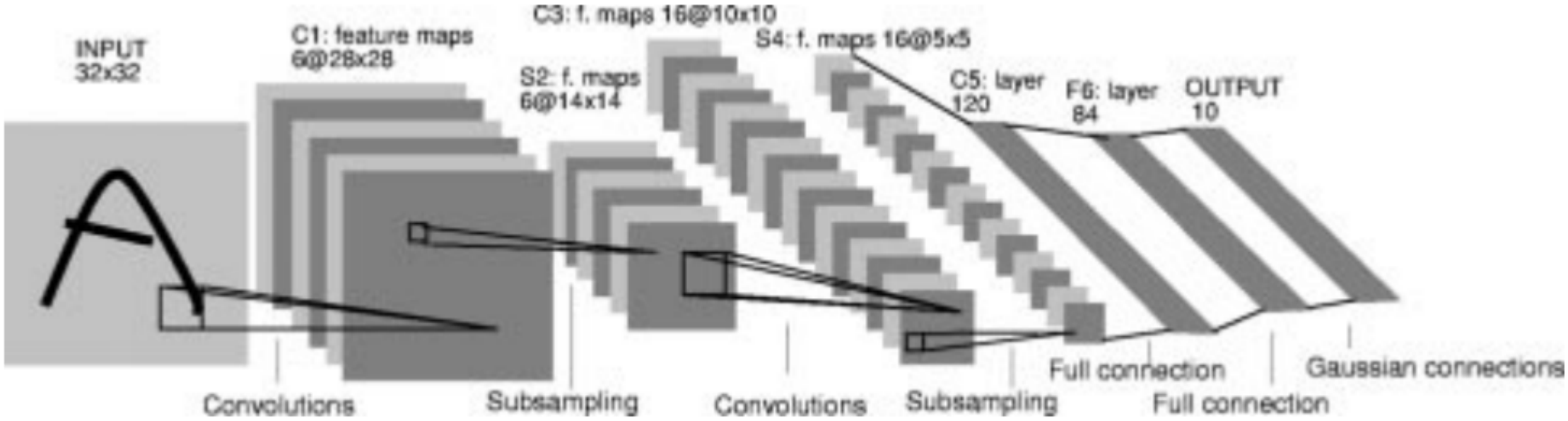
Study Successful Model Architectures

- LeNet-5
- AlexNet
- VGG
- GoogLeNet
- ResNet

LeNet-5

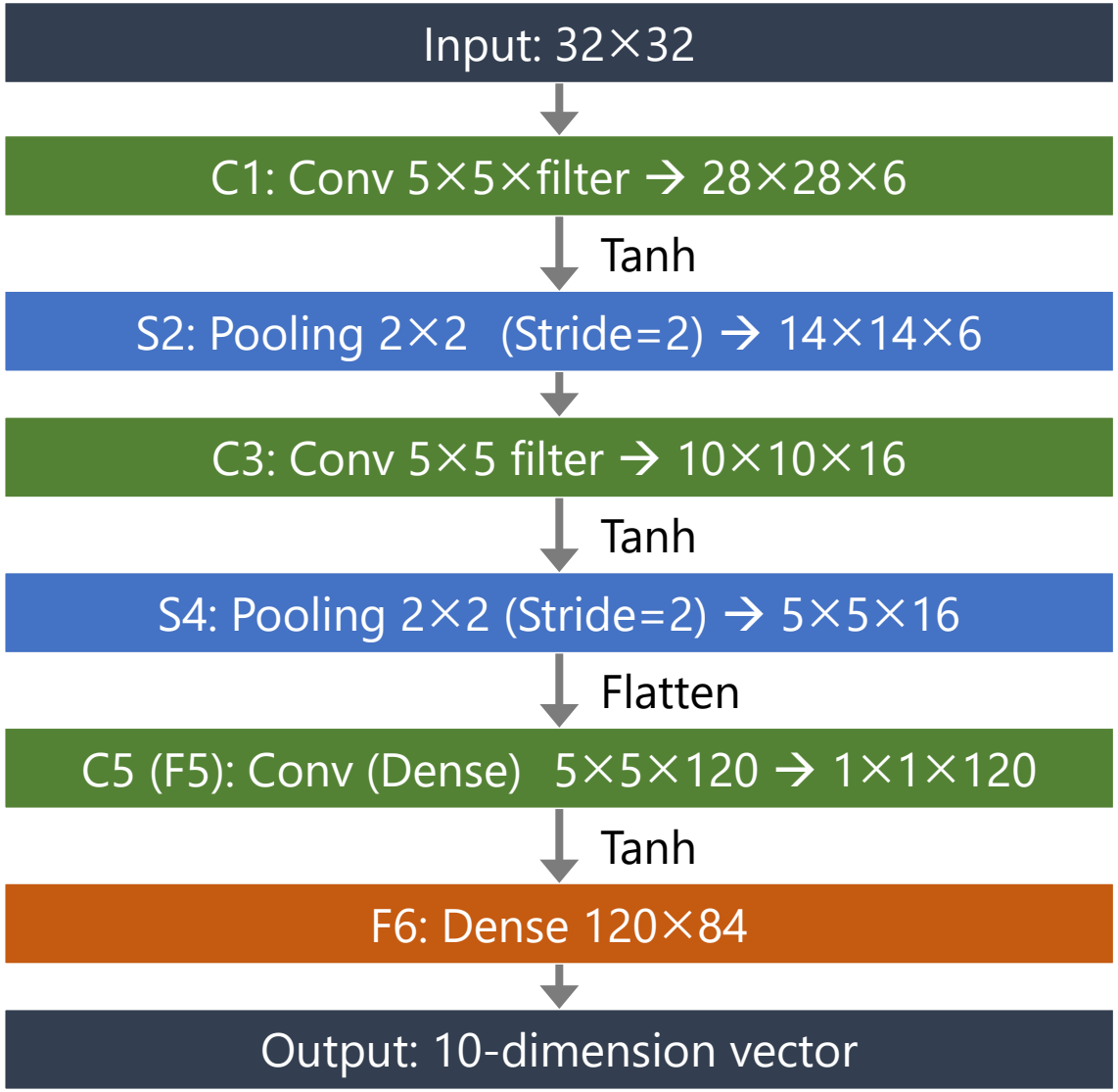
- LeNet-5 was proposed by Yann LeCun et al.
- It has two consecutive convolution-pooling combinations, followed by three fully-connected layers.
- LeCun et al. applied LeNet-5 to a handwritten character recognition problem.

LeNet-5 Architecture



LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

Summary of LeNet-5



Differences from Newer Popular Models

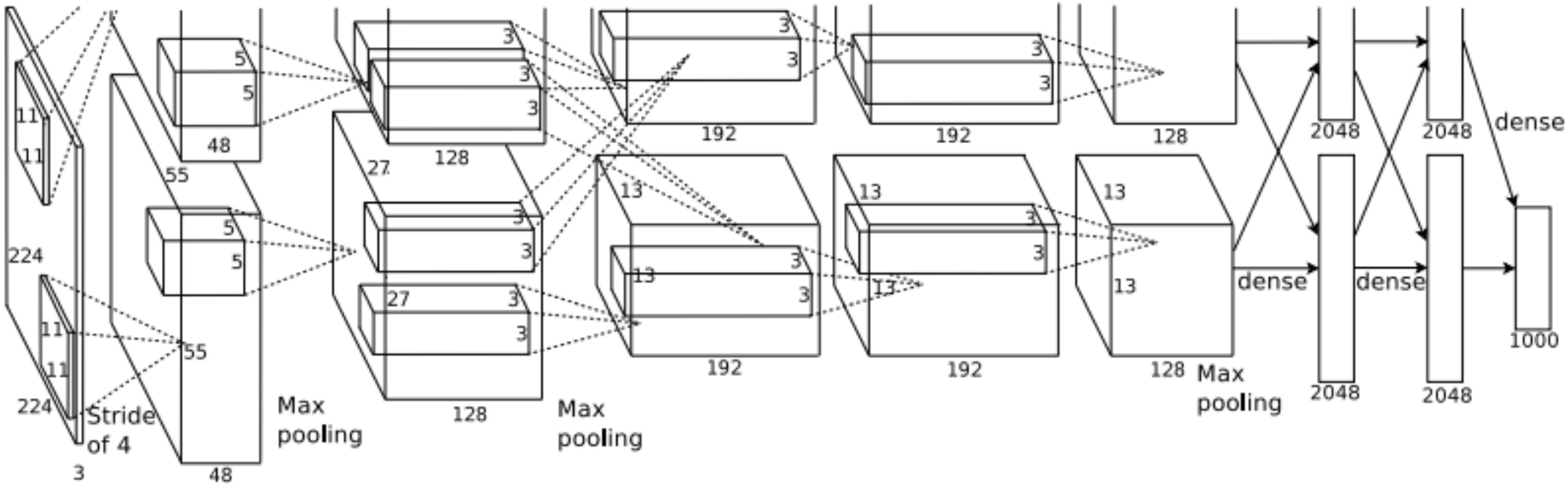
- Convolutional layers do not apply zero-padding.
- Mainly uses tanh function as the activation function.
- Pooling operations with no overlaps

13. AlexNet

AlexNet

- AlexNet-5 was proposed by Alex Krizhevsky et al.
- The first DL and CNN architecture applied to object recognition
- Achieved high performance using dropout and ReLU function.

AlexNet Architecture



Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25.

Summary of AlexNet

Summary of AlexNet

- Prevent overfitting by:
  - Dropout
  - Image data augmentation
  - ReLU function

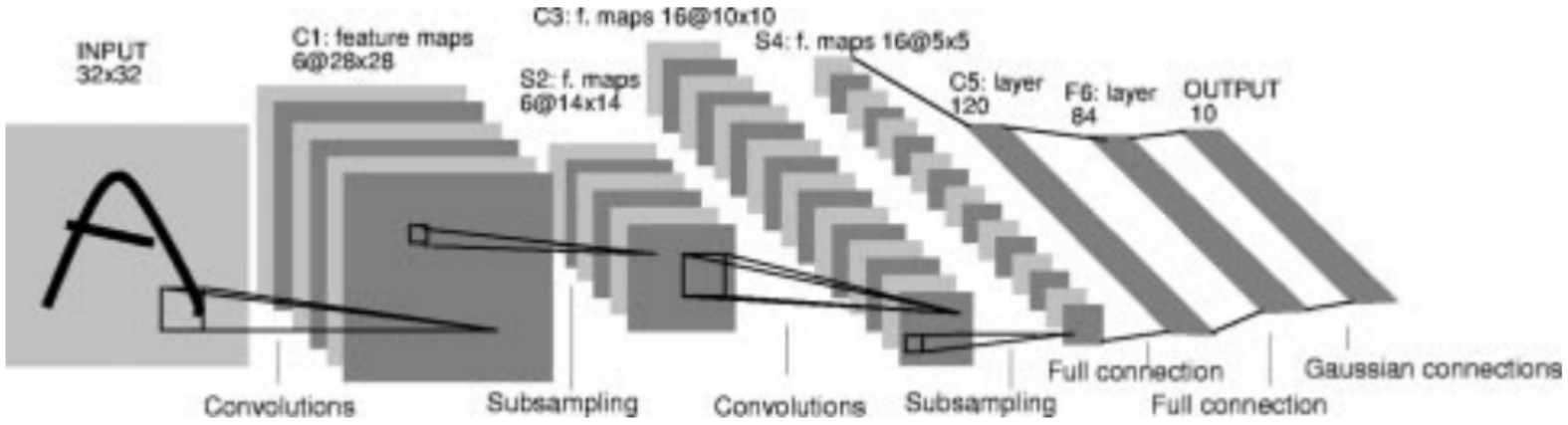
$$\text{cf: } \frac{d}{dx}\tanh x = \frac{1}{\cosh^2 x}$$

- Max-pooling to pick up the most salient feature

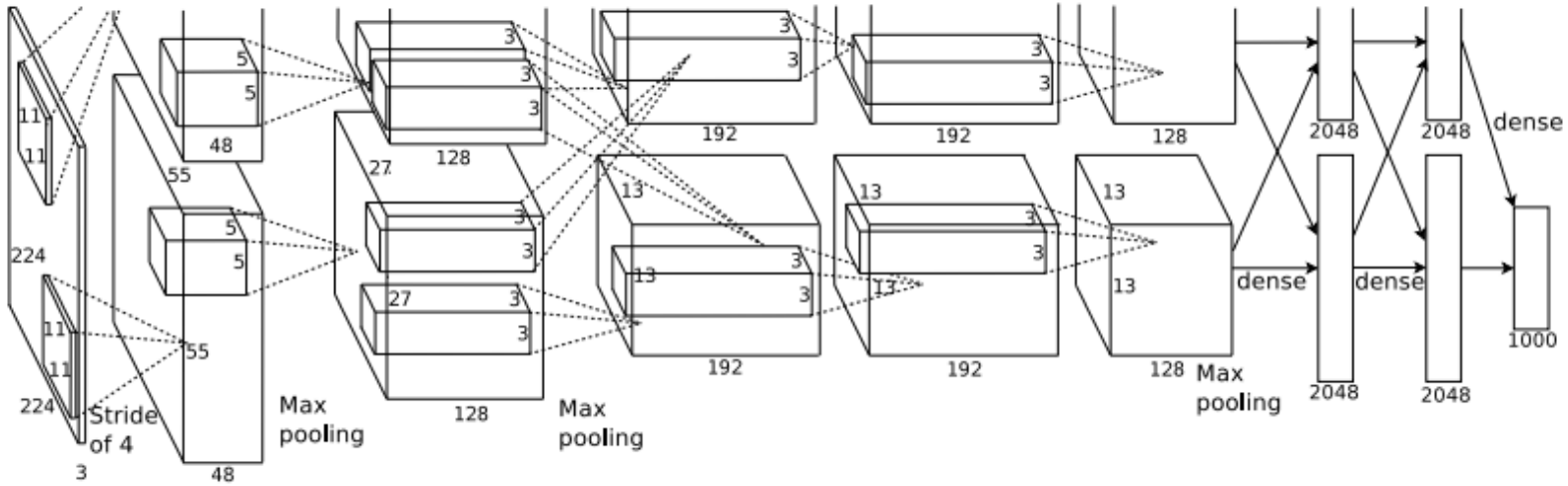
14. Multiclass Classification by LeNet & AlexNet

Multiclass Classification by LeNet & AlexNet

LeNet



AlexNet



Data Preparation

CIFAR-10

- Image data from Canadian Institute for Advanced Research
- Image dataset with 10 classes
- Can load from Keras

0：airplane
1：automobile
2：bird
3：cat
4：deer
5：dog
6：frog
7：horse
8：ship
9：truck

Import Libraries

```
# Import Libraries
import numpy as np
import pandas as pd

import keras
from keras import optimizers
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.layers import Conv2D, AveragePooling2D, MaxPooling2D
from keras.layers import BatchNormalization, Dropout
from keras.optimizers import Adadelta

from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Load Dataset

```
# Import Keras library for CIFAR-10 dataset
from keras.datasets import cifar10

# Download CIFAR dataset
(X_train, y_train), (X_test, y_test)=cifar10.load_data()
```

Load Dataset

```
# Import Keras library for CIFAR-10 dataset
from keras.datasets import cifar10

# Download CIFAR dataset
(X_train, y_train), (X_test, y_test)=cifar10.load_data()

# Show the shape of the dataset
print('X_train: ', X_train.shape)
print('X_test : ', X_test.shape)
print('y_train: ', y_train.shape)
print('y_test : ', y_test.shape)
```

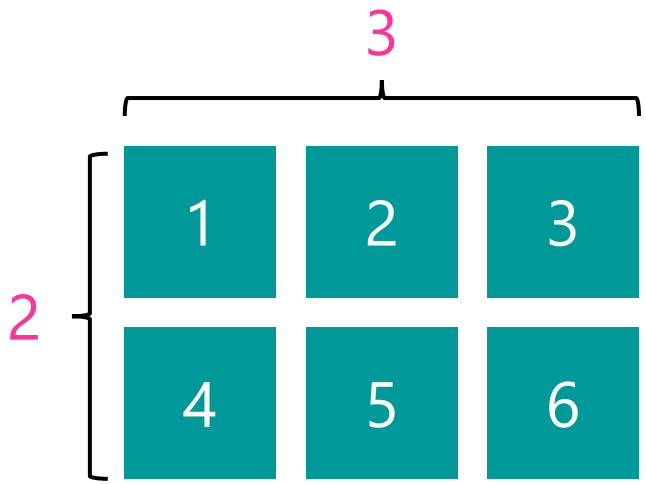
```
X_train:  (50000, 32, 32, 3)
X_test :  (10000, 32, 32, 3)
y_train:  (50000, 1)
y_test :  (10000, 1)
```

Show Some Images

```
# Show the first 6 images
fig = plt.figure()
for i in range(6):
    plt.subplot(2, 3, i+1)
    plt.imshow(X_train[i])
```

Show Some Images

```
# Show the first 6 images
fig = plt.figure()
for i in range(6):
    plt.subplot(2, 3, i+1)
    plt.imshow(X_train[i])
```



Show Data Range

```
# Show the range of image data
print('X_train range: ', np.ptp(X_train))
print('X_test range:  ', np.ptp(X_test))
----------------------------------------
X_train range:  255
X_test range:   255
```

Normalize Data

```
# Normalize image data

# Convert data to float
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# Divide image data by 255
X_train /= 255
X_test /= 255

# Check if the image datasets are properly normalized
print('Normalized X_train range: ', np.ptp(X_train))
print('Normalized X_test range:  ', np.ptp(X_test))
```

```
Normalized X_train range:  1.0
Normalized X_test range:   1.0
```

Check the Number of Classes

```
# Check the number of classes
print('N of classes: y_train = ', np.unique(y_train).size)
print('N of classes: y_test  = ', np.unique(y_test).size)
```

```
N of classes: y_train =  10
N of classes: y_test  =  10
```

One-Hot Encoding

```
# Apply one-hot encoding to label data

# The number of classes
num_classes = 10

# One-hot encoding
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# Check if the labels are one-hot encoded properly
print('y_train shape: ', y_train.shape)
print('y_test shape : ', y_test.shape)
```

```
y_train shape:  (50000, 10)
y_test shape :  (10000, 10)
```

Set Input Shape

```
# Set input shape: height, width, colors
img_rows, img_cols, channels = 32, 32, 3
input_shape = (img_rows, img_cols, channels)
```

LeNet

Create LeNet-5 Model

```
# Create LeNet-5 model
lenet_model = Sequential()
lenet_model.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1),
                       padding='same', activation='tanh', input_shape=input_shape))
lenet_model.add(AveragePooling2D((2, 2), strides=(2, 2)))
lenet_model.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1),
                       padding='valid', activation='tanh'))
lenet_model.add(AveragePooling2D((2, 2), strides=(2, 2)))
lenet_model.add(Flatten())
lenet_model.add(Dense(120, activation='tanh'))
lenet_model.add(Dense(84, activation='tanh'))
lenet_model.add(Dense(num_classes, activation='softmax'))

lenet_model.compile(
                        loss=keras.losses.categorical_crossentropy,
                        optimizer=keras.optimizers.Adadelta(),
                        metrics=['accuracy’]
                        )
```

Model Summary

print(lenet_model.summary())

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 6)         456
_________________________________________________________________
average_pooling2d_1 (Average (None, 16, 16, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 16)        2416
_________________________________________________________________
average_pooling2d_2 (Average (None, 6, 6, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               69240
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_3 (Dense)              (None, 10)                850
=================================================================
Total params: 83,126
Trainable params: 83,126
Non-trainable params: 0
_________________________________________________________________
None
```

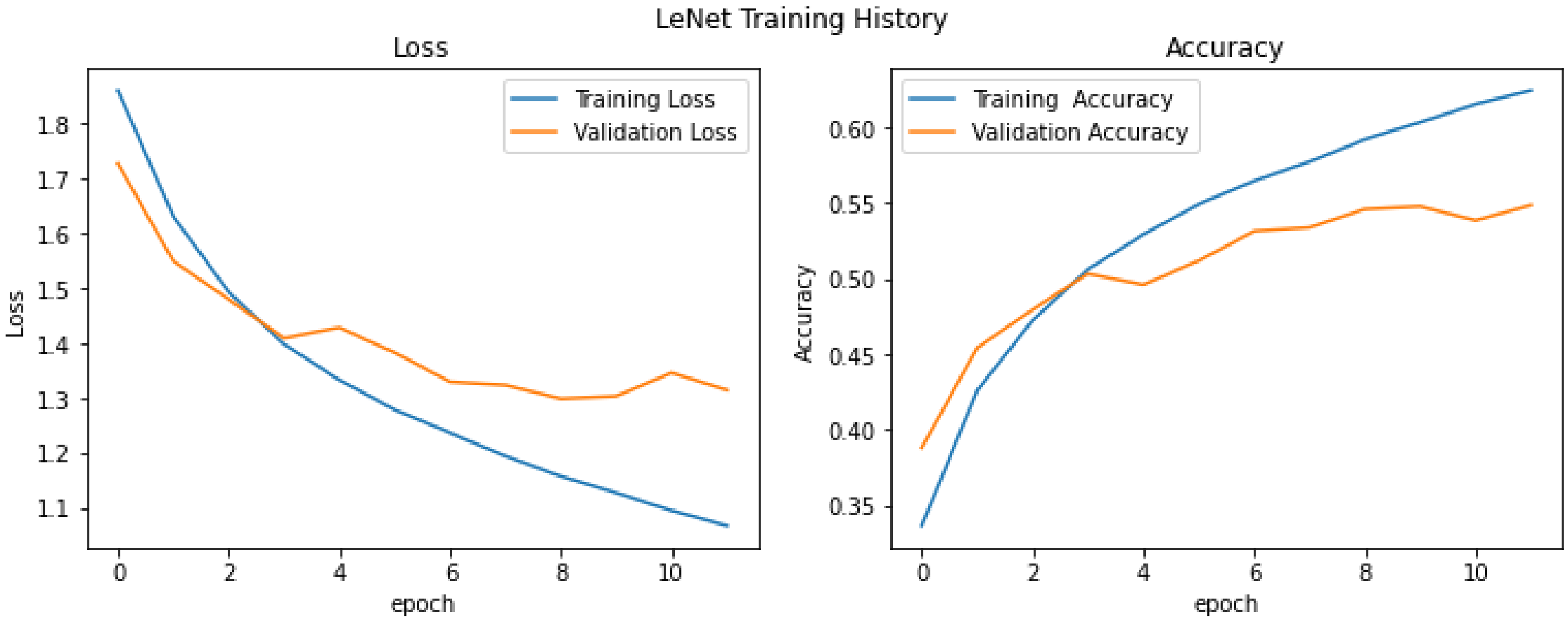
Model Training

```
# Fit LeNet-5 model and store the training history
l_history = lenet_model.fit(X_train, y_train,
                            batch_size=128,
                            epochs=12,
                            verbose=1,
                            validation_data=(X_test, y_test))
```

Visualize the Training History

```
# Visualize training history
plt.figure(figsize=(12,4))
plt.suptitle('LeNet Training History')
# Loss
plt.subplot(1, 2, 1)
plt.title('Loss')
plt.plot(l_history.history['loss'], label='Training Loss')
plt.plot(l_history.history['val_loss'], label='Validation Loss')
plt.legend(loc='best')
plt.xlabel('epoch')
plt.ylabel('Loss')
# Accuracy
plt.subplot(1, 2, 2)
plt.title('Accuracy')
plt.plot(l_history.history['accuracy'], label='Training  Accuracy')
plt.plot(l_history.history['val_accuracy'], label='Validation Accuracy')
plt.legend(loc='best')
plt.xlabel('epoch')
plt.ylabel('Accuracy')
```

Visualize the Training History



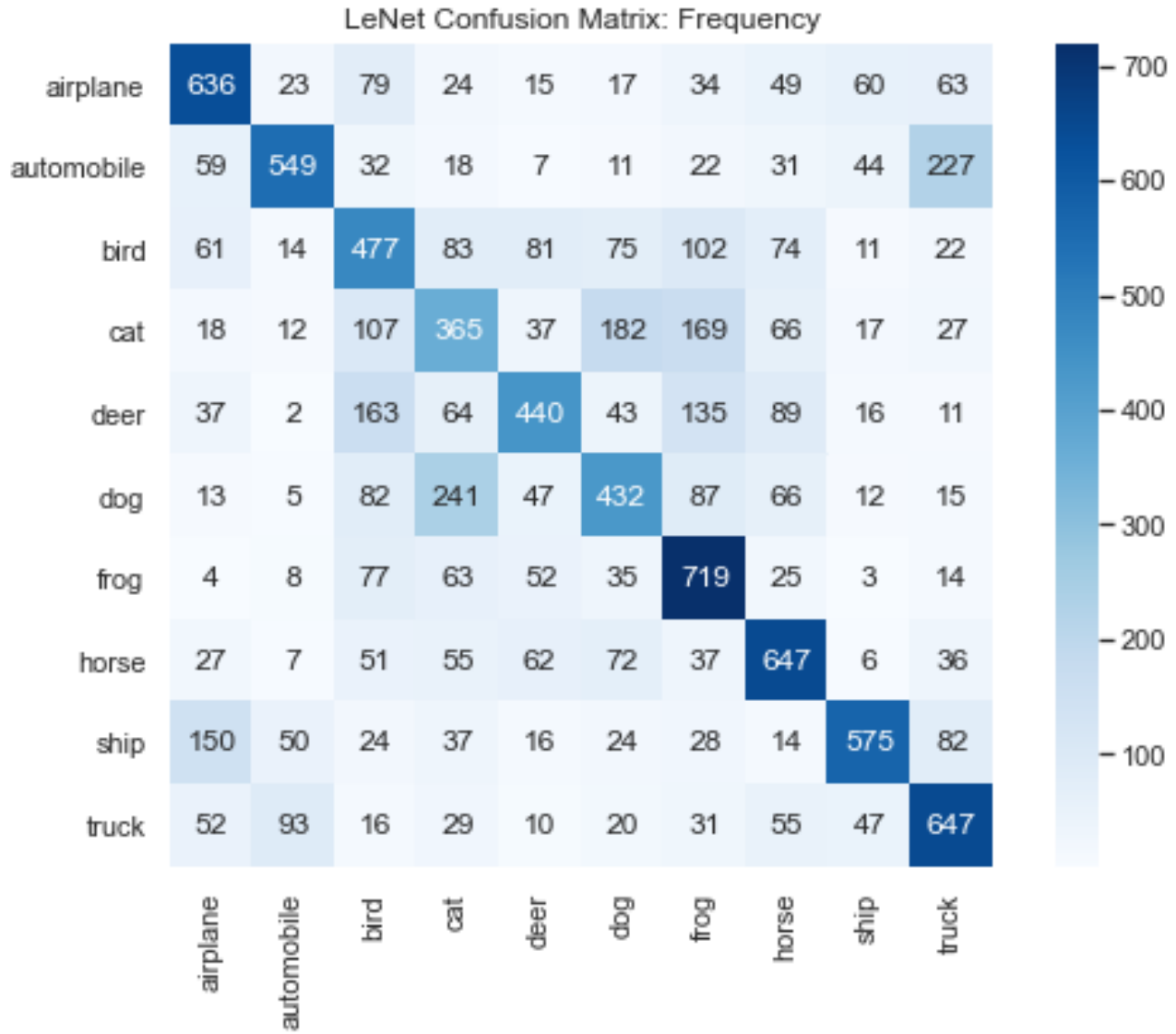
Prediction

```
# Making prediction
lenet_y_pred = lenet_model.predict_classes(X_test)
y_true=np.argmax(y_test, axis=1)
```

Confusion Matrix

```
# Create Confusion matrix with frequency
cm = confusion_matrix(y_true, lenet_y_pred)

labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

# Set class names to be displayed
cm = pd.DataFrame(cm,
                  index=labels,
                  columns=labels)

sns.set(rc={'figure.figsize':(9, 6)})
sns.heatmap(cm,
            square=True, cbar=True, annot=True,
            cmap='Blues',
            fmt='.0f').set(title='LeNet Confusion Matrix: Frequency')
```

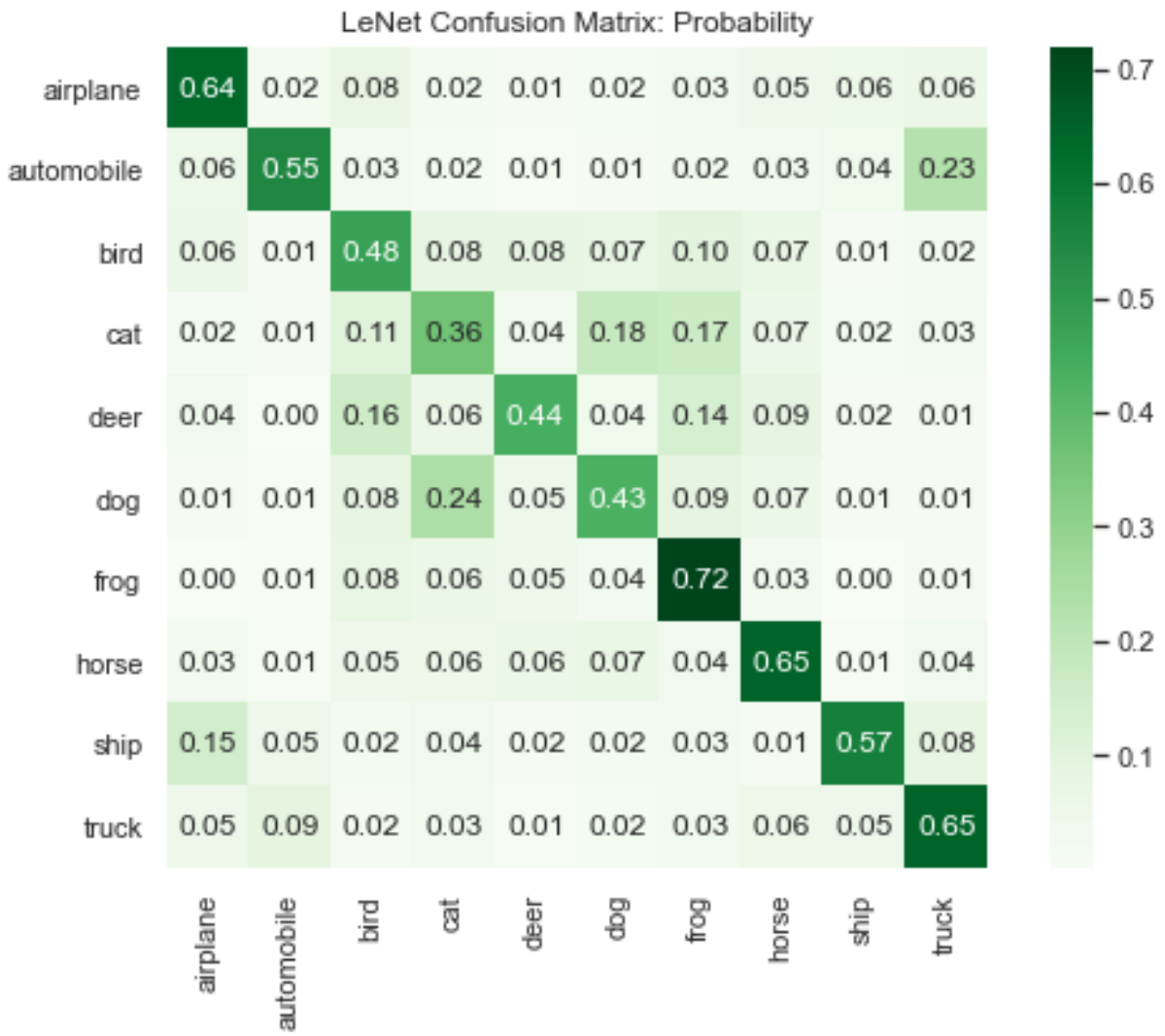
Confusion Matrix



Confusion Matrix (Probability)

```
# Create Confusion matrix with probability

# Compute the row probability
cmp = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

cmp = pd.DataFrame(cmp,
                   index=labels,
                   columns=labels)

sns.set(rc={'figure.figsize':(9, 6)})
sns.heatmap(cmp,
            square=True, cbar=True, annot=True,
            cmap='Greens',
            fmt='.2f').set(title='LeNet Confusion Matrix: Probability')
```

Confusion Matrix (Probability)



Show Model Performance

```
# Show the test loss and accuracy
score = lenet_model.evaluate(X_test, y_test, verbose=0)
print('LeNet: Loss and Accuracy')
print(f'Test loss score    : {score[0]:.3f}')
print(f'Test accuracy score: {score[1]:.3f}’)
```

```
LeNet: Loss and Accuracy
Test loss score    : 1.314
Test accuracy score: 0.549
```

AlexNet

Create AlexNet Model

```
# Create Alexnet model
alex_model = Sequential()
alex_model.add(Conv2D(96, kernel_size=(11, 11),
                      strides=(4, 4), padding='same',
                      activation='relu',
                      input_shape=input_shape))
alex_model.add(MaxPooling2D(pool_size=(3, 3),
                      strides=(2,2), padding='same'))
alex_model.add(BatchNormalization())
alex_model.add(Conv2D(256, kernel_size=(5, 5),
                      strides=(1, 1), padding='same',
                      activation='relu'))
alex_model.add(MaxPooling2D(pool_size=(3, 3),
                      strides=(2,2), padding='same'))
alex_model.add(BatchNormalization())
alex_model.add(Conv2D(384, kernel_size=(3, 3),
                      strides=(1, 1), padding='same',
                      activation='relu’))
alex_model.add(Conv2D(384, kernel_size=(3, 3),
                      strides=(1, 1), padding='same',
                      activation='relu'))
alex_model.add(Conv2D(256, kernel_size=(3, 3),
                      strides=(1, 1), padding='same',
                      activation='relu'))
alex_model.add(MaxPooling2D(pool_size=(3, 3),
                      strides=(2,2), padding='same'))
alex_model.add(BatchNormalization())

alex_model.add(Flatten())
alex_model.add(Dense(2048, activation='relu'))
alex_model.add(Dropout(0.5))
alex_model.add(Dense(2048, activation='relu'))
alex_model.add(Dropout(0.5))
alex_model.add(Dense(num_classes,
                     activation='softmax'))

alex_model.compile(
    loss=keras.losses.categorical_crossentropy,
    optimizer=keras.optimizers.Adadelta(),
    metrics=['accuracy']
)
```

Model Summary

alex_model.summary()

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 8, 8, 96)          34944
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 96)          0
_________________________________________________________________
batch_normalization_1 (Batch (None, 4, 4, 96)          384
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 4, 256)         614656
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
batch_normalization_2 (Batch (None, 2, 2, 256)         1024
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 2, 2, 384)         885120
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 2, 2, 384)         1327488
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 2, 2, 256)         884992
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 1, 1, 256)         0
_________________________________________________________________
batch_normalization_3 (Batch (None, 1, 1, 256)         1024
_________________________________________________________________
flatten_2 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 2048)              526336
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_5 (Dense)              (None, 2048)              4196352
_________________________________________________________________
dropout_2 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                20490
=================================================================
Total params: 8,492,810
Trainable params: 8,491,594
Non-trainable params: 1,216
_________________________________________________________________
```

Model Training

```
# Fit Alexnet model and store the training progress
a_history = alex_model.fit(X_train, y_train,
                           batch_size=128,
                           epochs=12,
                           verbose=1,
                           validation_data=(X_test, y_test))
```

Visualize the Training History

```
# Visualize the results
plt.figure(figsize=(12,4))
plt.suptitle('AlexNet Training History')

plt.subplot(1, 2, 1)
plt.title('Loss')
plt.plot(a_history.history['loss'], label='Training Loss')
plt.plot(a_history.history['val_loss'], label='Validation Loss')
plt.legend(loc='best')
plt.xlabel('epoch')
plt.ylabel('Loss')

plt.subplot(1, 2, 2)
plt.title('Accuracy')
plt.plot(a_history.history['accuracy'], label='Training  Accuracy')
plt.plot(a_history.history['val_accuracy'], label='Validation Accuracy')
plt.legend(loc='best')
plt.xlabel('epoch')
plt.ylabel('Accuracy')
```

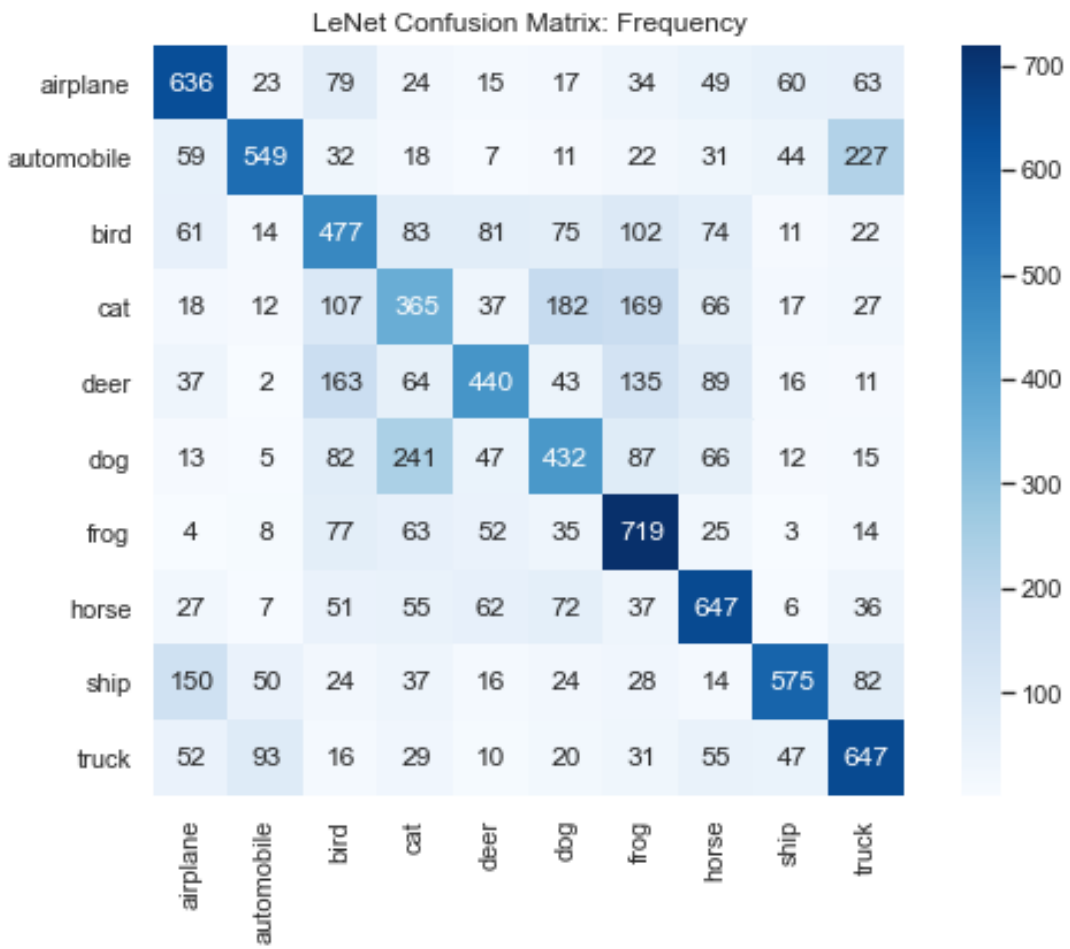
Visualize the Training History

Prediction & Confusion Matrix

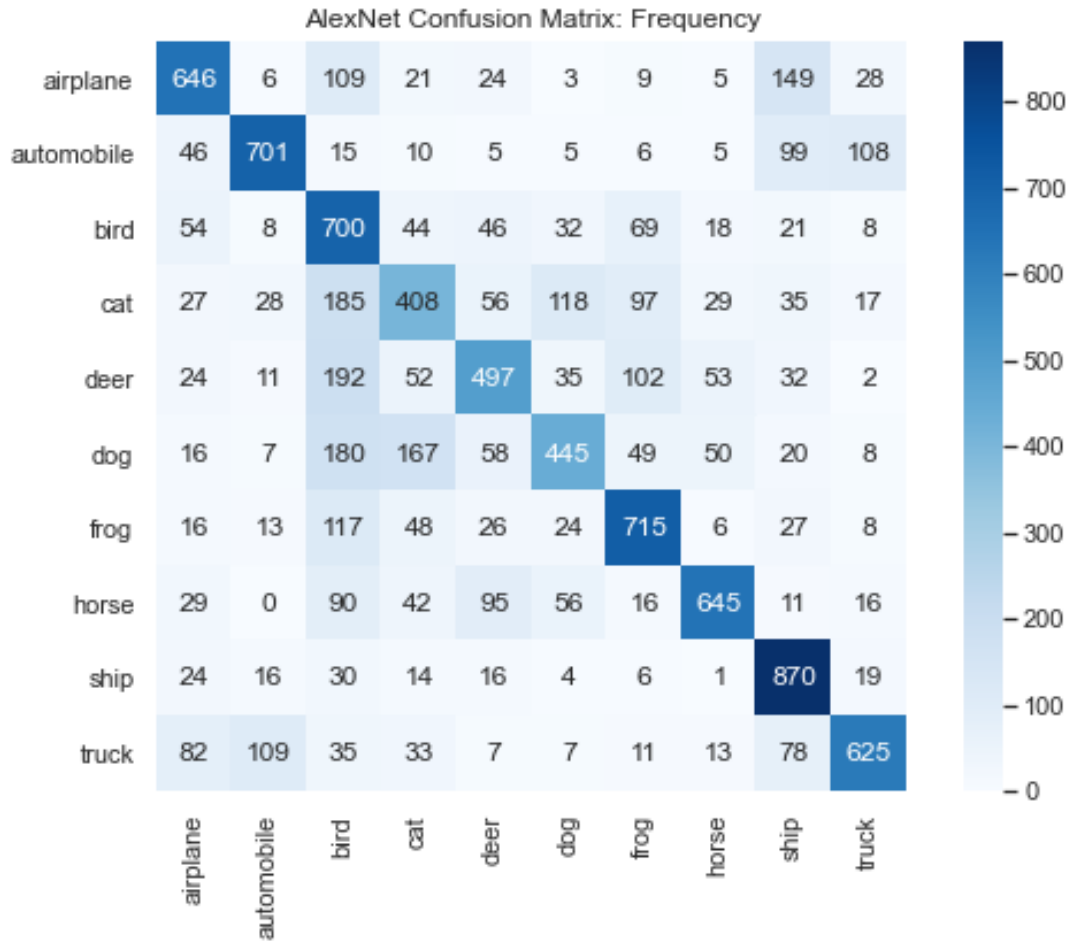
```
# Making prediction
alex_y_pred = alex_model.predict_classes(X_test)
y_true = np.argmax(y_test, axis=1)

# Create Confusion matrix with frequency
cm = confusion_matrix(y_true, alex_y_pred)

# Set class names to be displayed
cm = pd.DataFrame(cm,
                  index=labels,
                  columns=labels)

sns.set(rc={'figure.figsize':(9, 6)})
sns.heatmap(cm,
            square=True, cbar=True, annot=True,
            cmap='Blues',
            fmt='.0f').set(title='AlexNet Confusion Matrix: Frequency')
```

Confusion Matrix

LeNet



AlexNet



Show the Model Performance

```
# Show the test loss and accuracy
score = alex_model.evaluate(X_test, y_test, verbose=0)
print('AlexNet: Loss and Accuracy')
print(f'Test loss score    : {score[0]:.3f}')
print(f'Test accuracy score: {score[1]:.3f}')
```

AlexNet: Loss and Accuracy
Test loss score   : 1.277
Test accuracy score: 0.638

LeNet: Loss and Accuracy
Test loss score   : 1.314
Test accuracy score: 0.549

15. VGGNet

VGGNet

- VGGNet-5 was proposed by Karen Simonyan & Andrew Zisserman.

  "VGG": Visual Geometry Group at Oxford

- Deeper structure with many small filters

- VGG16 & VGG19

VGG Architecture

- 3×3 filters

  stride=1, padding=1

- ReLU function for each hidden layer

ConvNet Configuration				VGG16	VGG19
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

We Can Use Pre-trained VGGNet

- We can obtain pre-trained VGGNet and use it in our problems.
- We can make our work more efficient by using pre-trained model.
- Transfer learning

16. GoogLeNet

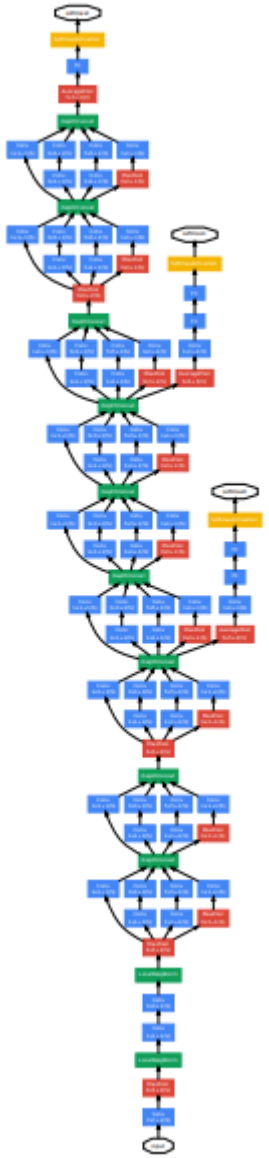
GoogLeNet

- DNN problems:
  - Overfitting
  - High memory use and computational cost
  - Vanishing gradient problem
- GoogLeNet is a DNN model designed to avoid the problems above.

Notable Characteristics of GoogleNet

- Pre-training
- Inception module
- Global Average Pooling (GAP)

GoogLeNet Structure



Convolutional layer

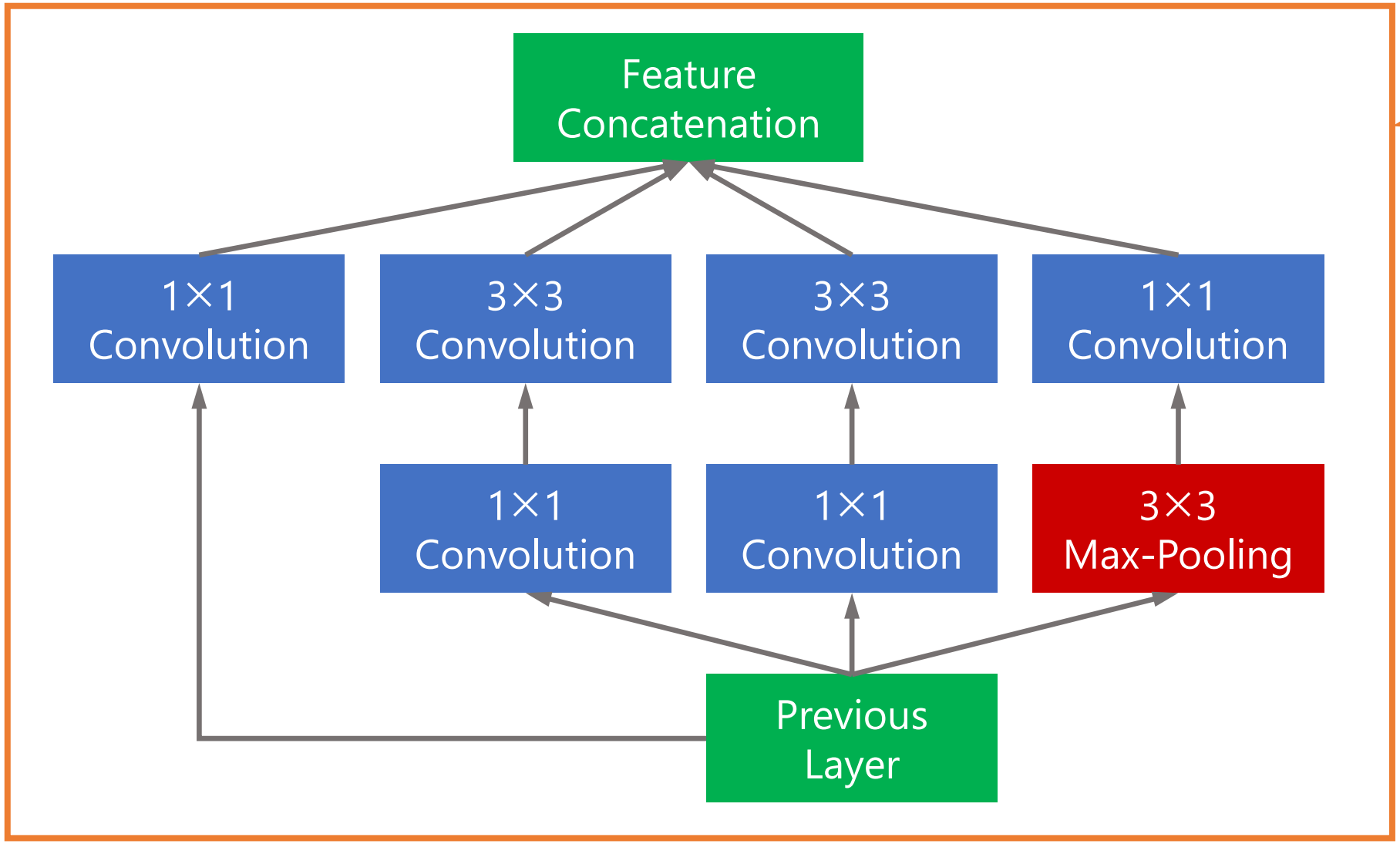
Max-pooling layer

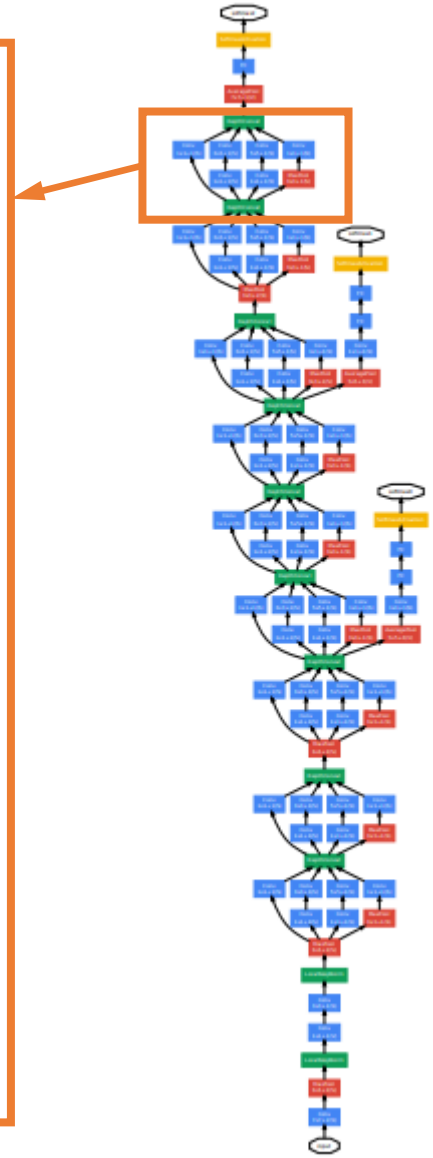
Feature Concatenation

Softmax layer

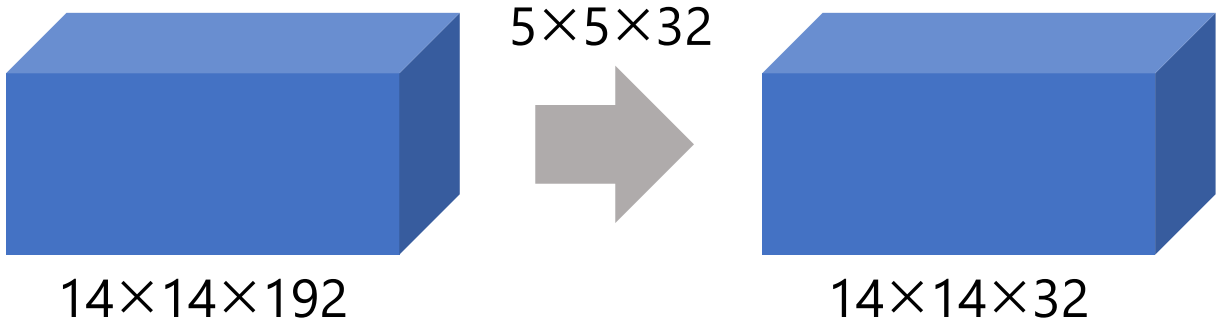
Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, Dl, Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

Inception Module





1 by 1 Convolution



N of Operations :

(14×14×32)×(5×5×192)

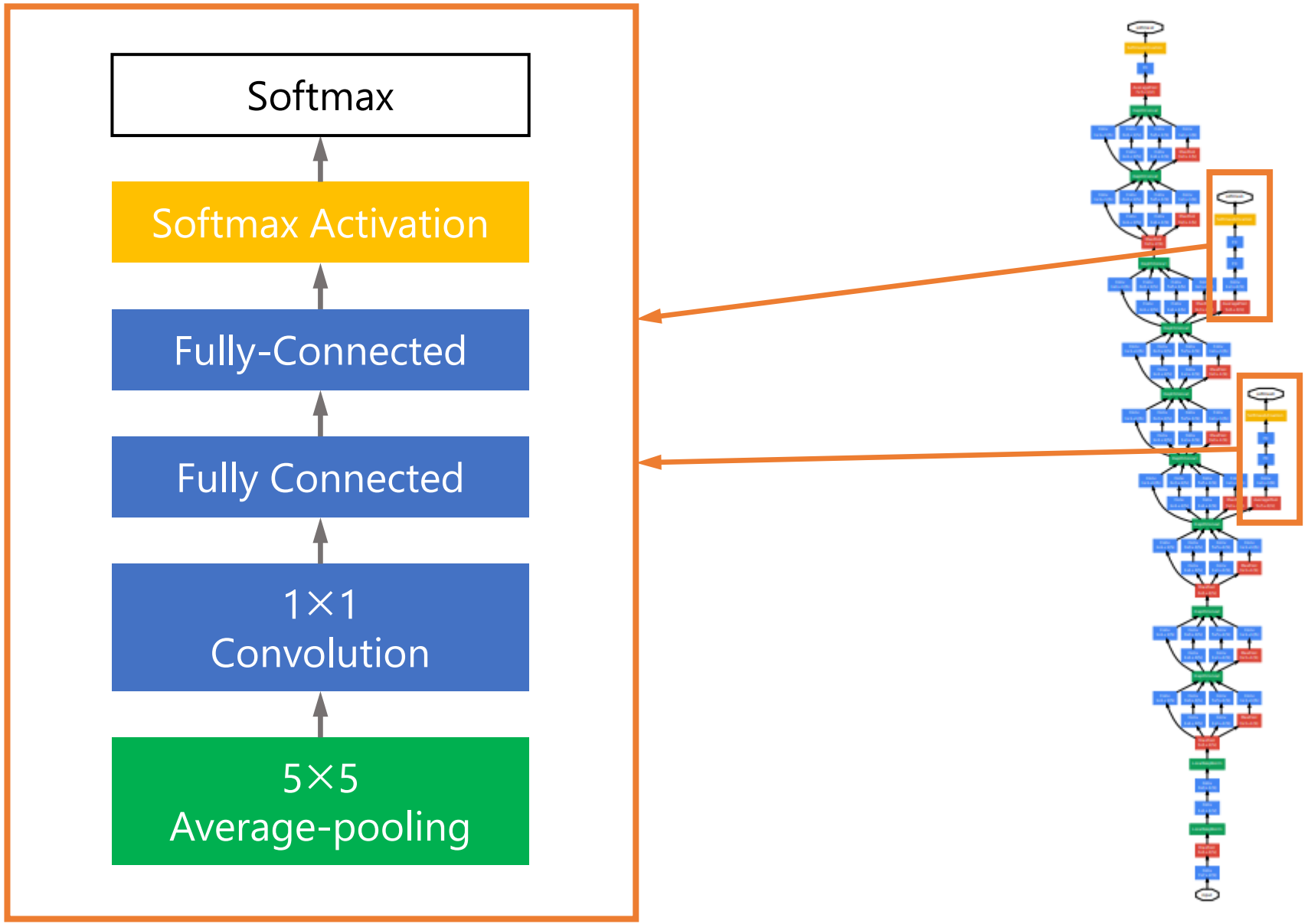
=**30,105,600.**

N of Operations: (14×14×16)×(1×1×192) =602,112

(14×14×32)×(5×5×16) =2,508,800

602,112 + 2,508,800 = **3,110,912**

Auxiliary Classifier



Global Average Pooling

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Szegedy et al., (2015)

17. ResNet

Problem of Deep CNN

- Owing to technical advancement, we can develop a deeper CNN than ever.
- Simply deepening a CNN does not improve the model performance.
- On the contrary, deepening the CNN structure increased the training error.

Degradation Problem

- The training error of a deep learning model stops lowering in the early stage of training.

- It differs from overfitting.
  - Degradation: Adding more layers increases the training error.
  - Overfitting: Adding more layers decreases error only for the training set, not for the test set.

ResNet

- Developed by Kaiminig He and colleagues to address degradation problems.

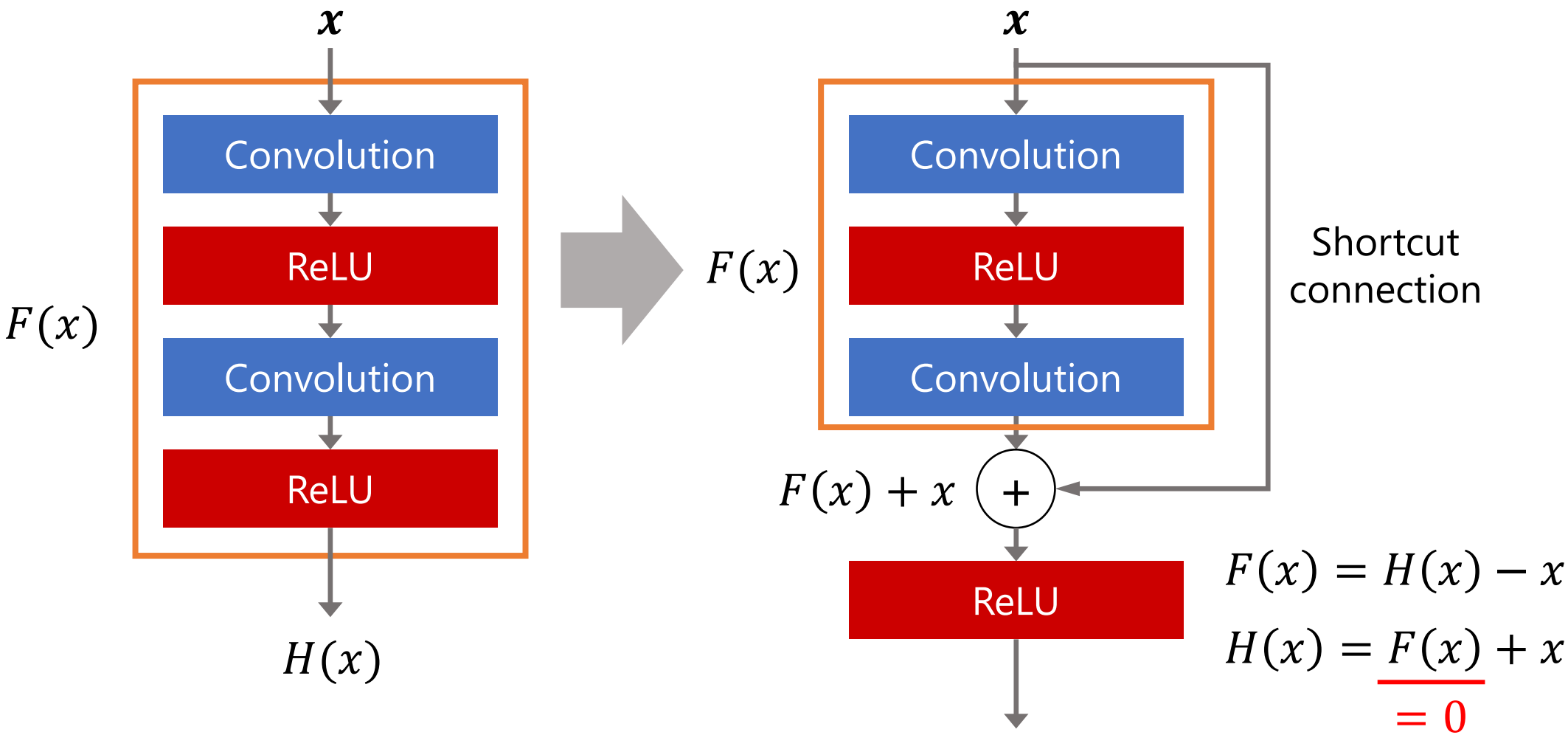
  *He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

- Degradation problems are caused by the difficulty in learning identity mapping

  Identity mapping: $f(x) = x$

- The use of shortcut connection was proposed to address this problem.

Residual Learning



Summary of ResNet

- Residual learning.
- Shortcut connections
- Residual blocks

18. Transfer Learning

Always Need to Develop a Model from Scratch?

- Training a deep CNN models with a large dataset is time-consuming.
- Sometimes we do not have sufficient training data.

Transfer Learning

- We can re-use the knowledge obtained in our previous work.

  Knowledge: The weights already learned in another problem.

- Transfer learning

  - Extracting the learned weights from a trained network
  - Transferring them to another network instead of training a new network from scratch.

Advantages of Transfer Learning

- Transfer learning can shorten the time for training.

- Transfer learning enables us to develop a new model with a relatively small dataset.

Problem Similarity and Transfer Learning

- In transfer learning, we reuse a pre-trained model to similar problems.

  Transfer learning will work well in similar problems.

- If we apply the pre-trained model to a widely different problem, the model will not perform well.

  e.g., Handwritten digits recognition → Speech Recognition

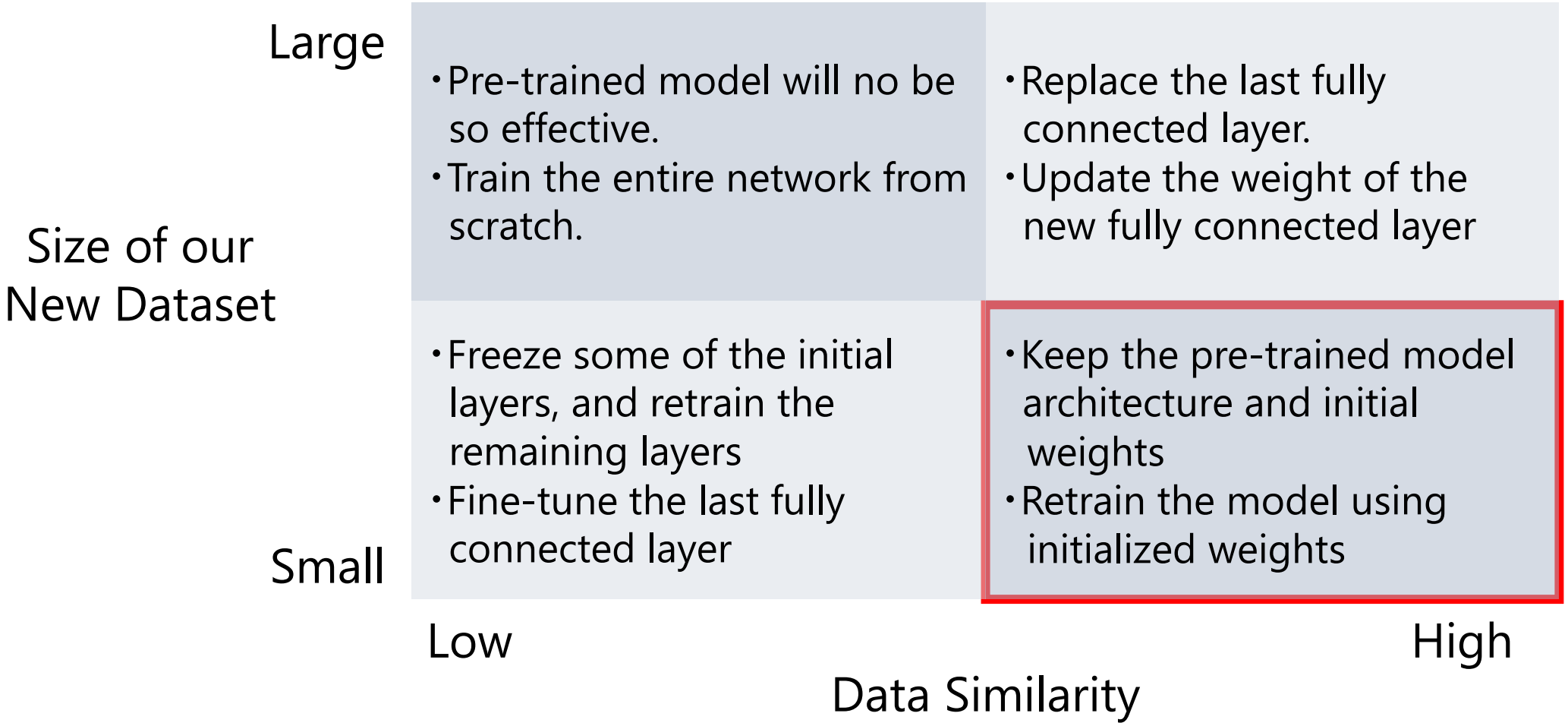
Dataset Size and Similarity

- Our new dataset is small and similar to the original dataset.
- Our new dataset is small and widely different from the original dataset.
- Our new dataset is large and similar to the original dataset.
- Our new dataset is large, and widely different from the original dataset.

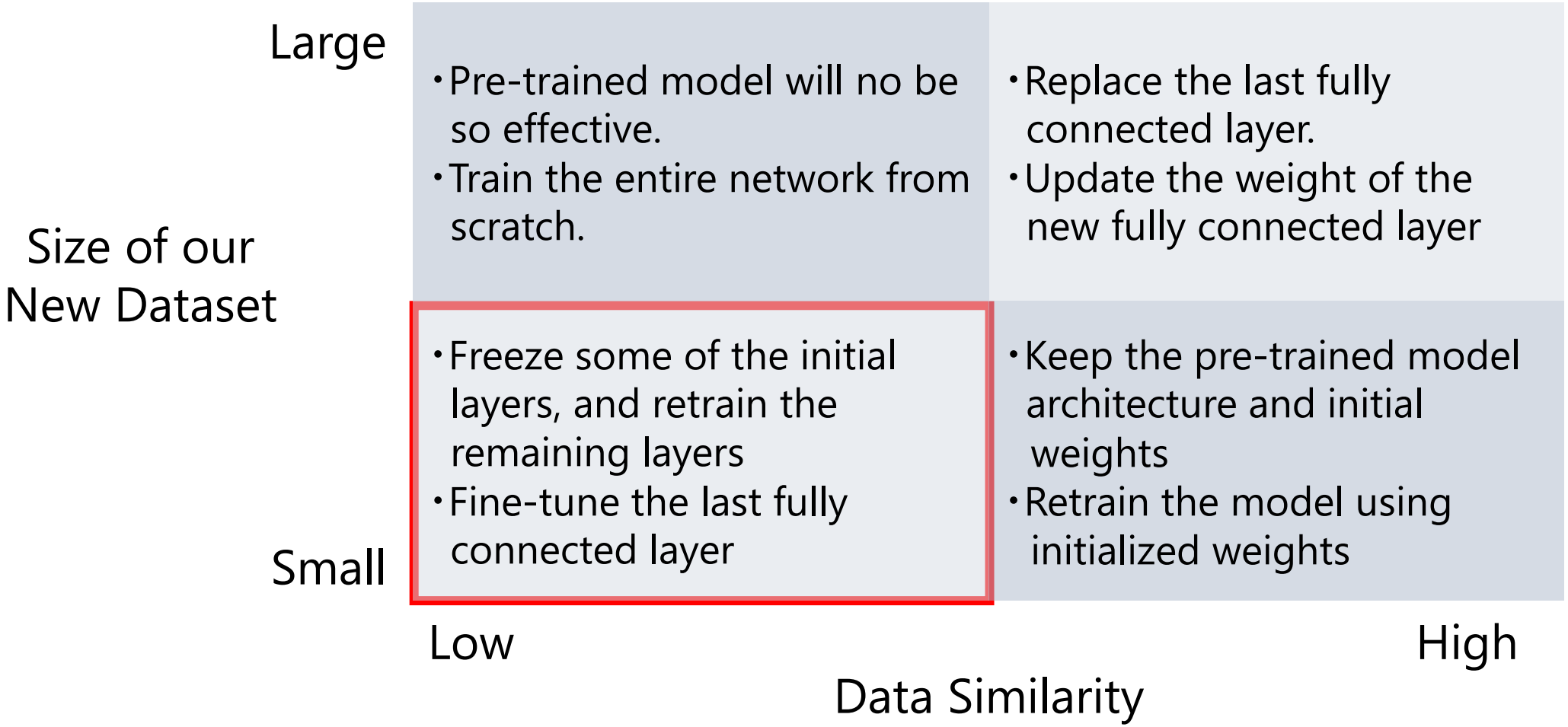
Transfer Learning Approaches

Feature extraction	Extract features using the pre-trained network other than the output layer.
Train only some layers while freezing others	In this approach, we reuse only some layers of the pre-trained model.
Use the architecture of a pre-trained model	Reuse the architecture of the pre-trained model but initialize the weights.

Case 1: Our new dataset is small and similar to the original dataset



Case 2: Our new dataset is small and different from the original dataset.

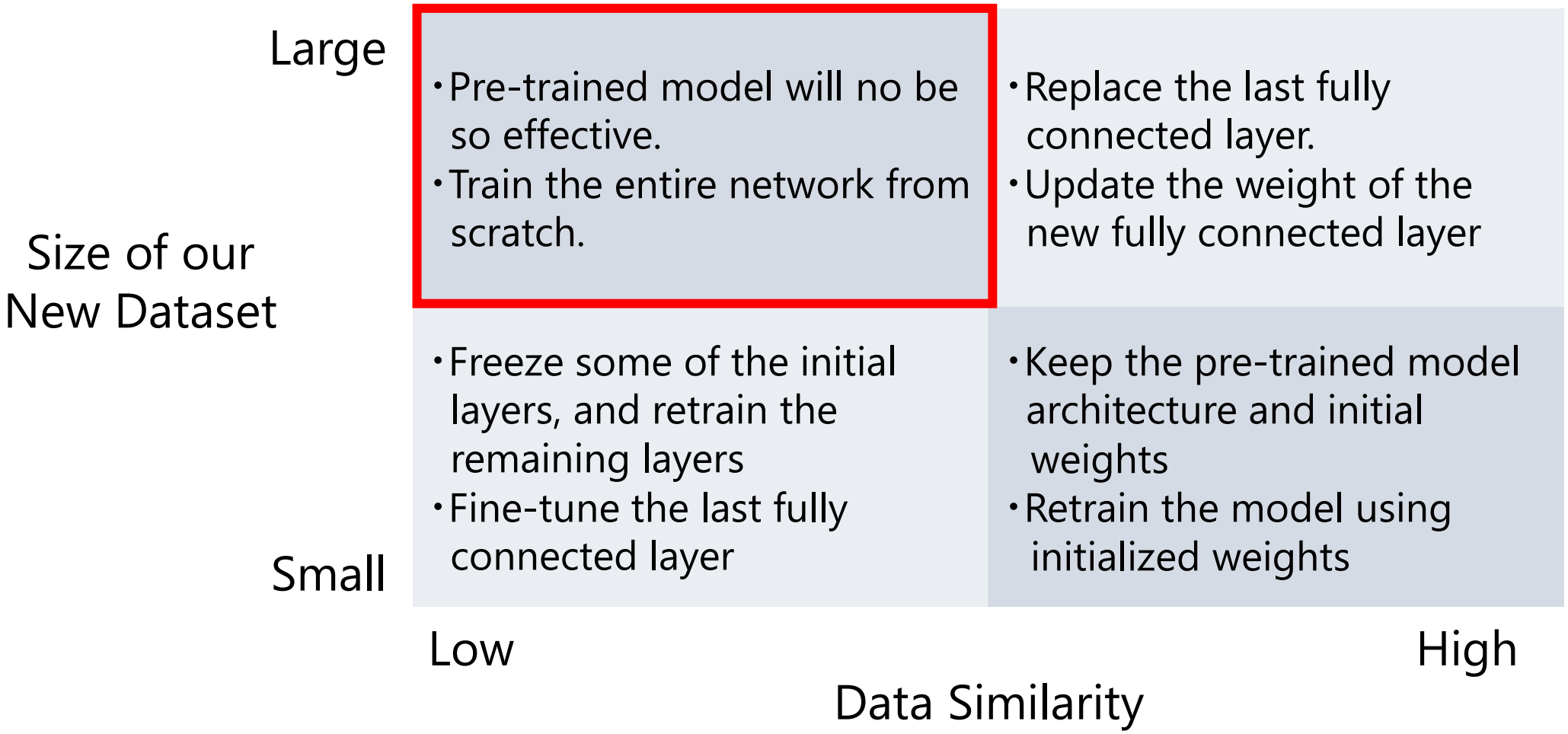


Case 3: Our new dataset is large and similar to the original dataset.

Size of our New Dataset	Low	High
Large	• Pre-trained model will no be so effective. • Train the entire network from scratch.	• Replace the last fully connected layer. • Update the weight of the new fully connected layer
Small	• Freeze some of the initial layers, and retrain the remaining layers • Fine-tune the last fully connected layer	• Keep the pre-trained model architecture and initial weights • Retrain the model using initialized weights

Data Similarity

Case 4: Our new dataset is large, and different from the original dataset.



How to Use Pre-trained Models

- We can download some of the top-performing models freely.
- They are available in the Keras library.

19. Binary Classification with Transfer Learning with Python

Import Libraries

```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from keras.preprocessing.image import ImageDataGenerator

from keras import models
from keras import layers
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.resnet50 import ResNet50

from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Flatten
```

Prepare Dataset

```
# Upload dataset
train_data_dir = "C:/Users/username/cats_or_dogs/train"
test_data_dir = "C:/Users/username/cats_or_dogs/test“

# Create data generator
datagen = ImageDataGenerator(rescale=1.0/255.0)

# Prepare iterators
train_it = datagen.flow_from_directory(train_data_dir,
                                       class_mode='binary', batch_size=32, target_size=(200, 200))
test_it = datagen.flow_from_directory(test_data_dir,
                                      class_mode='binary', batch_size=32, target_size=(200, 200))
```

```
Found 3000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

Load VGG16 Model

```
# Load VGG16 model
vgg_base = VGG16(weights='imagenet',
                 include_top=False,
                 input_shape=(200, 200, 3))

# Show model summary
vgg_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 200, 200, 3)]     0

block1_conv1 (Conv2D)        (None, 200, 200, 64)      1792

block1_conv2 (Conv2D)        (None, 200, 200, 64)      36928

block1_pool (MaxPooling2D)   (None, 100, 100, 64)      0

block2_conv1 (Conv2D)        (None, 100, 100, 128)     73856

block2_conv2 (Conv2D)        (None, 100, 100, 128)     147584

block2_pool (MaxPooling2D)   (None, 50, 50, 128)       0

block3_conv1 (Conv2D)        (None, 50, 50, 256)       295168

block3_conv2 (Conv2D)        (None, 50, 50, 256)       590080

block3_conv3 (Conv2D)        (None, 50, 50, 256)       590080

block3_pool (MaxPooling2D)   (None, 25, 25, 256)       0

block4_conv1 (Conv2D)        (None, 25, 25, 512)       1180160

block4_conv2 (Conv2D)        (None, 25, 25, 512)       2359808

block4_conv3 (Conv2D)        (None, 25, 25, 512)       2359808

block4_pool (MaxPooling2D)   (None, 12, 12, 512)       0

block5_conv1 (Conv2D)        (None, 12, 12, 512)       2359808

block5_conv2 (Conv2D)        (None, 12, 12, 512)       2359808

block5_conv3 (Conv2D)        (None, 12, 12, 512)       2359808

block5_pool (MaxPooling2D)   (None, 6, 6, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Stack Fully Connected Layer

```
# Stack fully connected layers on the base model
vgg_model = models.Sequential()
vgg_model.add(vgg_base)
vgg_model.add(layers.Flatten())
vgg_model.add(layers.Dense(256,
                           activation='relu'))
vgg_model.add(layers.Dense(1,
                           activation='sigmoid'))

vgg_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 6, 6, 512)         14714688

 flatten (Flatten)           (None, 18432)             0

 dense (Dense)               (None, 256)               4718848

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 19,433,793
Trainable params: 19,433,793
Non-trainable params: 0
_________________________________________________________________
```

Model Training

```
# Setting: Freeze VGG16 and Train added fully connected layers
vgg_base.trainable = False

# Compile the model
vgg_model.compile(loss='binary_crossentropy',
                  optimizer='rmsprop',
                  metrics=['acc'])

# Train the model
vgg_history = vgg_model.fit(train_it,
                            steps_per_epoch=len(train_it),
                            validation_data=test_it,
                            validation_steps=len(test_it),
                            epochs=20, verbose=1)
```

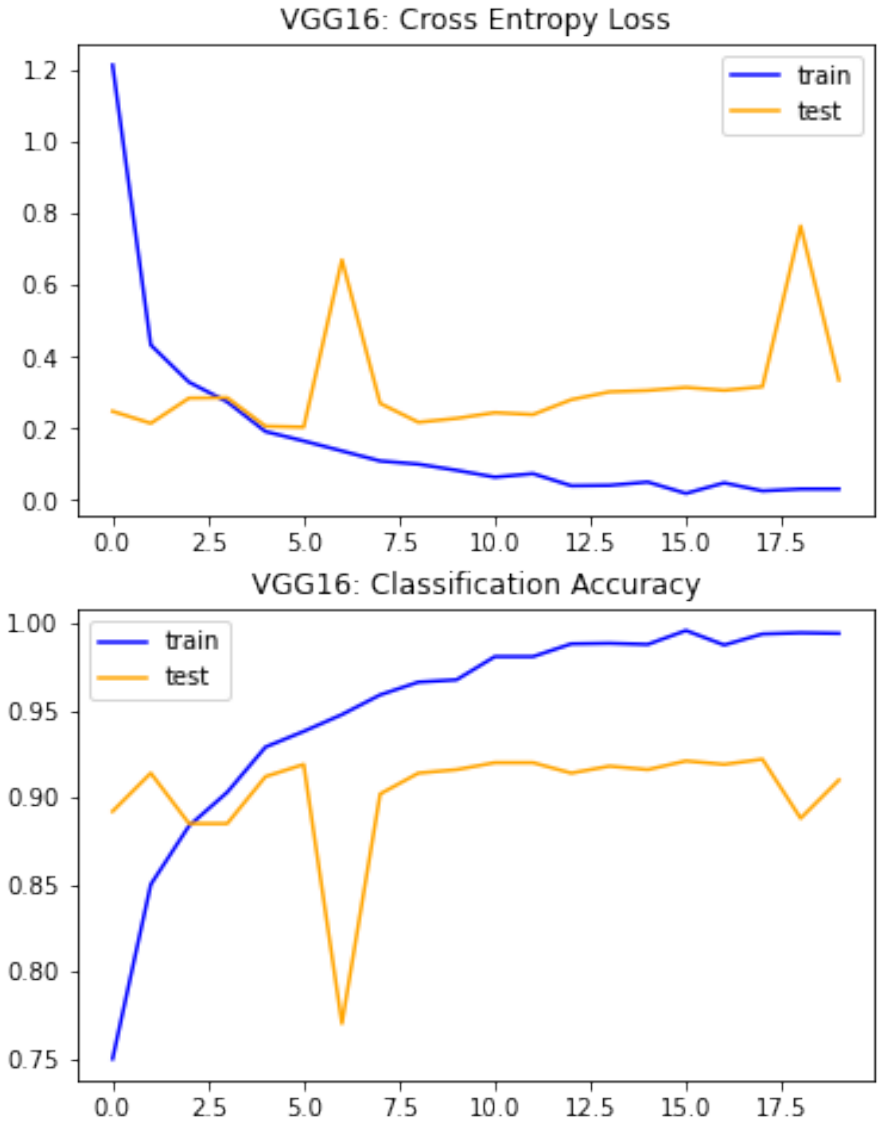
Model Evaluation

```
# Evaluate model
_, vgg_acc = vgg_model.evaluate(test_it, steps=len(test_it), verbose=0)
print('%.3f' % vgg_acc)
```

```
0.910
```

Visualize the Training History

```
# Plot training history
plt.figure(figsize=(6,8))
# Plot loss
plt.subplot(211)
plt.title('VGG16: Cross Entropy Loss')
plt.plot(vgg_history.history['loss'],
         color='blue', label='train')
plt.plot(vgg_history.history['val_loss'],
         color='orange', label='test')
plt.legend()
# Plot accuracy
plt.subplot(212)
plt.title('VGG16: Classification Accuracy')
plt.plot(vgg_history.history['acc'], color='blue',
         label='train')
plt.plot(vgg_history.history['val_acc'], color='orange',
         label='test')
plt.legend()
```



GoogleNet and ResNet

```
# Load GoogLeNet model
ggln_base = InceptionV3(weights='imagenet',
                        include_top=False,
                        input_shape=(200, 200, 3))
```

```
# Load ResNet model
resn_base = ResNet50(weights='imagenet',
                     include_top=False,
                     input_shape=(200, 200, 3))
```

Performance Comparison