AI VIETNAM
Free AI Course 2020

Convolutional Neural Network

(Draft)

Quang-Vinh Dinh
Ph.D. in Computer Science

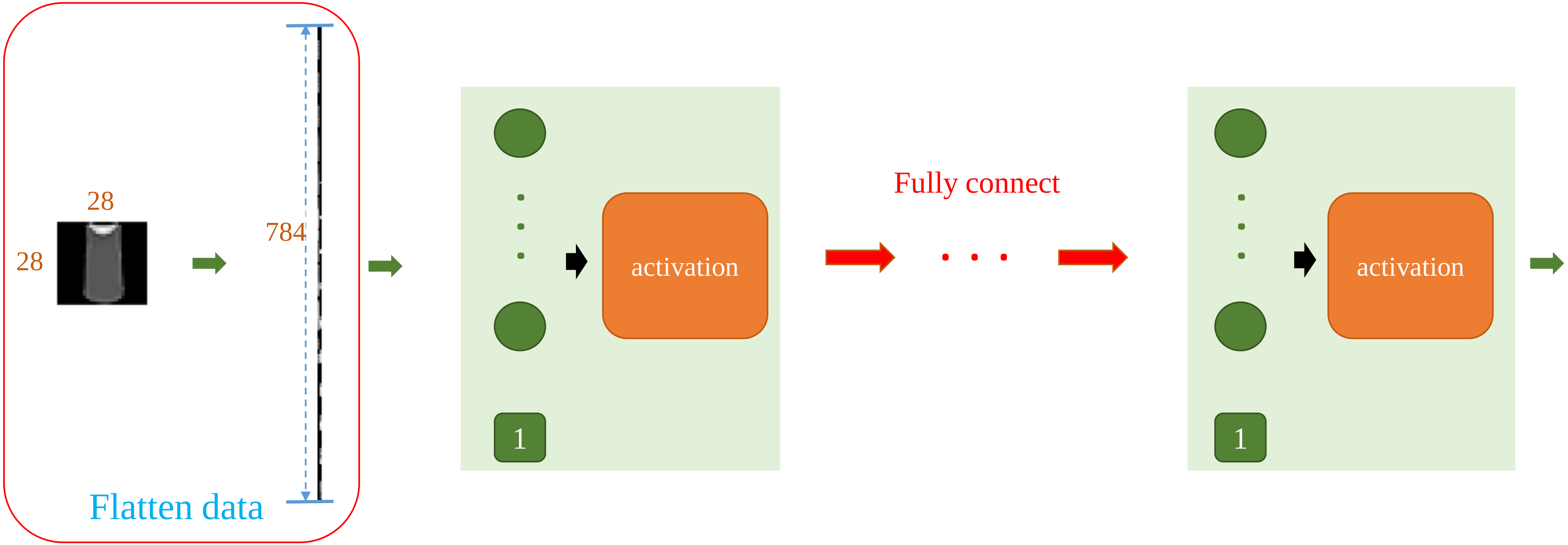
Year 2020

Outline

- From MLP to CNN
- Feature Map Down-sampling
- Padding
- 1x1 Convolution
- Image classification: Cifar-10 data

From MLP to CNN

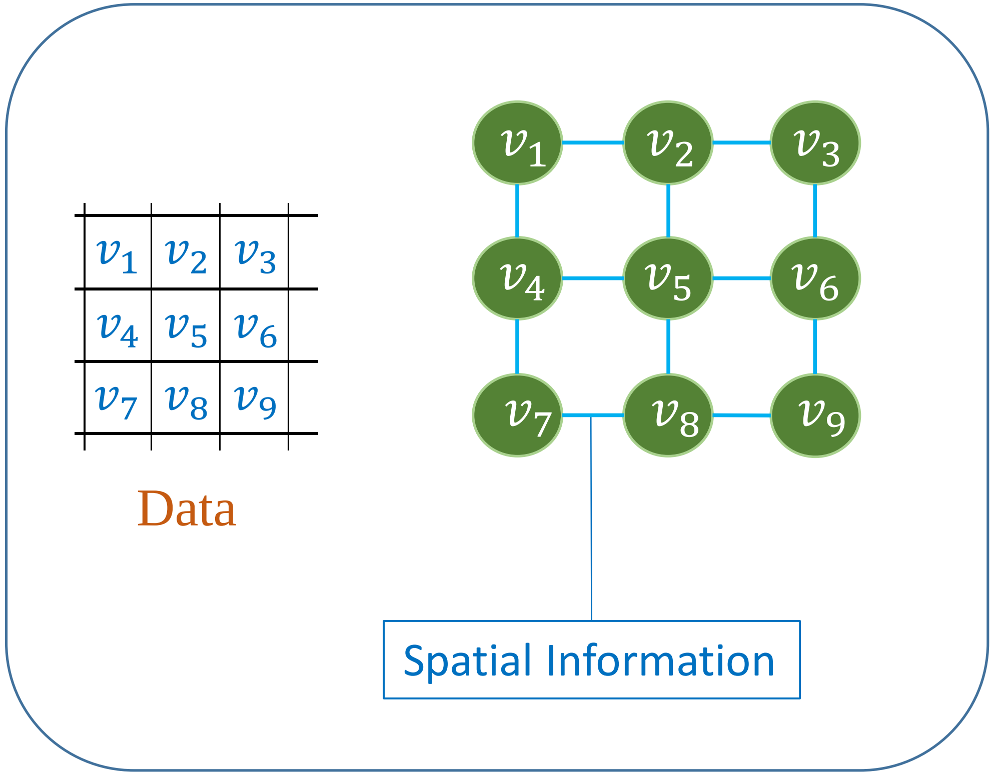
❖ Multi-layer Perceptron

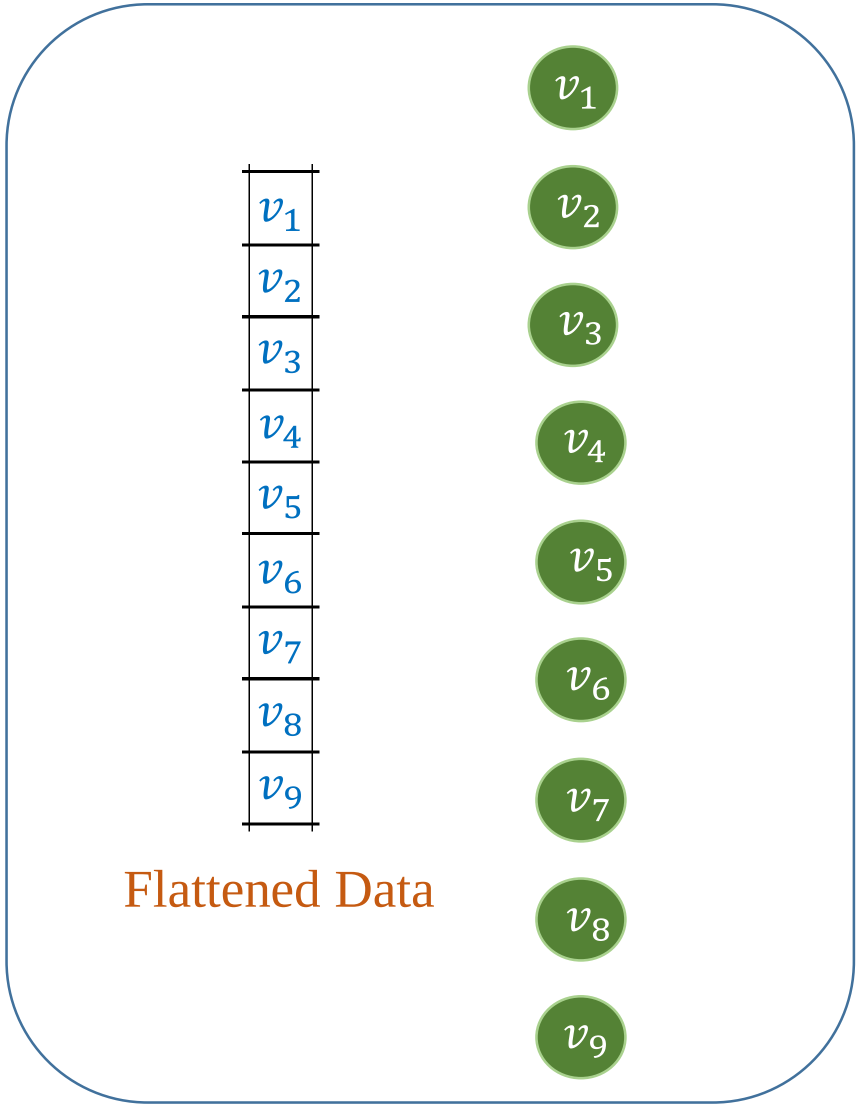


Problem: Remove spatial information of the data
Inefficiently have a large amount of parameters

From MLP to CNN

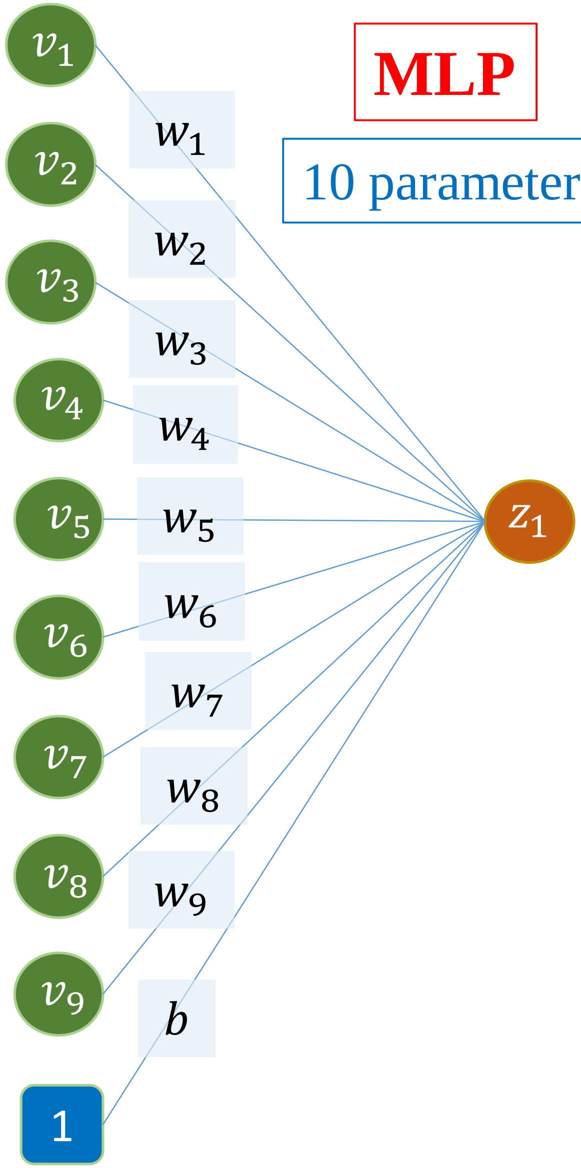
❖ Problem of flattening data





Remove spatial information of the data

From MLP to CNN



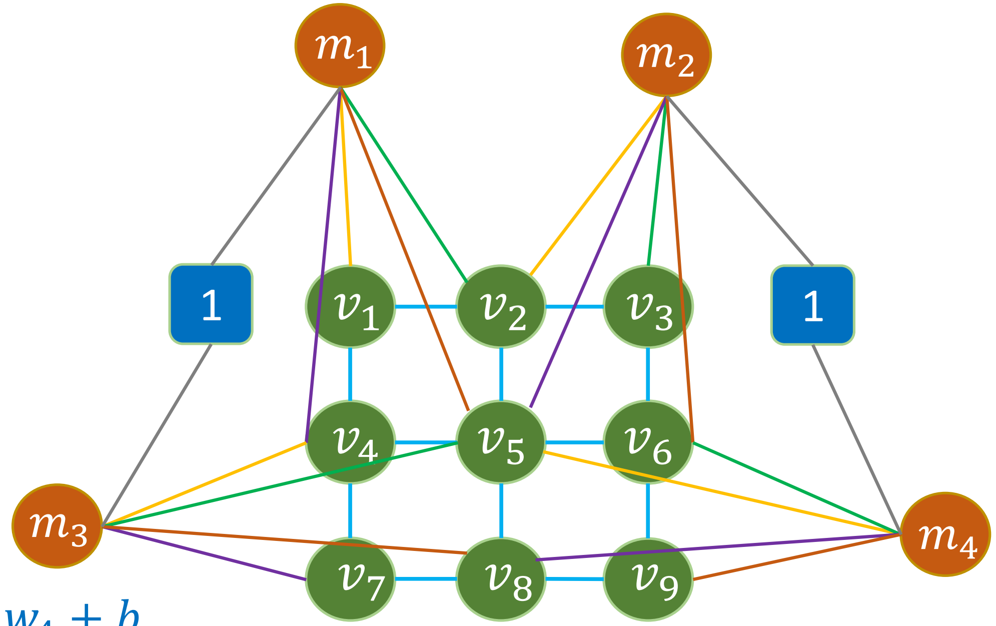
MLP

10 parameters

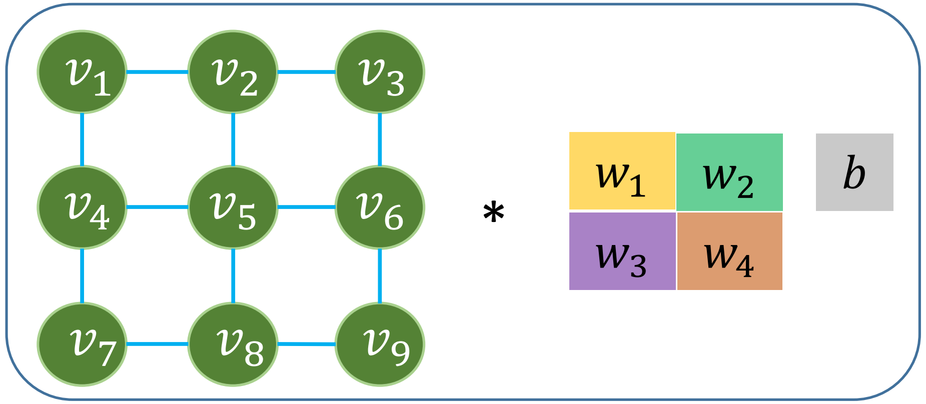
CNN

5 parameters

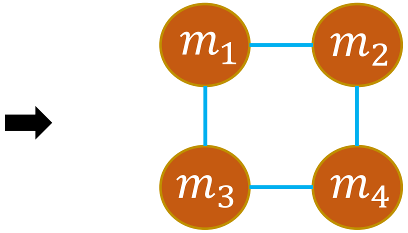
Kernel of parameters



$$m_1 = v_1w_1 + v_2w_2 + v_4w_3 + v_5w_4 + b$$



Convolution(~Correlation)

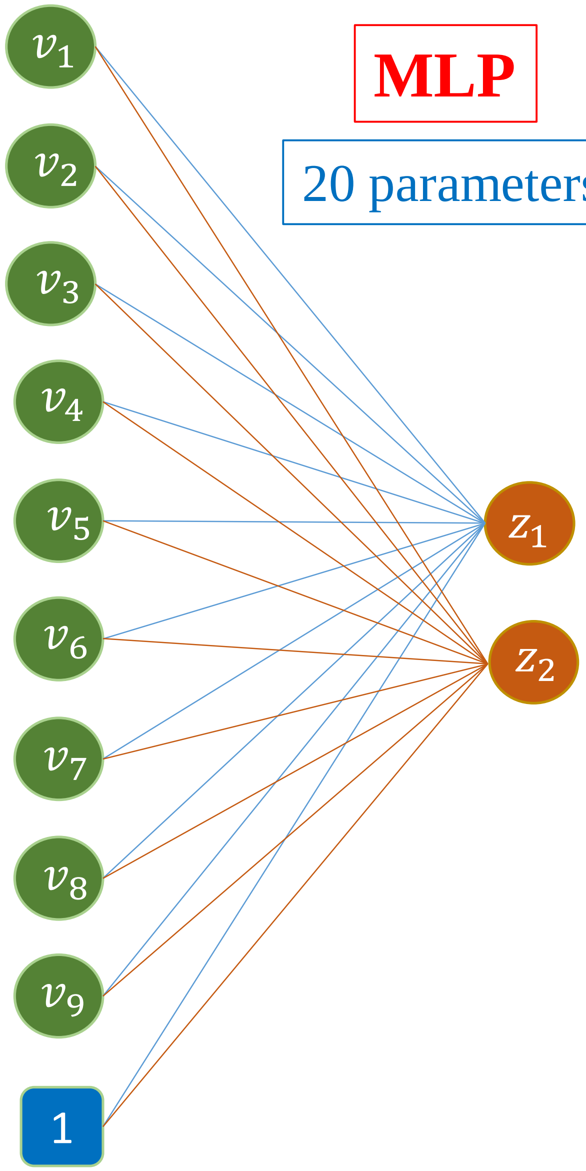


Feature Map

From MLP to CNN

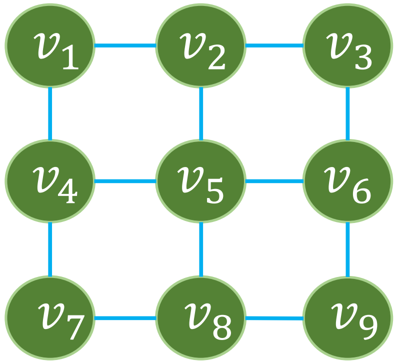
MLP

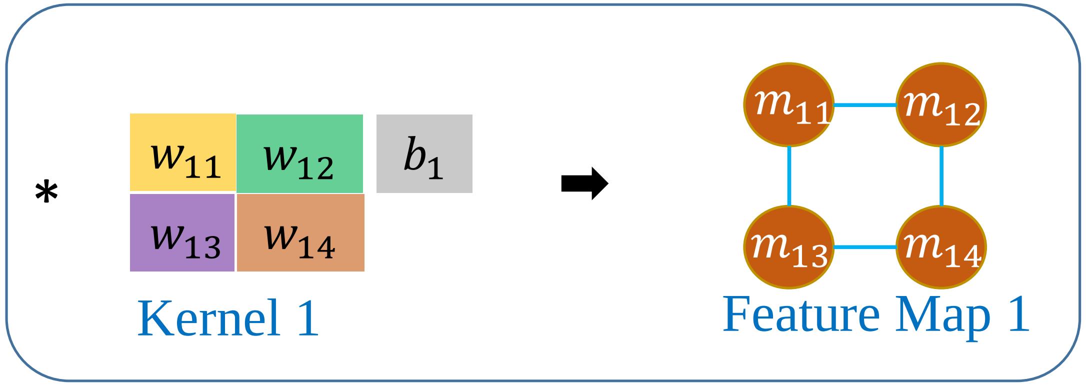
20 parameters

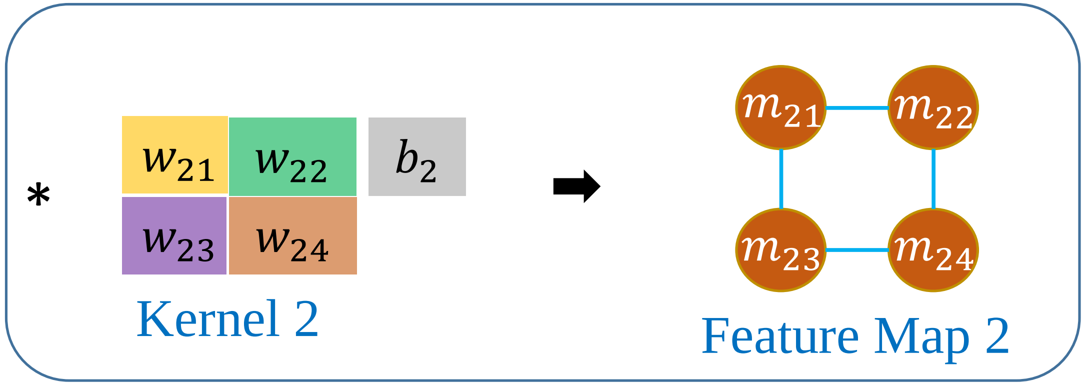


CNN

10 parameters



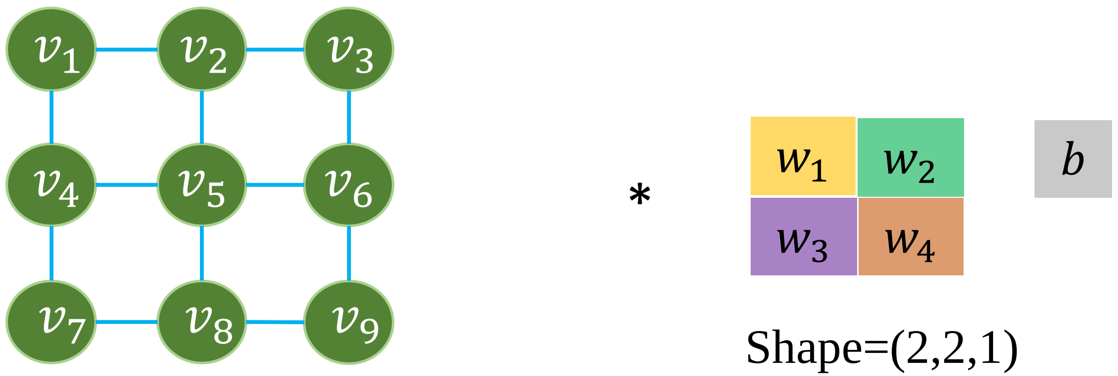




Kernel 1 $\neq$ Kernel 2

From MLP to CNN

❖ Understand convolution



(Height=3, Width=3, Channel=1)

Shape=(3,3,1)

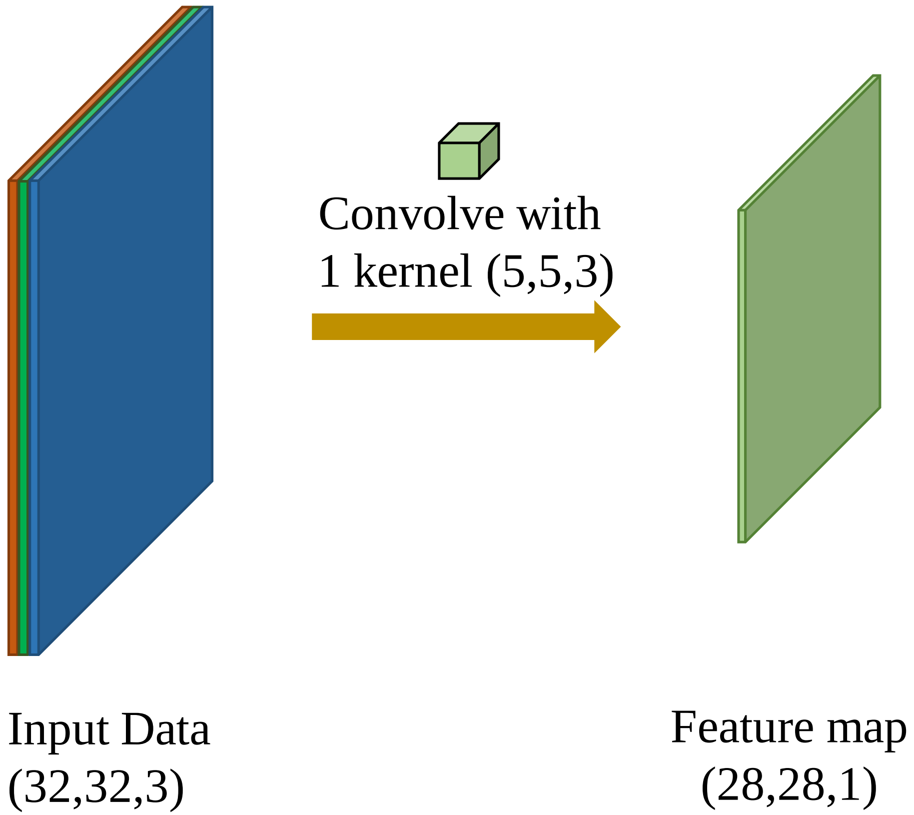
Shape=(2,2,1)

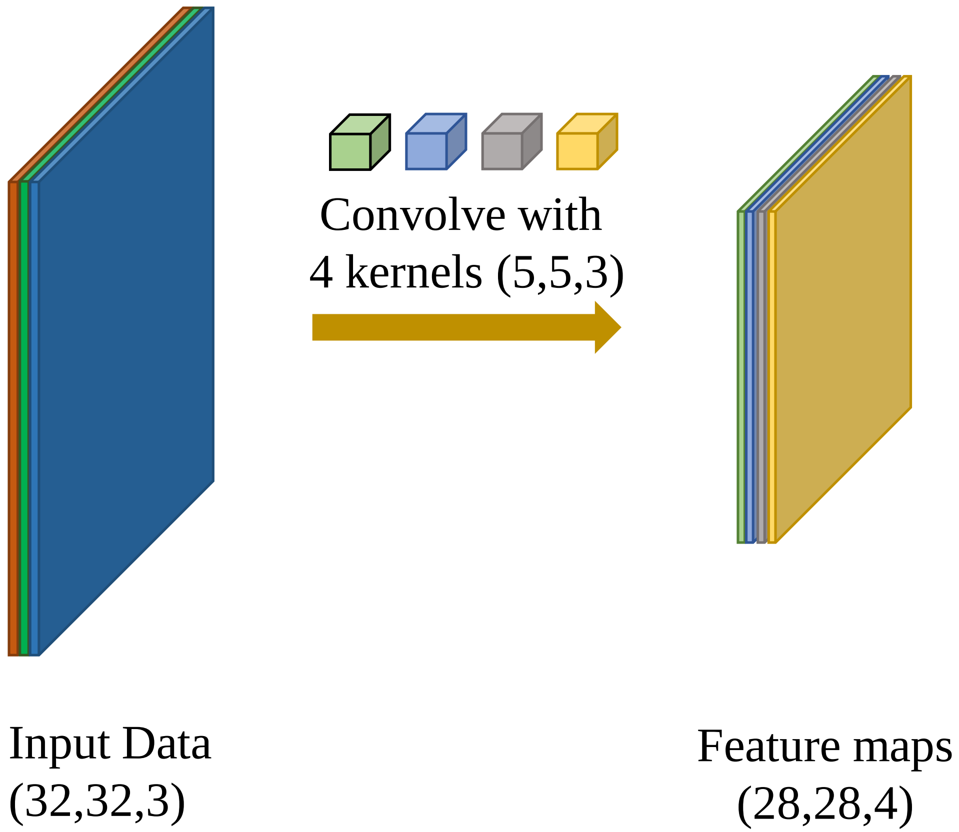
#parameters (including bias) = 5

#channels of data $\overset{\text{must}}{=}$ #channels of kernel

From MLP to CNN

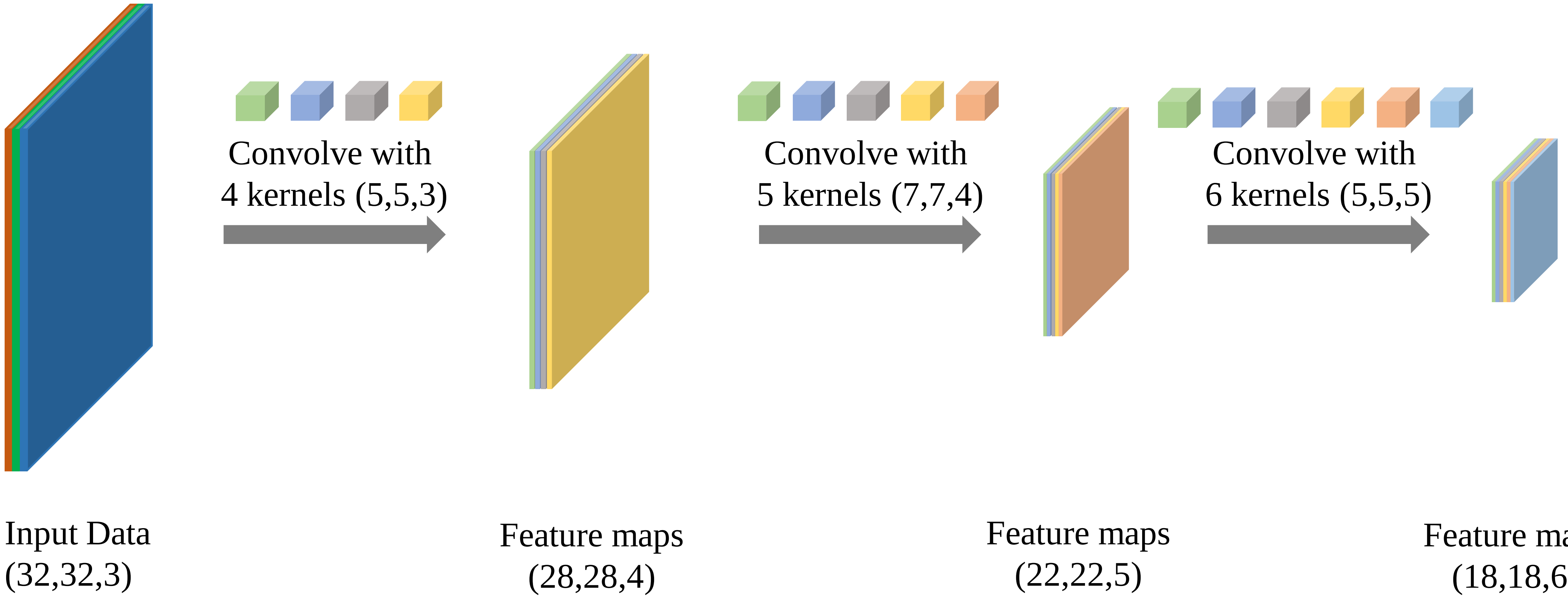
❖ Understand convolution





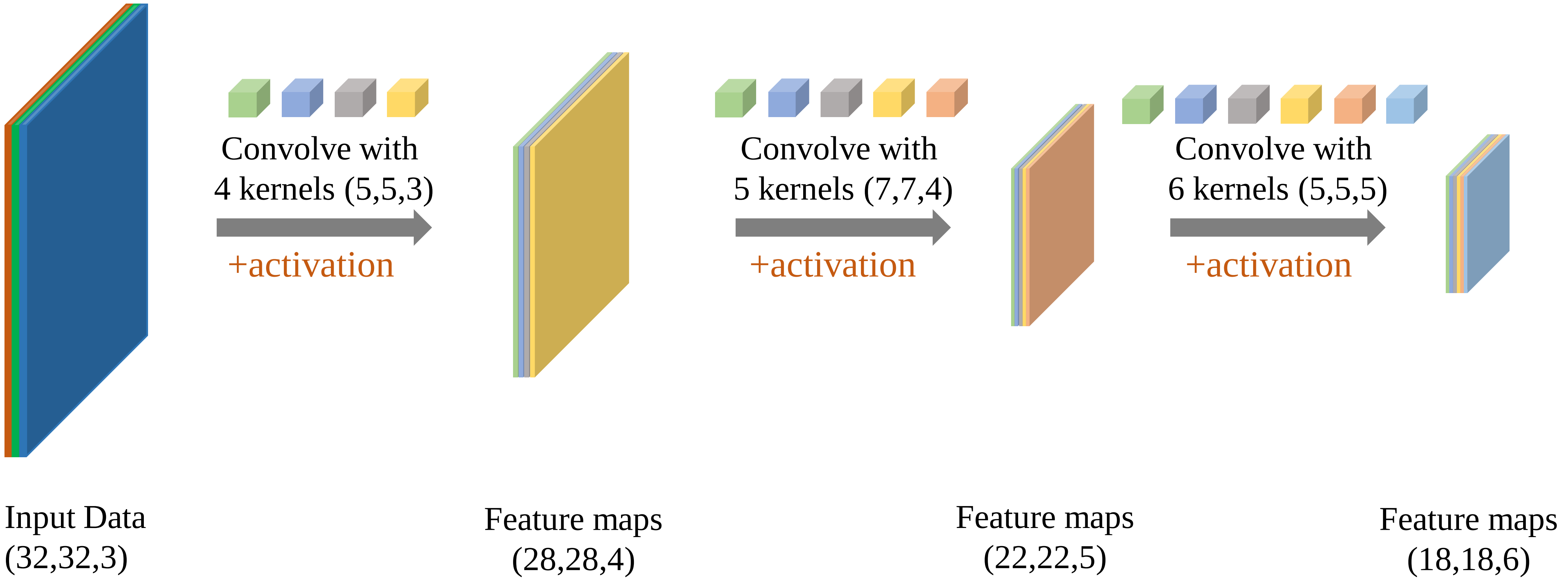
From MLP to CNN

❖ A stack of convolutions



From MLP to CNN

❖ **A stack of pairs of convolution+activation**



From MLP to CNN

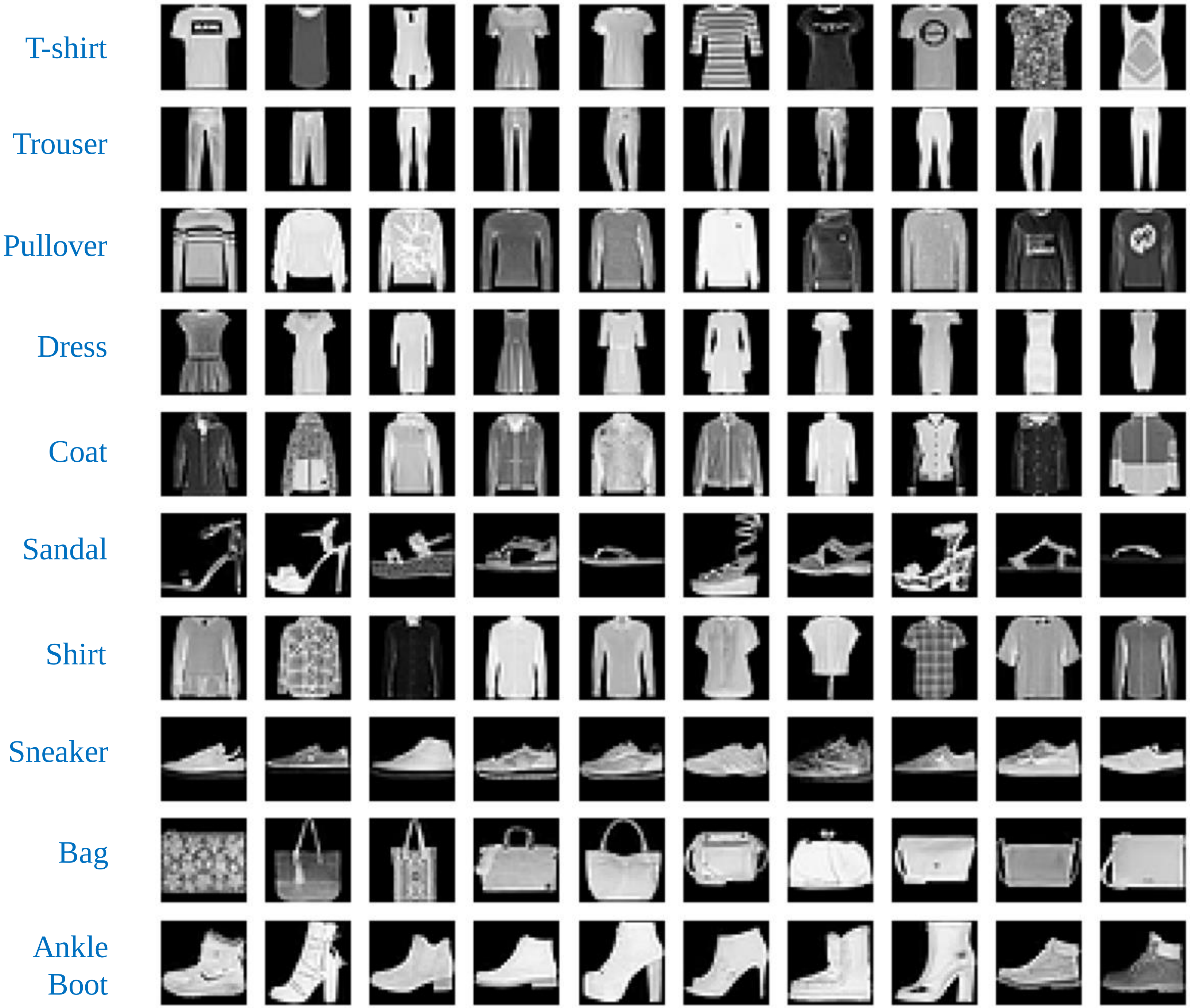
Fashion-MNIST dataset

Grayscale images

Resolution=28x28

Training set: 60000 samples

Testing set: 10000 samples



From MLP to CNN

```python
# model
model = keras.models.Sequential()

# input is with the shape of (28, 28, 1)
model.add(tf.keras.Input(shape=(28, 28, 1)))

# Convolve with 32 (7x7) kernel; Output: (22x22x32)
model.add(keras.layers.Conv2D(32, (7, 7), activation='relu'))
# Convolve with 64 (7x7) kernel; Output: (16x16x64)
model.add(keras.layers.Conv2D(64, (7, 7), activation='relu'))
# Convolve with 128 (7x7) kernel; Output: (10x10x128)
model.add(keras.layers.Conv2D(128, (7, 7), activation='relu'))
# Convolve with 256 (7x7) kernel; Output: (4x4x256)
model.add(keras.layers.Conv2D(256, (7, 7), activation='relu'))

# flatten
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(10, activation='softmax'))

# compile and train
model.compile(optimizer='adam', metrics=['accuracy'],
              loss='sparse_categorical_crossentropy')
model.fit(train_images, train_labels, epochs=10)

# testing
test_loss, test_acc = model.evaluate(test_images,
                                     test_labels, verbose=2)
print('Test accuracy:', test_acc)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 22, 22, 32)        1600
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 64)        100416
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 128)       401536
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 256)         1605888
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dense (Dense)                (None, 10)                40970
=================================================================
Total params: 2,150,410
Trainable params: 2,150,410
Non-trainable params: 0
```

```
Train on 60000 samples
Epoch 1/10
60000/60000 - 577s 10ms/sample - loss: 0.5021 - accuracy: 0.8138
Epoch 2/10
60000/60000 - 578s 10ms/sample - loss: 0.3388 - accuracy: 0.8757
Epoch 3/10
60000/60000 - 567s 9ms/sample - loss: 0.2993 - accuracy: 0.8880
Epoch 4/10
60000/60000 - 545s 9ms/sample - loss: 0.2726 - accuracy: 0.8995
Epoch 5/10
60000/60000 - 1254s 21ms/sample - loss: 0.2475 - accuracy: 0.9083
Epoch 6/10
60000/60000 - 563s 9ms/sample - loss: 0.2201 - accuracy: 0.9172
Epoch 7/10
60000/60000 - 571s 10ms/sample - loss: 0.1983 - accuracy: 0.9254
Epoch 8/10
60000/60000 - 581s 10ms/sample - loss: 0.1806 - accuracy: 0.9340
Epoch 9/10
60000/60000 - 581s 10ms/sample - loss: 0.1517 - accuracy: 0.9431
Epoch 10/10
60000/60000 - 2145s 36ms/sample - loss: 0.1378 - accuracy: 0.9495
10000/1 - 19s - loss: 1.2228 - accuracy: 0.8858

Test accuracy: 0.8858
```

Outline

- From MLP to CNN
- Feature Map Down-sampling
- Padding
- 1x1 Convolution
- Image classification: Cifar-10 data

Down-sample Feature Map

Max pooling: Features are preserved

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data

2x2 max pooling (

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

) =

m_1	m_2
m_3	m_4

$$m_1 = \max(v_1, v_2, v_5, v_6)$$
$$m_2 = \max(v_3, v_4, v_7, v_8)$$
$$m_3 = \max(v_9, v_{10}, v_{13}, v_{14})$$
$$m_4 = \max(v_{11}, v_{12}, v_{15}, v_{16})$$

Feature map (220x220)

max pooling (2x2)

Feature map (110x110)

max pooling (2x2)

Feature map (55x55)

Down-sample Feature Map

Example 1: Using max pooling

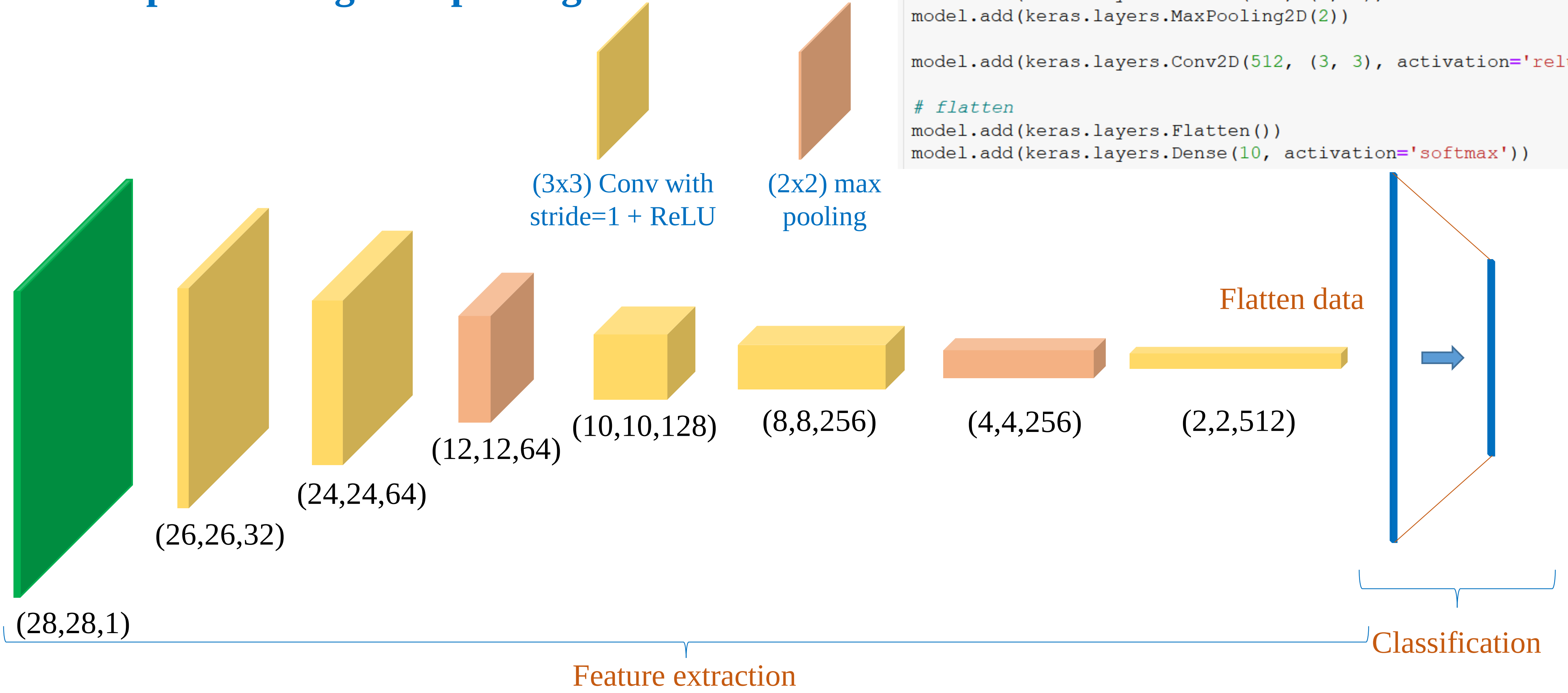
```
# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(28, 28, 1)))

model.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
model.add(keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D(2))

model.add(keras.layers.Conv2D(128, (3, 3), activation='relu'))
model.add(keras.layers.Conv2D(256, (3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D(2))

model.add(keras.layers.Conv2D(512, (3, 3), activation='relu'))

# flatten
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(10, activation='softmax'))
```



Down-sample Feature Map

❖ Example 1: Using max pooling

```
Epoch 1/10
60000/60000 [==============================] - 281s 5ms/sample - loss: 0.4243 - accuracy: 0.8423
Epoch 2/10
60000/60000 [==============================] - 295s 5ms/sample - loss: 0.2678 - accuracy: 0.9029
Epoch 3/10
60000/60000 [==============================] - 305s 5ms/sample - loss: 0.2200 - accuracy: 0.9199
Epoch 4/10
60000/60000 [==============================] - 298s 5ms/sample - loss: 0.1832 - accuracy: 0.9330
Epoch 5/10
60000/60000 [==============================] - 298s 5ms/sample - loss: 0.1510 - accuracy: 0.9447
Epoch 6/10
60000/60000 [==============================] - 302s 5ms/sample - loss: 0.1245 - accuracy: 0.9540
Epoch 7/10
60000/60000 [==============================] - 306s 5ms/sample - loss: 0.0988 - accuracy: 0.9639
Epoch 8/10
60000/60000 [==============================] - 292s 5ms/sample - loss: 0.0847 - accuracy: 0.9684
Epoch 9/10
60000/60000 [==============================] - 291s 5ms/sample - loss: 0.0709 - accuracy: 0.9741
Epoch 10/10
60000/60000 [==============================] - 289s 5ms/sample - loss: 0.0606 - accuracy: 0.9786
10000/1 - 8s - loss: 0.4003 - accuracy: 0.9125

Test accuracy: 0.9125
```

Down-sample Feature Map

❖ Convolve with stride

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data **D**

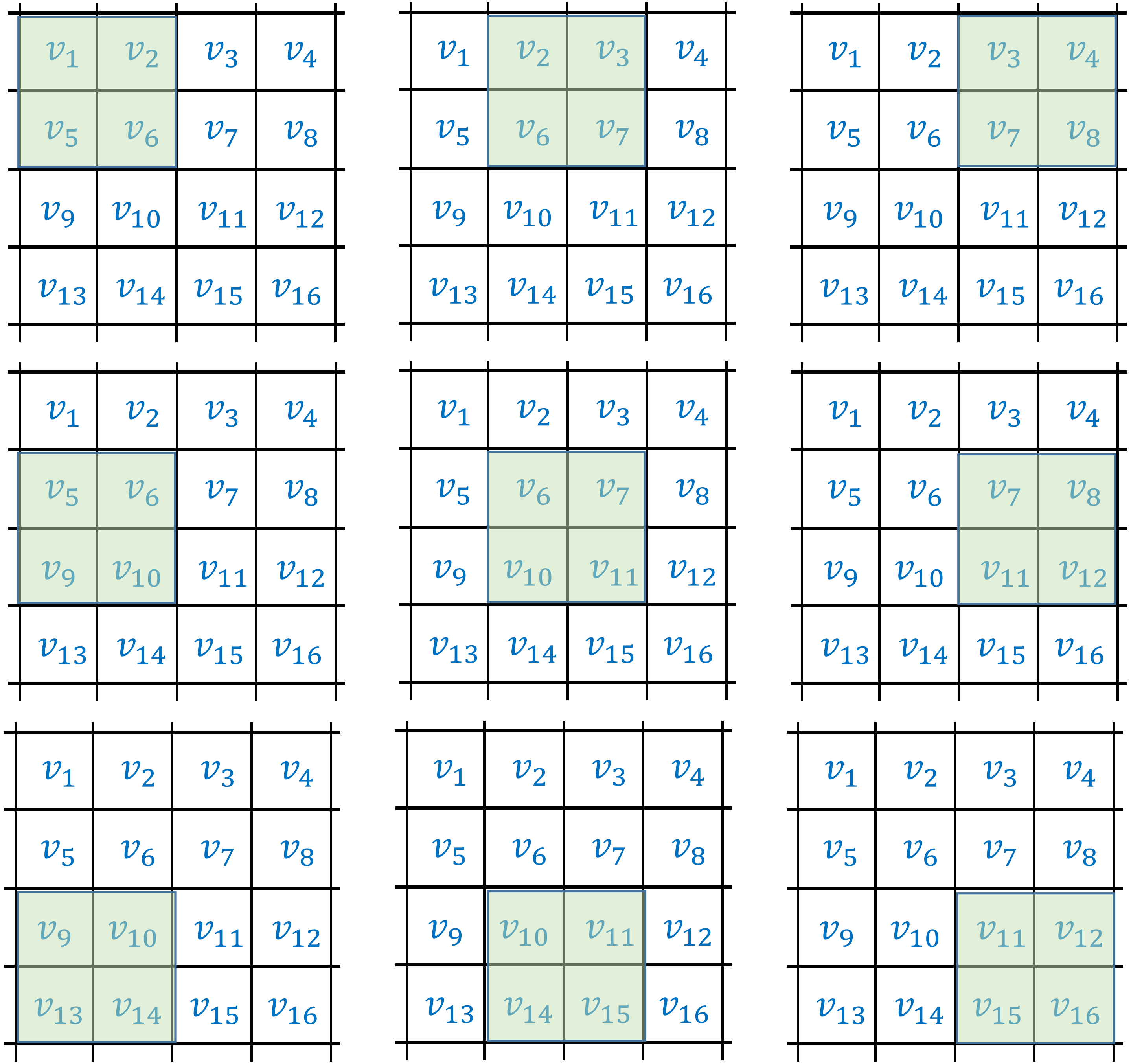
w_1	w_2	b
w_3	w_4	

Kernel of parameters

Convolve **D** with stride=1

m_1	m_2	m_3
m_4	m_5	m_6
m_7	m_8	m_9

Output



Down-sample Feature Map

❖ **Convolve with stride**

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data **D**

w_1	w_2
w_3	w_4

b

Kernel of parameters

Convolve **D** with stride=2

m_1	m_2
m_3	m_4

Output

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

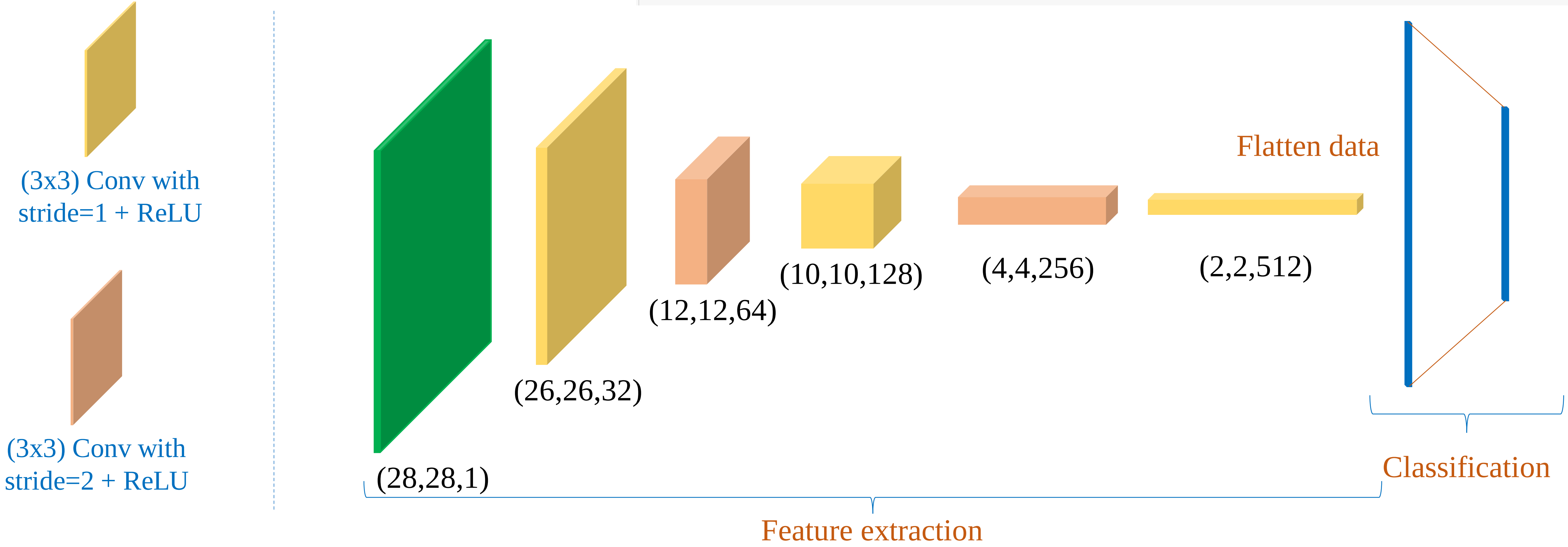
Down-sample Feature Map

Example 2: Using Conv with strides

```
# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(28, 28, 1)))

model.add(keras.layers.Conv2D(32, (3, 3), strides=1, activation='relu'))
model.add(keras.layers.Conv2D(64, (3, 3), strides=2, activation='relu'))
model.add(keras.layers.Conv2D(128, (3, 3), strides=1, activation='relu'))
model.add(keras.layers.Conv2D(256, (3, 3), strides=2, activation='relu'))
model.add(keras.layers.Conv2D(512, (3, 3), strides=1, activation='relu'))

# flatten
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(10, activation='softmax'))
```



Down-sample Feature Map

❖ Example 2: Using Conv with strides

```
Epoch 1/10
60000/60000 [==============================] - 276s 5ms/sample - loss: 0.4500 - accuracy: 0.8350
Epoch 2/10
60000/60000 [==============================] - 268s 4ms/sample - loss: 0.2934 - accuracy: 0.8922
Epoch 3/10
60000/60000 [==============================] - 285s 5ms/sample - loss: 0.2376 - accuracy: 0.9108
Epoch 4/10
60000/60000 [==============================] - 291s 5ms/sample - loss: 0.1942 - accuracy: 0.9283
Epoch 5/10
60000/60000 [==============================] - 288s 5ms/sample - loss: 0.1536 - accuracy: 0.9420
Epoch 6/10
60000/60000 [==============================] - 288s 5ms/sample - loss: 0.1198 - accuracy: 0.9548
Epoch 7/10
60000/60000 [==============================] - 289s 5ms/sample - loss: 0.0934 - accuracy: 0.9653
Epoch 8/10
60000/60000 [==============================] - 199s 3ms/sample - loss: 0.0746 - accuracy: 0.9730
Epoch 9/10
60000/60000 [==============================] - 200s 3ms/sample - loss: 0.0625 - accuracy: 0.9772
Epoch 10/10
60000/60000 [==============================] - 199s 3ms/sample - loss: 0.0566 - accuracy: 0.9797
10000/1 - 5s - loss: 0.3381 - accuracy: 0.9055

Test accuracy: 0.9055
```

Outline

- From MLP to CNN
- Feature Map Down-sampling
- Padding
- 1x1 Convolution
- Image classification: Cifar-10 data

Padding

Goal: Keep resolution of feature map

v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data $\boldsymbol{D}$
(4x4)

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

b

Kernel of parameters

Without using padding
or padding=0

Convolve with stride=1 $(\boldsymbol{D}) =$

m_1	m_2
m_4	m_5

Output
(2x2)

Padding = 1

v	v	v	v	v	v
v	v_1	v_2	v_3	v_4	v
v	v_5	v_6	v_7	v_8	v
v	v_9	v_{10}	v_{11}	v_{12}	v
v	v_{13}	v_{14}	v_{15}	v_{16}	v
v	v	v	v	v	v

Data $\boldsymbol{D}_p$

Convolve with stride=1 $(\boldsymbol{D}_p) =$

m_1	m_2	m_3	m_4
m_5	m_6	m_7	m_8
m_9	m_{10}	m_{11}	m_{12}
m_{13}	m_{14}	m_{15}	m_{16}

Output
(4x4)

Padding

Example

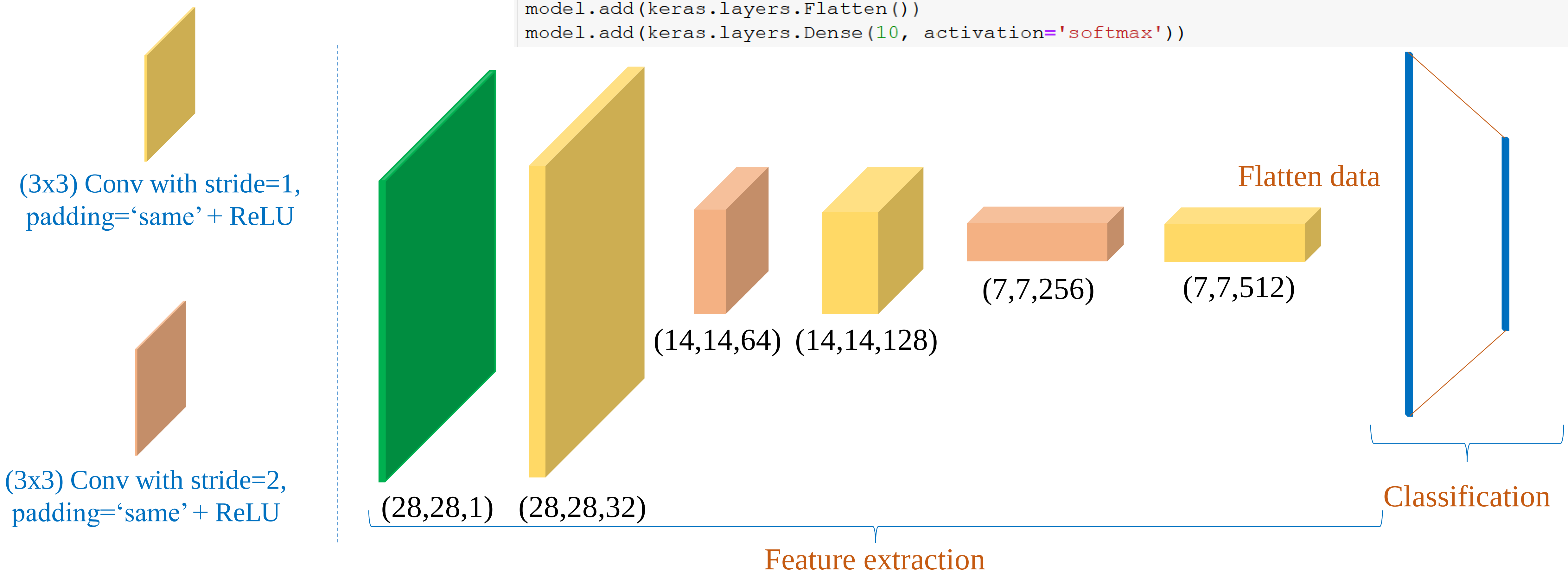
```python
# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(28, 28, 1)))

model.add(keras.layers.Conv2D(32, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(64, (3, 3), strides=2, padding='same', activation='relu'))

model.add(keras.layers.Conv2D(128, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(256, (3, 3), strides=2, padding='same', activation='relu'))

model.add(keras.layers.Conv2D(512, (3, 3), strides=1, padding='same', activation='relu'))

# flatten
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(10, activation='softmax'))
```

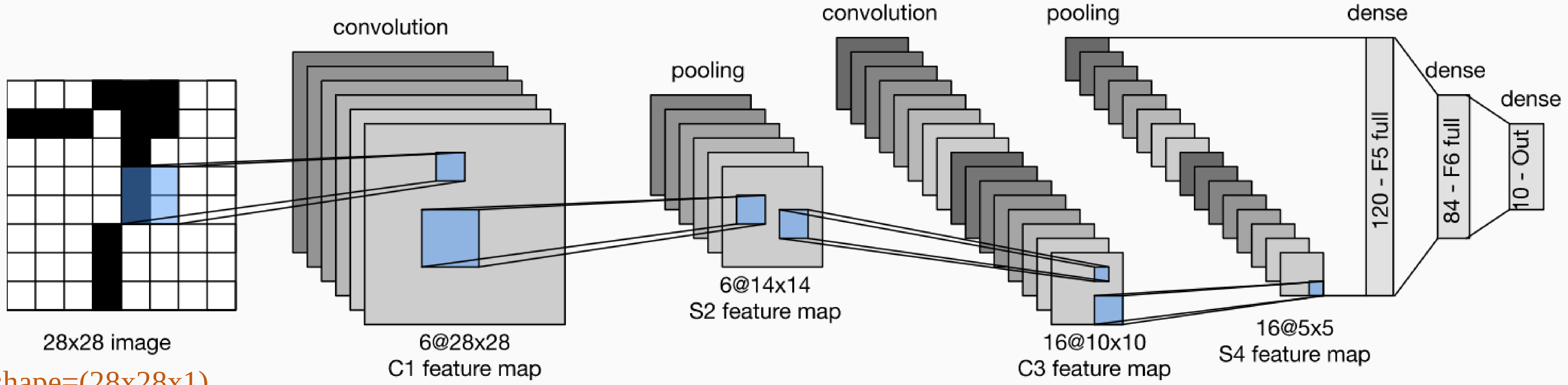


Padding

❖ LeNet (1990s)

2x2 max pooling

2x2 max pooling



shape=(28x28x1)

6 Convolutions
kernel=(3x3x1), pad='same'

16 Convolutions
kernel=(5x5x6)

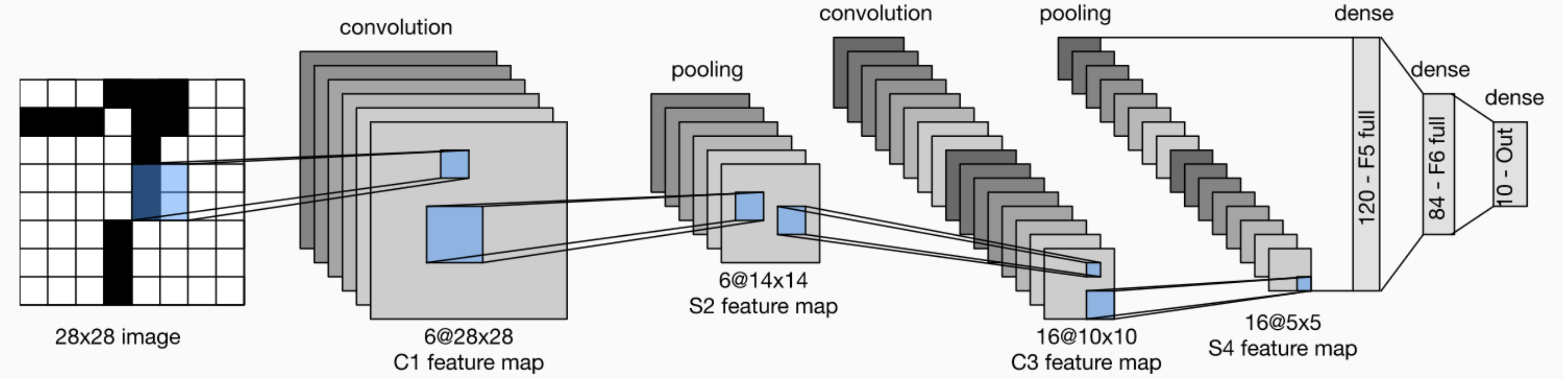
6 Convolutions
kernel=(5x5x1), pad='same'

https://d2l.ai/chapter_convolutional-neural-networks/lenet.html

Padding

❖ **LeNet (1990s)**

https://d2l.ai/chapter_convolutional-neural-networks/lenet.html




```python
# model architecture
model = tf.keras.Sequential()
# input shape (28,28,1)
model.add(tf.keras.Input(shape=(28, 28, 1)))

# convolution 1 and max pooling 1
model.add(tf.keras.layers.Conv2D(6, (5,5), padding='same', activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

# convolution 2 and max pooling 2
model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=5, activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))

# Flatten
model.add(tf.keras.layers.Flatten())

# fully connected
model.add(tf.keras.layers.Dense(120, activation='relu'))
model.add(tf.keras.layers.Dense(84, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

Outline

- From MLP to CNN
- Feature Map Down-sampling
- Padding
- 1x1 Convolution
- Image classification: Cifar-10 data

1x1 Convolution

❖ Why 1x1 Convolution

❖ Flexible input size

Yann LeCun
April 7, 2015 ·

In Convolutional Nets, there is no such thing as "fully-connected layers". There are only convolution layers with 1x1 convolution kernels and a full connection table.

It's a too-rarely-understood fact that ConvNets don't need to have a fixed-size input. You can train them on inputs that happen to produce a single output vector (with no spatial extent), and then apply them to larger images. Instead of a single output vector, you then get a spatial map of output vectors. Each vector sees input windows at different locations on the input.

In that scenario, the "fully connected layers" really act as 1x1 convolutions.

Yann LeCun

Yann LeCun in 2018

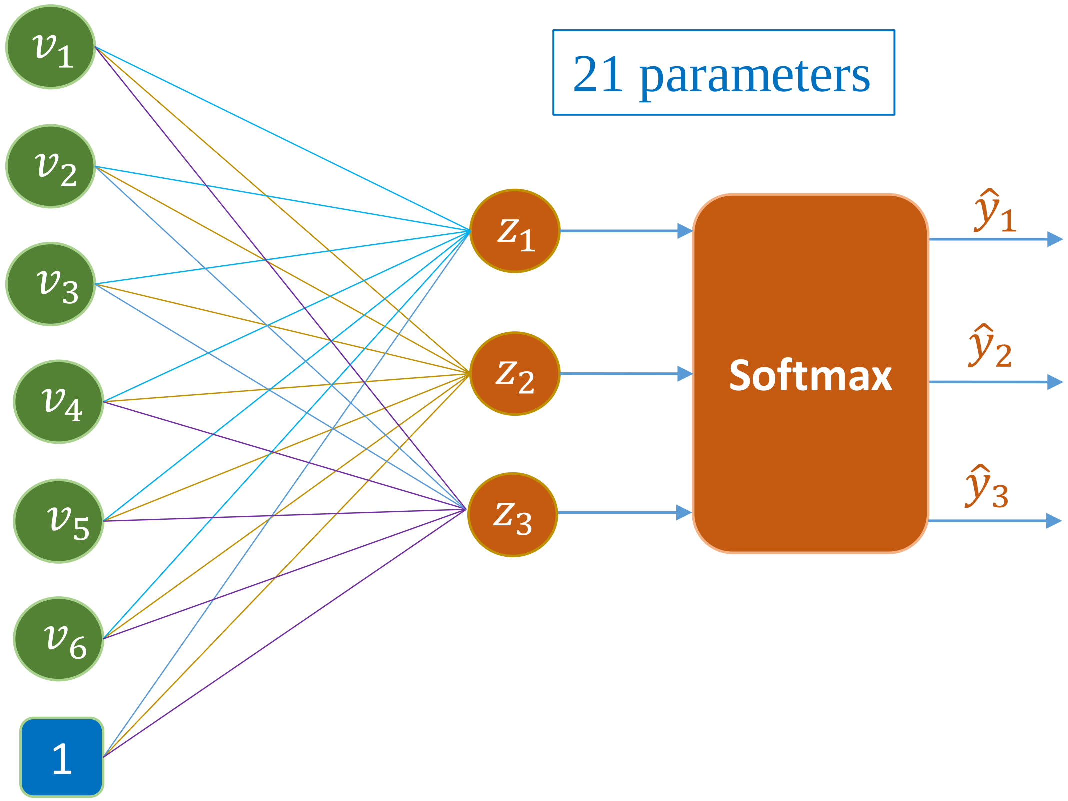
Born	July 8, 1960 (age 60) Soisy-sous-Montmorency, France
Alma mater	ESIEE Paris (MSc) Pierre and Marie Curie University (PhD)
Known for	Deep learning
Awards	Turing Award (2018) AAAI Fellow (2019) Legion of Honour (2020)
Scientific career	
Institutions	Bell Labs (1988-1996) New York University Facebook
Thesis	*Modèles connexionnistes de l'apprentissage (connectionist learning models)* (1987a)
Doctoral advisor	Maurice Milgram
Website	yann.lecun.com

1x1 Convolution

❖ Comparison

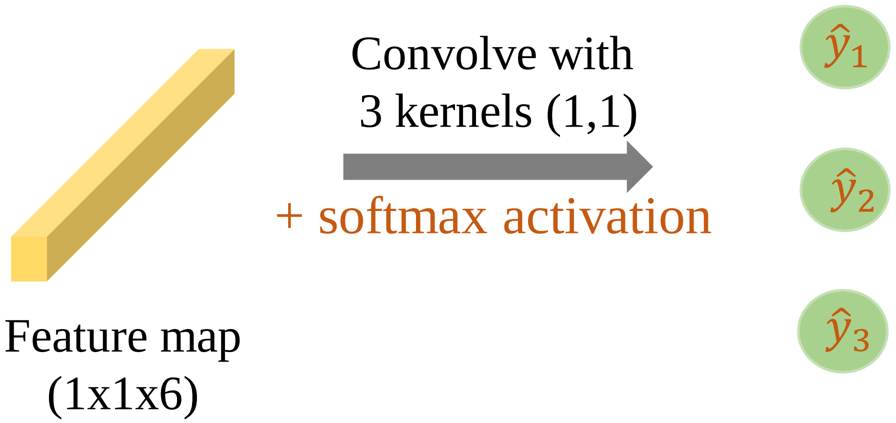
Fully connected layer

21 parameters



CNN

21 parameters



1x1 Convolution

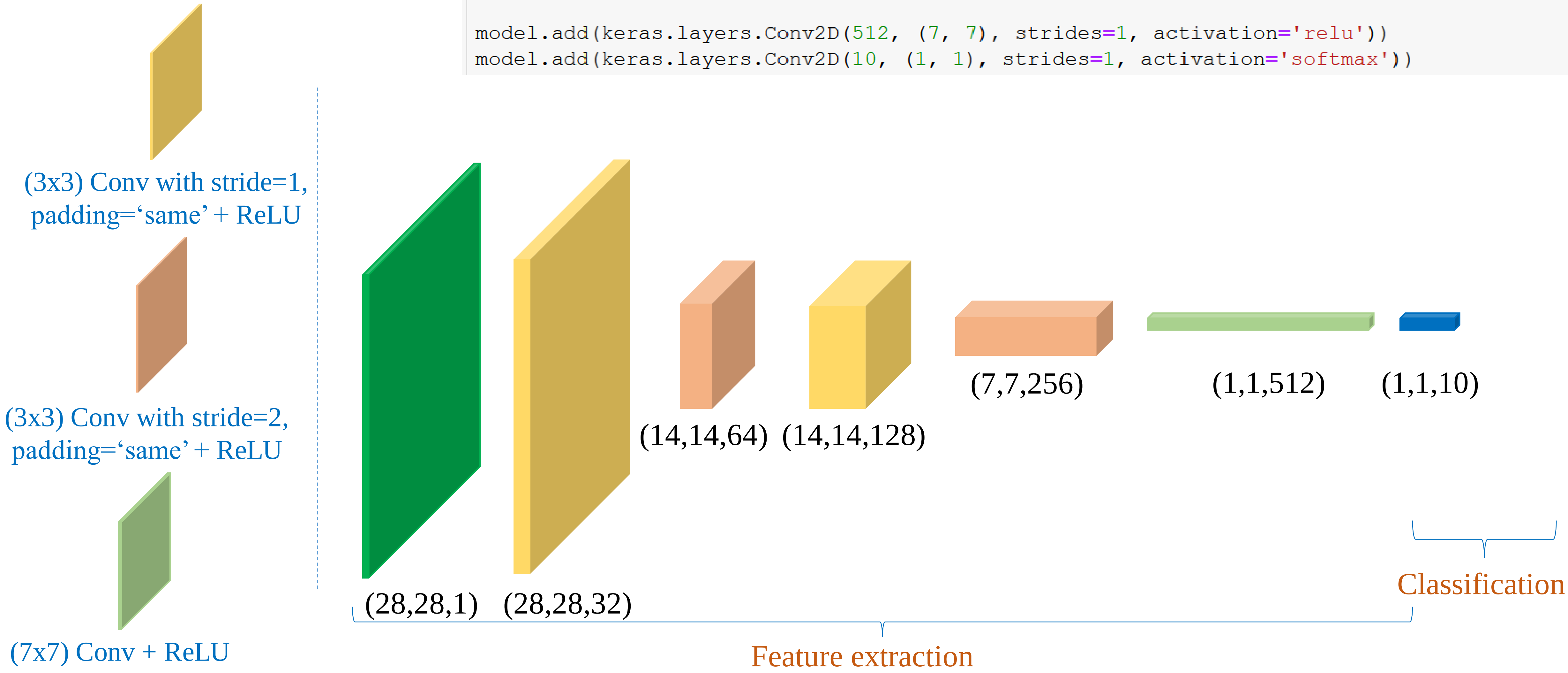
Replace FC by 1x1 Conv

```python
# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(28, 28, 1)))

model.add(keras.layers.Conv2D(32, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(64, (3, 3), strides=2, padding='same', activation='relu'))

model.add(keras.layers.Conv2D(128, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(256, (3, 3), strides=2, padding='same', activation='relu'))

model.add(keras.layers.Conv2D(512, (7, 7), strides=1, activation='relu'))
model.add(keras.layers.Conv2D(10, (1, 1), strides=1, activation='softmax'))
```



1x1 Convolution

Dynamic input sizes

```
# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(None, None, 1)))

model.add(keras.layers.Conv2D(32, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(64, (3, 3), strides=2, padding='same', activation='relu'))

model.add(keras.layers.Conv2D(128, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(256, (3, 3), strides=2, padding='same', activation='relu'))

model.add(keras.layers.Conv2D(512, (7, 7), strides=1, activation='relu'))
model.add(keras.layers.Conv2D(10, (1, 1), strides=1, activation='softmax'))
```

```
shape = (32,28,28,1)
data = tf.random.normal(shape)
print(data.shape)

output = model.predict(data)
print(output.shape)
```

```
(32, 28, 28, 1)
(32, 1, 1, 10)
```

```
shape = (32,64,64,1)
data = tf.random.normal(shape)
print(data.shape)

output = model.predict(data)
print(output.shape)
```

```
(32, 64, 64, 1)
(32, 10, 10, 10)
```

Shape=(batch size, height, width, channel)

1x1 Convolution

Dynamic input sizes

```
# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(None, None, 1)))

model.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
model.add(keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D(2))

model.add(keras.layers.Conv2D(128, (3, 3), activation='relu'))
model.add(keras.layers.Conv2D(256, (3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D(2))

model.add(keras.layers.Conv2D(512, (3, 3), activation='relu'))

# flatten
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(10, activation='softmax'))
```

```
C:\installed\lib\site-packages\tensorflow_core\python\keras\layers\core.py in build(self, input_shape)
   1013     input_shape = tensor_shape.TensorShape(input_shape)
   1014     if tensor_shape.dimension_value(input_shape[-1]) is None:
-> 1015       raise ValueError('The last dimension of the inputs to `Dense` '
   1016                        'should be defined. Found `None`.')
   1017     last_dim = tensor_shape.dimension_value(input_shape[-1])

ValueError: The last dimension of the inputs to `Dense` should be defined. Found `None`.
```

Outline

- From MLP to CNN
- Feature Map Down-sampling
- Padding
- 1x1 Convolution
- Image classification: Cifar-10 data

Image Classification

Cifar-10 dataset

Color images

Resolution=32x32

Training set: 50000 samples

Testing set: 10000 samples

Cifar-10 Image Classification

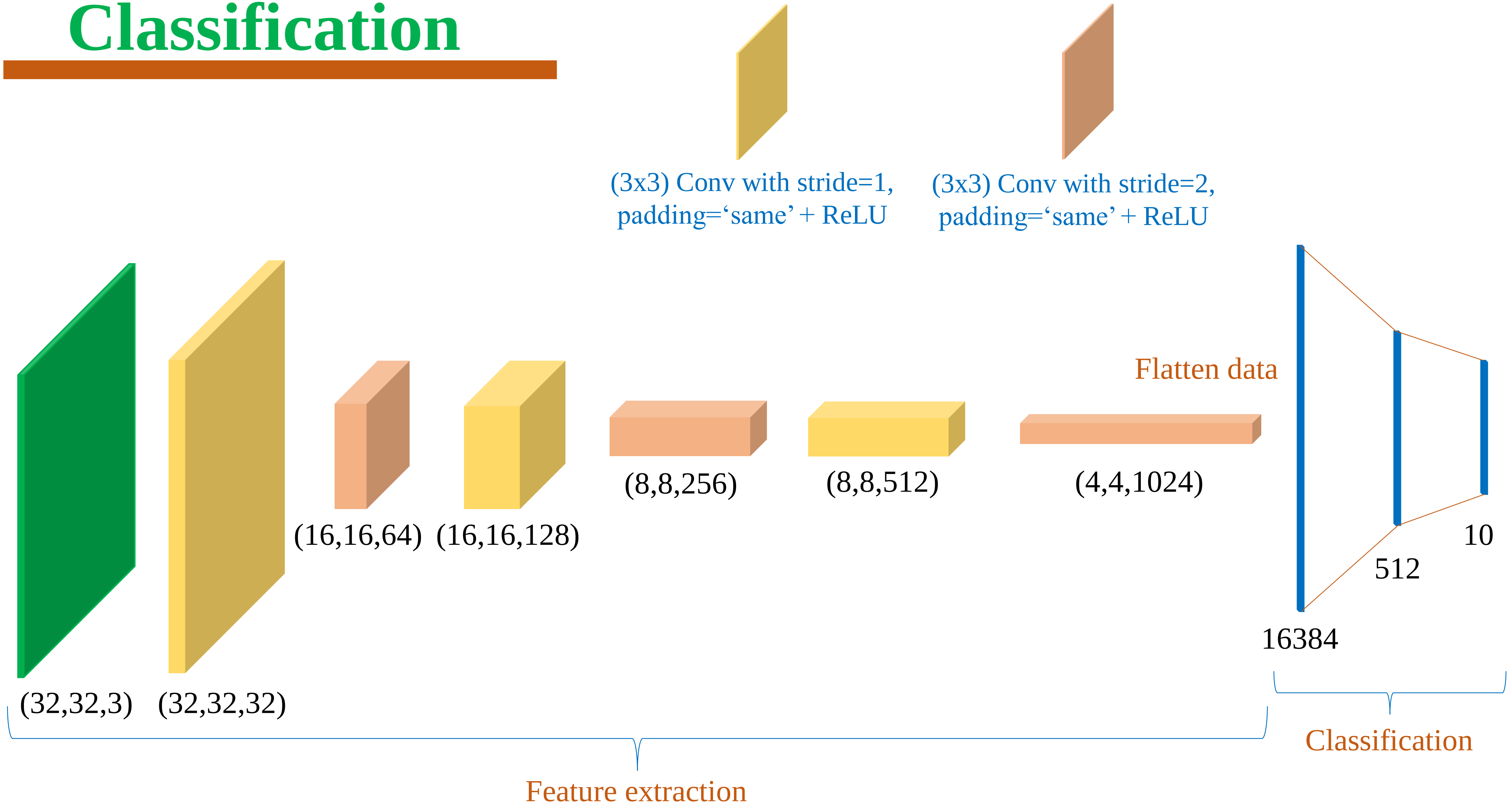


Image Classification

Cifar-10

```python
import tensorflow as tf

# data preparation
cifar10 = tf.keras.datasets.cifar10
(x_train, y_train),(x_test, y_test) = cifar10.load_data()

# normalize
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```python
# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(height, width, 3)))

model.add(keras.layers.Conv2D(32, (3, 3), strides=1, padding='same', activation = 'relu'))
model.add(keras.layers.Conv2D(64, (3, 3), strides=2, padding='same', activation='relu'))

model.add(keras.layers.Conv2D(128, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(256, (3, 3), strides=2, padding='same', activation='relu'))

model.add(keras.layers.Conv2D(512, (3, 3), strides=1, padding='same', activation='relu'))
model.add(keras.layers.Conv2D(1024, (3, 3), strides=2, padding='same', activation='relu'))

# flatten
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(512, activation='relu'))
model.add(keras.layers.Dense(10, activation='softmax'))
model.summary()
```

Image Classification

❖ Demo

```
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] ::
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>> for epoch in range(n_epochs):
...     sum_of_losses = 0
...     gradients = np.zeros((2,1))
...
...     for index in range(4):
...         xi = X_b[index:index+1]
...         yi = y[index:index+1]
```

Reading and Exercises

❖ Exercises

1) Use LeNet for the fashion-MNIST and Cifar-10 data sets

2) What are the advantages of using 1x1 Conv instead of FC

❖ Reading

https://cs231n.github.io/convolutional-networks/

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53


Thank you!