AI VIETNAM
AI Insight Course

Convolutional Neural Network

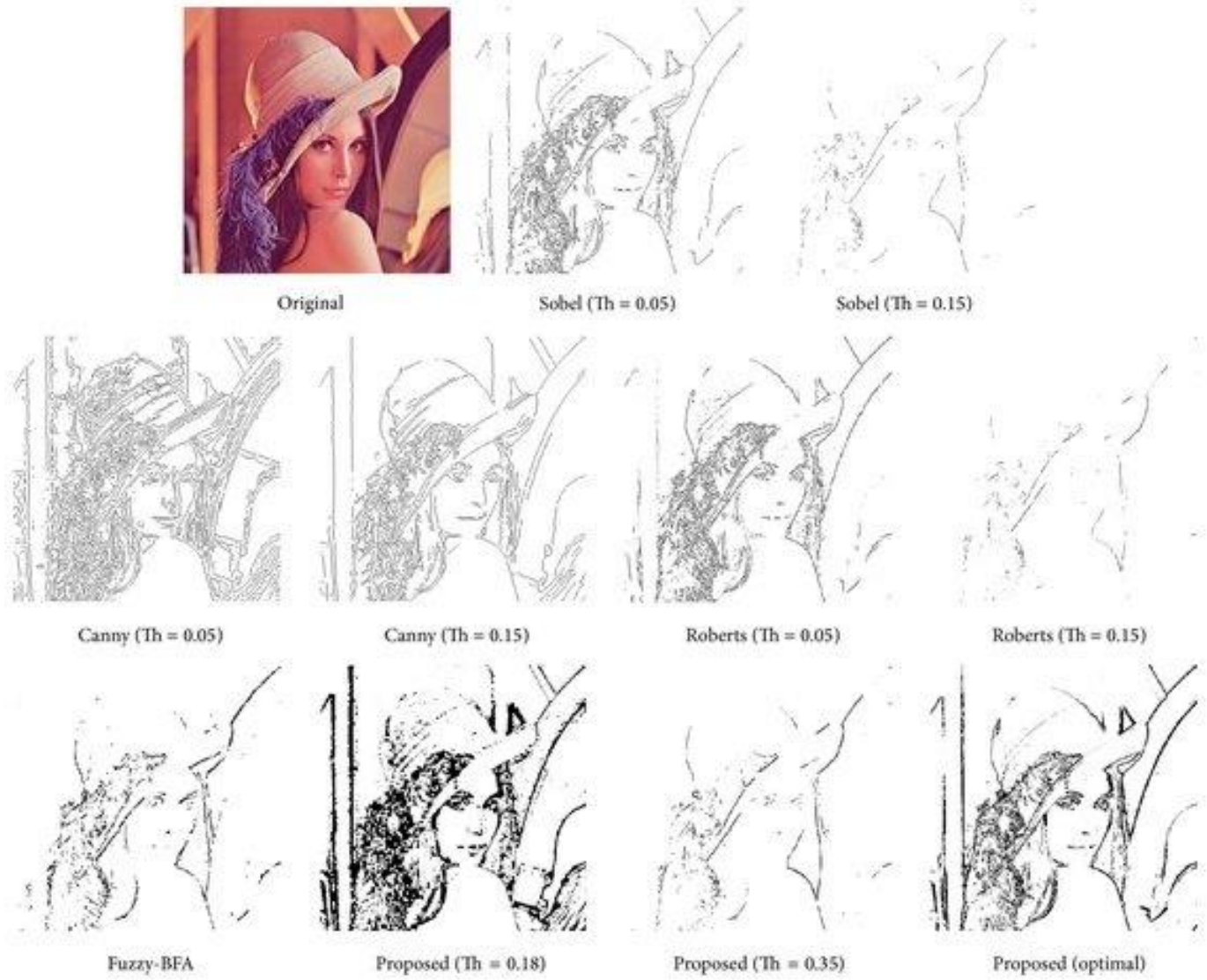
(Session 3)

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2020

Classic Feature Extractors

- **Edge detection**
- **Gradient**



Edge Detection via Edge-Strength Estimation Using Fuzzy Reasoning and Optimal Threshold Selection Using Particle Swarm Optimization

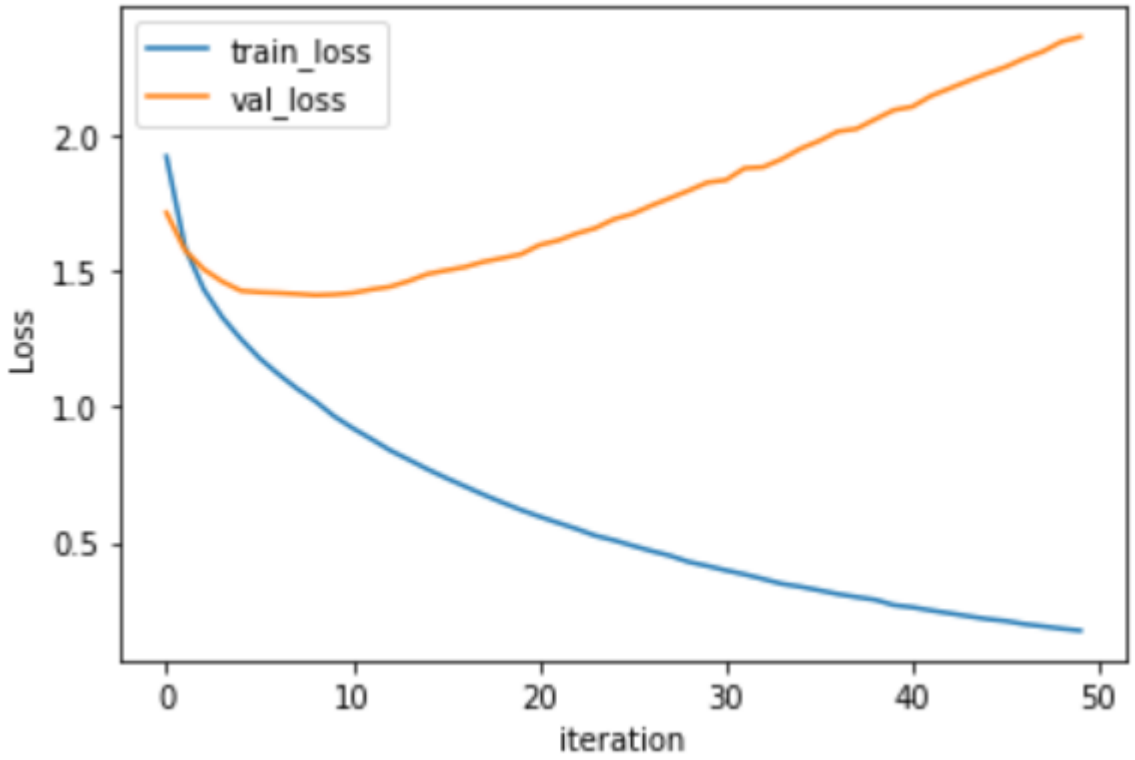
Case Study

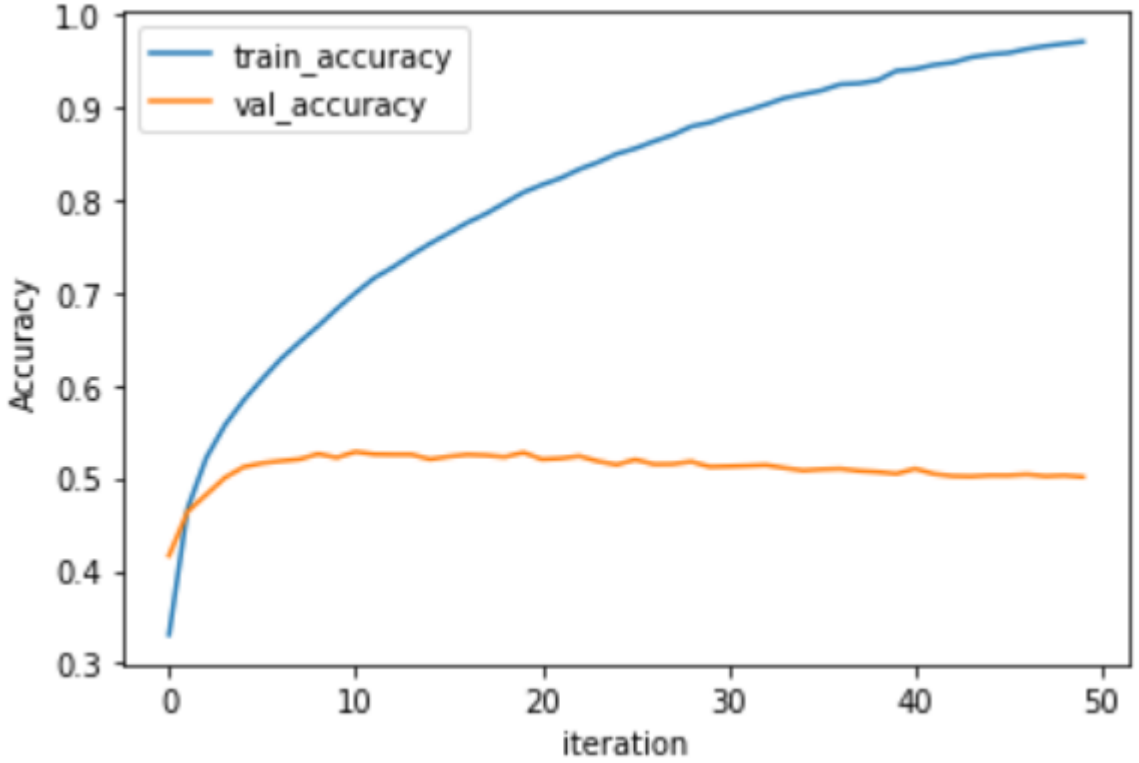
- Classic feature extractors vs. CNN filters

```python
import tensorflow as tf
from tensorflow import keras

def construct_model():
    inp = keras.layers.Input(shape=[32,32,3])

    # get edges
    edges = tf.image.sobel_edges(inp)
    edges = tf.keras.layers.Reshape((32,32,6))(edges)

    dx, dy = tf.image.image_gradients(inp)
    x = tf.keras.layers.Concatenate(axis=3)([dx, dy, edges])

    x = keras.layers.MaxPooling2D(2)(x)
    x = keras.layers.Flatten()(x)
    x = keras.layers.Dense(128, activation='relu')(x)
    x = keras.layers.Dense(10, activation='softmax')(x)

    return tf.keras.Model(inputs=[inp], outputs=x)

cifar10 = keras.datasets.cifar10
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

train_images = train_images / 255.0
test_images  = test_images / 255.0

train_images = tf.reshape(train_images, (50000, 32, 32, 3))
test_images  = tf.reshape(test_images, (10000, 32, 32, 3))

# model
model = construct_model()
model.summary()

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
history_data = model.fit(train_images, train_labels, batch_size=1024,
                         validation_data=(test_images, test_labels), epochs=50)
```

Case Study

❖ Classic feature extractors vs. CNN filters





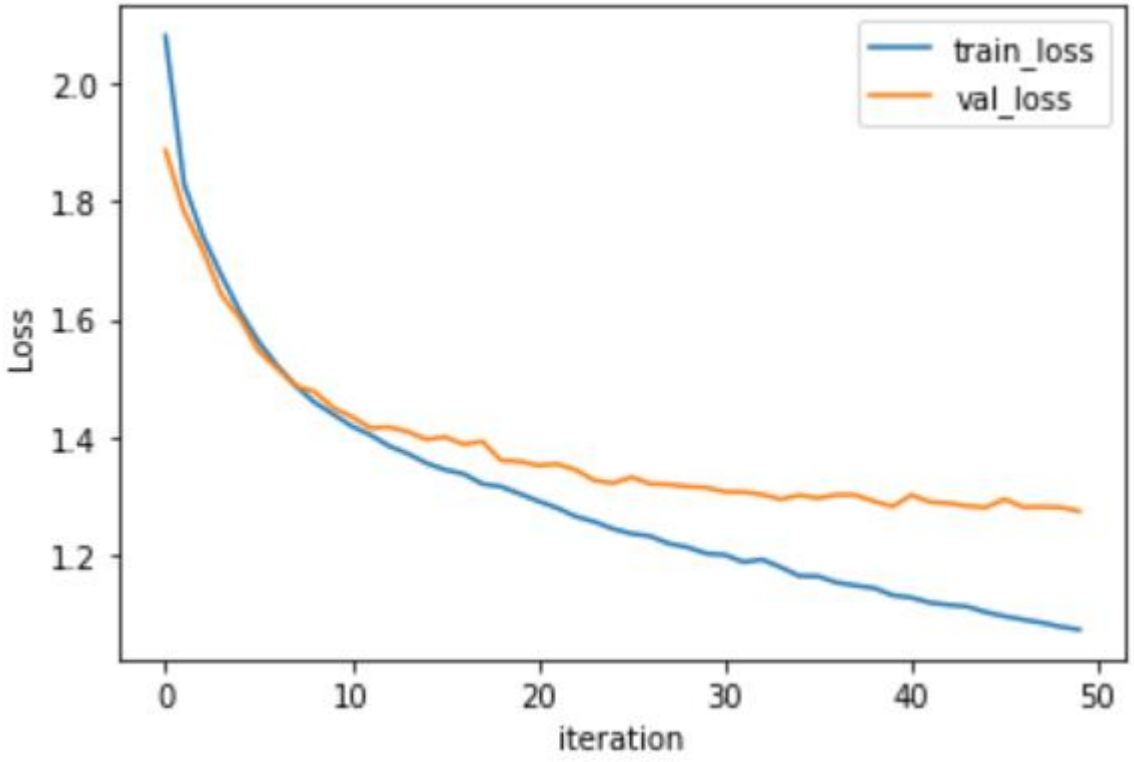
Case Study

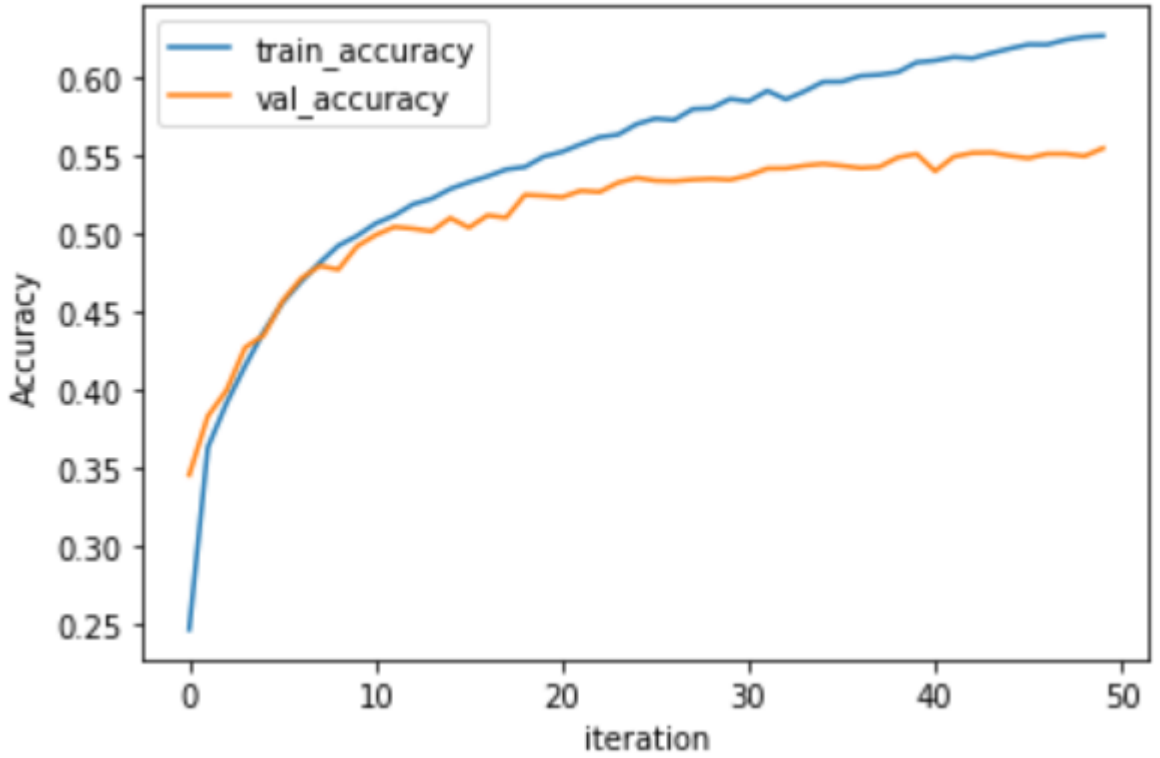
- **Classic feature extractors vs. CNN filters**

```python
import tensorflow as tf
from tensorflow import keras

cifar10 = keras.datasets.cifar10
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

train_images = train_images / 255.0
test_images  = test_images / 255.0

train_images = tf.reshape(train_images, (50000, 32, 32, 3))
test_images  = tf.reshape(test_images, (10000, 32, 32, 3))

# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(32, 32, 3)))
model.add(keras.layers.Conv2D(4, (3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D(2))

# flatten
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(128, activation='relu'))
model.add(keras.layers.Dense(10, activation='softmax'))

model.summary()

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
history_data = model.fit(train_images, train_labels, batch_size=1024,
                         validation_data=(test_images, test_labels), epochs=50)
```

Case Study

❖ Classic feature extractors vs. CNN filters





Down-sample Feature Map

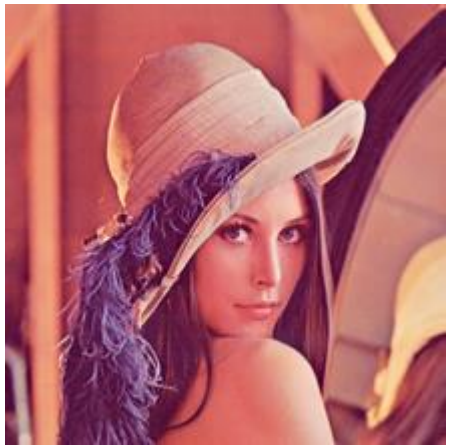
Max pooling: Features are preserved

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 $\left(\begin{array}{|c|c|c|c|}\hline v_1 & v_2 & v_3 & v_4 \\ \hline v_5 & v_6 & v_7 & v_8 \\ \hline v_9 & v_{10} & v_{11} & v_{12} \\ \hline v_{13} & v_{14} & v_{15} & v_{16} \\ \hline\end{array}\right) = \begin{array}{|c|c|}\hline m_1 & m_2 \\ \hline m_3 & m_4 \\ \hline\end{array}$

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



Feature map (220x220)

max pooling (2x2) →

Feature map (110x110)

max pooling (2x2) →

Feature map (55x55)

Down-sample Feature Map

Max pooling: Features are preserved

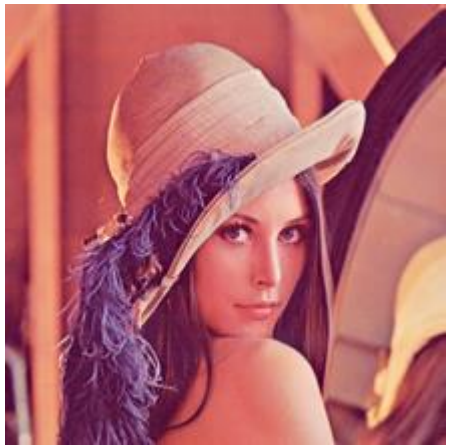
v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

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 |

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

keras.layers.MaxPooling2D(pool_size=2)



Feature map (220x220)

max pooling (2x2) →

Feature map (110x110)

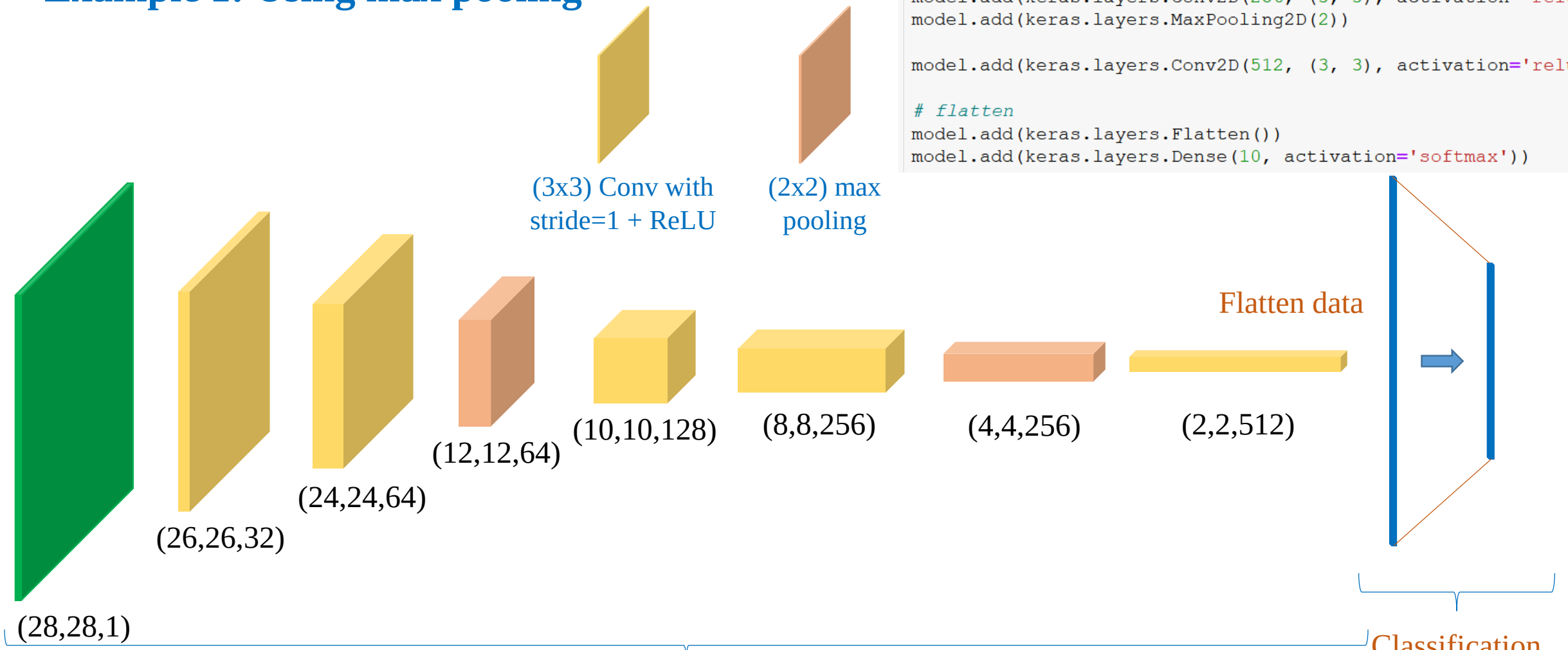
max pooling (2x2) →

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

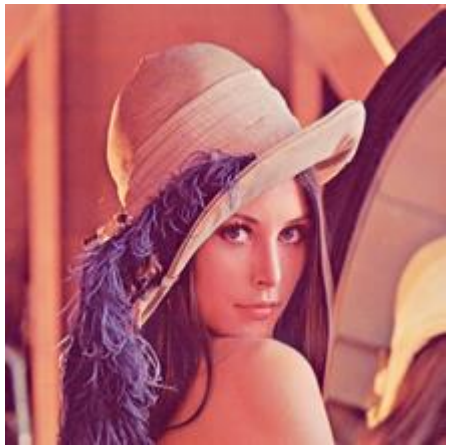
Average pooling: Features are preserved

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

average 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 |

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



Feature map (220x220)

Average Pooling (2x2) →

Feature map (110x110)

Average Pooling (2x2) →

Feature map (55x55)

Down-sample Feature Map

Average pooling: Features are preserved

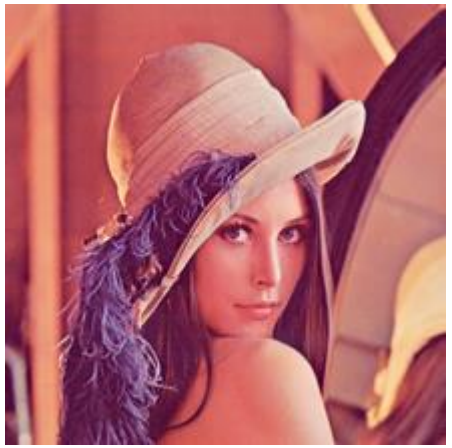
v_1	v_2	v_3	v_4
v_5	v_6	v_7	v_8
v_9	v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}	v_{16}

Data

average 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 |

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

keras.layers. AveragePooling2D (pool_size=2)



Feature map (220x220)

Average Pooling (2x2) →

Feature map (110x110)

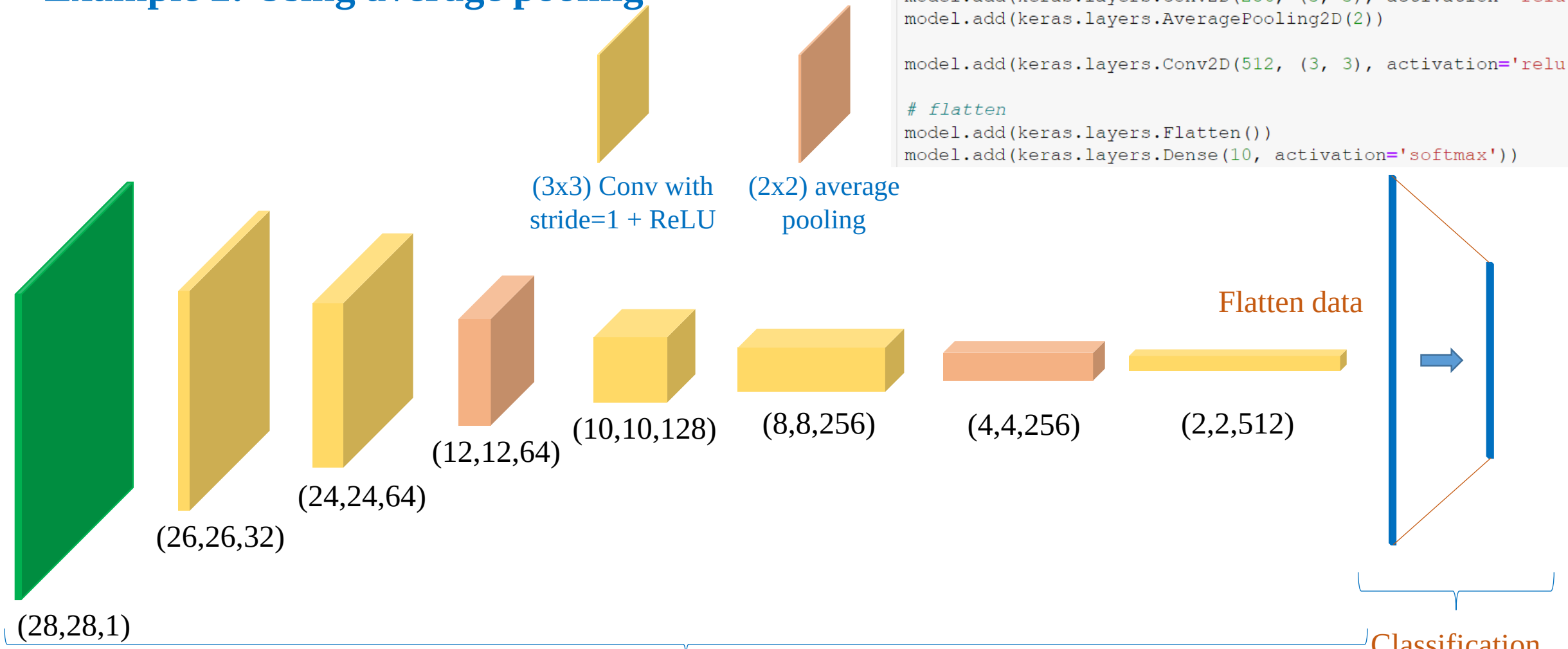
Average Pooling (2x2) →

Feature map (55x55)

Down-sample Feature Map

Example 2: Using average 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.AveragePooling2D(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.AveragePooling2D(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

Max pooling vs. Average pooling

Feature map (220x220)

max pooling (2x2) →

max pooling (2x2) →

Feature map (55x55)

Feature map (220x220)

Average Pooling (2x2) →

Feature map (110x110)

Average Pooling (2x2) →

Feature map (55x55)

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

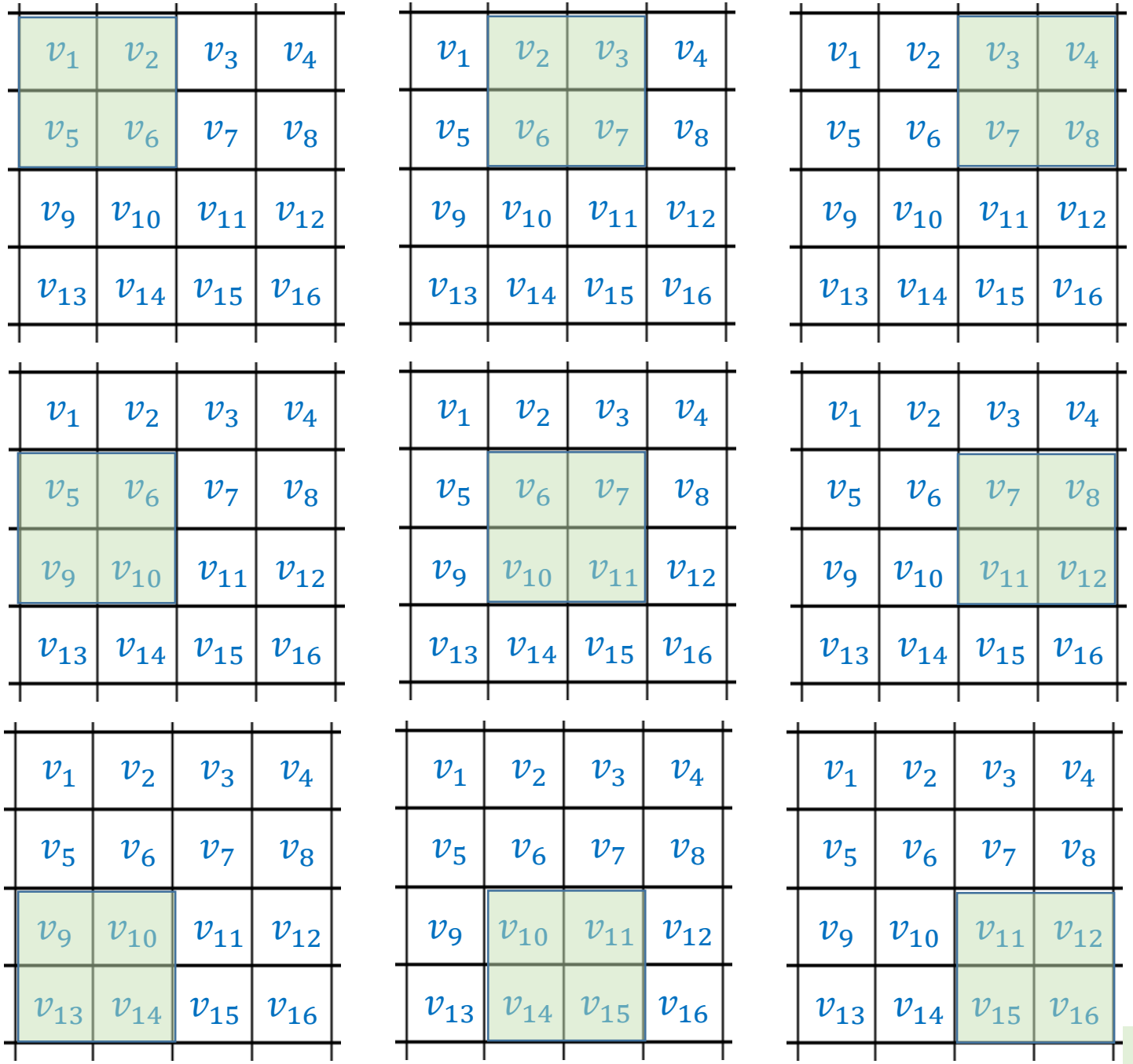
b

Kernel of parameters

Convolve **D** with stride=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**

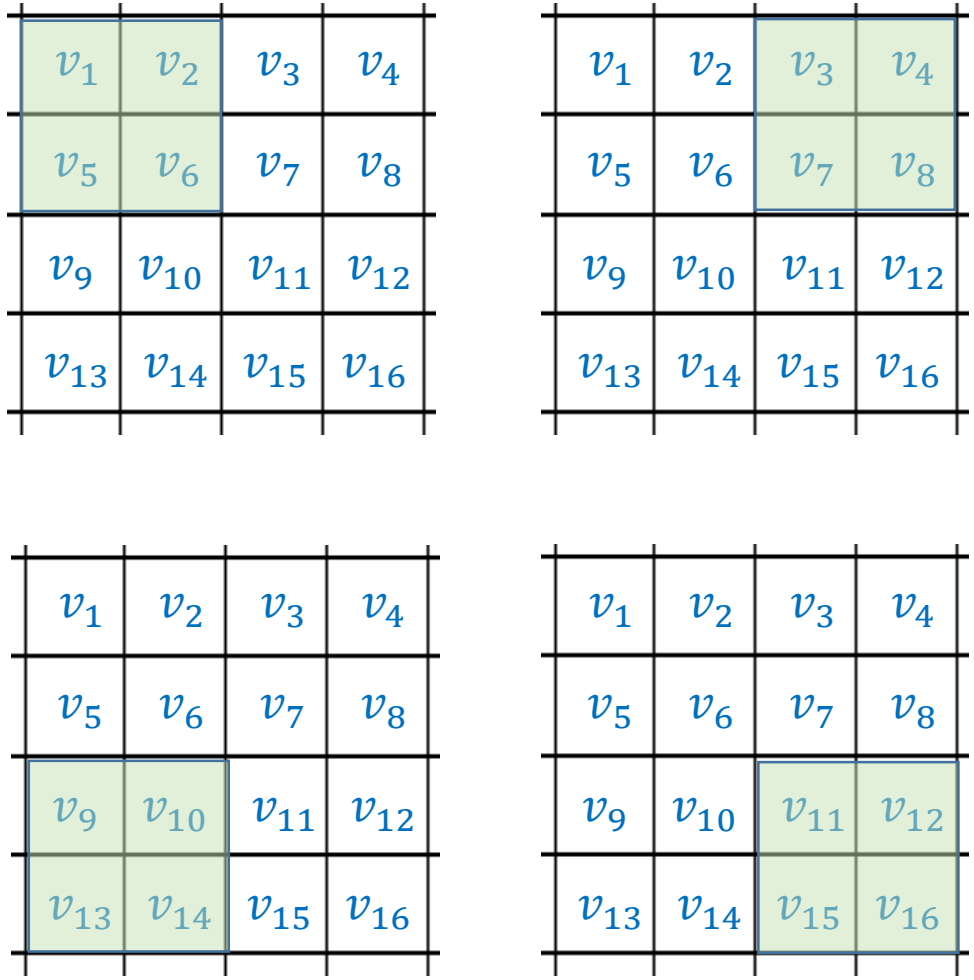
w_1	w_2	b
w_3	w_4	

Kernel of parameters

Convolve **D** with stride=2

m_1	m_2
m_3	m_4

Output



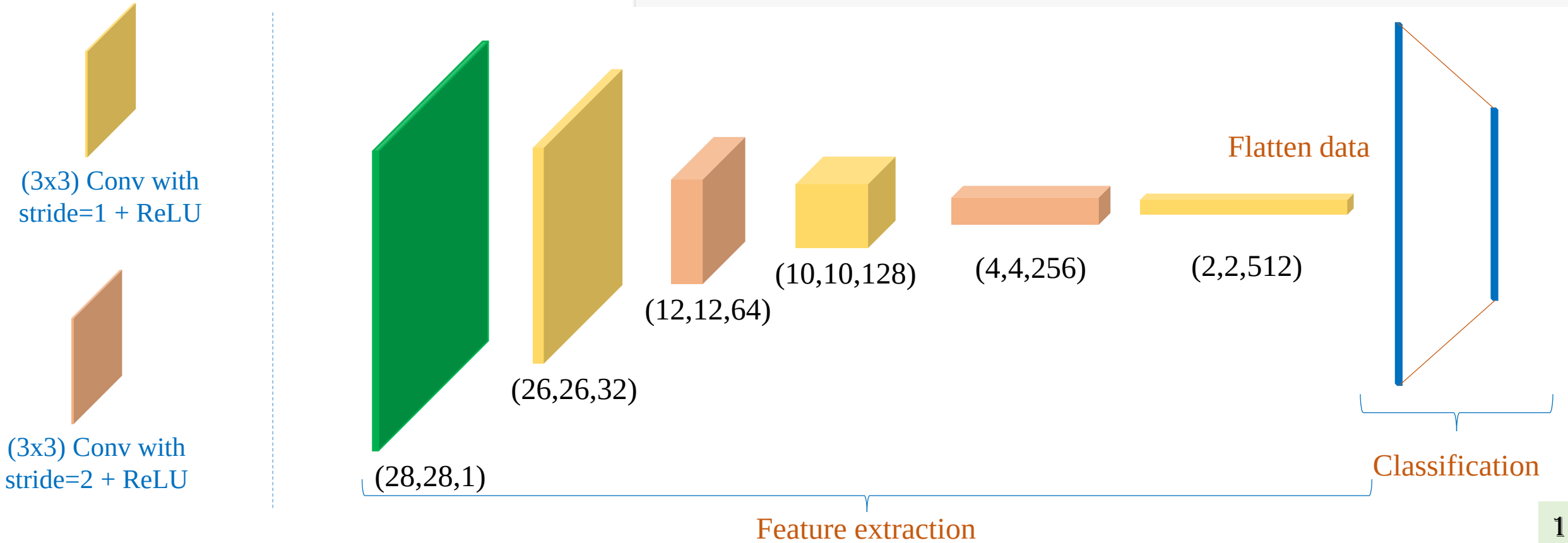
keras.layers.Conv2D(32, (3, 3), strides=1, activation='relu')

keras.layers.Conv2D(32, (3, 3), strides=2, activation='relu')

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'))
```



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)

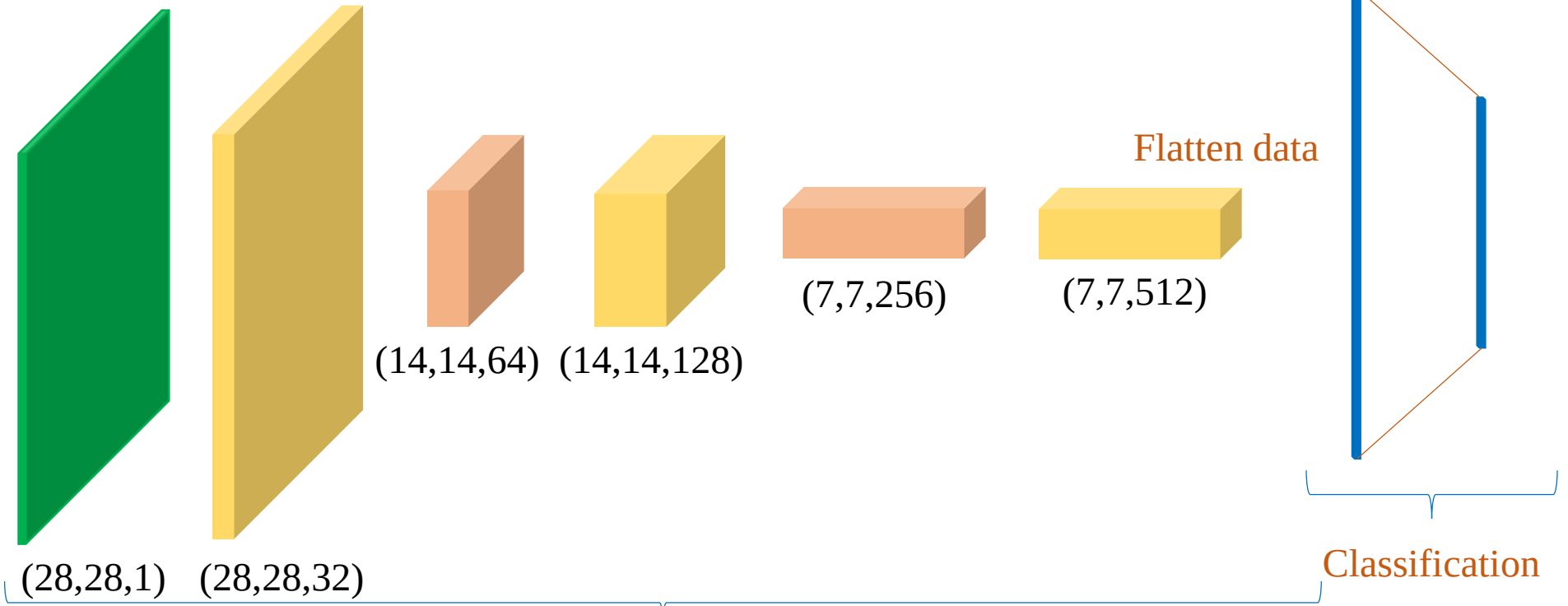
Padding

Example

```
# 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'))
```

(3x3) Conv with stride=1, padding='same' + ReLU

(3x3) Conv with stride=2, padding='same' + ReLU



Outline

- Common CNN Archtecture
- 1x1 Convolution
- Examples
- Backpropagation

Size of Feature Maps

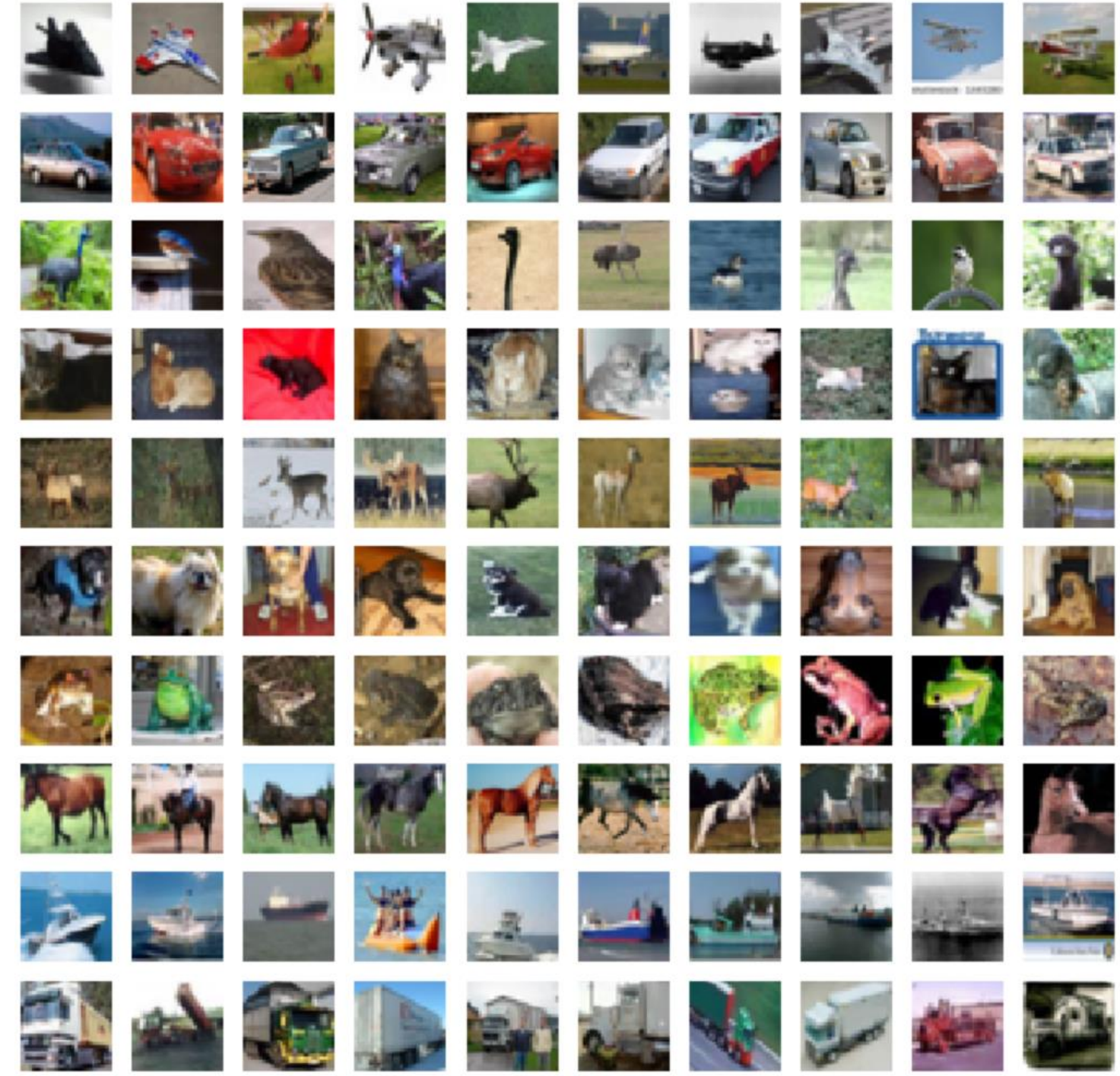
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Image Classification

Cifar-10 dataset

- Color images
- Resolution=32x32
- Training set: 50000 samples
- Testing set: 10000 samples



Cifar-10 Image Classification

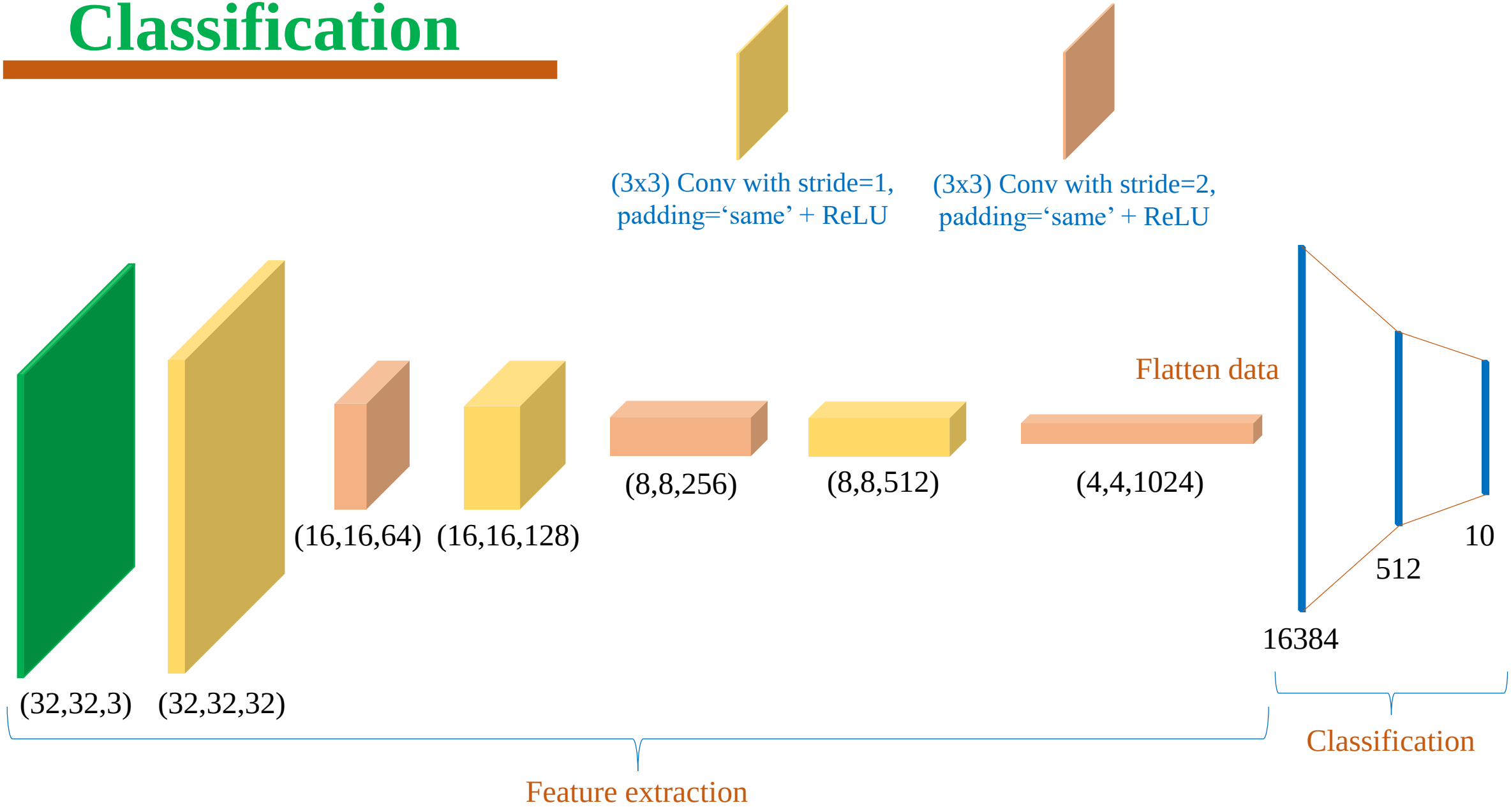


Image Classification

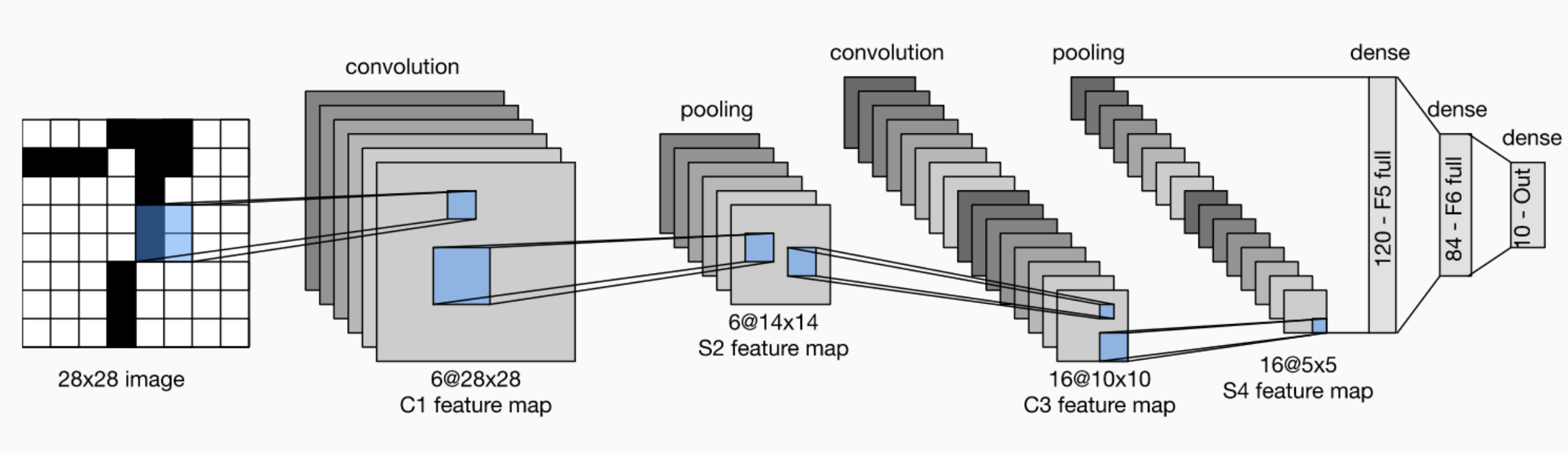
Cifar-10

```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()
```

LeNet Architecture

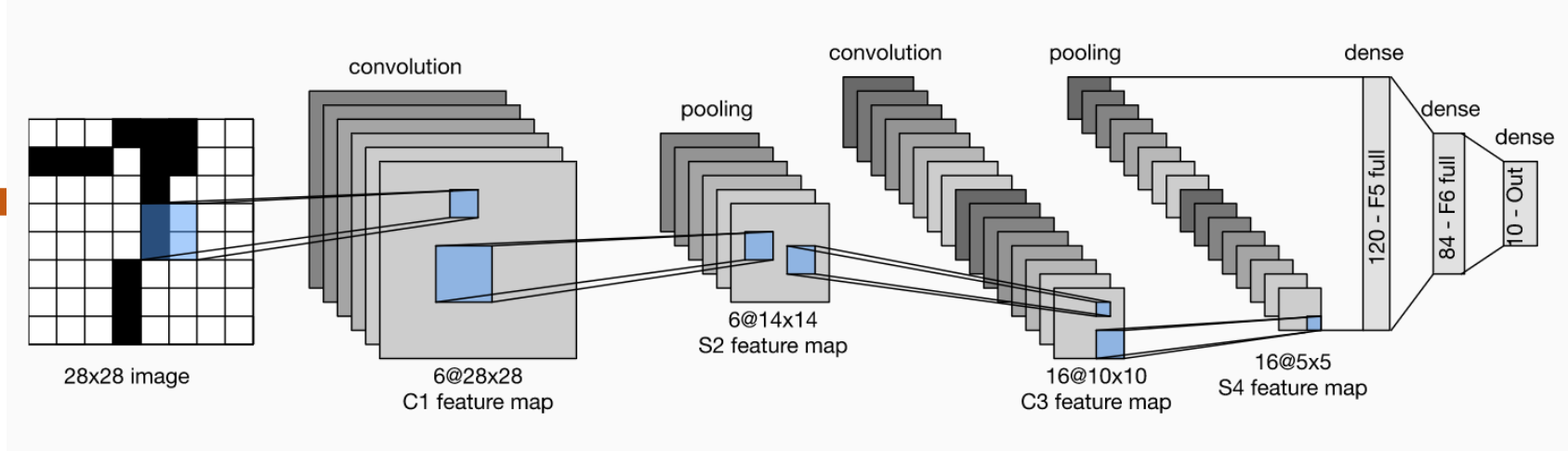
❖ Model Construction



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

LeNet Architecture

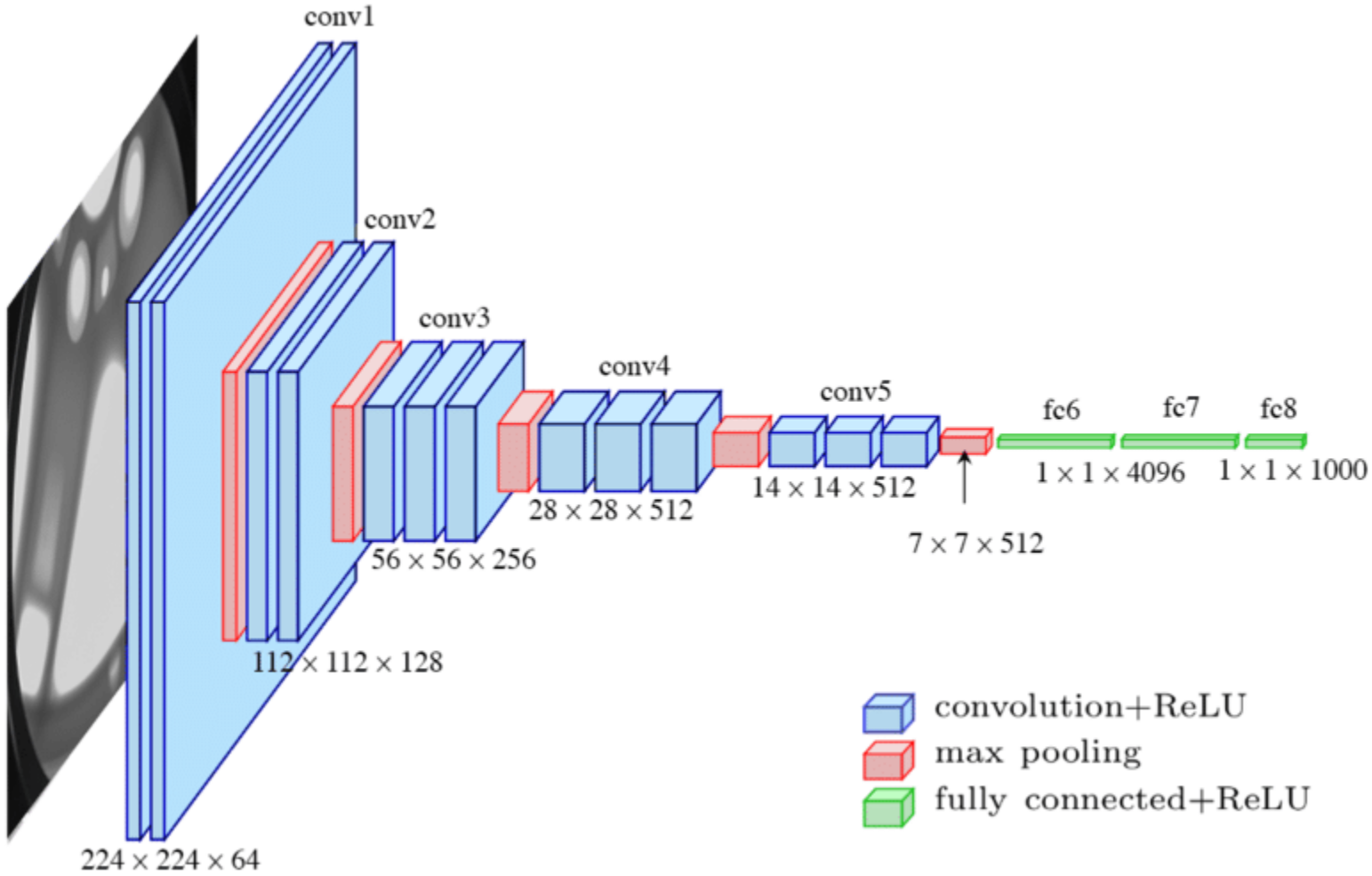
❖ **Model Construction**




```
# 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'))
```

VGG16 Architecture

❖ Model Construction



VGG16 Architecture

❖ **Model Construction**

```
def VGG_16():
    model = Sequential()
    model.add(Convolution2D(64, 3, padding='same', activation='relu',
                            input_shape=(224,224, 1)))
    model.add(Convolution2D(64, 3, padding='same', activation='relu'))
    model.add(MaxPooling2D((2,2), strides=(2,2)))

    model.add(Convolution2D(128, 3, padding='same', activation='relu'))
    model.add(Convolution2D(128, 3, padding='same', activation='relu'))
    model.add(MaxPooling2D((2,2), strides=(2,2)))


    model.add(Convolution2D(256, 3, padding='same', activation='relu'))
    model.add(Convolution2D(256, 3, padding='same', activation='relu'))
    model.add(Convolution2D(256, 3, padding='same', activation='relu'))
    model.add(MaxPooling2D((2,2), strides=(2,2)))

    model.add(Convolution2D(512, 3, padding='same', activation='relu'))
    model.add(Convolution2D(512, 3, padding='same', activation='relu'))
    model.add(Convolution2D(512, 3, padding='same', activation='relu'))
    model.add(MaxPooling2D((2,2), strides=(2,2)))

    model.add(Convolution2D(512, 3, padding='same', activation='relu'))
    model.add(Convolution2D(512, 3, padding='same', activation='relu'))
    model.add(Convolution2D(512, 3, padding='same', activation='relu'))
    model.add(MaxPooling2D((2,2), strides=(2,2)))

    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(1000, activation='softmax'))

    return model
```

Outline

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

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

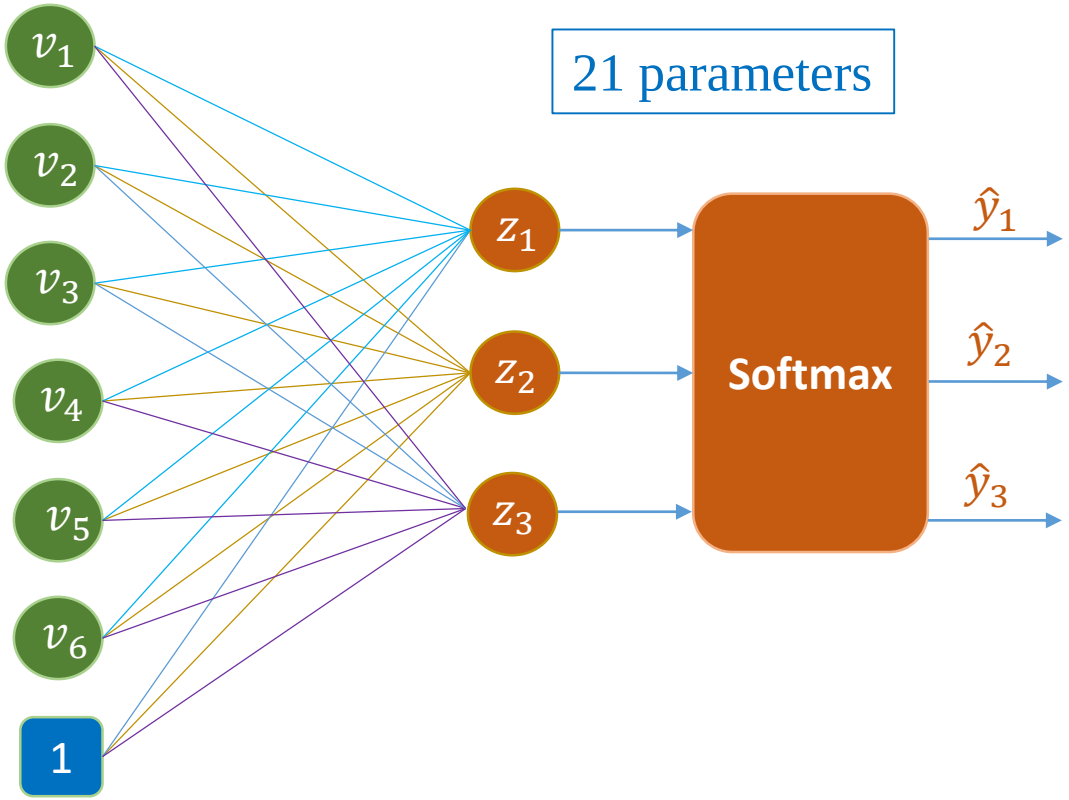
Born	July 8, 1960 (age 60) Soisy-sous-Montmorency, France
Alma mater	ESIEE Paris (MSc) Pierre and Marie Curie University (PhD)
Known for	Deep learning
Awards	Turing Award (2018) AAAI Fellow (2019) Legion of Honour (2020)
Scientific career	
Institutions	Bell Labs (1988-1996) New York University Facebook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

1x1 Convolution

❖ Comparison

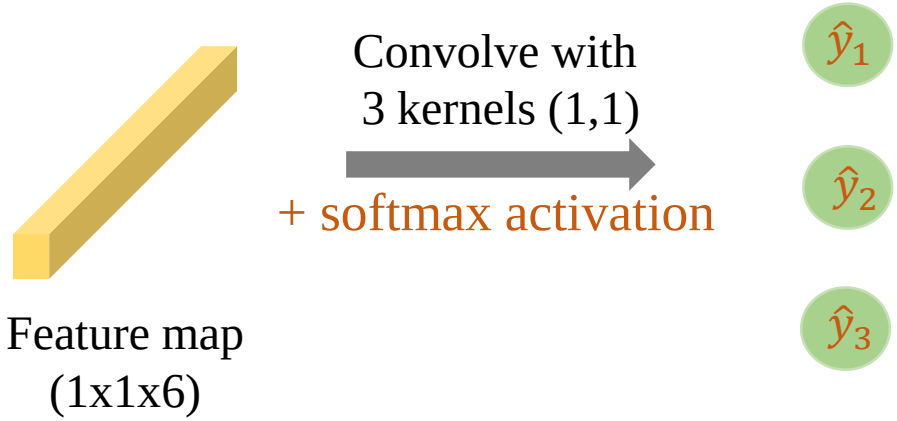
Fully connected layer

21 parameters



CNN

21 parameters

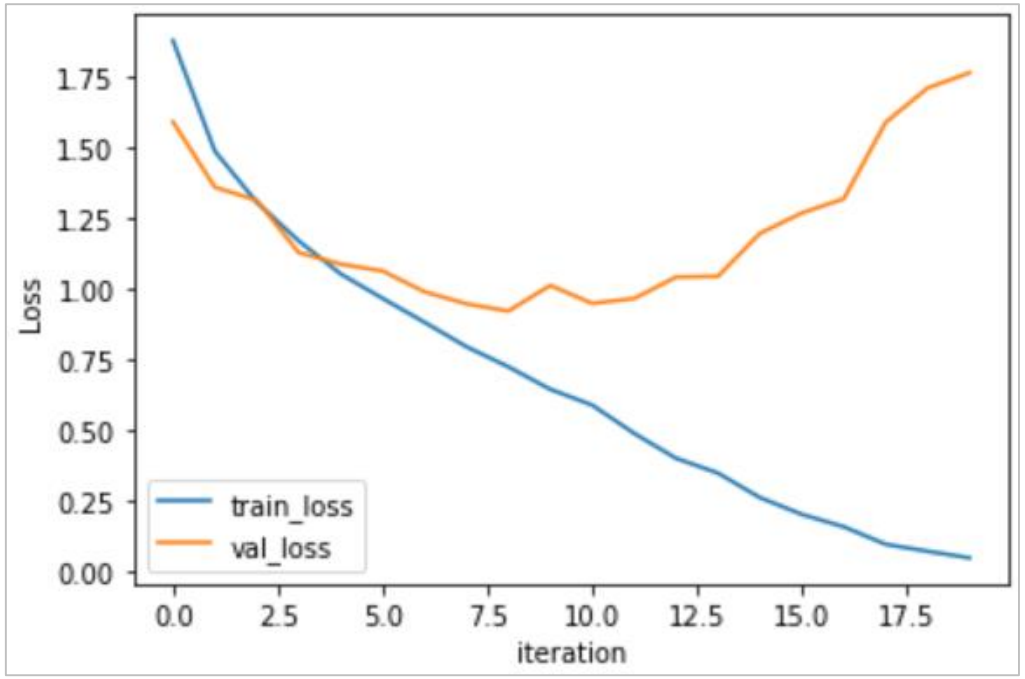


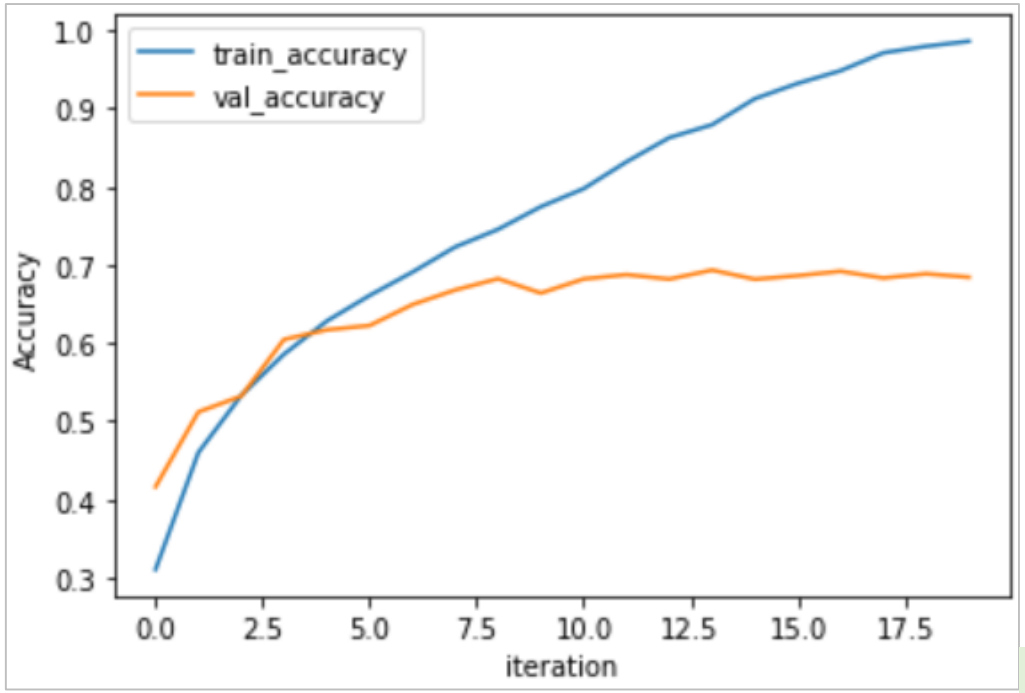
1x1 Convolution

```
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras

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

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

# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(32, 32, 3)))

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, (5, 5), strides=1, activation='relu'))

# flatten
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(128, activation='relu'))
model.add(keras.layers.Dense(10, activation='softmax'))
model.summary()

# training
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=1024,
                    validation_data=(x_test, y_test), epochs=20, verbose=1)
```

```
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 14, 14, 64)        18496
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 12, 12, 128)       73856
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 5, 5, 256)         295168
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 1, 1, 512)         3277312
_________________________________________________________________
flatten_2 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 3,732,682
Trainable params: 3,732,682
Non-trainable params: 0
_________________________________________________________________
```

```python
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras

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

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

# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(32, 32, 3)))

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, (5, 5), strides=1, activation='relu'))

# flatten
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(128, activation='relu'))
model.add(keras.layers.Dense(10, activation='softmax'))
model.summary()

# training
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=1024,
                    validation_data=(x_test, y_test), epochs=20, verbose=1)
```



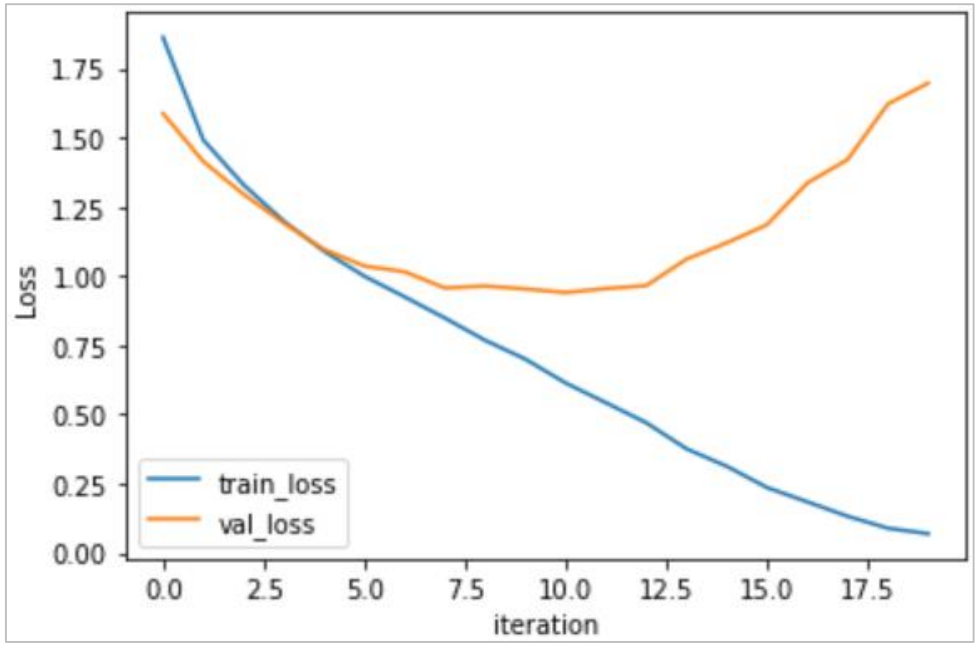


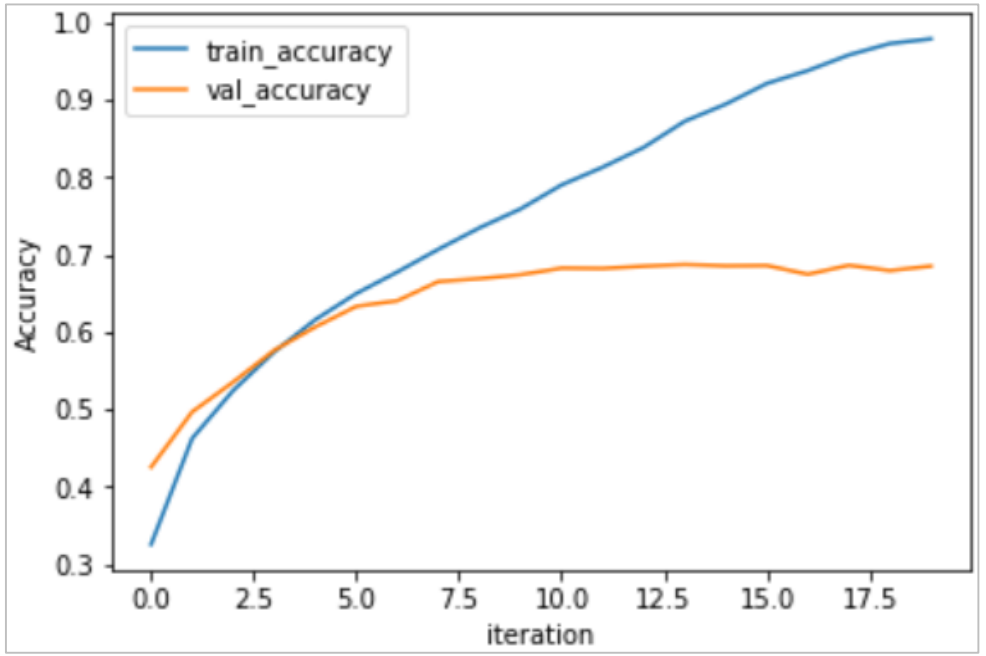
1x1 Convolution

```
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras

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

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

# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(32, 32, 3)))

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, (5, 5), strides=1, activation='relu'))

model.add(keras.layers.Conv2D(128, (1, 1), strides=1, activation='relu'))
model.add(keras.layers.Conv2D(10, (1, 1), activation='softmax'))
model.add(keras.layers.Reshape((10,)))
model.summary()

# training
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=1024,
                    validation_data=(x_test, y_test), epochs=20, verbose=1)
```

```
conv2d_22 (Conv2D)           (None, 14, 14, 64)        18496
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 12, 12, 128)       73856
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 5, 5, 256)         295168
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 1, 1, 512)         3277312
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 1, 1, 128)         65664
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 1, 1, 10)          1290
_________________________________________________________________
reshape (Reshape)            (None, 10)                0
=================================================================
Total params: 3,732,682
Trainable params: 3,732,682
Non-trainable params: 0
```

```python
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras

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

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

# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(32, 32, 3)))

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, (5, 5), strides=1, activation='relu'))

model.add(keras.layers.Conv2D(128, (1, 1), strides=1, activation='relu'))
model.add(keras.layers.Conv2D(10, (1, 1), activation='softmax'))
model.add(keras.layers.Reshape((10,)))
model.summary()

# training
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=1024,
                    validation_data=(x_test, y_test), epochs=20, verbose=1)
```





1x1 Convolution

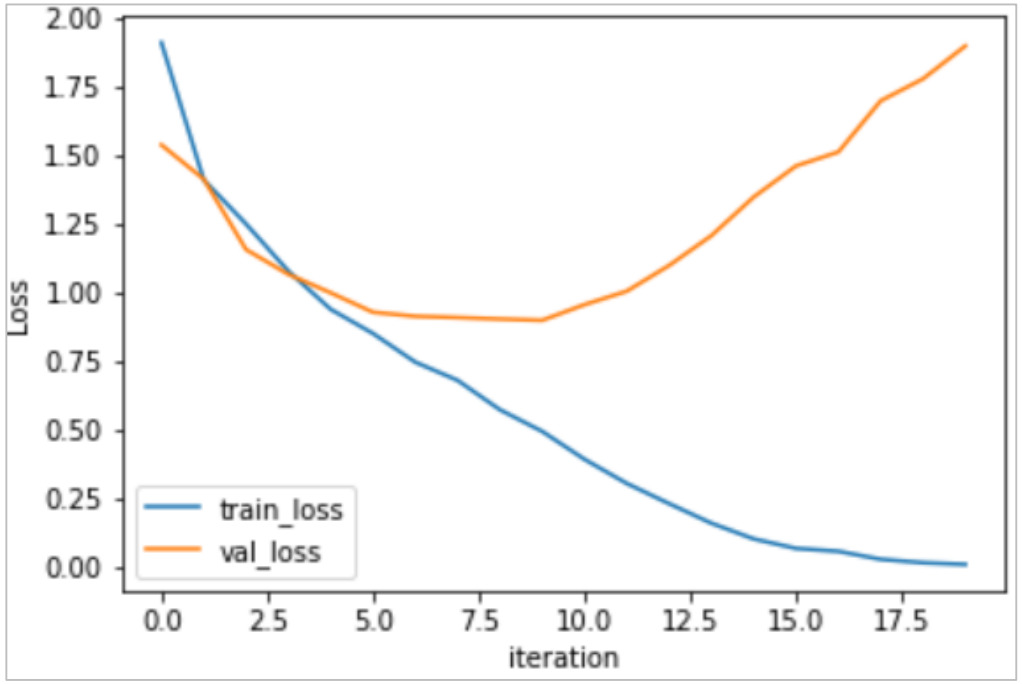
- Convert from FCL to 1x1 Conv
  - Another example

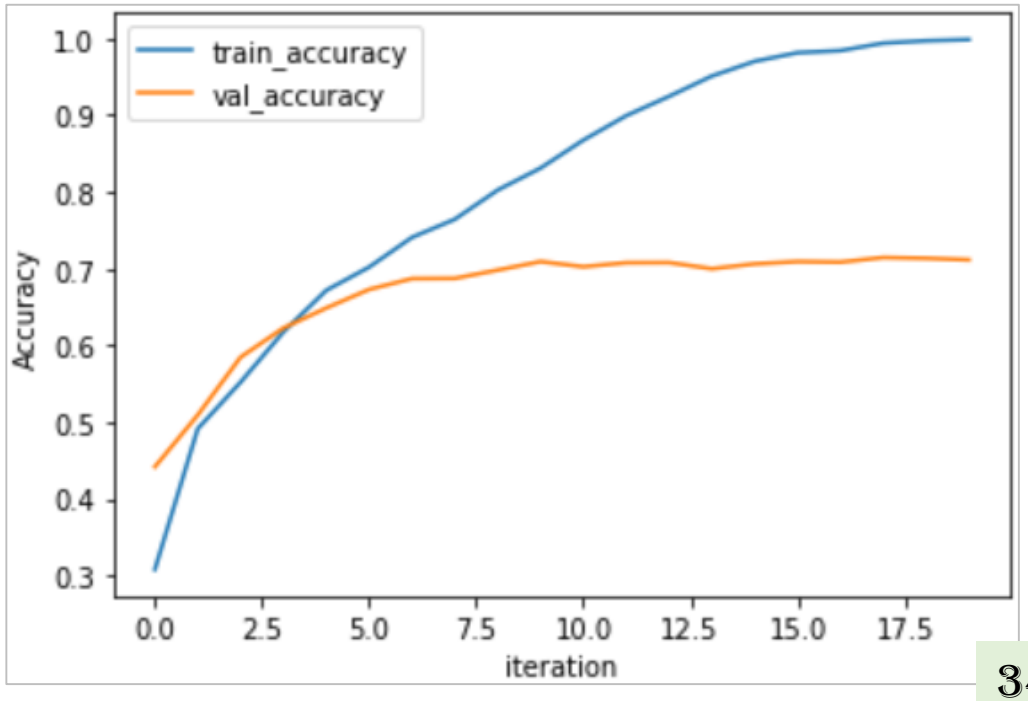
1x1 Convolution

```python
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras
import tensorflow.keras.layers as layers

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

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

# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(32, 32, 3)))

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

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

# training
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=1024,
                    validation_data=(x_test, y_test), epochs=20, verbose=1)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 256)         295168
_________________________________________________________________
flatten (Flatten)            (None, 16384)             0
_________________________________________________________________
dense (Dense)                (None, 512)               8389120
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 8,782,666
Trainable params: 8,782,666
Non-trainable params: 0
_________________________________________________________________
```

```python
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras
import tensorflow.keras.layers as layers

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

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

# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(32, 32, 3)))

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

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

# training
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=1024,
                    validation_data=(x_test, y_test), epochs=20, verbose=1)
```



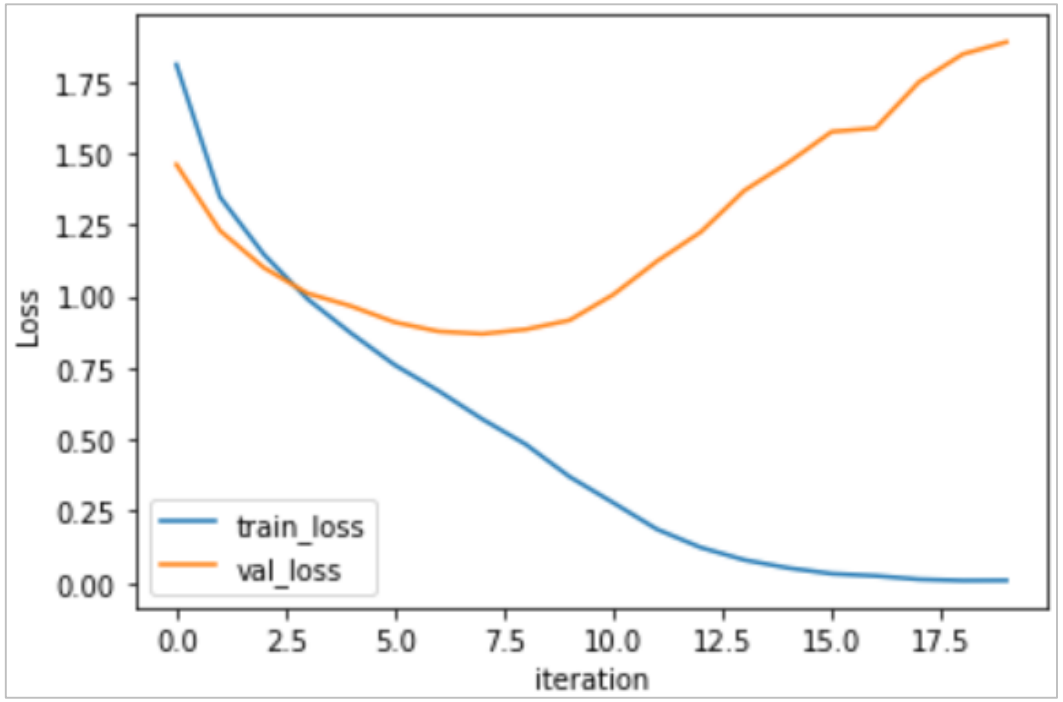


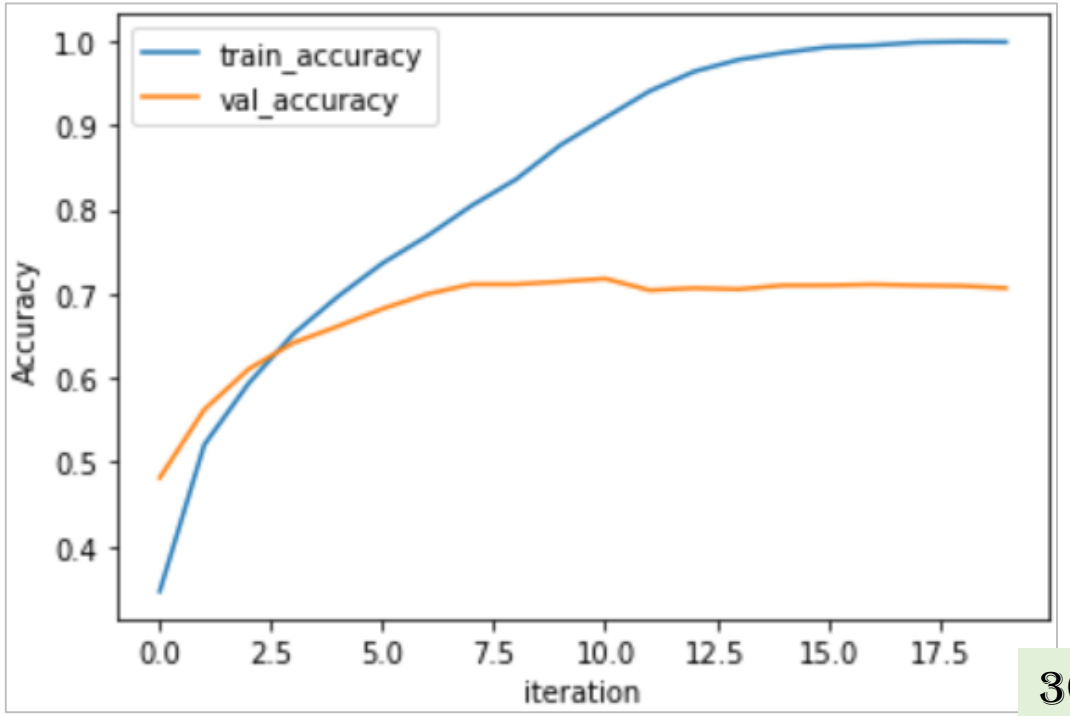
1x1 Convolution

```python
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras
import tensorflow.keras.layers as layers

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

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

# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(32, 32, 3)))

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

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

# flatten
model.add(layers.Conv2D(512, (8, 8), activation='relu'))
model.add(layers.Conv2D(10, (1,1), activation='softmax'))
model.add(layers.Reshape((10,)))
model.summary()

# training
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=1024,
                    validation_data=(x_test, y_test), epochs=20, verbose=1)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_34 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 16, 16, 128)       73856
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 8, 8, 256)         295168
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 1, 1, 512)         8389120
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 1, 1, 10)          5130
_________________________________________________________________
reshape_3 (Reshape)          (None, 10)                0
=================================================================
Total params: 8,782,666
Trainable params: 8,782,666
Non-trainable params: 0
_________________________________________________________________
```

```python
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras
import tensorflow.keras.layers as layers

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

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

# model
model = keras.models.Sequential()
model.add(tf.keras.Input(shape=(32, 32, 3)))

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

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

# flatten
model.add(layers.Conv2D(512, (8, 8), activation='relu'))
model.add(layers.Conv2D(10, (1,1), activation='softmax'))
model.add(layers.Reshape((10,)))
model.summary()

# training
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=1024,
                    validation_data=(x_test, y_test), epochs=20, verbose=1)
```





1x1 Convolution

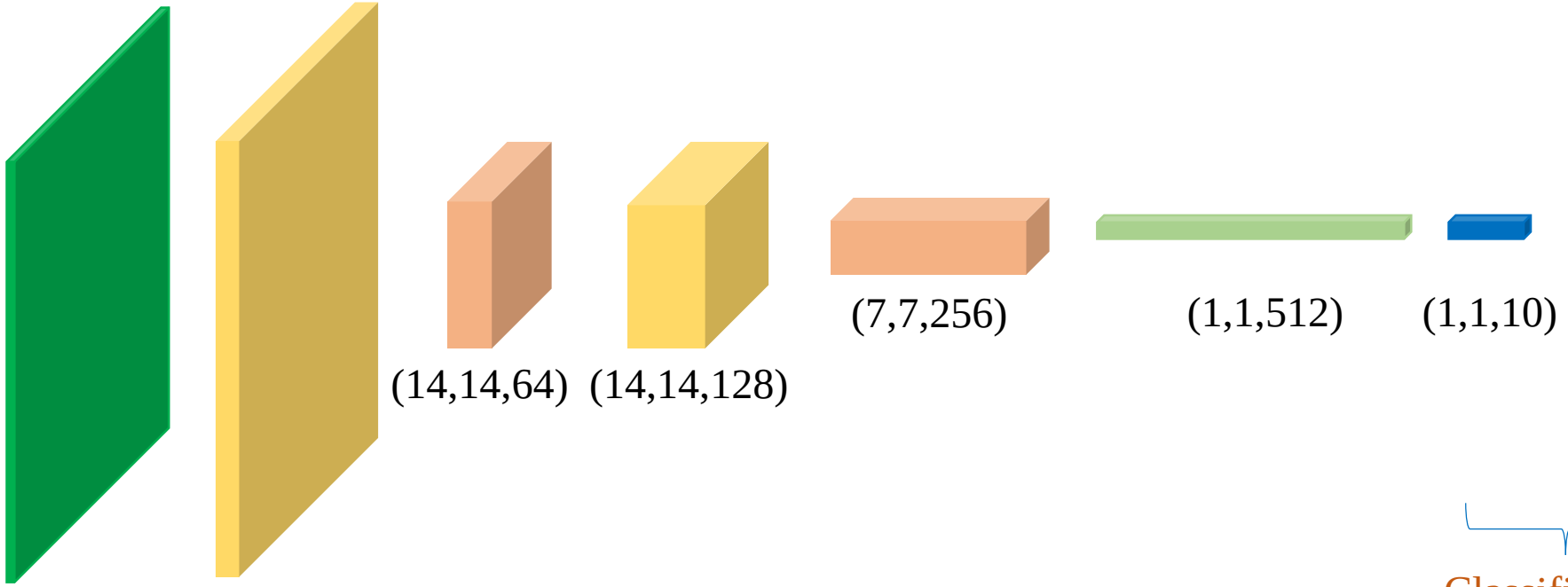
Replace FC by 1x1 Conv

```
# 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'))
```

(3x3) Conv with stride=1, padding='same' + ReLU

(3x3) Conv with stride=2, padding='same' + ReLU

(7x7) Conv + ReLU



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'))
```

```
1 shape = (32,28,28,1)
2 data = tf.random.normal(shape)
3 print(data.shape)
4
5 output = model.predict(data)
6 print(output.shape)
```

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

```
1 shape = (32,64,64,1)
2 data = tf.random.normal(shape)
3 print(data.shape)
4
5 output = model.predict(data)
6 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
- Backpropagation

Sigmoid Function

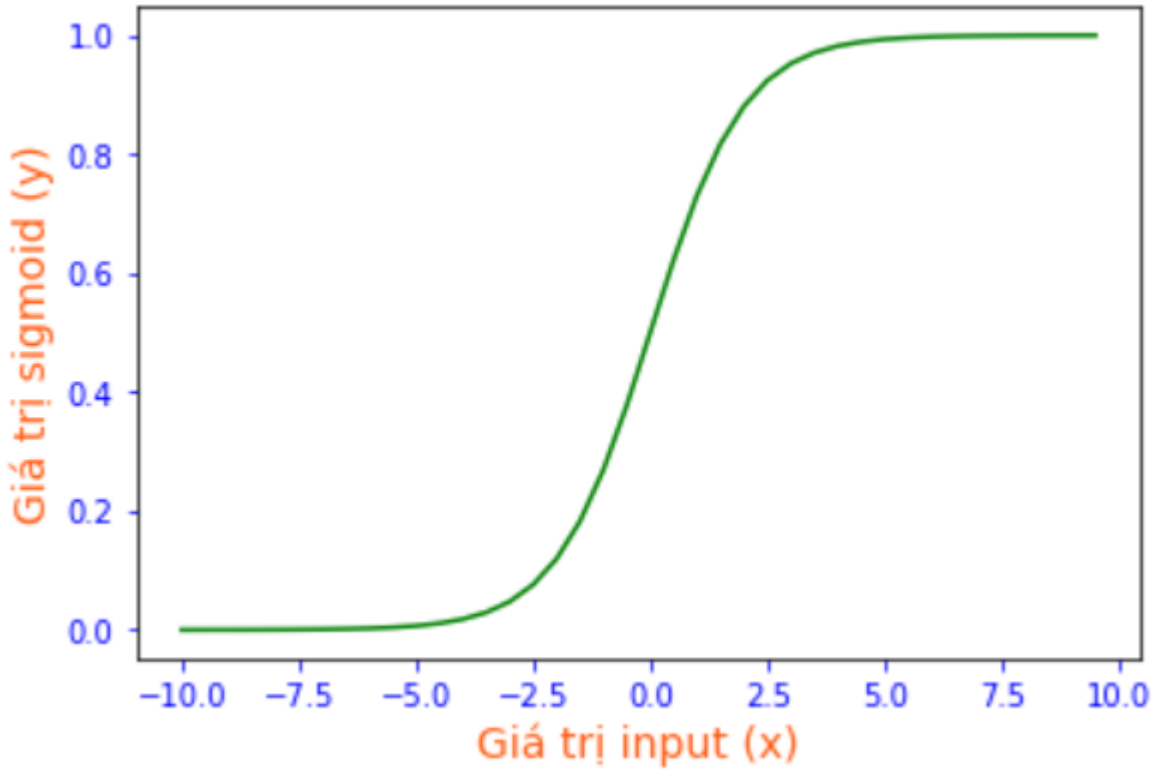
❖ Sigmoid function

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

data =	1	5	-4	3	-2

data_a = sigmoid(data)

data_a =	0.731	0.993	0.017	0.95	0.119



Sigmoid Function

Sigmoid function

$$y = \sigma(u) = \frac{1}{1 + e^{-u}}$$

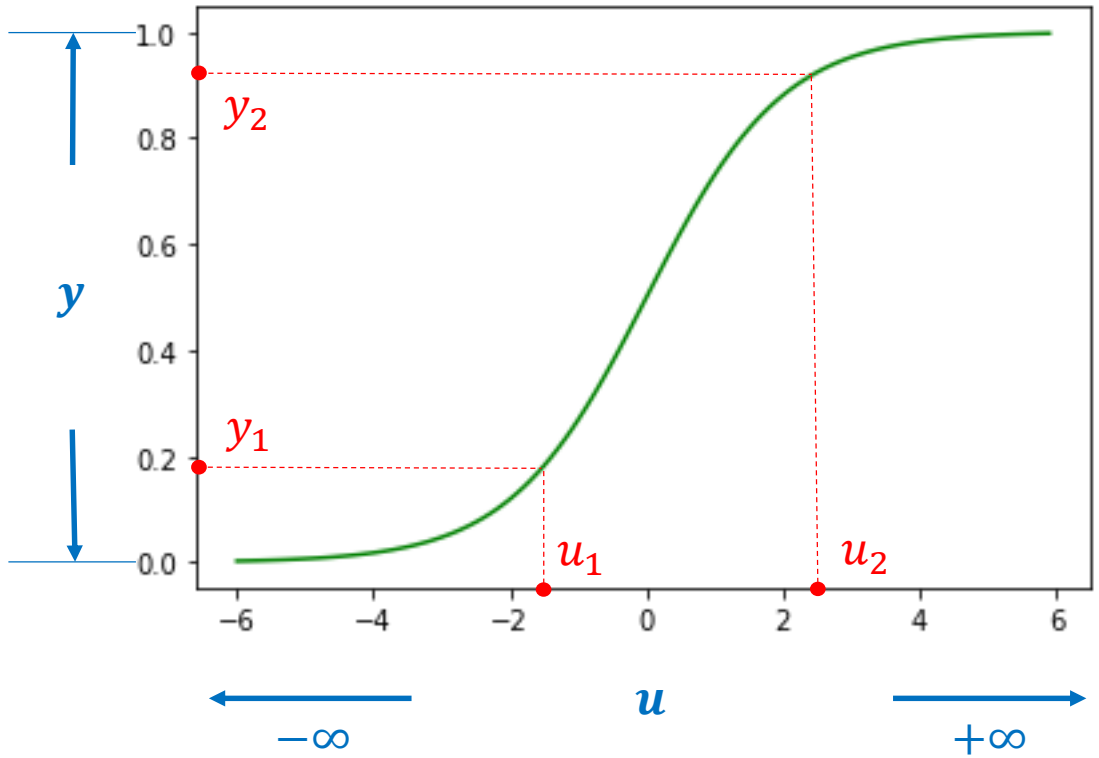
$$u \in (-\infty \quad +\infty)$$

$$y \in (0 \quad 1)$$

Property

$$\forall u_1 u_2 \in [a \ \ b] \text{ và } u_1 \leq u_2$$

$$\rightarrow \sigma(u_1) \leq \sigma(u_1)$$



Sigmoid Function

❖ Construct loss

Model and Loss

$$z = \boldsymbol{\theta}^T \boldsymbol{x}$$

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\mathrm{L} = \frac{1}{N}[-y^T \log(\hat{y}) - (1 - y^T)\log(1 - \hat{y})]$$

Derivative

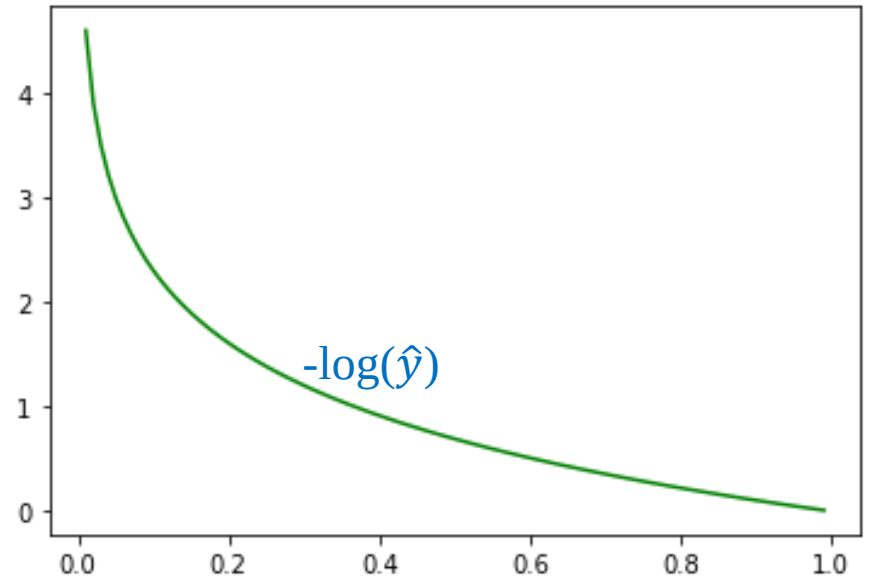
$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial \theta}$$

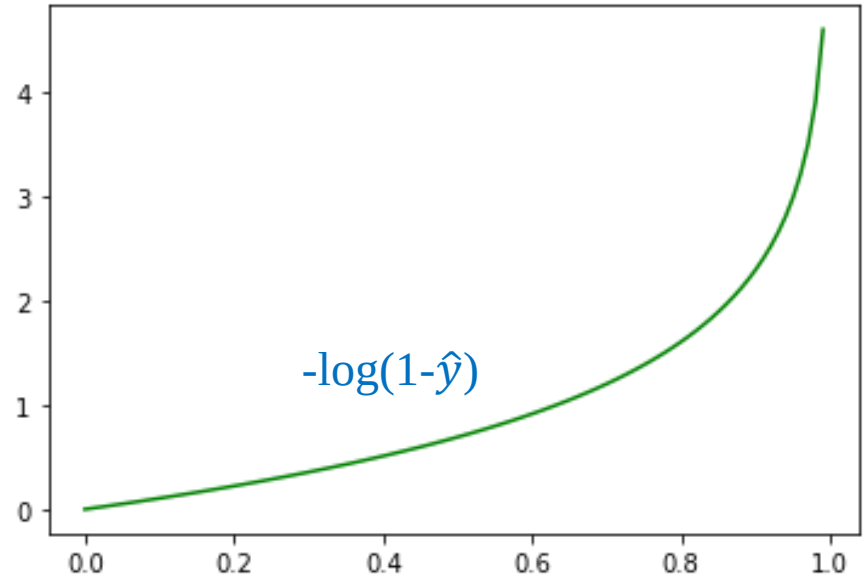
$$\frac{\partial L}{\partial \hat{y}} = \frac{1}{N}\left(-\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}}\right) = \frac{1}{N}\frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y})$$

$$\frac{\partial z}{\partial \theta} = x$$

$$\frac{\partial L}{\partial \theta} = \frac{1}{N} x^T (\hat{y} - y)$$





Sigmoid Function

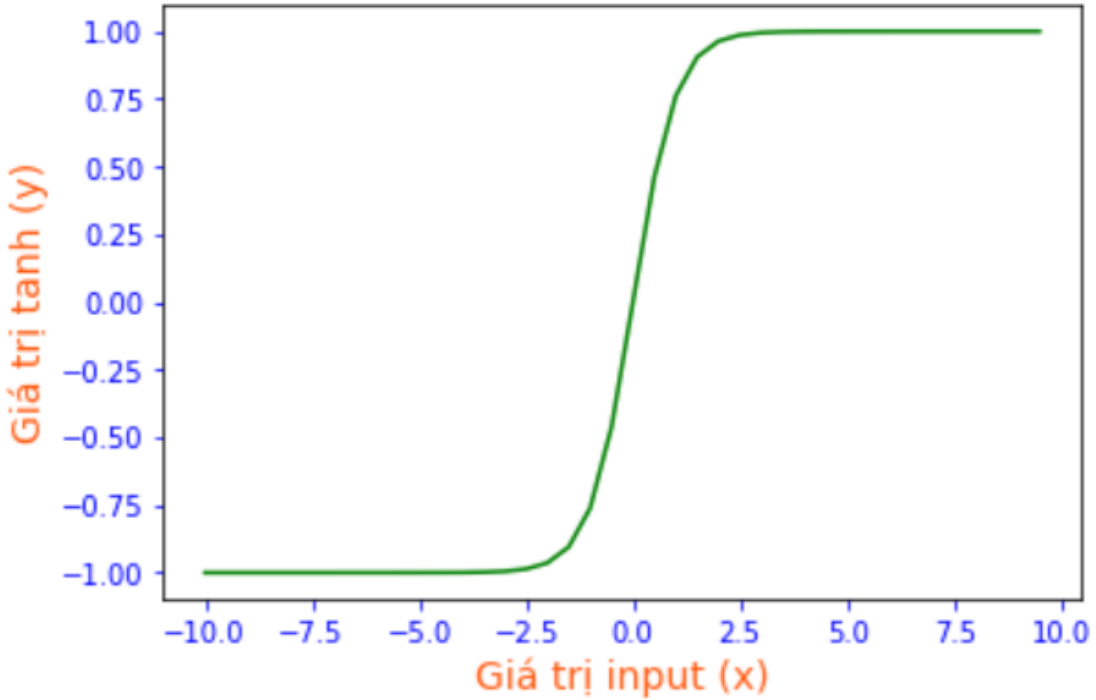
```python
# sigmoid

import tensorflow as tf
import tensorflow.keras as keras
import numpy as np

sigmoid_activation = keras.activations.sigmoid

data_X = np.array([[-3, 9, 2],
                   [5, -8, 4],
                   [-7, 3, 5]])
data_X = data_X.reshape(1, 3, 3, 1)
data_X = tf.Variable(data_X, dtype=tf.float32)

data_y = np.ones((1, 3, 3, 1))
data_y = tf.Variable(data_y, dtype=tf.float32)

print('\n\n data_X: \n', data_X[0,:,:,0])
print('\n\n data_y: \n', data_y[0,:,:,0])

with tf.GradientTape(persistent=True) as tape:
    data_after = sigmoid_activation(data_X)
    print('\n\n data_after: \n', data_after[0,:,:,0])

    loss = tf.reduce_mean((data_after-data_y)**2)
    print('\n\n loss: \n', loss)

dloss = tape.gradient(loss, data_after)
print('\n\n dloss: \n', dloss[0,:,:,0])

ddata_X = tape.gradient(loss, data_X)
print('\n\n ddata_X: \n', ddata_X[0,:,:,0])
```

```
 data_X:
 tf.Tensor(
[[-3.  9.  2.]
 [ 5. -8.  4.]
 [-7.  3.  5.]], shape=(3, 3), dtype=float32)


 data_y:
 tf.Tensor(
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]], shape=(3, 3), dtype=float32)


 data_after_mp:
 tf.Tensor(
[[4.7425866e-02 9.9987662e-01 8.8079709e-01]
 [9.9330711e-01 3.3542514e-04 9.8201376e-01]
 [9.1102719e-04 9.5257413e-01 9.9330717e-01]], shape=(3, 3), dtype=float32)


 loss:
 tf.Tensor(0.32464194, shape=(), dtype=float32)


 dloss:
 tf.Tensor(
[[-2.1168314e-01 -2.7418137e-05 -2.6489535e-02]
 [-1.4873081e-03 -2.2214767e-01 -3.9969417e-03]
 [-2.2201978e-01 -1.0539082e-02 -1.4872948e-03]], shape=(3, 3), dtype=float32)


 ddata_X:
 tf.Tensor(
[[-9.5631359e-03 -3.3824765e-09 -2.7812310e-03]
 [-9.8877599e-06 -7.4488918e-05 -7.0596914e-05]
 [-2.0208179e-04 -4.7612048e-04 -9.8875844e-06]], shape=(3, 3), dtype=float32)
```

Tanh Function

❖ Tanh function

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



data =

1	5	-4	3	-2

data_a = tanh(data)

data_a =

0.761	0.999	-0.999	0.995	-0.964

Tanh Function

❖ Construct loss

Model and Loss

$$z = \boldsymbol{\theta}^T \boldsymbol{x}$$

$$\hat{y} = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$\mathrm{L} = \frac{1}{N}[-y^T \log(\hat{y}) - (1 - y^T)\log(1 - \hat{y})]$$

Derivative

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial \theta}$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{1}{N}\left(-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right) = \frac{1}{N}\frac{\hat{y} - y}{\hat{y}(1-\hat{y})}$$

$$\frac{\partial \hat{y}}{\partial z} = 1 - \hat{y}^2$$

$$\frac{\partial z}{\partial \theta} = x$$

$$\frac{\partial L}{\partial \theta} = \frac{1}{N} x^T \frac{(\hat{y} - y)(1 + \hat{y})}{\hat{y}}$$

Tanh Function

```python
# tanh

import tensorflow as tf
import tensorflow.keras as keras
import numpy as np

tanh_activation = keras.activations.tanh

data_X = np.array([[-1, 0.6, 0.3],
                   [5, -8, 4],
                   [-7, 3, 5]])
data_X = data_X.reshape(1, 3, 3, 1)
data_X = tf.Variable(data_X, dtype=tf.float32)

data_y = np.ones((1, 3, 3, 1))
data_y = tf.Variable(data_y, dtype=tf.float32)

print('\n\n data_X: \n', data_X[0,:,:,0])
print('\n\n data_y: \n', data_y[0,:,:,0])

with tf.GradientTape(persistent=True) as tape:
    data_after = tanh_activation(data_X)
    print('\n\n data_after: \n', data_after[0,:,:,0])

    loss = tf.reduce_mean((data_after-data_y)**2)
    print('\n\n loss: \n', loss)

dloss = tape.gradient(loss, data_after)
print('\n\n dloss: \n', dloss[0,:,:,0])

ddata_X = tape.gradient(loss, data_X)
print('\n\n ddata_X: \n', ddata_X[0,:,:,0])
```

```
data_X:
tf.Tensor(
[-1.   0.6  0.3]
[ 5.  -8.   4. ]
[-7.   3.   5. ]], shape=(3, 3), dtype=float32)


data_y:
tf.Tensor(
[1. 1. 1.]
[1. 1. 1.]
[1. 1. 1.]], shape=(3, 3), dtype=float32)


data_after:
tf.Tensor(
[-0.7615942   0.53704965  0.2913126 ]
[ 0.99990916 -0.99999976  0.9993292 ]
[-0.99999833  0.9950547   0.99990916]], shape=(3, 3), dtype=float32)


loss:
tf.Tensor(1.3133101, shape=(), dtype=float32)


dloss:
tf.Tensor(
[-3.91465366e-01 -1.02877855e-01 -1.57486096e-01]
[-2.01861058e-05 -4.44444388e-01 -1.49064595e-04]
[-4.44444090e-01 -1.09895074e-03 -2.01861058e-05]], shape=(3, 3), dtype=float32)


ddata_X:
tf.Tensor(
[-1.6440541e-01 -7.3205583e-02 -1.4412135e-01]
[-3.6673100e-09 -2.1192760e-07 -1.9991120e-07]
[-1.4834922e-06 -1.0842378e-05 -3.6673100e-09]], shape=(3, 3), dtype=float32)
```

ReLU Function

$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

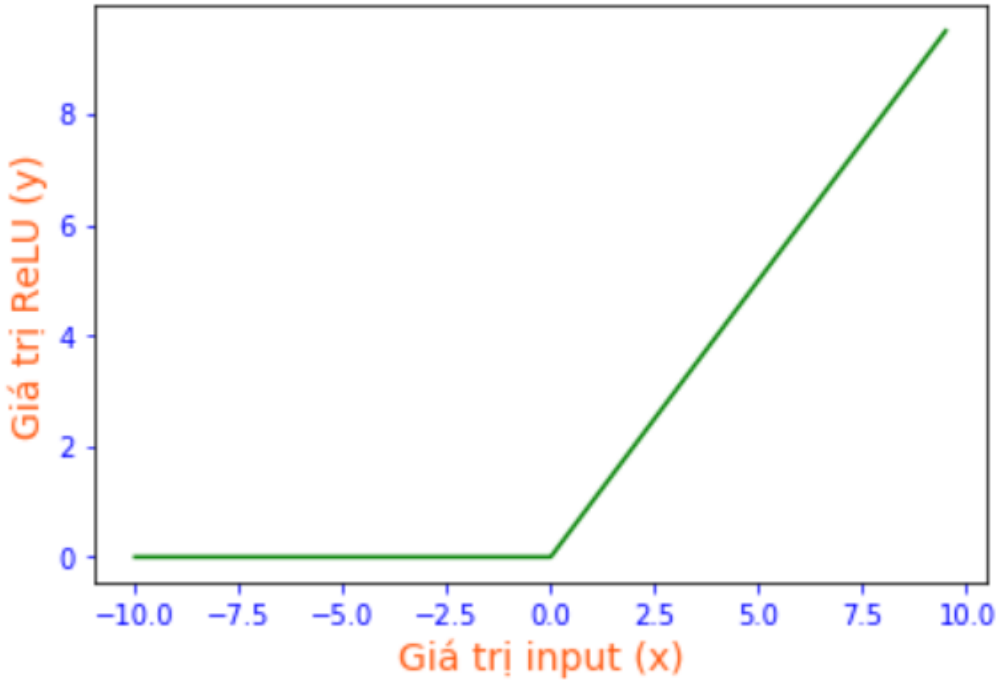
data =

1	5	-4	3	-2

data_a = ReLU(data)

data_a =

1	5	0	3	0



ReLU Function

```python
# relu

import tensorflow as tf
import tensorflow.keras as keras
import numpy as np

relu_activation = keras.activations.relu

data_X = np.array([[1, 5, 3, -7],
                   [7, -3, 9, 2],
                   [3, 5, -8, 4],
                   [8, -7, 3, 5]])
data_X = data_X.reshape(1, 4, 4, 1)
data_X = tf.Variable(data_X, dtype=tf.float32)

data_y = np.ones((1, 4, 4, 1))
data_y = tf.Variable(data_y, dtype=tf.float32)

print('\n\n data_X: \n', data_X[0,:,:,0])
print('\n\n data_y: \n', data_y[0,:,:,0])

with tf.GradientTape(persistent=True) as tape:
    data_after = relu_activation(data_X)
    print('\n\n data_after: \n', data_after[0,:,:,0])

    loss = tf.reduce_mean((data_after-data_y)**2)
    print('\n\n loss: \n', loss)

dloss = tape.gradient(loss, data_after)
print('\n\n dloss: \n', dloss[0,:,:,0])

ddata_X = tape.gradient(loss, data_X)
print('\n\n ddata_X: \n', ddata_X[0,:,:,0])
```

```
data_X:
tf.Tensor(
[[ 1.  5.  3. -7.]
 [ 7. -3.  9.  2.]
 [ 3.  5. -8.  4.]
 [ 8. -7.  3.  5.]], shape=(4, 4), dtype=float32)


data_y:
tf.Tensor(
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]], shape=(4, 4), dtype=float32)


data_after:
tf.Tensor(
[[1. 5. 3. 0.]
 [7. 0. 9. 2.]
 [3. 5. 0. 4.]
 [8. 0. 3. 5.]], shape=(4, 4), dtype=float32)


loss:
tf.Tensor(13.9375, shape=(), dtype=float32)


dloss:
tf.Tensor(
[[ 0.     0.5    0.25  -0.125]
 [ 0.75  -0.125  1.     0.125]
 [ 0.25   0.5   -0.125  0.375]
 [ 0.875 -0.125  0.25   0.5  ]], shape=(4, 4), dtype=float32)


ddata_X:
tf.Tensor(
[[ 0.     0.5    0.25  -0.   ]
 [ 0.75  -0.     1.     0.125]
 [ 0.25   0.5   -0.     0.375]
 [ 0.875 -0.     0.25   0.5  ]], shape=(4, 4), dtype=float32)
```

PReLU Function

❖ PReLU function

$$\text{PReLU}(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

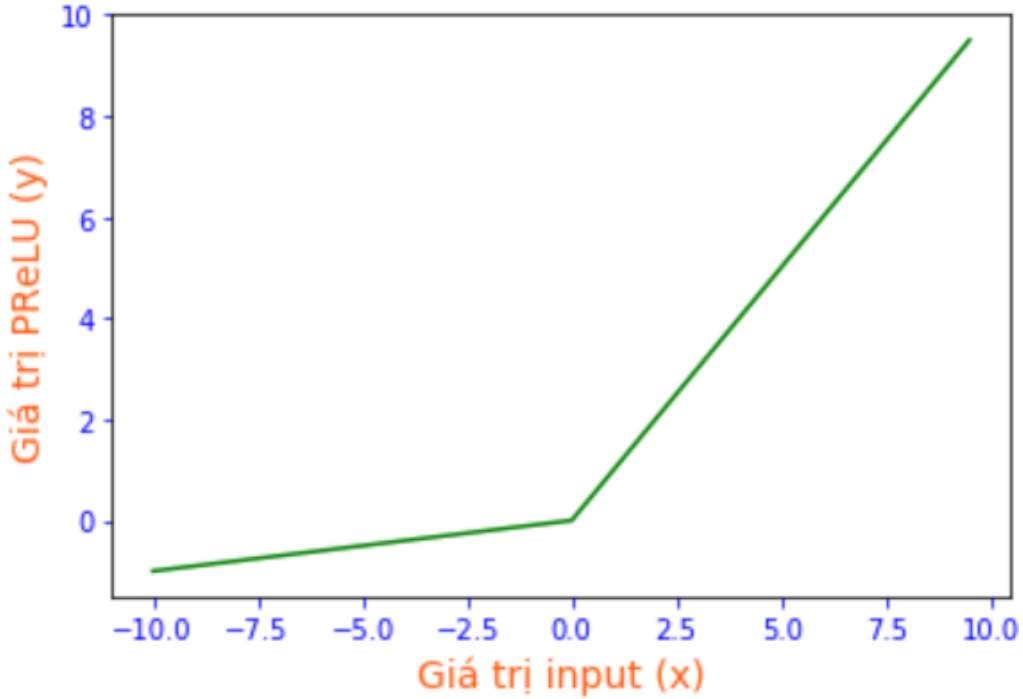
data =

1	5	-4	3	-2

data_a = PRELU(data)

data_a =

1	5	-0.4	3	-0.2



ELU Function

❖ ELU function

$$\text{ELU}(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

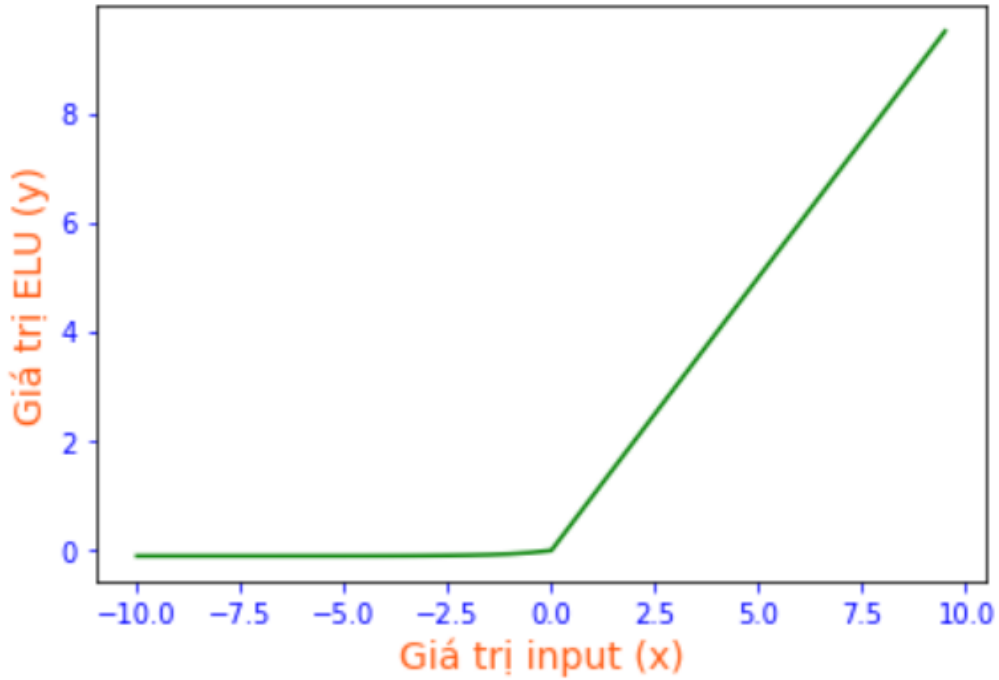
data =

1	5	-4	3	-2

data_a = ELU(data)

data_a =

1	5	-0.098	3	-0.086



Softplus Function

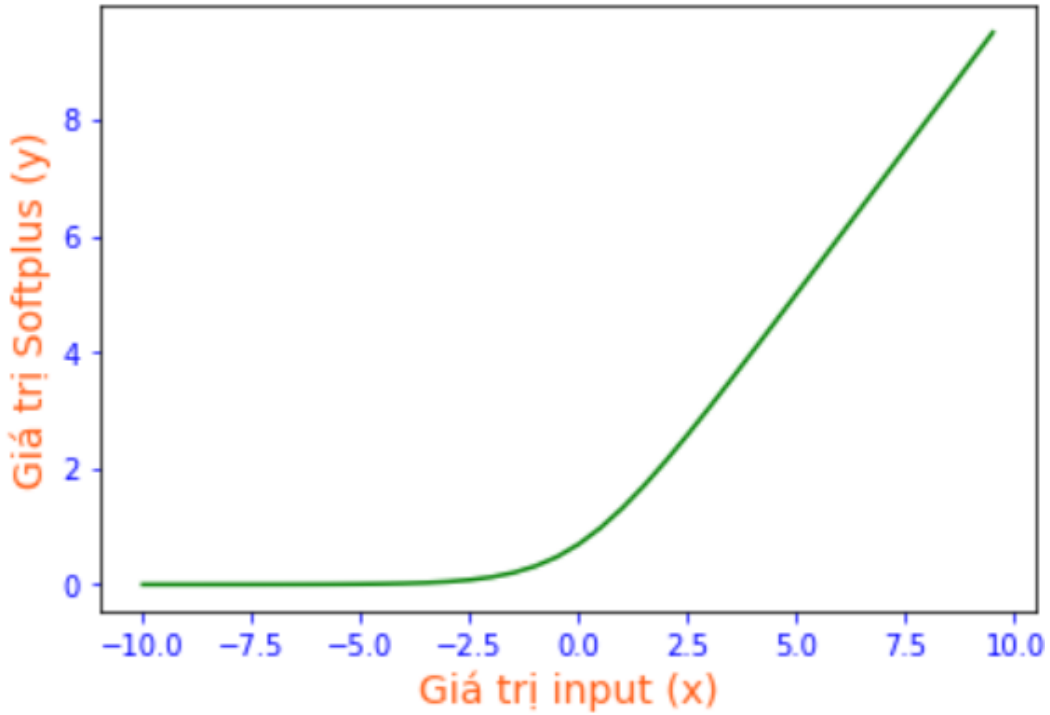
❖ Softplus function

$$\text{softplus}(x) = \log(1 + e^x)$$

data =	1	5	-4	3	-2

data_a = softplus(data)

data_a =	1.313	5.006	0.018	3.048	0.126



Implementation (straightforward)

Softmax function

Chuyển các giá trị của một vector thành các giá trị xác suất

Formula

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

$$0 \le f(x_i) \le 1$$

$$\sum_i f(x_i) = 1$$

Input

$x_1 = 1.0$

$x_2 = 2.0$

$x_3 = 3.0$

→ **Softmax** →

Probability

$f(x_1) = 0.09$

$f(x_2) = 0.24$

$f(x_3) = 0.67$

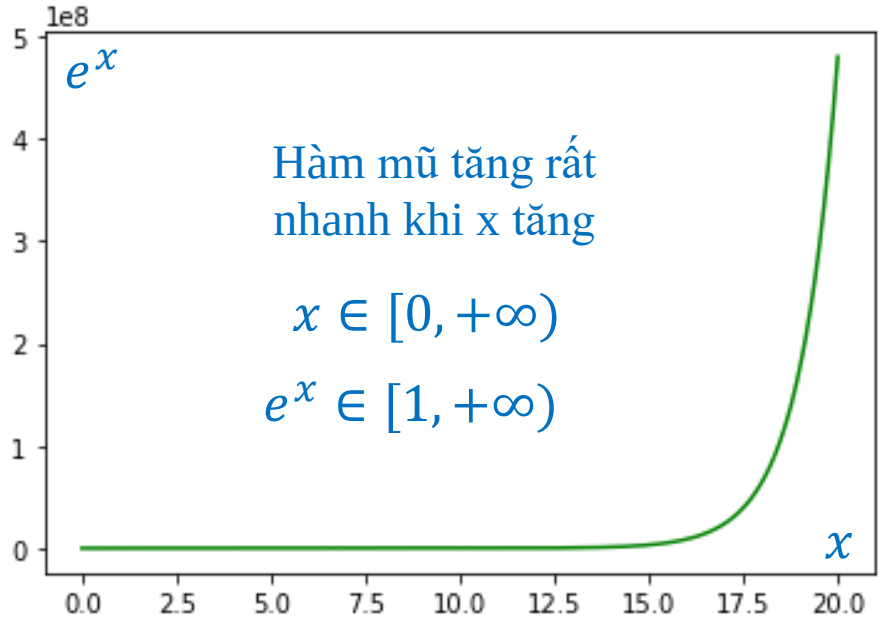
```python
import numpy as np

def softmax(X):
    exps = np.exp(X)
    return exps / np.sum(exps)
```

```python
X = np.array([1.0, 2.0, 3.0])
f = softmax(X)
print(f)
```

```
[0.09003057 0.24472847 0.66524096]
```

```python
X = np.array([1000.0, 1001.0, 1002.0])
f = softmax(X)
print(f)
```

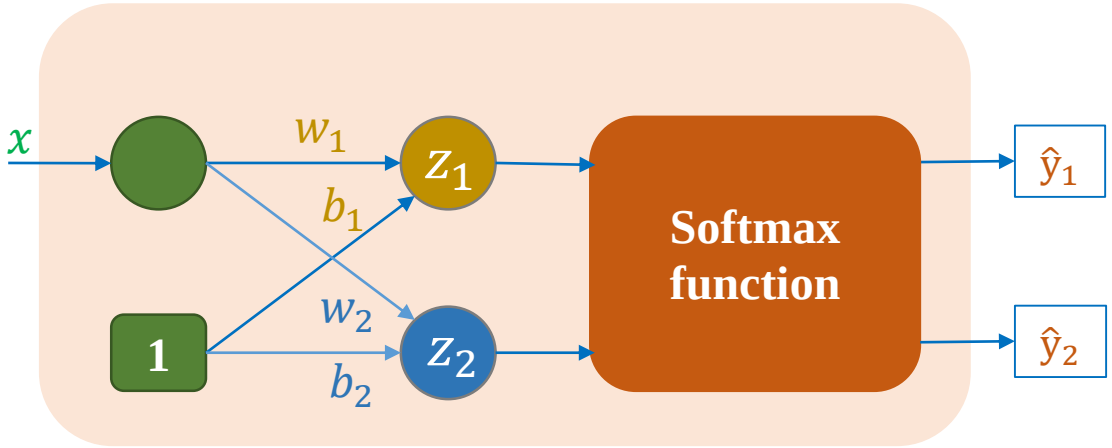
```
[nan nan nan]
```



Giá trị nan vì e^x vượt giới hạn lưu trữ của biến

Softmax function

Model



$$L(\boldsymbol{\theta}) = -\sum_{i=1}^{2} \delta(i, y) \log \hat{y}_i$$

$$\hat{y}_1 = \frac{e^{z_1}}{\sum_{j=1}^{2} e^{z_j}}$$

$$\hat{y}_2 = \frac{e^{z_2}}{\sum_{j=1}^{2} e^{z_j}}$$

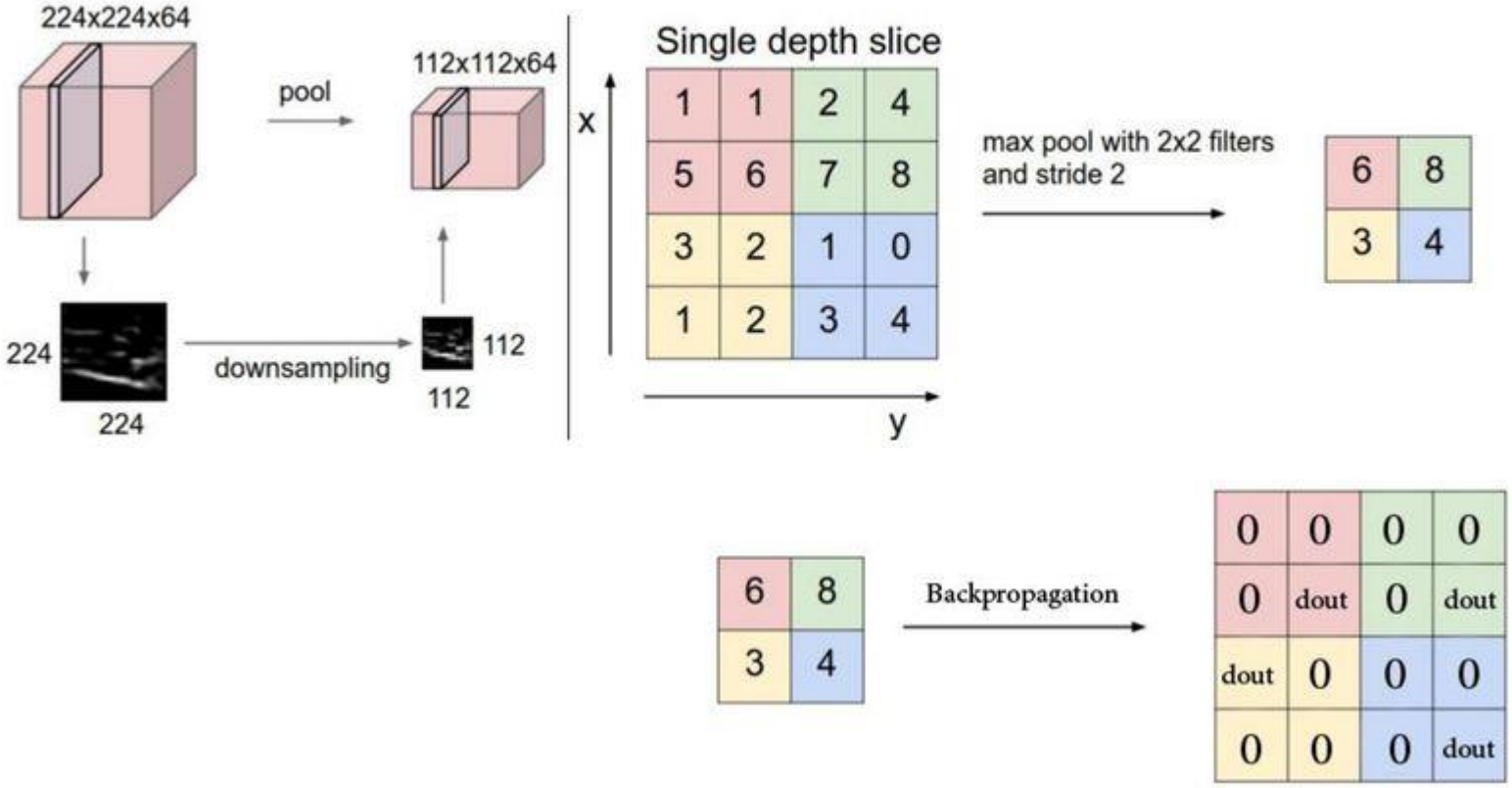
Derivative

$$\frac{\partial \hat{y}_i}{\partial z_j} = \hat{y}_i\left(\delta(i,j) - \hat{y}_j\right)$$

$$\frac{\partial L}{\partial z_i} = \hat{y}_i - \delta(i, y)$$

Max Pooling

Forward and Backward Max Pooling



https://slideplayer.com/slide/15983712/

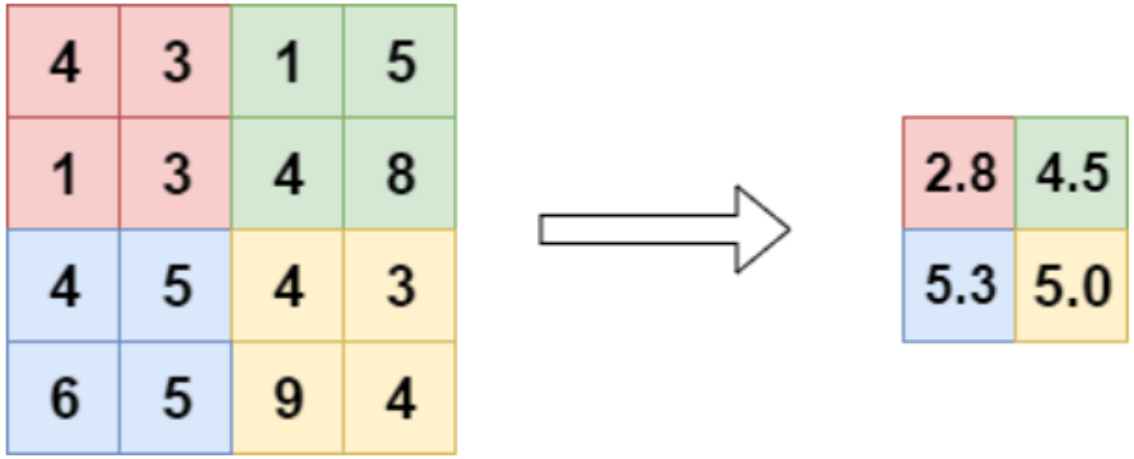
Max Pooling

```python
# max_pooling

import tensorflow as tf
import tensorflow.keras as keras
import numpy as np

max_pooling = keras.layers.MaxPooling2D(pool_size=2, strides=2)

data_X = np.array([[1, 5, 3, 7],
                   [7, 3, 9, 2],
                   [3, 5, 8, 4],
                   [8, 7, 3, 5]])
data_X = data_X.reshape(1, 4, 4, 1)
data_X = tf.Variable(data_X, dtype=tf.float32)

data_y = np.ones((1, 2, 2, 1))
data_y = tf.Variable(data_y, dtype=tf.float32)

print('\n\n data_X: \n', data_X[0,:,:,0])
print('\n\n data_y: \n', data_y[0,:,:,0])

with tf.GradientTape(persistent=True) as tape:
    data_after = max_pooling(data_X)
    print('\n\n data_after: \n', data_after[0,:,:,0])

    loss = tf.reduce_mean((data_after-data_y)**2)
    print('\n\n loss: \n', loss)

dloss = tape.gradient(loss, data_after)
print('\n\n dloss: \n', dloss[0,:,:,0])

ddata_X = tape.gradient(loss, data_X)
print('\n\n ddata_X: \n', ddata_X[0,:,:,0])
```

```
data_X:
 tf.Tensor(
[[1. 5. 3. 7.]
 [7. 3. 9. 2.]
 [3. 5. 8. 4.]
 [8. 7. 3. 5.]], shape=(4, 4), dtype=float64)


data_y:
 tf.Tensor(
[[1. 1.]
 [1. 1.]], shape=(2, 2), dtype=float64)


data_after:
 tf.Tensor(
[[7. 9.]
 [8. 8.]], shape=(2, 2), dtype=float64)


loss:
 tf.Tensor(49.5, shape=(), dtype=float64)


dloss:
 tf.Tensor(
[[3.  4. ]
 [3.5 3.5]], shape=(2, 2), dtype=float64)


ddata_X:
 tf.Tensor(
[[0.  0.  0.  0. ]
 [3.  0.  4.  0. ]
 [0.  0.  3.5 0. ]
 [3.5 0.  0.  0. ]], shape=(4, 4), dtype=float64)
```

Average Pooling

Formula



AI VIETNAM
AI Insight Course

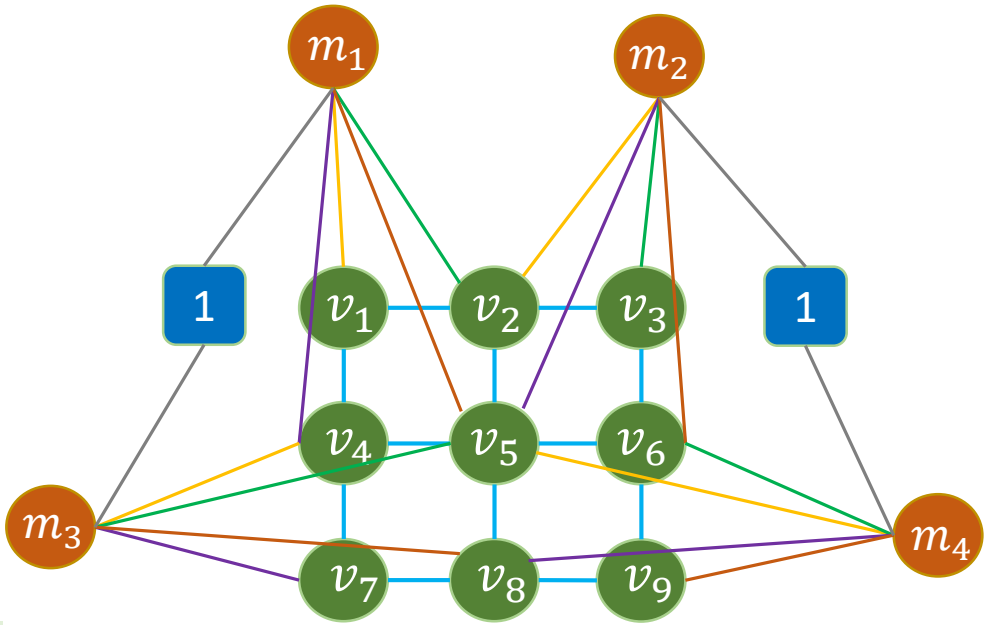
Average Pooling

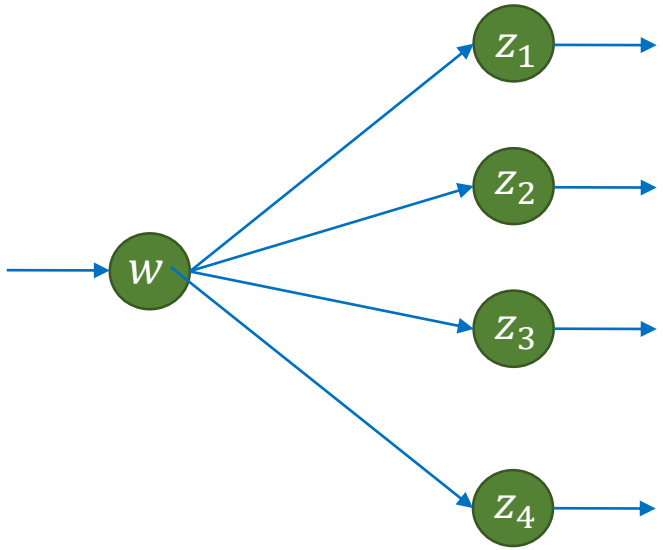
```python
# AveragePooling2D

import tensorflow as tf
import tensorflow.keras as keras
import numpy as np

avg_pooling = keras.layers.AveragePooling2D(pool_size=2, strides=2)

data_X = np.array([[1, 5, 3, 7],
                   [7, 3, 9, 2],
                   [3, 5, 8, 4],
                   [8, 7, 3, 5]])
data_X = data_X.reshape(1, 4, 4, 1)
data_X = tf.Variable(data_X, dtype=tf.float32)

data_y = np.ones((1, 2, 2, 1))
data_y = tf.Variable(data_y, dtype=tf.float32)

print('\n\n data_X: \n', data_X[0,:,:,0])
print('\n\n data_y: \n', data_y[0,:,:,0])

with tf.GradientTape(persistent=True) as tape:
    data_after = avg_pooling(data_X)
    print('\n\n data_after_ap: \n', data_after[0,:,:,0])

    loss = tf.reduce_mean((data_after-data_y)**2)
    print('\n\n loss: \n', loss)

dloss = tape.gradient(loss, data_after)
print('\n\n dloss: \n', dloss[0,:,:,0])

ddata_X = tape.gradient(loss, data_X)
print('\n\n ddata_X: \n', ddata_X[0,:,:,0])
```

```
 data_X:
 tf.Tensor(
[[1. 5. 3. 7.]
 [7. 3. 9. 2.]
 [3. 5. 8. 4.]
 [8. 7. 3. 5.]], shape=(4, 4), dtype=float32)


 data_y:
 tf.Tensor(
[[1. 1.]
 [1. 1.]], shape=(2, 2), dtype=float32)


 data_after_ap:
 tf.Tensor(
[[4.   5.25]
 [5.75 5.  ]], shape=(2, 2), dtype=float32)


 loss:
 tf.Tensor(16.40625, shape=(), dtype=float32)


 dloss:
 tf.Tensor(
[[1.5   2.125]
 [2.375 2.   ]], shape=(2, 2), dtype=float32)


 ddata_X:
 tf.Tensor(
[[0.375   0.375   0.53125 0.53125]
 [0.375   0.375   0.53125 0.53125]
 [0.59375 0.59375 0.5     0.5    ]
 [0.59375 0.59375 0.5     0.5    ]], shape=(4, 4), dtype=float32)
```

Convolution

Kernel of parameters





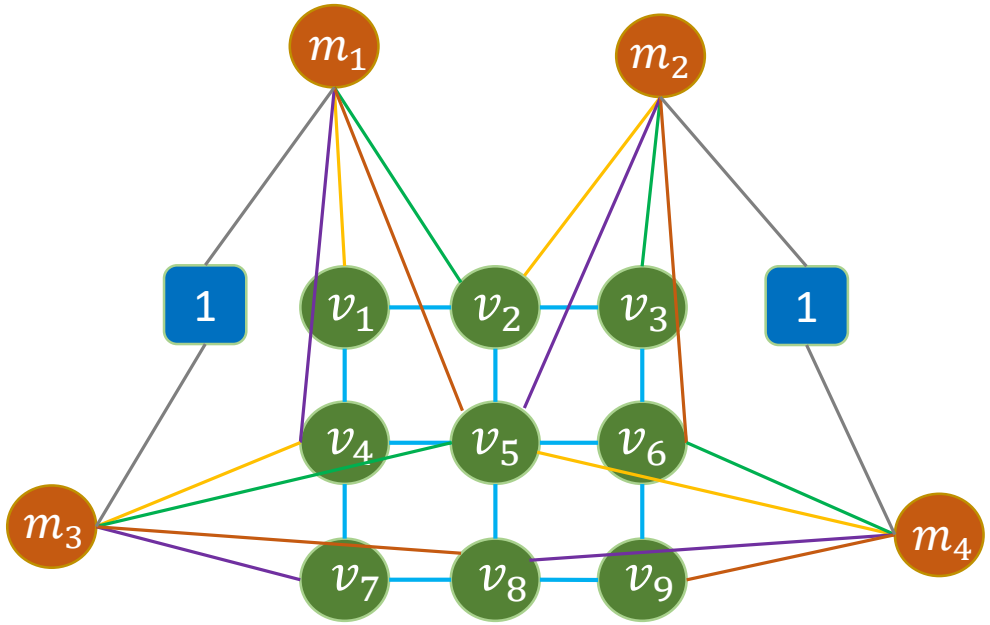
$$\frac{\partial L}{\partial w} = \sum_{i=1}^{4} \frac{\partial L}{\partial z_i} \frac{\partial z_i}{\partial w}$$

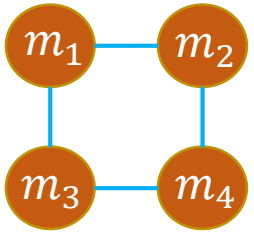
Convolution

w_1	w_2
w_3	w_4

b

Kernel of parameters





Feature Map

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

$$m_2 = v_2w_1 + v_3w_2 + v_5w_3 + v_6w_4 + b$$

$$m_3 = v_4w_1 + v_5w_2 + v_7w_3 + v_8w_4 + b$$

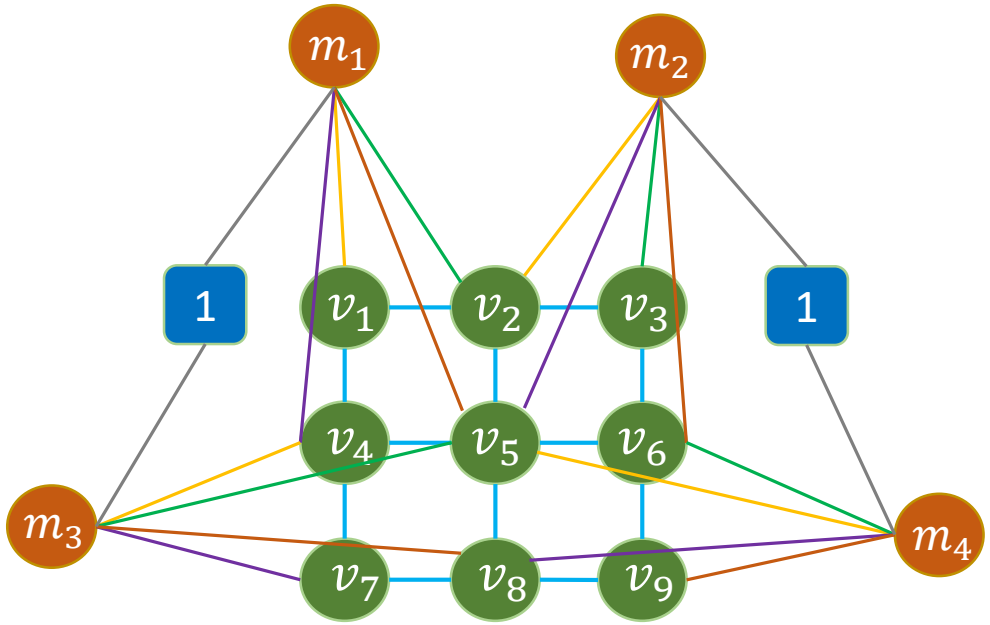
$$m_4 = v_5w_1 + v_6w_2 + v_8w_3 + v_9w_4 + b$$

Convolution

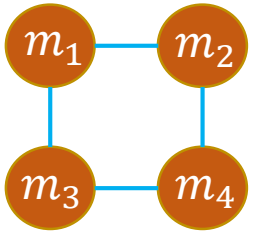
w_1	w_2
w_3	w_4

b

Kernel of parameters



$$\frac{\partial L}{\partial w_1} = \sum_i \frac{\partial L}{\partial m_i} \frac{\partial m_i}{\partial w_1}$$



Feature Map

$$m_1 = v_1 w_1 + v_2 w_2 + v_4 w_3 + v_5 w_4 + b$$

$$m_2 = v_2 w_1 + v_3 w_2 + v_5 w_3 + v_6 w_4 + b$$

$$m_3 = v_4 w_1 + v_5 w_2 + v_7 w_3 + v_8 w_4 + b$$

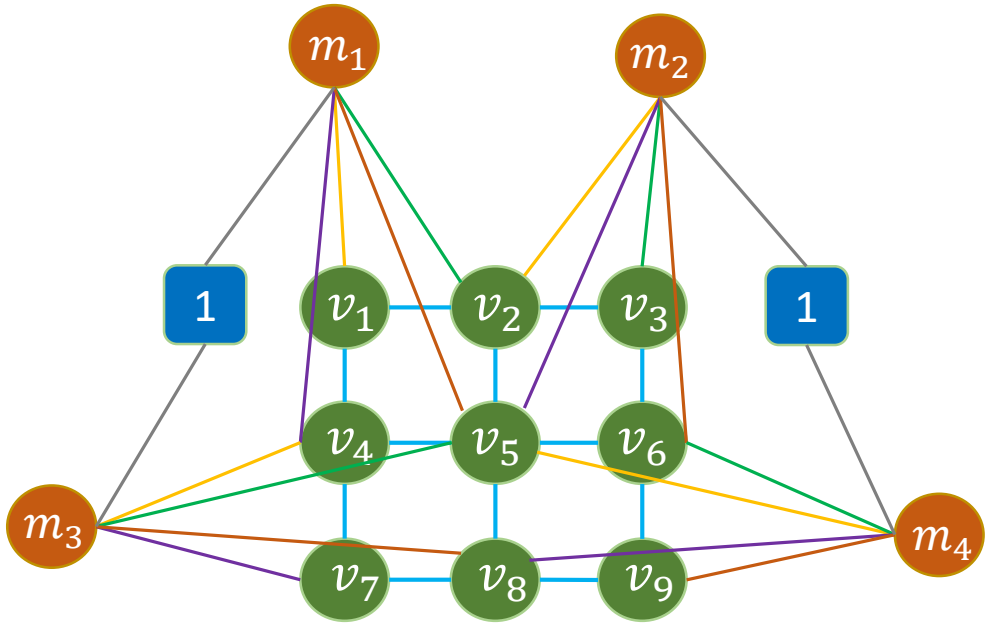
$$m_4 = v_5 w_1 + v_6 w_2 + v_8 w_3 + v_9 w_4 + b$$

Convolution

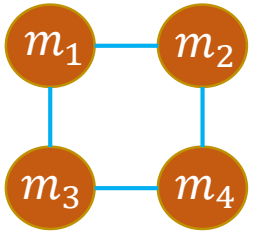
w_1	w_2
w_3	w_4

b

Kernel of parameters



$$\frac{\partial L}{\partial w_1} = \sum_i v_i \frac{\partial L}{\partial m_i}$$



Feature Map

$$m_1 = v_1 w_1 + v_2 w_2 + v_4 w_3 + v_5 w_4 + b$$

$$m_2 = v_2 w_1 + v_3 w_2 + v_5 w_3 + v_6 w_4 + b$$

$$m_3 = v_4 w_1 + v_5 w_2 + v_7 w_3 + v_8 w_4 + b$$

$$m_4 = v_5 w_1 + v_6 w_2 + v_8 w_3 + v_9 w_4 + b$$

AI VIETNAM
AI Insight Course

Convolution

```python
import tensorflow as tf
import tensorflow.keras as keras
import numpy as np

conv2D = keras.layers.Conv2D(1, (2, 2), activation='relu')

data_X = np.array([[1, 1, 2],
                   [1, 2, 1],
                   [2, 1, 1]])
data_X = data_X.reshape(1, 3, 3, 1)
data_X = tf.Variable(data_X, dtype=tf.float32)

data_y = np.ones((1, 2, 2, 1))
data_y = tf.Variable(data_y, dtype=tf.float32)

print('\n\n data_X: \n', data_X[0,:,:,0])
print('\n\n data_y: \n', data_y[0,:,:,0])

with tf.GradientTape(persistent=True) as tape:
    data_after_conv = conv2D(data_X)
    print('\n\n data_after_conv: \n', data_after_conv[0,:,:,0])

    # Check the values of the current conv weight
    print('\n\n conv_weight: \n', conv2D.trainable_weights[0][:,:,0,0])
    print('\n conv_bias: \n', conv2D.trainable_weights[1][0])

    loss = tf.reduce_mean((data_after_conv-data_y)**2)
    print('\n\n loss: \n', loss)

dloss = tape.gradient(loss, data_after_conv)
print('\n\n dloss: \n', dloss[0,:,:,0])

dconv_weight = tape.gradient(loss, conv2D.trainable_weights)
print('\n\n dconv_weight: \n', dconv_weight[0][:,:,0,0])
print('\n dconv_bias: \n', dconv_weight[1][0])
```

```
data_X:
 tf.Tensor(
[[1. 1. 2.]
 [1. 2. 1.]
 [2. 1. 1.]], shape=(3, 3), dtype=float32)


data_y:
 tf.Tensor(
[[1. 1.]
 [1. 1.]], shape=(2, 2), dtype=float32)
```

```
data_after_conv:
 tf.Tensor(
[[0.9292677 1.2405912]
 [1.2405912 1.6272811]], shape=(2, 2), dtype=float32)
```

```
conv_weight:
 tf.Tensor(
[[ 0.6779961  -0.3282364 ]
 [ 0.61954254 -0.02001727]], shape=(2, 2), dtype=float32)

conv_bias:
 tf.Tensor(0.0, shape=(), dtype=float32)
```

```
loss:
 tf.Tensor(0.12856321, shape=(), dtype=float32)
```

```
dloss:
 tf.Tensor(
[[-0.03536615  0.12029558]
 [ 0.12029558  0.31364053]], shape=(2, 2), dtype=float32)
```

```
dconv_weight:
 tf.Tensor(
[[0.83250606 0.75945675]
 [0.75945675 0.4834994 ]], shape=(2, 2), dtype=float32)

dconv_bias:
 tf.Tensor(0.5188656, shape=(), dtype=float32)
```

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

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks


Thank you!