機器學習研究應用
Study for Machine Learning and Its Applications

Deep Learning for Computer Vision

孫士韋

Shih-Wei Sun

swsun@newmedia.tnua.edu.tw

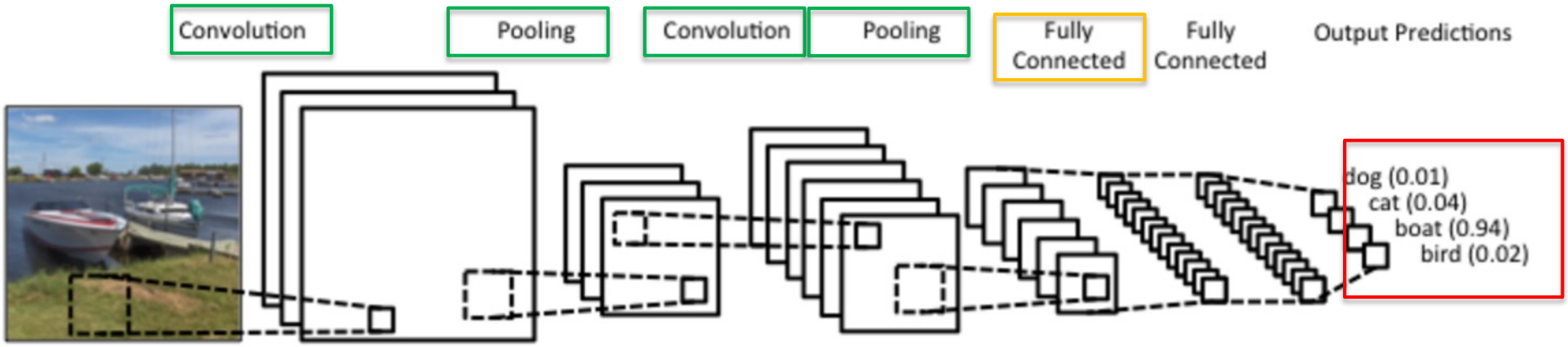
Outline

- Introduction to convnets
  - Convolution neural networks
  - Convolution operation
  - Max-pooling operation
- Training a convnet from scratch on a small dataset

Introduction to convnets

Convolutional Neural Networks

- CNN, convnets
- Deep learning model
  - Used in computer vision applications
- Image classification problems
  - Small training datasets



Introduction to convnets

- Take a practical look: a simple convnet
- Classify MNIST digits
- Instantiating a small convnet

Code:

```
1  from tensorflow.keras import layers
2  from tensorflow.keras import models
3  model = models.Sequential()
4  model.add(layers.Conv2D(32, (3, 3),
       activation='relu', input_shape=(28, 28,
       1)))
5  model.add(layers.MaxPooling2D((2, 2)))
6  model.add(layers.Conv2D(64, (3, 3),
       activation='relu'))
7  model.add(layers.MaxPooling2D((2, 2)))
8  model.add(layers.Conv2D(64, (3, 3),
       activation='relu'))


13 print(model.summary())
```

Step 1: add layers

Conv2D

input image size

MaxPooling2D

Results:

(height, width, channels)

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
None
```

28-2

26/2

13-2

11/2, int

5-2

max_pooling2D

-2 issue:

due to convolution, border effect

(32, 64): number of channels,

Controlled by the

First argument

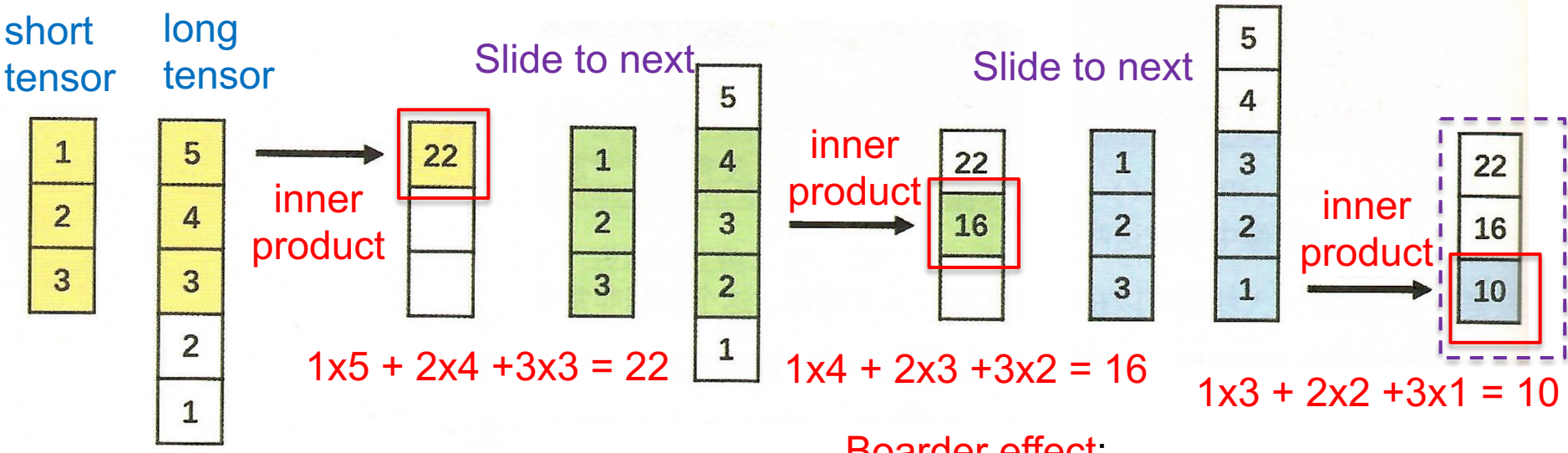
Convolution operation

- Basic concept

One kind of operations: like +, -, *, /

Convolution for two vectors:
Resulting still a vector

sliding – extracting – inner product

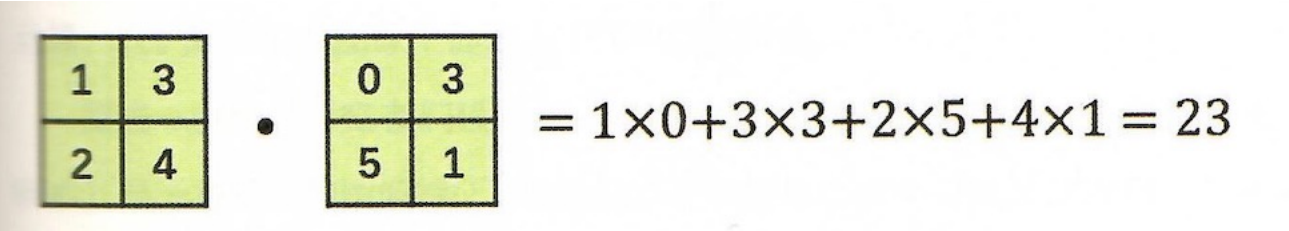


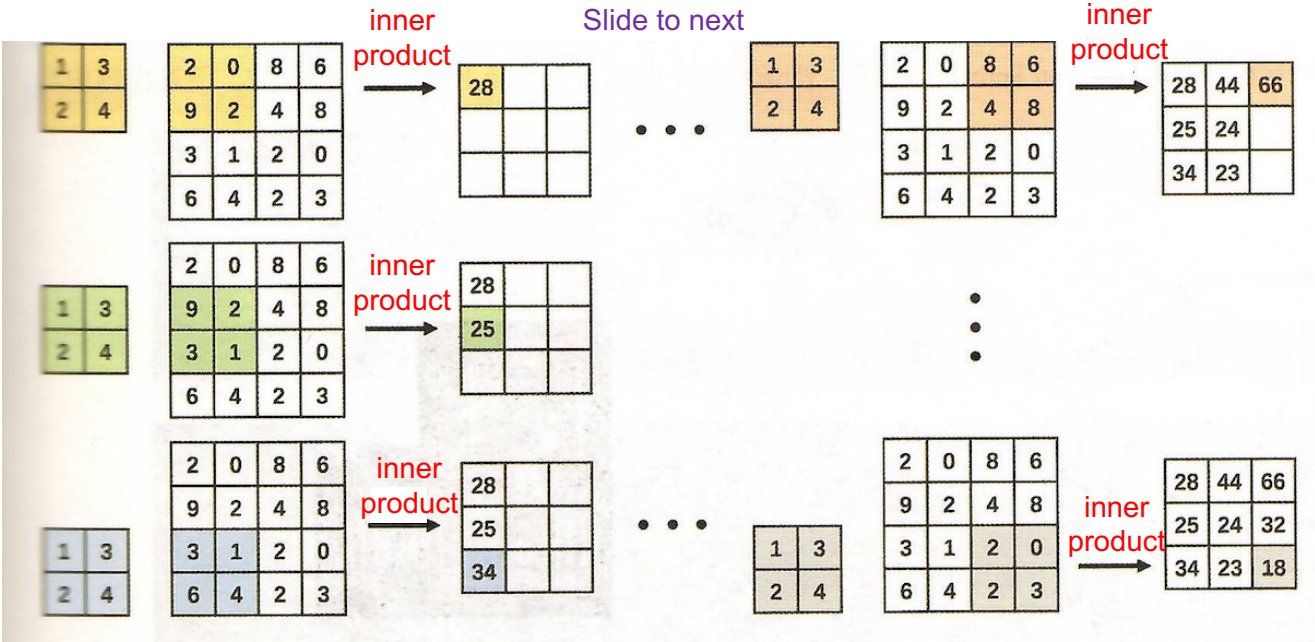
Boarder effect:
Original tensor: 5 elements
After convolution: 3 elements
5-2=3

2D Convolution

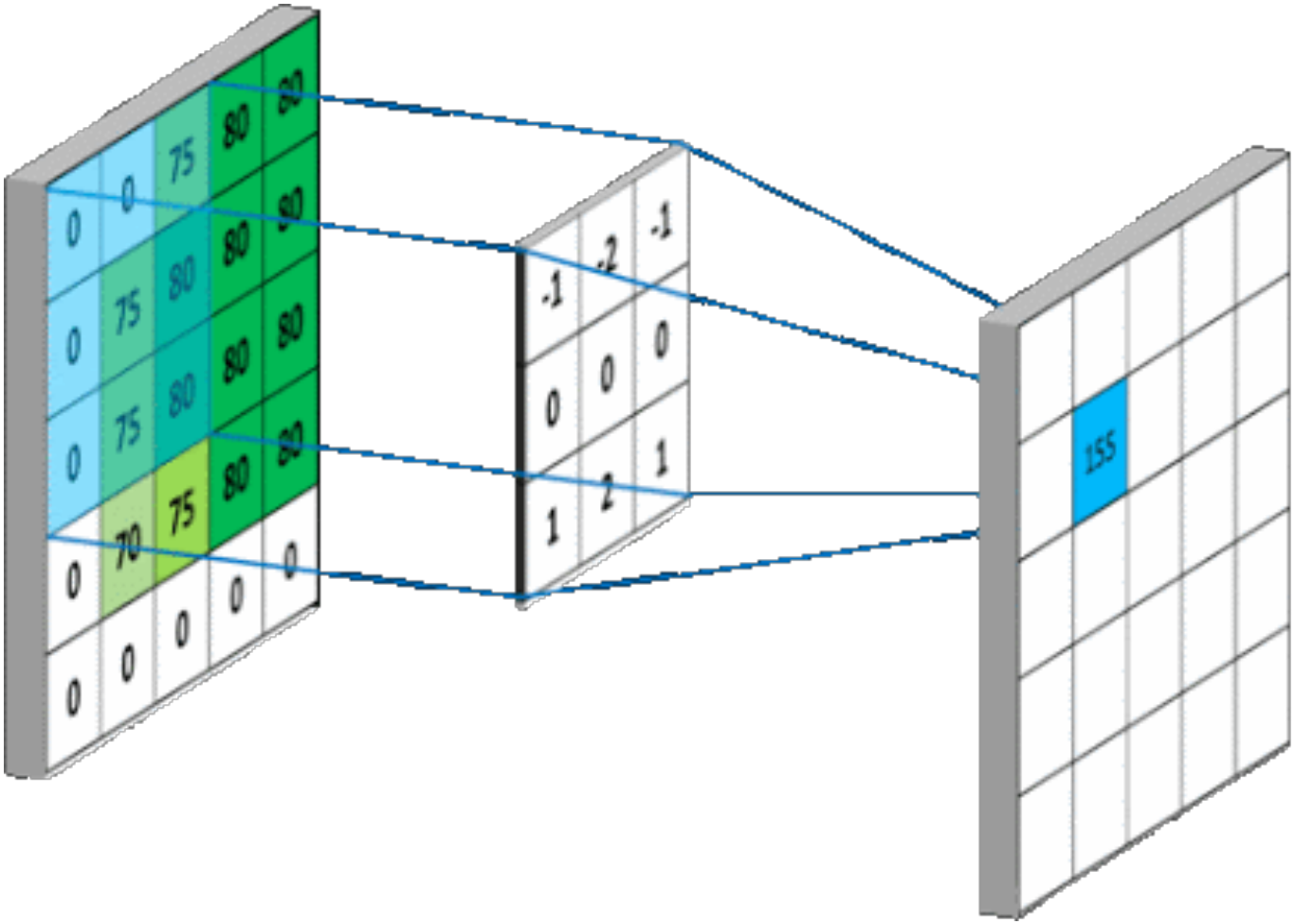
-An example of (2, 2) Conv2D

sliding – extracting – inner product





Visualization for 2D Convolution



Credit:
https://medium.com/swlh/haar-cascade-classifiers-in-opencv-explained-visually-f608086fc42c

The convolution operation (1/2)

- Why do we need convolution?
  - Dense layer:
    - Learn global patterns
  - Convolution layer:
    - Learn Local patterns
- Key property 1:
  - Translation invariant
    - No matter a local pattern moves

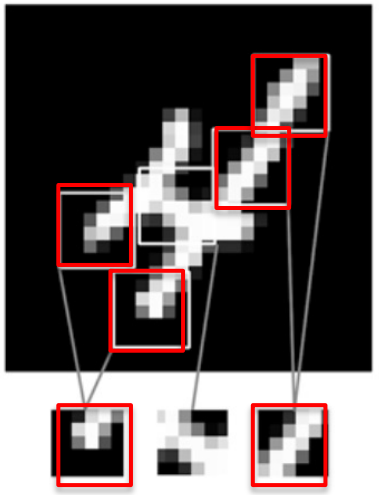
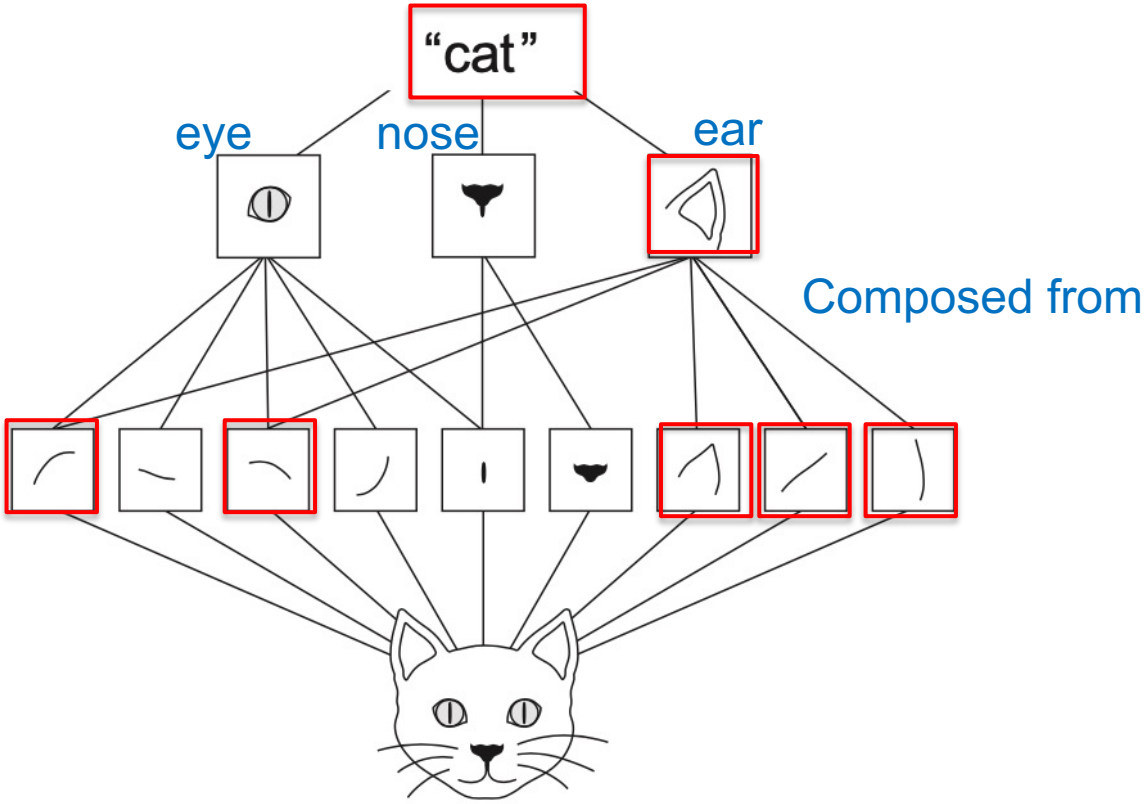


Figure 5.1 Images can be broken into local patterns such as edges, textures, and so on.

The convolution operation (2/2)

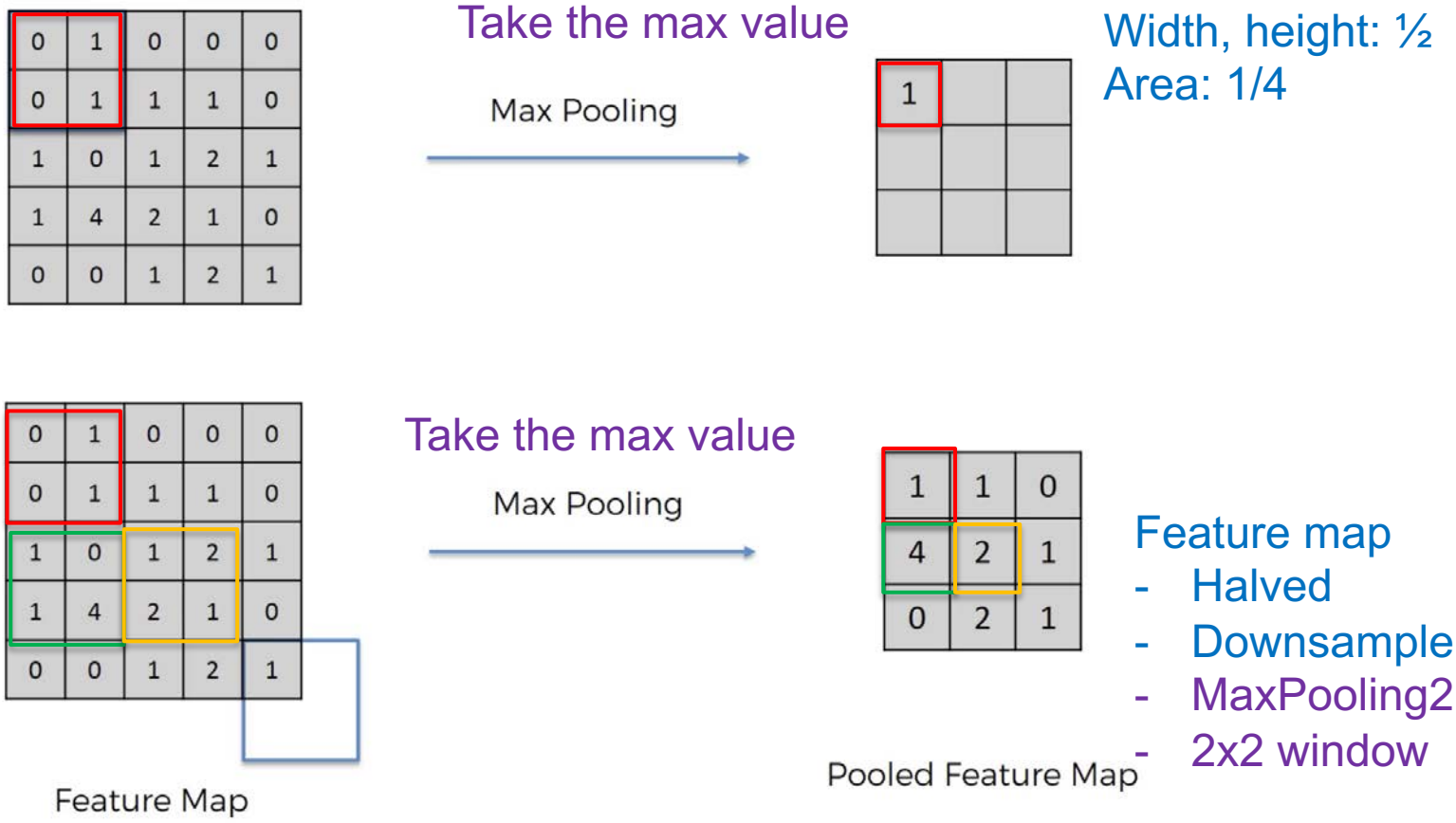
- Key property 2:
  - Learn spatial hierarchies of local patterns



Max-pooling operation

- Reduce the trained parameters
  - Reduce computations,
  - make the problem solvable

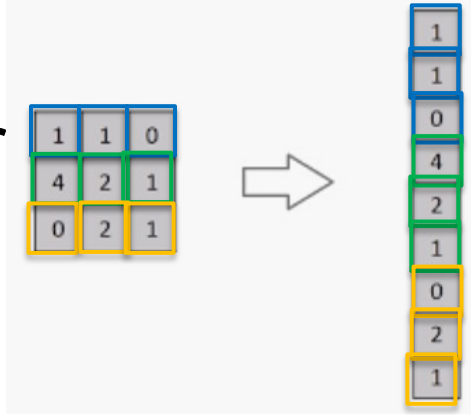
Max pooling:
-目標downsampling
卻不會損壞識別結果



Adding a classifier to the convnet

- Feed the last output tensor
  - Into a densely connected classifier
- Flatten, dense layers

Flatten:
2D tensor -> 1D tensor



Step 1: add layers

Flatten, 3D tensor to 1D tensor

Code:

```
10 model.add(layers.Flatten())
11 model.add(layers.Dense(64, activation='relu'))
12 model.add(layers.Dense(10,
       activation='softmax'))
13 print(model.summary())
```

Add 2 dense layers

Dense layer: learn global patterns

softmax:
multiclass, single-label problem
- Classified as 0~10 digits

Results:

3x3x64 = 576

```
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
None
```

2 dense layers

Training the convnet

- On MNIST images

Code:

```
15 from tensorflow.keras.datasets import mnist
16 from tensorflow.keras.utils import
       to_categorical
17
18 (train_images, train_labels), (test_images,
       test_labels) = mnist.load_data()
19
20 train_images = train_images.reshape((60000,
       28, 28, 1))
21 train_images = train_images.astype('float32')
       / 255
22 test_images = test_images.reshape((10000, 28,
       28, 1))
23 test_images = test_images.astype('float32') /
       255
24 train_labels = to_categorical(train_labels)
25 test_labels = to_categorical(test_labels)
26 model.compile(optimizer='rmsprop',
       loss='categorical_crossentropy',
       metrics=['accuracy'])
27 model.fit(train_images, train_labels,
       epochs=5, batch_size=64)
28 test_loss, test_acc =
       model.evaluate(test_images, test_labels)
29 print(test_acc)
```

Step 0: load data

Step 2: compile the model

Step 3: fit the model

Step 4: evaluate the model

Multi-class classifier: 10 classes

Results:

```
Epoch 2/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0491 - accura
cy: 0.9847
Epoch 3/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0329 - accura
cy: 0.9896
Epoch 4/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0259 - accura
cy: 0.9923
Epoch 5/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0198 - accura
cy: 0.9940
2022-02-12 17:45:44.656047: I tensorflow/core/grappler/optimizers/custom_graph_o
ptimizer_registry.cc:112] Plugin optimizer for device_type GPU is enabled.
313/313 [==============================] - 2s 7ms/step - loss: 0.0406 - accuracy
: 0.9887
0.9887000322341919
```

Training for 60,000 images,
938*64 = 60,032, close to;
for 5 epochs

Accuracy: 98.87%

Testing on 313*128
= 20, 032 samples,
2 times of 10,000 images
in 2.007 second

Training need a period of time

Testing from an image
to a *machine learning model*,
Really fast!!

Can be applied to real-time interactive apps!

The full code

```
from tensorflow.keras import layers
from tensorflow.keras import models
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3),
    activation='relu', input_shape=(28, 28,
    1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3),
    activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3),
    activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10,
    activation='softmax'))
print(model.summary())

```

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import
    to_categorical

(train_images, train_labels), (test_images,
    test_labels) = mnist.load_data()

train_images = train_images.reshape((60000,
    28, 28, 1))
train_images = train_images.astype('float32')
    / 255
test_images = test_images.reshape((10000, 28,
    28, 1))
test_images = test_images.astype('float32') /
    255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
model.compile(optimizer='rmsprop',
    loss='categorical_crossentropy',
    metrics=['accuracy'])
model.fit(train_images, train_labels,
    epochs=5, batch_size=64)
```

Practice 1

- Show the accuracy

```
Epoch 2/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0491 - accura
cy: 0.9847
Epoch 3/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0329 - accura
cy: 0.9896
Epoch 4/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0259 - accura
cy: 0.9923
Epoch 5/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0198 - accura
cy: 0.9940
2022-02-12 17:45:44.656047: I tensorflow/core/grappler/optimizers/custom_graph_o
ptimizer_registry.cc:112] Plugin optimizer for device_type GPU is enabled.
313/313 [==============================] - 2s 7ms/step - loss: 0.0406 - accuracy
: 0.9887
0.9887000322341919
```

Outline

- Introduction to convnets
- Training a convnet from scratch on a small dataset
  - Starting from a real example
    - Binary classification for dogs /cats

Training a convnet from scratch on a small dataset

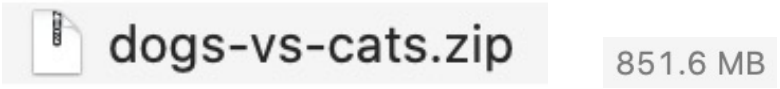
Training from a small dataset

- Download the Dogs vs. Cats dataset [2013]
  - https://www.kaggle.com/c/dogs-vs-cats/data
    - Create a free user account
    - https://www.dropbox.com/s/dbjkh21m6nyxrd4/train_cats_dogs.zip?dl=0
- 25,000 images of dogs and cats
  - 12,500 dogs
  - 12,500 cats

Figure 5.8 Samples from the Dogs vs. Cats dataset. Sizes weren't modified: the samples are heterogeneous in size, appearance, and so on.

Preprocessing - Copy images

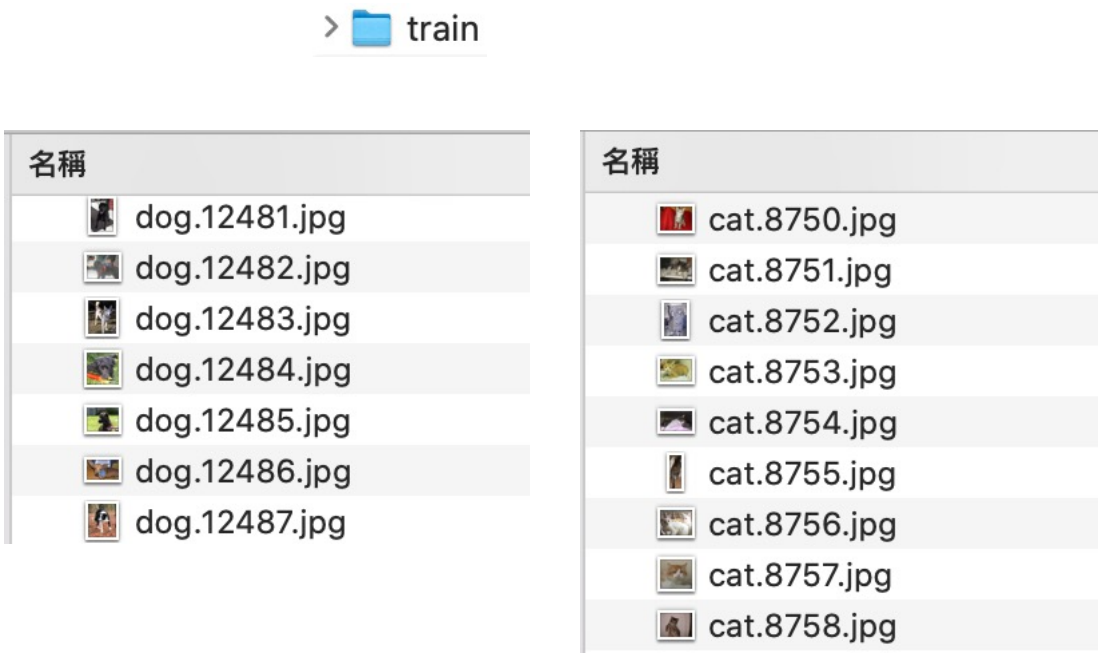
- After downloading
  - Uncompressing it



- Train your model on less than
  - 10% of the data
- To training, validation, and test directories
  - 1,000 samples/class: training
  - 500 samples/class: validation set

Uncompressing the files

- Uncompressing the zip file
- Rename the folder: train



Setup the directories

- Assigning for
  - the original file folder: images of *dogs/cats*
  - the a smaller dataset folder:
    - 1000 samples/training, 500 samples/class: validation

```
1  import os, shutil
2
3  # The path to the directory where the
       original
4  # dataset was uncompressed
5  original_dataset_dir =
       '/Users/swsun/Downloads/train'
6
7  # The directory where we will
8  # store our smaller dataset
9  base_dir =
       '/Users/swsun/Downloads/cats_and_dogs_s
       mall'
10 os.mkdir(base_dir)
```

Make a new directory

Copy the files (1/2)

```
# Directories for our training,
# validation and test splits
train_dir = os.path.join(base_dir, 'train')
os.mkdir(train_dir)
validation_dir = os.path.join(base_dir,
    'validation')
os.mkdir(validation_dir)
test_dir = os.path.join(base_dir, 'test')
os.mkdir(test_dir)
```

Making folders of sets:
train/
validation/
test

```
# Directory with our training cat pictures
train_cats_dir = os.path.join(train_dir,
    'cats')
os.mkdir(train_cats_dir)

# Directory with our training dog pictures
train_dogs_dir = os.path.join(train_dir,
    'dogs')
os.mkdir(train_dogs_dir)
```

training set:
cats/dogs

```
# Directory with our validation cat
    pictures
validation_cats_dir =
    os.path.join(validation_dir, 'cats')
os.mkdir(validation_cats_dir)

# Directory with our validation dog
    pictures
validation_dogs_dir =
    os.path.join(validation_dir, 'dogs')
os.mkdir(validation_dogs_dir)
```

validation set:
cats/dogs

```
# Directory with our validation cat
    pictures
test_cats_dir = os.path.join(test_dir,
    'cats')
os.mkdir(test_cats_dir)

# Directory with our validation dog
    pictures
test_dogs_dir = os.path.join(test_dir,
    'dogs')
os.mkdir(test_dogs_dir)
```

test set:
cats/dogs

Copy 1000 cats images to training set

```
# Copy first 1000 cat images to
    train_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in
    range(1000)]
for fname in fnames:
    src =
        os.path.join(original_dataset_dir,
        fname)
    dst = os.path.join(train_cats_dir,
        fname)
    shutil.copyfile(src, dst)
```

```
# Copy next 500 cat images to
    validation_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in
    range(1000, 1500)]
for fname in fnames:
    src =
        os.path.join(original_dataset_dir,
        fname)
    dst =
        os.path.join(validation_cats_dir,
        fname)
    shutil.copyfile(src, dst)
```

Copy 500 cats images to validation set

```
# Copy next 500 cat images to test_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in
    range(1500, 2000)]
for fname in fnames:
    src =
        os.path.join(original_dataset_dir,
        fname)
    dst = os.path.join(test_cats_dir,
        fname)
    shutil.copyfile(src, dst)
```

Copy 500 cats images to test set

```
# Copy first 1000 dog images to
    train_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in
    range(1000)]
for fname in fnames:
    src =
        os.path.join(original_dataset_dir,
        fname)
    dst = os.path.join(train_dogs_dir,
        fname)
    shutil.copyfile(src, dst)
```

Copy 1000 dogs images to training set

Copy the files (2/2)

Copy 500 dogss images to validation set

```
72
73  # Copy next 500 dog images to validation_dogs_dir
74  fnames = ['dog.{}.jpg'.format(i) for i in
        range(1000, 1500)]
75  for fname in fnames:
76      src = os.path.join(original_dataset_dir,
            fname)
77      dst = os.path.join(validation_dogs_dir,
            fname)
78      shutil.copyfile(src, dst)
79
```

```
80  # Copy next 500 dog images to test_dogs_dir
81  fnames = ['dog.{}.jpg'.format(i) for i in
        range(1500, 2000)]
82  for fname in fnames:
83      src = os.path.join(original_dataset_dir,
            fname)
84      dst = os.path.join(test_dogs_dir, fname)
85      shutil.copyfile(src, dst)
86
```

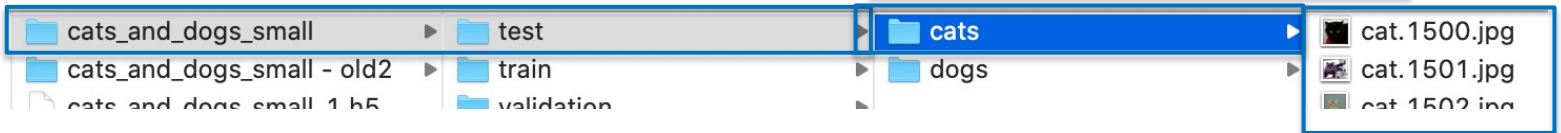
Copy 500 dogs images to test set

Display the number images in Each directories

```
87  ## leave as a file for download
88  print('total training cat images:',
        len(os.listdir(train_cats_dir)))
89  print('total training dog images:',
        len(os.listdir(train_dogs_dir)))
90  print('total validation cat images:',
        len(os.listdir(validation_cats_dir)))
91  print('total validation dog images:',
        len(os.listdir(validation_dogs_dir)))
92  print('total test cat images:',
        len(os.listdir(test_cats_dir)))
93  print('total test dog images:',
        len(os.listdir(test_dogs_dir)))
```

Results:

```
total training cat images: 1000
total training dog images: 1000
total validation cat images: 500
total validation dog images: 500
total test cat images: 500
total test dog images: 500
```



Comment the codes

```
import os, shutil

# The path to the directory where the original
# dataset was uncompressed
original_dataset_dir =
    '/Users/swsun/Downloads/train'

# The directory where we will
# store our smaller dataset
base_dir =
    '/Users/swsun/Downloads/cats_and_dogs_small'
#os.mkdir(base_dir)

# Directories for our training,
# validation and test splits
train_dir = os.path.join(base_dir, 'train')
#os.mkdir(train_dir)
validation_dir = os.path.join(base_dir,
    'validation')
#os.mkdir(validation_dir)
test_dir = os.path.join(base_dir, 'test')
#os.mkdir(test_dir)


# Directory with our training cat pictures
train_cats_dir = os.path.join(train_dir, 'cats')
#os.mkdir(train_cats_dir)

# Directory with our training dog pictures
train_dogs_dir = os.path.join(train_dir, 'dogs')
#os.mkdir(train_dogs_dir)

# Directory with our validation cat pictures
validation_cats_dir =
    os.path.join(validation_dir, 'cats')
#os.mkdir(validation_cats_dir)
```

Comment for: *mkdir*

```

# Directory with our validation dog pictures
validation_dogs_dir =
    os.path.join(validation_dir, 'dogs')
#os.mkdir(validation_dogs_dir)

# Directory with our validation cat pictures
test_cats_dir = os.path.join(test_dir, 'cats')
#os.mkdir(test_cats_dir)

# Directory with our validation dog pictures
test_dogs_dir = os.path.join(test_dir, 'dogs')
#os.mkdir(test_dogs_dir)
```

```
'''
# Copy first 1000 cat images to train_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in
    range(1000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir,
        fname)
    dst = os.path.join(train_cats_dir, fname)
    shutil.copyfile(src, dst)

# Copy next 500 cat images to validation_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in
    range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_dataset_dir,
        fname)
    dst = os.path.join(validation_cats_dir,
        fname)
    shutil.copyfile(src, dst)

# Copy next 500 cat images to test_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in
    range(1500, 2000)]
```

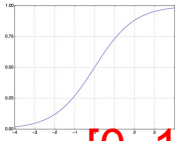
Comment for: multiple lines

```
for fname in fnames:
    src = os.path.join(original_dataset_dir,
        fname)
    dst = os.path.join(test_cats_dir, fname)
    shutil.copyfile(src, dst)

# Copy first 1000 dog images to train_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in
    range(1000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir,
        fname)
    dst = os.path.join(train_dogs_dir, fname)
    shutil.copyfile(src, dst)

# Copy next 500 dog images to validation_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in
    range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_dataset_dir,
        fname)
    dst = os.path.join(validation_dogs_dir,
        fname)
    shutil.copyfile(src, dst)


# Copy next 500 dog images to test_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in
    range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir,
        fname)
    dst = os.path.join(test_dogs_dir, fname)
    shutil.copyfile(src, dst)

## leave as a file for download
print('total training cat images:',
    len(os.listdir(train_cats_dir)))
print('total training dog images:',
    len(os.listdir(train_dogs_dir)))
print('total validation cat images:',
    len(os.listdir(validation_cats_dir)))
print('total validation dog images:',
    len(os.listdir(validation_dogs_dir)))
print('total test cat images:',
    len(os.listdir(test_cats_dir)))
print('total test dog images:',
    len(os.listdir(test_dogs_dir)))
'''
```

Step 0: Load Data → Step 1: Define Model → Step 2 : Compile Model → Step 3: Fit Model → Step 4: Evaluate Model

Build your network

Problem type	Last-layer activation
Binary classification	sigmoid



[0, 1]

Step1: add layers

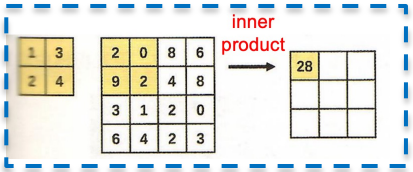
```
from tensorflow.keras import layers
from tensorflow.keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3),
    activation='relu', input_shape=(150, 150,
    3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3),
    activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3),
    activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3),
    activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

print(model.summary())
```

2 dense layers

'sigmoid' activation:

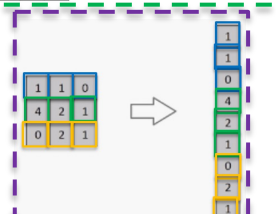
Binary classification: dog / cat

Conv2D (3,3)



MaxPooling2D (2,2)

Flatten()



Results:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

Configuring the model

- Compile the model

```
113
114 from tensorflow.keras import optimizers
115
116 model.compile(loss='binary_crossentropy',
        optimizer=optimizers.RMSprop(lr=1e-4),
        metrics=['acc'])
117
```

Step 2: compile the model

Assign the loss function: binary_crossentropy

Optimizer: RMSprop, as usual

Table 4.1 Choosing the right last-layer activation and loss function for your model

Problem type	Last-layer activation	Loss function
Binary classification *dog / cat*	`sigmoid`	`binary_crossentropy`

Results:

```
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)
_________________________________________________________________
dense_1 (Dense)              (None, 512)
_________________________________________________________________
dense_2 (Dense)              (None, 1)
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

Nothing changed

Data Preprocessing

```
118 from tensorflow.keras.preprocessing.image import
        ImageDataGenerator
119
120 # All images will be rescaled by 1./255
121 train_datagen =
        ImageDataGenerator(rescale=1./255)
122 test_datagen = ImageDataGenerator(rescale=1./255)
123
124 train_generator =
        train_datagen.flow_from_directory(
        train_dir, target_size=(150, 150),
        batch_size=20, class_mode='binary')
125
126 validation_generator =
        test_datagen.flow_from_directory(
        validation_dir, target_size=(150, 150),
        batch_size=20, class_mode='binary')
127
128 for data_batch, labels_batch in train_generator:
129     print('data batch shape:', data_batch.shape)
130     print('labels batch shape:',
            labels_batch.shape)
131     break
```

Preprocessing:

- Read the pictures
- Decode from JPEG to RGB
- Convert to floating point tensors
- Rescale from [0, 255] to [0,1]

Rescale by 1/255

ImageDataGenerator

training set

Resize to 150 x 150

Binary labels: dog/cat

validation set

Results:

Display Generator results

150, 150: image tensors

```
None
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
data batch shape: (20, 150, 150, 3)
labels batch shape: (20,)
```

20: batch size

Fitting and saving the model

- Fit the model
- Save the model!

instead of model.fit

Pass argument:
train_generator, validataion_generator

```
132
133 history = model.fit_generator(train_generator,
        steps_per_epoch=100, epochs=30,
        validation_data=validation_generator,
        validation_steps=50)
134 model
        .save
        ('/Users/swsun/Downloads/cats_and_dogs_small_
        1.h5')
135
```

Results:

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
data batch shape: (20, 150, 150, 3)
labels batch shape: (20,)
Epoch 1/30
100/100 [==============================] - 37s 373m
0.5155 - val_loss: 0.6832 - val_acc: 0.5000
Epoch 2/30
100/100 [==============================] - 40s 402m
0.5850 - val_loss: 0.6617 - val_acc: 0.6370
Epoch 3/30
100/100 [==============================] - 40s 402m
0.6530 - val_loss: 0.7222 - val_acc: 0.6510
Epoch 4/30
 31/100 [========>.....................] - ETA: 22s
```

Save the model after training

cats_and_dogs_small_1.h5

Training cost a lot of time,
Don‘t forget to save it!
- To be used by other interactive applications!

```
Epoch 27/30
100/100 [==============================] -
0.9790 - val_loss: 0.0939 - val_acc: 0.7480
Epoch 28/30
100/100 [==============================] -
9855 - val_loss: 1.1271 - val_acc: 0.7400
Epoch 29/30
100/100 [==============================] -
0.9865 - val_loss: 0.3695 - val_acc: 0.7320
Epoch 30/30
100/100 [==============================] -
0.9920 - val_loss: 0.7260 - val_acc: 0.7370
```

Practice 2

- Display the accuracy of
  - Training data
  - Validation data

```
Epoch 27/30
100/100 [==============================] - 36s 359ms/step - loss: 0.0683 - acc:
0.9790 - val_loss: 0.0939 - val_acc: 0.7480
Epoch 28/30
100/100 [==============================] - 322s 3s/step - loss: 0.0559 - acc: 0.
9855 - val_loss: 1.1271 - val_acc: 0.7400
Epoch 29/30
100/100 [==============================] - 1760s 18s/step - loss: 0.0501 - acc:
0.9865 - val_loss: 0.3695 - val_acc: 0.7320
Epoch 30/30
100/100 [==============================] - 44s 437ms/step - loss: 0.0423 - acc:
0.9920 - val_loss: 0.7260 - val_acc: 0.7370
```

Plot the results

```
136 import matplotlib.pyplot as plt
137
138 acc = history.history['acc']
139 val_acc = history.history['val_acc']
140 loss = history.history['loss']
141 val_loss = history.history['val_loss']
142
143 epochs = range(len(acc))
144
145 plt.plot(epochs, acc, 'bo', label='Training acc')
146 plt.plot(epochs, val_acc, 'b', label='Validation
        acc')
147 plt.title('Training and validation accuracy')
148 plt.legend()
149
150 plt.figure()
151
152 plt.plot(epochs, loss, 'bo', label='Training
        loss')
153 plt.plot(epochs, val_loss, 'b',
        label='Validation loss')
154 plt.title('Training and validation loss')
155 plt.legend()
156
157 plt.show()
```

Accuracy:
Training / validation

Overfitting issue:
- Training accuracy – 100%
- Validation accuracy – 70%
- after 5 epochs

Too few samples: 2000 images

loss:
Training / validation

Overfitting issue:

Improvement: Dropout,
weight decay (L2 regularization),
Reduce network size…

Full codes (1/3)

```python
import os, shutil

# The path to the directory where the original
# dataset was uncompressed
original_dataset_dir =
    '/Users/swsun/Downloads/train'

# The directory where we will
# store our smaller dataset
base_dir =
    '/Users/swsun/Downloads/cats_and_dogs_small'
#os.mkdir(base_dir)

# Directories for our training,
# validation and test splits
train_dir = os.path.join(base_dir, 'train')
#os.mkdir(train_dir)
validation_dir = os.path.join(base_dir,
    'validation')
#os.mkdir(validation_dir)
test_dir = os.path.join(base_dir, 'test')
#os.mkdir(test_dir)


# Directory with our training cat pictures
train_cats_dir = os.path.join(train_dir, 'cats')
#os.mkdir(train_cats_dir)

# Directory with our training dog pictures
train_dogs_dir = os.path.join(train_dir, 'dogs')
#os.mkdir(train_dogs_dir)

# Directory with our validation cat pictures
validation_cats_dir =
    os.path.join(validation_dir, 'cats')
#os.mkdir(validation_cats_dir)


# Directory with our validation dog pictures
validation_dogs_dir =
    os.path.join(validation_dir, 'dogs')
#os.mkdir(validation_dogs_dir)

# Directory with our validation cat pictures
test_cats_dir = os.path.join(test_dir, 'cats')
#os.mkdir(test_cats_dir)

# Directory with our validation dog pictures
test_dogs_dir = os.path.join(test_dir, 'dogs')
#os.mkdir(test_dogs_dir)


'''
# Copy first 1000 cat images to train_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in
    range(1000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir,
        fname)
    dst = os.path.join(train_cats_dir, fname)
    shutil.copyfile(src, dst)

# Copy next 500 cat images to validation_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in
    range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_dataset_dir,
        fname)
    dst = os.path.join(validation_cats_dir,
        fname)
    shutil.copyfile(src, dst)

# Copy next 500 cat images to test_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in
    range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir,
        fname)
    dst = os.path.join(test_cats_dir, fname)
    shutil.copyfile(src, dst)

# Copy first 1000 dog images to train_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in
    range(1000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir,
        fname)
    dst = os.path.join(train_dogs_dir, fname)
    shutil.copyfile(src, dst)

# Copy next 500 dog images to validation_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in
    range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_dataset_dir,
        fname)
    dst = os.path.join(validation_dogs_dir,
        fname)
    shutil.copyfile(src, dst)


# Copy next 500 dog images to test_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in
    range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir,
        fname)
    dst = os.path.join(test_dogs_dir, fname)
    shutil.copyfile(src, dst)

## leave as a file for download
print('total training cat images:',
    len(os.listdir(train_cats_dir)))
print('total training dog images:',
    len(os.listdir(train_dogs_dir)))
print('total validation cat images:',
    len(os.listdir(validation_cats_dir)))
print('total validation dog images:',
    len(os.listdir(validation_dogs_dir)))
print('total test cat images:',
    len(os.listdir(test_cats_dir)))
print('total test dog images:',
    len(os.listdir(test_dogs_dir)))
'''
```

Full code (2/3)

```
95
96  from tensorflow.keras import layers
97  from tensorflow.keras import models
98
99  model = models.Sequential()
100 model.add(layers.Conv2D(32, (3, 3),
        activation='relu', input_shape=(150, 150,
        3)))
101 model.add(layers.MaxPooling2D((2, 2)))
102 model.add(layers.Conv2D(64, (3, 3),
        activation='relu'))
103 model.add(layers.MaxPooling2D((2, 2)))
104 model.add(layers.Conv2D(128, (3, 3),
        activation='relu'))
105 model.add(layers.MaxPooling2D((2, 2)))
106 model.add(layers.Conv2D(128, (3, 3),
        activation='relu'))
107 model.add(layers.MaxPooling2D((2, 2)))
108 model.add(layers.Flatten())
109 model.add(layers.Dense(512, activation='relu'))
110 model.add(layers.Dense(1, activation='sigmoid'))
111
112 print(model.summary())
113
114  from tensorflow.keras import optimizers
115
116  model.compile(loss='binary_crossentropy',
         optimizer=optimizers.RMSprop(lr=1e-4),
         metrics=['acc'])
117
113
114 from tensorflow.keras import optimizers
115
116 model.compile(loss='binary_crossentropy',
        optimizer=optimizers.RMSprop(lr=1e-4),
        metrics=['acc'])
117
```

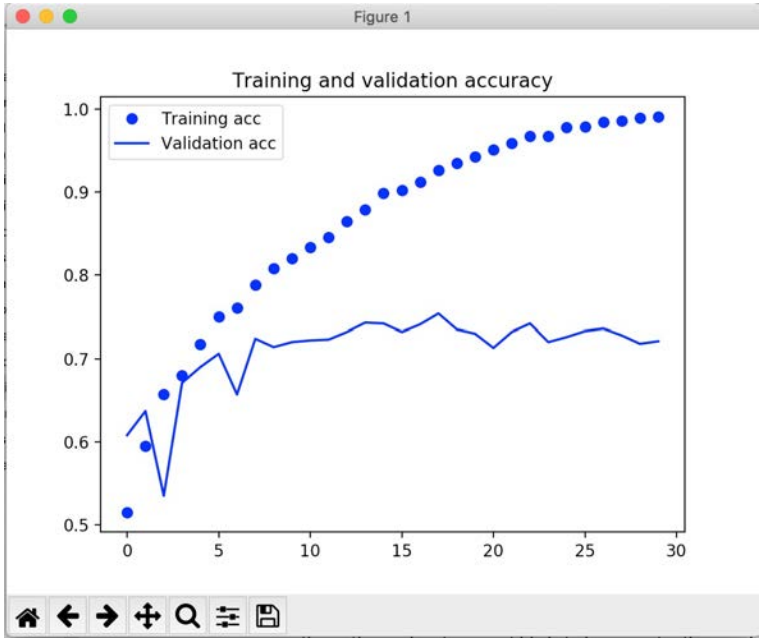
```
118 from tensorflow.keras.preprocessing.image import
        ImageDataGenerator
119
120 # All images will be rescaled by 1./255
121 train_datagen =
        ImageDataGenerator(rescale=1./255)
122 test_datagen = ImageDataGenerator(rescale=1./255)
123
124 train_generator =
        train_datagen.flow_from_directory(
        train_dir, target_size=(150, 150),
        batch_size=20, class_mode='binary')
125
126 validation_generator =
        test_datagen.flow_from_directory(
        validation_dir, target_size=(150, 150),
        batch_size=20, class_mode='binary')
127
128 for data_batch, labels_batch in train_generator:
129     print('data batch shape:', data_batch.shape)
130     print('labels batch shape:',
            labels_batch.shape)
131     break
132
133 history = model.fit_generator(train_generator,
        steps_per_epoch=100, epochs=30,
        validation_data=validation_generator,
        validation_steps=50)
134 model
        .save
        ('/Users/swsun/Downloads/cats_and_dogs_small_
        1.h5')
135
```

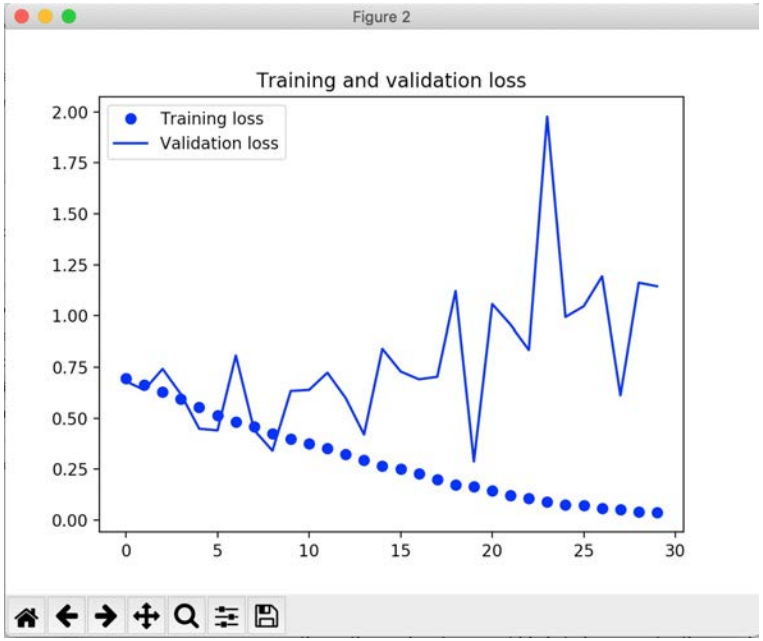
Full code (2/3)

```
136 import matplotlib.pyplot as plt
137
138 acc = history.history['acc']
139 val_acc = history.history['val_acc']
140 loss = history.history['loss']
141 val_loss = history.history['val_loss']
142
143 epochs = range(len(acc))
144
145 plt.plot(epochs, acc, 'bo', label='Training acc')
146 plt.plot(epochs, val_acc, 'b', label='Validation
        acc')
147 plt.title('Training and validation accuracy')
148 plt.legend()
149
150 plt.figure()
151
152 plt.plot(epochs, loss, 'bo', label='Training
        loss')
153 plt.plot(epochs, val_loss, 'b',
        label='Validation loss')
154 plt.title('Training and validation loss')
155 plt.legend()
```

Practice 3

- Plot the results
  - Accuracy: training / validation
  - Loss: training / validation

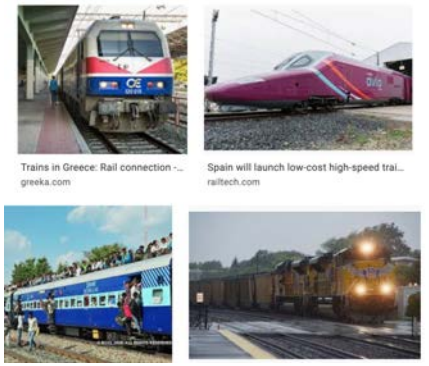




Homework 1

- Implementing a binary classification example
  - apples
  - bananas

- Implementing a multiclass classification example
  - cars, airplanes, trains, …

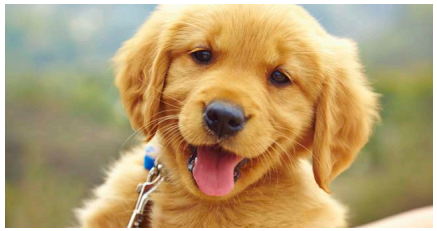


Outline

- Introduction to convnets
- Training a convnet from scratch on a small dataset
- Using a pretrained convnet

Using a pretrained convnet

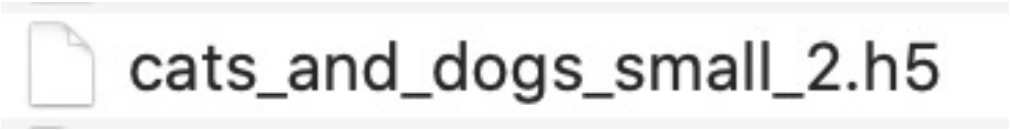
- According to a model
  - pretrained model previously
  - Pretrained from others
- 4 steps to predict the possible class
  - Step 1: Using a trained model
  - Step 2: Load an image
  - Step 3: Convert the image
  - Step 4: Predict the class



It’s a Dog!

Step 1: Using a trained model

- Load the convnet (CNN) model generated
  - previously



  - Find out the .h5 file (model) you generated

```
1  import os, shutil
2  from tensorflow.keras.models import
       load_model
3  from tensorflow.keras.preprocessing.image
       import load_img
4  from tensorflow.keras.preprocessing.image
       import img_to_array
5
6
7  # load the model
8  model =
       load_model
       ('/Users/swsun/Downloads/cats_and_dogs_
       small_1.h5')
9
```

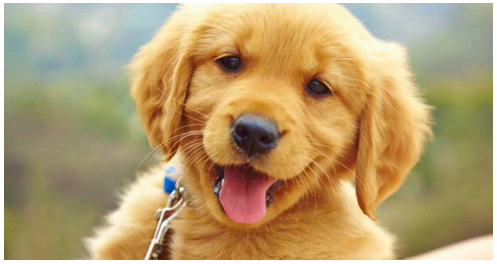
import necessary items

Load the model

assign the location of the file

Step 2: Load an image

- Load the image file



assign the name of the image file

```
10 #load the iamge
11 file_name='/Users/sunshih-wei/Documents/python/
       dog.jpg'
12 image = load_img(file_name,
       target_size=(150,150))
```

Load the image

- Resize the image
  - to the proper size:
  - (150, 150): what you trained before

Step 3: Convert the image

- Convert the image file
  - To a proper tensor (array)

```
14  #image to array, reshape
15  image = img_to_array(image)
16  image = image.reshape((1, image.shape[0],
        image.shape[1], image.shape[2]))
17
```

Image to array: img_to_array

1-st D

2-nd D

3-rd D

4-th D

- Reshape the tensor
  - as a 4D tensor

(1, 150, 150, 3)

Step 4: Predict the class

- Using the model's method
  - model: an object
  - Predict which class it belongs

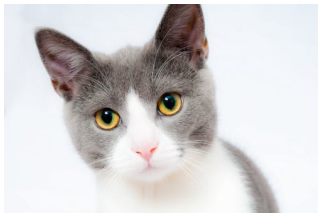
Predict the class

```
18 #predict the class: 0 is cat, 1 is dog
19 print(model.predict(image))
```

Results:

```
StreamExecutor device (0): Host, Default Version
[[1]]
```

- Binary classification
  - 2 classes: 0 or 1
    - 0: cat
    - 1: dog
      - or inverted



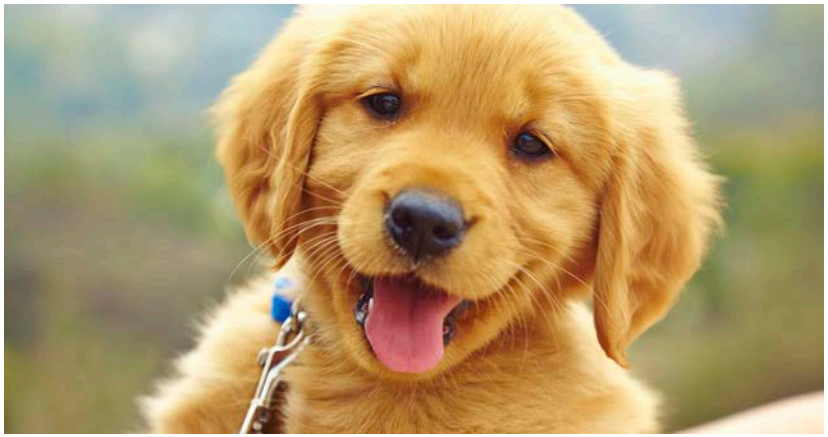
```
StreamExecutor device (0): Host, Default Version
[[0]]
```

Full code

```
import os, shutil
from tensorflow.keras.models import
    load_model
from tensorflow.keras.preprocessing.image
    import load_img
from tensorflow.keras.preprocessing.image
    import img_to_array


# load the model
model =
    load_model
    ('/Users/swsun/Downloads/cats_and_dogs_
    small_1.h5')

#load the iamge
file_name='/Users/sunshih-wei/Documents/python/
    dog.jpg'
image = load_img(file_name,
    target_size=(150,150))


#image to array, reshape
image = img_to_array(image)
image = image.reshape((1, image.shape[0],
    image.shape[1], image.shape[2]))

#predict the class: 0 is cat, 1 is dog
print(model.predict(image))
```

Practice 1

- Predict the photos
  - By your own model
- Display: dog / cat
  - Hint:
    - if … 0, else …
- Download other photos



It‘s a dog!

It‘s a cat!

Textbook Reading

- *Deep Learning with Python*
  - Ch 8, Introduction to deep learning for computer vision
    - 8.1, Introduction to convnets
    - 8.2 Training a convnet from scratch on a small dataset
  - p. 201 - p. 224