Import 必要模組

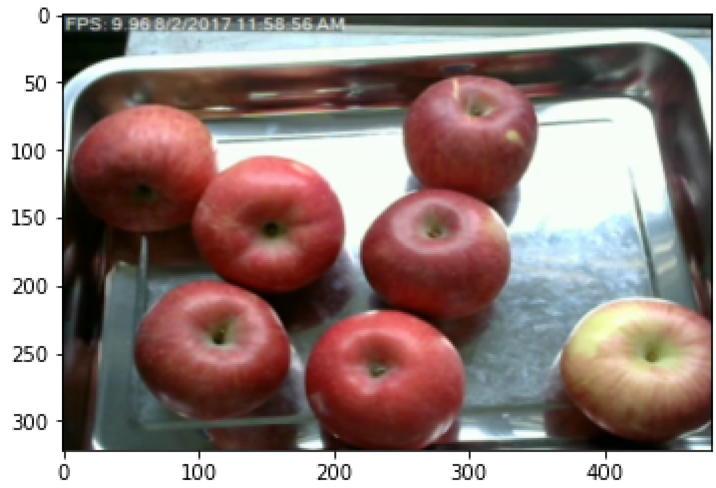
In [59]:
```
import tensorflow as tf
from tensorflow import keras
from keras.preprocessing import image
from tensorflow.keras import Sequential

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from tqdm import tqdm
```

data

- 注意檔案所在的目錄

In [63]:
```
# 輸入檔名檢查圖像
path = "c:/111/dog_vs_cat/"
# 類別名稱即為資料夾的名稱
CATEGORIES = ["dogs", "cats"]
# dogs 類別
directory = os.path.join(path,CATEGORIES[0])
# 第一筆資料
img = image.load_img(directory+"/apple001.jpg")
plt.imshow(img)
```

Out[63]: <matplotlib.image.AxesImage at 0x1a812095a30>



In [64]:
```
# CV2 讀取圖像檔轉為數字
img_array=cv2.imread(directory+"/apple001.jpg")
# 縱向 499 個像素
print (len(img_array))
 # 橫向 327 個像素
print (len(img_array[0]))
# 每個像素有三個數值
print (len(img_array[0][0]))
# 第一個像素的三個數值，代表 R G B 三個顏色
print (img_array[0][0])
# shape
print (img_array.shape)
```

322

```
480
3
[61 63 56]
(322, 480, 3)
```

In [65]:

```
# 各像表數值代表顏色組成
img_array
```

Out[65]:

```
array([[[ 61,  63,  56],
        [ 58,  60,  53],
        [ 60,  65,  52],
        ...,
        [ 67,  72,  59],
        [ 66,  70,  58],
        [ 66,  70,  58]],

       [[ 61,  63,  56],
        [ 55,  58,  51],
        [ 62,  60,  48],
        ...,
        [ 65,  69,  56],
        [ 66,  70,  58],
        [ 66,  70,  58]],

       [[ 60,  62,  55],
        [ 58,  60,  53],
        [ 68,  58,  42],
        ...,
        [ 62,  67,  54],
        [ 63,  68,  55],
        [ 63,  68,  55]],

       ...,

       [[  9,  16,  15],
        [  9,  16,  15],
        [  9,  16,  15],
        ...,
        [225, 229, 201],
        [217, 221, 193],
        [214, 217, 189]],

       [[  9,  16,  15],
        [  9,  16,  15],
        [  9,  16,  15],
        ...,
        [254, 255, 237],
        [250, 254, 233],
        [247, 252, 231]],

       [[  9,  16,  15],
        [  9,  16,  15],
        [  9,  16,  15],
        ...,
        [250, 252, 239],
        [249, 253, 233],
        [249, 253, 233]]], dtype=uint8)
```

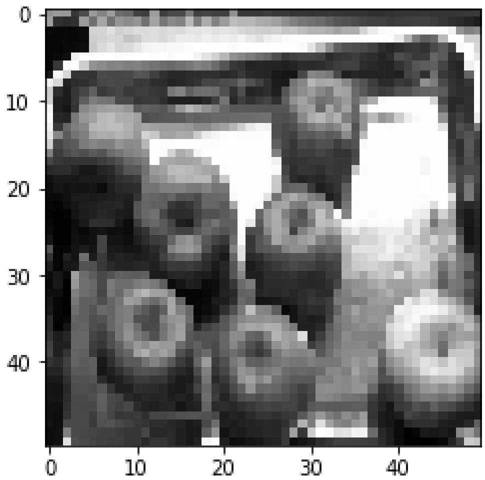
資料前處理

- size 縮小為 50*50
- 取黑白灰階

In [66]:

```
# img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)
IMG_SIZE = 50
img_array=cv2.imread(directory+"/apple001.jpg",cv2.IMREAD_GRAYSCALE)
```

```
new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
plt.imshow(new_array, cmap='gray')
plt.show()
```



In [67]:
```
new_array
```

Out[67]:
```
array([[145, 152, 150, ...,  55,  57,  59],
       [157,  50,  43, ...,  43,  38,  29],
       [ 10,   9,  10, ..., 146, 129, 137],
       ...,
       [ 19,  16,  74, ...,  30,  41,  96],
       [ 13,  17, 126, ..., 133, 155, 117],
       [ 13,  18, 223, ...,  65, 202, 194]], dtype=uint8)
```

In [68]:
```
# training_data
path = "c:/111/dog_vs_cat/"
# 類別，與資料夾名稱同
CATEGORIES = ["dogs", "cats"]
# 圖片大小設定
IMG_SIZE=50
IMG_SIZE-50
training_data = []
def create_training_data():
    for category in CATEGORIES:  # do dogs and cats
        directory = os.path.join(path,category)  # create path to dogs and cats
        class_num = CATEGORIES.index(category)  #  0=dog 1=cat
        # 讀取圖像，數字轉換，調整大小，只取灰階
        for img in tqdm(os.listdir(directory)):
            try:
                # os.path.join(directory,img) 即 directory+"/"+img
                img_array = cv2.imread(os.path.join(directory,img) ,cv2.IMREAD_GRAYS
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to
                # training data 是一個list, 每個元素，
                # 由 features 與 label 構成，label 是數字代碼
                training_data.append([new_array, class_num])
            except Exception as e:  # in the interest in keeping the output clean...
                pass
            #except OSError as e:
            #    print("OSErrroBad img most likely", e, os.path.join(path,img))
            #except Exception as e:
            #    print("general exception", e, os.path.join(path,img))

create_training_data()
# 總共筆數
print(len(training_data))
```

```
100%|████████████████████████████████████████
█| 692/692 [00:07<00:00, 97.90it/s]
```

```
100%|██████████████████████████████████████████████████|
2171/2171 [00:23<00:00, 92.63it/s]
2863
```

In [69]:
```
# 重新排序
import random
random.shuffle(training_data)
```

In [70]:
```
# features and label
X = []
y = []
for features,label in training_data:
    X.append(features)
    y.append(label)

# 重組 張量的 shape, -1 代表任意數，由電腦計算得之
# X 是 features
X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE)
# y 是label, 數字代碼
y=np.array(y).reshape(-1, 1)
```

In [71]:
```
X.shape
```

Out[71]:
```
(2863, 50, 50)
```

In [72]:
```
y.shape
```

Out[72]:
```
(2863, 1)
```

In [73]:
```
# 標準化
X = X/255.0
```

save

In [74]:
```
import pickle

pickle_out = open("X.pickle","wb")
pickle.dump(X, pickle_out)
pickle_out.close()

pickle_out = open("y.pickle","wb")
pickle.dump(y, pickle_out)
pickle_out.close()
```

In [75]:
```
pickle_in = open("X.pickle","rb")
X = pickle.load(pickle_in)

pickle_in = open("y.pickle","rb")
y = pickle.load(pickle_in)
```

Model, DNN1

- 2 個類別，最後一個神經層為
- keras.layers.Dense(1, activation='sigmoid')
- 注意 只有一個神經元, activation用'sigmoid'
- 其output layer 的數值為介於0 與1 之數字
- loss='binary_crossentropy'

- 愈接近零，愈可能是dog, 愈接近1愈可能是cat

In [76]:
```
model_DNN1 = keras.Sequential([
    # input layer (1) 像素攤平，成為 一個維度
    keras.layers.Flatten(input_shape=(50, 50)),
    # hidden layer (2)
    keras.layers.Dense(128, activation='relu'),
    # output layer (3)
    keras.layers.Dense(1, activation='sigmoid')
])
model_DNN1.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_4 (Flatten)          (None, 2500)              0         
_________________________________________________________________
dense_11 (Dense)             (None, 128)               320128    
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 129       
=================================================================
Total params: 320,257
Trainable params: 320,257
Non-trainable params: 0
_________________________________________________________________
```

In [77]:
```
model_DNN1.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Model,DNN2

- n個類別，最後一個神經層為
- keras.layers.Dense(n, activation='softmax')
- 注意 n=2, activation用'softmax'
- loss用 sparse_categorical_crossentropy
- 其 output layer 為兩個神經元，數值為各類別的機率

In [78]:
```
model_DNN2 = keras.Sequential([
    # input layer (1) 像素攤平，成為 一個維度
    keras.layers.Flatten(input_shape=(50, 50)),
    # hidden layer (2)
    keras.layers.Dense(128, activation='relu'),
    # hidden layer (2)
    keras.layers.Dense(200, activation='relu'),
    # hidden layer (2)
    keras.layers.Dense(200, activation='relu'),
    # hidden layer (2)
    keras.layers.Dense(128, activation='relu'),
    # output layer (3)
    keras.layers.Dense(2, activation='softmax')
])
model_DNN2.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_5 (Flatten)          (None, 2500)              0         
_________________________________________________________________
dense_13 (Dense)             (None, 128)               320128    
```

```
dense_14 (Dense)             (None, 200)               25800
_________________________________________________________________
dense_15 (Dense)             (None, 200)               40200
_________________________________________________________________
dense_16 (Dense)             (None, 128)               25728
_________________________________________________________________
dense_17 (Dense)             (None, 2)                 258
=================================================================
Total params: 412,114
Trainable params: 412,114
Non-trainable params: 0
_________________________________________________________________
```

In [79]:
```
model_DNN2.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

fit

In [80]:
```
X.shape
```

Out[80]:
```
(2863, 50, 50)
```

In [81]:
```
print(X[0][0])
```

```
[0.78431373 0.6        0.6        0.20392157 0.32156863 0.71764706
 0.62352941 0.59607843 0.41176471 0.36470588 0.36078431 0.38823529
 0.65882353 0.61176471 0.59215686 0.43921569 0.39607843 0.34901961
 0.3372549  0.59215686 0.30588235 0.50980392 0.34117647 0.42352941
 0.49411765 0.50196078 0.43921569 0.37647059 0.42745098 0.44705882
 0.57254902 0.77254902 0.81176471 0.63529412 0.56470588 0.67843137
 0.72941176 0.8        0.79215686 0.77647059 0.74901961 0.71372549
 0.64313725 0.56862745 0.66666667 0.65882353 0.58039216 0.34117647
 0.31764706 0.30980392]
```

In [82]:
```
y.shape
```

Out[82]:
```
(2863, 1)
```

In [83]:
```
history_DNN1 =model_DNN1.fit(X,y, batch_size=32, epochs=10,validation_split=0.1)    #
```

```
Epoch 1/10
81/81 [==============================] - 3s 6ms/step - loss: 0.6402 - accuracy: 0.76
76 - val_loss: 0.2287 - val_accuracy: 0.9547
Epoch 2/10
81/81 [==============================] - 0s 4ms/step - loss: 0.2236 - accuracy: 0.94
43 - val_loss: 0.1265 - val_accuracy: 0.9721
Epoch 3/10
81/81 [==============================] - 0s 4ms/step - loss: 0.1336 - accuracy: 0.96
87 - val_loss: 0.0886 - val_accuracy: 0.9791
Epoch 4/10
81/81 [==============================] - 0s 4ms/step - loss: 0.0840 - accuracy: 0.97
92 - val_loss: 0.0660 - val_accuracy: 0.9826
Epoch 5/10
81/81 [==============================] - 0s 3ms/step - loss: 0.0816 - accuracy: 0.97
79 - val_loss: 0.1228 - val_accuracy: 0.9547
Epoch 6/10
81/81 [==============================] - 0s 4ms/step - loss: 0.0705 - accuracy: 0.97
93 - val_loss: 0.0545 - val_accuracy: 0.9791
Epoch 7/10
81/81 [==============================] - 0s 3ms/step - loss: 0.0416 - accuracy: 0.98
90 - val_loss: 0.0419 - val_accuracy: 0.9895
Epoch 8/10
81/81 [==============================] - 0s 3ms/step - loss: 0.0387 - accuracy: 0.99
10 - val_loss: 0.0265 - val_accuracy: 0.9930
```

```
Epoch 9/10
81/81 [==============================] - 0s 3ms/step - loss: 0.0305 - accuracy: 0.99
51 - val_loss: 0.0216 - val_accuracy: 0.9930
Epoch 10/10
81/81 [==============================] - 0s 3ms/step - loss: 0.0255 - accuracy: 0.99
63 - val_loss: 0.0199 - val_accuracy: 0.9965
```

In [23]:
```
history_DNN2 =model_DNN2.fit(X,y, batch_size=32, epochs=10,validation_split=0.1)   #
```

```
Epoch 1/10
282/282 [==============================] - 2s 5ms/step - loss: 0.7148 - accuracy: 0.
5104 - val_loss: 0.6859 - val_accuracy: 0.5420
Epoch 2/10
282/282 [==============================] - 1s 4ms/step - loss: 0.6815 - accuracy: 0.
5696 - val_loss: 0.6659 - val_accuracy: 0.5860
Epoch 3/10
282/282 [==============================] - 1s 3ms/step - loss: 0.6755 - accuracy: 0.
5694 - val_loss: 0.6699 - val_accuracy: 0.5810
Epoch 4/10
282/282 [==============================] - 1s 3ms/step - loss: 0.6666 - accuracy: 0.
5914 - val_loss: 0.6669 - val_accuracy: 0.5810
Epoch 5/10
282/282 [==============================] - 1s 3ms/step - loss: 0.6635 - accuracy: 0.
5927 - val_loss: 0.6583 - val_accuracy: 0.6060
Epoch 6/10
282/282 [==============================] - 1s 4ms/step - loss: 0.6575 - accuracy: 0.
6081 - val_loss: 0.6788 - val_accuracy: 0.5630
Epoch 7/10
282/282 [==============================] - 1s 4ms/step - loss: 0.6568 - accuracy: 0.
6048 - val_loss: 0.6619 - val_accuracy: 0.6080
Epoch 8/10
282/282 [==============================] - 1s 4ms/step - loss: 0.6547 - accuracy: 0.
6054 - val_loss: 0.6573 - val_accuracy: 0.6060
Epoch 9/10
282/282 [==============================] - 1s 4ms/step - loss: 0.6521 - accuracy: 0.
6142 - val_loss: 0.6609 - val_accuracy: 0.6090
Epoch 10/10
282/282 [==============================] - 1s 4ms/step - loss: 0.6538 - accuracy: 0.
6135 - val_loss: 0.6732 - val_accuracy: 0.5810
```

看起來，正確率不高，無法令人滿意。

- 調整一下
- 你有辦法透過調整參數，改變神經層與神經元，提高正確率嗎?

evaluation

In [84]:
```
import pandas as pd
# history 轉為 dataframe 格式
hist = pd.DataFrame(history_DNN1.history)
# 新增 epoch 欄位
hist['epoch'] = history_DNN1.epoch
# 顯示 epoch, loss, val_loss
hist.tail()
```

Out[84]:

	loss	accuracy	val_loss	val_accuracy	epoch
5	0.054789	0.984472	0.054519	0.979094	5
6	0.042252	0.990295	0.041851	0.989547	6
7	0.034259	0.993401	0.026503	0.993031	7
8	0.032765	0.994177	0.021563	0.993031	8

	loss	accuracy	val_loss	val_accuracy	epoch
9	0.027596	0.994953	0.019936	0.996516	9

In [85]:

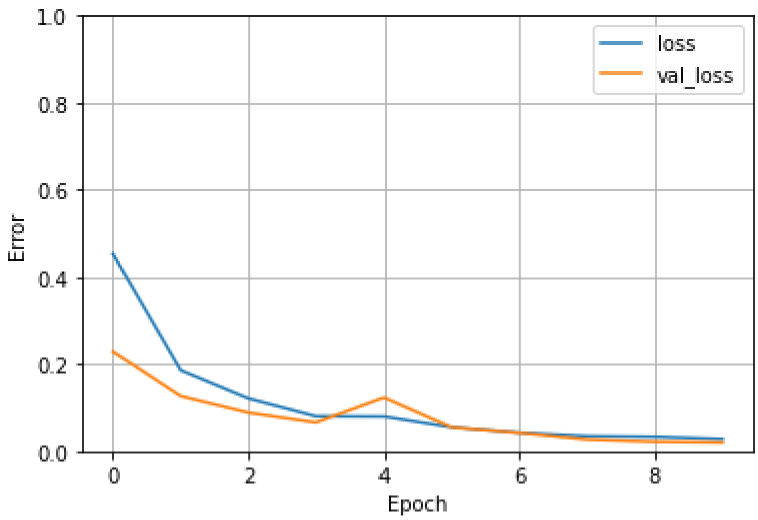
```
import pandas as pd
# history 轉為 dataframe 格式
hist = pd.DataFrame(history_DNN2.history)
# 新增 epoch 欄位
hist['epoch'] = history_DNN2.epoch
# 顯示 epoch, loss, val_loss
hist.tail()
```

Out[85]:

	loss	accuracy	val_loss	val_accuracy	epoch
5	0.656644	0.612778	0.678798	0.563	5
6	0.655804	0.607333	0.661894	0.608	6
7	0.656340	0.606778	0.657259	0.606	7
8	0.652015	0.617222	0.660856	0.609	8
9	0.652864	0.616556	0.673231	0.581	9

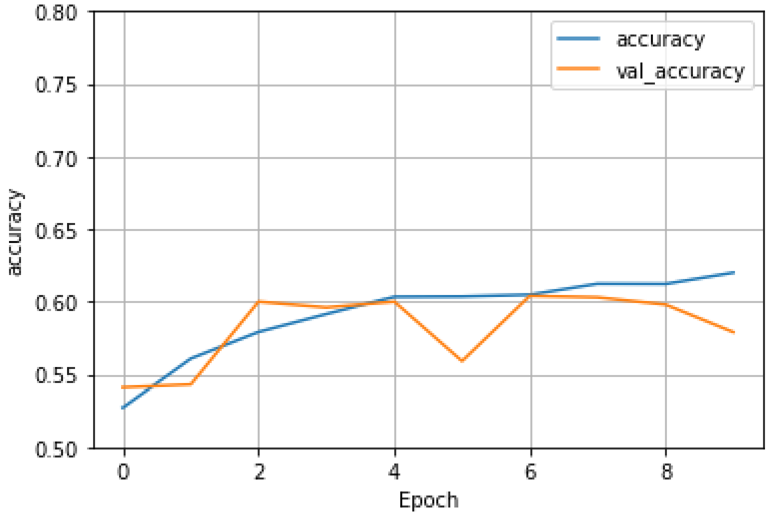
In [86]:

```
# 繪圖，顯示損失函數下降的趨勢
def plot_loss(history):
  plt.plot(history.history['loss'], label='loss')
  plt.plot(history.history['val_loss'], label='val_loss')
  plt.ylim([0, 1])
  plt.xlabel('Epoch')
  plt.ylabel('Error')
  plt.legend()
  plt.grid(True)
plot_loss(history_DNN1)
```



In [27]:

```
# 繪圖，顯示正確率上升的趨勢
def plot_acc(history):
  plt.plot(history.history['accuracy'], label='accuracy')
  plt.plot(history.history['val_accuracy'], label='val_accuracy')
  plt.ylim([0.5, 0.8])
  plt.xlabel('Epoch')
  plt.ylabel('accuracy')
  plt.legend()
  plt.grid(True)
plot_acc(history_DNN1)
```



模型儲存與預測

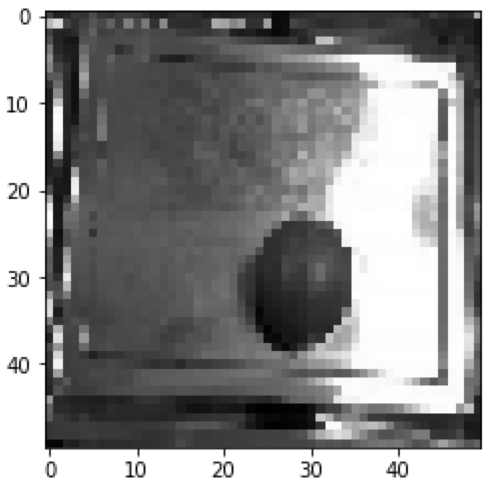
In [87]:
```
# save model
model_DNN1.save('c:/111/model_DNN1')
model_DNN2.save('c:/111/model_DNN2')
```

```
INFO:tensorflow:Assets written to: c:/111/model_DNN1\assets
INFO:tensorflow:Assets written to: c:/111/model_DNN2\assets
```

In [88]:
```
# load model
model_DNN1 = tf.keras.models.load_model('c:/111/model_DNN1')
model_DNN2 = tf.keras.models.load_model('c:/111/model_DNN2')
```

任選一張圖進行預測

In [90]:
```
# img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)
IMG_SIZE = 50
img_array=cv2.imread('c:/111/dog_vs_cat/cats/tomato030.jpg',cv2.IMREAD_GRAYSCALE)
new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
plt.imshow(new_array, cmap='gray')
plt.show()
```



In [91]:
```
# 從原來的 array 要調整成張量，三個維度
# 與模型訓練時的 input shape 一致
img_tf = tf.expand_dims(new_array, 0) # Create a batch
img_tf.shape
```

Out[91]:
```
TensorShape([1, 50, 50])
```

model_DNN1

In [92]:
```
# model_DNN1
predictions = model_DNN1.predict(img_tf)
score =predictions[0]
score
```

Out[92]:
```
array([1.], dtype=float32)
```

model_DNN1

In [93]:
```
# model_DNN2
predictions = model_DNN2.predict(img_tf)
score =predictions[0]
score
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_functio
n.<locals>.predict_function at 0x000001A80CE0BAF0> triggered tf.function retracing.
Tracing is expensive and the excessive number of tracings could be due to (1) creati
ng @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3)
passing Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option
that relaxes argument shapes that can avoid unnecessary retracing. For (3), please r
efer to https://www.tensorflow.org/guide/function#controlling_retracing and https://
www.tensorflow.org/api_docs/python/tf/function for  more details.
```

Out[93]:
```
array([1.2547241e-18, 1.0000000e+00], dtype=float32)
```

In [94]:
```
print("這應該是 {} 有百分之 {:.2f} 的信心 "
    .format(CATEGORIES[np.argmax(score)], 100 * np.max(score)))
```

```
這應該是 cats 有百分之 100.00 的信心
```

In [ ]:

In [ ]:

其他模型

- 以下兩個模型用到 CNN 神經層，複雜許多，執行的時間也會比較長，但準確率有明顯的提高。
- 各位同學只要本看結果即可，以後會細講。

CNN

In [95]:
```
# 重組 張量的 shape
# 注意維度有改變，如果是三個顏色，改為 3
X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE,1)
y=np.array(y).reshape(-1, 1)
```

In [96]:
```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Activation,MaxPooling2D,Conv2D,Flatten,Dense

model = Sequential()

model.add(Conv2D(256, (3, 3), input_shape=(50, 50,1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors

model.add(Dense(64))

model.add(Dense(1))
model.add(Activation('sigmoid'))

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

In [97]:

```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 48, 48, 256)       2560
_________________________________________________________________
activation_3 (Activation)    (None, 48, 48, 256)       0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 24, 24, 256)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 22, 22, 256)       590080
_________________________________________________________________
activation_4 (Activation)    (None, 22, 22, 256)       0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 11, 11, 256)       0
_________________________________________________________________
flatten_6 (Flatten)          (None, 30976)             0
_________________________________________________________________
dense_18 (Dense)             (None, 64)                1982528
_________________________________________________________________
dense_19 (Dense)             (None, 1)                 65
_________________________________________________________________
activation_5 (Activation)    (None, 1)                 0
=================================================================
Total params: 2,575,233
Trainable params: 2,575,233
Non-trainable params: 0
_________________________________________________________________
```

In [98]:

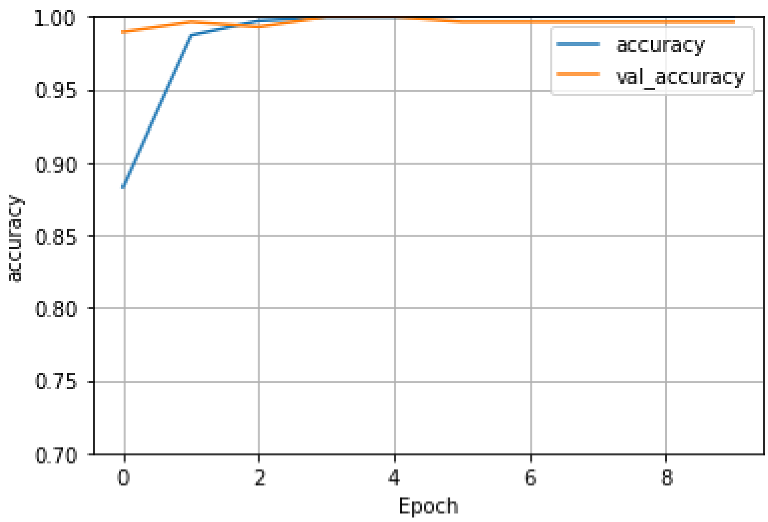
```
history =model.fit(X,y, batch_size=32, epochs=10,validation_split=0.1,verbose=2)
```

```
Epoch 1/10
81/81 - 45s - loss: 0.2990 - accuracy: 0.8832 - val_loss: 0.0453 - val_accuracy: 0.9
895
Epoch 2/10
81/81 - 55s - loss: 0.0458 - accuracy: 0.9872 - val_loss: 0.0134 - val_accuracy: 0.9
965
Epoch 3/10
81/81 - 44s - loss: 0.0114 - accuracy: 0.9973 - val_loss: 0.0186 - val_accuracy: 0.9
930
Epoch 4/10
81/81 - 46s - loss: 0.0035 - accuracy: 0.9996 - val_loss: 0.0029 - val_accuracy: 1.0
000
Epoch 5/10
81/81 - 45s - loss: 0.0014 - accuracy: 0.9996 - val_loss: 0.0042 - val_accuracy: 1.0
000
Epoch 6/10
81/81 - 45s - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.0079 - val_accuracy: 0.9
965
Epoch 7/10
```

```
81/81 - 44s - loss: 1.7778e-04 - accuracy: 1.0000 - val_loss: 0.0091 - val_accuracy:
0.9965
Epoch 8/10
81/81 - 44s - loss: 1.1326e-04 - accuracy: 1.0000 - val_loss: 0.0098 - val_accuracy:
0.9965
Epoch 9/10
81/81 - 45s - loss: 8.8431e-05 - accuracy: 1.0000 - val_loss: 0.0096 - val_accuracy:
0.9965
Epoch 10/10
81/81 - 44s - loss: 7.2157e-05 - accuracy: 1.0000 - val_loss: 0.0086 - val_accuracy:
0.9965
```

In [100…

```
# 繪圖，顯示損失函數下降的趨勢
def plot_acc(history):
  plt.plot(history.history['accuracy'], label='accuracy')
  plt.plot(history.history['val_accuracy'], label='val_accuracy')
  plt.ylim([0.7, 1.0])
  plt.xlabel('Epoch')
  plt.ylabel('accuracy')
  plt.legend()
  plt.grid(True)
plot_acc(history)
```



CNN2

In [101…

```
# 重組 張量的 shape
X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE,1)
y=np.array(y).reshape(-1, 1)
```

In [102…

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Activation,MaxPooling2D,Conv2D,Flatten,Dense

# 宣告 Sequential()
model = Sequential()
# Conv2D
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(50, 50, 1)))
# MaxPooling2D
model.add(MaxPooling2D((2, 2)))
# Conv2D
model.add(Conv2D(64, (3, 3), activation='relu'))
# MaxPooling2D
model.add(MaxPooling2D((2, 2)))
# Conv2D
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
```

```
model.add(Dense(2, activation='softmax'))
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 48, 48, 32)        320
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 24, 24, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 22, 22, 64)        18496
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 11, 11, 64)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 9, 9, 64)          36928
_________________________________________________________________
flatten_7 (Flatten)          (None, 5184)              0
_________________________________________________________________
dense_20 (Dense)             (None, 64)                331840
_________________________________________________________________
dense_21 (Dense)             (None, 2)                 130
=================================================================
Total params: 387,714
Trainable params: 387,714
Non-trainable params: 0
_________________________________________________________________
```

In [103…

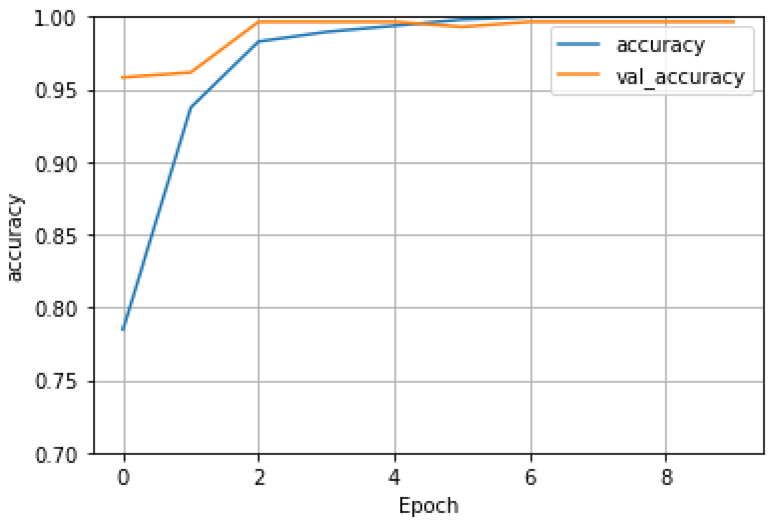
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(X, y, epochs=10,
                    validation_split=0.1)
```

```
Epoch 1/10
81/81 [==============================] - 7s 69ms/step - loss: 0.5191 - accuracy: 0.7
584 - val_loss: 0.2407 - val_accuracy: 0.9582
Epoch 2/10
81/81 [==============================] - 5s 67ms/step - loss: 0.2639 - accuracy: 0.9
204 - val_loss: 0.1602 - val_accuracy: 0.9617
Epoch 3/10
81/81 [==============================] - 6s 69ms/step - loss: 0.1882 - accuracy: 0.9
725 - val_loss: 0.1491 - val_accuracy: 0.9965
Epoch 4/10
81/81 [==============================] - 6s 68ms/step - loss: 0.1544 - accuracy: 0.9
899 - val_loss: 0.1140 - val_accuracy: 0.9965
Epoch 5/10
81/81 [==============================] - 6s 73ms/step - loss: 0.1205 - accuracy: 0.9
941 - val_loss: 0.1059 - val_accuracy: 0.9965
Epoch 6/10
81/81 [==============================] - 6s 79ms/step - loss: 0.1076 - accuracy: 0.9
985 - val_loss: 0.0912 - val_accuracy: 0.9930
Epoch 7/10
81/81 [==============================] - 6s 73ms/step - loss: 0.0949 - accuracy: 1.0
000 - val_loss: 0.0820 - val_accuracy: 0.9965
Epoch 8/10
81/81 [==============================] - 6s 68ms/step - loss: 0.0845 - accuracy: 1.0
000 - val_loss: 0.0741 - val_accuracy: 0.9965
Epoch 9/10
81/81 [==============================] - 6s 69ms/step - loss: 0.0797 - accuracy: 1.0
000 - val_loss: 0.0679 - val_accuracy: 0.9965
Epoch 10/10
81/81 [==============================] - 6s 68ms/step - loss: 0.0667 - accuracy: 1.0
000 - val_loss: 0.0654 - val_accuracy: 0.9965
```

In [107…

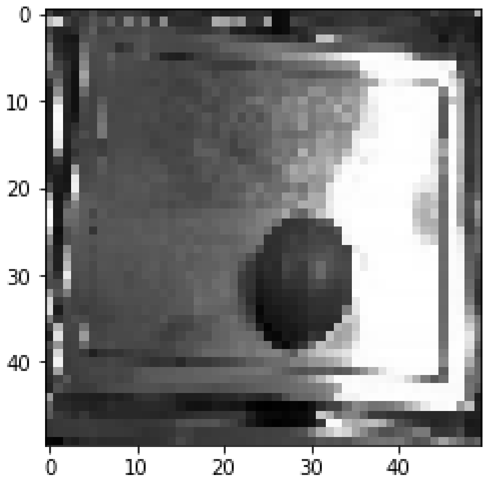
```
# 繪圖，顯示損失函數下降的趨勢
def plot_acc(history):
  plt.plot(history.history['accuracy'], label='accuracy')
```

```
    plt.plot(history.history['val_accuracy'], label='val_accuracy')
    plt.ylim([0.7, 1.0])
    plt.xlabel('Epoch')
    plt.ylabel('accuracy')
    plt.legend()
    plt.grid(True)
plot_acc(history)
```



In [108…

```
### 任選一張圖進行預測

# img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)
IMG_SIZE = 50
img_array=cv2.imread('c:/111/dog_vs_cat/cats/tomato030.jpg',cv2.IMREAD_GRAYSCALE)
new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
plt.imshow(new_array, cmap='gray')
plt.show()
```



In [109…

```
new_array
```

Out[109…

```
array([[ 70,  39,  44, ...,  49,  52, 177],
       [184, 215,  70, ...,  36,  38,  29],
       [ 70,  42,  41, ...,  44,  40,  27],
       ...,
       [ 59,  62,  63, ...,  56,  31,  33],
       [ 58,  62,  61, ...,  76,  36,  34],
       [ 14,  12,  41, ..., 114,  89, 102]], dtype=uint8)
```

In [110…

```
# 重組 張量的 shape
X = np.array(new_array).reshape(IMG_SIZE, IMG_SIZE,1)
X.shape
```

Out[110…
```
(50, 50, 1)
```

In [111…
```
# 從原來的 array 要調整成張量
# 與模型訓練時的 input shape 一致
img_tf = tf.expand_dims(X, 0) # Create a batch
img_tf.shape
```

Out[111…
```
TensorShape([1, 50, 50, 1])
```

In [112…
```
predictions = model.predict(img_tf)
score =predictions[0]

print("這應該是 {} 有百分之 {:.2f} 的信心 "
    .format(CATEGORIES[np.argmax(score)], 100 * np.max(score)))
```

```
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_functio
n.<locals>.predict_function at 0x000001A80FDC0550> triggered tf.function retracing.
Tracing is expensive and the excessive number of tracings could be due to (1) creati
ng @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3)
passing Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option
that relaxes argument shapes that can avoid unnecessary retracing. For (3), please r
efer to https://www.tensorflow.org/guide/function#controlling_retracing and https://
www.tensorflow.org/api_docs/python/tf/function for  more details.
這應該是 cats 有百分之 100.00 的信心
```

In [ ]:

In [ ]:

In [ ]: