VGG16、ResNet20、InceptionV3

7107029022 資管碩二 邱靖詒

- **VGG16**

1. 匯入套件

In [1]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
from keras import layers
from keras import models
from keras import optimizers
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import VGG16
```

```
D:\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarnin
to `np.floating` is deprecated. In future, it will be treated as
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

2. 兩種情況：VGG16 (w/o data augmentation)、VGG16 (with data augmentation)
3. VGG (w/o data augmentation)
   - 模型架構

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 150, 150, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

- 使用預先訓練的 convolutional base 萃取特徵

In [5]:
```
# 使用預先訓練的 convolutional base 萃取特徵
def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 4, 4, 512))
    labels = np.zeros(shape=(sample_count))
    generator = datagen.flow_from_directory(directory,
                                            target_size=(150, 150),
                                            batch_size=batch_size,
                                            class_mode='binary')
    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch)
        features[i * batch_size : (i + 1) * batch_size] = features_batch
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch
        i += 1
        print(i, end=' ') # 由於萃取需要較長的時間，我們印出 i 來檢視進度
        if i * batch_size >= sample_count:
            break
    return features, labels
```

- 資料展平

In [7]:
```
# 將資料展平
train_features = np.reshape(train_features, (2000, 4 * 4 * 512))
validation_features = np.reshape(validation_features, (1000, 4 * 4 * 512))
test_features = np.reshape(test_features, (1000, 4 * 4 * 512))
```

- 建置模型

In [8]:
```
# 建置模型
model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.RMSprop(lr=2e-5), loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(train_features, train_labels,epochs=30, batch_size=20,
                    validation_data=(validation_features, validation_labels))
```

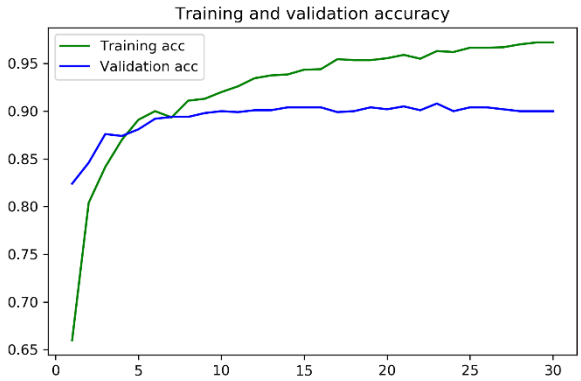
```
Epoch 21/30
2000/2000 [==============================] - 1s 669us/step - loss: 0.1262 - acc: 0.9590 - val_loss: 0.2346 - val_acc: 0.9050
Epoch 22/30
2000/2000 [==============================] - 1s 691us/step - loss: 0.1274 - acc: 0.9550 - val_loss: 0.2406 - val_acc: 0.9010
Epoch 23/30
2000/2000 [==============================] - 1s 684us/step - loss: 0.1192 - acc: 0.9630 - val_loss: 0.2343 - val_acc: 0.9080
Epoch 24/30
2000/2000 [==============================] - 1s 694us/step - loss: 0.1139 - acc: 0.9620 - val_loss: 0.2385 - val_acc: 0.9000
Epoch 25/30
2000/2000 [==============================] - 1s 705us/step - loss: 0.1034 - acc: 0.9665 - val_loss: 0.2367 - val_acc: 0.9040
Epoch 26/30
2000/2000 [==============================] - 1s 687us/step - loss: 0.1072 - acc: 0.9665 - val_loss: 0.2369 - val_acc: 0.9040
Epoch 27/30
2000/2000 [==============================] - 1s 685us/step - loss: 0.1008 - acc: 0.9670 - val_loss: 0.2366 - val_acc: 0.9020
Epoch 28/30
2000/2000 [==============================] - 1s 696us/step - loss: 0.0938 - acc: 0.9700 - val_loss: 0.2417 - val_acc: 0.9000
Epoch 29/30
2000/2000 [==============================] - 1s 690us/step - loss: 0.0916 - acc: 0.9720 - val_loss: 0.2441 - val_acc: 0.9000
Epoch 30/30
2000/2000 [==============================] - 1s 694us/step - loss: 0.0919 - acc: 0.9720 - val_loss: 0.2399 - val_acc: 0.9000
```

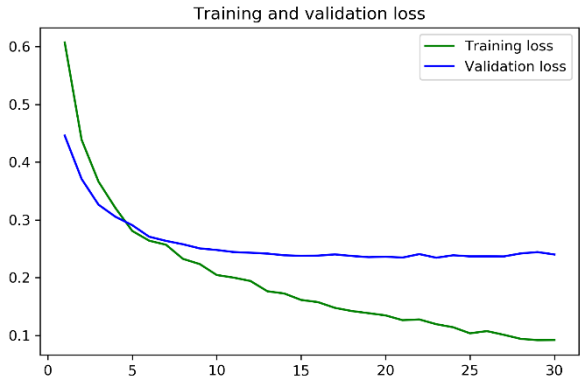
- 模型準確率 + 視覺化呈現

In [9]:
```
loss, accuracy = model.evaluate(train_features, train_labels)
print("\nLoss: %.2f, Accuracy: %.2f%%" % (loss, accuracy*100))
```

```
2000/2000 [==============================] - 1s 251us/step

Loss: 0.07, Accuracy: 98.05%
```





4. VGG (with data augmentation)

- 模型架構

  在 convolutional base 卷積基底上增加密集層分類器

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0
_________________________________________________________________
```

- 凍結卷積基底神經網路

In [12]:

```
print('This is the number of trainable weights '
'before freezing the conv base:', len(model.trainable_weights))

conv_base.trainable = False

print('This is the number of trainable weights '
'after freezing the conv base:', len(model.trainable_weights))
```

```
This is the number of trainable weights before freezing the conv base: 30
This is the number of trainable weights after freezing the conv base: 4
```

- 以凍結的 convolutional base 卷積基底進行從頭到尾完整的 model 訓練 + 建置模型

In [13]:

```
train_datagen = ImageDataGenerator(
                rescale=1./255, rotation_range=40, width_shift_range=0.2,
                height_shift_range=0.2, shear_range=0.2,
                zoom_range=0.2, horizontal_flip=True, fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(train_dir,
                    target_size=(150, 150), batch_size=20, class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
                       validation_dir, target_size=(150, 150),
                       batch_size=20, class_mode='binary')

model.compile( loss='binary_crossentropy', optimizer=optimizers.RMSprop(lr=2e-5), metrics=['acc'])

history = model.fit_generator(train_generator, steps_per_epoch=100,
        epochs=30, validation_data=validation_generator, validation_steps=50)
```

```
Epoch 21/30
100/100 [==============================] - 51s 505ms/step - loss: 0.3036 - acc: 0.8670 - val_loss: 0.2403 - val_acc: 0.8980
Epoch 22/30
100/100 [==============================] - 51s 506ms/step - loss: 0.2971 - acc: 0.8735 - val_loss: 0.2414 - val_acc: 0.9030
Epoch 23/30
100/100 [==============================] - 51s 505ms/step - loss: 0.3040 - acc: 0.8720 - val_loss: 0.2395 - val_acc: 0.9030
Epoch 24/30
100/100 [==============================] - 51s 507ms/step - loss: 0.2772 - acc: 0.8830 - val_loss: 0.2392 - val_acc: 0.9010
Epoch 25/30
100/100 [==============================] - 51s 506ms/step - loss: 0.2932 - acc: 0.8685 - val_loss: 0.2386 - val_acc: 0.8950
Epoch 26/30
100/100 [==============================] - 51s 507ms/step - loss: 0.2920 - acc: 0.8775 - val_loss: 0.2400 - val_acc: 0.9010
Epoch 27/30
100/100 [==============================] - 51s 505ms/step - loss: 0.2845 - acc: 0.8780 - val_loss: 0.2429 - val_acc: 0.9010
Epoch 28/30
100/100 [==============================] - 51s 505ms/step - loss: 0.2859 - acc: 0.8750 - val_loss: 0.2442 - val_acc: 0.9000
Epoch 29/30
100/100 [==============================] - 51s 506ms/step - loss: 0.2983 - acc: 0.8825 - val_loss: 0.2353 - val_acc: 0.8990
Epoch 30/30
100/100 [==============================] - 51s 506ms/step - loss: 0.2886 - acc: 0.8740 - val_loss: 0.2345 - val_acc: 0.9020
```

- 模型訓練

In [8]:
```
model.compile(optimizer = 'rmsprop', loss = 'categorical_crossentropy', metrics = ['accuracy'])
history = model.fit_generator(
        train_generator,
        steps_per_epoch = train_generator.samples // batch_size,
        validation_data = validation_generator,
        validation_steps = validation_generator.samples // batch_size,
        epochs = 20)

model.save('model-inceptionv3-final.h5')
```

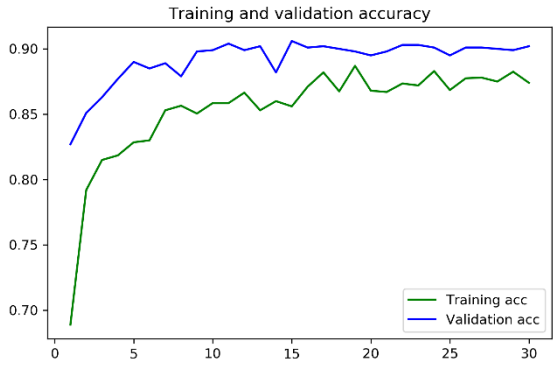
```
1000/1000 [==============================] - 131s 131ms/step - loss: 0.4315 - acc: 0.7765 - val_loss: 7.5340 - val_acc: 0.518
0
Epoch 15/20
1000/1000 [==============================] - 132s 132ms/step - loss: 0.4087 - acc: 0.8070 - val_loss: 7.5696 - val_acc: 0.515
00.4
Epoch 16/20
1000/1000 [==============================] - 132s 132ms/step - loss: 0.4517 - acc: 0.7640 - val_loss: 7.7938 - val_acc: 0.511
0
Epoch 17/20
1000/1000 [==============================] - 132s 132ms/step - loss: 0.4343 - acc: 0.7815 - val_loss: 7.7535 - val_acc: 0.511
0
Epoch 18/20
1000/1000 [==============================] - 131s 131ms/step - loss: 0.4261 - acc: 0.7895 - val_loss: 7.9017 - val_acc: 0.506
0
Epoch 19/20
1000/1000 [==============================] - 131s 131ms/step - loss: 0.4490 - acc: 0.7710 - val_loss: 7.8034 - val_acc: 0.512
0
Epoch 20/20
1000/1000 [==============================] - 131s 131ms/step - loss: 0.4296 - acc: 0.7815 - val_loss: 7.7969 - val_acc: 0.512
0
```

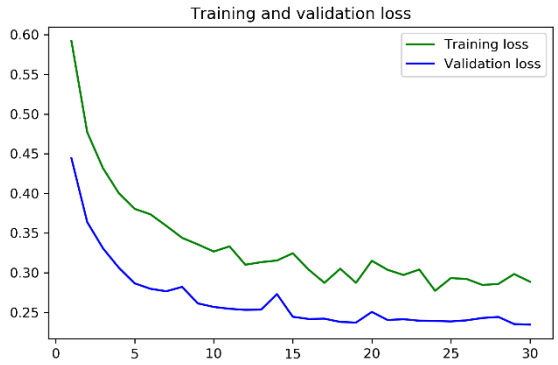
- 模型準確率 + 視覺化呈現

  模型儲存為「model-VGG16(withDA)-final.h5」

In [15]:
```
model.save('model-VGG16(withDA)-final.h5')
scores = model.evaluate_generator(generator=validation_generator, steps=validation_generator.samples // batch_size)
print("\nLoss: %.2f, Accuracy: %.2f%%" % (scores[0]/100, scores[1]*100))
```

```
Loss: 0.00, Accuracy: 90.20%
```





◆ ResNet20

1. 匯入套件

In [17]:
```
import sys
import numpy as np
from tensorflow.python.keras import backend as K
from tensorflow.python.keras.models import Model
from tensorflow.python.keras.layers import Flatten, Dense, Dropout
from tensorflow.python.keras.applications.resnet50 import ResNet50
from tensorflow.python.keras.optimizers import Adam
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.keras import backend as K
from tensorflow.python.keras.models import load_model
from tensorflow.python.keras.preprocessing import image
```

2. 參數設置

In [18]:
```
# 資料路徑
DATASET_PATH  = 'D:/Anaconda3/Scripts/5 上課資料/電腦視覺與人機互動/20191031HW/cats_and_dogs_small'
# 影像大小
IMAGE_SIZE = (224, 224)
# 影像類別數
NUM_CLASSES = 2
# 若 GPU 記憶體不足，可調降 batch size 或凍結更多層網路
BATCH_SIZE = 2
# 凍結網路層數
FREEZE_LAYERS = 2
# Epoch 數
NUM_EPOCHS = 20
# 模型輸出儲存的檔案
WEIGHTS_FINAL = 'model-resnet50-final.h5'
```

3. 透過 data augmentation 產生訓練與驗證用的影像資料

In [19]:
```
# 透過 data augmentation 產生訓練與驗證用的影像資料
train_datagen = ImageDataGenerator(rotation_range=40,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   channel_shift_range=10,
                                   horizontal_flip=True,
                                   fill_mode='nearest')
train_batches = train_datagen.flow_from_directory(DATASET_PATH + '/train',
                                                  target_size=IMAGE_SIZE,
                                                  interpolation='bicubic',
                                                  class_mode='categorical',
                                                  shuffle=True,
                                                  batch_size=BATCH_SIZE)

valid_datagen = ImageDataGenerator()
valid_batches = valid_datagen.flow_from_directory(DATASET_PATH + '/validation',
                                                  target_size=IMAGE_SIZE,
                                                  interpolation='bicubic',
                                                  class_mode='categorical',
                                                  shuffle=False,
                                                  batch_size=BATCH_SIZE)
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

4. 輸出各類別的索引值

In [20]:
```
# 輸出各類別的索引值
for cls, idx in train_batches.class_indices.items():
    print('Class #{} = {}'.format(idx, cls))
```

```
Class #0 = cats
Class #1 = dogs
```

5. 模型架構

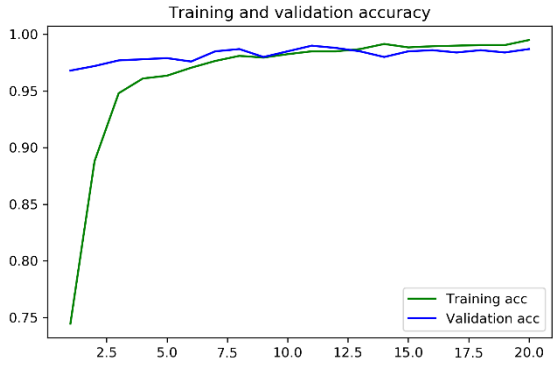
Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 224, 224, 3)	0	
conv1 (Conv2D)	(None, 112, 112, 64)	9472	input_1[0][0]
bn_conv1 (BatchNormalization)	(None, 112, 112, 64)	256	conv1[0][0]
activation (Activation)	(None, 112, 112, 64)	0	bn_conv1[0][0]
max_pooling2d (MaxPooling2D)	(None, 55, 55, 64)	0	activation[0][0]
res2a_branch2a (Conv2D)	(None, 55, 55, 64)	4160	max_pooling2d[0][0]
bn2a_branch2a (BatchNormalizati	(None, 55, 55, 64)	256	res2a_branch2a[0][0]
activation_1 (Activation)	(None, 55, 55, 64)	0	bn2a_branch2a[0][0]

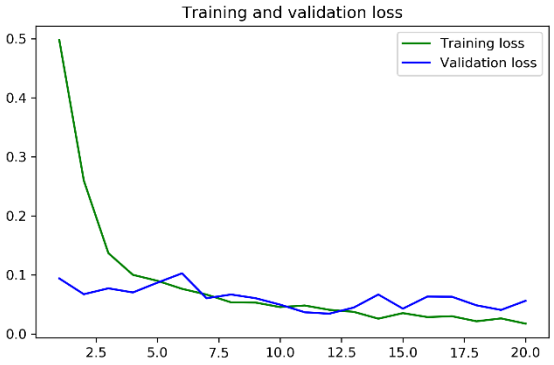
6. 模型準確率 + 視覺化呈現

模型儲存為「model-resnet50-final.h5」

In [22]:
```
scores = model.evaluate_generator(generator=validation_generator, steps=validation_generator.samples // BATCH_SIZE)
print("\nLoss: %.2f, Accuracy: %.2f%%" % (scores[0]/100, scores[1]*100))
```

```
Loss: 0.00, Accuracy: 90.20%
```





◆ InceptionV3

1. 匯入套件

In [1]:
```
import pandas as pd
import numpy as np
from keras.applications.inception_v3 import InceptionV3
from keras.applications.inception_v3 import preprocess_input
from keras.preprocessing.image import ImageDataGenerator, array_to_img
from keras.preprocessing import image
from keras.models import Model, Sequential
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, AveragePooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.optimizers import SGD
from keras import backend as K
```

2. 參數設置

In [2]:
```
# 影像維度
img_width, img_height = 299, 299

train_data_dir = 'D:/Anaconda3/Scripts/5 上課資料/電腦視覺與人機互動/20191031HW/cats_and_dogs_small/train'
validation_data_dir = 'D:/Anaconda3/Scripts/5 上課資料/電腦視覺與人機互動/20191031HW/cats_and_dogs_small/validation'
nb_train_samples = 2000
nb_validation_samples = 800
batch_size = 2
```

3. 凍結所有層(除了 Bottleneck Layers 以外)來進行微調透

In [5]:
```
# 凍結所有層(除了Bottleneck Layers以外)來進行微調
for layer in model.layers:
    if layer.name in ['predictions']:
        continue
    layer.trainable = False
```

4. 透過 data augmentation 產生訓練與驗證用的影像資料

In [7]:
```
train_datagen = ImageDataGenerator(rotation_range=40,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   fill_mode='nearest')
train_generator = train_datagen.flow_from_directory(directory='D:/Anaconda3/Scripts/5 上課資料/電腦視覺與人機互動/20191031HW/cat
                                                    target_size=[img_width, img_height],
                                                    batch_size=batch_size,
                                                    class_mode='categorical')

validation_datagen = ImageDataGenerator()
validation_generator = validation_datagen.flow_from_directory(directory='D:/Anaconda3/Scripts/5 上課資料/電腦視覺與人機互動/2019
                                                              target_size=[img_width, img_height],
                                                              batch_size=batch_size,
                                                              class_mode='categorical')
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

5. 輸出各類別的索引值

In [20]:
```
# 輸出各類別的索引值
for cls, idx in train_batches.class_indices.items():
    print('Class #{} = {}'.format(idx, cls))
```

```
Class #0 = cats
Class #1 = dogs
```

6. 模型架構

In [4]:
```
base_model = InceptionV3(weights='imagenet', include_top=False)

# 添加全域空間平均池化層
x = base_model.output
x = GlobalAveragePooling2D()(x)
# 增加全連階層
x = Dense(1024, activation='relu', name='fc1')(x)
prediction = Dense(2, activation='softmax', name='predictions')(x)
model = Model(inputs=base_model.input, outputs=prediction)
model.summary()
```

```
concatenate_2 (Concatenate)     (None, None, None, 7 0           activation_92[0][0]
                                                                 activation_93[0][0]
__________________________________________________________________________________________________
activation_94 (Activation)      (None, None, None, 1 0           batch_normalization_94[0][0]
__________________________________________________________________________________________________
mixed10 (Concatenate)           (None, None, None, 2 0           activation_86[0][0]
                                                                 mixed9_1[0][0]
                                                                 concatenate_2[0][0]
                                                                 activation_94[0][0]
__________________________________________________________________________________________________
global_average_pooling2d_1 (Glo (None, 2048)         0           mixed10[0][0]
__________________________________________________________________________________________________
fc1 (Dense)                     (None, 1024)         2098176     global_average_pooling2d_1[0][0]
__________________________________________________________________________________________________
predictions (Dense)             (None, 2)            2050        fc1[0][0]
==================================================================================================
Total params: 23,903,010
Trainable params: 23,868,578
Non-trainable params: 34,432
__________________________________________________________________________________________________
```

7. 模型準確率 + 視覺化呈現

模型儲存為「model-inceptionv3-final2.h5」

In [12]:
```
scores = model.evaluate_generator(generator=validation_generator, steps=validation_generator.samples // batch_size)
print("\nLoss: %.2f, Accuracy: %.2f%%" % (scores[0]/100, scores[1]*100))
```

```
Loss: 0.08, Accuracy: 50.00%
```