109403034 賴昀廷 資管三B [CoLab連結]

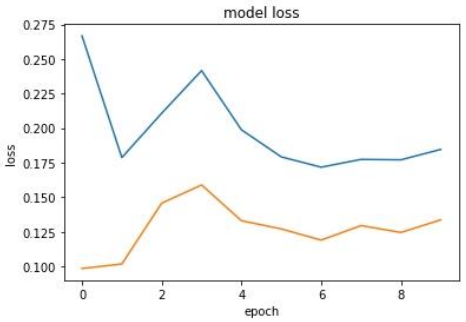
[其他] test & train 的路徑我下載到本地端後有稍微更改一下，所以跟原本助教給的路徑不一樣

[最高的TestACC]

[284]:
```
# 讀入測試資料並評估模型
#print(test_dir)
test_ds = make_dataset(test_dir)
test_ds = test_ds.batch(BATCH_SIZE)

score = model.evaluate(test_ds)

print("history")
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

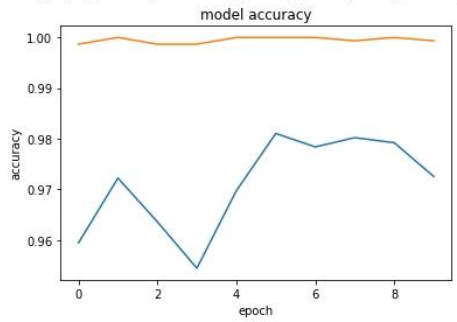
```
7/7 [==============================] - 1s 91ms/step - loss: 3.9705 - accuracy: 0.4539
history
Test loss: 3.9705116748809814
Test accuracy: 0.45389220118522644
```

[283]:
```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("model loss")
plt.ylabel("loss")
plt.xlabel("epoch")

plt.show()
```



[282]:
```
print(history.history.keys())

plt.plot(history.history["accuracy"])
plt.plot(history.history["val_accuracy"])
plt.title("model accuracy")
plt.ylabel("accuracy")
plt.xlabel("epoch")
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



[281]:
```
# todo
EPOCHS = 10 #None

##########
# todo #
##########
# model.compile 決定 learning strategy、Loss calculator


# onehot編碼用categorical_crossentropy
# model.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])


model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
history = model.fit(train_ds, epochs=EPOCHS, validation_data=val_ds, class_weight=class_weights)
```

```
Epoch 1/10
47/47 [==============================] - 16s 210ms/step - loss: 0.2667 - accuracy: 0.9594 - val_loss: 0.0984 - val_accuracy: 0.9987
Epoch 2/10
47/47 [==============================] - 16s 219ms/step - loss: 0.1787 - accuracy: 0.9722 - val_loss: 0.1017 - val_accuracy: 1.0000
Epoch 3/10
47/47 [==============================] - 20s 234ms/step - loss: 0.2108 - accuracy: 0.9636 - val_loss: 0.1457 - val_accuracy: 0.9987
Epoch 4/10
47/47 [==============================] - 19s 252ms/step - loss: 0.2416 - accuracy: 0.9545 - val_loss: 0.1589 - val_accuracy: 0.9987
Epoch 5/10
47/47 [==============================] - 20s 282ms/step - loss: 0.1988 - accuracy: 0.9697 - val_loss: 0.1330 - val_accuracy: 1.0000
Epoch 6/10
47/47 [==============================] - 19s 224ms/step - loss: 0.1792 - accuracy: 0.9811 - val_loss: 0.1270 - val_accuracy: 1.0000
Epoch 7/10
47/47 [==============================] - 16s 203ms/step - loss: 0.1717 - accuracy: 0.9784 - val_loss: 0.1189 - val_accuracy: 1.0000
Epoch 8/10
47/47 [==============================] - 16s 233ms/step - loss: 0.1773 - accuracy: 0.9802 - val_loss: 0.1295 - val_accuracy: 0.9993
Epoch 9/10
2023-04-09 14:53:34.578384: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:177] Filling up shuffle buffer (this may take a while): 6114 of 30000
  2/47 [>.............................] - ETA: 3s - loss: 0.2236 - accuracy: 0.9766
2023-04-09 14:53:36.504532: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:230] Shuffle buffer filled.
47/47 [==============================] - 22s 209ms/step - loss: 0.1771 - accuracy: 0.9792 - val_loss: 0.1245 - val_accuracy: 1.0000
Epoch 10/10
47/47 [==============================] - 15s 218ms/step - loss: 0.1846 - accuracy: 0.9726 - val_loss: 0.1336 - val_accuracy: 0.9993
```

[過程]

前前後後試了超多種model排列，也從本來用的colab換到家裡的電腦跑，因為後來發現colab額度太小。在調整過程中大概抓出了幾個比較會影響結果的參數，每次修改後都會將參數以及結果記錄起來，列舉一些，見下方表格。

後來上網也有查到可以做資料增強來提高準確度，像是對訓練集的圖片做水平、垂直翻轉之類的以增加訓練大小，但實作後一直卡data_generator那邊，所以最後就放棄了。印象最深刻的是最後有查到可以在model那邊加上預處理layers.experimental.preprocessing.RandomFlip("vertical")，一加上去準確度直接提升，這應該是在這過程中最開心的時候了吧!

input_size	batch	shuffle	Epochs	model	test_acc
128*128	128	20000	10	Conv2D(32,3*3) + MaxPooling2D(2*2) Conv2D(32,5*5) + MaxPooling2D(2*2) Dense(512)	25%
128*128	128	20000	10	Conv2D(32,3*3) + MaxPooling2D(2*2) Conv2D(32,3*3) + MaxPooling2D(2*2) Conv2D(64,3*3) + MaxPooling2D(2*2) Dense(512)	32%
256*256	128	20000	30	Conv2D(16,3*3) + MaxPooling2D(2*2) Conv2D(32,3*3) + MaxPooling2D(2*2) Conv2D(64,3*3) + MaxPooling2D(2*2) Conv2D(128,3*3) + MaxPooling2D(2*2) Dense(512) + Dense(256) + Dense(128) + Drop(0.5)	34.6%
300*300	128	20000	500	同上	28%
512*512	200	20000	50	同上	32.8%
256*256	128	30000	35	RandomFlip(horizontal) + 同上	36.77%
256*256	128	30000	按了好多次	RandomFlip(vertical) Conv2D(16,3*3) + MaxPooling2D(2*2) Conv2D(32,3*3) + MaxPooling2D(2*2) Conv2D(64,3*3) + MaxPooling2D(2*2) Conv2D(128,3*3) + MaxPooling2D(2*2) Dense(512) + Dense(256) + Drop(0.5)	42.28%
256*256	128	30000	按了好多次	RandomFlip(vertical) Conv2D(16,3*3) + MaxPooling2D(2*2) Conv2D(32,3*3) + MaxPooling2D(2*2) Conv2D(64,3*3) + MaxPooling2D(2*2) Conv2D(128,3*3) + MaxPooling2D(2*2) Dense(1024) + Drop(0.5)	45.39%

[心得]

好難... 但從完全看不懂範例程式碼，但最後大概了解整個架構，其實蠻有成就感的，尤其是最一開始根本不知從何下手，model真的是盲調，跟大海撈針一樣，到後面卻可以大概有個方向，知道怎麼調整比較有效率等等，蠻有趣的啦!

最累的就是看網路上的人都有做資料增強擴充訓練集，但不知為何會一直出錯，雖然最後沒能做出來，但至少也是有學到新的東西；另外也了解到ResNet等知名的模型真的很複雜，看都看不懂...

最後不知道能不能許個願，之後的作業能不能也給20天的時間寫，不然真的寫不完XD

[一些過程截圖]

其中一種Model的組合:

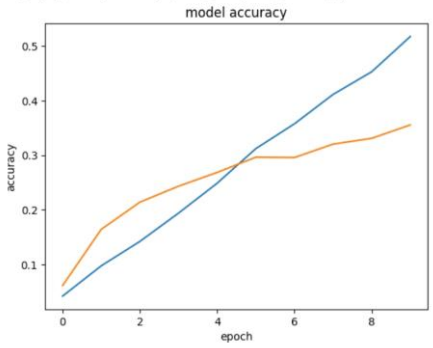
```
# model = None
model = keras.Sequential(
        [
                keras.Input(shape=input_shape),

                layers.Conv2D(16, kernel_size=(3, 3), activation="relu"), #捲積層
                layers.MaxPooling2D(pool_size=(2, 2)), #池化層

                layers.Conv2D(32, kernel_size=(3, 3), activation="relu"), #捲積層
                layers.MaxPooling2D(pool_size=(2, 2)), #池化層

                layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
                layers.MaxPooling2D(pool_size=(2, 2)),

                # layers.Conv2D(64, kernel_size=(5, 5), activation="relu"),
                # layers.MaxPooling2D(pool_size=(3, 3)),

                layers.Flatten(),
                layers.Dense(512, activation="relu"), #全連接層

                layers.Dropout(0.5),
                layers.Dense(num_classes, activation="softmax"),
        ]
)
```

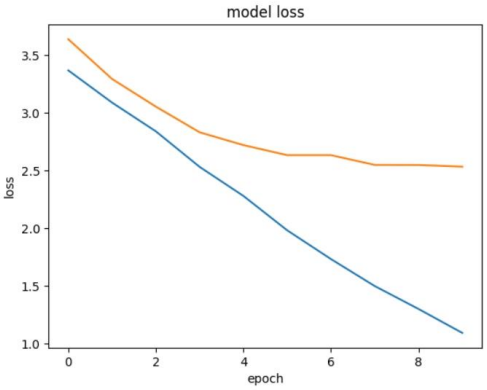
```
[20] # 讀入測試資料並評估模型

     test_ds = make_dataset(test_dir)
     test_ds = test_ds.batch(BATCH_SIZE)


     score = model.evaluate(test_ds)
     # score = model.evaluate(x_test, test_label, verbose=0)

     print("Test loss:", score[0])
     print("Test accuracy:", score[1])
```

5秒

```
7/7 [==============================] - 3s 120ms/step - loss: 2.7025 - accuracy: 0.3257
Test loss: 2.7024712562561035
Test accuracy: 0.3257485032081604
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```





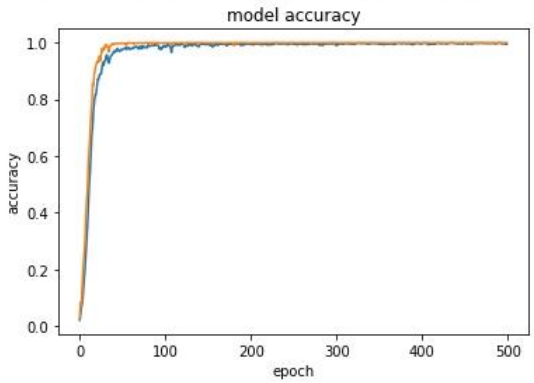
Epoch = 500

[95]:
```
# todo
EPOCHS = 500 #None

##########
# todo #
##########
# model.compile 決定 learning strategy、loss calculator


# onehot編碼用categorical_crossentropy
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])


history = model.fit(train_ds, epochs=EPOCHS, validation_data=val_ds, class_weight=class_weights)
```

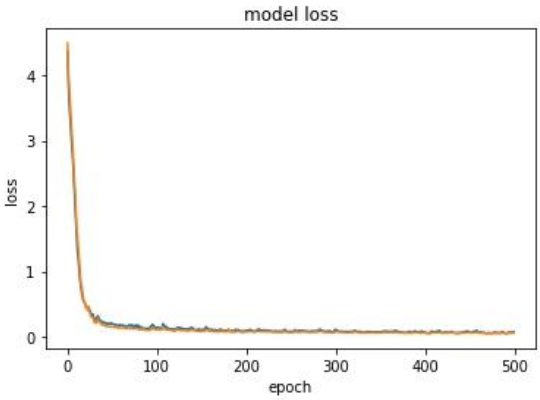
```
Epoch 1/500
47/47 [==============================] - 21s 277ms/step - loss: 4.3591 - accuracy: 0.0213 - val_loss: 4.4907 - val_accuracy: 0.0306
Epoch 2/500
47/47 [==============================] - 19s 270ms/step - loss: 3.8577 - accuracy: 0.0439 - val_loss: 4.0227 - val_accuracy: 0.0831
Epoch 3/500
47/47 [==============================] - 21s 280ms/step - loss: 3.5477 - accuracy: 0.0700 - val_loss: 3.7755 - val_accuracy: 0.0745
Epoch 4/500
47/47 [==============================] - 20s 286ms/step - loss: 3.3206 - accuracy: 0.0828 - val_loss: 3.5433 - val_accuracy: 0.1509
Epoch 5/500
47/47 [==============================] - 20s 277ms/step - loss: 3.1113 - accuracy: 0.1193 - val_loss: 3.2904 - val_accuracy: 0.2055
Epoch 6/500
47/47 [==============================] - 22s 310ms/step - loss: 2.8886 - accuracy: 0.1636 - val_loss: 3.0776 - val_accuracy: 0.2520
Epoch 7/500
47/47 [==============================] - 20s 282ms/step - loss: 2.6952 - accuracy: 0.1980 - val_loss: 2.8372 - val_accuracy: 0.2945
Epoch 8/500
47/47 [==============================] - 23s 328ms/step - loss: 2.3962 - accuracy: 0.2495 - val_loss: 2.5844 - val_accuracy: 0.3743
Epoch 9/500
47/47 [==============================] - 21s 295ms/step - loss: 2.1347 - accuracy: 0.3034 - val_loss: 2.3253 - val_accuracy: 0.4395
Epoch 10/500
47/47 [==============================] - 23s 327ms/step - loss: 1.8728 - accuracy: 0.3547 - val_loss: 2.0607 - val_accuracy: 0.4820
Epoch 11/500
```

[96]:
```
print(history.history.keys())

plt.plot(history.history["accuracy"])
plt.plot(history.history["val_accuracy"])
plt.title("model accuracy")
plt.ylabel("accuracy")
plt.xlabel("epoch")
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



[97]:
```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("model loss")
plt.ylabel("loss")
plt.xlabel("epoch")

plt.show()
```



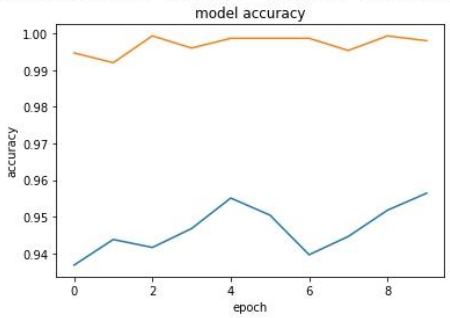
ACC 42.22

[391]:
```
# 讀入測試資料並評估模型
#print(test_dir)
test_ds = make_dataset(test_dir)
test_ds = test_ds.batch(BATCH_SIZE)

score = model.evaluate(test_ds)

print("history")
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

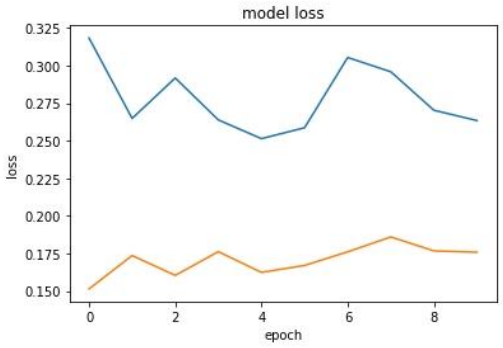
```
7/7 [==============================] - 1s 92ms/step - loss: 3.8329 - accuracy: 0.4228
history
Test loss: 3.83294939994812
Test accuracy: 0.4227544963359833
```

[389]:
```
print(history.history.keys())

plt.plot(history.history["accuracy"])
plt.plot(history.history["val_accuracy"])
plt.title("model accuracy")
plt.ylabel("accuracy")
plt.xlabel("epoch")
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



[390]:
```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("model loss")
plt.ylabel("loss")
plt.xlabel("epoch")

plt.show()
```



[388]:
```
# todo
EPOCHS = 10 #None

##########
# todo #
##########
# model.compile 決定 learning strategy、Loss calculator


# onehot編碼用categorical_crossentropy
# model.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])


model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
history = model.fit(train_ds, epochs=EPOCHS, validation_data=val_ds, class_weight=class_weights)
```

```
Epoch 1/10
47/47 [==============================] - 17s 223ms/step - loss: 0.3183 - accuracy: 0.9368 - val_loss: 0.1514 - val_accuracy: 0.9947
Epoch 2/10
47/47 [==============================] - 15s 206ms/step - loss: 0.2648 - accuracy: 0.9438 - val_loss: 0.1736 - val_accuracy: 0.9920
Epoch 3/10
47/47 [==============================] - 18s 256ms/step - loss: 0.2917 - accuracy: 0.9416 - val_loss: 0.1604 - val_accuracy: 0.9993
Epoch 4/10
47/47 [==============================] - 19s 247ms/step - loss: 0.2638 - accuracy: 0.9468 - val_loss: 0.1762 - val_accuracy: 0.9960
Epoch 5/10
47/47 [==============================] - 16s 221ms/step - loss: 0.2514 - accuracy: 0.9551 - val_loss: 0.1624 - val_accuracy: 0.9987
Epoch 6/10
47/47 [==============================] - 17s 227ms/step - loss: 0.2586 - accuracy: 0.9505 - val_loss: 0.1670 - val_accuracy: 0.9987
Epoch 7/10
47/47 [==============================] - 18s 264ms/step - loss: 0.3054 - accuracy: 0.9397 - val_loss: 0.1761 - val_accuracy: 0.9987
Epoch 8/10
47/47 [==============================] - 19s 259ms/step - loss: 0.2959 - accuracy: 0.9446 - val_loss: 0.1860 - val_accuracy: 0.9953
Epoch 9/10
47/47 [==============================] - 20s 250ms/step - loss: 0.2703 - accuracy: 0.9518 - val_loss: 0.1767 - val_accuracy: 0.9993
Epoch 10/10
47/47 [==============================] - 18s 232ms/step - loss: 0.2634 - accuracy: 0.9564 - val_loss: 0.1758 - val_accuracy: 0.9980
```