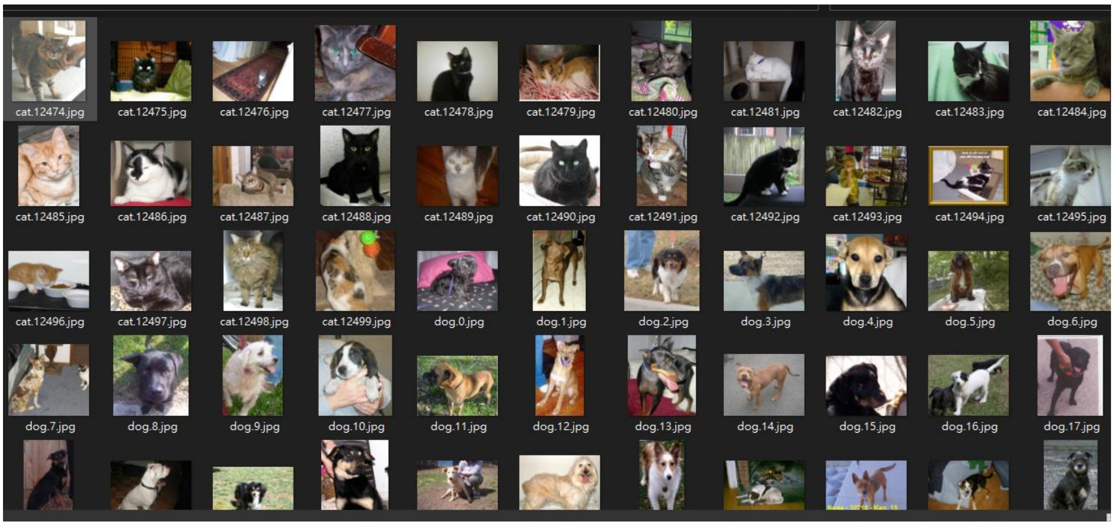
如何透過少量的數據，使建構模型達成與大量數據一致?

B0928012 王晟翰

首先，我們收集資料時往往並不一定能收集大量的資料，所以如何去修改模型，就成為很重要的因素之一。因此，以下目標為，數據量在大約 1/10 的前提下，是否能產生差不多的模型。

第一步，為了更加模擬現實的狀況，我是完全自己重頭收集資料，來更加擬真。我是在微軟的公開資料上，收集貓與狗的圖片，數據量為各 12500 張圖片。



再來就是去做資料上的處理與分類，主要分為訓練，驗證以及測試集。

```python
import os, shutil
# 原始目錄所在的路徑
original_dataset_dir = 'C:\\Users\\hank\\Desktop\\dogs-vs-cats\\train'

# 數據集分類後的目錄
base_dir = 'C:\\Users\\hank\\Desktop\\dogs-vs-cats\\train1'
os.mkdir(base_dir)

# # 訓練、驗證、測試數據集的目錄
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)

# 貓訓練圖片所在目錄
train_cats_dir = os.path.join(train_dir, 'cats')
os.mkdir(train_cats_dir)
```

```python
import os, shutil
# 原始目錄所在的路徑
original_dataset_dir = 'C:\\Users\\hank\\Desktop\\dogs-vs-cats\\train'

# 數據集分類後的目錄
base_dir = 'C:\\Users\\hank\\Desktop\\dogs-vs-cats\\train1'
os.mkdir(base_dir)

# # 訓練、驗證、測試數據集的目錄
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)

# 貓訓練圖片所在目錄
train_cats_dir = os.path.join(train_dir, 'cats')
os.mkdir(train_cats_dir)
```

程式內容，都有很詳細的註解，可以很好理解！

```
#輸出數據集對應目錄下圖片數量
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)))
```

```
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
```

我們可以從這張圖看出，我們先用 **1000** 筆的數據來做訓練，並看看最後訓練成果如何。

```
#神經網路模型構建
from keras import layers
from keras import models
#keras的序貫模型
model = models.Sequential()
#卷積層，卷積核是3*3，激活函數relu
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
#最大池化層
model.add(layers.MaxPooling2D((2, 2)))
#卷積層，卷積核2*2，激活函數relu
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
#最大池化層
model.add(layers.MaxPooling2D((2, 2)))
#卷積層，卷積核是3*3，激活函數relu
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
#最大池化層
model.add(layers.MaxPooling2D((2, 2)))
#卷積層，卷積核是3*3，激活函數relu
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
#最大池化層
model.add(layers.MaxPooling2D((2, 2)))
#flatten層，用於將多維的輸入一維化，用於卷積層和全連接層的過渡
model.add(layers.Flatten())
#全連接，激活函數relu
model.add(layers.Dense(512, activation='relu'))
#全連接，激活函數sigmoid
model.add(layers.Dense(1, activation='sigmoid'))
```

```
#輸出模型各層的參數狀況
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

這裡是比較通用的神經網路模型的建構，右邊則是圖形化這個結構

```
from keras.preprocessing.image import ImageDataGenerator

# 所有圖像將按1/255重新縮放
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # 這是目標目錄
        train_dir,
        # 所有圖像將調整為150x150
        target_size=(150, 150),
        batch_size=20,
        # 因為我們使用二元交叉熵損失，我們需要二元標籤
        class_mode='binary')

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

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

```
#查看上面對於圖片預處理的處理結果
for data_batch, labels_batch in train_generator:
    print('data batch shape:', data_batch.shape)
    print('labels batch shape:', labels_batch.shape)
    break
```

```
data batch shape: (20, 150, 150, 3)
labels batch shape: (20,)
```

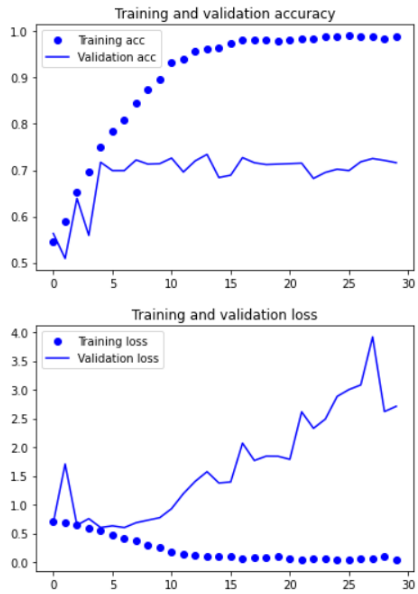
```
#模型訓練過程
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50)
```

```
C:\Users\hank\anaconda3\lib\site-packages\keras\engine\training.py:1915: UserWa
rning: `Model.fit_generator` is deprecated and will be removed in a future vers
ion. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/30
100/100 [==============================] - 35s 156ms/step - loss: 0.7493 - acc:
0.5302 - val_loss: 0.6841 - val_acc: 0.5630
Epoch 2/30
100/100 [==============================] - 6s 57ms/step - loss: 0.6894 - acc:
0.5894 - val_loss: 1.7068 - val_acc: 0.5090
Epoch 3/30
100/100 [==============================] - 6s 57ms/step - loss: 0.7529 - acc:
0.6275 - val_loss: 0.6414 - val_acc: 0.6390
Epoch 4/30
```

把圖形的大小先固定化，做一些地修改，然後最後就是訓練模型了！

```
#保存訓練得到的的模型
model.save('C:\\Users\\hank\\Desktop\\dogs-vs-cats\\cats_and_dogs_small_1.h5')
```

```
#對於模型進行評估，查看預測的準確性
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



如何把我們所建構的模型圖片化呢?

可以透過上方圖片中的下半段程式碼顯示出右邊那張圖，可以發現訓練到中半部，就可以發現，訓練已經過擬合，已經沒有訓練上的意義。重點是，epcho 也才 30 次而已，效果看起來不佳。

這時我們可以看看做 8000 筆的數據最後訓練模型會長怎樣

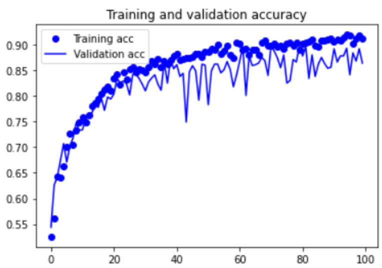
```
#輸出數據集對應目錄下圖片數量
print('total training cat images:', len(os.listdir(train_cats_dir)))
print('total training dog images:', len(os.listdir(train_dogs_dir)))
print('total validation cat images:', len(os.listdir(validation_cats_dir)))
print('total validation dog images:', len(os.listdir(validation_dogs_dir)))
print('total test cat images:', len(os.listdir(test_cats_dir)))
print('total test dog images:', len(os.listdir(test_dogs_dir)))
```

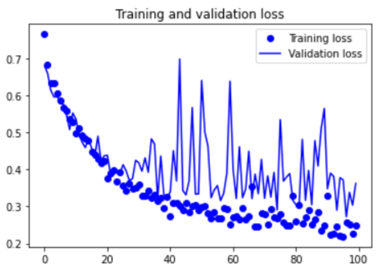
```
total training cat images: 8000
total training dog images: 8000
total validation cat images: 2000
total validation dog images: 2000
total test cat images: 2000
total test dog images: 2000
```

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=100,
      validation_data=validation_generator,
      validation_steps=40)
model.save('C:\\Users\\hank\\Desktop\\dogs-vs-cats\\cats_and_dogs_small_2.h5')
```

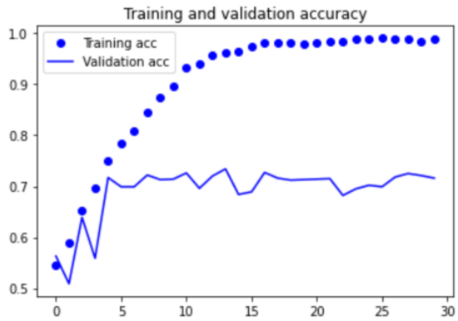
```
Epoch 1/100
100/100 [==============================] - 23s 199ms/step - loss: 0.8884 - acc:
0.4826 - val_loss: 0.7067 - val_acc: 0.5320
Epoch 2/100
100/100 [==============================] - 20s 195ms/step - loss: 0.6896 - acc:
0.5550 - val_loss: 0.6714 - val_acc: 0.6461
Epoch 3/100
100/100 [==============================] - 20s 196ms/step - loss: 0.6749 - acc:
0.5821 - val_loss: 1.0409 - val_acc: 0.5000
Epoch 4/100
100/100 [==============================] - 20s 197ms/step - loss: 0.6635 - acc:
0.6227 - val_loss: 0.6114 - val_acc: 0.6898
Epoch 5/100
```

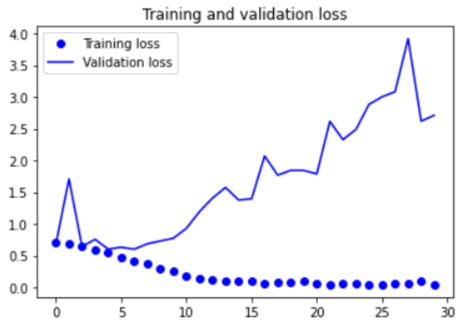
```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





我們來看看 8000 筆最後訓練的樣式，就可以發現訓練 epoch 達到 100 次，就會發現，整體趨勢好看不少，也因此我們目標就是，如何把修改的 1000 筆訓練模型，變得跟 8000 一樣。(可以對比 1000 的圖)





如何在較少資料情況下

還是能使正確率逐漸提高，且 loss 越來越少?

我在網路上找許多資料練習，會發現有個東西叫做數據增強。

什麼是數據增強?

就是透過以下方法，使訓練的圖片，可以有所變形

旋轉 | 反射變換(Rotation/reflection)

翻轉變換(flip) ***縮放變換(zoom)***

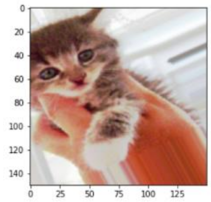
平移變換(shift) ***尺度變換(scale)***

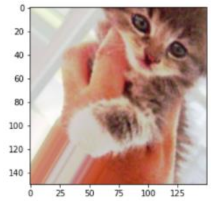
對比度變換(contrast) ***噪聲擾動(noise)***

顏色變化

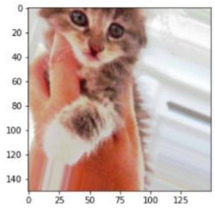
```python
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
    #一個角度值(0-180)，在這個範圍內可以隨機旋轉圖片
      rotation_range=40,
    #範圍(作為總寬度或高度的一部分)，在其中可以隨機地垂直或水平地轉換圖片用於隨機應用
      width_shift_range=0.2,
      height_shift_range=0.2,
    #用於隨機應用剪切轉換
      shear_range=0.2,
    #用於在圖片內部隨機縮放
      zoom_range=0.2,
    #用於水平隨機翻轉一半的圖像—當沒有假設水平不對稱時(例如真實世界的圖片)
      horizontal_flip=True,
    #用於填充新創建像素的策略，它可以在旋轉或寬度/高度移動之後出現
      fill_mode='nearest')
```

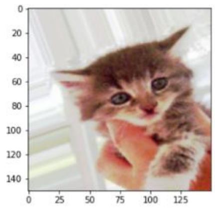
程式碼長這樣，這是我測試幾個參數，比較好的一個





```python
import matplotlib.pyplot as plt
# This is module with image preprocessing utilities
from keras.preprocessing import image
fnames = [os.path.join(train_cats_dir, fname) for fname in os.listdir(train_cat
# We pick one image to "augment"
img_path = fnames[3]
# Read the image and resize it
img = image.load_img(img_path, target_size=(150, 150))
# Convert it to a Numpy array with shape (150, 150, 3)
x = image.img_to_array(img)
# Reshape it to (1, 150, 150, 3)
x = x.reshape((1,) + x.shape)
# The .flow() command below generates batches of randomly transformed images.
# It will loop indefinitely, so we need to `break` the loop at some point!
i = 0
for batch in datagen.flow(x, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 4 == 0:
        break
plt.show()
```



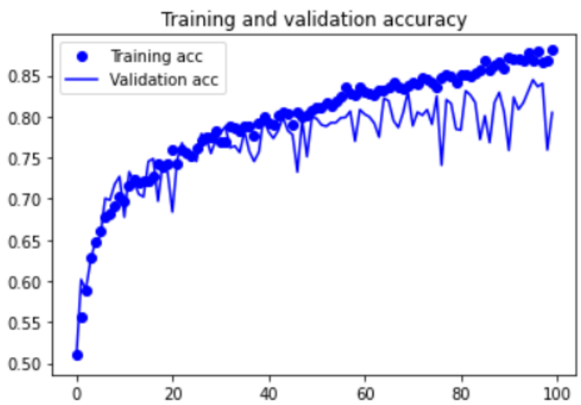


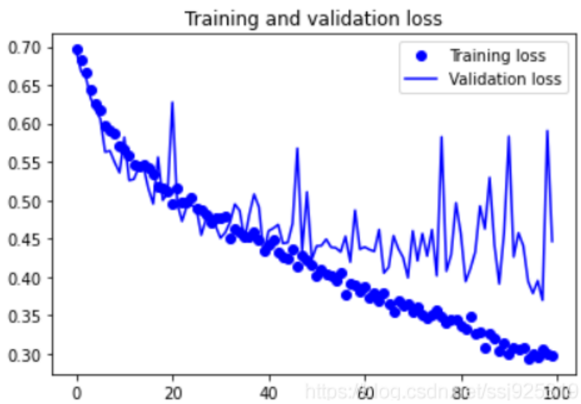
顯示出來，就是把同張貓照片，去做不同的轉換，增加訓練量。

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

```
C:\Users\hank\anaconda3\lib\site-packages\keras\engine\training.py:1915: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/100
100/100 [==============================] - 30s 142ms/step - loss: 1.7851 - acc: 0.4910 - val_loss: 0.6924 - val_acc: 0.5400
Epoch 2/100
100/100 [==============================] - 6s 57ms/step - loss: 0.7495 - acc: 0.5391 - val_loss: 0.6705 - val_acc: 0.6180
```

```python
#對於模型進行評估，查看預測的準確性
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





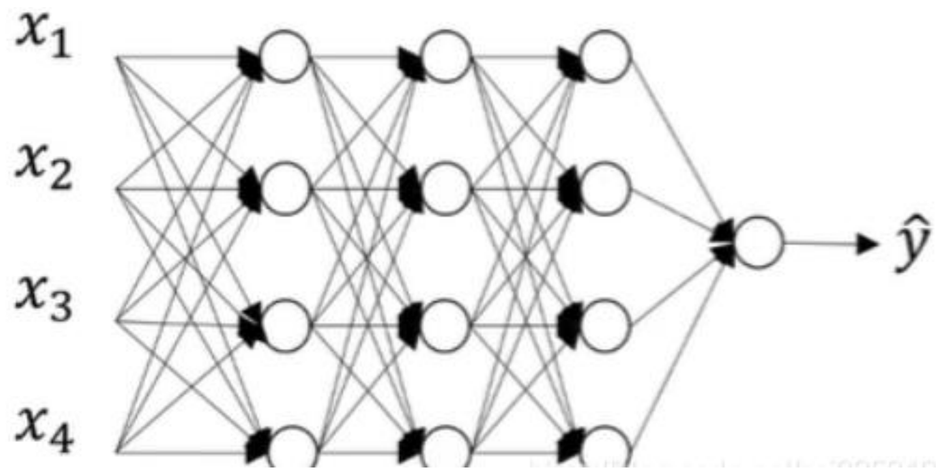
訓練完的結果可以發現，驗證曲線明顯接近訓練曲線很多，但可以看到有許多尖尖的高峰，接下來就是看看能不能處理這個問題。

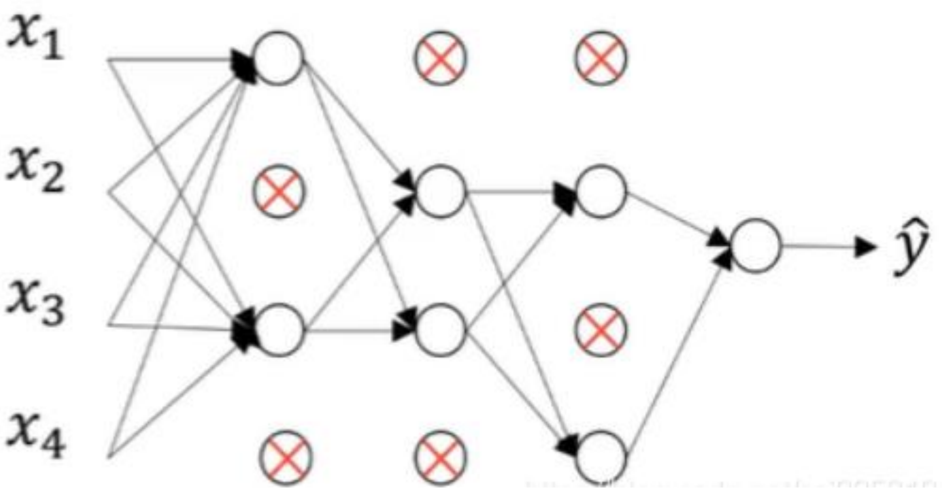
我在網路上找到一個較 dropout 層，聽說可以改善這個問題

什麼是 dropout 層？

Dropout 層在神經網路層當中是用來幹嘛的呢？它是一種可以用於減少神經網絡過擬合的結構，那麼它具體是怎麼實現的呢？

假設下圖是我們用來訓練的原始神經網絡





從上圖我們可以看到一些神經元之間斷開了連接，因此它們被 **dropout** 了！ **dropout** 顧名思義就是被拿掉的意思，正因為我們在神經網絡當中拿掉了一些神經元，所以才叫做 **dropout** 層。

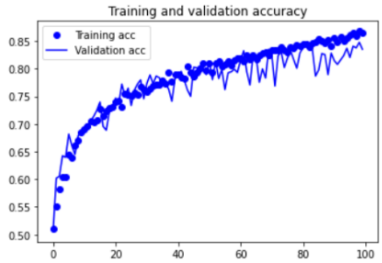
```python
#神經網路模型構建
from keras import layers
from keras import models
#keras的序貫模型
model = models.Sequential()
#卷積層，卷積核是3*3，激活函數relu
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
#最大池化層
model.add(layers.MaxPooling2D((2, 2)))
#卷積層，卷積核2*2，激活函數relu
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
#最大池化層
model.add(layers.MaxPooling2D((2, 2)))
#卷積層，卷積核是3*3，激活函數relu
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
#最大池化層
model.add(layers.MaxPooling2D((2, 2)))
#卷積層，卷積核是3*3，激活函數relu
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
#最大池化層
model.add(layers.MaxPooling2D((2, 2)))
#flatten層，用於將多維的輸入一維化，用於卷積層和全連接層的過渡
model.add(layers.Flatten())
#全連接，激活函數relu
model.add(layers.Dense(512, activation='relu'))
#全連接，激活函數sigmoid
model.add(layers.Dense(1, activation='sigmoid'))
```

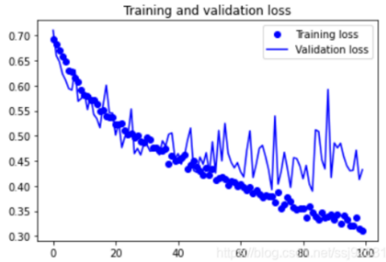
```python
#神經網路模型構建
from keras import layers
from keras import models
#keras的序貫模型
model = models.Sequential()
#卷積層，卷積核是3*3，激活函數relu
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
#最大池化層
model.add(layers.MaxPooling2D((2, 2)))
#卷積層，卷積核2*2，激活函數relu
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
#最大池化層
model.add(layers.MaxPooling2D((2, 2)))
#卷積層，卷積核是3*3，激活函數relu
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
#最大池化層
model.add(layers.MaxPooling2D((2, 2)))
#卷積層，卷積核是3*3，激活函數relu
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
#最大池化層
model.add(layers.MaxPooling2D((2, 2)))
#flatten層，用於將多維的輸入一維化，用於卷積層和全連接層的過渡
model.add(layers.Flatten())

#退出層
model.add(layers.Dropout(0.5))

#全連接，激活函數relu
model.add(layers.Dense(512, activation='relu'))
#全連接，激活函數sigmoid
model.add(layers.Dense(1, activation='sigmoid'))
```

在神經網路模型中，新增這一個項目，退出層，參數設定 0.5

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

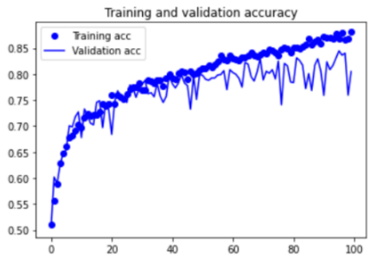
```
C:\Users\hank\anaconda3\lib\site-packages\keras\engine\training.py:1915: UserWa
rning: `Model.fit_generator` is deprecated and will be removed in a future vers
ion. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/100
100/100 [==============================] - 30s 142ms/step - loss: 1.7851 - acc:
0.4910 - val_loss: 0.6924 - val_acc: 0.5400
Epoch 2/100
100/100 [==============================] - 6s 57ms/step - loss: 0.7495 - acc:
0.5391 - val_loss: 0.6705 - val_acc: 0.6180
Epoch 3/100
100/100 [==============================] - 6s 57ms/step - loss: 0.6616 - acc:
0.6177 - val_loss: 0.6234 - val_acc: 0.6520
Epoch 4/100
100/100 [==============================] - 6s 57ms/step - loss: 0.6188 - acc:
```

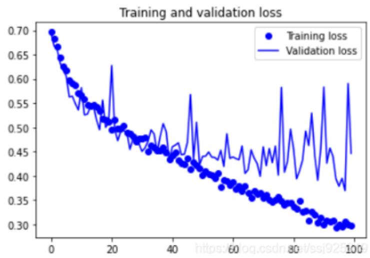




可以發現尖峰明顯少了不少且變低

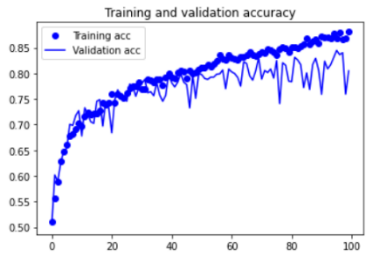
右邊是只用數據分析的圖，可以做比較

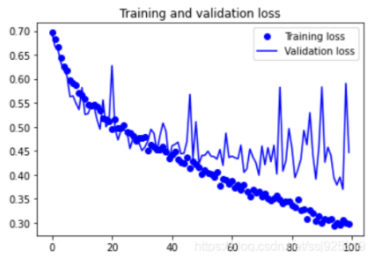




再來看 8000 筆數據的

可以明顯透過我們前面所做的修改，可以使訓練結果筆上 8000 筆的甚至更好。





結論：我們不應該單單要求大量的數據，應該也要能透過少量的數據，竟量分析出更多東西，或者是更好的模型。也透過這次比較發現，那怕數據量少了接近 10 倍，仍然能做出差不多甚至更好的模型。

可以再看看最一開始訓練的樣子

，很難想像，是同樣的數據訓練的。