

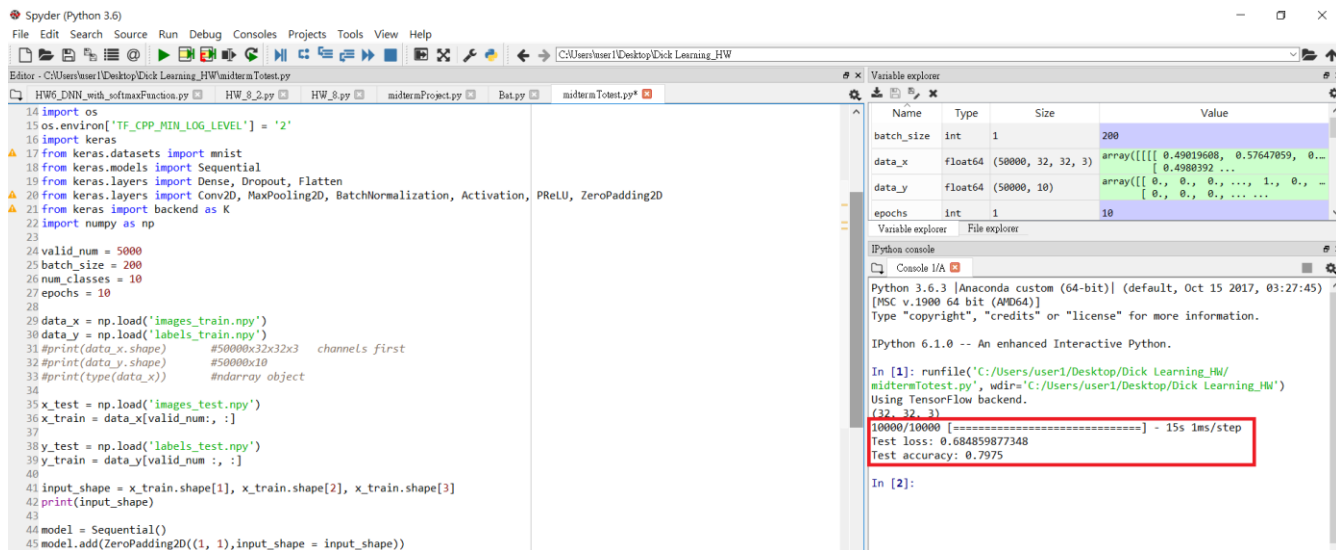
機器學習期中結報

姓名:林士恩

系級:電機 108 甲

學號:B043011031

一、測試集正確率：



The screenshot shows the Spyder Python IDE with a file named `midtermTotest.py` open. The code defines a Keras model with the following layers: `Conv2D(12, kernel_size=(3, 3), activation='relu')`, `MaxPooling2D(pool_size=(2, 2))`, `Conv2D(64, (3, 3), activation='relu')`, `MaxPooling2D(pool_size=(2, 2))`, `Flatten()`, `Dense(128, activation='relu')`, `Dense(500, activation='relu')`, and `Dense(num_classes, activation='softmax')`. The model is trained for 10 epochs with a batch size of 200. The console output shows the training progress and the final test accuracy of 0.7975.

```
14 import os
15 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
16 import keras
17 from keras.datasets import mnist
18 from keras.models import Sequential
19 from keras.layers import Dense, Dropout, Flatten
20 from keras.layers import Conv2D, MaxPooling2D, BatchNormalization, Activation, PReLU, ZeroPadding2D
21 from keras import backend as K
22 import numpy as np
23
24 valid_num = 5000
25 batch_size = 200
26 num_classes = 10
27 epochs = 10
28
29 data_x = np.load('images_train.npy')
30 data_y = np.load('labels_train.npy')
31 print(data_x.shape) #50000x32x32x3 channels first
32 print(data_y.shape) #50000x10
33 print(type(data_x)) #ndarray object
34
35 x_test = np.load('images_test.npy')
36 x_train = data_x[valid_num:, :]
37
38 y_test = np.load('labels_test.npy')
39 y_train = data_y[valid_num:, :]
40
41 input_shape = x_train.shape[1], x_train.shape[2], x_train.shape[3]
42 print(input_shape)
43
44 model = Sequential()
45 model.add(ZeroPadding2D((1, 1), input_shape = input_shape))
```

Variable explorer:

Name	Type	Size	Value
batch_size	int	1	200
data_x	float64	(50000, 32, 32, 3)	array([[[[0.49019608, 0.57647859, 0....
data_y	float64	(50000, 10)	array([[0., 0., ..., 1., 0., ...
epochs	int	1	10

Python console:

```
Python 3.6.3 [Anaconda custom (64-bit)] (default, Oct 15 2017, 03:27:45)
[MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

Python 6.1.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/user1/Desktop/Dick Learning_HW/
midtermTotest.py', wdir='C:/Users/user1/Desktop/Dick Learning_HW')
Using TensorFlow backend.
(32, 32, 3)
10000/10000 [=====] - 15s 1ms/step
Test loss: 0.684859877348
Test accuracy: 0.7975

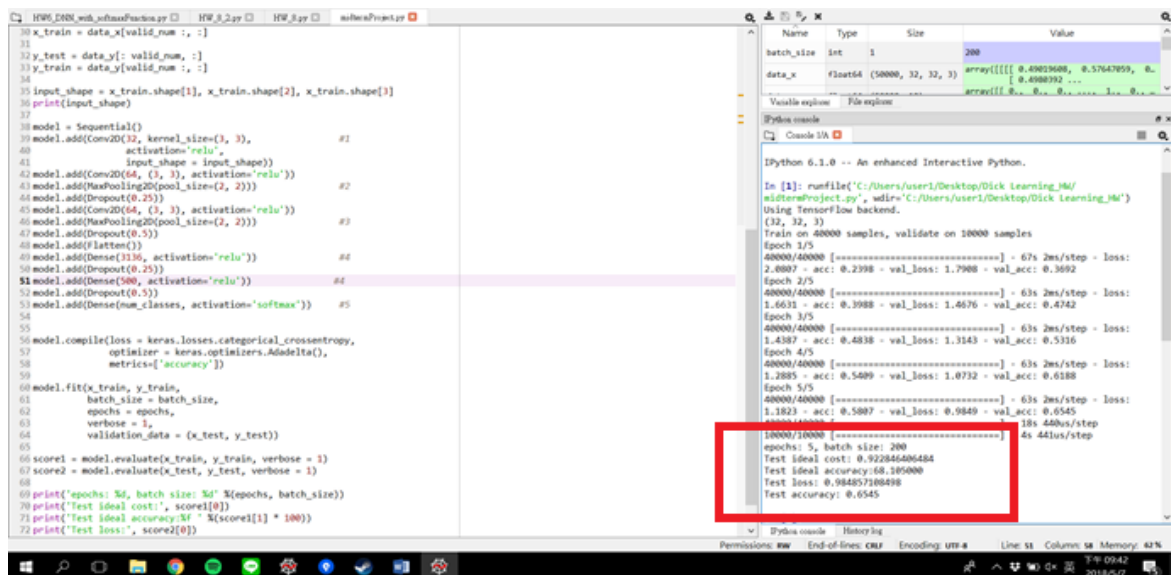
In [2]:
```

二、模型討論與分析：

這次訓練的模型大致上可以分成 2 種，一種是類似 VGG 的架構，另一種則是只有 3 層 CNN 加上 2 層 DNN 的混和架構。一開始我都是用後者來訓練模型。

1. 第一種 model 是 3 層 CNN 加上 2 層 DNN(based on HW8)

(資料切割成 4000 筆做 training 10000 筆做 testing, batch size = 200)



The screenshot shows the Spyder Python IDE with a file named `midtermProject.py` open. The code defines a Keras model with the following layers: `Conv2D(12, kernel_size=(3, 3), activation='relu')`, `MaxPooling2D(pool_size=(2, 2))`, `Conv2D(64, (3, 3), activation='relu')`, `MaxPooling2D(pool_size=(2, 2))`, `Flatten()`, `Dense(128, activation='relu')`, `Dense(500, activation='relu')`, and `Dense(num_classes, activation='softmax')`. The model is trained for 5 epochs with a batch size of 200. The console output shows the training progress and the final test accuracy of 0.6545.

```
10 x_train = data_x[valid_num:, :]
11
12 y_test = data_y[valid_num:, :]
13 y_train = data_y[valid_num:, :]
14
15 input_shape = x_train.shape[1], x_train.shape[2], x_train.shape[3]
16 print(input_shape)
17
18 model = Sequential()
19 model.add(Conv2D(12, kernel_size=(3, 3),
20 activation='relu',
21 input_shape = input_shape)) #1
22 model.add(MaxPooling2D(pool_size=(2, 2))) #2
23 model.add(Conv2D(64, (3, 3), activation='relu')) #2
24 model.add(MaxPooling2D(pool_size=(2, 2))) #3
25 model.add(Flatten()) #4
26 model.add(Dense(128, activation='relu')) #4
27 model.add(Dense(500, activation='relu')) #4
28 model.add(Dense(num_classes, activation='softmax')) #5
29
30 model.compile(loss = keras.losses.categorical_crossentropy,
31 optimizer = keras.optimizers.Adadelta(),
32 metrics=['accuracy'])
33
34 model.fit(x_train, y_train,
35 batch_size = batch_size,
36 epochs = epochs,
37 verbose = 1,
38 validation_data = (x_test, y_test))
39
40 score1 = model.evaluate(x_train, y_train, verbose = 1)
41 score2 = model.evaluate(x_test, y_test, verbose = 1)
42
43 print('epochs: %d, batch size: %d' % (epochs, batch_size))
44 print('Test ideal cost: %f, score1[0]: %f' % (score1[0], score1[1] * 100))
45 print('Test loss: %f, score2[0]: %f' % (score2[0], score2[1] * 100))
```

Variable explorer:

Name	Type	Size	Value
batch_size	int	1	200
data_x	float64	(50000, 32, 32, 3)	array([[[[0.49019608, 0.57647859, 0....
data_y	float64	(50000, 10)	array([[0., 0., ..., 1., 0., ...
epochs	int	1	10

Python console:

```
Python 6.1.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/user1/Desktop/Dick Learning_HW/
midtermProject.py', wdir='C:/Users/user1/Desktop/Dick Learning_HW')
Using TensorFlow backend.
(32, 32, 3)
Train on 40000 samples, validate on 10000 samples
Epoch 1/5
40000/40000 [=====] - 67s 2ms/step - loss: 2.0807 - acc: 0.2398 - val_loss: 1.7908 - val_acc: 0.3692
Epoch 2/5
40000/40000 [=====] - 63s 2ms/step - loss: 1.6631 - acc: 0.3988 - val_loss: 1.4676 - val_acc: 0.4742
Epoch 3/5
40000/40000 [=====] - 63s 2ms/step - loss: 1.4387 - acc: 0.4838 - val_loss: 1.3143 - val_acc: 0.5316
Epoch 4/5
40000/40000 [=====] - 63s 2ms/step - loss: 1.2885 - acc: 0.5489 - val_loss: 1.0732 - val_acc: 0.6188
Epoch 5/5
40000/40000 [=====] - 63s 2ms/step - loss: 1.1823 - acc: 0.5887 - val_loss: 0.9849 - val_acc: 0.6545
10000/10000 [=====] - 4s 440us/step
epochs: 5, batch size: 200
Test ideal cost: 0.92846406484
Test ideal accuracy: 68.185000
Test loss: 0.984857188498
Test accuracy: 0.6545
```

可以觀察到在 epoch = 5 之下，真實正確率高達 65.5%，且與訓練正確率僅相差 2.5%，完全不可能有 overfitting 的問題。之後希望把 accuracy 提高，所以增加 epoch，得到的結果如下：

```
Console I/A
40000/40000 [=====] - 63s 2ms/step - loss: 1.4614 - acc: 0.4763 - val_loss: 1.3102 - val_acc: 0.5256
Epoch 4/10
40000/40000 [=====] - 63s 2ms/step - loss: 1.3139 - acc: 0.5304 - val_loss: 1.2075 - val_acc: 0.5713
Epoch 5/10
40000/40000 [=====] - 63s 2ms/step - loss: 1.1884 - acc: 0.5792 - val_loss: 1.0721 - val_acc: 0.6184
Epoch 6/10
40000/40000 [=====] - 82s 2ms/step - loss: 1.0877 - acc: 0.6142 - val_loss: 0.9235 - val_acc: 0.6780
Epoch 7/10
40000/40000 [=====] - 91s 2ms/step - loss: 0.9992 - acc: 0.6468 - val_loss: 0.9663 - val_acc: 0.6621
Epoch 8/10
40000/40000 [=====] - 91s 2ms/step - loss: 0.9269 - acc: 0.6739 - val_loss: 0.8219 - val_acc: 0.7148
Epoch 9/10
40000/40000 [=====] - 92s 2ms/step - loss: 0.8596 - acc: 0.6999 - val_loss: 0.8302 - val_acc: 0.7080
Epoch 10/10
40000/40000 [=====] - 92s 2ms/step - loss: 0.8044 - acc: 0.7193 - val_loss: 0.7518 - val_acc: 0.7420
40000/40000 [=====] - 21s 522us/step
10000/10000 [=====] - 5s 531us/step
epochs: 10, batch size: 200
Test ideal cost: 0.55372608335
Test ideal accuracy: 82.142500
Test loss: 0.751780521393
Test accuracy: 0.742

In [2]:
Python console History log
ons: RW End-of-lines: CRLF Encoding: UTF-8 Line: 20 Column: 1 Memory: 81% 下午 11:02 2018/5/7
```

```
0.6240 - acc: 0.7827 - val_loss: 0.6719 - val_acc: 0.7665
Epoch 15/20
40000/40000 [=====] - 89s 2ms/step - loss: 0.5830 - acc: 0.7971 - val_loss: 0.6703 - val_acc: 0.7684
Epoch 16/20
40000/40000 [=====] - 90s 2ms/step - loss: 0.5455 - acc: 0.8085 - val_loss: 0.6740 - val_acc: 0.7719
Epoch 17/20
40000/40000 [=====] - 90s 2ms/step - loss: 0.5090 - acc: 0.8223 - val_loss: 0.6624 - val_acc: 0.7709
Epoch 18/20
40000/40000 [=====] - 90s 2ms/step - loss: 0.4732 - acc: 0.8345 - val_loss: 0.6487 - val_acc: 0.7797
Epoch 19/20
40000/40000 [=====] - 90s 2ms/step - loss: 0.4443 - acc: 0.8437 - val_loss: 0.6687 - val_acc: 0.7762
Epoch 20/20
40000/40000 [=====] - 91s 2ms/step - loss: 0.4168 - acc: 0.8547 - val_loss: 0.6615 - val_acc: 0.7779
40000/40000 [=====] - 33s 822us/step
10000/10000 [=====] - 7s 669us/step
epochs: 20, batch size: 200
Test ideal cost: 0.159053990713
Test ideal accuracy: 96.187500
Test loss: 0.66147166276
Test accuracy: 0.7779

In [2]: runfile('C:/Users/user1/Desktop/Dick Learning_HW/
midtermProject.py', wdir='C:/Users/user1/Desktop/Dick Learning_HW')

Kernel died, restarting
Python console History log
ons: RW End-of-lines: CRLF Encoding: UTF-8 Line: 48 Column: 21 Memory: 33% 下午 11:53 2018/5/7
```

(epoch = 10) (epoch = 20)

會發現到了 epoch = 20 的時候與 epoch 的時候 accuracy 相差僅 5%，但是 epoch = 20 時 testing 與 ideal accuracy 相差將近 20%，發生 overfitting，且 testing accuracy 已經非常接近最佳值，認定這個 model 沒有做得很好。

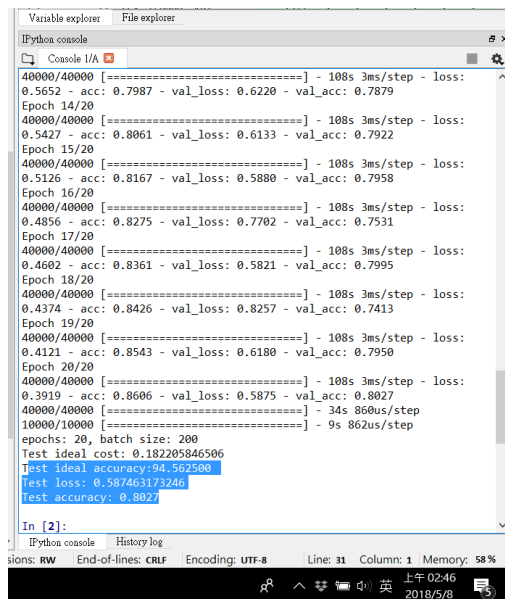
若是想要改善 epoch = 20 時的 overfitting，並且減少對於 initialization 的依賴性，在層與層之間加上 BatchNormalization，並在試著跑了一次，出來的結果 testing 正確率有上升到 79%，ideal accuracy 也有降低一些，overfitting 的情況有所改善，但是正確率也是卡在將近 79%，結果如下：

```
32 y_test = data_y[valid_num, :]
33 y_train = data_y[valid_num, :]
34
35 input_shape = x_train.shape[1], x_train.shape[2], x_train.shape[3]
36 print(input_shape)
37
38 model = Sequential()
39 model.add(Conv2D(32, kernel_size=(3, 3), #1
40               activation='relu',
41               input_shape = input_shape))
42 model.add(Conv2D(64, (3, 3), activation='relu'))
43 model.add(MaxPooling2D(pool_size=(2, 2))) #2
44 model.add(Dropout(0.25))
45 model.add(Conv2D(64, (3, 3), activation='relu'))
46 model.add(MaxPooling2D(pool_size=(2, 2))) #3
47 model.add(Dropout(0.5))
48 model.add(Flatten())
49 model.add(BatchNormalization()) #BatchNorm
50 model.add(Dense(3136, activation='relu')) #4
51 model.add(Dropout(0.25))
52 model.add(BatchNormalization()) #BatchNorm
53 model.add(Dense(500, activation='relu'))
54 model.add(Dropout(0.5))
55 model.add(Dense(num_classes, activation='softmax')) #5
56
57
58 model.compile(loss = keras.losses.categorical_crossentropy,
59               optimizer = keras.optimizers.Adadelta(),
60               metrics=['accuracy'])
61
62 model.fit(x_train, y_train,
63         batch_size = batch_size,
64         epochs = epochs,
65         verbose = 1,
```

```
Console I/A
40000/40000 [=====] - 71s 2ms/step - loss: 0.6636 - acc: 0.7656 - val_loss: 0.6748 - val_acc: 0.7729
Epoch 14/20
40000/40000 [=====] - 71s 2ms/step - loss: 0.6466 - acc: 0.7728 - val_loss: 0.6603 - val_acc: 0.7763
Epoch 15/20
40000/40000 [=====] - 71s 2ms/step - loss: 0.6232 - acc: 0.7787 - val_loss: 0.6883 - val_acc: 0.7731
Epoch 16/20
40000/40000 [=====] - 71s 2ms/step - loss: 0.6033 - acc: 0.7876 - val_loss: 0.6797 - val_acc: 0.7694
Epoch 17/20
40000/40000 [=====] - 71s 2ms/step - loss: 0.5811 - acc: 0.7970 - val_loss: 0.6420 - val_acc: 0.7831
Epoch 18/20
40000/40000 [=====] - 71s 2ms/step - loss: 0.5618 - acc: 0.8022 - val_loss: 0.6767 - val_acc: 0.7703
Epoch 19/20
40000/40000 [=====] - 71s 2ms/step - loss: 0.5459 - acc: 0.8076 - val_loss: 0.6623 - val_acc: 0.7740
Epoch 20/20
40000/40000 [=====] - 71s 2ms/step - loss: 0.5234 - acc: 0.8160 - val_loss: 0.6187 - val_acc: 0.7921
40000/40000 [=====] - 19s 467us/step
10000/10000 [=====] - 5s 470us/step
epochs: 20, batch size: 200
Test ideal cost: 0.263105883944
Test ideal accuracy: 92.865000
Test loss: 0.618665984917
Test accuracy: 0.7921

In [2]:
Python console History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 53 Column: 54 Memory: 61% 上午 12:21 2018/5/8
```

之後又試著改變原本的架構一點，得到最好的正確率如下：



```
Variable explorer | File explorer
Python console
Console 1/A
40000/40000 [=====] - 108s 3ms/step - loss: 0.5652 - acc: 0.7987 - val_loss: 0.6220 - val_acc: 0.7879
Epoch 14/20
40000/40000 [=====] - 108s 3ms/step - loss: 0.5427 - acc: 0.8061 - val_loss: 0.6133 - val_acc: 0.7922
Epoch 15/20
40000/40000 [=====] - 108s 3ms/step - loss: 0.5126 - acc: 0.8167 - val_loss: 0.5880 - val_acc: 0.7958
Epoch 16/20
40000/40000 [=====] - 108s 3ms/step - loss: 0.4856 - acc: 0.8275 - val_loss: 0.7702 - val_acc: 0.7531
Epoch 17/20
40000/40000 [=====] - 108s 3ms/step - loss: 0.4602 - acc: 0.8361 - val_loss: 0.5821 - val_acc: 0.7995
Epoch 18/20
40000/40000 [=====] - 108s 3ms/step - loss: 0.4374 - acc: 0.8426 - val_loss: 0.8257 - val_acc: 0.7413
Epoch 19/20
40000/40000 [=====] - 108s 3ms/step - loss: 0.4121 - acc: 0.8543 - val_loss: 0.6180 - val_acc: 0.7950
Epoch 20/20
40000/40000 [=====] - 108s 3ms/step - loss: 0.3919 - acc: 0.8606 - val_loss: 0.5875 - val_acc: 0.8027
40000/40000 [=====] - 34s 860us/step
10000/10000 [=====] - 9s 862us/step
epochs: 20, batch size: 200
Test ideal cost: 0.182205846506
Test ideal accuracy: 94.562500
Test loss: 0.587463173246
Test accuracy: 0.8027
In [2]:
```

(有 80% 的正確率)

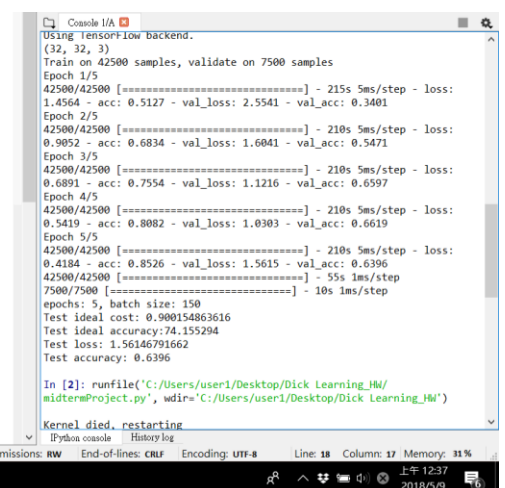
就算修改成(Training:45000, testing:5000 資料), 正確率並沒有因此而上升, 所以我認定這個 model 不管怎麼樣, 極限都將近 79 ~ 80%之間徘徊, 如果想要更高的正確率的話, 就需要修改 CNN 或 DNN 的整體架構。

之後參考了上課講義, 發現其實 CNN 架構在 Convolution layer 間如果訓練資料的維度如果變化太快的話, 會導致訓練不太起來, 所以我認為之前的問題可能就出現在這, 就加上 zero padding 來維持資料量, 甚至增加資料量, 並疊加 2 層 convolution layer 之後才經過 maxpooling 來減少資料數量, 並在 CNN 與 DNN 銜接的 hidden units 數量與 CNN 最後一層輸出的資料做 flatten 後的大小匹配, 所以不會像之前一樣少掉部分的資訊。得到的結果如下:

(在 4 層 CNN + 3 層 DNN 之下且 Training: 42500 Testing: 7500)

在 epoch = 5 之下會發現結果並沒有比之前來的好。

```
28 x_test = data_x[: valid_num, :]
29 x_train = data_x[valid_num :, :]
30
31 y_test = data_y[: valid_num, :]
32 y_train = data_y[valid_num :, :]
33
34 input_shape = x_train.shape[1], x_train.shape[2], x_train.shape[3]
35 print(input_shape)
36
37 model = Sequential()
38 model.add(ZeroPadding2D((1, 1), input_shape = input_shape))
39 model.add(Conv2D(32, (3, 3)))
40 model.add(BatchNormalization()) #BatchNorm
41 model.add(PReLU())
42 model.add(ZeroPadding2D((1, 1)))
43
44 model.add(Conv2D(32, (3, 3)))
45 model.add(BatchNormalization()) #BatchNorm
46 model.add(PReLU())
47 model.add(MaxPooling2D(pool_size=(2, 2))) #2
48 model.add(Dropout(0.5))
49 model.add(ZeroPadding2D((1, 1)))
50
51 model.add(Conv2D(64, (3, 3)))
52 model.add(BatchNormalization()) #BatchNorm
53 model.add(PReLU())
54 model.add(ZeroPadding2D((1, 1)))
55 model.add(Dropout(0.5))
56
57 model.add(Conv2D(64, (3, 3)))
58 model.add(BatchNormalization()) #BatchNorm
59 model.add(PReLU())
```



```
Console 1/A
Using tensorflow backend.
(32, 32, 3)
Train on 42500 samples, validate on 7500 samples
Epoch 1/5
42500/42500 [=====] - 215s 5ms/step - loss: 1.4564 - acc: 0.5127 - val_loss: 2.5541 - val_acc: 0.3401
Epoch 2/5
42500/42500 [=====] - 210s 5ms/step - loss: 0.9052 - acc: 0.6834 - val_loss: 1.6041 - val_acc: 0.5471
Epoch 3/5
42500/42500 [=====] - 210s 5ms/step - loss: 0.6891 - acc: 0.7554 - val_loss: 1.1216 - val_acc: 0.6597
Epoch 4/5
42500/42500 [=====] - 210s 5ms/step - loss: 0.5419 - acc: 0.8082 - val_loss: 1.0303 - val_acc: 0.6619
Epoch 5/5
42500/42500 [=====] - 210s 5ms/step - loss: 0.4184 - acc: 0.8526 - val_loss: 1.5615 - val_acc: 0.6396
42500/42500 [=====] - 55s 1ms/step
7500/7500 [=====] - 10s 1ms/step
epochs: 5, batch size: 150
Test ideal cost: 0.900154863616
Test ideal accuracy: 74.155294
Test loss: 1.56146791662
Test accuracy: 0.6396
In [2]: runfile('C:/Users/user1/Desktop/Dick Learning_HM/midtermProject.py', wdir='C:/Users/user1/Desktop/Dick Learning_HM')
Kernel died, restarting
Python console | History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 18 Column: 17 Memory: 31 %
2018/5/9
```

但如果把 DNN 減少成只有 2 層, 並重新調配成 Training: 45000 Testing:

5000，在 epoch = 5 之下，會發現正確率較之前的架構上升了不少，testing 正確率高達 72%。

The screenshot shows a Jupyter Notebook with a Keras model architecture and its training results. The model architecture is defined in the code cell, and the training results are shown in the output cell.

```
41 model.add(PReLU())
42 model.add(ZeroPadding2D((1,1)))
43
44 model.add(Conv2D(32, (3, 3)))
45 model.add(BatchNormalization()) #BatchNorm
46 model.add(PReLU())
47 model.add(MaxPooling2D(pool_size=(2, 2))) #2
48 model.add(ZeroPadding2D((1,1)))
49
50 model.add(Conv2D(64, (3, 3)))
51 model.add(BatchNormalization()) #BatchNorm
52 model.add(PReLU())
53 model.add(ZeroPadding2D((1,1)))
54 model.add(Dropout(0.5))
55
56 model.add(Conv2D(64, (3, 3)))
57 model.add(BatchNormalization()) #BatchNorm
58 model.add(PReLU())
59 model.add(MaxPooling2D(pool_size=(2, 2))) #3
60 model.add(Dropout(0.5))
61 model.add(Flatten())
62
63 model.add(Dense(4096)) #4
64 model.add(BatchNormalization()) #BatchNorm
65 model.add(PReLU())
66 model.add(Dropout(0.5))
67
68 model.add(Dense(2048)) #4
69 model.add(BatchNormalization()) #BatchNorm
70 model.add(PReLU())
71 model.add(Dropout(0.5))
72
73 model.add(Dense(1000)) #4
74 model.add(BatchNormalization()) #BatchNorm
75 model.add(PReLU())
76 model.add(Dropout(0.5))
77
78 model.add(Dense(num_classes, activation='softmax')) #5
79
80
81 model.compile(loss = keras.losses.categorical_crossentropy,
82               optimizer = keras.optimizers.Adadelta(),
83               metrics=['accuracy'])
```

The output cell shows the training results for 5 epochs. The model is trained on 45000 samples and validated on 5000 samples. The training results are as follows:

Epoch	Train on 45000 samples, validate on 5000 samples
Epoch 1/5	45000/45000 [=====] - 197s 4ms/step - loss: 1.4266 - acc: 0.5262 - val_loss: 1.4822 - val_acc: 0.5276
Epoch 2/5	45000/45000 [=====] - 193s 4ms/step - loss: 0.8795 - acc: 0.6922 - val_loss: 0.9326 - val_acc: 0.6758
Epoch 3/5	45000/45000 [=====] - 193s 4ms/step - loss: 0.6686 - acc: 0.7648 - val_loss: 0.9577 - val_acc: 0.6862
Epoch 4/5	45000/45000 [=====] - 192s 4ms/step - loss: 0.5164 - acc: 0.8191 - val_loss: 0.8198 - val_acc: 0.7386
Epoch 5/5	45000/45000 [=====] - 192s 4ms/step - loss: 0.4044 - acc: 0.8568 - val_loss: 0.9680 - val_acc: 0.7230

The output also shows the test results for 5000 samples:

Test Results
5000/5000 [=====] - 6s 1ms/step
epochs: 5, batch size: 200
Test ideal cost: 0.393585364321
Test ideal accuracy: 85.948889
Test loss: 0.967969408226
Test accuracy: 0.723

The screenshot shows a Jupyter Notebook with a Keras model architecture and its training results. The model architecture is defined in the code cell, and the training results are shown in the output cell.

```
17 valid_num = 5000
18 batch_size = 200
19 num_classes = 10
20 epochs = 5
21
22 data_x = np.load('images_train.npy')
23 data_y = np.load('labels_train.npy')
24 #print(data_x.shape) #50000x32x32x3 channels first
25 #print(data_y.shape) #50000x10
26 #print(type(data_x)) #ndarray object
27
28 x_test = data_x[: valid_num, :]
29 x_train = data_x[valid_num :, :]
30
31 y_test = data_y[: valid_num, :]
32 y_train = data_y[valid_num :, :]
33
34 input_shape = x_train.shape[1], x_train.shape[2], x_train.shape[3]
35 print(input_shape)
36
37 model = Sequential()
38 model.add(ZeroPadding2D((1, 1), input_shape = input_shape))
39 model.add(Conv2D(32, (3, 3)))
40 model.add(BatchNormalization()) #BatchNorm
41 model.add(PReLU())
42 model.add(ZeroPadding2D((1,1)))
43
44 model.add(Conv2D(32, (3, 3)))
45 model.add(BatchNormalization()) #BatchNorm
46 model.add(PReLU())
47 model.add(MaxPooling2D(pool_size=(2, 2))) #2
48 model.add(Dropout(0.5))
49 model.add(ZeroPadding2D((1,1)))
50
51 model.add(Conv2D(64, (3, 3)))
52 model.add(BatchNormalization()) #BatchNorm
53 model.add(PReLU())
```

The output cell shows the training results for 5 epochs. The model is trained on 45000 samples and validated on 5000 samples. The training results are as follows:

Epoch	Train on 45000 samples, validate on 5000 samples
Epoch 1/5	45000/45000 [=====] - 198s 4ms/step - loss: 1.4618 - acc: 0.5117 - val_loss: 1.7744 - val_acc: 0.4214
Epoch 2/5	45000/45000 [=====] - 193s 4ms/step - loss: 0.8984 - acc: 0.6852 - val_loss: 0.9986 - val_acc: 0.6598
Epoch 3/5	45000/45000 [=====] - 193s 4ms/step - loss: 0.6740 - acc: 0.7630 - val_loss: 1.0325 - val_acc: 0.6704
Epoch 4/5	45000/45000 [=====] - 193s 4ms/step - loss: 0.5332 - acc: 0.8124 - val_loss: 1.0256 - val_acc: 0.6956
Epoch 5/5	45000/45000 [=====] - 193s 4ms/step - loss: 0.4181 - acc: 0.8520 - val_loss: 0.8768 - val_acc: 0.7344

The output also shows the test results for 5000 samples:

Test Results
5000/5000 [=====] - 58s 1ms/step
epochs: 5, batch size: 200
Test ideal cost: 0.323370621173
Test ideal accuracy: 88.368889
Test loss: 0.876823072529
Test accuracy: 0.7344

```
61 #model.add(Dropout(0.5))
62 model.add(Flatten())
63
64 model.add(Dense(4096)) #4
65 model.add(BatchNormalization()) #BatchNorm
66 model.add(PReLU())
67 model.add(Dropout(0.5))
68
69 #model.add(Dense(2048)) #4
70 #model.add(BatchNormalization()) #BatchNorm
71 #model.add(PReLU())
72 #model.add(Dropout(0.5))
73
74 model.add(Dense(1000)) #4
75 model.add(BatchNormalization()) #BatchNorm
76 model.add(PReLU())
77 model.add(Dropout(0.5))
78
79 model.add(Dense(num_classes, activation='softmax')) #5
80
81 model.compile(loss = keras.losses.categorical_crossentropy,
82               optimizer = keras.optimizers.Adadelta(),
83               metrics=['accuracy'])
84
85 model.fit(x_train, y_train,
86         batch_size = batch_size,
87         epochs = epochs,
88         verbose = 1,
89         validation_data = (x_test, y_test))
90
91 score1 = model.evaluate(x_train, y_train, verbose = 1)
92 score2 = model.evaluate(x_test, y_test, verbose = 1)
93
94 outfile = 'cnn_model_with_TS45_BS200_EP7'
95 model.save_weights(outfile)
96
97 print('epochs: %d, batch size: %d' % (epochs, batch_size))
98 print('Test ideal cost:', score1[0])
99 print('Test ideal accuracy:%f ' % (score1[1] * 100))
100 print('Test loss:', score2[0])
101 print('Test accuracy:', score2[1])
102
103
```

Name	Type	Size	Value
batch_size	int	1	200
data_x	float64	(50000, 32, 32, 3)	array([[[[0.49019608, 0.57647059, 0.49080392, ...
data_y	float64	(50000, 10)	array([[0., 0., 0., ..., 1., 0., ...
epochs	int	1	7

Epoch 1/7
45000/45000 [=====] - 197s 4ms/step - loss: 1.4050 - acc: 0.5341 - val_loss: 1.1995 - val_acc: 0.5980
Epoch 2/7
45000/45000 [=====] - 193s 4ms/step - loss: 0.8541 - acc: 0.6998 - val_loss: 1.0661 - val_acc: 0.6646
Epoch 3/7
45000/45000 [=====] - 193s 4ms/step - loss: 0.6536 - acc: 0.7688 - val_loss: 0.9177 - val_acc: 0.7002
Epoch 4/7
45000/45000 [=====] - 193s 4ms/step - loss: 0.5206 - acc: 0.8144 - val_loss: 0.7555 - val_acc: 0.7504
Epoch 5/7
45000/45000 [=====] - 193s 4ms/step - loss: 0.4026 - acc: 0.8564 - val_loss: 0.7836 - val_acc: 0.7626
Epoch 6/7
45000/45000 [=====] - 193s 4ms/step - loss: 0.3026 - acc: 0.8920 - val_loss: 1.0106 - val_acc: 0.7182
Epoch 7/7
45000/45000 [=====] - 193s 4ms/step - loss: 0.2303 - acc: 0.9177 - val_loss: 1.1591 - val_acc: 0.7178
45000/45000 [=====] - 58s 1ms/step
5000/5000 [=====] - 6s 1ms/step
epochs: 7, batch size: 200
Test ideal cost: 0.274090516922
Test ideal accuracy: 90.317778
Test loss: 1.15905436516
Test accuracy: 0.7178

利用此架構下，並把 epoch 增加到 20 層，最好也就做到將近 80%而已，也有點 overfitting。

```
62 model.add(Flatten())
63
64 model.add(Dense(4096)) #4
65 model.add(BatchNormalization()) #BatchNorm
66 model.add(PReLU())
67 model.add(Dropout(0.25))
68
69 #model.add(Dense(2048)) #4
70 #model.add(BatchNormalization()) #BatchNorm
71 #model.add(PReLU())
72 #model.add(Dropout(0.5))
73
74 model.add(Dense(1000)) #4
75 model.add(BatchNormalization()) #BatchNorm
76 model.add(PReLU())
77 model.add(Dropout(0.5))
78
79 model.add(Dense(num_classes, activation='softmax')) #5
80
81 model.compile(loss = keras.losses.categorical_crossentropy,
82               optimizer = keras.optimizers.Adadelta(),
83               metrics=['accuracy'])
84
85 model.fit(x_train, y_train,
86         batch_size = batch_size,
87         epochs = epochs,
88         verbose = 1,
89         validation_data = (x_test, y_test))
90
91 score1 = model.evaluate(x_train, y_train, verbose = 1)
92 score2 = model.evaluate(x_test, y_test, verbose = 1)
93
94 outfile = 'cnn_model_with_TS45_BS200_EP20withDropout'
95 model.save_weights(outfile)
96
97 print('epochs: %d, batch size: %d' % (epochs, batch_size))
98 print('Test ideal cost:', score1[0])
99 print('Test ideal accuracy:%f ' % (score1[1] * 100))
100 print('Test loss:', score2[0])
101 print('Test accuracy:', score2[1])
102
103
```

Name	Type	Size	Value
batch_size	int	1	200
data_x	float64	(50000, 32, 32, 3)	array([[[[0.49019608, 0.57647059, 0.49080392, ...
data_y	float64	(50000, 10)	array([[0., 0., 0., ..., 1., 0., ...
epochs	int	1	20

Epoch 14/20
45000/45000 [=====] - 197s 4ms/step - loss: 0.4662 - acc: 0.8335 - val_loss: 0.6513 - val_acc: 0.7890
Epoch 15/20
45000/45000 [=====] - 197s 4ms/step - loss: 0.4442 - acc: 0.8418 - val_loss: 0.6689 - val_acc: 0.7794
Epoch 16/20
45000/45000 [=====] - 197s 4ms/step - loss: 0.4166 - acc: 0.8511 - val_loss: 0.5411 - val_acc: 0.8158
Epoch 17/20
45000/45000 [=====] - 197s 4ms/step - loss: 0.3858 - acc: 0.8604 - val_loss: 0.5679 - val_acc: 0.8106
Epoch 18/20
45000/45000 [=====] - 197s 4ms/step - loss: 0.3612 - acc: 0.8699 - val_loss: 0.5875 - val_acc: 0.8136
Epoch 19/20
45000/45000 [=====] - 197s 4ms/step - loss: 0.3462 - acc: 0.8766 - val_loss: 0.6587 - val_acc: 0.7984
Epoch 20/20
45000/45000 [=====] - 197s 4ms/step - loss: 0.3148 - acc: 0.8862 - val_loss: 0.6482 - val_acc: 0.7994
45000/45000 [=====] - 58s 1ms/step
5000/5000 [=====] - 6s 1ms/step
epochs: 20, batch size: 200
Test ideal cost: 0.157356066582
Test ideal accuracy: 94.757778
Test loss: 0.648216510391
Test accuracy: 0.7994

會沒辦法突破 80%，我覺得原因在於可能資料主要是分成動物與交通工具，導致在訓練過程中分散學習的重點到 2 個不同的領域，所以盡管訓練正確率可以高達 96%，但是測試時仍無法初步判斷輸入的資料是動物還是交通工具，我覺得解決方法在於再增加資料數量，不然就是只訓練一個針對動物的 model，另一個針對交通工具，這樣會有更好的表現。