Deep Learning HW2

電子所 陳惟民 0650273

1. CNN For Image Recognition

i. Image preprocessing approach:

Central crop to all images and resize to (512,512).

Below preprocessing code is for both training set and testing set.

```
32
33  normalize = transforms.Normalize(mean=[0.5, 0.5, 0.5],
34                                   std=[0.5, 0.5, 0.5])
35
36  TRANSFORM = transforms.Compose([transforms.CenterCrop(512),
37      transforms.RandomHorizontalFlip(),
38      transforms.ToTensor(),
39      normalize])
40
```

ii. CNN architecture: 3cnn+1fc (包含 v.的討論)

Three convolution layers and each layer is connected to a ReLU activation function and pooling layer respectively. Stride and filter size are showed below.

```
55  ###NET
56  class CNN(nn.Module):
57      def __init__(self):
58          super(CNN,self).__init__()
59          self.layer1 = nn.Sequential(
60              nn.Conv2d(3, 16, kernel_size = 5, padding=2),
61              nn.ReLU(),
62              nn.MaxPool2d(2))
63          self.layer2 = nn.Sequential(
64              nn.Conv2d(16, 32, kernel_size = 5, padding=2),
65              nn.ReLU(),
66              nn.MaxPool2d(2))
67          self.layer3 = nn.Sequential(
68              nn.Conv2d(32, 32, kernel_size = 5, padding=2),
69              nn.ReLU(),
70              nn.MaxPool2d(4))
71          self.fc = nn.Linear(32*32*32,11)
```

Accuracy:

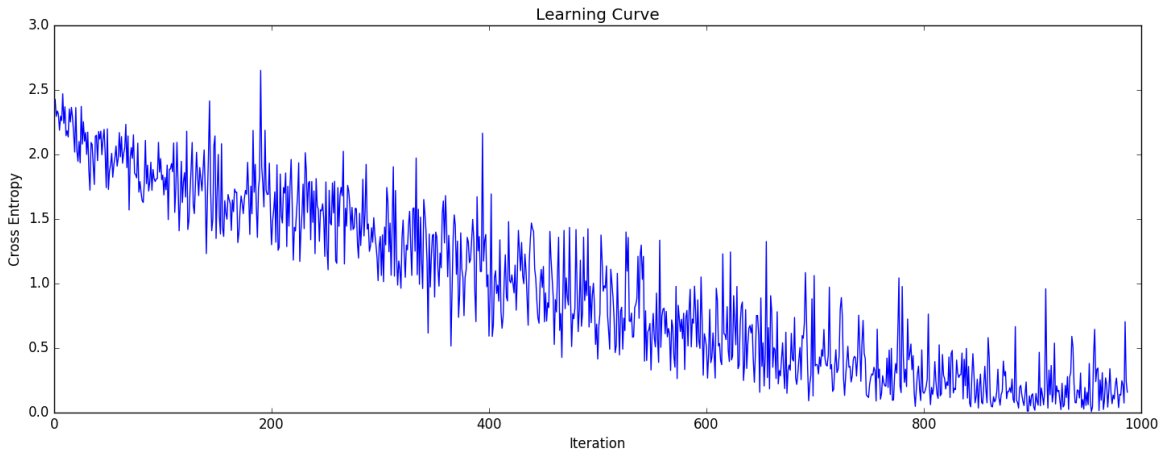
Training Accuracy: 96%

Testing Accuracy: 46%

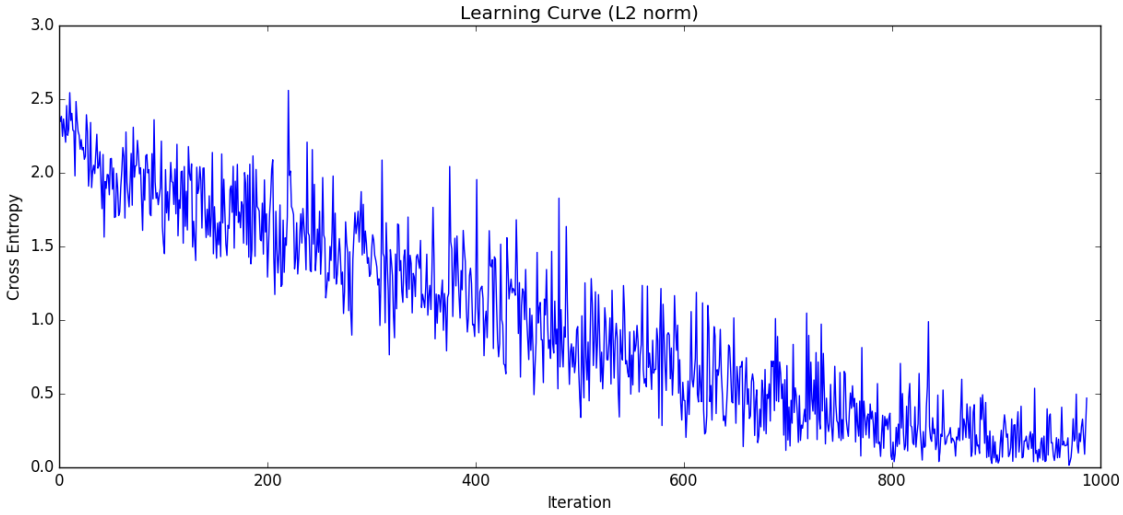
```
Accuracy on the training set: 96 %
-----Start Testing-----
Testing Accuracy: 46 %
wmchen@commlab1080:~/DL2018$
```

Learning Curve: 下降趨勢明顯，但無顯著不同

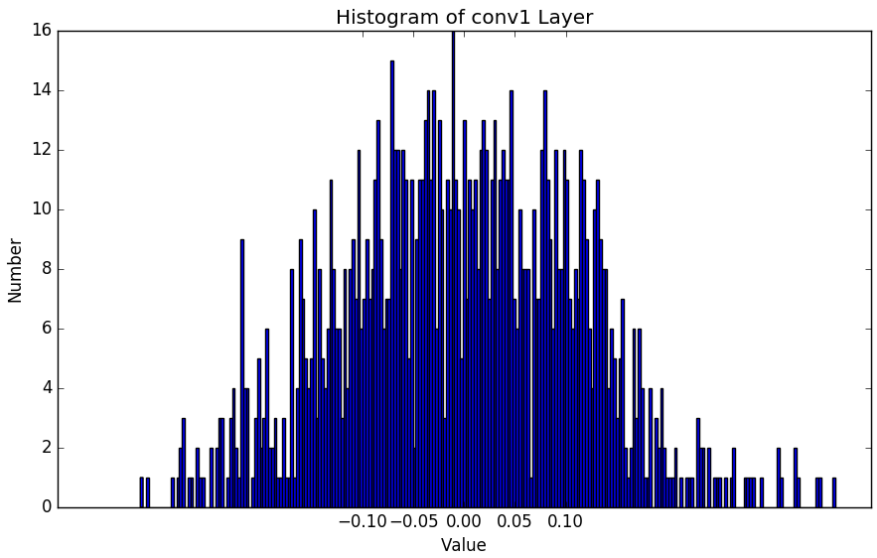
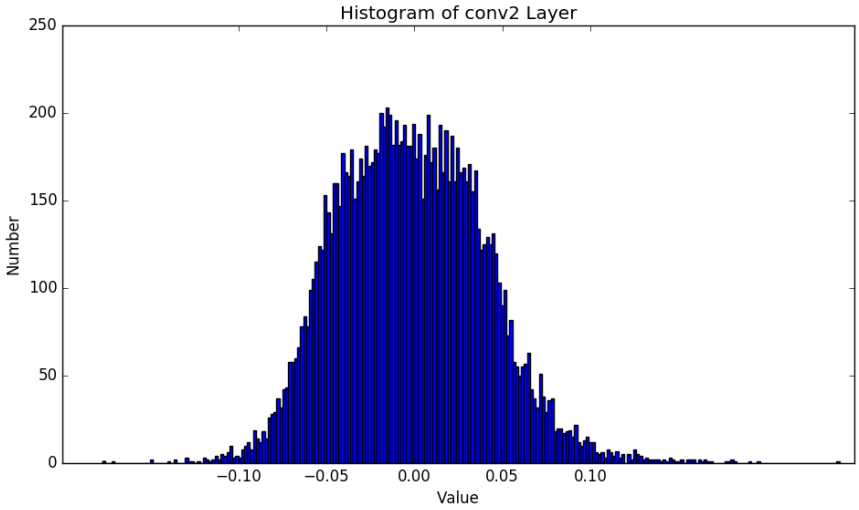
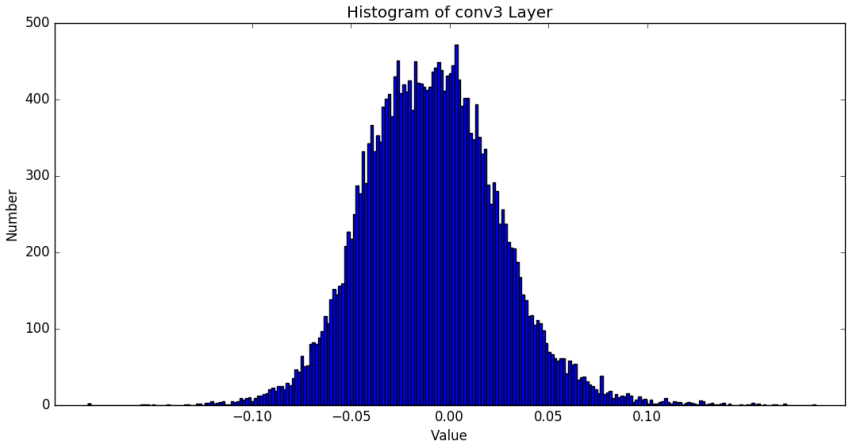
No L2 penalty:

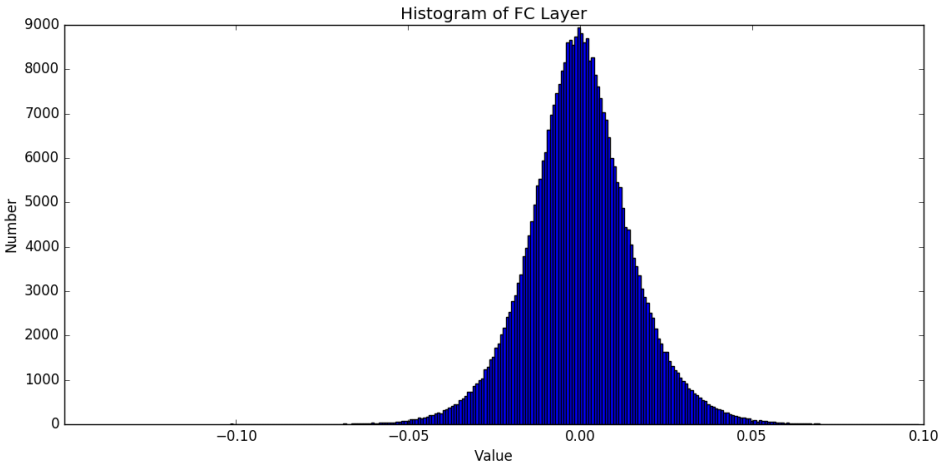


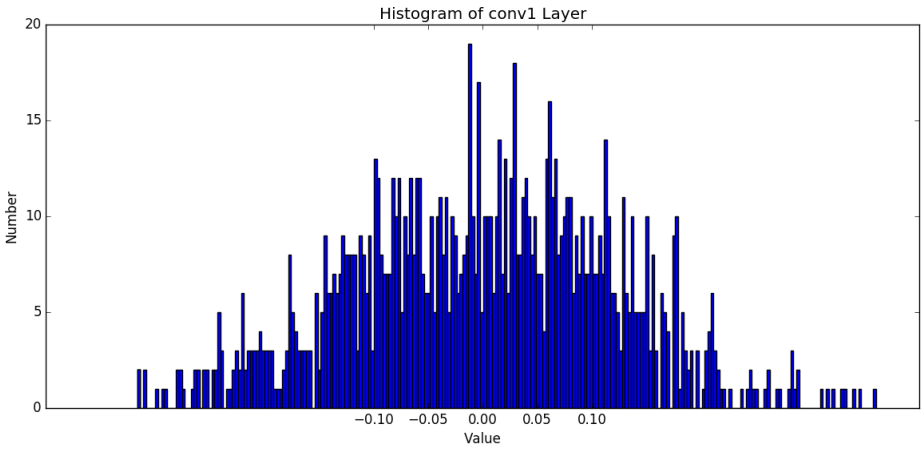
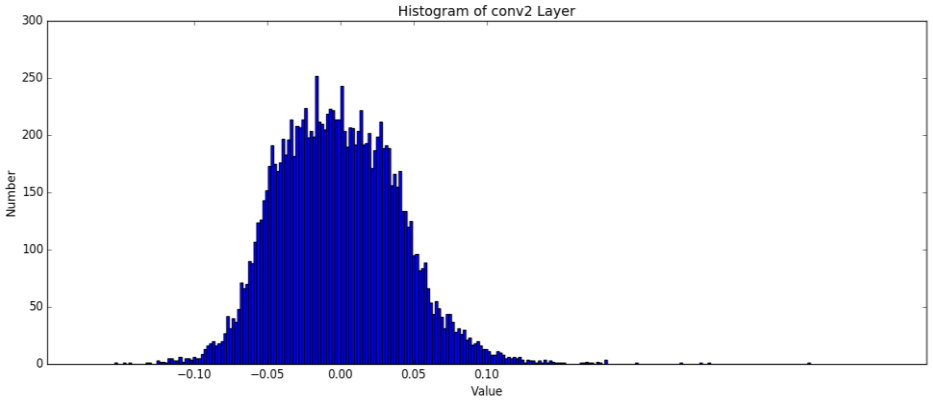
With L2 penalty:

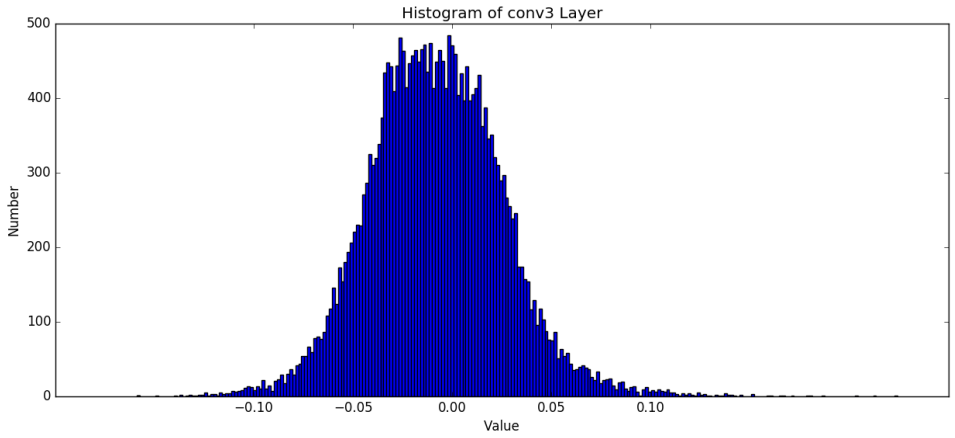
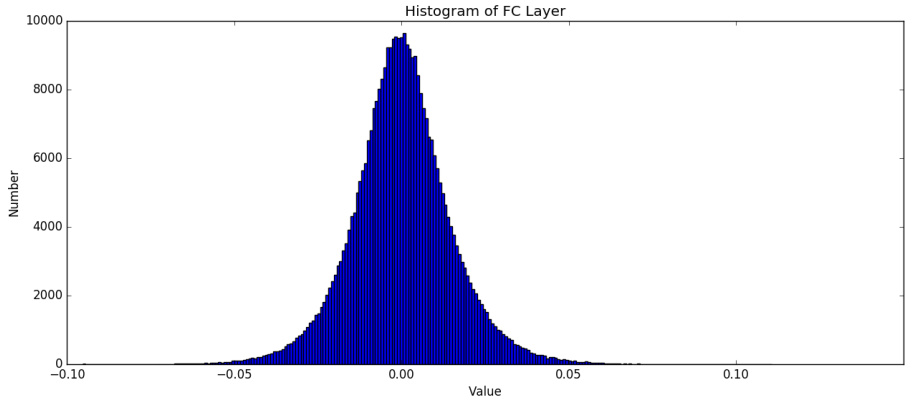


Weights、Bias distribution:

	No L2 penalty:
Conv1	
Conv2	
Conv3	

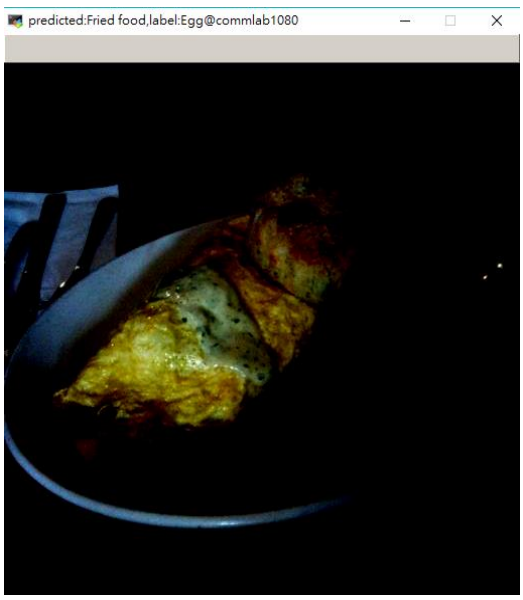
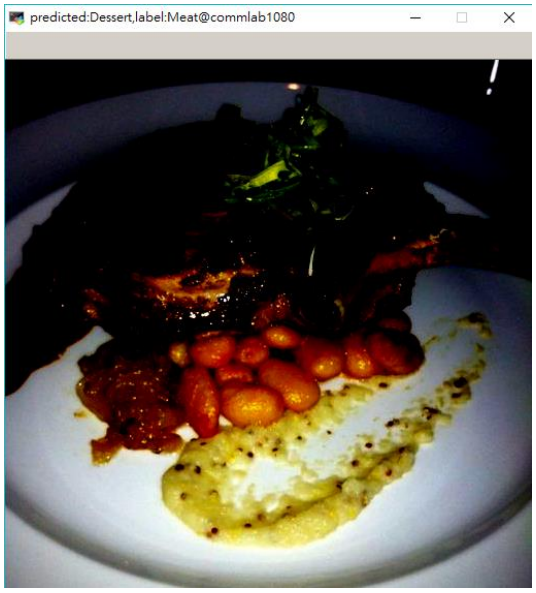
FC1	

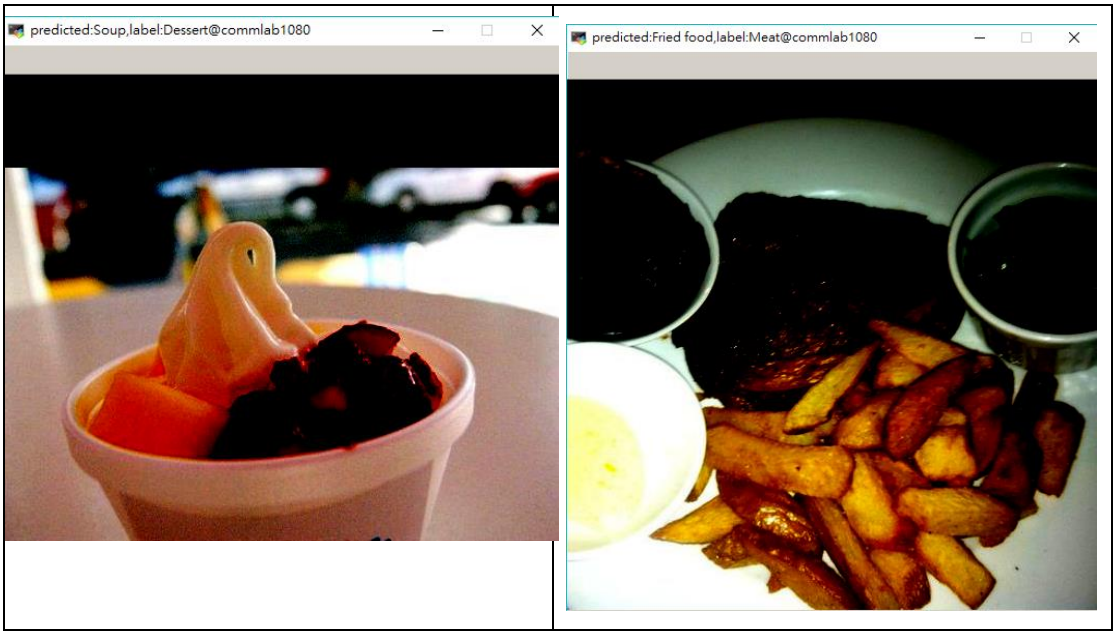
	With L2 penalty:
Conv1	
Conv2	

Conv3	
FC1	

iii. Show some results which predicted wrong

True label and classes name wrong predicted are show in the image file name. (Showed in upper toolbar of every image.)

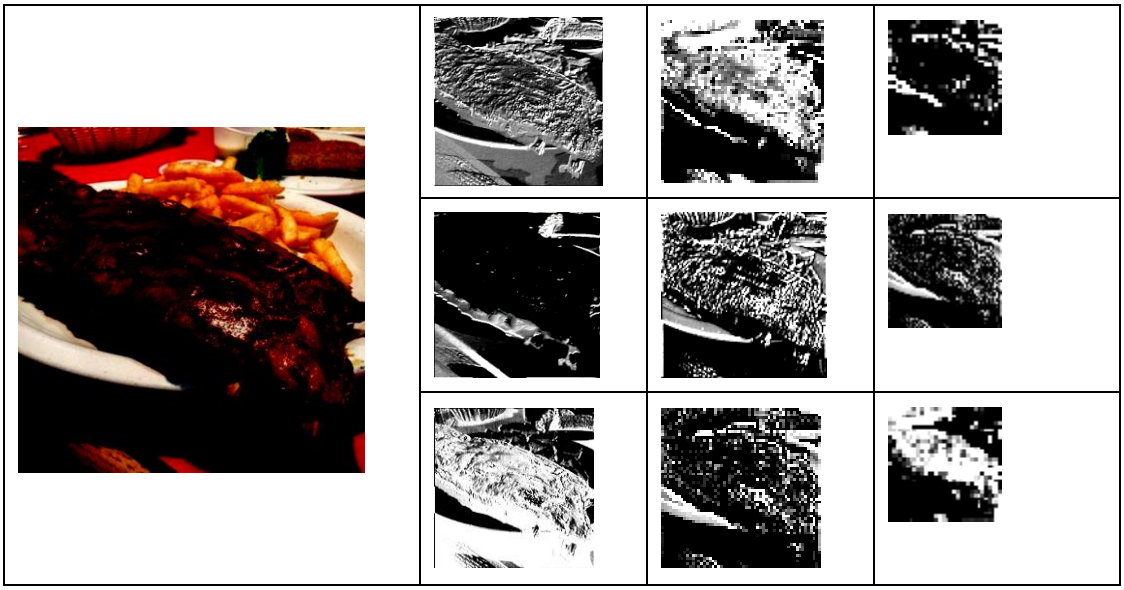
	

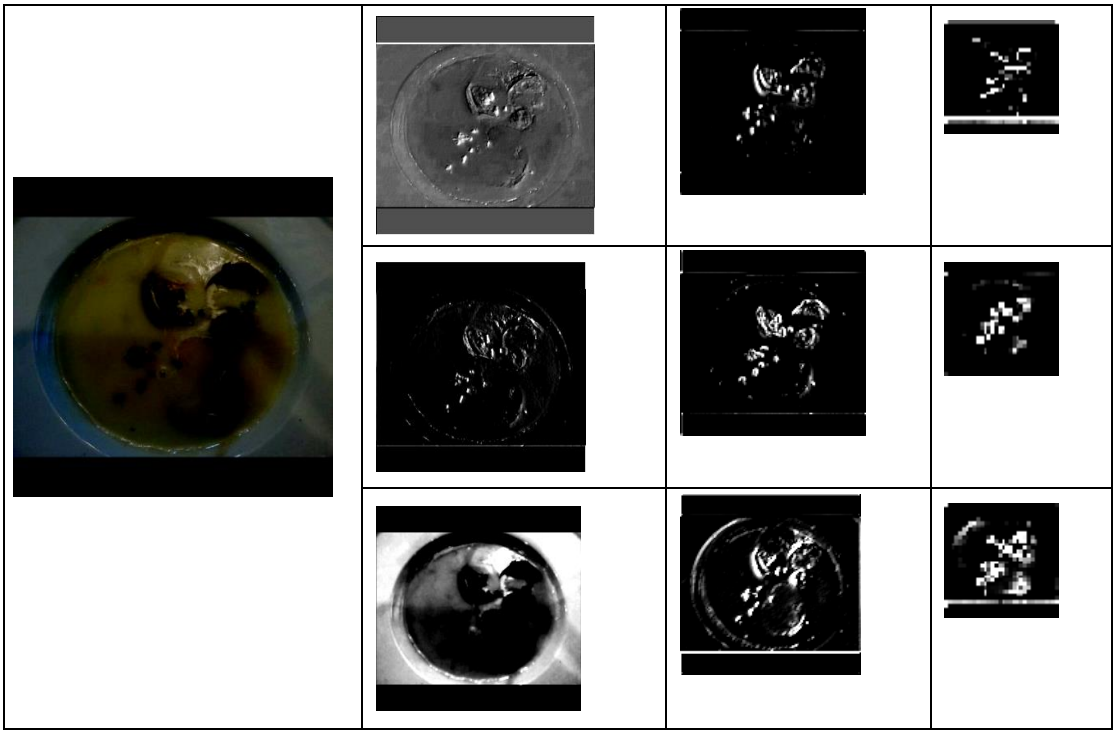


Discuss: 這些圖有的連肉眼都分不清是什麼，何況是 CNN。這是 dataset 本身的缺陷。

Ex: 那杯東西看起來像聖代(Dessert)，但解釋成羅宋湯(Soup)也完全合理。

iv. CNN layer feature visualization



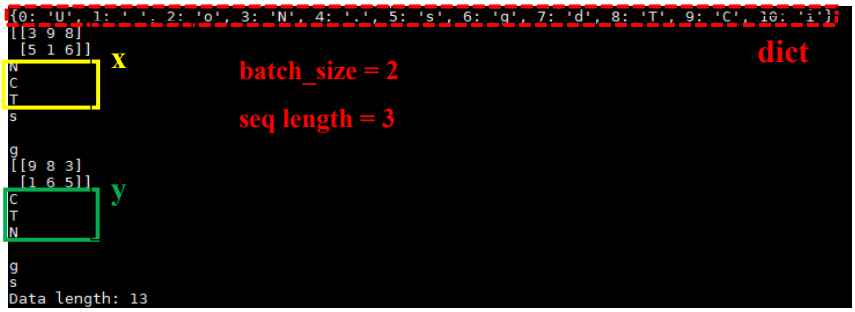


根據影像處理課程所學，2D convolution 後會取到邊界特徵，可進行非常粗糙的 image segmentation，從圖亦可見邊界被提取出來。

2. Recurrent Neural Network for Language Model

i. Explain how to separate the sentence data into mini-batches.

s = [N,C,T,U, ,i,s, , g,o,o,d]



batch_size = 2:

batch1 NCTU_i

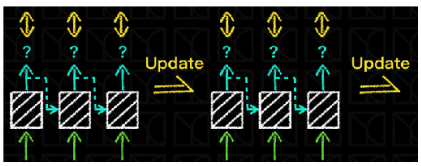
batch2 s _ good

time_step = 3:

batch1 NCTU_i

batch2 s _ good

[C,_][T,g][U,o] [_,o][i,d]

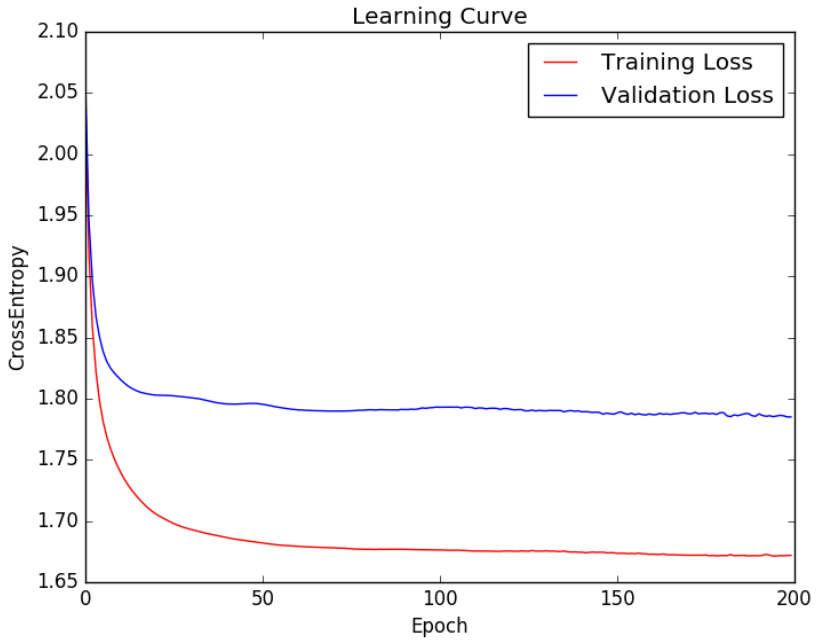
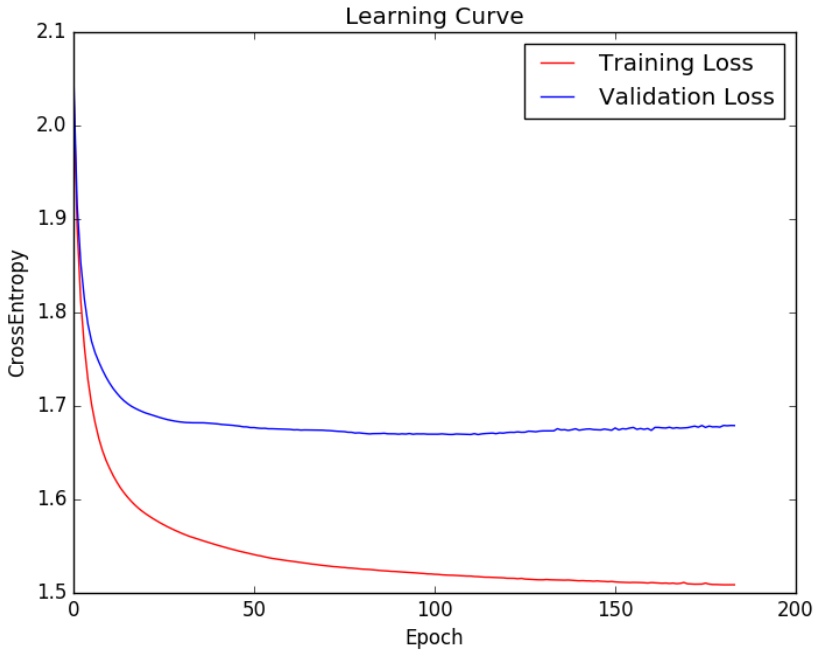


[N,s][C,_][T,g] [U,o][_,o]

ii. Bits-per-character(BPC)

$$\mathrm{BPC} = -\frac{1}{T}\sum_{t=1}^{T}\sum_{k=1}^{K} t_{t,k}\log_2 y_{t,k}(\mathbf{x}_t, \mathbf{w})$$

定義看起來像 entropy。

RNN	
LSTM	

iii. 在標準的 RNN 中，只有一個 tanh 層。LSTM 有四個 activation function，多了 forget gate，結構較 RNN 複雜，較能學習長期信息。訓練後 Training loss、Validation loss 都比 RNN 低。

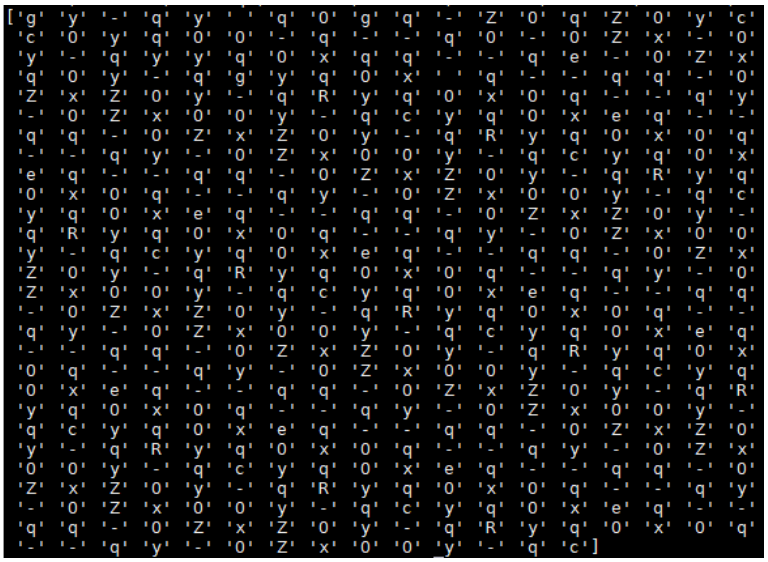
iv. Generate some words:

Actually, I can not identify the sentence what I generated, so I just explain the code I did.

```
264  print('-----Start generating-----')
265  prime = 'ROMEO'                                          作為初始放入的文字
266  generator = np.array([vocab_to_int[c] for c in prime], dtype=np.int32)
267  generator = np.expand_dims(generator, axis=0)
268  net = T.load(FILENAME).cuda()
269  #print(generator[0])                                     確定初始放入的文字 encode 正確
270  for i in generator[0]:
271      print(int_to_vocab[i])
272  hidden_ = net.init_hidden()
273  x_ = Variable(T.from_numpy(generator).long()).cuda()
274  #print(int_to_vocab)
275  pattern = []
276  for i in range(500):
277      
278      print(x_.size())                                     將 encoded 的文字放入，進行 round one
279      out, hidden_ = net(x_,hidden_)
280      print(out.size())
281      out = out.view(-1,N_CHARACTERS)
282      sentence = T.argmax(out,dim=1)                       Argmax 取出 dict(67 字母)中，機率最大為該字母
283      print(out.size())
284      nu = sentence.cpu().numpy()
285      print(nu)
286      for i in nu:
287          pattern.append(int_to_vocab[i])
288      
289      new_words = np.expand_dims(nu, axis=0)
290      x_ = Variable(T.from_numpy(new_words).long()).cuda()
291      #print(new_words)                                    將第一次的 output 重新 reshape，作為下一個 time step 的 input
292      print(new_words)
293      exit()
294
```

```
R
O
M
E
O
torch.Size([1, 5])
torch.Size([1, 5, 67])
torch.Size([5, 67])
[ 6  0 44 36 23]
[[ 6  0 44 36 23]]
wmchen@commlab1080:~/DL2018/HW2/Dataset$
```

Result:


['g' 'y' '-' 'q' 'y' ' ' 'q' 'O' 'g' 'q' '-' 'Z' 'O' 'q' 'Z' 'O' 'y' 'c'
'c' 'O' 'y' 'q' 'O' 'O' '-' 'q' '-' '-' 'q' 'O' '-' 'O' 'Z' 'x' '-' 'O'
'y' '-' 'q' 'y' 'y' 'q' 'O' 'x' 'q' 'q' '-' '-' 'q' 'e' '-' 'O' 'Z' 'x'
'q' 'O' 'y' '-' 'q' 'g' 'y' 'q' 'O' 'x' ' ' 'q' '-' '-' 'q' 'q' '-' 'O'
'Z' 'x' 'Z' 'O' 'y' '-' 'q' 'R' 'y' 'q' 'O' 'x' 'O' 'q' '-' '-' 'q' 'y'
'-' 'O' 'Z' 'x' 'O' 'O' 'y' '-' 'q' 'c' 'y' 'q' 'O' 'x' 'e' 'q' '-' '-'
'q' 'q' '-' 'O' 'Z' 'x' 'Z' 'O' 'y' '-' 'q' 'R' 'y' 'q' 'O' 'x' 'O' 'q'
'-' '-' 'q' 'y' '-' 'O' 'Z' 'x' 'O' 'O' 'y' '-' 'q' 'c' 'y' 'q' 'O' 'x'
'e' 'q' '-' '-' 'q' 'q' '-' 'O' 'Z' 'x' 'Z' 'O' 'y' '-' 'q' 'R' 'y' 'q'
'O' 'x' 'O' 'q' '-' '-' 'q' 'y' '-' 'O' 'Z' 'x' 'O' 'O' 'y' '-' 'q' 'c'
'y' 'q' 'O' 'x' 'e' 'q' '-' '-' 'q' 'q' '-' 'O' 'Z' 'x' 'Z' 'O' 'y' '-'
'q' 'R' 'y' 'q' 'O' 'x' 'O' 'q' '-' '-' 'q' 'y' '-' 'O' 'Z' 'x' 'O' 'O'
'y' '-' 'q' 'c' 'y' 'q' 'O' 'x' 'e' 'q' '-' '-' 'q' 'q' '-' 'O' 'Z' 'x'
'Z' 'O' 'y' '-' 'q' 'R' 'y' 'q' 'O' 'x' 'O' 'q' '-' '-' 'q' 'y' '-' 'O'
'Z' 'x' 'O' 'O' 'y' '-' 'q' 'c' 'y' 'q' 'O' 'x' 'e' 'q' '-' '-' 'q' 'q'
'-' 'O' 'Z' 'x' 'Z' 'O' 'y' '-' 'q' 'R' 'y' 'q' 'O' 'x' 'O' 'q' '-' '-'
'q' 'y' '-' 'O' 'Z' 'x' 'O' 'O' 'y' '-' 'q' 'c' 'y' 'q' 'O' 'x' 'e' 'q'
'-' '-' 'q' 'q' '-' 'O' 'Z' 'x' 'Z' 'O' 'y' '-' 'q' 'R' 'y' 'q' 'O' 'x'
'O' 'q' '-' '-' 'q' 'y' '-' 'O' 'Z' 'x' 'O' 'O' 'y' '-' 'q' 'c' 'y' 'q'
'O' 'x' 'e' 'q' '-' '-' 'q' 'q' '-' 'O' 'Z' 'x' 'Z' 'O' 'y' '-' 'q' 'R'
'y' 'q' 'O' 'x' 'O' 'q' '-' '-' 'q' 'y' '-' 'O' 'Z' 'x' 'O' 'O' 'y' '-'
'q' 'c' 'y' 'q' 'O' 'x' 'e' 'q' '-' '-' 'q' 'q' '-' 'O' 'Z' 'x' 'Z' 'O'
'y' '-' 'q' 'R' 'y' 'q' 'O' 'x' 'O' 'q' '-' '-' 'q' 'y' '-' 'O' 'Z' 'x'
'O' 'O' 'y' '-' 'q' 'c' 'y' 'q' 'O' 'x' 'e' 'q' '-' '-' 'q' 'q' '-' 'O'
'Z' 'x' 'Z' 'O' 'y' '-' 'q' 'R' 'y' 'q' 'O' 'x' 'O' 'q' '-' '-' 'q' 'y'
'-' 'O' 'Z' 'x' 'O' 'O' 'y' '-' 'q' 'c' 'y' 'q' 'O' 'x' 'e' 'q' '-' '-'
'q' 'q' '-' 'O' 'Z' 'x' 'Z' 'O' 'y' '-' 'q' 'R' 'y' 'q' 'O' 'x' 'O' 'q'
'-' '-' 'q' 'y' '-' 'O' 'Z' 'x' 'O' 'O' 'y' '-' 'q' 'c']