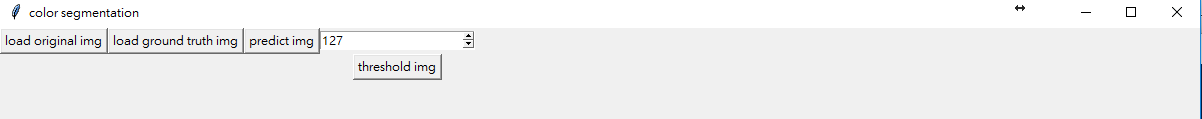
Image Processing HW1

P76064295 蔡侑軒

* **Environments**
* Windows 10
* Python 3.6.3
* Tensorflow 1.4.0
* **Interface**

在cmd中執行指令＂py main.py＂，便會跳出interface(如下所示)。



* **Questions**

從color image中切出傷口部分

* **Methods**

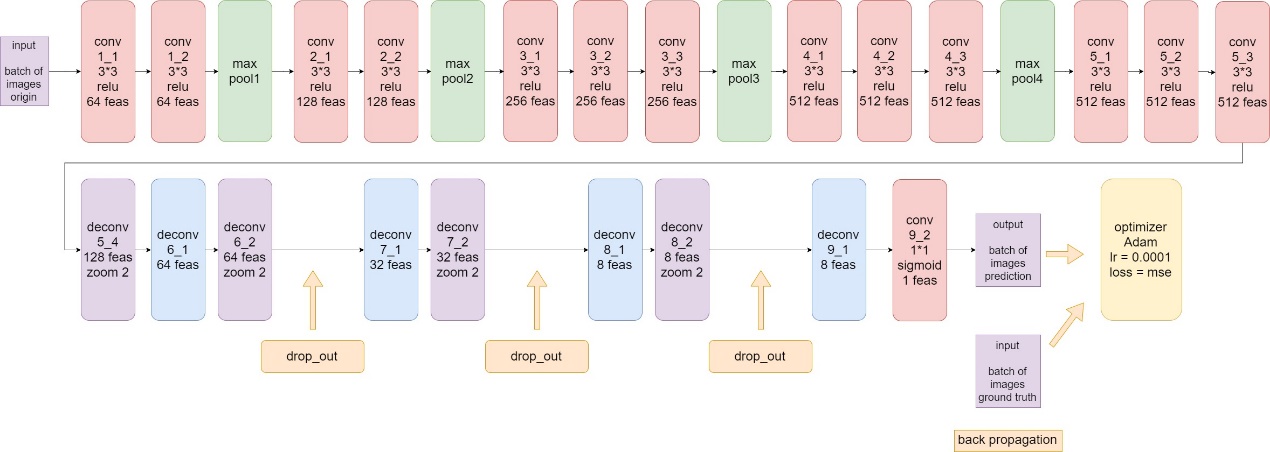
使用tensorflow，利用machine learning的方式預測出傷口位置

* **FCN**

這次使用的為FCN(fully convolution network)的主架構，選用FCN的原因有以下幾點：

* FCN的input size(換言之image size)可以為任意
* 參數個數少
* 輸出之結果可為image

FCN的整體架構如下：



* “conv 1\_1 3\*3 relu 64 feas”表示該layer為convolution layer、編號為1\_1、使用3\*3的filter、使用relu當作activation、此層輸出的結果會有64張feature maps等。
* “max pool1”表示該layer為max pool layer，編號為1，在此架構中，所有的max pool layer所使用的filter大小皆為2\*2，亦即會將輸入size縮小兩倍(max pool)後輸出。
* “deconv 5\_4 128 feas zoom2”表示該layer為deconvolution layer、編號為5\_4、此層輸出結果會有128張feature maps，並會放大兩倍size(若無zoom 2的表示單純做deconvolution)。對於放大用的deconv layer，其filter會先以bilinear的方式做初始化。
* “drop\_out”表示drop out layer。

其中，上半部的part(conv1\_1 ~ conv5\_3)為VGG16之架構，在概念上為使用多個小型的convolution filter(ex: 3\*3)來取代一個大型的convolution filter(ex: 5\*5)，如，兩個3\*3 filter的conv layer相連接，其實在效力上基本相等於一個5\*5 filter的conv layer，而使用兩層小的conv layer還會多經過一次activation function，因此效力會更勝一個5\*5 filter的conv layer。

在網路上有VGG16已經training完成的參數檔，可以將參數load進我們所寫的結構中，因為該參數是用非常龐大的dataset(Imageset)訓練，所以擷取feature的能力是值得肯定的。這樣的方法不僅能減少整體訓練的時間，同時準確度也會上升取多，更能彌補對應特定的題目時(如本次wound segmentation)training data量可能不足的問題。

在VGG16之後，接上多個deconvolution layer，概念上就是反轉前面VGG16之架構，期以得出能與原圖相似(size大小、輪廓等)但不同特徵(segmentation)之輸出。在實作上，並沒有完全反轉前面之VGG16架構，因為經過自己的測試，這樣的架構足以應對wound segmentation之問題，更多的layer反而會衍生其他問題(準確率下降、training、prediction時間長、記憶體不足等)，故有稍稍刪減一些layer。

上圖中橘色部分為drop out layer，這種layer只有在training時會加上，概念為隨機使得某些weight為0，用以預防overfitting，在此實作中dropped rate = 0.4；而黃色部分為optimizer，在此次實作上選用adam，learning rate為0.00001，選用mean squared error作為loss function

* **Data preprocessing**

為了加快訓練時間，我將所有data(ground truth 與 training data)的長寬各縮小2倍，而對於ground truth data，我對其做invert，讓傷口部分為255，其餘為0

觀察架構圖可知，VGG16做了4次的max pool layer，亦即縮小了16倍，故在deconvolution layer的部分，也有4層放大2倍的layer，共放大了16倍。但在實作上，max pool layer在縮小時，若某一維度(長或寬)為奇數時，其輸出size會取ceil，而後在經由deconvolution layer放大16倍之後，原始輸入與輸出的長寬可能會有所不同！故在前處理時，會將data之長寬縮放成16的倍數，以避免在神經網路中會出錯。

* **Data augmentation**

30筆data是不夠的，所以做data augmentation是必須的。在此寫一支簡單的程式來生成data(image\_preprocessing.py)，生成出的data，會以原圖做為標準，隨機旋轉與上下左右位移，並對相對應之ground truth也做相同的旋轉與上下左右位移，所產生之新data，雖然在人看來差異性不大，但對於電腦(矩陣觀點)而言以是截然不同的全新data!

在此，隨機旋轉角度為±30，隨機上下左右位移為0.2(邊長比例)

* **Training**

在training中，batch size = 2，batch size若是更大，應該能使得training與prediction的速度更快，在此只取2的原因是因為記憶體不足…

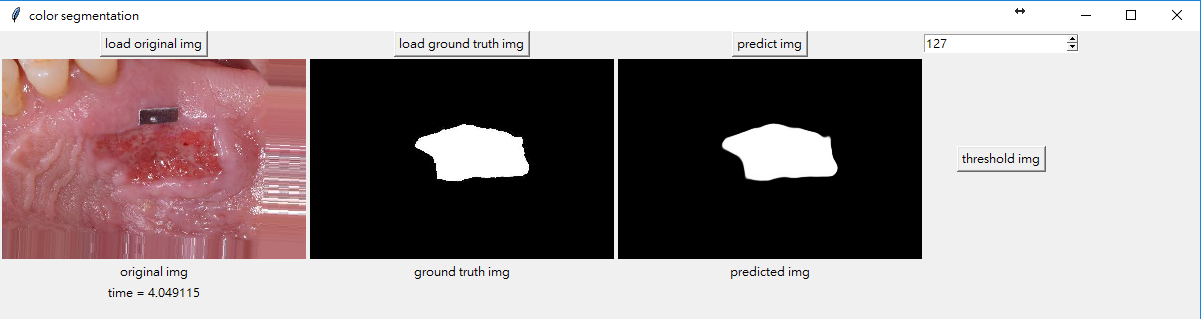
第一次的training中，使用120筆data，300 epoch，總共花費約13hr，以達到不錯之效果

而後加上n\_fold概念，將training data分為10份，1份作為validation data，9份作為training data，循環替換10次。在此實作n\_fold方式為，先對training data做3次的訓練，而後再對validation data做一次訓練，因此，神經網路會以training data為重，而已validation data作微調。

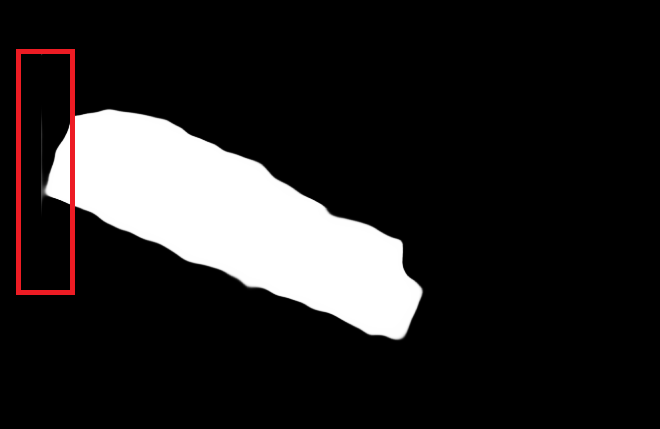
第二次training加上n\_fold概念，使用1200筆data，1 epoch，花費約9hr，達到更好的效果

* **Results**

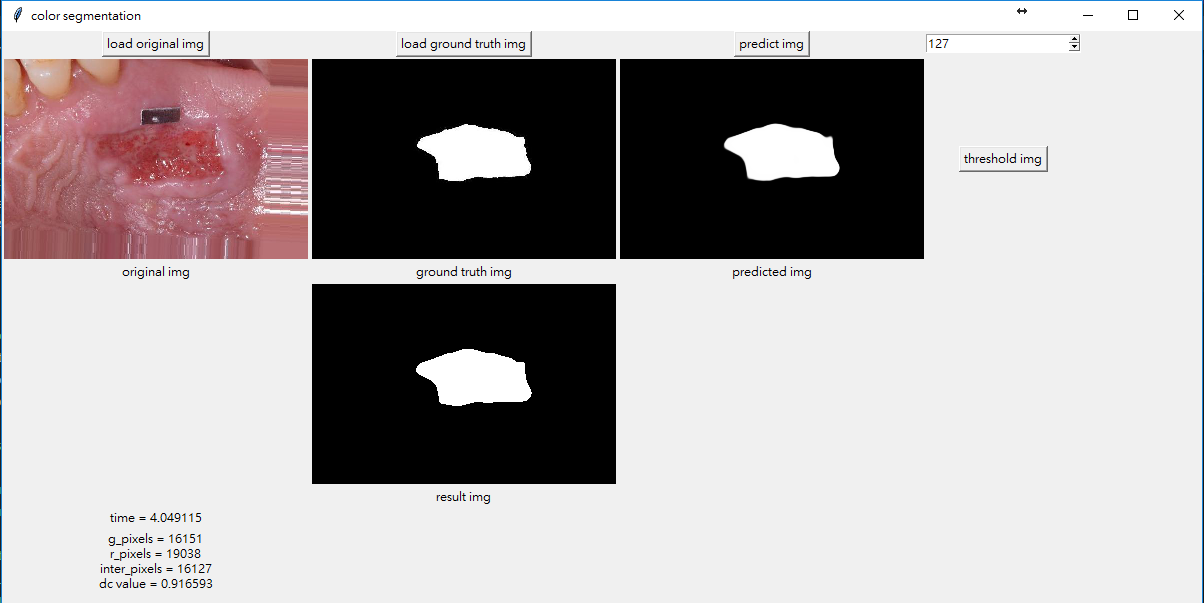
在interface上，先後load原始data與ground truth之data，而後按下predict img，便會開始運行神經網路，再做預測，會將預測結果儲存為predicted\_img.jpeg，並顯示結果圖與運算時間至interface上。



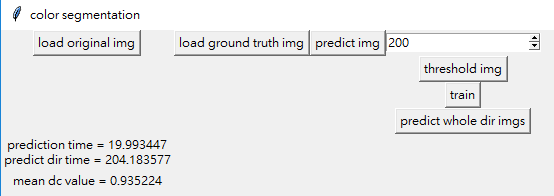
但有時結果並不會如此理想(如下圖之紅框區域)



故會再使用threshold來做最後判斷是否為傷口，右上角之spinbox之值即為threshold，決定好threshold後按下threshold img便會產生新圖，新圖會儲存為result\_img.jpeg，同時顯示並計算DC值



另外，machine learning有一大優點在於運算時間，但可以從上圖中發現時間需要4s，反而很慢。這是因為上述測試只測試單一張img，而神經網路必須呼叫GPU才能運算，其在初始化方面會需要較長的時間，所以才展生運算時間較久的錯覺



上圖為直接對某一資料夾內的wound imgs做segmentation，該資料夾內共有120筆data，可以觀察到其實神經網路的運算時間僅有20s左右(以單張img而言，即是按下predict img之步驟)，其餘的時間大部分都是在threshold img(這部分比較慢可能是因為python本身的關係，若能做平行處理等相信還能再快)，可見神經網路之速度。

另也可以看到，對這120筆資料，此神經網路能得到平均dc value = 0.94的成績

* **Discussion**

可以看的出來結果相當不錯，基本上對於傷口之判別都可以達到0.9以上之DC值。

有趣的是，大體而言prediction出來的結果通常會略大於ground truth一些，所以在threshold的選擇上，通常選得越高DC值也會越高(不過差異不是非常大)。

另外，與其他使用一般segmentation方法之朋友討論過後，聽聞他們大多會遇到一個問題，當傷口顏色淺的時候，傷口的判別容易不準，嚴重的情況甚至會直接判斷錯區域!但在我的實作中，目前是沒有發現類似的問題。

後言：這次第一次使用tensorflow，在實作時出了問題也很難明確的知道問題出在哪裡(各個地方都是黑盒子)，所以一開始處處碰壁，光是要跑出一正常結果就已經花費8成的時間，但是在弄清楚之後覺得愉悅許多!