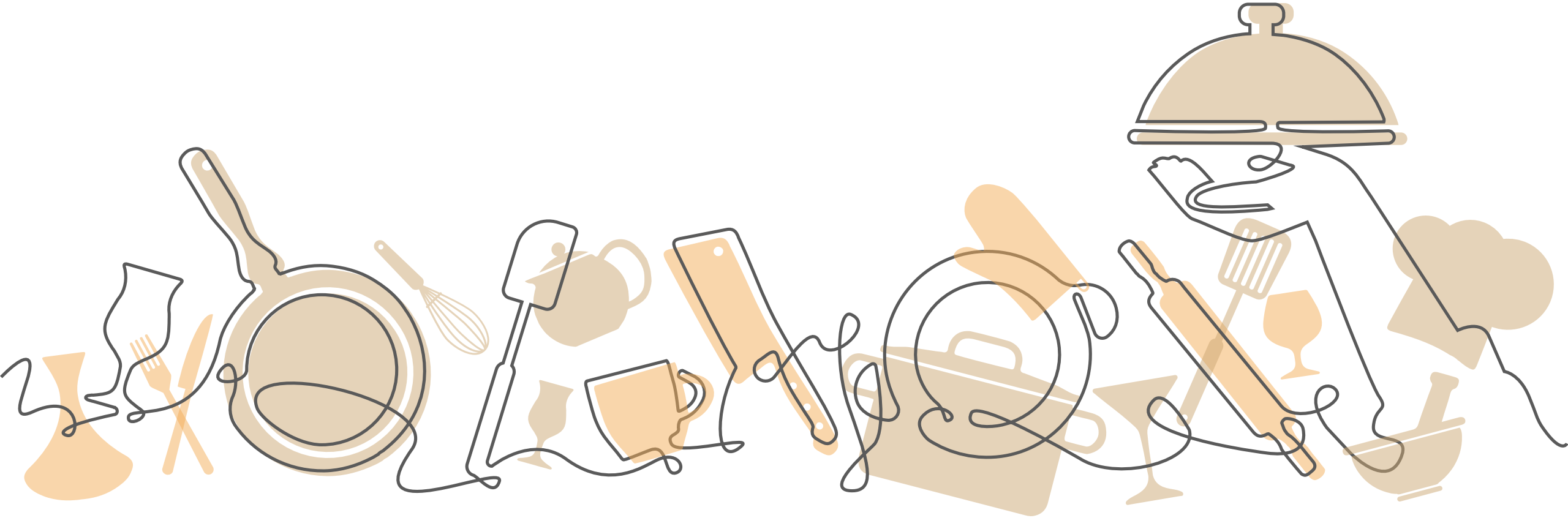


Computer Vision
Liver Segmentation with UNet

Installing

!pip install nibabel

Nibabel 是一個專門用於處理醫學影像的模組
可以用來讀取處理 .nii 或.nii.gz檔案
對於 NIfTI 影像格式支援度非常好。

https://www.slicer.org/

可以下載3D Slicer，透過此軟體顯示醫學影像檔案內的影像。

Dataset

volume-1.nii	2022/9/20 下午 01:27	NII 檔案	62,977 KB
volume-2.nii	2022/9/20 下午 01:27	NII 檔案	264,705 KB
volume-3.nii	2022/9/20 下午 01:27	NII 檔案	273,409 KB
volume-4.nii	2022/9/20 下午 01:27	NII 檔案	430,593 KB
volume-5.nii	2022/9/20 下午 01:27	NII 檔案	274,945 KB

CT

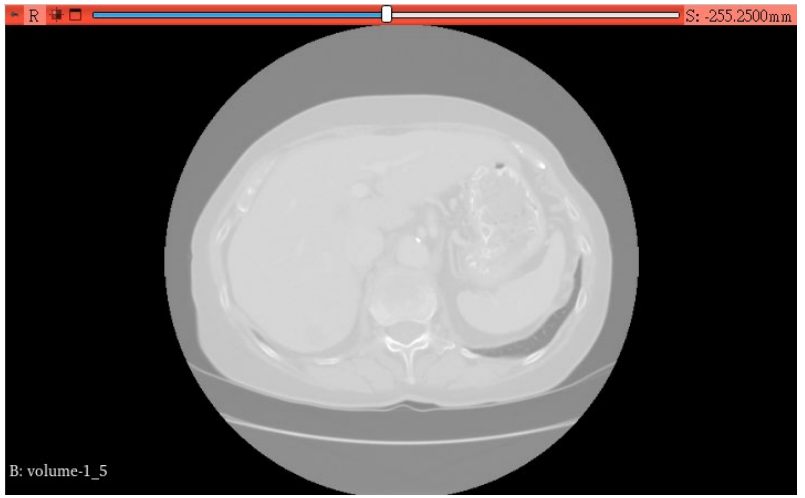
segmentation-1.nii	2022/9/20 下午 01:25	NII 檔案	31,489 KB
segmentation-2.nii	2022/9/20 下午 01:26	NII 檔案	132,353 KB
segmentation-3.nii	2022/9/20 下午 01:26	NII 檔案	136,705 KB
segmentation-4.nii	2022/9/20 下午 01:26	NII 檔案	430,593 KB
segmentation-5.nii	2022/9/20 下午 01:26	NII 檔案	137,473 KB

Mask

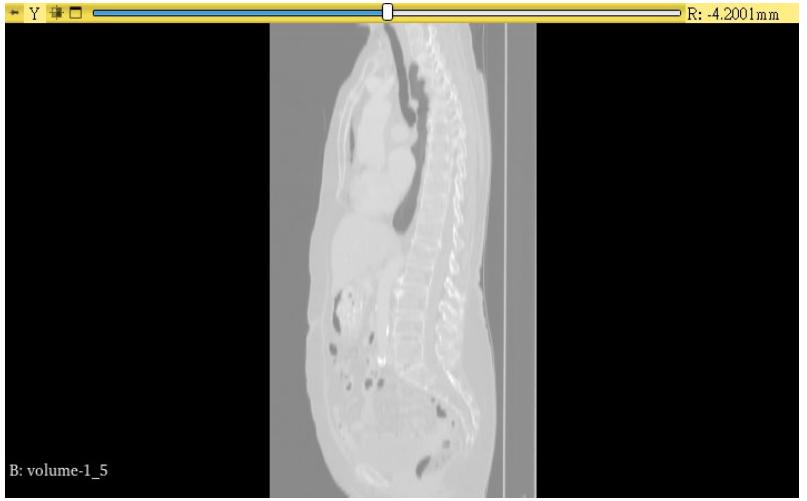
Dataset

CT

(a) coronal view

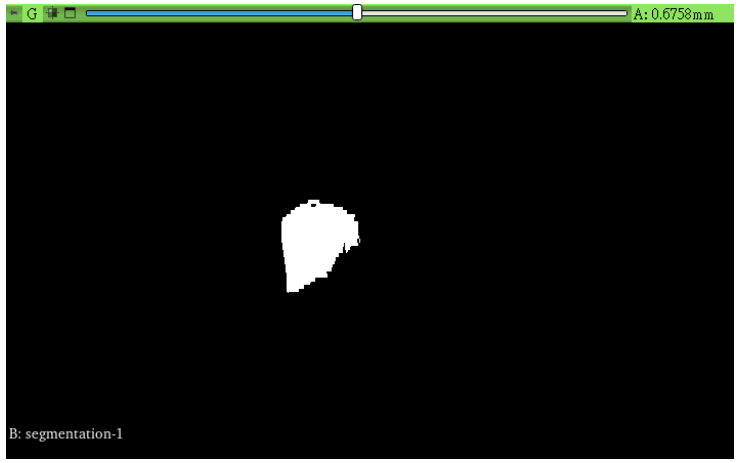


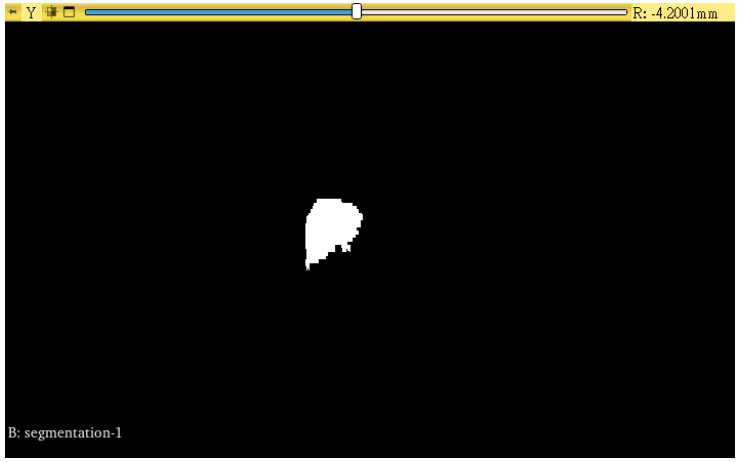
(b) axial view



(c) sagittal view

Mask



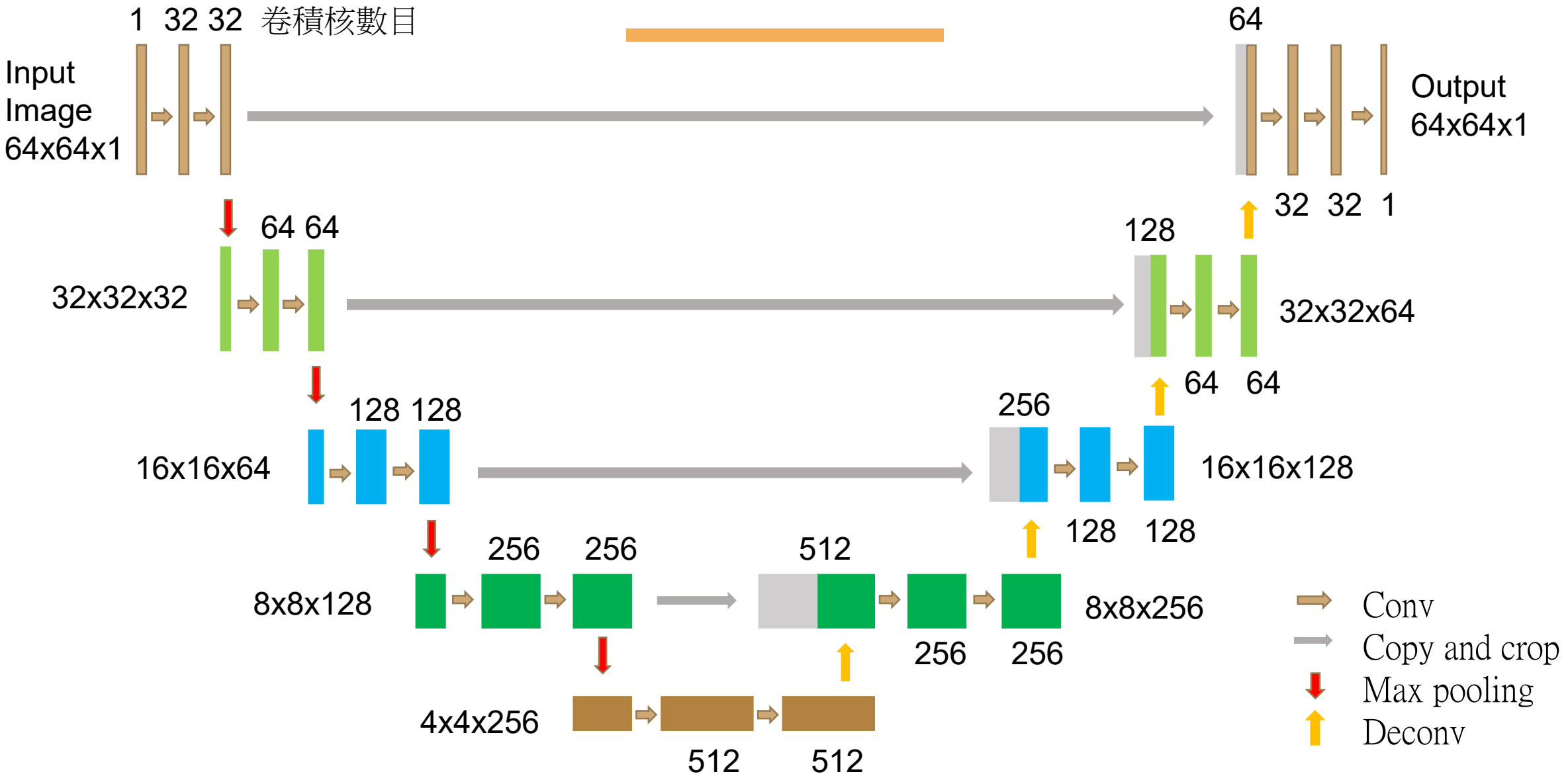


Data

1. 讀取Npy檔，上述醫學檔案已處理好儲存在一起。

```
total_patch  =  np.load("./drive/MyDrive/Unet/trainNpy/total_patch.npy")
total_mask  =  np.load("./drive/MyDrive/Unet/trainNpy/total_mask.npy")

total_patch.shape

(15318, 64, 64, 1)
```

Unet Model


1 32 32 卷積核數目
Input
Image
64x64x1
64
Output
64x64x1
32 32 1
64 64
32x32x32
128
32x32x64
64 64
128 128
16x16x64
256
16x16x128
128 128
256 256
8x8x128
512
8x8x256
256 256
4x4x256
512 512
Conv
Copy and crop
Max pooling
Deconv

Loss Function

Weighted binary crossentropy 用於二分類

$$\mathrm{BCELoss(O,T)} = -\frac{1}{n}\sum_{i}\left((T[i] * \log(O[i])) + (1 - T[i]) * \log(1 - O[i])\right)$$

True label — Prob of Positive Class — True label — Prob of Negative Class

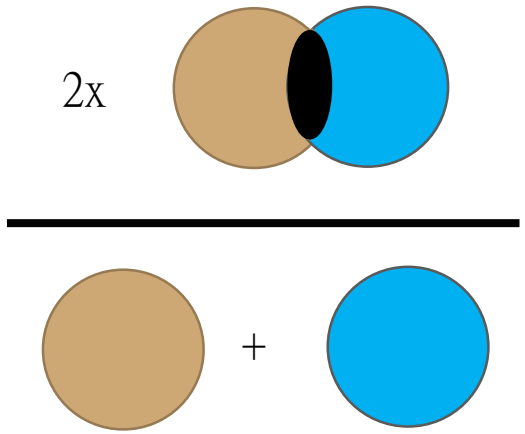
```
#loss function
def weighted_binary_crossentropy(y_true, y_pred):
    y_pred = tf.clip_by_value(y_pred, 10e-8, 1.-10e-8)  #把值壓縮到(min,max)之間，小於min的值變成min，大於max的值變成max
    loss = - (y_true * K.log(y_pred) * 0.90 + (1 - y_true) * K.log(1 - y_pred) * 0.10)

    return K.mean(loss)
```

因為正樣本(擁有肝臟的Mask)較少，因此要給予較大的權重。

Loss Function

Dice Coefficient Loss 計算樣本相似度(數值越大即重疊率越高)

```
smooth = 1.                                    #用於防止分母為0
def dice_coef(y_true, y_pred):
        y_true_f = K.flatten(y_true)           #攤平成一維
        y_pred_f = K.flatten(y_pred)
        intersection = K.sum(y_true_f * y_pred_f)
        return (2. * intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) + smooth)
```



Training

1. 設定訓練過程需要的優化器、損失函數和評估標準
   - optimizer : 優化器，梯度下降，使Loss可以越小越好
   - loss : 實際值和預測值的差距，是優化器要指引的對象
   - metrics : 與Loss相似，但結果並不會回饋到訓練過程中

```
adam = Adam(lr = 0.0001)
model.compile(optimizer = adam, loss = weighted_binary_crossentropy, metrics = [dice_coef])
```

2. 設定參數，開始訓練
   - total_patch : 訓練資料
   - total_mask : 訓練資料的答案
   - batch_size : 一次訓練的樣本數
   - epoch : 設定訓練整個資料集的次數

```
model.fit(total_patch, total_mask, batch_size = 128, epochs = 30)
```

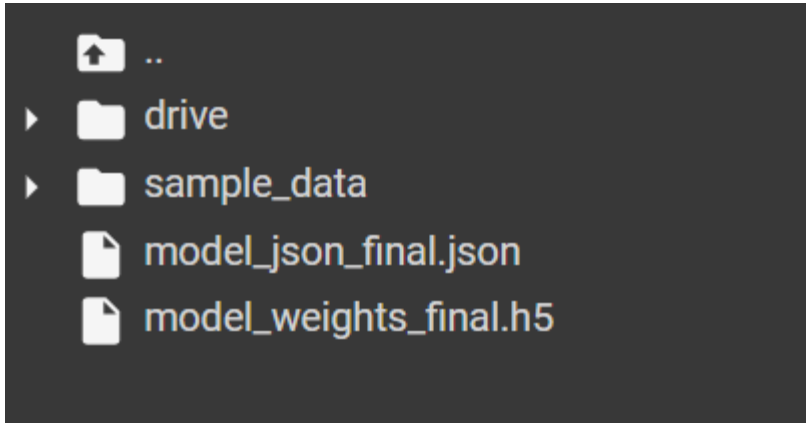
Training

```
Epoch 41/50
120/120 [==============================] - 27s 223ms/step - loss: 0.0245 - dice_coef: 0.9090
Epoch 42/50
120/120 [==============================] - 27s 223ms/step - loss: 0.0238 - dice_coef: 0.9099
Epoch 43/50
120/120 [==============================] - 27s 223ms/step - loss: 0.0266 - dice_coef: 0.9043
Epoch 44/50
120/120 [==============================] - 27s 223ms/step - loss: 0.0298 - dice_coef: 0.8990
Epoch 45/50
120/120 [==============================] - 27s 223ms/step - loss: 0.0247 - dice_coef: 0.9097
Epoch 46/50
120/120 [==============================] - 27s 223ms/step - loss: 0.0346 - dice_coef: 0.8993
Epoch 47/50
120/120 [==============================] - 27s 222ms/step - loss: 0.0263 - dice_coef: 0.9010
Epoch 48/50
120/120 [==============================] - 27s 223ms/step - loss: 0.0228 - dice_coef: 0.9118
Epoch 49/50
120/120 [==============================] - 27s 222ms/step - loss: 0.0217 - dice_coef: 0.9170
Epoch 50/50
120/120 [==============================] - 27s 222ms/step - loss: 0.0201 - dice_coef: 0.9177
<keras.callbacks.History at 0x7f3e1c094a60>
```

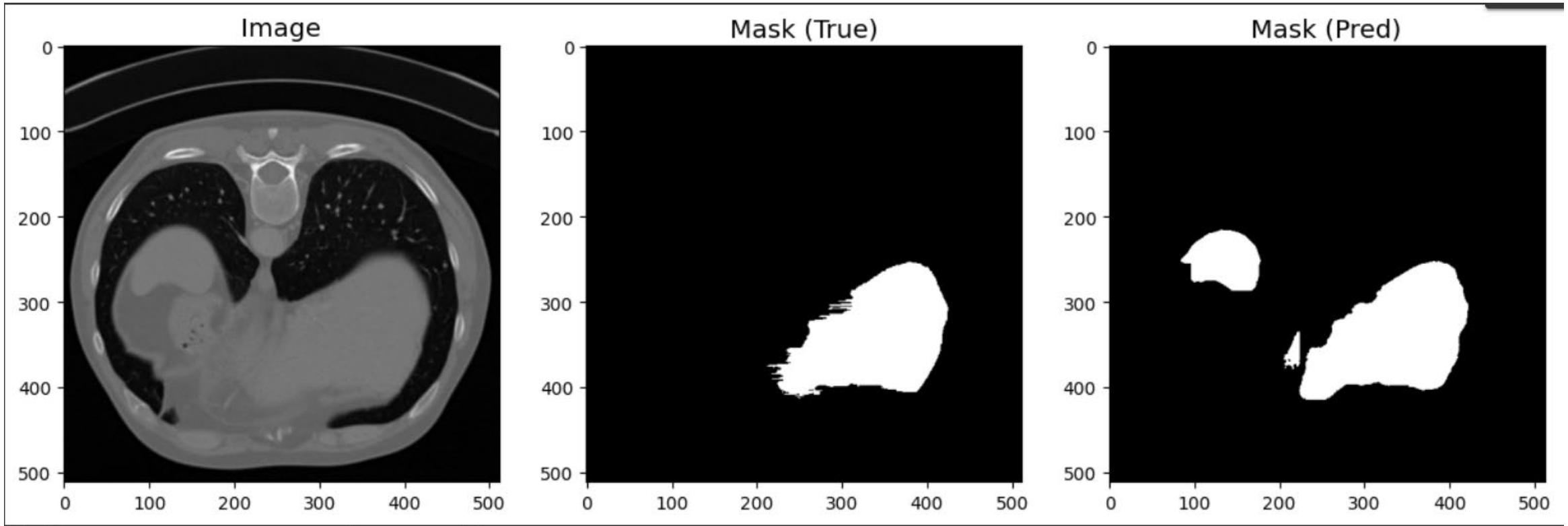
Saving Model

```
#儲存模型
model_json = model.to_json()
with open("./model_json_final.json", "w") as json_file:
        json_file.write(model_json)

# serialize weights to HDF5
model.save_weights("./model_weights_final.h5")
print("Saved model to disk")
```

```
# load weights into new model
json_file = open("./model_json_final.json", 'r')
loaded_model_json = json_file.read()
json_file.close()

loaded_model = model_from_json(loaded_model_json)
loaded_model.load_weights("./model_weights_final.h5")
print("Loaded model from disk")
```

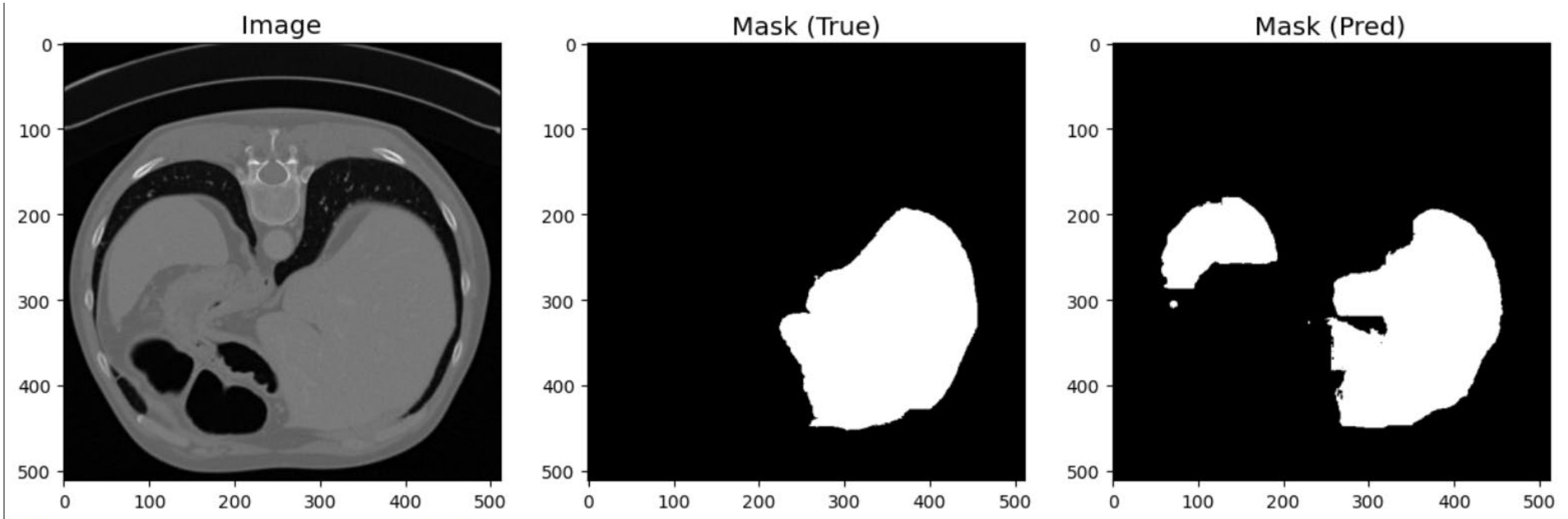


Prediction




```
8/8 [==============================] - 0s 12ms/step
Slice to Patch Shape: (225, 64, 64, 1)
Prediction Shape: (512, 512, 1)
Dice coef: tf.Tensor(0.8556331152982525, shape=(), dtype=float64)
```

Prediction




```
8/8 [==============================] - 0s 12ms/step
Slice to Patch Shape: (225, 64, 64, 1)
Prediction Shape: (512, 512, 1)
Dice coef: tf.Tensor(0.8408137781547451, shape=(), dtype=float64)
```

Thank you