機器學習於材料資訊的應用
Machine Learning on Material Informatics

陳南佑(NAN-YOW CHEN)

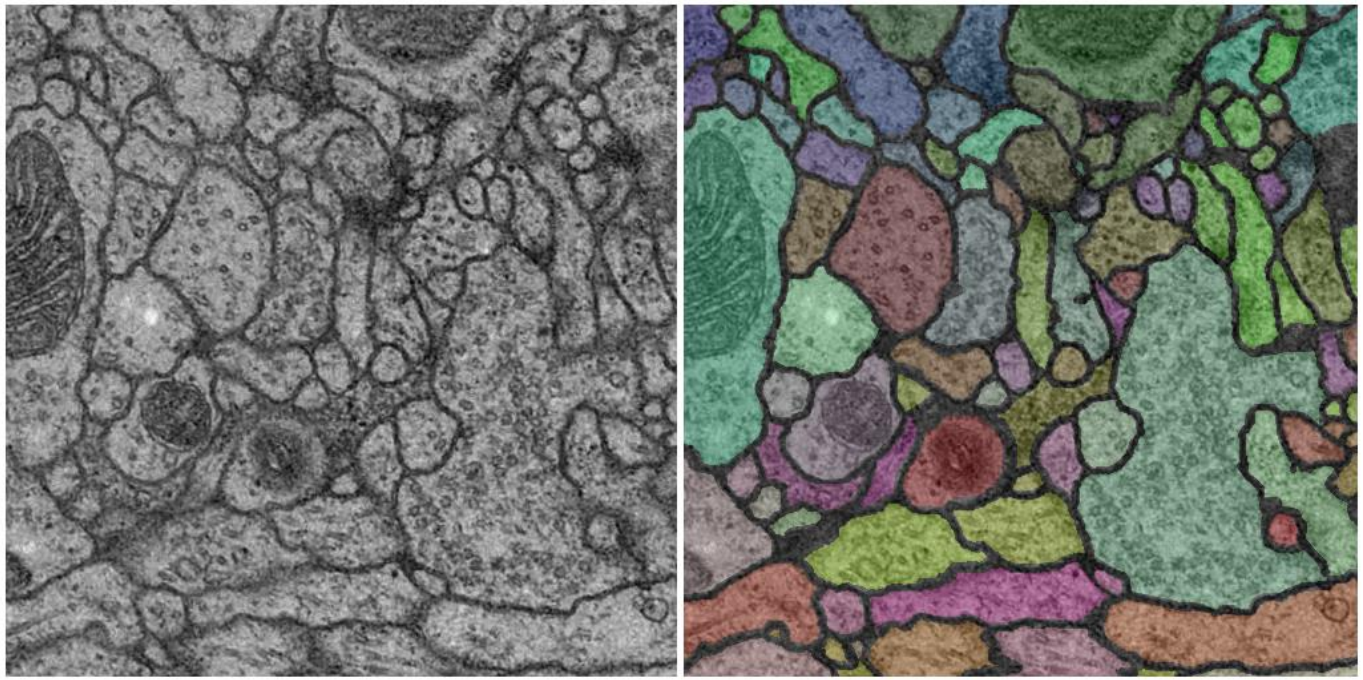
nanyow@narlabs.org.tw

楊安正(AN-CHENG YANG)

acyang@narlabs.org.tw

2D EM segmentation challenge

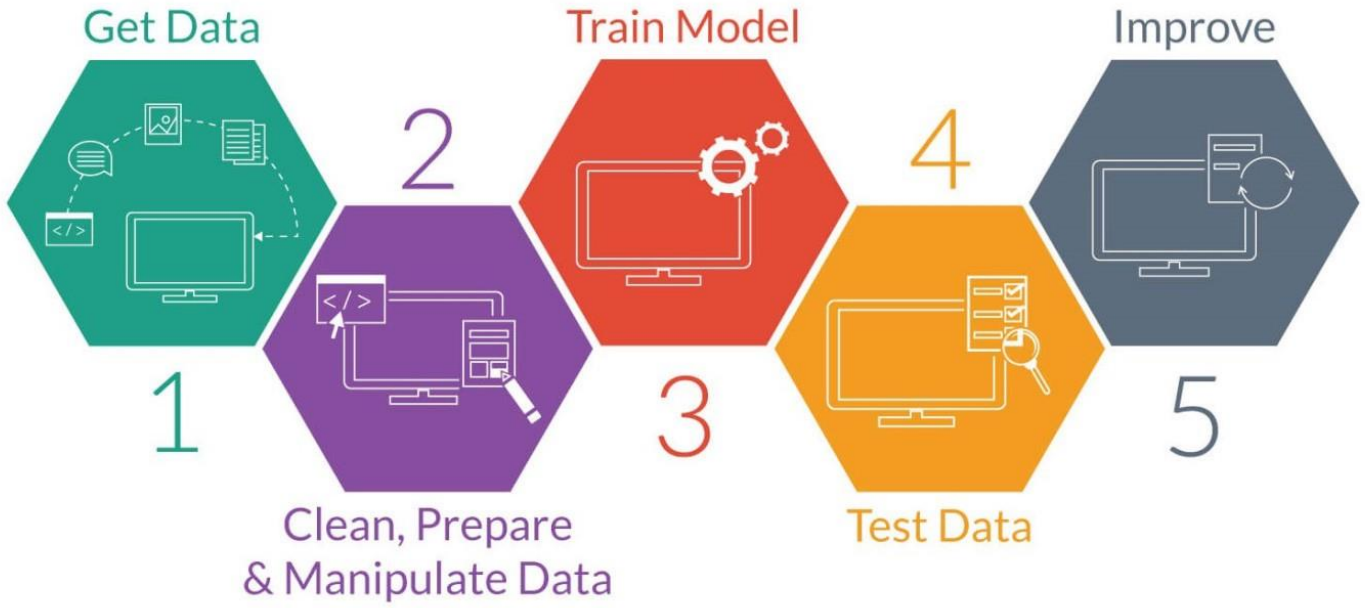
ISBI Challenge
Segmentation of neuronal structures in EM stacks



http://brainiac2.mit.edu/isbi_challenge/

ISBI Challenge

- This challenge was part of a workshop previous to the IEEE International Symposium on Biomedical Imaging (ISBI) 2012.
- First challenge on 2D segmentation of neuronal processes in EM images
- The training data is a set of 30 sections from a serial section Transmission Electron Microscopy (ssTEM) data set of the Drosophila first instar larva ventral nerve cord (VNC)神經索. The microcube measures 2 x 2 x 1.5 microns approx., with a resolution of 4x4x50 nm/pixel. (500x500x30)
- The results are expected to be submitted as a 32-bit [0 to 4294967295] TIFF 3D image, which values between 0 (100% membrane certainty) and 1 (100% non-membrane certainty).



從ssTEM取得資料

檔案處理

人工標記

建立網路

分類演算法

用測試資料檢驗演算法

改善人工標記

data.py

dataPrepare.ipynb

model.py

trainUnet.ipynb

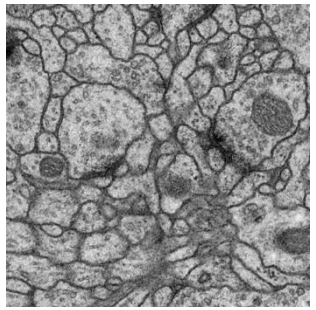
main.py

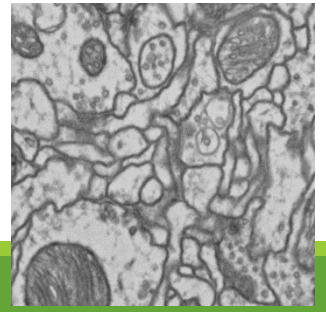
資料夾結構

- aug
  擴增影像的存放位置
- image
  原始影像資料
- label
  標記資料(mask)
- test
  測試資料

- Fixed
  標記影像斷點位置用的dataset
- Membrane
  分離細胞膜用的dataset
- Mitochondrion
  分離粒線體用的dataset

image



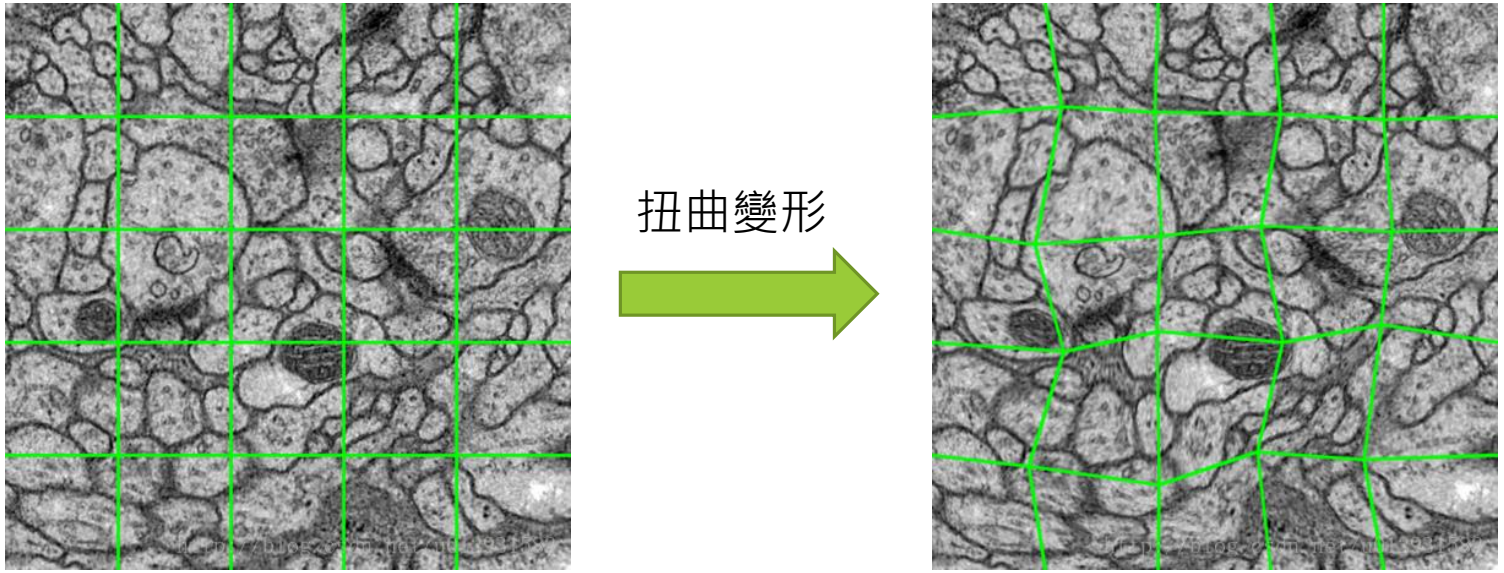


label

檔案處理

- 檔案處理主要仰賴keras.preprocessing.image 的 ImageDataGenerator功能。(https://keras.io/api/preprocessing/image/)
- 之所以可以進行資料擴增完全是基於一個假設:

扭曲圖像對於分類結果影響不大!!



檔案處理-dataPrepare.ipynb

- 資料擴增的引數透過dict形式存放，因為我們是依照ImageDataGenerator實作，所以我們的自訂函數也必須遵照ImageDataGenerator的設計。

```
data_gen_args = dict(rotation_range=20,
                     width_shift_range=0.05,
                     height_shift_range=0.05,
                     shear_range=0.05,
                     zoom_range=0.05,
                     horizontal_flip=True,
                     fill_mode='nearest')
```

rotation_range: 隨機旋轉的範圍
width_shift_range: 進行影像處理的範圍
height_shift_range:進行影像處理的範圍
shear_range: 扭曲的程度(逆時針扭)
zoom_range: 隨機縮放的範圍
horizontal_flip: 隨機水平翻轉
fill_mode: 填空模式

檔案處理-data.py

Generator是一個 Python 序列製作物件，可以用它來疊代一個可能很大的序列，在疊代的過程中所產生的值都是動態的，不需要將整個序列儲存在記憶體中， 就能產生更好的效能。
產生器表示式跟生成式的格式很像，差別在於將中括號「[]」改成小括號「()」，執行後可以看到產生器會回傳一個產生器物件 object，而不是串列。
產生器是記錄「產生值的方法」，而不是記錄值。
使用產生器中「產生的值只能取用一次」，無法重新啟動或重新取得 (因為不會紀錄)。

```
a = [i for i in range(10)]  # 序列生成式
b = (i for i in range(10))  # 產生器表示式
print(a)   # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(b)   # <generator object <genexpr> at 0x3c8hae835>
```

檔案處理-data.py

串列生成式，記憶體中保留了整份串列，再次取值時還是能得到數值，如果是產生器，再次取值時，就完全取不到值。

```
a = [i**2 for i in range(10)]
for i in a:
    print(i, end=' ')    # 0 1 4 9 16 25 36 49 64 81
for i in a:
    print(i, end=' ')    # 0 1 4 9 16 25 36 49 64 81
print()
b = (i**2 for i in range(10))
for i in b:
    print(i, end=' ')    # 0 1 4 9 16 25 36 49 64 81
for i in b:
    print(i, end=' ')    # 發生錯誤，取不到值
```

使用「**next**」的方法依序取值，但如果最後取不到值就會發生錯誤。

```
c = (i**2 for i in range(10))  # 產生器表示式
print(next(c))  # 0
print(next(c))  # 1
print(next(c))  # 4
print(next(c))  # 9
print(next(c))  # 16
print(next(c))  # 25
print(next(c))  # 36
print(next(c))  # 49
print(next(c))  # 64
print(next(c))  # 81
print(next(c))  # 發生錯誤，因為取不到值
```

檔案處理-data.py

普通函式是順序執行，遇到 return 語句就會返回。而 generator 函式會在每次調用 next() 的時候執行，遇到 yield 語句返回，再次執行時從上次返回的 yield 語句處繼續執行。

```
def f(max):
    n = 0
    while n<max:
        print(n)
        n += 1
f(5)

0
1
2
3
4
```

函式裡，包含 yield 陳述式，那麼這個函式就會變成一個產生器

```
def g(max):
    n = 0
    while n<max:
        yield(n)    # 換成 yield
        n += 1
g(5)
<generator object g at 0x7f45ebcc79d0>
h=g(5)
print(next(h))   #0
print(next(h))   #1
print(next(h))   #2
print(next(h))   #3
print(next(h))   #4
```

檔案處理-data.py

```
def trainGenerator(batch_size,train_path,image_folder,mask_folder,aug_dict,image_color_mode = "grayscale",
                   mask_color_mode = "grayscale",image_save_prefix  = "image",mask_save_prefix  = "mask",
                   flag_multi_class = False,num_class = 2,save_to_dir = None,target_size = (512,512),seed = 1):
```

```
train_path: ” Fixed”, “Membrane”, “Mitochondrion”三項任務擇一
image_folder: 影像存放位置 image
mask_folder: 標記資料存放位置label
```

檔案處理-data.py

- 同步處理影像集標記資料。

```
def trainGenerator(batch_size,train_path,image_folder,mask_folder,aug_dict,image_color_mode = "grayscale",
                   mask_color_mode = "grayscale",image_save_prefix  = "image",mask_save_prefix  = "mask",
                   flag_multi_class = False,num_class = 2,save_to_dir = None,target_size = (512,512),seed = 1):
```

```
image_generator =
image_datagen.flow_from_directory(
        train_path,
        classes = [image_folder],
        class_mode = None,
        color_mode = image_color_mode,
        target_size = target_size,
        batch_size = batch_size,
        save_to_dir = save_to_dir,
        save_prefix  = image_save_prefix,
        seed = seed)
```

```
mask_generator =
mask_datagen.flow_from_directory(
        train_path,
        classes = [mask_folder],
        class_mode = None,
        color_mode = mask_color_mode,
        target_size = target_size,
        batch_size = batch_size,
        save_to_dir = save_to_dir,
        save_prefix  = mask_save_prefix,
        seed = seed)
```

檔案處理-data.py

```
## zip() 函数用于將可疊代的物件作為參數，將物件中對應的元素打包成元組，然後返回由這些元組組成的物件。
## zip 方法在 Python 2 和 Python 3 中的不同：
## 在 Python 2.x zip() 返回的是一个列表。
## 在 Python 3.x 中為了減少記憶體，zip() 返回的是一個物件。如需展示列表，需手動使用 list() 轉換。
    train_generator = zip(image_generator, mask_generator)
    for (img,mask) in train_generator:
        …..
```

檔案處理-data.py

```
#調整成one-hot label
def adjustData(img,mask,flag_multi_class,num_class):
#產生測試資料
def testGenerator(test_path,num_image = 30,target_size = (512,512),flag_multi_class =
False,as_gray = True):
#以numpy格式存放資料
def geneTrainNpy(image_path,mask_path,flag_multi_class = False,num_class = 2,image_prefix
= "image",mask_prefix = "mask",image_as_gray = True,mask_as_gray = True):
#將結果套用顏色
def labelVisualize(num_class,color_dict,img):

def saveResult(save_path,npyfile,flag_multi_class = False,num_class = 2):
```

檔案處理-dataPrepare.ipynb

- 呼叫方式為:

```
myGenerator = trainGenerator(BatchSize,'Membrane','image','label',data_gen_args,save_to_dir = "Membrane/aug")
```

人工標記

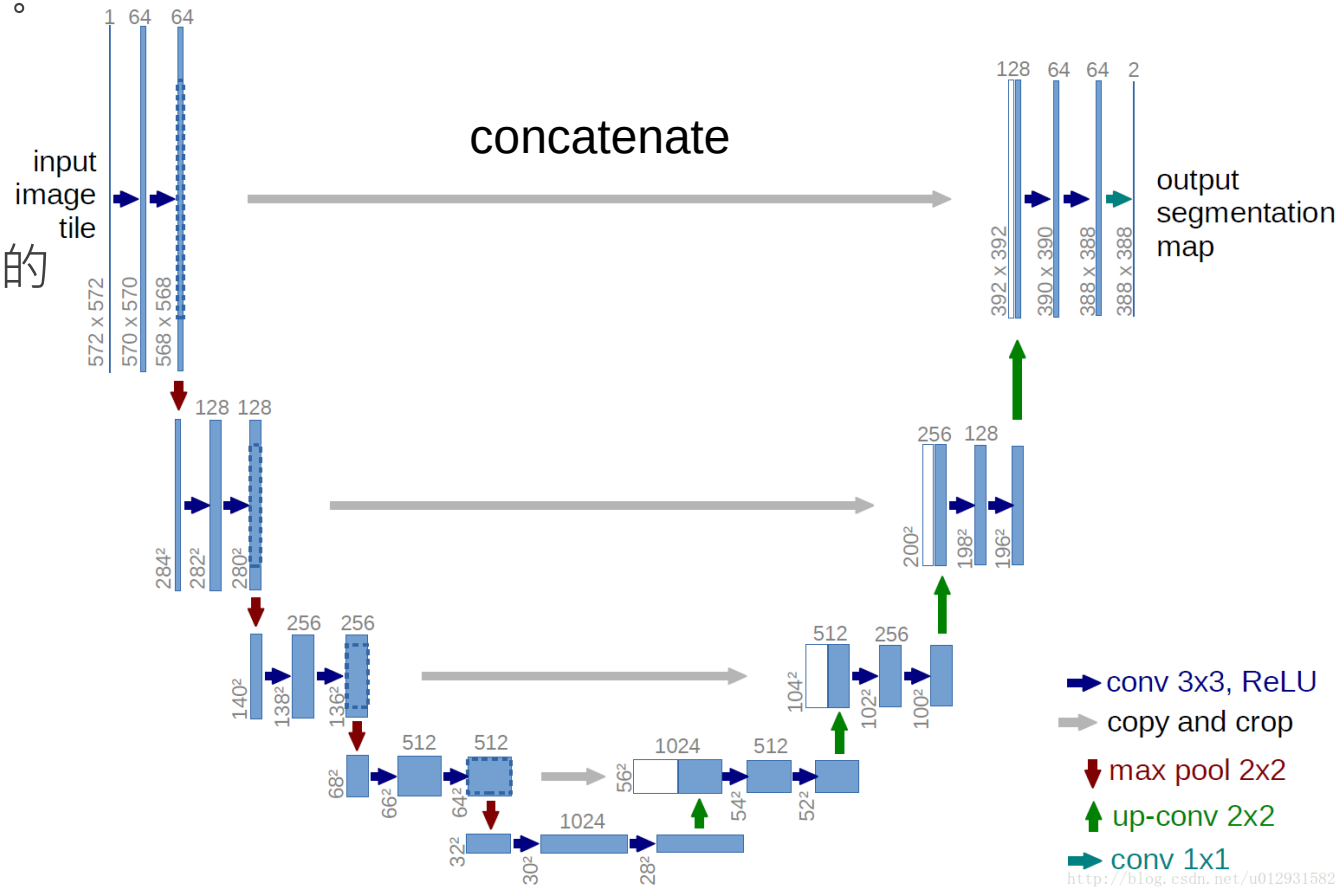
- 需要影像專家花時間投入。
- 感謝時任中研院物理所博士後研究員王定遠博士的高品質標記資料。

Down sampling

Up sampling

建立網路

- 每個藍色方塊代表的是multi-channel feature map。
- 藍色方塊上的數字代表的是channel數量。
- x-y-size代表影像本身的尺寸，標記在方塊左側。
- 白色方塊代表透過直通路徑 concatenate左側網路的feature map。
- 融合初階特徵已及高階特徵進行pixel的分類。
- 不同顏色箭頭代表不同的操作
  - 藍色:conv 3X3 ReLu
  - 白色: copy
  - 紅色: max pool 2X2
  - 綠色: up-conv 2X2
  - 藍綠: conv 1X1



https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

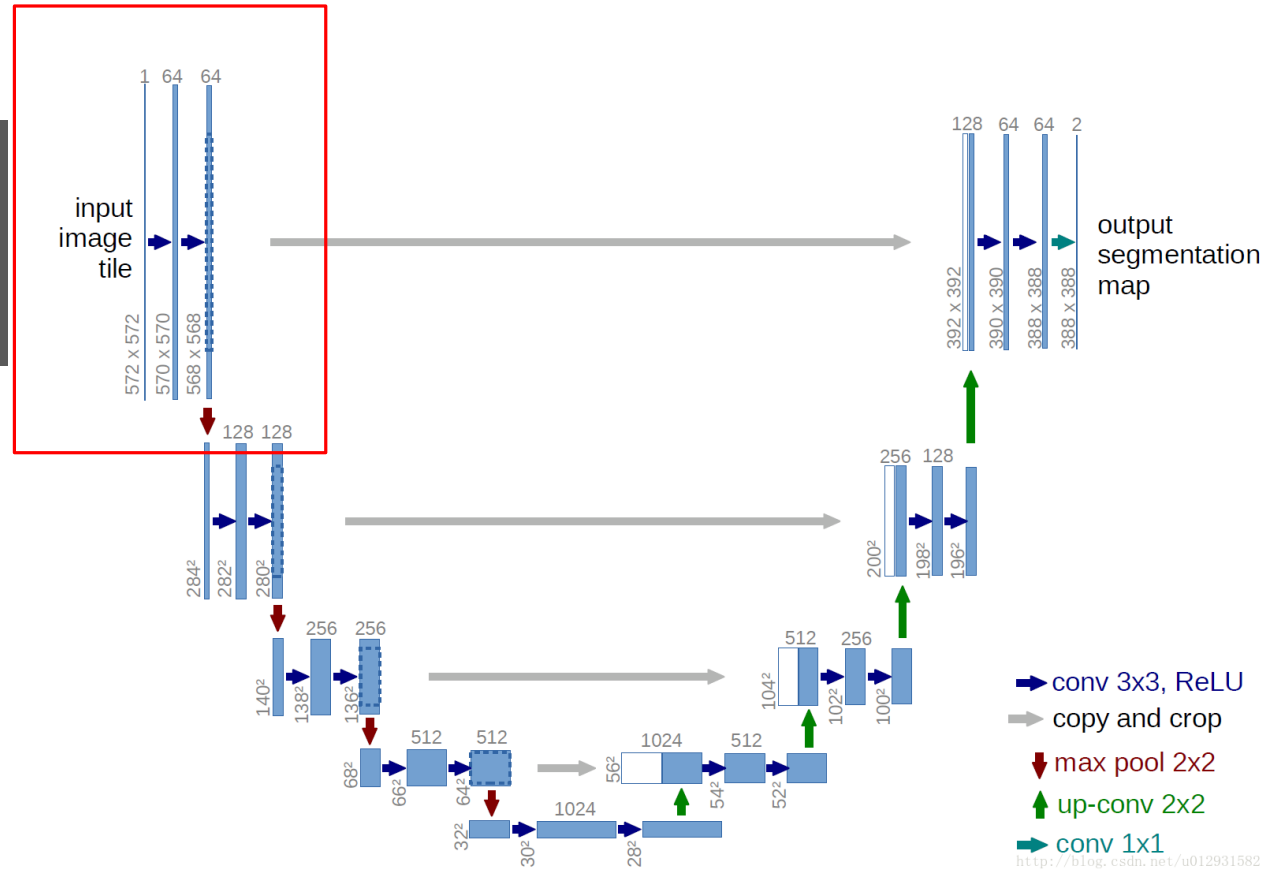
建立網路

```
import numpy as np
import os
import skimage.io as io
import skimage.transform as trans
import numpy as np
from keras.models import *
from keras.layers import *
from keras.optimizers import *
from keras.callbacks import ModelCheckpoint, LearningRateScheduler
from keras import backend as keras
```

建立網路

```
inputs = Input(input_size)
conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same',
kernel_initializer = 'he_normal')(inputs)
conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same',
kernel_initializer = 'he_normal')(conv1)
```

Kernel size都用3*3。
捲積層採用 same padding。(原始paper使用zero padding)

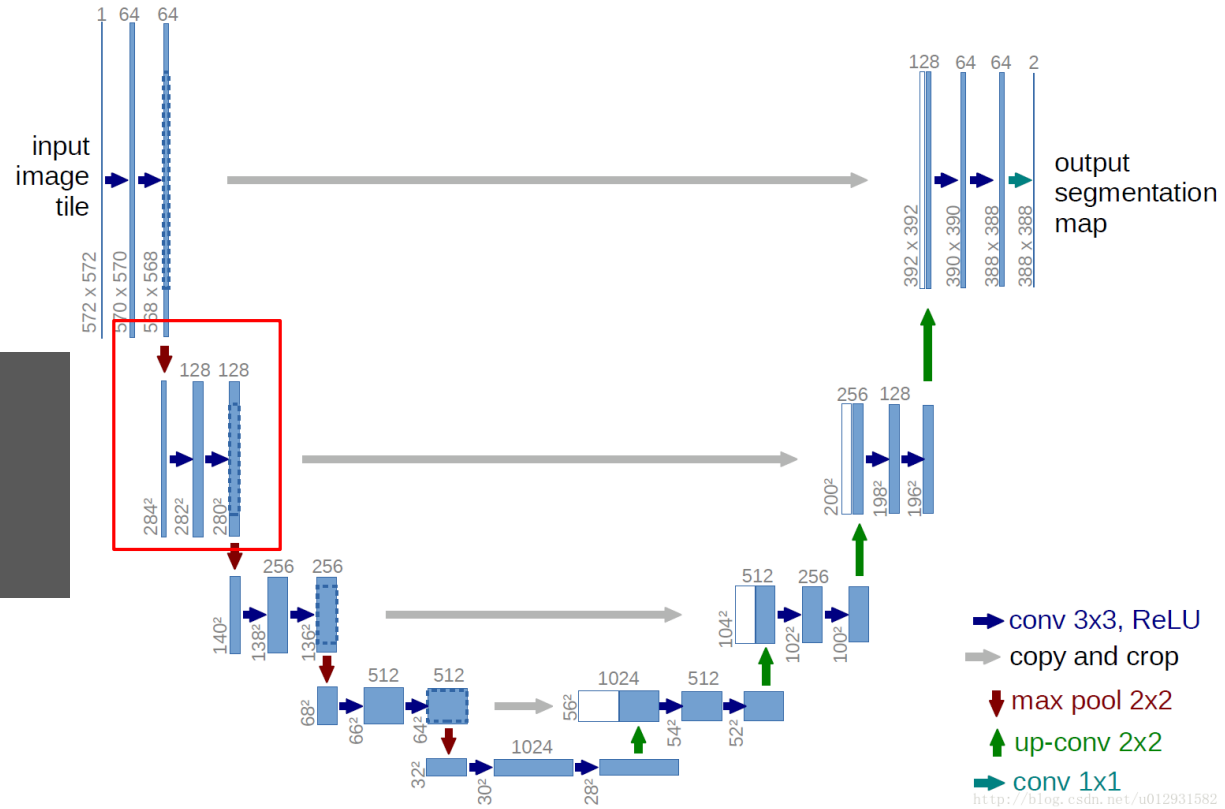


https://keras.io/zh/initializers/

建立網路

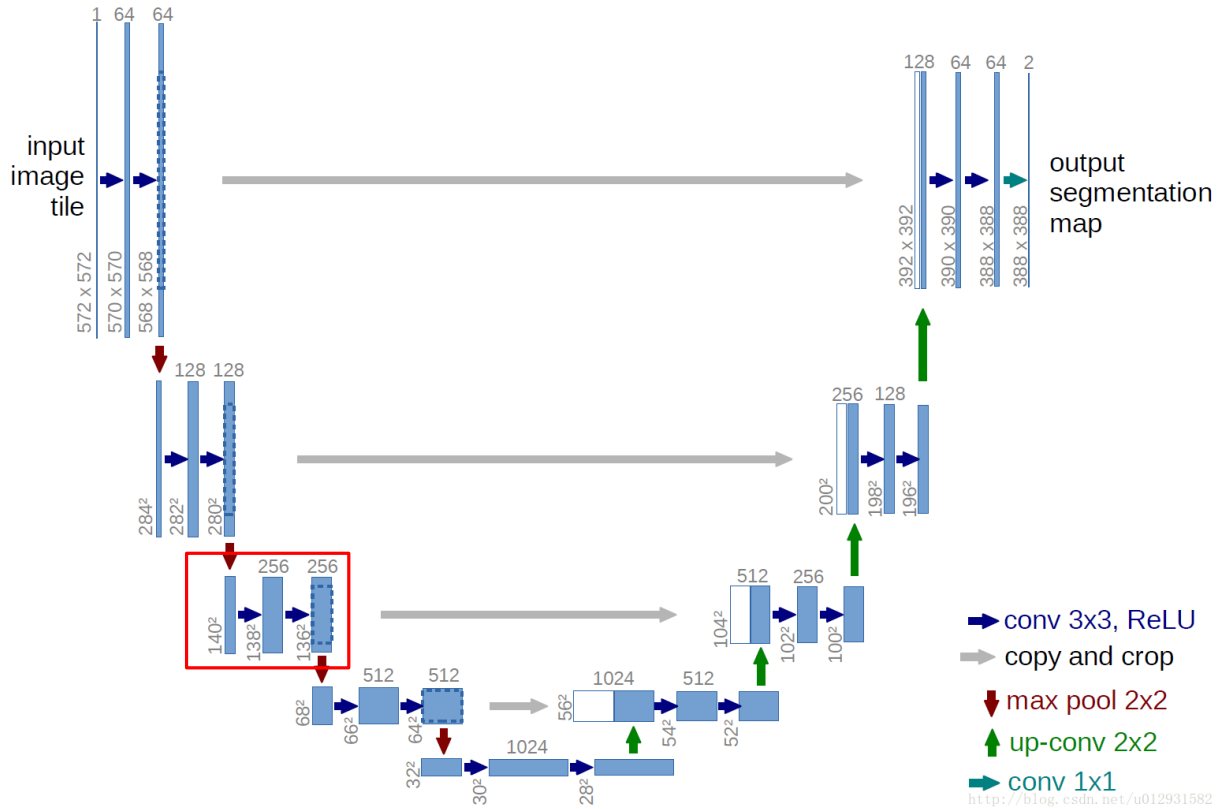
Down sampling採用MaxPooling。

```
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same',
kernel_initializer = 'he_normal')(pool1)
conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same',
kernel_initializer = 'he_normal')(conv2)
```



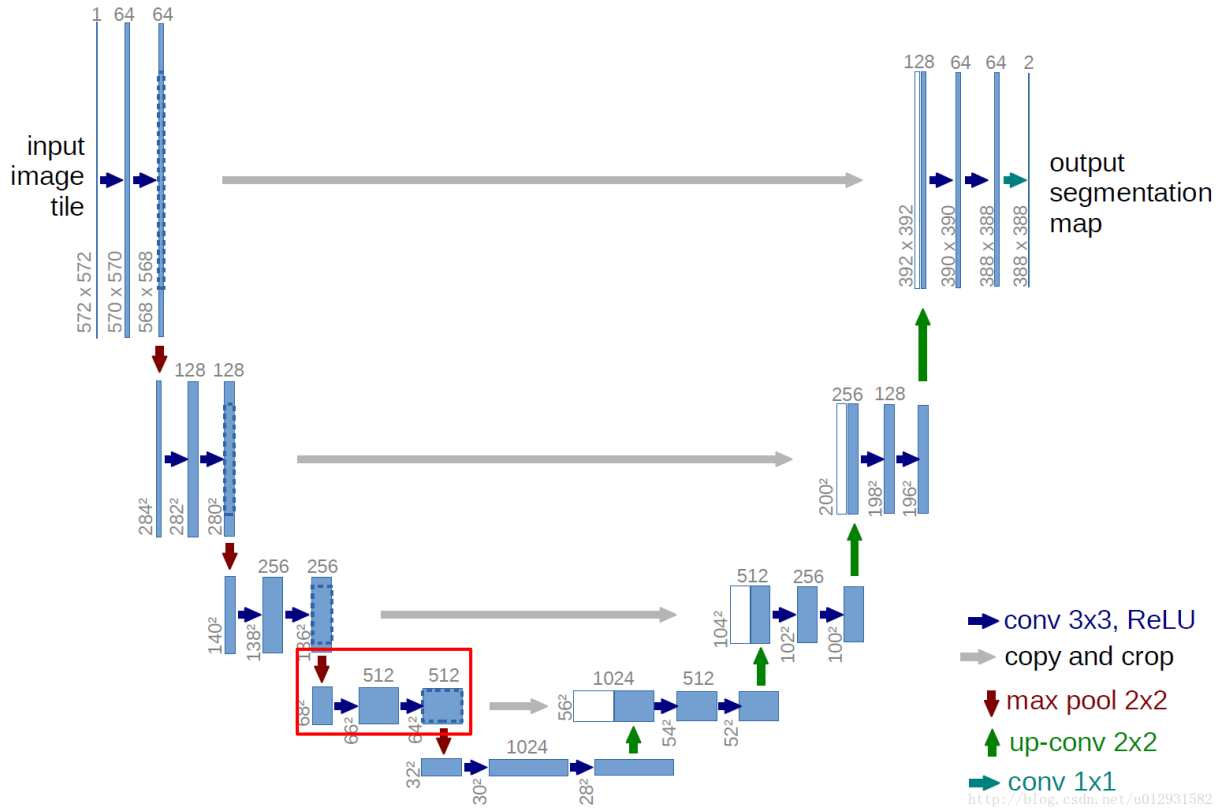
建立網路

```
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
    conv3 = Conv2D(256, 3, activation = 'relu', padding =
'same', kernel_initializer = 'he_normal')(pool2)
    conv3 = Conv2D(256, 3, activation = 'relu', padding =
'same', kernel_initializer = 'he_normal')(conv3)
```

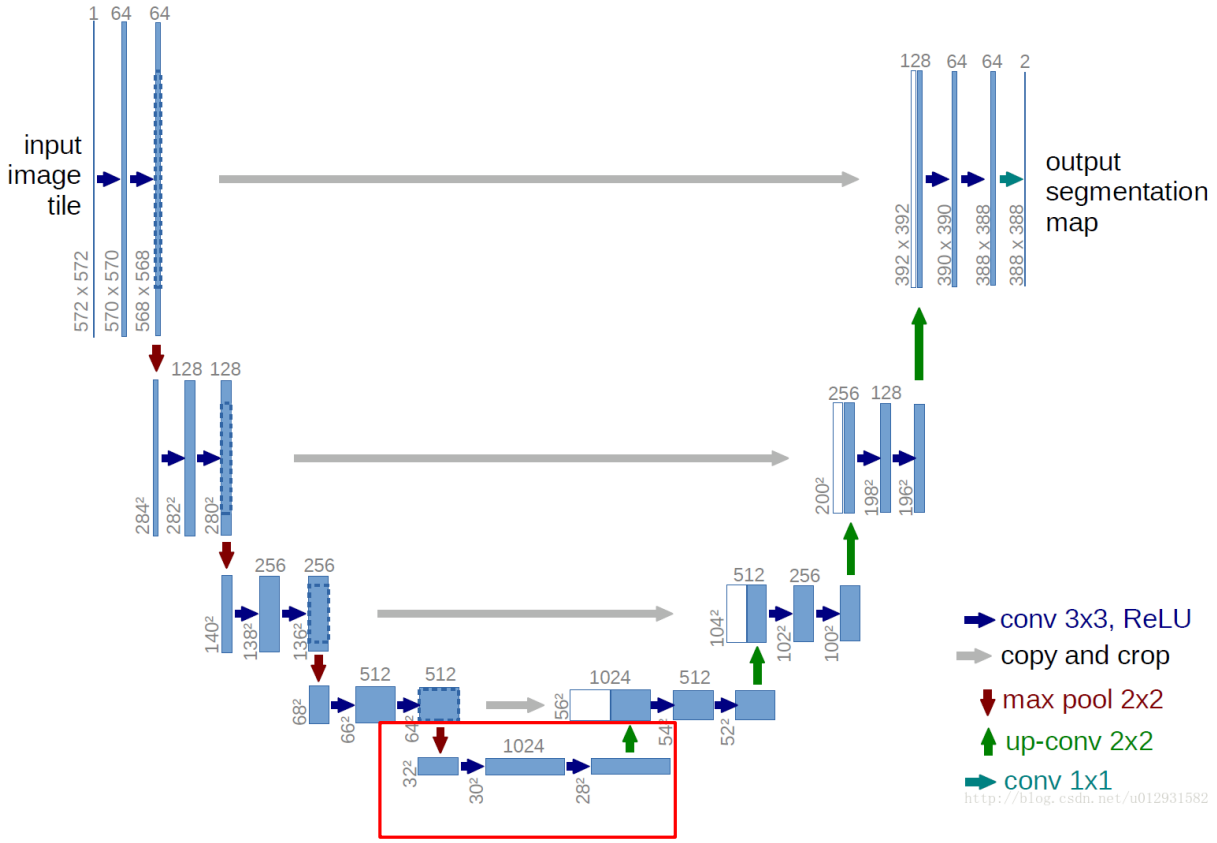


建立網路

```
pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
    conv4 = Conv2D(512, 3, activation = 'relu', padding =
'same', kernel_initializer = 'he_normal')(pool3)
    conv4 = Conv2D(512, 3, activation = 'relu', padding =
'same', kernel_initializer = 'he_normal')(conv4)
    drop4 = Dropout(0.5)(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)
```

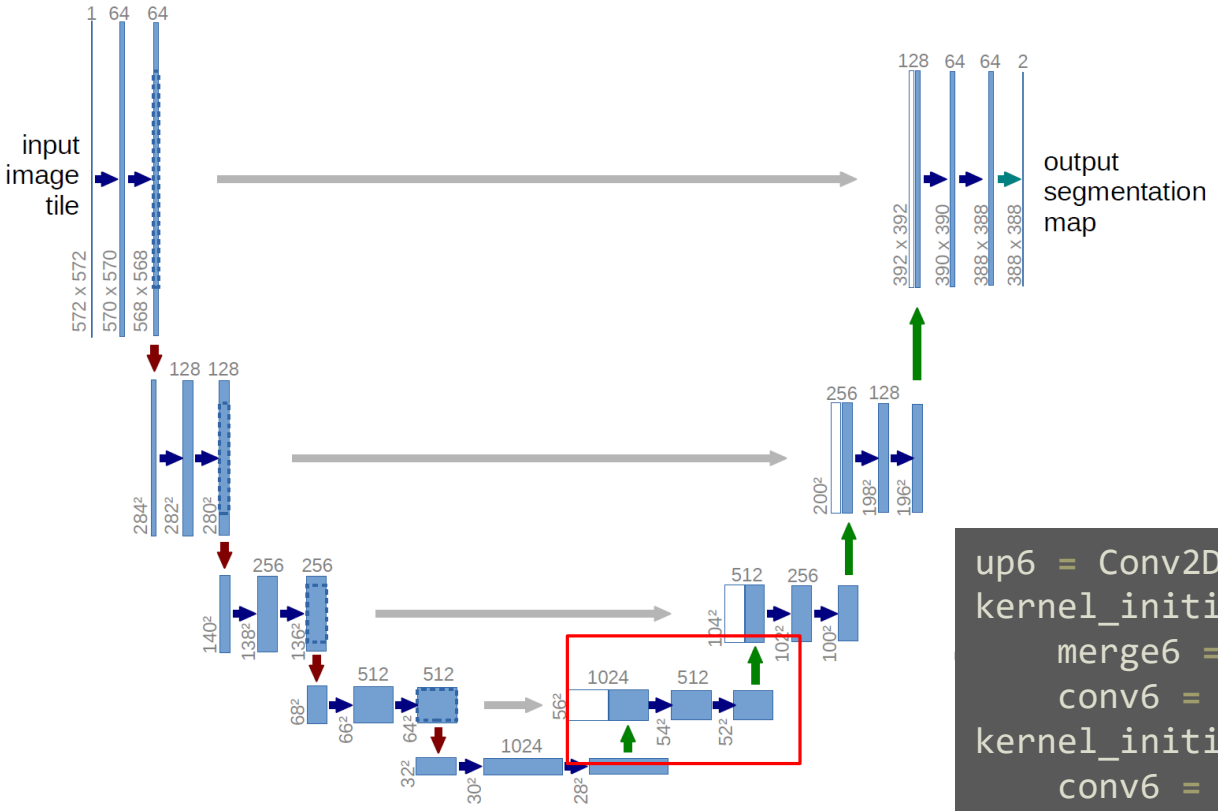


建立網路




```
conv5 = Conv2D(1024, 3, activation = 'relu', padding =
'same', kernel_initializer = 'he_normal')(pool4)
    conv5 = Conv2D(1024, 3, activation = 'relu', padding =
'same', kernel_initializer = 'he_normal')(conv5)
    drop5 = Dropout(0.5)(conv5)
```

建立網路

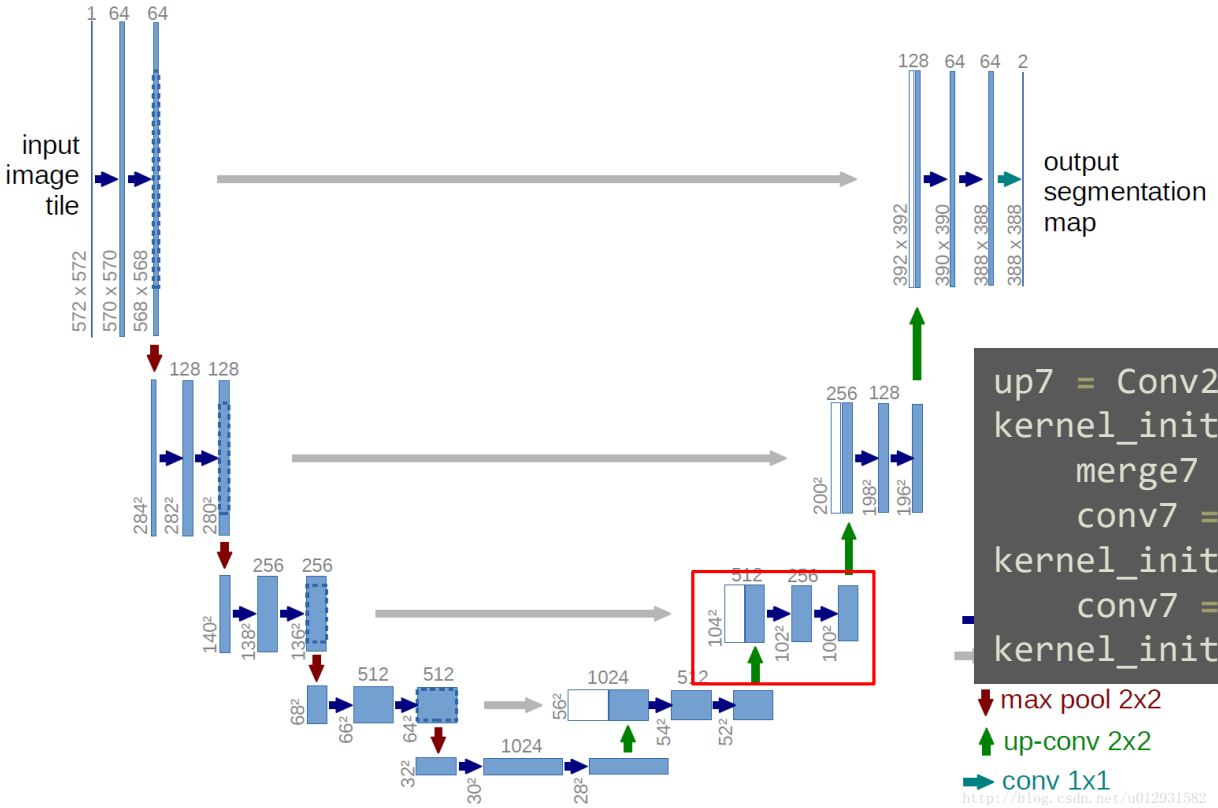


Up sampling操作使用UpSampling2D來放大。

```
up6 = Conv2D(512, 2, activation = 'relu', padding = 'same',
kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(drop5))
    merge6 = concatenate([drop4,up6], axis = 3)
    conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same',
kernel_initializer = 'he_normal')(merge6)
    conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same',
kernel_initializer = 'he_normal')(conv6)
```

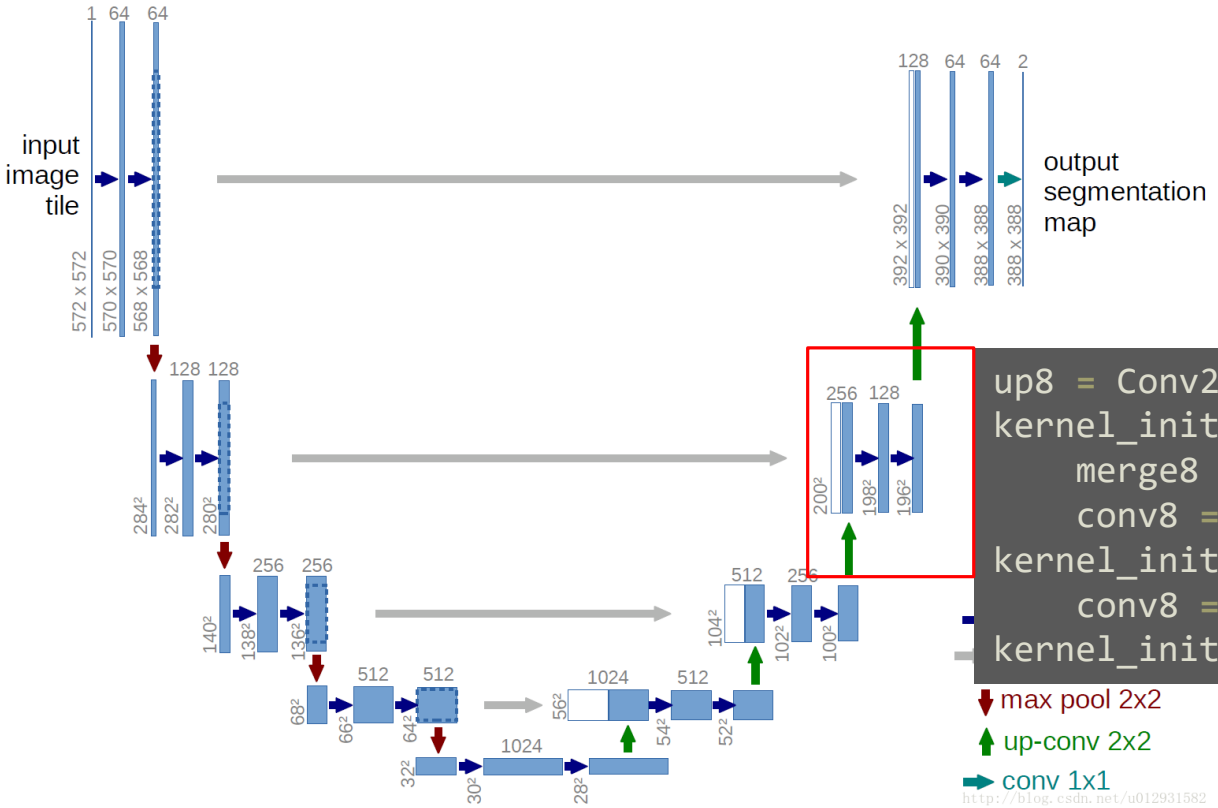
https://keras.io/api/layers/reshaping_layers/up_sampling2d/

建立網路



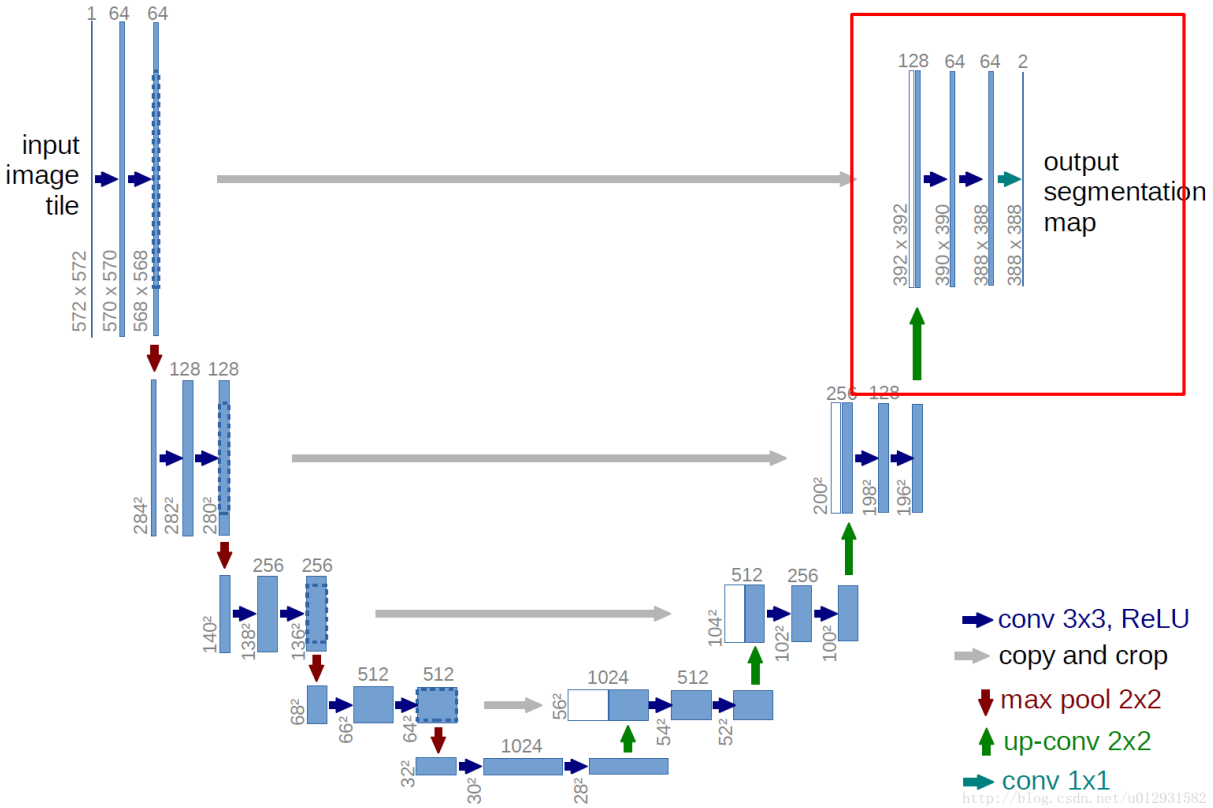

```
up7 = Conv2D(256, 2, activation = 'relu', padding = 'same',
kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(conv6))
    merge7 = concatenate([conv3,up7], axis = 3)
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same',
kernel_initializer = 'he_normal')(merge7)
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same',
kernel_initializer = 'he_normal')(conv7)
```

建立網路




```
up8 = Conv2D(128, 2, activation = 'relu', padding = 'same',
kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(conv7))
    merge8 = concatenate([conv2,up8], axis = 3)
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same',
kernel_initializer = 'he_normal')(merge8)
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same',
kernel_initializer = 'he_normal')(conv8)
```

建立網路



最後的分類使用1*1的捲積

```
model = Model(input = inputs, output = conv10)

    model.compile(optimizer = Adam(lr = 1e-4), loss =
'binary_crossentropy', metrics = ['accuracy'])

    #model.summary()

    if(pretrained_weights):
        model.load_weights(pretrained_weights)

    return model
```

訓練網路

```
myGene =
trainGenerator(BatchSize,'membrane/train','image','label',data_gen_
args,save_to_dir = None)

model = unet()
model_checkpoint = ModelCheckpoint('unet_membrane.hdf5',
monitor='loss',verbose=1, save_best_only=True)
model.fit_generator(myGene,steps_per_epoch=300,epochs=1,callbacks=[
model_checkpoint])
```

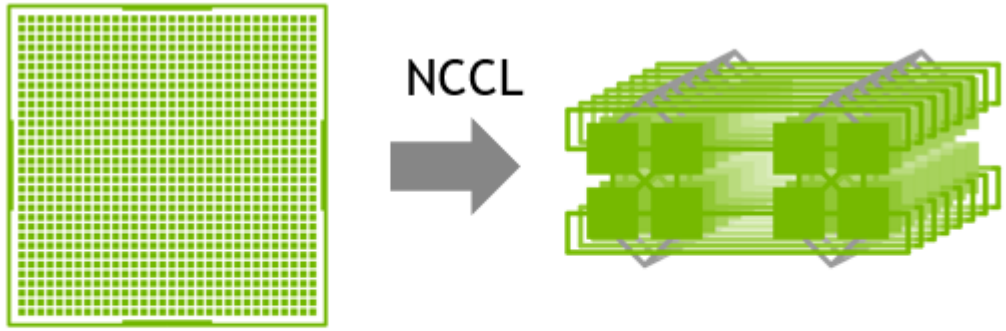
訓練網路

```
model.fit_generator(myGene,steps_per_epoch=2000,epochs=10,callbacks=[model_checkpoint])

#利用generator節省記憶體的消耗，還可以動態進行資料擴增，提高模型泛化的能力。

model.fit(myGene,steps_per_epoch=2000,epochs=10,callbacks=[model_checkpoint])

#對於小的數據資料集，沒有資料擴增，可以完整放入記憶體當中，
自從tf2.1開始，fit方法也支援generator
```

使用訓練過的網路(模型)

```
testGene = testGenerator("data/membrane/test")
model = unet()
model.load_weights("UNet_Membrane.hdf5")
results = model.predict_generator(testGene,30,verbose=1)
saveResult("data/membrane/test",results)
```

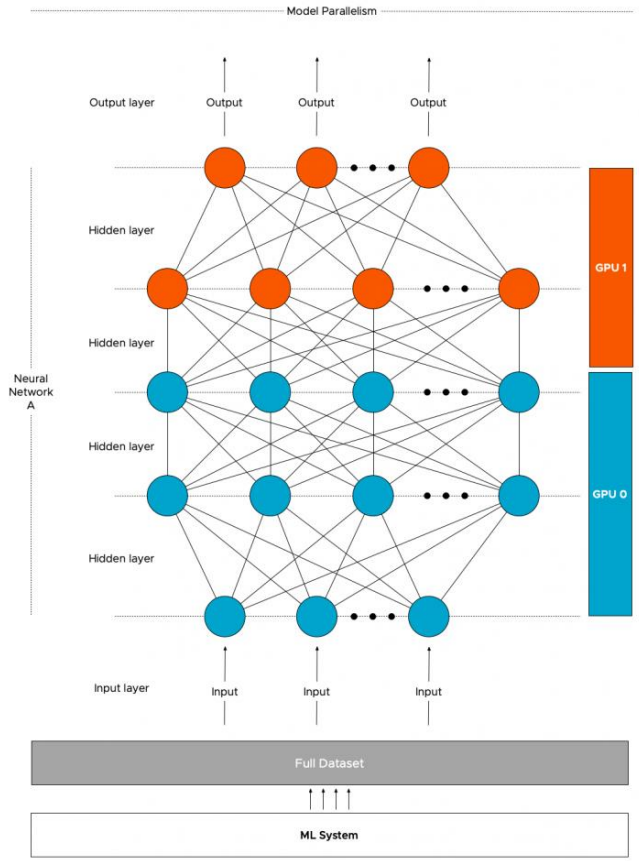
Multi-GPU and Distributed Deep Learning

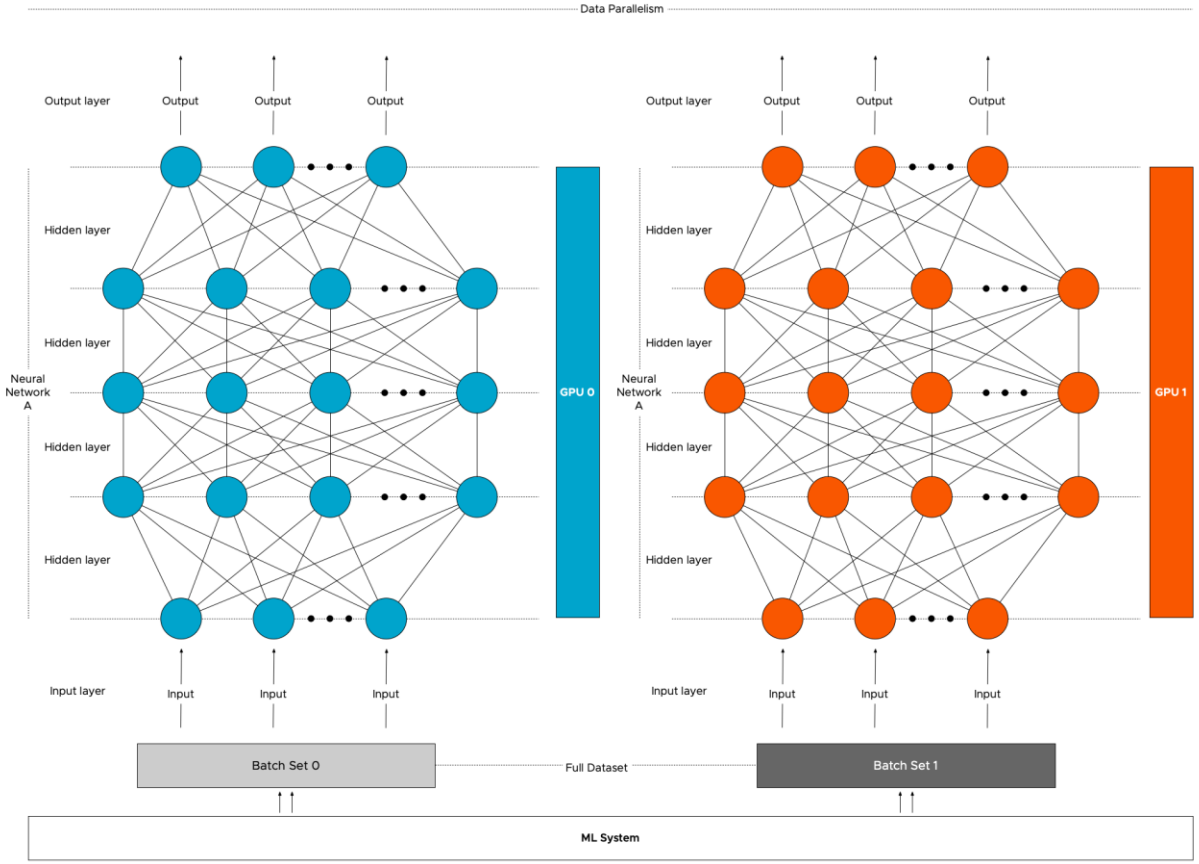


1 GPU

multi-GPU, multi-node

Model parallelism & Data Parallelism





https://frankdenneman.nl/2020/02/19/multi-gpu-and-distributed-deep-learning/

Distributed strategies

- MirroredStrategy
  - 適合Data Parallelism，將資料分在不同的硬體上，每個硬體有自己的訓練參數，利用all_reduce運算統合不同模型間的參數。
- TPUStrategy
  - 和MirroredStrategy雷同，專門為TPU特化，在colab可以用。
- ParameterServerStrategy
  - 適用在Model parallelism，不同硬體負責模型不同的部分，分為worker 和 parameter server，每個步驟worker都會去parameter server讀取參數，執行計算更新參數，再把參數同步回 parameter server。
- MultiWorkerMirroredStrategy
  - 適用在Model parallelism，在使用更多硬體時，模型的單一部分也可融入MirroredStrategy進行Data Parallelism，而整個大的模型依然是採用ParameterServerStrategy。
- CentralStorageStrategy
  - 發展中，參數不放在parameter server，而是放在CPU上，所有機器上的GPU都可以共享參數而不用複製一份，也不需要各GPU all_reduce參數，也不需要從worker同步回parameter server 。

Usage:

```
strategy = tf.distribute.MirroredStrategy()
print('Number of devices: {}'.format(strategy.num_replicas_in_sync))
```

```
BatchSize_per_Replica = 1
BatchSize = BatchSize_per_Replica * strategy.num_replicas_in_sync
print(BatchSize)
```

Usage:

```
with strategy.scope():

    model = unet()

    model_checkpoint = tf.keras.callbacks.ModelCheckpoint('UNet_Membrane.hdf5',
monitor='loss',verbose=1, save_best_only=True)


model.fit(myGene,steps_per_epoch=1000,epochs=5,callbacks=[model_checkpoint,tf.keras.
callbacks.TensorBoard(log_dir='./logs')])
```

Results

1GPU

Time=13xx.xxxsec

2GPU

Time=868.557274103164 7sec