Up Convolution

Convolution

$$\underbrace{\begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix}}_{\text{Learned Kernel } K\in\mathbb{R}^{3\times3}} \otimes \underbrace{\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 14 \end{bmatrix}}_{\text{Input}}$$

$$= \underbrace{\begin{bmatrix} k_{11}+2k_{12}+3k_{13}+5k_{21}+6k_{22}+7k_{23}+9k_{31}+10k_{32}+11k_{33} & 2k_{11}+3k_{12}+4k_{13}+6k_{21}+7k_{22}+8k_{23}+10k_{31}+11k_{32}+12k_{33} \\ 5k_{11}+6k_{12}+7k_{13}+9k_{21}+10k_{22}+11k_{23}+13k_{31}+14k_{32}+15k_{33} & 6k_{11}+7k_{12}+8k_{13}+10k_{21}+11k_{22}+12k_{23}+14k_{31}+15k_{32}+16k_{33} \end{bmatrix}}_{\text{Output}}$$

$$= \underbrace{\begin{bmatrix} o_{11} & o_{12} \\ o_{21} & o_{22} \end{bmatrix}}_{\text{Output}}$$

The 3×3 kernel takes the 4×4 input to a 2×2 output.

Unroll $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 14 \end{bmatrix}$, $\begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix}$ and $\begin{bmatrix} o_{11} & o_{12} \\ o_{21} & o_{22} \end{bmatrix}$ to rewrite the above convolution as

$$\underbrace{\begin{bmatrix} k_{11} & k_{12} & k_{13} & 0 & k_{21} & k_{22} & k_{23} & 0 & k_{31} & k_{32} & k_{33} & 0 & 0 & 0 & 0 & 0 \\ 0 & k_{11} & k_{12} & k_{13} & 0 & k_{21} & k_{22} & k_{23} & 0 & k_{31} & k_{32} & k_{33} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & k_{11} & k_{12} & k_{13} & 0 & k_{21} & k_{22} & k_{23} & 0 & k_{31} & k_{32} & k_{33} & 0 \\ 0 & 0 & 0 & 0 & 0 & k_{11} & k_{12} & k_{13} & 0 & k_{21} & k_{22} & k_{23} & 0 & k_{31} & k_{32} & k_{33} \end{bmatrix}}_{\text{Learned Kernel } K\in\mathbb{R}^{4\times16}} \underbrace{\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \end{bmatrix}}_{\text{Input } I\in\mathbb{R}^{16}}$$

$$= \underbrace{\begin{bmatrix} k_{11}+2k_{12}+3k_{13}+5k_{21}+6k_{22}+7k_{23}+9k_{31}+10k_{32}+11k_{33} \\ 2k_{11}+3k_{12}+4k_{13}+6k_{21}+7k_{22}+8k_{23}+10k_{31}+11k_{32}+12k_{33} \\ 5k_{11}+6k_{12}+7k_{13}+9k_{21}+10k_{22}+11k_{23}+13k_{31}+14k_{32}+15k_{33} \\ 6k_{11}+7k_{12}+8k_{13}+10k_{21}+11k_{22}+12k_{23}+14k_{31}+15k_{32}+16k_{33} \end{bmatrix}}_{\text{Output}}$$

$$= \underbrace{\begin{bmatrix} o_{11} \\ o_{12} \\ o_{21} \\ o_{22} \end{bmatrix}}_{\text{Output } O\in\mathbb{R}^{4}}$$

We can write this as

$$O = KI, \quad K \in \mathbb{R}^{4\times16}. \tag{1}$$

Transposed Convolution

Example Use a 3×3 kernel to take a 2×2 input to a 4×4 output.

$$\underbrace{\begin{bmatrix} o_{11} \\ o_{12} \\ o_{13} \\ o_{14} \\ o_{21} \\ o_{22} \\ o_{23} \\ o_{24} \\ o_{31} \\ o_{32} \\ o_{33} \\ o_{34} \\ o_{41} \\ o_{42} \\ o_{43} \\ o_{44} \end{bmatrix}}_{\text{Output}} = \underbrace{\begin{bmatrix} k_{11} & 0 & 0 & 0 \\ k_{12} & k_{11} & 0 & 0 \\ k_{13} & k_{12} & 0 & 0 \\ 0 & k_{13} & 0 & 0 \\ k_{21} & 0 & k_{11} & 0 \\ k_{22} & k_{21} & k_{12} & k_{11} \\ k_{23} & k_{22} & k_{13} & k_{12} \\ 0 & k_{23} & 0 & k_{13} \\ k_{31} & 0 & k_{21} & 0 \\ k_{32} & k_{31} & k_{22} & k_{21} \\ k_{33} & k_{32} & k_{23} & k_{22} \\ 0 & k_{33} & 0 & k_{23} \\ 0 & 0 & k_{31} & 0 \\ 0 & 0 & k_{32} & k_{31} \\ 0 & 0 & k_{33} & k_{32} \\ 0 & 0 & 0 & k_{33} \end{bmatrix}}_{\text{Learned Kernel } K^T \in \mathbb{R}^{16\times 4}} \underbrace{\begin{bmatrix} i_{11} \\ i_{12} \\ i_{21} \\ i_{22} \end{bmatrix}}_{\text{Input}}$$

$$\underbrace{\begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix}}_{\text{Learned Kerned } K^T} \underset{\text{up}}{\otimes} \underbrace{\begin{bmatrix} i_{11} & i_{12} \\ i_{21} & i_{22} \end{bmatrix}}_{\text{Input}} = \underbrace{\begin{bmatrix} o_{11} & o_{12} & o_{13} & o_{14} \\ o_{21} & o_{22} & o_{23} & o_{24} \\ o_{31} & o_{32} & o_{33} & o_{34} \\ o_{41} & o_{42} & o_{43} & o_{44} \end{bmatrix}}_{\text{Output}}$$

Example Use a 2×2 kernel to take a 2×2 input to a 4×4 output.

$$\underbrace{\begin{bmatrix} o_{11} \\ o_{12} \\ o_{13} \\ o_{14} \\ o_{21} \\ o_{22} \\ o_{23} \\ o_{24} \\ o_{31} \\ o_{32} \\ o_{33} \\ o_{34} \\ o_{41} \\ o_{42} \\ o_{43} \\ o_{44} \end{bmatrix}}_{\text{Output}} = \underbrace{\begin{bmatrix} k_{11} & 0 & 0 & 0 \\ k_{12} & 0 & 0 & 0 \\ 0 & k_{11} & 0 & 0 \\ 0 & k_{12} & 0 & 0 \\ k_{21} & 0 & 0 & 0 \\ k_{22} & 0 & 0 & 0 \\ 0 & k_{21} & 0 & 0 \\ 0 & k_{22} & 0 & 0 \\ 0 & 0 & k_{11} & 0 \\ 0 & 0 & k_{12} & 0 \\ 0 & 0 & 0 & k_{11} \\ 0 & 0 & 0 & k_{12} \\ 0 & 0 & k_{21} & 0 \\ 0 & 0 & k_{22} & 0 \\ 0 & 0 & 0 & k_{21} \\ 0 & 0 & 0 & k_{22} \end{bmatrix}}_{\text{Learned Kernel } K^T \in \mathbb{R}^{16\times 4}} \underbrace{\begin{bmatrix} i_{11} \\ i_{12} \\ i_{21} \\ i_{22} \end{bmatrix}}_{\text{Input}}$$

By equation (1) we have $O = KI$ with $K \in \mathbb{R}^{4\times 16}$. Multiplying this through by K^T gives

$$K^T O = K^T K I, \quad K^T K \in \mathbb{R}^{16\times 16}.$$

If $K^T K$ were invertible then we could solve for I according to $I = (K^T K)^{-1} K^T O$. However $K^T K$ is not invertible as $rank(K) \leq 4$ so $rank(K^T K) \leq 4$. The point here is that transposed (up) convolution will **not** lead to an inverse of the original (down) convolution.

1×1 Convolution Kernels

A kernel $k \in \mathbb{R}^{1\times1\times64}$ will take 64 maps/channels of size $d \times d$ to a single map of size $d \times d$. Then 32 kernels $K \in \mathbb{R}^{32\times1\times1\times64}$ will take 64 maps/channels of size $d \times d$ to 32 maps/channels of size $d \times d$.

Semantic segmentation

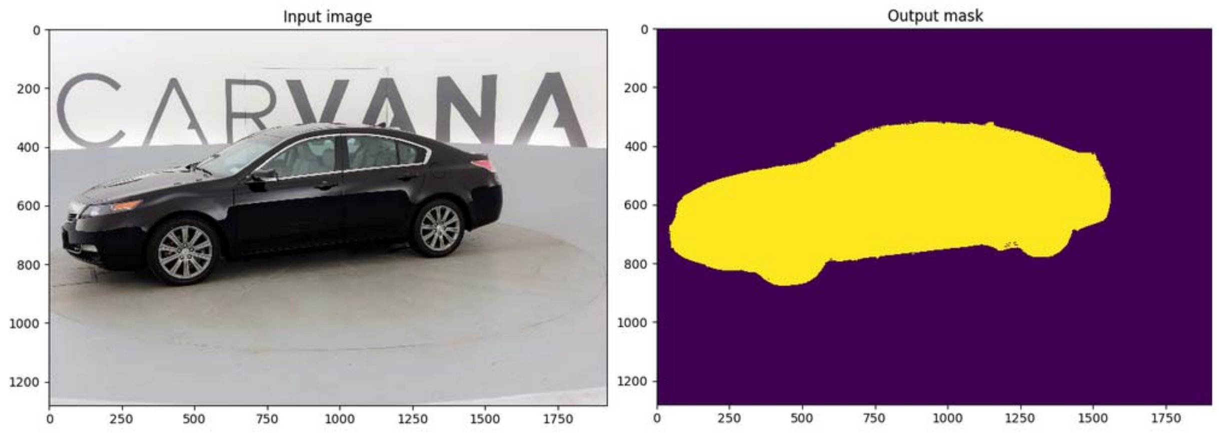

Figure 1: From *https://github.com/milesial/Pytorch-UNet*

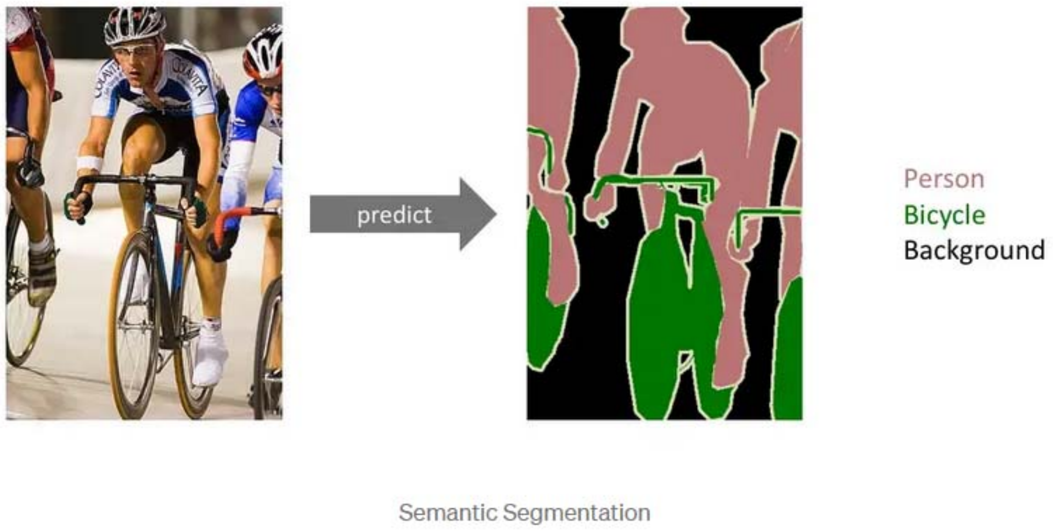

Figure 2: From *https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47*

Overlap tile strategy [1]

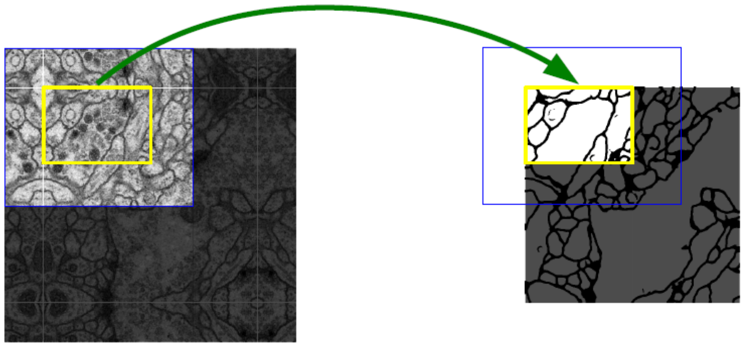

Fig. 2. Overlap-tile strategy for seamless segmentation of arbitrary large images (here segmentation of neuronal structures in EM stacks). Prediction of the segmentation in the yellow area, requires image data within the blue area as input. Missing input data is extrapolated by mirroring

Up convolution [1]

In the paper by Ronneberger et al. [1] *up convolution* is used instead of transpose convolution. A 2×2 up convolution is illustrated in Figure 3. Here, $k = 1, ..., 5$ denotes the input channel/map and ℓ denotes the output channel/map. There are 20 weights and 1 bias. In the figure it is stated that the output map has its *resolution* increased by a factor of 2. This not true. The *size* of the output channel is increased by a factor of 2 in width and height. The resolution is the same.

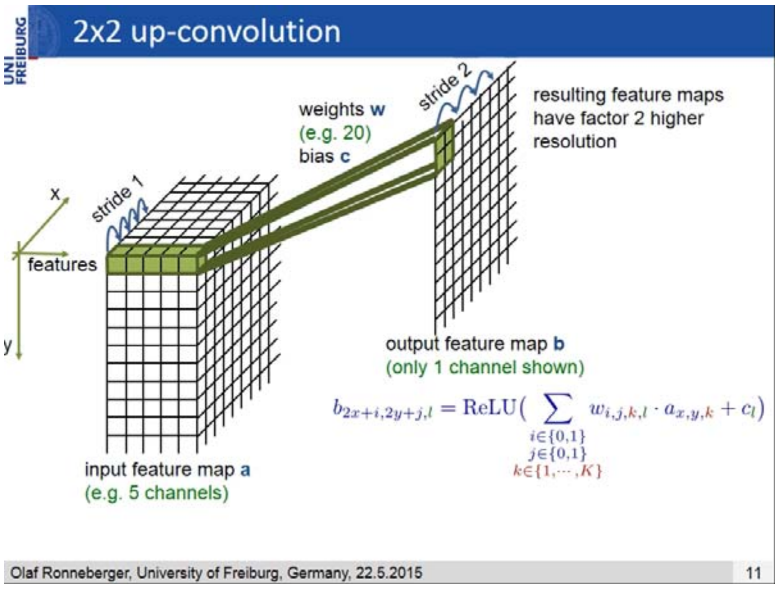
UNI FREIBURG
2x2 up-convolution
weights **w**
(e.g. 20)
bias **c**
stride 2
resulting feature maps have factor 2 higher resolution
stride 1
x
features
y
output feature map **b**
(only 1 channel shown)
$b_{2x+i,2y+j,l} = \mathrm{ReLU}(\sum_{i\in\{0,1\}, j\in\{0,1\}, k\in\{1,\cdots,K\}} w_{i,j,k,l} \cdot a_{x,y,k} + c_l)$
input feature map **a**
(e.g. 5 channels)
Olaf Ronneberger, University of Freiburg, Germany, 22.5.2015
11

Figure 3: Up Convolution.

Another way to write this up convolution is as follows.

$$\begin{bmatrix} b^{(\ell)}_{2x,2y} \\ b^{(\ell)}_{2x+1,2y} \\ b^{(\ell)}_{2x,2y+1} \\ b^{(\ell)}_{2x+1,2y+1} \end{bmatrix} = \mathrm{ReLU}\left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} w^{(\ell,1)}_{11} & w^{(\ell,2)}_{11} & w^{(\ell,3)}_{11} & w^{(\ell,4)}_{11} & w^{(\ell,5)}_{11} \\ w^{(\ell,1)}_{12} & w^{(\ell,2)}_{12} & w^{(\ell,3)}_{12} & w^{(\ell,4)}_{12} & w^{(\ell,5)}_{12} \\ w^{(\ell,1)}_{21} & w^{(\ell,2)}_{21} & w^{(\ell,3)}_{21} & w^{(\ell,4)}_{21} & w^{(\ell,5)}_{21} \\ w^{(\ell,1)}_{22} & w^{(\ell,22)}_{22} & w^{(\ell,3)}_{22} & w^{(\ell,4)}_{22} & w^{(\ell,5)}_{22} \end{bmatrix} \begin{bmatrix} a^{(1)}_{x,y} \\ a^{(2)}_{x,y} \\ a^{(3)}_{x,y} \\ a^{(4)}_{x,y} \\ a^{(5)}_{x,y} \end{bmatrix} + \begin{bmatrix} c^{(\ell)} \\ c^{(\ell)} \\ c^{(\ell)} \\ c^{(\ell)} \end{bmatrix} \right)$$

U-net Architecture

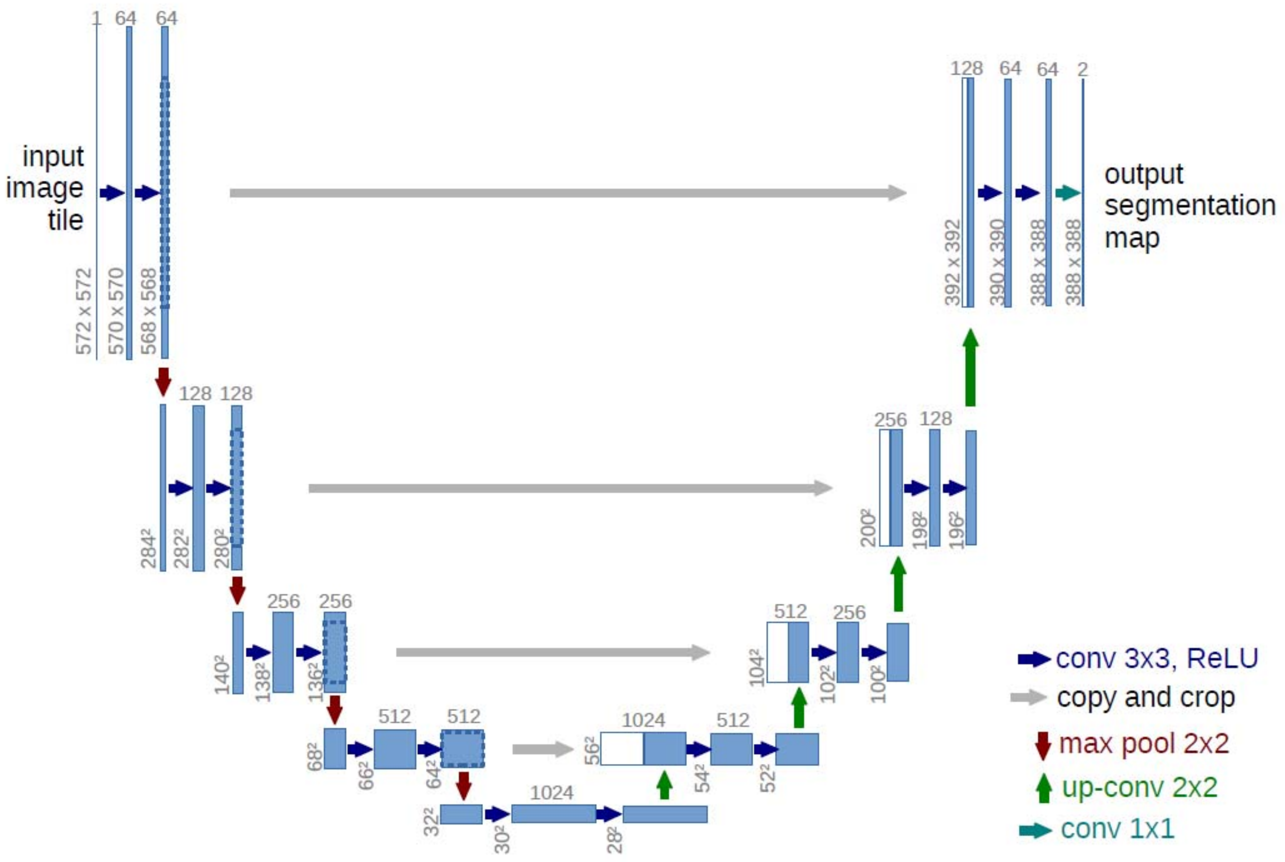

Figure 4: From *U-Net: Convolutional Networks for Biomedical Image Segmentation* by Olaf Ronneberger, Philipp Fischer, Thomas Brox. See https://arxiv.org/abs/1505.04597

- After every (down) convolution the pixels/neurons are passed through a ReLU.
- After every up convolution the pixels/neurons are passed through a ReLU.

Softmax and Maximum Likelihood

Suppose the problem is to determine for each pixel in an image whether it is part of a cancer cell or not. The output image has two channels $n \times n$ "pixels". That is, the output has shape $K \times n \times n$ for $K = 2$. In the first channel a pixel value is 1 if it is part of a cancer cell, otherwise it is zero. In the second channel a pixel has value 1 if is **not** part of a cancer cell, otherwise it is zero. More generally we can have K outputs.

Let $a(k, i, j)$ for $k = 0, 1, ..., K - 1$ and $i, j = 0, ..., n - 1$ denote the values in the UNET output. For each (i, j) compute the softmax of the values in the K channels, that is,

$$p_k(i,j) = \frac{e^{a(k,i,j)}}{\sum_{k=0}^{K-1} e^{a(k,i,j)}}.$$

With this softmax activiation we use the maximum log likelihood as the cost. That is, let $\ell \in \{0, 1, ..., K - 1\}$ be the correct classification of the pixel so the cost on that pixel is

$$-\log p_\ell(i,j).$$

The objective is to minimize the total maximum log likelihood cost given by

$$C = -\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} \log p_\ell(i,j).$$

The authors of [1] use a *weighted* maximum log likelihood given by

$$C = -\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} w(i,j) \log p_\ell(i,j)$$

with

$$w(i,j) = w_k(i,j) + w_0 e^{-\frac{(d_1(i,j)+d_2(i,j))^2}{2\sigma^2}}.$$

The weights $w_k(i, j)$ are used to balance the class frequencies. E.g., with K classes, suppose the image to be segmented contains m_1 pixels of class 1, m_2 pixels of class 2, ..., and m_K pixels of class K giving a total number of pixels equal to $m = m_1 + m_2 + \cdots + m_K$. Then, if the pixel at (i, j) is in class k, the weight is set as

$$w_k(i,j) = 1 - \frac{m_k}{m}.$$

$d_1(i, j)$ is the distance from a pixel at (i, j) of one class to the closest pixel of another class.
$d_2(i, j)$ is the distance from a pixel at (i, j) of one class to the second closest pixel of another class.
In [1] $w_0 = 10$ and $\sigma = 5$ (pixels).

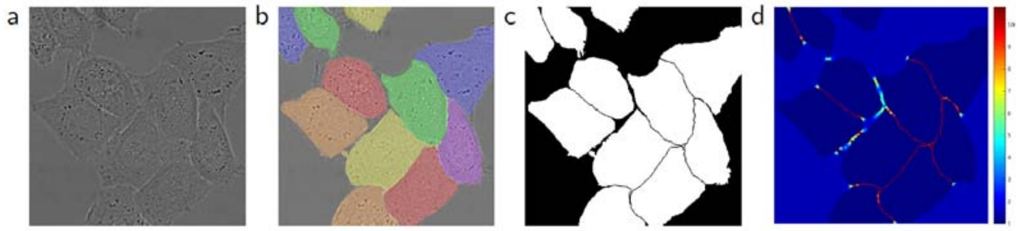

Fig. 3. HeLa cells on glass recorded with DIC (differential interference contrast) microscopy. (**a**) raw image. (**b**) overlay with ground truth segmentation. Different colors indicate different instances of the HeLa cells. (**c**) generated segmentation mask (white: foreground, black: background). (**d**) map with a pixel-wise loss weight to force the network to learn the border pixels.

Sigmoid and Cross Entropy

Let $a(k,i,j)$ for $k = 0, 1, ..., K-1$ and $i, j = 0, ..., n-1$ denote the values in the UNET output. These are passed through the sigmoid function $\sigma(\cdot)$ to obtain $\sigma(a(k,i,j))$.

For each $i, j = 0, ..., n-1$ let $g(k,i,j)$ for $k = 0, 1, ..., K-1$ be a one-hot vector giving the correct output "pixel".

The objective is to minimize the total cross entropy cost given by

$$C \triangleq -\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{K-1}\Big(g(k,i,j)\ln(\sigma(a^{(k)}(i,j))) \; + (1-g(k,i,j))\ln(1-\sigma(a^{(k)}(i,j)))\Big)$$

This cost is typically used when $K = 2$, that is, for binary classification.

Dice Cost [Wikipedia]

Let $a(k,i,j)$ for $k = 0, 1, ..., K-1$ and $i, j = 0, ..., n-1$ denote the values in the K maps/channels of the UNET output. For each (i,j) let

$$k_{\max}(i,j) \triangleq \arg\max_k \{a(k,i,j)\}.$$

Then define

$$p(k,i,j) \triangleq \begin{cases} 1, & k = k_{\max}(i,j) \\ 0, & \text{otherwise} \end{cases}$$

As in the case of the cross entropy cost, for each $i, j = 0, ..., n-1$ let $g(k,i,j)$ for $k = 0, 1, ..., K-1$ be a one-hot vector giving the correct output "pixel".

Note that for each (i,j) the quantity $\sum_{k=0}^{K-1} p(k,i,j)g(k,i,j)$ is either 1 or 0.

The **Dice** cost is given by

$$D \triangleq \frac{2\left(\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\left(\sum_{k=0}^{K-1} p(k,i,j)g(k,i,j)\right)\right)}{\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\left(\sum_{k=0}^{K-1} p^2(k,i,j) + \sum_{k=0}^{K-1} g^2(k,i,j)\right)}.$$

- $0 \le D \le 1$.
- $D = 1$ if and only if $p(k,i,j) = g(k,i,j)$ for $i, j = 0, ..., n-1$ and $k = 0, 1, ..., K-1$.

This has a simpler expression if $K = 2$ so there are two output channels. The Dice cost becomes

$$D \triangleq \frac{2\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} p(i,j)g(i,j)}{\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\left(p^2(i,j) \; + g^2(i,j)\right)}$$

where

$$p(i,j) \triangleq \begin{cases} 1, & \arg\max_{k\in\{0,1\}}\{a(k,i,j)\} = 1 \\ 0, & \arg\max_{k\in\{0,1\}}\{a(k,i,j)\} = 0 \end{cases}$$

and

$$g(i,j) \triangleq \begin{cases} 1, & \text{cancer cell} \\ 0, & \text{non cancer cell.} \end{cases}$$

References

[1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *arXiv: 1505.04597v1*, May 2015.