Paper review: U-Net: Convolutional Networks for Biomedical Image Segmentation
O. Ronneberger, P. Fischer, and T. Brox

Malcolm Davies

University of Houston

daviesm1@math.uh.edu

May 6, 2020

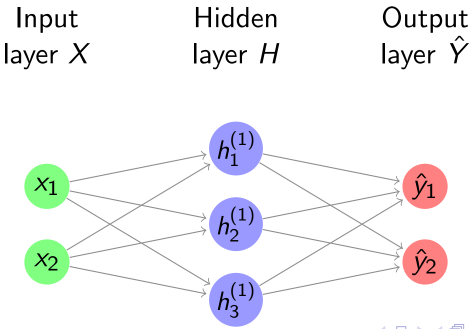
Overview

Abstract

- This paper introduces a new (as of May 2015) strategy for the design and training of neural networks.
- The Authors call their new method a **U-Net** after the U-like shape of the architecture when drawn out as a picture.
- Such a network can be trained end-to-end from very few images, and outperforms the prior best method on the ISBI challenge for segmentation of neuronal structures in electron microscopic stacks.
- The network is fast. Segmentation of a 512x512 image takes less than a second on a recent GPU.

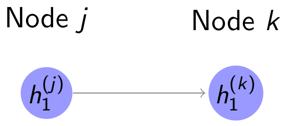
Basic Ideas: Neural Networks

- The goal of an artificial neural network (ANN) is to approximate some function f^*. In our case, this is a classification function $y = f^*(x)$, where $x \in \mathbb{R}^n$ is an input vector.
- The simplest architecture for an ANN is a directed graph composed of three layers of nodes, an **input layer**, a **hidden layer**, and an **output layer**, along with the connections between those nodes (the arrows in the graph).



Basic Ideas: Neural Networks

- The architecture of the network can be a lot more complicated than the above example with many more layers, numbers of nodes in layers, and types of connections.
- The network architecture will create a collection of formula relationships between the nodes defined by by connecting functions $f_{(j,k)}(x, \theta)$ for nodes (j, k) with parameters θ.



$$f_{(j,k)}(*, \theta) : h_1^{(j)} \longmapsto h_1^{(k)}$$

Basic Ideas: Neural Networks

- So the specific architecture of an ANN defines a mapping $\hat{y} = f(x; \theta)$ and then learns the value of the parameters θ that result in the best function approximation to $f^*(x)$ given that graph and those connecting functions.
- Often there will also be a **response function** g, either at the end of the network, or between layers, to further modify the outputs. Common examples are the **RELU function** $g(z) = \max(0, z)$, or the **Sigmoid function** $\sigma(z) = 1/(1 + e^{-z})$.

Introduction: Convolutional networks and biomedical imaging

- **Convolutional Neural Networks (CNNs)** are commonly used for the analysis of 2D (or 3D) images having either gray level pixel intensities, or multi-channels pixel intensities, such as Red/Blue/Green.

Basic Ideas: Convolution Operations in ANNs

- A CNN is an ANN where some of the primitive functions $f_{(j,k)}(x, \theta)$ act on their layers as a convolution operation (often called a **convolution filter**).
- In CNNs, convolution operations act on whole sections of an image file, so they are usually not single primitive functions $f_{(j,k)}(x, \theta)$, but rather are made up of such functions working together.
- Recall the definition of a **convolution** in the context of functions: Given two functions k and f, their convolution is defined

$$(k * f)(x) := \int_{\infty}^{\infty} k(y) f(x - y)\, dy.$$

The insight here is that the two functions are first offset by y, then multiplied together, and then their multiple is integrated over all y in their domain.

Basic Ideas: Linear Convolution Operations in ANNs

- Convolutions over an image file X are similar, except that the domain is the 2D discreet coordinate space, so the integral is actually a summation.
- Let V be the square "window" of integer coordinates (m, n) such that $|m| \leq S$ and $|n| \leq S$ for some integer S.
- Fix a "function" of arbitrary real numbers $K(m, n)$ defined for all (m, n) in V. The $(2S + 1)x(2S + 1)$ matrix K defines the "**kernel**" of a linear convolution filter with support included in V.
- This convolution filter acts linearly on an image X to generate another image G denoted $G = K * X$, computed as follows for each pixel (i, j)

$$G(i,j) = K * X(i,j) = \sum_{(m,n)\in V} K(m,n)X(i-m, j-n)$$

Basic Ideas: Convolution Layers and Non-Linear Convolution Operations in ANNs

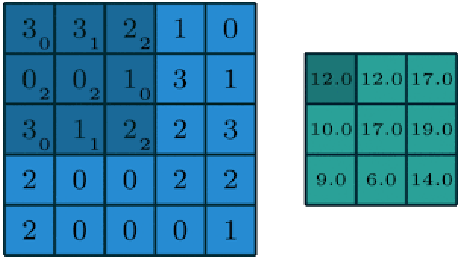
- Fix a kernel K with support in a square window V, and fix a single threshold parameter "b". Then (K, V, b) defines a **convolution layer** of size NxN, with one "neuron" (or "node") NOD_{ij} placed at each pixel position (i, j).
- When the current input is an NxN image X, the state $Y(i, j)$ of neuron NOD_{ij} is then computed by

$$Y(i,j) = g(b + K * X(i,j))$$

- Here g is a response function. When this is non-linear, it makes the convolution layer into a non-linear convolution layer.

The five operations: 3x3 Convolution with RELU

- The **3x3 convolution** operation with **RELU** in this paper is the standard unpaded version.
- A 3×3 kernel K (in grey) ranges over all contained intersections with an image X (in blue), i.e., those completely contained within the image.



The five operations: 3x3 Convolution with RELU

- The $3x3$ kernel operation reduces the size of the image by 2 in both dimensions.
- The RELU function then takes the outputted matrix and retains only the positive part.

The five operations: Copying and cropping

- An important aspect of the U-Net's architecture is the large number of feature channels in the up-sampling part. As a consequence, the expansive path is more or less symmetric to the contracting path, and yields the u-shaped architecture.
- The key insight here is that deep features can be obtained when going deeper, but spatial location information is also lost when going deeper, so output from shallower layers have more local information. We want to combine both to enhance the results.
- We do this through the **copying and cropping** operation. Every step in the expansive (upwards) path is concatenated with the correspondingly cropped feature map from the contracting (downwards) path. The cropping is necessary due to the loss of border pixels in every convolution.

The five operations: Copying and cropping

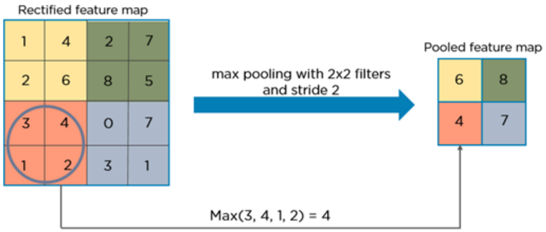
- There is a detail here that I do not fully understand concerning how the the concatenated data are merged. Figure 1 shows the concatenated data as a white and blue box stuck together, followed by a 3x3 conv operation, followed by a blue box of the same spatial dimensions but half the feature dimensions. However, it gives no details.

Speculation: Two possible answers

- The layers are somehow being gobbled up and merged by the following 3x3 convolution operation, similarly to how the 1x1 convolution operation works at the last step.
- The layers are simply being merged by element-wise addition. (This was the explanation given by an outside source, but the paper is too vague to say for sure.)

The five operations: 2x2 Max-Pooling Down-Sampling

- The **max pooling** operation in this paper partitions an input image X up into 2×2 squares, then takes the maximal element from each square and arranges them in a new image Y that is half the size in both dimensions.
- It can be thought of as a form of data compression that "picks the local highlights".



The five operations: 2x2 Max-Pooling Down-Sampling

- The **max pooling** operation in this paper involves increasing the number of feature channels.
- Each layer of data in a CNN is actually a three-dimensional array of size $h \times w \times d$, where h and w are spatial dimensions, and d is the feature or channel dimension.

Author explanation:

"At each down-sampling step we double the number of feature channels."

The five operations: 2x2 Max-Pooling Down-Sampling

- The exact mechanism for the max-pool feature creation currently eludes me. The paper itself does not give much detail here, though the authors do "provide the full Caffe[6]-based implementation and the trained networks".

Speculation:

- My guess is that the new features are "abstract" in that they are an extra avenue for the learning algorithm to encode information, but they wont be as "straightforward" as color levels.
- These new features correspond to using additional convolution Kernels.

The five operations: Convolution and 2x2 Up-Sampling

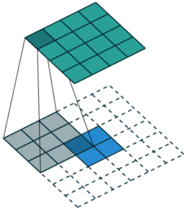


Figure: Action of the convolution operation over an image.

- Up-sampling is a method that gets the output size larger.
- This can be a convolution operation that allows the kernel K (in grey) to range over all intersections with an image X (in blue), not just those contained within the image.
- This is made concrete by expanding the image X to X' (all the white squares) by some method.

The five operations: Convolution and 2x2 Up-Sampling

- In this paper, the expansion of the image X to X' is done by mirroring the image where necessary.

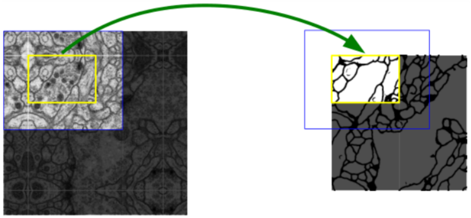


Fig. 2. Overlap-tile strategy for seamless segmentation of arbitrary large images (here segmentation of neuronal structures in EM stacks). Prediction of the segmentation in the yellow area, requires image data within the blue area as input. Missing input data is extrapolated by mirroring

The five operations: Convolution and 2x2 Up-Sampling

- Here again, the exact mechanism for this eludes me. The paper does not give much detail here.

Author explanation:

"Every step in the expansive path consists of an up-sampling of the feature map followed by a 2x2 convolution (“up-convolution”) that halves the number of feature channels”

The five operations: Basic 1x1 Convolution

- At the final layer a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes.
- A 1×1 convolution maps an input pixel, along with all it's channels (features), to an output pixel, not looking at anything around itself.

Network Architecture

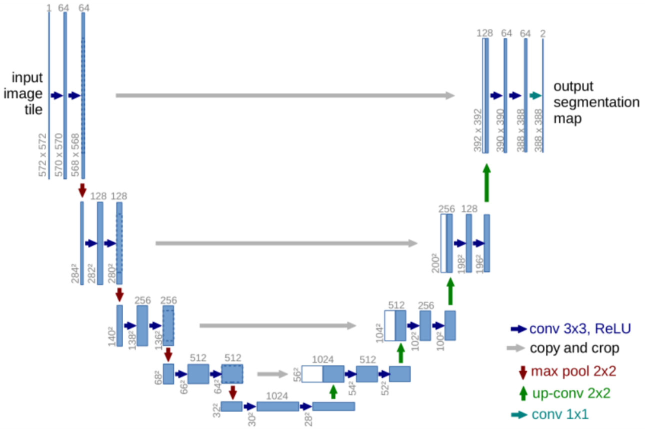


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Training and Data Augmentation

- Data augmentation is essential to teach the network the desired invariance and robustness properties, when only few training samples are available.
- "We apply elastic deformations to the available training images. This allows the network to learn invariance to such deformations, without the need to see these transformations in the annotated image corpus. This is particularly important in biomedical segmentation, since deformation used to be the most common variation in tissue and realistic deformations can be simulated efficiently."
- Additionally, "we primarily need shift and rotation invariance as well as robustness to deformations and gray value variations".

Experiments

The authors apply the u-net to three different segmentation tasks:

1. The segmentation of neuronal structures in electron microscopic recordings provided by the ISBI 2012 EM segmentation challenge.
2. Two sets of cell segmentation task in light microscopic images provided by the 2014 and 2015 ISBI cell tracking challenge.

Experiments: Results

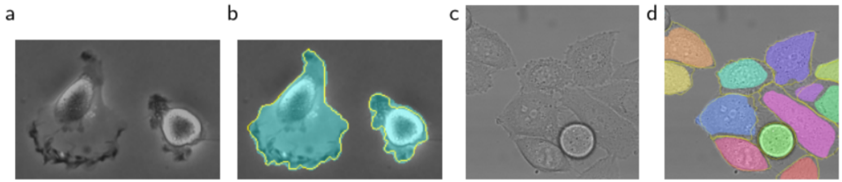


Fig. 4. Result on the ISBI cell tracking challenge. (**a**) part of an input image of the "PhC-U373" data set. (**b**) Segmentation result (cyan mask) with manual ground truth (yellow border) (**c**) input image of the "DIC-HeLa" data set. (**d**) Segmentation result (random colored masks) with manual ground truth (yellow border).

Table 2. Segmentation results (IOU) on the ISBI cell tracking challenge 2015.

Name	PhC-U373	DIC-HeLa
IMCB-SG (2014)	0.2669	0.2935
KTH-SE (2014)	0.7953	0.4607
HOUS-US (2014)	0.5323	-
second-best 2015	0.83	0.46
u-net (2015)	**0.9203**	**0.7756**

Conclusion

- The u-net architecture achieves very good performance on very different biomedical segmentation applications.
- Using data augmentation with elastic deformations, it needs very few annotated images, and has a very reasonable training time of only 10 hours on a NVidia Titan GPU (6 GB).

The End