FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Cell Segmentation with the U-Net Convolutional Network

Manuel Danner

Computer Vision Lab
Institute of Visual Computing & Human-Centered Technology
TU Wien

November 12, 2018

Supervisor: Roman Pflugfelder

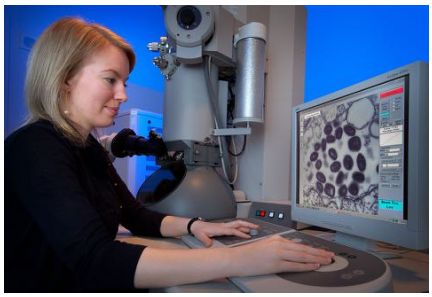
Contents

Abstract

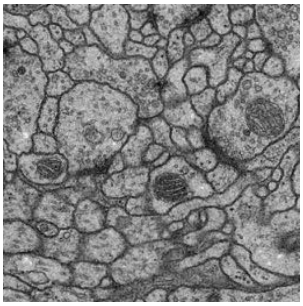
In biomedical research, electron microscopy images are required to investigate the detailed structure of tissues, cells, organelles and macromolecular complexes. Modern electron microscopy is used to obtain high resolution images of biological and non-biological specimens. Key regions need to be identified, to support clinical analysis and medical interventions. This process is sophisticated and tedious for human experts. To reduce the required amount of human labor by semiautomated systems, state-of-the-art methods use supervised learning with deep neural networks. In this work, the convolutional neural network u-net is reimplemented in PyTorch. Human expert labeling, for cell segmentation, is time consuming and demanding. To overcome the assumption of extensive manual labeling for training, we propose a new data augmentation method. The ISBI 2012 challenge is used to evaluate and compare our novel approach with two different elastic transformation methods. Furthermore, the use of different loss functions for the proposed network is outlined. Experiments show the benefit of the used data augmentations. Our proposed network achieves rank 24 of 143 participants, while the original u-net ranks at place 44. The difference at the foreground-restricted rand score value, between our proposed network and the first place at the ISBI 2012 challenge, is less than a value of 0.011.

1 Introduction

Modern electron microscopy (EM) is used to obtain high resolution images of biological and non-biological specimens. In biomedical research, EM images are required to investigate the detailed structure of tissues, cells, organelles and macromolecular complexes. On this account, large amounts of high resolution images from biological and non-biological specimens exist. Figure 1 shows a Disease Control and Prevention intern working with a transmission electron microscopes. The identification of key regions, such as organelles, tissues or cells, can support clinical analysis and medical interventions. This process is time consuming and demanding for human experts.



(a) Electron microscope



(b) EM image

Figure 1: The left image depicts a Disease Control and Prevention intern, working with a transmission electron microscope. The right image is from the ISBI 2012 challenge dataset [1].

Conventional image processing methods rely on hand-crafted models to extract and detect prominent features [2]. Good features contain discriminating information, which can distinguish between different objects in the image. No method produces efficient solutions on all types of medical images [3]. For cell segmentation, main characteristics of cells need to be handled properly. Cells touch and occlude each other, heaps of cells occur and flowing transitions between cell states and cell functions exist. Furthermore, EM images have low contrast and contain noise. To reduce the amount of human labor needed by semiautomated systems, state-of-the-art methods for cell segmentation use supervised learning with deep convolutional neural networks (CNNs) [1]. A condition for CNNs to generate accurate results are extensive labeled image datasets [4]. For example, the ImageNet dataset [5] contains roughly 1.2 million training images, 50.000 validation images and 150.000 testing images.

Even though a lot of unlabeled EM images for cells exist, deep neural networks need thousands of annotated training samples to generate acceptable results. It is not feasible to label the amount of EM images needed for neural networks by hand. Fortunately, data augmentation can be applied to small datasets labeled by hand, as cells and cell structures show limited variation of appearance.
State of the art in segmentation of cells is the convolutional network u-net provided by Ronneberger et al. [6].

Ronneberger et al. created a fully convolutional network to detect and segment

cells with the Caffe deep learning framework [7]. Excessive data augmentation is needed to train their network on the few annotated training samples available. First, a contracting path is used to capture context. As a next step, the path is expanded again to enable precise localization. Their network can be trained on few images, and the segmentation of a 512×512 image takes less than a second on a modern GPU.

In 2012, the former best result of the ISBI 2012 challenge [1] by Ciresan et al. [8] trained a deep neural network (DNN) as a pixel classifier. Each pixel label is predicted by a square window centered around the raw pixel values. Mild smoothing and thresholding is applied to the pixel probabilities. To create 3 million training examples, all membrane pixels are used as positive examples, while the corresponding amount of non-membrane pixels are used as negative examples. In their work, four networks with different depth and window size are trained. The largest used architecture contains 11 Layers. 1 input layer, 4 convolutional layers each followed by a max pooling layer, and 2 fully connected layers in the end.

In this work, the original u-net architecture is ported to a fast Python framework with strong GPU acceleration, called PyTorch 0.4. To overcome the assumption of extensive manual labeling for training, we propose a new data augmentation method. This approach is compared with two different elastic transformation methods. All approaches are implemented and evaluated on the ISBI 2012 challenge [1]. The novel approach occupies place 24, while the compared methods achieve place 41 and 67 at the ISBI 2012 challenge [1]. The Kullback-Leibler Divergence and Cross Entropy are both analyzed for the use as loss function. Both functions can be used equally, if a gradient method is used for network optimization. The goal of the practicum is to get a deeper understanding of the u-net network used by Ronneberger et al. [6] and improve the results on the ISBI 2012 challenge [1].

The report is organized as follows: preliminaries are explained in Section 2. All necessary layers to construct the u-net are explained in detail. The fundamental u-net architecture is described in Section 3. Section 4 describes the first challenge on 2D segmentation of neuronal processes in EM images (ISBI 2012 challenge) [1], the used evaluation metric and the achieved results at the challenge. In Section 5 a comparison of the proposed networks is given and a conclusion is drawn.

2 Fundamentals

CNNs are a subtype of neural networks. In at least one of their layers, a mathematical operation called convolution is used. A convolution is a linear operation. CNNs work best, when there is a temporal or spatial relationship between the data points. Distinctive features can be detected without human supervision. Section 2.1 covers technical terms, which are often used. Section 2.2 explains different layers and Section 2.3 the training of CNNs by describing a loss function.

2.1 Preliminary

First, Basic preprocessing data augmentation methods are described. In the following sections, basic terms to understand neural networks are explained in detail. Basic knowledge about image filtering is needed to develop and build such networks.

2.1.1 Basic Data Augmentation

Neural networks need huge labeled datasets, to improve their results. In biomedical image datasets, annotated images are rare. Only few labeled data exists. Instead of collecting additional annotated data at a high cost, the network model can be exposed to synthetic variations of images without any deductions, depending on the data. Additionally, overfitting is reduced and the networks ability to generalize is improved.
All augmentation methods are applied to the test image in Figure 2 for comprehensibility.

Figure 2: Test image of an microscope, which is used to show the effect of each data augmentation method.

As a preprocessing step, rotation can be applied to the training images of a network. To avoid missing image information, due to rotation, the image is reflected at the borders. This effect can be seen at the bottom left of Figure 3b and at the top right corner of Figure 3d. Depending on the characteristics of the data, a reflection at the borders can introduce new errors.

To expand the training set further, horizontal and vertical flipping can be used. This augmentation generates mirrored images at the given axis. Figure 4 shows the

(a) Original (b) 20 ° (c) 30 ° (d) 40 °

Figure 3: 10 ° rotations on the test image. For every input image, 35 additional rotated versions are created.

two different directions of flipping. The necessity of flipping depends on the used training images.

(a) Original (b) Vertical flip (c) Horizontal flip

Figure 4: Horizontal and vertical flipping applied to the test image.

2.1.2 Convolution

An input image can be represented in matrix form, where each element in the matrix corresponds to the intensity or color of a pixel. A convolution applies a predefined filter matrix, which is called kernel, at each pixel of the input image. The filter matrix size and depth can vary, depending on the underlying problem. The values of the filter are called weights. Figure 5 shows a simple convolution with filter size 3×3. Neurons are all elements in a neural network of the form: $\sigma(\sum_j w_j x_j + b)$. The weight w_j is the value at filter-matrix index j, x_j is the corresponding input element to the filter matrix and b is the bias term. The function $\sigma(.)$ represents an activation function. These functions are used to shift the value of the layer element into a predefined range. In Section 2.2.5 and Section 2.2.6, two activation functions (rectified linear unit and softmax) are explained in detail. Furthermore, input and output elements are also called neurons. Because multiple values are convolved into the output neuron, this neuron contains information about the receptive field. Therefore, the new calculated values of the activation map do not correspond to intensity or color values anymore, but coarser information, like edges or curves, are contained in this value. For example, if an edge exists in the input image, the neuron corresponding to the edge in the activation map is activated.

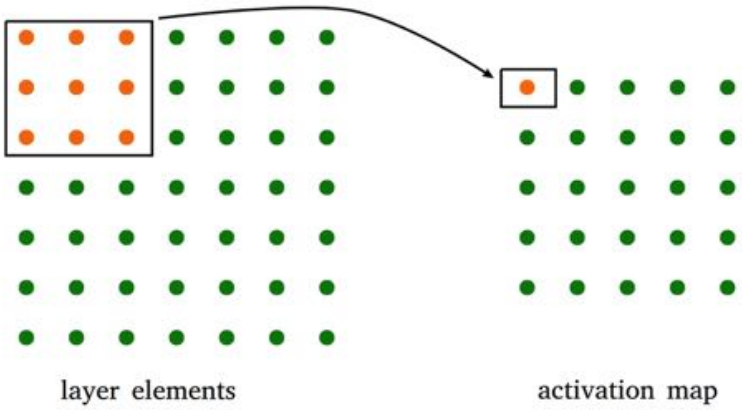


Figure 5: The computation of a convolution on an input image with stride 1 and no padding. A 3×3 filter is applied on a 7×7 image. The output is an activation map of size 5×5. Orange layer elements are the pixels which are currently used for calculations. The green layer elements are visited later. The receptive field of the orange elements in the activation map is shown as black rectangle in the left image. In this example, the receptive field is composed of 9 layer elements. The filter is slided over the whole input image to create the activation map.

2.1.3 Sparse Connectivity

As stated by Goodfellow et al. [9], traditional neural network layers use a dense connectivity. Every input neuron is connected to each following output neuron, therefore each output is affected by every input neuron. In contrast, CNNs have a sparse connectivity. On this account, the filter size is smaller than the input image size. A grayscale image of size 512×512, has 262.144 weights, if a dense connectivity is used, for each activation map of size 1×1. If a sparse connectivity is applied, a filter size of 3×3, uses only 9 weights for the resulting activation map of size 510×510, when no padding is applied. Padding, which is explained in detail in Section 2.1.4, is used to generate additional predefined space around elements. Small filters are used to decrease the amount of (floating-point) operations. For example, a 2×1 filter uses $511 \times 512 \times 3 = 784.896$ operations. (For each output pixel, two multiplications and one addition is needed).The same transformation with a dense matrix multiplication needs $512 \times 512 \times 511 \times 512$ operations. As a result, far less operations are needed to create the same activation map in a sparse connectivity system.

2.1.4 Padding

When a convolution is applied over an image, pixel information at the border is missing, depending on the filter size. Therefore, the output image size is reduced, by $\lfloor filtersize \div 2 \rfloor \times 2$ at each axis. A method to handle border pixels with estimated values is padding. There are different kinds of padding. For example, zero padding, replicate padding or mirror padding.
All examples are explained for a 2-D image. It can be extended easily to multiple

dimensions. Zero padding adds rows and columns of zero values at the four borders of the matrix. Instead of zeros, replicate padding adds a copy of the adjoining borders to be able to calculate the convolution for each neuron. Mirror padding reflects the pixels at the edge of the border. All of these methods introduce an error to the resulting convolution, because the additional information about the outside of the border is just assumed. A benefit of padding is the consistent matrix size after the convolution. Figure 6 shows an example of padding applied on an one-dimensional array.
Another method is to use no padding. As a result, no border errors are introduced, but the output matrix shrinks in size, depending on the filter size. In Figure 5, no padding is used, therefore the input matrix of size 7×7 gets reduced to an output matrix of size 5×5.

0 0 0 | 1 2 3 4 5 6 7 | 0 0 0

(a) zero padding

1 1 1 | 1 2 3 4 5 6 7 | 7 7 7

(b) replicate padding

4 3 2 | 1 2 3 4 5 6 7 | 6 5 4

(c) mirror padding

Figure 6: Three different types of padding shown on an one-dimensional array. Zero padding, replicate padding and mirror padding.

2.1.5 Stride

By default, the filter matrix is convolved over every layer element of the input matrix. With a custom set stride, one can skip a set amount of layer elements. For example, a stride of 1 does not skip a layer element. The receptive field is shifted over every pixel. A stride of 2 skips every second pixel in both directions. A stride of 3 skips 2 pixels at a time in both directions. It is important to note, that the row and column size of the output matrix has to be an integer value. No fractions are allowed. A benefit of using a stride is to remove the overlap of the receptive fields. The receptive fields overlap less, the higher the stride parameter is set. However, the resolution of the matrix gets reduced. In Figure 5, a stride of 1 is used, with no padding. In Figure 7, stride is set to two. Instead of shifting the receptive field over every pixel, only every second pixel is used.

2.1.6 Dilation

Dilation is similar to stride, but instead of skipping pixels from the input matrix, the filter kernel introduces spaces between its filter values. Figure 8 shows an example of a matrix with no padding and no stride, but with a dilation set to 2. A 3×3 filter is used and only the orange middle point and blue side values are used for the

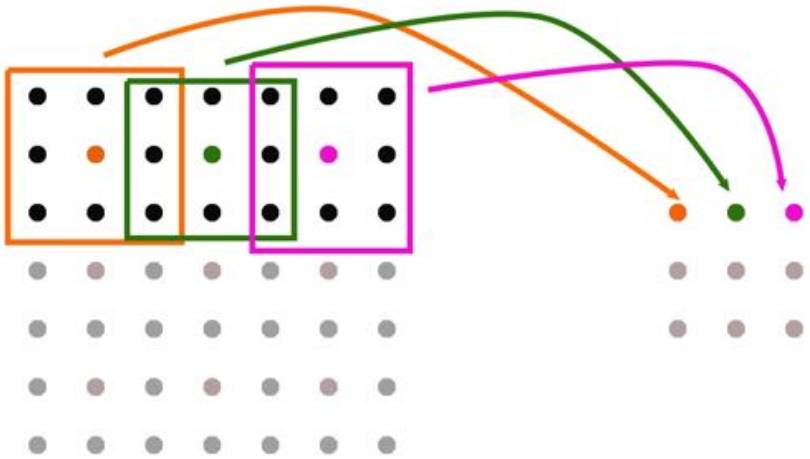


Figure 7: The computation of a convolution, with stride 2 and no padding, on an input image. A 3×3 filter is applied on a 7×7 matrix. The output is an activation map of size 3×3. The first row from the activation map on the right matrix, contains the corresponding convolutions for the orange, green and pink receptive fields from the left matrix.

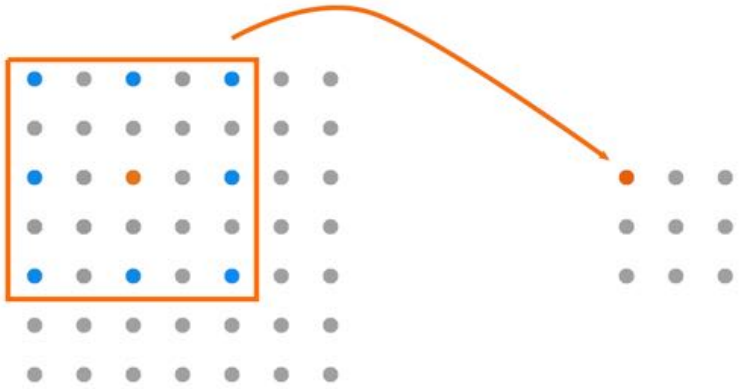
filter calculation.
Unlike the matrix reduction received by setting a stride, dilation does not reduce the output size if padding is used. Furthermore, dilation filters support exponentially expanding receptive fields, where no resolution or coverage is lost [10].

2.2 Layer Types

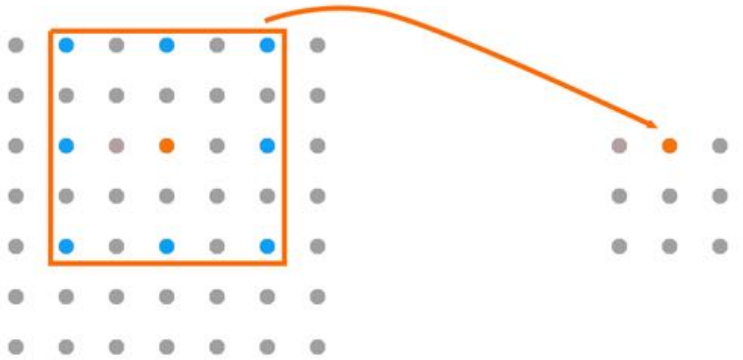
CNNs can be separated into three parts. Specifically, an input layer, one or more hidden layers and an output layer. The input layer is the first layer in the network. It is used to pass the data to the first hidden layer. Hidden layers transform the given input data with different methods. In the next section, six different types of hidden layers are explained in detail. For the u-net network, convolutional, pooling, transposed convolution, ReLU and dropout layers are used as hidden layers. To get a mapping to classes, a softmax layer is used. For simplification, in the next sections, the input layer is assumed to be a grayscale image. Table 1 lists coarse mathematical definitions of the used layers.

2.2.1 Convolutional Layer

The name CNN derives from the mathematical term of a convolution. Table 1 shows the coarse definition of a convolution. The convolution can be seen as a weighted average, where the weighting is implied by a different function. For CNNs, each convolutional layer needs user defined parameters. The channels of the input image, the wanted number of output channels, the filter size, the stride value, the padding type, the dilation value and whether a bias term should be added or not.
First, a convolutional layer gets a matrix with a predefined size $n \times m \times d$ as input. The number of rows is called n. m corresponds to the number of columns and d is



(a) dilation filter over the first pixel



(b) dilation filter over the second pixel

Figure 8: Dilation is applied on the image matrix with a value of 2 and a filter size of 3×3. No padding and no stride is added. The filter uses only the orange and blue pixels in the left matrix to calculate the orange value in the right matrix. One can imagine the dilation as a sparse filter of size 5×5, where all values are zero, expect for the blue and orange pixels.

the amount of channels (depth of the matrix). A filter of size $k_1 \times k_2 \times d$ is convolved with the input matrix, with $k_1 \ll n$ and $k_2 \ll m$. One filter creates one feature map. In the convolutional layer, the amount of feature maps e can be seen as the number of different filters which are needed. Therefore, a total of $k_1 \times k_2 \times d \times e$ weights are used, without the bias. The bias would add e weights [11].

A special case is the convolution with a $1 \times 1 \times k$ filter. This convolution can be used to reduce the depth of a matrix, and compress multiple feature maps into one.

A convolution is a linear function. Therefore, to introduce non-linearity to a CNN, non-linear activation functions σ, like the rectified linear unit (ReLU), or the classical sigmoid function $\sigma = \frac{e^x}{1+e^{-x}}$, are needed. Without these activation functions, multiple convolutions can be reduced to a single convolutional layer.

f_{conv}	$(f * g)(x) = \int f(\tau) \cdot g(x - \tau)$
f_{deconv}	$f_{deconv} = f_{conv}$
$f_{maxpool}$	$max(x_1, ..., x_n)$
$f_{dropout}$	$x_j \cdot P_j(x_j) \mid P_j \in [0, 1]$
f_{ReLU}	$max(0, x)$
$f_{softmax}$	$\frac{\exp(x_i)}{\sum_{n=0}^{k} \exp(x_n)}$, for $i = 0, 1, 2, ..., k$

Table 1: Coarse mathematical interpretation of different layer types. The convolution can be seen as a weighted average, where the weighting is implied by g. When max pooling is applied, only the highest value is utilized in the given window. A deconvolution can be described as a fully padded convolution with unit strides. The dropout function multiplies each layer element with zero or one, with a predefined probability. ReLU and softmax functions are used to shift a layer element into a given range.

2.2.2 Pooling Layer

This layer is used to introduce an invariance to object motion and deformations. Because of the decrease of the matrix size, the computational complexity is reduced. There are different types of pooling layers, which can be used to accomplish a matrix size reduction. For example average pooling or max pooling. Figure 9 shows max pooling and average pooling applied on the same input matrix. In this layer, the depth of the output matrix is equal to the input matrix. Only the width and height of the input matrix is reduced to create the output matrix. The position of the high activation is not important, as long as the high activation is inside the pooling region. It is important to note, that no weights are added in this layer.

2.2.3 Transposed Convolution Layer

A transposed convolution can be seen as a reversed convolution layer. A different name is fractionally strided convolution or deconvolution as mentioned by Dumoulin et al. [11]. Therefore, it can be used to up-sample an input. In computer vision, there are various techniques to up-sample an image. For example, nearest neighbor interpolation, bi-linear interpolation or bi-cubic interpolation [12]. With these interpolation techniques, the network would have no learnable up-sampling parameters. To let the network learn weights for up-sampling, transposed convolutions are used. The transposed convolution layer takes the same parameters as the convolution layer, which is defined in Section 2.2.1. A transposed convolution can always be emulated with a direct convolution, by adding rows and columns of zeros to the input. More efficient implementations of the transposed convolution exist, where the operation can be seen as a gradient of a convolution with respect to the input, as mentioned by Dumoulin et al. [11]. Figure 10 shows a convolution followed by a transposed convolution, which is used to up-sample the input. Zero padding is added at the top blue 2×2 matrix, to create the 4×4 green matrix. It is important to note, that the transposed convolution is not an inverse convolution operation, as the initial input values can not be recreated. Only the shape of the initial feature map is restored.

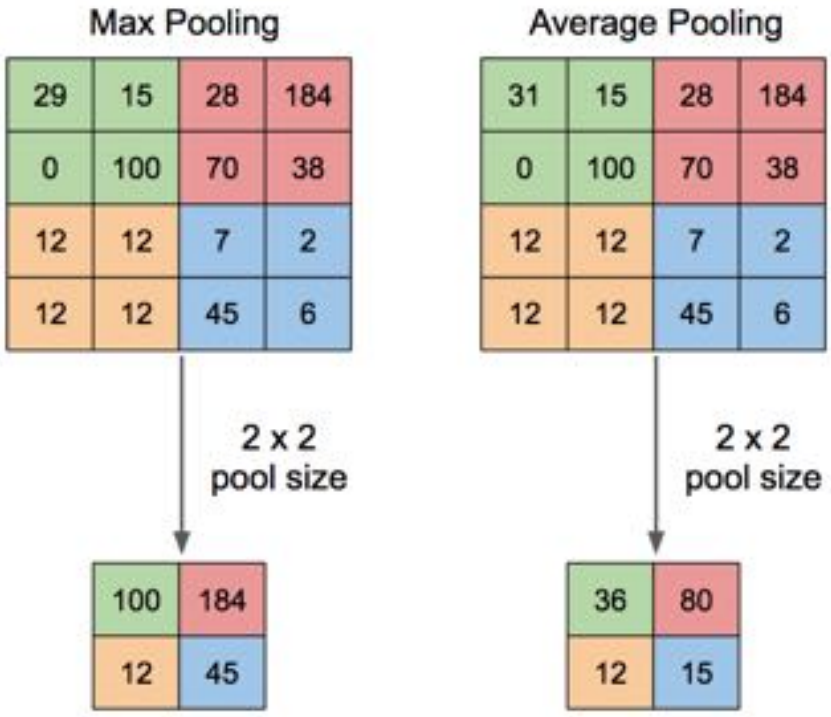


Figure 9: Max pooling with a 2×2 pool size is used on the left matrix. Only the highest value in each 2×2 block is propagated to the output. In comparison, average pooling with a 2×2 pool size applied on the same matrix. The average value of each 2×2 block is propagated. Both pooling types create different results.

2.2.4 Dropout Layer

This layer is only used during training. With a set probability p, elements of the input are replaced with zero. On every forward call, the chosen elements are randomized again. This method helps to reduce overfitting and prevents co-adaptions, as mentioned by Hinton et al. [13].

2.2.5 ReLU Layer

A rectified linear unit (ReLU) is an activation function. It can be applied after a convolution and prevents negative values in the feature map. Equation 1 shows the mathematical term of the activation function. x is the matrix result from a given layer, for example a convolution layer. Negative values are mapped to zero, while positive values do not change.

$$\sigma_{ReLU(x)} = max(0, x) \tag{1}$$

2.2.6 Softmax Layer

The softmax layer is used to generate a probability matrix for the output layer. Every layer element is assigned the probability to belong to each target class, therefore it is scaled between 0 and 1. For example, foreground and background pixel

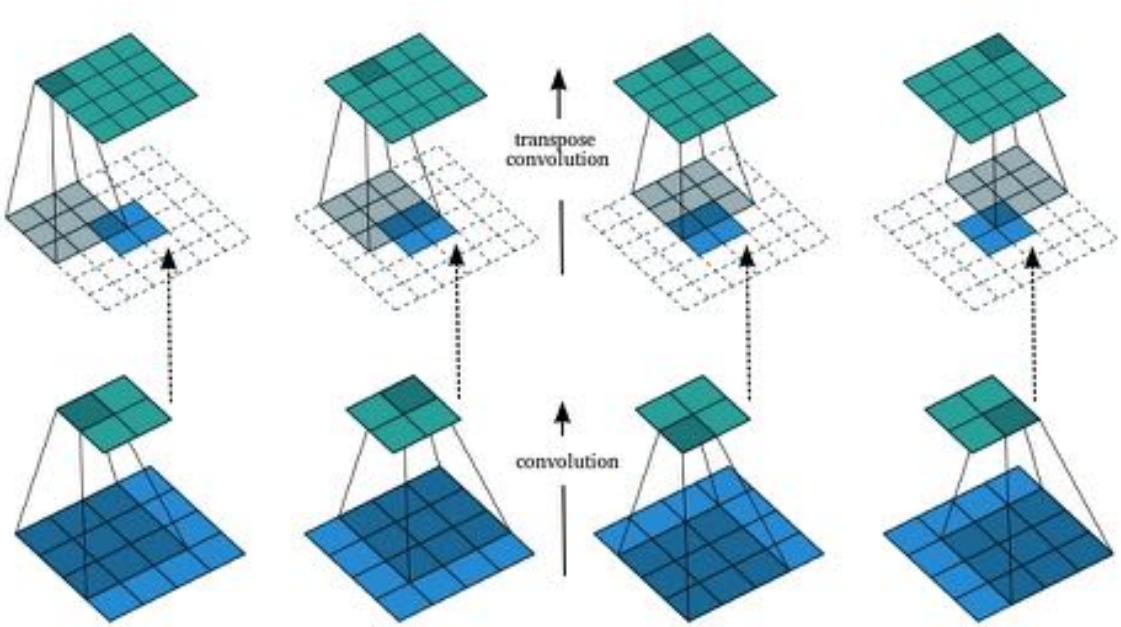


Figure 10: A 3×3 filter is convolved over the bottom blue 4×4 matrix, to create a feature map of size 2×2. To up-sample the resulting 2×2 matrix, a 3×3 filter is convolved over the top blue 2×2 matrix, which is padded by a 2×2 zero border and a stride of 1. This transpose convolution creates a 4×4 feature map. The transposed convolution is not an inverse convolution operation, as the initial input values can not be recreated. Image reproduced from [11].

affiliation can be seen as 2 different classes in image segmentation. The sum of the class probabilities for each layer element adds up to 1. Equation 2 shows the corresponding softmax function. k is the amount of classes -1. x_i is a layer element, belonging to class i.

$$\sigma_{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_{n=0}^{k} \exp(x_n)}, \text{for } i = 0, 1, 2, ..., k \tag{2}$$

2.3 Loss Function

As common in supervised machine learning, it is necessary to compare the actual output with the predicted output [9]. A loss function is needed to measure an error between the predicted value and the actual label. In Equation 3, the predicted label of an input image x_i, with the according weights W, is compared with the ground truth label y_i. To calculate the average loss over all examples, the sum of the losses for each example is divided by the amount of examples N. Therefore, the performance of the network can be connected to the loss function. For accurate predictions, the calculated error needs to be minimized.

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i) \tag{3}$$

In neural networks, backpropagation is used to minimize the calculated error. The weights of each layer are modified with an optimization function. Figure 11 shows the correlation of the loss function and the optimization. Briefly summarized, the gradient of the loss function, with respect to the weights and biases of the layers, is calculated. Until a minima of the loss function is reached, the weights are modified in the opposite direction of the gradient. In this work, the stochastic gradient descent is used as an optimization function. For the interested reader, more information on the topic of backpropagation and optimization can be found in the book Deep Learning by Goodfellow et al. [9].

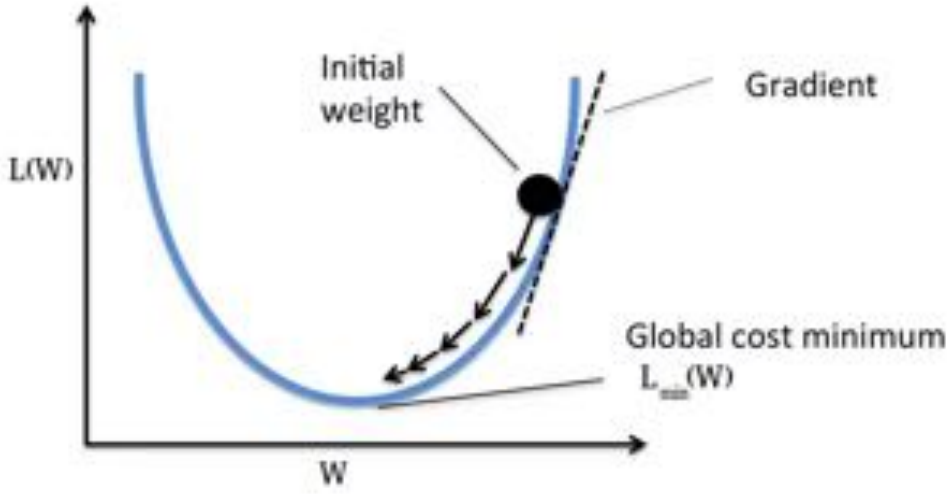


Figure 11: The gradient of a given loss function L with weights W is calculated. By modifying the weights in the opposite direction of the calculated gradient, a minimum can be found.

2.3.1 Entropy

The entropy H is defined as the sum of the probability P of each label i, times the information gain $-\log_2(P(i))$ of the label. Equation 4 shows the mathematical term. The entropy measures the average information content one can expect, if one of the labels occur. For example, if a coin is tossed and both head and tail have a probability of 50%, an entropy of 1 will occur. 1 bit is needed to transport the information. If both sides are head, the entropy will be 0, because no new information is needed to predict the outcome. It will always be head.

$$H(P) = -\sum_i P(i) \log_2(P(i)) \tag{4}$$

2.3.2 Kullback-Leibler Divergence

The Kullback-Leibler Divergence (also called relative entropy), measures how big the variance between two probability distributions is. Equation 5 shows how the Kullback-Leibler Divergence (KLD) is calculated. The difference between KLD and the entropy, lies in the additional term of $Q(i)$ in the logarithm. If both distributions are identical, the logarithm term evaluates to zero, hence the function result is zero.

$$D_{KL}(P \parallel Q) = \sum_i P(i) \log_2(\frac{P(i)}{Q(i)}) \tag{5}$$

Instead of measuring the average information content one can expect, the KLD measures the additional content which is needed, if the probability distribution Q is used. If both distributions are equal, the function can be zero. Therefore, if the predicted output of a network is seen as a probability, one can use the KLD as a loss function for the neural network.

2.3.3 Cross Entropy

In the implementation of the u-net architecture by Ronneberger et al. the Cross Entropy Loss is used [6]. Each predicted label is seen as a probability to belong to the right class. Therefore, the neural network creates a probability distribution. The other probability distribution is defined by the ground truth labels. The cross entropy measures the average information content one can expect, if the probability distribution Q is used, instead of P. The entropy and the KLD can be used to define the cross entropy, shown in Equation 6. The entropy of the ground truth data is added to the KLD.

$$H(P, Q) = H(P) + D_{KL}(P \parallel Q) \tag{6}$$

Inserting Equation 4 and Equation 5 into Equation 6 results in Equation 7.

$$H(P, Q) = \sum_i P(i) \log_2(\frac{P(i)}{Q(i)}) - \sum_i P(i) \log_2(P(i)) \tag{7}$$

With the logarithm quotient rule, Equation 7 can be split into three sums, as shown in Equation 8.

$$H(P, Q) = \sum_i P(i) \log_2(P(i)) - \sum_i P(i) \log_2(P(i)) - \sum_i P(i) \log_2(Q(i)) \tag{8}$$

Equation 9 shows the final term of the cross entropy for discrete distributions. $P(i)$ corresponds to the probability gathered by the ground truth data. $Q(i)$ is the probability generated by the neural network.

$$H(P, Q) = -\sum_i P(i) \log_2(Q(i)) \tag{9}$$

Equation 10 shows the calculation of the cross entropy for a single neuron.

$$\begin{aligned} \text{Class A prediction} = 30\% \quad & \text{Class A ground truth} = 0\% \\ \text{Class B prediction} = 70\% \quad & \text{Class B ground truth} = 100\% \\ H = -(0 \cdot \log(0.3) + 1 \cdot \log(0.7)) &= 0.154 \end{aligned} \tag{10}$$

Both the cross entropy and the KLD can be used as a loss function for neural networks. While KLD can be zero, the cross entropy will be equal to the entropy of the distribution, if both distributions are equal. For gradient optimizations, both methods generate the same end result in the network and can be equally used.

3 Methodology

In biomedical image processing, only little datasets are receivable or available [6]. Therefore, the architecture of neural networks differs, depending on the task it should be applied to. Ronneberger et al. [6] describe the u-net architecture, which is specifically developed for biomedical image segmentation. Key aspects of the network are the fast training, the use of data augmentation to enlarge the used datasets and the concatenation of higher resolution features in the up-sampled path to enable precise localization. Section 3.1 explains the structure of the used network. Three different data augmentation methods are explained in Section 3.2. Section 3.3 contains the used framework and system specifications.

3.1 Network Architecture

The architecture of the implemented network is based on the u-net convolutional network. Figure 12 shows an illustration of the u-net architecture. It comprises 4 contracting levels, 1 bottom level and 4 expansive levels. Every level of the contracting path consists of two convolution layers of size 3×3, each followed by a rectified linear unit (ReLU). In difference to the original implementation [6], reflect padding is used at each convolution layer, to create an output of the same size as the input. To down-sample the image, a 2×2 max pooling layer is used, with a stride set to 2. The number of features is doubled at each down-sampling. Special care has to be taken, that the layers are applied to even height and width inputs. No decimal numbers are allowed for height and width at any layer. For uneven inputs, cropping can be used to allow a seamless tiling of the output segmentation map. Another method is to add zero padding.
The bottom layer follows the same structure, but instead of using a down-sampling, an up-sampling operation is used, which halves the feature channels. A 2×2 transposed convolution layer, with stride 2, handles the up-sampling, followed by a ReLU. In the expansive path, a copy of the corresponding image of the contracting path (after the 2 convolution layers and the ReLU, but before the max pooling layer), is added to the feature map. Now, the first convolution halves the amount of features again, as seen in Figure 12 on the right path. If no padding is used, the corresponding contracting layer image has to be cropped to match the size of the expansive path image.
The final layer adds a 1×1 convolution, which maps the features to the number of defined classes.
In addition, a dropout layer with 50% probability is added before the last max pooling layer, and before the first up-sampling layer.
The network contains 19 convolutions, 4 transposed convolutions, 4 max pooling layers and 2 dropout layers.

3.2 Elastic Transformation

As mentioned by Ronneberger et al. [6], elastic transformations are a key concept for training, when working with very few annotated images. The differences between training the network with and without elastic transformations are described

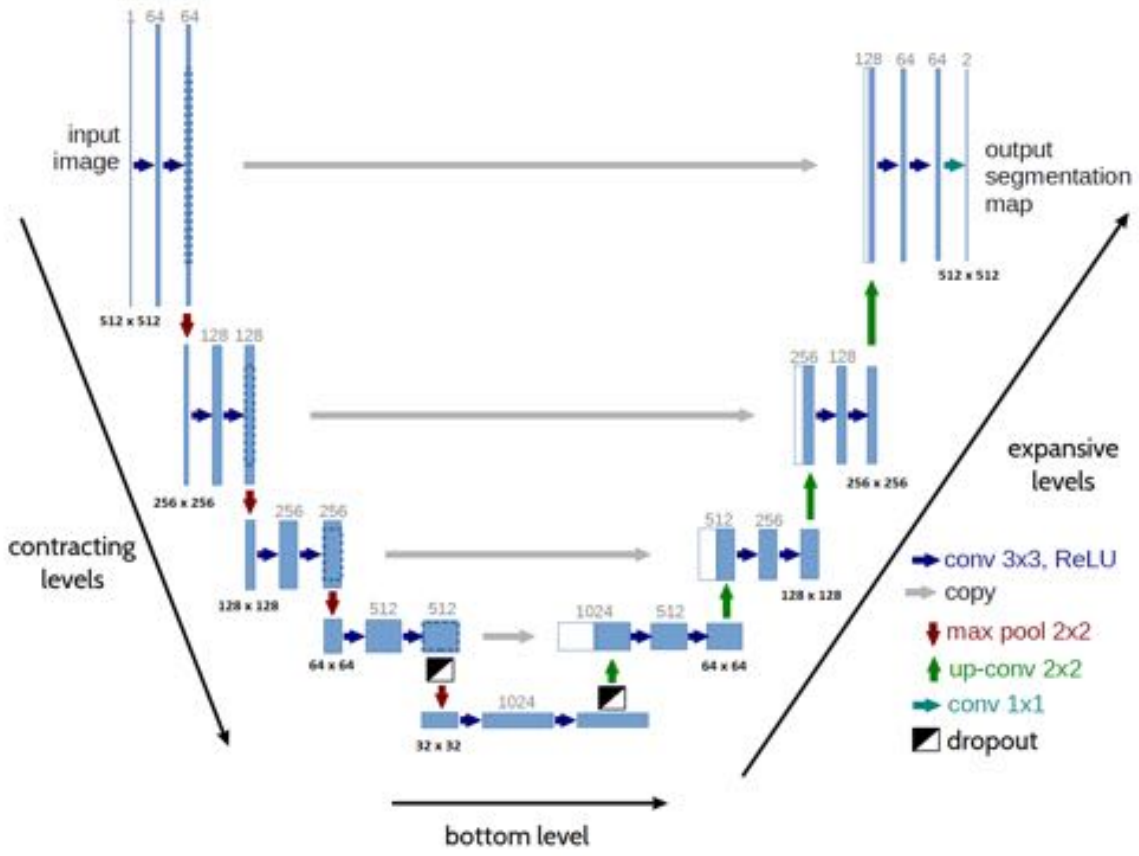


Figure 12: Adapted u-net architecture. The original u-net is proposed by Ronneberger et al. [6]. In comparison to Ronneberger et al. this u-net implementation uses padding to prevent a tiling of the input image. Image adapted from [6]

.

in Section 4.4.

In this section, three different methods to calculate elastic transformations are introduced. Section 3.2.1 and Section 3.2.2 discuss the use of random vector fields and Gaussian filtering to create elastic transformations. In Section 3.2.3, instead of using random vector fields, a gradient field is applied first. For comparability, the same random seed is chosen for each transformation.

3.2.1 Random Normalized Vector Field

First, we need to create a random field for the x and y direction with the same size as the input image. Each random field contains uniform distributed random numbers between -1 and 1. To both random fields, a Gaussian filter is applied. The filter size is calculated with respect to σ as shown in Equation 11 . For large values of σ, random values are close to zero in both fields. The random fields can be combined to create a displacement vector field. After normalizing each displacement vector to a

norm of 1, σ can be used to manipulate the look of the field. Small values of σ create random displacement fields. Choosing large values for σ let the displacement field look constant. Intermediate values of σ can create elastic looking deformations. Figure 13 shows different σ value vector fields. To control the intensity of the displacement a scaling value α is introduced, with which the displacement field is multiplicated [14].

$$FilterSize = 2 \cdot round(3 \cdot \sigma) + 1 \tag{11}$$

3.2.2 Random Scaled Vector Field

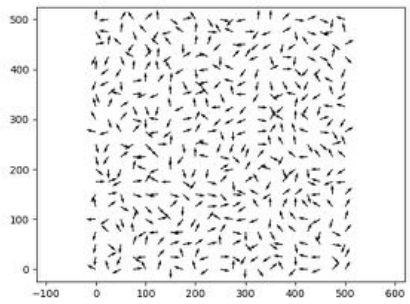
Instead of normalizing each displacement vector to a length of 1, like mentioned in Section 3.2.1, the displacement field is multiplied with a scaling value α directly. The effect can be seen in Figure 14. Random values, close to zero, do not impact the displacement, while long vectors have a bigger impact on the resulting transformation.

3.2.3 Elastic Gradient Field

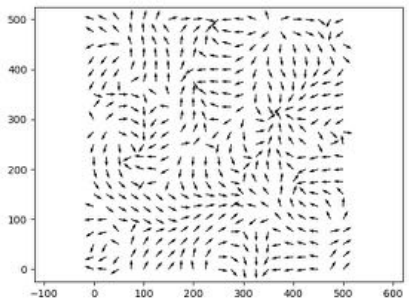
A different method to create elastic transformations, is the use of the image gradient. Instead of two random fields, the gradients dx and dy are calculated. Then, the gradient matrix values are scaled to be between -1 and 1. After that, a Gaussian filter is applied to the gradient fields. As mentioned in Section 3.2.1, the vectors can be normalized to length of 1, or no normalization can be applied as in Section 3.2.2. Two scaling factors, which are randomly chosen between two thresholds, are needed to introduce semi-randomness. Scaling factor α is multiplied with the first filtered gradient field, while scaling factor β is multiplied with the second gradient field. Figure 15 shows the elastic gradient fields with $\alpha = \beta$ and no normalization. Therefore, changing the value of the scaling factors randomly, the direction and intensity of the filtered gradient vectors can be manipulated.

3.3 Framework

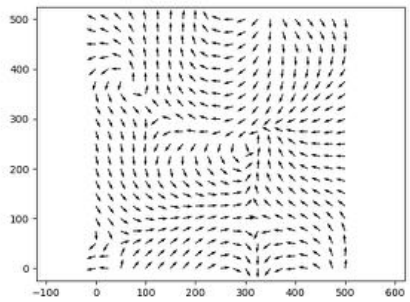
The original architecture of u-net network [6] is implemented in Caffe [7]. In this work, the architecture is converted to PyTorch 0.4 in Python 2.7, on an Ubuntu 16.04 system.
PyTorch is a deep learning framework, which is fast and flexible. It supports strong GPU acceleration with CUDA capable devices. Version 0.4 is currently the latest release version. Furthermore, version 1.0 will be released during 2018. The new version can be seen as a merge of Caffe2 and PyTorch.
The network is trained on an Intel i7-7700K with 16 GB RAM and a KFA2 Geforce GTX 1070 EXOC.



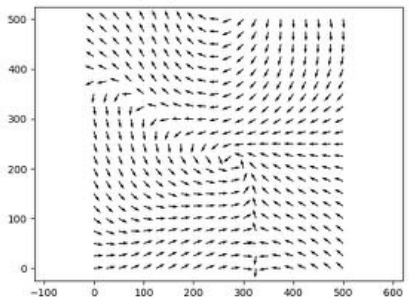
(a) $\sigma = 11$



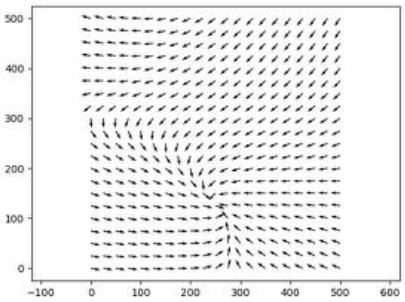
(b) $\sigma = 31$



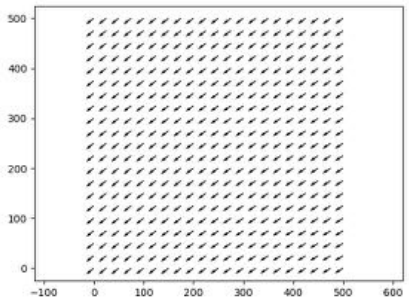
(c) $\sigma = 61$



(d) $\sigma = 91$

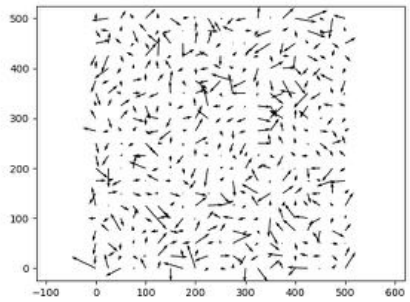


(e) $\sigma = 201$

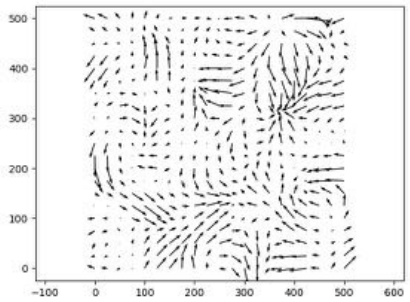


(f) $\sigma = 401$

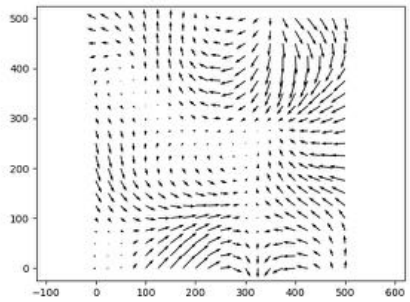
Figure 13: Normalized vector displacement field of size 512×512 with changing σ values. Every 25th vector is shown. Low values of σ create random fields, while high values of σ generate constant fields. Medium values create elastic looking displacement fields. For scaling, the field is multiplied with a scaling value α, after normalizing each displacement vector to length 1.



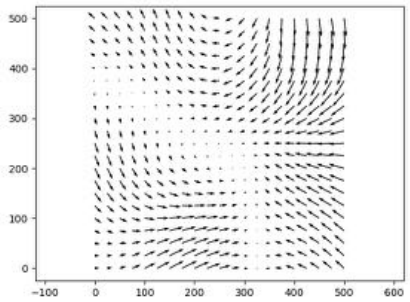
(a) $\sigma = 11$



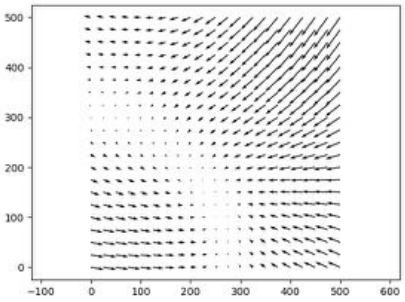
(b) $\sigma = 31$



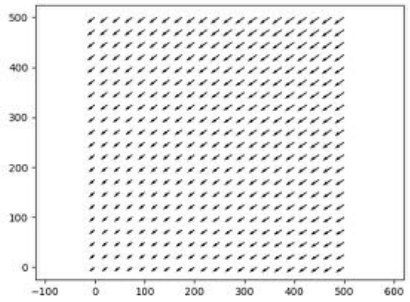
(c) $\sigma = 61$



(d) $\sigma = 91$

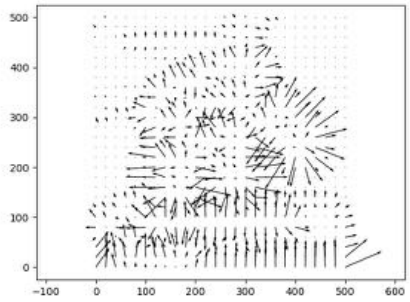


(e) $\sigma = 201$

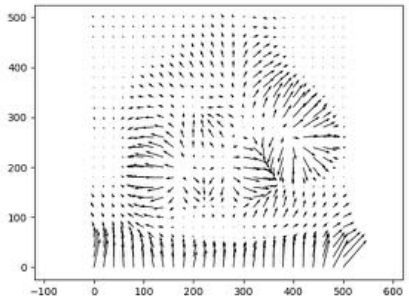


(f) $\sigma = 401$

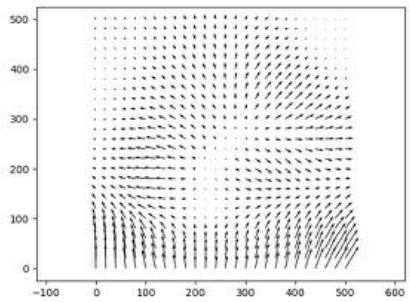
Figure 14: Scaled vector displacement field of size 512×512 with changing σ values. Every 25th vector is shown. Low values of σ create random fields, while high values of σ generate constant fields. Medium values create elastic looking displacement fields. For scaling, the field is multiplied with a scaling value α. No vector normalization is applied.



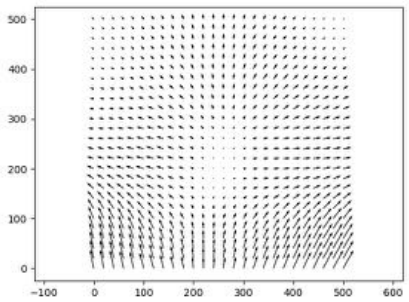
(a) $\sigma = 11$



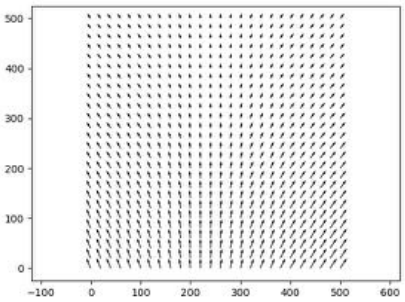
(b) $\sigma = 31$



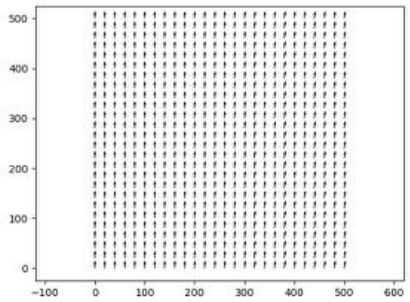
(c) $\sigma = 61$



(d) $\sigma = 91$



(e) $\sigma = 201$



(f) $\sigma = 401$

Figure 15: Scaled gradient displacement field of size 512×512 with changing σ values on the image in Figure 2. Every 25th vector is shown. High values of σ generate constant fields. Medium values create elastic looking displacement fields, which depend on the structure of the image. To introduce randomness, the field is multiplied with two random scaling factors α and β. No vector normalization is applied.

4 Results

Different augmentation methods are applied to the ISBI 2012 dataset [1], which is described in Section 4.1. The challenge dataset is made available after a registration and the leaderboard is updated daily. Section 4.2 explains the used metric for the ISBI 2012 competition [1]. Section 4.3 describes the training of the network with the experimental values. Finally, Section 4.4 shows the achieved results in the competition. A discussion about the achieved results is covered in Section 4.5.

4.1 Dataset

In may 2012, the first challenge on 2D segmentation of neuronal processes in EM images was launched [1]. Participants were given a training data set, which contains a set of 30 sequential sections from a serial section Transmission Electron Microscopy (ssTEM) data set of the Drosophila first instar larva ventral nerve cord (VNC). The ground truth for the 30 grayscale images is available too, where white pixels correspond to segmented objects and black is used for the rest, which contains mostly membranes.
Each training image is of size 512×512 in grayscale, with corresponding ground truth. 30 test images are made available without ground truth. These images are used as evaluation for the challenge. [1]
In this work, as a preprocessing step, the dataset is split into 20 training images and 10 validation images. From the 30 images, every third image is used for validation. Figure 16 shows 3 images from the training set with corresponding ground truth.

4.2 Evaluation

The main goal of the ISBI 2012 challenge [1] is to find an accurate boundary map. A simple measure is the pixel error. This method only classifies if a given pixel is correctly detected as a boundary pixel. The inability to detect merge errors, is a drawback of this method, as mentioned by Ignacio Arganda-Carreras et al. [1]. If a single pixel is missing between a border, only a small pixel error occurs, but a merge error is caused. Therefore, a new score was introduced in the challenge.

One can transform a boundary map into a segmentation by finding the connected components. First, a rand split score is defined in Equation 12. S is the predicted segmentation and T is the segmentation of the ground truth. p_{ij} can be defined as the probability that a randomly chosen pixel is part of segment i in S and segment j in T. $t_j = \sum_j p_{ij}$ is the probability of a randomly chosen pixel to belong to segment j in T. The numerator in Equation 12 , is the probability, that two randomly chosen pixels belong to the same segment in both S and T. The denominator is the probability that two pixels belong to the same segment in T. The whole equation can be seen as the probability of two randomly chosen pixels belonging to the same segment in S, with the condition that they belong to the same segment in T. A high rand split score, corresponds to fewer split errors [1].

$$V_{split}^{Rand} = \frac{\sum_{ij} p_{ij}^2}{\sum_k t_k^2} \tag{12}$$

The rand merge score is defined in Equation 13. $S_i = \sum_i p_{ij}$ is the probability of a randomly chosen pixel to belong to segment i in S. The Equation 13 can be described by the probability of two randomly chosen pixels belonging to the same segment in T, with the condition that they belong to the same segment in S. Fewer merge errors increase the merge score [1].

$$V_{merge}^{Rand} = \frac{\sum_{ij} p_{ij}^2}{\sum_k s_k^2} \tag{13}$$

With the weighted harmonic mean, both scores can be joined, to include both split and merge errors. If $\alpha = 0.5$, split and merge errors are weighted equally [1]. The official ranking system for the competition is seen in Equation 14. Borders between cells are often described differently by humans and algorithms. Therefore, border pixels in the ground truth boundary map are excluded in the calculation of the scores.

$$V_{\alpha}^{Rand} = \frac{\sum_{ij} p_{ij}^2}{\alpha \sum_k s_k^2 + (1-\alpha) \sum_k t_k^2} \tag{14}$$

A script to evaluate the training set is published at the website of the ISBI 2012 challenge [15]. The script evaluates a given ground truth and the resulting output segmentation map of the network. The segmentation map is thresholded in 0.1 steps and the best resulting score is used.

Information Theoretic Scoring is an alternative scoring system which was added later to the competition. The ranking depends solely on the Foreground-restricted rand scoring value. The Information Theoretic Scoring values are added for the sake of comparison. Both methods are explained in detail in the main publication about the competition, by Ignacio Arganda-Carreras et al. [1].

4.3 Training

Multiple versions of the same neural network are trained with different data augmentation methods applied. Data augmentation can be applied to cell images, because deformations are a common variation in tissues [6]. Realistic looking deformations can be approximated efficiently with the described data augmentation. Rotation and flipping of images is explained in Section 2.1.1. Elastic deformations are described in Section 3.2. The rotation is applied to the dataset as a preprocessing step in 10 ° steps. Flipping and elastic deformations are applied randomly (50% of the time) at runtime. Due to the imbalance of cell and membrane pixels, a rescaling weight is applied for each class. For the ISBI 2012 challenge [1], 2 classes exist. Class cell is scaled with 0.25, while the membrane class is scaled with 0.75. To generate these values, the ground truth from 5 images is used and the amount of black and white pixels is counted and averaged.
In comparison with the u-net [6] implementation, padding is used. The loss is calculated between the result and the ground truth with the cross entropy function, explained in Section 2.3.3. The stochastic gradient descent implementation of PyTorch 0.4 is applied with a learning rate of 0.001 and a high momentum of 0.99. These values gave the best results after training. The weights are initialized as mentioned by Ronneberger et al. [6]. The weights are drawn from a Gaussian

distribution. The standard deviation is $\sqrt{\frac{2}{N}}$. N corresponds to the number of incoming channels in the convolution layer. For a 3×3 convolution with 128 incoming channels N is equal to $9 * 128 = 1152$.
For the Random Normalized Vector Field in Section 3.2.1, $\sigma = 40$ and $\alpha = 32$ are used. The Random Scaled Vector Field in Section 3.2.2 uses the same $\sigma = 40$ value, but $\alpha = 1000$. The last approach, Elastic Gradient Field explained in Section 3.2.3, uses $\sigma = rand(70, 80)$, $\alpha = rand(-10, 10)$ and $\beta = rand(-10, 10)$, to create a semi randomness. Each network is trained for a maximum of 600 epochs, which corresponds to around 80 minutes of training time. After 600 epochs, the training is stopped. In all 7 trained networks, the best validation loss lies within the first 600 epochs.

Figure 17 shows the cross entropy loss for the training and validation sets, while using the elastic gradient field augmentation. Table 2 lists the best validation loss, the corresponding training loss and the best epoch for each tested network. Net 1 has the lowest validation and training loss, despite the lack of augmentation and class weights. An explanation for the good results is the sequential training set from the ISBI 2012 challenge [1]. The network tends to overfit, because of the similar looking images in the validation set.

	Best Epoch	**Train Loss**	**Val Loss**	**Data Augmentation**
Net 1	167	0.178	0.193	-
Net 2	148	0.203	0.231	W
Net 3	485	0.395	0.412	W, R
Net 4	234	0.201	0.222	W, F
Net 5	527	0.223	0.203	W, R, F, RNVF
Net 6	336	0.205	0.219	W, R, F, RSVF
Net 7	438	0.195	0.209	W, R, F, EGF

Table 2: Best epoch and best loss values for 7 different test networks. After the best epoch is reached, each network starts to overfit. A sign for the overfitting is the decreasing train loss while the validation loss increases. Symbols for data augmentation: W for class weights. R for image rotation. F for image flipping. RNVF for Random Normalized Vector Field. RSVF for Random Scaled Vector Field. EGF for Elastic Gradient Field.

4.4 Scores

The following section covers the achieved results at the ISBI 2012 challenge [1] for the implemented versions of u-net. Due to an improved data augmentation step, the adapted version of u-net in PyTorch 0.4, with Elastic Gradient Field, scores better results at the challenge than the original u-net version.

Currently, 145 submissions are registered at the leaderboard. 7 different self-trained u-net versions are submitted to the challenge and their results are listed in Table 3. All networks except Net 1 use class weights.

The best results for the implementation are reached by using Elastic Gradient Field (EGF). Table 4 shows the results of some preselected submission of the chal-

	Rank	Rand Score Thin	Information Score Thin	data augmentation
Net 1	92	0.947	0.978	-
Net 2	75	0.962	0.980	W
Net 3	57	0.970	0.983	W, R
Net 4	88	0.952	0.981	W, F
Net 5	41	0.973	**0.986**	W, R, F, RNVF
Net 6	67	0.966	0.983	W, R, F, RSVF
Net 7	**24**	**0.976**	0.985	W, R, F, EGV

Table 3: Rand score and Information score for the tested neural networks with data augmentation. Symbols for data augmentation: W for class weights. R for image rotation. F for image flipping. RNVF for Random Normalized Vector Field. RSVF for Random Scaled Vector Field. EGF for Elastic Gradient Field.

lenge. The top 40 submissions lie close to each other, all within less than 0.02 difference.

To conclude, the difference between the implemented method and the first rank is less than 0.011 for each scoring metric. A benefit of the self-trained u-net networks, are the fast training times. Within 80 minutes, the network is trained for 600 epochs on a KFA2 Geforce GTX 1070 EXOC and generates competitive results.

	Rank	Rand Score Thin	Information Score Thin
IAL MutexWS	1	0.9879	0.9918
CVLab	10	0.9811	0.9880
CUMedVision-motif	20	0.9765	0.9883
Net 7 (EGF)	24	0.9762	0.9856
Image Analysis Lab Freiburg	44	0.9727	0.9866

Table 4: Leaderboard ranking and their scores at the ISBI 2012 challenge [1]. Only one submit attempt is made for the self-implemented methods, to avoid exploiting information about the test dataset. Image Analysis Lab Freiburg is the original u-net implementation [6]. In this challenge, it is allowed to submit the result multiple times to achieve a better score.

4.5 Discussion

A surprising result is the score of Net 1. Despite the lack of any data augmentation and class weights, the trained network still achieves a high rand score of 0.947, as can be seen in Table 3. Because all networks with class weights scored on average 0.02 higher values at the ISBI 2012 challenge [1], the results of Net 1 are shown for comparison reasons only. A factor for the high score of networks without augmentation, is the thresholding step, which is applied before calculating the scores by the challenge. Therefore, low wrong probability values, do not influence the scoring system drastically.

Flipping of the dataset, as can be seen in the results in Table 3 of Net 4, yields worse results than no data augmentation. Net 2 scores a 0.01 better rand score than

Net 4. I assume, if the images in the training dataset are spatial dependent on each other, the hidden challenge dataset are spatially dependent as well. Therefore, the images do not need invariance to big rotations or mirroring. A deeper understanding of the Drosophila first instar larva ventral nerve cord (VNC) would be needed, to use data augmentation which exploits the conditions of the cells. Nevertheless, even with only $10\,^\circ$ rotations of the dataset, Net 3 achieves the third best results of the 7 trained networks.

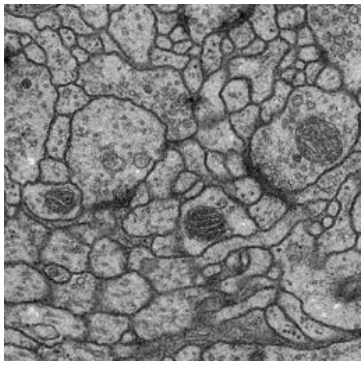
Finally, Figure 18 shows one image of the validation set with corresponding ground truth, on which all 7 networks are tested for comparison. Table 5 lists the rand score of the networks. Because the images in the dataset are of sequential nature, the validation set can not completely reduce overfitting. Therefore the values of datasets without data augmentation or only flipping, score good results. Figure 19 shows the corresponding output of the 7 self-trained networks.

Despite the highest score of all 7 trained networks in the challenge, Net 7 does not detect the black blob on the right half, as can be seen in Figure 19g. However, the bottom line segment gets detected to a higher degree.

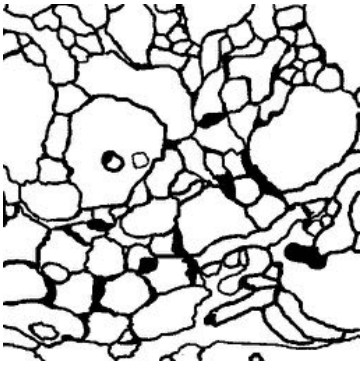
	Rand Score Thin	**Data Augmentation**
Net 1	0.777	-
Net 2	0.908	W
Net 3	0.686	W, R
Net 4	0.873	W, F
Net 5	0.895	W, R, F, RNVF
Net 6	0.899	W, R, F, RSVF
Net 7	0.854	W, R, F, EGV

Table 5: Rand Score of all trained networks for the validation image seen in Figure 18.

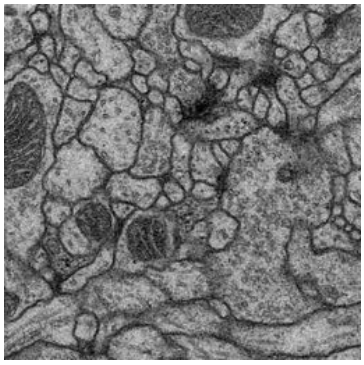
In conclusion, all 7 trained networks lie between a value smaller than 0.03 of each other. Factors for the close results are the sequential nature of the EM slice images and a thresholding which is applied before scoring evaluation. Because the images, with which the trained networks are tested at the ISBI 2012 challenge [1], are hidden from the public, only assumptions about the difference in the scores can be made. Both hidden and public dataset images seem to be fairly close to each other, in terms of the position and alignment of features of the Drosophila nerve cord.



(a) train image 1

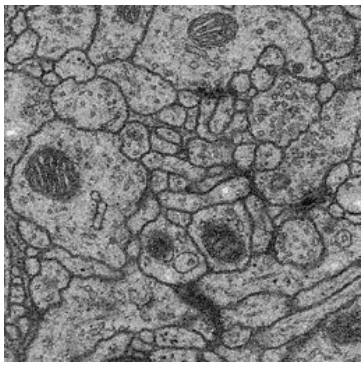


(b) ground truth 1

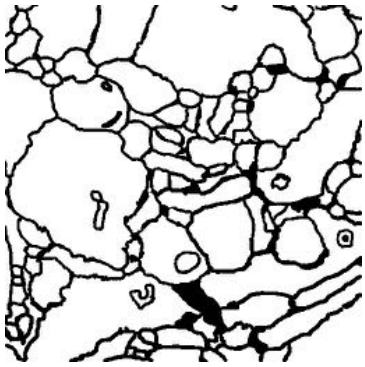


(c) train image 2

(d) ground truth 2



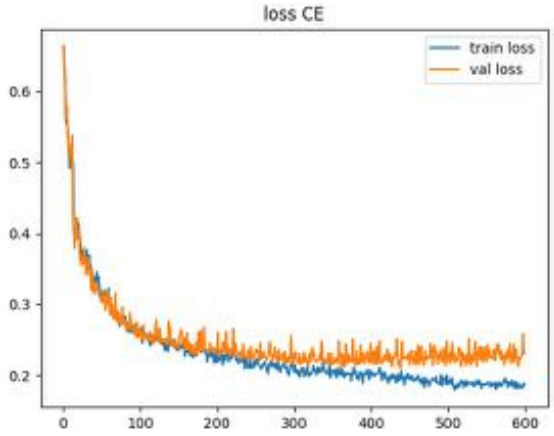
(e) train image 3



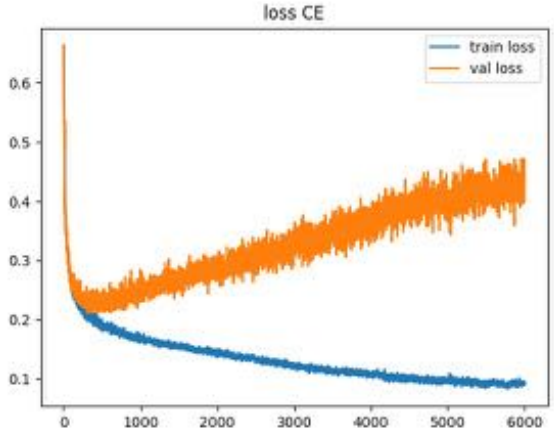
(f) ground truth 3

Figure 16: 3 images with corresponding ground truth from the ISBI 2012 challenge dataset [1]. White pixels in the ground truth belong to segmented objects, while black pixels correspond mostly to membranes. The ground truth is generated by two experts, who independently segmented the data, with two different software programs. The final ground truth was created by a consensus of both expert segmented boundary maps. It is stated that the ground truth itself can contain errors, relative to the underlying biological reality. Image reproduced from [1]

.

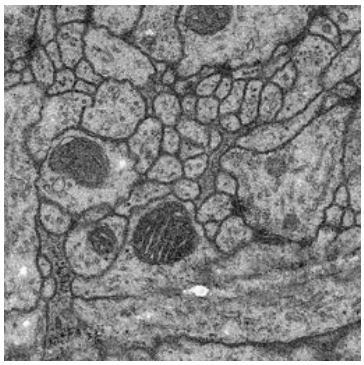


(a) loss while using Elastic Gradient Field 600 epochs
A discussion about the achieved results are in Section 4.5

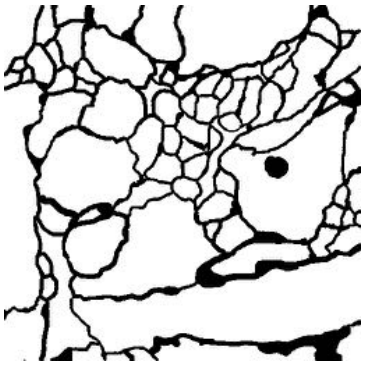


(b) loss while using Elastic Gradient Field 6000 epochs

Figure 17: Figures of the cross entropy loss for gradient data augmentation. The orange line represents the loss of the validation set for each epoch, while the blue line uses the training set for the loss calculation. In Subfigure 17b the overfitting of the data is shown. No improvement can be seen after 600 epochs.



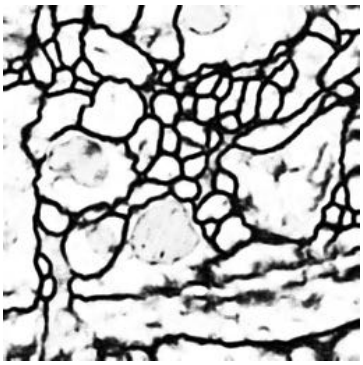
(a) Validation image 7



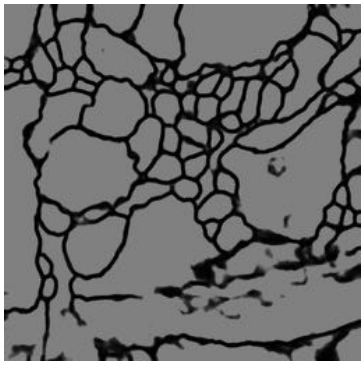
(b) Ground truth image 7

Figure 18: One image from the validation set with the corresponding ground truth.

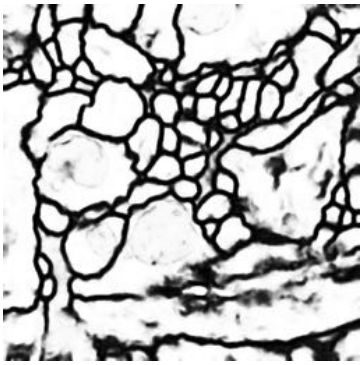
(a) Net 1



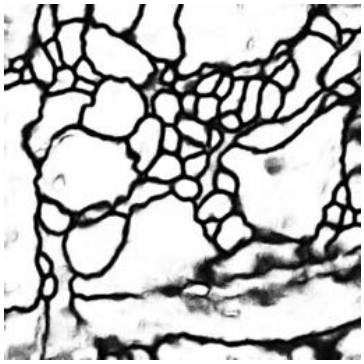
(b) Net 2



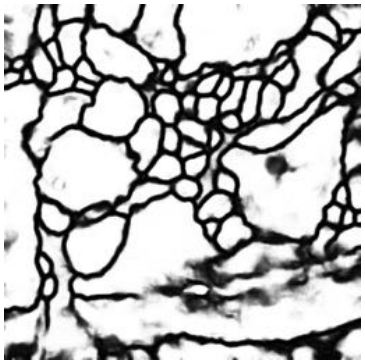
(c) Net 3



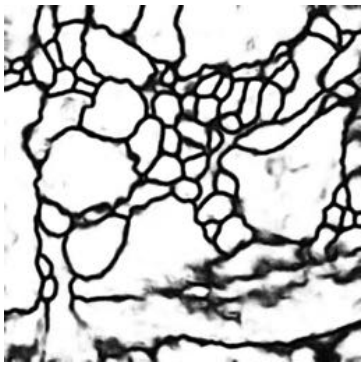
(d) Net 4



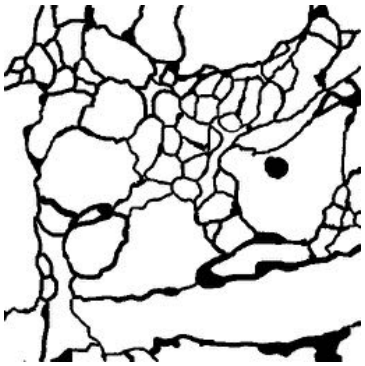
(e) Net 5



(f) Net 6



(g) Net 7



(h) Ground truth

Figure 19: 7 trained networks with different data augmentation, with their corresponding output of one validation image.

5 Conclusion

In this paper, an adaption from the u-net framework described by Ronneberger et al. [6] is implemented in PyTorch. First, preliminary knowledge, which is needed to understand and develop CNNs is given. Important layers of a CNN and the use of a loss function are explained. U-net is a convolutional neural network, which is specialized on little datasets. It comprises 4 contracting levels, 1 bottom level and 4 expansive levels as explained in Section 3.1. To compete at the ISBI 2012 challenge [1], three different image transformation methods are compared and used to improve the results on the challenge dataset. The trained network uses only 20 of the 30 available annotated images, and achieves a better result than the original u-net implementation, with far less training time. Using a Nvidia Geforce GTX 1070, the network takes 80 minutes to train for 600 epochs, containing 30 images each. Furthermore, only 1 solution for each of the three different image transformation networks is submitted to the ISBI 2012 challenge [1]. As a result, an adaptation of parameters to the test image set is avoided.
The PyTorch implementation is available at github, with an ISBI 2012 dataloader and without the trained models [16].

Results from Section 4 show, that, despite no use of data augmentation, Net 1, which scores the lowest of the 7 trained networks, would still be placed at rank 93 in the ISBI 2012 challenge [1]. With the introduction of only class weights, Net 2 places at rank 76. Surprisingly, adding random image flipping to the training images reduces the achieved rand score by nearly 0.01. One can argue, that a possible reason for this effect, is the sequential nature of the given EM slices of the Drosophilia first instar larva ventral nerve cord. Therefore, the hidden test images of the challenge have a sequential nature too. If the nerve cord is sliced at the same alignment every time, the flipping will introduce unnecessary mirroring. Invariance to such mirror images and large rotations, is maybe not needed. The applied augmentation learns the network invariance, which is not needed for the ISBI 2012 challenge, and therefore does not improve the rand score as expected.

As stated by Ronneberger et al. [6], u-net should use excessive data augmentation to allow the network to learn invariance. For the ISBI 2012 challenge, the usage of small rotation steps without any excessive data augmentation yields good results. Using elastic data augmentation only improved the resulting rand score by less than 0.007 as can be seen in Table 3. One can argue, that the scoring metric with the addition of a thresholding is not discriminative enough. For each evaluated image, the thresholding is adapted and only the best score is used. In real world applications, a fixed threshold, or user set threshold, would be used.

As future work, different loss functions can be evaluated and implemented. Additionally, the expansion of the u-net architecture with more layers could be evaluated. The use of the three different data augmentation methods explained in Section 3.2, needs to be reevaluated in different challenges, to confirm a increased benefit for the network.

References

[1] I. Arganda-Carreras, S. C. Turaga, D. R. Berger, D. Cireşan, A. Giusti, L. M. Gambardella, J. Schmidhuber, D. Laptev, S. Dwivedi, J. M. Buhmann, *et al.*, "Crowdsourcing the creation of image segmentation algorithms for connectomics," *Frontiers in neuroanatomy*, vol. 9, p. 142, 2015.

[2] G. Kumar and P. K. Bhatia, "A detailed review of feature extraction in image processing systems," in *Advanced Computing & Communication Technologies (ACCT), 2014 Fourth International Conference on*, pp. 5–12, IEEE, 2014.

[3] G. Dougherty, "Image analysis in medical imaging: recent advances in selected examples," *Biomedical imaging and intervention journal*, vol. 6, no. 3, p. e32, 2010.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255, Ieee, 2009.

[6] O. Ronneberger, P.Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, vol. 9351 of *LNCS*, pp. 234–241, Springer, 2015. (available on arXiv:1505.04597 [cs.CV]).

[7] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[8] D. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 2843–2851, Curran Associates, Inc., 2012.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[10] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *CoRR*, vol. abs/1511.07122, 2015.

[11] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[12] K. S. Reddy and D. K. R. L. Reddy, "Enlargement of image based upon interpolation techniques," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, no. 12, p. 4631, 2013.

[13] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012.

[14] P. Y. Simard, D. Steinkraus, J. C. Platt, *et al.*, "Best practices for convolutional neural networks applied to visual document analysis.," in *ICDAR*, vol. 3, pp. 958–962, 2003.

[15] "ISBI 2012 challenge evaluation script." http://brainiac2.mit.edu/isbi_challenge/evaluation. Accessed: 2018-07-26.

[16] "U-net pytorch 0.4 implementation with data augmentation." https://github.com/Mastercorp/U-Net-Pytorch-0.4. Accessed: 2018-07-27.