UMIT TIROL
DIE TIROLER PRIVATUNIVERSITÄT
Department for Biomedical Informatics and Mechatronics

Development of a preprocessing layer for the augmentation of 2D and 3D images in artificial neural networks

Aaron Bacher

Hall in Tirol, November 2, 2020

Bachelorarbeit

verfasst im Rahmen eines gemeinsamen Bachelorstudienprogramms von LFUI und UMIT – Joint Degree Programme

eingereicht an der UMIT – Private Universität für Gesundheitswissenschaften, Medizinische Informatik und Technik, Department für Biomedizinische Informatik und Mechatronik zur Erlangung des akademischen Grades

Bachelor of Science

Beurteiler:
DI Elias Tappeiner
Institut für Biomedizinische Bildanalyse

Betreuungsbestätigung

Ich befürworte die Abgabe der vorliegenden Abschlussarbeit, welche von mir betreut und insgesamt positiv bewertet wurde.

……
Datum und Unterschrift des/der Betreuer/in

Annahme durch das Studienmanagement

am:……

von:……

Kurzfassung

Um künstliche neuronale Netze ausreichend zu trainieren, so dass eine Überanpassung an den Trainingsdatensatz vermieden werden kann, werden riesige Mengen an Beispieldaten benötigt. Diese stehen oft nicht zur Verfügung und um dieses Problem zu bewältigen, wird Augmentation eingesetzt. Unter Augmentation versteht man den Prozess der Vergrößerung des Trainingsdatensatzes unter Verwendung desselben durch Anwendung verschiedener Techniken, in bildbezogenen neuronalen Netzen beispielsweise durch Transformation der Bilder. Um diese Transformationen von 2D- und 3D-Bildern in neuronalen Netzen zu verbessern, wurde ein neuer Preprocessing Layer entwickelt. Für die medizinische Bildanalyse, in der Trainingsdaten nur schwer zu erhalten sind, gibt es bereits verschiedene Frameworks, die solche Methoden zur Verfügung stellen. Eines davon ist NiftyNet, das mehrere verschiedene Preprocessing Layer für jeweils eine spezifische Bildtransformation bietet. Um dies zu verbessern, kombiniert der neue Layer alle verschiedenen Transformationen und erfordert daher nur eine einzige Interpolation für den gesamten Transformationsprozess. Dies wird durch die Kombination der Transformationsmatrizen der einzelnen Transformationen und die Verwendung einer elastischen deformation-Implementierung erreicht, die es erlaubt, Verzerrungen auf jeden einzelnen Pixel individuell anzuwenden. Durch die Einsparung von Mehrfachinterpolationen konnte die Zeit, die für dieselben Interpolationen benötigt wird, stark reduziert und mehr Bildqualität beibehalten werden. Während die Qualität des Ergebnisses des neuronalen Netzes nicht stark beeinflusst wurde, stellt die durch den Layer verursachte Zeitverkürzung eine große Verbesserung für den Trainingsprozess künstlicher neuronaler Netze dar.

Abstract

Artificial neural networks need a huge amount of example data to be trained sufficiently, in order to avoid overfitting to the training data set. To overcome the problem of having too little data, augmentation is used. Augmentation is the process of enlarging the training data set using the existing one by applying different techniques, in image related neural networks for example by transforming the images. To improve this process of transforming 2d and 3d images, a new preprocessing layer has been developed. For medical image analysis, where training data is scarce, a variety of frameworks have been in used which provide such methods. One of them is NiftyNet, which provides several different preprocessing layers, one for each specific image transformation. To improve this, the new layer combines all the pre-existing forms of transformations thus requiring only one interpolation for the entire transformation process. This is achieved by combining the transformation matrices of the single transformations and elastic deformation, which allows us to individually apply distortions on every single pixel. By saving multiple interpolations, the time needed for the same interpolations has been strongly reduced and image quality has been saved. While the quality of the outcome of the neural network has not been overly affected, the reduction in time caused by the layer is a major improvement for the training process of artificial neural networks.

Contents

List of Figures

List of Tables

1 Introduction

Artificial neural networks are a form of artificial intelligence trying to to emulate a biological brain. In the last few decades they have been improved enormously and today they are being used in many different fields of application. But regardless of their usage, they all work on the same principle: a large number of interconnected artificial neurons which are capable of learning by communicating with each other. A characteristic unheard of in conventional computer algorithms. The learning process is called training and is needed to be able to do a specific task afterwards. This training consists of the network trying the task over and over again, getting feedback if it was successful and if not, changing some parameters on its own to improve the outcome. Therefore training data is needed. Each single instance in this data set consists of an example and the corresponding ground truth. The ground truth has to be set by a human and represents the optimal result the network can produce for a certain input. The better and more reliable a network should be, the larger and more diverse this training data set must be. If these two properties are not met, the network will not be able to cope with other data than the once used in training. This problem is called overfitting.
One possible field of application is image analysis, in which neural networks are used to classify images or to detect, locate or segment objects in images. Segmentation is the process of marking areas in an image which belong to a specific object. Its result is a label map, in which the respective pixels contain the label of the object they belong to. To train a neural network for this purpose, the training data set has to consist of multiple diverse images and the associated correct labels. Depending on the scope, such data cannot always be easily found ready for use. For example in medical image analysis, where neural networks are used to find single organs in CT-images, training data is not as easy accessible as in other areas. To resolve this problem, augmentation is used to enlarge the data, so to create new images out of the existing ones. This can be achieved by transforming the image and the corresponding label map with the same affine transformations or elastic deformation.
NiftyNet[1] is an open source library for artificial neural networks, especially used in medical image analysis, which provides methods for achieving this type of augmentation. However, it performs all different transformations one at a time, which takes multiple interpolations to complete the augmentation process. To resolve this problem, a new layer has been developed, which unifies all different transformations and so reduces the amount of interpolations needed. This saves time as well as image quality. As the rest of the network, which in this case is based on NiftyNet, should not have to be changed, the layer must be structured in such a way that the original NiftyNet layers can easily be replaced with it.

[1] `https://NiftyNet.io/`

2 Related work

2.1 Artificial neural networks

General purpose

Artificial neural networks (ANN) have been subject to theoretical scientific discussion since the 1930s. Frank Rosenblatt released the first properly working neural network in 1958 as “Mark 1 Perceptron” [20]. Since then development in this field has made enormous progress, though the working principle has remained the same. By trying to imitate a biological brain or nerve system with all its neurons, ANNs are capable of making calculations which are not based on explicit knowledge but on empiric values, so knowledge is gained implicitly [14]. Today’s artificial neural networks are ready to be use in several fields of application, like speech recognition or image classification and segmentation.

Functionality

To get ANNs working, thousands of neurons must be connected to each other, as they also are in biological nerve systems. Each neuron takes multiple inputs from other neurons and multiplies these inputs with a separate weight for each one. Then it sums them up, possibly adding also a constant value named bias, which is individually assigned to each neuron. Finally, it applies a so-called activation function to the result of the addition and the outcome is the new input for another neuron. It is the aim of above the mentioned training process to find the optimum values for weights and biases. The activation function depends on the architecture of the network. Commonly used output functions are the so-called ReLU activation function [12], a rectified linear unit, or the classic tanh() function. Choosing the right activation function is crucial for the entire network to function properly and depends on the network’s purpose.
Next to the activation function, the choice of the architecture itself is key. The architecture defines the structure in which the single neurons are connected to each other. Basically, all neurons are organised in so-called layers. Each network consists of multiple layers, including one input layer, one output layer and potentially one or more hidden layers between the input and the output. Each of these layers can contain multiple neurons, which in conventional neural networks are fully connected, which means that each neuron receives an input from every single neuron of the previous layer and sends its output to every single neuron of the next layer.

Convolutional neural networks

Despite the fact that neuronal networks with fully connected layers have already proven themselves in various fields of application, on certain other tasks they do not work very well, as for example image segmentation. Images contain a huge amount of data resulting in even more individual weights which have to be trained and stored. The main advantage of convolutional neural networks (CNN) is their ability to systematically extract features out of the input [9]. To achieve this, CNNs instead of only using fully connected layers also dispose of some convolutional layers. In convolutional layers, different filter masks are applied on the image via discrete convolution. Convolution is a mathematical opera-

tion which replaces every value of a (multidimensional) array with the weighted average of the values surrounding it. Each convolutional layer may consist of different feature maps, whose neurons have the same shared weights and so perform the same operation on different regions of the image. These regions are called receptive fields. [9]
The use of shared weights in the feature maps reduces the amount of weights to be stored and trained and thereby the complexity of the network.

2.2 Network training

The fundamental basis for successfully applying any neural network is a large amount of example data which can be used to train it [28]. Training a network with too little data results in so-called overfitting, which means that after training the network is exclusively capable of working with its training-data set and nothing else. [24]
Depending on the area of application, plenty of such data is freely available on the internet, for example for the classification of images of dogs, cats, airplanes, ships and similar objects (e.g. CIFAR-10 and CIFAR-100 data set[2]). Such data sets are often used for competitions or courses and a huge variety thereof can already be found ready for use. This is, however, not the case in other areas of application, such as the segmentation of CT images in medical image analysis, where data is not as easily accessible [24]. Due to data protection regulations, these data sets are not freely available to everyone, but often only on request, and must be anonymized before they are handed over by hospitals or other health facilities. In addition, the amount of example images thus available is usually not very large, and consequently the resulting lack of data does not allow for a network to be properly trained in order for it to be used later on as a valid reference for all individual cases that might occur. At this point several different ways to avoid overfitting come into play.

Transfer learning [32] bases on the assumption that networks with similar purposes need similar basic capabilities. So a network which e.g. is used for image segmentation, can be trained first on another data set which is larger than the actual one. Afterwards, the network should be able to do basic segmentation and the actual data set can be used to train on specific details instead of starting from zero and using the valuable rare data to learn basic image segmentation. To do so, the weights of the convolutional layers from the finished first training serve as initialization values for the training with the actual data set. When using the technique of Transfer learning, the architecture of the second training network depends on the first one.
One step further in the development is Pretraining [4], where only some of the weights are copied and the architecture of the second training network can be chosen independently from the first one.
Still another method to prevent overfitting is Dropout [27]. When using Dropout, some activation values are randomly replaced with zero during training, which makes the network more robust. Further improvement to Dropout is Spatial dropout, which instead of deleting singe values resets entire feature maps to zero [30].

[2] `https://www.cs.toronto.edu/~kriz/cifar.html`

C. Shorten and T. M. Khoshgoftaar [24] list more techniques to avoid overfitting using different network setups, but those will not be discussed further at this point. Instead, a method aiming at the same objective but using a totally different approach, namely by trying to get to the root of the problem, is at the core of this thesis: image augmentation. In the field of medical image analysis, augmentation is defined as the enlargement of a data set (lat. augere = to increase, enlarge [19]), where new data is gained from the existing original data which can then also be used for training. For this purpose, the original data can be transformed either by data warping or oversampling. Data warping means applying e.g. color or geometric transformations onto the images, so that theoretically a new image is generated based on one specific image of the original data set. Oversampling, on the other hand, creates new images by mixing different images together. [24]

Image augmentation techniques

Depending on the area of application or the context of the image, different image transformations can be applied based on the image content. For images taken with a camera, rotations, translations and scalings in most cases are suitable, as the viewing angle and the distance between camera and object usually are not clearly specified or the size of the depicted instance of an object type may vary. Since many objects and living beings have an axially symmetric structure, reflecting can be used for augmentation as well and in some cases, also shear strains and cropping can deliver good contribution [16], [24].
Elastic deformation is another form of image manipulation changing the outline of depicted objects. In medical image analysis, where depicted objects are often soft tissues, e.g. organs which are non-rigid objects, they can be deformed easily by small forces which may cause minor changes on their outline and the ensuing image of the body not only looks different but provides for an entirely new image to be used in training. Elastic deformation aims at simulating those forces by applying a random field of distortion onto the medical image [2].
Beside the shape, also the colors are an important part of an image and can be used for augmentation. By changing the brightness [24], modifying the intensities of the single RGB channels [8] or applying various kernel-filter operations totally new images are created. For example, blurring the input image makes the network more robust against fuzzy images [24].

2.3 Transformations

All scaling, reflecting and rotating mentioned in previous the sections, belong to the range of linear transformations which are mathematically described with eq. 1. The parameters t_{ij} must be filled with appropriate values to get the equations for scaling, reflecting or rotating[1]. Shears are possible, too, but not further discussed in this thesis. x and y are the pixel coordinates in the input image and x' and y' their transformed positions.

$$\begin{aligned} x' &= t_{11} \cdot x + t_{12} \cdot y \\ y' &= t_{21} \cdot x + t_{22} \cdot y \end{aligned} \tag{1}$$

The above equation describes the two-dimensional case, for the three-dimensional case applies:

$$\begin{aligned} x' &= t_{11} \cdot x + t_{12} \cdot y + t_{31} \cdot z \\ y' &= t_{21} \cdot x + t_{22} \cdot y + t_{32} \cdot z \\ z' &= t_{31} \cdot x + t_{32} \cdot y + t_{33} \cdot z \end{aligned} \tag{2}$$

However, the pure linear transformation is not wanted, as it can only be applied on one specific point of the image, at the origin, to be precise [1]. As it is a convention in SciPy[3], OpenCV[4] and other image processing applications to set the origin of an image in one of its corners, a rotation as a pure linear transformation of 90° would result in the image not being located in the field of view any longer. Applying an affine transformation, instead, allows us to translate the origin into the centre of the image, which then also results in an image rotated by 90°, which, however, still lies within the field of view.
The essence of an affine transformation, therefore, is to combine a linear transformation with a translation, in order for the linear one to be applicable to any desired point of the image. Mathematically, this can be realised by simply adding a displacement vector to the linear equation:

$$\begin{aligned} x' &= t_{11} \cdot x + t_{12} \cdot y + u \\ y' &= t_{21} \cdot x + t_{22} \cdot y + v \end{aligned} \tag{3}$$

For simplicity reasons we stick with the two-dimensional case. The equations for the three-dimensional case can be combined analogously. Eq. (3) is a linear system and can therefore be written using matrices:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \underbrace{\begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{pmatrix}}_{\mathbf{T}} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} u \\ v \end{pmatrix} \tag{4}$$

What all transformations described by these equations have in common, is that they preserve parallelism, collinearity and noncollinearity, and the ratio of division. However, they can still be divided into two sub-groups.
On the one hand there are Similarities [1], which preserve angles. For example a radial transformation, also known as scaling, takes the single distances from the origin to every pixel and scales them with a positive constant. On the other hand, there is the group of Motions [1], which preserve the distances between pixels while transforming. From this perspective, motions are similarities if the constant scale factor equals 1. But there are other possibilities for motions, such as rotations, reflections and translations. Motions can be divided in orientation preserving and orientation reversing transformations [1]. While translations and rotations preserve orientation, reflections reverse it. The can be easily understood by looking at an image of a right turning arrow. The image can be translated and rotated multiple times, but the arrow will always turn right. However, if

[3] `https://www.scipy.org/`
[4] `https://www.opencv.org/`

the image is flipped once, the arrow will turn left. Going one step further, a combination of motions preserves the orientation of the image if the amount of orientation reversing operations is even ($\det\mathbf{T} = 1$), otherwise it reverses it ($\det\mathbf{T} = -1$) [1]. This will be important afterwards, when speaking of tissues specific to the left or to the right side of an object.
A crucial precondition of the orthogonal transformation matrix $\mathbf{T}$ is that its determinant must not be 0. If the determinant is not 0, the matrix has an inverse and the transformation can be undone if needed. [1]

As mentioned above, depending on the required transformation, all single values t_{ij}, with i and j corresponding to the row and column of its position in the matrix $\mathbf{T}$ respectively, have to be filled appropriately. So for the scaling with the two scale-factors a and b for the two spatial directions x and y respectively, the transformation matrix is:

$$\mathbf{T} = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix} \tag{5}$$

Negative scale-factors will mirror the image, so for example $-|a|$ causes reflection along the y-axis. In the three-dimensional case, $-|a|$ causes a reflection on the yz-plane.
Rotation-matrices are more complex to combine. Rotations can be described using polar coordinates, but since in matrix notation Cartesian coordinates are needed, trigonometric functions have to be used:

$$Rot\,(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \tag{6}$$

As it might not be obvious how to add a third dimension in this case, the rotation matrices for all three spatial axes have been listed in the following:

$$Rot\,(x,\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \tag{7}$$

$$Rot\,(y,\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \tag{8}$$

$$Rot\,(z,\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{9}$$

Homogeneous coordinates

Looking again at eq. (4), one will notice that for one transformation two mathematical operations must be executed: a multiplication and an addition, which in this notation cannot be further combined. Still, in order to reduce the amount of basic arithmetical operations, it would be desirable to do so and indeed this can be accomplished by adding another dimension and setting its default value to 1. Then, as we cannot multiply a 2x2 matrix with a 3x1 vector, the matrix needs to receive a third column, where the translation

vector is written. This way, the components of the translation vector is multiplied by 1, stay unchanged and are added to the intermediate result of the transformation matrix.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} t_{11} & t_{12} & u \\ t_{21} & t_{22} & v \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{10}$$

This principle is based on the so-called homogeneous coordinates. The additional dimension with default value 1 is the projective plane on which the image is located (or projective space in case of three dimensions), which allows for an affine transformation to be calculated as a linear one [26]. Taking one step further we can also add an additional line to the matrix:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \underbrace{\begin{pmatrix} t_{11} & t_{12} & u \\ t_{21} & t_{22} & v \\ p & q & 1 \end{pmatrix}}_{\mathbf{T}_{projective}} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{11}$$

While the value in the bottom right corner of the matrix has to remain 1 to keep the image on the projective plane, two new variables p and q are added, which also allow projective transformations. As those are of less importance in this thesis, p and q will be set to zero. Accordingly, the transformation matrix is:

$$\mathbf{T}_{projective} = \begin{pmatrix} \mathbf{T} & \vec{t} \\ \vec{0}^T & 1 \end{pmatrix} \tag{12}$$

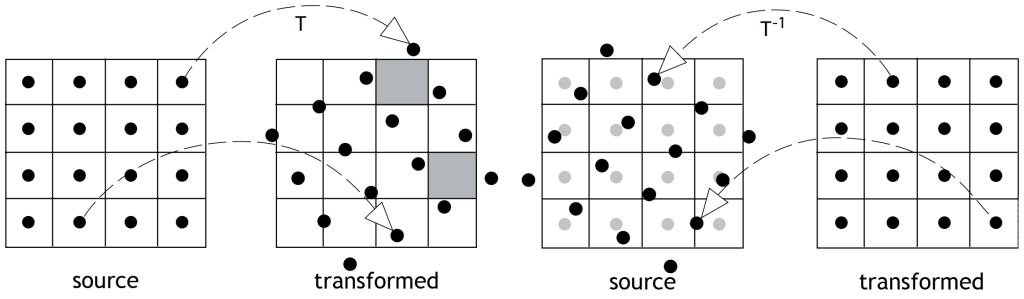
with the original transformation matrix $\mathbf{T}$ of the size nxn with the amount of columns (and rows) being n and the translation vector $\vec{t}$ also of length n. The newly created affine space has spatial dimension $n+1$ [22]. However, setting p and q to zero makes the last row of the matrix redundant, as z' will always be 1 and can therefore be omitted, which takes us back to eq. 10.

2.4 Interpolation

Images can be saved digitally in two different ways. On the one hand there is the vector graphic which consists of various mathematically defined forms. A significant advantage of this method is that the image can be scaled absolutely without data loss, as only the mathematical representation of the geometric shapes have to be adjusted. On the other hand, raster graphics such as CT-images consist of individual points with different colour values which are arranged on a grid in a specific area and form the image together. Therefore, those single picture elements are called pixels, while in the three-dimensional case voxel stands for volume element. For simplicity reasons, we will talk about pixels only hereafter, but the same applies to voxels.

The depiction of an image from multiple single points leads to a discrete data representation: the image is sampled. If a transformation is applied to it, the newly calculated position of a pixel does therefore, on average, not exactly correspond with another sampling position, but will be somewhere in the space between. However, in practice, image transformation is performed in the reversed direction. So not the source image will be transformed to the resulting one (forward mapping), but the pixels of the resulting image

will be transformed back to the source (backward mapping). The advantage of backward mapping is that, unlike forward mapping, neither holes nor overlaps are created in the transformation process, since coordinates which do not correspond to sampling positions can be interpolated from neighbouring sampling points [23]. This is visualized in fig. 1.



(a) forward mapping (b) backward mapping

Figure 1: [23] transforming an image with forward mapping creates holes (grey fields) in the output image

Interpolation is the process of finding all co-domain values for the full domain according to some sample points [17]. When interpolating, values are calculated based on the given nearby values. A variety of interpolation methods can be applied. The easiest one is the nearest neighbour interpolation, which takes the value of the nearest sampling point for any pixel situated in an interspace [33]. For the bilinear interpolation (with two dimensions; trilinear with three dimensions) the space between the sampling points will be completed with linear functions (for x, y and z-axis, respectively) of the neighbouring pixels and so any position in between can be determined more precisely. Still, there are more accurate methods. Using cubic instead of linear interpolation functions includes more sample points and so it better able to do justice to the original image. However, the maximum-minimum principle is no longer given. This means that it is no longer guaranteed that the interpolated values lie only within the maximum and minimum of the respective adjacent sampling points. A last interpolation method which should be mentioned are splines [21]. In contrast to the previously described cubic interpolation, which uses four sample points (per dimension, so 16 when using bicubic interpolation), splines use only two sample points while ensuring that the first and second derivative at the sample points are continuous. They fill the interspaces with low order polynomials whose order define the order of the splines.
In the end, every interpolation method must come up with new data to fill its empty spaces, thus causing the percentage of original pixel contained in the image to decrease and information to get lost. For this reason it is recommendable to avoid interpolations whenever possible.

2.5 Image quality metrics

While it is quite easy for the human eye, or maybe even a neural network, to compare and determine the differences between two images (subjective evaluation), a computer can only do so on pixel-level (objective metric) [29]. Image quality metrics work on a mathematical basis, accordingly every single pixel from one image will be compared to the corresponding pixel of the second one, using different measurements. Those methods usually only work with single channel images [6].
Objective comparison methods aim at giving a quantitative relation between two images. As most of the times not two random, and therefore quite differing, images need to be compared, but two very similar ones, for example the same image before and after a transformation, we are interested in image quality metrics (IQM). Three different cases can be distinguished: if the original image is given, the transformed one can be directly compared to it and the procedure is called full-reference; it is referred to as reduced-reference if the original image is only partly known and as no-reference if it is not known [31]. As in this thesis all images are known to the user, it will focus on full-reference metrics only.

Peak Signal-to-Noise-Ratio

One of the simplest and probably most frequently used IQM is the peak signal-to-noise-ratio (PSNR) [29], a variant of the mean squared error (MSE). The MSE is defined as the average difference between each of the pixels' gray values of images **x** and **y**. Two images are compared, one being the target and the other one the reference. The full-reference comparison is given as:

$$MSE(\mathbf{x},\mathbf{y}) = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2 \tag{13}$$

with N corresponding to the total amount of pixels contained in each of the images. As the differences are squared, the order of the inputs **x** and **y** is irrelevant.

PSNR is originally mentioned in signal analysis when comparing the average strength of a signal to its noise and it is called signal-to-noise-ratio (SNR). Since in image analysis the peak of the signal is used to perform this comparison, the highest possible value is used, which is 255 when working with 8 bit per pixel. For the noise we use the MSE and because the result is a ratio, it is converted to decibel dB:

$$PSNR = 10 \cdot \log_{10}\left(\frac{255^2}{MSE}\right) \tag{14}$$

Since the MSE is invariant to the order of the inputs, this applies also to PSNR.

Structural Similarity

Although the PSNR is a simple and intuitive method, it does not take into account the Human Vision System (HVS) [29] because only single pixels and no structures are compared. For example, totally different types of distortions applied on the same image could theoretically produce the same MSE. Therefore, Wang et al. have come up with a new

method which they name structural similarity (SSIM) [31] and which is made up of the multiplication of differences in luminance l, contrast c and structure s of two images. The influence of the three components can be defined by adjusting the parameters α, β and γ, respectively.

$$SSIM(\mathbf{x},\mathbf{y}) = [l(\mathbf{x},\mathbf{y})]^{\alpha} \cdot [c(\mathbf{x},\mathbf{y})]^{\beta} \cdot [s(\mathbf{x},\mathbf{y})]^{\gamma} \tag{15}$$

As we are talking about single channel images, with 0 is black and 255 being white, the value of each pixel is used to calculate the mean intensity of each image:

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{16}$$

The same will be applied on image $\mathbf{y}$. The luminance defines how close the two images are in terms of intensity [6] and is calculated by:

$$l(\mathbf{x},\mathbf{y}) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{17}$$

This function becomes 1 if the the mean intensities of both images are the same and decreases if they are different. C_1 is a constant to prevent the denominator from becoming 0. Wang et al. define it as $C_1 = (K_1 \cdot L)^2$, where $K_1 \ll 1$ is a small constant and L is the pixel value range (e.g. 255 for 8-bit gray scale images). To determine contrast, the standard deviations of each image are used, which are calculated by:

$$\sigma_x = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2} \tag{18}$$

The same applies to image $\mathbf{y}$, again N corresponds to the total amount of pixels in each image and with σ_x and σ_y the contrast can be calculated as in eq. (17) with a small constant $C_2 = (K_2 \cdot L)^2$, analogue to C_1:

$$c(\mathbf{x},\mathbf{y}) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{19}$$

For the above mentioned structure the Pearson correlation coefficient ρ [15] is used. The correlation coefficient is the standardisation of the covariance σ_{xy} and describes the connection between two random variables. If they are strongly connected, ρ approaches 1, else 0. If they are connected in a reversed way, ρ approaches -1. The covariance is calculated by:

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y) \tag{20}$$

and then inserted into the equation for the correlation coefficient to get the value for the structure. Again a small constant C_3 is introduced to prevent the denominator from becoming 0:

$$s(\mathbf{x},\mathbf{y}) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{21}$$

Having defined all three components l, c and s, it is now possible to combine them by multiplication. Following Wang et al. [31], all three parameters a, b and c are set to 1 to have the same influence on all parts and C_3 is set to $\frac{C_2}{2}$. The formula for structural similarity is given by:

$$SSIM(\mathbf{x},\mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{22}$$

This measurement method now has three major advantages. First, like PSNR, it is symmetrical and therefore independent of the order of the input images. But unlike PSNR, its outcome will always be in range $[-1,1]$, while PSNR's has no upper limit. This is due to the fact that the outcomes of l and c are in range $[0,1]$ and the one of s in range $[-1,1]$. Additionally, if two identical images are compared, its outcome will be 1. If two identical images are compared with the PSNR method, the result becomes infinite, as the denominator becomes 0. [31]

3 Methods

3.1 The downside of NiftyNet

CNNs used in medical image analysis are often using third party libraries for image augmentation. One of these is NiftyNet, which among other things provides a collection of preprocessing network-layers. Each of these layers performs a different transformation on an input image. For example, one could insert a layer for image flipping, another for scaling and a third one for rotating. Since these three layers do not work together but independently one after the other, the images get interpolated three times in total, once by every layer. Due to this procedure, not only does valuable image data get lost three times but also is time being spent which could otherwise be used to train the network, as the augmentation process relates only to the preprocessing steps. It is, therefore, the objective of this thesis to reduce the amount of interpolations as much as possible.

3.2 Principle

As has been outlined in section 2.3 on transformations, all three of the above-mentioned operations are performed by multiplying a transformation matrix with the raster coordinates of each pixel. The chaining of these transformations by multiplying their matrices allows us to pre-compute a combined transformation matrix which, when applied, can perform all the chained transformations at once. To perform the transformation - especially the rotation - in relation to the image's centre, its origin must be translated there. It is only afterwards that the actual transformation can be executed, whereupon the origin must be translated back. When using homogeneous coordinates, this can be realised by two matrix multiplications, before and after the transformation, respectively. These two translation matrices can thus be integrated in the combined transformation matrix.
To give an example, a volumetric image of size 128x176x176 is flipped along the xz-plane, scaled by 1.05 in each direction and rotated by 3° around the z-axis. For this operation, it takes five matrices of size 4x4 to be executed:

$$
\begin{aligned}
\mathbf{T} = & \begin{pmatrix} 1 & 0 & 0 & 64 \\ 0 & 1 & 0 & 88 \\ 0 & 0 & 1 & 88 \\ 0 & 0 & 0 & 1 \end{pmatrix} && \text{second translation matrix} \\
& \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(3^\circ) & -\sin(3^\circ) & 0 \\ 0 & \sin(3^\circ) & \cos(3^\circ) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} && \text{rotation matrix} \\
& \cdot \begin{pmatrix} 1.05 & 0 & 0 & 0 \\ 0 & 1.05 & 0 & 0 \\ 0 & 0 & 1.05 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} && \text{scale matrix} \\
& \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} && \text{flip matrix} \\
& \cdot \begin{pmatrix} 1 & 0 & 0 & -64 \\ 0 & 1 & 0 & -88 \\ 0 & 0 & 1 & -88 \\ 0 & 0 & 0 & 1 \end{pmatrix} && \text{first translation matrix} \\
= & \begin{pmatrix} 1.05 & 0 & 0 & -3.2 \\ 0 & -1.049 & -0.055 & 185.109 \\ 0 & -0.055 & 1.049 & 0.562 \\ 0 & 0 & 0 & 1 \end{pmatrix}
\end{aligned} \tag{23}
$$

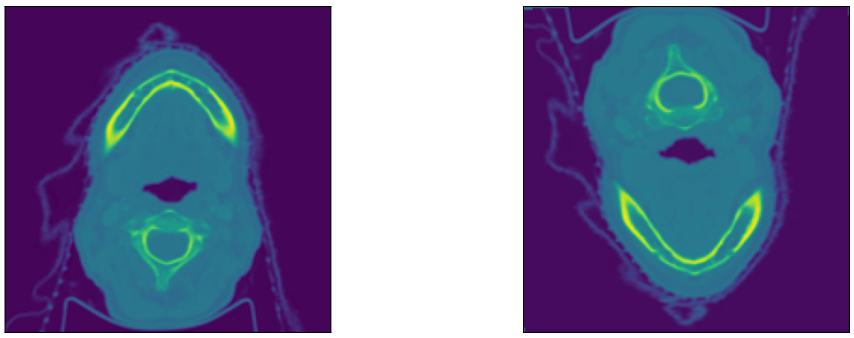
Note that the matrices are multiplied in reverse order since if they were multiplied one after the other by the vector of the image coordinates, the last would be multiplied first.

$$
\mathbf{x}' = \underbrace{\mathbf{T}_{trans2} \cdot \mathbf{T}_{rot} \cdot \mathbf{T}_{scale} \cdot \mathbf{T}_{flip} \cdot \mathbf{T}_{trans1}}_{\mathbf{T}} \cdot \mathbf{x} \tag{24}
$$

As mentioned in section 2.4, usually it is not the pixels of the original matrix which are mapped onto the output image, but the output pixels are mapped back to the original image. Therefore the resulting matrix is inverted.

$$
\mathbf{T}_{inv} = \begin{pmatrix} 0.952 & 0 & 0 & 3.048 \\ 0 & -0.952 & -0.05 & 176.08 \\ 0 & -0.05 & 0.952 & 8.691 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{25}
$$

The outcome of this transformation is depicted in fig. 2. As mentioned by Nikolov et al. [13], for the application of image segmentation, labels which are specific to one side of an object must be adjusted when flipped, regardless of the axis or plane. This is because reflections are not orientation preserving operations [1]. Adjusting these labels does not mean flipping them again, as then the label-map would not match with the image of the volume anymore, but swapping the identifiers. If for example label x is specific to the



(a) original (b) transformed according to eq. 25

Figure 2: example of an affine transformation

left and label y to the right, then after flipping label x must be renamed to y and vice versa. An example for this is shown in fig. 3.

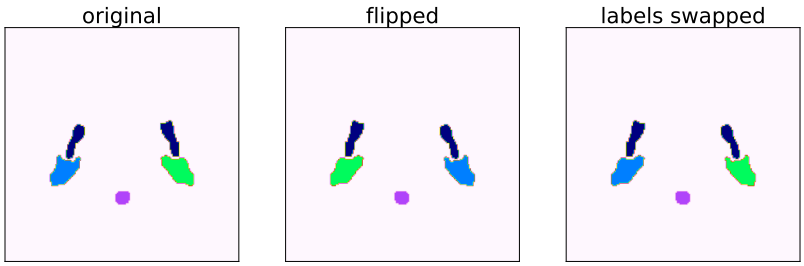


Figure 3: swapped labels on flipped label-map

3.3 Random Transformation Layer

So far it has been shown how the different transformations should be combined in theory. It is the aim of this thesis to construct one single preprocessing layer which can be used in the same way as the NiftyNet layers thus providing for the original layers to be replaced by it. Therefore a new individually configurable layer is being developed which, by combining some of NiftyNet's already existing layers, serves as an extension to it. Since it includes several different transformations, it is named Random-Transformation-Layer[5]. The layer is able to perform rotations, scalings, flips and elastic deformation, which can be specifically selected by the user. All different transformations can be switched on or off independently and the ranges of the single transformations, e.g. the minimal and maximal scaling percentage, can be set.

[5] `https://github.com/aaronbacher/RandomTransformationLayer`

Affine transformations

In image segmentation, it does not make sense to transform one single image randomly, as the image itself and the corresponding label-map always belong together. Therefore, the layer always takes a set of images which are transformed exactly in the same way. For each image set, within the given ranges, the layer generates a new random state which determines all relevant transformation parameters for every active transformation. These parameters are used to create the transformation matrices for every single transformation. As the transformation is performed via backward mapping, at the end the inverse transformation matrix is necessary. To avoid the necessity to invert the matrix at the end, this is already done for each single transformation matrix as they are constructed.
As the inverse matrix of an identity matrix the identity matrix, also a flip matrix does not change when being inverted, as its only difference to an identity matrix is a negative sign. Since a rotation matrix is orthogonal, its inverse can be calculated by transposing it [10]. The inverse scale matrix is realised not by multiplying the values by a factor but by dividing them by the same and the translation matrix gets inverted by changing the sign of all offset values.

$$\text{flip} \qquad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \overset{\text{invert}}{\Longleftrightarrow} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{26}$$

$$\text{rotation} \qquad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \overset{\text{invert}}{\Longleftrightarrow} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{27}$$

$$\text{scale} \qquad \begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \overset{\text{invert}}{\Longleftrightarrow} \begin{pmatrix} \frac{1}{x} & 0 & 0 & 0 \\ 0 & \frac{1}{y} & 0 & 0 \\ 0 & 0 & \frac{1}{z} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{28}$$

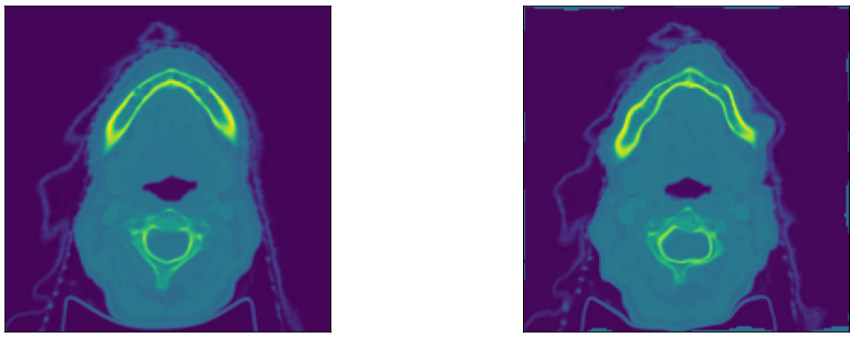
$$\text{translation} \qquad \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \overset{\text{invert}}{\Longleftrightarrow} \begin{pmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{29}$$

The construction of the single matrices is not very time consuming, and once the final transformation matrix is created, the amount of time needed for the affine transformation depends solely on the size of the image and not on how many different transformations have been combined.

Elastic deformation

Unlike NiftyNet, whose elastic deformation is based on an implementation of Milletari et al. [11], this layer uses a implementation developed by Simard et al. [25]. First, a field the size of the image is created and filled with random numbers between -1 and +1 before a Gaussian filter with standard deviation σ is applied, with σ being the elasticity

coefficient. If it is small, it affects small regions of the image independently from each other. The bigger σ is, the bigger the areas get and if it is very large, it finally ends up as a translation of the whole image in the same random direction. After applying the filter, the field of displacements is normalized and can be scaled with a second parameter α. A possible outcome of such a deformation is depicted in fig. 4. In practice, α is not usually that big, here, however, it has been set to 200 to make the differences clearly visible. As this method works with each pixel individually, it is better combinable with the affine transformation than NiftyNet's version which this only works with b-splines and a few control points.



(a) original (b) deformed with $\sigma = 5$ and $\alpha = 200$

Figure 4: example of an elastic deformation

Executing the transformations

Since the implementation of elastic deformation requires much more time than the sole affine transformation (see chapter 4.1 on quantitative layer evaluation), the layer chooses from three different ways of executing the transformation, depending on which transformations are needed.
If elastic deformation is not required, all the layer does is applying an affine transformation onto the image. If the layer is supposed to flip images only, even performing this flip with a matrix multiplication would be exaggerated, as it does not take interpolation for a flip along the main axes. Therefore, in this special case, instead of performing a matrix multiplication, NumPy's[6] flip method is used, which is much faster (see table 1).
If elastic deformation is used, the layer sets up a random state for random distortions. These distortions are temporarily saved in an array and are not applied to the image, as the image is not to be interpolated yet. This array of distortions can be imagined as a vector field, i.e. an image in which every pixel contains the coordinates which will be the pixel's new coordinates after the distortion is applied. To finalize, the affine image transformation must be combined with these distortions. If this step has been completed, the pixels can be mapped from the input to the output image.

[6] `https://numpy.org/`

Either way, one last step has to be made before the transformation can be completed. If the image being processed is the label-map of an image-set and it has be flipped, the labels have to been adjusted as visualised in fig. 3. However, if the image has been flipped twice, the labels are correct again and do consequently not need to be swapped anymore. Therefore, the layer keeps track of how many times an image has been flipped and only if this number is an odd one, the corresponding labels will be swapped.

3.4 Cython

Python is an interpreted and untyped programming language. In contrast to compiled languages whose instructions are translated into machine code before execution, the single instructions in interpreted languages are interpreted only during run-time, which makes execution slower. Additionally, in untyped programming languages no variable corresponds to a fixed data type, instead, it gets defined only during run-time and can be redefined multiple times. Therefore every time a variable is called, the compiler does not know its type and has to determine it, which takes some time. If this happens during a sequential program without major loops, it does not considerably affect the run-time of the program. However, during big loops, for example when iterating the pixels of an image, untyped variables and the instructions, which have to be interpreted in each iteration anew, have some serious impact on the duration of the program. Other programming languages, for example C, which are compiled before execution, require variables to be declared before first usage and do not allow for data types to be changed during run-time. They therefore can cope much better with big loops.
To deal with this weakness of Python, it is possible to write Python-modules in faster programming languages whose functions can still be called like any other normal python-function. The best known examples are NumPy and SciPy, which both provide methods to work extremely fast with huge amounts of data. Although they provide a wide range of possibilities, in some special cases their resources may not be sufficient and they do not offer the ability to create customized fast functions.
One way to achieve this is Cython[7]. Cython provides a C-extension for Python-programs and with that the possibility to boost single code-snippets. The idea is to rewrite the respective functions in a separate .pyx-file. In this file, not only can code in classic Python-syntax be supplemented for example with static type declarations but working with pointers is possible, too. After finishing the file, it must be compiled with the provided Cython-compiler, which returns compiled files, one of them being a shared library. Finally, this .so-file must be provided to the main python program, where it can be imported like any other module.
To demonstrate the improvements accomplished by just a few lines in Cython, let us look at the following example: a function takes an integer x as parameter and with a for-loop it counts from zero up to x. Additionally to the counter variable i of the loop, another variable j gets increased simultaneously. Obviously, this function is not very useful in everyday problems, but it is exactly this simplicity which it takes to show the differences between Cython and Python. Therefore, the same function is implemented once in Python and once in Cython. It is almost the same lines of code in both of them, except that in Cython, both i and j are declared as integers before the loop starts and, of

[7] `https://cython.org/`

course, that Cython gets compiled before execution. Then both functions are executed with the same parameters and the time needed is measured.

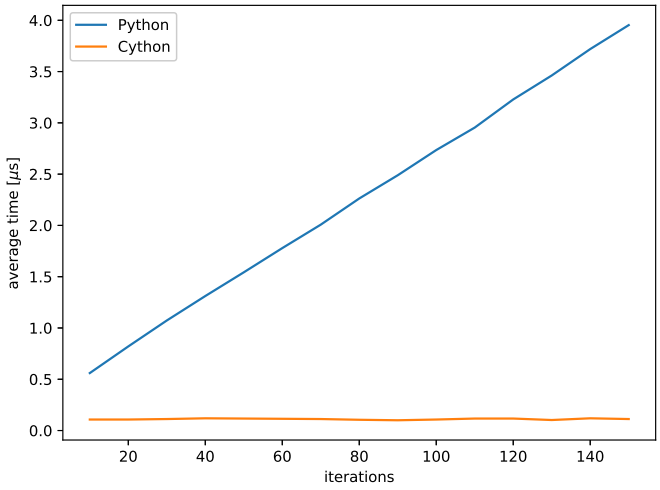


Figure 5: time consumed by Python and Cython loops with different numbers of iterations.

The outcome of the measurement can be seen in fig. 5. To receive accurate results, each iteration measured is repeated 100 times and then the average is taken. It is clearly visible that Cython is much faster, even if we are only in µs-range, and while the duration of the Python-function increases substantially with the number of iterations increasing, the duration of the Cython-function on this scale seems to remain constant.
Considering the amount of time it takes Python to come to terms with only few iterations, it is obvious, that it will be rather difficult to implement an image transformation which takes much more iterations (e.g. around 4 millions when speaking of volumetric images with 128x176x176 voxels). Therefore, the above described function to combine the distortions of elastic deformation with affine transformation is implemented in Cython.

4 Results

Comparing the original NiftyNet layers with the newly developed all-in-one random-transform layer (RTL), the latter seems to be advantageous in theory. Reducing the amount of required interpolations, so that no image ever gets interpolated more than once, should bring about an improvement in both run-time and image quality. The answer whether this is true is given by the following measurements. As the functionality to swap label identifiers if necessary is not provided by the NiftyNet code base, it should be mentioned beforehand that this very feature has been implemented in the NiftyNet rand_flip-layer in order to be able to compare the two versions properly.

4.1 Quantitative layer evaluation

The following measurements are executed on a laptop PC containing an Intel Core i7-7500U[8] as CPU with a maximum frequency of 2.7 GHz and 8 GB of RAM. The laptop is always connected to a power supply unit and there are no other major processes running.

Affine transformations

As a first comparison, the time consumed by each single layer is measured. Both versions are given the same set of two 128x176x176 test images (volume and respective label map) and the same parameter-ranges:

- possible flip axes = $(0, 1)$
- scaling percentage = $(-8.0\%, 8.0\%)$
- uniform rotation angle = $(-5.0°, 5.0°)$

Uniform means that the range given applies to all three possible rotation axes x, y and z. The layers are initialised, the time needed for the transformation itself is measured 100 times and the average is taken.

		flip	flip + scale	flip + scale + rotation
1 flip	NiftyNet	0.029 s	1.498 s	2.944 s
	RTL	**0.021 s**	**1.480 s**	**1.419 s**
2 flips	NiftyNet	0.058 s	1.535 s	2.997 s
	RTL	**0.008 s**	**1.406 s**	**1.426 s**

Table 1: time needed by layers for different combinations of affine transformations, averaged over 100 iterations

The results of the measurements are listed in table 1. As expected, performing only one flip is much faster than performing any matrix multiplication due to NumPy's highly efficient flip method. This is because when doing a flip along one of the main axes, each new pixel position matches exactly a grid position and no interpolation is needed. The procedure is the same in both layers and therefore also the time is approximately the same. Also, both layers must execute their functions to swap the labels.

[8] `https://www.intel.com/content/www/us/en/products/processors/core/i7-processors/i7-7500u.html`

When adding a scaling factor, the flip method is not enough anymore and the procedure has to be changed in the RTL. Now scaling and flipping are performed together as one single matrix multiplication. NiftyNet for this operation has to use two different layers: one is the already in use rand_flip layer, the other is the rand_spatial_scaling layer which works with matrix multiplication, similar to the RTL. These two layers are executed one after the other and in the end a slightly bigger difference between the RTL and the NiftyNet implementation is visible than before. However, the difference is still very small and can be ignored.
Adding a third transformation forces NiftyNet to add a third layer. This new transformation requires the image to be rotated by certain angle, which again is performed by matrix multiplication and additional interpolation. As expected, the time required by the NiftyNet layers doubles. Meanwhile, the RTL only has to combine the two matrices, calculate multiplication and interpolation once and therefore the time remains approximately the same.
Requiring the layer to perform two flips, NiftyNet flips the image twice while still using only one layer. The label-map is then swapped twice as well, whereby the second swap cancels every change made by the first one, resulting in the same image as the original one again. In the meantime, the RTL already anticipates this and does not swap the label-map in the first place, which leads to a bigger relative time difference. Combining the two flips with one scaling and one rotation has the same effect as with one flip on both versions.

Elastic deformation

When elastic deformation is added, the method used by the RTL is different from the respective NiftyNet layer. Therefore, before combining both implementations with the affine transformations, they should be first compared separately. For this purpose, on the RTL all other transformations are deactivated and for NiftyNet only the rand_elastic_deform layer is used. The first column of table 2 shows the outcome of this comparison. The implementation in NiftyNet takes about a second longer to execute but the implementation in the random transformation layer is still slow compared to the affine transformations.

	elastic deformation only	all transformations
NiftyNet	3.857 s	6.263 s
RTL	**2.653 s**	**2.917 s**

Table 2: time needed by layers for elastic deformation, averaged over 100 iterations

When combining elastic deformation with affine transformations, the difference between the two versions gets even bigger. As NiftyNet must now stack all four different layers one after the other, the time consumed by each gets added to the total time. With RTL, only the multiplication by the affine matrix must be added. Although this multiplication must be performed by Cython, which is not as fast as NumPy, the additional time is much shorter than the time invested by three new layers.

4.2 Qualitative layer evaluation

Doing fewer interpolations should not only save time but also improve image quality. To check this, different image quality metrics can be analysed. As there is no original transformed image which could be compared to the outcome of the layer, the test consists of transforming an image and then applying reverse transformation, thus reproducing the original image. Since this test only considers the impact of interpolation, the actual form of transformation does not matter. Accordingly, an image gets first rotated by 15° and then rotated back by -15°. As the second part of the test, the original image is rotated three times by 5° and then again three times by -5°. As it always gets interpolated between the individual steps, the first image should have been interpolated two times, the second one six times. Then both images are compared to the original one by comparing the PSNR and the SSIM. Again, the 128x176x176 image serves as a test image. Since both RTL and NiftyNet generate random states for their transformations, they are not meant to be told the exact parameters of an transformation. Therefore the transformations are not performed within a layer but by the exact same implementation as found in the RTL and NiftyNet outside any layer. The comparison is performed twice, once with splines of order one, linear interpolation, and once with cubic splines.

spline order	metric	2x interpolated	6x interpolated
1	PSNR	**19.5390**	12.6760
	SSIM	**0.9790**	0.9161
3	PSNR	**35.5752**	28.9698
	SSIM	**0.9990**	0.9962

Table 3: image quality metrics on different amounts of interpolation

The values of both structural similarity and peak-signal-to-noise-ratio are higher for the twice interpolated images, independently of spline order. According to the scales for both metrics this means that the images produced with only two interpolations are more similar to the original one than the ones interpolated six times. Less information was lost. Looking at the structural similarity index, it seems that reducing the number of interpolations pays off more when using low interpolation orders as the difference between the two and the six times interpolated images is bigger there. As the transformation containing two interpolations represents the RTL and the other containing six the NiftyNet layers, this test shows that transforming images with the RTL keeps more information than performing the same transformation with the single NiftyNet layers.

4.3 Influence on network training

Quantitative

As the random transformation layer has been developed for training in neural networks, its behaviour in such an environment has to be tested as well. Therefore, an implementation of a 2D-network developed by Kodym et al. [7] is trained on the data set of the MICCAI challenge 2015 [18] with different preprocessing options. The network is trained on a Nvidia GTX 1080[9] with 8 GB of VRAM and CUDA[10] version 10.0.130, while the images are preprocessed on a Intel Core i7-6800K[11] with a frequency of 3.4 GHz on one thread. First it is trained while using original NiftyNet layers, once without elastic deformation and once with all augmentation methods active. Then the same network is trained while using the new RTL, again once without elastic deformation and afterwards with it.

	no elastic deformation	full augmentation
NiftyNet	20.07 h (241 s)	20.94 h (251 s)
RTL	**18.39 h (221 s)**	**20.21 h (243 s)**

Table 4: time needed to train networks with 300 epochs, the average epoch time in brackets

The first thing to note is the time required by the different configurations. According to table 2, in both configurations the random transformation layer should be faster than NiftyNet. As presented in table 4, RTL is faster. While the configuration with the RTL and all augmentation methods is only 3.5 % faster than NiftyNet, without elastic deformation it is about 8.4 % faster.
While the training itself takes place on the graphics processing unit (GPU), the augmentation process is executed by the central processing unit (CPU). Since the augmentation process must be finished before the image can be given to the GPU, it may happen that the GPU is not always fully utilised when waiting for new images. Consequently, the improvement in time made by the RTL over NiftyNet should also be visible when looking at the degree of utilization of the GPU. Therefore, for every configuration usage has been recorded for one minute during training and the average has been taken. The outcome is shown in table 5. In both configurations utilisation is about 4 % higher with RTL than NiftyNet.

	no elastic deformation	full augmentation
NiftyNet	21.89 %	21.32 %
RTL	**25.79 %**	**25.26 %**

Table 5: GPU usage during training

[9] https://www.nvidia.com/en-gb/geforce/10-series/
[10] https://developer.nvidia.com/cuda-zone
[11] https://ark.intel.com/content/www/us/en/ark/products/94189/intel-core-i7-6800k-processor-15m-cache-up-to-3-60-ghz.html

Qualitative

As shown in section 4.2 on qualitative layer evaluation, the images returned by the RTL have lost less information through interpolation than those returned by the different NiftyNet layers. This should have a positive effect on the outcome of the network. To analyse this, after training with the training data set, the network is tested with a test data set which the network has not seen before. Then the Dice coefficient [3] and the Hausdorff distance [5] are measured on the test data set with the network version of every fifth epoch of the training process.
The Dice coefficient is a metric for the equality of two samples. It is calculated by the formula:

$$\text{Dice coef.}(\mathbf{x},\mathbf{y}) = \frac{2 \cdot |\mathbf{x} \cap \mathbf{y}|}{|\mathbf{x}| + |\mathbf{y}|} \tag{30}$$

with $|\mathbf{x} \cap \mathbf{y}|$ being the amount of same pixels/voxels in the images $\mathbf{x}$ and $\mathbf{y}$, and $|\mathbf{x}| + |\mathbf{y}|$ the absolute number of pixels/voxels in both images accumulated. The Hausdorff distance on the other hand does not compare the whole image pixel by pixel, but the outer curves of the depicted objects. Analytically it can be described by:

$$\text{Hausdorff dist.}(\mathbf{x},\mathbf{y}) = \max\left\{ \max_{x \in \mathbf{x}} \left[\min_{y \in \mathbf{y}} (dist(x,y)) \right], \max_{y \in \mathbf{y}} \left[\min_{x \in \mathbf{x}} (dist(y,x)) \right] \right\} \tag{31}$$

with $dist()$ being the Euclidean distance between two points. So the Hausdorff distance for every point on a curve in image $\mathbf{x}$ searches the nearest point of the equivalent curve in image $\mathbf{y}$, calculates the distance between them and then takes the largest of all these distances. As this process is not symmetric (starting from a curve in image $\mathbf{x}$ might not return the same result as when starting from a curve in image $\mathbf{y}$), both directions are calculated, and the bigger value taken as the final Hausdorff distance. In practice, to prevent aberration from distorting the result, not the maximum distance of all distances is taken but the maximum distance of the lower 95 % of all distances.
The results of the network in which elastic deformation has been deactivated are visualized in fig. 6 and fig. 7. As elastic deformation is implemented differently from NiftyNet, comparing the networks with active elastic deformation would correspond to comparing the two implementations rather than comparing the preprocessing layers themselves.
In both graphs the RTL is more reliable than NiftyNet. The RTL converges faster in both metrics, is more constant with fewer aberrations and at the Haussdorff distance it has a lower final value than the NiftyNet network.

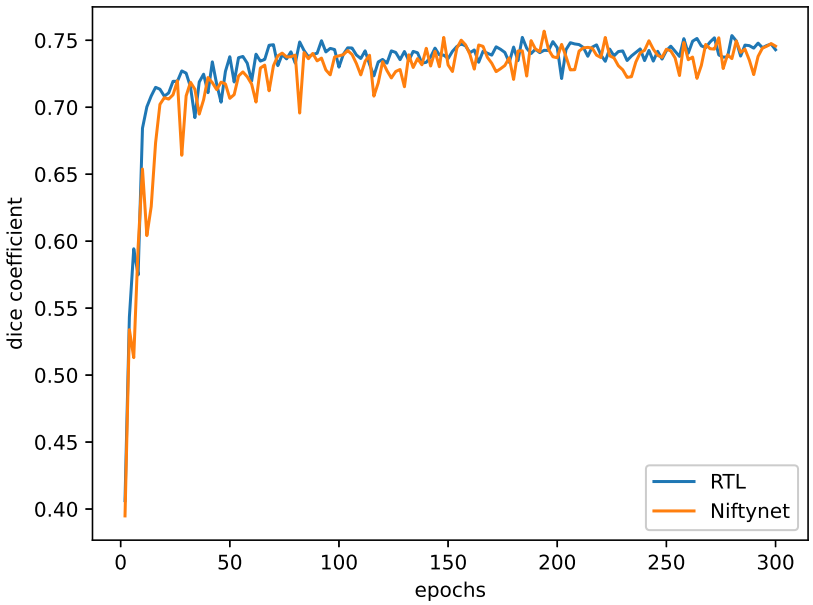


Figure 6: results of training without elastic deformation: Dice coefficient

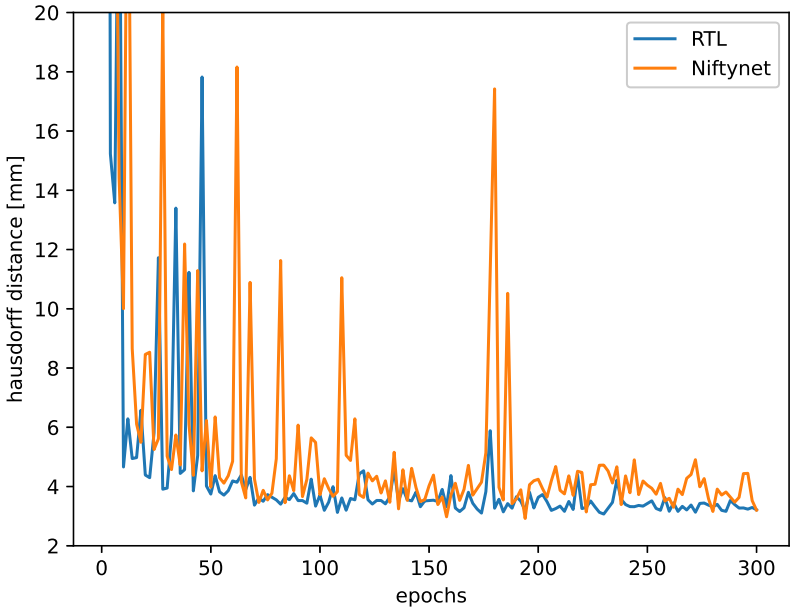


Figure 7: results of training without elastic deformation: Hausdorff distance

5 Conclusion

5.1 Layer quality evaluation

In order to optimise the augmentation process of images for artificial neural network training a new layer has been developed. In contrast to conventional libraries, as for example NiftyNet, the new layer combines all different types of transformations in one single preprocessing layer and reduces the amount of required interpolations applied on the images. It combines different affine transformations, such as scaling, flipping and rotating, but is also capable of applying elastic deformation onto the same image. It has therefore been denominated random transformation layer (RTL).
The comparisons made in section 4 show that the RTL does indeed have some advantages over the original NiftyNet layers. By reducing the amount of interpolations as well as optimizing the way elastic deformation are applied onto an image, a lot of time can be saved. This can be already seen when comparing single transformations but is best visible when the required time is halved when scaling and rotating the same image in one step. It also reduces the time by another second when applying elastic deformation on a single image.
The improvements brought about by the new layer, do not only refer to time but also to image quality, as less information gets lost. This has been shown by transforming an image in multiple steps, then reversing the transformation and comparing it to the original image.

5.2 Influence of the RTL on network training

The purpose of this new layer consists in improving the training behaviour of artificial neural networks. Therefore, the improvements noticeable when comparing the different layers regardless of any network should also be visible when making use of them to train a network. To this purpose, the two configurations (with and without elastic deformation) have been trained several times to determine the differences between NiftyNet and the RTL. Those can be seen above all when comparing the time required for training, as the random transformation layer takes more than 8 % less time to train a network than the NiftyNet layers do.
Next to time also improved image quality should have a positive impact on training. This is observed by comparing the Dice coefficient and the Hausdorff distance during the training process. But since on the one hand even the differences when comparing an image being interpolated two times with an image interpolated six times (see table 3) are not particularly distinct, and on the other hand both metrics are heavily fluctuating throughout the training process, no big differences regarding the Dice score are notable. In the Hausdorff distance graph differences can be seen more clearly, as the RTL graph shows less noise while training, leading to more stable end results regarding the Hausdorff distance. We hypothesize that this is due to smaller number of interpolations and the resulting higher image quality, affecting mainly small, only a few slices thick structures in the data set. Our hypothesis, however, needs to be evaluated in a future work.
Nevertheless, already by significantly reducing the time required for training, a huge step has been taken to further improve artificial neural networks based on NiftyNet.

5.3 Future work

As seen in section 4.2 where the layers have been compared in terms of quality, it is not possible to set up fixed parameters for transformation. Instead, parameters are chosen randomly from a given range. A further step could be to change the layer's handling of randomness in order to make the results repeatable. One of NiftyNet's successor MONAI's[12] improvements over NiftyNet is, indeed, reproducibility.
Despite the fact that Cython offers great advantages in time which are vital for the current form of the RTL with pure Python to be possible, there are some problems with its handling. Since the compiled binary files of Cython are hardware specific, the Cython file must be compiled every time it is used on a new computer. Therefore, a hardware-independent solution like other libraries such as NumPy or SciPy use, is requested.

[12] `https://monai.io/`

Bibliography

[1] Max K Agoston. *Computer graphics and geometric modeling*, volume 1. Springer, 2005.

[2] Eduardo Castro, Jaime S Cardoso, and Jose Costa Pereira. Elastic deformations for data augmentation in breast cancer mass detection. In *2018 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, pages 230–234. IEEE, 2018.

[3] Lee R Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[4] Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. Why does unsupervised pre-training help deep learning? In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 201–208, 2010.

[5] Felix Hausdorff. *Grundzüge der mengenlehre*, volume 7. von Veit, 1914.

[6] Alain Hore and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th international conference on pattern recognition*, pages 2366–2369. IEEE, 2010.

[7] Oldřich Kodym, Michal Španěl, and Adam Herout. Segmentation of head and neck organs at risk using cnn with batch dice loss. In *German Conference on Pattern Recognition*, pages 105–114. Springer, 2018.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Yann Lecun and Y. Bengio. Convolutional networks for images, speech, and time-series. 01 1995.

[10] Jörg Liesen and Volker Mehrmann. *Linear algebra*. Springer, 2015.

[11] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 fourth international conference on 3D vision (3DV)*, pages 565–571. IEEE, 2016.

[12] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010. URL `https://icml.cc/Conferences/2010/papers/432.pdf`.

[13] Stanislav Nikolov, Sam Blackwell, Ruheena Mendes, Jeffrey De Fauw, Clemens Meyer, Cían Hughes, Harry Askham, Bernardino Romera-Paredes, Alan Karthikesalingam, Carlton Chu, et al. Deep learning to achieve clinically applicable segmentation of head and neck anatomy for radiotherapy. *arXiv preprint arXiv:1809.04430*, 2018.

[14] Ikujirō Nonaka and Hirotaka Takeuchi. The knowledge-creating company. *Harvard business review*, 85(7/8):162, 2007.

[15] Karl Pearson. Vii. mathematical contributions to the theory of evolution.—iii. regression, heredity, and panmixia. *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, (187):253–318, 1896.

[16] Fábio Perez, Cristina Vasconcelos, Sandra Avila, and Eduardo Valle. Data augmentation for skin lesion analysis. In *OR 2.0 Context-Aware Operating Theaters, Computer Assisted Robotic Endoscopy, Clinical Image-Based Procedures, and Skin Image Analysis*, pages 303–311. Springer, 2018.

[17] Wilhelm Quade and Lothar Collatz. *Zur Interpolationstheorie der reellen periodischen Funktionen*. Verlag der Akademie der Wissenschaften, 1938.

[18] Patrik F Raudaschl, Paolo Zaffino, Gregory C Sharp, Maria Francesca Spadea, Antong Chen, Benoit M Dawant, Thomas Albrecht, Tobias Gass, Christoph Langguth, Marcel Lüthi, et al. Evaluation of segmentation methods on head and neck ct: auto-segmentation challenge 2015. *Medical physics*, 44(5):2020–2036, 2017.

[19] Hubert Reitterer and Wilfried Winkler. *Der kleine Stowasser*. G. Freytag, Munich 1980.

[20] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[21] Isaac Jacob Schoenberg. Contributions to the problem of approximation of equidistant data by analytic functions. part b. on the problem of osculatory interpolation. a second class of analytic approximation formulae. *Quarterly of Applied Mathematics*, 4(2), 1946.

[22] Oliver Schreer. Grundlagen der projektiven geometrie. *Stereoanalyse und Bildsynthese*, pages 9–36, 2005.

[23] Loren Arthur Schwarz. Non-rigid registration using free-form deformations. *Technische Universität München*, 2007.

[24] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[25] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, volume 3, 2003.

[26] Gerald Sommer. *Geometric computing with Clifford algebras: theoretical foundations and applications in computer vision and robotics*. Springer Science & Business Media, 2013.

[27] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[28] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017.

[29] Kim-Han Thung and Paramesran Raveendran. A survey of image quality measures. In *2009 international conference for technical postgraduates (TECHPOS)*, pages 1–4. IEEE, 2009.

[30] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 648–656, 2015.

[31] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[32] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):9, 2016.

[33] Jia Yonghong. Digital image processing (the second edition), 2010.

Verpflichtungs- und Einverständniserklärung

Ich erkläre, dass ich meine Bachelorarbeit selbständig verfasst und alle in ihr verwendeten Unterlagen, Hilfsmittel und die zugrunde gelegte Literatur genannt habe.
Ich nehme zur Kenntnis, dass auch bei auszugsweiser Veröffentlichung meiner Bachelorarbeit die Universität, das/die Institut/e und der/die Arbeitsbereich/e sowie die Leiterin bzw. der Leiter der Lehrveranstaltung, im Rahmen derer die Bachelorarbeit abgefasst wurde, zu nennen sind.
Ich nehme zur Kenntnis, dass meine Bachelorarbeit zur internen Dokumentation und Archivierung sowie zur Abgleichung mit der Plagiatssoftware elektronisch im Dateiformat pdf ohne Kennwortschutz bei der Leiterin bzw. beim Leiter der Lehrveranstaltung einzureichen ist, wobei auf die elektronisch archivierte Bachelorarbeit nur die Leiterin bzw. der Leiter der Lehrveranstaltung, im Rahmen derer die Bachelorarbeit abgefasst wurde, und das studienrechtliche Organ Zugriff haben.

Hall in Tirol am ....

Aaron Bacher