# Design of 3D-printed cranioplasty moulds using Neural Network

Jonathan Andersson

Master's thesis
2020:E86

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

LUND UNIVERSITY, FACULTY OF ENGINEERING

MASTER'S THESIS

Master of Science in Engineering, Engineering Mathematics

# Design of 3D-printed Cranioplasty moulds using Neural Network

*Author*
Jonathan Andersson

*Supervisor*
Einar Heiberg

*Co-Supervisor*
Finn Lennartsson

LUND UNIVERSITY

January 12, 2021

## Abstract

Cranioplasty is surgical repair of a skull bone defect due to a previous surgery or injury. Cranioplasty is most often performed with autologous bone flap, i.e the patient's own saved bone from previous surgery if this is available. If autologous bone is not available then custom procedure is to manually mould an implant using bone cement from the plastic polymethyl methacrylate (PMMA). Manually moulding implants during surgery has, albeit being a clinical routine, disadvantages and therefore Skåne University Hospital have developed a technique to 3D print patient-specific cranioplasty moulds based on a computed tomography (CT) scan of the skull bone. The shape of the mould is created by a combination of manual design process, mirroring and interpolation. This partly limits the technique to unilateral defects and the method can be tricky and time consuming for complicated cases.

The purpose of the thesis was to develop a method based on a neural network to reconstruct missing parts of the skull bone and overcome the limitations with the current mirroring method when designing implants or moulds for cranioplasty.

The process included developing a method to extract data from CT images for training of the neural networks. During the process numerous neural network structures and models were developed and evaluated with the best performing network being a convolutional autoencoder with skip connections. The network was trained with data from a total of 240 patients with simulated defects. The results of the network shows that it is able to handle both unilateral and bilateral defects with a mean error of 1.07mm. In comparison to the currently used method it performed as well or even better in some cases. Overall the developed method showed good enough results for it to be implemented as clinical routine.

**Acknowledgements**

First I would like to give a special thank you to my supervisor Einar Heiberg who had great patience with me and helped me during the whole process. You gave me good guidance throughout this project and made sure I had all resources and tools necessary. Your broad knowledge and your incredible drive is inspirational and working with you has been very educational.

Also a huge thank you to the whole Heart-MR Group at the Department of Clinical Physiology, Lund, and the Medviso team for providing me with necessary resources, a good work environment and being available for any kind of conversation. Especially to Finn Lennartsson who was the one originally collecting and ensuring the quality of all the raw data used for this project.

Finally a thank you to my friends, family and Marina for the support and encouragements throughout my studies and this project. A special thank you to my parents who made sure I could focus on the project, without them it would have been a lot tougher.

# Contents

# 1 Introduction

Cranioplasty is surgical repair of a skull bone defect due to a previous surgery or injury. There are a multitude of causes for skull defects or deformities. Examples of causes include the result of a trauma, tumor, infection, bone disease, or decompress surgery [1]. Cranioplasty can be traced far back in history as plates of gold and silver that have been found near the dead are believed to have been used as skull implants [2]. Nowadays cranioplasty is most often performed with autologous bone flaps i.e patient's own saved bone from previous surgery if this is available. If autologous bone is not available then, one could either use patient-specific implants in titan or other biocompatible materials such as hydroxyhepatite [3], or in the majority of cases an implant is moulded from bone cement and put in place during surgery. The bone cement is provided as a two-component material consisting of Polymethyl Methacrylate (PMMA) powder and a liquid that are mixed together before moulding. The casting of the bone cement during surgery is performed by placing a wet cloth on the exposed dura or arachnoid and applying the bone cement on top of the cloth and then manually moulding or shaping this to fit the void during the curation time which is approximately 10 minutes. This method has several drawbacks:

- The hardening process is exothermic and generates intensive heat that could damage surrounding tissue. Therefore it requires cooling which is done with sterile water.
- The hardening bone cement evaporates strong smells which affects the personal and can cause headaches.
- It may be difficult for the surgeon to get a good fit before the material hardens.
- If the implant has a sub optimal fit the cranioplasty flap needs to be grinded, spreading small dust particles in the operating theatre. As the PMMA bone cement is infused with the antibacterial substance gentamyecin, this becomes a work environment problem for the surgeon and the rest of the staff since the particles should not be inhaled.

Furthermore, up to 40% of all cranioplasties needs to be removed due to bleedings, infections or wound dehiscences [4].To address these problems a project was started last year at Skåne University Hospital [4]. The purpose of the project is to develop a method to create patient specific moulds for the implant based on a computed tomography (CT) scan. A CT scan is a medical imaging technique that reconstructs a volume from a set of X-ray projections. The different measurements are processed to produce cross-sectional images, so called slices of the object which makes it possible to visualize the inside of the skull. The signal from the X-ray is proportional to the tissue X-ray attenuation and is expressed in Hounsfield units (HU). It is a dimensionless unit that is obtained from a linear transformation of the measurements from the X-rays. It scales from -1000 HU for air to over 3000 HU for metals where 0 HU is the value for water [5]. This makes it possible to differentiate between different tissues in the CT image. The proposed method has a number of advantages compared to the old method:

- The casting of the implant can be done away from the patient instead of directly on the skull and in a location of the operating theatre with better ventilation.
- The surgery time can be shortened since one surgeon can work on the skull while another one is moulding the implant.
- The result can be done aesthetically better since the mould is created beforehand to a patient-specific shape ensuring a superior shape and less fine-tune grinding is required by the surgeon.

Currently the patient-specific mould design is based on mirroring shape from the non defect side to the defect side. In this process several manual corrections and adjustments are usually needed. After mirroring the healthy side to the defect side a number of points are placed around the inside and outside of the skull. Using interpolation between these points a template of the shape of the skull is created and from the template an implant is designed by filling the void of the hole between the inside and the outside surface at the defect area. A mould for the implant can then be designed from the template. The whole process is done with the software Segment 3DPrint[6] in Matlab[7]. Some of the steps for mirroring and creating the implant template can be seen in Fig. 1. The mould is 3D-printed on a Form3 3D printer (Formlabs Inc, MA, USA) using the Surgical Guide resin. Thereafter, the mould is sterilized in a clinical autoclave at 134°C. Since the skull is not perfectly symmetric the mirroring of the skull and the placement of the points for the template often requires manual corrections. Most importantly, the method can not handle defects that are bilateral, affecting both sides of the skull, i.e. when the defect is in the front or the back of the head as shown in Fig. 2.

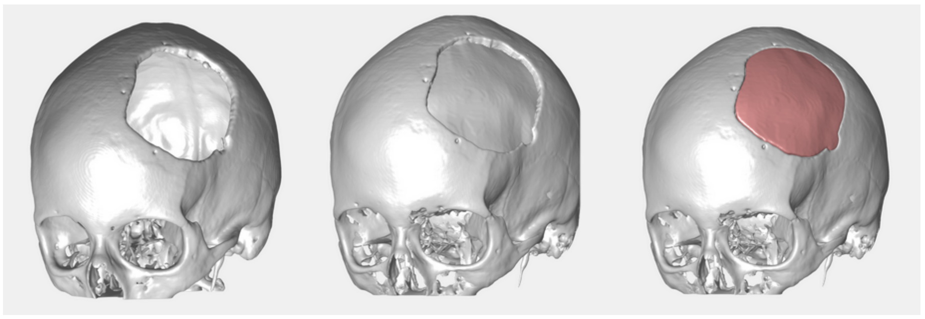


Figure 1: Steps for creating an implant. Left: Segmentation of the CT image. Middle: Mirrored and smoothed segmentation of the inside surface of the skull. Right: Design of implant by mirroring the outside of the skull and filling the void of the hole between the outside and inside surface. The mould is then designed by using the negative space of the implant.

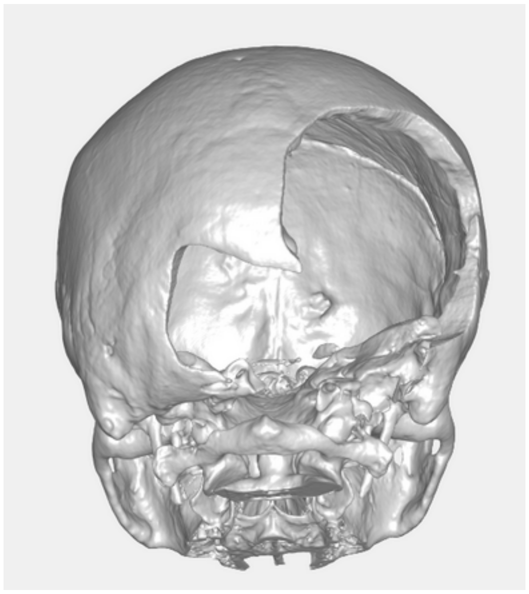


Figure 2: Subject with a large bilateral defect in the back of the head. This is an example where the current method of mirroring the defect does not work since the defect extends over the midline.

# 2 Aim

The aim of this master thesis project is to investigate the possibility to, instead of using the mirroring method currently used, use a neural network in the process to design implant moulds by using data from pre-operative CT images. Different network structures will be developed, evaluated and tested in the process to find a suitable network for the described problem. The network will be trained with data extracted from pre-operative CT images. The images will be pre-processed and a method to extract the data from the images will be developed. The data should be in a similar form as for the current method, i.e. points around the skull. For the training of the network the data will be manipulated to simulate defects in the skull. The aim for the network is then to be able to reconstruct the data as close to the original data before the defect was simulated and overcome the limitations with the current method.

# 3 Theory

## 3.1 Artificial Neural Networks

The artificial neural network (ANN) is loosely based on the biological neural network that can be found in the human brain. In a similar way as its biological counterpart the ANN is built up by neurons but in the artificial case they are called nodes. A neuron can simply be described as a switch. The switch receives inputs via connections, so called synapses, from other neurons and if a certain threshold of stimuli is reached the switch is activated. After the activation the neuron will in turn send out a pulse to other neurons [8]. A group of these neurons connected together is what makes up a neural network. The same design is found in the artificial nodes, as can be seen in Fig. 3. The nodes have inputs, an activation function and an output. A group of nodes that are connected together with links, corresponding to the biological synapses, creates an ANN. An ANN can then be trained to produce a desired output. The inputs to a node are either the outputs from previous nodes or, as in the case for the first layer of nodes, simply the inputs to the ANN. Each link has a different weight factor for the corresponding input which determines the effect the input has on the node and the node itself has a bias added as a scalar value. Both these parameters, the weight and bias, will continuously be updated during the training of the network to adjust the effect they will have on the final output and in that way minimize the error compared to the desired output. The node sums up the weighted inputs with the bias and the result is put into an activation function and depending on the result an output is sent out in a similar way as for the biological case when a neuron is activated and sends out a pulse. The output $y$ from a single node is given by

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{1}$$

where $f$ is the activation function, $x_i$ the inputs, $w_i$ the weights and $b$ the bias.

### 3.1.1 Perceptron

To easier understand the principles of an ANN let's look at one of the most basic one. It consists of only one node and is called a perceptron [9], the layout can be seen in Fig. 4. It is a binary classifier that, depending on if the sum of the weighted input and the bias is greater or less than 0, outputs 0 or 1. Following Eq. 1 the output $y$ of the perceptron is then defined as

$$y = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} w_i x_i + b \leq 0 \\ 1 & \text{if } \sum_{i=1}^{n} w_i x_i + b > 0 \end{cases} \tag{2}$$

where $x_i$ are the inputs, $w_i$ the weight corresponding to each input, $b$ the bias for the node. The activation function in this case is simply a step function.

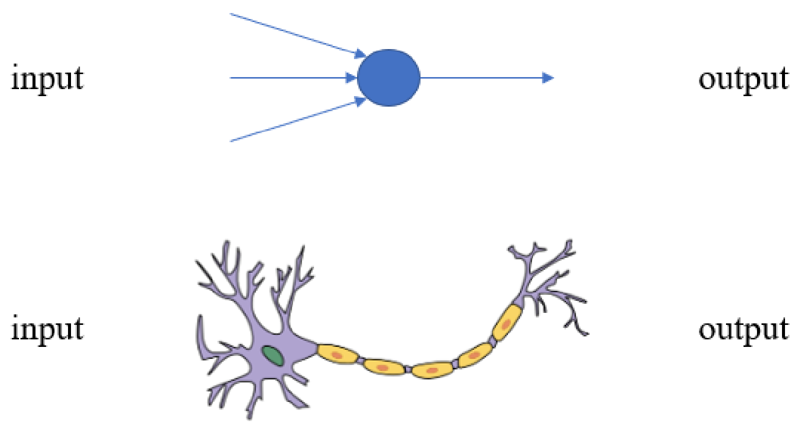


Figure 3: Visual comparison of a biological neuron and an artificial node.

Let's for example say the perceptron is used to decide how the weather is. If $y = 1$ the weather is good and if $y = 0$ the weather is bad. For inputs lets say that $x_1 = 1$ if the sun is out and $x_1 = 0$ if it is cloudy. Furthermore $x_2 = 1$ if it does not rain and $x_2 = 0$ if it rains. And last $x_3 = 1$ if it is windless and $x_3 = 0$ if it is windy. Now lets say that if it is sunny or not is not that important so $w_1 = 2$. However the rain and the wind plays a big part in deciding if the weather is good or bad so $w_2 = 5$ and $w_3 = 5$ i.e. more weight is put on these parameters compared to the one for the sun. Lastly the threshold for good weather is fairly high so $b = -7$. All these factors are summed up and passed to the activation function that in this case is activating depending on if the sum is greater or less than 0. The problem can be defined as in Eq. 2. If the sum is greater than 0 it is decided that the weather is good and vice versa. In this case it needs to be both windless and no rain for it to be considered good weather, the sun will not matter. By varying the weights and the bias different results can then be reached to find the model that best represents the desired criteria for good weather.

## 3.2 Activation functions

In the case with the perceptron the activation function used was the step function. This gives a binary result, the weather is either good or bad. Let's instead say that the value of how good the weather is should be on a scale from 0 to 1, then a sigmoid activation function can be used instead. It is defined as

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3}$$

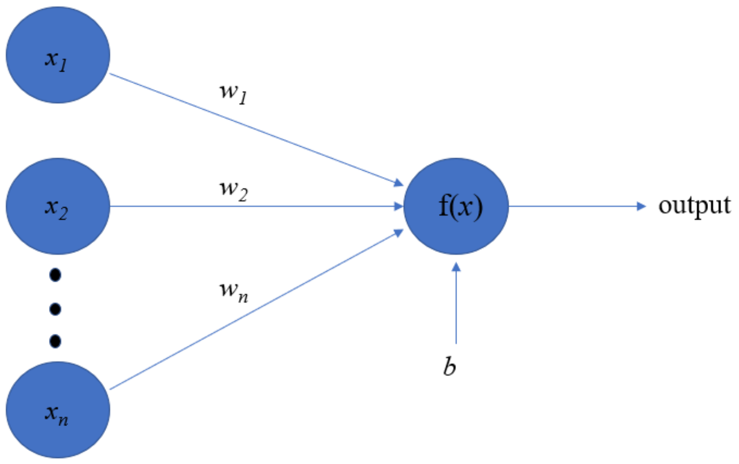


Figure 4: A schematic overview of a single node making up the basic perceptron.[9]

where $z \in \mathbb{R}$ and if we apply this to a node like in Eq. 1 we get the output from the sigmoid node as

$$y = \frac{1}{1 + e^{-\sum_{i=1}^{n} w_i x_i + b}} \tag{4}$$

where $y$ now, instead of being 0 or 1, takes a value between 0 and 1. So if we get a high value from our node we get $e^{-z} \approx 0$ so $\sigma(z) \approx 1$ meaning in the case of the weather classifier that the weather is very good.

There are a lot of different activation functions that are useful for different situations. Lets say instead that the node should output an intensity value for a pixel. Since the sigmoid function is limited to outputs between 0 and 1 it can not be used. Instead a commonly used activation function for such cases is the rectified linear unit (ReLU). It is defined as $f(z) = max(0, z)$ which for a node gives the output

$$y = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} w_i x_i + b \leq 0 \\ \sum_{i=1}^{n} w_i x_i + b & \text{if } \sum_{i=1}^{n} w_i x_i + b > 0 \end{cases} \tag{5}$$

## 3.3 Layers

The perceptron is the most basic model of a neural network. While it is not capable of doing complex tasks it gives an illustration on how a node works by weighing the inputs to produce an output. To be able to do more complex tasks multiple nodes are connected together to make up a network as can be seen in Fig. 5.

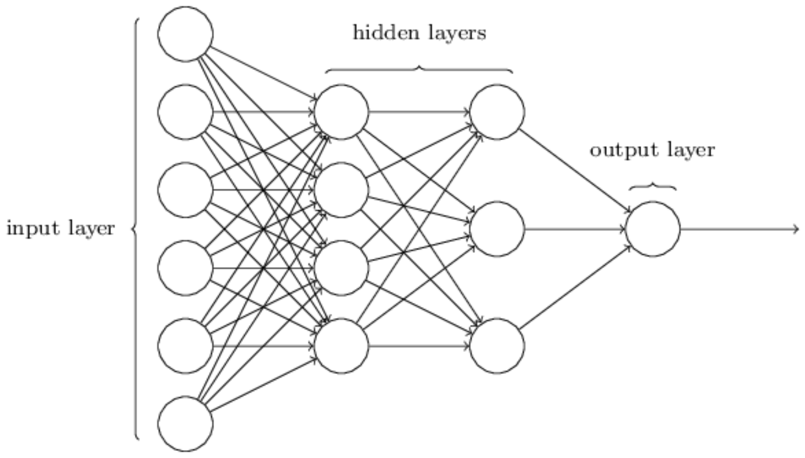


Figure 5: A schematic overview of multiple nodes making up an ANN.

Nodes that are lined up next to each other in a column are called a layer, e.g. in Fig. 5 there are 4 such columns representing 4 different layers. Nodes in a layer are not directly connected to each other and thus do not directly influence one another. They can however take in inputs from the same previous nodes and send out outputs to the same succeeding nodes. The first layer of a network is called the input layer and this is where the raw inputs enter the network. The last layer of the network is called the output layer and this is where the result of the entire network is output. In between the input and the output there are a number of layers called hidden layers. They are called "hidden" since they are neither input or output layers and thus not "visible" from outside of the network. Connecting multiple layers together creates an ANN and there are a lot of different layers with different properties that can be used depending on the purpose of the network. Some of those layers will be covered in the following sections.

### 3.3.1 Input Layer

The input layer is the first layer in an ANN and this is where the raw input data enters the network. The layer will be of the same size as the input data e.g. for an image of width 32, height 32 and 3 color channels the layer will be of size [32x32x3]. It is also common that the input data is normalized in this layer. Normalization of the data helps to speed up the training time by putting the input values roughly around the same range allowing for a faster learning rate[10]. A common normalization is the zero-center normalization where the values of the average image in the data set is subtracted from the image entering the network.[11]

### 3.3.2 Batch Normalization Layer

In the same way as it can be beneficial to normalize the input it can be beneficial to normalize the outputs from the activation layers before the data is passed on as inputs to the next layers. This makes it possible to use a higher learning rate and it reduces overfitting[12]. The normalization is done in the batch normalization layer by normalizing each input channel over a mini-batch of the data set. First the mini-batch mean, $\mu_B$ is calculated as

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{6}$$

where $m$ is the number of inputs and $x_i$ are the actual inputs. Then the mini-batch variance $\sigma_B$ is calculated with the mean $\mu_B$ from Eq. 6 as

$$\sigma_B = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{7}$$

Finally the normalized activations $\hat{x}_i$ are calculated with the mean $\mu_B$ from Eq. 6 and the variance $\sigma_B$ from Eq. 7 as

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{8}$$

where $\epsilon$ is a constant added to the mini-batch variance to improve the numerical stability. This can however effect what the following layer represent[12] so to avoid this the batch normalization layer scales and shift the normalized value $\hat{x}_i$ from Eq. 8 as

$$y_i = \gamma\hat{x}_i + \beta \tag{9}$$

where the offset $\beta$ and the scale factor $\gamma$ are parameters that are learned by the network during training. The batch normalization layer is often used right before a convolutional layer.[11]

### 3.3.3 Convolutional Layer

The convolutional layers main task in a network is to extract features from local and spatial dependencies in the input. In early layers of the network this could for example be edges or shapes and in the deeper layers of the network the features become more complex. The layer finds the features by applying sliding convolutional filters to the input and maps the result to a feature map. Convolution of two functions $f$ and $g$ is defined as

$$f * g = \int f(\tau)g(t-\tau)d\tau \tag{10}$$

In a discrete case of two dimensions, like for an image, and with a bias added this translates into

$$Y(i,j) = (I * K)_{ij} = B + \sum_{k=1}^{m}\sum_{l=1}^{n} K(k,l)I(i+k-1, j+l-1) \tag{11}$$

where $Y$ is the output, $K$ the filter, $I$ the input and $B$ the bias. The indices $i$ and $j$ are related to the output while $m$ and $n$ are related to the filter. In short the input is convolved by moving the filter horizontally and vertically over the image and computing the dot product of the input and the filter weights and then adding a bias. Each filter in a layer has its own weights and its own bias that is tuned during the training of the network. For the exact definition the filter is flipped along both axes before being applied, but this is irrelevant when it comes to the ANN. An example of how it can look is seen in Fig. 6.

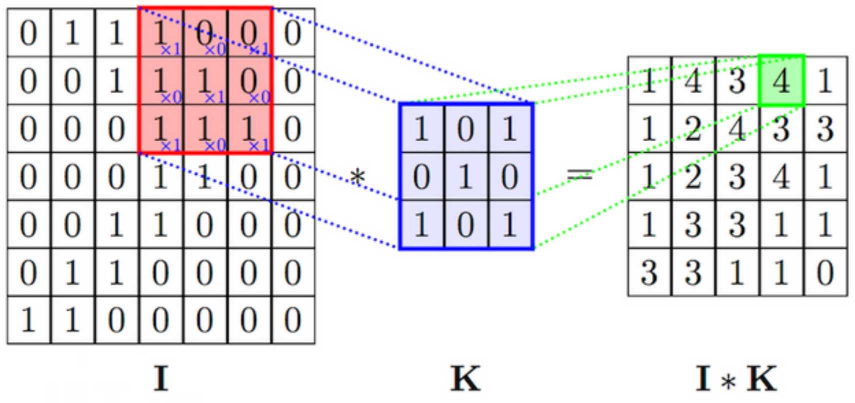


Figure 6: An example of a convolution filter being applied. **I** is the input image, **K** i the convolutional filter and **I*K** is the output feature map.[13]

In Fig. 6 we can see how the fourth entry in the first row of the output is calculated as $Y(1,4) = 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 = 4 \cdot$. The size of the filter is somewhat dependent on the size of the input. If the input is of size $m_1 \times n_1$ with $d_1$ channels the filter must have the same number of channels i.e.it must be of size $m_2 \times n_2 \times d_1$ where $m_2$ and $n_2$ can be selected independently of the input size. The size of the input together with the size of the filter decides the size of the output together with what is called the stride and the amount of zero-padding. The stride is what determines how the filter moves, i.e. with a stride of 2 instead of 1 the filter will move 2 steps over the image between each convolution instead of 1. An example of this can be seen in Fig. 7. Zero-padding is when zeros are added both horizontally and vertically at the borders of the input. In that way the output size can be controlled. An example of this can be seen in Fig. 8.

The output size of a filter can be calculated as $Out = (W - F + 2P)/S + 1$ where $W$ is the input volume size, $F$ the filter size, $S$ the stride and $P$ the amount of zero-padding. For example as in Fig. 8 where there is an input of size $5 \times 5$, filter size of $3 \times 3$, stride 1 and 1 line of zero-padding added, both vertically and horizontally, the output from a layer with 64 different filters is calculated as $Out = ((5-3+2)/1+1) \times ((5-3+2)/1+1) \times (64) = 5 \times 5 \times 64$.[11]

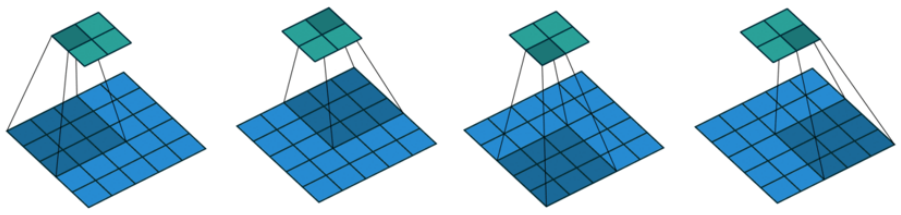


Figure 7: Filter with a stride of 2. The blue squares are the input, the dark blue area is where the filter is applied. The green squares are the output where the dark green square is the specific result from the convolution over the dark blue area. The result is a 2x2 output instead of a 3x3 as for stride 1.[14]

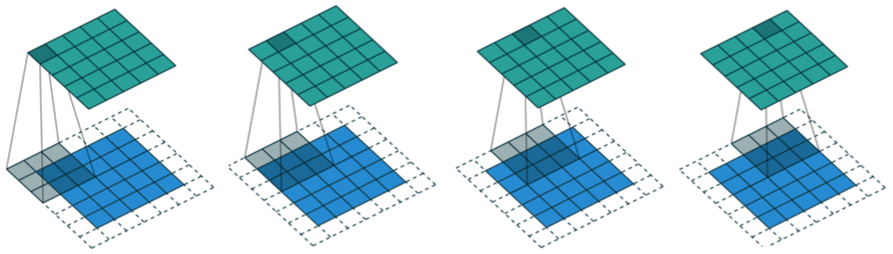


Figure 8: Input with one line zero-padding added around the borders. The blue squares are the input, the white squares are the zeros and the darker area is where the filter is applied. The green squares are the output where the dark green square is the specific result from the convolution over the dark area. This zero-padding increases the output size from 3x3 to 5x5, i.e. same size as the original input.[14]

### 3.3.4 Rectified Linear Unit Layer

The ReLU layer simply acts as an activation layer as it performs a threshold operation on the input. Each element of the input is run through an activation function similar to the one in Eq. 5. Any value less than zero is set to zero as

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{12}$$

The ReLU layer is positioned after the convolutional and the batch normalization layer. It does not affect the size of the data and has no learnable parameters.[11]

### 3.3.5 Max Pooling Layer

A max pooling layer consists of a filter that compares the values from an area of the input and outputs the max value from that specific area. Usually the main purpose of the max pooling layer is to down-sample the input and thus reduce the size of the data. In the same way as for the convolutional filter the max pooling filter slides both horizontally and vertically over the input and then outputs the max value of each region. The filter itself has no learnable parameters but depending on the filter size, the stride and the padding different output sizes can be achieved. The output size is calculated in the same way as for the convolutional layer. So for an input of size $W$, filter size of $F$, stride $S$ and padding $P$ the output size is given by $Out = (W - F + 2P)/S + 1$. An example of max pooling can be seen in Fig. 9.[11]

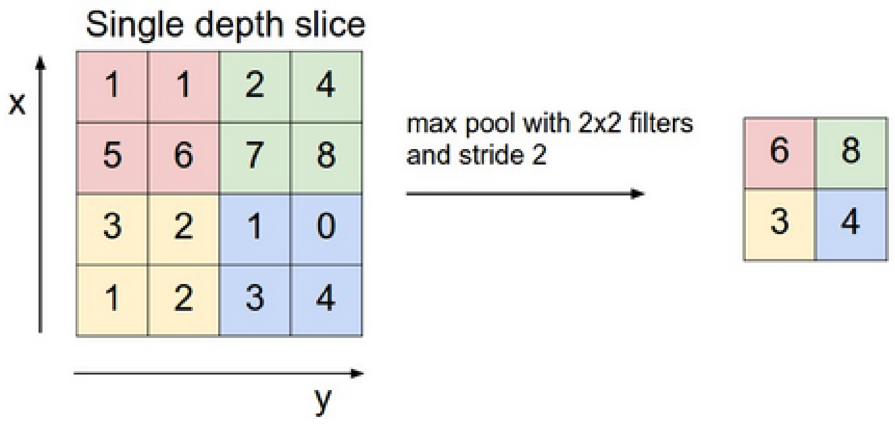


Figure 9: Max pooling filter of size 2x2 applied with stride 2 to input of size 4x4 resulting in a 2x2 output. The filter compares the values from each of the coloured areas and then outputs the largest value for each specific area.[15]

### 3.3.6 Transposed Convolutional Layer

The transposed convolutional layer is mainly used to upsample the input data of the layer. In a simplified way it can be described as a reversed convolutional layer. So instead of concatenating the information from the input the filter expands it, as in Fig. 10. A single value from the input is "expanded" and distributed to an area of the output.

The value from the input is multiplied with the weights of the filter and placed at the corresponding spots in the output. Just as for the convolutional layer the padding and the stride decides the size of the output, however the stride in the transposed convolutional layer is connected to the output instead of the input since every single value of the input should be run through the filter. Overlapping values in the output will then be added together, for there to be no overlap the stride must be equal to or larger than the size of the filter. An example of how the output is calculated and added up can be seen in Fig. 11. The output size can be calculated as $Out = (W - 1)S - 2P + (F - 1) + 1$ where $W$ is the input volume size, $F$ the filter size, $S$ the stride and $P$ the amount of zero-padding on the edges. The filters in the layer have learnable parameters in the form of weights and biases. The values for the parameters are learned during the training of the network.[11]

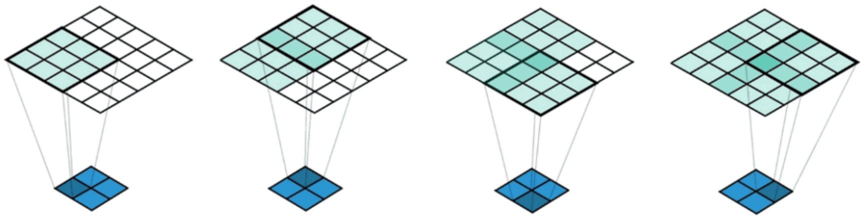
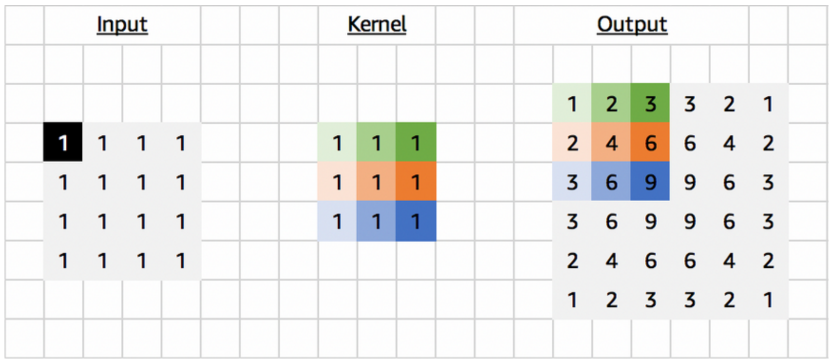


Figure 10: A 3x3 transposed convolutional filter applied to a 2x2 input with a stride of 2 resulting in a 5x5 output. The blue squares are the input with the darker one being the current value ran through the filter. White squares are the output where the green ones in the black frame are the output from the current input value. [16]



Input

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

Kernel

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

Output

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 3 | 2 | 1 |
| 2 | 4 | 6 | 6 | 4 | 2 |
| 3 | 6 | 9 | 9 | 6 | 3 |
| 3 | 6 | 9 | 9 | 6 | 3 |
| 2 | 4 | 6 | 6 | 4 | 2 |
| 1 | 2 | 3 | 3 | 2 | 1 |

Figure 11: Example of how the output is calculated and summed up in the output. A kernel of a 3x3 transposed convolutional filter applied to a 4x4 input with a stride of 1 resulting in a 6x6 output.[16]

### 3.3.7 Regression Layer

The regression layer is used as an output layer for regression problems i.e. when the output variable is a real or continuous value. It computes the half-mean-squared-error loss for each input to the layer compared to the target output.

For each output $y$ from the network and the target output $t$ the loss of the regression layer is calculated as

$$\text{loss} = \frac{1}{2} \sum_{p=1}^{HWC} (t_p - y_p)^2 \tag{13}$$

where $H$ is the height, $W$ the width and $C$ the number of channels of the output, and $p$ is the index for each element, i.e. pixel for an image-to-image regression network.[11]

## 3.4 Convolutional Neural Network

One of the most common types of ANN is the Convolutional Neural Network (CNN). The CNN is mostly used for computer vision and image analysis problems but also for any similar problems that have input data that have spatial or temporal dependencies. Usually the problems are some sort of classification or reconstruction/prediction problem. Just as the name indicates, the main feature of a CNN is the convolution. A CNN is simply an ANN with at least one convolutional layer. It consists of an input and an output layer and in between these there are a number of hidden layers usually consisting of a series of convolutional, activation (e.g. ReLU), pooling and normalization layers. In a way it can be said that the convolutional layer is used to find features and connections between the input data, the activation layer to decide which of the features are worth keeping, the pooling layer to downsample and enhance the features and the normalization layer to make the network more stable. Exactly what structure to use when it comes to specific layers, size of layers, number of layers etc. is dependent on what problem to solve. Usually there are a lot of different ways to go when it comes to the design that could all give satisfactory results. When it comes to data reconstruction a common type of CNN that is used is one called a convolutional autoencoder.

### 3.4.1 Convolutional Autoencoder

A convolutional autoencoder is a specific type of CNN where the input is first compressed and encoded and then reconstructed from the compressed representation to fit a desired output. In principle an autoencoder consists of 3 components: encoder, code and decoder. The encoder downsamples and encodes the data into the code in a bottleneck type of structure and the decoder then reconstructs the code to fit the desired output. In a convolutional autoencoder the encoding is often done with a combination of convolutional and maxpooling layers while the decoding is done with transposed convolutional layers. A typical structure of an encoder can be seen in Fig. 12.

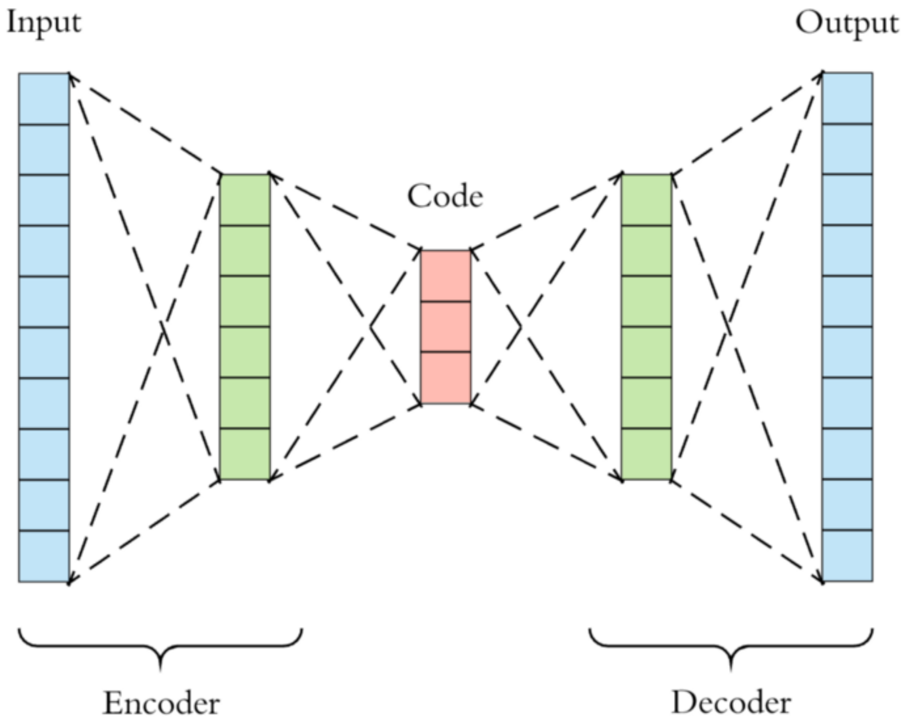


Figure 12: Typical structure of an encoder where the green bars represent a number of hidden layers.[17]

## 3.5 Training of a Network

Once the structure of the network is set the next step is to train the network for it to learn the optimal parameters i.e. the weights and biases for each different layer. Generally there are two types of learning when it comes to training a network, supervised and unsupervised. The difference between the two types of learning is that the supervised learning is done with a ground truth provided i.e. the network has knowledge on what the output should be. In that way the network can learn the optimal parameters to make the input match the desired output as good as possible by minimizing the loss. Supervised learning is typically used when it comes to classification and regression problems. Unsupervised learning on the other hand is done without any ground truth at all. It only uses the input to find structures within the data. The unsupervised learning is mostly used for clustering and dimensionality reduction.

### 3.5.1 Data sets

For the training and evaluation of the network the data set is often divided into three subsets: Training data, validation data and test data. The training data is typically the largest of the sets and is used during the training for the network to learn from and update its parameters. The validation data is used during training to validate the results from the training. It is not used for the training per say but to, during the training, keep track on how well the network performs on data that have not been used to update the parameters with. Thus it gives an unbiased evaluation of the network's performance which can be used to avoid overfitting. This is important since a common issue during training is for the network to become biased towards the training data by "memorising" it instead of learning general features and structures. The validation is done continuously during training in intervals for a set number of iterations. By comparing the loss of the training set to the loss of the validation set it can be assumed that the network is overfitting if the validation loss is diverging from the training loss. An example of this can be seen in Fig. 13. When the training of the networks is concluded the different results from the validation loss is used to select the network that performs the best. Even though the validation set is not directly used during the training, the result on the validation set is used when evaluating what network structure performs the best and to tune the hyper parameters. So as a final test for the selected network it is instead evaluated on the third subset, the test data. This set is completely unseen by the network and can give an unbiased result on the network's performance.

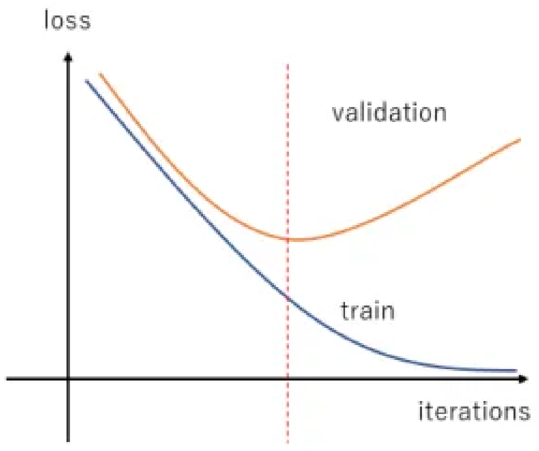


Figure 13: Visualization of overfitting during training of a network. The red dotted line shows the point of where the overfitting starts. This is where the loss of the validation data is starting to diverge from the loss of the training data. [18]

### 3.5.2 Hyper Parameters

Apart from all the parameter values that are learned during the training of the network, such as the weights and biases for the layers, there are also parameters that are set before the training. These parameters are called hyper parameters and will affect the structure of the network as well as how it behaves and learns. How to decide what hyper parameters to use when it comes to solvers, layers etc. is dependent on the problem, the data and the network. Even though there are some general guidelines on what to choose there are typically no clear answers to what the most optimal parameters are for specific cases. Instead it is common to do hyper parameter tuning to optimize the hyper parameters. The tuning could be done manually by trial and error where different parameters are tested and the results are observed. Another more sophisticated way to conduct the tuning is by doing what is called a grid search. Multiple values of each hyper parameter are lined up and all the different combinations of the parameters are systematically tested during the training to see which combination of parameters that yields the best result.

**Hyper Parameters related to the Network**
Once the general structure of the network is set there are a number of hyper parameters concerning the network structure that needs to be decided on. Primarily these are: How many filters there should be in each layer, what size the filters should have and also some parameters concerning the weights and biases that can be manually set for each layer. It is common to initialize the weights with independent samples from a uniform distribution. When it comes to deciding the size of the filters in the layers it depends on the structure of the data and where in the network the layers in question are placed. For an autoencoder it is common to have a larger sized convolutional filter in the early and late stage of the network and a smaller size in the middle. Typically the filter size varies between 3x3, 5x5 and 7x7 when it comes to images. For the pooling layer the most common size to use is 2x2 with stride 2, in that way the dimension of the data is reduced by a factor of 2 for each pooling layer.

When it comes to choosing the depth of the layers i.e. how many filters per layer, it depends on the complexity of the data and the problem that the network should solve. Too few layers will lead to the network not being able to find enough features and patterns in the data to solve the problem in a satisfactory way. And too many layers will lead to unnecessarily heavy computations and the possibility of overfitting which leads to bad generalization when it comes to different data.

**Hyper Parameters related to the training**
When it comes to the training of the network there are a few choices to be made and parameters to be set. The first choice is to decide what solver to use. The solver uses an optimization algorithm, typically gradient descent, which updates the network parameters by taking small steps in the direction of the negative gradient of the loss function and thereby minimising the loss. A variant of the

algorithm is called the stochastic gradient descent algorithm. Instead of using the entire training set at once to evaluate the gradients of the loss function, the stochastic gradient descent algorithm evaluates and updates the parameters using a subset of the training data. The subset is called a mini-batch and for each iteration of the algorithm a different mini-batch is used. The update of the parameters, i.e. the weights and biases, is done by backpropagation. For every iteration the gradients of the loss function are used to calculate the gradients of each layer using the chain rule, starting from the last layer and moving backwards in the network to the first layer. When all the training data have been used in different mini-batches the network have done what is called one epoch of training. Both the size of the mini-batches and maximum number of epochs to be used for training can be specified. The maximum number of epochs to be used depends on how quickly the network converges. By using too many epochs there is a risk of overfitting, while using too few may cause the network to not converge. An example of overfitting that occurred during hyper parameter tuning for a network can be seen in Fig. 14. The optimal size of the mini-batch depends on the data and has to be evaluated during training. Common choices of sizes are multiples of 32.

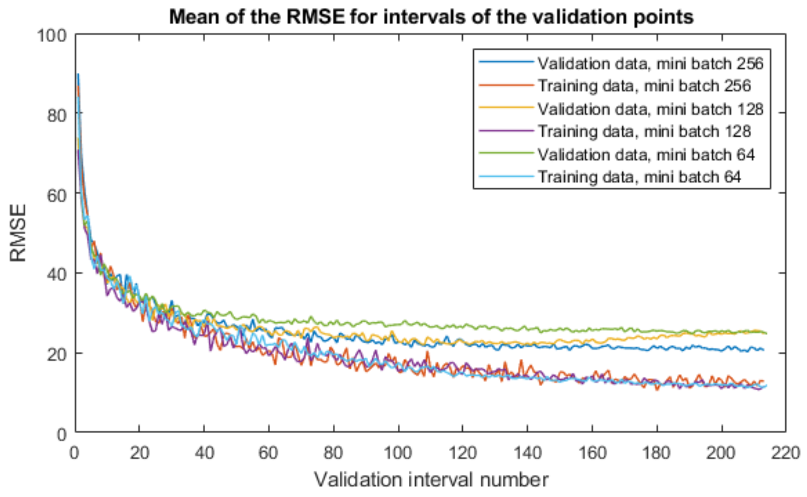


Figure 14: Example of overfitting during hyper parameter tuning of the mini batch size for the AES-CNN. It can be seen that the training with a mini batch size of 128 is overfitting since the validation root mean square error (RMSE, yellow line) is diverging from the training RMSE(purple line).

Another parameter that is important when it comes to training the network is the learning rate. The learning rate decides at what magnitude the network updates its parameters i.e. it specifies the step size of the gradient descent optimizer. Since the purpose of the optimizer is to minimize the loss of the network, it is searching for a global minimum and by having a too small learning rate the optimizer can get stuck in a local minimum or take too long to converge. While having a too large learning rate may result in missing the global minimum completely and the risk of not converging at all. An example of this can be seen in Fig. 15. A good alternative is to use a decaying learning rate so that after each epoch the learning rate is reduced by some factor. In that way the optimizer can start with a larger step size and then decrease the size as the network starts to converge to a minimum, this will hopefully speed up the convergence while keeping the accuracy.

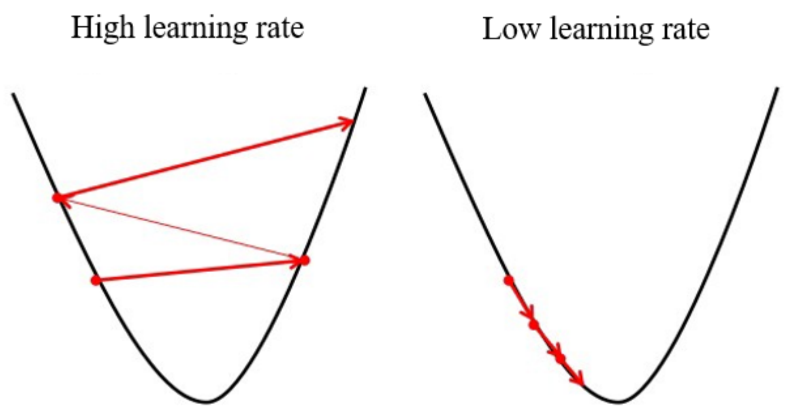


Figure 15: Visualization of the effect from different learning rates for gradient descent. On the left a too high learning rate causing the optimizer to take too large steps, missing the minimum and diverging. On the right a low learning rate, converging towards the minimum albeit a little slow. [19]

A common version of the stochastic gradient descent optimizer is the adaptive moment estimation (Adam) optimizer. It uses momentum in the algorithm and thus updating the weights more aggressively for steeper gradients. This increases the speed and accuracy of the optimizer and since it in that way regulates the learning rate by itself there is typically no need to manually set a learning rate decay.[20]

# 4 Material and Methods

## 4.1 Data

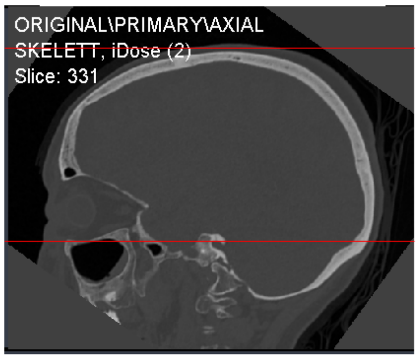
Since there were not a lot of cases of patients with skull defects that had CT scans of their healthy skulls it was hard to get enough data with good ground truths from real cases. The data was instead made up of CT images of none-defect skulls and the defects were manually created. In that way there were non defect versions of the skulls that could be used as ground truths and versions of the same skulls with simulated defects that could be used for training. The data for training the network was extracted from a set of CT scans with a total of 300 skulls. The CT scans were obtained from the routine clinical workflow at Skåne University Hospital in Lund. All CT scans were visually screened by a radiologist to exclude cases with cranial pathologies (e.g. bone diseases) or abnormalities (e.g. cases with large skull deformations). It consisted of 150 female and 150 male patients with birth years ranging from 1920 to 1999. Other than this there was no information available about the patients as the data was purposely and completely anonymized. All of the patients were considered healthy and had no major defects in their skulls. The images came in the form of DICOMs (Digital Imaging and Communications in Medicine) which is a standard format for medical imaging information and related data. Each patient scan consisted of around 330-430 slices of size 512x512 in the transverse plane.

Around the bottom of the orbita, i.e. under the eyes, the bone structure starts to become too complicated to produce an implant mould with an automated method like this. Furthermore, around the orbital region there are several surgical constraints that need to be taken into account for the implant design. It was therefore decided to limit the region of interest for the skulls in the project to the top of the skull down to the bottom of the orbita. In Fig. 16 the region of interest can be seen as well as the point where the bone structure is starting to become too complicated.

In order to get the data for the training of the network in the desired form each skull was loaded into Segment 3DPrint, resampled to be isotropic and then manually rotated to be in a strict axial orientation. The image stacks were then saved down to .mat files to be processed in Matlab.

### 4.1.1 Pre-processing

The image slices were represented in Matlab as stacks of 515x512 images with pixel values in HU. In order to segment the bone the images were thresholded to values above 150 HU, this removed all the soft tissues and left a segmented binary image. Since there were some components of the CT scanner visible in some of the images only the largest component, i.e. the skull, was kept in each image. The segmented binary images were then processed with a morphological operation, consisting of a dilation followed by an erosion. This smoothened the surface of the skull and removed protruding pixels. An example of a processed image is seen in Fig. 17. It somewhat distorted the area around the face but



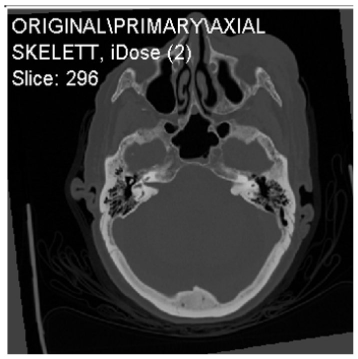


Figure 16: To the left: The region of interest in the sagittal plane is seen in between the red lines. To the right: The corresponding slice of the bottom red line seen in the transverse plane. Here we can see how complicated the bone structure starts to become.

since that is an area which the moulds are not intended for, it will have no effect on the final results. The morphological operation ensured that the data points could be placed around the skull without any outliers. It also ensured that holes around the orbita area were "closed". This was important because these holes were used to decide the region of interest since it was discovered that the structure started to become too complex around the point at where there were 4 of these holes. In that way the region of interest could automatically be set for each skull. This can be seen in Fig. 17.

### 4.1.2 Data points

From each morphological processed image the center of the skull was calculated and from the center 40 equally distributed lines were drawn, i.e. with an equal angle between all the lines. For each line all the skull pixels that were crossed were listed and the pixel furthest from the center point was selected as in Fig. 18. In that way 40 points around the outer edge of the segmented skull were selected with the first one being straight down from the center, i.e. starting in the neck and then moving clockwise around the skull. The Cartesian coordinates for these points were then saved in 2x40 matrices where the first row represented the X-coordinates of the points and the second row the Y-coordinates. These matrices were the raw data that was used to train the networks. In average there were 229 slices in the region of interest for each skull making up the total data set of 68625 slices.

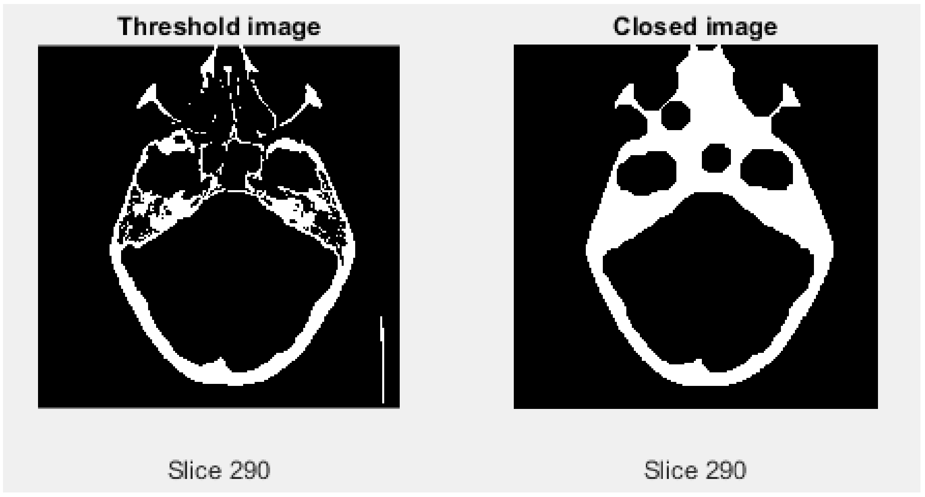


Figure 17: To the left: Example of segmented binary image that has been thresholded. To the right: The same image after morphological processing. The smooth surface enables good data point placements and the resulting 4 holes were used to automatically decide the region of interest.

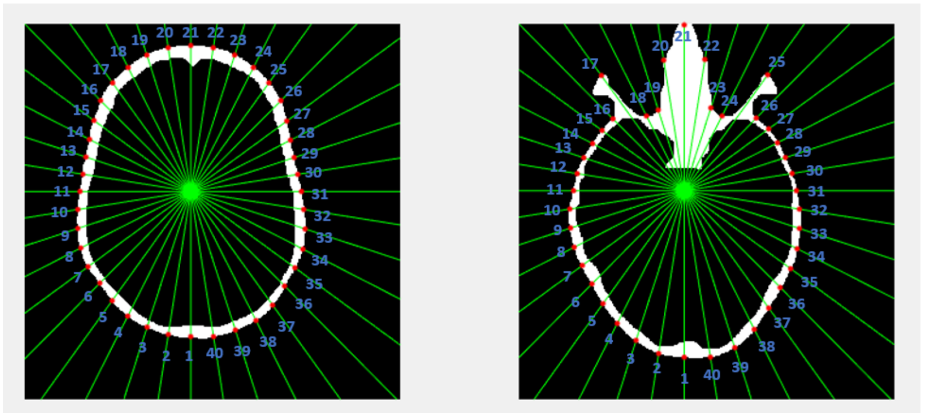


Figure 18: Two examples of how the data points of the skulls were extracted from the images.

### 4.1.3 Simulated defects

To simulate defects on the skulls in the data, a patch of consecutive points were removed to create artificial holes. The size of the holes were randomly assigned to between 1-20 points for each slice. Points were 'removed' by setting the coordinates to zero and in 1/10 of the holes there were also a small patch of points added since not all defects for real cases can be represented with convex hulls. It was avoided to make holes extending over the entire face since the method is not meant to be used for that area. Examples of the artificial holes can be seen in Fig. 19. The data from the skulls with the holes were used as input and the data from the skulls without holes was used as the ground truth when training the networks.

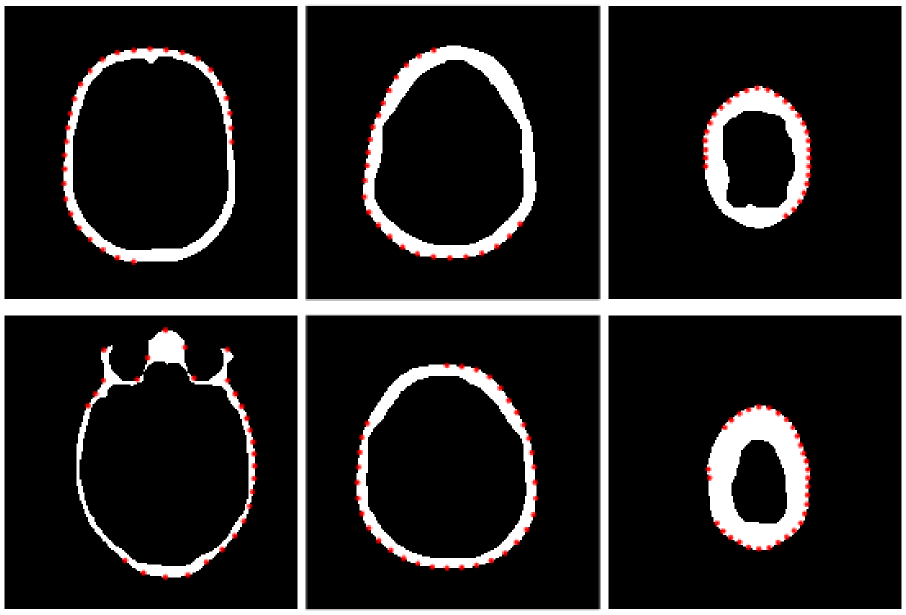


Figure 19: Examples of cases where the artificial holes were made by removing patches of data points. The points are plotted over the real skulls to show the unaltered shapes.

### 4.1.4 Data sets

Since the first point was geometrically close to the last point there was a circular dependency in the data. Because of this it was also tested to pad the start of the data with the last 20 data points and the end of the data with the 20 first data points. This resulted in a second data set with 2x80 matrices whose results from the network would be compared to those of the 2x40. For the 2x80 data set the network was trained with the 2x80 matrices and the result was then evaluated with the networks estimation of only the 40 original points.

The data sets were split up into three parts. A training set of 240 skulls with a total of 54668 slices used during the training of the network. A validation set of 30 skulls with a total of 6987 slices used during the training to validate the training process. And a test set of 30 skulls with a total of 6970 slices used to evaluate the trained network. There was no patient overlap between the sets.

## 4.2 Neural Network design and training

During the project numerous different network structures have been tested and evaluated. Since no previous work on this exact problem formulation could be found the structures were gradually developed and evaluated from scratch with inspiration from commonly used structures in other similar problems. In addition to the final network this report will briefly look at 2 of the networks that were tried out on the path to the final design to show the progress along the way. The networks were optimized using half mean squared error loss with the Adam solver. Pre-processing and training was performed in Matlab, and the network was trained on a local compute-server with a 22-core 2.37GHz Intel Xeon Gold 6152 and dual NVIDIA Titan RTX graphics cards.

### 4.2.1 5C-CNN: Basic CNN

Even though the data did not consist of images, the structure of the data was similar to images in the way that the matrix entries have spatial dependencies since the coordinates next to each other in the matrix represents data points close to each other in the image slice. This made a CNN a good starting structure since it is a commonly used network when it comes to image analysis. One of the earliest structures that was tested was a simple CNN, named 5C-CNN. The structure consisted of 5 convolutional layers of sizes 2x7, 2x5 and 2x3. The sizes were based on the size of the input data being 2x80 or 2x40 and the dimensions of the data was kept the same apart from the depth which was increased by a factor of 2 for each layer. The convolutional layers were applied with stride 1 and with added padding to keep the dimensions. The exact size and depth of the layers were tuned by trial and error. Following each of the convolutional layers there was an ReLU activation layer and a batch normalization layer. There was no structured hyper parameter tuning, only trial and error since the network did not perform good enough. The structure of the network can be seen in Fig. 20. The version of the network that gave best results was trained for 400 epochs with mini batch size 256 and initial learn rate 0.01. An extract of the training progress can be seen in Fig. 21. The network was first trained on the 2x40 set but after evaluation of the results the 2x80 set was created. Due to the enhanced performance of the 2x80 set this was the only network that was trained on both of the sets, the other ones were only trained on the 2x80.

### 4.2.2 AE-CNN: Convolutional autoencoder

Moving on from the 5C-CNN it was clear that a more complex structure was needed. A commonly used structure when it comes to image reconstruction and de-noising is the autoencoder structure. By encoding the data the network puts more focus on the overall structures which could be beneficial when it comes to reconstruction of data. The network was named AE-CNN and the structure consisted of an encoder with 3 blocks of convolutional, batch normalization, ReLU and maxpooling layers that downsampled the data. Following the encoder

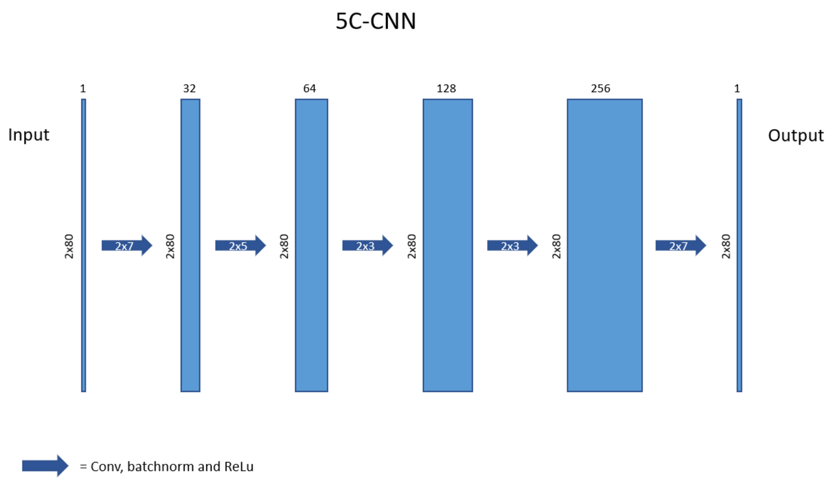


Figure 20: Structure of the 5C-CNN.

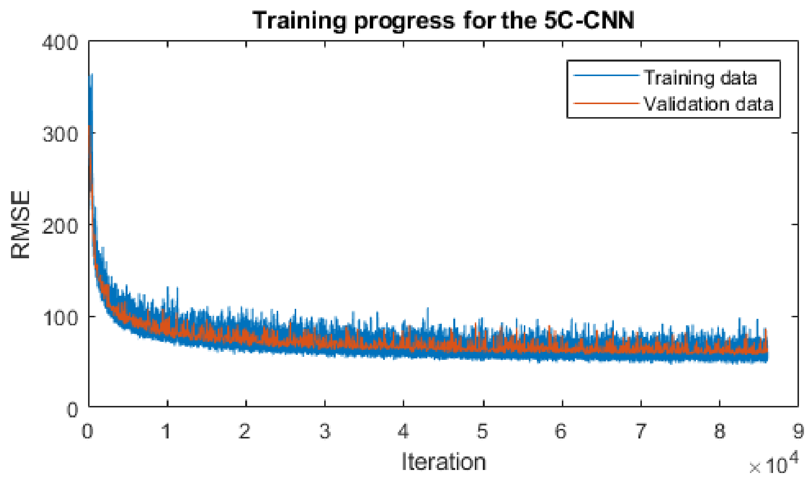


Figure 21: Training progress for the 5C-CNN on the 2x80 data set, the initial loss was so high that the first iterations were excluded from the plot to show the convergence better.

there was a decoder that consisted of 3 blocks with transposed convolutional, batch normalization and ReLU layers that upsampled the data. At the end of the network there was one last convolutional layer to adjust the data to the correct output dimension. The convolutional layers varied between 2x7, 2x5 and 2x3 with the larger ones in the beginning of the network and the smaller ones in the middle. They were applied with stride 1 and added padding to keep the dimensions of the data. Each convolutional layer increased the depth of the data by a factor of 2. The maxpooling layers were of size 2x2 and were applied with stride 2 and added padding to not reduce the first dimension of the data but to reduced the second dimension of the data by a factor of two i.e. for the 2x80 data set in to 2x40, 2x20, 2x10. The transposed convolutional layers also varied between 2x7, 2x5 and 2x3 with the smaller ones in the middle of the network and the larger ones in the end. They were applied with a stride of [1 2] and cropping to not change the first dimension of the data but to upsample the second dimension of the data with a factor of 2 i.e. from 2x10 to 2x20, 2x40 and 2x80. After each transposed convolutional layer the depth of the data was reduced by a factor of 2. Even though the Adam solver somewhat adjusts the learning rate it was found out during the evaluations of the training that a drop factor of the learn rate made the convergence of the network smoother. Furthermore the depth of the layers and the size of the mini batch were tuned with a grid search for mini batch size [64 128 256] and initial filter depth [8 16 32 64]. The final structure of the AE-CNN can be seen in Fig. 22. The best version of the network was trained with initial learn rate 0.01 and a drop factor of 0.9925 ending in a learn rate of 0.001 for 300 epochs, mini batch size of 128 and initial filter depth of 32. An extract of the training progress can be seen in Fig. 23.

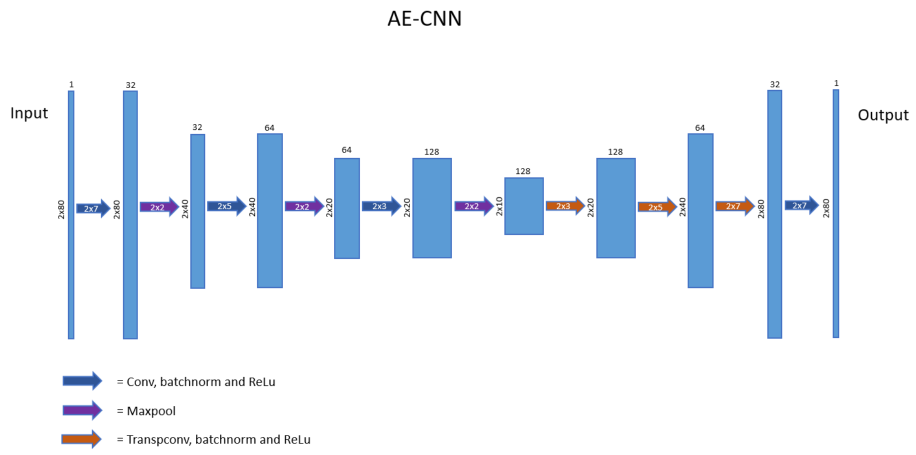


Figure 22: Structure of the AE-CNN.

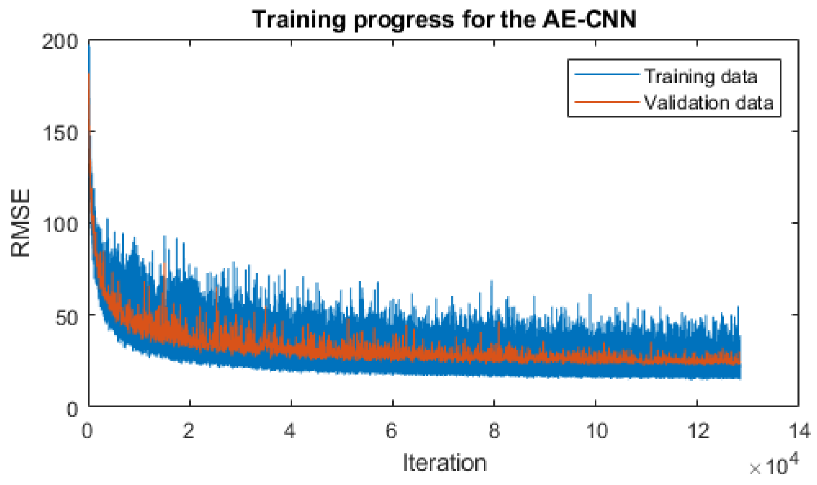


Figure 23: Training progress for the AE-CNN, the initial loss was so high that the first iterations were excluded from the plot to show the convergence better.

### 4.2.3 AES-CNN: Convolutional autoencoder with skip connections

Building on the structure of the AE-CNN the next significant change was to add skip connections between the encoder and the decoder. The idea was inspired by a well known network called the UNET[21] and builds upon feed forwarding a copy of the data before each downsample in the encoder to the corresponding place in the decoder. In that way any important structure and information is passed on and not lost during the downsample. The new network was named AES-CNN and had in a similar way as the AE-CNN blocks with convolutional, batch normalization and ReLU layers followed by downsampling with a max-pooling layer in the encoder. Then 3 blocks of transposed convolution, batch normalization and ReLU layers to upsample the data in the decoder. In this structure each transposed convolutional block was also followed by a convolutional block and the network also ended with one last convolutional layer to adjust the data to the correct output dimension. The convolutional layers varied between 2x7, 2x5 and 2x3 with the larger ones in the beginning of the network and the smaller ones in the middle. They were applied with stride 1 and added padding to keep the dimensions of the data apart from the depth of the data that was increased by a factor of 2 for each layer in the encoder. The maxpooling layers were of size 2x2 and were applied with stride 2 and added padding to only reduce the second dimension of the data by a factor of two, just as in the AE-CNN. The transposed convolutional layers varied between 2x7, 2x5 and 2x3 with the smaller ones in the middle of the network and the larger ones in the end. They were applied with a stride of [1 2] and cropping to only upsample the second dimension of the data with a factor of 2, also just as for the AE-CNN. Each transposed convolutional layer also reduced the depth of the data by a factor of 2. The final structure of the AES-CNN can be seen in Fig. 24. The mini batch sizes [64 128 256] and the learning rate drop factors [0.99 0.9925 0.995] were tuned with a grid search. Also the number of epochs was tuned to avoid problems with overfitting. The best version of the network was trained with initial learn rate 0.01 and a drop factor of 0.9925 for 500 epochs, mini batch size of 64 and initial filter depth of 32. An extract of the training progress can be seen in Fig. 25.

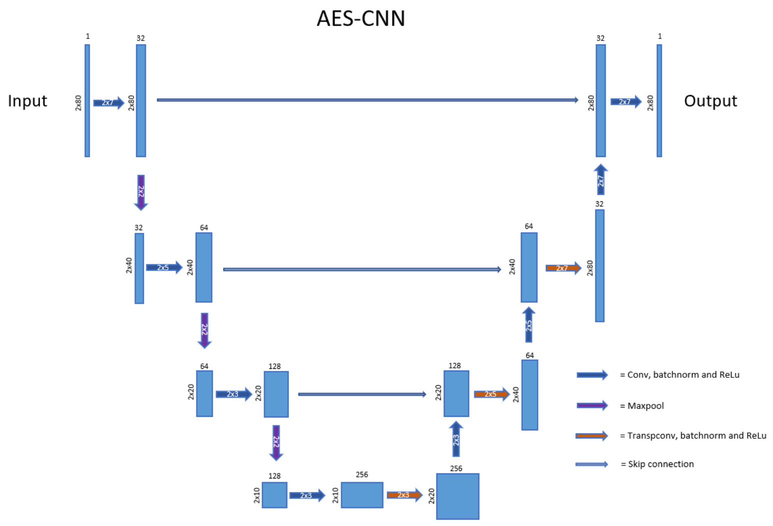


Figure 24: Structure of the AES-CNN.

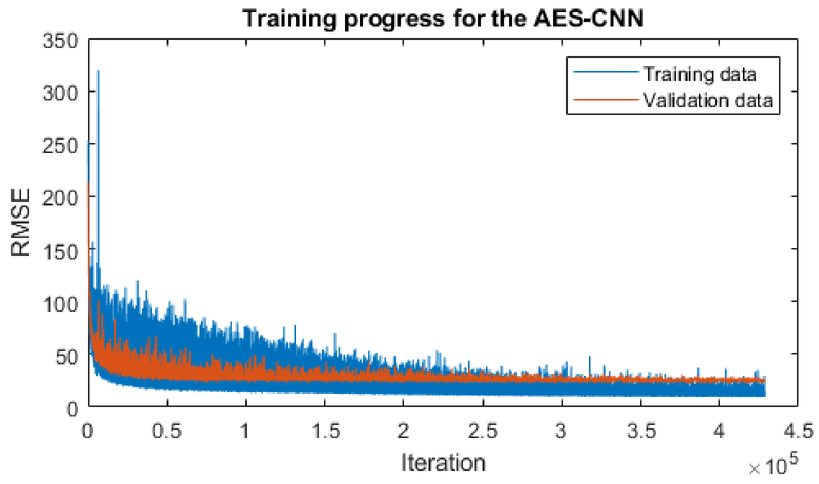


Figure 25: Training progress for the AES-CNN, the initial loss was so high that the first iterations were excluded from the plot to show the convergence better.

## 4.3 Evaluation

During the development of the networks the training loss was not the only result being taken into consideration. In order to understand how the networks actually performed at estimating the missing points and what problems they had there were evaluations done on the validation set. The evaluations included measuring the error between estimated and real points and visually investigating bad results. Since the data set was padded and the training was done by minimizing each matrix entry, i.e. each x and y coordinate and not the spatial distance between each point, it was hard to draw any conclusions on only the loss during training. The loss was calculated for all the points while the only points that really mattered were the missing ones. So after training the networks they were used on the validation set slice by slice for the 30 skulls, in total 6874 slices. The estimations of the missing points were visualised on top of the CT scans to investigate the results. The euclidean distance between the estimated points and the actual points from the ground truth were calculated and used for the evaluation. All the results presented will be based on the estimation of the missing points.

In order to see how the network performed in comparison to the current mirroring and interpolation method a few "real" cases were artificially created from a non defect skull in Segment 3DPrint in Matlab. In that way there were cases that could be mirrored by an operator at the hospital by the standards currently set and then the results could be compared to those from the network. The "real" cases can be seen in Fig. 34. The first case is with a hole on the front of the skull, second case a hole on the back of the skull and the third case a hole on the side of the skull. Since the current method with mirroring is not possible to use in case 1 and 2 the operator manually had to create the reconstructions with the help of subjective design and interpolation.

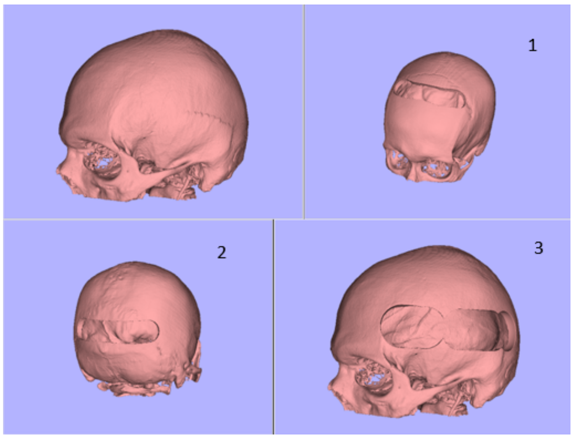


Figure 26: Artificial holes made in Segment 3DPrint on a non defect skull to simulate real cases. In the top left corner a view of the original skull, the numbers on the other skulls represents the case numbers.

# 5 Results

## 5.1 Data set comparison

As discussed previously the only network that was trained on both the 2x40 and 2x80 set was the 5C-CNN. The reason to include the result of this is to simply highlight the reasoning behind only using the 2x80 set for the other networks. The networks were trained on the two different data sets and then evaluated on the validation set. In Table 1 the result on the two different sets is presented and in Fig. 27 a representation of the result for each point. It can be seen that both networks perform bad with the 2x80 set being slightly better. In particular in Fig. 27 the 2x80 performs a lot better then the 2x40 when it comes to the first and last points. In Fig. 28 an example of the result from the 5C-CNN trained on the 2x40 set compared to the one trained on the 2x80 set highlighting the difference in performance that was seen in Fig. 27.

Table 1: Percentages of the missing points that were estimated with less than 2 and 4mm as well as the mean error distance of the missing points. Results are from the validation part of both the data sets on the 5C-CNN.

| Data set comparison | <2mm | <4mm | Mean error |
|---|---|---|---|
| 2x40 | 42.4% | 72.3% | 3.66mm |
| 2x80 | 48.0% | 79.0% | 2.85mm |

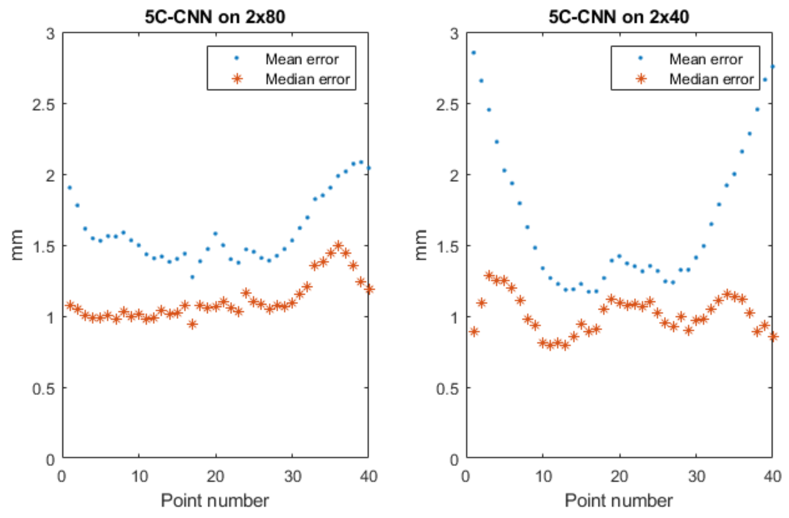


Figure 27: Total mean and median error for all the estimations of the missing points in the two different validation sets. Point numbers 1-40 represents the different locations of the skull from where the points were taken. It can be seen how the 2x40 set has significantly larger errors at the first and last points.

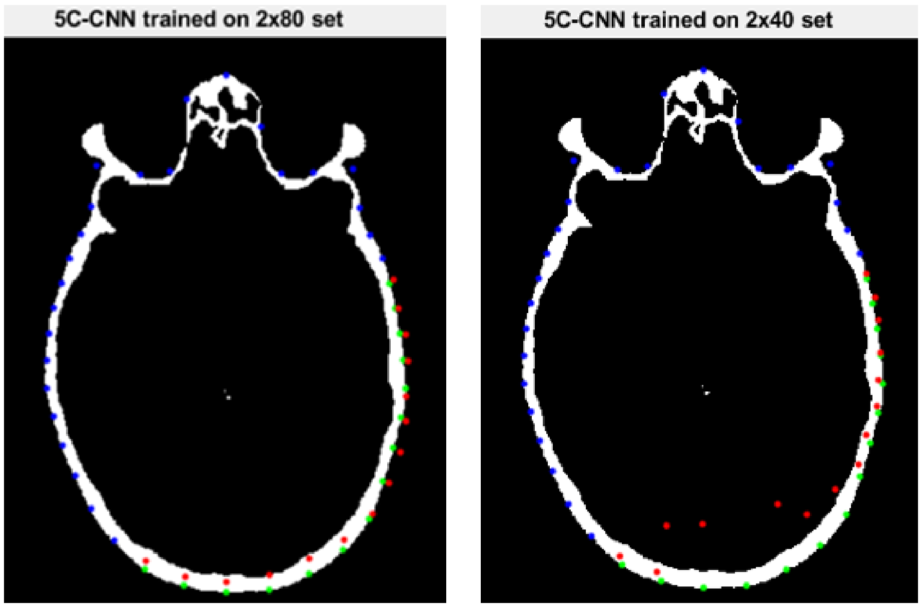


Figure 28: Example of the difference in result from the 5C-CNN trained on the 2x40 and the 2x80 data set. Blue dots are the original points used as input, green dots are the simulated hole, i.e. the missing points, and the red dots are the networks estimation for the missing points.

## 5.2 Network comparison

The best versions of each network structure was compared to each other. All the networks were evaluated on the validation set of the 2x80 data. It can be seen in Fig. 29 how they compared during training. The AE-CNN and the AES-CNN converged to around the same loss for the validation set while the training loss was lower for AES-CNN. They both had lower losses compared to the 5C-CNN in general. In Table 1 it can be seen how they all performed when estimating all the missing points for the validation set. Fig. 30 shows the mean of the total errors for each point around the skull for all three networks. Both the AE-CNN and the AES-CNN performed better than the 5C-CNN. The difference between the AE-CNN and the AES-CNN was small but the AES-CNN performed slightly better overall. A visualized example of an estimation of a large hole for each of the three networks is shown in Fig. 31.

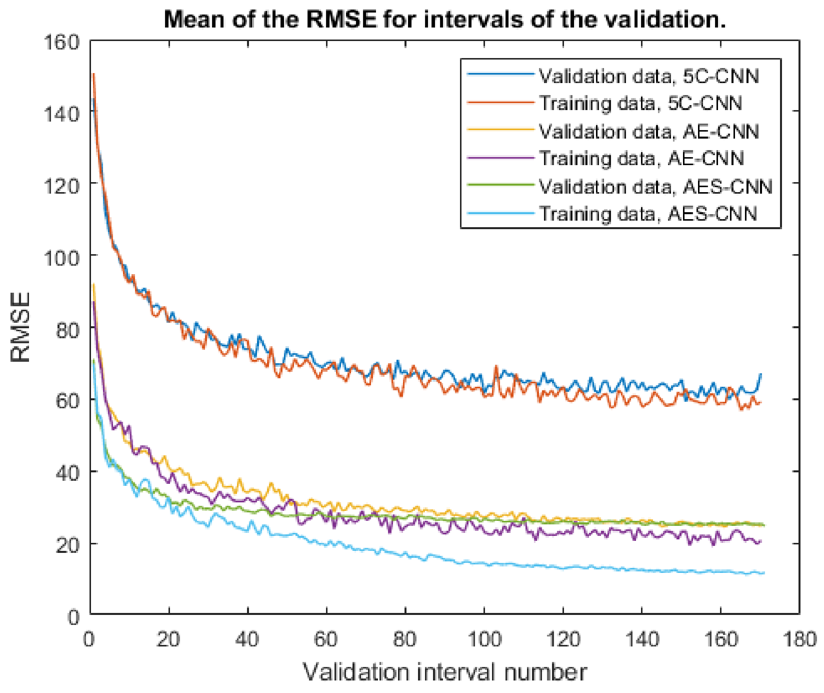


Figure 29: Training progress for the three different network structures shown as mean of the RMSE from Fig. 21, Fig. 23 and Fig. 25. The means are normalized over a certain number of validations for each network to make it easier to overview and compare.

Table 2: Percentages of the missing points that were estimated with less than 2 and 4mm as well as the mean error distance of all the missing points. Results are from the validation data in the 2x80 set.

| Network comparison | <2mm | <4mm | Mean error |
|---|---|---|---|
| 5C-CNN | 48.0% | 79.0% | 2.85mm |
| AE-CNN | 86.8% | 98.5% | 1.12mm |
| AES-CNN | 87.1% | 98.5% | 1.07mm |

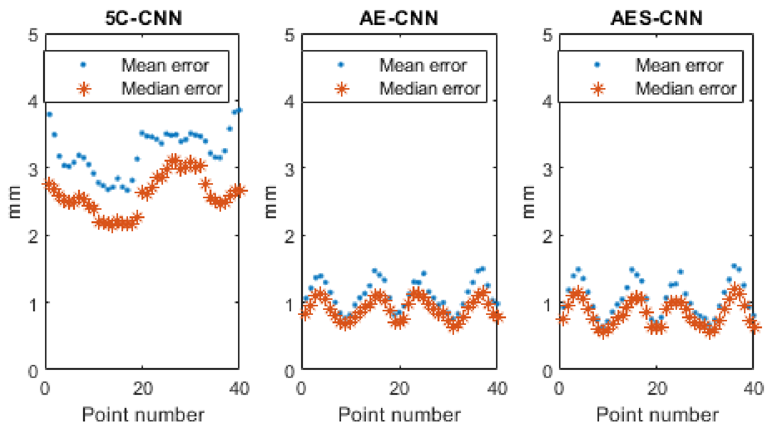


Figure 30: Total mean and median error for all the estimations of the missing points in the 2x80 validation set for all the three networks. Point numbers 1-40 represents the different locations of the skull from where the points were taken.

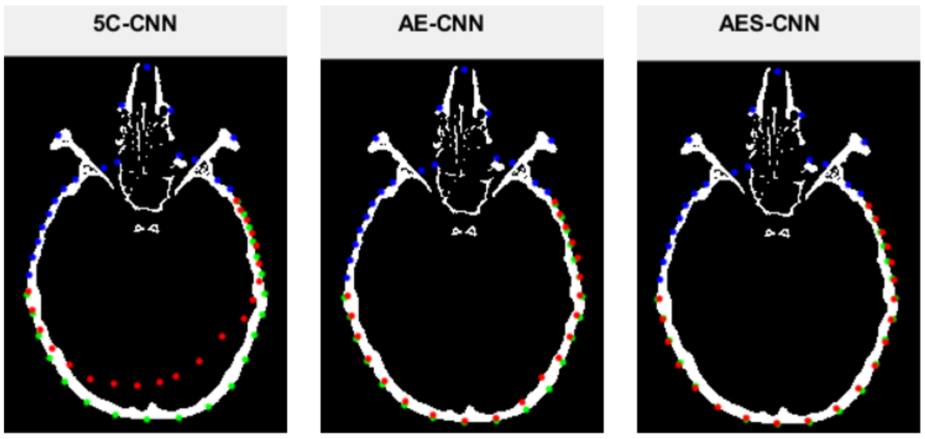


Figure 31: A visualized example from the validation set of estimation of missing points for the different networks on a slice with a large patch of missing points. Blue dots are the original points used as input, green dots are the simulated hole i.e. the missing points and the red dots are the networks estimation for the missing points.

## 5.3 Evaluation on test set for AES-CNN

Since the AES-CNN showed the best performance out of all the networks evaluated on the validation set it was in the end also evaluated on the test set. Fig. 32 shows a few random typical examples of how the reconstructions looked like. As can be seen in Table 3 the performance on the test set more or less matched the one on the validation set seen in Table 2. In Fig. 33 the mean and median errors for each specific location of the estimated points are shown. The plot shows that the errors of the points located on the "corners" of the skull are a bit higher than the points located at the posterio-lateral aspects of the skull. In the Table 4 the difference in performance when it comes to the size of the hole for the missing points. Smaller holes are estimated with a smaller error compared to the large ones. Furthermore, the total mean error per point when it comes to each slice can be seen in Table 5 showing that a very high percent of slices have a small mean error. The slices with the largest mean errors per estimated points can be seen in Fig. 34.

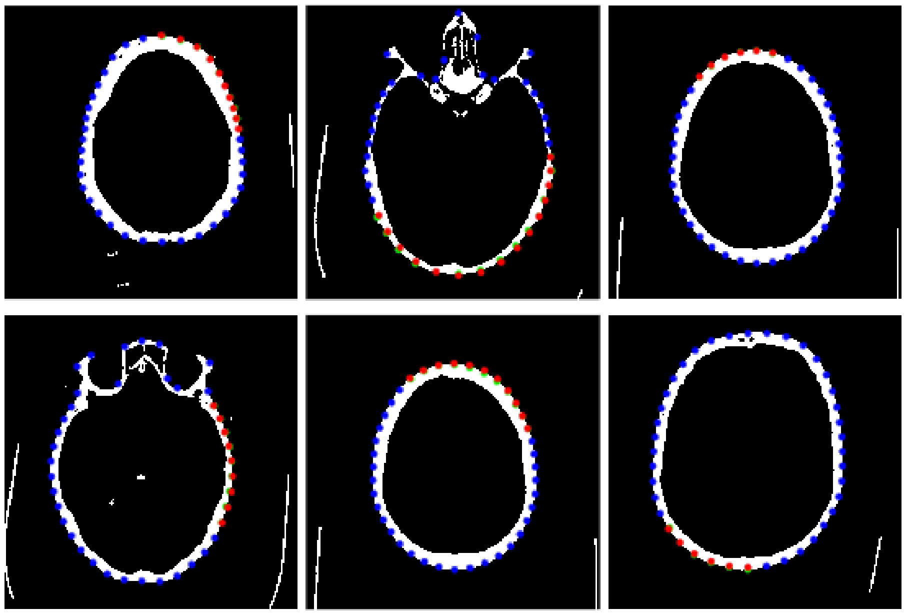


Figure 32: Typical random examples of estimations of missing points for the AES-CNN on the 2x80 test set. Blue dots are the original points used as input, green dots are the missing points and red dots are the networks estimation.

Table 3: Percentages of the missing points that were estimated with less than 2 and 4mm as well as the mean error distance of the missing points. Results are from the test data in the 2x80 set.

| Test set | <2mm | <4mm | Mean error |
|---|---|---|---|
| AES-CNN | 86.79% | 98.58% | 1.07mm |

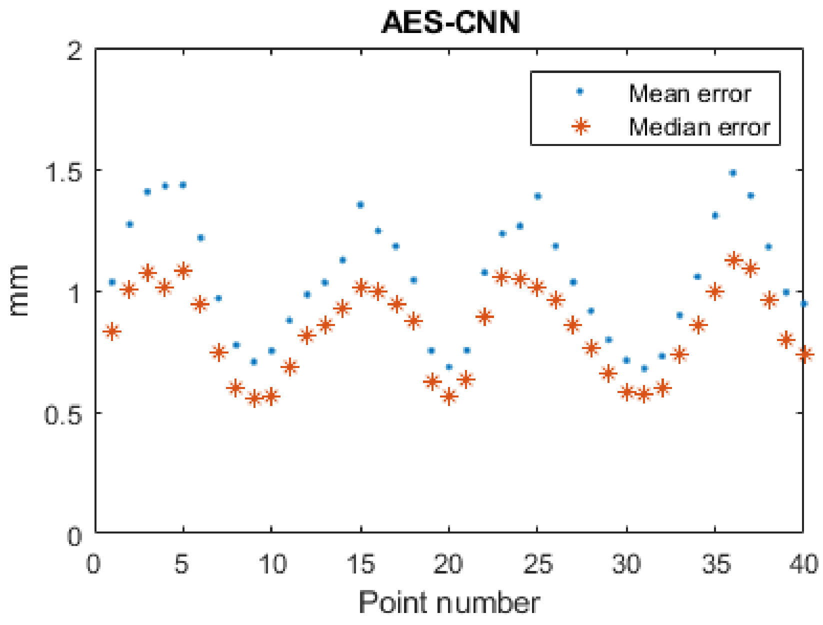


Figure 33: Total mean and median error for all the estimations of the missing points in the 2x80 test set. Point numbers 1-40 represents the different locations of the skull from where the points were taken. It can be seen how the errors are higher on the "corners" of the skull and lower on the sides, front and back.

Table 4: Percentages of the missing points that were estimated with less than 2mm error and the mean error for different sizes of missing points. Results are from the test data in the 2x80 set.

| Size of hole | 1-5 points | 6-10 points | 11-15 points | 15-20 points |
|---|---|---|---|---|
| Mean error mm | 0.80 | 0.98 | 1.08 | 1.16 |
| Percent <2mm | 95.01% | 89.68% | 86.57% | 84.08% |
| Percent <4mm | 99.73% | 99.19% | 98.64% | 98.09% |

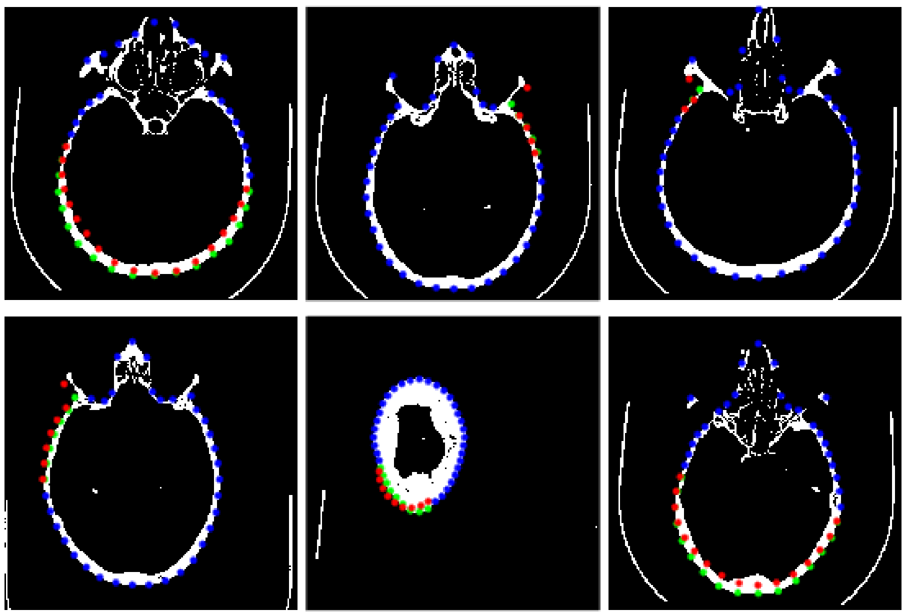


Figure 34: The slices with the highest mean error per point estimated.

Table 5: Percentages of slices in the test set with mean error per point less than 2 respectively 4 mm. Results are from the 6875 slices in the 2x80 test set.

| Mean error per slice | <2mm | <4mm |
|---|---|---|
| AES-CNN | 94.76% | 99.96% |

## 5.4 Comparison to current mirroring method

The results from case 1 are listed in Table 6 and an example of reconstructions of a slice can be seen in Fig. 35. The network had a lower mean error and also a higher number of total points with error <2mm. Neither of the methods had any errors >4mm. The results from case 2 can be seen in Table 7 and an example of reconstructions of a slice can be seen in Fig. 36. The network had a slightly higher mean error and a lower number of total points with error <2mm. Neither of the methods had any errors >4mm. Finally the results from case 3 are seen in Table 8 and an example of reconstructions of a slice from the case can be seen in Fig. 37. The network had a significantly lower mean error and a higher number of total points with error <2mm. However neither of the methods had any errors >4mm.

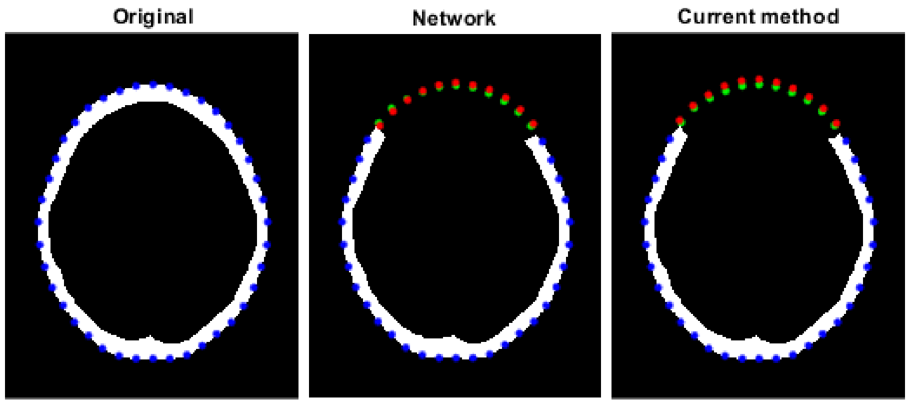


Figure 35: Example of reconstruction with the network compared to the current method. To the left the original whole skull, in the middle the networks estimation of missing points and to the right the points from the current method. Blue dots are the original points, green dots are the missing points and red dots are the estimations.

Table 6: Percentages of the missing points that were estimated with less than 2 and 4mm as well as the mean error distance of the missing points. The results are from the first artificial case with a hole on the front of the head which covers 33 slices in total. The slice with the largest hole was missing 9 points.

| Case 1 | <2mm | <4mm | Mean error |
|---|---|---|---|
| AES-CNN | 66.67% | 100% | 1.69mm |
| Mirroring | 45.24% | 100% | 2.06mm |

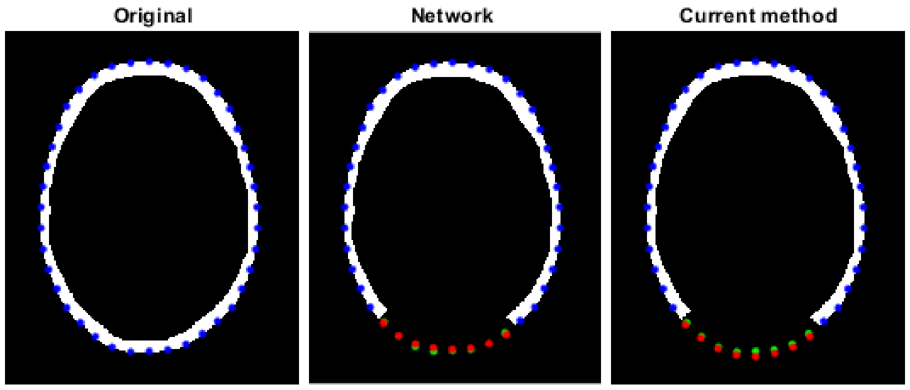


Figure 36: Example of reconstruction with the network compared to the current method. To the left the original whole skull, in the middle the networks estimation of missing points and to the right the points from the current method. Blue dots are the original points, green dots are the missing points and red dots are the estimations.

Table 7: Percentages of the missing points that were estimated with less than 2 and 4mm as well as the mean error distance of the missing points. The results are from the second artificial case with a hole on the back of the head which covers 51 slices in total. The slice with the largest hole was missing 9 points.

| Case 2 | <2mm | <4mm | Mean error |
|---|---|---|---|
| AES-CNN | 87.10% | 100% | 1.16mm |
| Mirroring | 93.43% | 100% | 0.92mm |

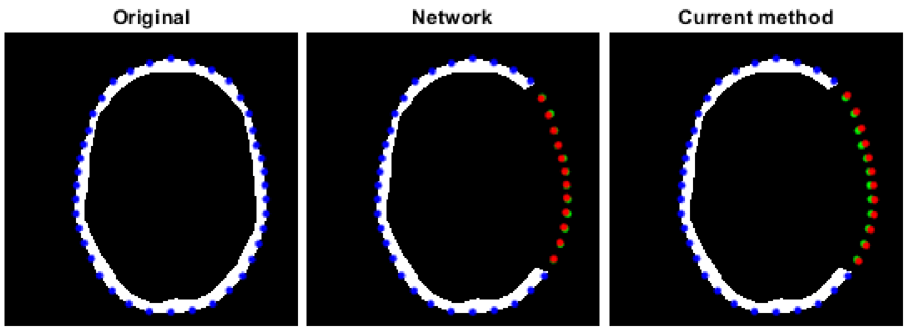


Figure 37: Example of reconstruction with the network compared to the current method. To the left the original whole skull, in the middle the networks estimation of missing points and to the right the points from the current method. Blue dots are the original points, green dots are the missing points and red dots are the estimations.

Table 8: Percentages of the missing points that were estimated with less than 2 and 4mm as well as the mean error distance of the missing points. The results are from the third artificial case with a hole on the side of the head which covers 81 slices in total. The slice with the largest hole was missing 12 points.

| Case 3 | <2mm | <4mm | Mean error |
|---|---|---|---|
| AES-CNN | 90.25% | 100% | 1.16mm |
| Mirroring | 25.62% | 100% | 2.35mm |

# 6 Discussion

The presented work is a proof-of-concept for using neural networks for skull reconstruction and design of cranioplasty moulds and implants. It was shown that the network had overall good results as it could handle varying defects all around the skull with relatively small errors.

**Data set**
Looking at the results of the two different data sets, 2x40 and 2x80, it was shown that it was beneficial to in some way capture the circular dependency of the data originating from the input being the coordinates of points around the skull. The 2x80 set had much better results compared to the 2x40 set as seen in Fig. 27 and Fig. 28. In particular around the back of the head where the data extraction started and ended for each slice. The padding of the start and end of the data made sure that the circular structure was somewhat represented. Since this method immediately gave good results no other methods were explored but there were ideas to instead link the end of the data to the start. This would however be a lot more complicated to execute when it comes to the training. It would also be possible to represent the points as only the distance from the center instead of the specific coordinates since they were evenly sampled radially around the skull. I.e. each point was placed on a line out from the centre so the only parameter needed would be the distance from the centre. The reason to not use that method was to in the future have the possibility to reconstruct more complex structures with cavities etc. and to only represent points with distances would make that more difficult. Finding another method that can extract a certain number of points equally spread out is hard when the data representation must be consistent in order to be used for training a network. It can also be noted that since the same amount of points (40) is used for all the slices the spatial distance between the points varies for each different slice i.e. slices with smaller areas have the points spatially closer to each other. This could potentially mean that a certain error could have a larger effect when the points are sampled closer together, e.g. for slices at the top of the head, compared to when they are further apart.

**Neural Networks**
When it comes to the network structure it was clear from Table 2 that the convolutional autoencoder with the skip-connections, the AES-CNN, was the best performing one. However, the difference between the AES-CNN and the AE-CNN was not major and making the AES-CNN deeper with more layers did close to nothing with the results. This could indicate that a peak was reached when it comes to the complexity of the network structure. What could be explored more is instead the different possibilities when it comes to the training. The hyper parameter tuning did increase the performance, as in Fig. 14, but there are still some parameters that could be explored such as different solvers, activation functions etc. It would also be a possibility to customize the loss

function since currently the loss is calculated for each separate entry, i.e. x and y coordinate. However, what matters the most is the actual euclidean distance between the estimation and the real points. Since the featured training loop for networks in Matlab ends up choosing the network parameters of the last iteration, and not the one actually performing the best during training, it could be an idea to customize the training loop to save the best performing parameters during training. There is however no such feature in the 2019a Matlab version which was used for this project. The only option would be to save the network for every n:th iteration.

**AES-CNN**
The evaluation on the test set for the best performing network, the AES-CNN seen in Table 3 and Fig. 33, showed about the same results as for the validation set with a low mean error of 1.07mm. This shows that the network has good generalisation over different data. In Table 3, Table 4 and Table 5 it can be seen that the network performed better for smaller holes but also that the 94.76% of slices with a mean point error of < 2mm is higher compared to the total 86.79% of points with mean error of < 2mm. This indicates that there are a few slices with large holes that increase the mean error. Looking at the slices with the highest mean errors in Fig. 34 it can be seen that there were typically 3 cases of large mean errors: A large hole in the back of the head, a single point "missing" the protruding bone of the orbita or a slice just at the top of the head making the structure irregular. For the first case it is clear that the information to the network is not sufficient. The hole is of such a size that the network can not estimate the points accurately. The second case is hard to evaluate since the distance to the point is large but the distance to the protruding bone is small. This gives an unjustly large error measurement. The third case comes from the irregular shape that occurs at the top of the head. For cases like this, instead of reconstructing the head in a strict axial orientation the head should be slightly tilted, this eliminates the problem with irregular shapes of the top slices. How large of an error that would be considered acceptable is an aesthetic question as regardless if the points are incorrectly placed from the intended skull surface, the implant will still fit since the moulds edges are based on the pre-operative CT. Since the skull is not perfectly symmetric the margin for the error would have to be considered from case to case. It could be argued that as long as the reconstructed part fits smoothly and the asymmetry in the implants is smaller than the skulls asymmetry, the fit would be acceptable.

**Case tests**
The cases in section 5.4 were created in order to get an idea of how the network performed compared to what is acceptable, with the currently used mirroring and interpolation method. The evaluation showed that the method gave as good or in some cases even better results compared to the current method. The size of the errors are somewhat misrepresented for the network compared to the current method since a point from the network with an error could still be on the bone

i.e. the error could be parallel to the skull surface. But for the current method all errors are radially from the center. An example of this can be seen in Fig. 38. A more fair error measurement could be measuring the distance to any part of the skull instead of a certain point. It can also be seen that if the mirroring is slightly off it will likely affect all of the points. This will create a systematic error for the implant that will result in a larger or smaller skull at the affected area compared to the original. Even though the size of the test cases was small it did show the potential of the network. One notable thing when inspecting the results of the test cases was that since the network is only trained on single slices it does not give any consideration to the slices above and below who are also affected by the hole. This could create irregularities moving from slice to slice. The potential problem could be fixed by interpolating the estimated points for all the slices once they are estimated. Another option to investigate would be to increase the depth of the data set for the network i.e. include more than one slice per hole of a skull. So instead of 2x80 data there could for example be three consecutive slices making up 2x80x3 data or 6x80 data depending on how they are represented. In that way the network would have to take the surrounding slices into consideration as well. This could potentially also help to reduce the errors of the estimations on the large holes in the back of the head discussed in the previous paragraph since more information about the shape of the skull would be available for the network.

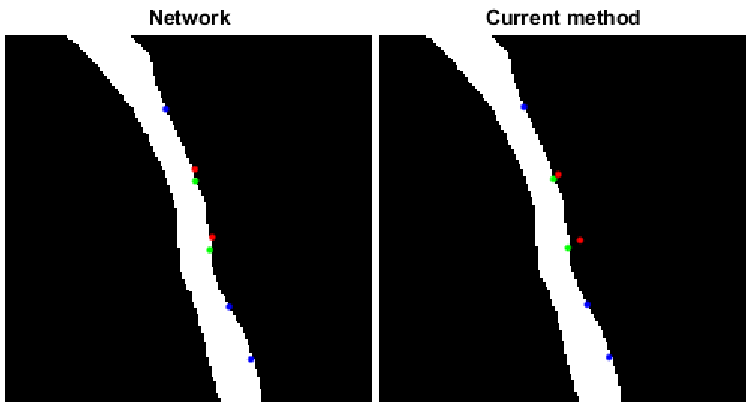


Figure 38: Zoomed-in example of estimated points that gave approximately the same error but where the estimations of the network would result in a better fitting mould compared to the current method since the estimated points are still on the skull surface. Blue dots are the original points, green dots are the missing points and red dots are the estimations.

## 6.1 Future work

Even though the developed method already is showing results good enough for it to be implemented for clinical routine there are several future ideas of possible improvements that could be explored.

- The data set consisted of points from single slices. It should be explored to increase the depth of the data set and include more slices per input. In that way there is more information for the network for it to be able to estimate large holes and to get more consistent results over a number of consecutive slices when it comes to real cases.
- Results could possibly be improved by exploring untuned network parameters such as solver and activation function but more importantly a custom loss function during training, to minimize the euclidean distance between the points instead of the coordinates, should be evaluated.
- When evaluating the networks the error measure should be calculated as the distance from the estimated point to the surface of the real skull instead of the distance from the estimated point to the specific point location.
- Points around the orbita are inconsistent and should either be excluded or a higher number of points should be extracted to make sure it is represented consistently.
- Use a more extensive size of "real" cases to compare the network to the current method and evaluate the results produced.
- Implement the method with the network in Segment 3DPrint software and create an algorithm that interpolate between the estimated slices to make the structure of the mould more smooth.

# 7 Conclusion

In conclusion, a method based on a neural network to reconstruct missing parts of the skull was successfully developed. A method to extract the points for the training of the networks in a suitable way was created and during the process numerous network structures have been tested and evaluated. The method overcomes the limitations with the current mirroring method in creation of implants or moulds for cranioplasty. The results overall and from the comparison to the current method shows that it is possible to use a neural network in the process to design implant moulds by using data from pre-operative CT images. The results were deemed good enough for the method to be implemented as clinical routine at the hospital. An example of an implant for a real case created with the method developed in this project can be seen in Fig. 39.

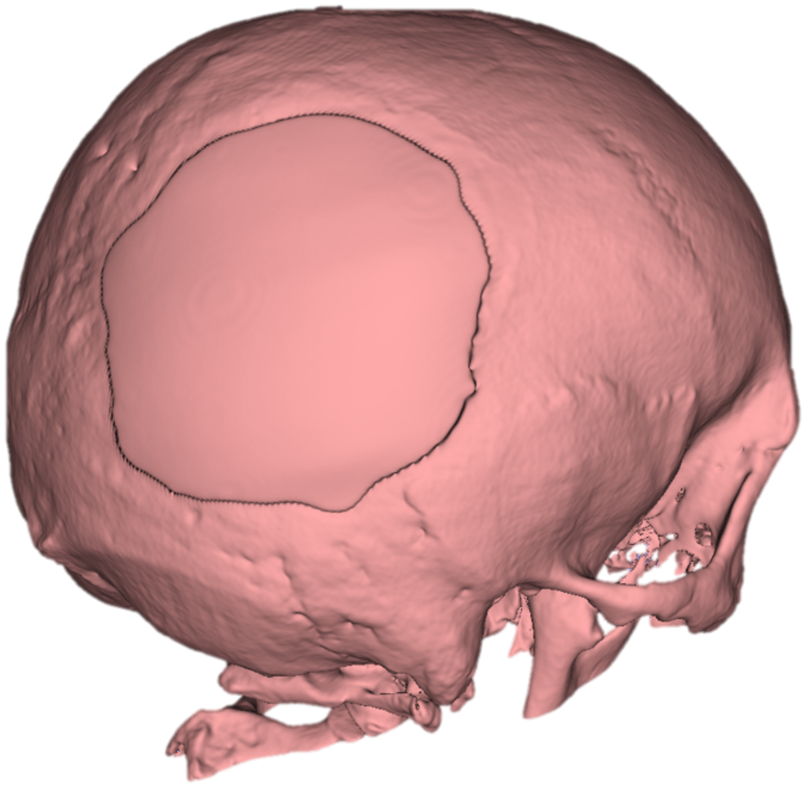


Figure 39: Implant created with the method developed in this project.

## References

[1] L. C. Hieu et al. “Design and manufacturing of personalized implants and standardized templates for cranioplasty applications”. In: *2002 IEEE International Conference on Industrial Technology, 2002. IEEE ICIT ’02.* Vol. 2. Dec. 2002, 1025–1030 vol.2. DOI: 10.1109/ICIT.2002.1189312.

[2] Manuel Dujovny et al. “Cranioplasty: Cosmetic or therapeutic?” In: *Surgical Neurology* 47.3 (1997), pp. 238–241. ISSN: 0090-3019. DOI: https://doi.org/10.1016/S0090-3019(96)00013-4. URL: http://www.sciencedirect.com/science/article/pii/S0090301996000134.

[3] Lars Kihlström Burenstam et al. “Patient-Specific Titanium-Reinforced Calcium Phosphate Implant for the Repair and Healing of Complex Cranial Defects”. In: *World Neurosurgery* 122 (Feb. 2019), e399–e407. DOI: https://doi.org/10.1016/j.wneu.2018.10.061.

[4] Siesjo, P., Heiberg, E., and Bernborg, P. “3D printade mallar for kranioplastik.” In: Forskningsansokan (2019).

[5] M.H. Lev and R.G. Gonzalez. “17 - CT Angiography and CT Perfusion Imaging”. In: *Brain Mapping: The Methods (Second Edition)*. Ed. by Arthur W. Toga and John C. Mazziotta. Second Edition. San Diego: Academic Press, 2002, pp. 427–484. ISBN: 978-0-12-693019-1. DOI: https://doi.org/10.1016/B978-012693019-1/50019-8. URL: http://www.sciencedirect.com/science/article/pii/B9780126930191500198.

[6] Medviso AB. *Segment 3DPrint*. 2020. URL: http://medviso.com/segment-3dprint/8.

[7] MATLAB. *version 9.6.0.1335978 (R2019a) Update 8*. Natick, Massachusetts: The MathWorks Inc., 2020.

[8] David Kriesel. *A Brief Introduction to Neural Networks*. 2007. URL: http://www.dkriesel.com/en/science/neural_networks.

[9] M. Nielsen. *Neural Networks and Deep Learning*. 2019. URL: http://neuralnetworksanddeeplearning.com/index.html.

[10] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[11] MATLAB. *List of Deep Learning Layers*. Natick, Massachusetts: The MathWorks Inc., 2020. URL: https://se.mathworks.com/help/deeplearning/ug/list-of-deep-learning-layers.html.

[12] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[13] Anna Ostrovskaya, N Semenov, and A Rubtsov. “Determination of the Requirements for the Neural Network for Recognition Algorithm UHRSI”. In: *IOP Conference Series: Materials Science and Engineering* 476 (Feb. 2019), p. 012021. DOI: 10.1088/1757-899X/476/1/012021.

[14] *convolution2dLayer*. 2020. URL: https://se.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.convolution2dlayer.html.

[15] *Stanford lecture notes for CS231n: Convolutional Neural Networks for Visual Recognition*. 2020. URL: https://cs231n.github.io/convolutional-networks/.

[16] Thom Lane. *Transposed Convolutions explained*. 2020. URL: https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8.

[17] Arden Dertat. *Applied Deep Learning - Part 3: Autoencoders*. 2017. URL: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798.

[18] Keita Kurita. *Machine Learning Explained*. 2018. URL: https://mlexplained.com/2018/04/24/overfitting-isnt-simple-overfitting-re-explained-with-priors-biases-and-no-free-lunch/.

[19] Pranoy Radhakrishnan. *What are Hyperparameters*. 2017. URL: https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a.

[20] MATLAB. *List of Deep Learning Layers*. Natick, Massachusetts: The MathWorks Inc., 2020. URL: https://se.mathworks.com/help/deeplearning/ref/trainingoptions.html.

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

Master's Theses in Mathematical Sciences 2020:E86
ISSN 1404-6342

LUTFMA-3434-2020

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/