



Școala doctorală interdisciplinară  
Domeniul de doctorat: Calculatoare și Tehnologia Informației

TEZĂ DE DOCTORAT

## **3D Analysis of the Normal and Pathological Coronary Morphology**

### **Analiza 3D a morfologiei coronariene normale și patologice**

doctorand:

**Alexandru DOROBANȚIU**

Conducător Doctorat:

**Prof. Dr. Ing. Remus BRAD**





This thesis is focused on automated coronary centerline extraction from Cardiac Computed Tomography Angiography (CCTA) data. This belongs to the domain of medical image segmentation, which, at the time I chose the subject for my thesis, was a hot topic in the imaging research community [1].

The work is divided into three chapters. **Chapter 1** is focused on information extraction from images and improving general image compression. With increased usage of image acquisition devices, including cameras and medical imaging instruments, the amount of information ready for long term storage is also growing. In this chapter we give a detailed description of the state-of-the-art lossless compression software PAQ8PX applied to grayscale image compression. We propose a new online learning algorithm for predicting the probability of bits from a stream. We then proceed to integrate the algorithm into PAQ8PX's image model. To verify the improvements, we test the new software on three public benchmarks. Experimental results show better scores on all of the test sets.

Edge detection plays an important role in many computer vision systems. Therefore, in **Chapter 2**, we propose a novel application agnostic algorithm for prediction of probabilities based on the contextual information available and then apply the algorithm for estimating the probability of pixels belonging to an edge using surrounding pixel values as local contexts. We then proceed to test different image transformations as input layers, such as the Canny edge detector. We propose two different architectures, one single layered and one multilayered, which approach the scaling problem by creating scaled side outputs and combining them via a logistic regression layer. We tested our approach on the BSDS500 edge detection dataset with optimistic results.

In **Chapter 3**, the focus is turned to medical image segmentation, where the mesh-type coronary model, obtained from three-dimensional reconstruction using the sequence of images produced by computed tomography (CT) can be used to obtain useful diagnostic information, such as extracting the projection of the lumen (planar development along an artery). We propose an automated coronary centerline extraction from cardiac computed tomography angiography proposing a 3D version of U-Net architecture, trained with a novel loss function and with augmented patches. We have obtained promising results in terms of accuracy (between 90–95%) and overlap (between 90–94%) with various network training configurations on the data from the Rotterdam Coronary Artery Centerline Extraction benchmark. We have also demonstrated the ability of the proposed network to learn despite the huge class imbalance and sparse annotation present in the training data.

## Chapter 1

In this chapter we describe the state-of-the-art image compression method called PAQ8PX and introduce a new algorithm for on-line automated learning. We tailored the implementation for our proposed method by integrating it with PAQ8PX, which resulted in an improved 8bpp grayscale model. We tested our implementation and obtained improvements on four datasets belonging to three benchmarks. The research was published in [2].

PAQ is a series of experimental lossless data compression software aiming at the best compression ratio at the cost of computing resources or keeping backwards version compatibility. Compressing a file goes through 4 main stages: preprocessing, model prediction, context mixing and probability refining. PAQ includes context modeling for many types of data, including images. Contexts are defined using specific modeling which includes direct modeling, indirect modeling, and last square modeling. The probability predictions from the

context models are blended using context mixing, which is a particular case of a Gated Linear Network. The output of the blending is refined through a network of adaptive probability maps.

The proposed method for improving the probability predictor is inspired from ensemble models, and the idea behind the algorithm is to encode probabilities in a memory-like structure. For resilience to noise, not all the keys should find a match in the memory.

We choose a simple model for contexts for predicting each individual bits of the pixels. We use rays in four directions with various lengths, and the quantized derivatives along the rays. We obtain a value from the memory for each context. We average all the obtained memory values then convert the average into a probability using the sigmoid function. For updating the values in the memory, we use reinforcement learning. Since we don't know the true value of the probability that a bit is 0 or 1 in a given context, we cannot use supervised learning. We back propagate the binary outcome in the network and try to minimize the cumulative logistic loss in an on-line manner. The square loss can be also used, but we are trying to minimize the wasted coding space. In order to pass mixing information to the weak learners, we propose a dual objective minimization function: in respect to the output of the network – global error, and in respect to the output of the individual nodes (side predictions) – local error. All the memory values are then updated by subtracting their local and the global error. Instead of updating weights of the mixture, we update directly the values which contribute to the average. We have no layer to separate the context weights from the input probabilities, making the method different from the context mixing algorithm.

The algorithm makes no assumption on how to organize the memory structure. We do not restrict the access to the memory to a precise scheme, but suggest three approaches: simple lookup, tagged lookup, and bucket lookup.

We implemented a tool for evaluating the pixelwise compression improvements on the PAQ8PX prediction mechanism. The tool takes as input two maps coming from two different predictor versions which are the coding cost for each bit of the input stream. The cost for each group of consecutive 8 bits (or 24 bits for color images) can be mapped to a single value and squashed into the 0-255 interval to create a pixelwise loss map for a predictor version. The mapping function can be tuned to emphasize different aspects of the prediction, and thus help estimate the potential gains should one change parameters of the model. The tool outputs other useful pixelwise information, such as the number of times a predictor got better loss, and the number of times a predictor got better loss on a given bit position in pixel.

In order to test the effectiveness of the algorithm, we applied the augmented version of the PAQ8PX algorithm with Contextual Memory to four test sets. We present our results for all the images in the datasets in order to prove that we did not tune the algorithm to a selected few. The images are compressed separately (as in not a solid archive) to prevent reusing correlations.

The main contribution of this chapter is an application agnostic algorithm for predicting probabilities based on the contextual information available with learning done in an on-line manner. The usefulness of the algorithm is demonstrated by integrating it with the PAQ8PX algorithm and testing it on several image compression benchmarks. The results show an overall compression ratio improvement across all the datasets without having special crafted features. One important difference from existing ensemble mixing algorithms is that, in our algorithm, we assume that various contexts apply together and the prediction benefits from the synergy of the side predictions, unlike the ensembles which assume model independence.

## Chapter 2

In this chapter we generalize the algorithm for automated learning introduced in the previous chapter, and which stands as a basis for various applications. We provide a description of the algorithm and point out the differences between the tested implementations. To prove its effectiveness, we applied and tested the algorithm on an edge detection benchmark. Up to this time, the algorithm was applied only for 2D images. The research was published in [3].

Edge detection has been a subject of research for many years, with papers as early as 1975 [4]. Since then, a large number of techniques have been approached, targeting different aspects of edge detection, like closed contours, human-like perception, or fast detection. The Berkeley Computer Vision Group provides a public benchmark for contour detection and image segmentation. Performance is evaluated by measuring the precision and recall, and combining them via harmonic mean into an F1 score (F measure).

Context modeling describes how the context information is structured and maintained. Depending on the problem, different types of discriminator contexts are useful. Local context refers the aspect of the data examined, as opposed to context value, which represents the numeric value of that context. The context value will be used for accessing the memory structure. The memory takes a context value as input, and outputs a value used for computing the output probability of belonging to a certain class.

When discussing edge detection and describing whether a pixel belongs to an edge or not, one of the most important aspects to take into consideration are the neighboring pixels. We can define contexts as rays, starting from the focused pixel and going in a straight line away in a given direction. Rays are defined by direction and length.

We extend the prediction technique introduced in Chapter 1 with an adaptive probability map to refine the result. The adaptive probability map (APM), also called secondary symbol estimation (SSE), is used to fine tune a probability, and works in the following way: select a set of interpolation points according to a context value, find the two points indexes between which the input value falls, then output the probability as the weighted average of the two values from the points, where the weight is selected from how far the input is from the two points.

For the case of edge detection, we can use both reinforcement learning, using the ground truth as the value zero or one, and supervised learning, using the probability of the pixel to be selected as part of the ground truth in one user submission. The global error is computed using the output probability which was refined by the adaptive probability map. This is not mandatory, but our tests show better results when the refined probability is used. This can be seen as allowing the model to learn something that can be corrected.

As an optimization for all three memory types, if the APM is designed to stretch the probability first before quantizing, the two steps that squash (after averaging) and then stretch again (before quantizing) cancel each other out, which means that the two operations, which are quite computational costly, can be skipped, as well as the value forwarded for quantization. The training phase of the memory leads to data dependencies between successive runs. The dependencies come from accessing the same memory locations, and also from accessing the adaptive probability map. However, there is room for improvement. For instance, computing the context values for accessing the memory locations is fully parallelizable. Depending on the memory type used, the memory locations can also be computed in parallel. In the testing phase, no read after write dependencies exist anymore, so the whole process can be run in parallel.

We implemented the contextual memory algorithm for an edge detector application. When analyzing about two dimensional images, color images have more layers in the dimension of the RGB colors. Hence three layers can go as an input for the algorithm. These layers are preprocessed using a chain of preprocessors. We used a Gauss filter for eliminating the noise in the input images. This takes the original RGB layers as input and outputs a three-layer image. We used a filter of size 5 and a sigma value of 1.4.

Since the algorithm makes no assumption of the data behind contexts, it can be benefic to include transformations of the color layers. Such transformations are appended, in the same way as the other layers, to the algorithm's input. We optionally used the Sobel filter, the Canny edge detection algorithm and a Kirsch edge detection algorithm as input, all of which use the color channels and output another layer. This makes the final input image have three or more layers. Having a Canny layer, or any edge detector as input is a sort of domain specific knowledge added in the model.

The single layer architecture takes a preprocessed image as an input and uses a given set of color channels to compute an output image which consists of a single layer grayscale image

in which the pixels represent the probability that the position in the original image belongs to an edge. The single layer architecture combined with the simple rays as contexts does not take into account information about the edge being kept the same at different zoom levels. To tackle the zooming problem, the multilayer architecture behaves in this way: take the original image, apply preprocessing, obtain the output; then take the original image, apply the preprocessing, append the previously obtained output as a layer, resize the image (meaning all its layers) and use it as an input for the algorithm. If one decides to separate the memory used by the algorithm at different sizes, we have a multilayer architecture. Each layer is trained separately starting from the largest image and going towards resized images. An algorithm layer can have different configuration from the other layers, with various options that include the length of the longest ray, the preprocessing done, the memory size and others. The result of the multilayer architecture is a set of grayscale images of varying sizes, which are called side outputs. These side outputs are then combined (blended) using a logistic regression layer, to form a single image. Before blending, the images are scaled to have the same size. The weights of the logistic regression layer are also trained on the training set.

Before computing the score for the benchmark, the output images are subjected to a non-maximal suppression technique and subsequently to an edge thinning. The benchmark provides a tool for evaluation which has an automated search in the space of thresholding, so that the user feels free to leave grayscale images instead of making the binarization himself.

Experimental results show that the learning is shifted towards not taking any risks, since the better F1 Score is achieved in the low threshold settings. In order to obtain an overall better F1 Score, class balancing must be considered when applying the loss function.

We made another analytical comparison using the Cross-Entropy measure. If we have two probability distributions, we can measure the number of bits needed to identify an event drawn from a set if a coding scheme is used using a probability distribution other than the true distribution of the set. Since the pixel intensities in the resulting images can be modeled as a probability of a pixel belonging to an edge, we can measure the cross entropy for the output images. We show a comparison with the Canny algorithm for the first 50 images in the test set of the benchmark.

The main contribution of this chapter is an off-line application agnostic algorithm for the prediction of probability that a pixel belongs to a class or another, based on the contextual information available, where learning can be done in a single pass over the training data. More than this, it does not impose any constraints on how to choose or model the context. It also allows it to be part of larger learning and prediction structures, having loose requirements on what information is needed for feedback.

### **Chapter 3**

In this chapter we present a machine learning algorithm for vessel centerline extraction which uses a proposed 3D adaptation of the U-NET architecture which outputs the probability for each voxel of a full volume to be part of the centerline. Also, based on an extensive state of the art review, an adapted loss function was proposed to handle sparse annotation, meaning that not all the centerlines are part of the ground truth of the dataset, and class imbalance, as very few voxels from an entire volume belong to a centerline. The research was published in [1] and [5].

Vessel centerlines can be extracted by either a segmentation and thinning pipeline, or by direct tracking. The extracted centerlines can serve as input for tracking algorithms for segmenting the required vessel tree.

The idea for creating a 3D U-Net network was based on the observation of the results of U-Net Convolutional Networks for Biomedical Image Segmentation [6], which allows a fully convolutional neural network to provide a good segmentation even when trained with a small training dataset.

The design of the U-Net follows two important steps, similar to an autoencoder network. The first one is the contraction, where successive rounds of two convolutions and a max-pooling to reduce the output size by half are applied on the input. For each round, the number of filters for the convolution is doubled. The bottom layer will have the most feature maps, but will also be the smallest in size. Its purpose is to learn an encoded representation of what it needs to be segmented. The convolution with 3x3x3 kernels means that one pixel from all the borders will be lost. To alleviate this problem, padding was employed. The second one is the expansion. Starting from the bottom layer, successive rounds of up-sampling, concatenate, and two convolutions (also with padding) are applied. The up-sampling resizes the feature vector. Together with the information concatenated from the same-size input of the contraction, the two convolutions can reconstruct the image in its original size. After the last expansion, another convolution is applied with the number of kernels equal to the number of features to extract. Since centerline extraction aims for binary segmentation, only one last kernel with a sigmoid activation function was required. The loss function is chosen such that the segmentation behaves like a voxel-wise classification function.

The limited amount of the memory of the GPU renders impossible the feeding an entire CT volume to a neural network. In 2D images, memory size is not a problem, even when working with large batches. Our first approach was to resize the volumes (and the ground truth) to a volume which would be small enough to fit in the video RAM, however the loss in precision when upscaling made the output useless for segmenting thin centerlines. Training the model using resizing posed problems with the dataset being too small. The second approach was proposed to divide the input into smaller patches, cutting only small parts of the volume (and the ground truth) and feeding them to the network. This method does not have any of the downsides of the previous one, but introduces a different one: the lack of context. Therefore, a tradeoff is reached between increasing the patch size so that the available contextual information for the model is enough to make a good prediction, and the size of the model so that the model is large enough to capture the correlations about where the centerline is positioned and that it is continuous and also crosses the patches. To generate the full output, the solution employed was to split the input in blocks of the shape of the patch, pass each of the blocks through the model, and recombine the predictions into a full 3D volume again.

For gradually adjusting weights during the training, a loss function is required, and should be chosen to quantify how much and in what direction to adjust the parameters so that, on the next iteration, the outputs are closer to the objective. None of the numerous loss functions proposed in literature fit the specifics of our dataset. A function to work with sparsely annotated centerlines is needed, meaning that examples would be contradictory, and, at the same time, with a huge class imbalance. We introduce a loss function which is a combination between the focal loss and an overlap loss, thus combining a local loss function with a global one.

To test the implementation of the proposed neural network, we use the dataset from the Rotterdam Coronary Artery Algorithm Evaluation Framework [7]. This is a public benchmark to evaluate algorithms for the task of Centerline Extraction from CTA data. The dataset is split in a training set and a testing set. The training set consists of 8 volumetric images and the test set consists of 24 volumetric images. All the images have four target vessel descriptions with each containing four reference points. The training set images additionally contain the ground truth in the form of a reference file for the target vessels. This makes it challenging for supervised learning algorithms, since many of the vessels are not reported as part of the ground truth. The 8 input volumes with ground truth were split into a training set of seven and a validation set with the remaining volume, leaving one volume out, therefore the holdout validation was employed. While the number is small, using augmented patches helps reduce the problem of a really small dataset.

The models were trained and tested on an NVIDIA GeForce GTX 1060 with 6GB VRAM. Training required around 8 hours on the formerly mentioned video card. The predicting processing time for a patch size input varies between 400ms and 600ms, depending on the patch size and model size. For a full volume, this time is multiplied by the number of patches inside

a volume. At around 80 patches and 500ms per patch, the full output is computed in around 40 seconds. Stitching and post-processing for saving is negligible in time. The training time makes hyperparameter tuning a strenuous operation, and better configurations can be achieved by integrating the algorithm into automated design space exploration frameworks. The parameter search space must be more rigidly defined, and the proposed parameters here are a good start.

Following the format of the reference file, we have created a tool to generate a volumetric mask for each input image of the training set, which serves as the ground truth in the supervised learning algorithm. The tool can be parametrized to specify the width in voxels of the centerline.

The training of the proposed architecture was performed with different configurations for the following parameters: input patch size, layer reduction, batch size, feeding or not the network patches with no single voxel of ground truth. The hard constraint put on the parameters is given by the limited amount of VRAM of the video card. An example constraint: if the input patch size was increased, the batch size or the number of convolutional kernels in the layers needs to be reduced.

The Batchgenerators framework [8] was used because it provided a wide range of transforms and included spatial augmentation, suitable for the 3D input data. The patches were augmented with spatial transformations, color transformations, noise transformations. Only the spatial transformations are applied, with the same parameters, to the ground truth. Without augmentations, the dataset was too small for the training to converge.

The classical notation of epoch, where epoch means one pass over the entire training dataset, will no longer apply when working with randomly cut patches. The term epoch is defined here by multiplying the number of volumes in the training set with the largest input volume size divided on each axis with the corresponding patch size, with the result divided by the batch size. The results are reported for different epochs given various input patch sizes. For the training dataset, the values for one epoch are computed as the average across all the input patches. For the validation dataset, the values for one epoch are computed by assembling the whole volume from the output patches and comparing it with the whole ground truth. The results are presented in the form of a pair of binary accuracy and overlap. Visual validation for the output volumes is provided.

The main contribution of this chapter is an implementation of a 3D U-Net, suitable for segmenting the coronary artery centerline. We have built upon a 2D U-Net base implementation network, originally designed for retina vessel segmentation, and developed a 3D version capable of segmenting with voxel precision. We conceived a novel loss function designed to combat two simultaneous problems, usually related to volumetric medical segmentation: huge class imbalance and sparse annotation. Without this loss function, the convergence of the model was not guaranteed even when using augmented inputs. We have demonstrated the training convergence using a second loss function starting with a network pretrained with another loss function. Without the pretraining, the training using directly the second loss function would not converge.



Focusul acestei teze este procesul de segmentare automată a liniei centrale coronariene din angiografia tomografiei computerizate cardiace (CCTA). Aceasta aparține domeniului imagisticii medicale, care, la momentul în care am ales tema, era un subiect important în comunitatea științifică de imagistică medicală [1].

Lucrarea este împărțită în trei capitole. **Capitolul 1** este axat pe extragerea informațiilor din imagini și îmbunătățirea compresiei de date a imaginilor. Odată cu creșterea numărului de dispozitive de achiziție a imaginilor, inclusiv a camerelor și a instrumentelor de imagistică medicală, crește și cantitatea de informații necesară pentru stocarea pe termen lung. În acest capitol oferim o descriere detaliată a software-ului de compresie fără pierderi de ultimă generație PAQ8PX, cu aplicare pe compresia imaginilor în tonuri de gri. Propunem un nou algoritm de învățare continuă pentru prezicerea probabilității biților dintr-un flux de date. Apoi, descriem integrarea algoritmului în modelul de imagine al PAQ8PX. Pentru a demonstra îmbunătățirile, testăm noul software pe trei benchmark-uri publice. Rezultatele experimentale arată scoruri mai bune pentru toate seturile de date.

Detectarea muchiilor obiectelor dintr-o imagine are un rol important în multe sisteme de viziune computerizată. Prin urmare, în **capitolul 2**, propunem un nou algoritm independent de domeniu pentru predicția probabilităților, care este bazat pe informațiile contextuale disponibile, și apoi aplicăm algoritmul pentru estimarea probabilității pixelilor de a aparține unei margini. Sunt folosite valorile pixelilor înconjurători ca și contexte locale. Următorul pas descris este testarea diferitelor transformări de imagine ca și straturi de intrare, cum ar fi detectorul de contur Canny. Propunem două arhitecturi diferite, una cu un singur strat și una multistrat, care abordează problema scalării prin crearea de rezultate secundare redimensionate, și apoi combinarea acestora printr-un strat de regresie logistică. Am testat abordarea noastră pe setul de date BSDS500 de detectare a conturilor cu rezultate optimiste.

În **capitolul 3**, accentul este pus pe segmentarea de imagini medicale, unde modelul coronarian de tip mesh, obținut din reconstrucția tridimensională folosind secvența de imagini produse prin tomografie computerizată (CT) poate fi folosit mai departe pentru a obține informații utile de diagnostic, cum ar fi extragerea proiecției lumenului (dezvoltare plană de-a lungul unei artere). Propunem o extracție automată a liniei centrale coronariene din angiografia tomografiei computerizate cardiace, prin crearea unei versiuni 3D a arhitecturii U-Net, antrenată cu o funcție de pierdere nouă și cu patch-uri augmentate. Am obținut rezultate promițătoare în ceea ce privește acuratețea (între 90-95%) și suprapunerea (între 90-94%) cu diferite configurații de antrenament ale rețelei pe datele din benchmark-ul Rotterdam de extragere a liniei centrale a arterelor coronariene. De asemenea, am demonstrat capacitatea rețelei propuse de a învăța, în ciuda debalansării imense între cele două clase și a adnotării rare prezente în datele de antrenament.

## Capitolul 1

În acest capitol descriem metoda de compresie de imagini de ultimă generație numită PAQ8PX, și introducem un nou algoritm pentru învățarea automată continuă. Am adaptat implementarea metodei propuse, integrând-o cu PAQ8PX, ceea ce a dus la un model mai bun de predicție pentru imagini în tonuri de gri. Am testat implementarea și am obținut îmbunătățiri pentru patru seturi de date din trei benchmark-uri diferite. Cercetarea a fost publicată în [2].

PAQ este o serie de programe experimentale de compresie fără pierderi a datelor, care vizează cel mai bun raport de compresie în defavoarea dimensiunii resurselor de calcul și fără păstrarea compatibilității cu versiunile mai vechi. Comprimarea unui fișier trece prin 4 etape

principale: preprocesare, predicția modelului, combinarea contextuală, și rafinarea probabilităților. PAQ include modelarea contextului pentru multe tipuri de date, inclusiv imagini. Contextele sunt definite folosind modelare specifică, care include modelarea directă, modelarea indirectă și modelarea regresiei liniare. Predicțiile de probabilitate din modelele contextuale sunt convertite într-o singură predicție folosind combinarea contextuală, care este un caz particular al unei rețele neuronale de tipul Gated Linear Network. Rezultatul combinării este apoi rafinat printr-o rețea de funcții de transfer adaptive.

Metoda propusă pentru îmbunătățirea metodei de predicție a probabilității este inspirată din Ensemble learning, iar ideea din spatele algoritmului este de a codifica probabilitățile într-o structură asemănătoare unei memorii. Pentru toleranță la zgomot, nu toate intrările ar trebui să găsească o potrivire în memorie.

În scopul modelării contextelor pentru prezicerea fiecărui bit individual al pixelilor, alegem o metodă simplă: folosim raze în patru direcții cu diferite lungimi și derivatele cuantizate de-a lungul acestora. Obținem o valoare din memorie pentru fiecare context. Facem apoi media tuturor valorilor obținute, apoi convertim media într-o probabilitate folosind funcția sigmoidă. Pentru actualizarea valorilor din memorie, folosim reinforcement learning. Deoarece nu cunoaștem adevărata valoare a probabilității ca un bit să fie 0 sau 1 într-un context dat, nu putem folosi învățarea supervizată (supervised learning). Propagăm înapoi rezultatul binar prin rețea și încercăm să minimizăm pierderea logistică cumulativă într-o manieră continuă. Pierderea pătratică poate fi, de asemenea, utilizată, dar încercăm să minimizăm spațiul de codare irosit. Pentru a transmite informațiile de combinare nodurilor ansamblului, propunem o funcție dublă de minimizare a obiectivului: în ceea ce privește ieșirea rețelei - eroarea globală, și în ceea ce privește ieșirea nodurilor individuale (predicții reziduale) - eroarea locală. Toate valorile corespunzătoare din memorie sunt apoi actualizate prin scăderea erorii lor locale și globale. În loc să actualizăm ponderile combinării, actualizăm direct valorile care contribuie la medie. Nu avem niciun strat adițional care să separe ponderile de context de probabilitățile de intrare, făcând diferită metoda propusă de algoritmul de combinare contextuală.

Algoritmul nu face nici o presupunere asupra unui mod de a structura memoria. Nu restricționăm accesul la memorie la o schemă precisă, ci sugerăm trei abordări: căutare simplă, căutare etichetată și căutare locală.

Am implementat un program pentru evaluarea îmbunătățirilor de compresie a datelor pentru pixeli individuali pe mecanismul de predicție PAQ8PX. Programul primește ca intrare două matrici provenind din două versiuni diferite de predictor, matrici care reprezintă costul de codare pentru fiecare bit al fluxului de intrare. Costul pentru fiecare grup de 8 biți consecutivi (sau 24 de biți pentru imaginile color) poate fi convertit la o singură valoare și limitat la intervalul 0-255 pentru a crea o matrice de pierdere la nivel de pixel pentru o versiune a predictorului. Funcția de conversie poate fi ajustată pentru a sublinia diferitele aspecte ale predicției și, astfel, ajută la estimarea câștigurilor potențiale în cazul în care se schimbă parametrii modelului. Programul furnizează și alte informații utile, cum ar fi de câte ori un predictor a avut o pierdere mai bună și de câte ori un predictor a avut o pierdere mai bună pe o anumită poziție a bitului în pixel.

Pentru a testa eficiența algoritmului, am aplicat versiunea extinsă a algoritmului PAQ8PX cu memoria contextuală pe patru seturi de test. Prezentăm rezultatele pentru toate imaginile din seturile de date pentru a demonstra că nu am ajustat algoritmul doar la câteva imagini selectate. Imaginile sunt comprimate individual (nu într-o arhivă solidă) pentru a preveni reutilizarea corelațiilor.

Contribuția principală a acestui capitol este un algoritm agnostic de aplicație pentru prezicerea probabilităților pe baza informațiilor contextuale disponibile, cu învățarea realizată într-o manieră continuă. Utilitatea algoritmului este demonstrată prin integrarea acestuia cu algoritmul PAQ8PX și testarea acestuia pe mai multe benchmark-uri de compresie de imagini. Rezultatele arată o îmbunătățire generală a raportului de compresie pentru toate seturile de date fără a avea caracteristici particularizate datelor. O diferență importantă față de algoritmi

existenți de Ensemble learning este că, în algoritmul nostru, presupunem că diferite contexte se aplică împreună și predicția finală beneficiază de sinergia predicțiilor reziduale, spre deosebire de ansamblurile care pleacă de la ipoteza independenței constituenților.

## Capitolul 2

În acest capitol generalizăm algoritmul pentru învățarea automată descris în capitolul anterior, care stă la bază pentru diferite aplicații. Oferim o descriere a algoritmului și subliniem diferențele implementărilor testate. Pentru a dovedi eficacitatea, am aplicat și testat algoritmul pe un benchmark de detectare a conturului obiectelor din imagini. Până în prezent, algoritmul a fost aplicat doar pentru imagini 2D. Cercetarea a fost publicată în [3].

Detectarea conturului imaginilor a fost subiect de cercetare de mai mulți ani, cu lucrări publicate încă din 1975 [4]. De atunci, un număr mare de tehnici au fost abordate, vizând diferite aspecte ale detectării marginilor obiectelor, cum ar fi contururile închise, percepția asemănătoare cu a omului, sau detectarea rapidă. Berkeley Computer Vision Group oferă un benchmark public pentru detectarea conturului și segmentarea imaginilor. Performanța este evaluată prin măsurarea preciziei și a recall-ului și combinarea acestora prin media armonică într-un scor F1 (măsură F).

Modelarea contextuală descrie modul în care informațiile de context sunt structurate și menținute. În funcție de problemă, sunt utile diferite tipuri de contexte discriminatoare. Contextul local se referă la aspectul datelor examinate, spre deosebire de valoarea contextului, care reprezintă valoarea numerică a aceluși context. Valoarea contextului va fi utilizată pentru accesarea structurii memoriei. Memoria ia o valoare de context ca intrare și scoate o valoare utilizată pentru calcularea probabilității de ieșire a apartenenței la o anumită clasă.

Atunci când se discută despre detectarea conturului și se descrie dacă un pixel aparține sau nu unei margini, unul din cele mai importante aspecte care trebuie luate în considerare este cel al pixelilor vecini. Putem defini contexte ca raze, pornind de la pixelul focalizat și mergând în linie dreaptă într-o direcție dată. Razele sunt definite prin direcție și lungime.

Extindem tehnica de predicție introdusă în capitolul 1 cu o funcție de transfer adaptivă pentru a rafina rezultatul. Funcția de transfer adaptivă, numită și estimarea secundară a simbolului, este utilizată pentru a ajusta fin o probabilitate și funcționează în felul următor: se selectează un set de puncte de interpolare în funcție de o valoare contextuală, se găsesc cele două puncte index între care cade valoarea de intrare, apoi se emite probabilitatea ca medie ponderată a celor două valori ale celor două puncte, unde ponderea este selectată de distanța valorii de intrare față de cele două puncte.

Pentru cazul detectării muchiilor, putem utiliza atât Reinforcement Learning, folosind clasa adevărată ca valoare de zero sau unu, cât și învățarea supervizată, folosind probabilitatea ca pixelul să fie selectat ca parte a clasei adevărate dintr-o marcă a unui utilizator din mai mulți posibili. Eroarea globală este calculată folosind probabilitatea finală care a fost rafinată de funcția de transfer adaptivă. Acest lucru nu este obligatoriu, dar testele noastre arată rezultate mai bune atunci când este utilizată probabilitatea rafinată. Acest lucru poate fi văzut ca permiterea modelului să învețe ceva care poate fi corect.

Ca o optimizare pentru toate cele trei tipuri de memorie, dacă funcția de transfer adaptivă este proiectată să extindă spațiul probabilității înainte de cuantizare, cei doi pași care restrâng spațiul (după aplicarea mediei) și apoi care îl extind (înainte de cuantizare) se anulează reciproc, ceea ce înseamnă că cele două operații care sunt destul de costisitoare computațional pot fi omise, iar valoarea de intrare este cuantizată direct. Faza de antrenament a memoriei duce la dependențe de date între rulări succesive. Dependențele provin din accesarea acelorași locații de memorie și, de asemenea, din accesarea funcției de transfer. Cu toate acestea, există loc de îmbunătățire. De exemplu, calculul valorilor contextelor poate fi complet paralelizat. În funcție de tipul de memorie utilizat, locațiile din memorie pot fi, de asemenea, calculate în paralel. În faza de testare, nu mai există dependențe de citire după scriere, astfel încât întregul proces poate fi rulat în paralel.

Am aplicat algoritmul de memorie contextuală pentru un program detector de contur. Când se analizează imagini bidimensionale, imaginile color au mai multe straturi descrise de spațiul culorilor RGB. Prin urmare, trei straturi sunt intrare pentru algoritm. Aceste straturi sunt preprocesate folosind un lanț de preprocesoare. Am folosit un filtru Gauss pentru eliminarea zgomotului din imaginile de intrare. Aceasta ia straturile RGB originale ca intrare și produce o imagine cu trei straturi. Am folosit un filtru de dimensiune 5 și o valoare sigma de 1,4.

Deoarece algoritmul nu face nicio presupunere a datelor din spatele contextelor, poate fi benefic să includem transformări ale straturilor de culoare. Rezultatul acestor transformări devine, la fel ca și celelalte straturi, intrare a algoritmului. Am folosit filtrul Sobel, algoritmul de detectare a muchiilor Canny, și un algoritm de detectare a muchiilor Kirsch ca și intrare, algoritmi care folosesc canale de culoare și produc un alt strat. Acest lucru face ca imaginea de intrare finală să aibă trei sau mai multe straturi. Având un strat Canny sau orice detector de contur ca intrare este echivalent cu adăugarea în model de cunoștințe specifice domeniului.

Arhitectura cu un singur strat ia o imagine preprocesată ca intrare și folosește un set dat de canale de culoare pentru a calcula o imagine de ieșire care constă dintr-o singură imagine în tonuri de gri în care pixelii reprezintă probabilitatea ca poziția din imaginea originală să aparțină unui contur. Arhitectura cu un singur strat combinată cu razele simple ca și contexte nu ia în considerare informațiile despre conturul păstrat la fel la diferite niveluri de scalare. Pentru a rezolva problema scalării, arhitectura multistrat se comportă în acest fel: se ia imaginea originală, se aplică procesarea, se obține o ieșire; apoi se ia imaginea originală, se redimensionează, se aplică procesarea, se adăugă rezultatul obținut anterior ca strat, și se folosește ca intrare pentru algoritm. Dacă se decide separarea memoriei utilizate de algoritm la diferite nivele de redimensionare, avem o arhitectură multistrat. Fiecare strat este antrenat separat începând de la cea mai mare imagine și mergând spre imagini de dimensiune mică. Un strat al algoritmului poate avea o configurație diferită de celelalte straturi, cu diverse opțiuni care includ lungimea celei mai lungi raze, preprocesarea efectuată, dimensiunea memoriei și altele. Rezultatul arhitecturii multistrat este un set de imagini în tonuri de gri de diferite dimensiuni, care se numesc ieșiri reziduale. Aceste ieșiri reziduale sunt apoi combinate (amestecate) folosind un strat de regresie logistică pentru a forma o singură imagine. Înainte de combinare, imaginile sunt redimensionate pentru a avea toate aceași dimensiune. Ponderile stratului de regresie logistică sunt de asemenea calculate folosind același set de antrenament.

Înainte de a calcula scorul pentru benchmark, imaginile ieșire sunt supuse unei tehnici de suprimare non-maxime și, ulterior, unei subțieri de margine. Benchmark-ul oferă un instrument de evaluare care are o căutare automată în spațiul pragului de binarizare, astfel încât utilizatorul să poată lăsa imagini în tonuri de gri în loc să facă el însuși binarizarea.

Rezultatele experimentale arată că învățarea este orientată spre a nu-și asuma niciun risc, deoarece scorul F1 mai bun este atins în setările de prag de binarizare scăzut. Pentru a obține un scor F1 mai bun, trebuie luată în considerare echilibrarea procentului de exemple aparținând unei clase atunci când se aplică funcția de pierdere.

Prezentăm încă o comparație analitică folosind măsura Cross-Entropy. Dacă avem două distribuții de probabilitate, putem măsura numărul de biți necesari pentru a identifica un eveniment extras dintr-o mulțime dacă o schemă de codificare este utilizată cu o distribuție de probabilitate diferită de distribuția adevărată a mulțimii. Deoarece intensitățile pixelilor din imaginile rezultate pot fi modelate ca probabilitate ca un pixel să aparțină unei margini, putem măsura entropia încrucișată pentru imaginile de ieșire. Arătăm o comparație cu algoritmul Canny pentru primele 50 de imagini din setul de testare al benchmark-ului.

Contribuția principală a acestui capitol este un algoritm agnostic de aplicație pentru predicția probabilității că un pixel aparține unei clase sau alteia, predicție făcută pe baza informațiilor contextuale disponibile, unde învățarea se poate face într-o singură trecere peste datele de antrenament. Mai mult decât atât, nu se impune nicio constrângere asupra modului de alegere sau modelare a contextului. De asemenea, este posibilă integrarea algoritmului în

structuri mai mari de învățare și predicție, având cerințe scăzute referitoare la informațiile necesare pentru antrenament.

### Capitolul 3

În acest capitol este propus un algoritm de învățare automată pentru extragerea liniei centrale a vaselor de sânge coronariene, care utilizează o adaptare 3D a arhitecturii U-NET care generează probabilitatea ca fiecare voxel (volum pixel) al unui volum complet să fie parte a liniei centrale a unui vas de sânge. De asemenea, pe baza unei revizuirii extinse a literaturii actuale, a fost propusă o funcție de pierdere adaptată pentru a gestiona adnotările rare, ceea ce înseamnă că nu toate liniile centrale sunt marcate în setul de date, și al dezechilibrului de clasă, deoarece foarte puțini voxelii dintr-un întreg volum aparțin unei linii centrale. Cercetarea a fost publicată în [1] și [5].

Liniile centrale ale unui vas de sânge pot fi extrase fie printr-o segmentare și apoi un algoritm de subțiere, fie prin urmărire directă. Liniile centrale extrase pot servi drept intrare pentru algoritmi de urmărire pentru segmentarea întregului arbore coronarian.

Ideea pentru crearea unei rețele 3D U-Net a fost construită pe observarea rezultatelor rețelelor U-Net convoluționale pentru segmentarea imaginilor biomedicale [6], care permite unei rețele neuronale complet convoluționale să ofere o bună segmentare chiar și atunci când este antrenată cu un mic set de date pentru antrenament.

Structura rețelei de acest tip urmează doi pași importanți, similari cu o rețea de tipul Autoencoder. Primul pas este contracția, unde sunt aplicate pe datele de intrare runde consecutive de două convoluții și apoi o operație max pooling pentru a reduce dimensiunea ieșirii. Pentru fiecare rundă, numărul de filtre pentru convoluție este dublat iar dimensiunea ieșirii față de intrare este înjumătățită. Stratul inferior va avea cele mai multe filtre, dar va fi și cel mai mic ca dimensiune. Scopul său este de a învăța o reprezentare codificată a ceea ce trebuie segmentat. Convoluția cu nuclee de  $3 \times 3 \times 3$  înseamnă că se va pierde un pixel din toate marginile volumelor de intrare. Pentru a elimina această problemă, s-a folosit procesul de padding. Al doilea pas este extinderea. Începând cu stratul inferior, se aplică runde succesive de upsampling, concatenare și două convoluții (de asemenea, cu padding). Procesul de upsampling redimensionează vectorul de caracteristici. Împreună cu informațiile provenite prin concatenarea intrării de aceeași dimensiune din pasul contracției, cele două convoluții pot reconstrui imaginea în dimensiunea sa originală. După ultima expansiune, se aplică o altă convoluție cu numărul de nuclee egal cu numărul de caracteristici de extras. Deoarece extracția liniei centrale vizează segmentarea binară, a fost necesar doar un ultim strat de neuroni cu funcție de activare sigmoidă. Funcția de pierdere este aleasă astfel încât segmentarea să se comporte ca o funcție de clasificare la nivel de voxel.

Cantitatea limitată de memorie video face imposibilă utilizarea ca intrare pentru o rețea neurală a unui întreg volum CT. În imaginile 2D, dimensiunea memoriei nu este o problemă, chiar și atunci când se lucrează cu loturi mari. Prima abordare a fost de a redimensiona volumele (și volumul cu valoarea de adevăr) la un volum care ar fi suficient de mic pentru a se potrivi în memoria plăcii video. Cu toate acestea, pierderea de precizie la redimensionarea la mărimea originală a făcut rezultatul inutil pentru segmentarea liniilor centrale care sunt prin natura lor subțiri. Antrenarea modelului folosind redimensionarea a ridicat și probleme cu setul de date care era prea mic. A doua abordare propusă a fost de a împărți volumele în patch-uri, tăind doar părți mici din volum (și valoare de adevăr) pentru intrarea în rețea. Această metodă nu are niciunul dintre dezavantajele celei propuse anterior, dar introduce una diferită: lipsa întregului context. Prin urmare, se ajunge la un compromis între creșterea dimensiunii patch-urilor, astfel încât informațiile contextuale disponibile pentru model să fie suficiente pentru o predicție bună, și dimensiunea modelului, astfel încât modelul să fie suficient de mare pentru a capta corelațiile cu privire la locul unde se află poziționată linia centrală și că aceasta este continuă și poate traversa patch-urile. Pentru a genera segmentarea completă, soluția utilizată a fost împărțirea

intrării în blocuri de forma patch-ului, trecerea fiecărui bloc prin model și recombinarea predicțiilor într-un volum 3D complet.

Pentru ajustarea treptată a ponderilor modelului în timpul antrenamentului este necesară o funcție de pierdere care ar trebui aleasă pentru a cuantifica cât și în ce direcție se ajustează parametrii, astfel încât, la următoarea iterație, ieșirile să fie mai aproape de obiectiv. Niciuna dintre numeroasele funcții de pierdere propuse în literatura științifică nu se potrivește cu specificul setului nostru de date. Este necesară o funcție care să funcționeze cu linii centrale rar adnotate, ceea ce înseamnă că exemplele ar putea fi contradictorii și, în același timp, cu un imens dezechilibru de clasă. Introducem o funcție de pierdere care este o combinație între pierderea focală și o pierdere prin suprapunere, combinând astfel o funcție de pierdere locală cu una globală.

Pentru a testa implementarea rețelei neuronale propuse, folosim setul de date din cadrul benchmark-ului Rotterdam [7]. Acest benchmark are ca scop evaluarea algoritmilor pentru extragerea liniei centrale din datele angio CT. Setul de date este împărțit într-un set de antrenament și un set de testare. Setul de antrenament este format din 8 imagini volumetrice, iar setul de testare este format din 24 de imagini volumetrice. Fiecare volum conține patru linii centrale, fiecare conținând încă patru puncte de referință. Imaginile setului de antrenament conțin în plus valoarea de adevăr a segmentării sub forma unui fișier de referință pentru vasele de sânge țintă. Acest lucru îl face dificil pentru algoritmii de învățare supervizați, deoarece multe dintre vasele de sânge nu sunt raportate ca parte a segmentării. Cele 8 volume de intrare segmentate au fost împărțite într-un set de antrenament cu șapte volume și un set de validare cu volumul rămas, lăsând un volum în afară, de aceea a fost utilizată validarea holdout. Deși numărul este mic, utilizarea patch-urilor augmentate ajută la reducerea problemei unui set de date foarte mic.

Modelele au fost antrenate și testate pe un NVIDIA GeForce GTX 1060 cu 6 GB VRAM. Timpul de antrenare a fost măsurat la aproximativ 8 ore pe placa video menționată anterior. Timpul de predicție și procesare pentru o intrare de dimensiunea patch-ului variază între 400ms și 600ms, în funcție de dimensiunea patch-ului și dimensiunea modelului. Pentru un volum complet, acest timp este înmulțit cu numărul de patch-uri care compun un volum. La aproximativ 80 de patch-uri și 500ms pe patch, ieșirea completă este calculată în aproximativ 40 de secunde. Combinarea patch-urilor și post-procesarea pentru salvarea rezultatului sunt neglijabile ca și timp. Timpul de antrenament face ca explorarea spațiului hiperparametrilor modelului să fie o operație costisitoare, și se pot realiza configurații mai bune prin integrarea algoritmului în platforme de explorare automată a spațiului hiperparametrilor. Spațiul de căutare trebuie definit mai rigid, iar parametrii propuși aici sunt un început bun.

Urmărind formatul fișierului de referință, am creat un instrument pentru a genera o mască volumetrică pentru fiecare imagine de intrare a setului de antrenament, care servește drept volum de adevăr în algoritmul de învățare supervizat. Instrumentul poate fi parametrizat pentru a specifica lățimea în voxeli a liniei centrale.

Antrenarea rețelei cu arhitectura propusă a fost realizată cu diferite configurații pentru următorii parametri: dimensiunea patch-ului de intrare, reducerea numărului de neuroni în nuclee, dimensiunea lotului, folosirea sau nu a patch-urilor fără nici un singur voxel marcat ca fiind linie centrală. Constrângerea dură aplicată parametrilor este dată de cantitatea limitată de memorie a plăcii video. Un exemplu de constrângere: dacă dimensiunea patch-ului de intrare a fost mărită, dimensiunea lotului sau a numărului de nuclee convoluționale din straturi trebuie redus.

Platforma Batchgenerators [8] a fost utilizată deoarece a furnizat o gamă largă de transformări și include augmentarea spațială, potrivită pentru datele de intrare 3D. Patch-urile au fost augmentate cu transformări spațiale, transformări de culoare, transformări de zgomot. Doar transformările spațiale sunt aplicate, cu aceiași parametri, volumului de adevăr. Fără augmentări, setul de date a fost prea mic pentru ca antrenamentul să fie convergent.

Noțiunea clasică a epocii, în care epoca înseamnă o trecere peste întregul set de date de antrenament, nu se va mai aplica atunci când se lucrează cu patch-uri tăiate aleatoriu. Termenul epocă este definit aici prin înmulțirea numărului de volume din setul de antrenament cu cea mai mare dimensiune a volumului de intrare și apoi împărțită pe fiecare axă cu dimensiunea corespunzătoare a patch-ului, și rezultatul împărțit la dimensiunea lotului. Rezultatele sunt raportate pentru diferite epoci, în funcție de diferitele dimensiuni ale patch-urilor de intrare. Pentru setul de date de antrenament, valorile pentru o epocă sunt calculate ca medie pe toate patch-urile de intrare. Pentru setul de date de validare, valorile pentru o epocă sunt calculate prin asamblarea întregului volum din patch-urile de ieșire și compararea acestuia cu întregul volum de adevăr. Rezultatele sunt prezentate sub forma unei perechi de precizie binară și suprapunere. Este prezentată și validarea vizuală pentru volumele de ieșire.

Contribuția principală a acestui capitol este implementarea unui U-Net 3D, adecvat pentru segmentarea liniei centrale a arterelor coronariene. Arhitectura are la bază o rețea neuronală U-Net 2D, concepută inițial pentru segmentarea vaselor retinei, și dezvoltarea unei versiuni 3D capabilă să segmenteze cu precizie de voxel. Am creat o nouă funcție de pierdere concepută pentru a combate două probleme simultane, de obicei legate de segmentarea medicală volumetrică: dezechilibrul imens de clasă și adnotare rară. Fără această funcție de pierdere, convergența antrenării modelului nu a fost garantată chiar și atunci când se utilizează intrări augmentate. Am demonstrat convergența antrenamentului utilizând o a doua funcție de pierdere, începând cu o rețea preantrenată cu funcția de pierdere propusă. Fără pregătire, antrenamentul direct cu cea de-a doua funcție de pierdere nu ar converge.

I wish to express my deepest gratitude to those who assisted me and understood the tough journey of all these academic years.



# CONTENTS



Introduction.....	22
1. The objectives of the thesis.....	22
2. Thesis structure and content.....	23
1. Lossless image compression with contextual memory.....	26
1.1 Overview.....	26
1.2. Related work.....	26
1.3. PAQ8PX algorithm for lossless image compression in detail.....	28
1.3.1. Description.....	28
1.3.2. General aspects.....	29
1.3.3. Modeling.....	29
1.3.4. Image compression.....	30
1.3.4.1. Direct modeling.....	30
1.3.4.2. Indirect modeling.....	30
1.3.4.3. Least squares modeling.....	31
1.3.4.4. Correlations.....	31
1.3.4.5. Grayscale 8bpp.....	33
1.3.5. Context mixing.....	33
1.3.6. Adaptive probability maps.....	34
1.3.7. Other considerations.....	34
1.4. The proposed method – contextual memory.....	35
1.4.1. Context modeling.....	36
1.4.2. Description of the contextual prediction.....	37
1.4.2.1. Model Prediction.....	37
1.4.2.2. Interpretation of values.....	37
1.4.2.3. Updating the model.....	38
1.4.3. Memory implementation and variations.....	39
1.5. Quantifying the error.....	41
1.6. Experimental Results.....	42
1.6.1. PAQ8PX Contextual Memory implementation details.....	42
1.6.2. Evaluation on the benchmarks.....	42
1.6.3. Discussion on the results.....	45
1.7. Conclusions.....	46
2. A novel contextual memory algorithm for edge detection.....	48
2.1 Overview.....	48
2.2 Related work.....	48
2.3 Berkeley Edge Detection Benchmark.....	49
2.4 Basis for the contextual memory and the processing pipeline.....	50
2.4.1 Logistic Regression.....	50
2.4.2 Ensemble learning.....	51

2.4.3 Context modeling .....	53
2.3.3 Resources and hashing as a solution .....	53
2.4 An original method for contextual prediction .....	54
2.4.1 Model Prediction .....	55
2.4.2 Updating the proposed model.....	57
2.4.3 Implementation details .....	59
2.5 Results .....	60
2.5.1 Inputs, preprocessing and processing architecture .....	60
2.5.2 Results on Berkeley Edge Detection Benchmark.....	61
2.6 Conclusions.....	73
3. Coronary centerline extraction from CCTA using 3D-UNet .....	76
3.1 Overview.....	76
3.2 Related Work .....	77
3.2.1. Rule based centerline extraction .....	78
3.2.2. Machine learning based centerline extraction .....	78
3.3. Proposed Method .....	79
3.3.1. Neural Network Architecture .....	79
3.3.2. Resize or patches.....	81
3.3.3. Loss functions .....	83
3.3.3.1. Local loss .....	83
3.3.3.2. Global loss.....	83
3.3.3.3. Combined loss functions .....	84
3.3.3.4. Proposed loss function .....	84
3.3.4. Generating the full output .....	85
3.4. Experimental Setup .....	86
3.4.1. Coronary dataset.....	86
3.4.2. Visualization .....	87
3.4.3. Execution setup .....	88
3.4.4. Training the network.....	88
3.5. Results.....	90
3.5.1 Network parameters .....	99
3.5.2 3D U-NET Python dependencies .....	103
3.6. Discussion .....	104
3.7. Conclusions.....	106
4. Final conclusions.....	108
4.1 Conclusions.....	108
4.2 Personal Contributions .....	109
4.2.1 Lossless image compression .....	109
4.2.2 Image segmentation with edge detection .....	110
4.2.3 Automated coronary centerline extraction from CCTA .....	110
4.3 Dissemination of the research results.....	110

# LIST OF FIGURES

Figure 1 PAQ8 Image compression diagram .....	29
Figure 2 Causal pixel neighborhood.....	32
Figure 3 Block scheme of the proposed prediction using the contextual memory method .....	35
Figure 4 Contexts as rays .....	36
Figure 5 Block scheme for the proposed update algorithm .....	38
Figure 6 Quantified error for image "8068" from BSD500 [38] with (a) original image, (b) pixelwise coding cost, (c) high contrast centered difference of coding costs, (d) computed disagreement. ....	42
Figure 7 (a) Example BSD500 test image "2018" with (b) ground truth .....	50
Figure 8 (a) Position of pixels as rays of length 5; (b) ray length 3 (green), length 4 (blue), length 5 (orange) .....	53
Figure 9 The prediction scheme for each pixel of one channel of a layer .....	55
Figure 10 (a) uniform distribution of points; (b) non-uniform distribution of points .....	56
Figure 11 Block scheme for the proposed update algorithm .....	58
Figure 12 Contextual memory single layer processing architecture .....	61
Figure 13 Contextual memory multilayer processing architecture .....	61
Figure 14 (a) sample image from the benchmark along with (b) the ground truth .....	62
Figure 15 Output example after NMS and thinning .....	67
Figure 16 Output of the proposed method on the training image "197017", (a) original image, (b) proposed output, (c) ground truth. ....	68
Figure 17 Output of the proposed method on the test image "51084", (a) original image, (b) proposed output, (c) ground truth. ....	69
Figure 18 Output of the individual layers on the test image "51084", (a) first layer (b) second layer (c) third layer (d) fourth layer. ....	70
Figure 19 F1 Score plotted against threshold.....	71
Figure 20 Cross Entropy for the first 50 images of the dataset (lower is better) .....	72
Figure 21 Precision in respect to the threshold (bigger is better) .....	73
Figure 22 Result of centerlines (left) to vessel segmentation (right) using sphere fitting, from [78]. ....	77
Figure 23 Example vessel segmentation from extracted centerline using set evolution active contour, from [81].....	78
Figure 24 The original U-NET architecture, from [6] .....	79
Figure 25 Example Retina Vessel Segmentation using a 2D U-Net Architecture .....	80
Figure 26 The proposed 3D U-Net architecture .....	80
Figure 27 Downscaled and upscaled centerline ground truth (80x80x64) plotted in 3D to highlight the difficulty of using it for training .....	82
Figure 28 2D slice of an augmented patch with ground truth next to it.....	82
Figure 29 Output volume composed from patches with visible stitching.....	85
Figure 30 3D Slicer Volume Rendering with CT-Coronary Arteries preset .....	87
Figure 31 (a) Training 128x128x96, batch size 1, reduction 1, only patches with ground truth (b) Validation .	91
Figure 32 (a) Training 128x128x128, batch size 2, reduction 2, only patches with ground truth (b) Validation	91
Figure 33 (a) Training 256x256x128, batch size 1, reduction 2, only patches with ground truth (b) Validation	91
Figure 34 (a) Training 256x256x128, batch size 1, reduction 2 (b) Validation.....	92
Figure 35 (a) Training 320x320x64, batch size 1, reduction 4 (b) Validation .....	92
Figure 36 (a) Training 384x384x48, batch size 1, reduction 4 (b) Validation .....	92
Figure 37 Patch size 128x128x96 rendering of ground truth (red) and centerline segmentation (transparent yellow) .....	93
Figure 38 Patch size 128x128x96 rendering of ground truth (red) and thinned centerline segmentation (green).....	93
Figure 39 Patch size 128x128x128 rendering of ground truth (red) and centerline segmentation (transparent yellow) .....	94
Figure 40 Patch size 128x128x128 rendering of ground truth (red) and thinned centerline segmentation (green).....	94

Figure 41 Patch size 256x256x128, training with only patches with ground truth, rendering of ground truth (red) and centerline segmentation (transparent yellow) ..... 95

Figure 42 Patch size 256x256x128, training with only patches with ground truth, rendering of ground truth (red) and thinned centerline segmentation (green) ..... 95

Figure 43 Patch size 256x256x128 rendering of ground truth (red) and centerline segmentation (transparent yellow) ..... 96

Figure 44 Patch size 256x256x128 rendering of ground truth (red) and thinned centerline segmentation (green)..... 96

Figure 45 Patch size 320x320x64 rendering of ground truth (red) and centerline segmentation (transparent yellow) ..... 97

Figure 46 Patch size 320x320x64 rendering of ground truth (red) and thinned centerline segmentation (green)..... 97

Figure 47 Patch size 384x384x48 rendering of ground truth (red) and centerline segmentation (transparent yellow) ..... 98

Figure 48 Patch size 384x384x48 rendering of ground truth (red) and thinned centerline segmentation (green)..... 98

Figure 49 Loss function switch (a) Patch size 128x128x128 (b) Patch size 256x256x128 ..... 105

# LIST OF TABLES



Table 1 Waterloo gray test set 1.....	43
Table 2 Waterloo gray test set 2.....	44
Table 3 Imagecompression.info 8bpp gray new test images.....	44
Table 4 Squeezechart 8bpp grayscale .....	45
Table 5 Compression running time comparison on image lena2 (expressed in seconds) .....	45
Table 6 Results for the single layer contextual memory edge detector .....	62
Table 7 Results for the multilayer architecture of the contextual memory edge detector .....	66
Table 8 Comparative results of the contextual memory edge detector .....	71
Table 9 The size and resolution of each CTA volume from the training dataset [84].....	86
Table 10 The overlap and accuracy results for the selected hyperparameters .....	90
Table 11 Network parameters for Input Size 128x128x96, model reduction 1.....	99
Table 12 Network parameters for Input Size 128x128x128, model reduction 2.....	100
Table 13 Network parameters for Input Size 256x256x128, model reduction 4.....	101
Table 14 Network parameters for Input Size 320x320x64, model reduction 4.....	102
Table 15 Network parameters for Input Size 384x384x48, model reduction 4.....	103





## Introduction

Multislice computed tomography (MSCT) imaging sequences can be used to clarify coronary anatomy and to determine if blood vessels are narrowed or even blocked. Deposits that stop blood flow are called coronary lesions. Visualization of lesions that can cause ischemia and their identification is useful for non-invasive diagnosis. The characteristics of these lesions can be used to predict whether they are at risk of rupture or displacement, causing acute coronary events. Computer vision and image processing techniques such as segmentation and feature extraction can be used to isolate, highlight, and classify these features.

### 1. The objectives of the thesis

This work is focused on automated coronary centerline extraction from Cardiac Computed Tomography Angiography (CCTA) data. This belongs to the domain of medical image segmentation, which, at the time I picked the subject for my thesis, was a hot topic in the imaging research community [1]. The numerous research challenges related to this issue strengthened the idea that this topic is a promising one. Furthermore, having a large community working in that direction meant benchmarks and other required datasets would be available.

This thesis approaches three domains, seemingly distinct, and I would like to clarify how these three go together towards the final goal.

Starting from the idea of segmenting the centerlines of the coronary vessel tree, the analysis of this topic led me towards the domain of information extraction. The simplest way to validate the accuracy of understanding the information conveyed by the pixels of an image is to try modeling the features of the image and predicting the values of the pixels in a given context. This process is used in data compression, where a better predictor leads to a better compress ratio, and the results are quantifiable. This process is done in an on-line manner, meaning that the predictor always starts from an empty model, and has to update the internal state with each prediction made. Without starting with an empty model, for the decompressor to work correctly it needs the predictor model which was used for compressing. This implies that, depending on the model size, the gains of the compression will be diminished, or it may even lead to inflating the stream. Starting from an empty model also puts a big burden on the predictor, meaning that feature extraction needs to be done as fast as possible, and in a single pass over the bitstream. Whether the images are 2D or 3D, the techniques used for compression are similar. Machine learning is a helpful tool for extracting statistics from the data. These statistics are then used for generating next-bit probabilities, which, together with an entropy coding strategy, form a lossless file compressor. The first step in improving the existing state of the art compression methods was to understand the details and where the predictive power comes from. I then proceeded to extend the existing algorithm with an original idea, inspired from ensemble models, combined with reinforcement learning. There is always a compromise between the quality of prediction and computational cost. For the proposed algorithm, several optimizations have also been provided.

By plotting the pixelwise coding cost of a statistical lossless image compressor, it can be seen that the resulting image is very similar to the output of an edge detector, and the big difficulty the predictor has in the strive for accuracy is to find the position and the global shape

of the objects' edges. Naturally, the next step was to implement a contour detector, which is a form of image segmentation – detecting which pixels from the image belong to the contour. This is also a case of thin segmentation. The feature extraction needed for this task does not work so well as with on-line learning, since better results can be yielded with domain knowledge. I extended the proposed algorithm for data compression to an off-line learning framework, and I added the ability to use supervised learning. The benchmark used for validation provided the segmented edges for the ground truth coming from multiple user submissions. Most of the submissions did not agree on all the objects to segment, because the semantic segmentation was subjective. In order to address this, I extended the processing pipeline from a single layer architecture to a multiple layer architecture in the hope of catching these semantic details. The multilayer architecture proved to give better results.

Medical image segmentation borrows the techniques from the general-purpose image segmentation, and introduces domain specific knowledge. It is used to obtain new information and projections from the available inputs. My attention turned to deep learning, where the basis of feature extraction was to add many layers to the network architecture, in the hope that intermediate layers will provide useful abstractions. By useful I do not wish to convey that they are meaningful to humans, but useful for prediction. Convolutional neural networks (CNNs) are designed in such a way that they make the assumption that neighboring inputs share a correlated cause. The assumption holds for images, may them be planar or volumetric. CNNs often contain pooling layers after convolutional ones, which help reduce the number of neurons needed. They make the network less susceptible to overfitting and faster to train. There are downsides to using deep learning, such as the need for many training samples for the right abstractions to be generated in the deep layers. In the medical imaging domain, large enough and completely annotated datasets may be hard to procure.

Applications of medical image segmentation include shape analysis, for example melanoma shape analysis used for skin cancer detection, volume and volume evolution analysis, used for example in the tumor analysis, and mass detection, where material densities are added as domain knowledge to the volume analysis. Other applications are related to computer aided diagnosis (CAD), by providing tools for assisting human experts, finding regions of interest, visualization, and the option to isolate, highlight and classify features.

Some of the challenges in medical image segmentation with machine learning which had to be overcome at the time of this writing include:

- Finding the appropriate network architecture along with a good strategy to train the models, with no silver bullet expected [1]
- Creating or finding representative datasets, with enough data to train the algorithms, and with high enough variability for the network output to be reliable
- 3D architectures require a large amount of computation power and computing system memory
- A lack of good open-source resources, since many research projects are funded by private companies
- Few frameworks for result validation and comparison
- Difficulties of integrating the algorithms (due to their experimental nature) into production pipelines

## **2. Thesis structure and content**

Developing the ideas presented above, the thesis is organized as follows:

**Chapter 1** is focused on information extraction from images and improving general image compression. With the increased use of image acquisition devices, including cameras

and medical imaging instruments, the amount of information ready for long term storage is also growing. In this chapter we give a detailed description of the state-of-the-art lossless compression software PAQ8PX applied to grayscale image compression. We propose a new online learning algorithm for predicting the probability of bits from a stream. We then proceed to integrate the algorithm into PAQ8PX's image model. To verify the improvements, we test the new software on three public benchmarks. Experimental results show better scores on all of the test sets.

Thin segmentation and edge detection play an important role in many computer vision systems. Therefore, in **Chapter 2**, we propose a novel application agnostic algorithm for prediction of probabilities based on the contextual information available and then apply the algorithm for estimating the probability of pixels belonging to an edge using surrounding pixel values as local contexts. We then proceed to test different image transformations as input layers, such as the Canny edge detector. We propose two different architectures, a single layered one and a multilayered one, which approach the scaling problem by creating scaled side outputs and combining them via a logistic regression layer. We tested our approach on the BSDS500 edge detection dataset with optimistic results.

**Chapter 3**, focuses on medical image segmentation, where the mesh-type coronary model, obtained from three-dimensional reconstruction using the sequence of images produced by computed tomography (CT) can be used to obtain useful diagnostic information, such as extracting the projection of the lumen (planar development along an artery). We propose an automated coronary centerline extraction from cardiac computed tomography angiography (CCTA) proposing a 3D version of U-Net architecture, trained with a novel loss function and with augmented patches. We have obtained promising results for accuracy (between 90–95%) and overlap (between 90–94%) with various network training configurations on the data from the Rotterdam Coronary Artery Centerline Extraction benchmark. We have also demonstrated the ability of the proposed network to learn despite the huge class imbalance and sparse annotation present in the training data.

**Chapter 4** draws the final conclusions, highlighting the main contributions of this thesis, and presents the dissemination of the results mainly in journal papers [1–3,5,9], and in [10].





## 1. Lossless image compression with contextual memory

### 1.1 Overview

The first part of the thesis is focused on information extraction from images, and a good objective metric of how well an image is understood is through compression size.

Why is compression a difficult problem? In general, when it comes to predicting something, you need to understand the process behind the result. This requires acquisition of knowledge about the environment and the potential dynamics. For example, if you know the English language, it will be rather easy to predict the letters missing from a truncated sentence. Predicting the value of the pixels in an image requires deep understanding of what is represented in the image. The predictor needs to create an internal representation of segments that correspond to features of the image, like shapes, patterns, textures, borders, and then make the guess based on which part of the segmented image it is currently in.

An important application of image compression is in the field of medical imaging. Whether the images come from radiography, magnetic resonance imaging, ultrasonography or by other methods, the number of acquired images is growing, which makes it increasingly necessary to use advanced compression methods. There are two important operations that require improvement: the storage of images, be it for long or short term (archiving), and the transmission of images via networks. When it comes to quality, lossy methods need to keep the quality of the image high to prevent mispronounced diagnostics. There may be cases where medical law would state that a copy of the medical images should be long term stored in lossless mode to allow diagnostic reconsideration in case of legal proceedings.

In this chapter we describe the state-of-the-art image compression method called PAQ8PX and introduce a new algorithm for on-line automated learning. We tailored the implementation for our proposed method by integrating it with PAQ8PX, which resulted in an improved 8bpp grayscale model. We tested our implementation and obtained improvements on four datasets belonging to three benchmarks. The research was published in [2].

### 1.2. Related work

Since compression is a difficult problem, the techniques used come from many branches of algorithmics. We provide a review of several algorithms published in the recent literature and some of which use a similar contextual method as ours.

Wavelet compression involves decorrelating the neighboring pixel values by convoluting them with a basis function and then entropy encoding the resulting coefficients. Burrows Wheeler transform involves applying a reversible sorting algorithm to the data, making it compressible using simple operations. Since these methods remove the contextual correlation in the data stream, data compression falls into the category of non-contextual methods. There is ongoing research in the area of non-contextual methods applied for two-dimensional or three-dimensional images.

Lossless wavelet compression was improved in [11] by introducing a new family of update-then-predict integer lifting wavelets. In [12], the authors extended the Burrows Wheeler transform to two dimensions. The Bi-level Burrows Wheeler Compression Algorithm applies the well-known block sorting algorithm on the rows of the image and then on the columns, for an improved homogeneity in the 2D space. It then uses a modified Kernel Move-To-Front for the 2D subspace before the entropy coding stage.

A mixture of lossless and lossy wavelet based color image compression has been described in [13], where region of interest based on the saliency of the image is taken into account when sending the image progressively through the communication network. It was applied for wildlife photography where the images are sent through a limited bandwidth channel. The ROI is extracted using a convolutional neural network to create a mask. Two wavelet encoding types are then used: for the lossless part SPIHT coding, for the lossy one EZW coding.

Deep learning for residual error prediction has been described in [14]. Here, a Residual-Error Predictive Convolutional Neural Network (REP-CNN) is introduced with the scope of refining the prediction of the LOCO-I and CALIC predictors. In total, three REP-CNN are trained, one for direct prediction and two for predicting the residuals of the aforementioned predictors. The big disadvantage of such a method is that, in order for a decoder to work, the entire trained neural network needs to be sent along with the compressed representation.

Contextual methods are still the base for both lossless and lossy image compression. There is a lot of diversity in the literature about the choice of context and how it is used. An example of a lossy image compression applied for medical ultrasound images relies on contextual vector quantization, as shown in [15]. In this algorithm, a separation method based on region growing distinguishes a region of interest in the image starting from a seed point. Different vector sizes are chosen for background and the contextual ROI. The regions are afterwards encoded with high and low compression ratios respectively, and then are merged in a final result.

Another lossy image compression for medical imaging [16] relies on contextual prediction of the quantized and the normalized sub-band coefficients after a discrete wavelet transform was applied.

Extending the Prediction by Partial Matching for two dimensions for lossless compression of indexed raster images has been presented in [17]. Context models for sparse order lengths are created and stored in an AVL-tree structure. A parallelization of the coding algorithm is presented by splitting the image into independent blocks and compressing them individually.

Context Based Predictor Blending for lossless compression of color images is described in [18], which is an extension of the algorithm CBPB [19], where the image is interpreted as an interleaved sequence generated by multiple sources so that non-stationary signals are better predicted. The blending prediction weights are selected based on the texture of the surrounding pixels and a Pearson correlation coefficient is computed for adjusting these weights. The final prediction also takes into account a template matching prediction. The CBPB algorithm was also ported to parallel execution via a CUDA implementation [20].

Vanilc is a lossless image compression framework described in [21] for 8bpp, multichannel color images, and 16bpp medical 3D volumes. The main contribution of this chapter is a pixel probability distribution predictor based on a weighted least squares approach which uses a weighting function that generalizes some of the proposed contextual schemes in the literature and provides good results when it comes to the non-stationarities in the image while having only a few tuning parameters.

A lossless image compression algorithm is described in [22]. It is based on multi-resolution compression for progressive transmission. It improves on prior work from [23], where the image is decomposed into a pyramidal data structure and an edge adaptive hierarchical interpolation is applied for coding and progressive transmission. The prediction accuracy is improved here by using context-conditioned adaptive error modeling and by passing

the estimates through an error remapping function. In this way, it improves both the final bitrate and the visual quality of the reconstructed images at intermediate stages.

Another lossless medical imaging compression algorithm using geometry-adaptive partitioning and least square-based prediction is described in [24]. Because of the similarities of the images obtained from the same imaging method, a prior segmentation via geometry adaptive partitioning and quadtree partitioning of the image allows a good selection of a least squares optimized predictor for sections of the image.

For lossless compression of 3D medical images, an extension of the Minimum Rate Predictors from 2D to 3D has been developed in [25]. Here, 3D shaped predictors were introduced to benefit from the volumetric redundancy, then volume-based optimizations are applied, and hybrid 3D block splitting and classification is done. The algorithm was also extended from 8bpp images to 16bpp images because they provide better diagnostic quality.

Lossless compression of Multitemporal Hyperspectral Images can also exploit the temporal correlations besides the spatial and spectral ones. In [26], the Fast Lossless predictor, a variation of the Least Means Square applied to the causal context [27], has been extended to 4D to incorporate the temporal aspect in the prediction. The residuals are computed as the difference between the prediction and the current pixel and are then encoded using the fast Golomb-Rice Coding.

### **1.3. PAQ8PX algorithm for lossless image compression in detail**

#### **1.3.1. Description**

PAQ is a series of experimental lossless data compression software aiming at the best compression ratio for a wide range of file types without focus on using few computing resources or keeping backwards version compatibility. It was started by Matt Mahoney and later developed by more than 20 developers in different branches of compression. PAQ8PX is a branch of PAQ started by Jan Ondrus in 2009 and which has recently adopted the best image compression models in the series with the help of Márcio Pais. In short, we refer to version 167 of PAQ8PX.

A detailed description of the software in its current phase is not available in literature. The reason may be the everchanging filetype specific models and the amount of version branching this software receives, from simplified models for fast compression, to platform specific optimization test and generalization of the algorithms used. However, a description of the PAQ series of compressors from the perspective of machine learning is available at [28].

The description of the overall compression algorithm and the techniques used can be found in [29] and [30]. The PAQ8PX version has a development thread which can be found at [31]. The source code is written in the C++ programming language and is contained in only one file with more than 12000 lines of code. The logic was not broken into different files in order to make it easier to compile to any platform. The big downside of this is that it makes the code very difficult to read. Another thing that makes the code difficult to develop is that plenty optimization techniques were inserted along the code, which can slow down the understanding of what is going to execute and when.

The general compression scheme for image compression is presented in Figure 1.

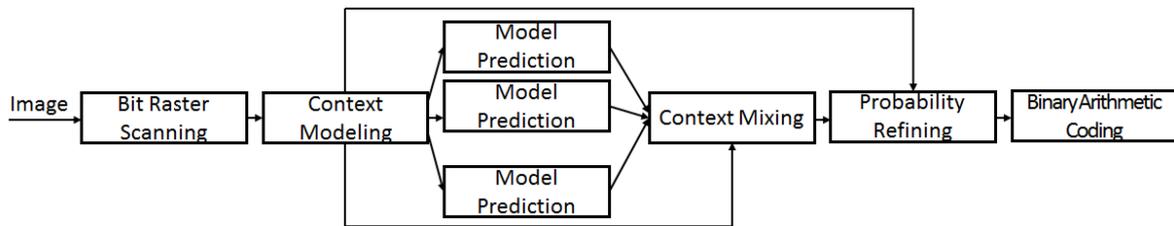


Figure 1 PAQ8 Image compression diagram

### 1.3.2. General aspects

Compressing a file goes through 4 main stages: preprocessing, model prediction, context mixing and probability refining. An optional pre-training phase can be activated via command line parameters.

The preprocessing phase is also split into three parts. At first, it searches through the file to be compressed for known stream types. Based on these types, different models are activated for the second stage of compressing. For example, it searches for image (1bpp, 4bpp, 8bpp, 8bpp grayscale, 24bpp, 32bpp, png 8bpp, png grayscale 8bpp, png 24bpp, png 32bpp), jpeg, gif, text, audio (8 and 16 bit mono and stereo), exe, base64, zlib streams, file containers and others. After this stage, an optional transform phase is applied for certain stream types such as text, where an End of Line transform can be applied, or EXE, where certain instructions are replaced with others. The transform phase is then applied in reverse and if the result matches the original stream, the transform is kept.

In the case of images, the preprocessing phase extracts the file header, which is compressed separately, and the byte stream containing the pixel values of the image. The width and the bit depth of the image are extracted from the header and the width is passed on to the image model selected by bit depth.

The model prediction and context mixing phase happen consecutively. Probabilities of individual bits from the input stream are predicted by many specialized models. All the probabilities are combined into one probability via the context mixing algorithm. The output probability is refined using a network of adaptive probability maps. The final prediction is used to encode the bit from the stream using a binary arithmetic coder. The algorithm is symmetrical, meaning that both the coder and the decoder do the same operations ending up with the same final probability. The decoder uses the probability to decode the bit from the compressed stream.

### 1.3.3. Modeling

The term model is used with double meaning throughout the compressor. At first, it is used to denote the unit of the algorithm which outputs a probability which will participate to the mixing phase. One can interpret this as an “elementary” model. The second meaning is the collection of units which are modeling a given type of data. The output of such models is, evidently, a collection of probabilities. Example models are: TextModel (for language specific language stemming and word modeling), MatchModel (for repeatable long matches of data), RecordModel (for data structured in records), SparseModel, JpegModel (for specific jpeg data), WavModel, ExeModel, DmcForest (a collection of Dynamic Marcov Coding models), XmlModel, PpmModel (various order prediction by partial matching), ImageModels (for different bit depth image data) and many more. One or more of this type of models is selected according to the input stream type and compression parameters.

It is out of the scope of this chapter to explain models unrelated to image compression. These can be further detailed in a paper on general compression.

### 1.3.4. Image compression

In the case of image streams, the match model can be optionally activated and can bypass the image model if there is a long match found. But our focus will be set on the 8bpp image model. The output of this model contains predictions for four types of input streams: 8bpp indexed color or grayscale and 8bpp png indexed or grayscale. If the stream is png, a part of the filtering scheme used is undone in order to obtain the true pixel value.

Depending on the type of image, different correlations can be expected and, thus, exploited by specific modeling. Before describing how the specific contexts look like, we should describe which types of operations are possible with the contexts. Three major types of models can be identified: direct, indirect and least squares modeling. All of these models expect byte level context values (as data coming from a file comes in byte chunks) and can output direct probabilities, stretched probabilities or both. The context mixing stage expects probabilities in the logistic domain (stretched probabilities) and different operations are applied to probabilities to fit or skew them into this domain.

#### 1.3.4.1. Direct modeling

Direct modeling is implemented with the use of stationary context maps. This type of map takes as input a context value and outputs a weighted stretched probability and a weighted probability centered around zero (skewing). It is implemented using a direct lookup table where each entry stores a probability (which is then stretched and skewed) and a hit counter. On the update phase, an error is computed as the difference between the stored probability and the value of the bit. The error is weighted with a value dependent on the hit counter. Fewer hits on the context value indicate a more rapid update rate. This is implemented via a lookup table containing the values of an inverse linear function of the hit count.

For each context requiring a direct modeling, a new map must be created. This protects the contexts from colliding with each other.

#### 1.3.4.2. Indirect modeling

Unlike direct modeling which updates the probability based on the last probability predicted, the indirect modeling tries to learn the answer based on a similar sequence from the past. Indirect modeling is implemented with the use of indirect context maps, which use two step mapping. An optional run context map is also included, which is used for modeling runs of bits.

The first mapping is between a context value and a bit history called state. The state is modeled as an 8-bit value with the following meaning. A zero value means the context value has never been seen before. States from 1 to 30 map all the possible 4-bit histories. The rest of the states represent bit counts of zero and one or an approximation of the ratio between zeroes and ones if the number of previously seen bits exceeds a count of 16. The states are used as indexes in a state table which contains transitions to the next state depending on the value of the next bit. The states were empirically chosen to try to model non-stationarity and different state maps were proposed in other compression programs [32].

The states are kept in a hash map implemented as a table with 64-byte entries to fit in a cache line. The entries contain checksums for the context value to prevent collisions and up to

7 state values. Since the map expects byte data, at bit 0, 2 and 5, the bucket for a context value is recomputed via a dispersion function. The 7 state values can hold information about no bits known (one value), one bit known (two values), and two bits known (four values). At bit zero, only 3 states are needed and, as an optimization, the next four bytes implement a run map which predicts the last byte seen in the same context value, logarithmically weighted by the length of the run. The hash map implements a *least frequently used* eviction policy and a *priority eviction* based on the state of the first element in the bucket. States are indexed based on the total number of bits seen, therefore the more information available is favored.

The next mapping is between the state and one or more probabilities. This is done in a similar manner as in direct modeling by using a state map. For each input, four probabilities are returned, one stretched, one skewed, and two depending on the bit counts of zero and one for that state. The fifth probability out of the indirect context map comes from the run map.

Unlike stationary maps, more contexts can be added to the indirect map, meaning that they share the same memory space and are identified by an index. Each context has its own state map accessed by the index. Having states modeled as 8-bit values makes them more memory efficient than the 32-bit representation for stationary maps.

#### 1.3.4.3. Least squares modeling

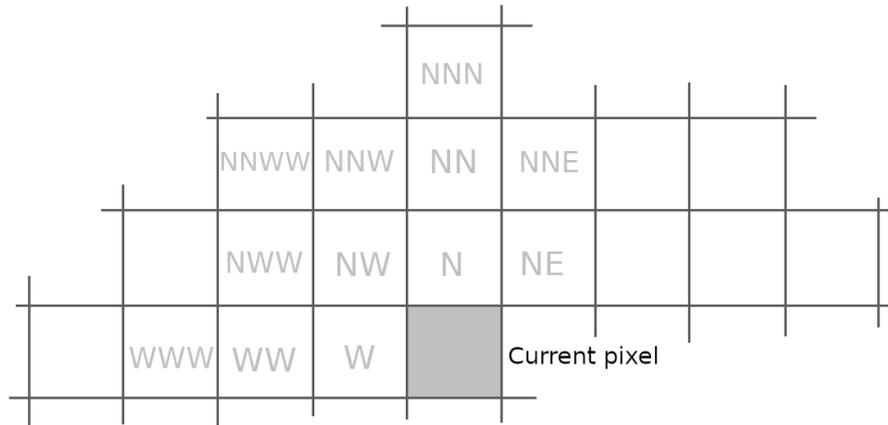
An Ordinary Least Squares modeling is used for predicting the value of the next pixel (not bit prediction) based on a given set of context values and acts as a maximum likelihood estimator. The prediction is a linear combination of the regressors, which are the explanatory variables. The update phase tries to minimize the sum of squared differences of the true pixel value and the predicted value. Finding the values of the weight vectors is done on-line by the method of normal equations which uses a Cholesky decomposition which factors the design matrix into an  $n$ -by- $n$  lower triangular matrix, where  $n$  is the number of regressors. The matrix is then used to analytically find the weight values. The bias vector and the covariance matrix are updated using parametrized momentum.

The value of the prediction is not used directly, but is used in combination with the known bits of the current byte and the bit position in byte as a key into a stationary context map.

#### 1.3.4.4. Correlations

Different types of correlations are exploited for the type of images supported since we have varying expectation of what the byte values from the input stream represent in the image. It is difficult to describe all the operations used and only a minimal description will be provided. This section does not cover the png modeling.

The neighboring pixels are the best estimators for searching correlations. They form the causal pixel neighborhood. Various notations are used for representing the position of the pixels. A simple and meaningful representation is by using the cardinal points on a compass (see Figure 2).



**Figure 2** Causal pixel neighborhood

Each time a cardinal point is mentioned, a step of the size of the pixel is taken into that direction relative to the pixel that is being predicted.

*Palette color indexed images*, as the name suggests, use the byte value to index the true RGB color in the palette table. This means that the direct values cannot be used with linear predictors because a linear combination would also be an index and might end up suggesting a completely different color. Another problem is that quantizing the values will also result in different indexes which are not matches to the expected texture in the image. Moreover, since we know that we have 8-bit indexes, we expect that only a small portion of the entire color plane is used. This makes the use of indirect context maps useful and context values will be computed, for example, by hashing the W, N and NW values together.

*Grayscale images* or individual color planes in color images require different modeling which is dependent on what the content of the image represents. If the source of the image is artificial (meaning computer generated, renders, drawings or screenshots) hard edges and continuous tone regions may be expected. Photographic images may present noise, which makes the process of prediction more cumbersome.

Of course, like for palette images, texture tracking via indirect context maps is useful. Contexts can now be computed also by quantizing the values or computing intensity magnitude levels using logarithm functions of direct values or of logarithms of the difference of quotient of two values.

Additionally, modeling for the expected pixel value is needed. The results are used as keys into stationary maps. Various prediction techniques work in many directions, including horizontal, vertical and diagonals.

Inspired from video compression schemes, half-pixel, quarter pixel, n-th pixel interpolation and extrapolation provide predictions and can be combined with other predictions by averaging, gradients and other interpolation techniques.

Linear pixel value combinations are used, such as averaging or gradients. For example, if the two pixels from above have values 60 and 50, a combination of the form  $N*2 - NN$  will output 40. An averaging combination of the form  $(N + NN) / 2$  and will output 55. Another type of combination can be a Lagrange polynomial used for extrapolating, like  $NNN*3 - NN * 3 + N$ . Extrapolated values from different directions are then combined by linear combinations for new predictions. The result of a prediction can be negative or above the maximum value of 255, therefore two functions are applied to the result. The Clip function restricts the value in the  $[0, 255]$  interval. The Clamp function is similar to the strategy employed by the LOCO

predictor for keeping the prediction in the same plane as the neighboring pixel values which are also passed as parameters to the function.

*Color images* exploit the same correlations as the grayscale images, but include modeling for the spectral correlation of the color planes. This means that an increased gradient in the red color plane can also mean increased gradients in the other planes. The magnitude of the change in a previous plane can be used to make predictions in a current plane or a prediction in the current plane can be refined based on the residual of the prediction in the previous plane.

#### 1.3.4.5. Grayscale 8bpp

In the analyzed version of PAQ8PX, a number of 62 stationary maps are used for grayscale images. Five of them are used in conjunction with OLS modeling, in order to model quadrants of the causal pixel neighborhood of different lengths. The others accept as keys various clipped and clamped predictions. An indirect context map is used which accepts 27 entries as keys computed as hashed predictions. This means that the estimated number of probabilities which are the output of the image model for grayscale images is  $62 * 2 + 27 * 5 = 259$ .

### 1.3.5. Context mixing

Encoding of a bit needs only one probability and the bit to code. Modeling produces, as we've seen, many probabilities which need to be combined to obtain a final probability. One option would be to do a linear combination of the probabilities and adjust the weights accordingly after the true value is available.

The solution in the PAQ8 family of compressors is to use a gated linear network, and context mixing being one implementation of such a network. The details of GLNs are described in detail in [33] which also include the mathematical proof of the convergence guarantee. The description of the network is split into 3 parts: Geometric Mixing, Gated Geometric Mixing and Gated Linear Networks.

Geometric Mixing is an adaptive online ensemble which was analyzed in depth and whose properties are described in [34], [35] and [36]. The main difference to a linear mixing, which implies weighting the probabilities directly, is that the probabilities are first transformed into the logistic domain using the logit function (sometimes referred to as stretch in this chapter).

$$\text{logit}(x) = \log\left(\frac{x}{1-x}\right) \quad (1)$$

Then, they are linearly combined and then the result is transformed back into a probability using a sigmoid function (sometimes referred to as squash in this chapter).

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (2)$$

The weights are updated using an online gradient descent together with a logarithmic loss. In this way, a weighted redundancy minimization in the coding space can be achieved (minimized Kullback-Leibler divergence) [35].

An advantage of this method when compared to regular probability weighting also comes with the fact that weights don't need to be normalized or clipped to the positive domain.

Gated Geometric Mixing means adding a context selector. So far, we have a neuron which takes as input stretched probabilities and has weights associated with the input. If, instead, we had a set of weights from which we select one based on an index, we would create a gate. The

index can be computed as a function of a context or as additional information. We can now say that the neuron has specialized weights.

Gated Linear Networks are a network of stacked Gated Geometric Mixing layers of neurons. The output of a Gated Geometric Mixing neuron is a probability. A set of neurons which works on the same input forms a layer. The set of outputs of one layer form the input for another layer. A final probability is obtained when a layer contains only one neuron. At a first glance, the network looks similar to a multi-layer perceptron, but, in this case, the learning is not done via backpropagation. Instead, each neuron output tries to approximate the end probability, and, since each layer constructs on the output of the previous, it further improves the result.

We present below some important considerations. The loss function is convex, which implies a simplified training of a deep network. The network rapidly adapts to the input, making it a perfect candidate for on-line learning. Weights can be initialized in more ways and random assignment is not necessary because of the convexity of the loss function. The PAQ8 compressors initialize all the weights to zero with the implication that no predicting model has any importance in the beginning and allowing a rapid update towards selecting the best specialist. Clipping the weights and regularization techniques are also presented in [33], but are not used in PAQ.

PAQ8PX uses a network with two layers for image compression, the first layer having 7 neurons and using as contexts functions of immediate pixels and column information, which means that the pixel position in the image is taken into account.

### **1.3.6. Adaptive probability maps**

Adaptive probability maps (APM), sometimes referred to as secondary symbol estimation, have as input a probability and a context value and as output another probability. The context value serves as an index in a set of transfer functions. Once selected, a set of interpolation points are available. In the initial state, they should map the input probability to the same value. The input probability is quantized between two points of the set; the output value is the linear interpolation of the value of the points weighted by the distance from them. In the update phase, the two end values are updated so that the output probability is closer to the value of the predicted bit.

There are variations of the APM. One of them, which is used in PAQ8, is to have as input a stretched probability with the benefit of having more interpolation points towards the zero and one probabilities, where compression benefits from the fine tuning. Other compression software products use APM with two quantized predictions as input with a 2D interpolation plane.

It is not necessary to use only a single APM, since they can be connected in a network. PAQ8PX uses different architectures based on the type of stream detected. For 8bpp grayscale images three APMs are used. Two of them take as input the output of the context mixing phase and have as contexts functions which include the current known byte bits, the number of mispredictions in the past and whether the prediction falls in a neighborhood plane or not. The output of the first APM is again refined and the final prediction is a fixed weight linear combination of the three probabilities.

### **1.3.7. Other considerations**

Predictions need to be perfectly identical when compressing and decompressing, otherwise the decoder will rely on false data. Floating point operations cannot guarantee cross compiler, cross processor and cross operating system this hard constraint. It became even more

important to have fixed rules when support for streaming instructions like SSE and AVX was added. It was decided that fixed point arithmetic will be used across all operations. Even setting initial values for lookup operation tables like stretching, squashing or logarithm was done by interpolation of initial integer values or by numerical integration. Components described here use fixed point values with varying point position. The representation can be on 16bit or 32bit integers. For example, having the weights of the context mixing algorithm represented on 16bits is useful when using vector instructions, since more values fit into operands. Some exception to this rule was made for the sake of maximum compression for the wav model and the ordinary least squares algorithm used in image compression.

Another unintuitive part is that the update part of each model takes place right before the predict part. The first prediction is by default 0.5 since it relies on no information. Afterwards, each time the predictor is queried, it does the update with the known bit and then computes a prediction. This is done as an optimization since the accessed memory locations during the update might still be loaded in cache and the prediction might need the same locations.

### 1.4. The proposed method – contextual memory

The idea behind the algorithm is to encode probabilities in a memory-like structure. The probabilities are accessed by using a set of keys computed on a known context. Resilience to noise (since lossless compression for photographic images will mostly have to deal with noise) would be handled by allowing that not all the keys should find a match in the memory. The block scheme of the prediction method is presented in Figure 3.

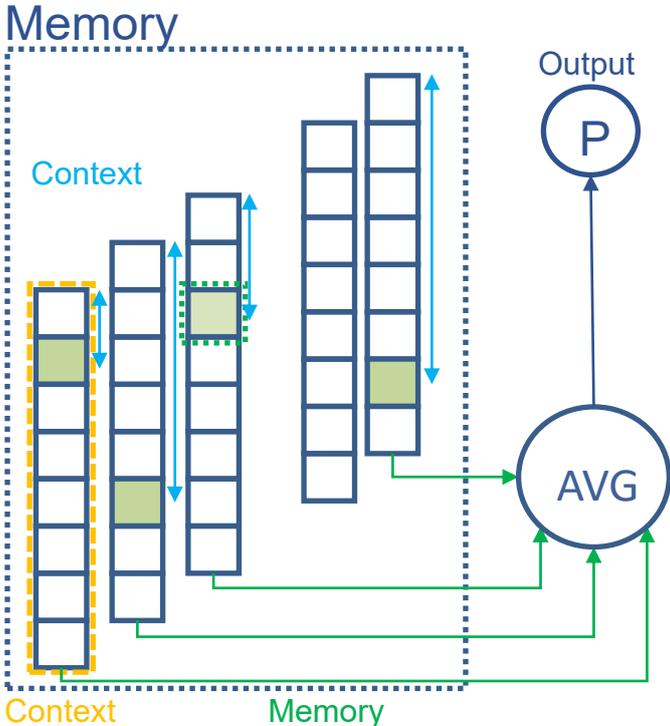


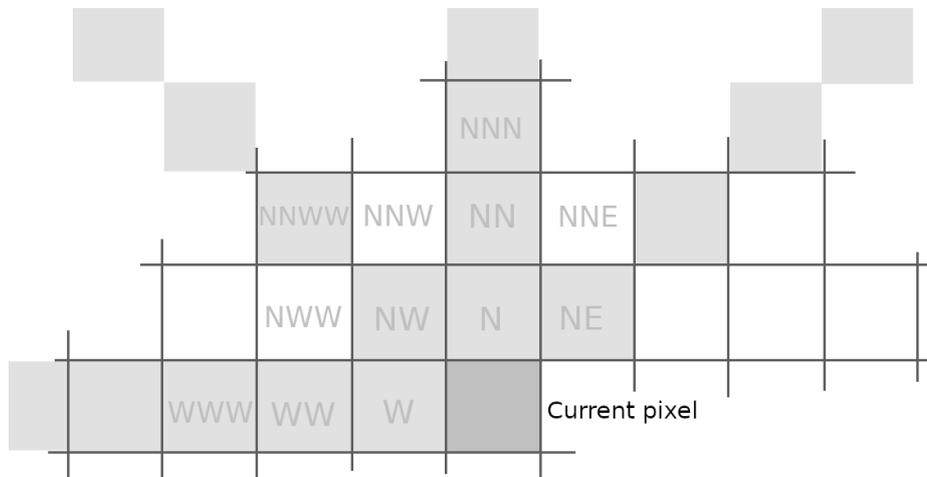
Figure 3 Block scheme of the proposed prediction using the contextual memory method

### 1.4.1. Context modeling

When it comes to predictive compression, we need to decide what best describes the part of the image we are currently trying to predict. This means that we need to look around the target pixel in the hope that the information will be enough to help a decision mechanism recognize which part of the image we are in and choose to use the appropriate representation of the internally created segmentation of the image.

Before continuing, we need to define the terms used in Figure 3. We denote by the word *context* the region of the image which participates in the prediction mechanism. *Context value* is the numeric value of the context, either a direct value or a function of that value, which will be used as index in the *memory* structure. The algorithm makes no assumption about the memory structure, but we provide some implementation details. The output of indexing the memory is the *memory value*.

We choose a simple model for contexts for predicting the bits of the pixels. We use rays in four directions and having various lengths, and the quantized derivatives along the rays. The rays are depicted in Figure 4. Since the pixels of the image are predicted from left to right, top to bottom, the only information we can rely on are known pixels, which means the directions are to the west, 45 degrees north-west, north and 45 degrees north-east. We choose rays of varying lengths from length 1 to 7, but use this as a parameter. The derivative with respect to the intensity value is computed as the difference between the consecutive pixels of the ray and quantizing is done by masking the lower order bits from the derivative. We use three levels of quantizing, each cutting out one more bit than the other. The current pixel participates in the contexts only with the currently known bits.



**Figure 4** Contexts as rays

In order to compute the context value from the contexts, we use a hashing function. We chose Fowler–Noll–Vo hash function (FNV) which is a non-cryptographic one byte at a time, however designed to compute fast and with a low collisions rate. It is found to be particularly suited for hashing nearly identical strings [37].

As an optimization, since we know that we will need to compute hashes for rays, we exploit the fact that FNV computes one-byte-at-a-time hashes and pass as input only the longest ray and output all the intermediate results. We apply the same optimization for the quantized derivatives of the rays.

## 1.4.2. Description of the contextual prediction

### 1.4.2.1. Model Prediction

To make a prediction, we propose the following algorithm:

We obtain a value from the memory for each context. One way to do that is to index the hash of the *context value* in a table

We average all the obtained *memory values*

Convert the average into a probability using the sigmoid function

$$p = \sigma \left( \frac{k}{n} \sum_{i=0}^n v_i \right), v_i = M[i][hash(c_i)] \quad (3)$$

$p$  is the output probability (that a bit is one),

$n$  is the number of input contexts,

$c_i$  is the *context value* of the  $i$ -th context,

$v_i$  is the *memory value* from the memory  $M$  for context  $i$ ,

$k$  is some ad-hoc constant

$\sigma$  is the sigmoid function.

### 1.4.2.2. Interpretation of values

Logistic regression is a way of combining probabilities when they are fed as input to the algorithm. Using stretched probabilities as input (applying logit function to them), logistic mixing becomes optimal for minimizing wasted coding space (Kullback-Leibler divergence) [34] because the weighting becomes geometric.

$$mix = \beta_0 + \beta_1 x_{1,i} + \beta_2 x_{2,i} + \dots + \beta_k x_{k,i} \quad (4)$$

With  $x_{j,i}$  being a probability, the equation becomes:

$$logisticMix = \beta_0 + \beta_1 t_{1,i} + \beta_2 t_{2,i} + \dots + \beta_k t_{k,i} \quad (5)$$

where:

$$t_{j,i} = \text{logit}(x_{j,i}) \quad (6)$$

The update formula for minimizing the relative entropy is:

$$\beta_j = \beta_j + \alpha * t_j * (y_i - p_i) \quad (7)$$

The set of weights carries a part of the predictive part of the ensemble, and they get updated to better represent the potential of individual components. In the case of PAQ8, the components gather statistics independently and the network independently mixes the statistics. Adding more weights to the mixer can result in improved predictive power since the model can discriminate better the contexts. But what if instead of separating the mixer from the statistics we shift the mixing information towards the components? How can we pass the mixing information to the weak learners of the ensemble?

So far, we know that the memory value  $v_i$  is taken from a memory structure. The index in the memory is computed based on the context value. But the feature is the context, not the memory value. We can assume that the *memory value* is

$$v_i = \beta_i * t_i \quad (8)$$

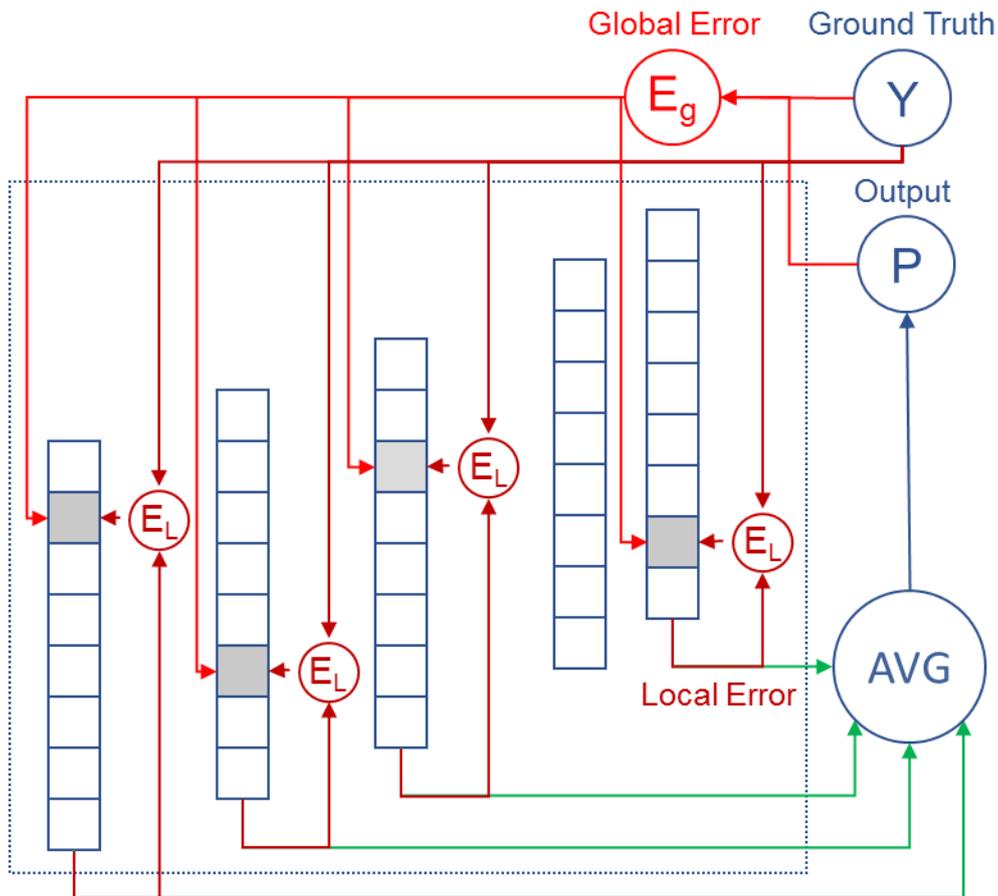
with  $t_i$  a stretched probability and  $\beta$  the weight of the probability in the ensemble. Computing the output probability resembles logistic regression, with the main difference that we apply averaging. The average in itself is a weighted stretched probability

$$average = \frac{k}{n} \sum_{i=0}^n (\beta_i * t_i) \quad (9)$$

which is converted into a regular probability by applying the sigmoid function.

### 1.4.2.3. Updating the model

The block scheme for updating the model is presented in Figure 5.



**Figure 5** Block scheme for the proposed update algorithm

In order to pass mixing information to the weak learners, we propose a dual objective minimization function:

In respect to the output of the network – global error

In respect to the output of the individual nodes (side predictions) – local error

Like PAQ8, we use reinforcement learning. Since we don't know the true value of the probability that a bit is 0 or 1 in a given context, we cannot use supervised learning. We back propagate the binary outcome in the network and try to minimize the cumulative logistic loss in an on-line manner. The square loss can be also used, but we are trying to minimize the wasted coding space.

For minimizing the logistic loss, the formula we used for the global error is:

$$E_g = \beta_g(p - y) \quad (10)$$

If we wanted to minimize the square loss, the formula would be:

$$E_g = \beta_g(p - y) * p * (1 - p) \quad (11)$$

with  $E_g$  as the global error,  $\beta_g$  the global error learning rate,  $p$  is the output probability of the entire network, and  $y$  the binary ground truth.

The local error is computed for each *memory value* in a similar fashion to the global error.

$$E_l = \beta_l * (p_i - y), p_i = \sigma(k_v * v_i) \quad (12)$$

with  $E_l$  as the local error,  $\beta_l$  the local error learning rate,  $p_i$  is the output probability for the  $i$ -th context (side prediction) multiplied by an ad-hoc value  $k_v$ , computed as the sigmoid of the *memory value*  $v_i$ , and  $y$  is the binary ground truth.

All the *memory values* are then updated by subtracting the local and the global errors:

$$v_i = v_i - E_l - E_g \quad (13)$$

Instead of updating weights of the mixture, we update directly the values which contribute to the average. We have no layer to separate the context weights from the input probabilities, making the method different from the context mixing algorithm.

### 1.4.3. Memory implementation and variations

The algorithm makes no assumption on how to organize the memory structure. We do not restrict the access to the memory to a precise scheme, but suggest three approaches, chosen as a compromise for speed and ease of use, and we describe here potential implementation and give more details to the implementation we chose to use. The proposed implementations are based on hash tables since they give fast retrieval given the fact that the context values are computed by hashing series of pixel values. We chose the 32-bit FNV hashing with table sizes the power of two so that indexing an entry will be done by masking. We use separate tables for each context so we have collision independence. The difference between the memory types comes from the way collisions and new entries are treated.

*simple lookup* - the simplest (and fastest) version, we ignore the potential collisions and average the memory values, multiply the result by an ad-hoc constant and then apply the sigmoid function:

$$p = \sigma(\text{sum} * c), c = \frac{k}{n} \quad (14)$$

where  $n$  is the number of contexts and  $k$  is an ad-hoc constant. Once the number of contexts becomes known,  $c$  becomes a constant and can be computed only once.

*tagged lookup* – for each *memory value* a small tag is added which is computed by taking the higher order bits of the context value. The memory location is computed by using the lower order bits from the context value. If a table address size is less than 32 bits, the remainder bits still can bring value to the indexing. The memory location is a pair of the memory value and the tag. If we don't have a hit, then we don't sum up the memory value. We also count the number of memory hits, because we will use this information when computing the probability. If the tag matches, we can use the value for the average.

$$p = \sigma\left(\frac{sum * k}{n_t}\right) \quad (15)$$

where  $n_t$  is the number of tag matches. In an empirical study we concluded that instead of simply averaging the values, we can get better results by dividing the sum by the average of the number of contexts and the number of tag matches. The formula becomes:

$$p = \rho\left(\frac{sum * k}{\frac{n_t + n}{2}}\right) \quad (16)$$

This is an approximate weighting of the confidence of the output based on how many inputs participate to the result.

On update, we update the tag of the location where it does not match. The value of the location can be reset to zero or the old value can be kept and the regular updated formula used. Keeping the old value sometimes gives better results and we believe it is because the collisions generated by noise can reset a very biased context value. This method uses more memory and has a more complex update rule, but gives better results than the simple lookup with the cost of improved computing complexity.

*bucket lookup* – the context value indexes a bucket with an array of tagged values. The selection of the memory value is done by searching the bucket for a matching tag. Inside the bucket we have an array of tagged locations. The memory value is linearly searched by tag. In this way we can implement complex replacement rules for the values inside the bucket. We provide a “least bias” eviction rule when no tag is matched in the bucket. This means kicking the location with the value closest to zero. In this way, we keep the values which can bring benefits to the compression. Computing the output and the update rules are the same as in tagged lookup. If the bucket size is kept small (4 to 8 entries), the linear search is done in the same cache line, making the speed comparable to the tagged lookup.

The tests we performed yielded different results for the proposed memory implementations. Each implementation should be chosen by taking into account the balance of speed versus the quality of prediction (from the first to the third).

As an optimization, instead of having the memory values represented as floating point numbers which occupy 32 bits, we quantize the value to a fixed-point represented by a short integer which only uses 16 bits. In the future, we could replace this representation with the FP16 standard. For tagged values we have found that 8bit values are enough, making the whole tag-value pair to be 24bits. In this way we can use lookup tables for computing the local error, since we know that the values are constricted to 65536 possibilities and we avoid the multiply and squash operations.

Memory implementations consider the 8 bits per byte structure of the image. This means that the buckets should be indexed using a proximity function for faster memory access. We

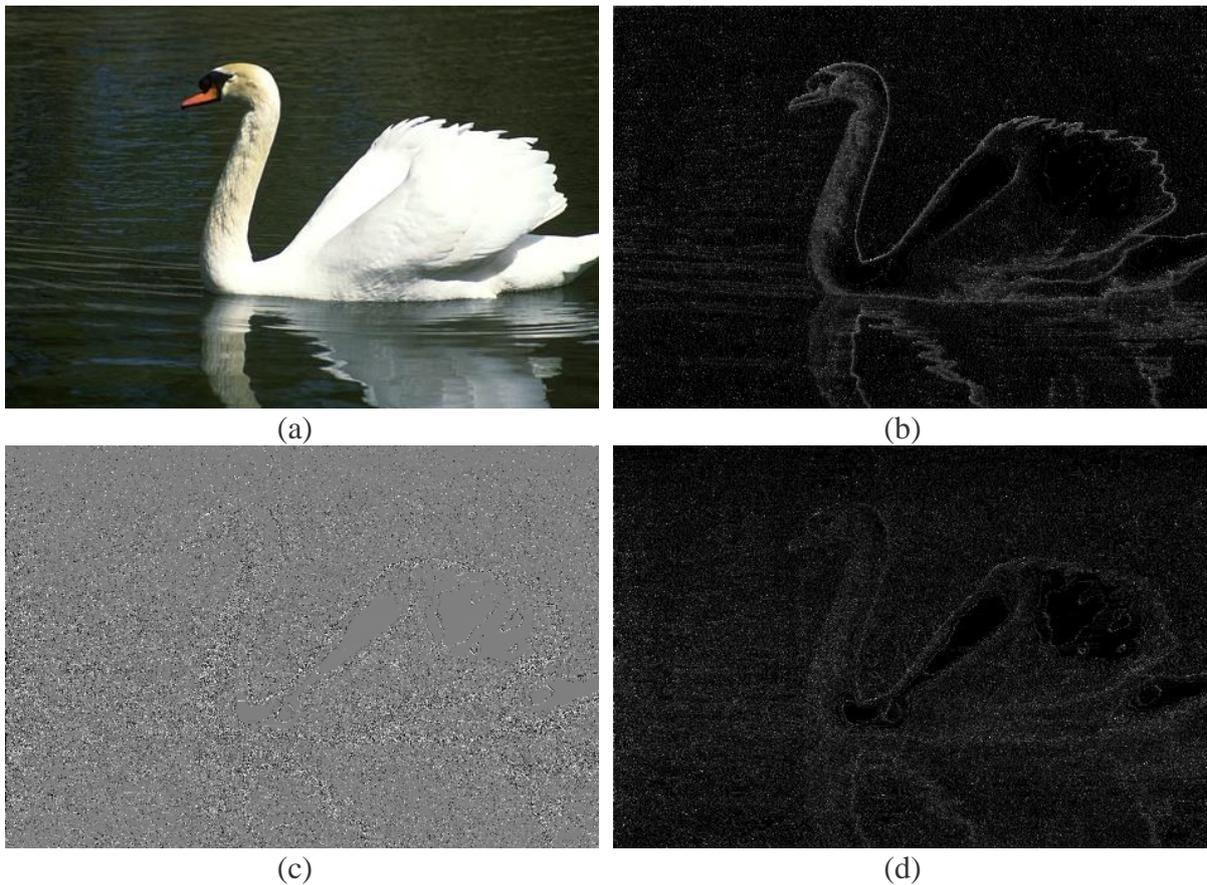
used the XOR function of the initial context value hash with the known partial nibble (4 bits of a byte) to create a new index. This ensures that the new context value will fall inside the same (or at most another) cache line of 64 bytes. After 4 bits, a new hash is computed with the known full nibble.

### **1.5. Quantifying the error**

We implemented a tool for evaluating the pixelwise compression improvements on the PAQ8PX prediction mechanism. The tool takes as input two maps coming from two different predictor version, and are the coding cost for each bit of the input stream. The cost for each group of consecutive 8 bits (or 24 bits for color images) can be mapped to a single value and squashed into the 0-255 interval to create a pixelwise loss map for a predictor version. The mapping function can be tuned to emphasize different aspects of the prediction, and thus help estimate the potential gains should one change parameters of the model. When two such maps are created, the difference of the two can be computed and then remapped into an image suitable for visual validation.

An example of such visualization can be seen in Figure 6. The pixelwise coding cost was computed using a simple average of the coding costs for all the bits belonging to a pixel. The output map was clamped and the result squashed into the 0-255 interval. The difference of coding costs was computed by subtracting the coding cost maps corresponding to the two versions of the predictor and then fit to a centered range for squashing into 0-255. The computed disagreement is the squared values of the difference, and it helps identify where the two predictors behave differently.

Besides the visualizations, the tool outputs other useful pixelwise information, such as the number of times a predictor got better loss, and the number of times a predictor got better loss on a given bit position in pixel.



**Figure 6** Quantified error for image "8068" from BSD500 [38] with (a) original image, (b) pixelwise coding cost, (c) high contrast centered difference of coding costs, (d) computed disagreement.

## 1.6. Experimental Results

### 1.6.1. PAQ8PX Contextual Memory implementation details

We implemented the contextual memory algorithm for the 8bpp lossless image compression. The application is written in the C++ programming language, since the paq8px was already implemented in this language. The code is compiled in Visual Studio and separates the original code, the changes required for compilation and the additional implementation into different commits, making clear which part is which.

The source code containing a full implementation of the algorithm is publicly available at the GitHub page [39], repository [40].

### 1.6.2. Evaluation on the benchmarks

In order to test the effectiveness of the algorithm, we applied the augmented version of the PAQ8PX algorithm with Contextual Memory to four test sets. The command line option for the compression for all the images selects memory level 8 and adaptive learning rate (-8a).

The Waterloo image compression benchmark [41] contains two test sets with image sizes from 256 x 256 pixels up to 1118 x 1105. For this benchmark, we used as parameters for the

contextual memory ray length 5 and memory size 20. The results for the two test sets are presented in Table 1 and Table 2.

A newer benchmark, Sachin Garg’s test images corpus [42], contains images ranging 2268 x 1512 pixels up to 7216 x 5412. For this benchmark, we used as parameters for the contextual memory ray length 7 and memory size 23. The results are presented in Table 3.

We also included the Squeezechart 8bpp grayscale images test set [43] where four of the five images are medical images. For this benchmark, we used as parameters for the contextual memory ray length 6 and memory size 20. The results are presented in Table 4. The best results in the table are highlighted using bold font weight.

We used as learning constants global learning rate  $\beta_g=0.9$  and local learning rate  $\beta_l=0.1$  and the constants  $k$  and  $k_v$  were set to 0.4.

The results in Table 1, Table 2, Table 3, and Table 4 are expressed in bits per pixel which is an image size independent absolute measure of the compression ratio. It represents the average number of bits needed to encode the pixel information from an image. It is computed as the compressed size of the image divided by the number of pixels. This is not to be confused with bits per byte, which measures the compressed ratio of a general file, though in our case the two values coincide since the size of a pixel in an 8bit color depth grayscale image is one byte. The header of the compressed file should be excluded when computing the bits per pixel, but it is not always the case since the header is usually a minor payload compared to the content. However, it should be specified if the header is included or not in computing the bits per pixel so that the results can be verified. The proposed algorithm’s bpp was computed with a tool we created, available on the GitHub page [44], and includes the header size in the computation.

**Table 1** Waterloo gray test set 1

Set	JPEG 2000	JPEG- LS	MRP	ZPAQ	Vanilc WLS D	Paq8px167	Paq8px167+CM (proposed)
<b>bird</b>	3,6300	3,4710	3,2380	4,0620	2,7490	<b>2,6073</b>	2,6077
<b>bridge</b>	6,0120	5,7900	5,5840	6,3680	5,5960	5,5074	<b>5,5037</b>
<b>camera</b>	4,5700	4,3140	3,9980	4,7660	3,9950	3,8176	<b>3,8173</b>
<b>circles<sup>1</sup></b>	0,9280	0,1530	0,1320	0,2300	0,0430	<b>0,0281</b>	0,0282
<b>crosses<sup>1</sup></b>	1,0660	0,3860	0,0510	0,2120	0,0160	0,0176	<b>0,0171</b>
<b>goldhill1</b>	5,5160	5,2810	5,0980	5,8210	5,0900	5,0220	<b>5,0197</b>
<b>horiz<sup>1</sup></b>	0,2310	0,0940	0,0160	0,1220	0,0150	<b>0,0139</b>	0,0140
<b>lena1</b>	4,7550	4,5810	4,1890	5,6440	4,1230	4,1302	<b>4,1293</b>
<b>montage<sup>1</sup></b>	2,9830	2,7230	2,3530	3,3350	2,3630	2,1505	<b>2,1501</b>
<b>slope<sup>1</sup></b>	1,3420	1,5710	0,8590	1,5040	0,9600	<b>0,7186</b>	0,7194
<b>squares<sup>1</sup></b>	0,1630	0,0770	0,0130	0,1770	<b>0,0070</b>	0,0129	0,0128
<b>text<sup>1</sup></b>	4,2150	1,6320	3,1750	0,4960	0,6210	0,1053	<b>0,1052</b>
<b>Average</b>	2,9510	2,5060	2,3920	2,7280	2,1310	2,0109	<b>2,0103</b>

<sup>1</sup> Non-natural / artificially generated image

Table 2 Waterloo gray test set 2

Set	JPEG 2000	JPEG- LS	MRP	ZPAQ	Vanilc WLS D	Paq8px167	Paq8px167+CM (proposed)
<b>barb</b>	4,6690	4,7330	3,9100	5,6720	<b>3,8710</b>	3,9319	3,9297
<b>boat</b>	4,4150	4,2500	3,8720	4,9650	3,9280	3,8165	<b>3,8145</b>
<b>france<sup>1</sup></b>	2,0350	1,4130	0,6030	0,4220	1,1590	0,0992	<b>0,0966</b>
<b>frog</b>	6,2670	6,0490	_ <sup>2</sup>	3,3560	5,1060	2,4656	<b>2,4581</b>
<b>goldhill2</b>	4,8470	4,7120	4,4650	5,2830	4,4630	4,4227	<b>4,4214</b>
<b>lena2</b>	4,3260	4,2440	3,9230	5,0660	3,8680	3,8608	<b>3,8604</b>
<b>library<sup>1</sup></b>	5,7120	5,1010	4,7650	4,4870	4,9110	3,3253	<b>3,3200</b>
<b>mandrill</b>	6,1190	6,0370	5,6790	6,3690	5,6780	5,6364	<b>5,6339</b>
<b>mountain</b>	6,7120	6,4220	6,2210	4,4930	5,2150	4,0799	<b>4,0744</b>
<b>peppers2</b>	4,6290	4,4890	4,1960	5,0950	4,1740	4,1493	<b>4,1470</b>
<b>washesat</b>	4,4410	4,1290	4,1470	2,2900	1,8900	1,7478	<b>1,7466</b>
<b>zelda</b>	4,0010	4,0050	<b>3,6320</b>	4,9200	3,6330	3,6437	3,6435
<b>Average</b>	4,8480	4,6320		4,3680	3,9910	3,4316	<b>3,4288</b>

\_<sup>1</sup> Non-natural / artificially generated image

\_<sup>2</sup> Supported image size only multiple of eight

Table 3 Imagecompression.info 8bpp gray new test images

Set	JPEG 2000	JPEG- LS	MRP	ZPAQ	Vanilc GraLIC	WLS D	Paq8px167	Paq8px167+CM (proposed)
<b>artificial<sup>1</sup></b>	1,1970	0,7980	0,5170	0,6730	0,4464	0,6820	0,3188	<b>0,3186</b>
<b>big_building</b>	3,6550	3,5920	_ <sup>2</sup>	4,3350	3,1777	3,2430	3,1250	<b>3,1216</b>
<b>big_tree</b>	3,8050	3,7320	_ <sup>2</sup>	4,4130	3,4080	3,4680	3,3823	<b>3,3803</b>
<b>Bridge</b>	4,1930	4,1480	_ <sup>2</sup>	4,7250	3,8700	3,8420	3,7958	<b>3,7953</b>
<b>cathedral</b>	3,7100	3,5700	3,2600	4,2390	3,1900	3,3020	3,1539	<b>3,1519</b>
<b>Deer</b>	4,5820	4,6590	_ <sup>2</sup>	4,7280	4,3116	4,3760	4,1788	<b>4,1750</b>
<b>fireworks</b>	1,6540	1,4650	1,3010	1,5550	1,2500	1,3640	<b>1,2324</b>	1,2325
<b>flower_foveon</b>	2,1980	2,0380	_ <sup>2</sup>	2,4640	1,7761	1,7470	1,6944	<b>1,6943</b>
<b>hdr</b>	2,3440	2,1750	1,8540	2,5890	1,9197	1,8730	1,8330	<b>1,8327</b>
<b>leaves_iso_200</b>	4,0830	3,8200	3,4000	4,7430	3,2630	3,5370	4,0509	<b>4,0473</b>
<b>leaves_iso_1600</b>	4,6810	4,4860	4,1860	5,2600	4,0720	4,2430	3,2168	<b>3,2130</b>
<b>nightshot_iso_100</b>	2,3000	2,1300	1,8390	2,5760	1,8240	1,8750	1,7811	<b>1,7805</b>
<b>nightshot_iso_1600</b>	4,0380	3,9710	3,7430	4,2680	3,6610	3,7820	3,6295	<b>3,6272</b>
<b>spider_web</b>	1,9080	1,7660	<b>1,3490</b>	2,3640	1,4441	1,4220	1,3498	1,3502
<b>zone_plate<sup>1</sup></b>	5,7550	7,4290	2,8340	5,9430	0,8620	0,9110	0,1257	<b>0,1257</b>
<b>Average</b>	3,3400	3,3190		3,6580	2,5650	2,6500	2,4579	<b>2,4564</b>

\_<sup>1</sup> Non-natural / artificially generated image

\_<sup>2</sup> Supported image size only multiple of eight

**Table 4** Squeezechart 8bpp grayscale

Set	MRP	cmix v14f	GraLIC	Paq8px167	Paq8px167+CM (proposed)
<b>blood8</b>	2,1670	2,1600	2,3200	2,1308	<b>2,1304</b>
<b>cathether8</b>	<b>1,5350</b>	1,5351	1,6580	1,5382	1,5380
<b>fetus</b>	4,0650	3,9730	4,1310	3,8236	<b>3,8225</b>
<b>shoulder</b>	<b>2,8660</b>	2,9080	3,1130	2,8697	2,8676
<b>sigma8</b>	2,6870	2,6290	2,7200	2,6266	<b>2,6263</b>
<b>Average</b>	2,6640	2,6410	2,7880	2,5978	<b>2,5970</b>

### 1.6.3. Discussion on the results

The reason for choosing these benchmarks is that they are publicly available and that they are provided without conflicts of interest. They contain images of various types such as artificially generated, edited, photographs, scans. This makes them suitable for publications and there are published papers using them, such as [21]. We present our results on all the images in the datasets in order to prove that we did not tune the algorithm to a selected few. The images are compressed separately (as in *not a solid archive*) to prevent reusing correlations.

The PAQ8 family of algorithms was designed to achieve good compression ratios at the expense of a long compression time and a large memory footprint. Even though there are some optimizations applied, the running time will be much larger than some of the other algorithms. Furthermore, the contextual memory algorithm does not contain too many speed optimizations in its provided form. Therefore, a running time comparison is out of the scope of this thesis, but in order to give the reader a sense of the execution time scale, we provide a relative comparison on the image *lena2* from the Waterloo gray test set (see Table 5). The algorithms were run on the same processing architecture.

**Table 5** Compression running time comparison on image *lena2* (expressed in seconds)

Image	MRP	JPEG 2000	JPEG- LS	GraLIC	Paq8px167	Paq8px167+CM (proposed)
<b>lena2</b>	258s	0.04s	0.02s	0.25s	12s	24s

The compression running time does not equal the decompression running time for all the algorithms. For instance, the MRP algorithm is highly asymmetric due to its multiple pass optimizations, decompression of the same image taking only 0.6 seconds. The timings were measured using the x64 version of the Timer 14.00 tool created by Igor Pavlov and available for public domain on the 7-cpu website [45].

Memory requirements depend on the compression parameters. PAQ8PX reports using 2493MB of memory for command line parameter `-8a`. The contextual memory can increase the resources needed depending on the parameters set. We can estimate the memory consumption of the current implementation by multiplying the number of rays by: ray length, table size ( $2^{\text{memory size}}$ ), 4 (no quantization plus 3 quantized derivatives), 3 (number of bytes per memory

location). For ray length 5 and memory size 20 we estimate that it adds another 240MB of memory.

The results in Table 1, Table 2, and Table 3 for all the compressors, except Paq8px167 and the proposed method, were taken directly from [21]. The results in Table 4 were taken from the PAQ8PX thread [31]. The results for GraLIC and the results for Paq8px167 and the proposed method are computed using a tool we created for this purpose, available at [44]. The tool does not exclude the file headers when computing the bits per pixel.

## 1.7. Conclusions

This chapter provides a description of the state-of-the-art compression program PAQ8PX from the point of view of grayscale image compression. The main contribution of this chapter is an application agnostic algorithm for predicting probabilities based on the contextual information available with learning done in an on-line fashion. The usefulness of the algorithm is demonstrated by integrating it with the PAQ8PX algorithm and testing it on several image compression benchmarks. The results show an overall compression ratio improvement across all the datasets without having special crafted features. One important difference from existing ensemble mixing algorithms is that, in our algorithm, we assume that various contexts apply together and the prediction benefits from the synergy of the side predictions, unlike the ensembles which assume model independence.

In its current state, the algorithm was not applied to color or 16 bpp images. In the future, we intend to extend the algorithm for three-dimensional medical image compression and support 16 bpp depth. Context modeling can be done similarly to the work described in [25].

The architecture chosen for the algorithm does not take into account parallel optimization techniques which are suitable for hardware implementation like the CCSDS developed image compression algorithms [27]. Unlike the CCSDS standard, the focus for PAQ8 is set to provide the unconstrained liberty for exploring techniques to improve the compression ratio. However, if one wants to standardize an epoch of development, the code can be converted to the ZPAQ open standard [46] which, among other things, aims for forward and backwards compatibility of archives. Although not presented in this chapter, in future research papers we can compare the compression ratio with the CCSDS family of algorithms.

In [47], a context function is proposed to statistically discriminate error residual classes of full pixel prediction, useful for lossless and near-lossless compression schemes. Due to its properties, such a function could be integrated within the context mixing layer and the adaptive probability maps.

Design space exploration needs to be done for context modeling. Moving contexts from other modeling types to the contextual memory predictor might also bring benefits. The side predictions of the algorithm can be also be passed to the context mixing network. Splitting various contexts into two or more contextual memory structures can also lead to finding better correlations.





## 2. A novel contextual memory algorithm for edge detection

### 2.1 Overview

One of the subclasses of image segmentation is thin segmentation. In this category are both centerline segmentation and edge detection. The second chapter of this thesis focuses on thin segmentation.

The process of segmentation selects a set of pixels from an image, based on rules and patterns. The labeling of the extracted sets, allows the user to obtain more information from the image. Typical rules for segmentation include the grouping of pixels by color, intensity or texture. Nevertheless, by targeting information extraction, rules help finding edges of objects, areas, specific shapes and volumes, when applied to stacks of images. Automatic annotation of images and video, gains more support each year.

Practical applications of image segmentation include: machine vision, control systems, object detection (for example face detection, pedestrian detection), recognition tasks (for example fingerprint and iris recognition), and medical imaging. Medical image segmentation borrows from many general-purpose segmentation techniques but combines them with domain specific knowledge in order to obtain better results. Shape analysis and volume evolution make medical image segmentation important for diagnostics and treatment plans.

Recent methods, like deep learning through Convolutional Neural Networks (CNN), proved to give state of the art results in 2D benchmarks, but neural network architectures for 3D image processing are only now starting to emerge. Medical image segmentation integrated these techniques for both 2D [48] and 3D [49] segmentations.

In this chapter we introduce a general algorithm for automated learning, which stands as a basis for various applications. We provide a description of the algorithm and point out differences from various implementations tested. To prove the effectiveness, we have applied and tested the algorithm on an edge detection benchmark, with promising results. To this date, the algorithm was applied only for 2D images, but in the future, we plan to extend it for volumetric images, as an application in medical imaging. The research was published in [3].

### 2.2 Related work

Edge detection has been a subject of research for many years, with papers as early as 1975 [4]. Since then, a large number of techniques have been approached, targeting different aspects of edge detection, like closed contours, human-like perception, or fast detection. In the following subchapters, we provide an overview of more recent methods used, organized by the main strategy of the algorithm.

In [38], a method based on oriented gradient signals is described. These are obtained from splitting the input image into the three CIELAB color channels and a texture channel. After applying filtering on the channels with 17 Gaussian kernels, the results are clustered using a K-means algorithm. An image is formed using the result for each pixel on which a non-maximal

suppression is applied. So far, only local information was used for each pixel. For the global information spectral clustering is used. The probability of a pixel belonging to the contour is a weighted combination of the local and global information.

Realtime framerates have been recently achieved using random decision for edge detection [50]. Local image patches were used as a basis for learning structure labels. These are then mapped into a discrete space using a decision tree which splits the data based on a decision function. Learning was done by independently training the trees in a recursive manner using information gain criteria.

Deep learning architectures started to prove state of the art results at impressive processing speed of 0.4s using the Holistically-Nested Edge Detection (HED) architecture [51]. The networks map whole image to image predictions which provide a significant advantage as a global information method. A convolutional neural network with hierarchical representations provides the high-level features, visible as side outputs of the network. Deep supervision is used for training the layers, which are blended using a weighted fusion layer to provide a final image.

Relaxed deep supervision (RDS), proposed in [52], also relies on deep learning, but the main difference from HED is that RDS accepts as input predictions from other edge detectors such as Canny, called relaxed labels. HED itself is used as a provider for relaxed labels. The network tries to eliminate most of the false positives from all the intermediate layers.

The current state of the art is another recent deep learning approach based on Richer Convolutional Features [53]. Similar to HED, it has a phase of producing side outputs, but in this case a VGG16 architecture was used. Instead of using only the final convolutional layers for merging like previous approaches did, this architecture encapsulates, in a holistic manner, features from all convolutional layers and then trains the network via back propagation.

Although the research on deep network focuses only on developing network architectures and not new techniques, other approaches are continually tested in the literature with promising improvements for low level features, with the main advantage being that you don't need a powerful video card to run these algorithms, and, thus, they can run on most equipment and the results are good enough for further processing. One such algorithm which improves the Canny edge detector is described in [54]. After applying an improved anisotropic diffusion filter, gradient templates are used for four directions. Subsequently, an adaptive threshold is computed based on the histogram of the image, making the output more resilient to noise.

Another algorithm of this type revolves around hierarchical graph-partitioning [55]. The algorithm employed is called Divide-and-Link and it is used for a hierarchical network clustering. Unlike our proposed method, pixels are here modeled as nodes in a graph and a dissimilarity order between pixels determines the clusters in the graph. The regions are then transformed into a boundary map with a selection of the largest area of neighbor regions to provide the border.

### **2.3 Berkeley Edge Detection Benchmark**

The Berkeley Computer Vision Group provides a public benchmark for contour detection and image segmentation [38]. The provided contains 500 natural images, split into 300 training and validation images and 200 testing images.

Ground truth is provided from handmade human annotations from five different subjects on average for each image. An example image from the benchmark, along with its ground truth, is presented in Figure 7.



Figure 7 (a) Example BSD500 test image “2018” with (b) ground truth

## 2.4 Basis for the contextual memory and the processing pipeline

Before detailing the proposed algorithm, we will present some methods and techniques which stand as a basis for the concept of contextual memory and the processing pipeline.

### 2.4.1 Logistic Regression

For a regression model, where the dependent variable is a categorical, logistic regression can be used. In our case, the outcome is binary, where an image pixel belongs to a certain group or not. Binary logistic regression is useful for estimating the probability of class membership. Useful for making such predictions, a single layer neural network has the output probability:

$$P_r(Y_i = 1|X_i) = p_i = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_{(1,i)} + \beta_2 x_{(2,i)} + \dots + \beta_k x_{(k,i)})}} \quad (1)$$

where  $p_i$  is the probability that the output  $Y_i$  belongs to the class, defined as a sigmoid function of a linear combination of the  $k$  explanatory variables  $X$ , and  $\beta_j$  for  $j = 1 \dots k$  are the parameters to be estimated, usually called coefficients or weights. The sigmoid function takes

any input  $x \in \mathcal{R}$  and outputs a value between zero and one, making it interpretable as a probability. This function is also preferred because it has a continuous derivative:

$$y = \frac{1}{1+e^{-f(x)}}, \frac{dy}{dx} = \frac{y(1-y)df}{dx} \quad (2)$$

The inverse of the logistic function, sometimes called the stretch function, is defined as:

$$g(t) = \ln\left(\frac{t}{1-t}\right) \quad (3)$$

There are numerous numerical ways to estimate the coefficients [56], relevant here is the Stochastic Gradient Descent. Minimizing a function by following the gradient of the cost is called Gradient Descent. If the loss is accounted for the entire training set or a subset of the training set, the method is called Batch Gradient Descent. If the batch is the size of one, we will have a Stochastic Gradient Descent. For each instance  $i$  of the training set, we will make a prediction and then suffer a loss. If we apply the backpropagation algorithm, the weights will be updated according to the following rule:

$$\beta_j = \beta_j + \alpha x_j (y_i - p_i) p_i (1 - p_i) \quad (4)$$

where  $\beta_j$  is the weight for the  $j$ th input variable  $x_j$ ,  $p_i$  is the output probability for the instance  $i$ , and  $y_i$  is the label of the instance (0 or 1). The learning rate  $\alpha$ , usually chosen empirically, limits the amount of correction for each coefficient. This is the gradient descent in weight space which minimizes the root mean square error.

Instead, if we want to minimize the relative entropy:

$$loss = -\log(1 - p_i) \text{ if } y_i=1 \text{ or } loss = -\log(p_i) \text{ if } y_i=0 \quad (5)$$

then we will apply equation 6:

$$\beta_j = \beta_j + \alpha x_j (y_i - p_i) \quad (6)$$

This minimizes the amount of information lost when a prior probability distribution  $Q$  is used to approximate a posterior probability distribution  $P$ .

## 2.4.2 Ensemble learning

Combining multiple hypotheses in order to obtain a better hypothesis is called ensemble learning. Ensembles are supervised learning meta-algorithms which, after training, can be used to make predictions. The ensemble also represents a hypothesis. The set of hypotheses which compose the ensemble do not necessarily contain the output hypothesis, making it likely to better describe the data. If not carefully prevented, this can lead to overfitting the training data.

As expected, using ensemble methods require more resources (memory and computation time) than single models. Various techniques make a tradeoff between the speed of computation, amount of memory used and the accuracy of the model.

In the process of designing artificial neural networks, creating multiple models and combining them is called ensemble averaging. The ensemble should perform better than the individual models because the error averages out. Instead of picking the prediction of only one of all the models as many ensemble techniques do, all the models are kept and the ones which

are more prone to error are assigned a smaller weight. This can be expressed as a linear combination of experts. The output of the ensemble can be computed as in equation 7.

$$y(X; \alpha) = \sum_{j=1}^k \alpha_j y_j(x) \quad (7)$$

with  $\alpha$  a set of weights,  $y_j$  as the  $j$ th model prediction. Numerically optimizing  $\alpha$  is already a solved problem when applying the neural network learning rules.

The properties of the models upon which the ensemble averaging is built upon are [57] :

“  
 In any network, the bias can be reduced at the cost of increased variance.  
 In a group of networks, the variance can be reduced at no cost to bias.  
 ”

False assumptions in the learning algorithm lead to bias error. High bias leads to missing the correlations between the data features and the target outputs (underfitting). Sensitivity to small fluctuations in the training set causes variance error. High variance can cause overfitting, meaning the model has learned mostly noise instead of generalizing for unseen data. Given trained models with low bias but high variance, the result of the ensemble averaging is expected to have both low bias and low variance.

One of the variants of ensemble averaging is the Negative Correlation Learning. This algorithm attempts to train and to combine individual networks in an ensemble in the same learning process [58].

Boosting is meta-algorithm which aims to create a strong learner from a collection of weak learners. As a rule of thumb, a series of weak classifiers are trained in respect to a probability distribution and are added to the set which is the basis for the strong classifier. This sequential introduction of weak learners keeps them focused on the samples previous learners misclassified.

AdaBoost, short for "Adaptive Boosting", is a type of ensemble learning. A boost classifier has the following form:

$$F_T(x) = \sum_{t=1}^T f_t(x) \text{ with } f_t(x) = \alpha_t h(x) \quad (8)$$

$f_t(x)$  is a weighted weak learner, while  $x$  is the sampled input. For a binary classification task, the output of the learner is considered of class 0, if the value is negative or class 1 if the value is positive. The absolute value of the output should be interpreted as confidence in the result.

The output of a weak learner for the sample  $i$  is called the hypothesis  $h(x_i)$ . At iteration  $t$ , a weak learner is chosen and weighted with a coefficient  $\alpha_t$ . The value of  $\alpha_t$  should minimize  $E_t$  which is the training error at stage  $t$ . The error is computed using equation 9.

$$E_t = \sum_i E(F_{t-1}(x_i) + \alpha_t h(x_i)) \quad (9)$$

with  $F_{t-1}(x)$  is the boosted classifier built up to the previous stage and  $E(F)$  is an arbitrarily chosen error function. Recently, Convex Potential Boosters received criticism regarding convergence where random classification noise is present [59].

Stacking refers to blending the predictions of multiple machine learning models. With a specifically given combiner algorithm, stacking can represent most if not all ensemble techniques. Usually, a logistic regression layer is used as the combiner.

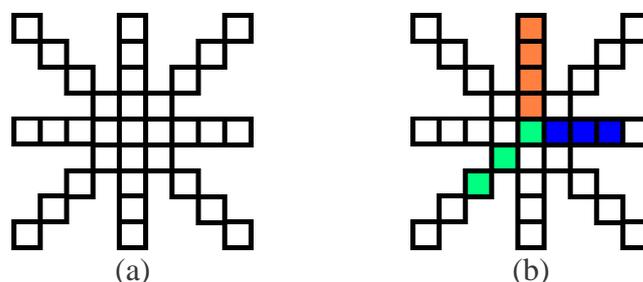
### 2.4.3 Context modeling

Context modeling describes how the context information is structured and maintained. Depending on the problem, different types of discriminator contexts are useful. Local context refers the aspect of the data examined, as opposed to context value, which represents the numeric value of that context. The context value will be used for accessing the memory structure. The memory takes a context value as input, and outputs a value used for computing the output probability of belonging to a certain class.

When discussing edge detection and describing whether a pixel belongs to an edge or not, one of the most important aspects to take into consideration are the neighboring pixels. For instance, as a context, we can take the top pixel and left pixel. The context value for this example would be top pixel intensity 255, left pixel intensity 20.

Choosing neighboring pixels means using a local context, because we do not input the entire image when deciding if the focused pixel is an edge or not. Even more, there is no clear rule on how far away we should look when making such an assumption.

We can define contexts as rays, starting from the focused pixel and going in a straight line away in a given direction. Rays are defined by direction and length. Rays could be chosen from 1 to  $L$ , the maximum context length. We define direction by the angle of the ray, and we choose the angle by dividing 360 degrees by the number of rays we want to use. For example, if we use 8 rays, we have 4 rays, one for each axis, and 4 rays for the diagonals, as in Figure 8, for length 5 and 8 rays:



**Figure 8** (a) Position of pixels as rays of length 5; (b) ray length 3 (green), length 4 (blue), length 5 (orange)

The context value needs not be only pixel values; we can take any function of the pixel values as well. Instead of directly using the values of the pixels, we can take the numerical derivative of the pixels in the direction of the ray. Of course, one can mask or quantize the values of the pixels or the value of the derivative. In the case of color images, rays for each color channel can be individually taken as contexts.

### 2.3.3 Resources and hashing as a solution

If we allow contexts of any length to be used, we quickly hit the wall of practical limitations. Take for instance context values as pixels from a color channel which are represented as bytes. A context of length only 4 already means  $2^{32}$  possible values. But not all the values will be relevant, as most of the possible values will represent noise.

To overcome this issue, we use hashing functions to map them to the chosen data structures. A hashing function maps data of arbitrary size to data of fixed size. This makes it useful for data structures like hash tables. One-at-a-time hash functions are useful for data which comes in byte chunks, especially when we want to keep the intermediary hash values.

For the current work, we analyzed in particular two different types of hash functions: Jenkins hash function [60] and Fowler–Noll–Vo hash function [61]. Both of them are non-cryptographic but were chosen for their speed of computation and rather low collision rate.

The Jenkins hash functions, named after Bob Jenkins was formally published in 1997. We provide pseudocode on how to compute a variant of this hash:

```
jenkins_one_at_a_time_hash(*data, length)
    hash = 0
    for (i = 0; i < length; i++)
        hash += data [i]
        hash += hash << 10
        hash ^= hash >> 6
        hash += hash << 3
        hash ^= hash >> 11
        hash += hash << 15
    return hash
```

Fowler–Noll–Vo hash functions (short FNV) are named after Glenn Fowler, Landon Curt Noll, and Kiem-Phong Vo. We provide pseudocode on how to compute the FNV-1a variant of the hash:

```
FNVA_one_at_a_time_hash(*data, length)
    hash = FNV_offset_basis
    for (i = 0; i < length; i++)
        hash = hash XOR data[i]
        hash = hash × FNV_prime
    return hash
```

*FNV\_offset\_basis* and *FNV\_prime* are two given constants chosen depending on the output hash length. For instance, for 32 bit hashing, the values are 2166136261 and 16777619 respectively.

Notice that while the Jenkins Hash has more instructions per byte, it uses only xor, add and shifts, as opposed to FNV-1a which uses multiplication. Both functions can be altered to keep the intermediate results of the hash value, useful when working with rays of increasing length.

Both functions can be altered to keep the intermediate results of the hash value, useful when working with rays of increasing length.

## 2.4 An original method for contextual prediction

Applied on images, the context modeling part of the algorithm takes as input an image and a position. The position is used as a basis for the rays which are modeled as position deltas

from the focused pixel. The values of the pixels are taken from the color channel of the image and are then hashed. The result of the hash is the context value, which we later use for indexing. As a result, we have as many context values (numbers) as the number of contexts.

The memory can be organized as a map with a separate table for each context in order to prevent collisions between contexts. There are more ways to organize the memory, and some suggestions will be discussed in the implementation details.

### 2.4.1 Model Prediction

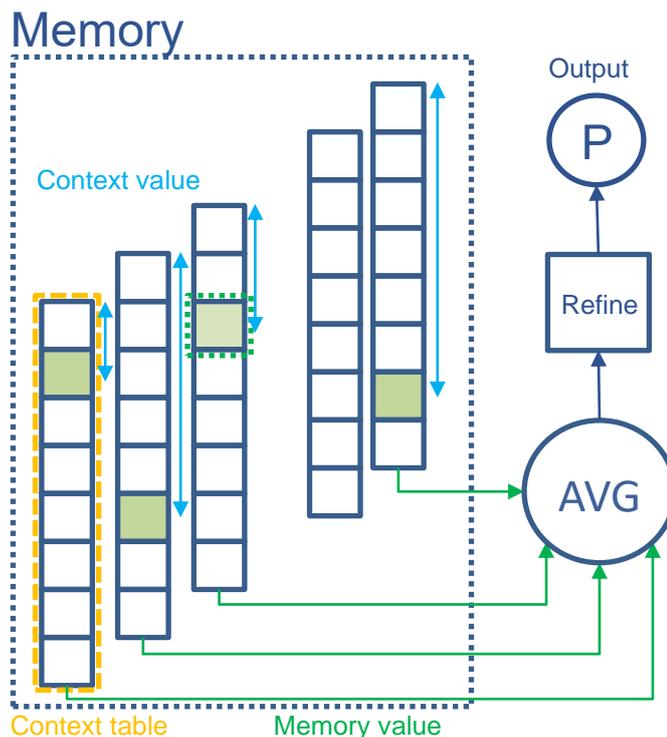
In order to make a prediction, we have proposed the following algorithm:

- We obtain a value from the memory for each context. One way to do that is to index the hash of the context value in a table
- We average all the obtained memory values
- Convert the average into a probability using the sigmoid function
- Refine the probability using a transfer function and obtain the output probability (equation 10)

$$p = T \left( \sigma \left( \frac{k}{n} \sum_{i=0}^n v_i \right), C \right) \text{ with } v_i = M[i][\text{hash}(c_i)] \quad (10)$$

Where  $p$  is the probability that the pixel belongs to a class (output probability),  $n$  is the number of input contexts,  $c_i$  is the context value of the  $i$ -th context,  $v_i$  is the value from the memory  $M$  for context  $i$ ,  $k$  is some ad-hoc constant,  $T$  is an adaptive probability map transfer function which takes as input a probability and a small context value  $C$  and outputs a refined probability and  $\sigma$  is the sigmoid function.

The prediction scheme used for each of the pixels from one layer is detailed in Figure 9. It adds a probability refinement from the proposed algorithm in Chapter 1.



**Figure 9** The prediction scheme for each pixel of one channel of a layer

The adaptive probability map (APM), sometimes called secondary symbol estimation (SSE), is used to fine tune a probability, and works in the following way:

Select a set of interpolation points according to a context value  $C$ :  $points = pointset[C]$

Find the two points indexes between which the input value falls: index low and index high

Output the probability as the weighted average of the two values from the points. The weight is selected from how far the input is from the two points:

$$output = points[index\ low] * (1 - weight) + points[index\ high] * weight \quad (11)$$

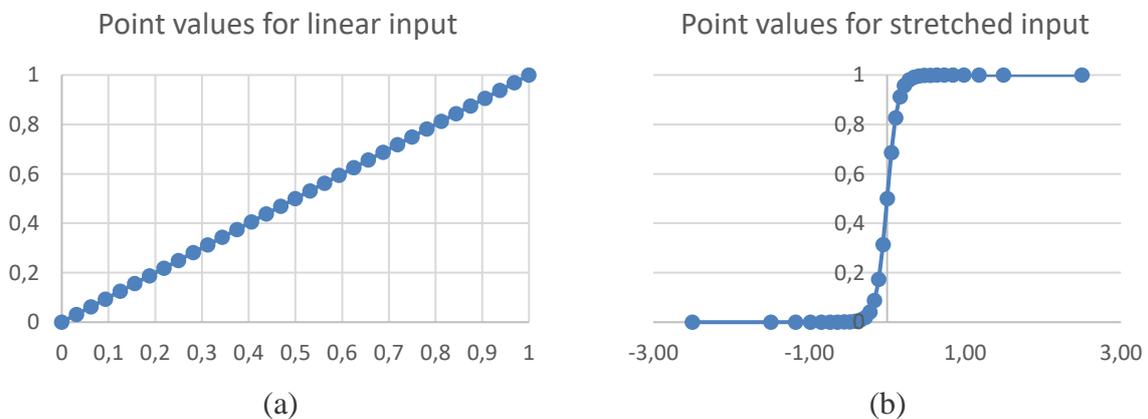
The input probability  $p$  can be mapped to point indexes in more ways. A simple way is to quantize the probability linearly to the number of points  $Np$ . This is equivalent to

$$index\ low = [p * Np], index\ high = [p * Np] + 1 \quad (12)$$

where  $[ ]$  denotes the *floor* operation.

Another way to quantize the probability is to stretch the probability first and then quantize linearly to the number of points. This serves the purpose to allocate more points close to zero and one, where fine tuning makes more sense. The input value for the APM is now a stretched probability.

The following two diagrams from Figure 10 plot the initial point values for the two methods described in respect to the input. When no value was changed, the output of the APM should be equal to the input probability. The X axis represents the input value to be quantized, and the Y axis represents the point values (probability).



**Figure 10** (a) uniform distribution of points; (b) non-uniform distribution of points

In logistic regression, every feature is individually weighted before applying the sigmoid function to the result. There is no assumption about the origin of the numeric value of the feature. When we know that the input features are probabilities, logistic regression can be seen as a way of combining them. Mattern [34] proved that if instead of using plain probabilities as input we use stretched probabilities (inverse logistic function), that logistic mixing is optimal in the sense of minimizing Kullback-Leibler divergence, or wasted coding space, of the input predictions from the output mix. Stretching the input probabilities makes logistic regression a form of geometric weighting instead of linear weighting:

$$mix = \beta_0 + \beta_1 x_{1,i} + \beta_2 x_{2,i} + \dots + \beta_k x_{k,i} \quad (13)$$

where  $x_{j,i}$  is a probability, becomes

$$logisticMix = \beta_0 + \beta_1 t_{1,i} + \beta_2 t_{2,i} + \dots + \beta_k t_{k,i} \quad (14)$$

where  $t_{j,i} = \ln\left(\frac{x_{j,i}}{1-x_{j,i}}\right)$  and the update formula for minimizing the relative entropy could be written:

$$\beta_j = \beta_j + \alpha t_j (y_i - p_i) \quad (15)$$

Coming back to our case, we ask the question: where does the predictive power of an ensemble come from? Where does this information lay, in the aggregating algorithm or the individual components? It's obvious that if we use a form of logistic regression, the set of weights carries some information. It can be argued if that more weights are somehow added to the aggregating algorithm, it would be beneficial to its predictive power.

To some extent following this, an algorithm that adds more layers of logistic regression is the context mixing algorithm. The algorithm has been successfully applied in state-of-the-art data compression programs like PAQ [29] and *cmix* [62], which rank first in most text and general data compression benchmarks for their compression ratio.

The algorithm works in the following way:

- instead of mixing the input probabilities with only one set of weights, a number  $N$  of buckets of sets of weights is created
- choose one set of weights from each of the  $N$  buckets according to a function of context
- applying logistic regression with each set of weights will result in  $N$  probabilities, which will be mixed by another set of weights chosen from its own bucket.

One could underline that some information regarding context is passed on to the mixing ensemble, hence the name context mixing. If instead we want to move the mixing information from the ensemble towards the weak learners, we need to take into consideration the following: how to pass information back to the learners? We made little assumptions so far about the value of  $v_i$ . We know that the value comes from a big bucket of entries, where its index is dependent on the context. In a regression sense, the feature is the context, not  $v_i$ . We interpret this value like  $v_i = \beta_i + t_i$ , where  $t_i$  is a stretched probability for the context value and  $\beta$  is the weight of the probability in the ensemble. Averaging the memory values:

$$average = \frac{k}{n} \sum_{i=0}^n \beta_i t_i \quad (16)$$

which itself is a stretched probability. Applying the sigmoid function converts this average back into a probability.

## 2.4.2 Updating the proposed model

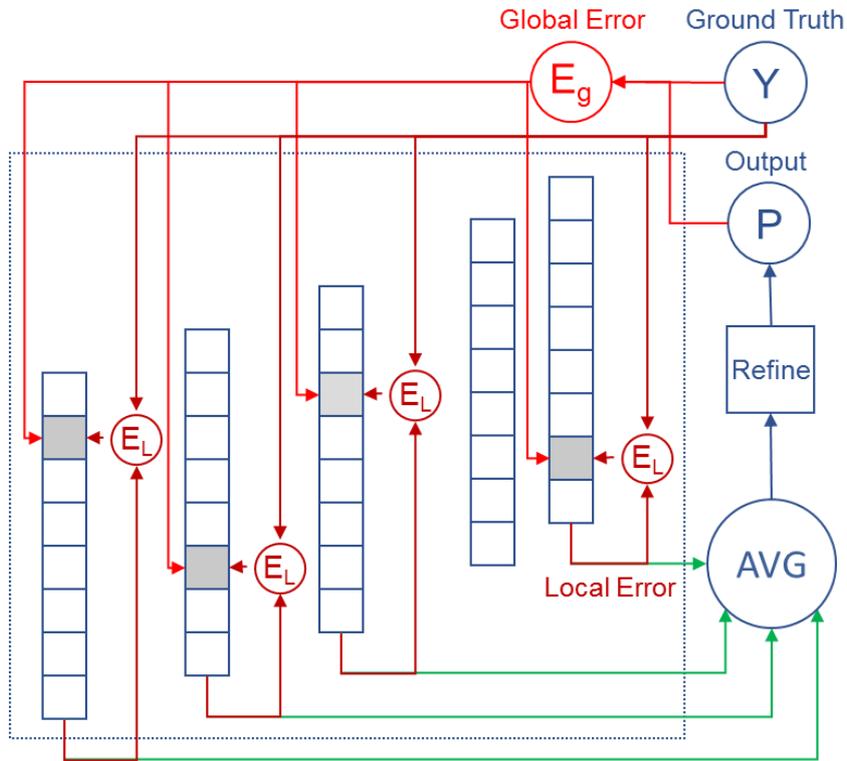
To update the model, we propose dual objective minimization:

- minimize the error in respect to the output of the entire network

- minimize the error in respect with the output of the individual node

An important observation here is that we can update the model from a supervised learning point of view or from a reinforcement learning point of view. Supervised learning means that we know the precise probability for the case we were predicting, and we use that for back propagation. Reinforcement learning means we do not know the exact probability for the case, and don't even know how to obtain it, but instead we rely on maximizing a cumulative reward in an on-line manner, given the interaction with the environment. In our case, instead of backpropagating a probability, we can use the binary outcome, and try to minimize either the cumulative logistic loss or the cumulative square loss.

In the case of edge detection, we can use both reinforcement learning, using the ground truth as a zero or one value, and as supervised learning, using the probability of the pixel to be selected as part of the ground truth in one user submissions.



**Figure 11** Block scheme for the proposed update algorithm

Working with stretched probabilities for logistic regression results in the global update error to be:

$$E_g = \beta_g(p - y) \quad (17)$$

for minimizing logistic loss, or

$$E_g = \beta_g(p - y)p(1 - p) \quad (18)$$

for minimizing the square loss, where  $E_g$  is the global error,  $\beta_g$  is the global error learning rate,  $p$  is the output probability and  $y$  is the information available as ground truth, that can be a binary outcome or a probability.

After computing the global error, we proceed to computing local errors for each memory value and updating them. The error is defined as:

$$E_l = \beta_l(p_i - y), E_g = \beta_g(p - y), \text{ with } p_i = \sigma(v_i) \quad (19)$$

for minimizing logistic loss, or

$$E_l = \beta_l(p_i - y)p_i(1 - p_i), E_g = \beta_g(p - y)p(1 - p), \text{ with } p_i = \sigma(v_i) \quad (20)$$

for minimizing the square loss, where  $E_l$  is the local error,  $E_g$  is the global error,  $\beta_l$  is the local error learning rate,  $\beta_g$  is the global error learning rate,  $p_i$  is the output probability for the  $i$ -th context (side prediction), computed as the sigmoid of the memory value  $v_i$ , and  $y$  is the ground truth.

Each memory value is then updated by subtracting both the local and the global errors:

$$v_i = v_i - E_l - E_g \quad (21)$$

It is important to notice that the global error is computed using the output probability which was refined by the adaptive probability map. This is not mandatory, but our tests show better results when the refined probability is used. This can be seen as allowing the model to learn something that can be corrected.

Updating the adaptive probability map works in the following way: the point values for *index high* and *index low* are adjusted to reduce the prediction error. Error for index low is

$$error_{low} = (points[index\ low] - y) * (1 - weight) * learning\ rate \quad (22)$$

and the error for *index high* is

$$error_{high} = (points[index\ high] - y) * weight * learning\ rate \quad (23)$$

where  $y$  is the ground truth available. Of course, variations of this can be applied, such as updating only the closest value.

Compared with the logistic regression, this method does not update the weights of the mixture, since the combining function is not a dot product, but it updates directly the values which participate in the average.

It is still a form of ensemble learning, particularly ensemble averaging, where the elements of the ensemble here are the memory values. The ensemble error is the global error.

One can argue that this method is similar to boosting, regarding the fact that each side prediction is a weak learner, and the output after mixing is the strong learner. Depending on the memory implementation chosen, which will be discussed in the next chapter, additional and possible longer contexts with unseen data make up for the bias of shorter contexts and can be added or evicted when accounting for the memory size limitations. Even though the error is backpropagated depending on the context, it also differs from context mixing, because there is no mixing layer to separate the context weights from the input probabilities.

### 2.4.3 Implementation details

As described in the previous chapter, the memory implementations can be any of the following:

- Direct lookup
- Tagged lookup

- Bucket lookup

As an optimization for all three memory types, if the APM is designed to first stretch the probability before quantizing, the two steps that squash (after averaging) and then stretch again (before quantizing) cancel each other out, and it means that the two operations, which are quite computationally costly, can be skipped and the value forwarded for quantization.

The training phase of the memory leads to data dependencies between successive runs. The dependencies come from accessing the same memory locations, and also accessing the adaptive probability map. But there is room for improvement. For instance, computing the context values for accessing the memory locations can be fully parallelizable. Depending on the memory type used, the memory locations can also be computed in parallel.

In the testing phase, no read after write dependencies will exist anymore, so the whole process can be run in parallel.

## 2.5 Results

### 2.5.1 Inputs, preprocessing and processing architecture

We implemented the contextual memory algorithm for an edge detector application. This section describes the architecture and some of the implementation details of the application. The application is implemented in the C# programming language, since we wanted to not restrict the testing and usage of the application to a closed scripting environment such as MATLAB. Using a strongly typed programming language also helps with choosing better data structures. The source code is publicly available at the GitHub page [63].

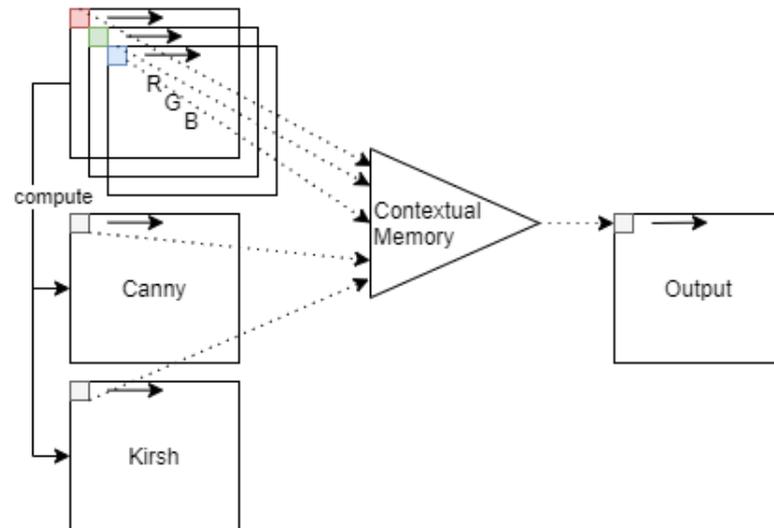
Even when talking about two dimensional images, color images have more layers in the dimension of the RGB colors. Hence three layers can become an input for the algorithm. These layers are preprocessed using a chain of preprocessors. We used a Gauss filter for eliminating the noise in the input images. This takes the original RGB layers as input and outputs a three-layer image. We used a filter of size 5 and a sigma value of 1.4.

Since the algorithm makes no assumption of the data behind contexts, it can be benefic to include transformations of the color layers. Such transformations are appended as other layers for the algorithm input image. We optionally used the Sobel filter, the Canny edge detection algorithm and a Kirsch edge detection algorithm as input, both which take the color channels and output another layer. This makes the final input image to have three or more layers. Having a Canny layer, or any edge detector as input is a sort of domain specific knowledge added in the model.

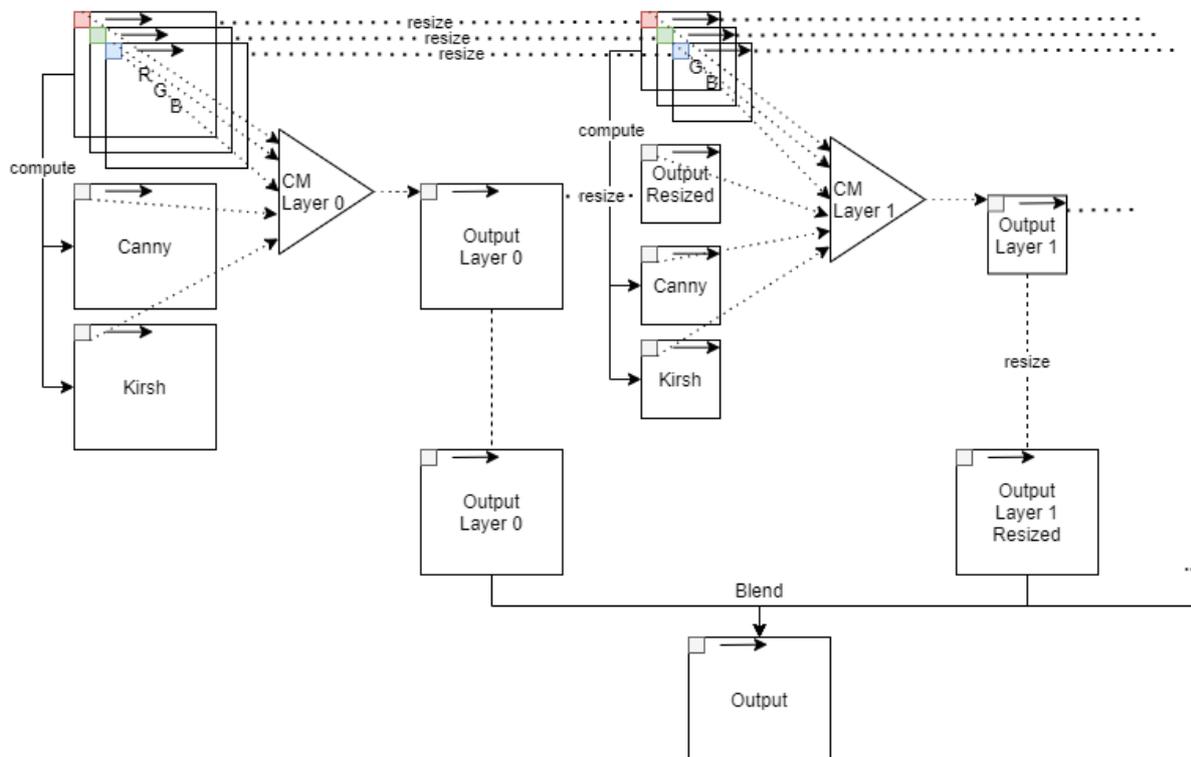
The single layer architecture takes a preprocessed image as an input and uses a given set of color channels to compute an output image which consists of a single layer grayscale image in which the pixels represent the probability that the position in the original image belongs to an edge. The single layer architecture is described in Figure 12.

The single layer architecture combined with the simple rays as contexts does not take into account information about the edge being kept the same at different zoom levels. To tackle the zooming problem, the multilayer architecture behaves in this way: take the original image, apply preprocessing, obtain the output; then take the original image, apply the preprocessing, append the previously obtained output as a layer, resize the image (meaning all its layers) and use it as an input for the algorithm. If one decides to separate the memory used by the algorithm at different sizes, we have a multilayer architecture. Each layer is trained separately starting from the largest image and going towards resized images. An algorithm layer can have different configuration from the other layers, and options can include the length of the longest ray, the preprocessing done, the memory size and others. The result of the multilayer architecture is a

set of grayscale images of varying sizes, which are called side outputs. These side outputs are then combined (blended) using a logistic regression layer, to form a single image. Before blending, the images are scaled to have the same size. The weights of the logistic regression layer are also trained on the training set. The multilayer architecture is described in Figure 13.



**Figure 12** Contextual memory single layer processing architecture



**Figure 13** Contextual memory multilayer processing architecture

## 2.5.2 Results on Berkeley Edge Detection Benchmark

In this section we present one example of the output images from the algorithm using image “326025” from the mentioned benchmark dataset along with some of the parameters used and a short description of the differences. We also provide an analytical evaluation of the improvements compared to the Canny Edge detection method.

The above-mentioned image along with its ground truth is shown in Figure 14. An example output, after non maximal suppression and thinning is applied, is presented in Figure 15.



**Figure 14** (a) sample image from the benchmark along with (b) the ground truth

The global parameter values used in the tests are:

- $\beta_l = 0.25$
- $\beta_g = 0.5$
- $k = 2$
- quantized derivative rays have the 2 least significant bits quantized

The results for the single layer architecture for various model parameters are presented in

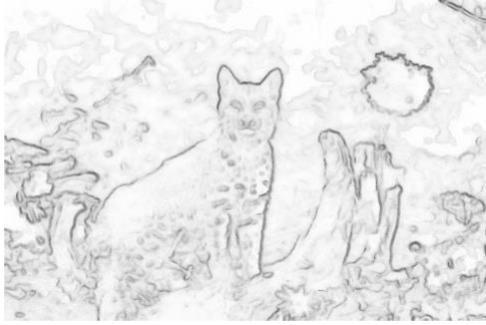
**Table 6.** The results for the multilayered architecture, also varying parameters, are presented in **Table 7.**

**Table 6** Results for the single layer contextual memory edge detector

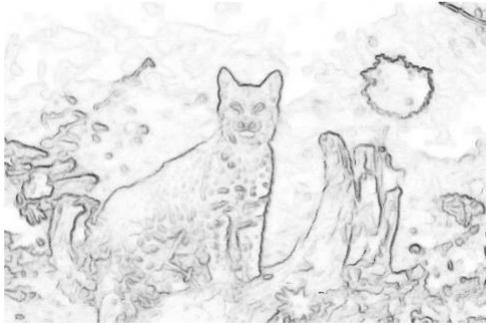
Algorithm Output	Description
	Context model: 16 rays, max length: 8, include quantized derivative rays Memory: direct lookup, table size $2^{18}$ Loss function: square loss Number of passes over training set: 1 Preprocess: Gauss
	Context model: 16 rays, max length: 8, include quantized derivative rays Memory: direct lookup, table size $2^{20}$ Loss function: square loss Number of passes over training set: 1 Preprocess: Gauss



Context model: 16 rays, max length:  
8, include quantized derivative rays  
Memory: direct lookup, table size  $2^{22}$   
Loss function: square loss  
Number of passes over training set: 1  
Preprocess: Gauss



Context model: 16 rays, max length:  
8, include quantized derivative rays  
Memory: direct lookup, table size  $2^{22}$   
Loss function: square loss  
Number of passes over training set: 2  
Preprocess: Gauss



Context model: 16 rays, max length:  
8, include quantized derivative rays  
Memory: direct lookup, table size  $2^{22}$   
Loss function: square loss  
Number of passes over training set: 3  
Preprocess: Gauss



Context model: 16 rays, max length:  
8, include quantized derivative rays  
Memory: tagged lookup, table size  $2^{20}$   
Loss function: square loss  
Number of passes over training set: 1  
Preprocess: Gauss



Context model: 16 rays, max length:  
8, include quantized derivative rays  
Memory: bucket lookup, table size  $2^{20}$   
Loss function: square loss  
Number of passes over training set: 1  
Preprocess: Gauss



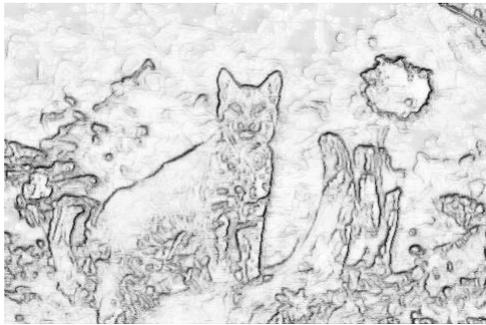
Context model: 8 rays, max length: 8,  
include quantized derivative rays  
Memory: direct lookup, table size  $2^{20}$   
Loss function: square loss  
Number of passes over training set: 1  
Preprocess: Gauss



Context model: 32 rays, max length:  
8, include quantized derivative rays  
Memory: direct lookup, table size  $2^{20}$   
Loss function: square loss  
Number of passes over training set: 1  
Preprocess: Gauss



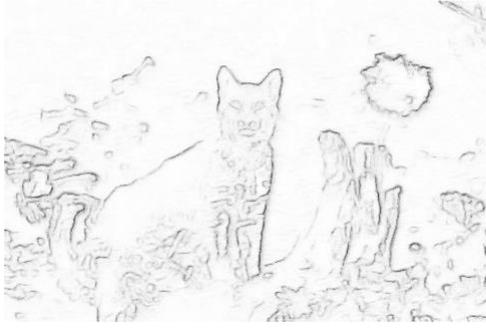
Context model: 16 rays, max length:  
8, include quantized derivative rays  
Memory: direct lookup, table size  $2^{20}$   
Loss function: square loss  
Number of passes over training set: 1  
Preprocess: Gauss, Canny



Context model: 16 rays, max length:  
8, include quantized derivative rays  
Memory: bucket lookup, table size  $2^{20}$   
Loss function: square loss  
Number of passes over training set: 1  
Preprocess: Gauss, Canny



Context model: 16 rays, max length:  
8, include quantized derivative rays  
Memory: bucket lookup, table size  $2^{20}$   
Loss function: square loss  
Number of passes over training set: 1  
Preprocess: Gauss, Canny, Kirsch



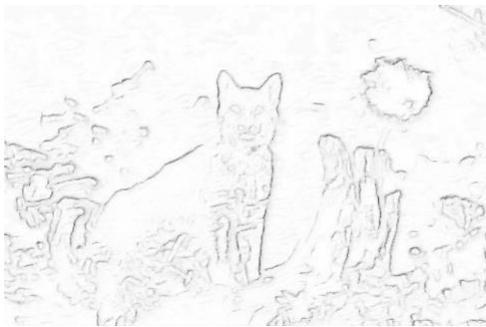
Context model: 16 rays, max length: 8, include quantized derivative rays  
Memory: bucket lookup, table size  $2^{20}$   
Loss function: square loss, GT  
threshold: 63  
Number of passes over training set: 1  
Preprocess: Gauss, Canny, Kirsch



Context model: 16 rays, max length: 8, include quantized derivative rays  
Memory: bucket lookup, table size  $2^{20}$   
Loss function: entropy loss, GT  
threshold: 63  
Number of passes over training set: 1  
Preprocess: Gauss, Canny, Kirsch



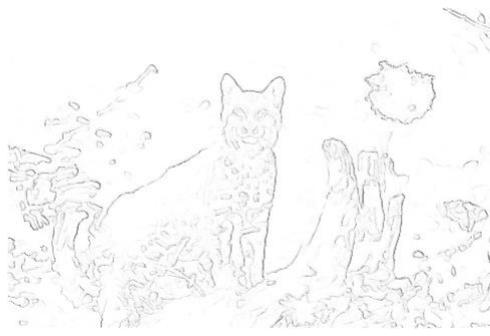
Context model: 16 rays, max length: 8, include quantized derivative rays  
Memory: direct lookup, table size  $2^{20}$   
Loss function: square loss, GT  
threshold: 63  
Number of passes over training set: 1  
Preprocess: Gauss, Canny, Kirsch



Context model: 16 rays, max length: 8, include quantized derivative rays  
Memory: direct lookup, table size  $2^{20}$   
Loss function: square loss, GT  
threshold: 63  
Number of passes over training set: 3  
Preprocess: Gauss, Canny, Kirsch

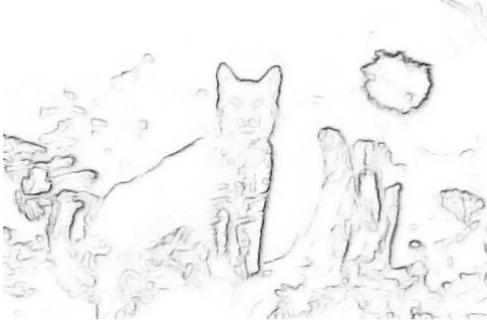
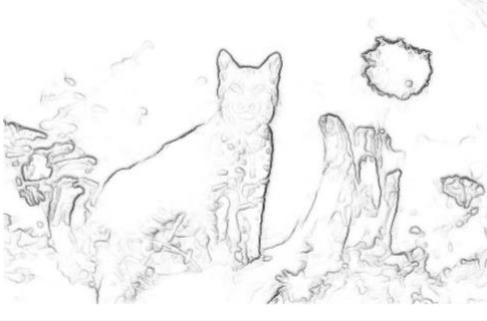
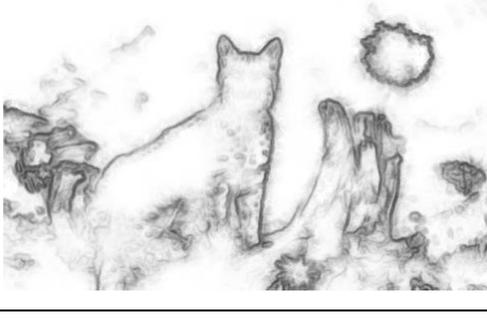


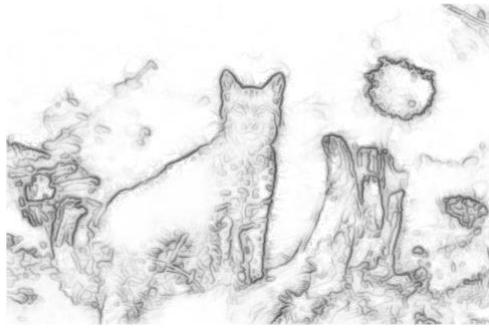
Context model: 16 rays, max length: 8, include quantized derivative rays  
Memory: direct lookup, table size  $2^{20}$   
Loss function: entropy loss, GT  
threshold: 63  
Number of passes over training set: 1  
Preprocess: Gauss, Canny, Kirsch



Context model: 16 rays, max length: 8, include quantized derivative rays  
 Memory: direct lookup, table size  $2^{20}$   
 Loss function: entropy loss, GT  
 threshold: 63  
 Number of passes over training set: 3  
 Preprocess: Gauss, Canny, Kirsch

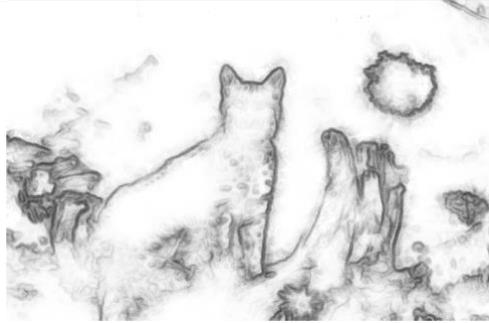
**Table 7** Results for the multilayer architecture of the contextual memory edge detector

Algorithm Output	Description
	Context model: 16 rays, max length: 8, include quantized derivative rays Memory: direct lookup, table size $2^{18}$ Loss function: square loss, GT threshold: 63 Number of passes over training set: 1 Preprocess: Gauss, Canny, Kirsch Layers scale: 1, 2, 4, 8, 16
	Context model: 16 rays, max length: 8, include quantized derivative rays Memory: bucket lookup, table size $2^{18}$ Loss function: square loss, GT threshold: 63 Number of passes over training set: 1 Preprocess: Gauss, Canny, Kirsch Layers scale: 1, 2, 4, 8, 16
	Context model: 16 rays, max length: 8, include quantized derivative rays Memory: bucket lookup, table size $2^{18}$ Loss function: entropy loss, GT threshold: 63 Number of passes over training set: 1 Preprocess: Gauss, Canny, Kirsch Layers scale: 1, 2, 4, 8, 16
	Context model: 16 rays, max length: 8, include quantized derivative rays Memory: bucket lookup, table size $2^{19}$ Loss function: entropy loss, GT threshold: 63 Number of passes over training set: 1 Preprocess: Gauss Layers scale: 1, 3, 5, 7



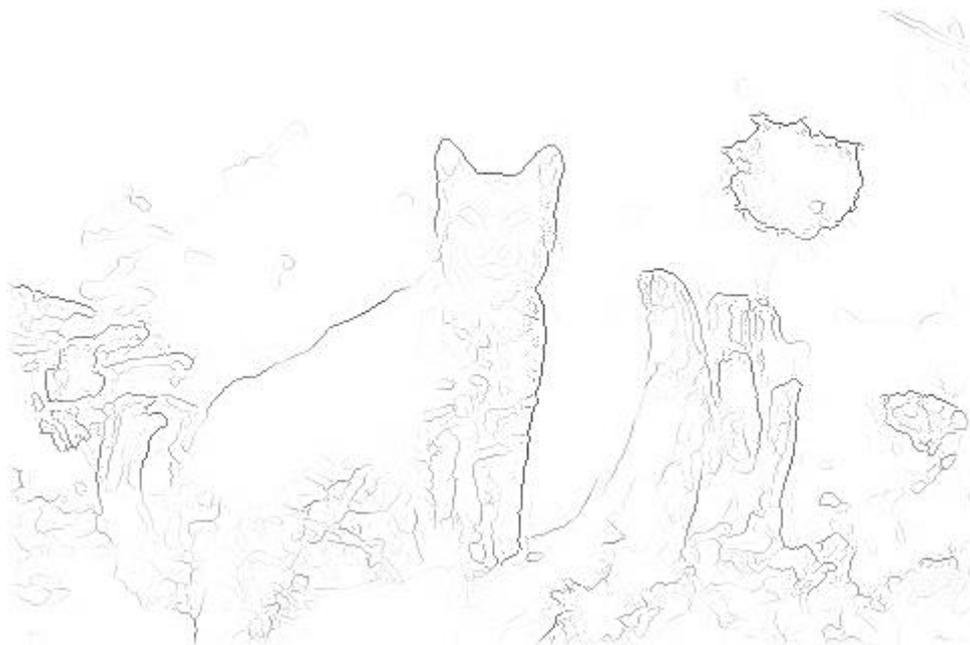
Context model: 16 rays, max length: 8, include quantized derivative rays  
Memory: bucket lookup, table size  $2^{19}$   
Loss function: entropy loss, GT  
threshold: 63  
Number of passes over training set: 1  
Preprocess: Gauss, Canny  
Layers scale: 1, 3, 5, 7

---



Context model: 16 rays, max length: 8  
Memory: bucket lookup, table size  $2^{19}$   
Loss function: entropy loss, GT  
threshold: 63  
Number of passes over training set: 1  
Preprocess: Gauss, Canny, Kirsch  
Layers scale: 1, 3, 5, 7

---

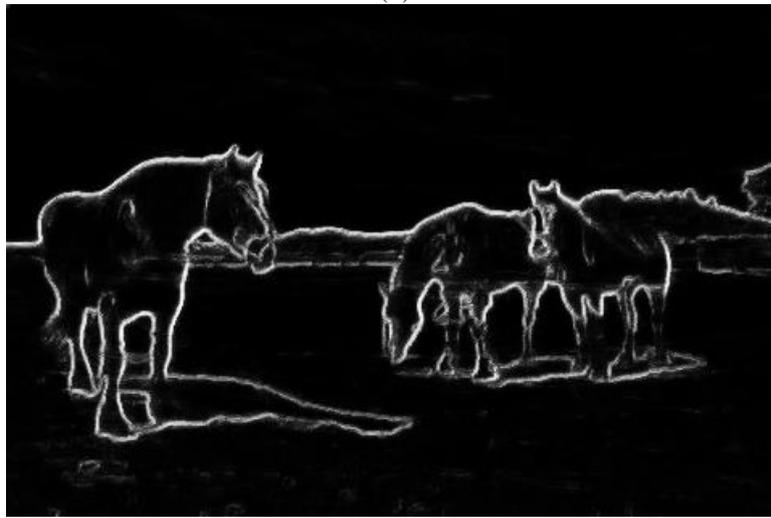


**Figure 15** Output example after NMS and thinning

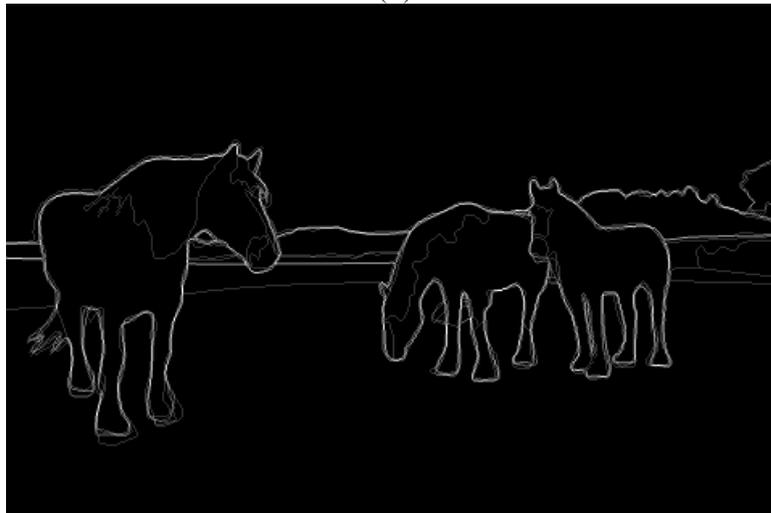
In Figure 16 we depict the output of our proposed method when we input one of the training images. We can see that the detected edged closely follow the ground truth.



(a)



(b)



(c)

**Figure 16** Output of the proposed method on the training image “197017”, (a) original image, (b) proposed output, (c) ground truth.

In Figure 17 we depict the output of our proposed method when we input one of the test images, and in Figure 18 we present the output of each of the four layers, each layer being half

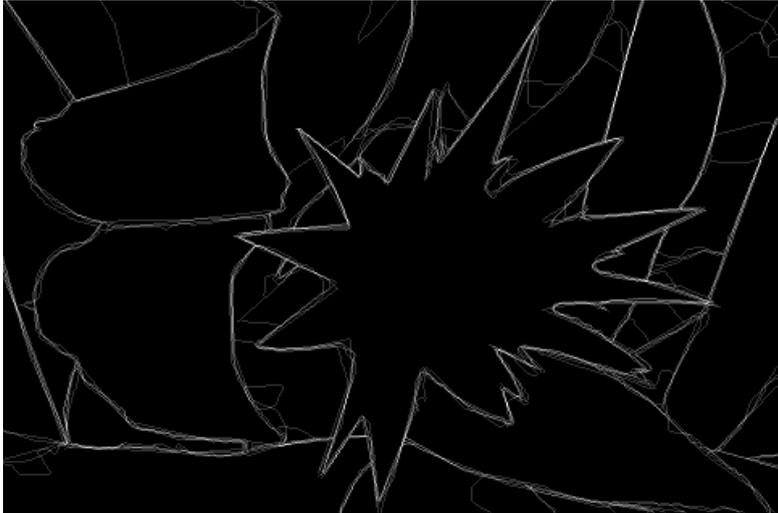
the size of the previous one. We can see which details each layer captured and how the blending mixed them.



(a)

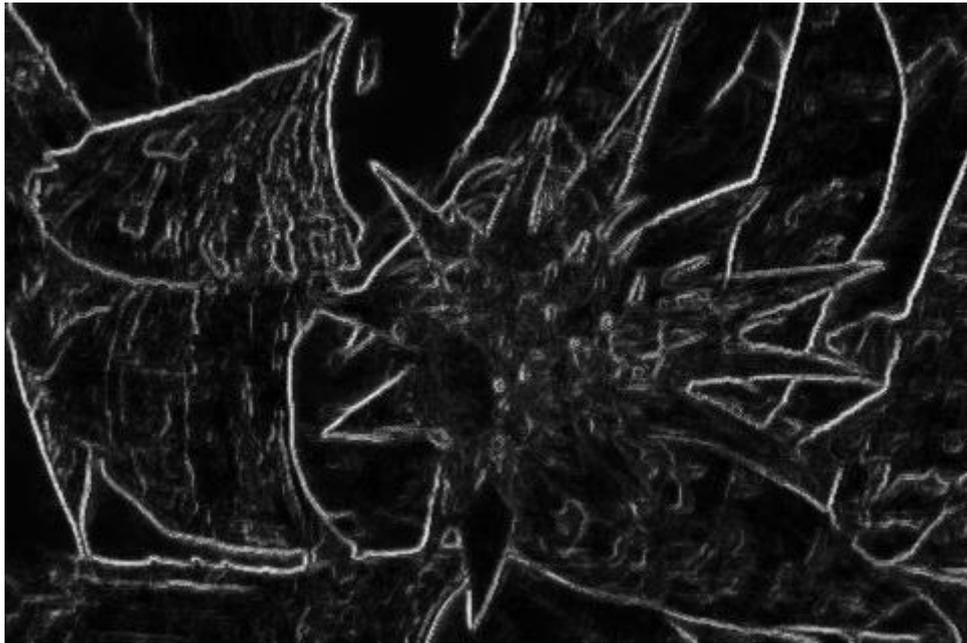


(b)

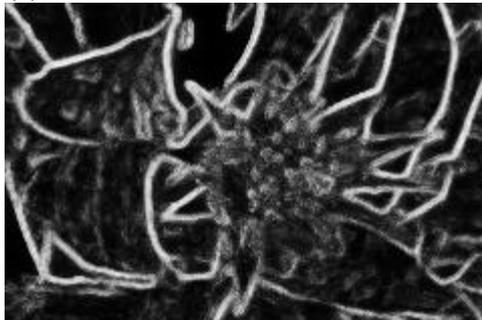


(c)

**Figure 17** Output of the proposed method on the test image “51084”, (a) original image, (b) proposed output, (c) ground truth.



(a)



(b)



(c)



(d)

**Figure 18** Output of the individual layers on the test image “51084”, (a) first layer (b) second layer (c) third layer (d) fourth layer.

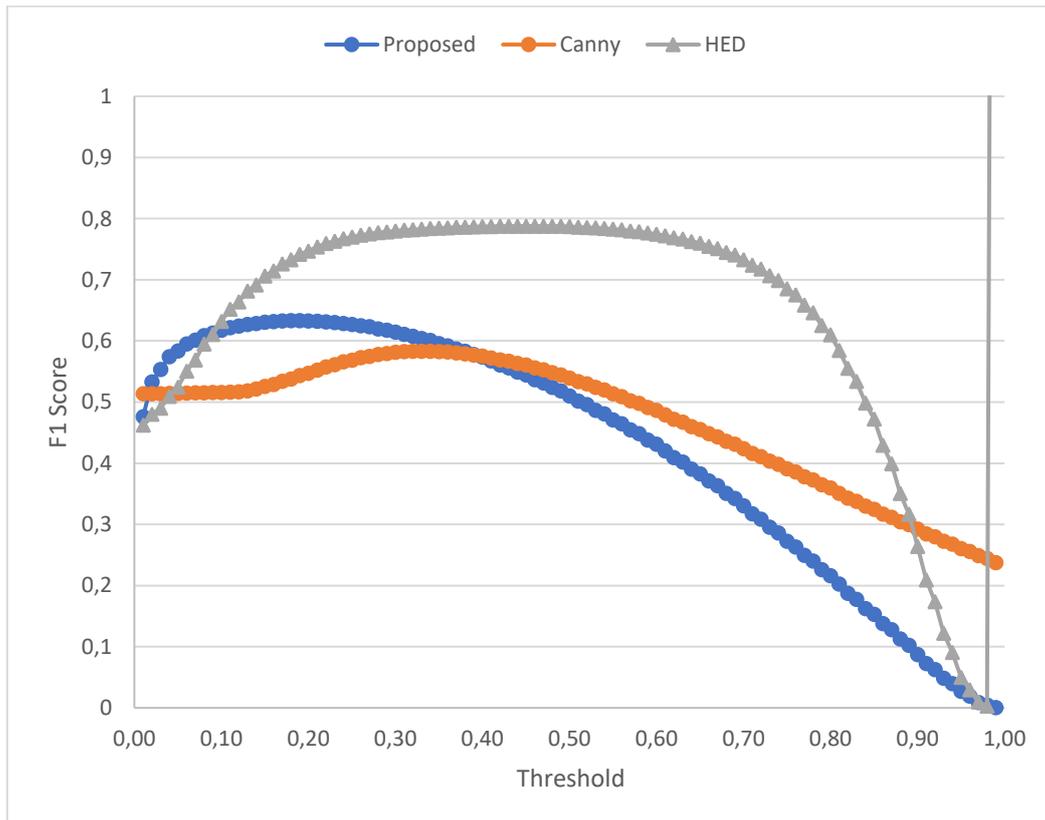
Before computing the score for the benchmark, the output images are subjected to a non-maximal suppression technique and subsequently to an edge thinning. This is done in MATLAB, using an adapted version of the Piotr Dollar’s Structured Edge Detection Toolbox, available at [64].

The benchmark provides a tool for evaluation which has an automated search in the space of thresholding, so that the user feels free to leave grayscale images instead of making the binarization himself. We compare the algorithm with the well-known Canny Edge Detector, the recent ID&L algorithm [55] and to the state-of-the-art deep learning algorithms HED [51],

RDS [52], RCF-ResNet101-MS [53], CED-VGG16 [65], AMH-ResNet50 [66], CASENet [67] and CEDN [68]. The comparison results are presented in Table 8.

**Table 8** Comparative results of the contextual memory edge detector

Algorithm	F1 Score	Precision	Recall	Threshold
Canny	0.583	0.500	0.698	0.33
ID&L [55]	0.610	0.590	0.680	N/A
<b>Contextual Memory (proposed)</b>	<b>0.640</b>	<b>0.605</b>	<b>0.686</b>	<b>0.19</b>
CASENet [67]	0.767	N/A	N/A	N/A
HED [51]	0.787	0.803	0.772	0.46
RDS [52]	0.792	N/A	N/A	N/A
CED-VGG16 [65]	0.794	N/A	N/A	N/A
AMH-ResNet50 [66]	0.798	N/A	N/A	N/A
CEDN [68]	0.788	N/A	N/A	N/A
RCF-ResNet101-MS [53]	0.819	N/A	N/A	N/A

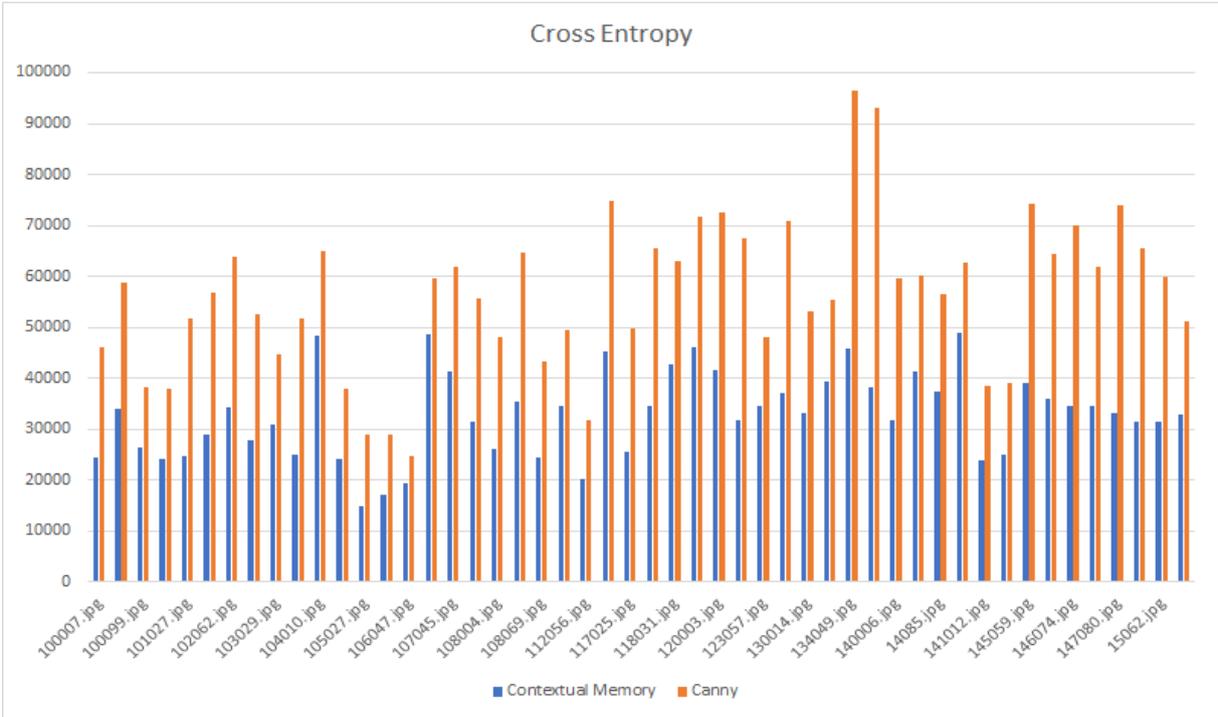


**Figure 19** F1 Score plotted against threshold

We can clearly see from Figure 19 that the learning is shifted towards not taking any risks, since the better F1 Score is achieved in the low threshold settings. In order to obtain an overall better F1 Score, class balancing must be considered when applying the loss function.

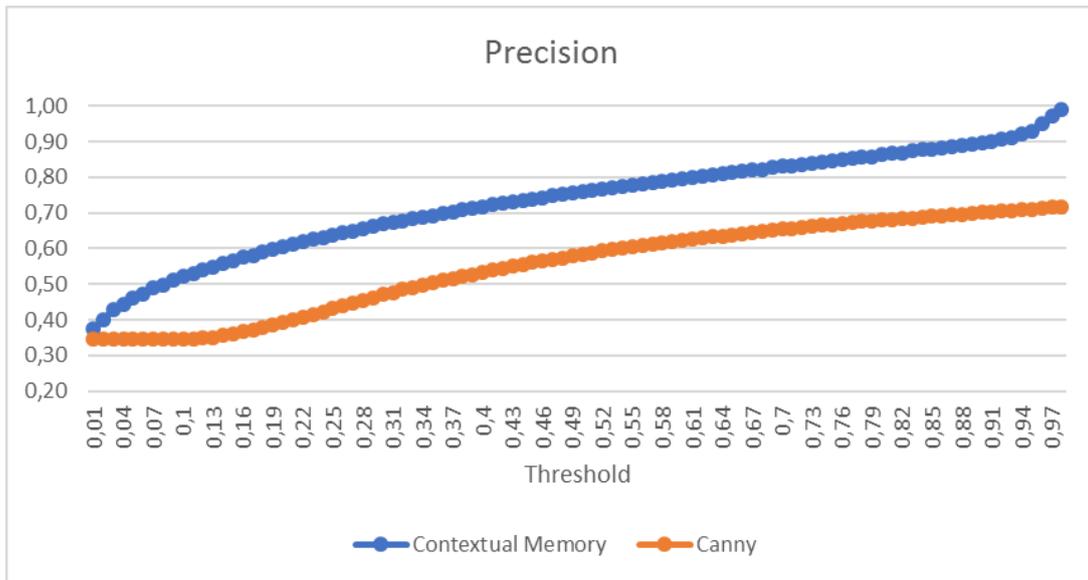
We made another analytical comparison using the Cross-Entropy measure. If we have two probability distributions, we can measure the number of bits needed to identify an event drawn from a set if a coding scheme is used using a probability distribution other than the true

distribution of the set. Since the pixel intensities in the resulting images can be modeled as a probability of a pixel belonging to an edge, we can measure the cross entropy for the output images. We show a comparison with the Canny algorithm for the first 50 images in the test set of the benchmark in Figure 20. A better probability modeling of the true source of the edges should have a lower cross entropy. For the overall set, the proposed method obtained a cross entropy of 6610784. In comparison, the Canny algorithm obtained a score of 11372700. This means that our algorithm surpassed the Canny algorithm by a factor of 1.72.



**Figure 20** Cross Entropy for the first 50 images of the dataset (lower is better)

To prove the potential of the proposed method, we also considered the precision in respect to the threshold. Precision is the fraction of relevant instances among the retrieved instances. A high value represents a small rate of false positives. Compared with the Canny algorithm, the proposed method offers a steady increase of precision and a roughly constant better precision in respect to the threshold. This can be seen in Figure 21.



**Figure 21** Precision in respect to the threshold (bigger is better)

Provided the results on the three measures, we show that the proposed algorithm has potential over the Canny method.

## 2.6 Conclusions

The main contribution of this chapter is an off-line application agnostic algorithm for the prediction of probability that a pixel belongs to a class or another, based on the contextual information available, where learning can be done in a single pass over the training data. More than this, it does not impose any constraints on how to choose or model the context. It also allows it to be part of larger learning and prediction structures, having loose requirements on what information is needed for feedback.

In order to demonstrate the usefulness of the method, a 2D edge detection application using the method was implemented, with two processing architectures, a single and a multilayer one. The results are promising, considering that no prior handcrafted knowledge has been added in the model to help with the prediction.

Allocating too much memory and more training passes over the training set leads to overfitting. The algorithm will learn by heart the training set, and will reproduce meticulously the ground truth, but the quality of the results on the test set will downgrade. We do not propose any solution to the overfitting problem.

In the future, we intend to experiment further with contextual modeling, which means exploring the space of choosing the appropriate contexts for various applications. We would also like to extend the existing application for three-dimensional images and apply it for medical image segmentation. Some of the changes needed to make when switching from 2D to 3D will be to model 3D contexts, which can be chosen as rays in three dimensions instead of two. Also, centerline and segmentation benchmarks have different intended objectives, so the feedback mechanism and the output metrics will have to be adapted to obtain results relevant to those benchmarks.

In order to improve the existing application, the following can be implemented:

- replace the loss function with a cost sensitive loss function, as described in [58], because the distribution of edge/non-edge pixels in the benchmark is biased 90% in favor of non-edges

- model rays as individual blocks, whose output will be combined using a context mixing layer. This means each ray will output a probability, and the set of probabilities will be further combined into a single probability
- replace the blending algorithm with an improved context mixing layer, maybe even with a fully connected layer such as in CNNs
- add more convolutional layers as input, having a set of learnable filters
- add the output of the state-of-the-art edge detectors as input layers, to see how the algorithm balances the precision and recall
- implement a GPGPU version of both the single or multilayer algorithm
- adapt the training phase to iterate over transformed versions of the input images, such as rotations and scaling, to gain more training data
- exclude unconvincing ground truth data, when less than a half of the human subjects agree to the edge position
- last but not least, design space exploration in respect to the parameters

Integrating with the Relaxed Deep Supervision [52] algorithm should provide interesting results, since the algorithm provides high precision on lower thresholds. We also expect the proposed network to integrate well with a deep learning architecture, especially with a CNN network, as a layer after the Rectified Linear Units (ReLU) layer, side by side with the fully connected layer.





### 3. Coronary centerline extraction from CCTA using 3D-UNet

#### 3.1 Overview

Non-invasive methods of medical imaging have gained much popularity with the increasing resolution of image acquisition and new possibilities to acquire volumetric images. These technological changes have led to an increase in the accuracy of patient diagnosis, without the need for invasive checks. Currently, different image segmentation methods are available, but most of them only consider two-dimensional images. Segmentation involves identifying, individualizing, or isolating one or more regions in the image, based on a uniform characteristic of the region in question. Thus, three-dimensional segmentation, analogous to two-dimensional segmentation, involves isolating a volume. Segmentation can be used in this case, for the automatic or semi-automatic identification of the coronary anatomy in the sequence of images produced by the Computed Tomography (CT). It can also be used to automatically identify lesions and extract their features, such as geometry or volume.

The mesh-type coronary model can be employed for operations such as determining the degree of blockage of the vessel, determining positive remodeling [69] (if the vessel has changed shape due to differences in dynamic pressure caused by injury to the vessel) and measuring distances along the vessel. Recent reconstruction models have come to the performance of simulating fluid dynamics [70], [71]. The problems of three-dimensional reconstruction are the identification of the artifacts introduced by the tomograph and the introduction of a priori knowledge, specific to the scanned area.

When it comes to volumetric medical image interpretation, algorithms could help reducing the variability between observers having different experience levels in searching for features in volumetric images [72]. Information such as automatic or semi-automatic detection of lesions, plaques, positive remodeling, and calculation of fractional flow reserve (FFR) are targeted for extraction. Once obtained, they can be presented to the specialist in the form of statistics, annotations and highlights on the model. Vessel segmentation is a prerequisite in the automated pipeline of diagnosis for vascular related diseases [73]. Accurate detection of coronary artery diseases can be done by combining the results of machine learning and image based morphological feature extraction [74].

Assigning labels to voxels in biomedical applications has been a cornerstone in medical image processing, with most of the fully automated segmentation techniques being atlas-based, where a set of expert pre-segmented cases stand as a pillar to high accuracy results [75].

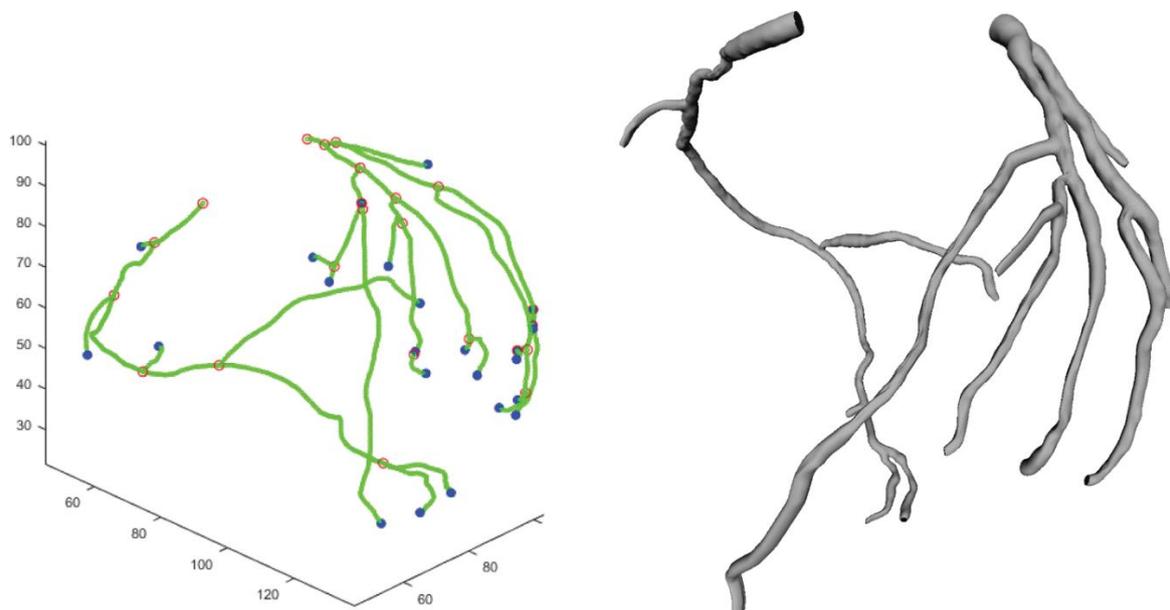
The proposed method presented in this chapter is a machine learning centerline extraction algorithm which uses a proposed 3D adaptation of the U-NET architecture which outputs the probability for each voxel of a full volume to be part of a vessel centerline. Furthermore, based on an extensive state of the art review, an adapted loss function was proposed to handle sparse annotation, meaning that not all the centerlines are part of the ground truth of the dataset, and class imbalance, as very few voxels from an entire volume belong to a centerline. The research was published in [5] and [1].

The chapter is organized as follows: section 3.2 provides a brief description of the current state of the art methods in coronary centerline extraction, followed by section 3.3 presenting our

approach using a 3D-UNET neural network; based on data publicly available. The experimental setup is described in section 3.4 and the obtained results are illustrated in section 3.5 and discussed in section 3.6 together with the final conclusions presented in the last section.

### 3.2 Related Work

Vessel centerlines can be extracted by either a segmentation and thinning pipeline [76], or by direct tracking [77]. The extracted centerlines can serve as input for tracking algorithms for segmenting the required vessel tree. In [78], the authors proposed Bayesian tracking combined with sphere fitting for segmentation. Vessel branching is done using statistical models with three categories: locally disconnected vessel (disappearance in some of the slices in the CTA), locally disconnected branch, and branch occurrence (domain specific knowledge about coronary vessels). An example sphere fitting to obtain vessel segmentation starting from centerlines can be seen in Figure 22.

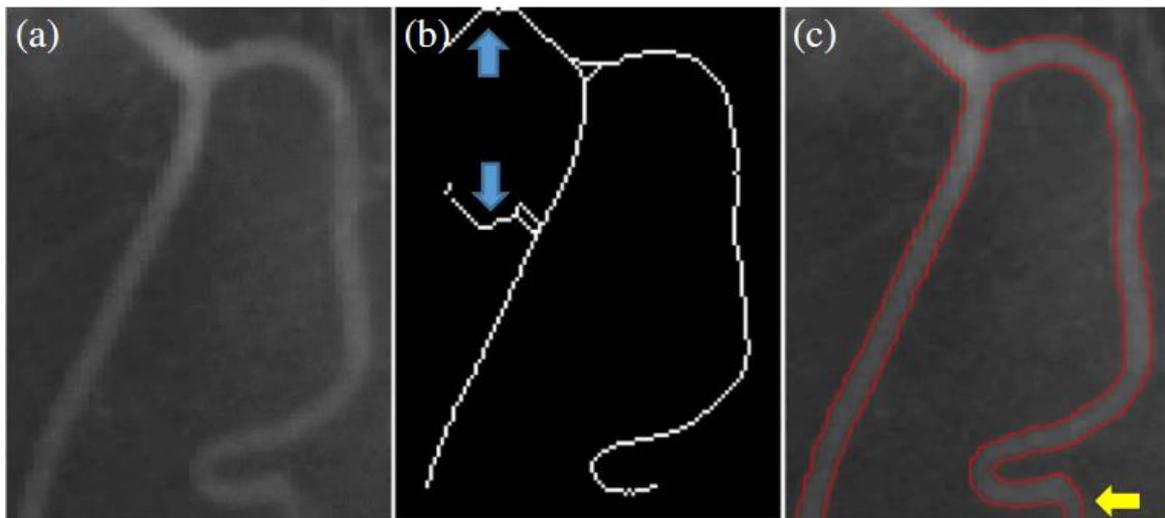


**Figure 22** Result of centerlines (left) to vessel segmentation (right) using sphere fitting, from [78].

In [79], the lumen segmentation is performed by tracking, using a minimal cost path based on a cost function with prior model information on three measurable parameters: vesselness measure, intensity similarity, and directional information.

Tree-structured segmentation is also achieved in [80] using the centerline extracted using a deep learning approach. A tree-structured Convolutional Gated Recurrent Unit network is trained for tracking segmentation and achieves good results, especially at vessel bifurcations.

Another method for vessel segmentation using a centerline constrained level set method is proposed in [81]. It improves on two active contour models, Chan-Vase and Curve Evolution for Vessel Segmentation, by integrating the centerline into the set evolution, as shown in Figure 23.



**Figure 23** Example vessel segmentation from extracted centerline using set evolution active contour, from [81]

There are two major categories of vessel and centerline segmentation methods: rule based and machine-learning based.

### 3.2.1. Rule based centerline extraction

Dynamic programming of coronary borders was used in [82] for centerline extraction, with a prior localization of the aorta using the Hough transform, and followed by a registration step of the borders in many cross-sectional images.

A fully automated centerline extraction is proposed in [83]. The pipeline proposed includes preprocessing by multiscale vessel enhancement filtering, and then using a tracking algorithm based on an automatic acquisition of an initial point and direction, combined with forward and backwards ridge positioning.

Another centerline extractor starting from segmented vessels is described in [84], where the authors compute a 3D gradient vector flow field which is then used to compute a timetable of arrive speed from one point to another. Combined with a branch tracing algorithm and a measure function for vesselness, the centerlines were extracted in about 16 minutes of processing time.

### 3.2.2. Machine learning based centerline extraction

A single voxel wide centerline extraction was implemented in [85]. It employs a fully convolutional neural network (FCN) which generates distance maps and detects branch endpoints. A distance map is a volume with each voxel representing the probability that a voxel belongs to the centerline, proportional to the distance from the true centerline. Using this information, a minimal path extractor given a root point can extract the centerline tree.

A state of the art convolutional neural network (CNN) for automatic centerline extraction using an orientation classifier was implemented in [86]. Here, the authors feed the network small patches with a size of  $19 \times 19 \times 19$  (which is a rather small context). The architecture of the network compensates for the small input size by using dilation convolution kernels. The output of the network is a quantized direction of the centerline of the vessel in the patch, and also the radius of the vessel. Generating a full centerline relies on an initial point placed anywhere inside

the vessel. Using this, a tracking algorithm iteratively feeds patches in the network and constructs the centerline. The tracking stops when the confidence of the output direction is below a given threshold. Due to the nature of the tracking algorithm, it would seem that a CNN with recurrent layers [87] was a missed opportunity. The recurrent layers are helpful when the successive inputs fed to a neural network are correlated [9]. This work was extended in [88] by improving the bifurcation detection. The bifurcation angle is an important feature in CAD classification [74].

Machine learning was combined in a hybrid approach with global geometry learning in [89] in order to improve the connectivity of the result of a CNN. Geometry aware grouping was also added to further improve the continuity of the vessel tree.

### 3.3. Proposed Method

#### 3.3.1. Neural Network Architecture

The idea for creating a 3D U-Net network came after observing the results of U-Net Convolutional Networks for Biomedical Image Segmentation [6], which allowed a fully convolutional neural network to provide a good segmentation even when trained with a small training dataset. The network architecture is depicted in Figure 24. Additionally, the belief that the network architecture was the right choice was further consolidated when observing the results of the U-Net applied for liver segmentation and vessel exclusion [90], and for Retina Vessel Segmentation, a similar task to centerline segmentation. An example of the full retina vessel segmentation can be found in Figure 25. The work on Retina Vessel Segmentation was refined many times, in several papers [91], [92], [93].

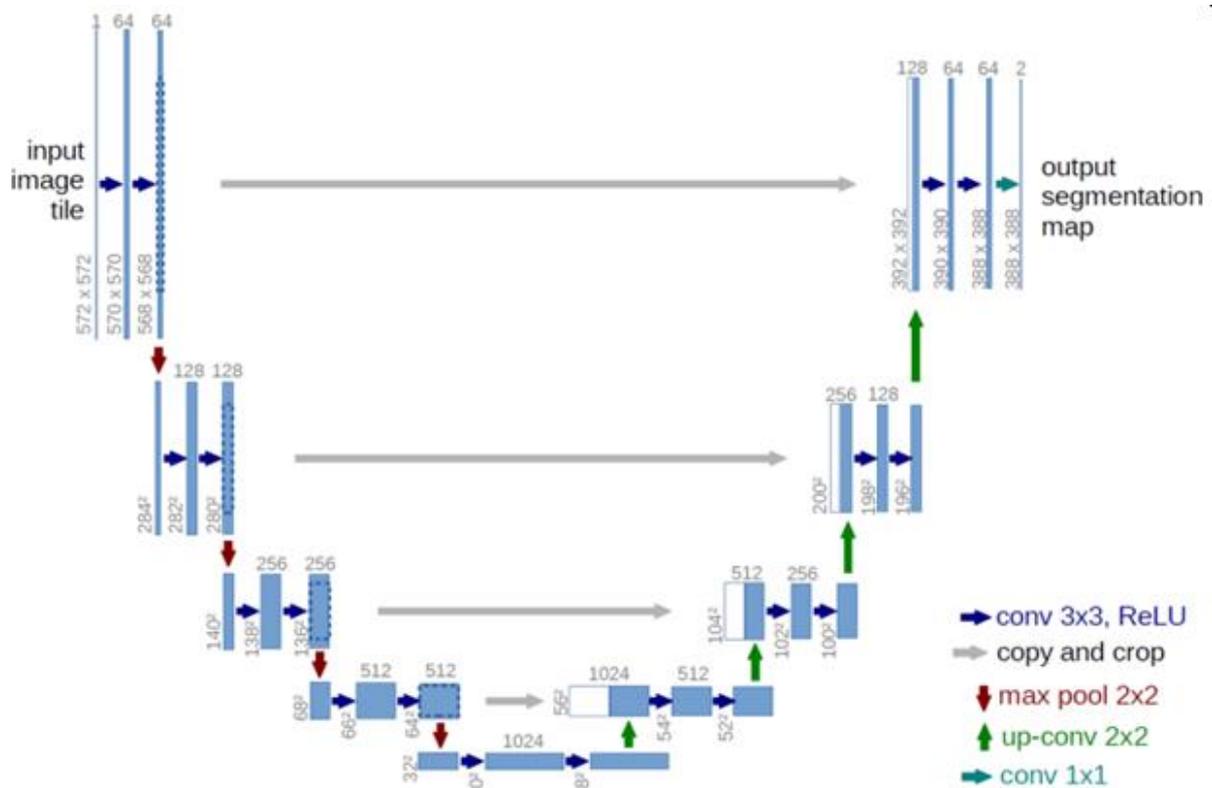
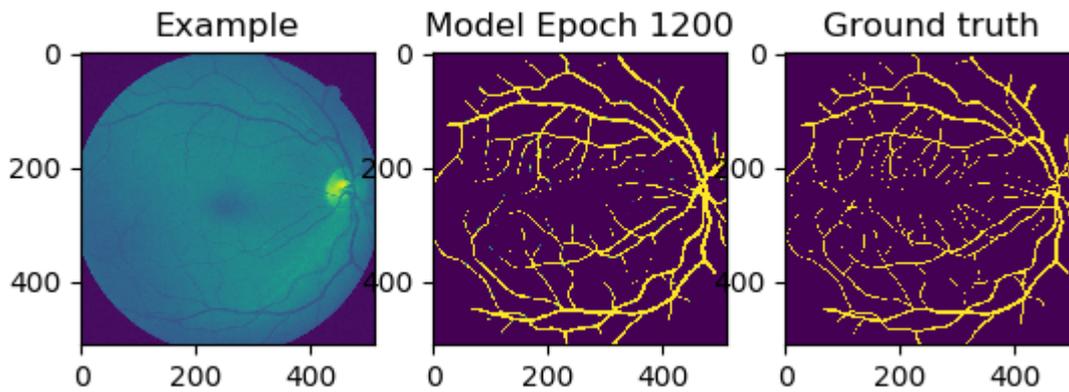
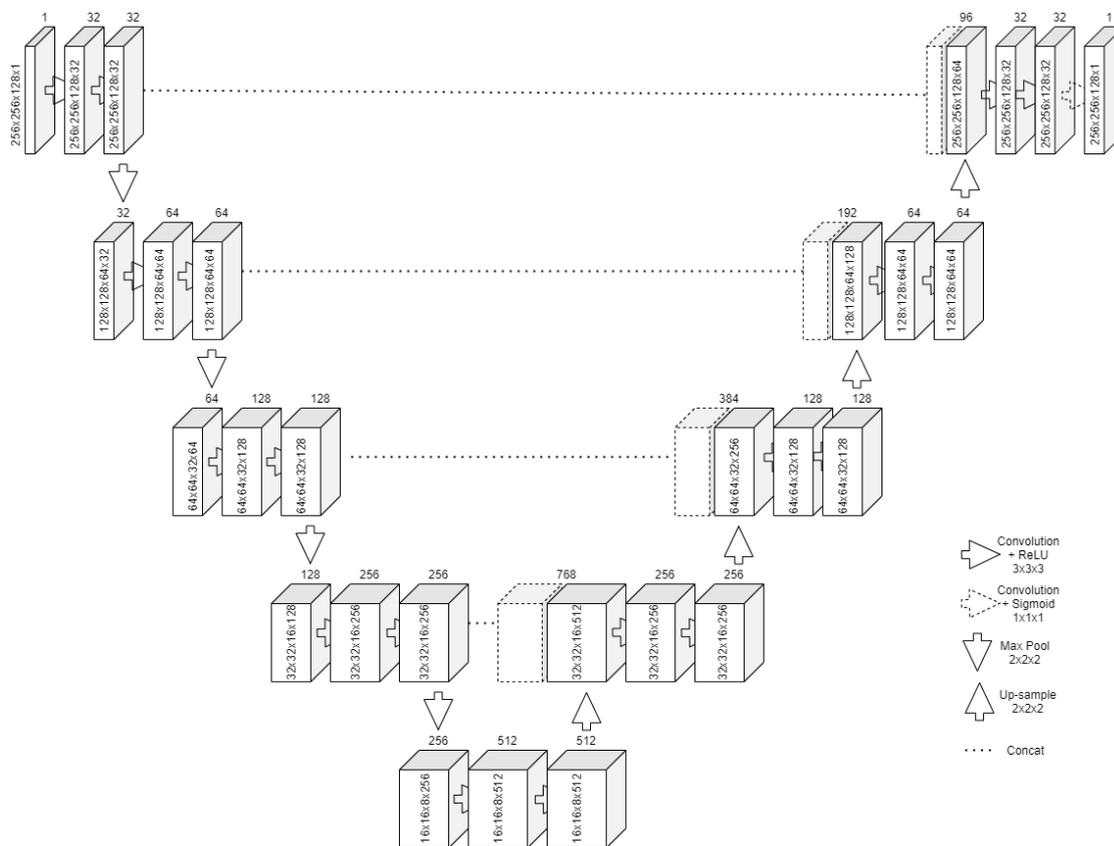


Figure 24 The original U-NET architecture, from [6]



**Figure 25** Example Retina Vessel Segmentation using a 2D U-Net Architecture



**Figure 26** The proposed 3D U-Net architecture

A similar extension of the network architecture to 3D was also presented in [94], for Xenopus kidney segmentation, where the ground truth segmentation was completed after training using sparse annotation. This is, in a way, similar to the annotation for the dataset used in this chapter, where not every vessel is annotated and the network needs to learn to extrapolate the provided cases. Another variation of the 3D U-NET was introduced in [95] for multiple organ segmentation tasks.

The design of the U-Net follows two important steps, similar to an autoencoder network. The first one is the contraction, where successive rounds of two convolutions and a max-pooling to reduce the output size by half are applied on the input, as presented in Figure 26. For each round, the number of filters for the convolution is doubled. The bottom layer will have the most feature maps, but will also be the smallest in size. Its purpose is to learn an encoded representation of what it needs to be segmented. The convolution with 3x3x3 kernels means that one pixel from all the borders will be lost. To alleviate this problem, padding was employed.

The second one is the expansion. Starting from the bottom layer, successive rounds of up-sampling, concatenate, and two convolutions (also with padding) are applied. The up-sampling resizes the feature vector. Along with the information concatenated from the same-size input of the contraction, the two convolutions can reconstruct the image in its original size. Transposed convolution can be used as an operation instead of up-sampling. However, this leads to an increased number of parameters to train.

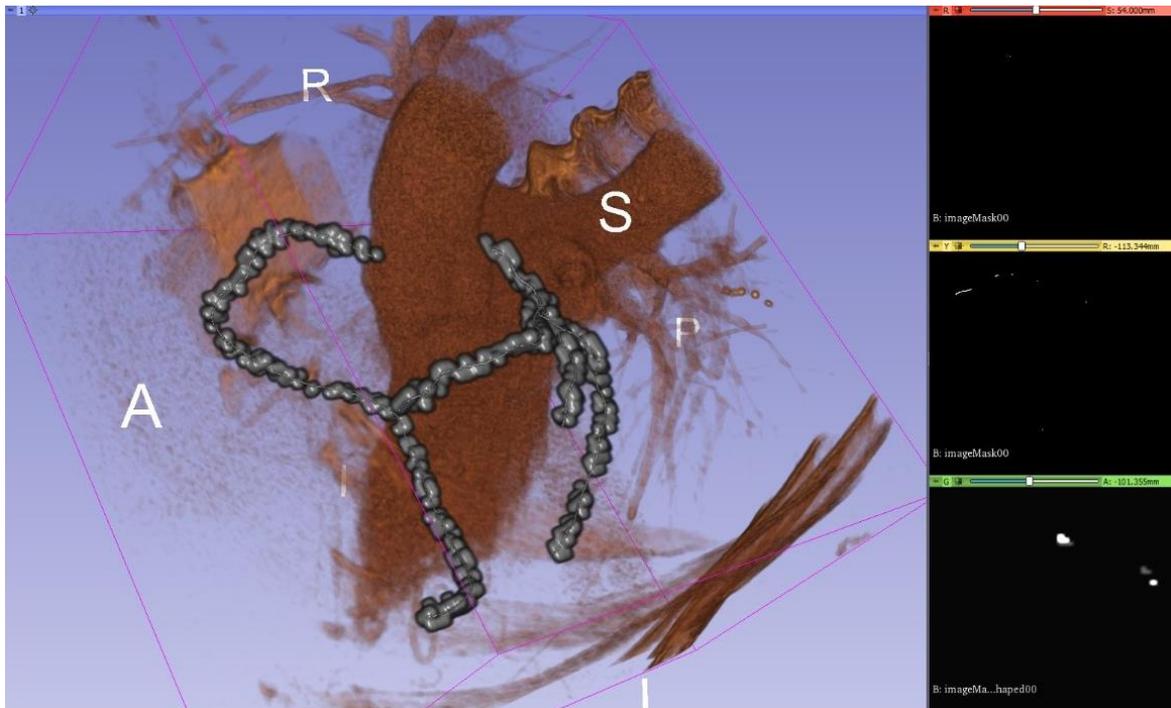
After the last expansion, another convolution is applied with the number of kernels equal to the number of features to extract. Since centerline extraction aims for binary segmentation, only one last kernel with a sigmoid activation function was required.

The loss function is chosen such that the segmentation behaves like a pixel-wise classification function.

### **3.3.2. Resize or patches**

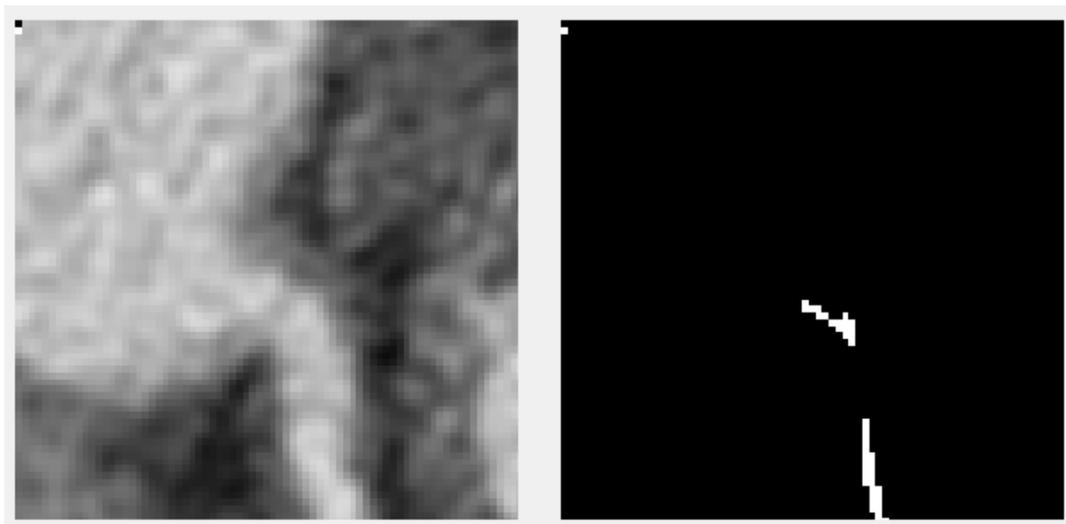
The limited amount of the memory of the GPU puts one in the impossibility of feeding an entire CT volume to a neural network. In 2D images, memory is not a problem, even when working with large batches. In 3D, the equivalent would be like feeding more than 500 images at once in a network.

Our first approach was to resize the volumes (and the ground truth) to a volume which would be small enough to fit in the video RAM, but large enough to not lose the details needed for segmentation. The first drawback was that the volumes from the benchmark did not come in a standard size. The X and Y axis were always of size 512, but the Z axis ranged from 272 to 388. This meant that the aspect ratio after any resize to a fixed value would not be constant. Another important loss would be the impossibility to have augmentation under the given circumstances, since even the 90-degree rotations would cause big ratio changes. The second drawback of this method was the loss in precision when upscaling. Extracting the centerline would imply getting as good a precision as possible, since the vessel fitting algorithms which take as input a centerline depend on this accuracy. Upscaling precision loss can be observed in Figure 27, where the ground truth centerline is discontinuous and inhomogeneous. Any algorithm without 100% precision would result in something even worse. Training the model using resizing posed problems with the dataset being too small.



**Figure 27** Downscaled and upscaled centerline ground truth (80x80x64) plotted in 3D to highlight the difficulty of using it for training

A second approach was proposed to divide the input into smaller patches, cutting only small parts of the volume (and the ground truth) and feeding them to the network. An example slice of a patch together with the associated ground truth is shown in Figure 28. This method does not have any of the downsides of the previous one, but introduces a different one: the lack of context. Therefore, a tradeoff is reached between increasing the patch size so that the available contextual information for the model is enough to make a good prediction and the size of the model so that the model is large enough to capture the correlations about where the centerline is positioned and that it should be continuous and also cross the patches.



**Figure 28** 2D slice of an augmented patch with ground truth next to it

### 3.3.3. Loss functions

Training neural networks means finding good parameters which make the network approximate a given objective function. For classification tasks having good accuracy is an objective. For image segmentation tasks good average pixel-wise classification accuracy could be an objective.

For gradually adjusting weights during the training, a loss function is required. The purpose of the training is to gradually minimize the loss function. A loss function should be chosen to quantify how much and in what direction to adjust the parameters such that on the next iteration the outputs are closer to the objective.

For image segmentation, numerous loss functions have been proposed in literature. Some of the existing loss functions for binary segmentation tasks are enumerated here. A new combined loss function which fits the objective of segmenting centerlines is proposed here.

#### 3.3.3.1. Local loss

*Cross entropy* (CE) is defined as:

$$CE(p, \hat{p}) = -(p \log(\hat{p}) + (1 - p) \log(1 - \hat{p})) \quad (1)$$

The predicted pixel class is the result of a sigmoid and is interpreted as a probability of belonging to a class. It is then compared with the true class (from the ground truth). Logistic regression is obtained using this loss function.

*Weighted cross entropy* (WCE) is defined similarly to the cross entropy but adds a coefficient for the positive examples:

$$WCE(p, \hat{p}) = -(\beta p \log(\hat{p}) + (1 - p) \log(1 - \hat{p})) \quad (2)$$

It serves in the case of class imbalance in the training dataset. The training might not converge in such cases. The  $\beta$  param controls the expected number of false positives or negatives.

*Balanced cross entropy* (BCE) is another variation which adds the coefficient for positive examples and the inverted coefficient for negative examples:

$$BCE(p, \hat{p}) = -(\beta p \log(\hat{p}) + (1 - \beta)(1 - p) \log(1 - \hat{p})) \quad (3)$$

*Focal loss*, introduced in [96] for dense object detection, aims to reduce the problem of huge class imbalances, where the cross entropy loss or any of its variations is not enough to achieve a stable training. It down-weights the “easy” examples to allow training on the rare events, thus the name of focal loss. It is also derived from the cross entropy, but adds a focusing parameter  $\gamma$ :

$$FL(p, \hat{p}) = -(\alpha(1 - \hat{p})^\gamma p \log(\hat{p}) + (1 - \alpha)\hat{p}^\gamma (1 - p) \log(1 - \hat{p})) \quad (4)$$

Another variation called *class balanced focal loss* was introduced in [97].

#### 3.3.3.2. Global loss

*Dice loss* (DL) aims to optimize the F1 score, defined as:

$$F1 = \frac{2TP}{2TP+FP+FN} = \frac{2|X \cap Y|}{|X|+|Y|}, \quad (5)$$

which is the harmonic mean between the precision and recall. It is considered an overlap measure, like the Jaccard Index. The loss function can be written as:

$$DL(p, \hat{p}) = 1 - \frac{2\sum p\hat{p}}{\sum p + \sum \hat{p}}, \quad (6)$$

where  $p \in \{0,1\}$  and  $\hat{p} \in [0, 1]$ .

*Balanced dice loss* (BDL), also known as Tversky loss, adds a  $\beta$  param to control the expected number of false positives or negatives. It is defined as:

$$BDL(p, \hat{p}) = 1 - \frac{p\hat{p}}{p\hat{p} + \beta(1-p)\hat{p} + (1-\beta)p(1-\hat{p})} \quad (7)$$

### 3.3.3.3. Combined loss functions

If none of the simple loss functions fits the objective of the neural network, or if the training does not converge, multiple standard loss functions can be combined. An example would be combining a dice coefficient loss with the cross-entropy loss:

$$\text{Combined} = \text{CE}(p, \hat{p}) + \text{DL}(p, \hat{p}) \quad (8)$$

To do that, the cross-entropy, which outputs individual pixel loss values (called local information) needs to be summed with a loss function on the level of the entire image (called global information). The global loss can be spread to local losses or the local losses can be averaged.

*Distance to the border of the nearest cell* (DNC) was introduced for U-Net segmentation in [6]. It combines cross entropy with an averaged distance function for a positive class to the border of two nearest cells. Morphological operations are applied on the image beforehand to compute the border map in the training dataset. The distance to the cells is defined as:

$$w(p) = w_c(p) + w_0 \cdot \exp\left(-\frac{(d_1(p)+d_2(p))^2}{2\sigma^2}\right), \quad (9)$$

and the loss function as:

$$\text{DNC}(p, \hat{p}) = -(w(p)p \log(\hat{p}) + w(p)(1-p) \log(1-\hat{p})) \quad (10)$$

For the U2-Net segmentation [95], a loss function combining Lovász-Softmax (an overlap measure) with the focal loss was used.

### 3.3.3.4. Proposed loss function

None of the above-mentioned loss functions fit the specifics of our dataset. A function to work with sparsely annotated centerlines is needed, meaning that examples would be contradictory, and in the same time with a huge class imbalance.

The proposed function is a combination between the focal loss and a simple overlap loss. The overlap loss, defined as follows:

$$OL(p, \hat{p}) = 1 - \frac{\sum p \hat{p}}{\sum p}, \quad (11)$$

ensures the stability of the training in spite of the contradicting examples. Its purpose is to penalize the misclassification of positives, since the positive cases are very rare. The focal loss was the only loss with enough power to ignore the huge accuracy of choosing the “easy” example. The focal loss was used with the proposed parameters from the original paper  $\gamma = 2$  and  $\alpha = 0.25$ .

### 3.3.4. Generating the full output

Because the model takes the input in the shape of patches, it cannot make predictions using a full 3D volume. The solution employed was to split the input in blocks of the shape of the patch, pass each of the blocks through the model, and recombine the predictions into a full 3D volume again. If the prediction is qualitative, no artifacts are seen at the border of the combined patches when a segmentation threshold is set.

The result is exported as a 3D volume in the same format as the input volumes and generated input masks, so that it can be loaded by similar software. One can see in Figure 29 the output volume composed from patches, where the threshold is lowered to make it clear where the patches meet.

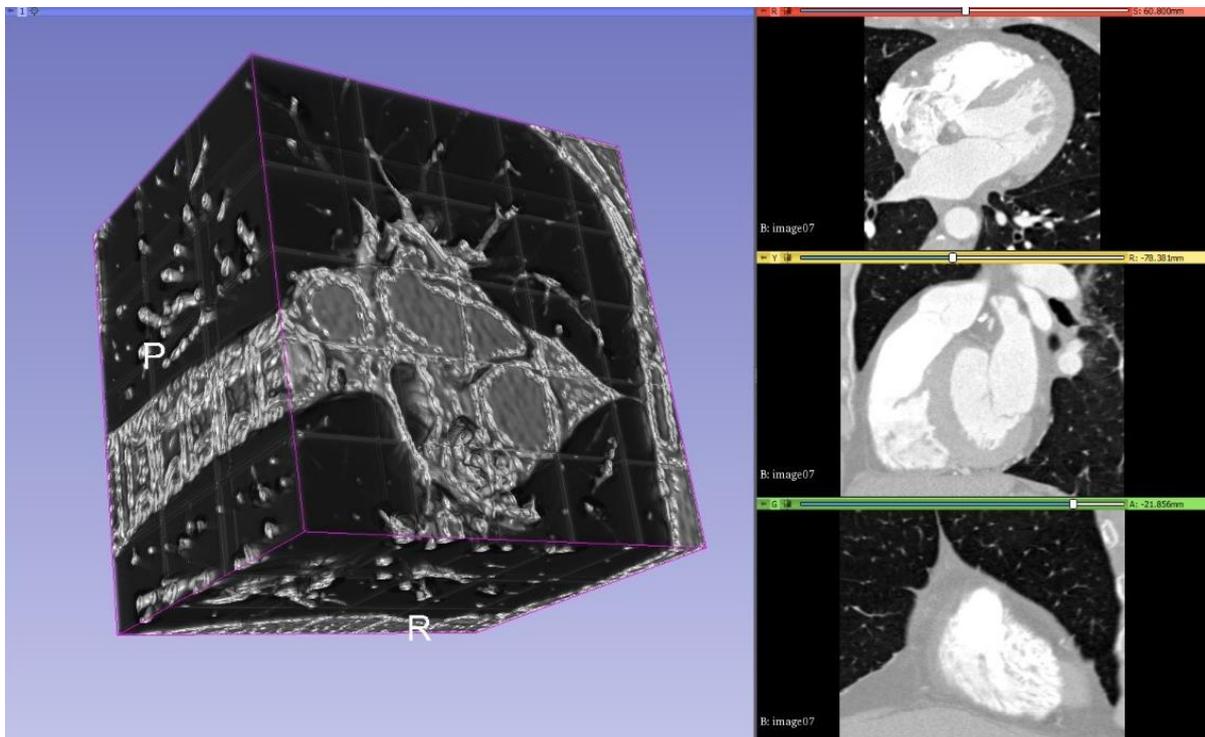


Figure 29 Output volume composed from patches with visible stitching

## 3.4. Experimental Setup

### 3.4.1. Coronary dataset

To test the implementation of the proposed neural network, we chose the dataset from the Rotterdam Coronary Artery Algorithm Evaluation Framework [7]. This is a public benchmark to evaluate algorithms for the task of Centerline Extraction from CTA data. The training dataset is publicly available, but non-anonymous registering is required. A standardized quantitative method for evaluating the centerline extraction algorithms is also provided [98].

The dataset is split in a training set and a testing set. The training set consists of 8 volumetric images and the test set consists of 24 volumetric images. All the images have four target vessel descriptions with each containing four reference points (described later). The training set images additionally contain the ground truth in the form of a reference file for the target vessels. This makes it challenging for supervised learning algorithms, since many of the vessels are not reported as part of the ground truth. The size and resolution for each training volume is described in Table 9.

The images are packed in two files. One is a “.mhd” ASCII human readable header file, which describes volume dimensions (e.g. 512x512x304), the offset and element spacing (voxel size in real world). The other is a “.raw” file which contains the voxel (volume pixel) intensities, expressed in Hounsfield Units (HU), following the formula  $\text{intensity} = \text{HU} + 1024$ , with a voxel value of zero corresponding to -1024 HU. The format is ITK compatible.

The ground truth reference files contain a list of the points that are on the centerline of the vessel. The points are described as x,y,z real world coordinates, the radius of the vessel at that point, and inter-observer variability at that position.

There are three categories for extraction: fully automated, minimal user interaction, and interactive extraction [99].

For the fully automated method, two points of reference are provided, one inside the distal part of the vessel, which uniquely identifies the vessel to be tracked, and one at a short distance to the starting point of the centerline.

For the minimal user interaction, only one point is allowed, but can be chosen from more options which include: the two points described before, a starting point of a centerline, an end point of a centerline, a user defined point. The first four points are provided with the downloadable data. A vessel tree can be obtained if one starts from the starting point of the centerline. If that is the case, other reference points can be used.

For the interactive extraction, any given number of reference points can be used, even user defined points by manual clicking.

**Table 9** The size and resolution of each CTA volume from the training dataset [84]

CTA	Size	Resolution (mm <sup>3</sup> )
dataset00	512x512x272	0.363x0.363x0.4
dataset01	512x512x338	0.363x0.363x0.4
dataset02	512x512x288	0.334x0.334x0.4
dataset03	512x512x276	0.371x0.371x0.4
dataset04	512x512x274	0.316x0.316x0.4
dataset05	512x512x274	0.322x0.322x0.4
dataset06	512x512x268	0.320x0.320x0.4
dataset07	512x512x304	0.287x0.287x0.4

### 3.4.2. Visualization

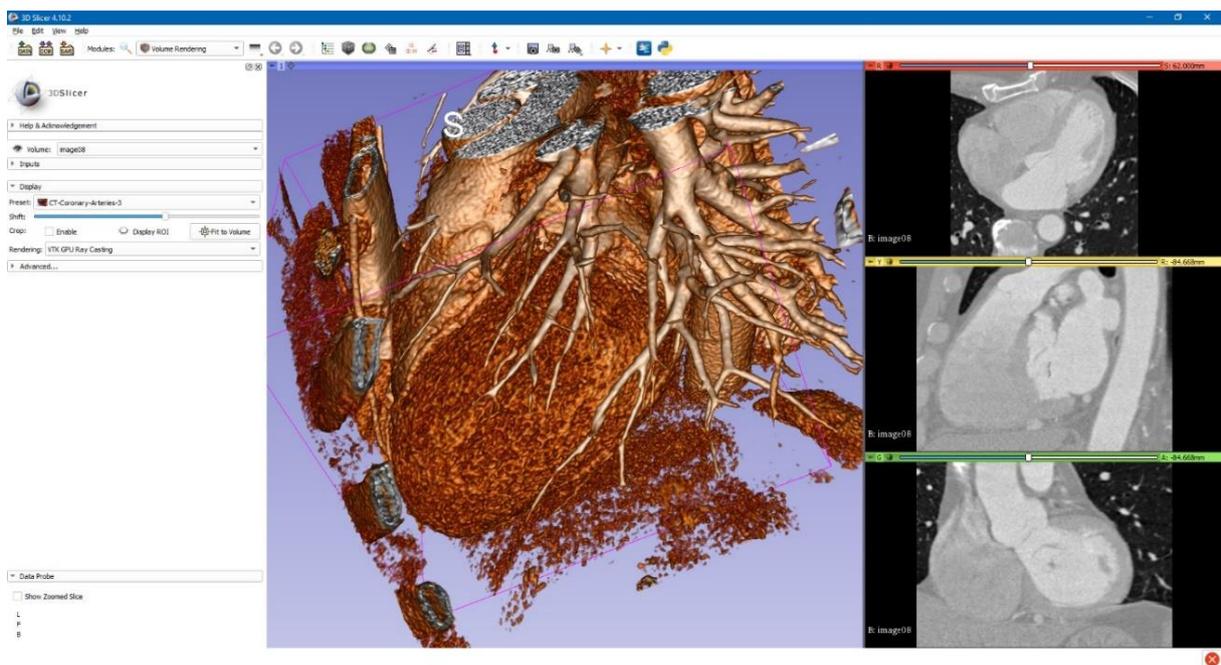
3D Slicer (available from [100]) is a free and open source software platform useful for three-dimensional visualization useful for image guided therapy (IGT), image processing with registration and interactive segmentation, and medical image informatics. It receives funding support from national health institutes and development support from a large worldwide community of developers with the code available on GitHub (available from [101]). It is directed by National Alliance for Medical Image Computing (NA-MIC), the Neuroimage Analysis Center (NAC), the National Center for Image-Guided Therapy (NCIGT) and others.

The code compiles cross-platform and it's extensible via plug-ins for different algorithms and applications. It supports python scripting for the customization of the look and workflow.

It supports loading DICOM file format for different type of image types like CT, MRI, nuclear medicine and microscopy. For usage, it contains functionality supported through modules for different organs and contains a bi-directional interface for integration with medical devices.

3D Slicer works with scenes which are a list of elements to be loaded like images, volumes, models, transforms, fiducial markers. Elements extensions include .vtk (Visualization ToolKit), .nrrd (Nearly Raw Raster Data), .png (Portable Network Graphics) and along with the scene file which is a .mrml (Medical Reality Modeling Language) they are archived into a .mrb file (Medical Reality Bundle).

Also, 3D Slicer provides an out-of-the-box module called Volume Rendering, used for visually inspect the input volumes, input masks and output predictions by overlapping them in a 3D space. It provides presets for easy rendering from volumetric images when trying to catch different aspects of the investigation. A “Shift” control allows the user to tamper with the thresholding for rendering features. An example rendering of the coronary arteries using one of the presets is presented in Figure 30.



**Figure 30** 3D Slicer Volume Rendering with CT-Coronary Arteries preset

More than this, the whole visualization process can be automated if needed using the SlicerJupyter software [102], which is an integration of a Jupyter Notebook with a Slicer Kernel which starts Slicer in the background but allows controlling it programmatically.

### 3.4.3. Execution setup

The algorithm was implemented in python using Tensorflow with Keras. The full source code is publicly available at [39]. The provided source code contains the dataset converter and the segmentation algorithm in the form of Python files with executable cells. The network layout is available in the subchapter 3.5.1 Network parameters

**Table 11**, Table 11, Table 12, Table 13, Table 14, Table 15. For loading the dataset, we used the SimpleITK python library, and for thinning the obtained centerlines, the skimage package was utilized.

The 8 input volumes with ground truth were split into a training set of seven and a validation set with the remaining volume, leaving one volume out, therefore the holdout validation was employed. While the number is small, using augmented patches helps reduce the problem of a really small dataset.

The models were trained and tested on an NVIDIA GeForce GTX 1060 with 6GB VRAM. Training required around 8 hours on the formerly mentioned video card. The predicting processing time for a patch size input varies between 400ms and 600ms, depending on the patch size and model size. For a full volume, this time is multiplied by the number of patches inside a volume. At around 80 patches and 500ms per patch, the full output is computed in around 40 seconds. Stitching and post-processing for saving is negligible in time. The training time makes hyperparameter tuning a strenuous operation, and better configurations can be found by integrating the algorithm into automated design space exploration frameworks. The parameter search space must be more rigidly defined, and the parameters proposed here are a good start.

Following the format of the reference file, we have created a tool to generate a volumetric mask for each input image of the training set, which serves as the ground truth in the supervised learning algorithm. The tool can be parametrized to specify the width in voxels of the centerline. Instead of having a single voxel for one reference point, a cube of a given size is generated, with the center voxel overlapping the centerline. This was found useful in combating the huge class imbalance when working with single voxel wide centerline. The tool outputs the mask in the same format as the input images, with the exception that it can compress the raw data, which is useful when the mask is very sparse. The network will learn centerlines which are thicker than one voxel. A new step of morphological thinning (as described in [103]) must be applied to convert the output of the network into thin centerlines. The overlap and the accuracy are presented in respect to the wide centerlines, not the thinned ones.

### 3.4.4. Training the network

The training of the proposed U-NET architecture was performed with different configurations for the following parameters:

- Input patch size
- Layer reduction
- Batch size
- Feeding or not the network patches with no single voxel of ground truth

The hard constraint put on the parameters is given by the limited amount of VRAM of the video card. An example constraint: if the input path size was increased, the batch size or the number of convolutional kernels in the layers needs to be reduced.

The input patch size is described in voxel dimensions as an (X, Y, Z) pair. The tested patch sizes were: 128x128x96, 128x128x128, 256x256x128, 320x320x64, 384x384x48.

The layer reduction is the divisor of the number of convolution kernels (the number of output filters in the convolution) for each convolutional layer. Batch size is the number of input volumes fed to the network before backpropagating the error.

When the ground truth is very sparsely distributed along the volume, many of the patches will not have any parts of the segmentation. As measured by experiment, this tends to make the network fall into a local optimum and to always say that there is never anything to segment in the input volume. To prevent this, for patches of a smaller size the network is fed only inputs which contain at least one segmented ground truth voxel.

A difficult challenge in the biomedical image segmentation is the small number of images available for training neural networks. A way to get past this problem is to use augmentation.

The *Batchgenerators* framework [8] was used because it provided a wide range of transforms and included spatial augmentation, suitable for the 3D input data. The framework allows single or multithreaded computation of the augmented data. An alternative framework which uses the GPU to faster compute some of the transformations is available in [95].

The following spatial transformations were used:

- Random cropping. This provides patch size outputs with the center randomly selected from inside the input volume. We do not allow cropping to go outside the borders of the input (and thus we need no padding)
- Elastic deformations with a deformation scale of 0 to 0.25. Deformation scale is expressed in percentage of the magnitude of deformation in respect to the patch size (in all dimensions)
- Rotation on all axes with randomly selected angles between -7 and 7 degrees
- Scaling, with a randomly selected zoom between 0.75 and 1.25
- Mirroring on all axes

The spatial transformations are applied with a probability per input sample. We used 0.1 for elastic deformation, 0.1 for rotations, 0.1 for scaling, and 0.5 for mirroring (for each axe).

The following color transformations were added:

- Brightness transformation, with a multiplier range from 0.7 to 1.5
- Gamma transformation, with a correction range between 0.5 and 2
- Reverse gamma transformation, with a correction range between 0.5 and 2. This is done by negating the image, applying the transform, then negating the image back

All the color transformations are applied with a probability of 0.15 per sample.

Noise transformations were also utilized:

- Gaussian noise, with a variance between 0 and 0.05
- Gaussian blur, with a sigma between 0.5 and 1.5

The noise transforms are applied with a probability of 0.15 per sample.

Only the spatial transformations are applied, with the same parameters, to the ground truth. This is done in order to keep the annotation in sync with the input volume. Without augmentations, the dataset was too small for the training to converge.

Elastic deformations help create sufficiently realistic looking data in the case of vessels, whether 2D or 3D.

An alternative to the data augmentation presented above will be the use of synthetic data generated using Generative Adversarial Neural Networks (GAN), which were applied for medical imaging with success in increasing the accuracy for liver lesion classification [104].

### 3.5. Results

The classical notation of epoch, where epoch means one pass over the entire training dataset, will no longer apply when working with randomly cut patches. The term epoch is defined here by multiplying the number of volumes in the training set with the largest input volume size divided on each axis with the corresponding patch size, with the result divided by the batch size. As an example, for a patch size 256x256x128, a maximum size of an input volume of 512x512x388, and a batch size of 2, the number of input patches considered to be an epoch is  $7 * (512 / 256) * (512 / 256) * (388 / 128) / 2 = 7 * 2 * 2 * 3 / 2 = 42$ . This formula was used as a rough approximation of feeding the entire volumes once through the network. Because of this definition, the results are reported for different epochs given various input patch sizes. For the training dataset, the values for one epoch are computed as the average across all the input patches. For the validation dataset, the values for one epoch are computed by assembling the whole volume from the output patches and comparing it with the whole ground truth. The distribution of network parameters for the network configurations is available in Table 11, Table 12, Table 13, Table 14, and Table 15.

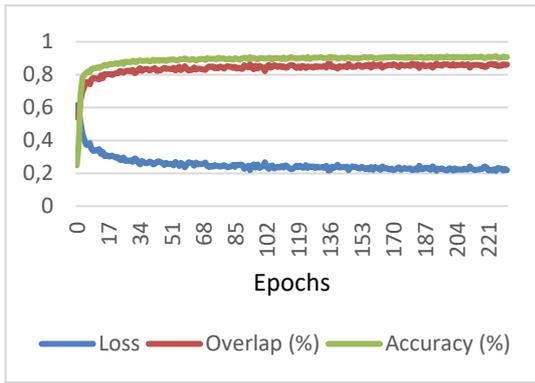
The overlap and accuracy measures on the validation data are presented in Table 10. Overlap is defined as the percentage of voxels correctly detected as part of the centerline (analogous to recall), with equation (11). Accuracy is defined as the percentage of voxels segmented as in the ground truth.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (12)$$

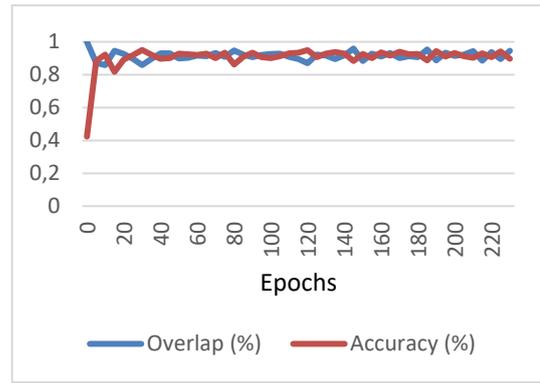
**Table 10** The overlap and accuracy results for the selected hyperparameters

Patch size	Batch size	Model Reduction	Patches with ground truth only	Epoch	Overlap	Accuracy
128x128x96	1	1	true	231	0.934	0.921
128x128x128	2	2	true	260	0.895	0.940
256x256x128	1	2	true	232	0.917	0.943
256x256x128	1	2	false	392	0.944	0.911
320x320x64	1	4	false	1250	0.883	0.953
384x384x48	1	4	false	1500	0.910	0.928

Since the ground truth is presented as sparse centerlines, meaning that only a subset of the coronary vessel tree is annotated, the results cannot be presented using the F1 score or Jaccard index. The number of “false positives” detected by the network will skew the results heavily. Instead, the results are presented in the form of a pair of binary accuracy and overlap (inspired by the results published in [86]). The graphs provided also include the value of the loss function to demonstrate that the learning converges, as depicted in Figure 31, Figure 32, Figure 33, Figure 34, Figure 35, Figure 36.

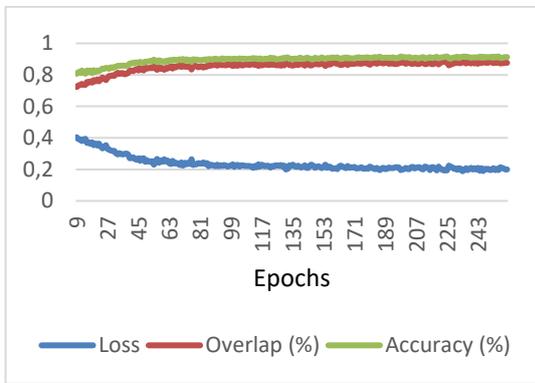


(a)

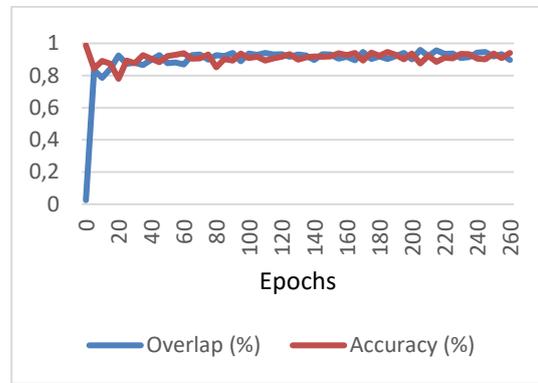


(b)

**Figure 31 (a) Training 128x128x96, batch size 1, reduction 1, only patches with ground truth (b) Validation**

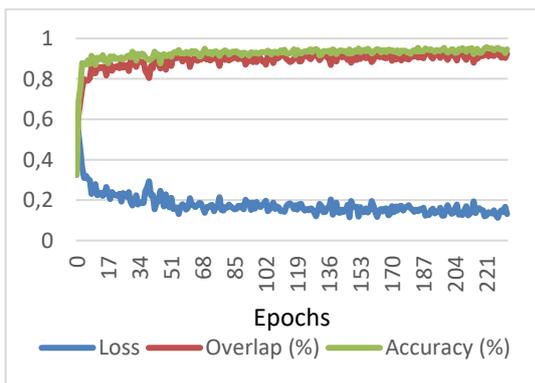


(a)

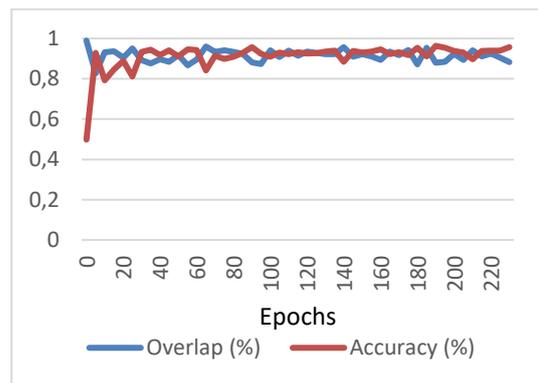


(b)

**Figure 32 (a) Training 128x128x128, batch size 2, reduction 2, only patches with ground truth (b) Validation**

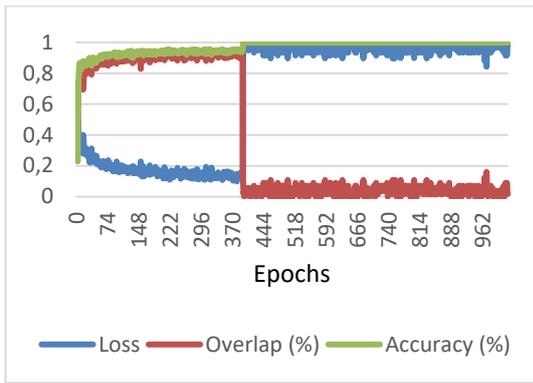


(a)

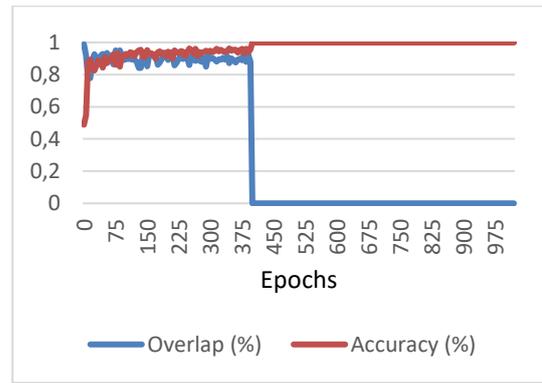


(b)

**Figure 33 (a) Training 256x256x128, batch size 1, reduction 2, only patches with ground truth (b) Validation**

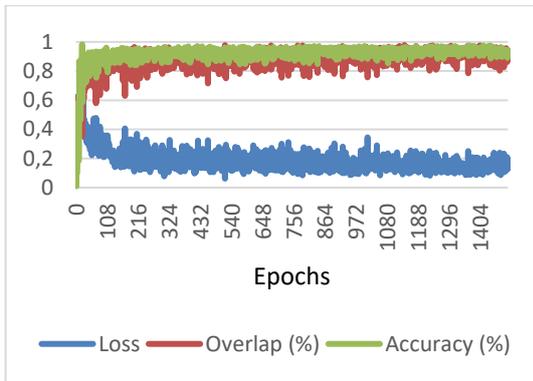


(a)

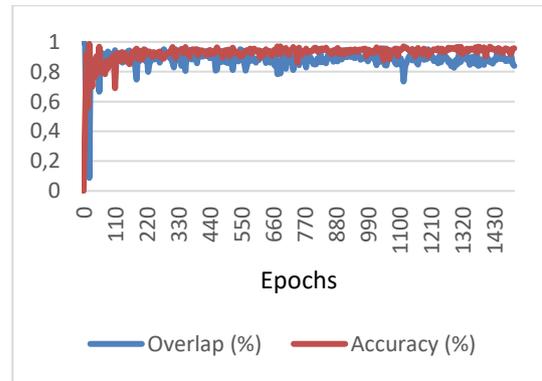


(b)

**Figure 34 (a)** Training 256x256x128, batch size 1, reduction 2 **(b)** Validation

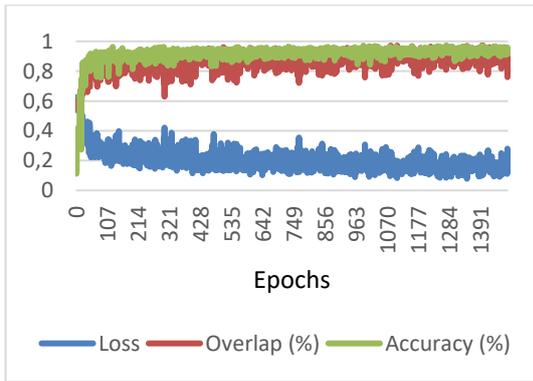


(a)

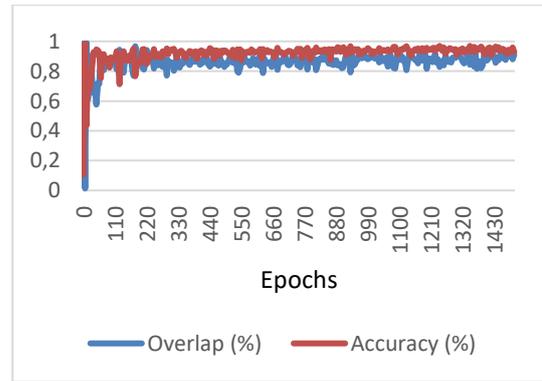


(b)

**Figure 35 (a)** Training 320x320x64, batch size 1, reduction 4 **(b)** Validation



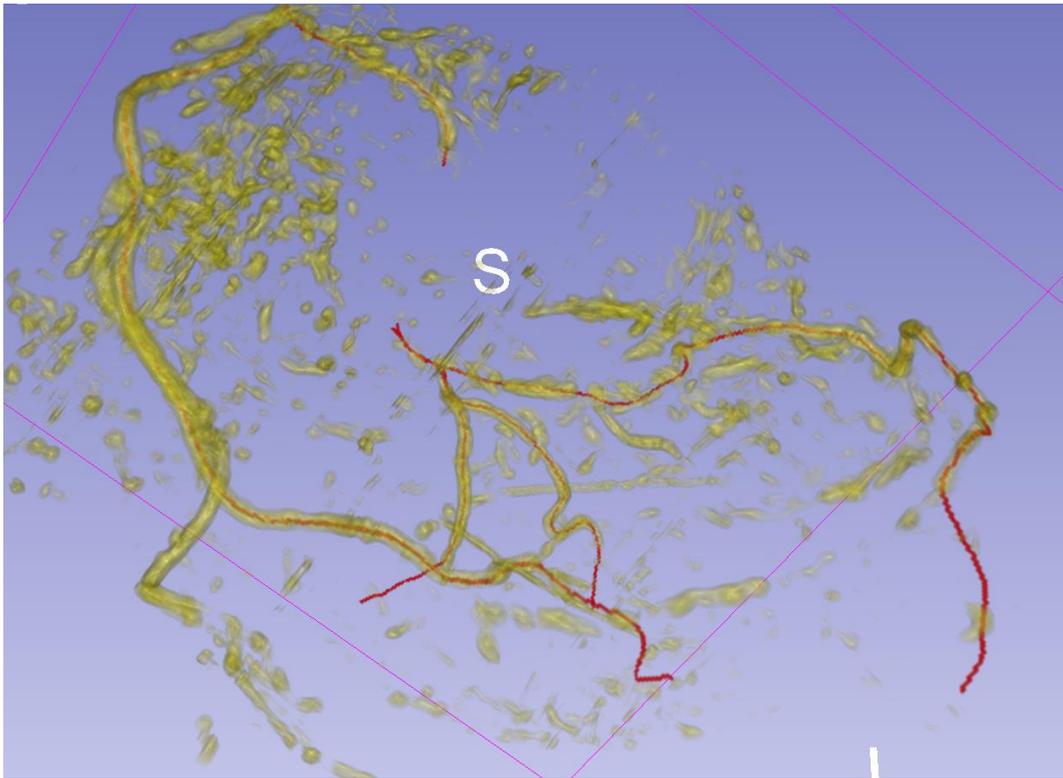
(a)



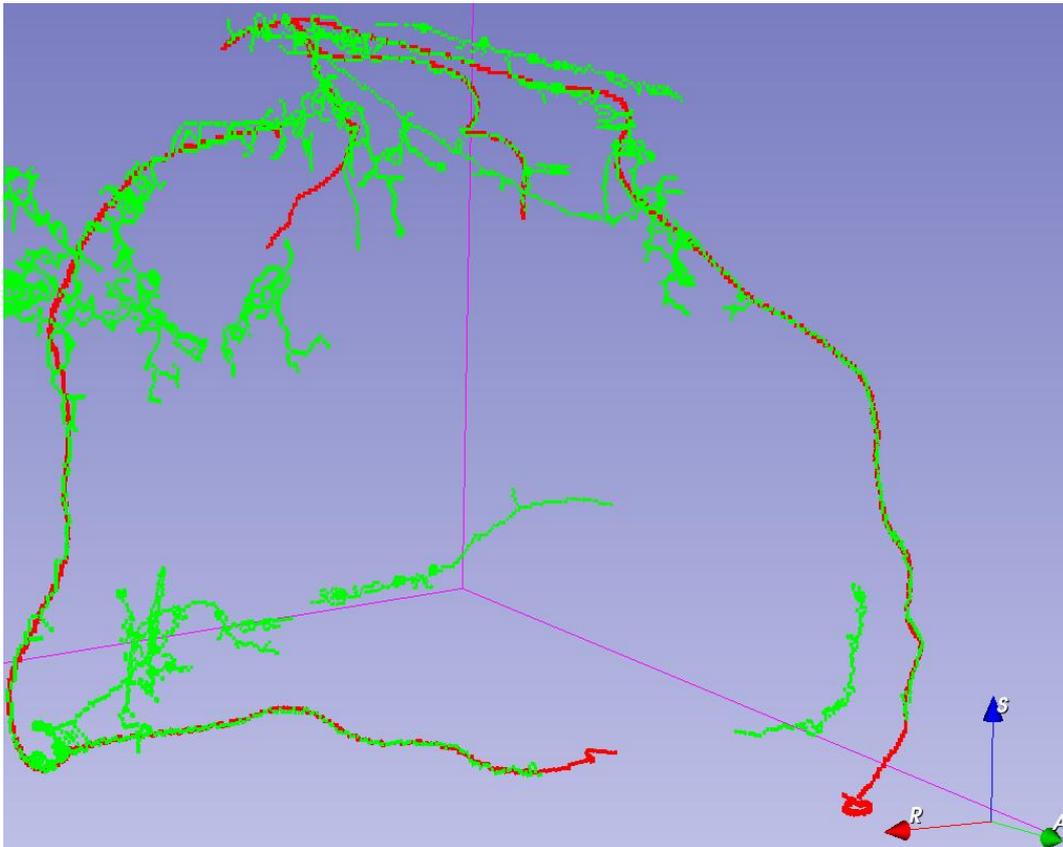
(b)

**Figure 36 (a)** Training 384x384x48, batch size 1, reduction 4 **(b)** Validation

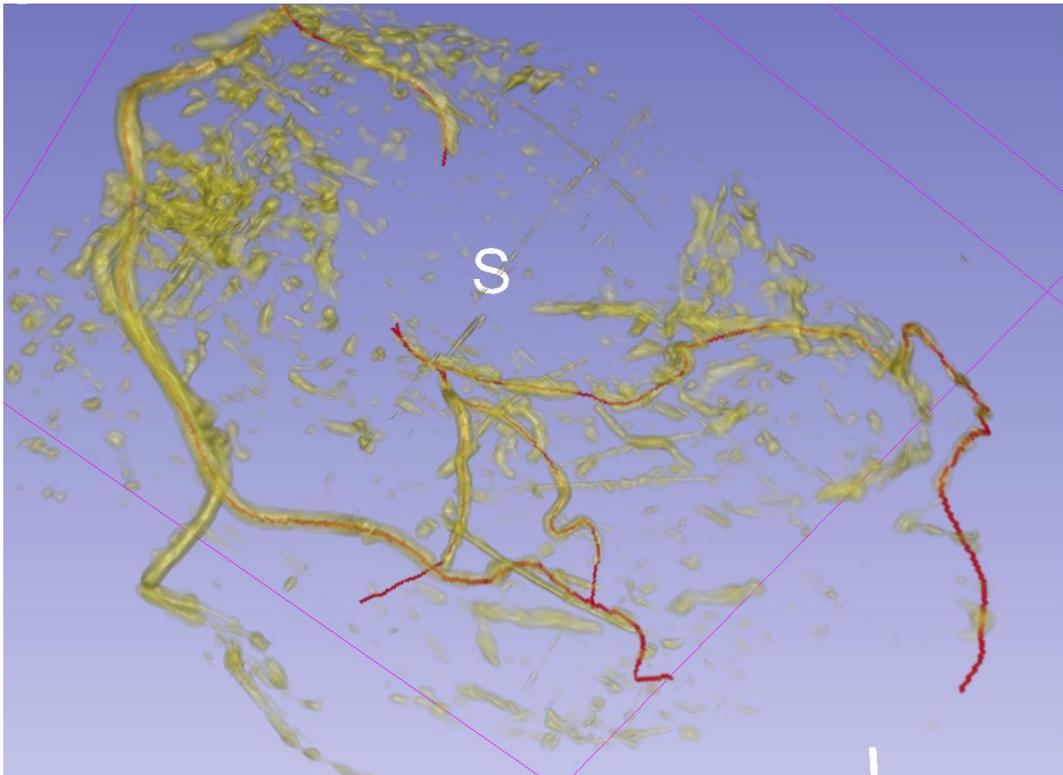
Visual validation for the output volumes is provided. The ground truth centerline is depicted in red. The segmentation for the centerlines is seen in transparent yellow (Figure 37, Figure 39, Figure 41, Figure 43, Figure 45, Figure 47), and for the thinned centerlines is seen in green (Figure 38, Figure 40, Figure 42, Figure 44, Figure 46, Figure 48), making it easier to check that the segmentation follows the centerline.



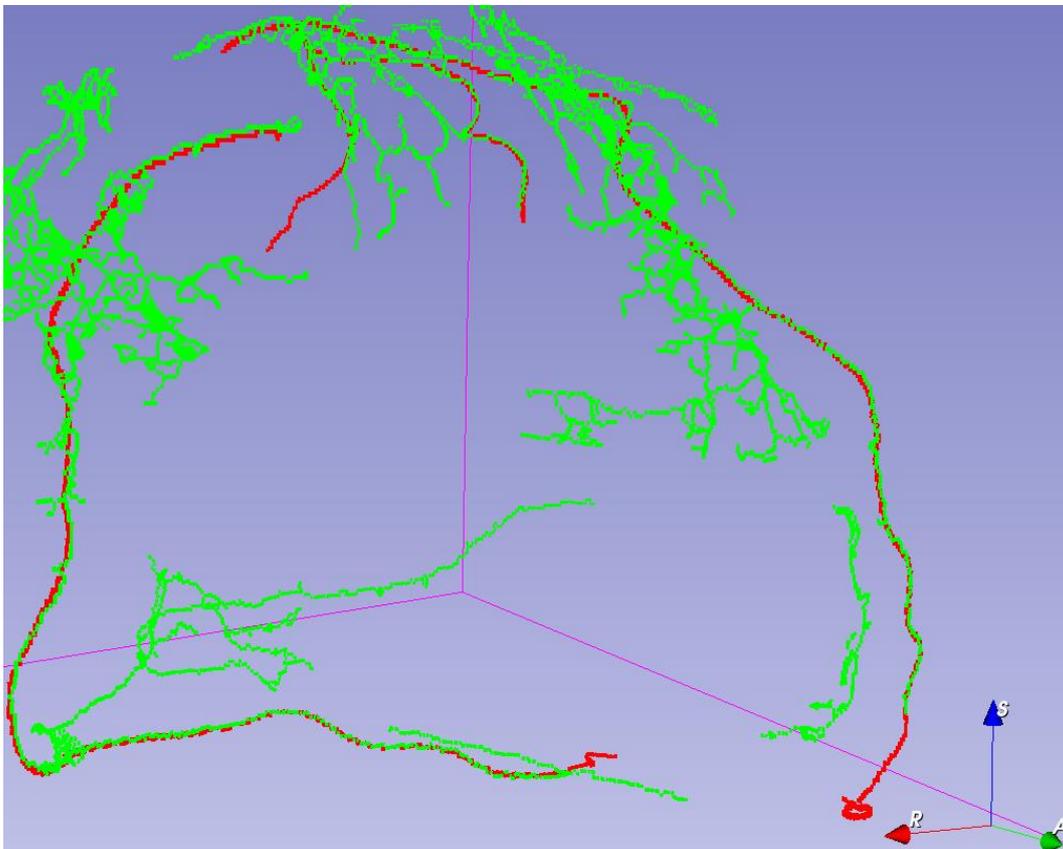
**Figure 37** Patch size 128x128x96 rendering of ground truth (red) and centerline segmentation (transparent yellow)



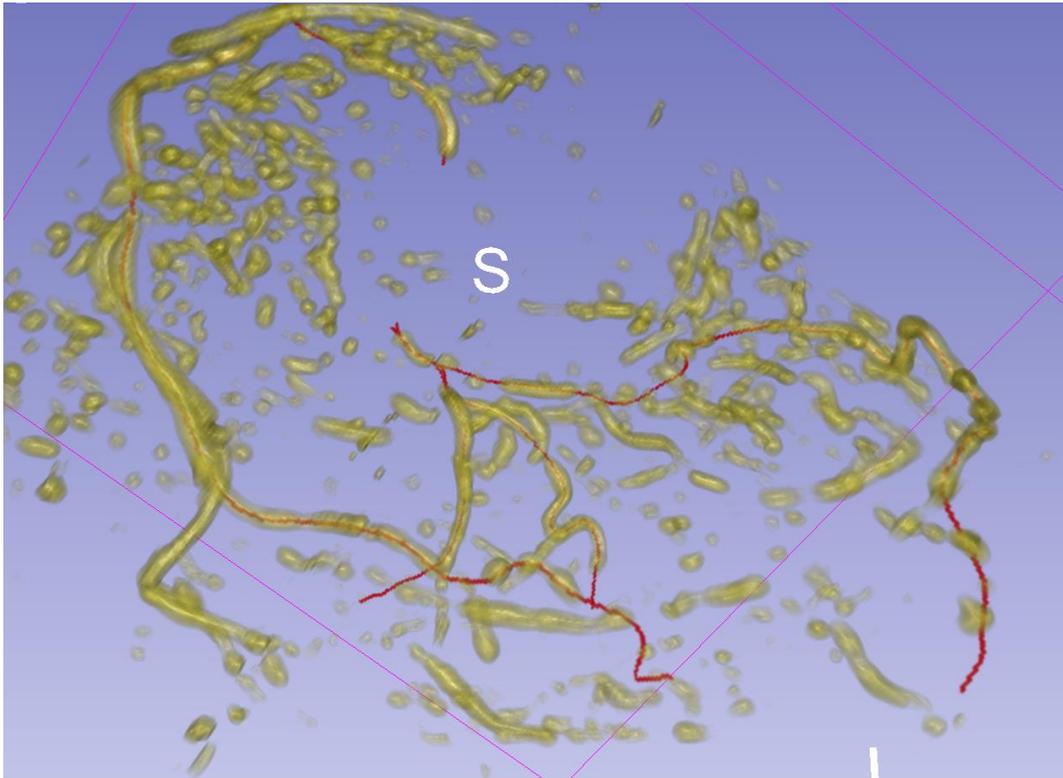
**Figure 38** Patch size 128x128x96 rendering of ground truth (red) and thinned centerline segmentation (green)



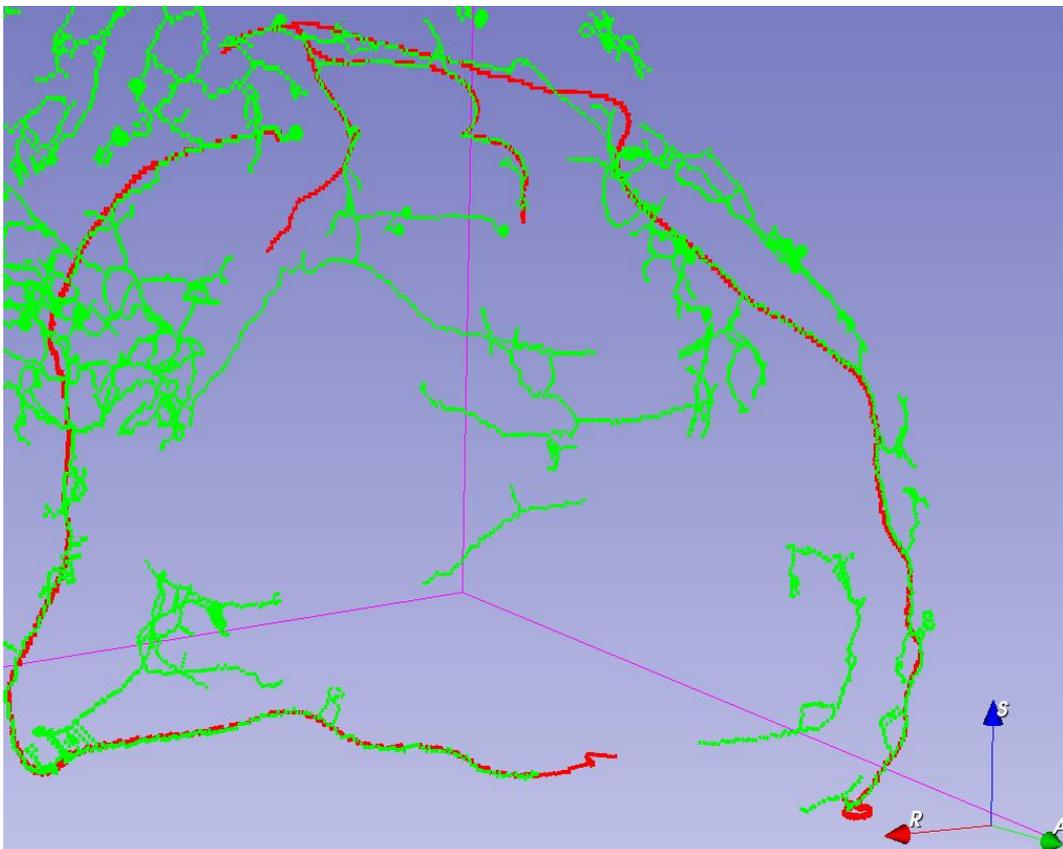
**Figure 39** Patch size 128x128x128 rendering of ground truth (red) and centerline segmentation (transparent yellow)



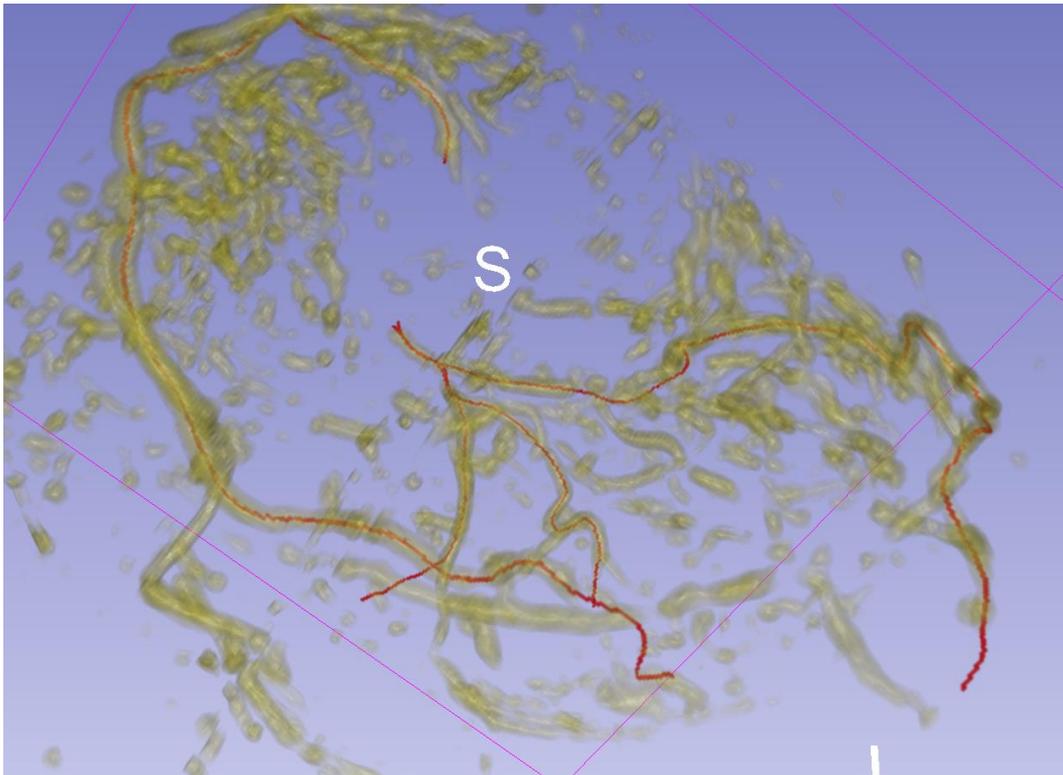
**Figure 40** Patch size 128x128x128 rendering of ground truth (red) and thinned centerline segmentation (green)



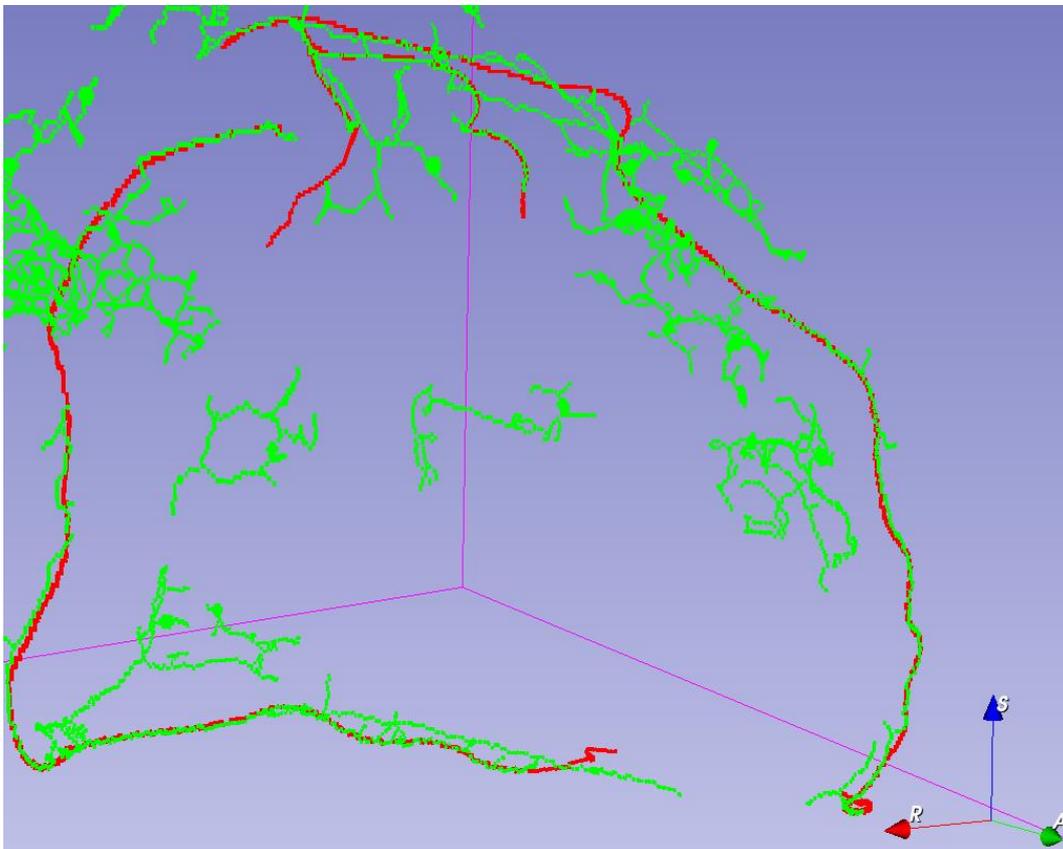
**Figure 41** Patch size 256x256x128, training with only patches with ground truth, rendering of ground truth (red) and centerline segmentation (transparent yellow)



**Figure 42** Patch size 256x256x128, training with only patches with ground truth, rendering of ground truth (red) and thinned centerline segmentation (green)



**Figure 43** Patch size 256x256x128 rendering of ground truth (red) and centerline segmentation (transparent yellow)



**Figure 44** Patch size 256x256x128 rendering of ground truth (red) and thinned centerline segmentation (green)

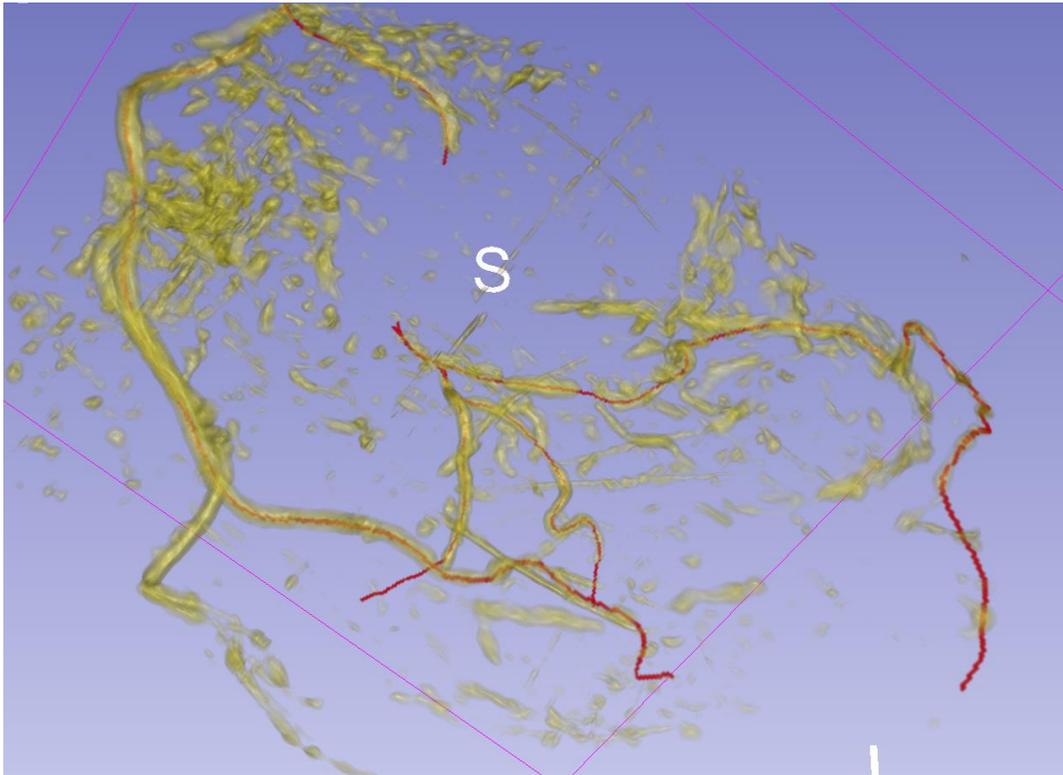


Figure 45 Patch size 320x320x64 rendering of ground truth (red) and centerline segmentation (transparent yellow)

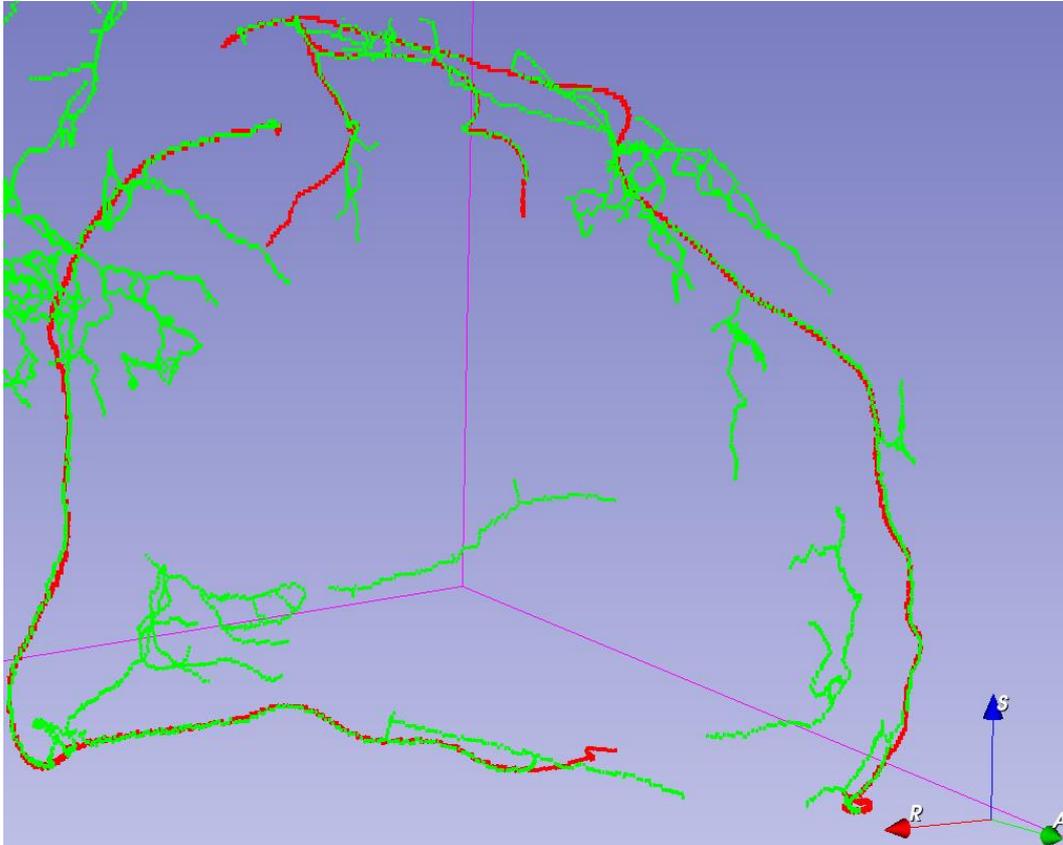
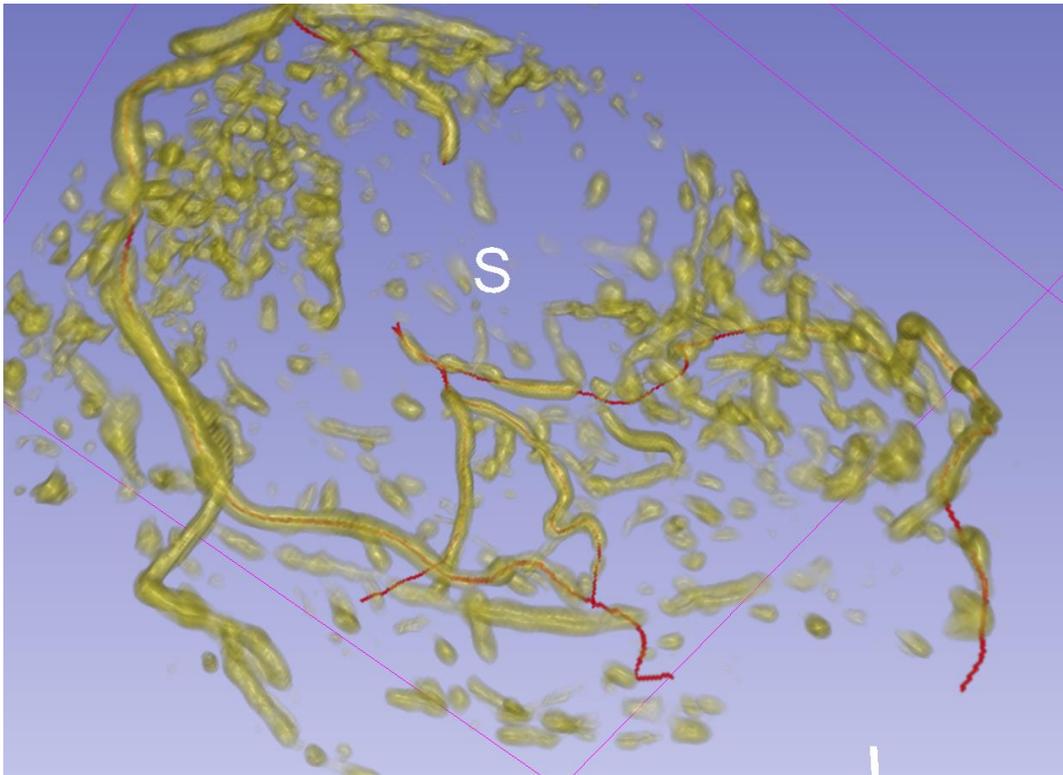
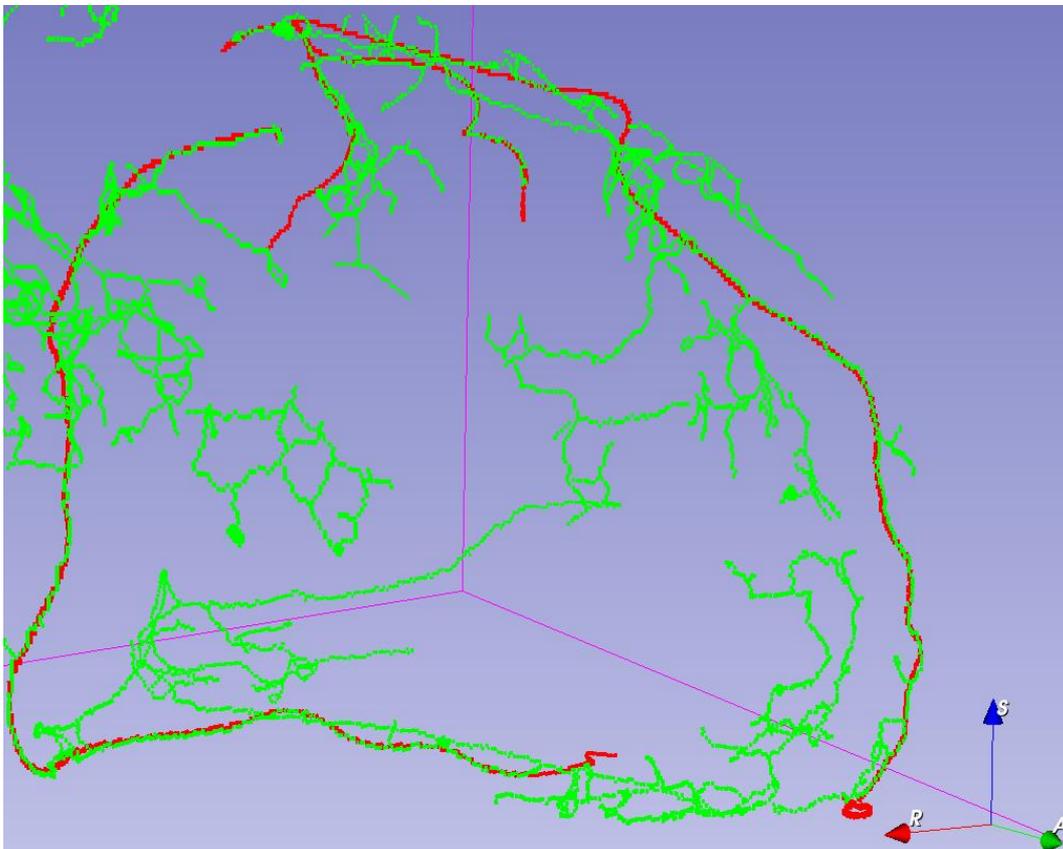


Figure 46 Patch size 320x320x64 rendering of ground truth (red) and thinned centerline segmentation (green)



**Figure 47** Patch size 384x384x48 rendering of ground truth (red) and centerline segmentation (transparent yellow)



**Figure 48** Patch size 384x384x48 rendering of ground truth (red) and thinned centerline segmentation (green)

### 3.5.1 Network parameters

**Table 11** Network parameters for Input Size 128x128x96, model reduction 1

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1, 96, 128, 128)	0	
conv3d (Conv3D)	(None, 32, 96, 128, 128)	896	input_1[0][0]
conv1 (Conv3D)	(None, 32, 96, 128, 128)	27680	conv3d[0][0]
max_pooling3d (MaxPooling3D)	(None, 32, 48, 64, 64)	0	conv1[0][0]
conv3d_1 (Conv3D)	(None, 64, 48, 64, 64)	55360	max_pooling3d[0][0]
conv3d_2 (Conv3D)	(None, 64, 48, 64, 64)	110656	conv3d_1[0][0]
max_pooling3d_1 (MaxPooling3D)	(None, 64, 24, 32, 32)	0	conv3d_2[0][0]
conv3d_3 (Conv3D)	(None, 128, 24, 32, 32)	221312	max_pooling3d_1[0][0]
conv3 (Conv3D)	(None, 128, 24, 32, 32)	442496	conv3d_3[0][0]
max_pooling3d_2 (MaxPooling3D)	(None, 128, 12, 16, 16)	0	conv3[0][0]
conv3d_4 (Conv3D)	(None, 256, 12, 16, 16)	884992	max_pooling3d_2[0][0]
conv3d_5 (Conv3D)	(None, 256, 12, 16, 16)	1769728	conv3d_4[0][0]
max_pooling3d_3 (MaxPooling3D)	(None, 256, 6, 8, 8)	0	conv3d_5[0][0]
conv3d_6 (Conv3D)	(None, 512, 6, 8, 8)	3539456	max_pooling3d_3[0][0]
conv5 (Conv3D)	(None, 512, 6, 8, 8)	7078400	conv3d_6[0][0]
up_sampling3d (UpSampling3D)	(None, 512, 12, 16, 16)	0	conv5[0][0]
concatenate (Concatenate)	(None, 768, 12, 16, 16)	0	up_sampling3d[0][0] conv3d_5[0][0]
conv3d_7 (Conv3D)	(None, 256, 12, 16, 16)	5308672	concatenate[0][0]
conv3d_8 (Conv3D)	(None, 256, 12, 16, 16)	1769728	conv3d_7[0][0]
up_sampling3d_1 (UpSampling3D)	(None, 256, 24, 32, 32)	0	conv3d_8[0][0]
concatenate_1 (Concatenate)	(None, 384, 24, 32, 32)	0	up_sampling3d_1[0][0] conv3[0][0]
conv3d_9 (Conv3D)	(None, 128, 24, 32, 32)	1327232	concatenate_1[0][0]
conv7 (Conv3D)	(None, 128, 24, 32, 32)	442496	conv3d_9[0][0]
up_sampling3d_2 (UpSampling3D)	(None, 128, 48, 64, 64)	0	conv7[0][0]
concatenate_2 (Concatenate)	(None, 192, 48, 64, 64)	0	up_sampling3d_2[0][0] conv3d_2[0][0]
conv3d_10 (Conv3D)	(None, 64, 48, 64, 64)	331840	concatenate_2[0][0]
conv3d_11 (Conv3D)	(None, 64, 48, 64, 64)	110656	conv3d_10[0][0]
up_sampling3d_3 (UpSampling3D)	(None, 64, 96, 128, 128)	0	conv3d_11[0][0]
concatenate_3 (Concatenate)	(None, 96, 96, 128, 128)	0	up_sampling3d_3[0][0] conv1[0][0]
conv3d_12 (Conv3D)	(None, 32, 96, 128, 128)	82976	concatenate_3[0][0]
conv9 (Conv3D)	(None, 32, 96, 128, 128)	27680	conv3d_12[0][0]
conv3d_13 (Conv3D)	(None, 1, 96, 128, 128)	33	conv9[0][0]
Total params: 23,532,289			
Trainable params: 23,532,289			
Non-trainable params: 0			

**Table 12** Network parameters for Input Size 128x128x128, model reduction 2

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1, 128, 128, 128)	0	
conv3d (Conv3D)	(None, 16, 128, 128, 128)	448	input_1[0][0]
conv1 (Conv3D)	(None, 16, 128, 128, 128)	6928	conv3d[0][0]
max_pooling3d (MaxPooling3D)	(None, 16, 64, 64, 64)	0	conv1[0][0]
conv3d_1 (Conv3D)	(None, 32, 64, 64, 64)	13856	max_pooling3d[0][0]
conv3d_2 (Conv3D)	(None, 32, 64, 64, 64)	27680	conv3d_1[0][0]
max_pooling3d_1 (MaxPooling3D)	(None, 32, 32, 32, 32)	0	conv3d_2[0][0]
conv3d_3 (Conv3D)	(None, 64, 32, 32, 32)	55360	max_pooling3d_1[0][0]
conv3 (Conv3D)	(None, 64, 32, 32, 32)	110656	conv3d_3[0][0]
max_pooling3d_2 (MaxPooling3D)	(None, 64, 16, 16, 16)	0	conv3[0][0]
conv3d_4 (Conv3D)	(None, 128, 16, 16, 16)	221312	max_pooling3d_2[0][0]
conv3d_5 (Conv3D)	(None, 128, 16, 16, 16)	442496	conv3d_4[0][0]
max_pooling3d_3 (MaxPooling3D)	(None, 128, 8, 8, 8)	0	conv3d_5[0][0]
conv3d_6 (Conv3D)	(None, 256, 8, 8, 8)	884992	max_pooling3d_3[0][0]
conv5 (Conv3D)	(None, 256, 8, 8, 8)	1769728	conv3d_6[0][0]
up_sampling3d (UpSampling3D)	(None, 256, 16, 16, 16)	0	conv5[0][0]
concatenate (Concatenate)	(None, 384, 16, 16, 16)	0	up_sampling3d[0][0] conv3d_5[0][0]
conv3d_7 (Conv3D)	(None, 128, 16, 16, 16)	1327232	concatenate[0][0]
conv3d_8 (Conv3D)	(None, 128, 16, 16, 16)	442496	conv3d_7[0][0]
up_sampling3d_1 (UpSampling3D)	(None, 128, 32, 32, 32)	0	conv3d_8[0][0]
concatenate_1 (Concatenate)	(None, 192, 32, 32, 32)	0	up_sampling3d_1[0][0] conv3[0][0]
conv3d_9 (Conv3D)	(None, 64, 32, 32, 32)	331840	concatenate_1[0][0]
conv7 (Conv3D)	(None, 64, 32, 32, 32)	110656	conv3d_9[0][0]
up_sampling3d_2 (UpSampling3D)	(None, 64, 64, 64, 64)	0	conv7[0][0]
concatenate_2 (Concatenate)	(None, 96, 64, 64, 64)	0	up_sampling3d_2[0][0] conv3d_2[0][0]
conv3d_10 (Conv3D)	(None, 32, 64, 64, 64)	82976	concatenate_2[0][0]
conv3d_11 (Conv3D)	(None, 32, 64, 64, 64)	27680	conv3d_10[0][0]
up_sampling3d_3 (UpSampling3D)	(None, 32, 128, 128, 128)	0	conv3d_11[0][0]
concatenate_3 (Concatenate)	(None, 48, 128, 128, 128)	0	up_sampling3d_3[0][0] conv1[0][0]
conv3d_12 (Conv3D)	(None, 16, 128, 128, 128)	20752	concatenate_3[0][0]
conv9 (Conv3D)	(None, 16, 128, 128, 128)	6928	conv3d_12[0][0]
conv3d_13 (Conv3D)	(None, 1, 128, 128, 128)	17	conv9[0][0]
Total params: 5,884,033			
Trainable params: 5,884,033			
Non-trainable params: 0			

**Table 13** Network parameters for Input Size 256x256x128, model reduction 4

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1, 128, 256, 256)	0	
conv3d (Conv3D)	(None, 8, 128, 256, 256)	224	input_1[0][0]
conv1 (Conv3D)	(None, 8, 128, 256, 256)	1736	conv3d[0][0]
max_pooling3d (MaxPooling3D)	(None, 8, 64, 128, 128)	0	conv1[0][0]
conv3d_1 (Conv3D)	(None, 16, 64, 128, 128)	3472	max_pooling3d[0][0]
conv3d_2 (Conv3D)	(None, 16, 64, 128, 128)	6928	conv3d_1[0][0]
max_pooling3d_1 (MaxPooling3D)	(None, 16, 32, 64, 64)	0	conv3d_2[0][0]
conv3d_3 (Conv3D)	(None, 32, 32, 64, 64)	13856	max_pooling3d_1[0][0]
conv3 (Conv3D)	(None, 32, 32, 64, 64)	27680	conv3d_3[0][0]
max_pooling3d_2 (MaxPooling3D)	(None, 32, 16, 32, 32)	0	conv3[0][0]
conv3d_4 (Conv3D)	(None, 64, 16, 32, 32)	55360	max_pooling3d_2[0][0]
conv3d_5 (Conv3D)	(None, 64, 16, 32, 32)	110656	conv3d_4[0][0]
max_pooling3d_3 (MaxPooling3D)	(None, 64, 8, 16, 16)	0	conv3d_5[0][0]
conv3d_6 (Conv3D)	(None, 128, 8, 16, 16)	221312	max_pooling3d_3[0][0]
conv5 (Conv3D)	(None, 128, 8, 16, 16)	442496	conv3d_6[0][0]
up_sampling3d (UpSampling3D)	(None, 128, 16, 32, 32)	0	conv5[0][0]
concatenate (Concatenate)	(None, 192, 16, 32, 32)	0	up_sampling3d[0][0] conv3d_5[0][0]
conv3d_7 (Conv3D)	(None, 64, 16, 32, 32)	331840	concatenate[0][0]
conv3d_8 (Conv3D)	(None, 64, 16, 32, 32)	110656	conv3d_7[0][0]
up_sampling3d_1 (UpSampling3D)	(None, 64, 32, 64, 64)	0	conv3d_8[0][0]
concatenate_1 (Concatenate)	(None, 96, 32, 64, 64)	0	up_sampling3d_1[0][0] conv3[0][0]
conv3d_9 (Conv3D)	(None, 32, 32, 64, 64)	82976	concatenate_1[0][0]
conv7 (Conv3D)	(None, 32, 32, 64, 64)	27680	conv3d_9[0][0]
up_sampling3d_2 (UpSampling3D)	(None, 32, 64, 128, 128)	0	conv7[0][0]
concatenate_2 (Concatenate)	(None, 48, 64, 128, 128)	0	up_sampling3d_2[0][0] conv3d_2[0][0]
conv3d_10 (Conv3D)	(None, 16, 64, 128, 128)	20752	concatenate_2[0][0]
conv3d_11 (Conv3D)	(None, 16, 64, 128, 128)	6928	conv3d_10[0][0]
up_sampling3d_3 (UpSampling3D)	(None, 16, 128, 256, 256)	0	conv3d_11[0][0]
concatenate_3 (Concatenate)	(None, 24, 128, 256, 256)	0	up_sampling3d_3[0][0] conv1[0][0]
conv3d_12 (Conv3D)	(None, 8, 128, 256, 256)	5192	concatenate_3[0][0]
conv9 (Conv3D)	(None, 8, 128, 256, 256)	1736	conv3d_12[0][0]
conv3d_13 (Conv3D)	(None, 1, 128, 256, 256)	9	conv9[0][0]

Total params: 1,471,489  
 Trainable params: 1,471,489  
 Non-trainable params: 0

**Table 14** Network parameters for Input Size 320x320x64, model reduction 4

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1, 64, 320, 320)	0	
conv3d (Conv3D)	(None, 8, 64, 320, 320)	224	input_1[0][0]
conv1 (Conv3D)	(None, 8, 64, 320, 320)	1736	conv3d[0][0]
max_pooling3d (MaxPooling3D)	(None, 8, 32, 160, 160)	0	conv1[0][0]
conv3d_1 (Conv3D)	(None, 16, 32, 160, 160)	3472	max_pooling3d[0][0]
conv3d_2 (Conv3D)	(None, 16, 32, 160, 160)	6928	conv3d_1[0][0]
max_pooling3d_1 (MaxPooling3D)	(None, 16, 16, 80, 80)	0	conv3d_2[0][0]
conv3d_3 (Conv3D)	(None, 32, 16, 80, 80)	13856	max_pooling3d_1[0][0]
conv3 (Conv3D)	(None, 32, 16, 80, 80)	27680	conv3d_3[0][0]
max_pooling3d_2 (MaxPooling3D)	(None, 32, 8, 40, 40)	0	conv3[0][0]
conv3d_4 (Conv3D)	(None, 64, 8, 40, 40)	55360	max_pooling3d_2[0][0]
conv3d_5 (Conv3D)	(None, 64, 8, 40, 40)	110656	conv3d_4[0][0]
max_pooling3d_3 (MaxPooling3D)	(None, 64, 4, 20, 20)	0	conv3d_5[0][0]
conv3d_6 (Conv3D)	(None, 128, 4, 20, 20)	221312	max_pooling3d_3[0][0]
conv5 (Conv3D)	(None, 128, 4, 20, 20)	442496	conv3d_6[0][0]
up_sampling3d (UpSampling3D)	(None, 128, 8, 40, 40)	0	conv5[0][0]
concatenate (Concatenate)	(None, 192, 8, 40, 40)	0	up_sampling3d[0][0] conv3d_5[0][0]
conv3d_7 (Conv3D)	(None, 64, 8, 40, 40)	331840	concatenate[0][0]
conv3d_8 (Conv3D)	(None, 64, 8, 40, 40)	110656	conv3d_7[0][0]
up_sampling3d_1 (UpSampling3D)	(None, 64, 16, 80, 80)	0	conv3d_8[0][0]
concatenate_1 (Concatenate)	(None, 96, 16, 80, 80)	0	up_sampling3d_1[0][0] conv3[0][0]
conv3d_9 (Conv3D)	(None, 32, 16, 80, 80)	82976	concatenate_1[0][0]
conv7 (Conv3D)	(None, 32, 16, 80, 80)	27680	conv3d_9[0][0]
up_sampling3d_2 (UpSampling3D)	(None, 32, 32, 160, 160)	0	conv7[0][0]
concatenate_2 (Concatenate)	(None, 48, 32, 160, 160)	0	up_sampling3d_2[0][0] conv3d_2[0][0]
conv3d_10 (Conv3D)	(None, 16, 32, 160, 160)	20752	concatenate_2[0][0]
conv3d_11 (Conv3D)	(None, 16, 32, 160, 160)	6928	conv3d_10[0][0]
up_sampling3d_3 (UpSampling3D)	(None, 16, 64, 320, 320)	0	conv3d_11[0][0]
concatenate_3 (Concatenate)	(None, 24, 64, 320, 320)	0	up_sampling3d_3[0][0] conv1[0][0]
conv3d_12 (Conv3D)	(None, 8, 64, 320, 320)	5192	concatenate_3[0][0]
conv9 (Conv3D)	(None, 8, 64, 320, 320)	1736	conv3d_12[0][0]
conv3d_13 (Conv3D)	(None, 1, 64, 320, 320)	9	conv9[0][0]
Total params: 1,471,489			
Trainable params: 1,471,489			
Non-trainable params: 0			

**Table 15** Network parameters for Input Size 384x384x48, model reduction 4

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1, 48, 384, 384)	0	
conv3d (Conv3D)	(None, 8, 48, 384, 384)	224	input_1[0][0]
conv1 (Conv3D)	(None, 8, 48, 384, 384)	1736	conv3d[0][0]
max_pooling3d (MaxPooling3D)	(None, 8, 24, 192, 192)	0	conv1[0][0]
conv3d_1 (Conv3D)	(None, 16, 24, 192, 192)	3472	max_pooling3d[0][0]
conv3d_2 (Conv3D)	(None, 16, 24, 192, 192)	6928	conv3d_1[0][0]
max_pooling3d_1 (MaxPooling3D)	(None, 16, 12, 96, 96)	0	conv3d_2[0][0]
conv3d_3 (Conv3D)	(None, 32, 12, 96, 96)	13856	max_pooling3d_1[0][0]
conv3 (Conv3D)	(None, 32, 12, 96, 96)	27680	conv3d_3[0][0]
max_pooling3d_2 (MaxPooling3D)	(None, 32, 6, 48, 48)	0	conv3[0][0]
conv3d_4 (Conv3D)	(None, 64, 6, 48, 48)	55360	max_pooling3d_2[0][0]
conv3d_5 (Conv3D)	(None, 64, 6, 48, 48)	110656	conv3d_4[0][0]
max_pooling3d_3 (MaxPooling3D)	(None, 64, 3, 24, 24)	0	conv3d_5[0][0]
conv3d_6 (Conv3D)	(None, 128, 3, 24, 24)	221312	max_pooling3d_3[0][0]
conv5 (Conv3D)	(None, 128, 3, 24, 24)	442496	conv3d_6[0][0]
up_sampling3d (UpSampling3D)	(None, 128, 6, 48, 48)	0	conv5[0][0]
concatenate (Concatenate)	(None, 192, 6, 48, 48)	0	up_sampling3d[0][0] conv3d_5[0][0]
conv3d_7 (Conv3D)	(None, 64, 6, 48, 48)	331840	concatenate[0][0]
conv3d_8 (Conv3D)	(None, 64, 6, 48, 48)	110656	conv3d_7[0][0]
up_sampling3d_1 (UpSampling3D)	(None, 64, 12, 96, 96)	0	conv3d_8[0][0]
concatenate_1 (Concatenate)	(None, 96, 12, 96, 96)	0	up_sampling3d_1[0][0] conv3[0][0]
conv3d_9 (Conv3D)	(None, 32, 12, 96, 96)	82976	concatenate_1[0][0]
conv7 (Conv3D)	(None, 32, 12, 96, 96)	27680	conv3d_9[0][0]
up_sampling3d_2 (UpSampling3D)	(None, 32, 24, 192, 192)	0	conv7[0][0]
concatenate_2 (Concatenate)	(None, 48, 24, 192, 192)	0	up_sampling3d_2[0][0] conv3d_2[0][0]
conv3d_10 (Conv3D)	(None, 16, 24, 192, 192)	20752	concatenate_2[0][0]
conv3d_11 (Conv3D)	(None, 16, 24, 192, 192)	6928	conv3d_10[0][0]
up_sampling3d_3 (UpSampling3D)	(None, 16, 48, 384, 384)	0	conv3d_11[0][0]
concatenate_3 (Concatenate)	(None, 24, 48, 384, 384)	0	up_sampling3d_3[0][0] conv1[0][0]
conv3d_12 (Conv3D)	(None, 8, 48, 384, 384)	5192	concatenate_3[0][0]
conv9 (Conv3D)	(None, 8, 48, 384, 384)	1736	conv3d_12[0][0]
conv3d_13 (Conv3D)	(None, 1, 48, 384, 384)	9	conv9[0][0]
Total params: 1,471,489			
Trainable params: 1,471,489			
Non-trainable params: 0			

### 3.5.2 3D U-NET Python dependencies

The python source code provided was executed with the following dependencies

- tensorflow-gpu (version 1.10.0)
- keras
- jupyter
- matplotlib
- scikit-image

- scikit-learn
- vtk
- itk
- SimpleITK
- Batchgenerators
- BatchViewer

### 3.6. Discussion

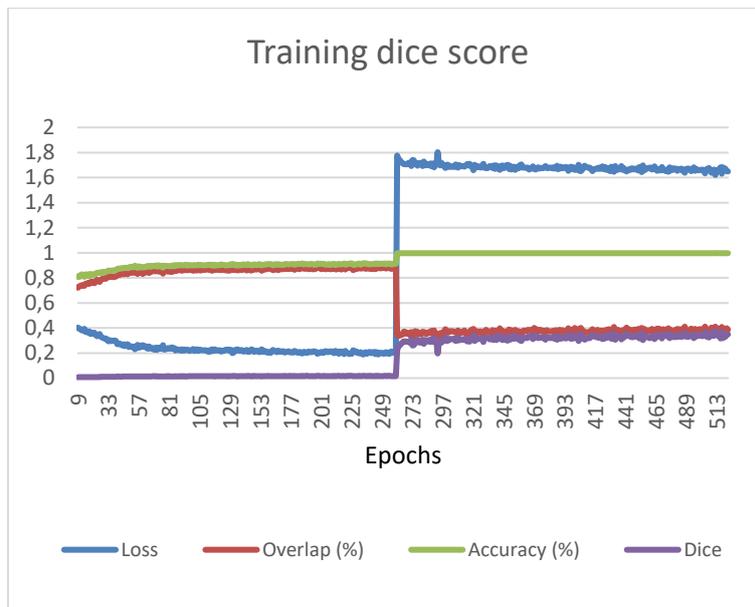
As the patch size was increased, the clutter created by smaller vessels is decreasing. This can be explained by the fact that patches without any ground truth are not avoided, since the probability of a long run of them is small. The network better learns to discern when the centerline feels relevant. However, this comes with the cost of also partly losing some of the overlap, but this effect could also be explained by losing the homogeneity of size for the Z dimension.

A strong correlation between the patch size and the variance of the training overlap, accuracy and loss can be seen. This is explained by the smaller number of training examples in an epoch and a greater distortion in the training examples due to augmenting.

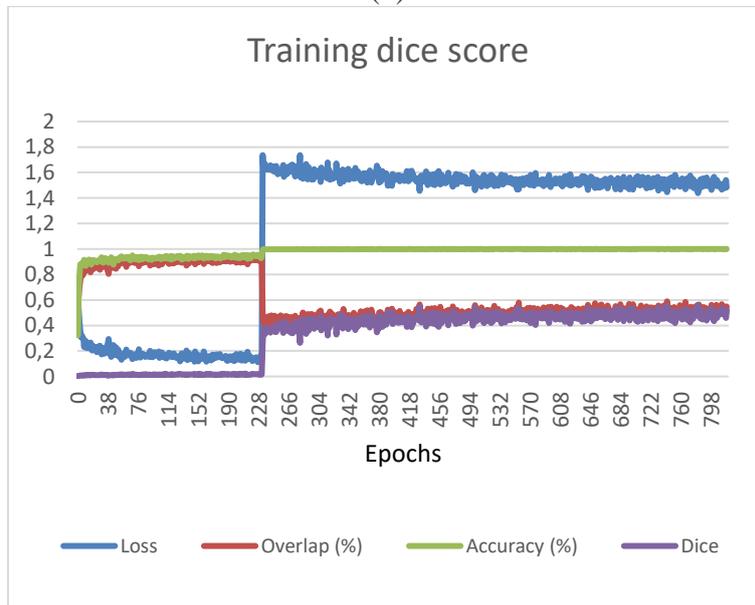
It is visible that, as the patch size increases, the number of visual artifacts decreases. We assume that this is due to the network having more contexts to discern which vessels are relevant. Unfortunately, due to constrain on the model size, adding more kernels for bigger patch sizes was not an option.

In the case of patch size 256x256x128, batch size 1, reduction 2, on epoch 393, the randomness in patch selection made the model fall into a local minimum of favoring the accuracy too much. In Figure 9 a huge drop in the overlap and a huge rise in the loss value can be seen. The accuracy jumps to almost 100% due to the class imbalance.

Starting from a random initialized model, with a dice coefficient loss function, which optimizes the tradeoff between precision and recall, the training converges quickly to predicting nothing. This happens because of two reasons: the huge class imbalance and the sparse annotation of the centerlines, where the model receives mixed signals about what represents a centerline and what doesn't. A different validation approach was used by starting to train the model with the proposed loss function, and when the model is sufficiently trained, switching to dice loss. The training now converges and an increase in the dice coefficient can be seen in Figure 49. Of course, this is not necessarily the objective, since the data is sparsely annotated, but this is another confirmation that the results are on the right track. We further present the results in term of dice coefficient after switching the training objective. A similar behavior is observed for a patch size 128x128x128 or for a patch size 256x256x128 (Figure 49).



(a)



(b)

**Figure 49** Loss function switch (a) Patch size 128x128x128 (b) Patch size 256x256x128

The loss function switch determines a rapid growth of the loss, marking a clear transition. The accuracy goes to almost 100%, because the model is now inclined to almost predict nothing. However, unlike the previous instability case, where the overlap fell to zero, it can now be seen that the overlap jumps to around 40% and starts to slowly increase with each epoch. For the first part of the training, the dice coefficient was close zero, since we had a lot of “false positives” in respect to the annotated ground truth. The second part of the training gives an increase in the dice coefficient, expected because of the loss function optimizes for that. Since the dice coefficient closely matches the overlap, it can be assumed that the model doesn’t predict too much outside of the high confidence zone. A simple dice function would output a number between -1 and 1. We added 1 to the result to avoid displaying negative values.

Centerline detection seems to be challenging in very low contrast vessels having fuzzy walls.

### 3.7. Conclusions

We have provided an open-source (available at [105]) implementation of a 3D U-Net, suitable for segmenting the coronary artery centerline. We have built upon a 2D U-Net base implementation network, originally designed for retina vessel segmentation, and developed a 3D version capable of segmenting with voxel precision. We conceived a novel loss function designed to combat two simultaneous problems, usually related to volumetric medical segmentation: huge class imbalance and sparse annotation. Without this loss function, the convergence of the model was not guaranteed even with using augmented inputs. We have built a customizable tool to generate annotated ground truth for the Rotterdam Coronary Artery Centerline Extraction from reference files. We have integrated a 3D augmentation algorithm [8] with the dataset to combat the problem of very few training examples. Without integrating one, the model would not generalize the centerline extraction. We have demonstrated that the training convergence using a second loss functions starting with a network pretrained with another loss function. Without the pretraining, training directly with the second loss function would not converge.

As future improvements, the existing algorithm could benefit from using Transfer Learning which was previously applied for U-Net architectures in [106], or for medical image classification models [107]. Even if the training dataset is small for coronary artery centerlines, one can use other segmentation tasks to pre-train the model. Using deconvolution layers instead of up-sampling layers in the proposed U-Net architecture may improve the pixelwise accuracy of the segmentation. The results can be perfected by incorporating a wider training set by with data from multiple benchmarks in the same way as in [83], [85], [86]. It is expected that the results will greatly increase, since it is known that neural networks need plenty training examples. In [85] 220 out of 620 CTA volumes were used for training (the rest were used for testing). In [83] 6 datasets were used with 3 levels of quality and summing more than 10000 fully annotated CTA volumes. Other promising improvements:

- Use dilated convolution layers to compensate for the smaller input patch sizes.
- Compute the Hausdorff distance for tracked vessels.
- More design space exploration using newer models of video cards having more VRAM and more cores. The use of a professional video card can improve the training time at least 10-fold.

Cardiovascular diseases (CVDs), classified as a group of disorders, are the number 1 cause of death globally [108]. Automated extraction of information about the coronary artery is on step in the computer-aided diagnosis of CVDs. Based on the extracted centerlines, other methods can be employed for visualizations, artery segmentation, plaque detection, and stenosis evaluation. The proposed method can help improving the visualization, detection and segmentation of various features such as soft plaque or calcified plaque if included in a stenosis score processing pipeline.





## 4. Final conclusions

### 4.1 Conclusions

Vessel segmentation is a prerequisite in the automated pipeline of diagnosis for vascular related diseases [73]. In this work, the focus was set on automated coronary centerline extraction from Cardiac Computed Tomography Angiography data.

The output can be used to obtain new information and projections useful for diagnosis by:

- extracting the projection of the lumen (planar development along an artery).
- determining the degree of blockage of the vessel
- determining positive remodeling [69] (if the vessel has changed shape due to differences in dynamic pressure caused by injury to the vessel)
- simulating fluid dynamics [70] [71] for calculation of the fractional flow reserve (FFR)

Segmenting the centerlines of the coronary vessel tree is a difficult problem, which was broken down into smaller tasks.

The first task was feature extraction from images, a subject which was better understood by the work done towards information extraction. Validation was done using cross entropy as a method to quantify the ability of an algorithm to understand the information carried by the pixels in images. The work for this task was creating a new prediction mechanism inspired from ensemble models, which was then integrated with the state-of-the-art image compressor paq8px. The result was an improved on-line reinforcement learning predictor. The feature extraction was done in a single pass over the entire data. Several optimizations were implemented as part of the algorithm for improving on the compromise between the quality of prediction and the computational cost.

The pixelwise residual cost of image was plotted and the results were similar to an edge segmentation. As a result, the next task was to improve the prediction model by implementing a contour detector. To achieve this, the proposed contextual memory algorithm was further developed to incorporate supervised off-line learning. For injecting domain specific knowledge, a processing pipeline was implemented, which included consecrated edge detection algorithms such as Canny edge detector and Kirsch edge detector. The result was a framework which encapsulated contextual memory with single and multilayered architectures. The results were validated on the Berkeley edge detection benchmark where the output of the framework was compared with the user submitted semantic segmentation ground truth. The score obtained was higher for the multilayer architecture, where the nested layer captured some aspects of the semantic segmentation.

The sparse distribution and the thin segmentation of object contours are some of the challenges of centerline extraction. The third task was to develop a deep neural network able to capture in the nested layers the abstractions needed. The network architecture chosen falls in the class of Fully Convolutional Neural Networks (FCNNs). For applying this network type for coronary centerline extraction from CCTA volumetric images, several problems were solved.

The design of the network was started from a U-NET implementation which provided state-of-the-art results for retina vessel segmentation. The original network used 2D kernels and the proposed network used 3D kernels. The network was trained using the Rotterdam Coronary Centerline Extraction dataset. There are only 8 training volumes in the dataset and are sparsely annotated. This data is not enough to directly train the network. Given the nature of the segmentation, augmentation was possible. Full size volumetric inputs require too much processing power for training, and resizing was not feasible when working with centerlines. Patches were used to limit the amount of computation power and the memory needed. A novel loss function was implemented to allow training to converge.

Little to no source code was available for the papers found in the literature at the time of this writing, and I decided to make all the results of this thesis open source.

## 4.2 Personal Contributions

The personal contributions are split into three categories:

- Lossless image compression
- Image segmentation with edge detection
- Automated coronary centerline extraction from Cardiac Computed Tomography Angiography (CCTA)

### 4.2.1 Lossless image compression

The main contributions are as follows:

- Created an original ensemble algorithm, useful for the prediction of bit probabilities with no assumption on the context modeling. The main difference from the existing ensemble mixing algorithms is: contexts apply together and the prediction benefits from the synergy of the side predictions - the ensembles assume model independence. The freedom on the context modeling opens the option for the algorithm to be used in areas like one-dimensional data, such as text information or data sequences, two-dimensional data, such as images, and three-dimensional data, such as volumetric images or image slices which represent volumetric information.
- Integrated Contextual Memory using on-line reinforcement learning with PAQ8PX in an open source project, available at [40], and proved the advantage by testing it on 4 image compression test sets and obtaining compression ratio improvement across all of them.
- Proposed and implemented 3 types of memory layouts for the contextual memory, each with low level speed optimizations.
- Implementation of an open source tool for automatic computation of bits-per-pixel for images, available at [44].
- Implementation of an original tool for pixelwise compression cost visualization, useful for the design space exploration of image compressors.
- Provided a detailed description of the state-of-the-art compression program PAQ8PX from the point of view of grayscale image compression.

## 4.2.2 Image segmentation with edge detection

For edge detection, the following contributions were made:

- The Contextual Memory algorithm was extended with off-line learning and supervised training, and integrated it with an original open source edge detection framework, available at [63].
- To prove the versatility of the algorithm, two processing pipelines were designed and implemented, one using a single layer architecture, and the other a multilayered one.
- The results of the edge detection framework were validated on the Berkeley edge detection benchmark with promising results as compared with CPU-only algorithms.
- Several low level and parallelization optimizations were provided for the extended version of the algorithm.
- We present more validation metrics as compared to the Canny edge detection algorithm.

## 4.2.3 Automated coronary centerline extraction from CCTA

To summarize, the main contributions are:

- Implemented and made open-source (available at [105]) a 3D U-Net for segmenting the coronary artery centerline. The base implementation was created by extending from 2D to 3D a network originally designed for retina vessel segmentation.
- A novel loss function designed to combat two simultaneous problems, usually related to volumetric medical segmentation: huge class imbalance and sparse annotation.
- A customizable tool to generate annotated ground truth for the Rotterdam Coronary Artery Centerline Extraction from reference files.
- Integrated a 3D augmentation algorithm with the dataset to combat the problem of very few training samples.
- Demonstrated training convergence using a second loss functions starting with a network pretrained with another loss function. Without the pretraining, training directly with the second loss function would not converge.

## 4.3 Dissemination of the research results

The results of the work done during the PhD studies were published in a number of international scientific journals. More specifically, 3 research papers have been published as a first author, and key contributions were made to another one. Also, key contributions were made to a conference proceedings paper. Based on my medical image segmentation experience, I was invited as a speaker to the Brain IT- Brain Revealed: innovative Technologies in Neurosurgery Study, that was held between 26th-28th of August 2021 in Sibiu, presenting the topic “Medical Imaging Segmentation with Machine Learning” [10].

The papers are as follows:

- **Dorobanțiu, A.**; Brad, R. Improving Lossless Image Compression with Contextual Memory. *Applied Sciences* 2019, 9, 2681, doi: 10.3390/app9132681, **ISI JCR - Q2** with IF 2.679 (2020), WOS: 000477031900100
- **Dorobanțiu, A.**; Brad, R. A novel contextual memory algorithm for edge detection. *Pattern Analysis and Applications* 2019, doi: 10.1007/s10044-019-00808-0, **ISI JCR – Q3** with IF 2.58 (2020), WOS: 000528015400024
- Ocrean, V.; **Dorobanțiu, A.**; Brad, R. Deep Learning Architectures and Techniques for Multi-organ Segmentation. *IJACSA* 2021, 12, doi: 10.14569/IJACSA.2021.0120104, indexed **ISI**, WOS: 000621697400004
- **Dorobanțiu, A.**; Ocrean, V.; Brad, R. Coronary Centerline Extraction from CCTA Using 3D-UNet. *Future Internet* 2021, 13, 101, doi: 10.3390/fi13040101, indexed **ISI**, WOS: 000643047700001
- Precup, S.-A.; Gellert, A.; **Dorobanțiu, A.**; Zamfirescu, C.-B. Assembly Process Modeling Through Long Short-Term Memory. In *Recent Challenges in Intelligent Information and Database Systems; Communications in Computer and Information Science; Springer Singapore: Singapore, 2021; Vol. 1371, pp. 28–39* ISBN 9789811616846, doi: 10.1007/978-981-16-1685-3\_3

In this short period of time starting from 2019, the paper [2] has already been cited in high ranked journals by:

- Khalaf, W., Al Gburi, A. and Zaghar, D., 2019. Pre and Postprocessing for JPEG to Handle Large Monochrome Images. *Algorithms*, 12(12), p.255, doi: 10.3390/a12120255
- Liu, X., An, P., Chen, Y. and Huang, X., 2021. An improved lossless image compression algorithm based on Huffman coding. *Multimedia Tools and Applications*, pp.1-15, doi: 10.1007/s11042-021-11017-5

The citations emphasize the potential impact of this work.



# REFERENCES

1. Ogorean, V.; Dorobanțiu, A.; Brad, R. Deep Learning Architectures and Techniques for Multi-organ Segmentation. *IJACSA* **2021**, *12*, doi: 10.14569/IJACSA.2021.0120104.
2. Dorobanțiu, A.; Brad, R. Improving Lossless Image Compression with Contextual Memory. *Applied Sciences* **2019**, *9*, 2681, doi: 10.3390/app9132681.
3. Dorobanțiu, A.; Brad, R. A novel contextual memory algorithm for edge detection. *Pattern Analysis and Applications* **2019**, doi: 10.1007/s10044-019-00808-0.
4. Fram, J.R.; Deutsch, E.S. On the quantitative evaluation of edge detection schemes and their comparison with human performance. *IEEE Trans Comput* **1975**, *C-24*:6, doi: 10.1109/T-C.1975.224274.
5. Dorobanțiu, A.; Ogorean, V.; Brad, R. Coronary Centerline Extraction from CCTA Using 3D-UNet. *Future Internet* **2021**, *13*, 101, doi: 10.3390/fi13040101.
6. Ronneberger, O.; Fischer, P.; Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*; Navab, N., Hornegger, J., Wells, W.M., Frangi, A.F., Eds.; Lecture Notes in Computer Science; Springer International Publishing: Cham, 2015; Vol. 9351, pp. 234–241 ISBN 978-3-319-24573-7.
7. Rotterdam Coronary Artery Algorithm Evaluation Framework Available online: <http://coronary.bigr.nl/centerlines/> (accessed on Jun 11, 2020).
8. MIC-DKFZ/batchgenerators Available online: <https://github.com/MIC-DKFZ/batchgenerators> (accessed on Aug 29, 2020).
9. Precup, S.-A.; Gellert, A.; Dorobanțiu, A.; Zamfirescu, C.-B. Assembly Process Modeling Through Long Short-Term Memory. In *Recent Challenges in Intelligent Information and Database Systems*; Communications in Computer and Information Science; Springer Singapore: Singapore, 2021; Vol. 1371, pp. 28–39 ISBN 9789811616846.
10. BrainIt: Brain revealed | Brain revealed: innovative Technologies in Neurosurgery Study.
11. Chen, D.; Li, Y.; Zhang, H.; Gao, W. Invertible update-then-predict integer lifting wavelet for lossless image compression. *EURASIP Journal on Advances in Signal Processing* **2017**, *2017*, 8, doi: 10.1186/s13634-016-0443-y.
12. Khan, A.; Khan, A.; Khan, M.; Uzair, M. Lossless image compression: application of Bi-level Burrows Wheeler Compression Algorithm (BBWCA) to 2-D data. *Multimedia Tools and Applications* **2017**, *76*, 12391–12416, doi: 10.1007/s11042-016-3629-2.
13. Feng, W.; Hu, C.; Wang, Y.; Zhang, J.; Yan, H. A Novel Hierarchical Coding Progressive Transmission Method for WMSN Wildlife Images. *Sensors* **2019**, *19*, 946, doi: 10.3390/s19040946.
14. Schiopu, I.; Munteanu, A. Residual-error prediction based on deep learning for lossless image compression. *Electronics Letters* **2018**, *54*, 1032–1034, doi: 10.1049/el.2018.0889.
15. Hosseini, S.M.; Naghsh-Nilchi, A.-R. Medical ultrasound image compression using contextual vector quantization. *Computers in Biology and Medicine* **2012**, *42*, 743–750, doi: 10.1016/j.compbiomed.2012.04.006.
16. Eben Sophia, P.; Anitha, J. Contextual Medical Image Compression using Normalized Wavelet-Transform Coefficients and Prediction. *IETE Journal of Research* **2017**, *63*, 671–683, doi: 10.1080/03772063.2017.1309998.
17. Borusyak, A.V.; Vasin, Yu.G. Development of an algorithm for adaptive compression of indexed images using contextual simulation. *Pattern Recognition and Image Analysis* **2016**, *26*, 4–8, doi: 10.1134/S1054661816010041.

18. Strutz, T. Context-Based Predictor Blending for Lossless Color Image Compression. *IEEE Transactions on Circuits and Systems for Video Technology* **2016**, *26*, 687–695, doi: 10.1109/TCSVT.2015.2416611.
19. Knezovic, J.; Kovac, M.; Mlinaric, H. Classification and Blending Prediction for Lossless Image Compression. In Proceedings of the MELECON 2006 - 2006 IEEE Mediterranean Electrotechnical Conference; IEEE: Benalmadena, Spain, 2006; pp. 486–489, doi: 10.1109/MELCON.2006.1653144.
20. Strizic, L.; Knezovic, J. Optimization of lossless image compression method for GPGPU. In Proceedings of the 2016 18th Mediterranean Electrotechnical Conference (MELECON); IEEE: Lemesos, Cyprus, 2016; pp. 1–6, doi: 10.1109/MELCON.2016.7495398.
21. Weinlich, A.; Amon, P.; Hutter, A.; Kaup, A. Probability Distribution Estimation for Autoregressive Pixel-Predictive Image Coding. *IEEE Transactions on Image Processing* **2016**, *25*, 1382–1395, doi: 10.1109/TIP.2016.2522339.
22. Biadgie, Y.; Kim, M.; Sohn, K.-A. Multi-resolution Lossless Image Compression for Progressive Transmission and Multiple Decoding Using an Enhanced Edge Adaptive Hierarchical Interpolation. *KSII Transactions on Internet and Information Systems* **2017**, *11*, 6017–6037, doi: 10.3837/tiis.2017.12.018.
23. Biadgie, Y. Edge Adaptive Hierarchical Interpolation for Lossless and Progressive Image Transmission. *KSII Transactions on Internet and Information Systems* **2011**, *5*, 2068–2086, doi: 10.3837/tiis.2011.11.011.
24. Song, X.; Huang, Q.; Chang, S.; He, J.; Wang, H. Lossless medical image compression using geometry-adaptive partitioning and least square-based prediction. *Medical & Biological Engineering & Computing* **2018**, *56*, 957–966, doi: 10.1007/s11517-017-1741-8.
25. Lucas, L.F.R.; Rodrigues, N.M.M.; da Silva Cruz, L.A.; de Faria, S.M.M. Lossless Compression of Medical Images Using 3-D Predictors. *IEEE Transactions on Medical Imaging* **2017**, *36*, 2250–2260, doi: 10.1109/TMI.2017.2714640.
26. Shen, H.; Jiang, Z.; Pan, W. Efficient Lossless Compression of Multitemporal Hyperspectral Image Data. *Journal of Imaging* **2018**, *4*, 142, doi: 10.3390/jimaging4120142.
27. Consultative Committee for Space Data Systems CCSDS RECOMMENDED STANDARD FOR IMAGE DATA COMPRESSION 2017.
28. Knoll, B.; Freitas, N. de A Machine Learning Perspective on Predictive Coding with PAQ8. In Proceedings of the 2012 Data Compression Conference; IEEE: Snowbird, UT, USA, 2012; pp. 377–386, doi: 10.1109/DCC.2012.44.
29. Mahoney, M.V. Adaptive Weighing of Context Models for Lossless Data Compression. 6.
30. Data Compression Explained Available online: [http://mattmahoney.net/dc/dce.html#Section\\_43](http://mattmahoney.net/dc/dce.html#Section_43) (accessed on May 11, 2019).
31. paq8px thread Available online: <https://encode.ru/threads/342-paq8px> (accessed on May 11, 2019).
32. Chartier, M. *MCM file compressor*. <https://github.com/mathieuchartier/mcm>; 2019;
33. Veness, J.; Lattimore, T.; Bhoopchand, A.; Grabska-Barwinska, A.; Mattern, C.; Toth, P. Online Learning with Gated Linear Networks. *arXiv:1712.01897 [cs, math]* **2017**.
34. Mattern, C. Mixing Strategies in Data Compression. In Proceedings of the 2012 Data Compression Conference; IEEE: Snowbird, UT, USA, 2012; pp. 337–346, doi: 10.1109/DCC.2012.40.
35. Mattern, C. Linear and Geometric Mixtures - Analysis. In Proceedings of the 2013 Data Compression Conference; IEEE: Snowbird, UT, 2013; pp. 301–310, doi: 10.1109/DCC.2013.38.

36. Mattern, C. On Statistical Data Compression. PhD Thesis, Technische Universität Ilmenau, Germany, 2016.
37. Fowler–Noll–Vo hash functions Available online: <http://www.isthe.com/chongo/tech/comp/fnv/index.html> (accessed on May 11, 2019).
38. Arbelaez, P.; Maire, M.; Fowlkes, C.; Malik, J. Contour detection and hierarchical image segmentation. *IEEE TPAMI* **2011**, *33*, doi: 10.1109/TPAMI.2010.161.
39. Alexandru Dorobanțiu - GitHub Available online: <https://github.com/AlexDorobantiu> (accessed on May 11, 2019).
40. Dorobanțiu, A. Paq8px167ContextualMemory Available online: <https://github.com/AlexDorobantiu/Paq8px167ContextualMemory> (accessed on May 13, 2019).
41. Image Repository of the University of Waterloo Available online: <http://links.uwaterloo.ca/Repository.html> (accessed on May 11, 2019).
42. S. Garg. The New Test Images - Image Compression Benchmark Available online: [http://imagecompression.info/test\\_images/](http://imagecompression.info/test_images/) (accessed on May 11, 2019).
43. Squeeze Chart • Lossless Data Compression Benchmarks Available online: <http://www.squeezechart.com/> (accessed on May 11, 2019).
44. Dorobanțiu, A. *Compute Bits Per Pixel for compressed images.* <https://github.com/AlexDorobantiu/BppEvaluator>; 2019;
45. 7-cpu Available online: <https://www.7-cpu.com/Utils.html> (accessed on Jun 13, 2019).
46. Mahoney, M. The ZPAQ Open Standard Format for Highly Compressed Data - Level 2. 23.
47. Aiazzi, B.; Alparone, L.; Baronti, S. Context modeling for near-lossless image coding. *IEEE Signal Processing Letters* **2002**, *9*, 77–80, doi: 10.1109/97.995822.
48. Gaonkar, B.; Hovda, D.; Martin, N. Deep learning in the small sample size setting: cascaded feed forward neural networks for medical image segmentation. *Proc SPIE* **2016**, 9785.
49. Milletari F, Navab N, Ahmadi S (2016) V-Net: fully convolutional neural networks for volumetric medical image segmentation. In: IEEE fourth international conference on 3D vision, pp 565–571.
50. Dollár, P.; Zitnick, L.C. Fast edge detection using structured forests. *IEEE Trans Pattern Anal Mach Intell* **2015**, *37*, doi: 10.1109/TPAMI.2014.2377715.
51. Xie, S.; Tu, Z. Holistically-nested edge detection. *Proc IEEE Int J Comput Vis* **2017**, *125*, doi: 10.1007/s11263-017-1004-z.
52. Liu Y, Lew MS (2016) Learning relaxed deep supervision for better edge detection. In: IEEE conference on computer vision and pattern recognition (CVPR), pp 231–240.
53. Liu Y, Cheng MM et al (2019) Richer convolutional features for edge detection. In: IEEE transactions on pattern analysis and machine intelligence; <http://mftp.mmcheng.net/Papers/19PamiEdge.pdf>. Accessed 05 Nov 2018.
54. Fu F, Wang C et al (2018) An improved adaptive edge detection algorithm based on Canny. In: Proceedings of SPIE 10827 icOPEN. Accessed 24 Jul 2018.
55. Guadaa, C.; Edwin, Z. A novel edge detection algorithm based on a hierarchical graph-partition approach. *J Intell Fuzzy Syst* **2018**, *34*, doi: 10.3233/JIFS-171218.
56. Minka T (2017) A comparison of numerical optimizers for logistic regression. <https://tminka.github.io/papers/logreg/minka-logreg.pdf>. Accessed 05 Feb 2017.
57. Naftaly, U.; Intrator, N.; Horn, D. Optimal ensemble averaging of neural networks. *Netw Comput Neural Syst* **1999**, *8*.
58. Liu, Y.; Yao, X. Ensemble learning via negative correlation. *Neural Netw* **1999**, *12*, doi: 10.1016/S0893-6080(99)00073-8.
59. Long, P.M.; Servedio, R.A. Random classification noise defeats all convex potential boosters. *Mach Learn* **2010**, *78*, doi: 10.1007/s10994-009-5165-z.
60. <http://www.burtleburtle.net/bob/hash/doobs.html>. Accessed 05 Feb 2017.
61. <http://www.isthe.com/chongo/tech/comp/fnv/index.html>. Accessed 05 Feb 2017.

62. <http://www.byronknoll.com/cmix.html>. Accessed 05 Nov 2018.
63. Dorobanțiu, A. *Contextual Memory Edge Detection* <https://github.com/AlexDorobantiu/ContextualMemoryEdgeDetection>; 2018;
64. Dollar, P. Structured Edge Detection Toolbox <https://github.com/pdollar/edges> Available online: <https://github.com/pdollar/edges> (accessed on Sep 27, 2021).
65. Wang Y, Zhao X et al (2018) Deep crisp boundaries. From boundaries to higher-level tasks. arXiv preprint arXiv:1801.02439.
66. Xu D, Ouyang W et al (2017) Learning deep structured multi-scale features using attention-gated CRFs for contour prediction. In: Advances in neural information processing system, pp 3961–3970.
67. Yu Z, Feng C et al (2017) CASENet: deep category-aware semantic edge detection. In: IEEE conference on computer vision and pattern recognition (CVPR), pp 21–26.
68. Yang J, Price B et al (2016) Object contour detection with a fully convolutional encoder-decoder network. In: IEEE conference on computer vision and pattern recognition (CVPR), pp 193–202.
69. Galal, H.; Rashid, T.; Alghonaimy, W.; Kamal, D. Detection of positively remodeled coronary artery lesions by multislice CT and its impact on cardiovascular future events. *Egypt Heart J* **2019**, *71*, 26, doi: 10.1186/s43044-019-0029-8.
70. Pantos, I.; Katritsis, D. Fractional Flow Reserve Derived from Coronary Imaging and Computational Fluid Dynamics. *Interventional Cardiology Review* **2014**, *9*, 145, doi: 10.15420/icr.2014.9.3.145.
71. Zhao, Y.; Ping, J.; Yu, X.; Wu, R.; Sun, C.; Zhang, M. Fractional flow reserve-based 4D hemodynamic simulation of time-resolved blood flow in left anterior descending coronary artery. *Clinical Biomechanics* **2019**, *70*, 164–169, doi: 10.1016/j.clinbiomech.2019.09.003.
72. Williams, L.H.; Drew, T. What do we know about volumetric medical image interpretation?: a review of the basic science and medical image perception literatures. *Cogn. Research* **2019**, *4*, 21, doi: 10.1186/s41235-019-0171-6.
73. Zhao, F.; Chen, Y.; Hou, Y.; He, X. Segmentation of blood vessels using rule-based and machine-learning-based methods: a review. *Multimedia Systems* **2019**, *25*, 109–118, doi: 10.1007/s00530-017-0580-7.
74. Chen, X.; Fu, Y.; Lin, J.; Ji, Y.; Fang, Y.; Wu, J. Coronary Artery Disease Detection by Machine Learning with Coronary Bifurcation Features. *Applied Sciences* **2020**, *10*, 7656, doi: 10.3390/app10217656.
75. Danilov, A.; Pryamonosov, R.; Yurova, A. Image Segmentation for Cardiovascular Biomedical Applications at Different Scales. *Computation* **2016**, *4*, 35, doi: 10.3390/computation4030035.
76. Guo, Z.; Bai, J.; Lu, Y.; Wang, X.; Cao, K.; Song, Q.; Sonka, M.; Yin, Y. DeepCenterline: a Multi-task Fully Convolutional Network for Centerline Extraction. *arXiv:1903.10481 [cs]* **2019**.
77. Bates, R.; Irving, B.; Markelc, B.; Kaeppler, J.; Muschel, R.; Grau, V.; Schnabel, J.A. Extracting 3D Vascular Structures from Microscopy Images using Convolutional Recurrent Networks. *arXiv:1705.09597 [cs]* **2017**.
78. Han, D.; Shim, H.; Jeon, B.; Jang, Y.; Hong, Y.; Jung, S.; Ha, S.; Chang, H.-J. Automatic Coronary Artery Segmentation Using Active Search for Branches and Seemingly Disconnected Vessel Segments from Coronary CT Angiography. *PLoS ONE* **2016**, *11*, e0156837, doi: 10.1371/journal.pone.0156837.
79. Liu, L.; Xu, J.; Liu, Z. Automatic segmentation of coronary lumen based on minimum path and image fusion from cardiac computed tomography images. *Cluster Comput* **2019**, *22*, 1559–1568, doi: 10.1007/s10586-018-2548-6.
80. Kong, B.; Wang, X.; Bai, J.; Lu, Y.; Gao, F.; Cao, K.; Xia, J.; Song, Q.; Yin, Y. Learning tree-structured representation for 3D coronary artery segmentation. *Computerized*

- Medical Imaging and Graphics* **2020**, *80*, 101688, doi: 10.1016/j.compmedimag.2019.101688.
81. Lv, T.; Yang, G.; Zhang, Y.; Yang, J.; Chen, Y.; Shu, H.; Luo, L. Vessel segmentation using centerline constrained level set method. *Multimed Tools Appl* **2019**, *78*, 17051–17075, doi: 10.1007/s11042-018-7087-x.
  82. Gao, Z.; Liu, X.; Qi, S.; Wu, W.; Hau, W.K.; Zhang, H. Automatic segmentation of coronary tree in CT angiography images. *Int J Adapt Control Signal Process* **2019**, *33*, 1239–1247, doi: 10.1002/acs.2762.
  83. Sheng, X.; Fan, T.; Jin, X.; Jin, J.; Chen, Z.; Zheng, G.; Lu, M.; Zhu, Z. Extraction Method of Coronary Artery Blood Vessel Centerline in CT Coronary Angiography. *IEEE Access* **2019**, *7*, 170690–170702, doi: 10.1109/ACCESS.2019.2955710.
  84. Cui, H.; Xia, Y. Automatic Coronary Centerline Extraction Using Gradient Vector Flow Field and Fast Marching Method From CT Images. *IEEE Access* **2018**, *6*, 41816–41826, doi: 10.1109/ACCESS.2018.2859786.
  85. Guo, Z.; Bai, J.; Lu, Y.; Wang, X.; Cao, K.; Song, Q.; Sonka, M.; Yin, Y. DeepCenterline: A Multi-task Fully Convolutional Network for Centerline Extraction. In *Information Processing in Medical Imaging*; Chung, A.C.S., Gee, J.C., Yushkevich, P.A., Bao, S., Eds.; Lecture Notes in Computer Science; Springer International Publishing: Cham, 2019; Vol. 11492, pp. 441–453 ISBN 978-3-030-20350-4.
  86. Wolterink, J.M.; van Hamersvelt, R.W.; Viergever, M.A.; Leiner, T.; Išgum, I. Coronary artery centerline extraction in cardiac CT angiography using a CNN-based orientation classifier. *Medical Image Analysis* **2019**, *51*, 46–60, doi: 10.1016/j.media.2018.10.005.
  87. Sainath, T.N.; Vinyals, O.; Senior, A.; Sak, H. Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks. In Proceedings of the 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP); IEEE: South Brisbane, Queensland, Australia, 2015; pp. 4580–4584, doi: 10.1109/ICASSP.2015.7178838.
  88. Salahuddin, Z.; Lenga, M.; Nickisch, H. Multi-Resolution 3D Convolutional Neural Networks for Automatic Coronary Centerline Extraction in Cardiac CT Angiography Scans. *arXiv:2010.00925 [cs, eess]* **2020**.
  89. He, J.; Pan, C.; Yang, C.; Zhang, M.; Wang, Y.; Zhou, X.; Yu, Y. Learning Hybrid Representations for Automatic 3D Vessel Centerline Extraction. In *Medical Image Computing and Computer Assisted Intervention – MICCAI 2020*; Martel, A.L., Abolmaesumi, P., Stoyanov, D., Mateus, D., Zuluaga, M.A., Zhou, S.K., Racoceanu, D., Joskowicz, L., Eds.; Lecture Notes in Computer Science; Springer International Publishing: Cham, 2020; Vol. 12266, pp. 24–34 ISBN 978-3-030-59724-5.
  90. Irving, B.; Hutton, C.; Dennis, A.; Vikal, S.; Mavar, M.; Kelly, M.; Brady, J.M. Deep Quantitative Liver Segmentation and Vessel Exclusion to Assist in Liver Assessment. In *Medical Image Understanding and Analysis*; Valdés Hernández, M., González-Castro, V., Eds.; Communications in Computer and Information Science; Springer International Publishing: Cham, 2017; Vol. 723, pp. 663–673 ISBN 978-3-319-60963-8.
  91. Jin, Q.; Meng, Z.; Pham, T.D.; Chen, Q.; Wei, L.; Su, R. DUNet: A deformable network for retinal vessel segmentation. *Knowledge-Based Systems* **2019**, *178*, 149–162, doi: 10.1016/j.knosys.2019.04.025.
  92. Zhuang, J. LadderNet: Multi-path networks based on U-Net for medical image segmentation. *arXiv:1810.07810 [cs, eess]* **2019**.
  93. Guo, C.; Szemenyei, M.; Yi, Y.; Wang, W.; Chen, B.; Fan, C. SA-UNet: Spatial Attention U-Net for Retinal Vessel Segmentation. *arXiv:2004.03696 [cs, eess]* **2020**.
  94. Çiçek, Ö.; Abdulkadir, A.; Lienkamp, S.S.; Brox, T.; Ronneberger, O. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. *arXiv:1606.06650 [cs]* **2016**.
  95. Huang, C.; Han, H.; Yao, Q.; Zhu, S.; Zhou, S.K. 3D U2-Net: A 3D Universal U-Net for Multi-domain Medical Image Segmentation. In Proceedings of the Medical Image

- Computing and Computer Assisted Intervention – MICCAI 2019; Shen, D., Liu, T., Peters, T.M., Staib, L.H., Essert, C., Zhou, S., Yap, P.-T., Khan, A., Eds.; Springer International Publishing: Cham, 2019; Vol. 11765, pp. 291–299, doi: 10.1007/978-3-030-32245-8\_33.
96. Lin, T.-Y.; Goyal, P.; Girshick, R.; He, K.; Dollár, P. Focal Loss for Dense Object Detection. *arXiv:1708.02002 [cs]* **2018**.
  97. Cui, Y.; Jia, M.; Lin, T.-Y.; Song, Y.; Belongie, S. Class-Balanced Loss Based on Effective Number of Samples. *arXiv:1901.05555 [cs]* **2019**.
  98. Schaap, M.; Metz, C.T.; van Walsum, T.; van der Giessen, A.G.; Weustink, A.C.; Mollet, N.R.; Bauer, C.; Bogunović, H.; Castro, C.; Deng, X. Standardized evaluation methodology and reference database for evaluating coronary artery centerline extraction algorithms. *Medical Image Analysis* **2009**, *13*, 701–714, doi: 10.1016/j.media.2009.06.003.
  99. Rotterdam Coronary Artery Challenge Categories Available online: <http://coronary.bigr.nl/centerlines/about.php> (accessed on Jun 11, 2020).
  100. 3D Slicer Available online: <https://www.slicer.org/> (accessed on Jun 11, 2020).
  101. GitHub Slicer Available online: <https://github.com/Slicer/Slicer> (accessed on Jun 11, 2020).
  102. Slicer/SlicerJupyter Available online: <https://github.com/Slicer/SlicerJupyter> (accessed on Jun 16, 2020).
  103. Jonker, P.P. Morphological Operations on 3D and 4D Images: From Shape Primitive Detection to Skeletonization. In *Discrete Geometry for Computer Imagery*; Borgefors, G., Nyström, I., di Baja, G.S., Eds.; Lecture Notes in Computer Science; Springer Berlin Heidelberg: Berlin, Heidelberg, 2000; Vol. 1953, pp. 371–391 ISBN 978-3-540-41396-7.
  104. Frid-Adar, M.; Diamant, I.; Klang, E.; Amitai, M.; Goldberger, J.; Greenspan, H. GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification. *Neurocomputing* **2018**, *321*, 321–331, doi: 10.1016/j.neucom.2018.09.013.
  105. AlexDorobantiu/CoronaryCenterlineUnet Available online: <https://github.com/AlexDorobantiu/CoronaryCenterlineUnet> (accessed on Aug 29, 2020).
  106. Igloukov, V.; Shvets, A. TeraNet: U-Net with VGG11 Encoder Pre-Trained on ImageNet for Image Segmentation. *arXiv:1801.05746 [cs]* **2018**.
  107. Manzo, M.; Pellino, S. Bucket of Deep Transfer Learning Features and Classification Models for Melanoma Detection. *J. Imaging* **2020**, *6*, 129, doi: 10.3390/jimaging6120129.
  108. WHO - Cardiovascular diseases (CVDs) fact sheet Available online: [https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-\(cvds\)](https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds)) (accessed on Sep 24, 2020).
-

