Departamentul Automatică şi Informatică Industrială
Facultatea Automatică şi Calculatoare
Universitatea POLITEHNICA din Bucureşti

LUCRARE DE DIPLOMĂ

Sistem biomedical de segmentare a leziunilor hepatice bazat pe tehnici avansate de prelucrare a imaginilor prin rețele neuronale convoluționale

Biomedical system for liver lesion segmentation based on advanced convolutional neural network architectures

Coordonator
Prof.univ.Dr.Ing. Dan Popescu

Absolvent
Stanciulescu Andrei

2022

Contents

1 Introduction

The liver is the largest organ of the human body, responsible for multiple essential functions. It accounts for around 2% of an adult's body weight and is an unique organ due to its dual blood supply: the portal vein and the hepatic artery. Bile production plays a vital role in the excretion of substances which cannot be processed by the kidneys, while also aiding in the absorption and digestion of lipids. The liver's metabolic functions consist in fat-soluble vitamin storage and also the decomposition of xenobiotics, such as various toxins and carcinogens. Lastly, the liver is also responsible for the processing of bilirubin, which is one of the products of hemoglobin breakdown [Kal22]. Given its essential role in body functions, liver cancer is the third most common cause of death from cancer, despite being the sixth most common type of cancer worldwide [WCF21]. Following the advancements in medicine and image processing, medical image segmentation methods are getting a lot of attention. The most widely used method for liver disease diagnosis is computed tomography (CT).

In practice, liver lesions are observed and diagnosed by comparing patterns which appear when using contrast imaging techniques. The liver has soft-tissue density, which is similar to the tumors which affect it, so plain CT images lack the contrast necessary to distinguish between areas of interest. To combat this setback, an intravenous contrast substance is injected in the patient. The substance reaches the tumor in a higher concentration, so the contrast is enhanced in the arterial phase. However, this behavior is typically observed in hyper vascular tumors. Other types of lesions can present a different pattern, which can be identified by the radiologist by observing the Native, Arterial and Portal Venous phase [Bar13].

Several computer-based systems are proposed by researchers to aid radiologists in interpreting CT images. Computer Aided Detection systems are used to analyze and report areas in the images that may be of interest. On the other hand, Computer Aided Diagnosis systems not only detect the regions of interest in CT images, but also present the likelihood that the area is a specific type of lesion, for example malign or benign. The main objective of both types of systems is to reduce the number of false negatives caused by human error. CAD systems are developed to search for and mark similar features in digital images that the radiologist would look for when performing an investigation [Cas05]. The development of these systems requires the integration of multiple disciplines, such as image processing, pattern matching and artificial intelligence. The workflow necessary for hepatic CAD consists of the following stages: pre-processing, segmentation, feature extraction and selection and classification. Depending on the method used for the feature extraction step, these computer aided diagnosis tools can be classified as conventional or deep learning based [Vai20].

The area of image semantic segmentation using neural networks has seen a lot of research interest in the last decade, as the hardware necessary to train and evaluate complex models became more common. As automated learning systems in general, and artificial neural networks in particular, become increasingly more popular and easier to implement, these types of solutions make their way into all domains, industrial, medical and that of entertainment and business. In the case of liver diseases, automated diagnostic techniques have seen a lot of interest, which is reflected by the steady increase of research papers on this subject. An important aspect which drives this initiative is the presence of public datasets, such as LiTS17 and 3Dircad, which contain anonymous CT examinations along with reference segmentation masks provided by certified radiologists. This allows independent researchers who do not have any affiliation with medical institutions to perform research and advance the technology in this field.

This project will first provide an overview of the existing systems used in biomedical imaging, specifically liver and tumor segmentation methods, which achieved high scores in competitions and provide a starting point for developing a new system. Then, I will provide detailed descriptions of the methods used to develop a segmentation model, starting from convolutional neural networks used for segmentation which are converted to feature extractors. Image processing techniques which can increase segmentation quality will also be explored, along with the performance increase which can be brought by the use of a decision fusion system. Details on the model training regime will be presented, along with all the details of the implementation which would make this work reproducible, such as test/training splits and the method for computing the performance metrics which will appear further in this work. Then, I will make a comparative analysis between this system’s results and other relevant implementations presented in scientific literature.

All the code and implementations used for this project will be made publicly available under an open-source license, such that others can verify, reproduce and extend this work.

2 Related Works

The goal of this chapter is to provide an overview of the current technologies used in medical image segmentation and diagnosis. While completely automated solutions have been getting a lot of attention recently, there are also semi-automatic methods for segmentation.

One study on automated segmentation techniques explored tumor identification using patch analysis [Li.W15]. Before feeding the images to a convolutional neural network, a gaussian filter was applied to remove part of the noise, then the images were down sampled by a factor of 2. To classify each pixel, a patch of 17x17 pixels centered around it was passed through a CNN composed of 5 convolutional layers, two fully connected layers which resulted in a classification output of 'tumor' and 'non-tumor'. Their method achieved a Dice Similarity Coefficient (DSC) of 80.06% ± 1.63%, which placed it above traditional machine learning techniques, namely AdaBoost, Random Forests and support vector machine (SVM). However, the method presented encountered difficulties when segmenting tumors of heterogeneous intensity or which had fuzzy borders.

An improvement to CNNs used in medical image analysis was brought by the development of U-Net. Frequently, in medical applications it is not enough to classify an image, instead a pixel-wise classification is necessary. Patch analysis methods work in some cases, but they have their drawbacks. For example, using a patch size that is too big requires additional max pooling layers, which in turn causes the localization accuracy to decrease. Reducing the patch size increases localization accuracy but in turn decreases contextual information. U-Net networks work as an encoder-decoder sequence. The standard contracting structure of a CNN is supplemented with upsampling and convolutional layers to bring the image to a dimensionality similar to the input, while giving a classification probability to each resulting pixel. To aid in upsampling, high resolution features are transferred from the contracting layers. To demonstrate the capabilities of this network, it was trained and tested for three different segmentation tasks: segmentation of neural structures from electron microscopy images, and two different cell tracking problems with images acquired by light microscopy. Because a small set of training data was used, data augmentation techniques such as shifting, rotation and elastic deformation were employed to prevent overfitting and to teach the network the desired invariance needed for medical image segmentation. Across all three tests, the U-Net achieved better segmentation metrics than other networks commonly used in image segmentation [Ron15].

Zhe Liu et al. proposes an improved U-Net architecture which addresses the issue of low contrast between the liver and surrounding tissue, image noise and difference of organ shape [Liu19]. This method is based on using an improved U-Net for initial liver segmentation, then employs a graph cut algorithm to refine the segmentation from the probability maps. Graph cut is

a semi-supervised segmentation algorithm which requires the user to mark a few pixels as background and a few as foreground. The image pixels as viewed as graph vertices and the segmentation process is based on separating the initial graph into two subgraphs based on an energy minimization function. The modification brought to the neural network consists in the method of transferring feature maps from the encoder to the decoder. In the original U-Net, the features are transferred after two convolution layers and a ReLU activation layer, while in the proposed method the features are transferred directly from the pooling layer, resulting in a lower loss of image features. Also, the proposed, improved network is deeper, featuring 3 additional downsampling blocks. The experiments evaluating this network were performed on **LiTS17**, a liver segmentation competition dataset containing 131 sequences. The proposed method was compared to other popular segmentation neural networks, scoring a DSC value of 95.05%. the performance was better that FCN-8s (89.36%) and U-Net (82.6%), coming second to H-DenseUNet (96.5%).

While almost all top-ranking segmentation algorithms in the MICCAI 2017 competition employ either U-Net or VGG-Net, Lu Meng et al. [Men20] propose a three-dimensional dual path multi-scale convolutional neural network (TDP-CNN), followed by conditional random fields (CRF) which refine the segmentation result. This method takes advantage of the fact that CT images are usually taken as a series of slices through the body, thus providing 3D contextual information which is not used by 2D CNNs. Like in other methods, the data was preprocessed by applying a Gaussian filter, normalization, and subsampling. To overcome the computational difficulties introduced by the usage of 3D convolution and 3D feature maps, the initial volume is broken down into small segments and only a few segments are input in the CNN at a time. Segments of different sizes but centered on the same voxel are used, hence the term “multiscale”. The network is structured into two different paths. Segments with smaller image size but higher resolution are processed by the local path and trained to capture features such as texture and contour. The bigger segments go through the global path, providing contextual and background information. The feature maps resulted from these paths are combined and used together in the final classification layers. The network classifies image regions into three labels: liver, liver tumor and background. To correct mis-segmentation points, this implementation used fully connected CRFs. This method achieved a 96.5% DICE index for liver segmentation and 68.9% Dice for tumor segmentation, indicating it can accurately segment the liver and liver tumors from 3D abdominal CT images.

Another variation of the traditional U-Net architecture employs attention aware and residual structures [Wan21]. The standard blocks which perform the encoding and decoding are wrapped in residual feed-forward structures, which address the problem of vanishing gradients in network training. Squeeze and Excitation blocks are introduced before max pooling layers. These blocks suppress irrelevant areas of the image while highlighting the significant ones. SE blocks consist of a global pooling layer followed by a FC layer, a ReLU activation function, another FC layer,

then a sigmoid activation function. The output from the block has the same dimensions as the input, so a pooling layer is still necessary. Another difference from the original U-Net is how the transition from the encoder to the decoder is handled. Instead of using a standard convolution layer, the solution employs Atrous Spatial Pyramid Pooling. Atrous convolution is a process which dilates the convolution kernel such that the elements from the original image are no longer neighboring pixels. By using multiple dilation factors for the atrous convolution, the context of the image can be captured at different scales. This enables the resulting feature map to contain richer semantic information. In this implementation, passing through the ASPP block doubles the number of features. The proposed system was evaluated using the **LiTS17** training dataset. Several preprocessing steps were applied to the CT images: removal of irrelevant images, clipping the Housenfield intensity, histogram equalization and normalizing the pixel values to [0, 1]. For the training phases, the dataset was augmented with scaling, rotations, flipping and B-spline deformations. On this dataset, the network scored a 95.71% DSC, outperforming Attention U-Net, U-Net and FCN. The optimality of the solution was also checked by starting with a classic U-Net and applying the proposed changes step by step and evaluating each resulting network. These experiments show that the changes bring a cumulative improvement to the starting architecture.

Combining residual attention-aware U-Nets with 3D convolutional neural networks, Qiangguo Jin et al. [Jin20] proposes a novel architecture called RA-Unet, which can segment the liver and tumors from 3D volumetric images. Their implementation obtained competitive results when tested on the MICCAI 2017 **LiTS17** dataset. Overall, the architecture consists of 3 individual stages. First, using a 2D residual attention-aware U-Net, a bounding box is placed around the liver in each slice of the volume to reduce redundant information for the next step. Next, the volume and the coarse bounding box are sent to a 3D RA-Unet which will define a more precise volume of interest which contains only the liver. The volume is then sent to a second 3D RA-Unet which extracts the tumor regions. Both the 2D and 3D versions of the RA-Unet share the same architecture, the only difference being the shape of the feature maps. Starting from the traditional U-Net structure, each downsampling and upsampling convolutional block is wrapped into a residual structure to combat the problem of vanishing gradients. In the original U-Net implementation, feature maps from the encoder part are transferred directly to the corresponding layer in the decoder part, the only processing being border cropping to ensure size compatibility. However, in the proposed architecture, the feature maps are passed through a residual attention aware structure before being transferred to the decoder. This module is divided between two branches: the trunk branch processes the original features through a series of convolutional layers while the soft mask branch selects features which are identical between feature maps and suppresses irrelevant noise. The soft mask branch consists of two encoder-decoder blocks followed by two convolutional layers and a Sigmoid normalization layer. For training, the loss function was based on the Dice coefficient. When tested on the **LiTS17** dataset,

the system scored a DSC of 96.1% for liver segmentation and 59.5% for tumor segmentation, the latter being higher than proposed 2D segmentation methods. To prove the generalization of the solution, it was also tested on the 3DIRCADb dataset, where is scored even higher performance indices.

While previously presented methods rely on semantic segmentation of only one set of images with the purpose of highlighting the liver area and potential lesion areas, Koichiro Yasaka et al. [Yas18] takes a different approach and proposes a system for tumor diagnosis which relies on the differentiation of liver masses in dynamic contrast agent-enhanced CT. This study used liver images acquired over three phases: noncontrast-agent enhanced, arterial and delayed, to classify liver masses in five categories: classic hepatocellular carcinomas, malignant tumors other than HCCs, indeterminate lesions or mass-like lesions, hemangiomas, and cysts. This study focuses on classification of images of liver masses, rather than determining the exact location of a lesion. The proposed network consists in a structure of two convolutional layers and one max-pool layer, structure which is repeated three times. This is followed by three fully connected layers, which in the end output the probability for each type of liver mass. The patient images were captured in the DICOM format, then only the relevant slices were selected from each patient. To remove redundant information, a windowing function was applied to remove HU values outside the liver tissue range. The window ranges were adjusted to consider the differences between standard images and contrast-enhanced images. After the initial image preprocessing, data augmentation methods were used to make the system robust to disturbances such as enlarging, rotation, parallel shift as well as different levels of noise. Using these variations on the original images, 52 different datasets were obtained from the original data. Some images were cropped to include only the liver parenchyma and the liver masses, while others also contained the surrounding organs. The proposed architecture was used to train five different models which used images from different phases of the CT scanning process: triphasic, arterial and delayed phase, unenhanced, arterial phase, and delayed phase. In the testing phase, the accuracy of each model in classifying liver masses in five different categories was evaluated. The triphasic and the arterial/delayed models had the best performance compared to the other three. The first model performed the best on the test data, with an accuracy of 84%, and 95% on the training data. The arterial/delayed model performed slightly worse on the test data and slightly better on the training data, but the differences are statistically insignificant. However, the other methods performed significantly worse, thus showing the importance of comparing images at different contrast phases when performing the diagnosis.

Most of the implementations presented above work on segmenting 2D images independently. However, this approach does not take advantage of the contextual information provided by processing multiple sequential slices of the same volume. Abdominal CT scans are usually acquired as a 3D image, with slices taken at a known distance, so volumetric information is available, but training a 3D CNN comes with high GPU memory requirements and longer

training time. Xiao Han proposes a 2.5D network architecture, which can take advantage of the contextual 3D information without processing the whole volume in one pass [Han17]. To segment each slice, several adjacent axial slices are selected and processed together. The additional slices only provide contextual information to the network, while the center slice in the stack is the one being segmented. Overall, the architecture is based on U-Net, which was adapted for 3D images to process the image stacks, and to which were added residual structures in both the encoder and decoder. For this implementation, stacks of 5 adjacent slices were passed through two similar sequential networks. The first provides a coarse segmentation of the liver, while the second one refines the liver segmentation and discriminates between healthy liver tissue and lesions. The images were preprocessed only by thresholding to eliminate pixels with values outside the liver intensity range. The resulting segmentation was post-processed by thresholding the confidence value of the segmentations to values above 0.8. This implementation achieved an 67% Dice score on the LiTS17 database when considering both liver and tumor segmentation. Another study performed by Wardhana et al. investigates the effects of using different numbers of slices as input, the effects of contrast enhancement techniques and whether adding an additional encoder-decoder structure brings significant improvements [War21]. The slice arrangement study evaluated network performance when trained with one, three, five seven or nine slices and the best performance was achieved when using three slices (one center slice with one adjacent slice on either side). Increasing the number of slices led to a degradation of the network's performance due to overfitting the training data. The study on the effects of contrast enhancement techniques on network performance shows that applying histogram equalization will reduce segmentation accuracy, since datasets will be affected differently by this technique based on the distribution of pixel values. The method with the highest Dice score was bilateral filtering, which uses a Gaussian filter to remove the noise, but it can also remove small lesions. Taking this effect into account, the authors decided to use only a basic contrast enhancement technique based on value windowing and normalization. Lastly, the study investigates whether any improvements can be made by adding a second encoder-decoder structure to the network, such that the output of the first structure feeds into the second one. The liver segmentation Dice score is similar between the two networks, but the extended network has a lower overall score when it comes to tumor segmentation. As a result of higher sensitivity, the extended network can identify smaller tumors which are not observed by the simple network, while it also misidentifies healthy tissue for tumors and exaggerates tumor size.

As it was discussed in the previous paragraphs, segmentation using volumes or stacked slices can prove to be beneficial for liver and tumor segmentation, compared to processing individual slices, as it gives contextual information and provides better tissue localization. However, 3D convolutions come with a greater cost of memory consumption and computation time. Besides the 2.5D methods presented above, there are solutions which use different size networks to process intra-slice and inter-slice features, then combine the results to create a final

segmentation. One such method is proposed by Chi et al. [Chi21], which uses a multi-branch UNet-like network to create three-dimensional liver and tumor segmentations. The proposed XNet architecture consists of 3 UNets which share a down-sampling and an up-sampling path. The main tumor feature extraction network uses a Dense UNet encoder, which provides multi-scale, deep feature analysis using a low amount of parameters, which is also shared with the liver mask segmentation network. The decoder path combines shortcut connections from the Dense encoder as well as the filtered liver region network. The latter is a 4 stage feature pyramidal network which is employed to generate liver features at 4 different scales, as well as to restrain tumor segmentation to the liver area, thus eliminating erroneous segmentations which appear outside the liver tissue. This XNet structure generates intra-slice liver and tumor feature maps which encode the differences and relations between liver lesions and healthy tissue. To gain inter-slice contextual information, a modified 3D UNet processes the filtered liver volume generated by the previous network. Based on the assumption that contextual information of a small area in a slice is decided mostly by a similarly sized neighborhood in the adjacent slices, all convolutional layers are sized to 3×3×n. Thus, it can detect changes along the z axis while prohibiting the influence of pixels which are outside the specified neighborhood. This modification also greatly reduce the computation strain, as it performs a low amount of mathematical operations for each volume. To generate a final segmentation, the intra-slice and inter-slice features are concatenated, then passed through a fully convolutional layer, which reduces the features dimensionality to a volumetric lesion segmentation. The training regime used also uses a modified loss function, which penalizes region errors and contour errors in a different manner for better training result. The system was evaluated on the LiTS dataset, where it achieved state of the art performance: 97.1% global Dice for liver segmentation and 84.3% global Dice for tumor segmentation.

Based on the results described in the papers above, I will choose a set of CNN models suitable for the described application, which will be responsible for segmenting, sequentially, the liver tissue and lesion tissue.

3 Materials And Methods

The proposed method for segmentation involves using multiple classification networks for feature extraction, then using a dedicated segmentation head to perform the segmentation from the resulting feature maps. This works in a similar way to an UNet, where the image classification network acts as an encoder and the segmentation head as a decoder, reducing the dimensionality of the tensor while upscaling it to the original size. Then, the segmentations obtained from these networks will be used as input for a weighted decision system, which will fuse the inputs into a single segmentation. This process will be performed separately, first to segment the liver tissue, then to determine the lesion areas, using only the liver as input. As presented in [Gru19], sequential segmentation approaches yield significantly better results compared to multiclass segmentation in one step.

3.1 Image dataset

The reference dataset for liver and liver lesion segmentation from CT images is **LiTS17**. This dataset was created in collaboration with seven hospitals and research institutions and was reviewed by three independent radiologists. It includes 131 CT examinations with several types of tumor contrast levels, abnormalities in tissue size and a varying number of lesions. Another 70 undisclosed volumes are used for model evaluation in the ongoing MICCAI competition. Since the examinations are provided by several different sources, they present a mix of different CT scanners and acquisition protocols. The image quality, resolution and slice thickness also vary between volumes. Some of the examinations also present imaging artifacts, such as metal artifacts, which can also be present in other clinical data [Bil19].

3.2 Convolutional Neural Networks Architecture

In recent years, convolutional neural networks (CNN) have seen immense development. This type of network has become the most representative in the broader field of deep learning. The evolution of CNNs is based first of all on the development of artificial neural networks, which are based on mathematical models of neurons, called perceptrons. The drawback of networks based on single-layer perceptrons is that they cannot efficiently work with linearily inseparable problems [Li.Z21]. When working with data structured in a grid, such as digital images, convolutional neural networks are more widely used, since they can learn hierarchies of spatial features of various sizes. CNNs are traditionally composed of 3 main types of layers: convolution, pooling and fully connected [Yam18].

3.2.1 Convolutional layers

The convolutional layer is the fundamental building block of any CNN architecture. Its role is to perform feature extraction and to increase or decrease the number of feature maps. Usually, this type of layer consists of two operations: convolution and activation function. The convolution is a linear mathematical operation which is performed by sliding a kernel over the image and taking the element-wise product of the overlapping regions, then summing the values and placing them in the resulting matrix. This procedure can be repeated to obtain an arbitrary number of feature maps from an input tensor. The two key hyperparameters which define this operation is the kernel size, which usually ranges from 3×3 to 7×7, depending on the scale at which the image is analyzed, and the number of such kernels, which defines the number of resulting feature maps. Since applying this operation reduces the x and y size of the resulting image compared to the input, zero-padding is usually used, such that after the operator is applied, the output size is the same as the input. However, the convolution operation is not enough to efficiently train a network, since it is a linear operation, by itself it can only model linear functions, while most processes and features present nonlinearities. This is why convolutional operations are almost always paired with a nonlinear activation function. Nonlinear functions such as the sigmoid or hyperbolic tangent provide a more accurate representation of the biological neuron firing function, the most widely used is the Rectified Linear Unit (ReLU), which is defined as [Yam18]:

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

The learnable parameters of this layer are the convolution kernel weights and, even thought it is less widely used, the kernel bias.

3.2.2 Pooling layers

Pooling layers are used as a downsampling mechanism, as they reduce the in-plane dimensionality of a tensor, while keeping the number of feature maps the same. This operation has a dual purpose: firstly, it introduces translational invariance to small shifts in the image plane, while also decreasing the number of operations for the downstream layers. This operation has no trainable parameters, since the filter size, stride and padding are considered layer hyperparameters and are defined by the network architecture. Max pooling is the most widely used pooling function. It analyzes in-plane patches of the initial tensor and keeps only the maximum value in each patch. The dimensionality reduction is determined by the filter size, padding and patch overlap. For example, a 3x3 filter without padding and overlap will reduce the tensor width and height by a factor of 3. Global average pooling performs a most extreme downsampling operation, as it reduces the dimension of each feature map to only one value, which is the in-plane average. This is usually used only once in the network, before the fully connected layer to drastically reduce the number of features before the final classification step. The advantage of this operation over using more fully connected layers is that it enforces

correspondence between feature maps and classes and eliminates all variations introduced by feature or image translations [Lin13].

3.2.3 Batch Normalization

One complication which comes with training Deep Neural Networks is the fact that the distribution of intermediate feature maps changes with each epoch, as previous layer parameters change. This phenomenon is referred to as internal covariate shift and due to it, networks require lower learning rates to avoid divergence, which in turn slows down the training process. The use of batch normalization layers (also referred to as BatchNorm) aims to eliminate this effect by performing normalization across each training minibatch independently by fixing the mean and variance of layer inputs [Iof15]. However, the normalization step by itself might hurt network performance as it could limit the inputs to a linear section of a nonlinear activation function, such as the sigmoid, which is linear when inputs approach 0. To avoid this issue, two learnable layer parameters are added, which perform a translation and a shift on the values with mean of 0 and unit variance. The operation performed by the BatchNorm layer can be described as [Iof15]:

$$\mu \leftarrow \frac{1}{n}\sum_{1}^{n} x_i \qquad \textit{minibatch mean} \tag{2}$$

$$\sigma^2 \leftarrow \frac{1}{n}\sum_{1}^{n} (x_i - \mu)^2 \qquad \textit{minibatch variance} \tag{3}$$

$$\hat{\boldsymbol{x}} \leftarrow \frac{\boldsymbol{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} \qquad \textit{minibatch normalization} \tag{4}$$

$$\boldsymbol{y} \leftarrow \gamma\, \hat{\boldsymbol{x}} + \beta \qquad \textit{scaling} \text{ and } \textit{shifting} \tag{5}$$

When using BatchNorm layers in the network architecture, convolution bias should not be used, since its effect is canceled when the mean becomes 0 and is replaced by the shifting parameter β. The introduction of this structures has the added effect of improving gradient flow through the network by reducing its dependence on the scale of the parameters and allows for increased learning rates while maintaining convergence.

3.2.4 Dropout layers

The purpose of dropout layers is to prevent co-adaptation in large networks, which happens when the network becomes too reliant on a specific path and leads to overfitting [Sri14]. Dropout is a technique which addresses this issue by randomly eliminating layer connections during training. The most widely used dropout rate is p = 0.5, which means that each connection in that specific layer has a 50% probability of being eliminated for a specific batch. This leads to multiple, thinner variations of the base network which are evaluated and adjusted during training.

During testing, it would be impractical to evaluate and average the results of these random network variations, so the averaging process is simulated by scaling the layer weights in the base network by p.

3.2.5 Fully connected layers

The fully connected layers (FC) are used exclusively in classification CNNs, and they usually represent the last few layers. Up to that point, the tensor processed by the network resembles an image, or at least low level features of an image. After the last convolutional or pooling layer, these values are flattened into a 1D array of values. In this type of layer, every input is connected to every output through a connection with trainable weight, a structure which resembles artificial neural networks based on perceptrons. A series of FC layers maps the input space to the number of classes by successively reducing the number of features. The last fully connected layer produces the values which will be napped to class probabilities through a sigmoid function in the case of binary classification, or a softmax function in case of multiclass classification.

3.3 Training Convolutional Neural Networks

The training of artificial neural networks in general refers to finding a combination of network parameters which minimizes a given cost function. Like many other minimization problems, it relies on computing the cost function gradients with respect to the objective function's parameters, then uses these gradients to adjust said parameters [Goo16]. The first step in the training process is to compute the function gradients. This can be very computationally expensive, depending on the method employed.

The most popular algorithm used in artificial neural network training is back-propagation, which is the idea of computing derivatives by propagating information through the network. ANNs and specifically CNNs perform a set of mathematical operations in a specific sequence, so for a certain network architecture a computational graph can be created, where each node represents a single value or a tensor variable. If a variable is used to compute another, a directed edge is drawn from the first to the second variable node. The back-propagation algorithm starts from the output node, and calculates the gradient of the cost function with respect to the node variables. Then the algorithm processes all the nodes which have directed edges connected to the output node, and computes the gradients of the output node function with respect to the current node variable. Since information propagation through the network on a certain path can be described as a function which is composed of all the functions performed by the intermediate layers, the chain rule of calculus can be used to compute the gradients of the cost function with respect to any network parameter after a recursive back-propagation pass. Given the gradient of the cost function with respect to the activations of layer l a^l , the gradient corresponding to the previous layer activations a^{l-1} can be computed using the following relation [Goo16]:

$$\nabla_{a^{l-1}} C = \left(\frac{\partial a^l}{\partial a^{l-1}} \right)^T \nabla_{a^l} C \tag{6}$$

where $\nabla_{a^l} C$ was computed at the previous back-propagation step, and the Jacobian matrix is easily computed since the function performed by layer l is known.

Gradient computation is the first part of the problem, now the network weights have to be adjusted accordingly to minimize the cost function. The most used methods are based on gradient descent, which is an algorithm which adjusts the function parameters taking into account the gradient magnitude and direction in an iterative process [Bot12]. At each step the parameters are adjusted in the following manner:

$$w_{i+1} \leftarrow w_i - \alpha \nabla_w C \tag{7}$$

where α is the assigned learning rate. For strictly convex problems, this method will provide an accurate approximation of the optimum. However, in other cases, this method can only converge in points of local minimum or saddle points, depending on learning rate and starting point. A drastic simplification to this algorithm is stochastic gradient descent (SGD), which is the method I used for network parameter optimization. Instead of taking into account all the training data, at each step the algorithm evaluates the objective function for one randomly picked sample and adjusts the weights accordingly. This greatly speeds up the process and, over a large amount of iterations, it presents a similar optimization behavior to classic GD.

3.4 Classifier Networks

For these experiments, I chose to use four popular image classification convolutional networks, namely: ResNet, ResNeXt, DenseNet and Inception. Overall, most image classification networks share the same structure, having multiple similar blocks, which differ only in the values of the hyperparameters, and inside the blocks, the individual layers follow the same structure, which represents the particularity of each network. Each block reduces the size of the input and generates a higher number of feature maps which represent lower-level details. To perform the classification task, the last feature maps are flattened and passed through a structure of fully connected linear layers, which in the end provides a prediction for the specific class the image belongs to. However, in this application, only the final feature maps are of interest, so only the convolutional part of the networks will be kept, and the output will be passed to the classifier head. The particularities of modifying and training the networks in this manner will be presented in the Implementation chapter.

3.4.1 ResNet152

The ResNet architecture was first proposed by He et al. in [He15]. They reformulated the learning functions to reference the layer inputs as opposed to being unreferenced functions. Their

research showed that this structure leads to networks which are easier to optimize and gain considerable accuracy from increasing the depth. Moreover, compared to VGG classification networks, ResNet can achieve 8 times the depth while still using fewer parameters. The proposed ResNet architecture is based on residual training blocks, which transfer the input of the block and add it to the output without changing it. This feedforward residual path addresses the problem of vanishing gradients and allows the network to benefit from increased depth. Starting with a plain convolutional network with 34 layers, residual connections were added every two layers. This primitive implementation of the ResNet achieved a significant decrease in the top-1 error while having the exact number of parameters as the plain network. It also showed increased accuracy gain from increasing the depth compared with a plain network. To address the higher training times that come with increased network depth, a bottleneck structure was proposed. This consists of 3 convolution layers: 1×1, 3×3, 1×1. The 1×1 convolutions are used to first reduce, then restore the dimensions of the input feature maps, leaving the 3×3 layer with smaller dimensions (Figure 1). Using this bottleneck structure, networks of various depths were designed. The chosen architecture, the 152-layer ResNet is built from 50 sequential bottleneck blocks grouped in 4 blocks, where each block doubles the number of channels processed by the bottleneck layer. Being designed to be trained and evaluated on the ImageNet dataset, the convolutional sequence of the network is followed by a fully connected linear layer with 1000 outputs, which represent the predictions for the 1000 classes in the dataset. However, since I am using this network as a feature extractor, the output of the last convolutional layer will be transferred to the DeepLab module. In the case of ResNet152, this output consists of 2048 8×8 feature maps for inputs of size 256×256. The following figure presents the structure of the bottleneck block used in the current ResNet implementation.

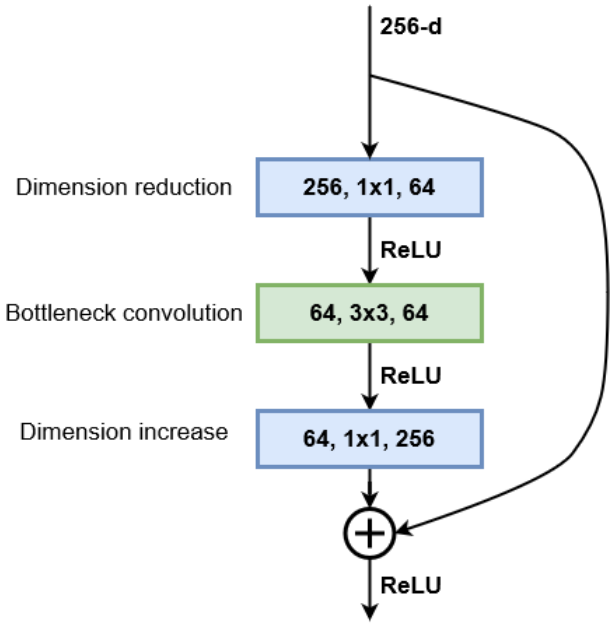


Figure 1 Structure of residual bottleneck building block (# in channels, filter size, # out channels)

3.4.2 ResNeXt101

Building on the standard ResNet structure, Xie et al. [Xie16] developed a modified version of this architecture which implements aggregated transformations. The general design follows the modularity of VGG and ResNets, stacking residual blocks with the same topology. Blocks processing the same number of feature maps have the same width and filter size, and when the spatial map is downsampled by a factor of 2, the block width is multiplied by the same factor. Taking inspiration from Inception networks, the ResNeXt blocks implement the *split-transform-aggregate* strategy. The bottleneck structure from the original ResNet is split into multiple paths where the 3×3 convolution layer operates on a significantly smaller number of channels (Figure 2). The number of distinct paths in a block is referred to as *cardinality* and is considered as an additional hyperparameter when creating the network structure. At the end of the block the outputs of all paths are summed together, and the block input is added via the residual connection. Their experiments showed that, while maintaining the same number of parameters, a ResNeXt50 network with a cardinality of 32 and bottleneck width 4 outperforms a ResNet50 model with bottleneck width 64, having a top-1 error 1.7 percentage points lower. Further examinations showed that increasing the cardinality of the aggregated transformation structures produces much better results than increasing the network depth or widening the bottleneck layer. The ResNeXt101 variant which will be used is built from 33 stacked bottleneck modules grouped in 4 blocks, which follow the same rule for increasing the number of channels as the ResNet variant presented above. Similarly, the convolutional layers end with a fully connected layer which computes the activation values for the 1000 classes of the ImageNet

dataset. This layer is of no interest for the implementation, since I will be using the network as a feature extractor, not a classifier.

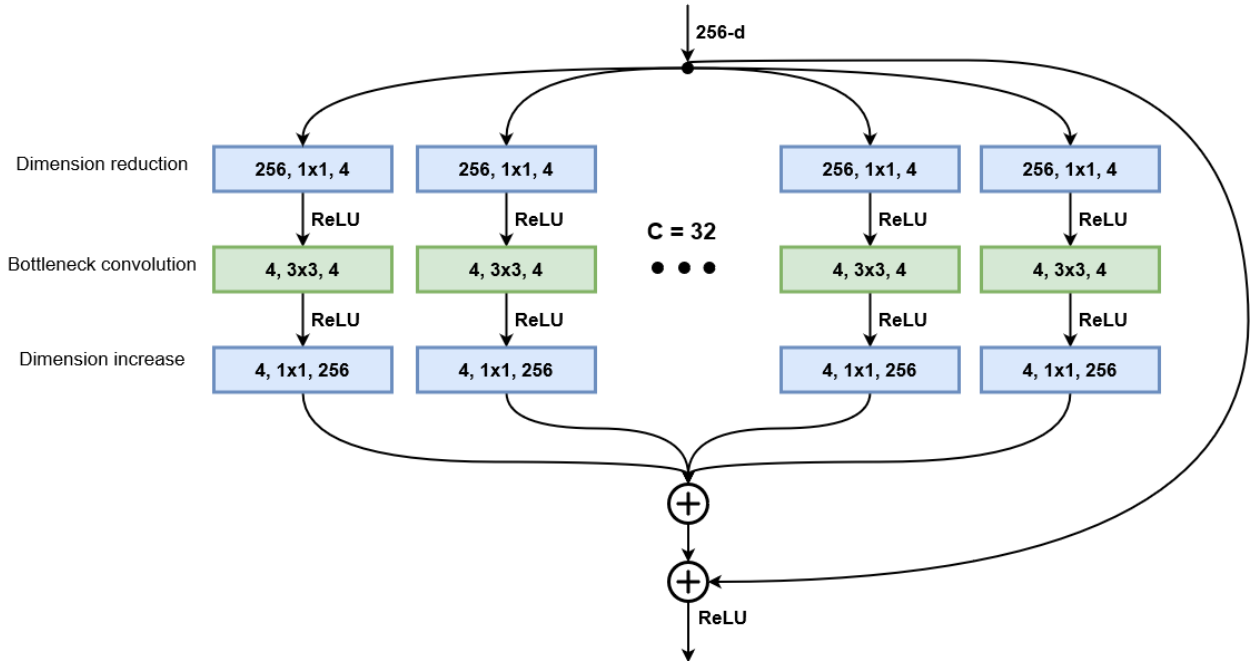


Figure 2 Structure of the split-transform aggregate block with cardinality 32 and bottleneck width 4 (# in channels, filter size, # out channels)

3.4.3 DenseNet201

Much like the other networks presented above, the DenseNet structure was designed to perform as an image classifier network. As described in [Hua16], this network architecture continues the philosophy of stacking convolutional layers with the same structure and different hyperparameters. However, instead of relying on network depth or width for increased performance, this family of CNNs exploits the potential of feature reusing. Building on the ResNet concept of feedforwarding the layer input its output, this implementation introduces direct connections from every layer to all subsequent layers in a feedforward fashion. In this case, the feedforward connection is done by concatenation instead of summation, such that each layer receives the output feature maps from all the layers before it. The composition function of a dense layer is composed of six consecutive operations: batch normalization, ReLU activation, 1×1 convolution, batch normalization, ReLU activation and a 3×3 convolution. The first three modules function as a bottleneck, while the next three are responsible for the actual image processing. Multiple layers are stacked in a dense block and share the same hyperparameters. The dense feedforward connections only happen inside the blocks, not between them since the size of the feature maps changes. Between blocks, a transitional layer composed of a batch normalization, 1×1 convolution followed by 2×2 average pooling is responsible for reducing the dimensions of the feature maps for the next block. The DenseNet201 variant is built from 4

dense blocks with a total of 98 dense layers. The output if the last convolutional layer is connected to a fully connected linear layer with 1000 class outputs. Their tests showed that DenseNets are more efficient in terms of parameters than ResNets. In terms of prediction accuracy on ImageNet, DenseNet201 is equivalent to ResNet101 while having less than half the number of trainable parameters. Another positive side-effect of more efficient usage of parameters is the resiliency to overfitting, the networks achieving improvements of around 30% on the C10 dataset when data augmentation is removed during training.

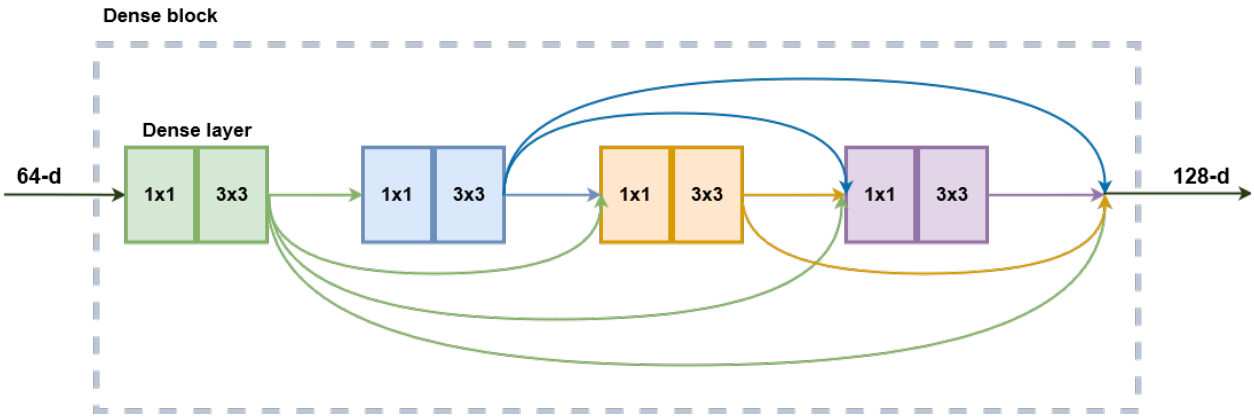


Figure 3 DenseNet building block with 4 dense layers. Each rectangle corresponds to a sequence of BatchNorm-ReLU-Convolution

3.4.4 InceptionV3

The creation of the Inception architecture, as presented in [Sze15], had the objective of scaling up networks in a way that the added computational complexity could be used as efficiently as possible. The computational cost of Inception networks is significantly lower than that of VGGNet or its successors with higher performance, however its structural complexity makes it more difficult to make changes to the network. For example, trying to increase the network capacity through doubling the filter bank size will lead to four times increase in computational cost and network parameters, but the performance increase might be insignificant. The paper presents some of the design principles that went into the design of the Inception networks. First, the design follows the principle of avoiding bottlenecks, especially in the initial stages of the network and those with extreme compression. Instead, it reduces the dimensions step by step when going deeper into the network. Also, higher dimensional representations are easier to process locally, the network training faster when the activations per tile are increased. However, the authors argue that spatial aggregations such as spread-out convolutions can be done on lower dimensional embeddings without significant performance loss since there is a string correlation between adjacent features. Regarding network hyperparameters, the Inception design balances the width and the depth of the network, increasing them in parallel to achieve optimal improvements. The original Inception architecture [Sze14] is built from multiple Inception blocks. Each block sends its inputs along 4 parallel paths, and the outputs of each path

are concatenated at the end of the block. In the naïve implementation, these 4 paths contain the following operations: 1×1 convolution, 3×3 convolution, 5×5 convolution and a 3×3 max pooling. The reasoning is that instead of choosing a single type of operation, the network could train the parameters for each operation as needed at various depths. However, this approach leads to immense computational costs. To address this issue, the original network adds a 1×1 convolution before the 3×3 and 5×5 convolutions to function as a bottleneck layer, thus leaving fewer dimensions for the complex convolutional layers. The new Inception design goes one step further to improve the efficiency by replacing the 5×5 convolution with 2 sequential 3×3 convolutions. Since the 5×5 convolution is 2.78 times more computationally expensive than the 3×3 variant, this approach proves to be more effective. The efficiency can be increased even further by employing spatial factorization into asymmetric convolutions. This way, a 3×3 convolution is split in a 3×1 followed by a 1×3 convolution. This two-layer solution is 33% cheaper in terms of computational complexity. This method was shown to not work well in the early layers but is effective when used for the factorization of $n \times n$ grids with n between 12 and 20. The figure below present the general structure of an Inception block with filter size $n \times n$. The last implemented change is to use the two asymmetrical convolutions in parallel, thus expanding the size of the filter bank. These changes are implemented gradually in the network. To reduce the grid size between Inception block groups, the last block of the group implements a stride of 2 in all parallel operations, thus halving the width and height of the feature maps. At the time of its creation, this network set a new state of the art in single crop evaluation on the ILSVRC 2012 with a 21.2% top-1 and 5.6% top-5 error.

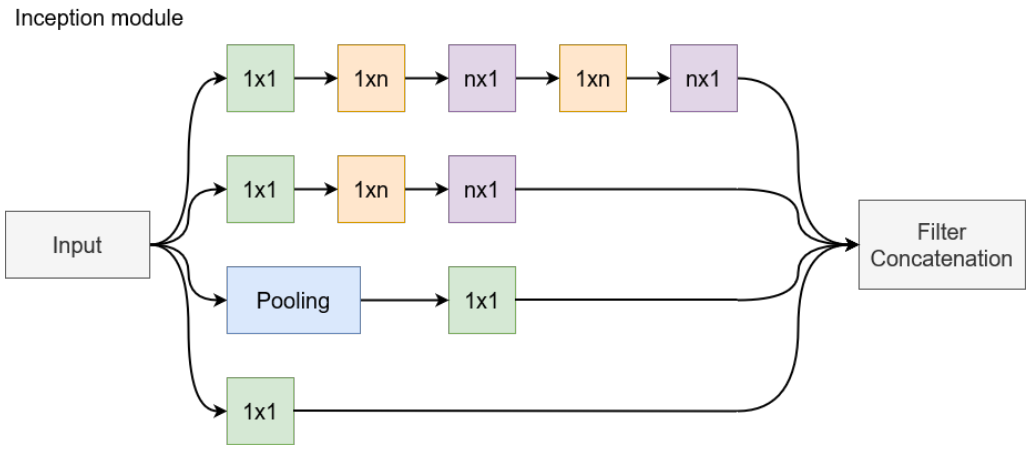


Figure 4 General structure of an inception module after the factorization of an NxN filter

3.5 Segmentation Head

Atrous convolution is a powerful method which can be used to adjust a filter's field of view, as well as to accurately control the resolution of features computed by a CNN. In the research paper "Rethinking Atrous Convolution for Semantic Image Classification", Chen et al

[Che17] present a design for solving semantic segmentation at multiple scales by using parallel and cascade atrous convolution modules and using image-level features to further boost performance. The DeepLabV3 network structure expands on the concepts of atrous convolution, which allows the network to train feature detection at different scales without downsampling the image. Modern Deep Convolutional Networks used for segmentation or classification usually decrease the input image size by a factor of 32 through multiple max pooling and striding layers. To restore the image to the original size, transposed convolutions are usually used. The motivation behind using atrous convolutions is that the image only needs to be decimated to a lesser extent, which allows us to control the features resolutions through the kernel dilation rate. This technique relies on upsampling convolution kernels at different rates by spacing out the kernel and inserting zeros between consecutive kernel values.

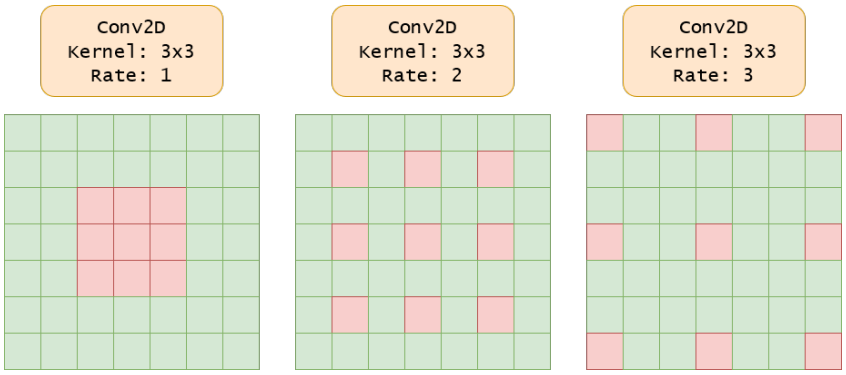


Figure 5 Atrous convolution example

Their experiments started from a standard ResNet structure. The last block was repeated 4 times and added in sequence to the network, each time increasing the dilation of the atrous convolution kernel. This method proved to be better than repeating the least layer without any modifications, however improvements were still possible. To allow all the different resolution filters to work on unaltered data, the additional filters were added in a parallel manner, using a concept called Atrous Spatial Pyramid Pooling. This structure produces features at multiple resolutions and concatenates them for the final segmentation layers. This method allows the network to achieve multiple effective field-of-views, which leads to the ability to encode multi-scale feature information. However, the dilation rate of the kernel should be carefully chosen in relation to the size of the feature maps. If the dilated extremities of the kernel fall outside the valid image space, then the convolution becomes effectively a 1×1 filter, which is not desirable. Along with the multi-resolution feature maps, image level features are added by resampling the input at the expected output size. This semantic segmentation network was trained and tested with a ResNet50 and ResNet101 backbone and performed better other state-of-the-art models on the PASCAL VOC 2012 dataset, achieving an 86.9% mIOU score. However, this segmentation method could be adapted to work with any image classification network with minor adjustments, effectively transforming it in a semantic segmentation network.

The actual semantic segmentation structure passes the input through 4 parallel paths, each with a convolutional layer employing a different atrous rate. The outputs are concatenated with the image level features and are then passed through another 2 Conv-BatchNorm-ReLU blocks. The first block reduces the number of feature maps through a 1×1 convolution and is followed by a Dropout layer with 0.5 probability, which randomly zeroes some of the elements of the input tensor, based on a Bernoulli distribution. The use of Dropout layers was proved to be an effective tool for network regularization during training and prevents neuron co-adaptation, as described in [Hin12]. In the end, a 1×1 filter reduces the number of feature maps from 256 to the number of classes, providing per-pixel classification. If necessary, the image is the bilinearly upsampled back to its original size.

This structure will be appended to the image classification networks described above to generate liver and lesion segmentations from the provided feature maps.

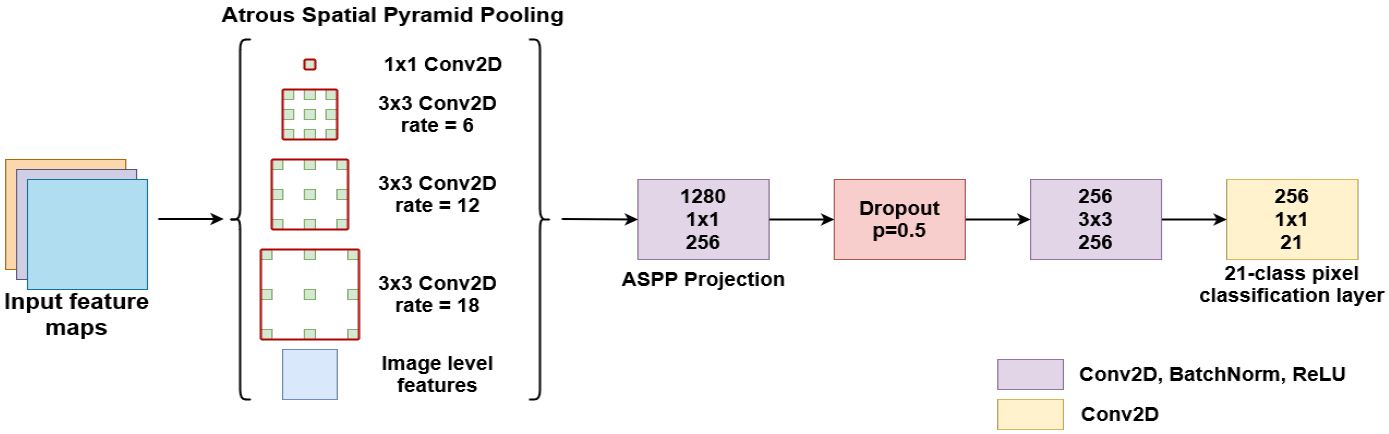


Figure 6 DeepLabV3 segmentation structure

3.6 Decision Fusion System

The proposed system, as described until now, produces four different segmentations for one input image passed through the four networks described above. These results will be fused into a single segmentation which should have improved accuracy metrics. To achieve this, I propose a weighted average system, where the weights can be individually adjusted for each network. The reasoning behind this approach is that each different network will learn different low-level features, such that the segmentation errors will appear in different places in the image for each network result. Applying a weighted average over these results may result in the errors from each one network will be corrected by the other three networks.

The input image is passed as input to each segmentation network and the results are computed. Then, each segmentation is multiplied by its corresponding weight, and they are all added together. In the end, the result is passed through a Sigmoid activation function and the values are rounded to the nearest integer to provide a binary segmentation mask.

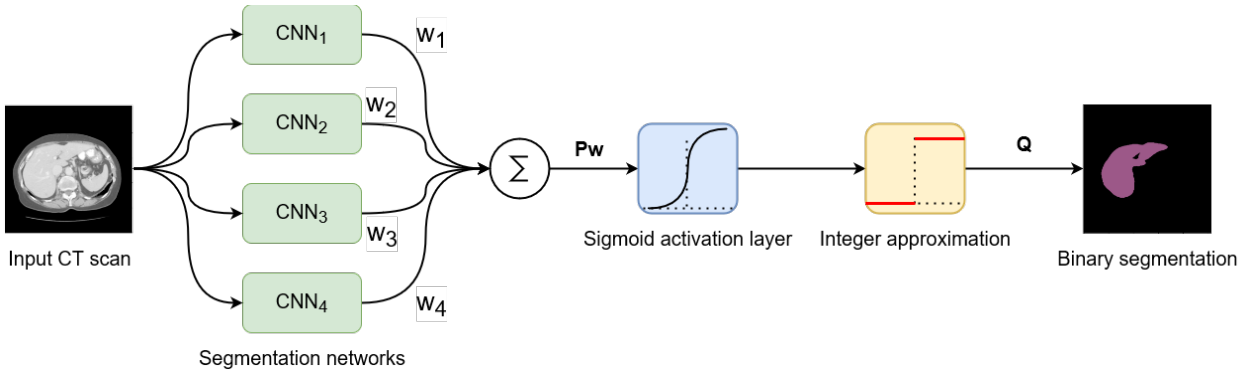


Figure 7 Modular representation of the decision fusion system

Suppose that P_1 through P_4 are the segmentations provided by the four segmentation networks and w_1 through w_4 their corresponding weights. Then, the fusion system's output Q can be described thus:

$$\begin{aligned} \boldsymbol{P} &= [P_1, P_2, P_3, P_4] \\ \boldsymbol{w} &= [w_1, w_2, w_3, w_4]^T \\ \sigma(x) &= \frac{1}{1+e^{-x}} \\ \boldsymbol{Q} &= round(\sigma(\boldsymbol{P}\boldsymbol{w})) \end{aligned} \tag{8}$$

The trivial choice of weights would be to give them the same value. However, since network performance is not the same across all networks, weighting based on performance could be a better idea, since better performing networks would have a higher influence on the final segmentation. My choice is to programmatically adjust the weights individually to minimize a cost function, in the same way linear neural networks are trained. The criterion used in the proposed method is binary cross entropy, so the following cost function to minimize is obtained, where G is the ground truth segmentation mask:

$$\min_{\boldsymbol{w}\in\mathbb{R}^4} \left| \frac{1}{N} \sum \left(G \log(\sigma(\boldsymbol{P}\boldsymbol{w})) + (1-G) \log(1-\sigma(\boldsymbol{P}\boldsymbol{w})) \right) \right| \tag{9}$$

The set of weights which minimizes the cost function can be found using iterative gradient descent algorithms, such as Stochastic Gradient Descent. The details of weight adjustment after the networks are trained will be presented in the implementation chapter.

3.7 Segmentation performance metrics

In the domain of biomedical image segmentation, the most widely used metrics to assess performance are the Dice Similarity Coefficient (DSC) and Jaccard index. In this application, I will evaluate the DSC on binary segmentation masks, separately for liver segmentation and for

lesion segmentation. This metric measures the degree of overlap between the ground truth and the evaluated system segmentation. Given two binary masks, the Dice coefficient is evaluated using the following formula:

$$DSC(\boldsymbol{A},\boldsymbol{B}) = \frac{2|\boldsymbol{A} \cap \boldsymbol{B}|}{|\boldsymbol{A}| + |\boldsymbol{B}|} \tag{10}$$

This formula produces a value in the range [0, 1], where 1 signifies a perfect overlap. False positives, as well as false negatives have a negative impact on the Dice metric. The result depends however on the size of the erroneous region compared to the cumulative size of the reference and evaluated segmentations. This means that missing a small lesion in an image which has multiple larger lesions will have a low impact on the score, while missing a small lesion in an image which presents only that lesion will produce a score of 0. To address this variation based on the total segmentation mask area, two metrics are usually computed. Global Dice is evaluated for all examinations combined into a single volume, and is more sensitive to larger lesions. Dice per case is computed for each individual volume, then averaged over all examinations, and it will apply a higher penalty to volumes with smaller lesions.

4 Implementation

4.1 Scientific and Deep Learning frameworks

The programming language used for this implementation is Python 3.8. I chose this language as it is very flexible for prototyping, has a very good ecosystem for scientific computing through the **SciPy**, provides Matlab-like syntax for matrix and vector operations with **NumPy**, and provides an interface for traditional machine learning methods through **scikit-learn**. Additionally, to process the initial CT image database, which is in the Neuroimaging Informatics Technology Initiative (NifTI) format, the **NiBabel** library [Bre22] was employed to extract the raw data from the volumetric examinations. To perform the image pre-processing and post-processing, I used **scikit-image**, a community developed library which provides easy access to a vast collection of image processing algorithms [Wal14]. Since not all medical images come in the same format, even though the database was stored in the NifTI format, other databases such as 3Dircad and local hospital resources are stored in the DICOM format, so I used the **pydicom** library [Mas22] to extract the pixel values, modify metadata and add colored ROIs on the segmented examinations. To create all visualizations related to the project, I used **Matplotlib**, a Python library for creating static and interactive visualizations, from statistics to images [Hun07].

Last but not least, I will discuss in more detail the choice for the deep learning framework used. The two most popular frameworks for Python development are PyTorch, developed by Facebook and TensorFlow, developed by Google. I chose to use PyTorch [Pas19], as it has several qualities which make it preferable to prototype new architectures on. Firstly, it supports the code as a model design principle, which enables easier modification of existing models or creating new models with minimal effort, since network layers are viewed as classes which process some input through a forward function and provide a built-in differentiation mechanism. This design model is further facilitated by the use of dynamic computation graph, which provides extended flexibility and the options of modifying model structures at runtime, as opposed to TensorFlow, which used static computation graphs until later versions. While TensorFlow requires a separate debugger to examine the data flow inside of models, PyTorch allows the standard Python debugger to break execution and examine variables inside models during runtime.

Besides these advantages, this framework presents a comprehensive list of features which facilitate fast development of deep learning workflows. Like most other deep learning frameworks, PyTorch is built upon a reverse-mode automatic differentiation algorithm, which lets it automatically compute gradients of custom models, or even for common programs which perform mathematical calculations and employ gradient-based optimizations. The differentiation

mechanism is based on operator overloading, which stores the mathematical function, every time a supported function is applied to a tensor which tracks gradients. To aid with workflow development, PyTorch implements dataloader classes tailored for image classification, which automatically generate labels and split the dataset. Moreover, reference models, such as AlexNet, VGGNets, and even newer models, such as ConvNeXt and Vision Transformers are available to train, evaluate and modify using the **torchvision** module. It also provides implementations of various cost functions, such as cross-entropy loss, MSE loss and L1 norm loss. To perform the actual network training, a series of optimizers can be used, starting with the well-known Stochastic Gradient Descent with momentum, and offering a multitude of other algorithms, such as Adam, Adagrad, RMSprop, Adadelta and LBFGS. To accelerate the deep-learning routines, this framework provides an easy for training using GPUs, thanks to its C++ core which interface with the CUDA runtime, thus providing low level, fast access to graphics computational resources.

4.2 Initial CT image pre-processing

The CT images in the LiTS17 dataset are stored in the NIfTI format, each file representing an entire examination volume. From the 131 examinations, 121 were used for network training and the remaining 10 for testing and determining the Dice scores. To make data loading easier, the volumes were split in individual slices. In this step, a windowing function is applied, which clamps the image values between -250 and 250 HU. Then, the pixel values are rescaled to the interval [0, 1]. This ensures that there won't be major contrast differences between the images during testing. To reduce the file size, all metadata was eliminated, and the slices are stored as raw pixel values. This way, individual slices can be randomly loaded without loading the entire volume. The initial image size is 512×512 pixels. To reduce the training time and memory requirements, I downsampled both the images and ground truth segmentation masks used for training to 256×256 pixels. Of course, training on downsampled images will reduce the segmentation accuracy, since the evaluation will be done on full resolution examinations, but it is a trade-off between performance, training time and the practicality to train on available hardware. Furthermore, the networks used in this implementation contain Batch Normalization layers for feature map regularization, and these layers learn the statistical properties of the training dataset. By applying the mentioned downsampling, the training batch size can be increased by a factor of 4, assuring that the BatchNorm layers will learn more statistically significant metrics, provided by a larger sample size. For the testing dataset however, only the images were downsampled, which ensures a fair evaluation of Dice scores, which would not be possible if the ground truth masks would also be downsampled.

4.3 Segmented liver image processing

As discussed, the segmentation system operates in two distinct stages, such that the lesion segmentation is performed on images which contain only the liver, without any surrounding tissue. To perform the liver extraction and to prepare the images for training, a series of steps are executed, some of which are similar to the CT pre-processing. Firstly, I apply the same windowing function which limit the intensity values to -250 to 250 HU, then the values are shifted by 250 and divided by 500 to occupy the range [0, 1]. The next step is to apply the liver segmentation, which was performed in the previous stage, as a transparent mask, which is done by multiplying the individual CT image pixels with the mask pixels. This will retain the original values inside the hepatic region, while setting the surrounding tissue and background to a value of 0. Then, the x, y, and z direction extremities of the liver volume need to be determined to generate a volume of interest. This is done by iterating through the slices from the three axes and retaining the minimum and maximum slice index where there is a non-zero value. The smaller in-plane dimension is increased symmetrically to match the larger one, resulting in a VOI with square cross-section. This is done to maintain the aspect ratio of the region of interest after resizing. The volume of interest is cropped from the original CT examination, thus eliminating any unnecessary regions and all slices which do not contain liver tissue, then each slice is resized to 256×256 pixels, such that all images have the same size during training. This method relies on the previous segmentation being correct.

Since the liver lesions have a similar intensity to the surrounding tissue, a contrast enhancing method is used to create better differentiation. First, I pass through each volume slice and assign the background pixels, which have a value of zero, a value equal to the mean of nonzero pixels in the slice. This both limits the image initial dynamic range and also eliminates strong edges around the liver, which have a strong effect on network activation but are not needed for lesion segmentation. The method used in this implementation is based on Contrast Limited Adaptive Histogram Equalization [Zui94]. Histogram equalization is a non-linear processing function which is based on the assumption that the gray-level mapping should be done based on the initial distribution of gray levels, such that a gray level class with more pixels should be assigned a bigger range in the output space. This process results in a mostly constant histogram, but depending on the image quality, it can increase details or just amplify background noise. Adaptive HE aims to improve some of the drawbacks by dividing the image into 8×8 blocks, then performing histogram equalization on each block individually, which incorporates contextual information and improves contrast, at the cost of amplifying noise in constant regions. Contrast Limited AHE limits contrast enhancement, specifically in homogeneous regions, where the histogram presents very high and narrow peaks. In this regions, the slope associated with the gray-level redistribution function is reduced, limiting noise amplification. The result of this histogram equalization method can be seen in the image below, where the lesion area becomes more visible.

Figure 8 CLAHE increases lesion contrast

4.4 From image classification to semantic segmentation

As it was already discussed in the previous chapters, an important contribution of this project is to study how image classification networks can be used as feature extractors to perform semantic segmentation, as well as to implement a solution based on this methodology. This chapter will provide detailed descriptions on the structural changes made to the networks, and how the data flow was modified to fit the task I want to achieve. The general rule is that, since classification networks are composed of a convolutional stage which performs feature extraction and a linear stage which performs classification, the latter should be removed, and instead a segmentation head, such as FCN or DeepLab, can be connected.

4.4.1 ResNet152

The software implementation of this ResNet variant is composed of multiple grouped layers. First, there is a 7×7 convolutional layer, which takes images with 3 channels and outputs a tensor with 64 feature maps. This layer was changed with one of the same kernel size, but which accepts single channel images, since the medical examinations are grayscale. This is followed by a sequence of 4 blocks which contain 3, 8, 36 and 3 bottleneck blocks, which share the same hyperparameters inside the blocks. The tensor dimensions are reduced immediately after the first convolutional layer through a Max Pooling layer. Each bottleneck block contains 3 individual convolutional layers, thus the 152 layers which characterize this network are obtained. This convolutional part is followed by a global average pooling and a fully connected layer, but these are eliminated since they are only necessary for classification. The output is taken directly from the fourth structural block, which has a size of $(b, 32, 32, 2048)$, where b is batch size. This means that each 256×256 input image was transformed into 2048 32×32 feature maps. In

the modified network architecture, this feature map block is passed to the DeepLab segmentation head. In the original DeepLabV3 implementation, the kernel dilation rates were set to 12, 24, and 36, since it was trained on significantly larger images of the COCO dataset. For feature maps of size 32×32, both kernels with dilation 24 and 36 would act as 1×1 convolutions, since the outer kernel elements will be outside the image bounds. This is why the dilation rates were reduced to 4, 8 and 12. Also, the last convolutional layer of the segmentation dataset was replaced, such that it outputs an image with only one channel, which represent the pixel probabilities of containing liver or lesion tissue, depending on the training case.

4.4.2 ResNeXt101

The software implementation of this ResNeXt variant is composed of multiple grouped layers, just like the network described above. Similar to its ResNet counterpart, there is a 7×7 convolutional layer, which takes images with 3 channels and outputs a tensor with 64 feature maps. This layer was changed with one of the same kernel size, but which accepts single channel images, since the medical examinations are grayscale. This is followed by a sequence of 4 blocks which contain 3, 4, 23 and 3 bottleneck blocks, which share the same hyperparameters inside the blocks. However, since this network implementation uses grouped convolution for the bottleneck blocks, I also have to specify its variant, which in this case is **32×8d**. This means that inside each bottleneck layer, a set of 32 grouped convolutions is computed, each having an input and output size of 8×8 pixels. This convolutional part is followed by a global average pooling and a fully connected layer, but these are eliminated since they are only necessary for classification, just like in the previous case. The output of the fourth structural block has the same shape as for the previous network, so I execute the same modifications, based on the same reasoning, for this network.

4.4.3 DenseNet201

The software implementation of this DenseNet variant is composed of multiple different structural blocks: dense blocks and transition blocks. First, there is a 7×7 convolutional layer, which takes images with 3 channels and outputs a tensor with 64 feature maps. This layer was changed with one of the same kernel size, but which accepts single channel images. It is followed by a sequence of 4 dense blocks composed of 6, 12, 48 and 32 sequences of BatchNorm-ReLU-Conv2D-BatchNorm-ReLU-Conv2D. These sequences are densely connected inside the structural blocks, but no dense connections appear between the dense blocks. Between each two dense blocks, there is a transition block, which performs a convolution and an average pooling with kernel size 2 and stride 2, effectively reducing the in-plane feature map dimensions by a factor of 2 along each axis. Just like before, the linear classification part of the network was removed. In the original formulation of the network, the tensor shape of the last convolutional layer for an input of size 256×256 was $(b, 8, 8, 1920)$. This tensor has in plane dimensions which are too small to generate a satisfactory segmentation result when they are upscaled to

256×256 by the segmentation head. To address this issue, I modified the transition blocks, such that only the first one performs average pooling, while in the second and third it is eliminated. This increases the output tensor in-plane dimension to 32×32, identical to the ResNet variants, but with a different number of channels. The next steps are repeated in the same manner as for the previous networks.

4.4.4 InceptionV3

The software implementation of the Inception architecture is composed of a convolutional sequence and an inception sequence. First, there is a 3×3 convolutional layer, which takes images with 3 channels and outputs a tensor with 32 feature maps. This layer was changed with one of the same kernel size, but which accepts single channel images. This is the first in a group of 5 convolutional layers, which also contains max pooling layers after the third and fifth convolutional layer. This sequence outputs a tensor of size $(b, 35, 35, 192)$, which is then passed to the Inception sequence. This is composed of 3 different Inception designs, each with 3, 5, and 2 modules. Each design implements successively more complex and deeper asymmetric convolutions. After this sequence, there is the classification part of the network, which I eliminated to extract just the feature maps, which have a size of $(b, 8, 8, 2048)$. Similar to the case presented above, the small in-plane dimensions make this a less than ideal case for upsampling when generating segmentation masks. The issue was addressed by removing the max pooling layers from the convolutional sequence, thus increasing the in-plane dimensions to 32×32. The steps to connect the segmentation head to the feature extraction output are the same as with the other three models. The Inception model implementation also provides an auxiliary output, which taps into the intermediate output of the network, after 8 Inception blocks. This is generally used to improve convergence during training by combating the vanishing gradients problem. However, it is not used in this case, as it would require an additional DeepLab module, which would greatly increase training time and memory cost.

4.5 PyTorch Dataset and Dataloader

In the PyTorch framework, data structured for training is loaded using extensions of the Dataset class. This provides an interface between the routines which read the data from disk and perform transformations, and the routines which arrange data into randomized training batches, with their corresponding labels or masks. PyTorch offers pre-built Dataset classes for most common image datasets, such as CIFAR, COCO, Cityscapes and some video datasets, such as Kinetics and HMDB51. It also provides a template dataset for creating a custom image dataset, where classification labels are generated based on the folder name where the images are in. Since there is no implementation for the LiTS17 dataset, I had to implement my own solution. To do this, I first created a class which inherits the base Dataset class. It takes as parameters a root directory for the images and masks folders, a transformation for the image and one for the mask. These transformations can be used to resize the images according to specific needs, adjust

contrast and introduce data augmentations in the case of smaller datasets. When the lass is initialized, it generates a list of all files found in the images and masks sub-directories of the root directory, and sorts them alphabetically. Then, when a certain image index is requested, the class looks up the corresponding image and mask path and loads them into NumPy arrays, interpreting them as binary files. Then, knowing the pixel count, the arrays are reshaped into 2D matrices, to which the specified transforms are applied, if any. The output of this loading routine is a pair of PyTorch tensors, one representing the image data and one the mask data. By default, the only transformation I use is image normalization with mean 0.5 and standard deviation 0.5. This will move the pixel values into the range [-1, 1]. No resizing is done at this point, as it was performed in the pre-processing stage of the original dataset. The routines implemented only handle loading the data from disk and preparing it, while another DataLoader class generates the training batches. For each epoch, it randomizes the order of image indices in the dataset. Then, it sequentially samples blocks of 32 images and concatenates them to generate the training mini-batches. This step of mini-batch randomization is essential for successfully training Batch Normalization layers, since successively processing a random subset of samples will give better statistics on the global dataset.

4.6 Model training

4.6.1 Training the segmentation networks

The training of the models which make up the proposed system was done in a sequential manner: first I trained liver segmentation variants, then the lesion segmentation variants of the same models. However, training in this order is not compulsory, since lesion training was done with the reference segmentations, and not with the segmentations provided by the previous step. All four segmentation models were trained individually. The feature extractor part of every network uses pretrained weights from the PyTorch database. These are the weights corresponding to models trained on the 1000 class ImageNet dataset. This means that the models already can extract features relevant for classification, which should reduce training time for this new task. The weights and biases used for the segmentation head of each model are initialized using the Kaiming method. Even if the feature extraction section uses pretrained weights, these weights are also optimized by the training algorithm. Since the first convolutional layer of each network was replaced to accommodate grayscale images, it means that it has random weights and should be optimized, so its gradients have to be computed. Since this can only be done by computing the gradients of all downstream layers, all layers can be optimized with a negligible increase in execution time. Most of the computation time is dedicated to network inference and gradient back-propagation, not to weight adjustment via gradient-based methods.

Each liver segmentation model was trained for 200 epochs using the stochastic gradient descent (SGD) optimizer. I set the learning rate to 0.05 and a momentum of 0.9. As shown by

[Sut13], using momentum and choosing a suitable model initialization can lead to better performance and faster convergence, as it accelerates gradient descent in the direction of persistent decrease of the cost function. To aid with convergence as the model parameters approach a point of minimum, a learning rate scheduler is used, which decreases the learning rate every 10 epochs by 5%. This way, the step size of the gradient descent optimizer becomes smaller as it approaches a solution so it cannot "jump" to a nearby local minimum. This also decreases overshoot when searching for the optimum point, so it can converge in fewer steps. The training was done in batches of 32 images, which was the biggest size that could be loaded on the available hardware.

For the lesion segmentation models, I used the network parameters learned in the previous step as a starting point for model training. The reasoning behind this is that the previous networks learned to distinguish between liver features and other organs, so this information could be transferred faster to particularities of the liver tissue, such as lesions. Since I am starting with models which can already perform liver localization, a higher learning rate of 0.1 was set, along with a more aggressive learning rate scheduler, where the learning rate is halved after each epoch. This ensures that the model parameters will move faster towards the optimum in the beginning of the optimization, then the later epochs will perform a very fine parameter tuning to move as close as possible towards the minimum point. The training is done for only 20 epochs, as further iteration caused the learning rate to become to small to make effective improvements. The learning rate for the last epoch is $\frac{0.1}{2^{19}} \approx 1.9\text{e-}7$. These values for learning rate decay, initial learning rate and total number of epochs were determined empirically after running multiple tests and comparing the evolution of the loss function. More tests with more various training hyper-parameters value combinations may result in a better starting point and schedule, but I was limited by computation time restrictions.

To accelerate the training process, I used the high performance computing cluster provided by the Faculty of Automatic Control and Computer Science. The machine used for training is configured as a SLURM partition, where jobs can be submitted by individual users. It is equipped with two 20-core Intel Xeon E5-2670 CPUs, 64 GB of DDR3 RAM and three NVIDIA Tesla K40m GPUs with 12GB of GDDR5 VRAM, each processing unit having 2880 cores.

4.6.2 Training the Fusion system

After the training process for all the models is finished, the resulting segmentations for the training dataset are saved to disk. The model-generated masks are needed for this step, and running inference on the network is much more time consuming than loading the images from memory.

I defined the Fusion model in the PyTorch framework as an extension to the base Module class. This allows us to define custom network parameters and to override the forward method, which runs model inference. The Fusion model can take as input either a batch of 2D images with 4 channels or a batch of 2D images with one channel. In the first case, the four channels represent the masks generated by each segmentation model. This mode was used for Fusion system training, as it can load the previously generated masks without passing the images through the segmentation networks. The second case is used when a new CT image has to be evaluated through the Fusion system. The model loads internally all segmentation models and evaluates their output, which is then processed in the same manner as in the first case. The Fusion system has the role of weighting the masks provided by each segmentation model, then adding them and producing a final mask. To perform this operation, four independent weights could be defined as model parameters, enable gradient tracking and compute the result in a weighted sum function. However, a much more elegant solution, which takes advantage of the convolution mechanism provided by the PyTorch framework, is to compute this weighted sum through a convolutional layer. The layer takes an input with 4 channels, computes the convolution with a 1×1 kernel, and outputs the single-channel image. The trainable parameter of this layer is a tensor of shape $(4,1,1,1)$, essentially 4 values which weight each mask accordingly and sum the result into one value. The convolutional layer has the possibility to also train a bias parameter, but this was disabled, since it would add a constant value to each binary mask, which cannot improve performance for this application. This method also takes advantage of the built-in autograd mechanism, allowing us to optimize this parameter in the same way one would train a traditional model.

Using the same stochastic gradient descent algorithm, the weight corresponding to each segmentation network is updated in such a way as to minimize the cross-entropy loss between the liver class and background class. The training dataset consisted in the collection of masks generated by each model as input, and the LiTS17 reference segmentations as ground truth. This optimization process is performed in an iterative fashion over 5 epochs, with a learning rate of 10 and momentum 0.9. Since this model has a very small parameter space, the choiche of number of iterations and learning rate scheduling is less strict. In my experiments, I observed that the 4 optimized values converged to the same value after roughly 4 epochs, regardless of learning rate, as long as it was in the interval [1, 15]. For bigger values, the solution had the tendency to oscillate around the optimum. Small values for the learning rate caused the model to converge slower, but the minimum point was approximately the same, within a margin of 1e-5.

Since the idea of segmentation fusion is based on the fact that each network will produce errors in different spots in the image and a decision system could filter out individual errors, the segmentation errors were also calculated as the absolute difference between the ground truth segmentation and the segmentations provided by each network. Then, the Dice similarity coefficient is calculated between every two models over all validation segmentations. A low dice

value would be desired since it would mean that the errors do not overlap and have the potential to be eliminated by the fusion system. The figure below presents the matrix of Dice coefficients. Between all models before fusion, the error overlap is under 70% Dice similarity.

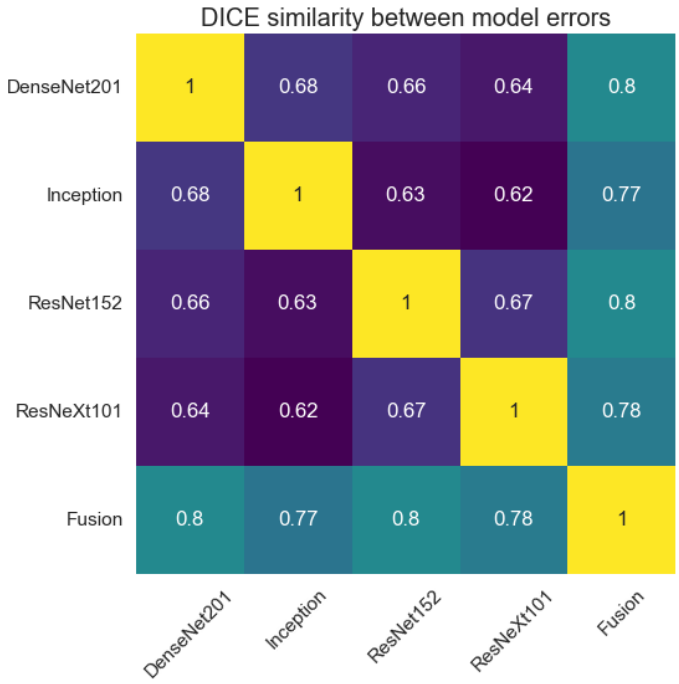


Figure 9 *Dice similarity between segmentation errors for liver tissue*

4.7 Liver segmentation post-processing

After the segmentations are generated by the networks, a sequence of additional processing operations will be applied on them to remove segmentation artifacts which do not belong to the liver area. Each slice is a binary segmentation mask and is processed individually. The first step is to find all connected regions and label them individually. If the number of regions is less than 2, the image is left as is. Otherwise, the centroid and area metrics are computed for each connected region and sort them descending by area. The region with the biggest area is considered to be the main liver area. Then, I iterate through all the remaining regions and remove them if the distance to the main area is greater than 200 pixels or its area is less than 8 times smaller than that of the main liver region. These decision thresholds were found empirically, trying to eliminate as many of the erroneous segmentations from the spleen or intestine area without removing any correct liver segmentations. The figure below shows an example when this method removes unwanted regions from the segmentation while leaving the liver region intact.

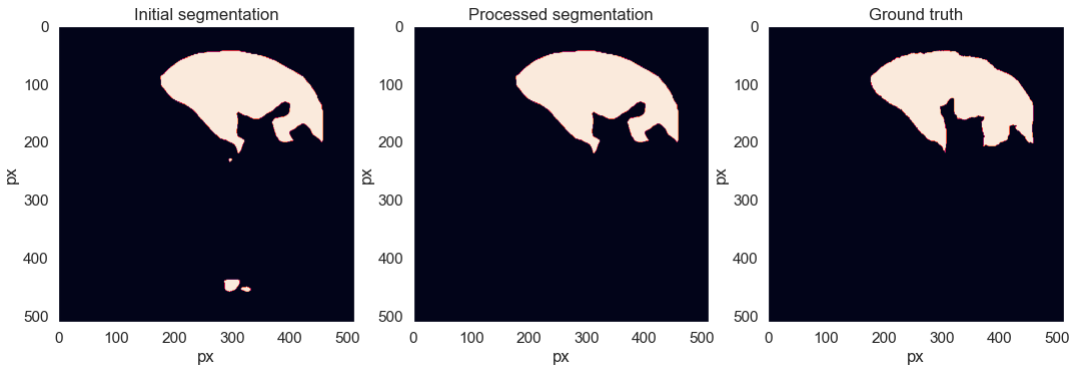


Figure 10 Example of segmentation post-processing

4.8 Lesion segmentation post-processing

To refine the lesion segmentation masks the same method used for liver masks cannot be employed since these lesions can appear anywhere inside the liver tissue area and can have various sizes, independent of one another. Even though the liver images also have some background pixels, none of the models produced masks which overlap the background, so this is not a concern. However, the masks will be processed by setting a minimum area for a tumor to be registered, otherwise it will be deleted. The threshold chosen for this operation is 10 pixels. This value was determined empirically, as it eliminates false positives caused by the image granularity and noise.

4.9 Creating the final segmentation

Until this point, the workings of each individual system were described, but without a detailed description of how they relate to one another. The two fusion systems that were trained each output a segmentation mask for each input image, one for the liver tissue and one for the lesion tissue, each mask having a size of 256×256 pixels. The liver tissue mask can easily be upscaled using bilinear interpolation to match the input image size, and will have a direct overlap with the liver tissue in the initial CT image. However, the same steps cannot be applied to the lesion mask, as it was cropped via the determined volume of interest from the volumetric representation of the examination. To be able to overlap the mask on the initial image, the same VOI has to be determined using the same method described before. Then, we resize the 256×256 segmentation mask to match the in-plane dimensions of the VOI. Taking into account the position and dimensions of the VOI, the resized mask is padded with zeros in all directions to match the 512×512 dimension of the initial examination. Moreover, empty masks need to be added before and after the lesion masks to account for the examination slices which do not contain liver tissue and were not processed in the second segmentation step. By following these

steps, two maps are obtained, which match the dimensions of the CT examination, so they can be used either as semi-transparent coloured masks to display the result in a way that is easier to understand for a human operator, or can be used to compute additional metrics which could be helpful for diagnostics, such as liver volume or lesion volume, also referred to as tumor burden.

4.10 Medical application

Having a model which provides good segmentation metrics is essential for a functional biomedical diagnostics system, but it also has to be functional in a real scenario, such as a clinical environment, not only in the development environment, where it processes well-structured data with a known standardized format and its performances are analyzed internally against a reference ground truth. This is why I have developed a solution which can load DICOM examinations and generate corresponding result in the DICOM format, such that they can be easily loaded in specialized programs which are available on computers in medical institutions. These programs provide an extensive array of measurement tools that can aid in deciding on a diagnostic, so saving the results in this standard medical format is essential. Usually, CT examinations saved in the DICOM format are stored as individual slices, where all slices of an examination are stored in the same folder, where the alphabetical order of files defines their order in the CT sequence. These files contain a large amount of metadata which defines their their order in the volume, pixel size, slice thickness, pacient data, CT acquisition protocol and X-ray energy among others. These fields are organized by DICOM tags.

First, the application loads each file listed in the selected directory using a dedicated Python library. Then, the raw pixel values are extracted and processed by a modality lookup table specified by the DICOM header. The data structures are transformed to NumPy arrays and are then processed in the same way to the LiTS dataset: pre-processing, two-stage segmentation, post-processing and mask resizing and padding.

To generate the output DICOM, the initial object is copied with all metadata. The original CT image si converted from grayscale to RGB by repeating the gray channel 2 times. This allows for colors to be added on the image. The liver mask is given a green color and alpha value of 0.3 and is overlaid on the initial examination. The lesion mask has the same transparency value but red color, and is overlaid in the same manner. However, in order for DICOM viewers to be able to register these changes, a few modifications to the metadata need to be made. First, the PhotometricInterpretation tag has to be set to RGB, and the SamplesPerPixel to 3, since each pixel is now represented by 3 bytes, one for each color channel. Next, the HighBit tag is set to 7 to specify that the data is stored in the little-endian format. The series number is incremented, so the file viewer will see them as 2 different datasets, then the files can be saved to disk. The result is that the segmentation results can be viewed interactively as colored regions overlaid on the initial CT examination.

5 Experimental Results And Discussions

This section will present the experimental results which are the outcome of the work presented in this document. The metric I will be focusing on is the Dice similarity index, as this allows me to make comparisons to other solutions trained on the same dataset. The system's performance was evaluated on the testing dataset, ensuring that none of the images were seen during the training phase. Even though the system processes images at a smaller size, the segmentations were upsampled to match the ground truth size provided in the dataset, to create a fair comparison. The Dice score was calculated per examination (i.e., for a whole volume, not for each individual slice) and then averaged over the 10 volumes reserved for testing. Table 1 below shows the segmentation scores for each model with and without post-processing, as well as for the fusion system, which was evaluated with the post-processed images. In the case of lesion masks, the proposed post-processing method did not provide a measurable change in performance metrics, so only one set of scores is displayed as seen in table 2.

Model	InceptionV3	DenseNet201	ResNet152	ResNeXt101	Fusion
Without post-processing	94.41%	94.94%	95.54%	95.28%	-
With post-processing	95.04%	95.25%	95.59%	95.27%	**95.67%**

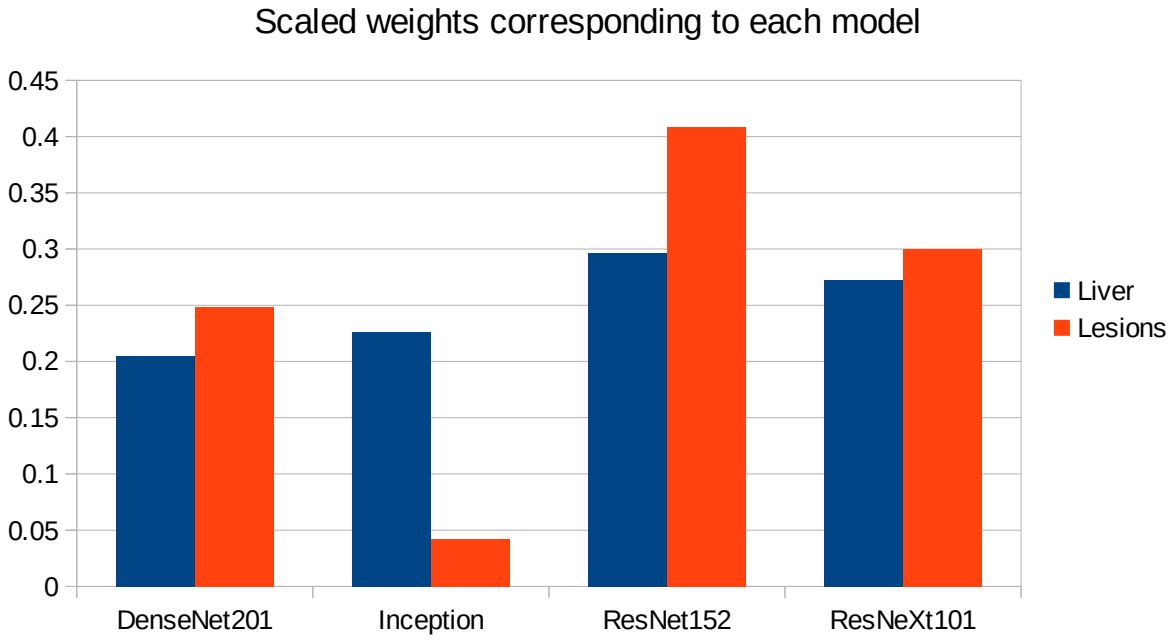
Table 1: Dice scores for liver segmentation

Model	InceptionV3	DenseNet201	ResNet152	ResNeXt101	Fusion
Dice score	57.38%	70.15%	75.12%	74.61%	**75.83%**

Table 2: Dice scores for tumor segmentation

As expected, not all models have the same performance, this being influenced by the model's architecture and number of parameters. This difference in segmentation performance can also be seen in the proportion each model's decision has in the result of the Fusion system. Models which have better performance individually get assigned a higher relative weight, which means that their segmentation has a higher chance to reach the final result. Since no model has a weight higher than 0.5 and the final decision is done by rounding the summed and weighted masks, it means that no model decides by itself what the output mask looks like, and at least 2 or 3 of the models have to agree on a pixel value for it to be marked as part of the final mask.

Starting with equal weights of 25% percent, the optimized values are as follows, for the two fusion systems implemented:



Even though in the case of liver segmentation the model weights are reasonably balanced since the performance values are closer, in the second case we can see that the Inception-generated masks have a weight under 5% due to its significantly lower Dice score.

The figures below show a set of randomly picked CT scans from the testing dataset along with their corresponding segmentations and ground truth. The second sample (Figure 11) shows how an incorrect segmentation, which appears only in the case of the Inception network is removed by the decision fusion system. On the contrary, the fourth sample highlights a case when two networks segment both liver lesions while the other identify only the main area, and the fusion system eliminates the smaller area. This suggests that the post-processing parameters have to be adjusted further to ensure that there are no liver regions left out by the segmentations. Figure 12 shows model outputs for a randomly selected sample of liver images. For cases where the tumor burden is small, and there is only one lesion area, the reference mask and the fusion mask overlap almost perfectly. On the other hand, as the tumor load increases and more regions appear, some are missed by the proposed system. Lastly, darker areas in some images can be falsely detected as lesions even though they are shadows caused by the X-ray scanning procedure or signify the presence of denser tissue.

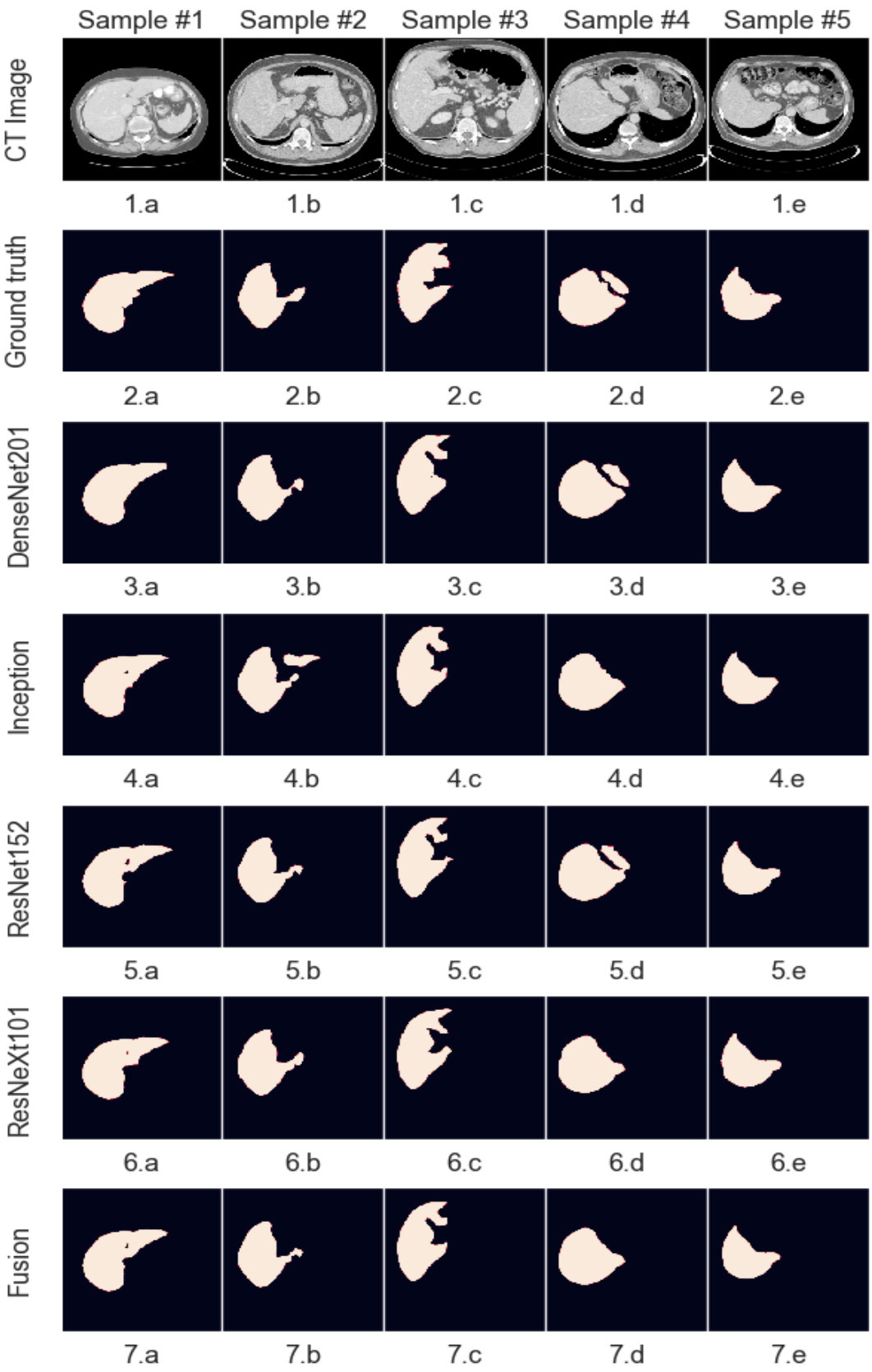


Figure 11: Sample liver segmentations

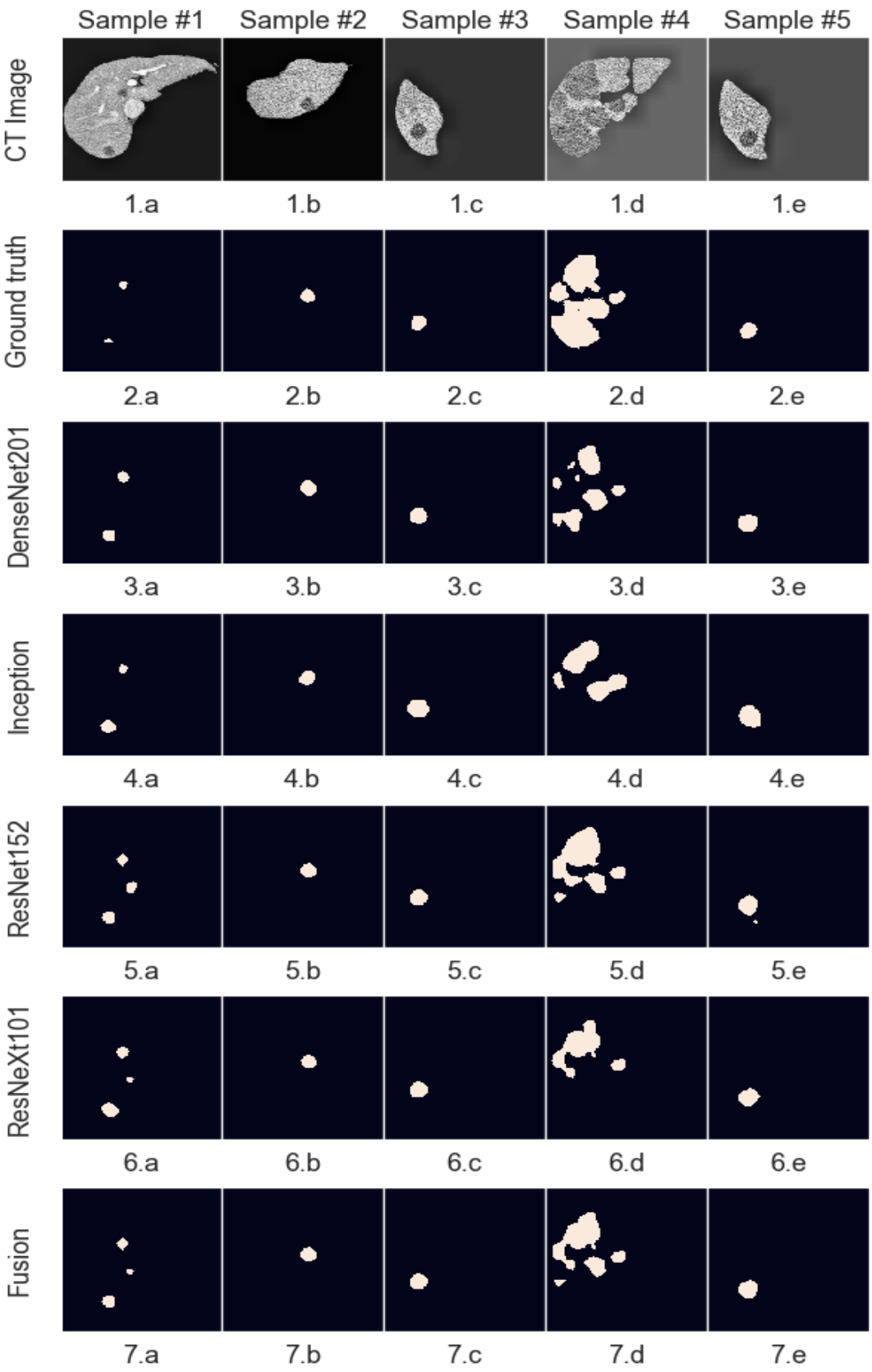


Figure 12: Sample lesion segmentations

Comparing the proposed method against state of the art solutions, it can be seen that some improvements still have to be made. The comparison table below shows that some methods that are based on more advanced variations of the UNet architecture achieve better performance metrics. Also, when discussing performance, model size should also be taken into account. To achieve these scores, I used variations of classification networks which have a high number of layers when compared to encoder-decoder networks. For example, for the proposed system, the size of all model weights stored on disk reaches almost 1800 MB, while UNets occupy a size around 30 MB for the base variant and up to 120 MB for more complex variations. This means that the Fusion system along with its internal models is a lot slower to load, needs significantly more computational resources to train, the inference times are also higher. However, this being a medical application, there is no need for instant segmentation results, so the high processing times aren't necessarily a significant drawback.

Method		Liver	Lesion
UNet + graph cut	[Liu19]	95.05%	-
TDP-CNN	[Men20]	96.5%	68.9%
SAR-UNet	[Wan21]	95.71%	-
3D RA-UNet	[Jin20]	96.1%	59.5%
2.5D residual UNet	[Han17]	-	67%
2.5D UNet	[War21]	95.3%	78.4±16%
XNet	[Chi21]	97.1%	84.3%
Proposed method		**95.67%**	**75.83%**

Table 3 Performance comparison between the proposed system and other methods

To showcase the potential this model has to be used in a practical environment, I used an anonymous DICOM examination provided by the Fundeni hospital. This means that the images are acquired using different devices than the ones in the LiTS dataset, thus allowing a better understanding of the model's behavior when used on data from a different source. The image below shows an input DICOM volume slice and the corresponding model output:

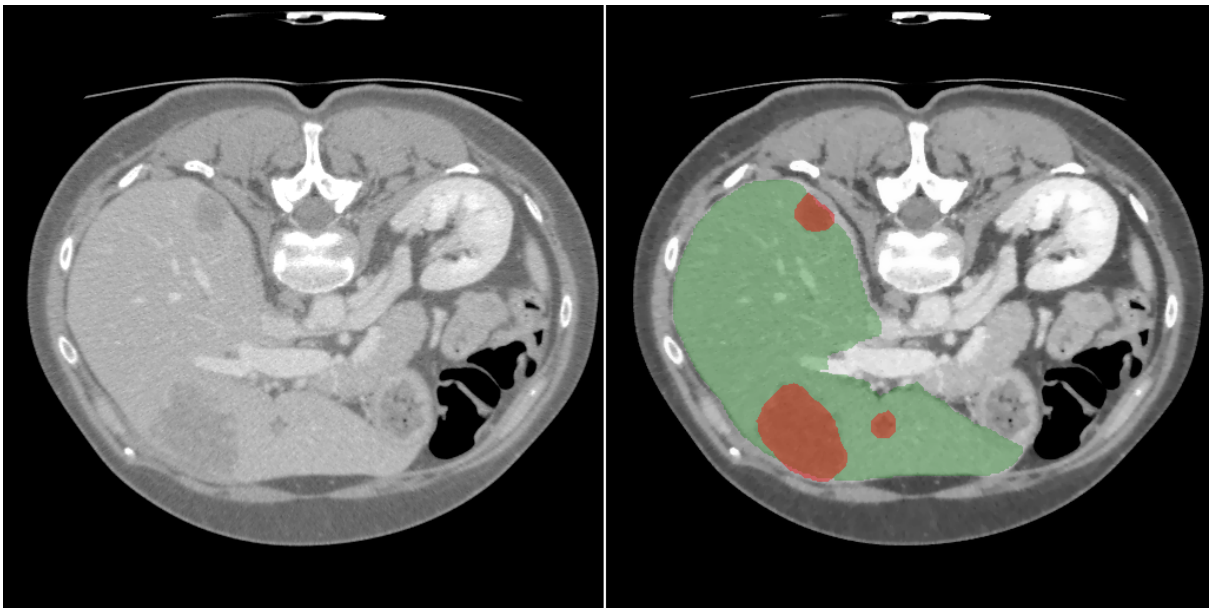


Figure 13 Sample DICOM input and its corresponding output

Even though the liver and lesion segmentations seem to overlap correctly in the model output, a performance score cannot be computed for the masks generated by the proposed model in this experiment. This is due to the fact that only the DICOM examinations were provided, without any reference segmentation masks against which the model results can be compared.

6 Conclusions

In this project we have proven that multiple image classification convolutional neural networks can be used as feature extractors for semantic segmentation in the context of medical imaging, specifically liver segmentation, and that a decision fusion system can further improve the performance of these networks by removing individual errors. Model performance was also increased by employing various pre-processing and post-processing techniques on the images and resulting masks. From the results and the performance metrics obtained, we deduce that this solution produces satisfactory results, but still needs adjustments and parameter fine-tuning to be on par with state-of-the-art methods, such as improved UNets.

After the system parameters are adjusted to match the needs in clinical settings, the next steps would be to devise an artificial intelligence system that can analyze the identified tumor patches across multiple CT contrast phases. Based on the tissue behavior across these phases, the specific type of lesion could be theoretically identified. However, to achieve this, a separate database needs to be created, one which contains lesion classifications based on expert opinions or biopsies, and which also contains examinations corresponding to the three contrast phases for all patients. A system that could provide a specific diagnosis based on these CT scans would bring great value in clinical settings, as costly and invasive biopsies could be avoided in some cases. This being said, reliable automated diagnostics systems still have a long way to become independent on expert opinion, based partly on the fact that there is great variability in the human body, which cannot be captured by a medical image dataset, regardless of how large it is.

Bibliography

[Bar13] Baron, R. L. (2013). Understanding and optimizing use of contrast material for CT of the liver. *163*(2), 323–331. https://doi.org/10.2214/AJR.163.2.8037023

[Ber19] Bertels, J., Eelbode, T., Berman, M., Vandermeulen, D., Maes, F., Bisschops, R., & Blaschko, M. (2019). *Optimizing the Dice Score and Jaccard Index for Medical Image Segmentation: Theory & Practice*. https://doi.org/10.1007/978-3-030-32245-8_11

[Bil19] Bilic, P., Christ, P. F., Vorontsov, E., Chlebus, G., Chen, H., Dou, Q., Fu, C.-W., Han, X., Heng, P.-A., Hesser, J., Kadoury, S., Konopczynski, T., Le, M., Li, C., Li, X., Lipkovà, J., Lowengrub, J., Meine, H., Moltz, J. H., … Menze, B. H. (2019). *The Liver Tumor Segmentation Benchmark (LiTS)*. http://arxiv.org/abs/1901.04056

[Bot12] Bottou, L. (2012). Stochastic Gradient Descent Tricks. In: Montavon, G., Orr, G.B., Müller, KR. (eds) Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, vol 7700. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-35289-8_25

[Bre22] Brett, M., Markiewicz, C. J., Hanke, M., Côté, M.-A., Cipollini, B., McCarthy, P., Jarecka, D., Cheng, C. P., Halchenko, Y. O., Cottaar, M., Larson, E., Ghosh, S., Wassermann, D., Gerhard, S., Lee, G. R., Wang, H.-T., Kastman, E., Kaczmarzyk, J., Guidotti, R., … freec84. (2022). *nipy/nibabel: 4.0.0rc0*. https://doi.org/10.5281/ZENODO.6617127

[Cam09] Campadelli, P., Casiraghi, E., & Esposito, A. (2009). Liver segmentation from computed tomography scans: A survey and a new algorithm. *Artificial Intelligence in Medicine*, *45*(2–3), 185–196. https://doi.org/10.1016/J.ARTMED.2008.07.020

[Cas05] Castellino, R. A. (2005). Computer aided detection (CAD): an overview. *Cancer Imaging*, *5*(1), 17. https://doi.org/10.1102/1470-7330.2005.0018

[Che17] Chen, L.-C., Papandreou, G., Schroff, F., & Adam, H. (2017). *Rethinking Atrous Convolution for Semantic Image Segmentation*. https://arxiv.org/abs/1706.05587v3

[Chi21] Chi, J., Han, X., Wu, C., Wang, H., & Ji, P. (2021). X-Net: Multi-branch UNet-like network for liver and tumor segmentation from 3D abdominal CT scans. *Neurocomputing*, *459*, 81–96. https://doi.org/10.1016/j.neucom.2021.06.021

[Goo16] Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron (2016). "6.5 Back-Propagation and Other Differentiation Algorithms". Deep Learning. MIT Press. pp. 200–220. ISBN 9780262035613.

[Gru19] Gruber, N., Antholzer, S., Jaschke, W., Kremser, C., & Haltmeier, M. (2019). *A Joint Deep Learning Approach for Automated Liver and Tumor Segmentation*. http://arxiv.org/abs/1902.07971

[Han17] Han, X. (2017). *Automatic Liver Lesion Segmentation Using A Deep Convolutional Neural Network Method*. https://doi.org/10.1002/mp.12155

[He15] He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition*. http://arxiv.org/abs/1512.03385

[Hin12] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). *Improving neural networks by preventing co-adaptation of feature detectors*. http://arxiv.org/abs/1207.0580

[Hua16] Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2016). *Densely Connected Convolutional Networks*. http://arxiv.org/abs/1608.06993

[Hun07] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science and Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

[Iof15] Ioffe, S., & Szegedy, C. (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. http://arxiv.org/abs/1502.03167

[Jin20] Jin, Q., Meng, Z., Sun, C., Cui, H., & Su, R. (2020). RA-UNet: A Hybrid Deep Attention-Aware Network to Extract Liver and Tumor in CT Scans. *Frontiers in Bioengineering and Biotechnology*, *8*. https://doi.org/10.3389/fbioe.2020.605132

[Kal22] Kalra, A., Yetiskul, E., Wehrle, C. J., & Tuma, F. (2022). Physiology, Liver. *StatPearls*. https://www.ncbi.nlm.nih.gov/books/NBK535438/

[Li.W15] Li, W., Jia, F., & Hu, Q. (2015). Automatic Segmentation of Liver Tumor in CT Images with Deep Convolutional Neural Networks. *Journal of Computer and Communications*, *03*(11), 146–151. https://doi.org/10.4236/jcc.2015.311023

[Li.Z21] Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2021). A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 1–21. https://doi.org/10.1109/TNNLS.2021.3084827

[Lin13] Lin, M., Chen, Q., & Yan, S. (2013). *Network In Network*. http://arxiv.org/abs/1312.4400

[Liu19] Liu, Z., Song, Y.-Q., Sheng, V. S., Wang, L., Jiang, R., Zhang, X., & Yuan, D. (2019). Liver CT sequence segmentation based with improved U-Net and graph cut. *Expert Systems With Applications*, *126*, 54–63. https://doi.org/10.1016/j.eswa.2019.01.055

[Mas22] Mason, D., scaramallion, mrbean-bremen, rhaxton, Suever, J., Vanessasaurus, Orfanos, D. P., Lemaitre, G., Panchal, A., Rothberg, A., Herrmann, M. D., Massich, J., Kerns, J., Golen, K. van, Robitaille, T., Biggs, S., moloney, Bridge, C., Shun-Shin, M., … Wortmann, J. (2022). *pydicom/pydicom: pydicom 2.3.0*. https://doi.org/10.5281/ZENODO.6394735

[Men20] Meng, L., Tian, Y., & Bu, S. (2020). Liver tumor segmentation based on 3D convolutional neural network with dual scale. *Journal of Applied Clinical Medical Physics*, *21*(1), 144–157. https://doi.org/10.1002/acm2.12784

[Nay20] Nayantara, P. V., Kamath, S., Manjunath, K. N., & Rajagopal, K. v. (2020). Computer-aided diagnosis of liver lesions using CT images: A systematic review. *Computers in Biology and Medicine*, *127*, 104035. https://doi.org/10.1016/J.COMPBIOMED.2020.104035

[Pas19] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, *32*. https://doi.org/10.48550/arxiv.1912.01703

[Ron15] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9351*, 234–241. https://arxiv.org/abs/1505.04597v1

[Sri14] Srivastava, N., Hinton, G., Krizhevsky, A., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In *Journal of Machine Learning Research* (Vol. 15).

[Sut13] Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). *On the importance of initialization and momentum in deep learning*.
[Sze14] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). *Going Deeper with Convolutions*. http://arxiv.org/abs/1409.4842
[Sze15] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). *Rethinking the Inception Architecture for Computer Vision*. http://arxiv.org/abs/1512.00567
[Vai20] Vaidehi Nayantara, P., & Kamath, S. (2020). Computer-aided diagnosis of liver lesions using CT images: A systematic review. *Computers in Biology and Medicine*, *127*. https://doi.org/10.1016/j.compbiomed.2020.104035
[Wan21] Wang, J., Lv, P., Wang, H., & Shi, C. (2021). SAR-U-Net: Squeeze-and-excitation block and atrous spatial pyramid pooling based residual U-Net for automatic liver segmentation in Computed Tomography. *Computer Methods and Programs in Biomedicine*, *208*, 106268. https://doi.org/10.1016/j.cmpb.2021.106268
[War21] Wardhana, G., Naghibi, H., Sirmacek, B., & Abayazid, M. (2021). Toward reliable automatic liver and tumor segmentation using convolutional neural network based on 2.5D models. *International Journal of Computer Assisted Radiology and Surgery*, *16*(1), 41–51. https://doi.org/10.1007/s11548-020-02292-y
[WCF22] *Liver cancer statistics | World Cancer Research Fund International*. (n.d.). Retrieved December 8, 2021, from https://www.wcrf.org/dietandcancer/liver-cancer-statistics/
[Wal14] van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., Yu, T., & the scikit-image contributors. (2014). scikit-image: image processing in Python. *PeerJ*, *2*, e453. https://doi.org/10.7717/peerj.453
[Wel22] Welper, G. (2022). Universality of gradient descent neural network training. *Neural Networks*, *150*, 259–273. https://doi.org/10.1016/j.neunet.2022.02.016
[Xie16] Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2016). *Aggregated Residual Transformations for Deep Neural Networks*. http://arxiv.org/abs/1611.05431
[Yam18] Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, *9*(4), 611–629. https://doi.org/10.1007/S13244-018-0639-9/FIGURES/15
[Yas18] Yasaka, K., Akai, H., Abe, O., & Kiryu, S. (2018). Deep learning with convolutional neural network for differentiation of liver masses at dynamic contrast-enhanced CT: A preliminary study. *Radiology*, *286*(3), 887–896. https://doi.org/10.1148/radiol.2017170706
[Zui94] K. Zuiderveld: Contrast Limited Adaptive Histogram Equalization. In: P. Heckbert: Graphics Gems IV, Academic Press 1994, ISBN 0-12-336155-9