Universidade Estadual de Campinas
Instituto de Computação

William Hideki Azana Tustumi

Automatic Liver Tumor Segmentation from Computed Tomography Images Based on 2D and 3D Deep Neural Networks

Segmentação Automática de Tumores do Fígado a Partir de Imagens de Tomografia Computadorizada Baseada em Redes Neurais Profundas 2D e 3D

CAMPINAS
2020

William Hideki Azana Tustumi

Automatic Liver Tumor Segmentation from Computed Tomography Images Based on 2D and 3D Deep Neural Networks

Segmentação Automática de Tumores do Fígado a Partir de Imagens de Tomografia Computadorizada Baseada em Redes Neurais Profundas 2D e 3D

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/Orientador: Prof. Dr. Hélio Pedrini
Co-supervisor/Coorientador: Prof. Dr. Guilherme Pimentel Telles

CAMPINAS
2020

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

T878a

Tustumi, William Hideki Azana, 1993-
Automatic liver tumor segmentation from computed tomography images based on 2D and 3D deep neural networks / William Hideki Azana Tustumi. – Campinas, SP : [s.n.], 2020.

Orientador: Hélio Pedrini.
Coorientador: Guilherme Pimentel Telles.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Segmentação de imagens. 2. Segmentação de imagens médicas. 3. Inteligência artificial. 4. Redes neurais (Computação). I. Pedrini, Hélio, 1963-. II. Telles, Guilherme Pimentel, 1972-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Segmentação automática de tumores do fígado a partir de imagens de tomografia computadorizada baseada em redes neurais profundas 2D e 3D
Palavras-chave em inglês:
Image segementation
Medical image segmentation
Artificial intelligence
Neural networks (Computer science)
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Hélio Pedrini [Orientador]
Ronaldo Cristiano Prati
Levy Boccato
Data de defesa: 29-10-2020
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-6275-1614
- Currículo Lattes do autor: http://lattes.cnpq.br/9484833877562200

Universidade Estadual de Campinas
Instituto de Computação

William Hideki Azana Tustumi

Automatic Liver Tumor Segmentation from Computed Tomography Images Based on 2D and 3D Deep Neural Networks

Segmentação Automática de Tumores do Fígado a Partir de Imagens de Tomografia Computadorizada Baseada em Redes Neurais Profundas 2D e 3D

Campinas, 29 de outubro de 2020

Acknowledgements

- To my family members for their support.
- To the Institute of Computing (IC) of the University of Campinas (UNICAMP) for the infrastructure.
- To the National Council for Scientific and Technological Development (CNPq), process number 168436/2017-5 for the scholarship.
- To my advisors for their supervision and guidance.

Resumo

A segmentação de tumores do fígado é o processo de classificação de voxels entre tumor e tecido saudável realizada a partir de um volume de tomografia computadorizada. A crescente qualidade dos métodos de aquisição de imagens médicas tem permitido a identificação, a localização e o diagnóstico de doenças, evitando-se métodos mais invasivos. Essa análise é vital para decidir sobre o tratamento mais adequado para o paciente. Em especial, a segmentação dos tumores é usada na decisão da viabilidade da extração do tumor e auxilia a especificação do plano operatório. O processo de segmentação das regiões atingidas pelos tumores, quando realizado manualmente, requer tempo e experiência dos especialistas médicos, pois envolve criar uma máscara do tumor para cada uma das fatias da tomografia. Esta tarefa é particularmente desafiadora quando os pacientes estão localizados em regiões carentes e distantes de serviços médicos especializados. Dois tipos de abordagens têm sido tradicionalmente propostas para agilizar e facilitar a segmentação de tomografias, uma completamente automática e outra que se baseia na intervenção humana. Neste trabalho, focamos em técnicas completamente automáticas para a segmentação de tumores do fígado. Apesar das redes neurais convolucionais alcançarem resultados significativos nas áreas de segmentação e classificação de imagens, a segmentação de volumes tomográficos apresenta novos desafios, como a introdução de artefatos oriundos da extração das imagens e número limitado de exemplos para treinamento. Em nossa pesquisa, investigamos o balanceamento entre os recursos computacionais e a qualidade da segmentação. Inicialmente, analisamos o desempenho de várias redes convolucionais e testamos camadas de diferentes redes seguindo o modelo de balanceamento da EfficientNet. A seguir, expandimos as camadas para convoluções tridimensionais e testamos camadas que tratavam as dimensões do volume separadamente. Finalmente, avaliamos o tempo de execução dos nossos modelos em equipamentos com processamento e memória limitados. Embora nossos modelos tenham obtido resultados inferiores em termos de eficácia com relação a outros métodos da literatura, sua execução se mostrou viável em um ambiente computacional restritivo. Os experimentos foram realizados na base de dados *Liver Tumor Segmentation Challenge* (LiTS).

Abstract

Segmentation of liver tumors is the process of voxel classification between tumor and healthy tissue performed from a volume of computed tomography. The increasing quality of medical image acquisition methods has allowed the identification, location and diagnosis of diseases, avoiding very intrusive surgeries. This analysis is vital to decide the most appropriate treatment for the patient. In particular, the segmentation of the tumors is used to decide the feasibility of extracting the tumor and helps to specify the operative plan. The segmentation process of regions affected by tumors, when performed manually, requires time and experience from medical specialists, as it involves creating a tumor mask for each of the slices of the tomography. This task is particularly challenging when patients are located in underserved regions and away from specialized medical services. Two types of approaches have traditionally been proposed to speed up and facilitate the segmentation of CT scans, one completely automatic and the other based on human intervention. In this work, we focus on fully automatic techniques for segmenting liver tumors. Despite the convolutional neural networks achieving significant results in the areas of image segmentation and classification, the segmentation of tomographic volumes presents new challenges, such as the introduction of a dimension of spatial relationships, artifacts from the extraction of images and a limited number of examples for training. In our research, we investigated the trade-off between computational resources and segmentation quality. Initially, we analyzed the performance of several convolutional networks and tested layers of different networks following the EfficientNet balancing model. Next, we expanded the layers for three-dimensional convolutions and tested layers that handled the dimensions of the volume separately. Finally, we evaluated the execution time of our models in equipment with limited processing and memory. Although our models have obtained inferior results in terms of effectiveness when compared to other methods in the literature, their execution proved to be viable in a restrictive computational environment. The experiments were performed on the Liver Tumor Segmentation Challenge (LiTS) database.

List of Figures

List of Tables

List of Abbreviations

2D	Two-Dimensional
3D	Three-Dimensional
ASSD	Average Symmetric Surface Distance
CAD	Computer-aided Diagnosis
CNN	Convolution Neural Network
CPU	Central Processing Unit
CT	Computed Tomography
CXR	Chest X-Ray
DST-ASPP	Depthwise Spatiotemporal Atrous Spatial Pyramid Pooling
DSTS	Depthwise Spatiotemporal Separate
DNN	Deep Neural Network
ELU	Exponential Linear Unit
FC	Fully Connected
FN	False Negative
FP	False Positive
FLOP	Floating-Point Operation
GAN	Generative Adversarial Network
HU	Hounsfield Unit
IC	Institute of Computing
IEEE	Institute of Electrical and Electronics Engineers
ILSVRS	ImageNet Large Scale Visual Recognition Challenge
IRCAD	Institut de Recherche contre Les Cancers de I'Appareil Digestif
ISBI	International Symposium on Biomedical Imaging
LiTS	Liver Tumor Segmentation
LIV	Laboratory of Visual Informatics
LW_HCN	Light-Weight Hybrid Convolutional Network
MAC	Memory Access Cost
MC	Model Cascade
MICCAI	Medical Image Computing and Computer Assisted intervention
MRI	Magnetic Resonance Imaging
MSSD	Maximum Symmetric Surface Distance
NIfTI	Neuroimaging Informatics Technology Initiative
PReLU	Parametric Rectified Linear Unit
ReLU	Rectified Linear Unit
R-CNN	Region-based Convolutional Neural Network
RGB	Red-Green-Blue
RVD	Relative Volume Difference
SE	Squeeze-and-Excitation
SVD	Singular Value Decomposition

SWA	Stochastic Weight Average
TN	True Positive
TP	True Negative
US	Ultrasonography
UNICAMP	University of Campinas
VOE	Volume Overlap Error
VOI	Volume of Interest
WCE	Weighted Cross Entropy

Contents

Chapter 1

Introduction

In this chapter, we describe the automatic liver segmentation problem and provide the motivation that lead us to investigate it. Then, we provide the main questions, objectives and contributions associated with this work. Finally, we provide the text structure.

1.1 Problem Motivation

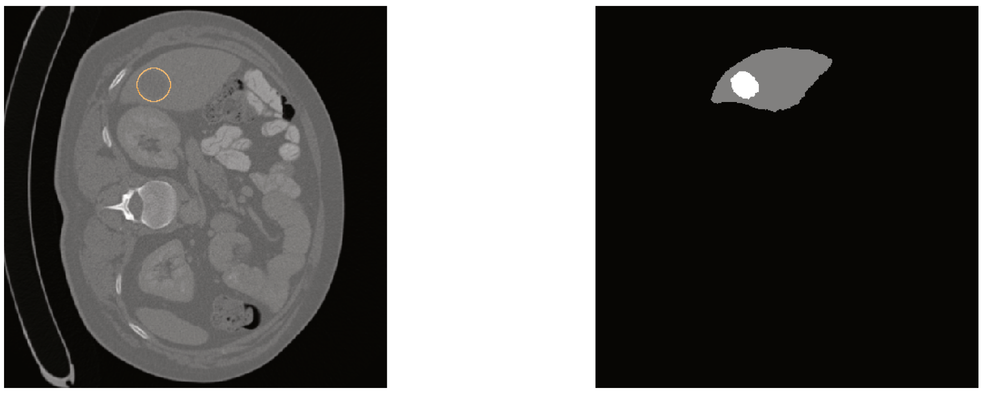
The technology for medical image acquisition has advanced significantly over the last few decades, transitioning from film images to digital images, which allows for higher image resolution and three-dimensional (3D) visualization. A number of methods that allow visualization of internal tissues have been developed over the years. Some of these medical modalities include computed tomography (CT), chest X-Ray (CXR), magnetic resonance imaging (MRI) and ultrasonography (US).

Medical image acquisition techniques have achieved sufficient resolution and reliability, so that it is possible to detect and classify abnormal tissues from the images. Specialists can identify a malignant tumor nodule in a vital organ by means of CT or CXR images without the need to extract the tissue for analysis.

An example of a tumor nodule found in the liver is shown in Figure 1.1. On the left image, we have the CT slice corresponding to the segmentation mask, where the circle highlights the location of the tumor. On the right image, we have a segmentation mask of the liver and tumor, where the gray area illustrates the liver segmentation, whereas the white area illustrates the tumor segmentation.

As of 2018, cancer was the second leading cause of death in the world, with approximately 9.6 million deaths [62]. Although liver cancer is not among the five most common types of cancer, it is the fourth most deadly, with an estimated 782,000 deaths yearly.

A substantial amount of investment has been directed towards the treatment, diagnosis and detection of this type of disease. However, there is a disparity between the resources available in wealthy and poor countries, that is, while 90% of the high-income countries reported availability of treatment services, less than 30% of low-income countries have these services. In addition, approximately 70% of cancer deaths occur in low- and middle-income countries.



(a) image of a tumor nodule (b) segmentation mask

Figure 1.1: Example of an axial CT image and its segmentation mask. Source: LiTS database [37].

1.2 Problem Characterization

Often, primary tumors are located in the abdomen region and spread metastases to the liver. Therefore, the liver and liver lesions are routinely analyzed when breast, pancreas, and colon cancer are diagnosed. The results of the segmentation of the liver tumor images are vital for planning, diagnosis and monitoring of the treatment response time. It is also an important resource for treatments such as thermal percutaneous ablation [48], percutaneous ethanol injection [39], radiotherapy surgical resection [2], among others. However, the number of available specialists is insufficient to cover all demanding patients, especially in isolated areas and underdeveloped countries. Furthermore, the manual segmentation process of a CT scan is costly, time consuming, poorly reproducible, and operator dependent [5].

Due to the aforementioned difficulties, several methods for speeding up the detection and segmentation of liver tumor have been proposed in the literature. They are commonly divided into two groups: computer-aided diagnosis (CAD) and fully automated diagnosis. The CAD approach aims to develop software to help specialists to segment the nodules quickly and precisely, whereas the second approach focuses on developing a program that segments the nodules without human intervention.

Both approaches are based on image characteristics and spatial voxel correlation, but the second presents the additional challenge of having no human intervention to assist the process. Many segmentation algorithms employ user input as a strict constraint on object segmentation. For instance, region growing receives a user input that indicates where the object is in the image. From the given starting area, a homogeneity criterion is applied to determine the foreground and background of the object [72].

As mentioned before, the liver and tumor segmentation process is expensive, so the datasets available to train machine learning models for this problem are relatively limited compared to the datasets used in broader image segmentation problems, where deep learning models have already surpassed human accuracy. In addition to the limited datasets, the region of interest in each CT scan is a fraction of the total volume, which creates an unbalanced class distribution that can bias the model to a certain classification.

Some other challenges arise due to the method used to obtain the CT volumes. Modern CT scanners have achieved resolutions ten times larger than previous models, which increases the heterogeneity of the database. In addition to the difference in technologies used to collect the images, some artifacts can occur due to materials present during the image extraction, which can obscure important tissues or confuse less robust methods for these exceptions.

1.3 Objectives and Contributions

This work aims to develop an automatic method for liver tumor segmentation from 3D CT images using Deep Neural Networks (DNNs). Promising results were already achieved with DNNs in skin lesion segmentation [53, 57, 64], brain tumor segmentation [58, 9, 6], and also liver tumor segmentation [60, 36, 19]. We use the architecture proposed by Han [19] as a baseline, then investigate and extend the applicability of new methods, such as depth and width balance [52], 3D-convolutional layers [44], separable convolutions layers [25], and cascade architecture [35].

In order to achieve our general objective, some specific objectives have to be fulfilled:

- investigation of recent approaches on the subject.
- dataset preparation.
- reproduction of the baseline results.
- extension of the baseline architecture based on approaches that have achieved successful results on related problems.
- performance evaluation of the new model.
- comparison of results with the state of the art.

As a main contribution, we proposed and analyzed an automatic method for liver tumor segmentation based on convolutional neural networks. The approach is based on 2D and 3D convolutions, as well as efficient parameter scaling.

1.4 Research Questions

In this section, we present some research questions that motivate our dissertation proposal:

- Can we develop a model that balances efficiency and effectiveness for the liver tumor segmentation problem?
- How optimized the EfficientNet is for the tumor liver segmentation problem?
- Does our model improve with the use of 3D convolutions?
- Can our model viably run on machines without high-end hardware?

1.5 Text Structure

This text is organized into five chapters and one appendix. In Chapter 2, we introduce some key concepts and briefly describe relevant work available in the literature in the investigated field. In Chapter 3, we describe our deep learning model, along with the reasoning behind each architectural decision. In Chapter 4, we report our experimental setup and analyze our results, along with a comparison with the state of the art in the field. In Chapter 5, we present some concluding remarks and directions for future work. In Appendix A, we provide complementary implementation details.

Chapter 2

Background

In this chapter, we briefly describe basic concepts related to computed tomographic volume acquisition, image preprocessing and segmentation, data augmentation, and neural networks, which are necessary to understand the subsequent sections of the research work. Then, we provide a brief explanation of deep neural network architectures, some relevant results found in the literature and how they are related to our work.

2.1 Concepts and Techniques

In this section, we describe some relevant concepts and techniques related to the topic under investigation.

2.1.1 Segmentation

Image segmentation is the process of delimiting object boundaries in an image [16]. The segmentation problem is an expansion of the localization and detection problems, since their responses can be extracted directly from the segmentation solution.

Segmentation typically combines different image features to achieve a successful solution. Many segmentation algorithms search for pattern discontinuations and pixel similarities to find object boundaries. However, most of them are dependent on the image type and characteristics, which makes it difficult to reuse solutions from different problems.

The concept of image segmentation can be extended to volume segmentation, where the object delimitation is not defined by the boundaries of pixels, but by the boundaries of voxels.

2.1.2 Computed Tomography

Hounsfield [23] developed the Computed Tomography (CT) scanning technology. The process generates a faithful 3D distribution of X-Ray attenuation values per volume unit, which generates the inverse Radon transform from projections in an axial slice. Modern scanners that use multi-slice technology can achieve less than 1 mm per slice depending on the level of detail needed for the region of interest [55], which is far better than the slice granularity achieved in the early days of CT when the thickness was normally above 5 mm.

The attenuation values from the CT volume are normalized to the Hounsfield Unit (HU) scale to make them independent from the X-ray energy output. The normalization is expressed in Equation (2.1).

$$\mathrm{HU}(\mu) = 1000\frac{\mu - \mu_{water}}{\mu_{water} - \mu_{air}} \tag{2.1}$$

where μ_{air} and μ_{water} are the linear attenuation coefficients of air and water, respectively.

The values of HU usually range from -1000 to 3000, which can be represented in two bytes. The attenuation values of air, fat, and water are considerably different, whereas the difference between distinct soft tissues is minimal, which makes their classification more difficult. Table 2.1 shows the HU values for different materials and body tissues.

Air	Fat	Water	Blood	Muscle	White Matter	Gray Matter	Bone
-1000	-100	0	30-45	40	20-30	37-45	>150

Table 2.1: Values of Hounsfield Unit (HU) for different materials and body tissues. Values extracted from [55].

2.1.3 Filters

The most common filters used in image segmentation are median, mean and Gaussian filters [16]. These filters aim to attenuate the occurrence of noise while maintaining sharp features, such as corners and edges, that can be blurred from the image.

When working with computer tomography, it is common to ignore HU values outside a certain band, as they do not correspond to the type of material that causes the problem under investigation.

2.1.4 Deep Neural Networks

Deep Neural Network is a subset of machine learning methods, which approximates complex feature relations through multiple layers of non-linear units. Each unit combines the output of many units from previous layers to compute an activation function. This structure was vaguely inspired by how neurons work. The layered structure allows computers to create a hierarchical representation of features using examples and accumulate knowledge of the problem.

The history of neural networks traces back to 1943, when McCulloch and Pitts developed a neural network using circuits to describe how neurons could work. Later, in 1960, Widrow solved the phone line noise problem using neural networks. However, the perceptron model could only compute weights for a single layer, which restricts its descriptive power. In 1986, Rumelhart et al. introduced backpropagation as a gradient evaluator to calculate the magnitude and direction of each step update, which resolved the limited number of trainable layers.

Lately, neural networks have become popular due to the advance of computational power and storage space, as well as the expansion of data available for training. Three architectures that leverage both aspects are: (i) AlexNet [34], which achieved unprecedented accuracy on the ImageNet [12] classification contest in 2012; (ii) ResNet [21], which achieved higher than human-level accuracy on the ImageNet; and FixEfficientNet [56], which is considered the state of the art for image classification on the ImageNet dataset.

Convolutional Neural Networks (CNNs) are the most prevalent type of DNNs for image classification, image segmentation and feature extraction [65, 71, 61, 66, 54]. Researchers devised the concept of CNN from studies of the visual cortex [15]. However, the use of CNNs is not limited to the image analysis field. For instance, the method achieved great success in the area of speech recognition [1, 4].

Some crucial drawbacks of DNNs include the computational time required to train the network, vanishing/exploding gradients and the necessity to use large amounts of training data. The first one is mitigated by the use of Graphical Processing Units (GPUs) with highly optimized matrix/vector functions. The second problem is partially addressed with Rectified Linear Units (ReLU) [45] and batch normalization [28]. Finally, the use of data augmentation techniques partially solves the third drawback.

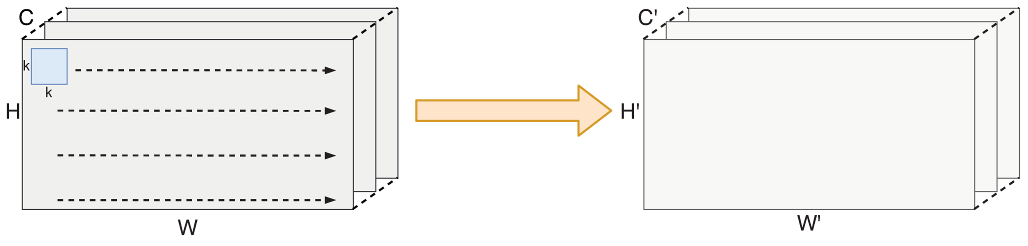
The development of ReLU and batch normalization techniques allowed the construction of deeper architectures, which improved model performance, since representation depth is beneficial for classification accuracy [51]. The large size of current models made impractical to start an architecture from scratch because most models that achieve state-of-the-art results may require a few weeks to train from randomly initialized weights. A common practice is to use an efficient pre-trained model, modify a few of its layers to adapt it to a specific problem, and then fine tune (adjust) the model using samples from the database of the target problem.

2.1.5 3D Convolution

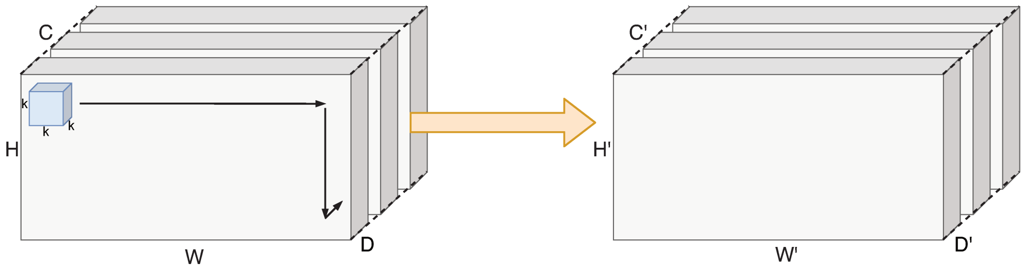
3D convolutions are a natural expansion of 2D convolutions for volumetric data. While 2D-convolution filters have 4 dimensions (height, width, channel input and channel output), 3D convolution filters have 5 dimensions, where an additional depth dimension is added after the width. This new dimension adds a z-axis to the filter movement. The added dimension makes it straightforward to capture the relationship between features on the newly created axis, but it also increases the computational cos of the model. Therefore, many models opt to use a 2.5D approach that uses volumetric data with 2D convolutions, where the depth dimension is mapped to the input filters. Figure 2.1 depicts the differences between 2D convolution and 3D convolution.

2.1.6 Separable Convolution

Separable convolutions are divided into two groups: spatial and depthwise. The first occurs in the spatial dimension; in the case of images, it separates width and height of the kernel, as in Figure 2.2. The second type separates the convolution in spatial aggregation and feature combination; in the case of images, the first filter has its original width and height, whereas the second filter is a 1×1 filter that transforms the number of features into the desired output. Figure 2.3 illustrates the convolution process.



(a) 2D convolution



(b) 3D convolution

Figure 2.1: Example of 2D convolution 2.1a and 3D convolution 2.1b diagrams. The traced arrows in the 2D convolution show the filter movement through the input. The filter in a 3D convolution must follow a similar path from the 2D convolution. However, since it has an additional dimension to move, it repeats the 2D-movement for each entry on the z-axis. Figure 2.1b indicates the dimensions by which the filter will move without the complete path to avoid cluttering the image. H, W, D and C denote the input: width, height, depth and channels, respectively, while H', W', D' and C' are their output counterparts. Source: Elaborated by the author.

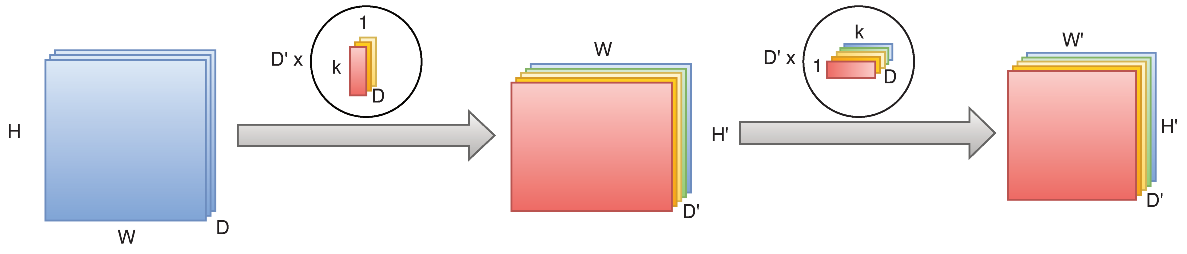


Figure 2.2: Diagram of a spatial separable convolution. The values on the left side of the circle represent how many repetitions of the kernel are used in the convolution. Source: Elaborated by the author.

The depthwise separable convolutions are the basis for more efficient architectures, because they reduce the number of computations and parameters without losing the descriptive power, which is not the case with spatial separable convolutions, where the only type of kernels that they can represent are those composed of separable filters.

The depthwise convolution separates the convolution into a group convolution and a pointwise convolution. The group convolution maintains the number of feature maps and transforms the spatial dimension. The pointwise convolution uses a 1×1 kernel with stride 1 that transforms the number of channels, but maintains the spatial dimension.

From Table 2.2, it is possible to conclude that the conventional convolution scales worse and the depthwise convolution scales better than the spatial convolution for values

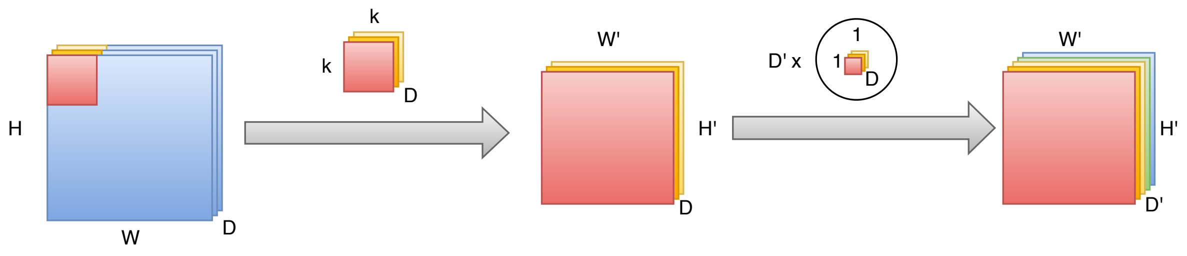


Figure 2.3: Diagram of a depthwise separable convolution. The first convolution separates the input into D layers, where each layer is convolved with a layer of the kernel. In the image, this operation is represented by its colors. The value on the left side of the circle represents how many repetitions of the kernel are used in the pointwise convolution. Source: Elaborated by the author.

Convolution	Number of Parameters
Conventional	$k^2 \times c_i * c_o$
Spatial Separable	$2 \times k \times c_i \times c_o$
Depthwise Separable	$k^2 \times c_i + c_i \times c_o$

Table 2.2: Expressions for the number of parameters from different 2D convolutions, where k is the size of the kernel, c_i is the number of input channels, and c_o is the number of output channels.

small values of k, which is the usual.

2.1.7 Residual Layer

Residual Network (ResNet) is considered the state of the art for image classification. It won the first place on the ImageNet Large Scale Visual Recognition Challenge 2015 (ILSVRC'2015) with an error percentage of 3.57% [21]. The complete architecture is a CNN composed of 152 layers, with 11.3 billion Floating Point Operations (FLOPs).

ResNet achieved its accuracy improvement by applying the concept of residual learning. Figure 2.4 shows a diagram of two residual layers. It is worth observing that the link connection from input x to the combinator $\oplus$ acts as a shortcut connection. The shortcut addition comes from the hypothesis that a residual function $F(x) - x$ is easier to be optimized than the unreferenced function $F(x)$. For example, if the identity function is the optimal solution, it is easier to simply reduce the weights from the residual layer than creating a stack of layers that fit the identity function.

2.1.8 Squeeze-and-Excitation

Squeeze-and-Excitation (SE) module aims to improve inter-channel representations by modeling the relationships between channels explicitly. It achieves that by creating a side route that acts as a self-attention for each channel.

Figure 2.5 illustrates the diagram for the SE module. The module is composed of a pooling layer that reduces the image space dimensionality to 1, while maintaining the

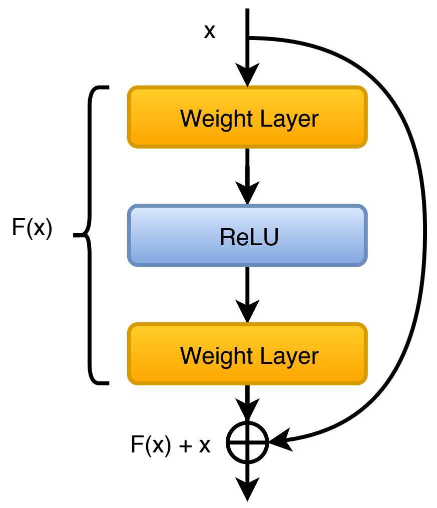


Figure 2.4: Residual layer diagram based on the work developed by He et al. [21]. Source: Elaborated by the author.

same number of channels. The output passes through two Fully Connected (FC) layers with a ReLU activation function between them. The first FC layer acts as a bottleneck, reducing the number of channels by an r factor, whereas the second layer expands it back to its original size. A sigmoid function follows the second layer, which is then combined by means of a broadcast multiplication with the main flow of the model. The SE module can be attached at the output of most layers, which allows it to be integrated into standard architectures.

2.1.9 Depthwise Spatiotemporal Separate

The Depthwise Spatiotemporal Separate (DSTS) module, proposed by Zhang et al. [68], separates 3D convolutions into two branches: a temporal and a spatial one. Both branches use depthwise convolutions, but with different filters. The temporal side has filters of size $3\times1\times1$ and the spatial side uses filters of size $1\times3\times3$. A pointwise convolution combines the features after a concatenation.

Figure 2.6 shows a diagram of the DSTS module. In addition to forcing the network to learn different relationships at each branch, the separation has the added benefit of reducing the number of parameters and the computational cost. The number of parameters is reduced from $27c^2$ with a normal 3D convolution with kernel size 3 to $c^2 + 12c$. The variable c represents the number of channels in that layer.

2.1.10 U-Net

U-Net is a CNN created to segment biomedical images that uses its architecture and training strategy to achieve a segmentation performance higher than the sliding-window convolutional network. It also relies heavily on data augmentation and, therefore, can be trained on a small dataset [47]. The network also runs very fast, such that it can segment an image of 512×512 in less than one second using an NVIDIA Titan GPU.

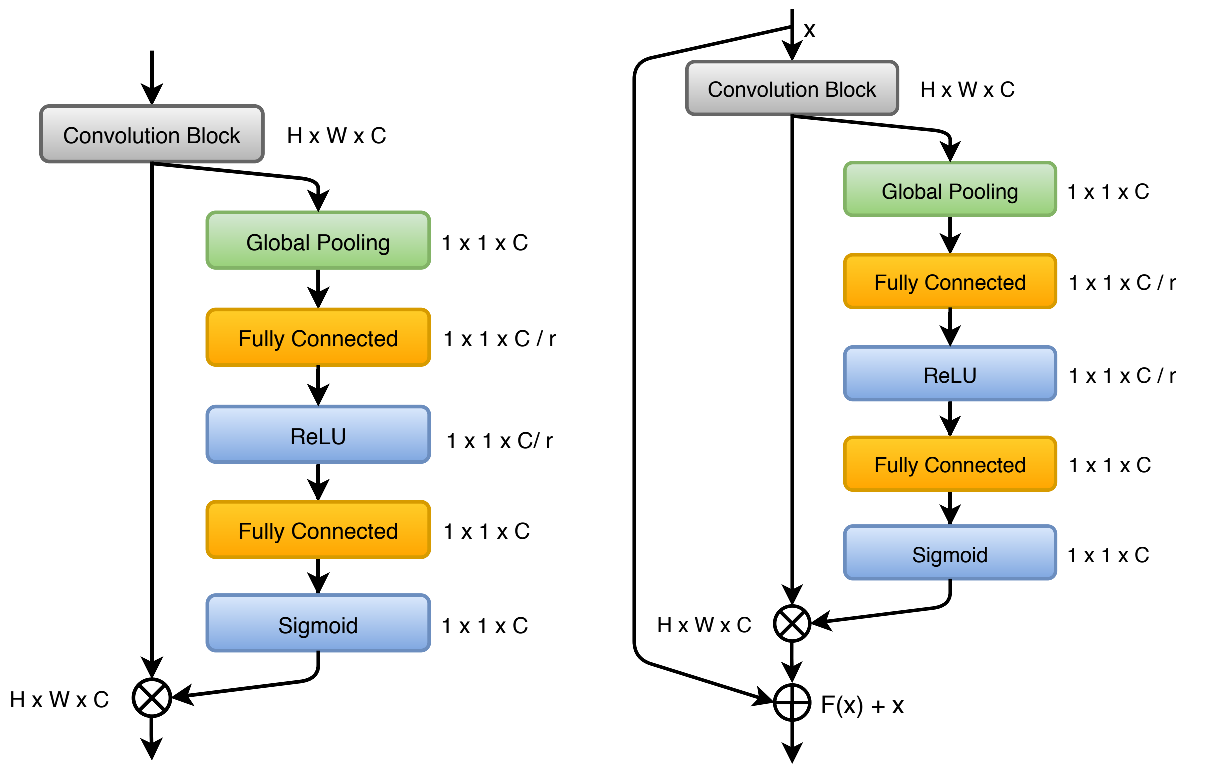


Figure 2.5: Squeeze-and-Excitation module diagram. The original module is shown on the left. On the right, the module is shown with a residual connection. Source: Elaborated by the author.

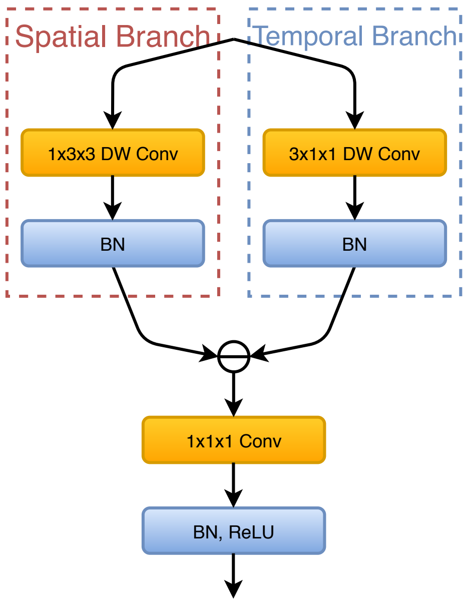


Figure 2.6: Diagram of a Depthwise Spatio-Temporal Separate module. Source: Elaborated by the author.

Figure 2.7 shows a diagram of the U-Net model. The left side of the architecture shrinks the image dimensions and increases its depth, whereas the right side expands the image to create the segmentation map. The horizontal blue arrows represent shortcuts on the model, which combine previous layers from the encoder to the decoder. The layers are combined using a concatenation operation. The difference between the dimension of the original image and the segmentation map is caused by unpadded convolutions that reduce the output dimension at each step.

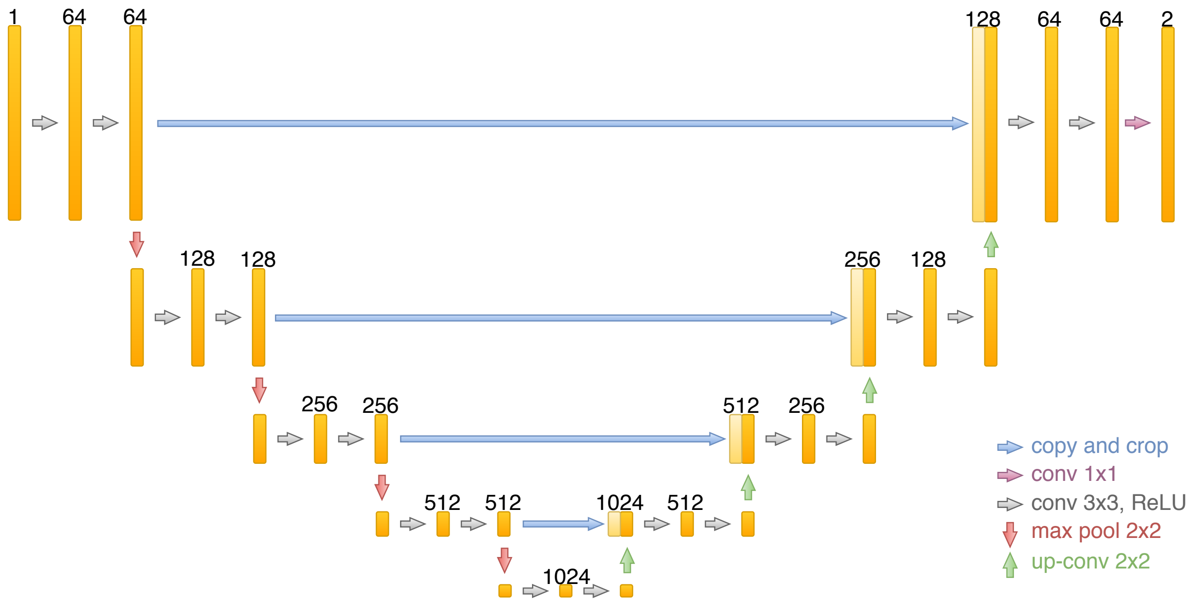


Figure 2.7: Diagram of the U-Net segmentation architecture. Source: Elaborated by the author.

A variant of the U-Net, named dense U-Net, was used by Li et al. [36] for liver tumor segmentation, achieving a global Dice score of 0.829 on the Liver Tumor Segmentation Challenge 2017 (LiTS) dataset [37]. The model used the ResNet architecture for feature extraction, instead of feeding the output to the dense U-Net to perform segmentation.

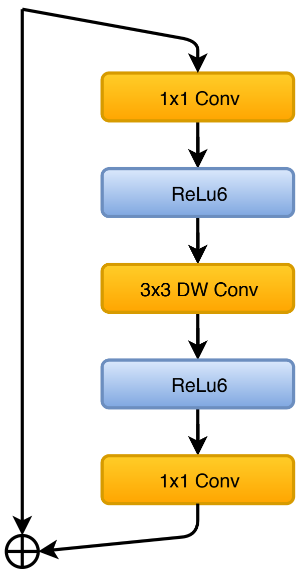
2.1.11 MobileNet

The MobileNet family of architectures passed by three iterations. The first iteration created an efficient architecture based on depthwise separable convolutions and introduced two global hyper-parameters to control the size of the network: the first one controls the number of channels in each layer and the second controls the input image resolution [25].

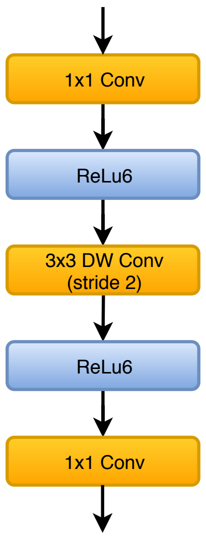
The second iteration expanded the work of MobileNet-V1 by proposing reverse residual structure, and modifying the architecture to solve segmentation problems, where the previous work focused on object recognition and object detection. Figure 2.8 shows the diagram of the reverse residual block used as building block for MobileNet-V2 [50].

Reverse residual blocks use shortcut connections for the same reason as classical residual blocks, as it facilitates gradient propagation between layers. However, the connection is between bottleneck layers because the bottlenecks should contain all the information considered important for the model. An additional benefit of connecting bottleneck layers is memory efficiency, since less information needs to be in memory at the same time.

The third version of the MobileNet added squeeze-and-excitation [26] (explained in Section 2.1.8) to its building block and introduced the h-swish activation function, an approximation of the swish function that allows for faster computation (more details in Section 2.2.5).



(a) Inverted residual block



(b) Inverted residual downsampling block

Figure 2.8: Inverted residual block diagrams from MobileNet-V2 [50]. Source: Elaborated by the author.

2.1.12 ShuffleNet

The ShuffleNet architecture was developed for very small computational constrains and has two iterations. The first introduced the pointwise group convolution and channel shuffle operation [70], whereas the second version dropped the pointwise group convolution, introduced a split channel layer, and focused its analysis on performance implementation gain instead of reducing the number of operations and memory usage [41].

The group pointwise convolution reduces the computation necessary for the pointwise convolution of the depthwise separable convolution, but it limits the connections of the input channels and the output channels. To address this limitation, the authors proposed the use of a shuffle layer that aggregates channels from different groups of the pointwise operation. The channel shuffle layer interlaces the features to output a homogeneous distribution. In Figure 2.9, we show an example with three groups.

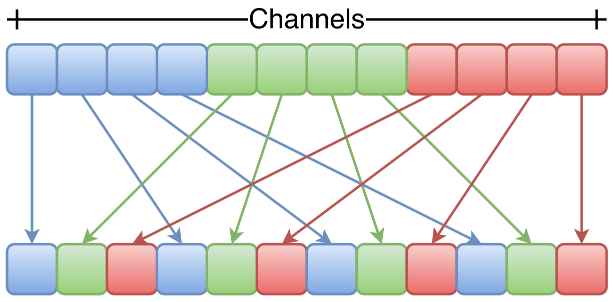


Figure 2.9: Channel shuffle example with three groups, where each group is color coded. The arrows point to the end position of each entry. Source: Elaborated by the author.

The second iteration of the ShuffleNet was based on the following guidelines: G1- equal channel width minimizes memory access cost (MAC); G2- excessive group convolution increases MAC; G3- network fragmentation reduces the degree of parallelism; and G4- elementwise operation is not negligible. The pointwise group convolution was dropped from the previous iteration because of G2. In its place, a split channel layer was introduced that separates the data flow into two branches: one branch remains the same to respect G3, while the other passes by a depthwise separable convolution. Then, the two branches are concatenated and passed through a channel shuffle layer. In Figure 2.10, we show a diagram of the building blocks of both versions of ShuffleNet.

It is worth noting that the ShuffleNet-V2 blocks maintain the same number of channels in the depthwise convolution to follow G1, and the channel increment comes from the absence of a channel split layer in the downsampling block. The last elementwise activation function was moved to before the concatenation operation to comply with G4.

2.1.13 EfficientNet

Tan and Le [52] observed that models that balanced their depth, width and resolution reached a higher accuracy for the same computational cost than models that favored only one of these coefficients. With that in mind, they proposed a compound scaling method that balances these three coefficients, given a target computational constraint. In their experiments, they chose the number of FLOPs to determine the model size and used the accuracy on ImageNet to measure its performance.

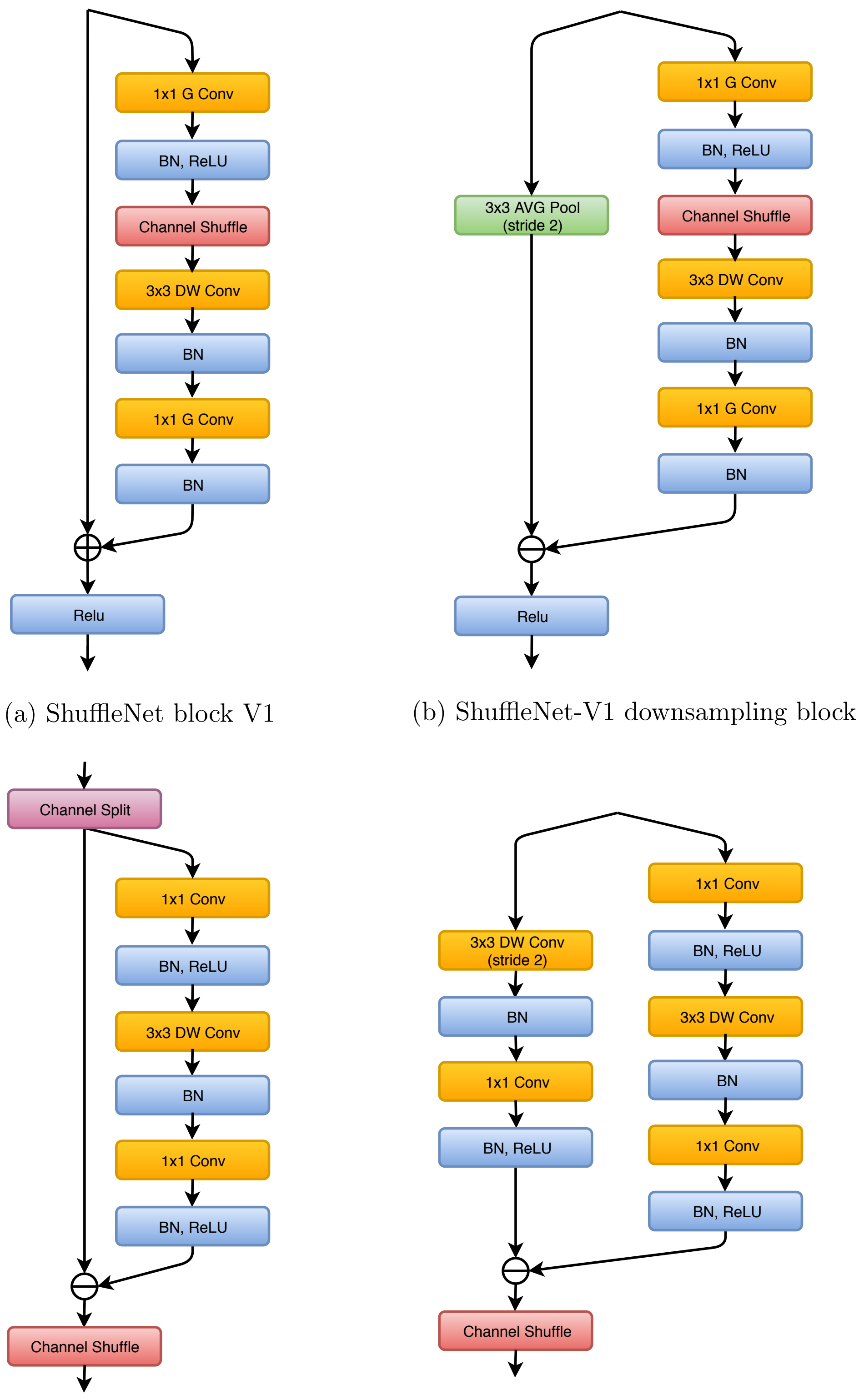
The baseline coefficients were discovered through a grid search with a target FLOPs of 400M. Notably, the scaling factor of the network depth is linear, while the scaling factor of width and resolution is quadratic. In other words, doubling the network depth doubles the number of FLOPs, while doubling either the width or the resolution quadruples the number of FLOPs. The base model uses the mobile inverted bottleneck from [25] coupled with squeeze-and-excitation module from [26] as a building block. With the baseline model and the compounding scaling, they created a family of network architectures, named EfficientNet-BX, ranging from B0 to B7, increasing with the target FLOPs.

2.2 Model Cascade

Model Cascade (MC) implements multiple stages, each one with different classifiers. This approach increases speed and accuracy, because earlier stages can classify voxels easier, while the later stages focus on the hardest voxels (boundary voxels). Each stage is trained separately, where previous stage's weights are frozen, while the next stage is trained, so that it does not interfere with the current stage's training.

2.2.1 Deep Layer Cascade

Li et al. [35] proposed a model that benefits from the multi-stage focus from the model cascade. However, this approach discards the need to train each stage separately. This was achieved by creating a mask at each stage that blocks the features passed to the next.



(a) ShuffleNet block V1 (b) ShuffleNet-V1 downsampling block

(c) ShuffleNet-V2 block (d) ShuffleNet-V2 downsampling block

Figure 2.10: 2.10a shows the diagram of the basic ShuffleNet-V1 block, 2.10b shows the downsampling variant of 2.10a from [70], 2.10c is the basic block diagram of ShuffleNet-V2 block, and 2.10d is its downsampling variant from [41]. Source: Elaborated by the author.

Figure 2.11 shows a cascade architecture diagram. The first stage creates a probability mask $L1$ and a feature block A. Values in $L1$ above the confidence threshold are set to 0, otherwise to 1. The resulting mask is applied to A. This focuses the network attention on the areas where the classification is unsure. The second stage works analogously to the first stage with B as the feature block and $L2$ as the probability mask. The last stage generates a probability mask $L3$ that, combined with $L1$ and $L2$, creates the final prediction.

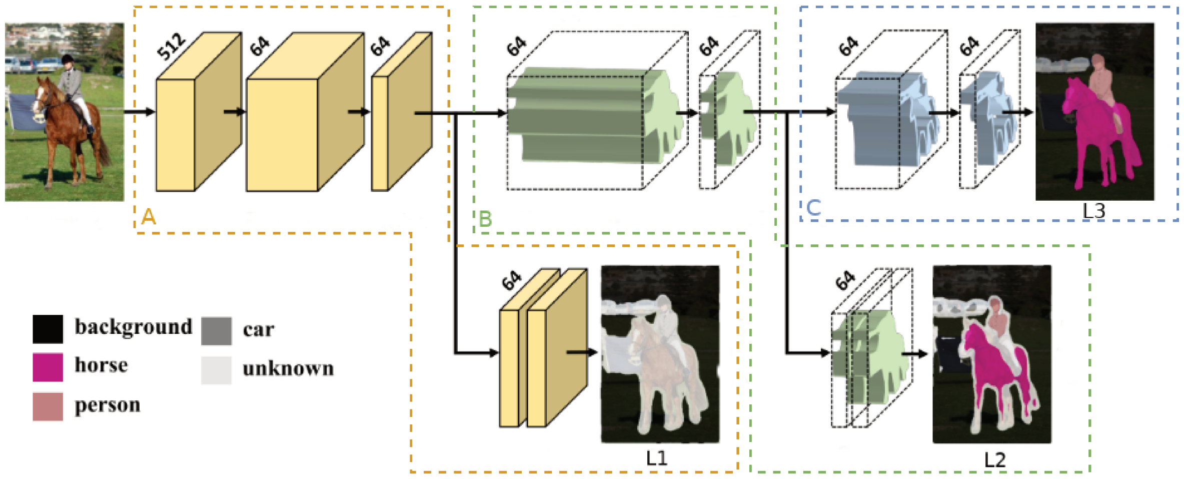


Figure 2.11: Diagram of the cascade architecture. Source: Edited from [35].

2.2.2 Data Augmentation

A relevant aspect of deep learning is the capacity of a model to generalize its knowledge to new samples of a problem. To achieve a generalized model, it is important to use a sizable and diverse training database, since the model can only understand the relations between features of a problem through examples; therefore, data augmentation is widely used to expand the training data.

Conventional data augmentation techniques add rotation, translation, scaling and intensity transformations to existing samples [40]. More sophisticated methods attempt to expand the possible types of transformations that can be used to generate samples, for instance, the Generative Adversarial Networks (GANs) [3]. Data augmentation must be carefully applied so that new samples cannot incorporate incorrect or misleading information. In this case, the participation of medical specialists would be beneficial for the evaluation of the data augmentation stage.

2.2.3 Transfer Learning

Transfer learning is a common technique employed when the database size of the target problem is insufficient or when it is desired to improve the model convergence time. The method consists of using a larger dataset from a similar problem to train the model first, then use the original dataset to fine-tune the model to the specific problem.

The idea behind transfer learning is that some information learned from the first dataset will be useful for solving the target problem and may no be learned. The use of transfer learning is particularly common in medical image recognition tasks, since the number of images is typically small compared to the number of samples in other categories of images.

2.2.4 Model Compression

The continuous development of architectures leads to accuracy improvements; however, the resources needed to train the model are largely increased. Due to that, efforts have been directed to simplifying the models by scaling down the number of variables or decreasing variable representation.

In 2014, Denton et al. [13] applied a singular value decomposition (SVD) to a CNN model to achieve model compression with minimal accuracy degradation. One year later, Han et al. [18] used thresholds to prune the weights of their architecture and improve the data compression with sparse matrix representation. In the same year, they used Huffman encoding to extend their work.

The search for smaller models with similar representation power has also generated promising results. SqueezeNet [27] is one of the most notorious models, which achieved accuracy comparable with AlexNet [34], while using 50 times fewer parameters. Following this result, MobileNet [25], ShuffleNet [70] and EfficientNet [52] further improved the field with smaller architectures with impressive results.

2.2.5 Activation Function

The capacity of model description depends on the non-linearities provided by its activation functions. The ReLU function is the most common activation function due to its simplicity and speed. The ReLU function has the additional benefit of mitigating vanishing and exploding gradient, since its gradients are either zero or one. However, the ReLU function can suffer from inactive neurons (dead neurons) [11], which occur when the ReLU function outputs the same result for any input in the dataset space. This can only happen in the negative portion of the function where all values are set to zero, thus cutting off the gradient flow.

Many activation functions have been proposed to improve the performance of the ReLU, such as ELU [11], LeakyReLU [42], PReLU [20], ReLU6 [33], swish [46] and h-swish [24]. The most significant for our work are ReLU, ReLU6, swish and h-swish.

ReLU6 limits the positive value of the ReLU to six, which the authors believe that forced the model to distribute the knowledge to features that are close to the activation limit. The swish function showed improved or equal results compared to the ReLU function in a wide range of tests that none of the other activation functions had achieved so far. However, the swish functions use sigmoid, a very costly function, especially when compared to the simple ReLU. The h-swish function is an approximation of the swish function that replaces the sigmoid with a ReLU6 and a division.

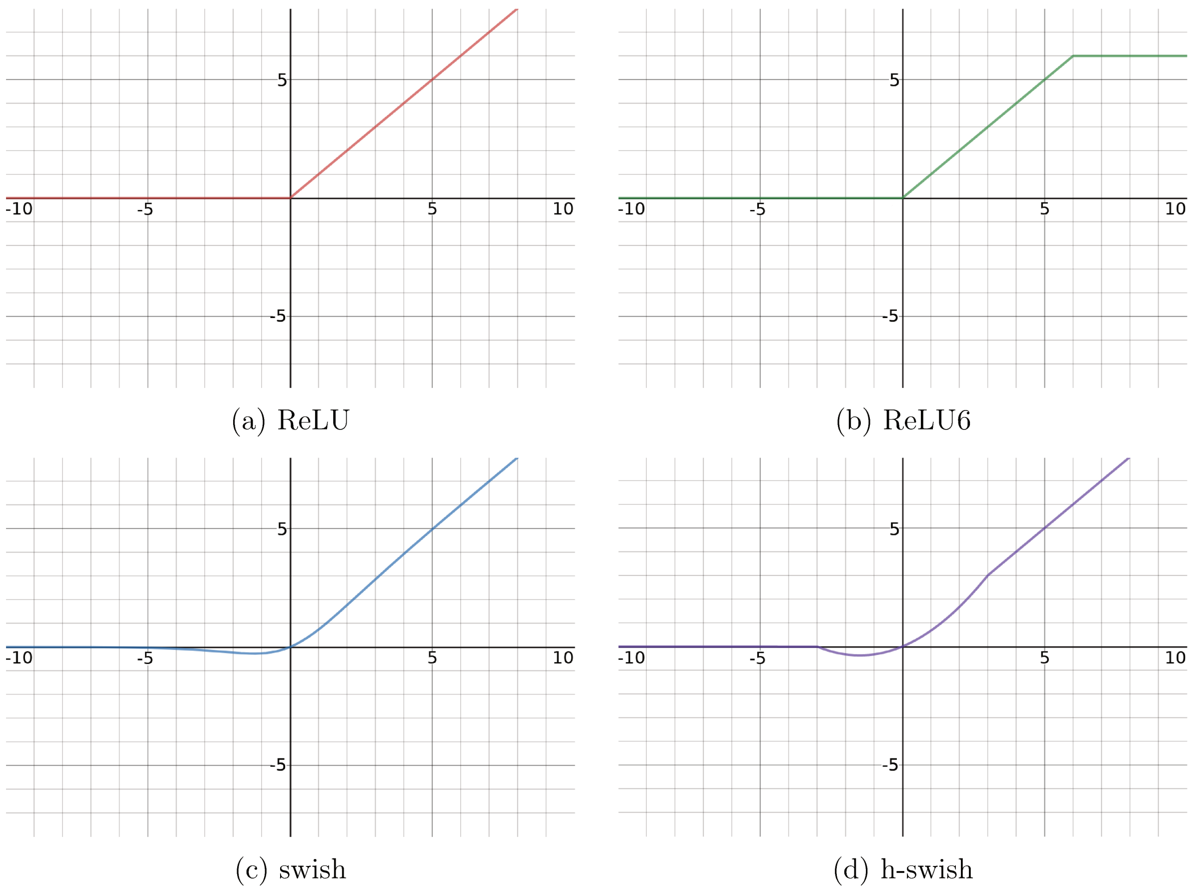
We compare all three functions and the ReLU in Figure 2.12. It is possible to notice that both swish and h-swish functions allow a small portion of the negative values of x to pass, while maintaining zero for large negative values. We show the formulas of all four functions in Equations (2.2) to 2.5 for completion sake.

$$\mathrm{ReLU}(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases} \tag{2.2}$$

$$\mathrm{ReLU6}(x) = \begin{cases} 6, & \text{if } x > 6 \\ x, & \text{if } 0 < x < 6 \\ 0, & \text{if } x \leq 0 \end{cases} \tag{2.3}$$

$$\mathrm{swish}(x) = \frac{x}{1 + e^{-x}} \tag{2.4}$$

$$\text{h-swish}(x) = x \times \frac{\mathrm{ReLU}(x) - 3}{6} \tag{2.5}$$



(a) ReLU

(b) ReLU6

(c) swish

(d) h-swish

Figure 2.12: Graphs of activation functions with center at (0,0). Source: Edited from `desmos.com`.

2.2.6 Loss Function

A loss function guides how the weights within the learning method are updated. It can be formulated either from the difference between expected output and predicted output or based on a desired feature that the output must present. The former can only be used in supervised training, whereas the later is used in unsupervised training or as a complementary loss to a supervised training. The most common loss functions used for image segmentation architectures are WCE (Weighted Cross Entropy), Dice distance and Jaccard Distance.

The WCE function treats each voxel as a classification problem and penalizes each error according to the correct class weight, these weights are hyperparameters that mitigate the effect of class imbalance present in the data. Since the weights are extracted from the class distribution present in the trainig set, they are specifically optimized for the training examples. In Equation (2.6), N is the number of voxels, C is the number of classes w_i^c is the weight related to that voxel, y_i^c is the correct voxel classification in one-hot encoding and $\hat{y}_i^c$ is the predicted probability for that voxel.

$$\text{WCE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} w_i^c y_i^c \log \hat{y}_i^c \tag{2.6}$$

The Dice distance and Jaccard distance are very similar, as both originate from metrics used to measure the segmentation quality. The Dice distance is based on the Sorensen-Dice Coefficient, whereas the Jaccard distance is based on the Jaccard index. The Dice coefficient is equal to twice the size of the intersection between two sets divided by the sum of the elements in both sets. Equation (2.7) shows the Dice equation with Y as one set and $\hat{Y}$ as the other set.

$$\text{Dice} = \frac{2|Y \cap \hat{Y}|}{|Y| + |\hat{Y}|} \tag{2.7}$$

In order to use the Dice equation as a loss function for supervised training of a neural network, it is necessary to adapt its formula, since the function was proposed to work with discrete data. Equation (2.8) shows the Dice loss function adapted for a minimization problem that allows gradient backpropagation.

$$\text{Dice distance} = 1 - \frac{2 \times \sum_{i=1}^{N} y_i \times \hat{y}_i}{\sum_{i=1}^{N} y_i^2 + \hat{y}_i^2} \tag{2.8}$$

As in Equation (2.8), N is the number of voxels, y_i is a voxel from the ground-truth mask, where y_i is equal to 1 when the voxel belongs to the object and 0 otherwise, and $\hat{y}_i$ is a voxel with the predicted probability of belonging to the object. The intersection operand was replaced by a multiplication that allows derivation. The size operator was replaced by the sum of the values of all voxels.

The Jaccard function replaces the divisor of the Dice function by a union of the sets Y and $\hat{Y}$.

$$\text{Jaccard} = \frac{|Y \cap \hat{Y}|}{|Y \cup \hat{Y}|} \tag{2.9}$$

The Jaccard index, similar to the Dice coefficient, has to be adapted in order to fit as a loss function. Equation (2.10) shows the adapted function.

$$\text{Jaccard distance} = 1 - \frac{\sum_{i=1}^{N} y_i \times \hat{y}_i}{\sum_{i=1}^{N} y_i^2 + \sum_{i=1}^{N} \hat{y}_i^2 - \sum_{i=1}^{N} y_i \times \hat{y}_i} \tag{2.10}$$

The new adaptation necessary for the Jaccard distance is the substitution of the union operator that is used in the divisor part of the fraction. The union is separated in two parts: the first one calculates the sum of the sizes of both sets and the second part subtracts the two sets.

The Dice coefficient can also be adapted to work with multi-class segmentation, by calculating the mean dice value for each class using one-hot encoding, which results in Equation (2.11).

$$\text{Multi-Dice} = 1 - \frac{1}{|C|} \sum_{c \in C} \frac{2 \sum_{i=1}^{N} y_{ci} \times \hat{y_{ci}}}{\sum_{i=1}^{N} y_{ci}^2 + \hat{y}_{ci}^2} \tag{2.11}$$

2.3 Related Work

In this section, we describe some relevant machine learning methods used to address the liver tumor segmentation problem.

Han [19] won the LiTS challenge at the 2017 Institute of Electrical and Electronics Engineers (IEEE) International Symposium on Biomedical Imaging (ISBI). The proposed solution uses two FCNNs with long-range shortcut connections from U-Net [47] and short-residual connections from ResNet [21]. The first network creates a coarse segmentation of the liver, which is used to limit the number of slices employed in the second network to the slices within the liver segmentation. The second network then creates a refined segmentation of the liver and the lesion segmentation.

The authors decided to use 2D convolution to reduce computational and memory resources to leverage the z-axis relation between voxels. Five slices are used to predict the middle slice. This method is called 2.5D convolution network, since it still considers volume relations, but uses 2D filters. The authors addressed the difference in spacing between slices by undersampling slices so that each volume had a spacing of 1mm$\times$1mm$\times$2.5mm. The undersampling method was only used in the first network, whereas the second network receives the input with the original spacing, since crucial information could be lost with the undersampling.

The networks were trained using stochastic gradient descent through 50 epochs and took 4 days for each network on a single NVIDIA TitanX GPU. Each prediction slice required 0.2s to be processed. The code was implemented in Caffe package. The model had 24M parameters. The solution obtained a global Dice coefficient of 0.67, volume overlap error (VOE) score of 0.45, average symmetric surface distance (ASSD) of 6.660

and maximum symmetric surface distance (MSSD) of 57.930 on the LiTS benchmark for lesion tumor segmentation.

Yuan [67] achieved the fifth place in different metrics at 2017 International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI) on the LiTS'2017 benchmark. The proposed solution has three steps: liver localization, liver segmentation and tumor segmentation. The liver localization uses a small fully convolutional neural network (FCNN) with 19 layers and 230,129K parameters. The localization step uses slices of 128×128 pixels and undersamples the slices to 3mm thickness to create a coarse segmentation that is used in the next step. A post-processing applies a threshold of 0.5 with connected components that finds and selects the large components.

The liver segmentation step creates a volume of interest (VOI) by expanding 10 voxels in each direction of the coarse segmentation of the previous step. The sampling frequency is increased, so that the space inter-slices is 2mm. Moreover, the resolution is expanded to 256×256×256. The network used in this step is more sophisticated with 5M parameters and 29 layers.

The tumor segmentation step uses the same network as the liver segmentation stage; however, the original slice spacing is used to avoid missing small lesions due to image blurring. During training, only the slice with tumors were used for training in order to reduce computational time. As post-processing, a filter was used to discard any tumor outside the predicted liver mask. The final liver tumor mask is the result of a bagging-like ensemble of six models. The loss function used was the Jaccard distance.

The networks were trained using Adam optimizer and required 1.57 days for liver localization and segmentation, whereas required 8.518 days for tumor segmentation on a single NVIDIA GTX 1060 GPU. Each test case took 33s for prediction on average. The model was implemented in Theano and used 36M parameters. The solution obtained a global Dice coefficient of 0.82, a Dice per case of 0.657, a VOE score of 0.378, an ASSD of 1.151 and an MSSD of 6.269 on the LiTS'2017 benchmark for lesion tumor segmentation.

Li et al. [36] improved the segmentation performance after the end of the 2017 MICCAI challenge on the LiTS'2017 benchmark. The proposed method used traditional 2D and 3D convolutions with a hybrid feature fusion layer to leverage both intra- and inter- slice features. The basic building blocks of the network used densely connected convolutions to gradually expand the number of channels and improve information flow between all closely connected layers. The model created a coarse liver segmentation using a 2D ResNet, which is used as input to the 2D DenseUNet and 3D DenseUNet. Both DenseUNets created feature maps that were combined and fed to the hybrid feature fusion that generates the final liver and tumor segmentation. The adopted loss function was weighted cross entropy as loss function; the class weights were determined through an empirical test.

The networks were trained using stochastic gradient descent and took 21 hours for the 2D DenseUNet and 9 hours for the 3D DenseUNet on 2 NVIDIA Xp GPUs. Each test case took between 30 and 200 seconds. The code was implemented in TensorFlow and the model has 114M parameters, achieving a global Dice coefficient of 0.82.4 and a Dice per case of 0.72.2 on the LiTS'2017 benchmark.

Zhang et al. [68] proposed an efficient hybrid convolutional model based on depthwise separable convolutions and spatial separable convolutions. Initially, they introduced a spatial and temporal factorization module to replace the traditional 3D convolutions. This module is explained in Section 2.1.9. The model was labeled hybrid since it uses 2D convolutions to extract low-level features, then uses 3D convolutions at deeper levels to reduce computation, while retaining the spatial connections from the 3D convolutions. The output block from the encoder is a depth spatio-temporal atrous pyramid polling (DST-ASPP) adapted from [7] and [8]. The DST-ASPP structure is used to combine multi-range features to calculate the segmentation output. The loss function used was a combination of cross entropy and multi-label Dice.

The network was trained with Adam optimizer and took 24 hours to train on an NVIDIA Tesla P100. Each test case run in 10 to 80 seconds. The model has 3.6M parameters and achieved a per case Dice of 0.730 and a global Dice of 0.820 on the LiTS'2017 benchmark.

The model proposed by Wang et al. [60] is based on the Mask R-CNN [22] architecture. This model generates bounding boxes around objects of interest in the image, then performs classification and segmentation in these patches. The main contribution of the work developed by [60] was the introduction of an attention module to generate the feature maps. The attention module was separated into spatial attention and channel attention. The channel attention is similar to the SE module of Hu et al. [26], whereas the spatial attention is inspired by the work of Wang et al. [59]. The model achieved a per case Dice score of 0.741 and a global Dice score of 0.813 on the LiTS'2017 benchmark. The hardware specification and the time spent to train the model were not mentioned by the work authors.

Chapter 3

Materials and Methods

In this chapter, we describe each stage of our model, the evaluation metrics, the dataset, and the computational resources used in our experiments to assess the efficiency and effectiveness of the proposed approaches.

3.1 Proposed Solution

Figure 3.1 shows a high-level diagram of our solution. The method is separated into liver segmentation and tumor segmentation. In the second stage, the mask generated in the first stage is used to focus the model on the important areas of the volume. We present a detailed explanation of each stage in Subsections 3.1.1 to 3.1.8.

3.1.1 Data Compression

We clamp the HU values of the CT volumes to the $[-200, 200]$ range to eliminate noise from materials that are not relevant to lesion or liver segmentation. The clamping operation eliminates all metals, bones and air from the volume.

We convert the values to 32-bit floating point numbers and apply a min-max normalization to limit the values to the range of 0 and 1, which generally improves the model convergence time and avoids floating-point precision errors.

Then, we compress the CT volume and the ground-truth together in an NpzFile to improve access speed and reduce disk space consumption. We also add voxel spacing and volume dimensions to the compressed file as metadata.

3.1.2 Batch Sampling

We adopted two different methods to generate samples, one for models assembled with 2D convolutions and the other for models assembled with 3D convolutions. Each sample has two components: axial slices from the CT volume and axial slices from the ground-truth mask.

A sample from a slice $s_k \in$ CT volume is created using the first method. The axial slices sampled from the CT volume are $s_k - w : s_k + w$ and the slice extracted from the ground-truth mask is s_k, where $w \in \mathbb{N}^+$ and the operator : is the concatenation operator of all slices between and including the two indices.

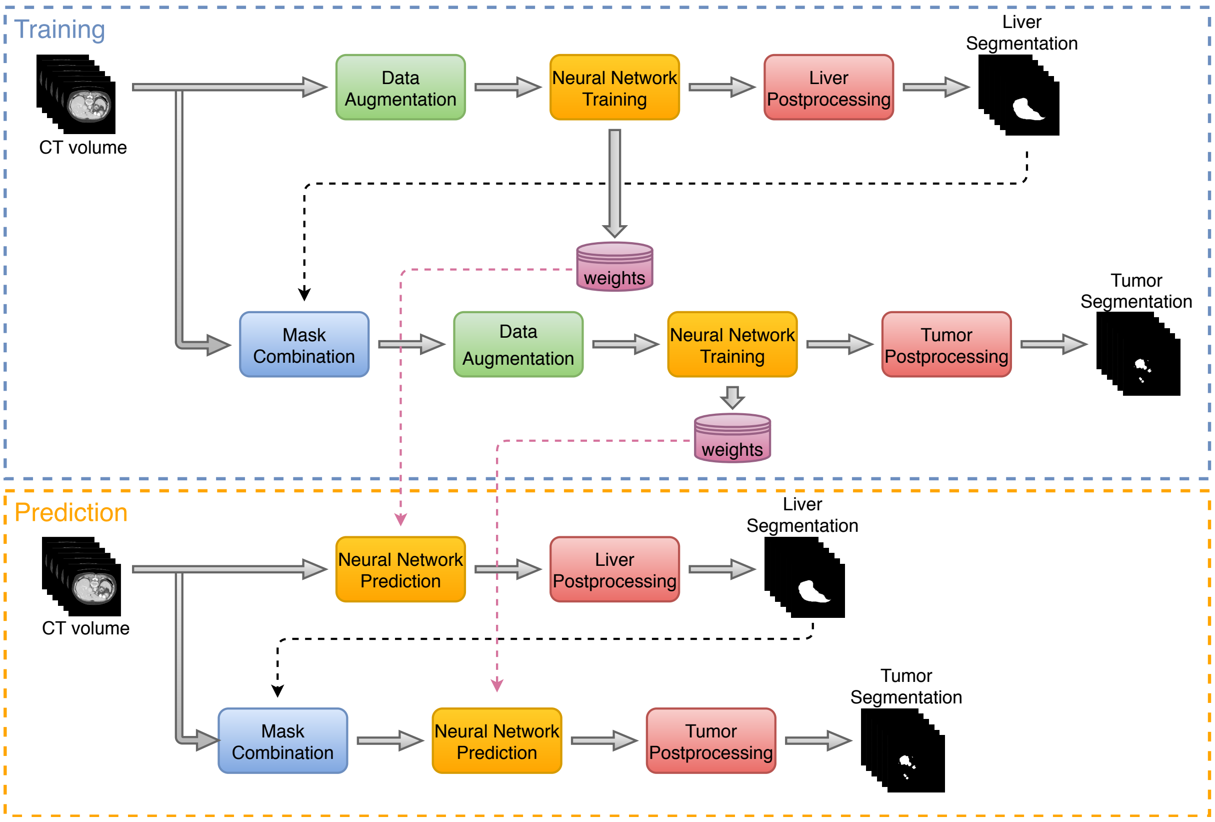


Figure 3.1: Overview diagram of the liver tumor segmentation. Source: Elaborated by the author.

A block of slices $s_k : s_i \in$ CT volume is extracted using the second method, since the model works with 3D filters. The axial slices from the CT volume are $s_k - w : s_i + w$ and the slices from the ground-truth mask are $s_k : s_i$, where $w = \dfrac{i-k}{2}$ and the : is the same as in the first sampling method. The edge cases are dealt with zero padding for the unavailable slices.

A batch generator samples every slice designated for training once per epoch during training. The sampling method includes the following steps: a random volume is chosen; each slice is used in that volume in a random order; another volume is chosen that was not elected at the current epoch. This method allows proper disk access without compromising the random sampling aspect of the batch generator.

3.1.3 Data Augmentation

The data augmentation process is only used at the training stage. We perform data augmentation using geometric and intensity transformations. The types of geometric transformations applied are: rotation, flipping, scaling, translation, shearing and elastic transformation. We use small random intensity perturbation as our only method of intensity transformation.

We limit the magnitude of the geometric transformations, because the model benefits from positional and structural information. We maintained small intensity disturbances to prevent the voxels from being perceived as different materials. Table 3.1 shows the parameters used in each transformation. Th elastic transformation is performed in only 10% of the images. In Figure 3.2, we show examples of our data augmentation.

Transformation	Range
Rotation	[-15, 15]
Shearing	[-10, 10]
Translation	[-16, 16]
Random Intensity Noise	[-0.5, 0.5]
Scaling	[-12.5, 12.5]

Table 3.1: Rotation and shearing ranges are in degrees, the translation range is in pixels, random intensity noise uses the measures of the intensity values after normalization, but the range is less than 2HU, the scaling is measured in percentage of the original image size.

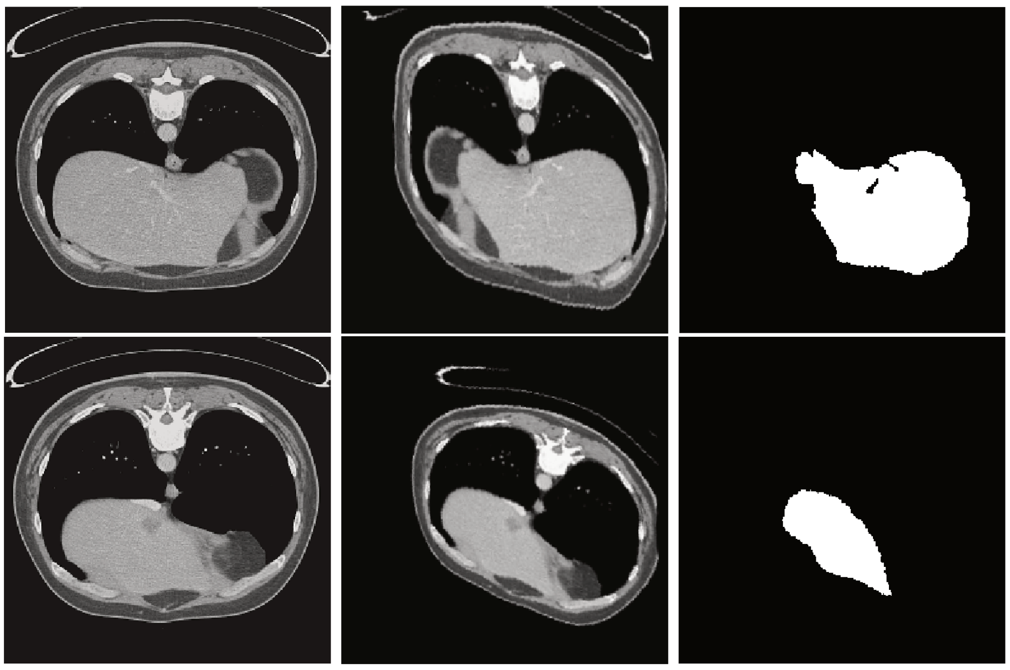

(a) Original (b) Transformed (c) Transformed Mask

Figure 3.2: Examples of data augmentation transformations.

3.1.4 Neural Network Architecture

All architectures used in the experiments can be separated into five blocks: stem, encoder, head, decoder and output. Figure 3.3 shows the connections between each block. The stem block is responsible for reducing the input dimension and adjusting the number of features. It also creates a shortcut connection to the output block before reducing the input dimensionality.

The encoder block is the most complex block of the model. It is responsible for extracting and aggregating the features from the volume, so the last blocks can generate

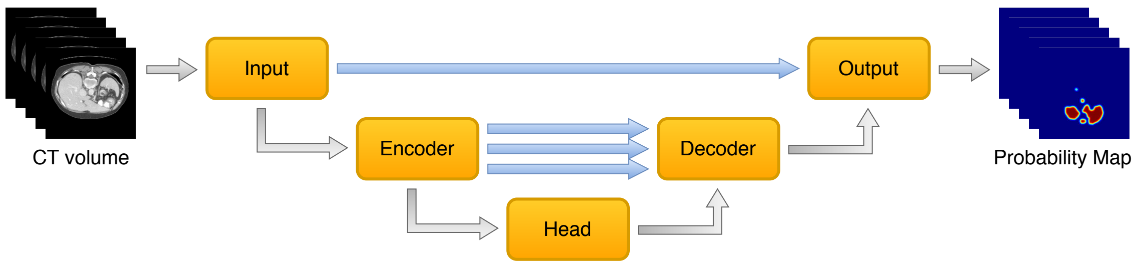


Figure 3.3: Diagram of the network architecture, where gray arrows show the main data flow and the blue arrows represent shortcut connections similar to the liver tumor segmentation. Source: Elaborated by the author.

the segmentation mask. Our encoder block is based on the EfficientNet architecture, explained in Subsection 2.1.13. We added shortcuts between downsampling layers of the encoder and the upsampling layers on the decoder block to facilitate information flow between different levels of representation, which help to mitigate any spatial information loss by dimensional reduction [47].

The head block typically maps the encoder output to the desired output space. In our model, the head block connects the encoder block to the decoder. The decoder block expands the feature map to the original volume size. The output block transforms the feature map into a probability map using the softmax function to create the segmentation probabilities for each voxel.

3.1.5 EfficientNet Modifications

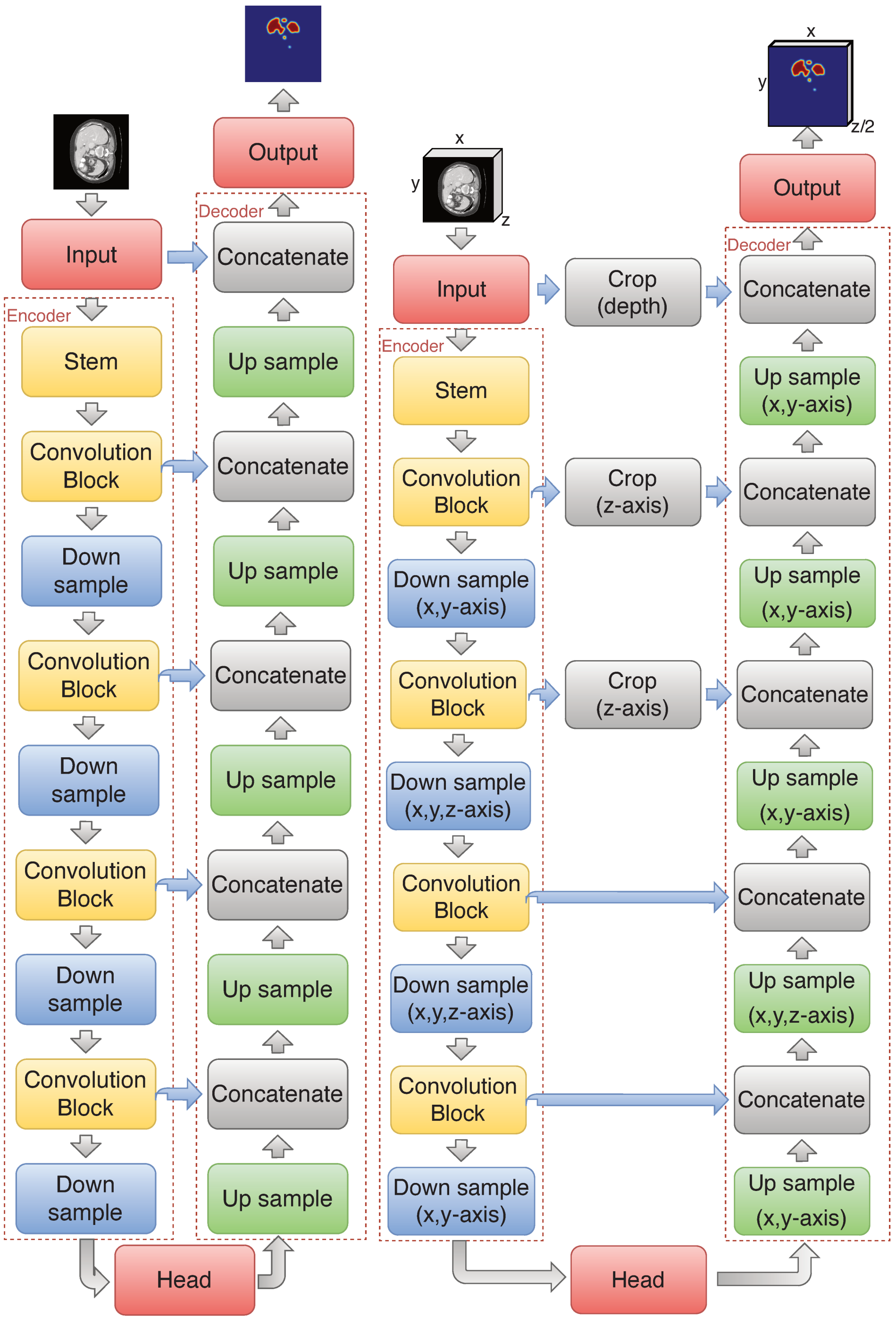
The EfficientNet was developed to achieve a balance between performance and resource usage in the image classification problem for the ImageNet dataset. The liver segmentation task shares many characteristics with the image classification problem. However, there are many peculiarities for each problem; with that in mind, we proposed different modifications to the EfficientNet architecture to optimize it to our problem.

The first modification was our input method that receives a sequence of 9 slices from the CT volume. Instead of an RGB image as in the original architecture, the input conversion is done by the stem block at the beginning of the network. The network generates a single probability map from the input that corresponds to the segmentation of the middle slice.

The EfficientNet uses inverted residual layers of the MobileNet-V2 as its main building block because, at the time, MobileNet-V2 was the state of the art in terms of accuracy per trainable variables. However, since then, different models have been proposed that reduce resource usage while maintaining the same level of accuracy, including a new version of the MobileNet. We selected the blocks proposed in the ShuffleNet-V2 [41] and MobileNet-V3 [24] as possible substitutes for the inverted residual layers in our encoder architecture.

A natural way to improve the spatial feature connectivity for volumetric data is to use 3D convolutions. Köpüklü et al. [32] already followed this idea with 3D convolutional versions of MobileNet-V1, MobileNet-V2, ShuffleNet-V1, ShuffleNet-V2 and SqueezeNet, achieving satisfactory results, although their research was focused on motion recognition.

To adapt our model for 3D convolutions, we resized our input to receive 8 slices and produce 4 slices of probability maps as output. Figure 3.4 shows a diagram of the network architecture for 2D and 3D inputs. This was done to guarantee that each probability map had sufficient amount of neighboring slices to extract information.



(a) architecture for 2D inputs (b) architecture for 3D inputs

Figure 3.4: Diagram of the model focused on the encoder block for 2D and 3D convolutions.

We achieved the desired input and output by performing dimensionality reduction on the z-axis only in the first and third rescaling and performing upsampling on the z-axis only in the first resizing of the decoder. The shortcut connections are cropped to the desired size on the encoder side to fit the decoder.

3.1.6 Liver Post-processing

We concatenate all the output slices to generate a volume probability map, which allows us to determine the voxel classification through a simple thresholding method. This generates a volumetric mask of our liver prediction that can be used in conjunction with an algorithm to find connected components, which allows us to find the largest connected component and classify it as the liver, thus eliminating most of the small false positives that may occur.

3.1.7 Combination Mask

We use the liver mask found in the previous stage to narrow the volume of interest for tumor segmentation. There are many ways to combine the input volume and the liver mask. We consider two methods in this work, described as follows.

The first approach is to mask the CT volume with the liver mask, effectively removing any tissue that is not considered liver. The second approach is to add the liver mask to the CT volume intensities, which would create a highlight in the voxels that are considered inside the liver. Both methods were chosen due to their simplicity and do not increase the input size of the model.

3.1.8 Tumor Post-processing

We use the same methods to create the probability map volume and connected components as in the liver post-processing step. However, instead of choosing only the largest component, we eliminate all components whose highest probability that a voxel presented in that component of being a tumor is below a threshold.

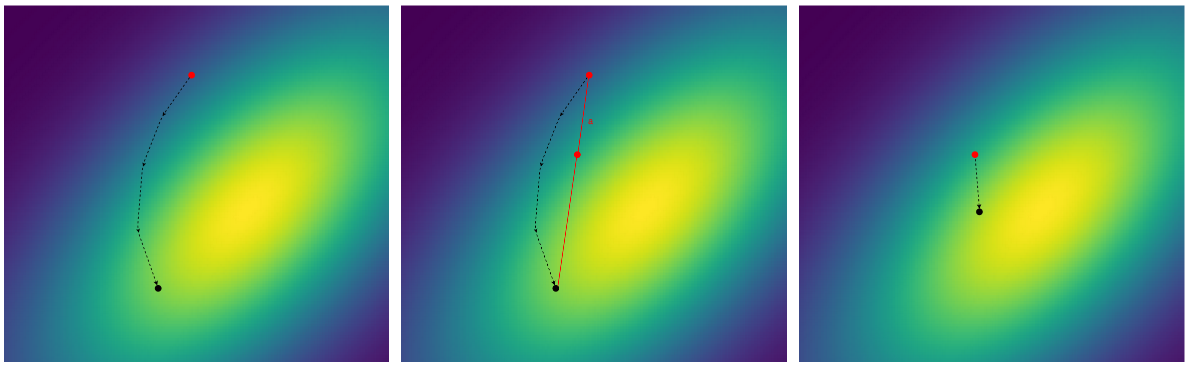
3.1.9 Network Training

Training a network is typically a very time consuming and sensitive process, which can decrease the model performance. With that in mind, we adopted modern techniques to reduce training time, improve model stability and reduce over-fitting.

Our choice of optimizer was RAdam [38], a stochastic adaptive algorithm based on the commonly used Adam [31] optimizer. RAdam rectifies its adaptive learning rate to stabilize its variance, which prevents the model from converging to biased/bad local optimal. The RAdam authors based their ideas on the improvement observed in high performance networks that gradually increased their learning rate for a few iterations, before reaching the desired learning rate and resuming their default training schedule. This technique is called linear warmup [17] and aims to stabilize the adaptive learning rate without skewing the network. We chose to use linear warmup in conjunction with RAdam, since the authors [38] who developed RAdam do not explicitly say that the optimizer eliminates the warmup necessity.

For the learning rate scheduler, we use a plateau reduction and lookahead mechanism [69]. The plateau reduction simply decreases the learning rate by a multiplicative factor when the network has not improved in the last couple of epochs.

The lookahead algorithm uses two sets of weights: a fast one that updates each iteration and the slow one that updates after k iterations. The slow set is updated with a linear interpolation of its current location and the position of the fast set of weights. Then, the fast set of weights starts the next iteration from the position of the slow set. Figure 3.5 shows a one-step representation of this process.



(a) fast weight path (b) slow weight update (c) beginning of a new iteration

Figure 3.5: 3.5a shows the algorithm after the fast weight updated k times. The black dot shows the fast weight position and the red dot shows the slow weight position. In 3.5b, we have the new position (in red) for the slow weights. In 3.5c, we have the first iteration of the fast weight in a new cycle of the algorithm. Source: Elaborated by the author.

We use the weights that achieved the best results in our validation set to avoid overfitting the training samples. In addition, we use early stopping. We also apply Stochastic Weight Average (SWA) to the weights with the best loss value for the training data.

Stochastic Weight Average

The Stochastic Weight Average (SWA) [29] method was developed based on two observations: (i) the loss surface observed on the training set does not necessarily matches the surface loss on the validation dataset and (ii) the stopping point of the optimization algorithm tends to remain on the boundary of the minimum surface, because the gradient within the minimum surface is too small to updated the weights.

SWA addresses these problems by averaging different weights. These weights are calculated by running the optimization method with a higher learning rate after the model has stabilized. The high learning rate will move the weights around the minimum surface border, which provides multiple points on the minimum surface that can be averaged to achieve a combination of weights that best generalizes to different datasets.

3.2 Evaluation Metrics

The evaluation metrics are divided into four groups: (i) detection metrics responsible for measuring the number of objects correctly detected, (ii) overlapping metrics that measure the number of voxels of the object correctly classified, (iii) size metrics that compare the predicted object volume with the ground truth and (iv) surface distance metrics that measure the distance between the borders of the predicted object and the border of the ground-truth.

3.2.1 Detection Metrics

The correct detection of an object is determined by the intersection over union ratio of the predicted mask and the ground-truth. The detection is considered in two values, greater than zero and greater or equal to 0.5, the first being more strict.

From the previous detection classification, it is possible to count occurrences of true positive (TP), false positive (FP) and false negative (FN), from which we can compute recall and precision metrics using Equations (3.1) and (3.2), respectively. Value TP corresponds to tumors found correctly, TF to tumors not found by the model and FN to the predictions that do not correspond to tumors.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.1}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{TF}} \tag{3.2}$$

The tumor prediction mask is not created for a specific lesion instance. Therefore, it is necessary to establish the correlation between the predicted tumor and a ground-truth tumor.

First, each lesion in the prediction and references is labeled using connected components. Then, any two reference lesions where there is a predicted lesion connecting them are given the same label, which transforms the lesion correspondence from many-to-many to a many-to-one problem. The last step is to combine any predicted lesion that overlaps the same reference lesion to generate a one-to-one mapping. To maintain the number of reference lesions invariant, the number of detection of a reference component is the same as the number of lesions merged in that component.

3.2.2 Overlapping Metrics

Overlapping-based methods can determine whether voxels from the predicted segmentation appear in the ground-truth segmentation.On the other hand, they cannot calculate the distance between the voxels that are present in one set and not in the other.

Dice Similarity Coefficient

Dice [14] proposed a metric to quantify the matches between two sets. Many other researchers have adopted the metric to evaluate the performance of medical segmentation. The Dice metric compares the overlapping ratio between the predicted mask and a manual segmentation done by a specialist.

Given two sets of voxels A and B, where A denotes a predicted segmentation of liver tumors and B denotes the ground-truth of liver tumors, we can calculate the Dice similarity coefficient through Equation (3.3), where $|A \cap B|$ is the intersection size between the two sets, and $|A|$ and $|B|$ are the sizes of the respective sets.

$$\text{DICE}(A, B) = \frac{2 \times |A \cap B|}{|A| + |B|} \tag{3.3}$$

Jaccard Volume Overlap Error

The Jaccard similarity coefficient [30] of two sets is defined as their intersection size over their union size. The Jaccard Volume Overlap Error (VOE) is the complement of the Jaccard coefficient. Using A and B as the same pair of sets of the Dice similarity coefficient, VOE can be written in terms of A and B, as shown in Equation (3.4).

$$\text{VOE} = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{3.4}$$

It is possible to observe that VOE measures the difference between two sets, so that similar sets have low VOE values, which is the opposite of the Dice similarity coefficient.

3.2.3 Size Metrics

Size-based metrics discard any information about voxel position and focus only on measuring the size difference between the segmentation results.

Relative Volume Difference

The Relative Volume Difference (RVD) metric is a dissimilarity metric that measures the relative difference between the number of voxels in the predicted segmentation set A and the ground-truth segmentation set B.

The RVD metric is expressed in Equation (3.5), where $|A|$ is the predicted segmentation set size and $|B|$ is the ground-truth segmentation set size.

$$\text{RVD} = \pm\frac{|A| - |B|}{|B|} \tag{3.5}$$

3.2.4 Surface Distance Metrics

Before describing a number of surface distance-based metrics, it is important to introduce some important concepts. Let x be the distance between a voxel and a set of voxels Y, defined as

$$\text{distance}(x, Y) = \min_{\forall y \in Y} \text{distance}(x, y) \tag{3.6}$$

The function distance in Equation (3.6) is the Euclidean distance between the coordinates of voxel x and y. Let B_A be the set of all voxels in A's border and B_B denote the set of all voxels in the B's border.

Metrics based on surface distance consider only voxels that belong to the segmentation surface. This characteristic can mask large volume errors that do not deform the segmentation surface.

Average Symmetric Surface Distance

The Average Symmetric Surface Distance (ASSD) metric averages the distance between all voxels in B_A to B_B and all voxels in B_B to B_A. This metric gives a general idea of how close the borders are in their entire extension. However, due to its robustness to outliers, it masks any isolated mistakes. Since a variation δ over a single distance has the same effect as a variation of $\frac{\delta}{|B_A|+|B_B|}$ spread over all distances, consequently, a large surface has a smaller variation in ASSD value.

$$\text{ASSD} = \frac{1}{|B_A| + |B_B|} \times \left(\sum_{x \in B_A} \text{distance}(x, B_B) + \sum_{y \in B_B} \text{distance}(y, B_A) \right) \tag{3.7}$$

Maximum Symmetric Surface Distance

As the ASSD metric cannot detect rare segmentation errors, the Maximum Symmetric Surface Distance (MSSD) is sensitive to them, which makes them complementary metrics.

$$\text{MSSD} = \max_{\forall x \in B_A} (\text{distance}(x, B_B)) \tag{3.8}$$

3.3 Dataset

The dataset used to evaluate the performance of our method is the LiTS'2017 benchmark, organized with 2017 IEEE International Symposium on Biomedical Imaging (ISBI) and 2017 International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI).

The dataset is composed of 201 CT volumes, from which 194 contain liver lesion. The volumes were collected from seven different institutions around the world. In particular, the LiTS benchmark includes the public 3D-IRCADb dataset in the training-set. Patients suffered from primary and secondary liver tumor and metastases.

The data contain liver and tumor labels created manually by trained radiologists and oncologists. The quality of labels correlated to the test set was verified by three experienced radiologists in a blind review [5]. The dataset was divided into 131 samples for training and 70 samples for testing. The testing samples were published without the label annotation.

In Figure 3.6, we show four examples of axial slices that represent some of the challenges that occur in the liver tumor segmentation problem. Figure 3.6a illustrates artifacts created due to the volume extraction method, where the artifact can obscure important information about the tomographic volume; the artifacts are similar to bright rays around the patient column. Figures 3.6b, 3.6c and 3.6d show examples of tumors with different shapes, sizes, positions and contrast in the LiTS training dataset. Figure 3.6b shows a small tumor with high contrast, but tangled with healthy tissue. Figure 3.6c shows a cluster of large and small tumors covering a large part of the liver. Finally, Figure 3.6d illustrates a tumor that is small and barely distinguishable from the liver.

As briefly mentioned about class imbalance in Section 1.2, Figure 3.7 shows the voxel count on a logarithmic scale for each of our three classes, background, liver and tumor.

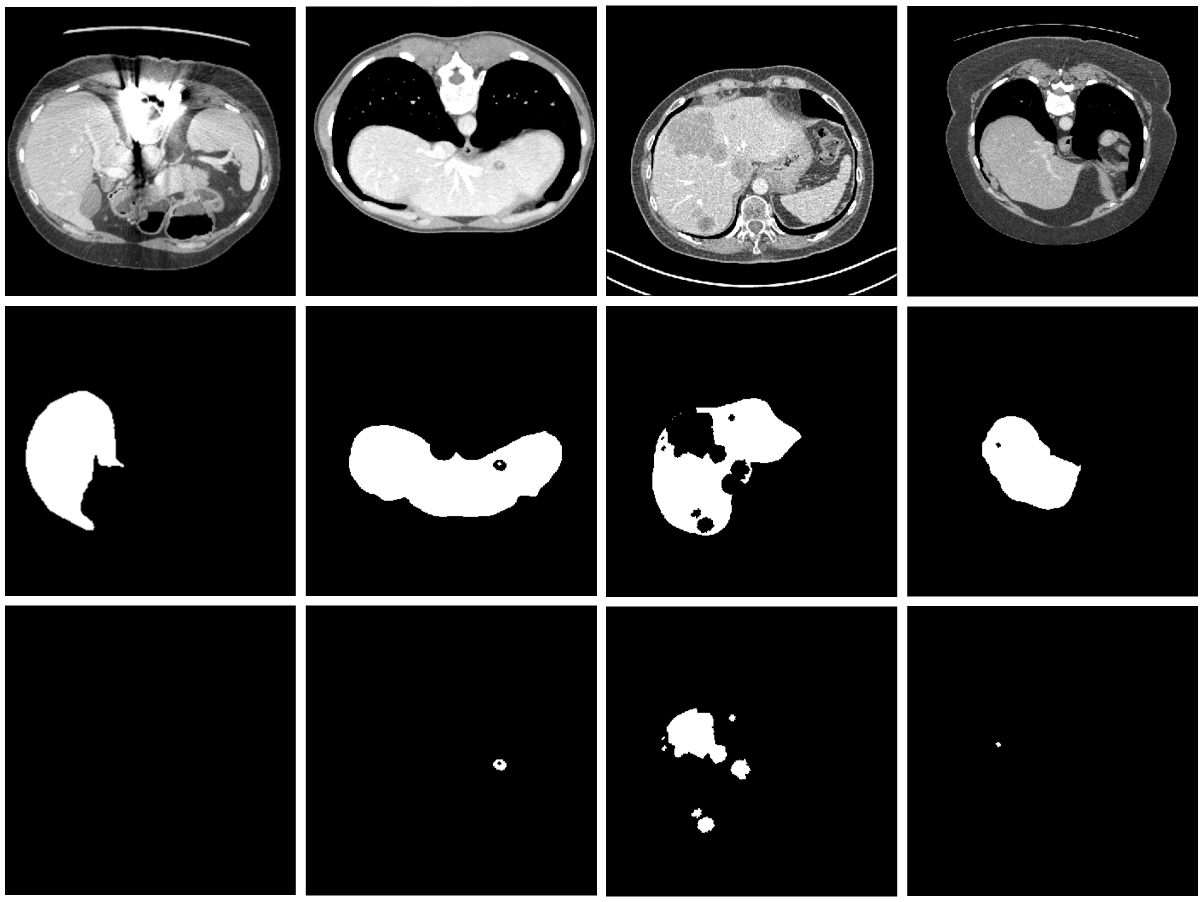


Figure 3.6: Axial slice examples from volumes of the LiTS dataset, with their respective liver and tumor ground truth masks. Source: LiTS database [37].

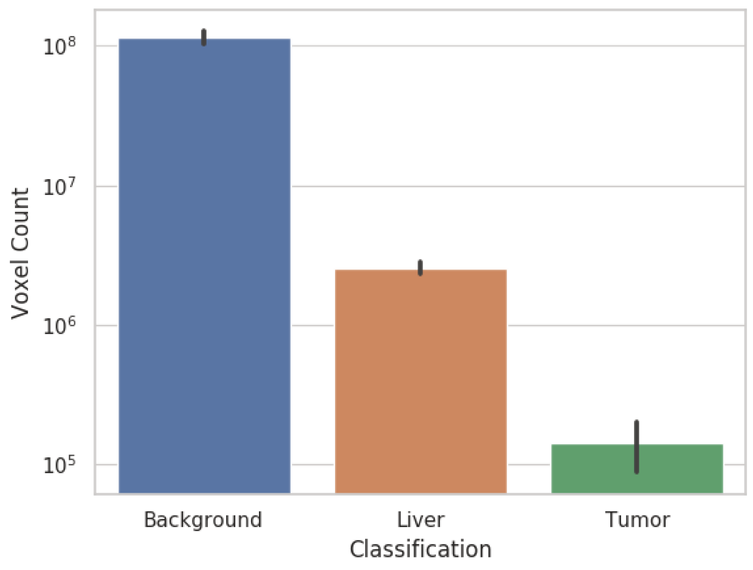
The graph on the left considers the entire volume, whereas the graph on the right focuses only on slices with liver.

It is possible to notice that the number of background voxels is close to 100 times larger than the number of liver voxels; and close to 1000 times larger than the number of tumor voxels. The variance across the volumes is considerably higher for tumors than the other two classes. The number of background voxels reduce to about a third without the slices with no liver tissue, but it is enough to affect the training process.

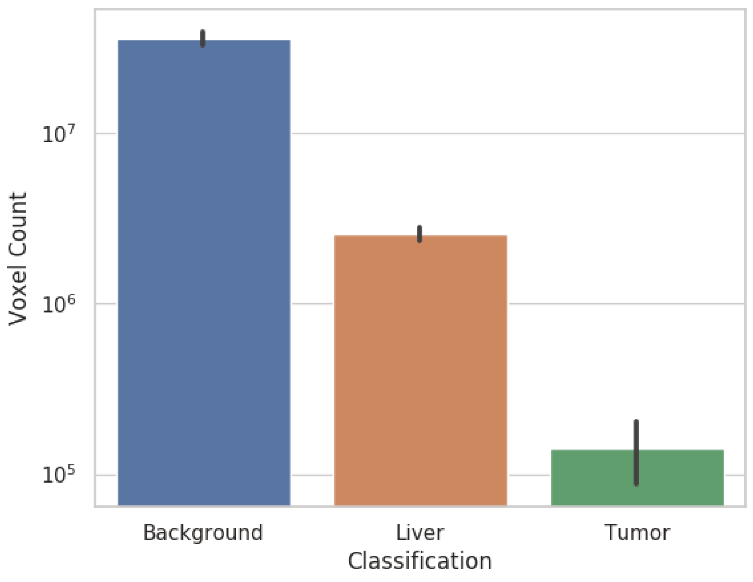
3.4 Computational Resources

The proposed method was implemented in Python programming language (version 3.x) due to the great availability of packages for machine learning, image manipulation, data analysis, numerical processing and data visualization. The main packages used in the implementation were NumPy, OpenCV, PyTorch, Pandas, SciPy, Seaborn, Matplotlib, SciPy, Scikit-Learn Scikit-Image, MedPy and TensorBoardX.

The experiments were conducted at the Laboratory of Visual Informatics (LIV) of the Institute of Computing (IC) at University of Campinas (UNICAMP). The computers are equipped with 3.5 Ghz Intel i7-3770 processors, 32 GB of RAM memory, an NVIDIA



(a) label count of the entire LiTS dataset



(b) label count from the slices classified with liver tissue

Figure 3.7: Label count of the LiTS dataset on a logarithmic scale. Source: Elaborated by the author.

GeForce GTX 1080ti graphics card, with 3584 CUDA cores at 1582 MHz, with default memory DDR5 SDRAM of 11 GB and 11 Gbps. The operating system was Ubuntu 18.04 LTS with CUDA 10.0 and cuDNN 7.5. For the CPU tests, we used a private machine equipped with an Intel Core i7-6700K CPU and 16 GB of RAM memory.

Additional resources from Google Colab[1] were used to prototype the first models of the project. Colab machine instances have 16 GB of memory with NVIDIA K80 or T4,

[1]`https://colab.research.google.com/`

with 12 GB and 16 GB of memory, respectively. The storage space used was provided by mounting a Google drive node on the server with the PyDrive library.

Chapter 4

Results

In this chapter, we describe and discuss the details of our experimental setup, present the results of our experiments and compare our solution with other approaches available in the literature.

4.1 Experimental Setup

We used two experimental setups: one that is performed locally and the other that is executed on the LiTS Challenge server. We opted to carry out local experiments because it allows us to visualize and compare the segmentation results on a case-by-case basis, which in turn helps us to identify the model's shortcomings.

Moreover, the comparison would be very hard to be made with only performance metrics, even with the diversity of metrics that the LiTS benchmark reports. When using a validation set, we also avoid overfitting our segmentation model to the test set, which could cause our model to perform better than a real use-case test would.

4.1.1 Local Evaluation

We divided the LiTS training set into two groups: the training group has 85% of the volumes and the validation group has 15%. The list of both sets are described in detail in Appendix A.1.

We downsampled the inputs used for training and prediction from 512×512 to 256×256 pixels. Then, we upsampled the predicted masks to the original size for evaluation. We opted to work with smaller image dimensions because it allows for faster experimental iterations. We also limited the experiments to the second stage of our model since several neural networks have already achieved liver segmentation with high Dice scores, while the tumor segmentation still poses a challenging problem.

In our preliminary test using an architecture similar to that proposed by Han [19], we observed a high variation of the segmentation quality across the volumes. In Figures 4.1 and 4.2, we show the Dice scores per CT volume, where the x-axis is ordered by the average number of voxels per tumor. It is clear that the network performed worse on volumes that presented smaller average tumor sizes.

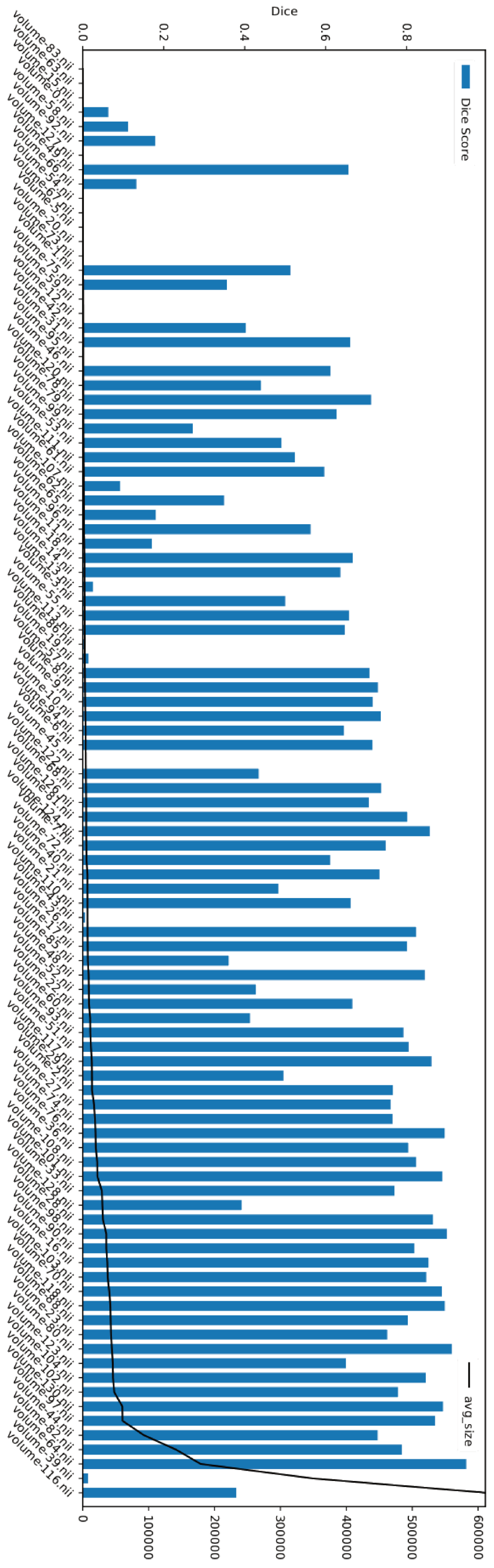


Figure 4.1: Dice score for each individual volume in the training set (ordered by the average size of each tumor). Volumes without any tumors were removed because their Dice values are binary.

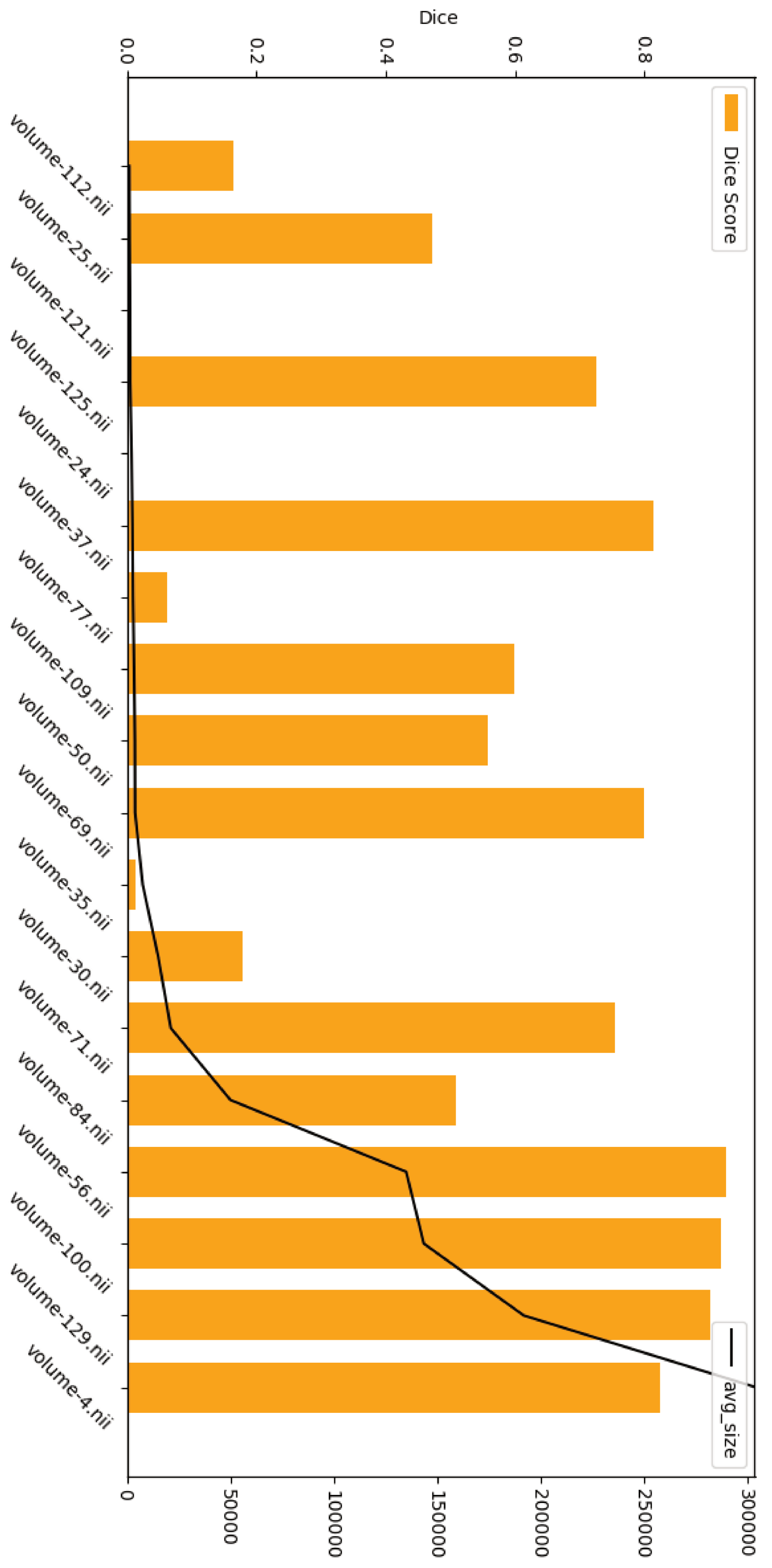


Figure 4.2: Dice score for each individual volume in the validation set (ordered by the average size of each tumor). Volumes without any tumors were removed because their dice values are binary.

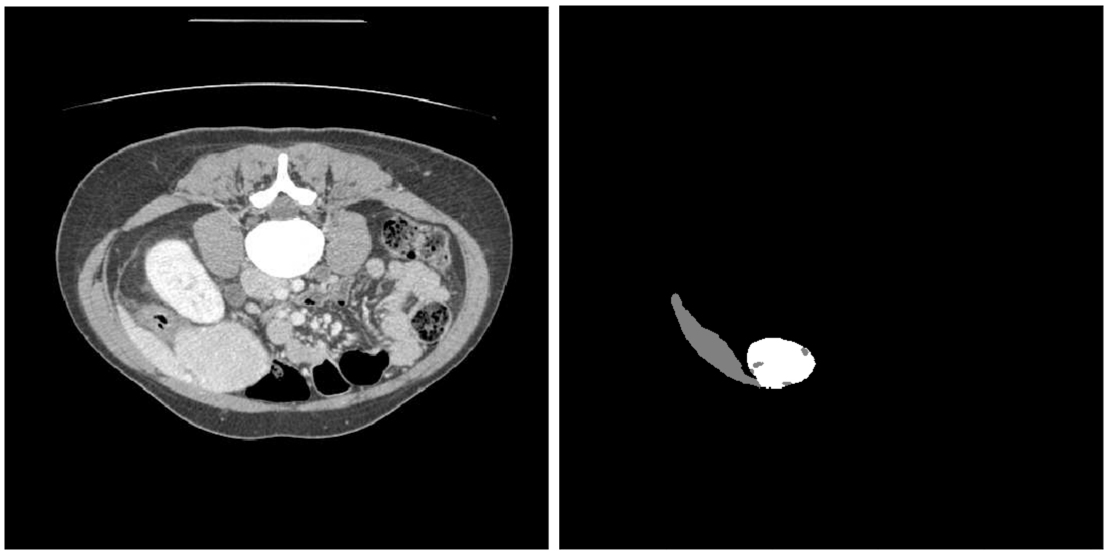

(a) CT slice (b) Mask

Figure 4.3: Example of a hard to segment large tumor; the mask (right) shows the segmentation for the liver in gray and the tumor in white.

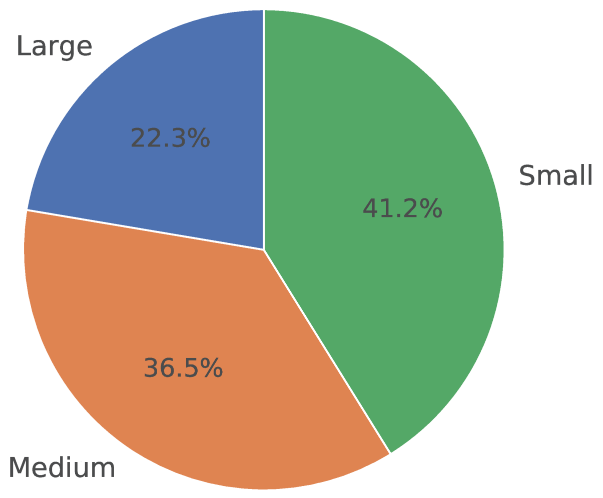
We examined volume 39 because the basic model performed poorly in this example, regardless of its average tumor size. Figure 4.3 shows a slice that exemplifies why we believe the basic model had difficulty identifying the tumor on volume 39. The tumor grew as an appendix of the liver that modified the neighboring information needed to perform the segmentation. The tumor also had an intensity value very similar to the healthy liver tissue.

In order to study the influence of tumor size on the performance of our models, we divided each tumor instance into three groups based on their number of voxels. The value range of each group is shown in Table 4.1, where v stands for the tumor volume.

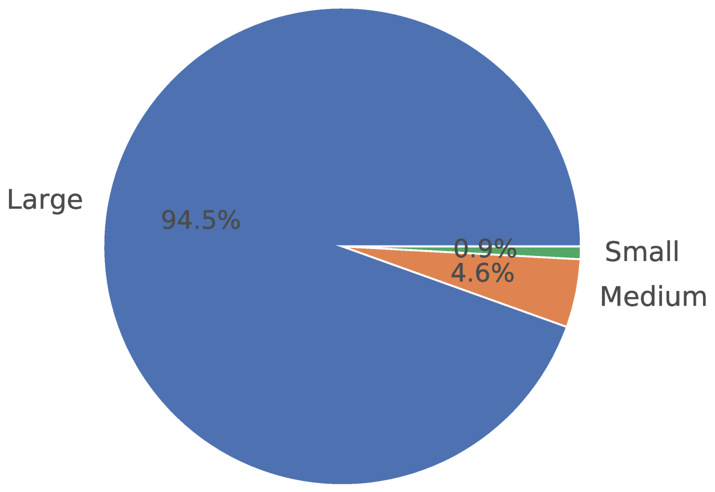
Group	Volume (voxels)
Small	$v < 1000$
Medium	$1000 \leq v \leq 10000$
Large	$v > 10000$

Table 4.1: Classification limits used to separate each tumor instance by its size.

Figure 4.4a shows the distribution of the number of tumors for each group. Figure 4.4b shows the sum of voxels per group. As we can observe, the number of small tumors is responsible for more than 40% of all tumors; on the other hand, the number of voxels that they represent is less than 1% of the total tumor volume. The vast majority of the tumor volume in the dataset is composed of large tumors. Even medium-sized tumors represent a small portion of the total volume.



(a) Number of tumors



(b) Sum of tumor volumes

Figure 4.4: (a) number of tumor for each group present in the training set; (b) total volume that each group of tumors represents in the training set.

2D Models

As mentioned in Section 3.1.5, we have three different 2D convolutional layer architectures. For each layer architecture, we experimented with three different sets of hyper-parameters: top, default and bottom.

The default set maintains the same proportions as used in EfficientNet-B0, the top set increases the shallower layers, whereas the bottom set increases the deeper layers (the top and bottom sets have decreased the number of parameters in their deeper and shallower layers to balance the Floating-Point Operation (FLOP) count with the default set). Appendix A.2 presents more details for all the hyper-parameters values used in each model.

We based the choice of our filter size and layer count on the sum of Floating-Point

Operation. Our target value was the same as the EfficientNet-B0. Table 4.2 shows the number of parameters and FLOPs for all the 2D models we tested.

Model	Layer Configuration	FLOPs	Parameters
MobileNet-V2	top	107.14M	4.64M
	default	123.89M	3.76M
	bottom	104.80M	6.37M
MobileNet-V3	top	103.01M	4.47M
	default	113.72M	3.64M
	bottom	103.06M	7.80M
ShuffleNet-V2	top	118.93M	1.44M
	default	122.31M	1.29M
	bottom	122.13M	1.47M

Table 4.2: Number of parameters and FLOPs for each 2D model, calculated for inputs of size 9×256×256. The values were measured using `https://github.com/sovrasov/flops-counter.pytorch`.

3D Models

In order to reduce memory consumption and computational time introduced by replacing 2D convolutions with 3D convolutions, we reduced the number of channels in each layer by half. Table 4.3 shows the number of parameters and FLOPs for each tested 3D architecture. The number of parameters is much smaller when compared to 2D models, which is expected since the number of channels in the filters was reduced by half.

We tested the same three-layer architectures from the 2D models. We also added the DSTS from Section 2.1.9 and tested three sets of different hyperparameters. They follow the same idea as those used in the 2D model tests, but the default set has its width reduced by half. Appendix A.2 provides more details for all hyperparameters values used in each model.

4.1.2 Training Times

In Tables 4.4 and 4.5, we show the average time to generate a slice of the probability map for each tumor segmentation. For 2D models, the time to create one slice is t/b, where t is the run time of one iteration and b the batch size. For 3D models, the time for one slice is $t/(bz)$, where t and b are the same as before and z is the z-axis size, whose value is 4 in our case.

The 3D models had an increase in training and prediction time of about 4 times in each model. The DSTS took the largest time, although it has a comparable number of FLOPS and parameters. We suspect that the network fragmentation [41], generated by the branching of each layer in spatial and temporal convolution, caused a decrease in computational performance.

Model	Layer Configuration	FLOPs	Parameters
MobileNet-V2	top	64.28M	1.26M
	default	61.69M	1.59
	bottom	64.03M	2.93M
MobileNet-V3	top	67.25M	0.99M
	default	62.44M	1.95M
	bottom	64.78M	3.29M
ShuffleNet-V2	top	68.84M	1.33M
	default	61.53M	1.59M
	bottom	56.37M	1.95M
DSTS	top	66.99M	1.68M
	default	61.22M	2.52M
	bottom	55.19M	2.88M

Table 4.3: Number of parameters and FLOPs for each 3D model, calculated for input of size 8×256×256. These values were obtained using the module from `https://github.com/sovrasov/flops-counter.pytorch`.

Model	Layer Configuration	Training		Prediction	
		Mean (s)	StDev	Mean (s)	StDev
MobileNet-V2	top	0.345	0.047	0.114	0.039
	default	0.237	0.014	0.072	0.001
	bottom	0.292	0.038	0.086	0.009
MobileNet-V3	top	0.242	0.024	0.079	0.007
	default	0.227	0.001	0.068	0.001
	bottom	0.303	0.004	0.090	0.002
ShuffleNet-V2	top	0.362	0.076	0.099	0.005
	default	0.295	0.031	0.083	0.009
	bottom	0.355	0.037	0.105	0.021

Table 4.4: Training and prediction times for each combination of block and hyperparameters of the 2D models.

4.2 Results and Discussion

This section is divided into two parts. The first presents and compares local evaluation results. The second compares our best models against the state of the art on the LiTS test dataset.

4.2.1 Local Evaluation

Table 4.6 shows the Dice value for each 2D model separated by tumor size. The results indicate that concentrating the number of parameters in the shallower layers helps in

Model	Layer Configuration	Training Mean (s)	Training StDev	Prediction Mean (s)	Prediction StDev
MobileNet-V2	top	1.15	0.039	0.428	0.0194
	default	1.14	0.0016	0.411	0.001
	bottom	1.24	0.013	0.428	0.008
MobileNet-V3	top	1.19	0.001	0.400	0.001
	default	1.065	0.001	0.381	0.008
	bottom	1.13	0.241	0.397	0.002
ShuffleNet-V2	top	1.31	0.001	0.447	0.001
	default	1.274	0.001	0.425	0.008
	bottom	1.270	0.002	0.417	0.001
DSTS	top	1.55	0.001	0.422	0.001
	default	NA	NA	NA	NA
	bottom	1.713	0.002	0.454	0.001

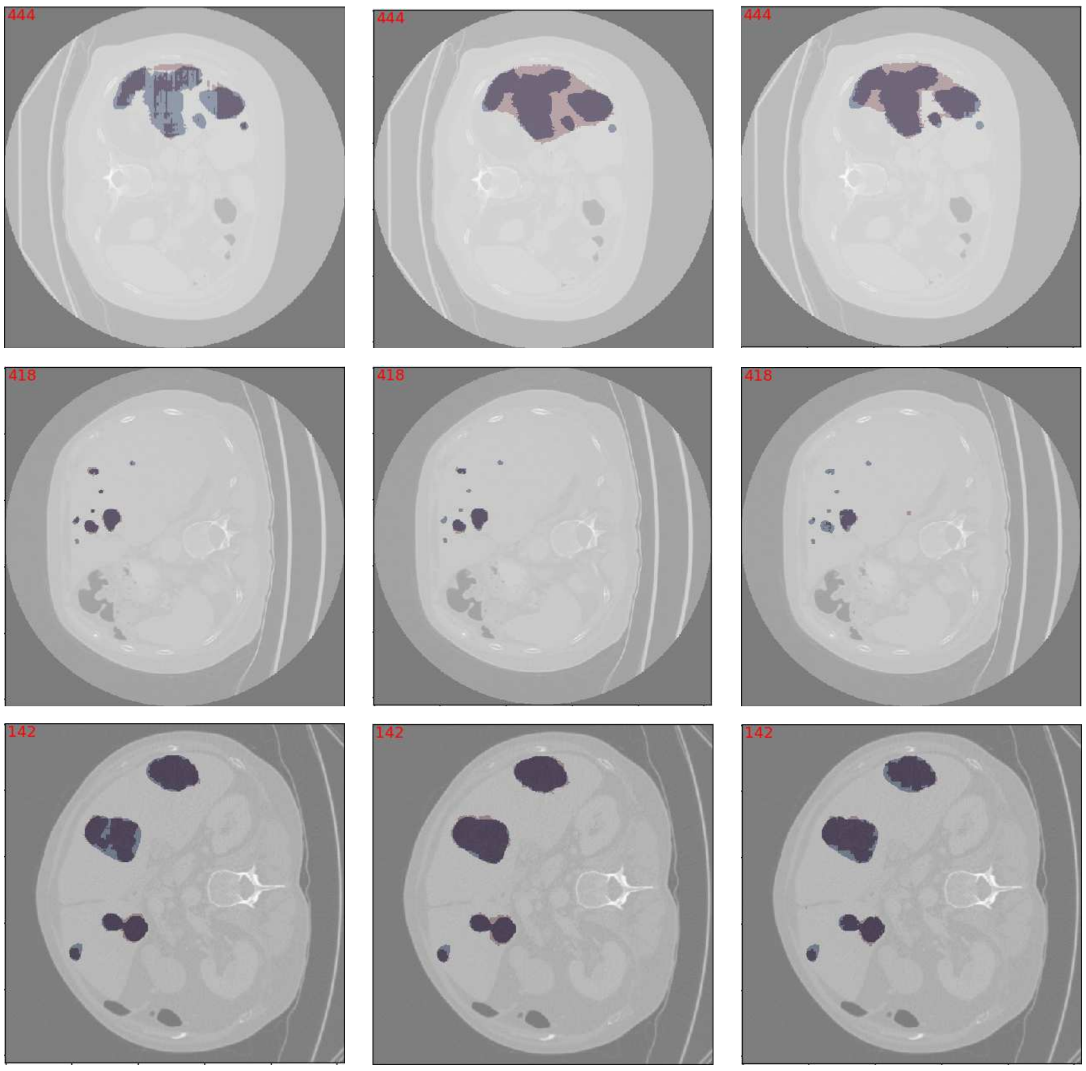
Table 4.5: Training and prediction times for each combination of block and hyperparameters of the 3D models.

the detection of smaller tumors, while concentrating the parameters in the deeper layers improves large tumor segmentation accuracy.

Model	Layer Configuration	Small	Medium	Large
MobileNet-V2	top	**0.233**	**0.470**	0.615
	default	0.079	0.182	**0.631**
	bottom	0.065	0.293	0.608
MobileNet-V3	top	0.134	0.405	0.548
	default	**0.140**	**0.418**	0.597
	bottom	0.104	0.337	**0.671**
ShuffleNet-V2	top	**0.163**	**0.385**	0.581
	default	0.058	0.339	**0.617**
	bottom	0.143	0.381	0.529

Table 4.6: Dice scores for each 2D model separated by tumor sizes. The numbers in bold highlight the best results.

On Figure 4.5, the left column model (parameters focused on shallow layers) presented problems with larger tumors (top row example), but was able to detect smaller tumors in the second example (middle row). This occurred because more of the shallower layers were able to use the detailed spatial information before the dimensionality reduction. The default EfficientNet model (middle column) did not detect as many small tumors as the left column model, but performed much better than the deeper model (right column). In the third example (bottom row), both the right and middle models performed better than



Top Default Bottom

Figure 4.5: Examples of segmentation for different hyperparameters with the EfficientNet using MobileNet-V2 block. The blue overlay represents the ground-truth segmentation and the red overlay represents the predicted segmentation. The left column represents the top set, the middle column represents the default set and the right column represents the bottom set. The examples were extracted from volume 4 (slice 444), volume 109 (slice 418) and volume 129 (slice 142). All examples are part of our validation set.

the left model, however, the right model presented insufficient precise spacial information to generate a smoother segmentation, as shown by the straight segmentation lines, which indicates that the model was unable to distinguish these pixels.

Table 4.7 shows the Dice scores for each 3D model separated by tumor size. The results of the 3D models do not clearly indicate how the parameter distribution influences the model performance. Based on the results, the best performing models are based on the DSTS layer.

In Figure 4.6, we show segmentation examples for three different models. The best 2D model (MobileNet-V2 - top left), its 3D counterpart (middle), and the best 3D architecture (DSTS - bottom right). The first example shows a slice where the 2D model failed to segment a large tumor, the middle model created a worse segmentation, which was

Model	Layer Configuration	Small	Medium	Large
MobileNet-V2	top	**0.195**	**0.455**	**0.646**
	default	0.125	0.359	0.577
	bottom	0.104	0.388	0.624
MobileNet-V3	top	0.196	0.415	**0.646**
	default	0.181	0.392	0.619
	bottom	**0.200**	**0.472**	0.631
ShuffleNet-V2	top	0.151	0.405	0.535
	default	**0.243**	**0.454**	**0.611**
	bottom	0.122	0.406	0.582
DSTS	top	0.192	0.483	0.631
	default	0.151	0.447	0.617
	bottom	**0.289**	**0.572**	**0.668**

Table 4.7: Dice scores of each 3D model separated by tumor sizes. The numbers in bold highlight the best results.

improved by the right model when compared to the previous ones, but that was much worse than the best 2D models shown in Figure 4.5. The jittering artifact can be explained by an underfitting of the model, caused by the lack of parameters.

In the second example, we have a group of small tumors. All models performed well in this example, but the left segmentation was the best followed by the right and, finally, the middle one. However, the right and middle models detected some tumors better than the left model, which indicates that some tumors are easier to segment by 3D models.

In the third example, the left and middle model generated an inferior segmentation in the large tumor than the right model, but the left model had a better segmentation in the small tumor at the bottom.

The fourth example shows a tumor that the left model failed to detect, but to which the middle and right models generated a good segmentation. The last example shows a small false positive from the left model that is not present in the middle and right model.

In Figure 4.7, we compare the average prediction time per slice for our best performing models. All models have average prediction time of less than one second per slice. The 3D models took approximately ten times longer than their 2D counterparts.

4.2.2 LiTS Challenge Evaluation

We submitted three predictions to the LiTS challenge: the first was based on the MobileNet-V2 with 2D convolutions, the second was based on the MobileNet-V2 with 3D convolutions and the last used the DSTS layers. We trained the models using the same splits as the local evaluation.

The evaluation of the LiTS challenge is done on their servers. The submission is composed of the mask for each volume, where the value 0 represents background, 1 represents liver, and 2 represents the tumor. The number of submissions is limited to 3 per day to avoid model bias. The metadata from the submitted Neuroimaging Informatics Technology Initiative (NIfTI) files must be appropriate to the corresponding CT volume. Table 4.8 lists the LiTS challenge entries alongside their segmentation evaluation metrics and parameter count.

Model	Dice per Case ($\uparrow$)	Global Dice ($\uparrow$)	VOE ($\downarrow$)	RVD ($\rightarrow 0$)	ASSD ($\downarrow$)	Parameters ($\downarrow$)
Volumetric Attention [60]	**0.741**	0.813	0.389	-0.177	1.1224	25M+
LW_HCN [68]	0.730	0.820	NA	NA	NA	3.6M
H-DenseUNet [36]	0.722	**0.824**	**0.366**	4.272	**1.102**	80M
DeepX [67]	0.657	0.820	0.378	0.288	1.151	30M
3D-UNet [10]	0.55	NA	NA	NA	NA	19M
X.Han [19]	NA	0.670	0.450	**0.040**	6.660	24M
DSTS-3D (ours)	0.548	0.768	0.400	-0.120	1.187	2.88M
2D MobileNet-V2 (ours)	0.536	0.678	0.429	-0.162	1.4232	4.64M
3D MobileNet-V2 (ours)	0.524	0.746	0.393	-0.041	1.1298	1.26M

Table 4.8: Segmentation metrics of the best performing published works on top, our model results separated by rule at the bottom. The $\uparrow$ indicates results with higher values are better; $\downarrow$ indicates results with lower values are better and $\rightarrow 0$ indicates results closer to 0 are better. Our models are at the bottom separated by a rule.

Our method did not achieve the performance necessary to compete with more specialized liver tumor segmentation methods. However, unlike most approaches, our method focused on limited resource environments, instead of maximizing performance.

Our best performing method achieved similar performance to 3D-UNet with 5 times less parameters. It also achieved proper results in the VOE and ASSD metrics, but its weakest metric was given by the Dice per case. If we compare the 2D MobileNet-V2 with 3D MobileNet-V2, we notice a drop in Dice per case, but an increase in ASSD, VOE and Dice global, which indicates that the 3D model created better overall segmentation, but the 2D model detected tumors correctly in more volumes.

The limited number of FLOPS that we imposed on our 3D models reduced its ability to detect smaller tumors. The only best performing solution with comparable number of parameters was the LW_HCN model, but it uses images four times larger than ours to generate its prediction.

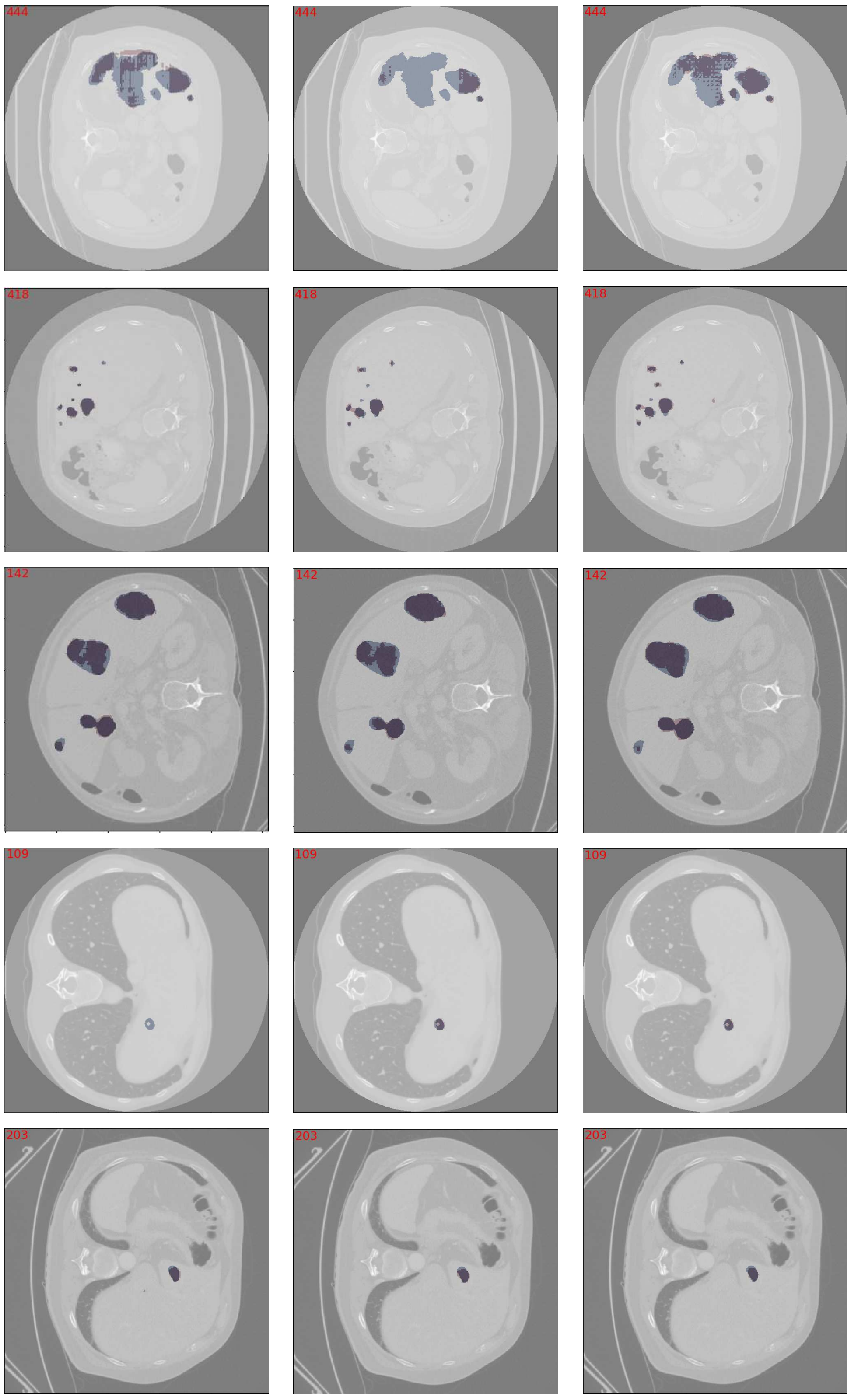


Figure 4.6: Examples of segmentation for the best performing 2D MobileNet-V2 architecture (left), its 3D counter part (middle) and the best performing 3D model (right). The blue overlay represents the ground-truth segmentation and the red overlay represents the predicted segmentation. The three first examples are the same as in Figure 4.5. The fourth example was extracted from volume 35 (slice 109) and the last example was extracted from volume 69 (slice 203), both from the validation set.

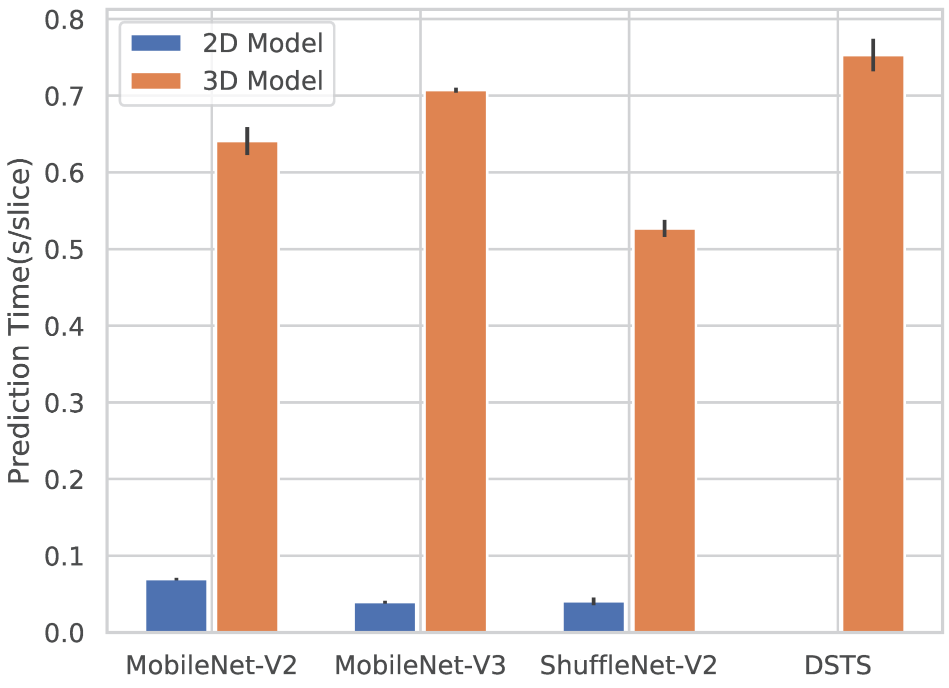


Figure 4.7: Illustration of average time to predict a slice of the CT volume. The standard deviation is represented by the black line in the middle of the bar.

Chapter 5

Conclusions and Future Work

In this work, we studied and analyzed the viability of efficient networks as encoder backbones for the liver tumor segmentation problem. We started from a baseline network to investigate its shortcomings and create a better understanding of the challenges posed by our problem.

The segmentation quality was correlated with the tumor size present in the volume. Large tumors presented a high segmentation quality and detection rate, while small tumors were poorly detected or overestimated. Based on our observations, we proposed a method to evaluate the segmentation quality that distinguishes tumors by their size, which allows a better understanding of how the model works under different circumstances.

We created two models, one based on 2D convolutions and the other based on 3D convolutions. Our 2D model is end-to-end segmentation based on the U-Net architecture with an adaptation of the EfficientNet-B0 as encoder. We tested two additional efficient layer architectures for the inverted residual layer (EfficientNet base layer), with three different layer distributions to determine how optimized the original EfficientNet architecture was for the tumor liver segmentation problem. Our results demonstrated that focusing on layers before the dimensionality reduction improved the detection of small tumors, with some degradation of the large tumor segmentation.

Our 3D-convolution model followed the same architecture as the 2D model. However, the input and output were adapted to fit the 3D convolutional filters. We repeated the experiments with the same layers as the 2D model and added another layer architecture, referred to as DSTS layer, described in Section 2.1.9. The DSTS model had the best performance, improving in all tumor sizes compared to any 3D model, but it still had issues that were not present in some 2D models.

Both 2D and 3D models were able to create segmentation predictions in less than 1 second per slice, on a machine without using a GPU. However, 3D models were almost ten times slower than their 2D counterparts, even with a lower number of parameters and FLOP count. We attribute this slowdown to the number of dependencies created by the 3D convolutions. Although the 3D models had some advantages over the 2D models, they were not enough to justify the reduced performance. A better approach could be a hybrid model, as described by Li et al. [36] and Zhang et al. [68].

Based on the results of our work, we can answer the research questions described in Section 1.4:

- Can we develop a model that balances efficiency and effectiveness for the liver tumor segmentation problem?

  Our models were not able to compete with state-of-the-art approaches. However, they were able to achieve proper results with a fraction of the number of parameters.

- How optimized the EfficientNet is for the tumor liver segmentation problem?

  The EfficientNet is not the most adequate architecture for liver tumor segmentation. The EfficientNet with its default filter sizes had the best segmentation performance for large tumors. However, it lacks detailed information to detect smaller tumors. We were able to improve the detection of small tumors by redistributing resources to the higher layers of the model.

- Does our model improve with the use of 3D convolutions?

  It was possible to notice a performance improvement between our 2D model using MobileNet-V2 layers (EfficientNet as encoder) and our model using the DSTS layer. However, there are situations in which the 2D model performed better than any of our 3D models, which suggests that a hybrid model that combines 2D and 3D convolutions can produce better results.

- Can our model viably run on machines without high-end hardware?

  According to our experiments, our largest model was able to segment liver tumors without using GPUs with an execution time of less than one second per slice.

As directions for future work, we intend to change the framework of our architecture. For instance, satisfactory results with Mask-RCNN were presented by Wang et al. [60], where this network uses ResNet-50 as backbone, which could be easily replaced by one of the encoders that we proposed in this work, reducing the computational cost of the model.

From Figure 4.6, we can observe that the 2D model has an advantage in locating smaller tumors. Therefore, an adaptation of the EfficientNet architecture that uses 2D convolutions for earlier layers and 3D convolutions for deeper layers could improve the detection of small tumors, while maintaining a low computation cost.

Finally, a great benefit of EfficientNet is its balance between computational resources and scalability. However, its parameters are better tuned for the image classification problem. A large study with 3D convolutions can benefit the overall performance of the model.

Bibliography

[1] O. Abdel-Hamid, L. Deng, and D. Yu. Exploring Convolutional Neural Network Structures and Optimization Techniques for Speech Recognition. In *Interspeech*, volume 2013, pages 1173–5, 2013.

[2] K. S. Albain, R. S. Swann, V. W. Rusch, A. T. Turrisi III, F. A. Shepherd, C. Smith, Y. Chen, R. B. Livingston, R. H. Feins, and D. R. Gandara. Radiotherapy plus Chemotherapy with or Without Surgical Resection for Stage III Non-Small-Cell Lung Cancer: a Phase III Randomised Controlled Trial. *The Lancet*, 374(9687):379–386, 2009.

[3] A. Antoniou, A. Storkey, and H. Edwards. Data Augmentation Generative Adversarial Networks. *arXiv preprint arXiv:1711.04340*, 2017.

[4] S. Bengio and G. Heigold. Word Embeddings for Speech Recognition. In *15th Conference of the International Speech Communication Association*, pages 1–5. Interspeech, 2014.

[5] P. Bilic, P. F. Christ, E. Vorontsov, G. Chlebus, H. Chen, Q. Dou, C.-W. Fu, X. Han, P.-A. Heng, and J. Hesser. The Liver Tumor Segmentation Benchmark (LiTS). *arXiv preprint arXiv:1901.04056*, 2019.

[6] H. Chen, Z. Qin, Y. Ding, L. Tian, and Z. Qin. Brain Tumor Segmentation with Deep Convolutional Symmetric Neural Network. *Neurocomputing*, 392:305–313, 2020.

[7] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFS. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2017.

[8] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking Atrous Convolution for Semantic Image Segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[9] S. Chen, C. Ding, and M. Liu. Dual-Force Convolutional Neural Networks for Accurate Brain Tumor Segmentation. *Pattern Recognition*, 88:90–100, 2019.

[10] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 424–432. Springer, 2016.

[11] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. Ieee, 2009.

[13] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

[14] L. R. Dice. Measures of the Amount of Ecologic Association Between Species. *Ecology*, 26(3):297–302, 1945.

[15] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[16] R. C. Gonzalez, R. E. Woods, and S. L. Eddins. *Digital Image Processing Using MATLAB*. Pearson Education India, 2004.

[17] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, Large Minibatch SGD: Training Imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[18] S. Han, J. Pool, J. Tran, and W. Dally. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[19] X. Han. Automatic Liver Lesion Segmentation Using a Deep Convolutional Neural Network Method. *arXiv preprint arXiv:1704.07239*, 2017.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep Into Rectifiers: Surpassing Human-Level Performance on Imangenet Classification. In *IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[21] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[22] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *IEEE International Conference on Computer Vision*, pages 2961–2969, 2017.

[23] G. N. Hounsfield. Computerized Transverse Axial Scanning (Tomography): Part 1. Description of System. *The British Journal of Radiology*, 46(552):1016–1022, 1973.

[24] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, and V. Vasudevan. Searching for Mobilenetv3. In *IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

[25] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*, 2017.

[26] J. Hu, L. Shen, and G. Sun. Squeeze-and-Excitation Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018.

[27] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and$<$ 0.5 MB Model Size. *arXiv preprint arXiv:1602.07360*, 2016.

[28] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint arXiv:1502.03167*, 2015.

[29] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson. Averaging Weights Leads to Wider Optima and Better Generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[30] P. Jaccard. Distribution de la Flore Alpine Dans le Bassin des Dranses et Dans Quelques Régions Voisines. *Bull Soc Vaudoise Sci Nat*, 37:241–272, 1901.

[31] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[32] O. Köpüklü, N. Kose, A. Gunduz, and G. Rigoll. Resource Efficient 3D Convolutional Neural Networks. In *IEEE/CVF International Conference on Computer Vision Workshop*, pages 1910–1919. IEEE, 2019.

[33] A. Krizhevsky and G. Hinton. Convolutional Deep Belief Networks on CIFAR-10. *Unpublished manuscript*, 40(7):1–9, 2010.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[35] X. Li, Z. Liu, P. Luo, C. Change Loy, and X. Tang. Not All Pixels are Equal: Difficulty-Aware Semantic Segmentation Via Deep Layer Cascade. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3193–3202, 2017.

[36] X. Li, H. Chen, X. Qi, Q. Dou, C.-W. Fu, and P.-A. Heng. H-DenseUNet: Hybrid Densely Connected UNet for Liver and Tumor Segmentation from CT Volumes. *IEEE Transactions on Medical Imaging*, 37(12):2663–2674, 2018.

[37] LiTS. Liver Tumor Segmentation Challenge, 2018. `https://competitions.codalab.org/competitions/17094`.

[38] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han. On the Variance of the Adaptive Learning Rate and Beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[39] T. Livraghi, A. Giorgio, G. Marin, A. Salmi, I. De Sio, L. Bolondi, M. Pompili, F. Brunello, S. Lazzaroni, and G. Torzilli. Hepatocellular Carcinoma and Cirrhosis in 746 Patients: Long-Term Results of Percutaneous Ethanol Injection. *Radiology*, 197(1):101–108, 1995.

[40] L. Lu, X. Wang, G. Carneiro, and L. Yang. *Deep Learning and Convolutional Neural Networks for Medical Imaging and Clinical Informatics*. Springer, 2019.

[41] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. Shufflenet v2: Practical Guidelines for Efficient CNN Architecture Design. In *European Conference on Computer Vision*, pages 116–131, 2018.

[42] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *30th International Conference on Machine Learning*, volume 28, pages 1–6, 2013.

[43] W. S. McCulloch and W. Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[44] F. Milletari, N. Navab, and S.-A. Ahmadi. V-net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. In *Fourth International Conference on 3D Vision*, pages 565–571. IEEE, 2016.

[45] V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *27th International Conference on Machine Learning*, pages 807–814, 2010.

[46] P. Ramachandran, B. Zoph, and Q. V. Le. Swish: a Self-Gated Activation Function. *arXiv preprint arXiv:1710.05941*, 7, 2017.

[47] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[48] S. Rossi, M. Di Stasi, E. Buscarini, P. Quaretti, F. Garbagnati, L. Squassante, C. Paties, D. Silverman, and L. Buscarini. Percutaneous RF Interstitial Thermal Ablation in the Treatment of Hepatic Cancer. *AJR. American Journal of Roentgenology*, 167(3):759–768, 1996.

[49] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Representations by Back-Propagating Errors. *Nature*, 323(6088):533–536, 1986.

[50] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted Residuals and Linear Bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[51] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[52] M. Tan and Q. V. Le. Efficientnet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv preprint arXiv:1905.11946*, 2019.

[53] P. Tang, Q. Liang, X. Yan, S. Xiang, W. Sun, D. Zhang, and G. Coppola. Efficient Skin Lesion Segmentation Using Separable-Unet With Stochastic Weight Averaging. *Computer Methods and Programs in Biomedicine*, 178:289–301, 2019.

[54] A. Tao, K. Sapra, and B. Catanzaro. Hierarchical Multi-Scale Attention for Semantic Segmentation. *arXiv preprint arXiv:2005.10821*, 2020.

[55] K. D. Toennies. *Guide to Medical Image Analysis*. Springer, 2017.

[56] H. Touvron, A. Vedaldi, M. Douze, and H. Jégou. Fixing the Train-Test Resolution Discrepancy: FixEfficientNet. *arXiv preprint arXiv:2003.08237*, 2020.

[57] P. Tschandl, C. Sinz, and H. Kittler. Domain-Specific Classification-Pretrained Fully Convolutional Network Encoders for Skin Lesion Segmentation. *Computers in Biology and Medicine*, 104:111–116, 2019.

[58] G. Wang, W. Li, T. Vercauteren, and S. Ourselin. Automatic Brain Tumor Segmentation Based on Cascaded Convolutional Neural Networks with Uncertainty Estimation. *Frontiers in Computational Neuroscience*, 13:56, 2019.

[59] X. Wang, R. Girshick, A. Gupta, and K. He. Non-Local Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7794–7803, 2018.

[60] X. Wang, S. Han, Y. Chen, D. Gao, and N. Vasconcelos. Volumetric Attention for 3D Medical Image Segmentation and Detection. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 175–184. Springer, 2019.

[61] J. Wei, Y. Xia, and Y. Zhang. M3Net: A Multi-Model, Multi-Size, and Multi-View Deep Neural Network for Brain Magnetic Resonance Image Segmentation. *Pattern Recognition*, 91:366–378, 2019.

[62] WHO. Latest Global Cancer Data, 2018. `https://www.who.int/cancer/PRGlobocanFinal.pdf`.

[63] B. Widrow. *Adaptive "Adaline" Neuron Using Chemical "Memistors"*. Stanford Electronics Laboratories, 1960.

[64] F. Xie, J. Yang, J. Liu, Z. Jiang, Y. Zheng, and Y. Wang. Skin Lesion Segmentation Using High-Resolution Convolutional Neural Network. *Computer Methods and Programs in Biomedicine*, 186:105241, 2020.

[65] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le. Self-Training with Noisy Student Improves Imagenet Classification. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10687–10698, 2020.

[66] A. Yang, X. Yang, W. Wu, H. Liu, and Y. Zhuansun. Research on Feature Extraction of Tumor Image Based on Convolutional Neural Network. *IEEE Access*, 7:24204–24213, 2019.

[67] Y. Yuan. Hierarchical Convolutional-Deconvolutional Neural Networks for Automatic Liver and Tumor Segmentation. *arXiv preprint arXiv:1710.04540*, 2017.

[68] J. Zhang, Y. Xie, P. Zhang, H. Chen, Y. Xia, and C. Shen. Light-Weight Hybrid Convolutional Network for Liver Tumour Segmentation. In *28th International Joint Conference on Artificial Intelligence, Macao, China*, pages 10–16, 2019.

[69] M. Zhang, J. Lucas, J. Ba, and G. E. Hinton. Lookahead Optimizer: k Steps Forward, 1 Step Back. In *Advances in Neural Information Processing Systems*, pages 9593–9604, 2019.

[70] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

[71] Q. Zhu, B. Du, and P. Yan. Boundary-Weighted Domain Adaptive Neural Network for Prostate MR Image Segmentation. *IEEE Transactions on Medical Imaging*, 39 (3):753–763, 2019.

[72] S. W. Zucker. Region Growing: Childhood and Adolescence. *Computer Graphics and Image Processing*, 5(3):382–399, 1976.

Appendix A

Additional Implementation Details

In this appendix, we present the dataset protocol and model hyperparameters that were defined in our work in order to allow reproducibility of our experimental results.

A.1 Dataset Protocol

Tables A.1 and A.2 present the training and validation splits, respectively, used in our experiments.

A.2 Model Hyperparameters

Tables A.3, to A.11 show the hyperparameters used in the 2D models. Tables A.12 to A.23 show the hyperparameters used in 3D models.

Filename	# Tumors	# Voxels	Filename	# Tumors	# Voxels	Filename	# Tumors	# Voxels
volume-0	11	2591	volume-1	12	7194	volume-10	6	24310
volume-101	17	378439	volume-102	1	47754	volume-103	2	75787
volume-104	6	275918	volume-105	0	0	volume-106	0	0
volume-107	4	5953	volume-108	66	1455033	volume-11	5	10549
volume-110	17	124406	volume-111	6	7300	volume-113	29	84270
volume-114	0	0	volume-115	0	0	volume-116	1	611132
volume-117	75	1034092	volume-118	10	419119	volume-119	0	0
volume-12	1	790	volume-120	6	5231	volume-122	17	79803
volume-123	5	228251	volume-124	14	72092	volume-126	1	5090
volume-127	1	298	volume-128	11	326553	volume-13	9	23651
volume-130	23	1383692	volume-14	1	2612	volume-15	5	875
volume-16	10	373122	volume-17	5	37676	volume-18	4	10210
volume-19	6	18528	volume-2	1	14131	volume-20	3	1669
volume-21	5	36271	volume-22	1	9424	volume-23	1	42904
volume-26	7	51809	volume-27	11	187743	volume-28	7	214268
volume-29	1	14062	volume-3	1	2793	volume-31	8	6736
volume-33	15	434721	volume-34	0	0	volume-36	2	39484
volume-39	1	348986	volume-40	17	122480	volume-41	0	0
volume-42	2	1638	volume-43	1	7337	volume-44	2	185990
volume-45	1	4539	volume-46	48	41416	volume-47	0	0
volume-48	4	35478	volume-49	22	8015	volume-5	1	540
volume-51	10	124895	volume-52	2	18296	volume-53	2	2261
volume-54	1	513	volume-55	1	2797	volume-57	2	6604
volume-58	5	1443	volume-59	1	697	volume-6	7	31467
volume-60	1	11368	volume-61	2	2916	volume-62	2	3196
volume-63	5	688	volume-64	1	179093	volume-65	1	1670
volume-66	5	2333	volume-67	1	525	volume-68	1	4888
volume-7	6	32422	volume-70	3	120665	volume-72	4	22364
volume-73	1	592	volume-74	2	37144	volume-75	4	2703
volume-76	10	193508	volume-78	14	12265	volume-79	10	9085
volume-8	6	20827	volume-80	2	88269	volume-81	1	5130
volume-82	1	141610	volume-83	1	98	volume-85	2	15544
volume-86	2	6051	volume-87	0	0	volume-88	6	254141
volume-89	0	0	volume-9	6	22179	volume-90	7	250893
volume-91	0	0	volume-92	9	2632	volume-93	30	348088
volume-94	14	62525	volume-95	3	2581	volume-96	23	40533
volume-97	12	725310	volume-98	17	608639	volume-99	13	12253

Table A.1: Volume information of the training dataset.

Filename	# Tumors	# Voxels	Filename	# Tumors	# Voxels	Filename	# Tumors	# Voxels
volume-100	11	1569689	volume-109	28	72822	volume-112	4	1181
volume-121	3	2123	volume-125	1	754	volume-129	10	1909785
volume-24	1	1422	volume-25	1	454	volume-30	1	14252
volume-32	0	0	volume-38	0	0	volume-35	3	20318
volume-37	8	14981	volume-4	5	1516253	volume-50	1	3014
volume-56	2	268297	volume-69	1	3113	volume-71	7	142741
volume-77	2	4234	volume-84	11	541305			

Table A.2: Volume information of the validation dataset.

Layers	Repetitions	Filter Size	Stride	Expansion Ratio	In Feature	Out Feature	SE Squeeze
Layer1	1	3	(1, 1)	1	32	16	0.25
Layer2	2	3	(2, 2)	6	16	24	0.25
Layer3	2	5	(2, 2)	6	24	40	0.25
Layer4	3	3	(2, 2)	6	40	80	0.25
Layer5	3	5	(1, 1)	6	80	112	0.25
Layer6	4	5	(2, 2)	6	112	192	0.25
Layer7	1	3	(1, 1)	6	192	320	0.25

Table A.3: Default hyperparameters used to construct the 2D encoder with the MobileNet-V2 block. Same hyperparameters as the work described by Tan and Le [52].

Layers	Repetitions	Filter Size	Stride	Expansion Ratio	In Feature	Out Feature	SE Squeeze
Layer1	1	3	(1, 1)	1	32	16	0.25
Layer2	1	3	(2, 2)	6	16	24	0.25
Layer3	1	5	(2, 2)	6	24	40	0.25
Layer4	3	3	(2, 2)	6	40	80	0.25
Layer5	4	5	(1, 1)	6	80	112	0.25
Layer6	4	5	(2, 2)	6	112	192	0.25
Layer7	2	3	(1, 1)	6	192	320	0.25

Table A.4: Hyperparameters used to construct the 2D encoder with the MobileNet-V2 block focusing resources on the deeper layers.

Layers	Repetitions	Filter Size	Stride	Expansion Ratio	In Feature	Out Feature	SE Squeeze
Layer1	2	3	(1, 1)	1	32	16	0.25
Layer2	2	3	(2, 2)	6	16	24	0.25
Layer3	2	5	(2, 2)	6	24	40	0.25
Layer4	2	3	(2, 2)	6	40	80	0.25
Layer5	2	5	(1, 1)	6	80	112	0.25
Layer6	3	5	(2, 2)	6	112	192	0.25
Layer7	1	3	(1, 1)	6	192	320	0.25

Table A.5: Hyperparameters used to construct the 2D encoder with the MobileNet-V2 block focusing resources on the shallower layers.

Layers	Repetitions	Filter Size	Stride	Expansion Ratio	In Feature	Out Feature	SE Squeeze
Layer1	1	3	(1, 1)	1	32	16	0.00
Layer2	2	3	(2, 2)	4	16	24	0.00
Layer3	2	5	(2, 2)	3	24	40	0.25
Layer4	3	3	(2, 2)	3	40	80	0.00
Layer5	3	5	(1, 1)	6	80	112	0.25
Layer6	4	5	(2, 2)	6	112	192	0.25
Layer7	1	3	(1, 1)	6	192	320	0.25

Table A.6: Default hyperparameters used to construct the 2D encoder with the MobileNet-V3. Values are based on the work described by Howard et al. [24].

Layers	Repetitions	Filter Size	Stride	Expansion Ratio	In Feature	Out Feature	SE Squeeze
Layer1	1	3	(1, 1)	1	32	16	0.00
Layer2	1	3	(2, 2)	4	16	24	0.00
Layer3	2	5	(2, 2)	3	24	40	0.25
Layer4	3	3	(2, 2)	3	40	80	0.00
Layer5	4	5	(1, 1)	6	80	112	0.25
Layer6	4	5	(2, 2)	6	112	192	0.25
Layer7	3	3	(1, 1)	6	192	320	0.25

Table A.7: Hyperparameters used to construct the 2D encoder with the MobileNet-V3 block focusing resources on the deep layers.

Layers	Repetitions	Filter Size	Stride	Expansion Ratio	In Feature	Out Feature	SE Squeeze
Layer1	2	3	(1, 1)	1	32	16	0.00
Layer2	2	3	(2, 2)	4	16	24	0.00
Layer3	2	5	(2, 2)	3	24	40	0.25
Layer4	2	3	(2, 2)	3	40	80	0.00
Layer5	2	5	(1, 1)	6	80	112	0.25
Layer6	3	5	(2, 2)	6	112	192	0.25
Layer7	1	3	(1, 1)	6	192	320	0.25

Table A.8: Hyperparameters used to construct the 2D encoder with the MobileNet-V3 block focusing resources on the shallower layers.

Layers	Repetitions	Filter Size	Stride	In Feature	Out Feature	SE Squeeze
Layer1	1	3	(1, 1)	32	16	0.25
Layer2	2	3	(2, 2)	16	24	0.25
Layer3	2	5	(2, 2)	24	40	0.25
Layer4	3	3	(2, 2)	40	80	0.25
Layer5	3	5	(1, 1)	80	112	0.25
Layer6	4	5	(2, 2)	112	192	0.25
Layer7	1	3	(1, 1)	192	320	0.25

Table A.9: Default hyperparameters used to construct the 2D encoder with the ShuffleNet-V2. Values based on MobileNet-V2 hyperparameters.

Layers	Repetitions	Filter Size	Stride	In Feature	Out Feature	SE Squeeze
Layer1	1	3	(1, 1)	32	16	0.25
Layer2	1	3	(2, 2)	16	24	0.25
Layer3	1	5	(2, 2)	24	40	0.25
Layer4	3	3	(2, 2)	40	80	0.25
Layer5	4	5	(1, 1)	80	112	0.25
Layer6	4	5	(2, 2)	112	192	0.25
Layer7	2	3	(1, 1)	192	320	0.25

Table A.10: Hyperparameters used to construct the 2D encoder with the ShuffleNet-V2 block focusing resources on the deeper layers.

Layers	Repetitions	Filter Size	Stride	In Feature	Out Feature	SE Squeeze
Layer1	3	3	(1, 1)	32	16	0.25
Layer2	3	3	(2, 2)	16	24	0.25
Layer3	2	5	(2, 2)	24	40	0.25
Layer4	2	3	(2, 2)	40	80	0.25
Layer5	2	5	(1, 1)	80	112	0.25
Layer6	2	5	(2, 2)	112	192	0.25
Layer7	1	3	(1, 1)	192	320	0.25

Table A.11: Hyperparameters used to construct the 2D encoder with the ShuffleNet-V2 block focusing resources on the shallower layers.

Layers	Repetitions	Filter Size	Stride	Expansion Ratio	In Feature	Out Feature	SE Squeeze
Layer1	1	3	(1, 1, 1)	1	16	8	0.25
Layer2	2	3	(1, 2, 2)	6	8	12	0.25
Layer3	2	5	(2, 2, 2)	6	12	20	0.25
Layer4	3	3	(2, 2, 2)	6	20	40	0.25
Layer5	3	5	(1, 1, 1)	6	40	56	0.25
Layer6	4	5	(1, 2, 2)	6	56	96	0.25
Layer7	1	3	(1, 1, 1)	6	96	160	0.25

Table A.12: Hyperparameters used to construct the 3D encoder with the MobileNet-V2 block, based on the hyperparameters described by [52], but adapted to 3D convolutions.

Layers	Repetitions	Filter Size	Stride	Expansion Ratio	In Feature	Out Feature	SE Squeeze
Layer1	1	3	(1, 1, 1)	1	16	8	0.25
Layer2	2	3	(1, 2, 2)	6	8	12	0.25
Layer3	2	5	(2, 2, 2)	6	12	20	0.25
Layer4	3	3	(2, 2, 2)	6	20	40	0.25
Layer5	4	5	(1, 1, 1)	6	40	56	0.25
Layer6	4	5	(1, 2, 2)	6	56	96	0.25
Layer7	4	3	(1, 1, 1)	6	96	160	0.25

Table A.13: Hyperparameters used to construct the 3D encoder with the MobileNet-V2 block focusing resources on the deeper layers.

Layers	Repetitions	Filter Size	Stride	Expansion Ratio	In Feature	Out Feature	SE Squeeze
Layer1	2	3	(1, 1, 1)	1	16	8	0.25
Layer2	2	3	(1, 2, 2)	6	8	12	0.25
Layer3	2	5	(2, 2, 2)	6	12	20	0.25
Layer4	2	3	(2, 2, 2)	6	20	40	0.25
Layer5	2	5	(1, 1, 1)	6	40	56	0.25
Layer6	3	5	(1, 2, 2)	6	56	96	0.25
Layer7	1	3	(1, 1, 1)	6	96	160	0.25

Table A.14: Hyperparameters used to construct the 3D encoder with the MobileNet-V2 block focusing resources on the shallower layers.

Layers	Repetitions	Filter Size	Stride	Expansion Ratio	In Feature	Out Feature	SE Squeeze
Layer1	1	3	(1, 1, 1)	1	16	12	0.0
Layer2	2	3	(1, 2, 2)	4	12	12	0.0
Layer3	3	5	(2, 2, 2)	3	12	20	0.25
Layer4	3	3	(2, 2, 2)	3	20	40	0.0
Layer5	3	5	(1, 1, 1)	6	40	56	0.25
Layer6	4	5	(1, 2, 2)	6	56	96	0.25
Layer7	2	3	(1, 1, 1)	6	96	160	0.25

Table A.15: Default hyperparameters used to construct the 3D encoder with the MobileNet-V3. Values are based on the work described by [24], but adapted to 3D convolutions.

Layers	Repetitions	Filter Size	Stride	Expansion Ratio	In Feature	Out Feature	SE Squeeze
Layer1	1	3	(1, 1, 1)	1	16	12	0.0
Layer2	2	3	(1, 2, 2)	4	12	12	0.0
Layer3	3	5	(2, 2, 2)	3	12	20	0.25
Layer4	3	3	(2, 2, 2)	3	20	40	0.0
Layer5	4	5	(1, 1, 1)	6	40	56	0.25
Layer6	4	5	(1, 2, 2)	6	56	96	0.25
Layer7	5	3	(1, 1, 1)	6	96	160	0.25

Table A.16: Hyperparameters used to construct the 3D encoder with the MobileNet-V3 block focusing resources on the deep layers.

Layers	Repetitions	Filter Size	Stride	Expansion Ratio	In Feature	Out Feature	SE Squeeze
Layer1	2	3	(1, 1, 1)	1	16	12	0.0
Layer2	2	3	(1, 2, 2)	4	12	12	0.0
Layer3	2	5	(2, 2, 2)	3	12	20	0.25
Layer4	2	3	(2, 2, 2)	3	20	40	0.0
Layer5	2	5	(1, 1, 1)	6	40	56	0.25
Layer6	2	5	(1, 2, 2)	6	56	96	0.25
Layer7	1	3	(1, 1, 1)	6	96	160	0.25

Table A.17: Hyperparameters used to construct the 3D encoder with the MobileNet-V3 block focusing resources on the shallower layers.

Layers	Repetitions	Filter Size	Stride	In Feature	Out Feature	SE Squeeze
Layer1	1	3	(1, 1)	32	16	0.25
Layer2	2	3	(2, 2)	16	24	0.25
Layer3	2	5	(2, 2)	24	40	0.25
Layer4	3	3	(2, 2)	40	80	0.25
Layer5	3	5	(1, 1)	80	112	0.25
Layer6	4	5	(2, 2)	112	192	0.25
Layer7	1	3	(1, 1)	192	320	0.25

Table A.18: Default hyperparameters used to construct the 3D encoder with the ShuffleNet-V2.

Layers	Repetitions	Filter Size	Stride	In Feature	Out Feature	SE Squeeze
Layer1	2	3	(1, 1, 1)	16	16	0.25
Layer2	2	3	(1, 2, 2)	16	32	0.25
Layer3	2	5	(2, 2, 2)	32	40	0.25
Layer4	3	3	(2, 2, 2)	40	80	0.25
Layer5	3	5	(1, 1, 1)	80	112	0.25
Layer6	4	5	(1, 2, 2)	112	192	0.25
Layer7	2	3	(1, 1, 1)	192	320	0.25

Table A.19: Hyperparameters used to construct the 3D encoder with the ShuffleNet-V2 block focusing resources on the deeper layers.

Layers	Repetitions	Filter Size	Stride	In Feature	Out Feature	SE Squeeze
Layer1	1	3	(1, 1, 1)	16	16	0.25
Layer2	2	3	(1, 2, 2)	16	32	0.25
Layer3	4	5	(2, 2, 2)	32	40	0.25
Layer4	4	3	(2, 2, 2)	40	80	0.25
Layer5	4	5	(1, 1, 1)	80	112	0.25
Layer6	5	5	(1, 2, 2)	112	192	0.25
Layer7	5	3	(1, 1, 1)	192	320	0.25

Table A.20: Hyperparameters used to construct the 3D encoder with the ShuffleNet-V2 block focusing resources on the shallower layers.

Layers	Repetitions	Filter Size	Stride	Expansion Ratio	In Feature	Out Feature	SE Squeeze
Layer1	3	3	(1, 1, 1)	16	24	0.25	
Layer2	4	3	(1, 2, 2)	24	32	0.25	
Layer3	4	5	(2, 2, 2)	32	40	0.25	
Layer4	4	3	(2, 2, 2)	40	80	0.25	
Layer5	4	5	(1, 1, 1)	80	112	0.25	
Layer6	4	5	(1, 2, 2)	112	192	0.25	
Layer7	4	3	(1, 1, 1)	192	320	0.25	

Table A.21: Default hyperparameters used to construct the 3D encoder with the DSTS block.

Layers	Repetitions	Filter Size	Stride	Expansion Ratio	In Feature	Out Feature	SE Squeeze
Layer1	2	3	(1, 1, 1)	16	24	0.25	
Layer2	4	3	(1, 2, 2)	24	32	0.25	
Layer3	4	5	(2, 2, 2)	32	40	0.25	
Layer4	4	3	(2, 2, 2)	40	80	0.25	
Layer5	4	5	(1, 1, 1)	80	112	0.25	
Layer6	5	5	(1, 2, 2)	112	192	0.25	
Layer7	5	3	(1, 1, 1)	192	320	0.25	

Table A.22: Hyperparameters used to construct the 3D encoder with the DSTS block focusing resources on the deeper layers.

Layers	Repetitions	Filter Size	Stride	Expansion Ratio	In Feature	Out Feature	SE Squeeze
Layer1	4	3	(1, 1, 1)	16	24	0.25	
Layer2	3	3	(1, 2, 2)	24	32	0.25	
Layer3	2	5	(2, 2, 2)	32	40	0.25	
Layer4	2	3	(2, 2, 2)	40	80	0.25	
Layer5	2	5	(1, 1, 1)	80	112	0.25	
Layer6	2	5	(1, 2, 2)	112	192	0.25	
Layer7	2	3	(1, 1, 1)	192	320	0.25	

Table A.23: Hyperparameters used to construct the 3D encoder with the DSTS block focusing resources on the shallower layers.