KU LEUVEN

Lightweight Convolutional Neural Networks for Brain Tumor Segmentation

Dr. Wouter Durnez

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Engineering and Computer Science

Thesis supervisors:
Prof. dr. ir. Sabine Van Huffel
Prof. dr. ir. Frederik Maes

Assessor:
Dr. ir. Christos Chatzichristos

Mentor:
Ir. Pooya Ashtari

Academic year 2020 – 2021

Preface

This may be my final chance to immortalize a word of thanks in an academic work, so I better make it count.

Nathalie, you are, and you will always be, the absolute love of my life. To do what you mean to me justice, I would have to write a thousand more theses, just so I could express that very thought in each preface (but I think you'll agree, two is enough).

Ellie, Juno, ik heb geprobeerd, maar ik kan niet in woorden vatten hoe belangrijk jullie voor mij zijn. Jullie zijn, gewoon buitengewoon, mijn hele wereld. Papa ziet jullie enorm graag.

A special thanks to prof. Van Huffel, for giving me this chance. I greatly appreciated you accepting me, a psychology major with two kids and a full-time job, as your thesis student. I also appreciate you acknowledging some of the silliness I couldn't help myself from inserting into presentations. (I could swear I at least heard a snicker at one point.)

پویا عزیز
با درود فراوان
ضمن تشکر بینهایت در خصوص کمک بی دریغ شما درمقابل سوال های فراوان من و پاسخ های بسیار دقیق و شفافتان سپاسگزارم .
باید خاطرنشان کنم که بردباری و صبرو حوصله ای که در مقابل بمباران ایمل هایم ار خود نشان دادید بسیار قابل تحسین است چون از تمام ایمیل هایم به قطوری یک کتاب رمان می توانست باشد (داستان یک خرمن مثنوی) .
در ضمن باید اشاره کنم که از شما بینهایت اموختم هم در مورد موضوع تزم و هم از نظر دانشی که در خصوص کامپیوتر بسیار گسترده و عمیق کاملا منطبق با علم .
شما برای من یک معلمی هستید که می توانستم در رویاهایم تصور کنم و همیشه بعنوان یک معلم فرهیخته برای من در خاطراتم باقی خواهید ماند .
از صمیم قلبم بهترین ارزوها را برای اتمام دوران تحصیلات عالیه دکترایتان و در کل زندگی شخصی تان دارم.
اینده بسیار روشن و سلامتی برایتان ارزو می کنم .

I also want to thank my copromotor, Frederik, and assessor, Christos, for the time they will invest/have invested in reading this thesis. Reading theses in summertime is not my favorite occupation either, so I sincerely appreciate the effort.

Anand, much obliged for ironing out whatever kinks you could find in this text. Live long and prosper!

The computational resources (Stevin Supercomputer Infrastructure) and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by Ghent University, FWO and the Flemish Government - department EWI.

Quite the prosaic athleticism I'm exercising here, squeezing you in again after a supercomputer acknowledgement, but Nathalie, once more for good measure: I love you!

Dr. Wouter Durnez

Contents

Abstract

Present-day medical imaging technologies yield large quantities of multi-modal data. Machine learning techniques can be used to help process this information. Semantic segmentation—the task of annotating pixels or voxels with class labels—has been tackled using a variety of traditional approaches. In recent years, however, progress within the computer vision domain has accelerated due to the advent of deep learning. A particular milestone was the genesis of convolutional neural networks, capable of processing input patches in a hierarchically structured manner. Nevertheless, such techniques are not without drawbacks. Convolutional layers are defined by weight tensors that, depending on the kernel size and number of input and output channels, can accumulate to a point where the total number of network parameters ranges in the millions. In addition, these large networks require significant computational effort, both during training and in the inference phase, hampering their real-time usefulness.

One way to address this issue is to replace convolutional weight tensors with low-rank constrained alternatives. To this effect, tensor network formats can be used to reparametrize the weight tensors in a rank-restricted manner, effectively imposing a degree of compression. In addition to allowing the network's memory demand and computational strain to be reduced, these sparse data formats leave less room for overfitting, encouraging the learning of generalizable representations instead.

In this work, we evaluated the impact of different tensor network parametrizations: the Canonical Polyadic format, the Tucker format, and two versions of the Tensor Train format. Building off of a U-Net architecture (the baseline network), convolutional layers were replaced with the aforementioned low-rank alternatives across a range of eight different compression rates. Our data indicated that, while this approach can significantly downscale the total number of network parameters, a minimal layer compression rate ($\times 10$) must be observed to lower the computational strain induced by the network. However, doing so has a non-negligible impact on the networks' performances compared to baseline, as indicated by the decrease in Dice score and increase in 95% Hausdorff distance. The most promising approach, all things considered, was the use of the Tensor Train format—a parametrization that yielded reduced, yet reasonable performance up to a layer compression rate of 20, while lessening the required computational effort.

In sum, these results suggest that lightweight convolutional layers can be leveraged to relax a network's computational and memory demands. Their application, however, comes at a performance cost, rendering them ill-suited for medical segmentation tasks. Nonetheless, other use cases may exist where computational feasibility is valued to the point where a small lapse in performance is acceptable.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

CNN	Convolutional neural network
CP	Canonical polyadic (format)
CPD	Canonical polyadic decomposition
CPU	Central processing unit
CT	Computerized tomography
FCN	Fully convolutional network
FLAIR	Fluid attenuated inversion recovery
FLOP	Floating point operation
GPU	Graphics processing unit
LR	Learning rate
MAC	Multiply-accumulate operation
MRI	Magnetic resonance imaging
PET	Positron emission tomography
RF	Radio frequency
RT	Repetition time
TNN	Tensorized neural network
TE	Time to echo
TT	Tensor train (format)

It's not a tumor!

Lacking conclusive medical evidence, Arnold Schwarzenegger is left to debate a child on the nature of his head bump.
Kindergarten Cop (1990)

Chapter 1

Introduction

1.1 Context

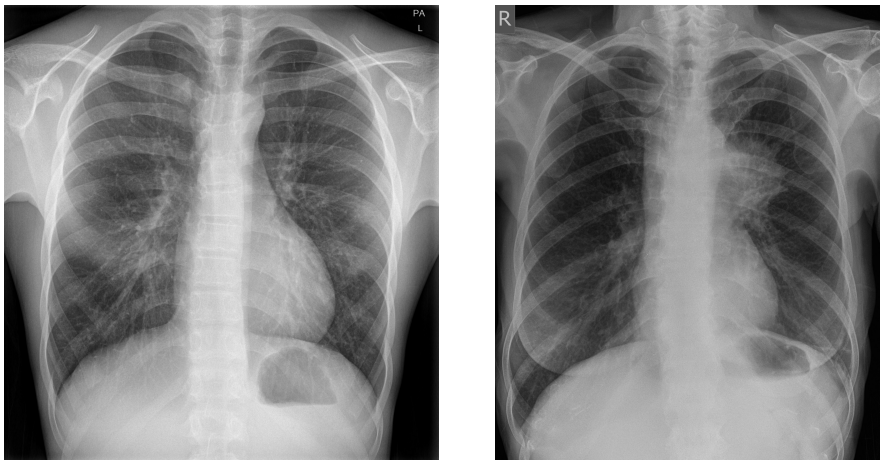
On November 8, 1895, dr. Wilhelm Conrad Röntgen haphazardly stumbled upon the X-ray [60]. While experimentally passing electrical discharge through various types of vacuum equipment, Röntgen noticed how an unidentified type of cathode ray—hence the 'X'—left a fluorescent mark on a small cardboard screen coated with barium platinocyanide paint. Intrigued by this unexpected observation, he continued his experiments, ushering in a new era of medical imaging.

Since then, our arsenal of medical imaging methods has greatly expanded [73]. Traditional X-ray radiography has been replaced by more advanced techniques, many of which have become commonplace. Mammography—a technique that makes use of low-energy X-rays—is routinely used as a screening instrument in the battle against breast cancer. Complex bone fractures can be quickly charted using X-ray computed tomography (CT) [15], a more sophisticated sibling to the conventional X-ray. In turn, positron emission tomography (PET) can, in conjunction with radioactive tracers, shed light on the metabolic function of different organs [82].

The data yielded by these imaging techniques, information-rich as it may be, remain useless without proper interpretation. The human body is quite the intricate machine. Medical experts, imbued with a deep anatomical knowledge and familiarity with the particularities of each imaging technique, are necessary to make sense of these images. Even an image as 'simple' as an X-ray—a single two-dimensional black-and-white image—can be highly difficult to gauge for abnormalities (fig. 1.1). Moreover, as our range of techniques widens, the amount of data they collectively yield increases. Contemporary diagnostic protocols can capture images in three dimensions, often combining multiple techniques (e.g., CT and PET) and multiple modalities (e.g., the use of different contrast agents). Consequently, the strain on the radiologist increases, as demand for their expertise keeps growing.

AI to the rescue

The advent of machine learning, and convolutional neural networks (CNN) [49] in particular, bore the promise of lightening that load. Such networks can be trained for the purpose



(a) (b)

Figure 1.1: **Nontrivial interpretation of medical images.** Image (a) shows a normal chest X-ray. Image (b) shows an X-ray with indications of hilar lung cancer. Or was it the other way around? Images courtesy of Dr Derek Smith, Radiopaedia.org, rID: 62093, and Dr Ian Bickle, Radiopaedia.org, rID: 50353.

of semantic segmentation: the classification of individual pixels (2D) or voxels (3D) into different categories (e.g., necrotic tissue vs. normal tissue). If a segmentation network is sufficiently effective, it can conceivably be used to support radiologists in their tasks, partially automating the interpretation of medical imagery. This is particularly useful in an intra-operative scenario, where imaging techniques are used in a (near) real-time manner (e.g., [15]).

1.2 Motivation

In recent years, numerous deep learning architectures have been proposed for the purpose of medical image segmentation. Several of them, including the well-known U-Net architecture designed by Ronneberger et al. [70], have been shown to be particularly robust. A common theme in deep learning, however, is that of overparametrization. In parallel with the growth of our hardware's computational abilities, researchers continuously increased the number and dimensions of hidden (convolutional) layers, as if following an unspoken dictum of 'more is better'. Indeed, overparametrized networks have been shown to exhibit some unexpected advantageous behavior. For instance, Neyshabur et al. [64] discuss how increasing a network's size beyond the requirements to achieve zero training error continues to benefit test performance, suggesting this behavior to be linked to "*some implicit regularization*" introduced by the optimization.

In contrast, Denil et al. [16] demonstrated how such architectures are significantly redundant by greatly constraining the weight matrices with insignificant loss of accu-

racy[1]. These findings suggest that there is room to optimize traditional neural network architectures. An ideal candidate for such optimization is the convolutional layer. Ever since its introduction in 1989 [49], convolutional neural networks have been front and center in the computer vision domain. At the same time, a traditional convolutional layer is associated with a heavy cost, both in terms of storage and the associated number of operations. In a 3D setting, it is characterized by a fifth-order weight tensor (also called the *kernel tensor*):

$$\mathcal{W} \in \mathbb{R}^{C_{in} \times C_{out} \times k_H \times k_W \times k_D} \tag{1.1}$$

In eq. (1.1), C_{in} and C_{out} denote the numbers of input and output channels, respectively, whereas the remaining parameters k_X represent the size of the 3D kernel along every spatial axis (height, width and depth).

Given the dimensionality of $\mathcal{W}$, and the central importance of convolutional layers in CNNs, reducing the number of parameters in this kernel tensor could be particularly beneficial to the performance of said models. There are several potential avenues to achieve this, some of which are described in chapter 3. In this thesis, we zoom in on a specific strategy: the use of low-rank or *lightweight* layers to replace their more expensive full-rank counterparts. More specifically, we discuss how low-rank constraints can be imposed onto convolutional weight tensors using so-called *tensor network* (TN) formats [11].

Tensor networks can be defined as "*special graph structures which break down high-order tensors into a set of sparsely interconnected low-order core tensors*" [11, p.4]. Through this approach, large-scale data can be represented using only a fraction of the original number of values, heavily reducing memory requirements. In addition, it can reduce the number of floating point operations (FLOPS) required to produce the output tensor.

In sum, exploring the use of tensor network formats to generate lightweight convolutional layers is appealing for two main reasons:

1. Doing so reduces a CNN's total parameter count, **reducing its memory footprint**, and
2. We can expect lightweight layers to require fewer calculations to attain the output, improving the network's **computational efficiency**, and resulting in better test-time performance.

1.3 Goals

In this thesis, **we explore how tensor networks can be leveraged to rank-constrain weight tensors, yielding a lightweight variant of the traditional convolutional layers**. We do so by training a baseline network on a popular set of MRI images: the 2020 Multimodal Brain Tumor Image Segmentation (BraTS) benchmark [58]. Then, we replace

[1] Interestingly, increasing the width of a network's hidden layers—i.e., increasing the number of units per layer, rather than the number of layers—also yields surprisingly good results. In experiments conducted by Neal et al. [63], both the bias *and* the variance decreased despite increasing numbers of hidden units (and therefore increased overparametrization), further belying the commonly held 'bias-variance trade-off' principle. Increasing a network's depth is a slightly more complex affair, however, due to the problems associated with deep architectures (e.g., vanishing gradient) [63, appendix C].

convolutional layers in this baseline network with various low-rank alternatives, using different combinations of tensor network formats and compression rates. We henceforth refer to these networks as *tensorized neural networks* (TNN) [66, 12]—i.e., networks in which (a subset of) the weight tensors are generated from lower-order data formats. In our evaluation, all networks were trained under the exact same set of training conditions, and their performance metrics juxtaposed against the baseline results.

1.4 Outline

The remainder of this thesis is structured as follows.

In chapter 2, we dive deeper into the emergence of deep (convolutional) neural networks, detailing various breakthroughs and how they pertain to the task of semantic segmentation. Remaining issues, particular to the use of deep architectures, are discussed.

We zoom in on the parameter-heavy convolutional layer in chapter 3, and discuss how they can be replaced with low-rank alternative layers. To that end, we briefly go over basic theoretical concepts pertaining to tensor networks, introduce tensor diagram notation, and review studies that used similar approaches. In the final sections of this chapter, we outline the approach for designing and implementing low-rank layers, as they were used in the context of this study.

Chapter 4 details our methodology. In it, the BraTS dataset is introduced, both in terms of the medical pathology it tackles, and the specifics of the medical images it contains. The remaining sections will focus on the training procedure—all aspects of which were kept equal between all models for the sake of a fair comparison. The chapter also gives an overview of the different TNNs, and the parameter choices that define them.

The resulting performance of all these networks, including how they stack up against the baseline, is discussed in chapter 5. This chapter zooms in on different performance metrics, as well as the core characteristics of each network—i.e., their memory demand, and the computational strain they impose.

Finally, in chapter 6, these results are framed in a broader context. Limitations of this study are discussed, along with potential implications for future research efforts.

We need to go deeper.

Leonardo DiCaprio expresses his belief that adding layers is a solution to his problems.
Inception (2010)

Chapter 2

Semantic Segmentation

When humans observe their physical environment, they do not merely experience a set of photons striking the retina. We continuously and automatically process our visual input to make sense of what we see, by grouping similar elements, recognizing patterns and simplifying complex images—an observation which famously led to *Gestalt theory* [83]. Computers, however, do not have the luxury of being the product of millions of years of evolution, and need some assistance in this regard[1].

Image segmentation is "*the process of partitioning an image into component regions or objects* [47, p. 4]". In other words, it entails a series of decisions about how atomic portions of an image group together, what exactly they represent within the image, or which part of the scene they belong to. When we apply techniques that look for specific object instances, this is called *instance segmentation* (e.g., [87]). *Semantic segmentation*, in turn, refers to the goal of assigning a class label to each pixel (in 2D) or voxel (in 3D) in an image, regardless of any object they may belong to [23, 74].

In the sections below, we describe how the domain of semantic segmentation evolved (or rather, dramatically pivoted) from a traditional, handcrafted approach to fully embracing deep learning as its lord and savior. We conclude with a description of U-Net, a major milestone in the field, and a parent to many of the present-day state-of-the-art algorithms.

2.1 Early segmentation approaches

Before neural networks and deep learning entered the mainstream, image segmentation was accomplished through a plethora of computer vision techniques, many of which are still in use today. There are several ways of grouping these algorithms [89], one of which leads to the following classification scheme [47]:

1. Region-based methods
2. Edge-based methods
3. Threshold-based methods
4. Feature-based clustering methods

[1] Or rather, a lot of assistance, as evidenced by the slow crawl toward fully autonomous driving.

Region-based methods A naive way of approaching image segmentation, is the idea that *similar* neighbouring pixels are part of the same image segment, whereas different neighbours are not. *Statistical region merging* [65], for instance, describes a process in which pixels are merged into atomic regions, which are then iteratively merged into larger regions until a certain criterion is reached. An application of this principle can be found in Maes et al. [55], who combined the automatic generation of primitive image regions (using the watershed of the gradient magnitude image) with a manual interactive phase. More specifically, users could 'paint' the region they wished to extract with a smart paintbrush, which took the earlier segmentation into account.

Edge-based methods Edges, in image segmentation, are lines that trace a discontinuity in the scene, i.e., a sudden change in a visual property such as hue or brightness. This allows us to divide images into segments, by considering the edges that separate them. Several edge detection operators have been proposed (fig. 2.1), many of which are still used and improved upon to this day (e.g., Sobel edge detection [84]). The arguably most popular edge detection algorithm is that of John Canny, who introduced his now-dubbed *Canny edge detector* in 1983 [8].

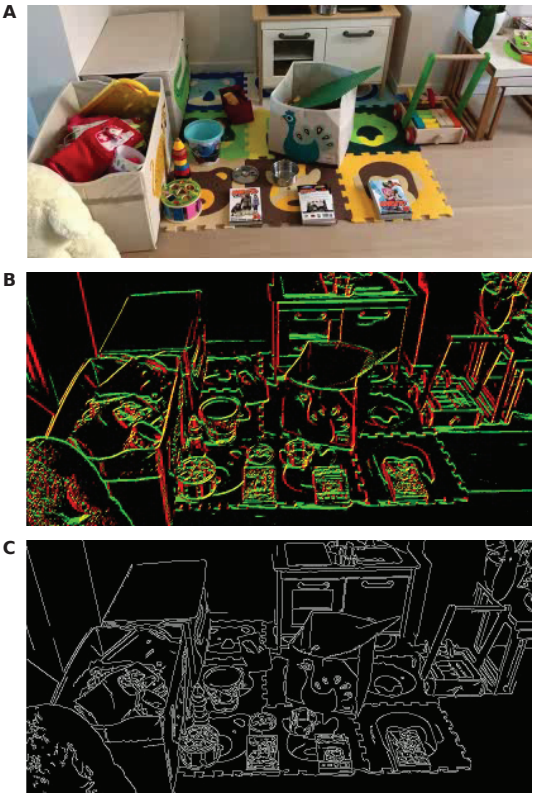


Figure 2.1: **Edge detection.** Segmenting my daughters' play corner. **A.** Original image. **B.** Sobel edge detection (vertical, diagonal, horizontal edges). **C.** Canny edge detection.

Threshold-based methods One of the simplest approaches to image segmentation is *thresholding*. Here, an image is converted to a binary format (e.g., based on pixel intensity), and a threshold value is selected: pixels with higher values form one (class of) region(s), pixels with lower values another. This threshold can be chosen globally (i.e., based on the full image), or locally (i.e., based on a part of the image) [61].

Feature-based clustering methods Clustering refers to the strategy of grouping pixels in an image based on the similarity of specific features. A typical example is *K-means clustering*, an algorithm that starts of with a set of cluster centers, and then iteratively computes distances between pixels and each of these centers, assigns each pixel to the nearest center, and recalculates the cluster centers based on the pixels they are linked

with. The distance metric can be composed of a variety of factors, such as color, brightness, location, or a combination of the above. There are many other clustering methods, including *Fuzzy clustering*, *hierarchical clustering*, and *partition-based clustering* [24]. In addition, classification algorithms can be counted among this category, such as the use of support vector machines [88] or random forest classifiers [27].

While this taxonomy covers a large portion of the existing computer vision approaches for image segmentation, it is certainly not exhaustive. There are other techniques, such as graph-based algorithms, that are hard to fit into any of the former categories. In addition, the boundaries of this taxonomy are not absolute: several approaches can be combined to yield a better overall performance (e.g., applying classification to primitive regions). An in-depth exploration of these methods is, however, not within the scope of this thesis.

2.2 Deep learning

The advent of deep learning—the use of neural networks with multiple hidden layers—has revolutionized the domain of machine learning, and that of computer vision in particular. At the same time, its inception dates back to 1943. In that year, Warren McCulloch and Walter Pitts [57] put forward a logical-mathematical framework designed to describe neural activity. Drawing inspiration from the workings of a human neuron, they designed small computational units that operated in parallel, and were capable of executing basic logic operations (AND, OR, and NOT).

Though Mcculloch and Pitts [57] opened the door for a refreshingly novel way of modelling neural activity, their ideas did not account for something that is central to the way our brain works: the ability to learn. At the time, there were several ideas on how to bridge that gap (see, for instance, the notion of *Hebbian learning* [29]). Arguably, one of the greatest milestones in that regard, is the conception of *backpropagation*—an algorithm to update network weights that is central to the majority of present-day (feedforward) neural networks. Its core principle was coined in the early sixties, with Henry Kelley presenting the mathematical blueprint [41], and Stuart Dreyfus refining it using the *chain rule* [18]. However, given the available computational hardware at that time, backpropagation was a labor-intensive affair, and therefore not very useful. Despite other attempts at the development of *deep learning* (see, for instance, [36, 37]), slowly but surely, research in the domain of artificial intelligence stagnated—the first *AI winter* set in (1974-1980), and the backpropagation algorithm was put on ice.

Convolutional neural networks Our sensory neurons do not fire at the behest of just any stimulus. The region in our sensory periphery to which they *do* respond, is called the *receptive field*. This concept inspired Fukushima [22] to create the *neocognitron* (1979), a "*self-organizing neural network model*", which included *convolutional layers* (and pooling layers) for the first time. In this architecture, neurons (or "cells" as they are called in the publication) only take input from their "receptive field"—a subregion of the output of the preceding layer. Despite this innovation it took a few more years for convolutional neural networks to be irreversibly brought onto stage. Backpropagation had been revisited in the years before by Rumelhart et al. [72], but it was LeCun et al. [49] who convincingly

demonstrated its potency in 1989 by designing a self-learning image classification network for zip codes. Their architecture[2] was heavily indebted to Fukushima's neocognitron, yet effectively expanded the concept of convolutional layers into how they are understood and used to this day.

Alas, winter came again. While promising, breakthroughs like LeCun's convolutional network did not scale well due to their computational hunger. Expectations for deep learning, at the time, were too high to be met, and funding for AI slowly dwindled. In the early nineties, little progress was made in the domain of neural networks. Other machine learning techniques, such as support vector machines [7], temporarily took the forefront.

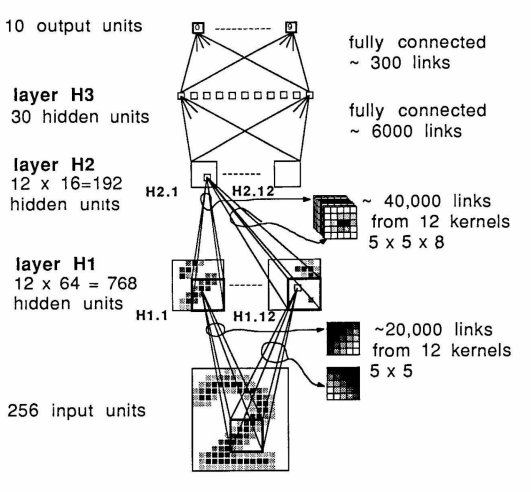


Figure 2.2: **Initial convolutional neural network**, famously published by LeCun et al. [49].

Long-awaited catalysts Then, as these things happen, video games changed everything[3]. In October 1999, the world's first *graphics processing unit* (GPU)—the GeForce 256—rolled off the production lines. Originally designed to accelerate 3D graphics for video games, which typically involves drawing numerous polygons in an ever-updating scene, GPUs were an exceptional fit for deep learning. A central processing unit (CPU) typically has a limited number of cores, i.e., units that carry out arithmetic operations. GPUs, on the other hand, have up to several thousands[4], which enables them to execute many calculations in parallel. In addition, GPUs also have a high memory bandwidth, allowing them to transfer large chunks of data in memory. These qualities make GPUs uniquely suited to the purpose of training neural networks, as their core functionality boils down to the calculation of large numbers of parallellizable matrix operations. Neural networks could compete with the then state-of-the-art machine learning approaches again.

A unique advantage of neural networks is their data-driven nature: as more training data becomes available, performance increases. By the grace of the Internet, such data became increasingly available. In 2009, Fei-Fei Lin ushered in the era of Big Data by launching ImageNet, an open database consisting of millions of labeled images. Between 2010 and 2017, the ImageNet project hosted a yearly challenge, inviting researchers to submit algorithms for object detection (i.e., instance segmentation) and image classification.

[2] A later iteration of this architecture now known as *LeNet*, after its principal researcher Yann LeCun [50].

[3] In a way, one could say that I've been pioneering the AI revolution all my life.

[4] The GeForce RTX 3090 has a whopping total of 10,496 1080 CUDA cores, along with 328 Tensor cores and 82 Ray Tracing cores.

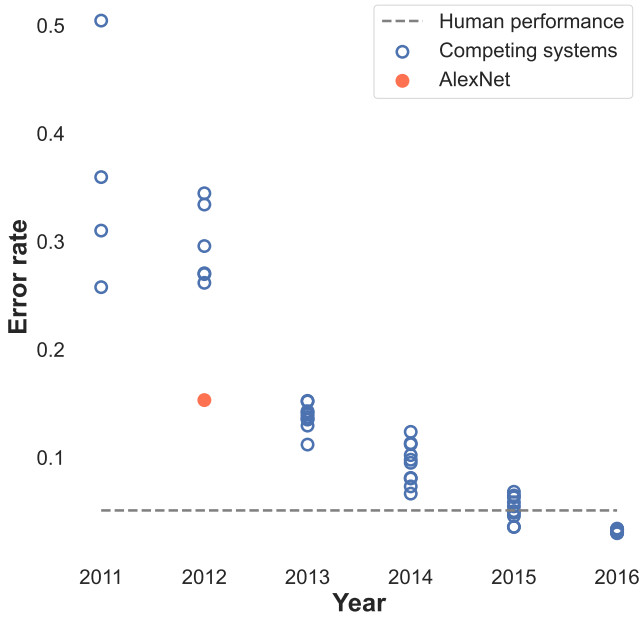


Figure 2.3: **ImageNet competition results.** Note the steep reduction in the error rate, given by AlexNet architecture in 2012 [46]. (Adapted from https://gist.github.com/germank/a542f22be0dad004b18775a7976d1a0b.)

Then came *AlexNet*, and everything changed again. The AlexNet architecture (fig. 2.4), designed by Krizhevsky et al. [46], won the 2012 ImageNet challenge by a significant margin (fig. 2.3). It was fueled by the combination of the aforementioned set of catalysts, in that it demonstrated how GPUs could be leveraged to boost performance, and that it was trained on a huge image dataset. The use of GPU acceleration was crucial indeed, as AlexNet boasts a total of over 62M network parameters.

Apart from showcasing an exceptional performance, AlexNet also popularized several now commonplace architectural components, such as the *MaxPool* layer (downsampling by selecting the maximum value in a sliding window), the *DropOut* layer (an effort towards regularization by randomly 'dropping out' a subset of output values), and the *Leaky Rectified Linear Unit* (see also: section 4.2.1). Given the network's incredible leap from the state of the art, many researchers followed suit, and began designing their own flavors of 'AlexNet'.

2.3 Segmentation revisited

While the work of Krizhevsky et al. [46] is, with right and reason, labeled as a milestone in the development of deep learning, it was not conceived of in a vacuum. Several parallel research efforts were ongoing around the same time, equally spurred on by the new computational capabilities GPUs had to offer. One very important team of researchers, in this regard, was that of Jurgen Schmidhuber. In fact, the first computer algorithm capable

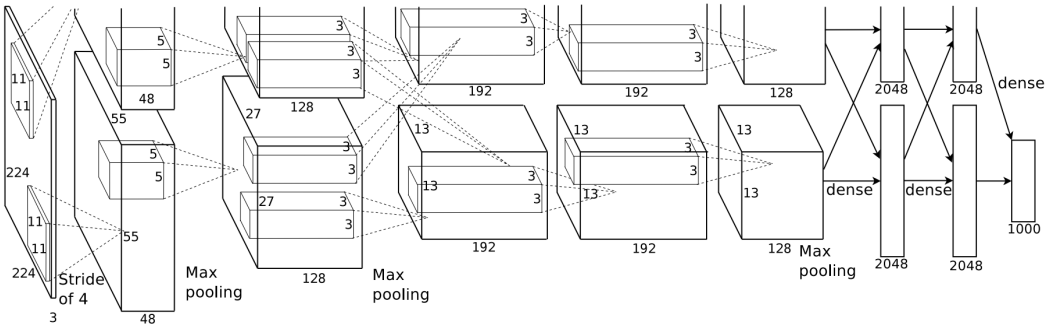


Figure 2.4: **AlexNet.** Krizhevsky et al. [46] won the 2012 ImageNet challenge, and revolutionized the field of deep learning with this seminal architecture.

of superhuman performance was *DanNet*, a deep convolutional architecture that went on to—in some adapted form—win four successive challenges[5] [13]. Importantly, DanNet won the ISBI 2012 Challenge [9]: a neuronal membrane segmentation task in electron microscopy images [14]. It was the first deep learning algorithm to win a competition in the image segmentation domain (as opposed to classification).

Cireşan's *DanNet* for image segmentation The ISBI'12 challenge was relatively straightforward: cellular images are given, and the algorithm must classify pixels as either part of the membranes ("the boundary between neurite cross sections") or the inside of the cells. Cireşan et al. [14] accomplished this through a sliding window setup, where pixel classification was accomplished by processing a patch around said pixel through a series of convolutional and max pooling layers. At the tail end of the network, a number of fully-connected dense layers were attached. The final of these dense layers contained two nodes (i.e., the number of classes), the output of which was passed to a *softmax* activation function. The softmax function is a normalized exponential function, used to to yield probability scores for each of the output classes[6].

This approach had some advantages. First, DanNet was capable of generating localized pixel-wise predictions by using patches around these positions as input, mirroring input values for any out-of-bounds part of the patch. Second, this use of patches was effectively a form of data augmentation, as there were more patches than training images. However, the algorithm is slow, as the network must run as many times as there are pixels (and therefore, patches) in the input image, in order to obtain a complete segmentation map. On top of that, a choice must be made in terms of patch size. A large window implies a maximal amount of contextual information is used in the prediction, though the spatial accuracy of the prediction is expected to be coarse. A small patch, on the other hand, improves localization accuracy at the cost of contextual richness. In other words, a trade-off between spatial and semantic accuracy presented itself.

[5] Though apparently, in the eyes of history, just not the *right* one.

[6] Note that this was, and is, a fairly common strategy for (pixelwise) classification purposes, such as the aforementioned AlexNet [46].

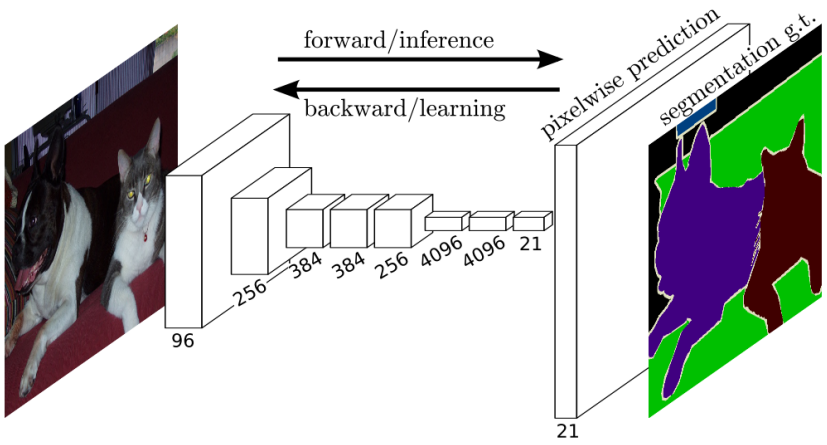


Figure 2.5: **Fully convolutional network.** This architecture, taken from [74], shows a traditional FCN: no dense layers are allowed, only convolutions, downsampling, and upsampling. Note how the input is contracted over the course of many convolutions, but very abruptly expanded at the tail end.

Ronneberger's *Original U-Net* In 2015, an elegant network model was proposed by Ronneberger et al. [70], in a bid to address these shortcomings. Its architecture drew inspiration from so-called *fully convolutional neural networks* (FCN) [74], which did not include any dense layers—only convolutions, upsampling (e.g., deconvolution), and subsampling (e.g., maxpool) (fig. 2.5). The local connectivity of such architectures not only reduced the overall number of parameters, they could also be used on variable-sized input (as opposed to dense architectures).

Traditionally, FCN architectures *contracted* the input image in a series of narrowing convolutions, which were then abruptly upsampled into a segmentation map. Ronneberger et al. [70] modified the upsampling stage, by modelling it as a gradual expansion. This expansive path contained a large number of feature channels, allowing the network to "*propagate context information to higher resolution layers*" [70, p. 3]. The architecture's symmetry—i.e., the mirroring of the encoding and decoding stages of the network—resembled a U-shape, hence it was christened *U-Net*. Importantly, skip connections were added between mirrored layers of the contracting and expansive path. More explicitly, high-res features from the contracting phase were concatenated with contextually-rich features from the expansive path, facilitating an optimal balance between spatial and semantic accuracy.

U-Net won two ISBI competitions in 2015: the Grand Challenge for Computer-Automated Detection of Caries in Bitewing Radiography[7], and the Cell Tracking Challenge[8]. U-Net's legacy as a ground-breaking segmentation architecture is further reflected in its descendants. The Brain Tumor Segmentation Challenge (BraTS, see section 4.1) was won by Kamnitsas et al. [40] in 2017, who used an ensemble approach that involved multiple

[7] http://www-o.ntust.edu.tw/~cweiwang/ISBI2015/challenge2/index.html
[8] https://lmb.informatik.uni-freiburg.de/people/ronneber/isbi2015/

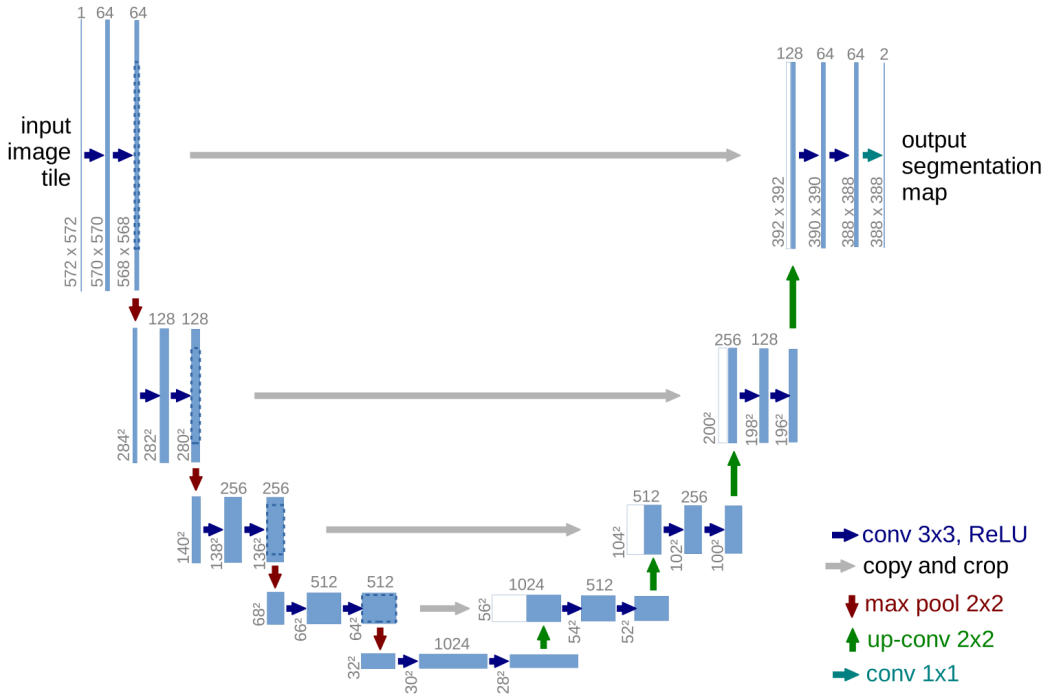


Figure 2.6: **U-Net [70].** The expansive path allows for contextual information to be propagated to higher levels of the architecture, setting it apart from earlier FCN. Note the skip connections, which allow the preservation of high resolution features from the expansive path.

3D U-Net [10] models. A year later, Isensee et al. [33] won second place with an only slightly altered U-Net architecture (*hence* nnU-Net, or 'No New-net'), reinforcing the reputation of U-Net's core architecture. In 2019, the challenge winner was a two-stage cascade of U-Net architectures [39]. The 2020 winner, finally, was a nnU-Net with additional BraTS-tailored post-processing, region-based training and data augmentation [35]. Ronneberger's dominion over segmentation networks appears, for all intents and purposes, unchallenged.

2.4 Deep-seated issues

At present, the deep learning revolution appears to be ongoing. In some cases, the adage appears to be 'deeper is better', as new architectures attempt to maximize performance by adding more layers, and ultimately, more training parameters. For instance, the 2015 ISBI classification challenge winner used a network that contained 152 layers [28]—AlexNet [46] had 8. A less dramatic example, where the original U-Net [70] architecture had 18 convolutional layers at its core, the nnU-Net [35] has 22. However, deep architectures are plagued by a number of problems, including those that are discussed below.

Vanishing gradient Neural networks are often trained using gradient-based training algorithms. Deep architectures, in particular, may suffer from the *vanishing gradient* problem [30]. As the learning error is further propagated to earlier layers, the weight update becomes increasingly smaller due to the partial differentiation of the error function. Ultimately, this renders the effect of the update process inert, and drastically slows down training convergence. There are several potential solutions to this issue. One of them is straightforward: rather than using sigmoid activation functions (which 'suffer' from derivation, see fig. 2.7), the use of the Rectified Linear Unit activation function provides a better, unbound alternative (see also section 4.2.1). Other helpful tools are the use of residual blocks, as residual connections do not suffer from the impact of the derivation, or normalization layers, which normalize the input, so the outer limits of the sigmoid activation (and minuscule derivative values) are not reached.

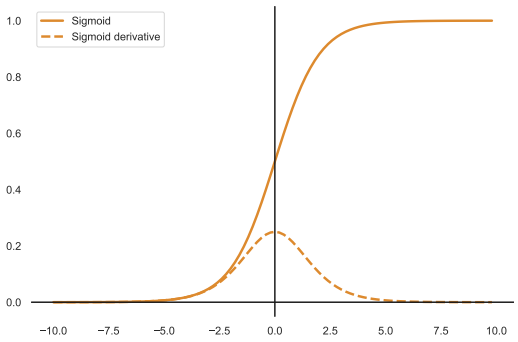


Figure 2.7: **Vanishing gradient.** Deriving the sigmoid function leads to small values, smothering the gradient.

Exploding gradient The inverse of diminishing gradients, is the phenomenon where backpropagation leads to a dramatic gradient increase. Whereas the former issue is caused by the use of certain activation functions, exploding gradients are caused when weights reach large values. In that case, the gradient with respect to the weight parameters may increase exponentially as it is propagated to the earlier layers [68]. Exploding gradients can be prevented in several ways, including through weight regularization. This strategy, which is also called *weight decay*, adds a term to the error function that penalizes large weights (as is the case in the *Adam* and *AdamW* optimizers, see section 4.3.2).

Memory and computational demand There is a simple downside to the trend of neural networks growing large and deeper. Passing input through such a network, and updating network weights based on the loss, evidently requires a great deal of computational and memory resources. This is especially true of CNNs, which—in spite of these encumbrances—represent the backbone of all modern computer vision architectures. The emergence and continued development of GPUs and TPUs notwithstanding, this strain remains a bottleneck, particularly when near real-time inference is required. For one thing, these accelerators may not always be available—a topical obstacle given the recent push towards so-called *edge AI*[9]. Examples of computationally limited platforms include autonomous vehicles, the majority of mobile or Internet-of-Things devices, or head-mounted displays for augmented reality. In the case of (medical) image segmen-

[9] Edge AI refers to any system that uses machine learning algorithms to process data locally (e.g., on a smartwatch).

tation, we may also consider an intra-operative imaging context, where a segmentation prediction is naturally plagued by time constraint.

Overfitting In addition to imposing a severe strain on the abilities of its host platform, overparametrized networks tend to *overfit*, particularly when the number of training instances is limited [16, 2]. There are several options to avoid overfitted models, including regularization strategies such as weight decay or dropout layers, or data augmentation to artificially increase the number of training instances. However, these techniques do not address the problems of memory and computational strain, as they do little to lower the number of parameters or connections.

In sum, the problem statement is as follows. Clearly, we must reduce the computational load of our algorithms. At the same time, the overparametrization of our state-of-the-art deep learning models is precisely what has led to their success. How, then, can we reduce the number of parameters to lessen the strain on memory resources, lower the number of operations required by our model, yet safeguard the quality of the predictions? In the next chapter, a promising remedial strategy is discussed, as we gradually introduce the concept of *tensorized neural networks*.

You're tearing me apart, Lisa!

Tommy Wiseau has yet to gain
appreciation for the benefits of
decomposed formats.
The Room (2003)

Chapter 3

Lightweight Convolution

In this chapter, we delve deeper into the inner workings of convolutional layers. First, some basic concepts from tensor calculus are explained, insofar as they are relevant to understand the subsequent sections. Convolutional layers are then dissected through this very lens. Next, we take a closer look at *tensor networks*, and particularly at their use to come up with low-rank representations of higher-order tensors. These low-rank representations can be, and have been, used to 'decompose'[1] the weight tensor that defines a convolutional layer, leading to more compact and speedier networks. We go over related work, discussing studies that have looked into similar approaches before. In the final section of the chapter, we zoom in on our own approach towards designing low-rank convolutional layers, and ultimately, tensorized neural networks.

3.1 Tensors and tensor networks

A **tensor** of the n^{th} order is an n-dimensional array. In that sense, it is generalization of a scalar (0^{th} order), a vector (1^{st} order), and a matrix (2^{nd} order) (see fig. 3.1). A tensor's dimensions are referred to as its **modes**, each of which has a certain size. Take, for instance, the following tensor:

$$\mathcal{X} \in \mathbb{R}^{N_1 \times N_2 \times ... \times N_k} \tag{3.1}$$

This tensor $\mathcal{X}$ is of the k^{th} order, with its k modes having sizes N_i ($i \in [1, 2, ..., k]$). An individual position in this tensor is denoted by lowercase letters:

$$\mathcal{X}(i_1, i_2, ...i_k) = x_{i_1, i_2, ...i_k} \tag{3.2}$$

The **outer product** in tensor calculus is, again, a generalization of the outer product for vectors or matrices. For two tensors, $\mathcal{X} \in \mathbb{R}^{N_1 \times ... \times N_k}$ and $\mathcal{Y} \in \mathbb{R}^{M_1 \times ... \times M_l}$, it is defined as:

$$\mathcal{Z} = \mathcal{X} \circ \mathcal{Y} \in \mathbb{R}^{N_1 \times ... \times N_k \times M_1 \times ... \times M_l} \tag{3.3}$$

[1] I use this term loosely here, since decomposing a tensor of learnt weights is different from learning weights in a reparametrized low-rank tensor format.

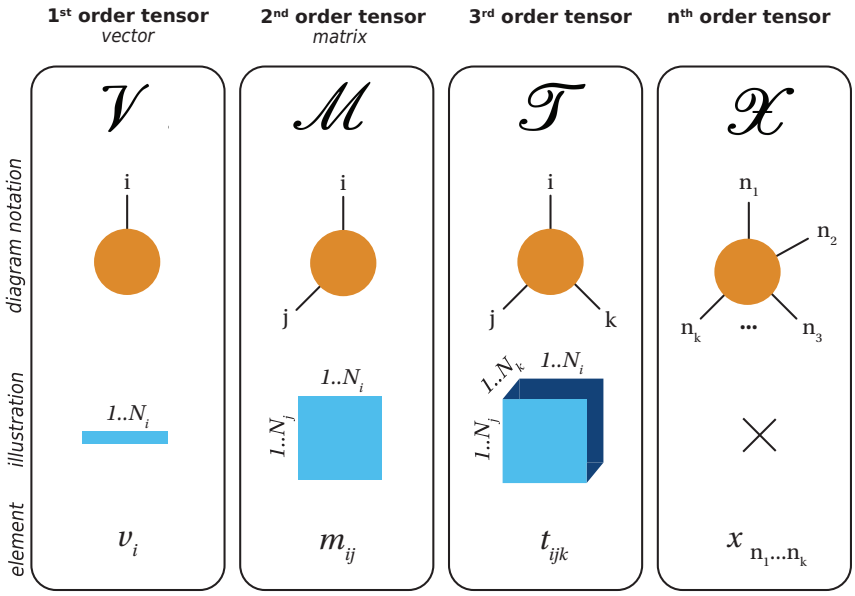


Figure 3.1: **Tensor (diagram) notation.** Vectors, matrices and 3rd-order tensors can be thought of as lines, quadrilaterals and beams, respectively. From 4th-order tensors onward, an intuitive illustration is no longer possible.

with

$$z_{n_1,\ldots,n_k,m_1,\ldots,m_l} = x_{n_1,\ldots,n_k} y_{m_1,\ldots,m_l} \tag{3.4}$$

The outer product of N 1st-order tensors (i.e., vectors) is a Nth-order **rank-1 tensor**—despite its higher dimensionality (N modes), it can be described efficiently using N vectors only. The concept of rank is central to the present work, and to the lightweight representations described in the subsequent sections. To define it, we first introduce the canonical polyadic decomposition[2].

A **canonical polyadic decomposition (CPD)** of a tensor $\mathscr{X}$ is its approximation by a finite sum of rank-1 tensors:

$$\mathscr{X} = \sum_{r=1}^{R} u_r^{(1)} \circ u_r^{(2)} \circ \ldots \circ u_r^{(k)} \qquad \text{with} \qquad u_r^{(i)} \in \mathbb{R}^{N_i} \tag{3.5}$$

In linear algebra, the rank of a matrix is the number of linearly independent row or column vectors. By extension, the **tensor rank** or CP rank, is the minimal number R of rank-1 terms required to obtain an *exact* CPD. As a consequence, the lower the rank of a tensor, the more compactly it can be described. Importantly, in practice, a finite algorithm to obtain such a decomposition does not exist for a 3rd- or higher-order tensor [45]. Hence, most algorithms decompose the tensor with varying (decreasing) values for R, until the approximation error meets a certain criterion.

[2] Also known as tensor rank decomposition, the polyadic form, the Kruskall tensor, PARAFAC, or CANDECOMP.

Tensor contraction Multilinear algebra extends linear algebra to include multidimensional arrays—tensors. Within the scope of this thesis, the most important of these operations is the **tensor contraction**. It can be considered as a "*higher-dimensional analogue of matrix multiplication, inner product and outer product*" [11, p. 12].

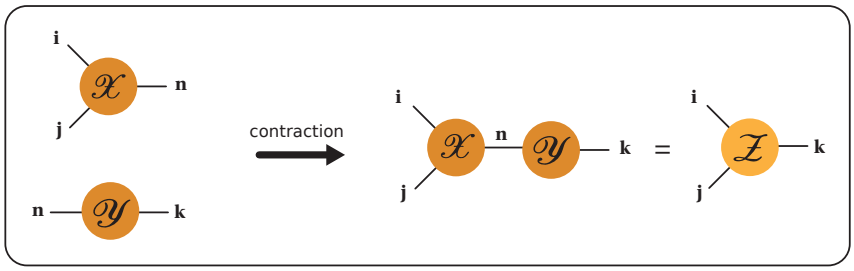


Figure 3.2: **Tensor contraction.** Tensor diagram notation of the contraction expressed in eq. (3.6).

The contraction of two tensors $\mathcal{X} \in \mathbb{R}^{N_1 \times \ldots \times N_k}$ and $\mathcal{Y} \in \mathbb{R}^{M_1 \times \ldots \times M_l}$ over common modes $N_i = M_j$ is defined as:

$$\mathcal{Z} = \mathcal{X} \times_{M_j}^{N_i} \mathcal{Y} \tag{3.6}$$

with

$$z_{n_1,\ldots,n_{i-1},n_{i+1},\ldots,n_k,m_1,\ldots,m_{j-1},m_{j+1},\ldots,m_l} = \sum_{n_i=m_j=1}^{N_i=M_j} x_{n_1,\ldots,n_i,\ldots,n_k} y_{m_1,\ldots,m_j,\ldots,m_l} \tag{3.7}$$

Contractions are often expressed more concisely, using **Einstein summation notation**. This notation is based on three principles:

1. Repeated indices are implicitly summed over.
2. Each index cannot appear more than twice in any term.
3. Each term must contain identical non-repeated indices.

For instance,

$$z_{ijk} = \sum_{n=1}^{N} x_{ijn} y_{nk} \tag{3.8}$$

becomes

$$z_{ijk} = x_{ijn} y_{nk} \tag{3.9}$$

Tensor diagram notation Inspired by the former notation, tensor diagrams were created as a convenient way to visually represent tensors, and contractions between them. In tensor diagram notation, there are two core principles:

1. A tensor is depicted by a solid shape, and its modes by dangling edges (fig. 3.1 and fig. 3.2).
2. A contraction between a common mode n of tensors **X** and **Y** is indicated by connecting these respective edges (fig. 3.2).

Using tensor diagram notation, we can easily visualize tensor networks. A **tensor network** is a specific type of graph structure, which can be used to represent a higher-order tensor as a set of sparsely interconnected, lower-order tensors. These component tensors are called **nodes** or **cores**, or **factor matrices** if their order is 2. The CPD, described in 3.5, can therefore be represented using a tensor network (see fig. 3.3).

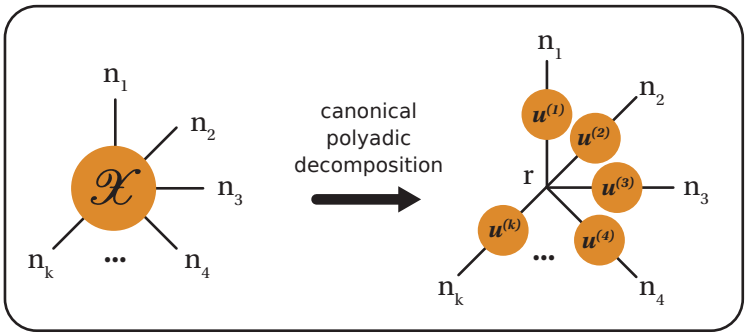


Figure 3.3: **Canonical polyadic decomposition.** Tensor diagram representation of a CPD, turning tensor X into a tensor network of sparsely connected $u^{(i)}$ nodes, which are (factor) matrices.

Contraction order matters (not) A CP tensor network is straightforward, in the sense that all nodes are simultaneously contracted over the r-mode. However, more complex tensor networks can easily be imagined, where several contractions may be executed in different orders (see, for instance, the Tucker and tensor train formats, discussed in section 3.4). As a trivial example, take the tensor network shown in fig. 3.4. This network can be contracted in two ways: by first contracting $\mathscr{X}$ and $\mathscr{Y}$ (over mode j), and then contracting the result with $\mathscr{Z}$ (over mode k), or in the reversed order. It can easily be shown that either order—also called a **contraction path**—will result in the same tensor. The number of calculations required, however, may differ greatly depending on the path taken.

In our example, assuming all modes have a size of 10, contracting $\mathscr{X}$ and $\mathscr{Y}$ requires 2k floating point operations (FLOPS, see also section 4.4.2). Contracting this result with $\mathscr{Z}$, then, adds another 200k, yielding a total of 202k FLOPS. In contrast, contracting $\mathscr{Z}$ and $\mathscr{Y}$ first, and $\mathscr{X}$ last, leads to a cumulative total of 400k FLOPS—almost double. This impact on complexity may become more pronounced as tensor networks become larger. As such, selecting the optimal contraction path is vital to benefit from the computational advantages tensor networks bring to bear. We will leverage this insight for our own approach, at the end of the chapter.

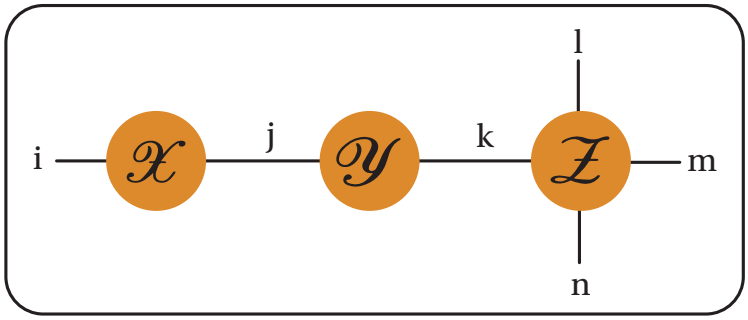


Figure 3.4: **Contraction path.** The cost of contracting a tensor network can vary greatly, depending on the path chosen.

3.2 Convolutional layer revisited

In section 2.2, the convolutional layer was touched upon: a neural network component that, using a sliding window approach, convolves a set of **kernels** with a given input array, such as an image. The values in these kernels—the *weights*—are learnt by the model using a backpropagation scheme. The combination of these kernels across all input channels is typically referred to as a **filter**. Convolving the filter with the input yields an **activation map**, representing the filter's response to each spatial position in the input array. A kernel can have different (odd-numbered) sizes. Often, kernels are 3×3 matrices (for 2D input) or $3 \times 3 \times 3$ tensors (for 3D input). However, other sizes can be used, such as $1 \times 1 (\times 1)$ for downscaling (see, for instance, Google's Inception architecture [79]).

The manner in which a kernel is 'slid' across the input, can be tuned using to the following parameters:

- **Padding** refers to the practice of adding values along the borders of the input, so the movement of the kernel (e.g., 5×5) across the input is not constrained by the original resolution, which would effectively enforce a downscaling. Several strategies can be used to select the padding values, such as reflecting (mirroring) the border values, or simply inserting zeroes (*zero-padding*).
- **Stride** denotes the 'jump' a kernel takes after each convolution. If stride is set to 1, the kernel moves a single position (along a certain axis) every time. If stride is set to 2, the kernel will skip a position, and so on.
- **Dilation** is the manner in which the kernel is 'inflated' by adding empty space in between the kernel weights. For instance, a dilation of 2 would yield a kernel that only considers input values every other position, whereas a dilation of 1 would cause whole, uninterrupted patches to be processed.

Consider a setting with 3-dimensional input, such as the processing of volumetric medical imaging data. The output of a convolutional layer is calculated as follows. Given an input tensor $\mathcal{X}$

$$\mathcal{X} \in \mathbb{R}^{C_{in} \times H \times W \times D} \tag{3.10}$$

with C_{in} the number of input channels, and (H, W, D) the dimensions of the input image; and given a kernel (weight) tensor $\mathcal{W}$

$$\mathcal{W} \in \mathbb{R}^{C_{in} \times C_{out} \times k_H \times k_W \times k_D} \tag{3.11}$$

with (k_H, k_W, k_D) representing the kernel size, and C_{out} the number of output channels. Assuming zero-padding, and dilation and stride set to 1, the output tensor $\mathcal{Y}$

$$\mathcal{Y} \in \mathbb{R}^{C_{out} \times H \times W \times D} \tag{3.12}$$

is defined as follows [2]:

$$y_{chwd} = \sum_{c'=1}^{C_{in}} \sum_{h'=1}^{k_H} \sum_{w'=1}^{k_W} \sum_{d'=1}^{k_D} x_{c'(h+h'-1)(w+w'-1)(d+d'-1)} v_{cc'h'w'd'} \tag{3.13}$$

with x, v and y denoting entries in the input, weight and output tensor, respectively.

Convolution in tensor diagram notation The aforementioned parametrization, where the indices corresponding to the input channels and kernel dimensions are summed over, invites another representation: convolution as a tensor diagram. To generate this diagram, we must first consider the notion of **unfolding** a tensor. In the present context, unfolding refers to the conversion of an input tensor to a format that contains the individual sliding window blocks (fig. 3.5). In other words, it is the creation of a new tensor that associates each entry in the original input tensor with its encompassing window, which will be convolved with each kernel. Note that this operation depends on the aforementioned spatial parameters (padding, stride and dilation), and may impact the resolution of the input.

The unfolded input tensor $\mathcal{X}'$ now shares modes (dangling edges) with the weight tensor $\mathcal{W}$—those belonging to the input channels and kernel dimensions—allowing both tensors to be contracted. Doing so yields the output tensor $\mathcal{Y}$. This conceptualization will be intuitively helpful in section 3.4, where alternative, decomposed formats for the weight tensor will be proposed.

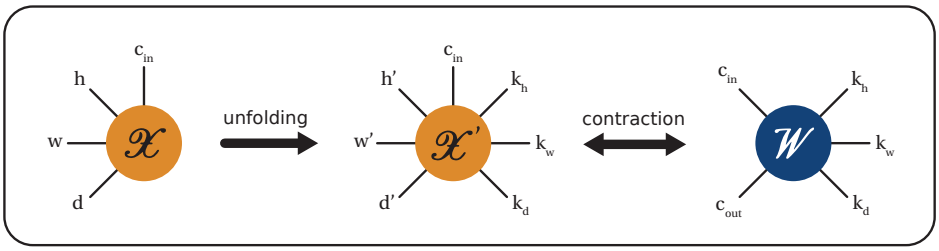


Figure 3.5: **Unfolding an input tensor.** To represent the functionality of a convolutional layer, we associate each position in the input tensor with a window, depending on the spatial characteristics of the kernel. This process can then be thought of as a contraction along the input channel and kernel dimension modes.

Low-rank representations In chapter 2, we discussed how convolutional layers are at the core of CNN, such as U-Net [70]. These layers are characterized by 4^{th}- or 5^{th}-order weight tensors, for 2D and 3D convolutional layers, respectively. Given this fact, and due to the increasing depth of contemporary CNNs, the total number of weights in such architectures is typically enormous. In section 2.4, we discussed how this overparametrization can lead to a number of issues, most notably a strain on computational resources and resources, as well as the tendency for the model to overfit.

Here we introduce an alternative approach that could offer a solution to all of the above problems: **low-rank convolutional layers**. As discussed in section 3.1, the rank of a tensor refers to the minimal number R of rank-1 terms in an exact CPD. For instance, if a 6^{th}-order tensor can be expressed using only (but no less than) two rank-1 terms, its rank is 2. This implies that if R is sufficiently small, we can express the same information using significantly fewer values, easing the strain on memory. Moreover, there are several established ways in which a higher-order tensor can be *decomposed* using tensor networks, of which the CP format is but one example. Depending on the optimal path along which the full tensor network—that is, the decomposed or reparametrized weight tensor connected to the input tensor—can be contracted, the convolution can be computationally optimized.

In the next section, we will look at earlier efforts to replace network layers with condensed components, such as low-rank tensor network representations. Finally, we discuss our own approach toward enforcing low-rank constraints onto convolutional weight tensors, as well as the characteristics of the tensor networks we employ to that end.

3.3 Related work

The overparametrization of deep learning architectures has been addressed from several different angles. One set of approaches can broadly be summarized under the category **pruning**—the removal of weights from a pretrained network according to a certain criterion or algorithm(e.g., [26, 21]). While certain pruning techniques have been shown to yield a parameter count decrease while maintaining performance, they require a network to be (partially) pretrained in order to identify which weights and connections are 'important', or which neurons should cluster together. The boons of this approach are thus limited to the inference phase, as they are of little benefit to the training process.

The importance of "*sparsely connected architectures, even inside the convolutions*" was also brought up by Szegedy et al. in their seminal paper [79, p. 3]. Here, the Google researchers drew inspiration from Arora et al. [1] who suggested that (on the assumption that the probability distribution of a dataset could be represented by a large, sparse deep network) optimal network architectures could be hand-crafted layer-by-layer by clustering highly correlated neuron activations. Such a strategy would also mimic biological structures more closely, in the Hebbian sense: *neurons that fire together, wire together*. Computing infrastructures tend to handle non-uniform sparse data structures poorly, however, as opposed to operations such as dense matrix multiplication. These technical hindrances led them to create the Inception module which aims to approximate an **optimal local sparse structure** in a CNN, instead. In simplified terms, Inception modules

use convolutions with differently sized kernels on the same level, followed by pointwise (i.e., 1×1) convolutions, and a concatenation step at the module's end. In combining these modules, a compact network is learnt, as opposed to compressing a large pretrained network. In a similar vein, Howard et al. [31] proposed a class of models they called 'MobileNets' after their intended use for mobile and embedded platforms. Akin to the make-up of an Inception module, MobileNets use what they dub "depthwise separable filters"—combinations of a depthwise convolution, which separately convolves a kernel slice with each layer of depth in the input, and a subsequent pointwise convolution to compress the depth.

We can think of the former models as a form of factorization: convolutions are pulled apart into smaller components. However, convolutional layers—or more accurately, the weight tensors that define them—can be compressed in a different way: by **imposing low-rank constraints**. Similar to the difference between pruning and the latter approaches, this too can be achieved in two ways: 1) by pretraining a network in a conventional way first, and decomposing the resulting weight tensor after, or 2) by training a low-rank constrained parametrization of the original weight tensor.

Low-rank approximation of pretrained networks

In order to compress CNNs, Denton et al. [17] used several approximation techniques, including the application of Singular Value Decomposition (SVD)[3] on the weight tensor. To make this possible—SVD is a matrix operation—all weight tensor dimensions save for the first two were folded together, leaving a 2^{nd}-order tensor. Due to the lossy nature of approximation techniques, which may in turn hurt the model's performance, an additional fine-tuning[4] step was applied to the network. The authors report a speedup with factor 2, a reduced memory footprint (convolutional layers: factor $2-3\times$, dense layers: factor $5-13\times$) and a minimal loss in accuracy (1%). In their discussion, the hint towards the potential of low-rank approximations for model regularization: as the amount of available parameters decreases, the model is forced to learn a more generalized representation.

Jaderberg et al. [38] used a comparable approach, a proposed layer-by-layer data reconstruction technique to approximate the weight tensors in subsequent layers, obtaining similar results. Here, after each low-rank approximation of a weight tensor, the layers above were fine-tuned to minimize the construction error.

The algorithm described by Jaderberg et al. [38] was tested by Lebedev et al. [48], who reported an inability to match the former results. Lebedev et al. [48] then used a different approach, exacting CPD onto the weight tensor using a non-linear least squares algorithm. With additional fine-tuning, the authors were able to obtain an average model speedup 8.5 CPU speedup with minimal loss of accuracy (1%).

Kim et al. [43], finally, used a similar strategy to generate CNNs capable of being run efficiently in a low-powered mobile setting. Weight tensors were decomposed using Tucker decomposition, a generalization of CPD. The rank of the decomposition was

[3] As an interesting segue into the coming sections, note that SVD can be thought of as the 2-dimensional precursor of CPD.

[4] fine-tuning refers to the continued training of a pretrained network, often—though not in this case—on a new dataset.

determined by a separate algorithm—that is, as the global analytic solution of variational Bayesian matrix factorization [62]. To make up for any loss in performance, the resulting model was fine-tuned. Given their problem statement, the authors focus on energy consumption, reporting significant improvements in runtime on the smartphone.

Low-rank constrained model training

Soon after, Novikov et al. [66] published a study in which dense layers were compressed using the Tensor Train (TT) format, reporting an ability to scale down dense weight matrices in a Very Deep VGG network [75] by a factor up to 20,000, reducing the full network size by a factor 7. The authors argue that the TT format is particularly interesting given its immunity to the curse of dimensionality (see also [4])—as opposed to the CPD and Tucker [81] formats—as well as because of its robust decomposition algorithms. Though these results are not immediately applicable to fully convolutional networks, which do not contain dense layers, they serve to highlight the potential of the TT format for low-rank approximation. More importantly, Novikov et al. [66] do not decompose a pretrained network, but rather insert layers of which the weights are stored in TT format (i.e., TT layers).

In 2016, Tai et al. [80] proposed a way to impose such low-rank constraints onto CNNs. They reparametrized the convolutional kernels, based on the low-rank decomposition used in the study of Jaderberg et al. [38] and colleagues, causing low-rank constraints to be naturally imposed during training. Whereas these reparametrized networks are also deeper, and therefore harder to train due to vanishing and/or exploding gradients, they were able to train these networks efficiently from scratch using *batch normalization* [32]. Interestingly, when juxtaposing this approach with the low-rank approximation of pretrained networks, the authors found that the former approach in some cases led to superior performance. They attribute this result to two possible causes: 1) a better weight initialization and discovery of a better local minimum, and/or 2) a regularizing effect of the low-rank former, in that a more compact representation leads to more generalizable results.

Recently, Ashtari et al. [2] adopted a similar approach, reparametrizing convolutional layers in a CP tensor network format. In their network—a U-Net style architecture—the rank R of each convolutional layer was tuned in proportion to the number of input channels. Their results reinforced the notion that low-rank constrained networks can, despite their lesser parameter count, even *outperform* an unconstrained counterpart. This was most apparent in the Hausdorff distances, which were in some cases substantially lower for the low-rank networks—a testament to the regularizing effect of low-rank representations.

3.4 Building lightweight layers

In the present work, we further explore the approach set out by Ashtari et al. [2]. We expand the potential parametrizations beyond the CP format, including the Tucker format and the TT format. To facilitate our discussion, we represent each network as an algebraic expression, as well as using tensor diagram notation. In addition, we discuss how each

format can be tuned to obtain a certain compression rate, compared to a traditional weight tensor. In the remaining chapters, we will investigate how different compression rates impact performance, and whether this effect differs among our chosen low-rank formats.

Canonical Polyadic format

First, we consider the CP format (fig. 3.6), which was also used by Ashtari et al. [2]. Its algebraic form is given by:

$$\mathcal{W} = \sum_{r=1}^{R} u^{(1)}_{r,c_{in}} \circ u^{(2)}_{r,c_{out}} \circ u^{(3)}_{r,k_h} \circ u^{(4)}_{r,k_w} \circ u^{(5)}_{r,k_d} \tag{3.14}$$

In the CP format, each mode of the weight tensor is represented by separate factor matrix—the cores of the network. Each of these cores has two modes: one corresponding to the weight tensor's mode it represents (e.g., the number of input channels c_{in}), and a common rank mode r. To obtain the weight tensor, all cores in the CP tensor network are simultaneously contracted over the r-mode.

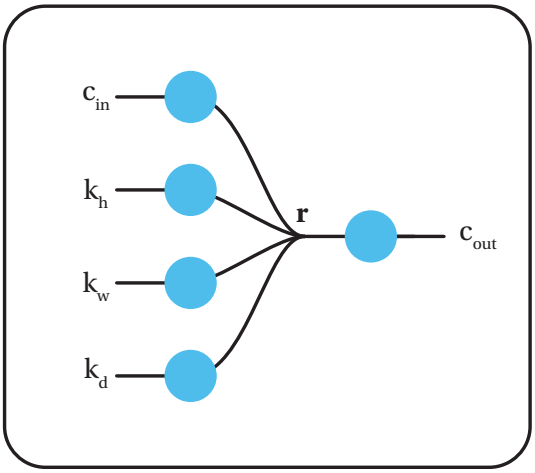


Figure 3.6: Convolutional weight tensor, represented in a **canonical polyadic format.**

The CP format can be tuned to obtain a specific compression rate. Let $N_{max} = (C_{in} + C_{out} + k_H + k_W + k_D)$ represent the total number of parameters in an unconstrained weight tensor, with C_{in} and C_{out} denoting the number of input and output channels, respectively, and (k_H, k_W, k_D) representing the kernel dimensions. To compress the representation with a factor k, the following equation can be trivially solved for r:

$$k = \frac{N_{max}}{N_{CP}} = \frac{N_{max}}{r(C_{in} + C_{out} + k_H + k_W + k_D)} \tag{3.15}$$

Tucker format

The Tucker format [81] is a generalization[5] of the CP format, in that it introduces a additional core G at the center of the tensor network, which connects to all others cores (fig. 3.7). Its general expression is:

[5] Replacing the center core of a Tucker format with a (hyper)diagonal tensor yields a CP format.

$$\mathcal{W} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} \sum_{r_4=1}^{R_4} \sum_{r_5=1}^{R_5} G_{r_1,r_2,r_3,r_4,r_5} \circ u^{(1)}_{r_1,c_{in}} \circ u^{(2)}_{r_2,c_{out}} \circ u^{(3)}_{r_3,k_h} \circ u^{(4)}_{r_4,k_w} \circ u^{(5)}_{r_5,k_d} \tag{3.16}$$

As a result, there are more bond dimensions—free dimensions in the network's cores—that can be tuned. In our Tucker decomposition, we set the bond dimensions corresponding to the 'kernel cores' to 3[6] [43]. We tune the remaining bond dimensions, from the central core to the cores corresponding to the input and output channel modes, proportionally to the number of input channels and output channels, respectively (i.e., as $\frac{C_{in}}{s}$ and $\frac{C_{out}}{s}$).

In the Tucker format, a compression rate k can be obtained by solving the following equation for s, and then tuning the network accordingly:

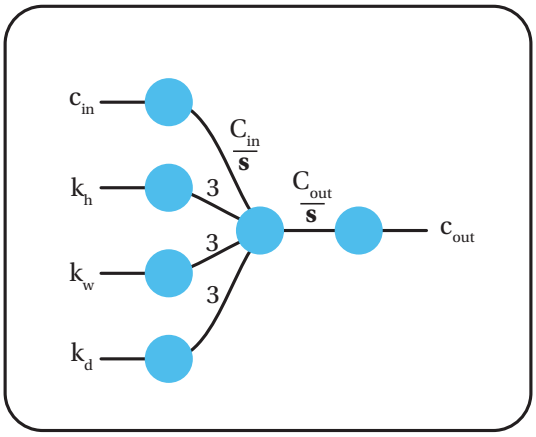


Figure 3.7: Convolutional weight tensor, represented in a **Tucker format.**

$$k = \frac{N_{max}}{N_{Tucker}} = \frac{N_{max}}{3(k_H + k_W + k_D) + \frac{C_{in}^2 + C_{out}^2}{s} + \frac{C_{in} C_{out} 3^3}{s^2}} \tag{3.17}$$

Note that this network, while seemingly more complex than the CP format, provides more options in terms of contraction paths. As such, there is more flexibility to find an optimal solution, reducing the required computation effort.

Tensor Train format

The final reparametrization we consider, is the Tensor Train format (fig. 3.8). It is expressed as follows:

$$\mathcal{W} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} \sum_{r_4=1}^{R_4} u^{(1)}_{c_{in},r_1} \circ u^{(2)}_{r_1,k_h,r_2} \circ u^{(3)}_{r_2,k_w,r_3} \circ u^{(4)}_{r_3,k_d,r_4} \circ u^{(5)}_{r_4,c_{out}} \tag{3.18}$$

Here, a tensor is decomposed by chaining together a series of cores, one for each mode of the original tensor. Each of these cores is a 3$^{\text{rd}}$-order tensor, except for the nodes at the head and tail of the 'train', which are 2$^{\text{nd}}$-order tensors. The bond dimensions between subsequent cores can be chosen, or tuned in accordance with the problem setting.

In our own TT reparametrization, we placed the cores corresponding to the channel modes (input and output) at the head and tail end. This was done to minimize the size of

[6] Another option was to simply not decompose the kernel modes, leaving only a central core and two channel-related cores—a so-called Tucker-2 decomposition [81].

each core. The channel modes typically have larger sizes than the kernel modes of the original weight tensor. Since the middle cores in the TT are 3rd-order tensors, it stands to reason to position the smaller kernel-related cores therein.

The remaining bond dimensions of the TT format were tuned in two ways, creating two versions of the TT reparametrization. The *first TT version* set all bond dimensions along the main TT 'axis' to a single variable, which could then be tuned for a desired compression rate by solving the following equation for tuning parameter r:

$$k = \frac{N_{max}}{N_{TT}} = \frac{N_{max}}{r(C_{in} + C_{out} + r(k_H + k_W + k_D))} \tag{3.19}$$

Given the higher-order nature of the middle cores, we further restricted their size by constraining the middle bond dimensions—i.e., those among the kernel cores—to 3 (*the second TT version)*. In that case, eq. (3.19) becomes:

$$k = \frac{N_{max}}{N_{TT2}} = \frac{N_{max}}{r(C_{in} + C_{out} + 3k_H + 3rk_D) + 9k_W} \tag{3.20}$$

Similar to the Tucker format, this TT network can be contracted along several paths, yielding additional opportunities to relax the computational strain of our network.

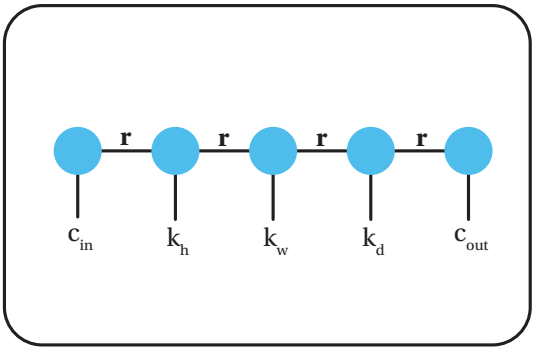


Figure 3.8: Convolutional weight tensor, represented in a **tensor train format.**

The proof of the pudding...

In practice, these lightweight layers are implemented as follows. When an input tensor is processed by the layer, it is first unfolded. The weight tensor is reparametrized in one of the aforementioned formats by creating a tensor network, in which the tuning parameter accommodates a compression rate that is chosen in advance. The unfolded input tensor is then connected to this network, and the optimal contraction path is obtained using a greedy search algorithm. The full tensor network, finally, is contracted along this path to yield the output tensor. Figure 3.9 summarizes our approach.

In the next chapter, a methodology is described to put these low-rank, lightweight layers to the test. Using an established, fully convolutional segmentation architecture—an nnU-Net [35]—as a baseline, we will evaluate the performance of each tensor network format with varying degrees of compression. To achieve this, all convolutional weight tensors in the network's double convolution blocks will be replaced by the respective low-rank variants, yielding a set of TNNs. These models will be trained, and their characteristics and performance metrics will be (statistically) evaluated.

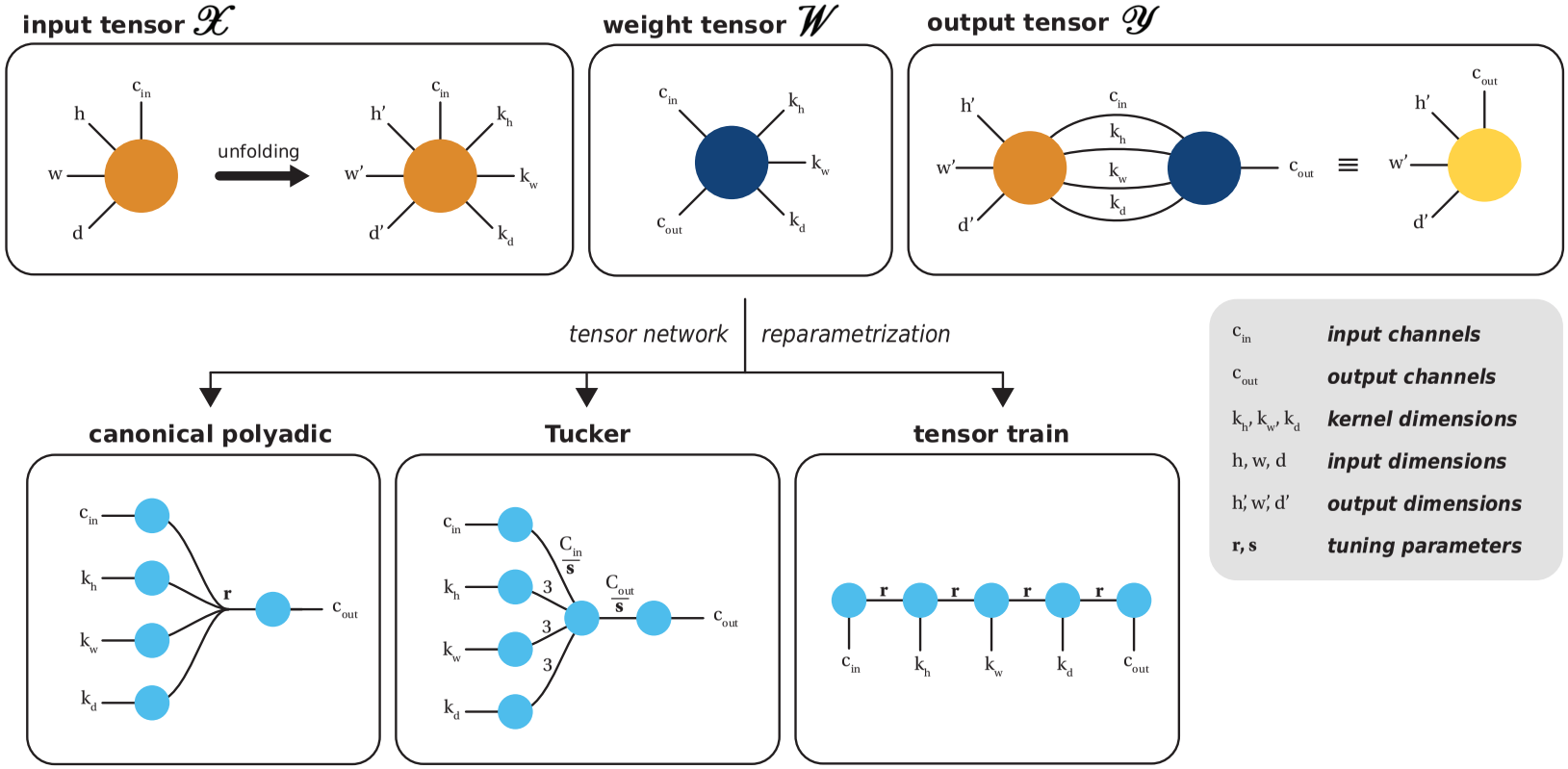


Figure 3.9: **Lightweight layers.** A convolutional layer can be represented by the contraction of an unfolded input tensor $\mathscr{X}$ with a weight tensor $\mathscr{W}$ to produce an output tensor. In our approach, the weight tensor is replaced by a (compressed) tensor network, consisting of smaller tensor nodes or *cores*. This tensor network is then connected to the input tensor along the kernel modes and the input channel mode of the appropriate nodes. The optimal contraction sequence is then obtained through recursive depth-first search. The result is a memory-friendly, computationally light alternative for the traditional convolutional layer. Note that the output dimensions may differ from the input dimensions, depending on the *stride*, *padding* and *kernel dimension* parameters (e.g., in Isensee et al. [35]'s use of stride to exact downsampling).

I have not failed. I've just found 10,000 ways that won't work.

Thomas A. Edison inadvertently sums up my experience exploring model parallelism in training across multiple GPUs.

Chapter 4

Method

The focus of this thesis is to evaluate the impact of different low-rank parametrizations of convolutional layers on memory, computational complexity and performance. To achieve this, we trained a sizeable number of segmentation models in a 5-fold cross-validation scheme.

The first of these models served as our **baseline**. More specifically, we drew from Isensee et al. [35]—a study in which the nnU-Net framework was applied to the 'Multimodal Brain Tumor Image Segmentation Benchmark' dataset (BraTS) of 2020 [58]. The nnU-Net ('no new-net') framework [33, 34] represents the idea (and observation) that, despite the introduction of various more recent architectures, state-of-the-art performance can still be reached for segmentation tasks using an *only slightly* modified U-Net model. Our own implementation thus heavily mirrors that of Isensee et al. [35], barring some differences that will be discussed throughout section 4.2.

The remaining models, i.e. the **tensorized** variants or TNNs, have an architecture that is largely parallel to the baseline, except for the *double convolution* blocks. In these blocks, all convolutional layers are replaced by 'lightweight' layers, which represent one of the parametrizations described in section 3.4. This parametrization is uniquely defined by four[1] factors:

1. the chosen **tensor network** (e.g., canonical polyadic) format
2. the **compression rate**
3. the number of **input channels**
4. the number of **output channels**

The first two factors are chosen ahead of training, whereas the numbers of input and output channels depend on where they occur in the architecture (e.g., level 3, see fig. 4.2).

To ensure a fair comparison of their respective performance, all models were trained using the same procedure. This includes the 5-fold data split, the preprocessing pipeline, and all training parameters (e.g., optimizer, learning rate scheduler). All of these parameters are detailed in the sections below, along with our rationale in support of their selection, where appropriate.

[1] Technically, the kernel size also plays a role in this parametrization. In our network, however, each kernel (in the double convolution blocks) is invariably a $3 \times 3 \times 3$ tensor.

4.1 Dataset

The BraTS dataset [58, 5, 6] is part of the Medical Segmentation Decathlon initiative [76], a collection of medical imaging datasets that are openly available in order to "*facilitate the development of semantic segmentation algorithms*" [76, p. 2]. The BraTS training dataset, specifically, contains a set of *volumetric brain images*, obtained from 369 *glioma* patients. A glioma is a common type of brain tumor that is associated with a somber prognosis, depending on the tumor variant and choice of treatment. Aggressive variants such as glioblastoma, for instance, have a median survival rate under two years [54]. Speedy and accurate diagnosis of the affliction, therefore, is of the essence.

The assessment strategy for suspected gliomas typically involves the use of imaging protocols [58], such as **MRI**. An MRI image, in essence, is the result of a question-answer process. First, a certain area of interest (e.g., the head) is subjected to an external uniform magnetic field. In doing so, the protons in the tissue's water nuclei are 'magnetized'—that is, they are coerced into alignment with this field. Next, a Radio Frequency (RF) energy pulse is applied to perturb this magnetized state, after which these nuclei return to their resting alignment. In doing so, they themselves emit RF energy, which is recorded by the MRI device after a fixed interval. This protocol can yield different types of images—that is, different *modalities*—depending on three factors [69]:

1. The **repetition time (RT)**, or the interval between successive RF pulses;
2. The **time to echo (TE)**, or the time interval between a pulse and the registration of its effect;
3. The use of **contrast agents**, most commonly *Gadolinium*.

Since different modalities tend to illuminate different aspects of the examined tissue—e.g., the use of contrast agents can shed light on certain vascular structures—their combination is particularly potent in the diagnostic process. The BraTS 2020 dataset contains MRI images in four modalities (fig. 4.1) [69]:

- T1-weighted (short TR and TE)
- T2-weighted (long TR and TE)
- Fluid Attenuated Inversion Recovery (FLAIR; very long TR and TE)
- T1-weighted post-contrast (after injection with Gadolinium).

In addition, the dataset holds **target labels**: manual segmentation labels that were added by one to four raters, and approved by expert neuro-radiologists (fig. 4.1). Each voxel (resolution: $1mm^3$) is assigned to one of the following categories: healthy tissue, enhancing tumor (ET), peritumoral edema (ED), or non-enhancing tumor core (NCR/NET). In this thesis, however, we use labels according to the **hierarchical majority vote** scheme [58, Algorithm 1, p.8], which produces the following nested label structure:

- class 1: **enhancing tumor** (ET)
- class 2: **tumor core** (TC: ET, non-enhancing tumor, and necrotic tissue)
- class 3: **whole tumor** (WT: tumor core and edema)

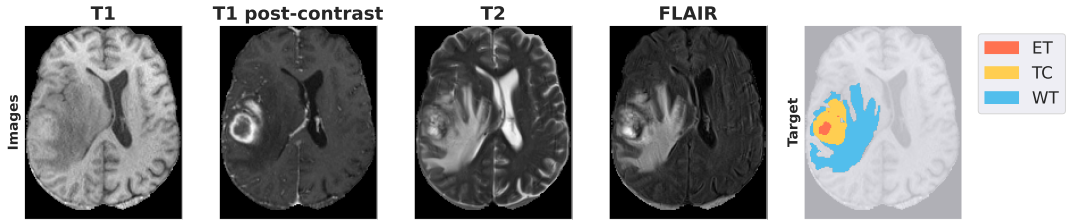


Figure 4.1: **BraTS dataset overview.** Horizontal slices of subject 102's data ($Z = 85$ after foreground cropping) in each available modality, followed by the target segmentation labels (ET = enhancing tumor, TC = tumor core, WT = whole tumor).

Finally, it is important to note that the BraTS 2020 dataset is the product of several preprocessing steps, such as skull-stripping, co-registration of all images to a single anatomical template, and interpolation. Any potential real-world application of segmentation models, such as the ones described in this thesis, would naturally have to include a similar preprocessing pipeline.

4.2 Architectures

All experiments were conducted using variants of a single U-Net variant, as described in Isensee et al. [35]. Minor modifications were made to the baseline architecture. Every TNN was derived from this architecture, swapping only the double convolution blocks.

4.2.1 Baseline network

Isensee et al. [35] presented a U-Net variant that was molded specifically to cater to the specifics of the BraTS 2020 challenge (fig. 4.2). In this section, we will describe its architecture, highlighting where our own models diverged from this blueprint.

The model's working principle is that of an autoencoder. Through a series of convolutional blocks and downsampling operations, the input is *encoded* into a 320 feature map of size $4 \times 4 \times 4$ (level 6). Then, a *decoding* phase follows, in which convolutional blocks are alternated with upsampling operations, reconstructing output of the same dimensions. As a result, we obtain predictions for each individual voxel. The classification of these voxels thus gives rise to one or more segmentation maps, depending on the labeling scheme and the number of output channels. Importantly, there are skip connections between mirroring convolution blocks, facilitating the preservation of details that may otherwise get lost as the receptive field of each subsequent convolution layer broadens.

Double convolution blocks The core operations in a U-Net architecture can be grouped in blocks, often labeled as *double convolution* blocks. In such a block, there is a two-fold succession of a convolutional layer, a normalisation layer, and an activation layer.

In all convolutional layers, input and output padding are set to 1 (zero-padding). The default stride is $(1,1,1)$, except where mentioned otherwise. Kernel size is $(3,3,3)$. The output of each 3D convolutional layer is passed through a *group normalization* layer [85].

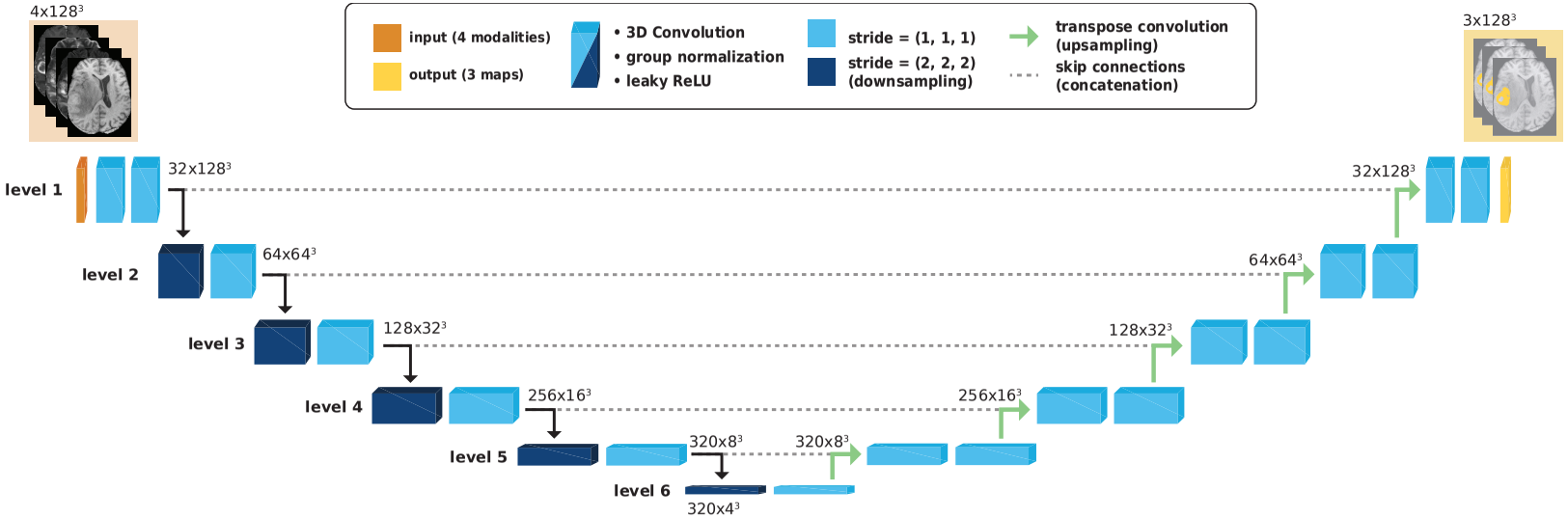


Figure 4.2: **Baseline U-Net architecture**, derived from Isensee et al. [35]. Feature map sizes are indicated for every double convolution block.

Contrary to *batch normalization* [32], in which case values are normalized across the batch dimension (and the spatial dimensions), group normalization normalizes a number of channel groups (i.e., along the spatial dimensions and a group of channels). Doing so is advantageous in terms of memory demands. A potential alternative is *instance normalization* (used in Isensee et al. [35]), which is functionally identical to batch normalization with a batch size of 1. However, group normalization has the added advantage of exploiting cross-channel dependency, making it the preferred option (as demonstrated in Wu and He [85]). In our architecture, groups are made up of 8 channels.

The output of the normalization is passed on to a *leaky rectified linear unit* (LeakyReLU) activation function (fig. 4.3). The traditional ReLU activation function has several advantages: it is computationally light, yields sparse activations (as many are set to 0), speeds up convergence by avoiding plateaus (as would be present in, e.g., the *tanh* function), and helps mitigate the 'vanishing gradient' problem. The downside of the aforementioned sparsity, however, is that certain neurons never escape 'inactivation'—a phenomenon known as the *dying ReLU* problem. The 'leaky' variant of ReLU adds a small slope (= 0.01) to the negative domain of the function, mitigating the issue. Moreover, there is empirical evidence that the use of leakyReLU benefits the networks performance [86].

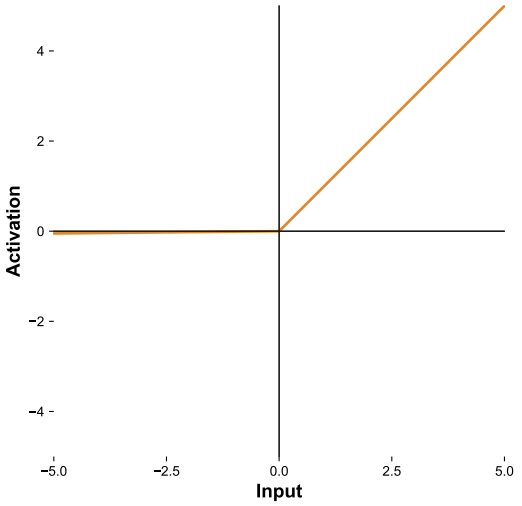


Figure 4.3: **Leaky Rectified Linear Unit activation.**

Downsampling In the encoder part (also: the *contracting path*), strided convolutions are used to downsample the input. To this end, the first convolutional layer in each of the encoder's double convolution blocks employs a stride of 2 along each spatial axis. Strided convolutions have some advantages over traditional downsample operations, such as max pooling (e.g., as used in Ronneberger et al. [70]): as they can combine the convolutional operation with a downsampling step, this approach puts less computational strain on the network. In addition, the operation is generalizable, in the sense that the weights associated with the convolution have to be learnt. As such, strided convolutions can adapt to maximize their performance in the context of the task. At the same time, it has been shown that this approach—trading traditional max pooling layers for strided convolutions—is of no detriment to the model's performance, particularly when the network is large enough [78].

Upsampling In the decoder section (also: the *expansive path*), upsampling was implemented using transposed convolution [19]. Parallel to what was described in the previous

paragraph, transposed convolution is a generalized technique—weights can be learnt to accommodate the task at hand. This is not the case for every other upsampling operation, such as *max unpooling*.

Final activation Training on nested tumor regions (see section 4.1) rather than the individual labels has been shown to improve performance on the BraTS dataset (see, for instance, BraTS 2019 challenge winner [39]). Therefore, rather than yielding a single output channel to be paired with a softmax activation, our final double convolution block yields three $128 \times 128 \times 128$ segmentation maps, one for each of the nested glioma subregions. Training then passes these values—i.e., the *logits*—through a *sigmoid activation layer*. For inference purposes, this output is then thresholded (at a value of 0.5) to yield discrete classification labels. Note that, unlike in Isensee et al. [35], no deep supervision scheme is used (see also: [51]).

4.2.2 Tensorized networks

All TNNs were created by exchanging all convolutional layers in the double convolution blocks with various tensor network formatted variant (our lightweight layers). As these layers represent the brunt of the effort required of the network, their replacement yields an opportunity to evaluate the impact of our layer tensorization on memory, computational complexity, and performance.

Implementation of these new, lightweight layers was as follows. In each convolutional layer, the weight tensor was replaced with a tensor network, representing a specific *format* with a certain degree of *compression*. This network was connected to the (unfolded) input tensor. The optimal (i.e., computationally most efficient) contraction path was obtained, and the tensor network was contracted. Bias was added to the resulting tensor.

Importantly, to facilitate a fair comparison, all weights were initialized using the same scheme that is used for the 3D convolution, as implemented in `Pytorch`—the Kaiming uniform distribution ($a = \sqrt{5}$) [28]. Similarly, bias was initialized using a normal distribution.

As indicated in chapter 3, the following tensor network formats were implemented:

1. Canonical polyadic format
2. Tensor train format (version 1)
3. Tensor train format (version 2)
4. Tucker format

Each of these formats leads to eight network variants, depending on the value chosen for the compression rate. Possible compression rates were **2**, **5**, **10**, **20**, **35**, **50**, **75**, and **100**.

4.3 Training

4.3.1 Preprocessing and augmentation

The raw image data, as provided by the BraTS challenge, was first passed through a preprocessing and augmentation pipeline consisting of the following steps.

1. Target labels were one-hot encoded according to the hierarchical scheme described in section 4.1.
2. The foreground (i.e., the brain portion of the image) was cropped from the volume, trimming away empty space.
3. (**augmentation**) A $128 \times 128 \times 128$ patch was randomly cropped out of the image.
4. (**augmentation**) A random affine transformation—that is, a geometric transformation preserving lines and parallelism—was applied to the image.
5. (**augmentation**) The intensity of the image was randomly scaled by $v = v \times (1 + f)$ ($f \in [-0.1, 0.1]$, with a probability of 0.5.
6. (**augmentation**) The intensity was also randomly shifted with an offset o ($o \in [-0.1, 0.1]$) with a probability of 0.5.
7. The image's intensity was normalized per channel.

4.3.2 Training procedure

Train-test split The BraTS dataset consists of two parts: 369 training image sets, which include segmentation labels, and 125 unlabeled validation (or more accurately: test) image sets. Since the latter images can only be scored using the online BraTS platform, and the scoring process is typically time-consuming, we opted to evaluate our models using a *5-fold crossvalidation scheme*, using only the BraTS 2020 training set. More explicitly, we randomly[2] divided the training set into 5 portions, after which for each model, 5 versions were trained. Each of these versions tried to predict one of the 5 data segments, after training on the other 4 segments. This way, we obtained predictions for every subject in our dataset.

Loss All models were trained using a smoothed *dice loss* [59], which was calculated as follows:

$$L_{dice} = 1 - \frac{2\,\langle P, G\rangle + 1e^{-5}}{\|P\|^2 + \|G\|^2 + 1e^{-5}} \tag{4.1}$$

where P represents the probability map yielded by the network; G is the (one-hot encoded) ground truth; $\langle \cdot, \cdot \rangle$ is the tensor *dot product*; and $\|\cdot\|$ is the Frobenius norm (i.e., $\|X\| = \langle X, X\rangle$). To avoid division by zero, the loss is smoothed by adding a small constant ($1e-5$) to both the numerator and the denominator.

[2] Random seeds were set across the entire project to ensure reproducibility.

Optimizer To guide the model towards its goal—minimizing the loss function—an optimizer is required. Optimizers are algorithms that determine how the error gradient translates to a network weight update. Here, we chose *Adam with decoupled weight decay (AdamW)* [53]. Like its cousin *Adam* [44], *AdamW* is an adaptive optimizer—it accommodates for a parameter-wise update in *learning rate (LR)*[3], a hyperparameter that dictates the magnitude of the gradient's impact on the weights. It is different from *Adam*, however, in that it removes the weight decay[4] from the gradient-based update. Loshchilov and Hutter [53] provide evidence that the weight decay step is unwarranted in the optimization procedure, and show how their modified optimization algorithm improves Adam's generalization performance. We used a(n initial) learning rate of $1e^{-4}$, and a weight decay of $1e^{-5}$.

Learning rate scheduler As the training process unfolds, a network moves towards an optimum. It is likely that, initially, the network state is far removed from said optimum, in which case a large LR will benefit convergence. As the optimum draws nearer, however, we must take care not to *overshoot*. Decreasing the LR, in this case, helps a model to gently inch closer. Adjusting the LR based on training parameters (such as the epoch, or the evolution of the loss function) is precisely the task of a scheduler.

Over the past years, several scheduler types have been proposed, from the vanilla *exponentially decreasing LR* to various *cyclical LRs* [77]. In this project, we used *cosine annealing with warm restarts* [52] (fig. 4.4), which combines several ideas. First, the learning rate starts off high, to then rapidly decrease along a cosine trajectory. This embodies the idea mentioned above: take great strides towards an optimum, then shuffle closer once you get nearer. After a number of epochs, however, the learning rate is reset to its initial value, but the weights are retained: a *warm restart*. The rationale here is that good weights have been learnt in the previous phase, placing us in a good position to try and learn a 'new' model.

In our implementation, the learning rate was reset every 50 epochs, and the minimum learning rate was set to $3e^{-5}$. Note that, despite *AdamW* being an adaptive optimizer, this does not preclude a learning rate scheduler from having a positive impact on training performance, as argued in Loshchilov and Hutter [53, p. 2].

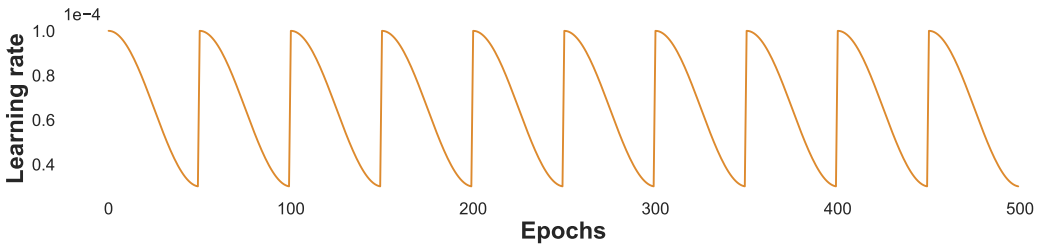


Figure 4.4: **Cosine Annealing with Warm Restarts.**

[3] "*The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.*" [44, p. 1]

[4] Weight decay is a technique that imposes a penalty on the complexity of our model weights—smaller values are preferred to avoid overfitting.

Batch size All models trained with a batch size of 1, to limit the baseline strain on GPU memory (in Isensee et al. [35], the batch size is 2 or 5, depending on the model).

Test inference To obtain test scores, predictions were obtained using a sliding window inference scheme, using a $128 \times 128 \times 128$ patch size, and a 0.25 overlap. Predictions were given the same weight across the patch—that is, predictions were not weighed differently depending their position in the patch (e.g., using a Gaussian distribution).

4.4 Measures

In order to compare the baseline model to the TNNs, we calculated model characteristics and performance metrics.

4.4.1 Performance

To quantify performance, two main metrics were used: the Dice score and the 95% Hausdorff distance.

Dice score

The Dice score, which also inspired the loss function (eq. (4.1)), is calculated as follows:

$$D(P,G) = \frac{2\langle P,G\rangle}{\|P\| + \|G\|} \tag{4.2}$$

From this equation, it follows that a perfect overlap of the prediction P and ground truth G will yield a score of 1, whereas a total lack of commonality will give a 0 score.

95% Hausdorff distance

The Hausdorff distance refers to the largest distance between an area and the nearest point in a second area. It is defined as:

$$H(P,G) = \max(h(A,B), h(B,A)) \tag{4.3}$$

with

$$h(P,G) = \max_{p\in P} \min_{g\in G} \|p-g\| \tag{4.4}$$

In eq. (4.3) and eq. (4.4), P and G represent the boundary point sets of the prediction and ground truth, respectively. p and g, in turn, represent points (i.e., voxels) in these respective sets, and the $\|\cdot\|$ represents the Frobenius norm.

The 95% Hausdorff distance slightly differs from this definition, as it is based on the 95^{th} percentile of the distances (rather than the maximum), leaving the metric less vulnerable to outliers.

Note that the Hausdorff distance metric is sensitive to geometrical characteristics, such as the regularity and size of the ground truth and predicted shapes.

4.4.2 Model characteristics

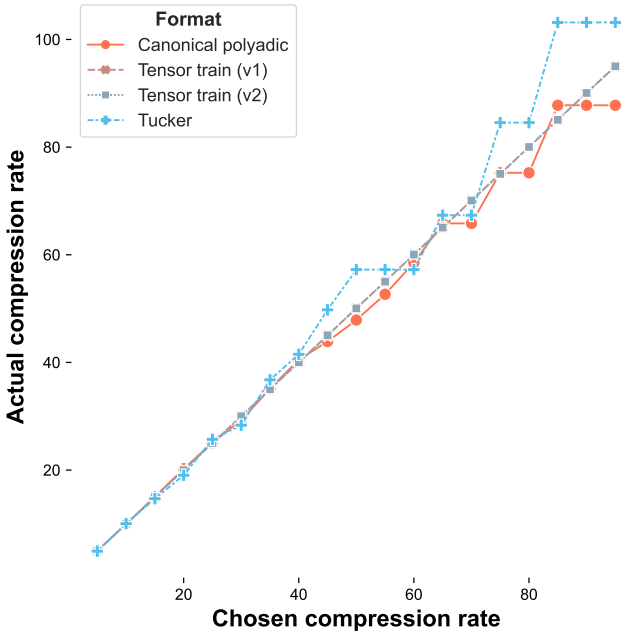


Figure 4.5: **Chosen compression rate versus actual compression rate.** Higher compression rates are harder to tune for, particularly for the CP and Tucker formats.

For each model, both the number of **multiply-accumulate (MAC)** operations and the total number of **model parameters** were obtained.

MACs

A MAC represents an atomic set of operations (a multiplication and an addition)[5], and thus functions as a unit measure for a model's computational demand. The larger its MAC count, the more calculations the network must execute en route to its output, and the more expensive it is to operate.

To compare MAC counts across different models, we use a percentual difference measure:

$$\Delta\text{MACs}_{TN}(\%) = 100 \times (1 - \frac{\text{MACs}_{TN}}{\text{MACs}_{baseline}}) \tag{4.5}$$

Note that negative values, when using this measure, correspond to an increase in MACs compared to baseline (ergo, *diminished* computational performance), whereas positive values signify a decrease in MACs (and therefore *improved* computational performance).

[5]A different, commonly used measure is the **floating point operation (FLOP)**. One MAC, then, is two FLOPs. For consistency's sake, we will only use MACs from here on out.

Network parameters

The number of model parameters, in turn, represent how heavily the network weighs on the available memory resources. Note that, while compression rates were set to fixed values across all TNNs, the total number of parameters could still vary. This is due to the fact that a compression rate was effected by tuning each respective tensor network's hyperparameter (see chapter 3) to approximate the required number of layer parameters. However, as these tuning parameters were necessarily *integers*—a bond dimension *cannot* be represented by a decimal number—this approximation could be *not exact*. As the compression rate increases, the dimensionality of the tensor cores becomes smaller. The impact of tuning parameter values is thus coarser, and the actual compression rate less precise. Figure 4.5 illustrates this point. The higher the requested compression rate, the more the approximation worsens.

Aside from the total number of network parameters, we calculate a network compression rate: the degree with which the full network is compressed as a result of our lightweight layer substitutions.

4.5 Implementation

The full code repository can be found here:
`https://github.com/WouterDurnez/airhead`.

All models, including the lightweight layers, were implemented in `Pytorch` [67]. To facilitate training, we made use of the `Pytorch Lightning` package [20]. Various bits of useful functionality, such as performance metrics and data transforms, were implemented using the `MONAI` package. Model parameters and MAC counts were obtained using the `ptflops` package. For the Tucker and tensor train formats, tuning parameters were calculated using the symbolic mathematics package `sympy`. Under the constraints that a solution must be positive and real, a unique solution was always found. As was mentioned earlier, the bond dimensions that were determined by these tuning parameters were rounded to the nearest integer, which could cause our actual compression rate to deviate slightly from the chosen value.

All models were trained on the Joltik cluster of the HPC of Ghent University. Each model was trained on a single NVIDIA Volta V100 GPU[6].

Theory, meet practice The implementation used for model training and inference deviated somewhat from the theoretical plan. In theory, the idea behind our 'lightweight layers' is to reparametrize the weight tensor into a 'decomposed' format, which then forms a tensor network along with an (unfolded) input node. We then use the `opt_einsum` package to determine the optimal contraction sequence, as described in section 4.2.2.

[6] Per model, the requested walltime was 50 hours—actual training time was around 40 hours. Cumulatively, not including failed attempts, the project's total GPU time was approximately 275 days. Prompted by a joking colleague, prof. dr. ir. Pieter Simoens, I have planted two trees via `http://www.treesforall.nl` to offset this footprint.

This approach should lead to an optimal (i.e., minimal) MAC count. Importantly, this very approach was used to obtain said MAC counts for all networks.

However, when training was initiated, some practical problems arose. Unfolding the input tensor manually for contraction with the decomposed weight tensor consistently led to *out of memory* errors, as the unfolding operation took a heavy toll on GPU RAM. Model parallelism was explored (e.g., using packages such as `fairscale`), but ultimately deemed out of scope, given the experimental nature of such functionality. Therefore, in our training implementation, the weight tensor network (i.e., not including the unfolded input tensor) was first contracted, after which `pytorch`'s native `conv3d` implementation was used to produce the convolution with the input tensor. While both implementations should lead to the same output at all times, the latter approach effectively constrains the search for an 'optimal contraction path': the input is always contracted last.

This also illustrates why *test time* would be a poor measure to compare our tensorized models with the baseline: the implementation used for training does not fully reflect our intention. However, even if memory were no issue, and we were able to contract tensor networks that include the unfolded input tensor, a comparison on the basis of test time would not be fair. Libraries of `pytorch`'s caliber are usually optimized in a lower-level language (e.g., C [42]), which is superior to anything we could implement in pure python. This is particularly true for expensive operations like convolutional layers. As such, a comparison in terms of (theoretical) MACs is more appropriate, as this (again, theoretically) extrapolates to a test time difference between would-be optimized implementations.

You never know
what you're gonna get.

Tom Hanks would not be surprised by the fickle impact of low-rank parametrizations on CNN performance.
Forrest Gump (1994)

Chapter 5

Results

In this chapter, the characteristics and performances of our networks are discussed. We determine the impact of our interventions (substituting lightweight layers) by evaluating the performance and model characteristics of our tensorized neural networks. In particular, we consider the differential impact of different tensor network formats and compression rates, and compare their performance on specific parts of the target annotation.

5.1 Network characteristics

The characteristics of our model are illustrated in fig. 5.1. In table A.1 we supply the following data: MAC counts, network parameter counts, network compression rates (per layer compression rate), and the percentual difference in the resulting MAC counts.

Network parameters Our baseline model has 28,325,699 parameters. Replacing the convolutional layers in the double convolution blocks with lightweight layers has an impact on this number as expected. The choice of tensor network format has little impact on the number of parameters when the compression rate is fixed. While the actual compression rate of a low-rank layer may differ slightly from the chosen compression rate (due to rounding issues, see section 4.4.2), such inaccuracies likely cancel each other out when multiple weight tensors of different sizes are compressed in a single model. Note that the compression rate, which we applied to a subset of (convolutional) layers, is not proportional to the total number of parameters, nor to the resultant network compression rate. This is due to the residual *uncompressed layers* (e.g., the transposed convolutions), which limit the maximal obtainable scale factor. In the remainder of our analyses, we will discuss compression rates in terms of either the *layer compression* for which we tuned our lightweight layers, or the resultant overall *network* compression.

MACs The computational effort that is associated with a pass through our baseline network, amounts to 495,867,248,640 MACs. For the CP format, this number is only improved upon when the chosen layer compression rate is 20 and greater. The other formats—the two TT formats and the Tucker format)—lead to better-than-baseline MAC counts from compression rate 10 onward. The Tucker format, finally, shows a minute

improvement over the baseline for a compression rate of 5. All other format × compression rate combinations are at least as computationally taxing as the baseline network, with the CP format showing a marked increase in MAC counts for low compression rates (2 and 5).

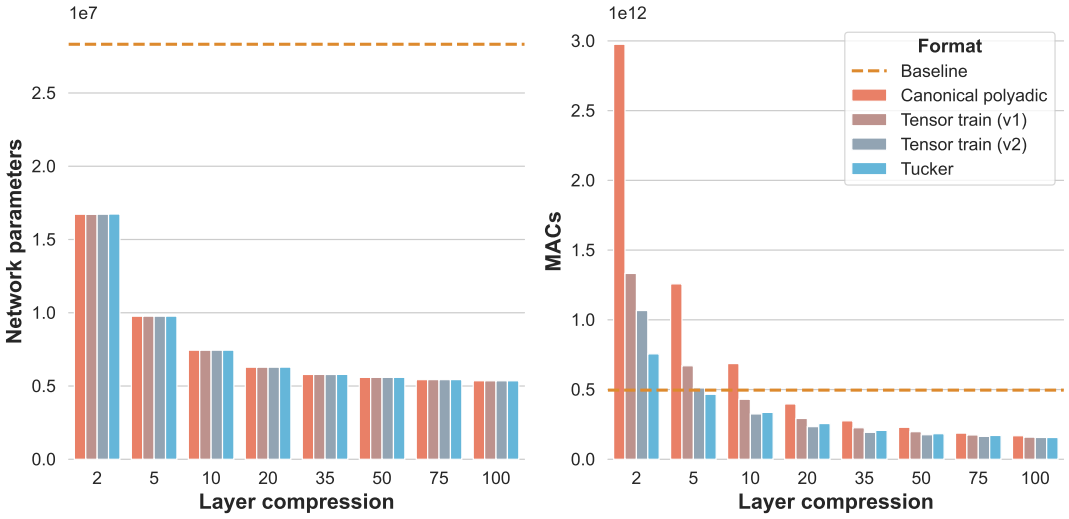


Figure 5.1: **Network characteristics.** Total number of network parameters (left) and MAC counts (right) for both the baseline network (dashed line), and each of the tensorized networks for all compression rates considered (solid bars).

5.2 Network performances

5.2.1 Baseline network

The Dice scores and 95% Hausdorff distances for the baseline network are shown in fig. 5.2. Aggregated metrics are given in table 5.1. While this performance is acceptable, these scores are by no means competitive with the results obtained by state-of-the-art networks. This was not the main goal of the present work: in this instance, the baseline network served merely as a reference for the experimental tensorized networks. Note that we do not apply any post-processing, nor do we use techniques such as deep supervision [33].

Dice scores indicate that our baseline network had more difficulty predicting smaller regions—whole tumor regions are better predicted than tumor core regions, which are more easily inferred than the notoriously difficult enhancing tumor region. This is to be expected; a similar pattern can be observed in other networks (e.g., [35, 39]).

5.2.2 Tensorized networks

In this section, the performances of the experimental, tensorized networks are compared to results from the baseline network. Figure 5.3 visualizes predictions given by the baseline network, as well as a subset of our tensorized networks (compression rates 35 and 75 are not included in the image). For simplicity's sake, we refer to models by their tensor

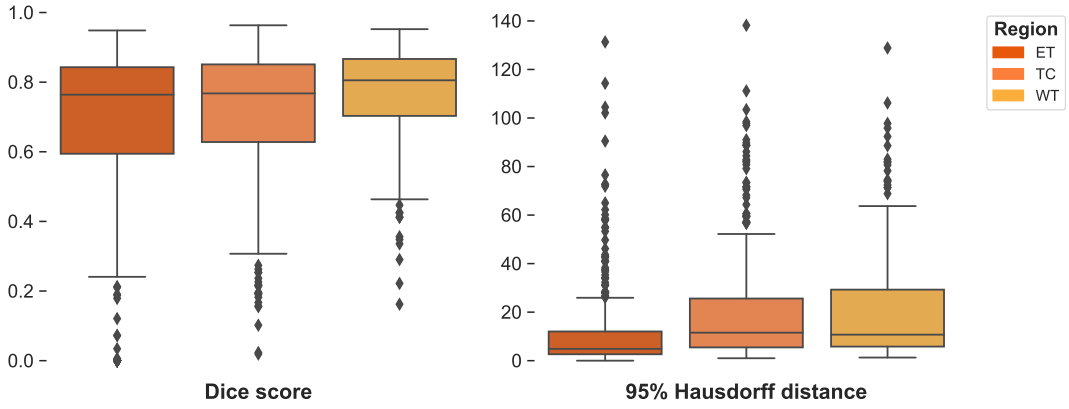


Figure 5.2: **Baseline performance metrics.** Box plot of the distribution of the Dice scores and Hausdorff distances, split out over the hierarchical target regions.

Table 5.1: Baseline performance metrics

	Dice score			**95 % Hausdorff distance**		
	ET	*TC*	*WT*	*ET*	*TC*	*WT*
Mean	0.664	0.714	0.771	12.088	20.347	20.207
SD	0.263	0.190	0.132	18.290	23.147	21.489
Median	0.764	0.768	0.805	4.863	11.534	10.720

network format and layer compression rate—e.g., when discussing the model in which convolutional layers were substituted with CP low-rank layers, with a compression rate of 2, we refer to it as CP-2, and so on.

A note on statistical comparisons Statistically comparing every possible network pair (e.g., baseline vs. CP-2, or Tucker-10 vs Tucker-100) for each of the label regions would yield an enormous number of tests. The figures, however, can be intuitively interpreted to a degree based on the following principle. When 95% confidence intervals do not overlap, two sets of data points differ with statistical significance [3]. The opposite is not true: should 95% confidence levels overlap, there may still be a significant difference. The following sections, however, focus on comparing the TNN performances with that of the baseline network.

Given the non-normal nature of our data (see, for instance, the skewed distributions in fig. 5.2), statistical comparisons were made using the non-parametric Mann-Whitney U test [56]. Since we are evaluating a large body of results, we did not adjust for multiple comparisons to avoid inflating the type II error (i.e., accepting the null hypothesis, "*there is no difference*", when in reality, the samples are different) [71].

These choices (focusing on baseline comparisons, and not correcting for multiple comparisons) were also made from the perspective that, in a medical setting, losing

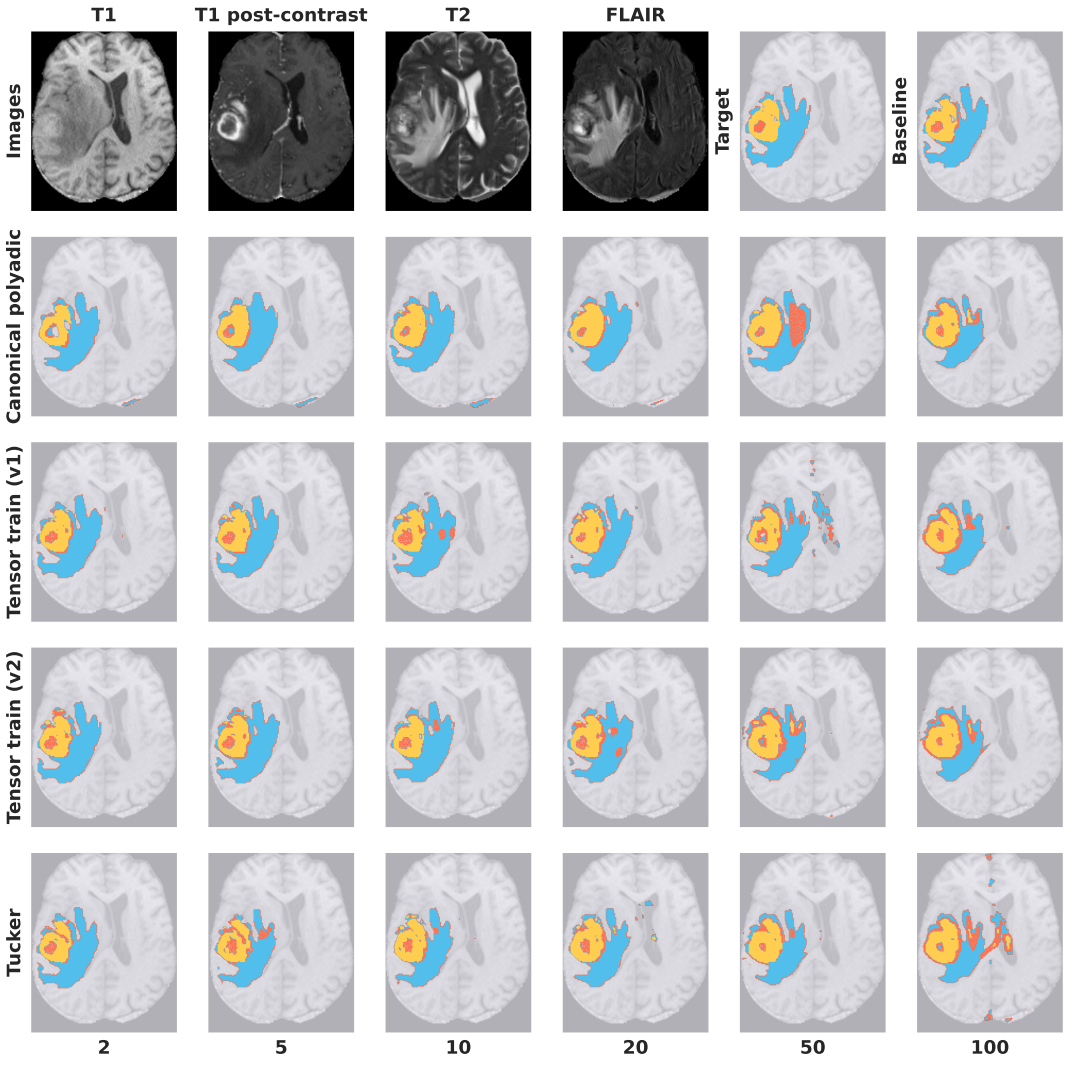


Figure 5.3: **Network prediction overview.** *First row*: Horizontal slices of subject 102's data ($Z = 85$ after foreground cropping) in each available modality, followed by the target segmentation labels (ET = enhancing tumor, TC = tumor core, WT = whole tumor), and the baseline model prediction (for the appropriate fold in the crossvalidation scheme). *Remaining rows*: predictions given by all tensor network formats for a selected range of compression rates. Note that, with growing compression rates, the ET region becomes particularly hard to accurately predict.

performance and failing to detect it is worse than performing on par with the baseline, but finding a statistical difference nonetheless.

Figure 5.4 shows the Dice scores and 95% Hausdorff distances yielded by our TNNs, juxtaposed against our baseline model, for different degrees of network compression. Figure 5.5 shows the same results, plotted against the networks' MAC counts. All means and standard deviations are provided in table A.2. Statistical comparisons to baseline performance, for both performance metrics and across all compression rates, are listed in table A.3.

Dice scores

The CP format underperformed consistently, failing to statistically meet the reference model at any compression rate. Tucker yielded slightly better results, reaching baseline performance at the lowest compression rate (Tucker-2), yet only for the ET ($U = 67338.0$, $p = .399$) and TC ($U = 65118.0$, $p = .153$). Tucker-2 corresponded to a 115.109 % increase in network MACs, and a network compression rate of 1.7. Scores for the WT region were significantly lower ($U = 57098.0$, $p < .001$), as were all Tucker network performances at higher compression rates (all $p < .001$).

In comparison, both versions of the TT format performed well up to layer compression rates of 20—that is, a network compression rate of 4.5, and a 50% MAC decrease—after which performance dropped sharply. Nonetheless, significant decreases in Dice scores were detected for all networks and regions, except for the ET ($U = 66811$, $p = .331$) and TC ($U = 67357$, $p = .401$) regions with the TT(v1)-2 network, and the TC ($U = 65601$, $p = .196$) region with TT(v2)-2 network. This corresponds to a 1.7 network compression rate, and a 168% and %115 MAC increase for the TT (v1) and TT (v2) networks, respectively. Larger layer compression rates always led to significant Dice score decreases for all regions.

95% Hausdorff distances

The CP format yielded an equal-to-baseline Hausdorff distance for the ET region ($U = 64956$, $p = .140$), yet *only* for a layer compression rate of 10 (network compression rate: 3.804, MAC increase of 38.407%). No other CP network reached baseline performance, however, for any of the regions ($p < .05$). Similarly, all of the Tucker networks underperformed significantly (all $p < .05$) for every region, compared to baseline.

The TT networks, however, performed markedly better. The TT (v1) networks, in particular, yielded equal-to-baseline performances for the ET region up to a layer compression rate of 20 (TT-2 (v1) $U = 65449$, $p = .182$; TT-5 (v1): $U = 64883$, $p = .131$; TT-10 (v1): $U = 66255$, $p = .264$; TT-20 (v1): $U = 65248$, $p = .164$), corresponding to a network compression rate of 4.508, and a 40.975% decrease in MACs. For the TC region, the TT-2 (v1) met baseline performance $U = 64284$, $p = .095$, and the TT-10 (v1) outperformed the baseline ($U = 62741$, $p < .05$). For TT (v1), this layer compression rate leads to a factor 1.694 network compression rate and a 168.741% increase in MACs.

The TT (v2) format yielded somewhat similar results. For a layer compression rate of 2 (network compression rate: 1.694, MAC increase: 115.109%), the network matched (ET: $U = 63539$, $p = .058$; WT: $U = 65552$, $p = .192$) or improved on (TC: $U = 60157$, $p < .01$) the

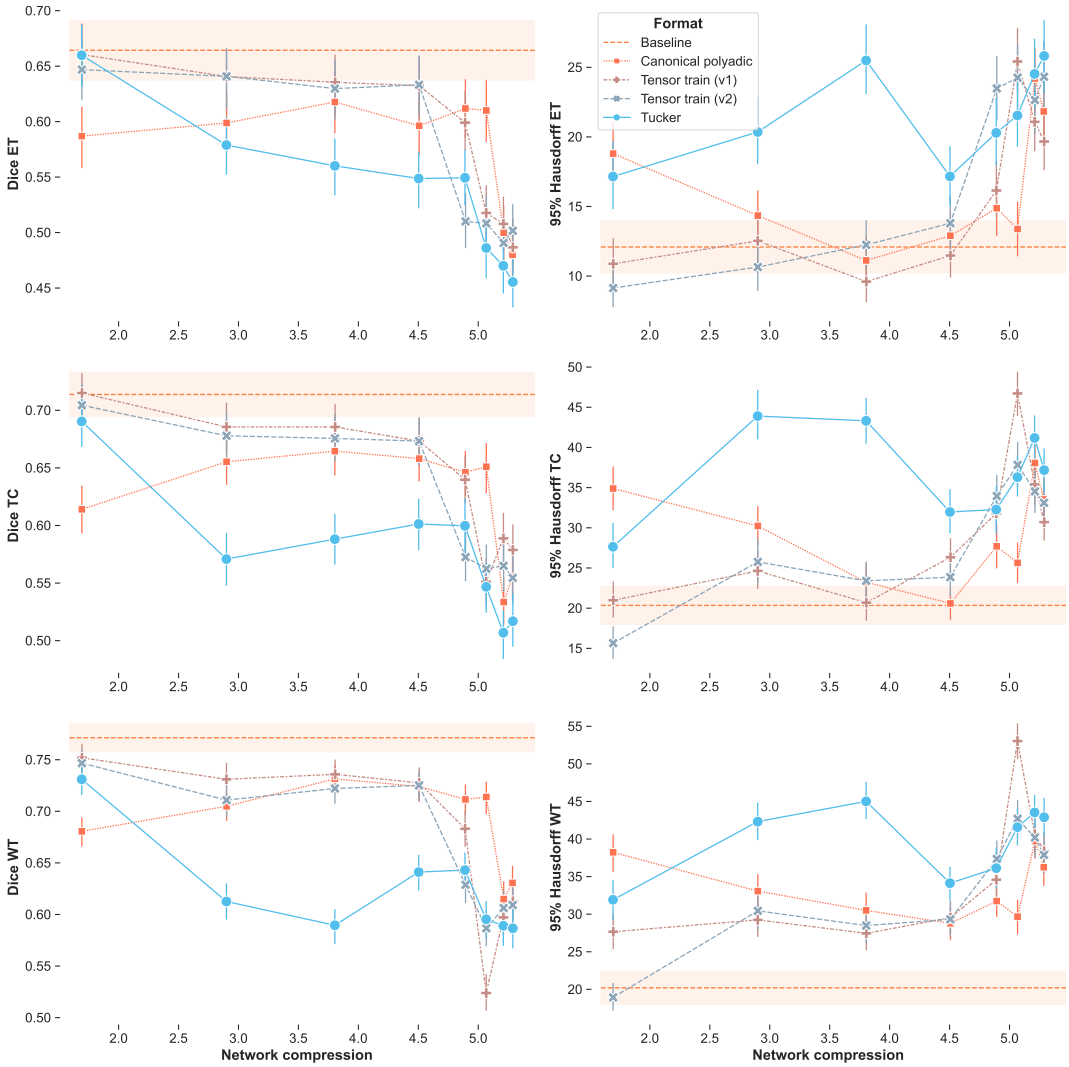


Figure 5.4: **Performance metrics versus network compression rate.** Dice scores and 95% Hausdorff distances for all subregions of the glioma, for varying network compression rates. The dashed orange line and its encompassing bar represent the performance of the baseline network, with its 95% confidence interval. The markers represent the performances obtained by different tensorized networks, for varying degrees of network compression. Whiskers again indicate 95% confidence intervals. Performances towards to top right (Dice score) or bottom right (Hausdorff distance) represent a better trade-off between performance and network compression.

baseline. The TT-5 (v2) network also achieved baseline performance ($U = 67349$, $p = .400$) for the ET region, corresponding to a network compression rate of 2.9, and MAC increase of 3.124%. All remaining TT networks significantly underperformed (all $p < .05$) relative to baseline.

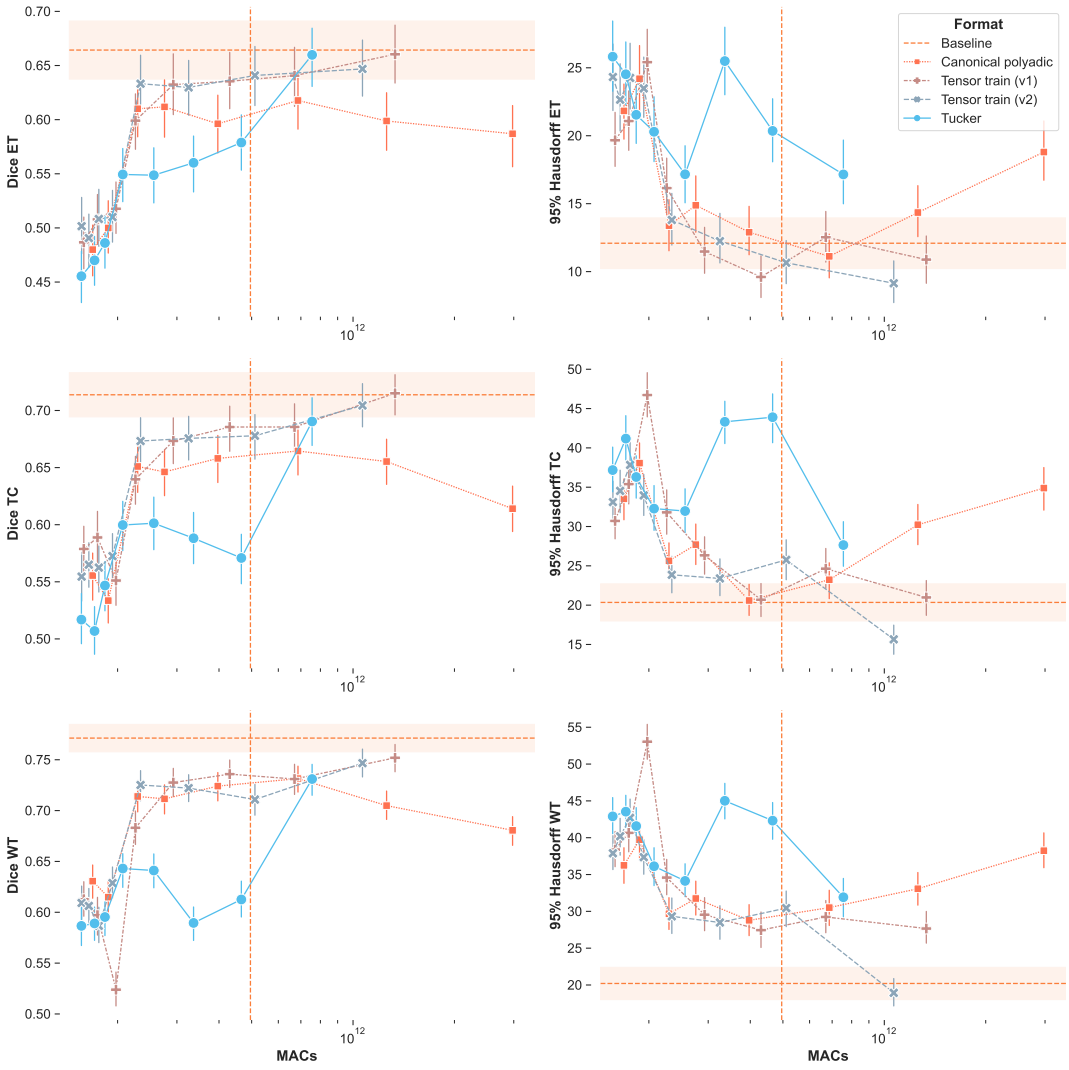


Figure 5.5: **Performance metrics versus MACs.** Dice scores and 95 % Hausdorff distances or all subregions of the glioma, against network MAC totals. The horizontal dashed orange line and its encompassing bar represent the performance of the baseline network, with its 95% confidence interval. The vertical dashed orange line represents the baseline network's MAC count. The markers represent the performances obtained by different tensorized networks, for varying degrees of network compression. Whiskers again indicate 95% confidence intervals. X-axes are logarithmically scaled. Performances towards to top left (Dice score) or bottom left (Hausdorff distance) represent a better trade-off between performance and computational efficiency.

The trick, William Potter, is not minding that it hurts.

Peter O'Toole muses upon
the balance between
success and sacrifice.
Lawrence of Arabia (1962)

Chapter 6

Discussion

Interpreting medical imaging data is a labor-intensive process that can only be performed by a scarce number of radiology experts. Semantic segmentation networks can be used support this diagnostic process. Nevertheless, such approaches come with problems of their own. Typically, CNNs are marked by both a large number of network parameters, and a significant computational demand. The latter issue, in particular, may complicate the use of such algorithms in real-time, depending on the setting in which they are deployed.

In this work, we explored a potential avenue to tackle these problems: the use of tensor network formats to reparametrize the convolutional layer's weight tensor. We evaluated four potential alternative network formats across eight different compression rates, relative to a baseline network.

In the next sections, the characteristics and performances of these tensorized networks will be discussed within the context of medical imaging. This has implications for the interpretation of the results. In a medical setting, it is reasonable to assume that any significant loss in performance is not acceptable. In contrast, this criterion may be relaxed in other contexts, depending on the primary focus of the application (e.g., using mobile platforms to perform simple classification tasks).

6.1 Performance

Baseline

While inspired by the nnU-Net architecture described in Isensee et al. [35], the performance of our baseline network was lower than that of said network. A direct comparison is not possible, since our test scheme was different from theirs (5 fold cross-validation on BraTS training set, vs. training on BraTS training set and testing on BraTS validation set). However, our implementation was also different in several ways, many of which could potentially have contributed to this difference in performance. For instance, our network was trained with a batch size of 1 due to memory constraints. Higher batch sizes can increase model performance [35]. Further, we did not apply any sort of post-processing strategy (e.g., thresholding ET regions), nor did our architecture make use of deep supervision, which can further aid the training process. None of these differences are particularly

relevant to the present work however, as the goal was to create an adequate baseline model to which other networks could be compared—a goal that was achieved.

Tensorized network performances

As expected, the total number of network parameters in all TNNs was significantly reduced compared to the baseline. As we only substituted convolutional layers in the double convolution blocks, which do not account for the entirety of trainable network parameters, the layer compression rate did not linearly correspond to the degree of network compression. Nonetheless, tensorization of the baseline network clearly reduced the strain on memory.

Computational demand, on the other hand, was only reduced from a certain level of layer compression (CP: layer compression rate 20, other formats: layer compression rate 10). In fact, low compression rates had an adverse effect on MAC counts, even causing a sixfold increase in computational strain for the CP-2 format. In that sense, the CP clearly underperformed. This is somewhat expected: the CP format offers the least flexibility in terms of finding an optimal contraction path, leaving less room for computational improvement.

Our data clearly indicates that, for our approach to have computational benefits, a minimal layer compression rate must be observed. However, with the exception of a few TNNs with small compression rates, all models yielded a significantly worse performance. More specifically, the Tucker and TT formats reached equal-to-baseline Dice scores at a factor 2 layer compression rate. The latter format, interestingly, accomplished baseline performance on the Hausdorff distance metric at a higher compression rate, with TT (v1) performing well up to a layer compression rate of 20. This result can likely be attributed to the regularization imposed by the low-rank constraints: steering the model away from overfitting (e.g., to odd, irregular training instances), towards more generalizable predictions, invites smoother contours in the predicted segmentation maps.

Foregoing statistical significance, results indicate that the TT format shows most promise for higher compression rates, particularly when all bond dimensions are set to a single tunable paramater (version 1). For these networks, Dice scores and 95% Hausdorff distances only dramatically worsened beyond a compression rate of 20. These networks' performances, moreover, are near-baseline for the regions that are most difficult to predict (i.e., the ET and TC). This is particularly important given the medical significance of these regions, which represent the most dangerous (i.e., life-threatening) sections in a glioma. Not only are TT networks able to at the very least approach baseline performance in a number of instances, they also lead to a sizeable reduction in computational demand (up to 52% reduction in MACs for a layer compression rate of 20).

Interestingly, these results are at odds with earlier findings (see section 3.3). Tai et al. [80] reported no significant loss in performance while gaining significant speed-up ($\times 1-2$), and a clear reduction in network parameters (up to 80% less). Still, all networks described in the study were trained for the purpose of classification—their performances are not directly comparable to our own.

A better point of reference is the study of Ashtari et al. [2], to which the present work is indebted. Here, the authors again found no apparent performance decrease when

training low-rank constrained networks using a CP format. While their study does not report statistical testing, in comparison, our own CP results indicate a distinctly outspoken decrease in average Dice scores, and mixed results for the Hausdorff distance. The training procedure they employed was different in several ways (e.g., use of residual blocks, fewer training epochs, adaptive learning rate schedule, post-processing strategy), though it is unclear which of these may have contributed to our diverging findings.

6.2 Implications

In the context of semantic segmentation for medical imaging, these results do not support our approach. Given the clinical importance of the predictions made by such a segmentation model, room for error is by default extremely narrow. This is particularly true for segmentation of hostile tissue in the brain, our most complex organ. Misidentifying healthy tissue as part of tumor, or conversely, failing to detect the entire tumor, is no trivial matter, particularly with a potential surgical intervention in mind.

Indeed, from this perspective, any deterioration in performance that is brought about by our manipulations is highly undesirable. However, CNNs are not deployed solely in performance-sensitive medical settings. Imagine a use case where one wishes to deploy a classification network on a mobile device with a limited computational capacity (e.g., [31, 43]). When attempting to move away from cloud-based solutions, the model's computational demand may conflict with any hope to use it in a near real-time manner. Under such conditions, it may be acceptable to tensorize the network using a TT format, yielding some performance in exchange for a 50% decrease in computational effort.

6.3 Limitations

In this thesis, we evaluated three of the most common tensor network types: the CP format, TT format and Tucker format. There are others, such as MERA, PEPS, and Hierarchical Tucker [11] (fig. 6.1), which we did not explore. Some successes were reported with the former network types, although studies were limited to the decomposition of fully connected layers [85, 25]. Interestingly, Wu and He [85] found that depending on the layer type (i.e., fully connected versus convolutional), different tensor network formats achieved better results. Extrapolating this finding, one may wonder: do different tensor network formats better accommodate the compression of convolutional weight tensors, depending on their shape? It is possible that, for instance, the initial convolutional layer (with few input and output channels) benefits from a different approach when compared to a bridge section convolutional layer (with hundreds of input and output channels). Similarly, the dimensionality of a weight tensor may have implications for the maximally attainable compression rate. Answering these questions, however, requires additional data.

Apart from the characteristics, hyperparameters and performances of our TNNs, there is another, practical consideration to be made when evaluating their real-world potential. As discussed in section 4.5, two separate sets of networks had to be built in order to get the desired metrics: a first set to calculate each network's associated MAC and

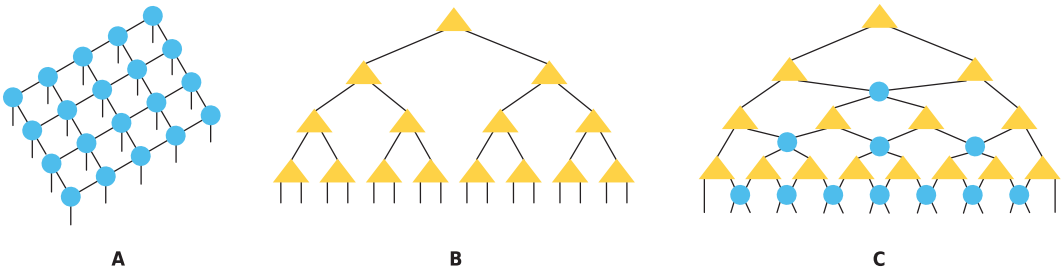


Figure 6.1: **Alternative tensor network formats.** A. Projected Entangled Pair States (PEPS). B. Hierarchical Tucker. C. Multiscale Entanglement Renormalization Ansatz (MERA).

parameters counts, and a second to train the reparametrized networks. The second set was created due to memory constraints caused by the unfolding of the input tensor, which hindered our efforts to train the tensorized networks. Thus, in order for our approach to have any merit in reality, an optimized, lower-level implementation would have to be programmed. Moreover, such an implementation would necessarily be more 'convoluted', as it should account for different compression rates and specific tensor network formats, each adding an extra set of hyperparameters to account for during training. Until such implementation is realized, any advantage due to the use of tensor network formats is by and large theoretical.

6.4 Conclusion

Low-rank convolutional layers can be helpful in reducing the space taken up in memory by a convolutional neural network. For small compression rates, tensor network formats help to reduce the number of network parameters noticeably while maintaining model performance. Simultaneously lowering the computational effort required for inference, however, is a more delicate affair: in order to achieve lower MAC counts—as opposed to actually *increasing* them in some cases—one must be willing to sacrifice predictive performance. The Tensor Train format, in that regard, yields the best trade-off curve.

Overall, the use of these lightweight layers depends on the priorities laid out by the context. While a medical setting can be expected to heavily emphasize the accuracy of the segmentation, networks hosted on mobile platforms may be better suited to trading performance for computational feasibility. In sum, advantages can be attained using the lightweight layers set out in this thesis, both in terms of memory and computational effort, though doing so comes at a cost.

Future research may focus on the interaction between different weight tensor sizes and the tensor network format chosen for reparametrization. Finally, for this approach to be useful in real-world applications, further work is needed to optimize the implementation of the lightweight layers described in this study.

Appendices

Appendix A

Supplementary tables

Table A.1: Model characteristics for baseline and tensorized networks

Baseline	**Layer compression**	**Format**			
		CP	*TT (v1)*	*TT (v2)*	*Tucker*
		Network parameters			
2.833+07	2	1.673E+07	1.672E+07	1.673E+07	1.674E+07
	5	9.765E+06	9.768E+06	9.766E+06	9.769E+06
	10	7.446E+06	7.442E+06	7.445E+06	7.446E+06
	20	6.285E+06	6.284E+06	6.286E+06	6.289E+06
	35	5.789E+06	5.792E+06	5.788E+06	5.794E+06
	50	5.590E+06	5.591E+06	5.590E+06	5.591E+06
	75	5.434E+06	5.436E+06	5.435E+06	5.437E+06
	100	5.359E+06	5.356E+06	5.357E+06	5.355E+06
		Network compression			
/	2	1.694	1.694	1.692	1.692
	5	2.900	2.900	2.900	2.900
	10	3.806	3.805	3.804	3.804
	20	4.508	4.506	4.504	4.504
	35	4.890	4.894	4.889	4.889
	50	5.067	5.068	5.067	5.067
	75	5.210	5.212	5.210	5.210
	100	5.289	5.288	5.289	5.289
		MACs			
4.959E+11	2	2.976E+12	1.333E+12	1.067E+12	6.981E+11
	5	1.258E+12	6.703E+11	5.114E+11	4.306E+11
	10	6.863E+11	4.304E+11	3.251E+11	3.121E+11
	20	3.968E+11	2.927E+11	2.340E+11	2.398E+11
	35	2.756E+11	2.260E+11	1.931E+11	1.960E+11
	50	2.294E+11	1.978E+11	1.759E+11	1.788E+11
	75	1.876E+11	1.743E+11	1.643E+11	1.632E+11
	100	1.687E+11	1.588E+11	1.564E+11	1.504E+11
		MAC Δ (%)			
/	2	-500.147	-168.741	-115.109	-52.445
	5	-153.638	-35.184	-3.124	6.018
	10	-38.407	13.197	34.446	32.209
	20	19.979	40.975	52.810	48.338
	35	44.414	54.433	61.049	58.214
	50	53.728	60.109	64.524	63.010
	75	62.163	64.850	66.862	65.549
	100	65.986	67.984	68.457	68.506

Table A.2: Performance metrics for tensorized networks (5-fold cross-validation)

	Compression		**Dice score**						**95% Hausdorff distance**					
	Layer	Network	ET		TC		WT		ET		TC		WT	
Format			M	SD	M	SD	M	SD	M	SD	M	SD	M	SD
CP	2	1.694	.587	.263	.614	.194	.681	.141	18.801	21.090	34.890	26.412	38.235	23.899
	5	2.901	.599	.262	.655	.192	.705	.131	14.347	18.995	30.226	24.879	33.070	22.231
	10	3.804	.618	.264	.665	.197	.731	.128	11.125	16.287	23.228	22.048	30.508	22.048
	20	4.507	.596	.256	.658	.194	.724	.135	12.898	17.182	20.593	19.838	28.785	20.670
	35	4.893	.612	.260	.646	.194	.712	.138	14.882	19.895	27.696	24.811	31.745	22.809
	50	5.067	.610	.263	.651	.202	.714	.144	13.384	18.587	25.636	23.958	29.672	22.414
	75	5.212	.500	.239	.534	.201	.615	.162	24.194	22.711	38.067	25.441	39.733	23.372
	100	5.286	.480	.240	.555	.196	.631	.162	21.827	22.045	33.522	25.534	36.247	23.423
TT (v1)	2	1.694	.660	.262	.715	.181	.752	.133	10.884	17.357	20.981	21.955	27.666	21.474
	5	2.900	.641	.267	.686	.204	.731	.148	12.531	17.565	24.637	23.351	29.240	21.910
	10	3.806	.635	.263	.686	.191	.736	.133	9.605	15.088	20.685	21.080	27.452	24.344
	20	4.508	.632	.263	.673	.202	.728	.142	11.477	16.458	26.336	23.868	29.542	21.855
	35	4.890	.599	.255	.640	.205	.683	.163	16.147	20.400	31.799	26.576	34.585	23.766
	50	5.067	.518	.243	.551	.211	.524	.166	25.415	22.603	46.721	26.831	53.037	22.633
	75	5.210	.508	.240	.589	.216	.597	.172	21.081	21.637	35.393	25.799	40.655	24.396
	100	5.289	.487	.234	.579	.213	.613	.166	19.667	20.224	30.710	23.516	38.385	23.914
TT (v2)	2	1.694	.647	.259	.704	.187	.747	.132	9.142	14.521	15.653	19.012	18.932	18.229
	5	2.900	.641	.262	.678	.199	.711	.154	10.651	16.080	25.759	25.682	30.446	24.541
	10	3.805	.630	.260	.676	.195	.722	.140	12.243	17.404	23.396	22.850	28.487	22.120
	20	4.506	.633	.260	.673	.203	.725	.144	13.799	19.329	23.859	23.647	29.325	22.871
	35	4.894	.510	.246	.573	.206	.629	.159	23.482	22.380	33.958	24.713	37.359	23.330
	50	5.068	.508	.247	.562	.203	.587	.170	24.255	24.054	37.834	27.030	42.698	23.907
	75	5.212	.491	.234	.565	.199	.606	.165	22.650	23.655	34.532	25.981	40.201	25.174
	100	5.288	.502	.242	.555	.209	.609	.161	24.322	23.580	33.111	24.673	37.891	23.481
Tucker	2	1.692	.660	.264	.690	.209	.731	.150	17.153	22.415	27.643	27.840	31.916	24.760
	5	2.899	.579	.253	.571	.217	.613	.171	20.355	23.124	43.903	30.056	42.319	25.369
	10	3.804	.560	.252	.588	.214	.590	.167	25.495	24.947	43.316	27.890	45.013	23.556
	20	4.503	.549	.251	.601	.222	.641	.162	17.164	20.680	31.969	27.503	34.132	23.132
	35	4.889	.549	.253	.600	.220	.643	.163	20.292	23.153	32.277	28.338	36.124	25.417
	50	5.066	.486	.240	.547	.203	.595	.171	21.541	21.734	36.298	26.526	41.580	23.475
	75	5.209	.470	.237	.507	.207	.589	.172	24.525	24.290	41.183	27.916	43.537	23.247
	100	5.289	.455	.236	.517	.219	.587	.181	25.818	25.292	37.181	27.392	42.899	24.530

Table A.3: Statistical comparisons of performance between tensorized networks and the baseline (Mann-Whitney U tests)

Format	Compression		MAC Δ (%)	Dice score comparison						95% Hausdorff distance comparison					
	Layer	*Network*		ET		TC		WT		ET		TC		WT	
				U	*p*	*U*	*p*	*U*	*p*	*U*	*p*	*U*	*p*	*U*	*p*
CP	2	1.694	-500.147	50844	.000	44854	.000	40392	.000	47967	.000	38014	.000	33175	.000
	5	2.901	-153.638	52675	.000	53222	.000	45309	.000	56022	.000	45409	.000	39240	.000
	10	3.804	-38.407	57159	.000	56287	.000	53175	.000	**64956**	**0.140**	56863	.000	43283	.000
	20	4.507	19.979	51674	.000	54453	.000	51597	.000	58183	.000	61204	.009	44083	.000
	35	4.893	44.414	55738	.000	51932	.000	48679	.000	56180	.000	50100	.000	42172	.000
	50	5.067	53.728	55840	.000	54188	.000	50096	.000	58992	.001	53903	.000	45210	.000
	75	5.212	62.163	35167	.000	31235	.000	27677	.000	37498	.000	33769	.000	30583	.000
	100	5.286	65.986	33202	.000	34262	.000	30643	.000	39293	.000	39050	.000	34722	.000
TT (v1)	2	1.694	-168.741	**66811**	**0.331**	**67357**	**0.401**	61350	.010	**65449**	**0.182**	**64284**	**0.095**	49326	.000
	5	2.900	-35.184	63184	.045	62738	.033	56323	.000	**64833**	**0.131**	57290	.000	46559	.000
	10	3.806	13.197	60786	.006	61082	.008	55446	.000	**66255**	**0.264**	**62741**	**0.033**	51571	.000
	20	4.508	40.975	60595	.005	59362	.001	54485	.000	**65248**	**0.164**	54185	0.000	45035	.000
	35	4.890	54.433	52199	.000	51903	.000	44590	.000	53824	.000	44688	.000	39297	.000
	50	5.067	60.109	37566	.000	35282	.000	16129	.000	36004	.000	26618	.000	18430	.000
	75	5.210	64.850	36455	.000	42245	.000	26488	.000	41756	.000	38437	.000	30910	.000
	100	5.289	67.984	33171	.000	39789	.000	28320	.000	42288	.000	42031	.000	33254	.000
TT (v2)	2	1.694	-115.109	63217	.047	**65601**	**0.196**	59152	.001	**63539**	**0.058**	**60157**	**0.003**	**65562**	**0.192**
	5	2.900	-3.124	62107	.020	60221	.003	50474	.000	**67349**	**0.400**	58557	.001	46386	.000
	10	3.805	34.446	59405	.001	59083	.001	52165	.000	62454	.026	58571	.001	47424	.000
	20	4.506	52.810	60076	.003	59498	.002	53863	.000	59795	.002	57885	.000	46547	.000
	35	4.894	61.049	37376	.000	38471	.000	30432	.000	38791	.000	38408	.000	33749	.000
	50	5.068	64.524	37375	.000	36070	.000	24035	.000	39098	.000	35255	.000	27739	.000
	75	5.212	66.862	33682	.000	36014	.000	26991	.000	40448	.000	38225	.000	32023	.000
	100	5.288	68.457	35722	.000	34933	.000	26188	.000	37702	.000	38403	.000	32277	.000
Tucker	2	1.692	-52.445	**67338**	**0.399**	**65118**	**0.153**	57098	.000	59293	.001	58501	.000	45534	.000
	5	2.899	6.018	48380	.000	40000	.000	29243	.000	47463	.000	33918	.000	31012	.000
	10	3.804	32.209	45275	.000	42271	.000	24452	.000	38441	.000	30754	.000	25630	.000
	20	4.503	48.338	43174	.000	45591	.000	33585	.000	50247	.000	47332	.000	37769	.000
	35	4.889	58.214	43384	.000	45170	.000	34033	.000	45499	.000	46085	.000	37157	.000
	50	5.066	63.010	34002	.000	33671	.000	25397	.000	40946	.000	37500	.000	28750	.000
	75	5.209	65.549	31801	.000	28334	.000	24333	.000	37580	.000	32384	.000	26883	.000
	100	5.289	68.506	30330	.000	30555	.000	25023	.000	37398	.000	36476	.000	28670	.000

Statistical comparisons to baseline at $\alpha = 0.5$ level: underperformance – **equal performance** – **overperformance**

Bibliography

[1] Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. *31st International Conference on Machine Learning, ICML 2014*, 1:883–891, 2014.

[2] Pooya Ashtari, Frederik Maes, and Sabine Van Huffel. Low-rank convolutional networks for brain tumor segmentation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12658 LNCS:470–480, 2021. ISSN 16113349. doi: 10.1007/978-3-030-72084-1_42.

[3] Peter C. Austin and Janet E. Hux. A brief note on overlapping confidence intervals. *Journal of Vascular Surgery*, 36(1):194–195, 2002. ISSN 07415214. doi: 10.1067/mva.2002.125015.

[4] Mingyuan Bai, S. T.Boris Choy, Xin Song, and Junbin Gao. Tensor-train parameterization for ultra dimensionality reduction. *Proceedings - 10th IEEE International Conference on Big Knowledge, ICBK 2019*, pages 17–24, 2019. doi: 10.1109/ICBK.2019.00011.

[5] Spyridon Bakas, Hamed Akbari, Aristeidis Sotiras, Michel Bilello, Martin Rozycki, Justin S. Kirby, John B. Freymann, Keyvan Farahani, and Christos Davatzikos. Advancing the cancer genome atlas glioma mri collections with expert segmentation labels and radiomic features. *Scientific Data*, 4(March):1–13, 2017. ISSN 20524463. doi: 10.1038/sdata.2017.117. URL http://dx.doi.org/10.1038/sdata.2017.117.

[6] Spyridon Bakas, Mauricio Reyes, Andras Jakab, Stefan Bauer, Markus Rempfler, Alessandro Crimi, Russell Takeshi Shinohara, Christoph Berger, Sung Min Ha, Martin Rozycki, Marcel Prastawa, Esther Alberts, Jana Lipkova, John Freymann, Justin Kirby, Michel Bilello, Hassan M. Fathallah-Shaykh, Roland Wiest, Jan Kirschke, Benedikt Wiestler, Rivka Colen, Aikaterini Kotrotsou, Pamela Lamontagne, Daniel Marcus, Mikhail Milchenko, Arash Nazeri, Marc Andr Weber, Abhishek Mahajan, Ujjwal Baid, Elizabeth Gerstner, Dongjin Kwon, Gagan Acharya, Manu Agarwal, Mahbubul Alam, Alberto Albiol, Antonio Albiol, Francisco J. Albiol, Varghese Alex, Nigel Allinson, Pedro H.A. Amorim, Abhijit Amrutkar, Ganesh Anand, Simon Andermatt, Tal Arbel, Pablo Arbelaez, Aaron Avery, Muneeza Azmat, B. Pranjal, Wenjia Bai, Subhashis Banerjee, Bill Barth, Thomas Batchelder, Kayhan Batmanghelich, Enzo Battistella, Andrew Beers, Mikhail Belyaev, Martin Bendszus, Eze Benson, Jose Bernal, Halandur Nagaraja Bharath, George Biros, Sotirios Bisdas, James Brown, Mariano Cabezas, Shilei Cao, Jorge M. Cardoso, Eric N. Carver, Adri Casamitjana, Laura Silvana Castillo,

Marcel Cat, Philippe Cattin, Albert Cérigues, Vinicius S. Chagas, Siddhartha Chandra, Yi Ju Chang, Shiyu Chang, Ken Chang, Joseph Chazalon, Shengcong Chen, Wei Chen, Jefferson W. Chen, Zhaolin Chen, Kun Cheng, Ahana Roy Choudhury, Roger Chylla, Albert Clrigues, Steven Colleman, Ramiro German Rodriguez Colmeiro, Marc Combalia, Anthony Costa, Xiaomeng Cui, Zhenzhen Dai, Lutao Dai, Laura Alexandra Daza, Eric Deutsch, Changxing Ding, Chao Dong, Shidu Dong, Wojciech Dudzik, Zach Eaton-Rosen, Gary Egan, Guilherme Escudero, Tho Estienne, Richard Everson, Jonathan Fabrizio, Yong Fan, Longwei Fang, Xue Feng, Enzo Ferrante, Lucas Fidon, Martin Fischer, Andrew P. French, Naomi Fridman, Huan Fu, David Fuentes, Yaozong Gao, Evan Gates, David Gering, Amir Gholami, Willi Gierke, Ben Glocker, Mingming Gong, Sandra Gonzlez-Vill, T. Grosges, Yuanfang Guan, Sheng Guo, Sudeep Gupta, Woo Sup Han, Il Song Han, Konstantin Harmuth, Huiguang He, Aura Hernndez-Sabat, Evelyn Herrmann, Naveen Himthani, Winston Hsu, Cheyu Hsu, Xiaojun Hu, Xiaobin Hu, Yan Hu, Yifan Hu, Rui Hua, Teng Yi Huang, Weilin Huang, Sabine van Huffel, Quan Huo, H. V. Vivek, Khan M. Iftekharuddin, Fabian Isensee, Mobarakol Islam, Aaron S. Jackson, Sachin R. Jambawalikar, Andrew Jesson, Weijian Jian, Peter Jin, V. Jeya Maria Jose, Alain Jungo, Bernhard Kainz, Konstantinos Kamnitsas, Po Yu Kao, Ayush Karnawat, Thomas Kellermeier, Adel Kermi, Kurt Keutzer, Mohamed Tarek Khadir, Mahendra Khened, Philipp Kickingereder, Geena Kim, Nik King, Haley Knapp, Urspeter Knecht, Lisa Kohli, Deren Kong, Xiangmao Kong, Simon Koppers, Avinash Kori, Ganapathy Krishnamurthi, Egor Krivov, Piyush Kumar, Kaisar Kushibar, Dmitrii Lachinov, Tryphon Lambrou, Joon Lee, Chengen Lee, Yuehchou Lee, Matthew Chung Hai Lee, Szidonia Lefkovits, Laszlo Lefkovits, James Levitt, Tengfei Li, Hongwei Li, Wenqi Li, Hongyang Li, Xiaochuan Li, Yuexiang Li, Heng Li, Zhenye Li, Xiaoyu Li, Zeju Li, Xiao Gang Li, Wenqi Li, Zheng Shen Lin, Fengming Lin, Pietro Lio, Chang Liu, Boqiang Liu, Xiang Liu, Mingyuan Liu, Ju Liu, Luyan Liu, Xavier Lladó, Marc Moreno Lopez, Pablo Ribalta Lorenzo, Zhentai Lu, Lin Luo, Zhigang Luo, Jun Ma, Kai Ma, Thomas Mackie, Anant Madabhushi, Issam Mahmoudi, Klaus H. Maier-Hein, Pradipta Maji, C. P. Mammen, Andreas Mang, B. S. Manjunath, Michal Marcinkiewicz, Steven McDonagh, Stephen McKenna, Richard McKinley, Miriam Mehl, Sachin Mehta, Raghav Mehta, Raphael Meier, Christoph Meinel, Dorit Merhof, Craig Meyer, Robert Miller, Sushmita Mitra, Aliasgar Moiyadi, David Molina-Garcia, Miguel A.B. Monteiro, Grzegorz Mrukwa, Andriy Myronenko, Jakub Nalepa, Thuyen Ngo, Dong Nie, Holly Ning, Chen Niu, Nicholas K. Nuechterlein, Eric Oermann, Arlindo Oliveira, Diego D.C. Oliveira, Arnau Oliver, Alexander F.I. Osman, Yu Nian Ou, Sebastien Ourselin, Nikos Paragios, Moo Sung Park, Brad Paschke, J. Gregory Pauloski, Kamlesh Pawar, Nick Pawlowski, Linmin Pei, Suting Peng, Silvio M. Pereira, Julian Perez-Beteta, Victor M. Perez-Garcia, Simon Pezold, Bao Pham, Ashish Phophalia, Gemma Piella, G. N. Pillai, Marie Piraud, Maxim Pisov, Anmol Popli, Michael P. Pound, Reza Pourreza, Prateek Prasanna, Vesnakovska Pr, Tony P. Pridmore, Santi Puch, Lodie Puybareau, Buyue Qian, Xu Qiao, Martin Rajchl, Swapnil Rane, Michael Rebsamen, Hongliang Ren, Xuhua Ren, Karthik Revanuru, Mina Rezaei, Oliver Rippel, Luis Carlos Rivera, Charlotte Robert, Bruce Rosen, Daniel Rueckert, Mohammed Safwan, Mostafa Salem, Joaquim Salvi, Irina Sanchez, Irina Snchez, Heitor M. Santos, Emmett Sartor, Dawid Schellingerhout, Klaudius Scheufele, Matthew R. Scott,

Artur A. Scussel, Sara Sedlar, Juan Pablo Serrano-Rubio, N. Jon Shah, Nameetha Shah, Mazhar Shaikh, B. Uma Shankar, Zeina Shboul, Haipeng Shen, Dinggang Shen, Linlin Shen, Haocheng Shen, Varun Shenoy, Feng Shi, Hyung Eun Shin, Hai Shu, Diana Sima, Matthew Sinclair, Orjan Smedby, James M. Snyder, Mohammadreza Soltaninejad, Guidong Song, Mehul Soni, Jean Stawiaski, Shashank Subramanian, Li Sun, Roger Sun, Jiawei Sun, Kay Sun, Yu Sun, Guoxia Sun, Shuang Sun, Yannick R. Suter, Laszlo Szilagyi, Sanjay Talbar, Dacheng Tao, Dacheng Tao, Zhongzhao Teng, Siddhesh Thakur, Meenakshi H. Thakur, Sameer Tharakan, Pallavi Tiwari, Guillaume Tochon, Tuan Tran, Yuhsiang M. Tsai, Kuan Lun Tseng, Tran Anh Tuan, Vadim Turlapov, Nicholas Tustison, Maria Vakalopoulou, Sergi Valverde, Rami Vanguri, Evgeny Vasiliev, Jonathan Ventura, Luis Vera, Tom Vercauteren, C. A. Verrastro, Lasitha Vidyaratne, Veronica Vilaplana, Ajeet Vivekanandan, Guotai Wang, Qian Wang, Chiatse J. Wang, Weichung Wang, Duo Wang, Ruixuan Wang, Yuanyuan Wang, Chunliang Wang, Guotai Wang, Ning Wen, Xin Wen, Leon Weninger, Wolfgang Wick, Shaocheng Wu, Qiang Wu, Yihong Wu, Yong Xia, Yanwu Xu, Xiaowen Xu, Peiyuan Xu, Tsai Ling Yang, Xiaoping Yang, Hao Yu Yang, Junlin Yang, Haojin Yang, Guang Yang, Hongdou Yao, Xujiong Ye, Changchang Yin, Brett Young-Moxon, Jinhua Yu, Xiangyu Yue, Songtao Zhang, Angela Zhang, Kun Zhang, Xuejie Zhang, Lichi Zhang, Xiaoyue Zhang, Yazhuo Zhang, Lei Zhang, Jianguo Zhang, Xiang Zhang, Tianhao Zhang, Sicheng Zhao, Yu Zhao, Xiaomei Zhao, Liang Zhao, Yefeng Zheng, Liming Zhong, Chenhong Zhou, Xiaobing Zhou, Fan Zhou, Hongtu Zhu, Jin Zhu, Ying Zhuge, Weiwei Zong, Jayashree Kalpathy-Cramer, Keyvan Farahani, Christos Davatzikos, Koen van Leemput, and Bjoern Menze. Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall survival prediction in the brats challenge. *arXiv*, 2018. ISSN 23318422.

[7] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 089791497X. doi: 10.1145/130385.130401. URL https://doi.org/10.1145/130385.130401.

[8] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, nov 1986. ISSN 0162-8828. doi: 10.1109/TPAMI.1986.4767851. URL https://ieeexplore.ieee.org/document/4767851.

[9] Ignacio Arganda Carreras, Srinivas C. Turaga, Daniel R. Berger, Dan Cire San, Alessandro Giusti, Luca M. Gambardella, Jürgen Schmidhuber, Dmitry Laptev, Sarvesh Dwivedi, Joachim M. Buhmann, Ting Liu, Mojtaba Seyedhosseini, Tolga Tasdizen, Lee Kamentsky, Radim Burget, Vaclav Uher, Xiao Tan, Changming Sun, Tuan D. Pham, Erhan Bas, Mustafa G. Uzunbas, Albert Cardona, Johannes Schindelin, and H. Sebastian Seung. Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in Neuroanatomy*, 9(November):1–13, 2015. ISSN 16625129. doi: 10.3389/fnana.2015.00142.

[10] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: Learning dense volumetric segmentation from sparse annotation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9901 LNCS:424–432, 2016. ISSN 16113349. doi: 10.1007/978-3-319-46723-8_49.

[11] Andrzej Cichocki. Tensor networks for dimensionality reduction, big data and deep learning. In *Studies in Computational Intelligence*, volume 738, pages 3–49. 2018. ISBN 9783319679457. doi: 10.1007/978-3-319-67946-4_1. URL http://link.springer.com/10.1007/978-3-319-67946-4%5F1.

[12] Andrzej Cichocki, A-H. Phan, Qibin Zhao, Namgil Lee, I. V. Oseledets, Masashi Sugiyama, and Danilo Mandic. Tensor networks for dimensionality reduction and large-scale optimizations. part 2 applications and future perspectives. *Foundations and Trends in Machine Learning*, 9(6):431–673, aug 2017. ISSN 19358245. doi: 10.1561/2200000067. URL http://arxiv.org/abs/1708.09165http://dx.doi.org/10.1561/2200000067.

[13] Dan C. Cireşan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. *IJCAI International Joint Conference on Artificial Intelligence*, (June 2014):1237–1242, 2011. ISSN 10450823. doi: 10.5591/978-1-57735-516-8/IJCAI11-210.

[14] Dan C. Cireşan, Alessandro Giusti, Luca M. Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pages 2843–2851, Red Hook, NY, USA, 2012. Curran Associates Inc.

[15] Karl Cuddy, Eric J. Dierks, Allen Cheng, Ashish Patel, Melissa Amundson, and R. Bryan Bell. Management of zygomaticomaxillary complex fractures utilizing intraoperative 3-dimensional imaging: The zygomas protocol. *Journal of Oral and Maxillofacial Surgery*, 79(1):177–182, 2021. ISSN 15315053. doi: 10.1016/j.joms.2020.08.028. URL https://doi.org/10.1016/j.joms.2020.08.028.

[16] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'aurelio Ranzato, and Nando De Freitas. Predicting parameters in deep learning. *Advances in Neural Information Processing Systems*, pages 1–9, 2013. ISSN 10495258.

[17] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *Advances in Neural Information Processing Systems*, 2(January):1269–1277, 2014. ISSN 10495258.

[18] Stuart Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45, aug 1962. ISSN 0022247X.

doi: 10.1016/0022-247X(62)90004-5. URL https://linkinghub.elsevier.com/retrieve/pii/0022247X62900045.

[19] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. pages 1–31, 2016. URL http://arxiv.org/abs/1603.07285.

[20] William A. Falcon. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

[21] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *7th International Conference on Learning Representations, ICLR 2019*, pages 1–42, 2019.

[22] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980. ISSN 03401200. doi: 10.1007/BF00344251.

[23] Dazhou Guo, Yanting Pei, Kang Zheng, Hongkai Yu, Yuhang Lu, and Song Wang. Degraded image semantic segmentation with dense-gram networks. *IEEE Transactions on Image Processing*, 29:782–795, 2020. ISSN 1057-7149. doi: 10.1109/TIP.2019.2936111. URL https://ieeexplore.ieee.org/document/8812903/.

[24] Manoj Kr Gupta and Pravin Chandra. A comparative study of clustering algorithms. *Proceedings of the 2019 6th International Conference on Computing for Sustainable Global Development, INDIACom 2019*, pages 801–805, 2019. ISSN 1942-9010. doi: 10.4018/jvcsn.2011070101.

[25] Andrew Hallam, Edward Grant, Vid Stojevic, Simone Severini, and Andrew G. Green. Compact neural networks based on the multiscale entanglement renormalization ansatz. *British Machine Vision Conference 2018, BMVC 2018*, (2014):1–9, 2019.

[26] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *Advances in Neural Information Processing Systems*, 2015-January:1135–1143, 2015. ISSN 10495258.

[27] Dennis Hartmann, Dominik Müller, Iñaki Soto-Rey, and Frank Kramer. Assessing the role of random forests in medical image segmentation. mar 2021. URL http://arxiv.org/abs/2103.16492.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 International Conference on Computer Vision, ICCV 2015:1026–1034, 2015. ISSN 15505499. doi: 10.1109/ICCV.2015.123.

[29] D. O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Wiley, New York, oct 1949. doi: 10.2307/1418888. URL https://www.jstor.org/stable/1418888?origin=crossref.

[30] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems*, 6(2):107–116, 1998. ISSN 02184885. doi: 10.1142/S0218488598000094.

[31] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. 2017. URL http://arxiv.org/abs/1704.04861.

[32] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015*, 1:448–456, 2015.

[33] Fabian Isensee, Philipp Kickingereder, Wolfgang Wick, Martin Bendszus, and Klaus H. Maier-Hein. No new-net. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11384 LNCS:234–244, 2019. ISSN 16113349. doi: 10.1007/978-3-030-11726-9_21.

[34] Fabian Isensee, Jens Petersen, Andre Klein, David Zimmerer, Paul F. Jaeger, Simon Kohl, Jakob Wasserthal, Gregor Koehler, Tobias Norajitra, Sebastian Wirkert, and Klaus H. Maier-Hein. nnu-net: Self-adapting framework for u-net-based medical image segmentation. *Informatik aktuell*, page 22, 2019. ISSN 1431472X. doi: 10.1007/978-3-658-25326-4_7.

[35] Fabian Isensee, Paul F. Jaeger, Peter M. Full, Philipp Vollmuth, and Klaus H. Maier-Hein. nnu-net for brain tumor segmentation. pages 1–15, 2020. URL http://arxiv.org/abs/2011.00848.

[36] A. Ivakhnenko, V. G. Lapa, and R. Mcdonough. Cybernetics and forecasting techniques. 1967.

[37] A. G. Ivakhnenko. Polynomial theory of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-1(4):364–378, oct 1971. ISSN 0018-9472. doi: 10.1109/TSMC.1971.4308320. URL http://ieeexplore.ieee.org/document/4308320/.

[38] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *BMVC 2014 - Proceedings of the British Machine Vision Conference 2014*, 2014. doi: 10.5244/c.28.88.

[39] Zeyu Jiang, Changxing Ding, Minfeng Liu, and Dacheng Tao. Two-stage cascaded u-net: 1st place solution to brats challenge 2019 segmentation task. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11992 LNCS(August):231–241, 2020. ISSN 16113349. doi: 10.1007/978-3-030-46640-4_22.

[40] K. Kamnitsas, W. Bai, E. Ferrante, S. McDonagh, M. Sinclair, N. Pawlowski, M. Rajchl, M. Lee, B. Kainz, D. Rueckert, and B. Glocker. Ensembles of multiple models and architectures for robust brain tumour segmentation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10670 LNCS:450–462, 2018. ISSN 16113349. doi: 10.1007/978-3-319-75238-9_38.

[41] Henry Kelley. Gradient theory of optimal flight paths. *ARS Journal*, 30(10):947–954, oct 1960. ISSN 1936-9972. doi: 10.2514/8.5282. URL https://arc.aiaa.org/doi/10.2514/8.5282.

[42] Brian W Kernighan and Dennis M Ritchie. *The C programming language*. 2006.

[43] Yong Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pages 1–16, 2016.

[44] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015.

[45] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, aug 2009. ISSN 0036-1445. doi: 10.1137/07070111X.

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, may 2017. ISSN 0001-0782. doi: 10.1145/3065386. URL https://dl.acm.org/doi/10.1145/3065386.

[47] Jiss Kuruvilla, Dhanya Sukumaran, Anjali Sankar, and Siji P Joy. A review on image processing and image segmentation. In *2016 International Conference on Data Mining and Advanced Computing (SAPIENCE)*, volume 1187, pages 198–203. IEEE, mar 2016. ISBN 978-1-4673-8594-7. doi: 10.1109/SAPIENCE.2016.7684170. URL http://ieeexplore.ieee.org/document/7684170/.

[48] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–11, 2015.

[49] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to digit recognition, 1989. URL https://www.ics.uci.edu/\protect\T1\textdollar\sim\protect\T1\textdollarwelling/teaching/273ASpring09/lecun-89e.pdf.

[50] Yann Lecun, Le'on Bottou, Yoshua Bengio, and Parick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86

(11):2278–2324, 1998. ISSN 00189219. doi: 10.1109/5.726791. URL http://ieeexplore.ieee.org/document/726791/#full-text-sectionhttp://ieeexplore.ieee.org/document/726791/.

[51] Chi Li, M. Zeeshan Zia, Quoc Huy Tran, Xiang Yu, Gregory D. Hager, and Manmohan Chandraker. Deep Supervision with Intermediate Concepts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):1828–1843, 2019. ISSN 19393539. doi: 10.1109/TPAMI.2018.2863285.

[52] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pages 1–16, 2017.

[53] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[54] David N. Louis, Arie Perry, Guido Reifenberger, Andreas von Deimling, Dominique Figarella-Branger, Webster K. Cavenee, Hiroko Ohgaki, Otmar D. Wiestler, Paul Kleihues, and David W. Ellison. The 2016 world health organization classification of tumors of the central nervous system: a summary. *Acta Neuropathologica*, 131(6): 803–820, 2016. ISSN 14320533. doi: 10.1007/s00401-016-1545-1.

[55] Frederik Maes, Dirk Vandermeulen, Paul Suetens, and Guy Marchal. Computer-aided interactive object delineation using an intelligent paintbrush technique. *Computer Vision, Virtual Reality and Robotics in Medicine*, pages 77–83, 2005. doi: 10.1007/bfb0034935.

[56] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1): 50–60, 1947. ISSN 0003-4851. doi: 10.1214/aoms/1177730491.

[57] Warren S Mcculloch and Walter Pitts. A logical calculus nervous activity. *Bulletin of Mathematical Biology*, 52(l):99–115, 1990. ISSN 00074985.

[58] Bjoern H. Menze, Andras Jakab, Stefan Bauer, Jayashree Kalpathy-Cramer, Keyvan Farahani, Justin Kirby, Yuliya Burren, Nicole Porz, Johannes Slotboom, Roland Wiest, Levente Lanczi, Elizabeth Gerstner, Marc André Weber, Tal Arbel, Brian B. Avants, Nicholas Ayache, Patricia Buendia, D. Louis Collins, Nicolas Cordier, Jason J. Corso, Antonio Criminisi, Tilak Das, Hervé Delingette, Çağatay Demiralp, Christopher R. Durst, Michel Dojat, Senan Doyle, Joana Festa, Florence Forbes, Ezequiel Geremia, Ben Glocker, Polina Golland, Xiaotao Guo, Andac Hamamci, Khan M. Iftekharuddin, Raj Jena, Nigel M. John, Ender Konukoglu, Danial Lashkari, José António Mariz, Raphael Meier, Sérgio Pereira, Doina Precup, Stephen J. Price, Tammy Riklin Raviv, Syed M.S. Reza, Michael Ryan, Duygu Sarikaya, Lawrence Schwartz, Hoo Chang Shin, Jamie Shotton, Carlos A. Silva, Nuno Sousa, Nagesh K. Subbanna, Gabor Szekely, Thomas J. Taylor, Owen M. Thomas, Nicholas J. Tustison, Gozde Unal, Flor Vasseur, Max Wintermark, Dong Hye Ye, Liang Zhao, Binsheng Zhao, Darko Zikic, Marcel Prastawa, Mauricio Reyes, and Koen Van Leemput. The multimodal brain tumor

image segmentation benchmark (brats). *IEEE Transactions on Medical Imaging*, 34 (10):1993–2024, 2015. ISSN 1558254X. doi: 10.1109/TMI.2014.2377694.

[59] Fausto Milletari, Nassir Navab, and Seyed-ahmad Ahmadi. V-net : Fully convolutional neural networks for. pages 1–11, 2016.

[60] R. F. Mould. The early history of x-ray diagnosis with emphasis on the contributions of physics 1895-1915. *Physics in Medicine and Biology*, 40(11):1741–1787, 1995. ISSN 00319155. doi: 10.1088/0031-9155/40/11/001.

[61] Henning Müller, Nicolas Michoux, David Bandon, and Antoine Geissbuhler. A review of content-based image retrieval systems in medical applications - clinical benefits and future directions. *International Journal of Medical Informatics*, 73(1):1–23, 2004. ISSN 13865056. doi: 10.1016/j.ijmedinf.2003.11.024.

[62] Shinichi Nakajima, Masashi Sugiyama, S. Derin Babacan, and Ryota Tomioka. Global analytic solution of fully-observed variational bayesian matrix factorization. *Journal of Machine Learning Research*, 14(1):1–37, 2013. ISSN 15324435.

[63] Brady Neal, Sarthak Mittal, Aristide Baratin, Vinayak Tantia, Matthew Scicluna, Simon Lacoste-Julien, and Ioannis Mitliagkas. A modern take on the bias-variance tradeoff in neural networks. 2018. URL http://arxiv.org/abs/1810.08591.

[64] Behnam Neyshabur, Ryota Tomioka, Ruslan Salakhutdinov, and Nathan Srebro. Geometry of optimization and implicit regularization in deep learning. pages 1–13, 2017. URL http://arxiv.org/abs/1705.03071.

[65] Richard Nock and Frank Nielsen. Statistical region merging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1452–1458, 2004. ISSN 01628828. doi: 10.1109/TPAMI.2004.110.

[66] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing neural networks. *Advances in Neural Information Processing Systems*, 2015-Janua: 442–450, sep 2015. ISSN 10495258. URL http://arxiv.org/abs/1509.06569.

[67] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[68] George Philipp, Dawn Song, and Jaime G. Carbonell. The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions. (2014), 2017. URL http://arxiv.org/abs/1712.05577.

[69] David C Preston. Magnetic resonance imaging (mri) of the brain and spine: Basics, Apr 2016. URL https://case.edu/med/neurology/NR/MRI%20Basics.htm.

[70] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9351:234–241, 2015. ISSN 16113349. doi: 10.1007/978-3-319-24574-4_28.

[71] Kenneth J. Rothman. No adjustments are needed for multiple comparisons. *Epidemiology*, 1(1):43–46, jan 1990. ISSN 1044-3983. doi: 10.1097/00001648-199001000-00010. URL http://journals.lww.com/00001648-199001000-00010.

[72] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, oct 1986. ISSN 0028-0836. doi: 10.1038/323533a0. URL http://www.nature.com/articles/323533a0.

[73] James H. Scatliff and Peter J. Morris. From roentgen to magnetic resonance imaging: the history of medical imaging. *North Carolina medical journal*, 75(2):111–113, 2014. ISSN 00292559. doi: 10.18043/ncm.75.2.111.

[74] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, apr 2017. ISSN 0162-8828. doi: 10.1109/TPAMI.2016.2572683. URL http://ieeexplore.ieee.org/document/7478072/.

[75] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–14, 2015.

[76] Amber L. Simpson, Michela Antonelli, Spyridon Bakas, Michel Bilello, Keyvan Farahani, Bram van Ginneken, Annette Kopp-Schneider, Bennett A. Landman, Geert Litjens, Bjoern Menze, Olaf Ronneberger, Ronald M. Summers, Patrick Bilic, Patrick F. Christ, Richard K. G. Do, Marc Gollub, Jennifer Golia-Pernicka, Stephan H. Heckers, William R. Jarnagin, Maureen K. McHugo, Sandy Napel, Eugene Vorontsov, Lena Maier-Hein, and M. Jorge Cardoso. A large annotated medical image dataset for the development and evaluation of segmentation algorithms. 2019. URL http://arxiv.org/abs/1902.09063.

[77] Leslie N. Smith. Cyclical learning rates for training neural networks. *Proceedings - 2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017*, (April): 464–472, 2017. doi: 10.1109/WACV.2017.58.

[78] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *3rd International Conference on Learning Representations, ICLR 2015 - Workshop Track Proceedings*, pages 1–14, 2015.

[79] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June-2015:1–9, 2015. ISSN 10636919. doi: 10.1109/CVPR.2015.7298594.

[80] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, and E. Weinan. Convolutional neural networks with low-rank regularization. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 1(2014):1–11, 2016.

[81] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966. ISSN 1860-0980. doi: 10.1007/BF02289464. URL https://doi.org/10.1007/BF02289464.

[82] Juan José Vaquero and Paul Kinahan. Positron emission tomography: Current challenges and opportunities for technological advances in clinical and preclinical imaging systems. *Annual Review of Biomedical Engineering*, 17(1):385–414, dec 2015. ISSN 1523-9829. doi: 10.1146/annurev-bioeng-071114-040723. URL http://www.annualreviews.org/doi/10.1146/annurev-bioeng-071114-040723.

[83] Johan Wagemans, James H. Elder, Michael Kubovy, Stephen E. Palmer, Mary A. Peterson, Manish Singh, and Rüdiger von der Heydt. A century of gestalt psychology in visual perception: I. perceptual grouping and figure-ground organization. *Psychological Bulletin*, 138(6):1172–1217, 2012. ISSN 1939-1455. doi: 10.1037/a0029333.

[84] Wenshuo Gao, Xiaoguang Zhang, Lei Yang, and Huizhong Liu. An improved sobel edge detection. In *2010 3rd International Conference on Computer Science and Information Technology*, pages 67–71. IEEE, jul 2010. ISBN 978-1-4244-5537-9. doi: 10.1109/ICCSIT.2010.5563693. URL http://ieeexplore.ieee.org/document/5563693/.

[85] Yuxin Wu and Kaiming He. Group normalization. *International Journal of Computer Vision*, 128(3):742–755, mar 2020. ISSN 0920-5691. doi: 10.1007/s11263-019-01198-w. URL http://link.springer.com/10.1007/s11263-019-01198-w.

[86] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. 2015. URL http://arxiv.org/abs/1505.00853.

[87] Jingru Yi, Pengxiang Wu, Menglin Jiang, Qiaoying Huang, Daniel J. Hoeppner, and Dimitris N. Metaxas. Attentive neural cell instance segmentation. *Medical Image Analysis*, 55:228–240, 2019. ISSN 13618423. doi: 10.1016/j.media.2019.05.004. URL https://doi.org/10.1016/j.media.2019.05.004.

[88] Tian Chi Zhang, Jing Yang, Jian Pei Zhang, and Jing Zhang. Svm methods in image segmentation. *The Sixth International Conference on Advanced Collaborative Networks, Systems and Applications*, (c):62–65, 2016.

[89] Yu-Jin Zhang. An overview of image and video segmentation in the last 40 years. *Advances in Image and Video Segmentation*, pages 1–16, 2011. doi: 10.4018/978-1-59140-753-9.ch001.