A workflow-integrated brain tumor segmentation system based on `fastai` and `MONAI`

Jostein Rådmannsøy Digernes
Carsten Ditlev-Simonsen

Master's thesis in Software Engineering at

Department of Computer Science, Electrical Engineering and Mathematical Sciences,
Western Norway University of Applied Sciences

Department of Informatics,
University of Bergen

June 1, 2022

Western Norway University of Applied Sciences

Abstract

Artificial intelligence (AI) has achieved great results in medical imaging tasks and has the potential to improve the experiences of clinicians and patients in the future, but on the way toward AI integration in medicine, there are many practical, technical, and societal challenges. In this thesis, we contribute to the development of AI integration in Helse Vest and present a brain tumor segmentation system integrated with their existing research PACS solution. We investigate to which degree integration of machine learning models is currently possible and if additional software development efforts are needed. The machine learning model used is developed with a library combining the two python-based deep learning libraries `fastai` and `MONAI`. This library is currently under development by researchers at Mohn Medical Imaging and Visualization Centre (MMIV), and we compare it with another state-of-the-art framework to quantify its potential usefulness. Additionally, we deploy it in a simple interactive web application. The thesis contains three studies that were conducted to discuss and answer our research goals. All studies used medical data from a data set coming out of the BraTS 2021 segmentation challenge, and our project is a part of MMIV's WIML project [1]. Our achieved results open the way for future developers to continue workflow integrated machine learning in research PACS, and we see many possible directions to take future research.

Acknowledgements

First and foremost, we would like to thank our supervisor Dr. Alexander Selvikvåg Lundervold for his excellent guidance and inspiration through both his courses as a lecturer at Western Norway University of Applied Sciences and as our supervisor. We also would like to thank our supervisor Sathiesh Kaliyugarasan for his great help and expertise throughout our master's degree.

Furthermore, we would like to thank Zhanbolat Satybaldinov for his very helpful work and contributions to our thesis. We also want to thank Hauke Bartsch for supporting us with his great competence.

Thanks to Mohn Medical Imaging and Visualization Centre at Haukeland University Hospital for providing tools and offices to conduct our experiments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Artificial intelligence (AI) has a lot of potential in the field of medicine, and machine learning (ML) algorithms have achieved outstanding results on many different tasks. Deep learning (DL), a sub-field of machine learning, has been particularly impactful [2, 3, 4]. One example with great potential for improvement in the effectiveness of treatment is the segmentation of organs. However, there are still many hurdles left before machine learning-based tools are mature enough for clinical deployment [5]. Figure 1.1 illustrates some of the many infrastructure-related components required for an actual ML system to be fully functional.

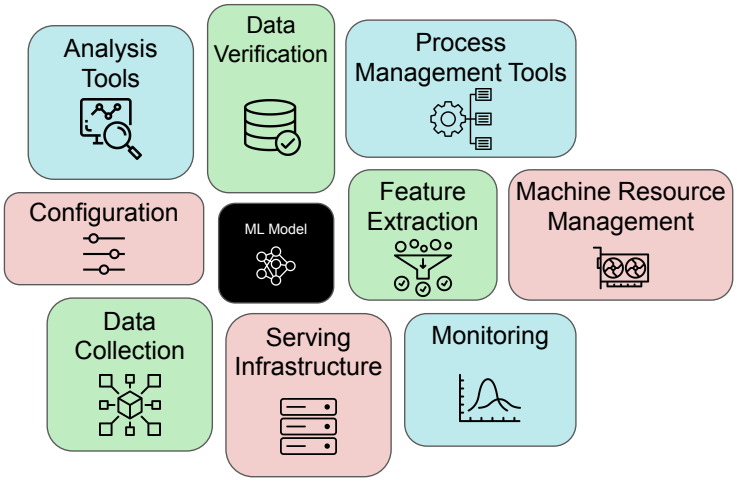

Figure 1.1: In practice, an ML model (the black box) alone only amounts to a small part of the whole picture in a real-world ML pipeline. The different boxes represent the infrastructure needed for an ML system in production. Figure modified from a figure in [6].

The thesis will present a deep learning-based system for fast and accurate brain tumor segmentation based on the *fastai* and *MONAI* deep learning libraries. These libraries are used to train a deep learning model that is then implemented

in a realistic medical environment. This developed model is trained and evaluated on data from the BraTS 2021 data sets[1] [7]. It is then further implemented in a proof-of-concept software application within Mohn Medical Imaging and Visualization Centre's (MMIV) *Workflow-Integrated Machine Learning* (WIML) project [1].

The project has two separate parts: First, constructing a deep learning model using a library extending `fastai` and `MONAI` (described in the following paragraphs), trained on the BraTS 2021 data set. Second, packaging this model in a Docker container that is integrated into the WIML research PACS infrastructure in Helse Vest RHF.

The medical AI research group at MMIV is currently developing an extension of the powerful `fastai` deep learning library [8] to support tasks related to three-dimensional imaging (classification, regression, segmentation), while also incorporating elements from the `MONAI` framework [9].

Such a library combining the best parts from `fastai` and `MONAI` is helpful in a wide variety of medical imaging settings, as it's very common to work with 3D imaging. For example, in magnetic resonance imaging (MRI) and computerized tomography (CT). Researchers at the MMIV have already successfully used the newly developed library in multiple projects: pulmonary nodule classification in lung CT in relation to lung cancer [10], skull-stripping in 3D MRI [11], measuring "brain age" directly from MR images [12], and segmentation of tumors in cervical cancers from MRI [13].

This MSc project contributes to the development of this library by adapting it to an important medical imaging challenge: brain tumor segmentation. Further, it is interesting to see this kind of algorithm applied and implemented through a realistic imaging platform: MMIV's Workflow-Integrated Machine Learning system. A fully functional proof of concept solution in this system can give insight into the hurdles and complications faced when developing such solutions. Figure 1.1 shows some of the many components in a general ML pipeline, and our proposed research PACS proof of concept could act as a tool to unveil these infrastructural challenges involved in medical ML deployment. Many of these components are already established in the hospital, e.g., the serving infrastructure is likely local due to privacy, and the data collection component is data from PACS. This motivates our proposed project of integrating an ML model into research PACS, which could act as a tool for testing, and unveiling these infrastructural challenges involved in medical ML deployment. It can also provide a more complete, visual, and hands-on experience for radiologists and healthcare professionals who ultimately could utilize the solution if ever deployed in practice. The current implementation uses Docker to package a containerized application [14] that can be integrated into the WIML research PACS infrastructure.

The data in the BraTS 2021 data set is three-dimensional medical imaging data. A challenge when working with medical images is the differences between image formats. The *DICOM* and *NIfTI* formats are two image file formats most relevant when working with neurological MRI. The DICOM format is designed for more generalized use and contains more information than the NIfTI

[1] `https://www.med.upenn.edu/cbica/brats2021/`

format. Furthermore, a DICOM image achieves three-dimensionality by layering multiple 2D image slices and saving them in different files as opposed to NIfTI, which is a single file 3D image array [15]. These differences present a challenge for conversion and for avoiding information loss [16].

1.1 Research questions

In this thesis, we wish to explore challenges and solutions for deploying a brain tumor segmentation system. Before attempting to construct a comprehensive system, we create a less complex proof-of-concept application. This approach lets us identify some of the challenges we might encounter later, specifically regarding training and exporting a model and how the model's predictions might be presented in an interface. Furthermore, this proof of concept application could help in further developing the extension of `fastai` combining `MONAI` by uncovering potential development directions of the library. Therefore, our first research question relates to what challenges we can reveal when developing and deploying a proof of concept application.

1. Is it possible to create and integrate a generic setup for easy model deployment within the existing `fastai` and `MONAI` framework extension?

As mentioned, MMIV's WIML and research PACS solution is the case we want to look at for integrating the model into an existing platform. To investigate how feasible this is in practice, we propose a research question related to the possibilities and challenges faced in such integration efforts. We aim to make a list of requirements to be incorporated into the current and future research PACS developments.

2. How can we integrate our trained models directly with the existing research PACS in Helse Vest? Are additional software development efforts and hardware for the research PACS needed?

The research in this thesis will be done using the `fastai` and `MONAI` framework extension, allowing us to investigate its usability and performance and test it in a production-like environment.

This extension aims at combining the simplicity of `fastai` and 3D medical imaging functionality of `MONAI` - two cutting-edge and exciting libraries solving important problems in their respective domains. Changing our deployed model from this extension to a model trained with a state-of-the-art model architecture and framework would be straightforward in theory, so we want to examine how. Comparing this extension with another framework will put it in context with the current best and can quantify its performance and potential usefulness.

Therefore the following question is proposed:

3. How does `fastai` combined with `MONAI` compare to another cutting-edge deep learning framework (nnU-net) regarding performance metrics?

By answering the research questions, the goal is to help the development of applications in WIML. Additionally, an indication of the usefulness of the frameworks utilized is desired, as well as an exploration of the challenges and advantages that lie in deployment.

Chapter 2

Background

2.1 Image segmentation

The field of image segmentation lies in the intersection between digital image processing and computer vision, which can be described as processing digital images through algorithms to gain a high-level understanding of the images. For image segmentation, the understanding is gained by constructing a simpler representation of the images that can be more meaningful to analyze. Image segmentation can be defined as partitioning a digital image into different regions, representing sets of pixels. Specifically, this means assigning labels to pixels with homogeneity among properties. The resulting segmentations can be viewed as nonempty subsets used to facilitate attribute extraction.

Image segmentation as a scientific field has a long history, dating back to the 1960s and 70s. In 1966, a professor named Seymour Papert at the Massachusetts Institute of Technology's AI lab launched a rather optimistic project called "the summer vision project", intending to solve "the machine vision problem" by the end of that summer [17]. Some of the general goals are described as dividing regions of a picture into "likely objects" and "likely background areas" (see section 2.1.1). Unfortunately, the challenges of computer vision are still not solved fifty years later. However, the project represents the birth of many aspects of the field, including image segmentation.

2.1.1 Types of image segmentation

Image segmentation can in general be classified into three sub-categories based on the information they output: instance segmentation (IS), panoptic segmentation (PS), and semantic segmentation (SS).

The fundamental difference between instance segmentation and semantic segmentation is that in IS, each segmentation map has a distinct identity regardless of class. I.e., all maps are treated as individual *instances*. In SS, however, each segmented map of the same class is treated as one single instance. Panoptic segmentation combines both of these aforementioned concepts. Detailed by Geremy Heitz and Daphne Koller [18], and later implemented as the basis for

segmentation categories in the famous Microsoft COCO data set [19, 20], let's distinguish categories of objects as being either *things* or *stuff*. Segmented *things* correspond to IS and are segmentation maps that can be counted through their distinct instance ID. It is usually an object with a well-defined shape, e.g., a fish or a person. Segmented *stuff*, on the other hand, represents amorphous categories that are harder to count, such as a background region (e.g. sky, sea, grass, etc.), and corresponds to SS (see figure 2.1). Using these definitions, PS utilizes some algorithm that enables it to distinguish between *things* and *stuff*.

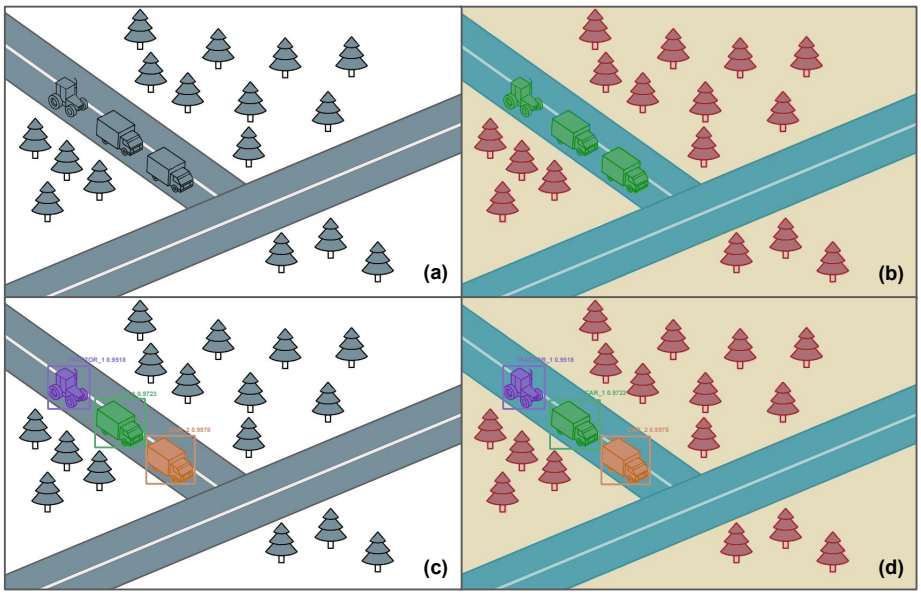

Figure 2.1: Overview of segmentation types: a: raw image, b: semantic segmentation, c: instance segmentation, and d: panoptic segmentation. To simplify, only vehicles are viewed as instances. One could view e.g. trees as instances as well, depending on the task at hand.

2.1.2 Medical image segmentation – an introduction

Image segmentation, in general, plays a vital role in medical image analysis, specifically in the domain of identifying and segmenting organs, tumors, and other regions of interest.

Assigning different sets of pixels in an image into individual segments can create an accurate mapping between that image segment and any structure in the body, e.g., tumors and organs. These segments are made to be internally homogeneous according to at least one similarity metric (e.g., belonging to tumor or non-tumor tissue).

In medical imaging, there is usually a particular region we are interested in inspecting, called region of interest (ROI). To enable further analysis of these regions, we need some way to separate this ROI from the rest of the image (background). After segmentation is performed, one can extract useful information such as organs or tumors' volume and surface area. This way of extracting and

analyzing quantitive images is called radiomics.

The process of manually sub-dividing tissue (i.e., by hand) in medical images by a radiologist in the clinic can be quite time-consuming. It is well documented that algorithmic solutions have shown great potential in making this procedure more efficient by assisting human experts [21, 22, 23].

Segmentation by hand is done every day in the hospital, and it is used for planning radiation therapy and surgery, as mentioned in [24]. Multiple image modalities are used, including positron emission tomography (PET), CT, and MRI. MRI is often used for tumor detection and clinical treatment. In this thesis, our focus is on MRI. With the magnitude of how often this clinical procedure takes place combined with its influence on people's lives, computational tools that can yield significant time savings in clinical practice are immensely useful. Integrating such practices into a radiologist's routine could have great value for both health care providers and patients.

2.1.3 Traditional image segmentation

Traditional image segmentation techniques originated from digital image processing coupled with optimization algorithms. Some of these algorithms set up regions in a picture by comparing neighboring pixel values. For example, region growing takes local averages of features such as grayscale intensity, color, or shape as the basis for pixel comparisons. Based on these comparisons related to some specified threshold, the algorithm assigns pixels to specified regions until a condition is met. This is one of many traditional image segmentation techniques considered state-of-the-art at their time (see figure 2.2). For reviews of broad sets of image segmentation techniques, see [25, 26, 27].

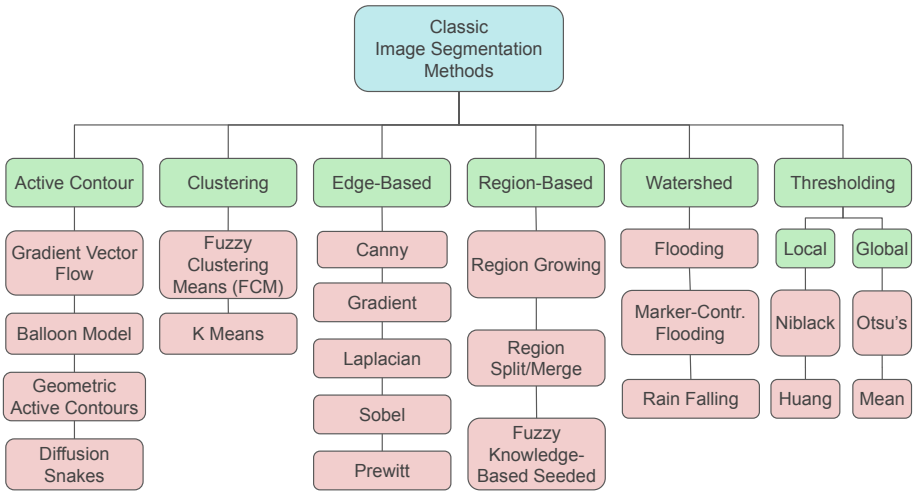

Figure 2.2: Overview of common traditional image segmentation techniques, and some of their respective implementations. These previous inventions illustrate years of effort that are still relevant even in the age of deep learning [28].

2.1.4 The machine learning approach

To understand the later concepts presented in this thesis, a basic familiarity with machine learning (ML) and its differences from traditional programming is essential. It is especially relevant to present this in the context of image segmentation. It's assumed that the reader has some basic knowledge of ML. With that in mind, we will give a basic overview of how ML relates to both traditional programming and image segmentation in this subsection.

Traditional programming and machine learning

In traditional programming, developers manually define and implement the use case and logic within the program. This program, along with data, is then executed on a machine, producing the resulting output. In machine learning, however, results combined with data are used to formulate the logic within the program (see figure 2.3). Note that this means that, in a sense, *the data becomes part of the program.*

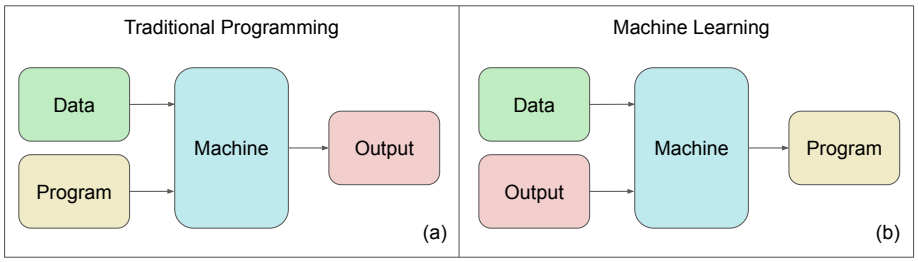

Figure 2.3: A simplified overview of traditional programming related to machine learning.

Developing programs through traditional programming can be a complex task requiring developers to manually maintain a large set of rules. E.g., in fraud detection, this would involve maintaining a blacklist and banning certain words and sentences.

More often than not, the logic of these rules needs to adapt at some point to keep up with factors such as data drift or concept drift (discussed in section 2.3.1). In addition, fraudsters will change their approach given enough time, and the definition of fraud itself might also change. To keep up with this in traditional programming, developers must manually adapt their fraud detection program. This means reprogramming and maintaining the aforementioned large set of rules as their tasks get increasingly complex.

In machine learning, one constructs programs that can learn from the underlying patterns it needs to solve the problem itself, such as detecting fraud or performing image segmentation. Such an approach makes it possible to create programs based on very subtle patterns in the data. Moreover, if modifications to the program are needed, it is possible to learn through new data and update parameters within the model. This is all made possible through a measure of

performance, often calculated by what are called *loss functions* (see figure 2.4 and Section 2.2.1).

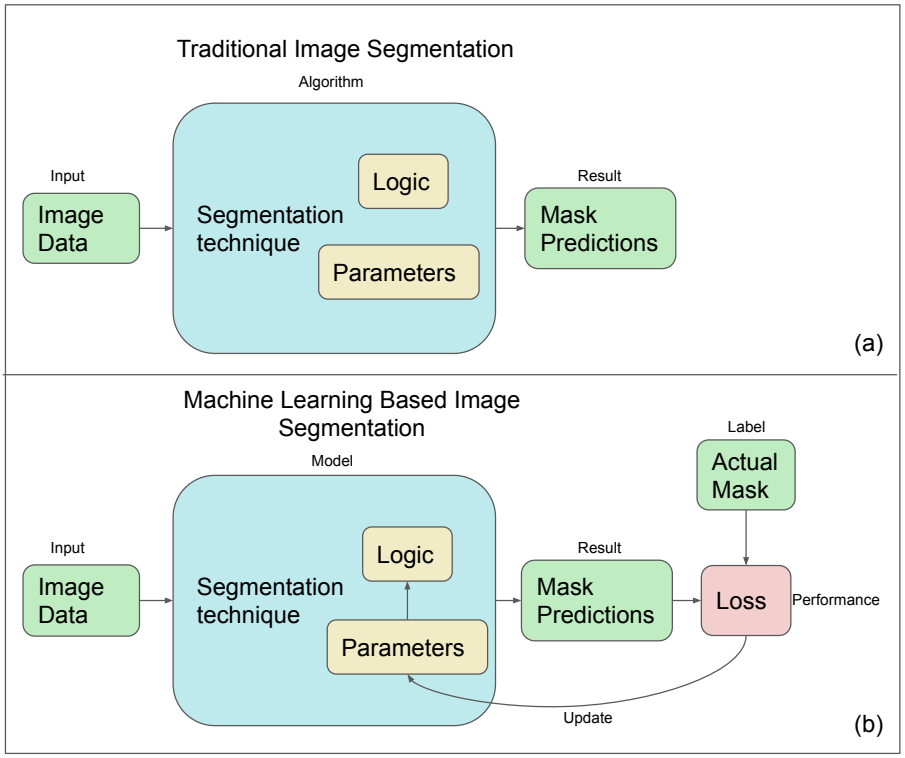

Figure 2.4: This figure illustrates the difference between traditional image segmentation techniques and those based on machine learning. In traditional image segmentation, we have segmentation techniques created with traditional programming. We can have different techniques, test all of them, and pick one based on performance, but as explained in 2.1.5, they are hard to optimize. As illustrated, the machine learning-based version automatically compares mask predictions to what it optimally should have predicted (actual mask), called a label. The relation between these two, called prediction and target, is used to calculate the loss. This, which acts as the performance measure, updates parameters that are often randomly initialized, inducing step-wise improvement. Thus, the way we calculate loss, through a loss function, is crucial to how our machine learning model performs.

2.1.5 Evaluating machine learning models

The goal of a machine learning model is for it to generalize well, and uncover underlying data patterns. This means that as the model is exposed to new, unseen data, it adapts properly. ML models have been shown to lead to some of the best results in a range of domains, but there are still limitations and challenges that have to be taken into account. Initially, the model is built, or *trained*, using *training data*. This is the data that is fed into the model *training loop* to iteratively update the model parameters (see figure 2.4). This training loop

can progress too far, particularly in complex models, where models adapt, or *fit* themselves too much based on the training data. This is called *overfitting* and causes models to generalize poorly. On the other hand, a model can sometimes not be complex enough or have too few iterations in the training loop causing it to generalize poorly as well. This is called *underfitting*. The dichotomy of balancing model complexity and optimizing generalization is commonly referred to as the *bias-variance tradeoff*.

After training a model, we need some way to evaluate its performance. The optimal way to achieve this is to test it on relevant, labeled data points that are unseen to approximate how it would perform in a real-world scenario. By this, we mean that to get an unbiased evaluation, it is important that this data remain hidden from both the model itself and data scientists so no assumptions are made based on it. This data set is called a *test set*.

After the test set is set aside for the final unbiased evaluation, it is common practice to reserve another partition of the training data, called *validation data*. The validation data set helps to evaluate how well a model reacts to unseen data. This is useful as it prevents overfitting during training, and is used to optimize *hyperparameters*. A hyperparameter is a different kind of parameter that is not derived via training but by the data scientist themselves. These control the learning process, how the model is structured and is defined before the learning process begins.

The validation set is used to evaluate the performance of a model during training. This introduces another type of loss, not derived via training, but validation called *validation loss*. Observing how hyperparameters affect validation loss gives an estimate of the influence they have on a model's ability to generalize well. Therefore, optimization, or the *tuning* of hyperparameters is measured based on its effect on validation loss.

When tuning hyperparameters, we are specifically tailoring, or *fitting* our model to this validation set. As tuning progresses, the model does not only acquire a bias toward the test set but increasingly so toward the validation set. This reiterates the importance of an unbiased test set, and it puts a restraint on how much a data scientist should tailor the model to the validation set as well with regard to the bias-variance tradeoff.

Finally, after a data scientist is satisfied with the hyperparameters, it is common practice to train the model a few iterations after merging the validation and test set. This is because, especially in cases with a limited amount of data, we want to use all data available and not let any of the data points in the validation set go to waste.

How we split our data set for training, validation, and testing serves as the basis for how the model views the data, and it can greatly affect the performance of the model. One composition of data might result in better performance than another, and choosing the optimal split is a hard task. K-fold cross-validation is a procedure used to optimize this composition on the cost of computational resources. The procedure splits the data set into k equally sized subsets and trains k models where each of the partitions gets to serve as the validation set for each model trained. After this, it is common to compare the models, and, in some cases *ensemble* them, meaning that they are jointly used to solve the same

problem. This way, we get more out of our data by ensuring all observations get the chance to appear as both training and validation data.

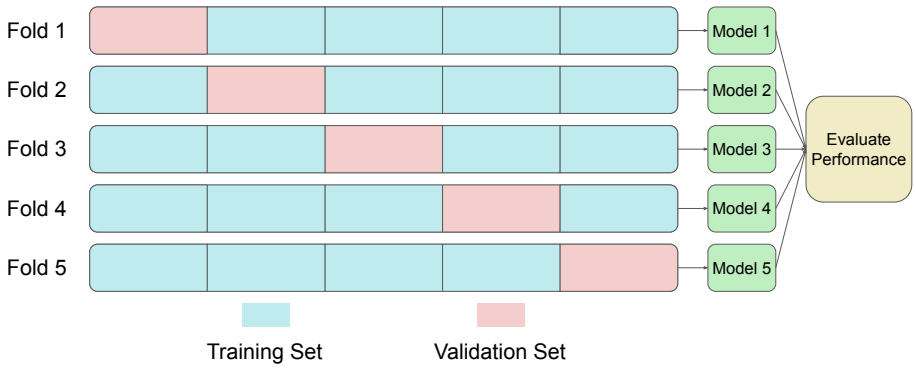

Figure 2.5: K-Fold Cross-Validation where $K = 5$. Fold 1 to 5 represents the same data set, where it is split into 5 equally sized subsets. All folds contain the same data, but the validation set is of equal size as the subsets, and distinct for each fold. This means that each data point gets to serve as validation data.

2.2 Supervised deep learning

Deep learning is a sub-field of machine learning, forming state-of-the-art solutions to a variety of problems, especially in computer vision and natural language processing. This was particularly acknowledged when deep learning methods started to outperform other methods in various image-analysis competitions like the well-known ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This yearly competition, which has now ended, was the benchmark challenge in object detection and image classification. In 2012, a deep learning model of the type *convolutional neural network* (CNN, see section 2.2.2), greatly reduced the error rate to a point no one had seen before.

The various algorithms within machine learning, and its sub-category deep learning, can be divided into three main types: supervised, unsupervised, and reinforcement learning. In this thesis, our focus will be on supervised deep learning. In supervised learning, a function is inferred based on labeled training data as ground truth, acting as a "supervisor". The labeled data tells the system whether it has learned correctly or not. Unsupervised learning learns patterns in data without labels. Finally, in reinforcement learning, a model is trained using trial and error, with rewards for the desired behavior and punishment for undesired behavior dealt out by a *reward function*. This thesis deals exclusively with supervision-based deep learning tasks, and we will not discuss unsupervised learning or reinforcement learning further.

2.2.1 Fundamentals of deep learning

Artificial neural networks (ANNs) are computational systems loosely inspired by neural networks in biological brains, such as the human brain. ANNs are built up by layers of numerous nodes called neurons, with interconnections between

the layers of neurons. An ANN consists of three *layers*: an input layer, one or more hidden layers, and an output layer (see figure 2.6). The input data is in the form of a multidimensional vector, which is distributed into the neural network through the input layer. The neurons in the hidden layers calculate the dot product between the inputs and their weights and add a bias. Finally, the results are passed through a nonlinear *activation function* which produces the neuron's output.

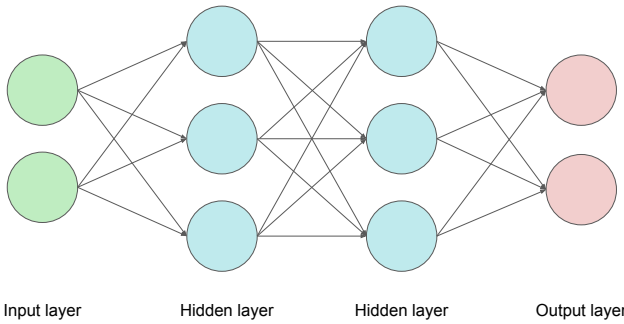

Figure 2.6: An illustration of a simple artificial neural network (a simple *feed-forward, fully-connected neural network*), showing the different types of layers and a possible connectivity pattern. In practice, there would be many more nodes and hidden layers, and the connectivity pattern would generally be more complex.

Training a neural network

Before training a neural network, it is generally a good practice to apply pre-processing techniques to data before feeding it to a neural network. In the case of data points having varying scales, data normalization is a technique that is applied to optimize and speed up the training process. For images, one way to bring those values to the same scale is to subtract the mean and then divide by the standard deviation pixel-wise (Z-Score normalization).

The performance of deep learning models are generally measured through a *loss function* (see also figure 2.4). Loss functions measure the discrepancy between the model output and a corresponding label. There are many ways to model this difference, and which function to apply depends on the problem to solve. Some examples of common loss functions are root mean squared error (RMSE), mean squared error (MSE), and cross-entropy. A simple example usually utilized for regression tasks, MSE assesses the average squared difference between labels and predicted values. This means that the larger a discrepancy is, the more it is penalized. The equation for MSE is as follows:

$$\sum_{i=1}^{D}(x_i - y_i)^2$$

As detailed in figure 2.4, a machine learning model iteratively improves performance metrics by updating parameters within the model based on a performance measure, usually provided as loss from a loss function. Weights and biases within

hidden layers represent such parameters in neural networks. *Gradient descent* is a widely used optimization algorithm used to update these parameters. The algorithm seeks to find the minimum value of a function and takes the derivative of the loss function as input. The goal is to find a particular combination of parameters where the loss value is at a minimum, such a point is called a *minima*. A *gradient* is a vector that represents the direction where the function increases the most, called the steepest ascent. This means that to approach a point with minimal loss (a minima), one has to take a step in the opposite direction of the gradient (the negative gradient). These steps define how it is possible to iteratively minimize model prediction error by repeating this process.

There are three categories of gradient descent: batch gradient descent, mini-batch gradient descent, and stochastic gradient descent. Stochastic gradient descent utilizes a single random training sample to calculate a gradient per iteration. Batch gradient descent loads the entire data set to memory before calculating the gradient per iteration, which is comparatively very computationally heavy. Mini-batch gradient descent utilizes randomly picked sub-sets of fixed size to calculate the gradient for each iteration.

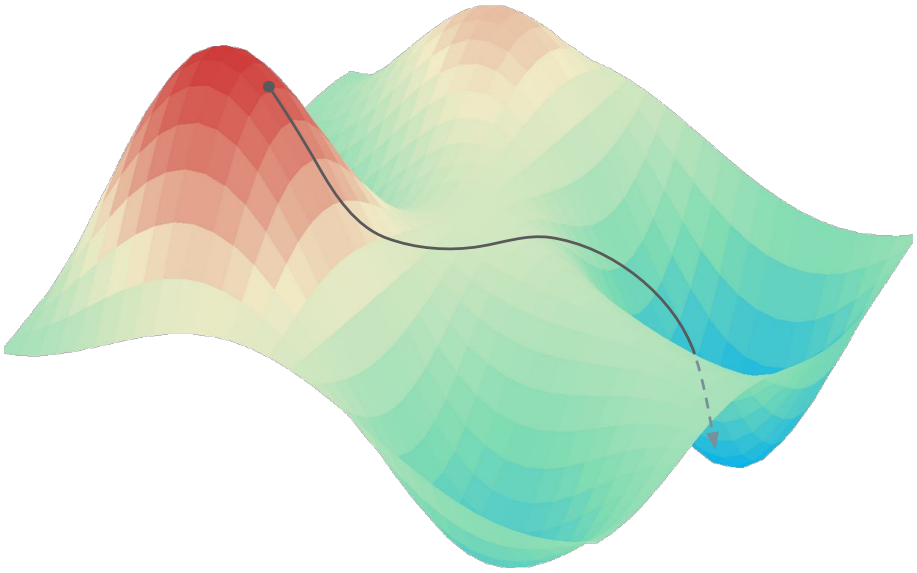

Figure 2.7: Illustration of gradient descent. The red part implies a combination of parameters where the loss is high. The arrow represents the stepwise traversal of the optimization algorithms' path toward the blue valley to find a minima.

Each input data point in the training set is *propagated* through the network to make a prediction. For each input, the loss function provides the performance error. This error is then propagated backward through the network to measure how the current weights affected it. This is a fundamental process within neural network training called *backpropagation*. An optimization method, such as a variant of gradient descent, can then be used to update the weights and biases, aimed at iteratively improving the network's performance as it is exposed to data.

The length of each step the optimization algorithm takes in the direction of the gradient is a hyperparameter called a *learning rate*. It is hard to determine the optimal learning rate before training a model. A large learning rate can converge quickly, but it might also pass over a minima, while a small learning rate converges slowly, and could get stuck. This leads to the need for ways of determining good learning rates and establishing learning rate schedules adapted to specific models, problems, and data sets.

2.2.2 Convolutional neural networks

Among the different architectures of artificial neural networks, the convolutional neural networks have shown great performance in computer vision tasks such as image classification and image segmentation. CNNs are a form of artificial neural networks, but they have some key features that make them more suitable for large inputs with a particular geometric structure, such as images. CNNs make some geometrical assumptions about the input: if detecting a pattern is important at some position in the input, it should also be important at other positions. This assumption means that a specific pattern in an image can be recognized even if its position is shifted. This is called *translational invariance* and is realized in CNNs through *weight sharing* (sometimes called *parameter sharing*). The layers that are most common in a CNN for classification are the convolutional layer, pooling layer, and fully connected layer.

Convolutional layer

The convolutional layer is central to CNNs, as the name implies. A first convolutional layer in a CNN might locate edges, colors, or other basic features in the image. Deeper into the CNN, the next convolutional layer could then find more advanced shapes or features, visualized in figure 2.8.

The convolutional layer consists of trainable matrices called *kernels*, or filters, which are smaller than the input matrix but share the same depth. The kernel glides (or convolves) over the input matrix with a *stride* at a time and calculates the convolution between the kernel and the *receptive field* (the region of input values with the same size as the kernel, see figure 2.9). This convolution can be thought of as weight sharing, as all neurons in a depth slice use the same fixed kernel. The output matrix produced by the convolution is called a *feature map* (or activation map). The feature map will have the same depth as the kernel, and each layer of the map is the result of a convolutional operation between the weight parameters within the kernel and the input data. The shape of the feature map is determined by its input shape and the kernel's shape and stride.

An important attribute of the convolutional layer is each neuron's local connectivity. Since the network is dealing with high dimensional inputs, connecting all neurons between each layer like an ANN is impractical. The number of weights between neurons increases exponentially with the size of the image input. Full connectivity also treats distant pixels as importantly as local pixels. Instead, each neuron is connected only to a small region of the input. The local connectivity's range is determined by the filter size, also called *receptive field size.*

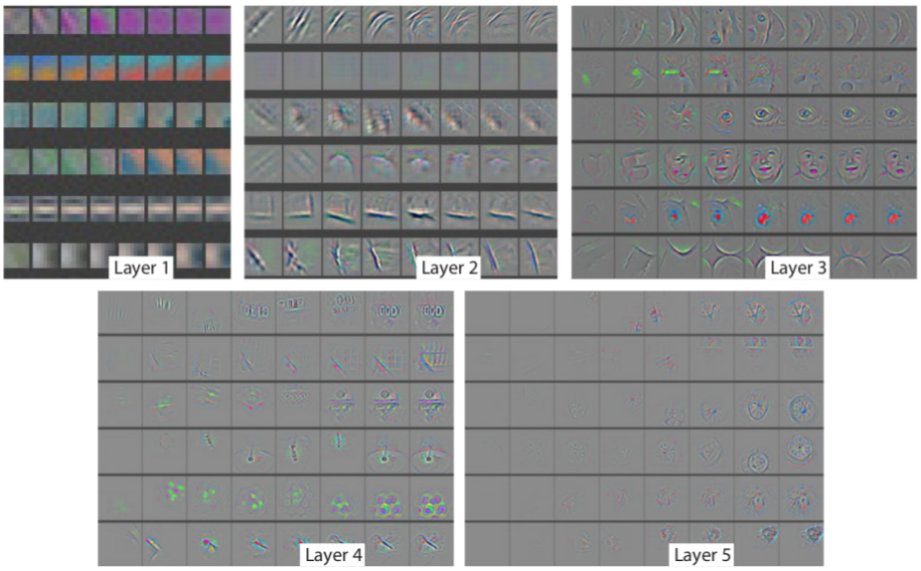

Figure 2.8: Visualization of shapes and features discovered in a CNN. Note how the features in early layers find basic shapes and colors, while in layer 3 advanced features such as faces are picked up. Figure is from [29].

A convolutional layer is followed by an activation layer. The feature map output of the convolutional layer is sent through a non-linear activation function, typically a *rectified linear unit* (ReLU): $f(x) = \max(0, x)$. This is an important step and needed for the model to approximate any non-linear function.

Pooling layer

The pooling layer's objective is to reduce the size of the activation maps. When the number of parameters in the activation map is reduced, the computational complexity also lessens. Down-sampling also helps to control the model from overfitting.

Similar to the convolutional layer, the operations in a pooling layer are done with a kernel. However, instead of calculating the convolution, it performs a pooling operation, usually *max pooling* or *average pooling*. In max pooling, the kernel glides over the input and selects the largest value (see figure 2.10). The size of the kernel is usually 2x2, with a stride of 2. A larger kernel window causes more information loss. Average pooling is similar but finds the average instead of the maximum.

Fully connected layer

In the fully connected layer, all neurons have connections to all neurons in the two adjacent layers, much like the neurons in a regular neural network. This layer is often added to the end of a CNN, where it performs the classification task based on features from previous layers.

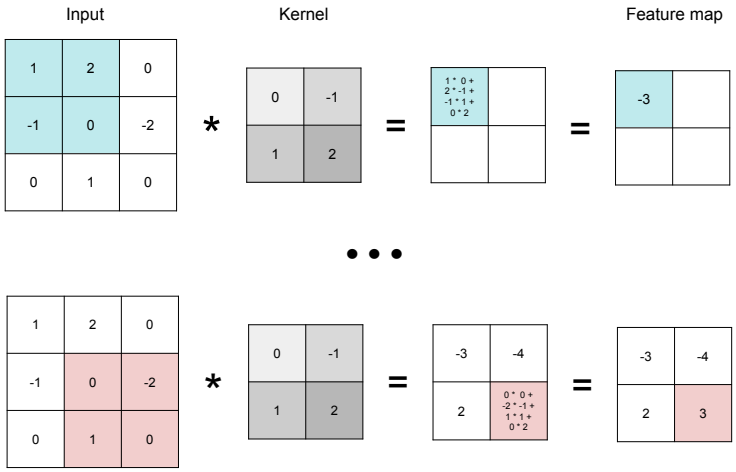

Figure 2.9: Illustration of a kernel calculating a convolution of some input. The dimension gets reduced after the calculation, this can be mitigated by adding padding. The kernel's stride is the number of pixels the kernel moves over the input at a time. The kernel in this figure has a stride of 1, and size of 2x2.

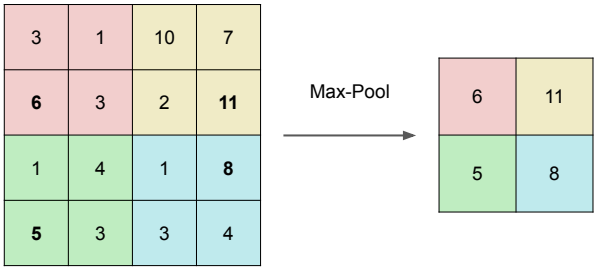

Figure 2.10: Illustration of a max pooling operation. Size is 2x2 and stride is 2. Note the 75% information loss: 16 data values become 4. Although a lot of information is lost by pooling, it improves efficiency and reduces the risk of overfitting in the network.

The CNN architecture

A typical CNN for image classification often consists of these layers together (see figure 2.11. When designing new CNN architectures, these layers are the basic building blocks, but often more advanced structures and layers are added on top of them, while some layers are dropped entirely as the pooling layer and fully connected layers can both be replaced by convolutional layers.

Typically the convolutional layer is followed by pooling layers. It is also common to follow a convolutional layer with another one before the pooling layer, enabling the convolutions to find more advanced patterns before down-sampling. The alternating layer pattern of convolutional layers and pooling layers repeats throughout the architecture. In a basic CNN for image classification, the last layer(s) is often a fully connected layer.

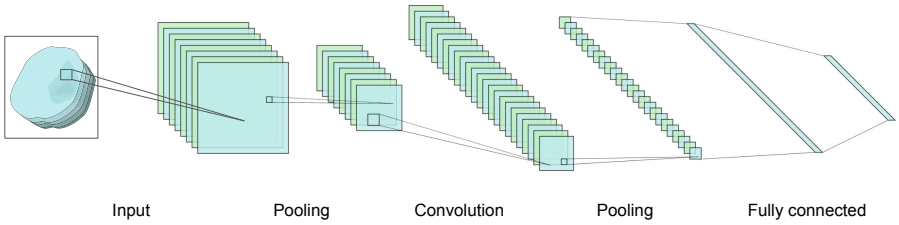

Figure 2.11: Illustration of building blocks of a typical convolutional neural network for classification. The layers listed below the figure are convolutional layers, pooling layers, and fully connected layers. Note that every convolution is also followed by a non-linear activation. Figure made using NN-SVG [30].

2.2.3 Convolutional neural networks for image segmentation

CNNs have commonly been used for image classification tasks, but they have also excelled at image segmentation, a more complex problem than classification. Instead of the model's output being a single prediction, it also includes the prediction's location within the image in the case of instance segmentation. Alternatively, the model outputs a whole image classifying each pixel in the case of semantic segmentation (see section 2.1.1).

Image segmentation tasks require a different architecture to classification tasks, given the difference in output. In semantic segmentation adding a 1x1 convolution at the end of the model changes the output to a feature mapping. This feature mapping has the same size as the input and a depth set to equal the number of label classes for the task. A CNN consisting of no fully connected layers is called a fully convolutional network (FCN) [31]. FCNs have lately gained attention in semantic segmentation tasks.

In addition to a different architecture, segmentation tasks require different functions to measure performance. In section 2.2.1, mean squared error and cross-entropy was mentioned as common loss functions. Segmentation tasks can be evaluated with different loss functions, a common one is dice loss which has the following formula:

$$Dice\ loss = 1 - \frac{2|A \cap B|}{|A| + |B|}$$

The dice metric measures similarity between two sets (illustrated in figure 2.12) and returns a value between 0 and 1 where 0 is no overlap and 1 is a perfectly overlapping segmentation. Dice is used both as a loss function and as an evaluation metric, along with other metrics for segmentation such as *Jaccard index* and *Hausdorff distance*. The latter calculates how close two subsets are to each other, by measuring how close every point of one subset is to some point in the other subset.

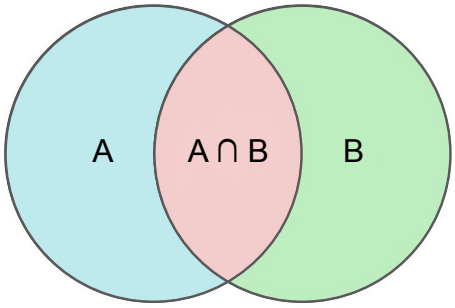

Figure 2.12: Visualization of dice coefficient, a performance measure in segmentation.

The U-Net architecture

The U-Net architecture [32] is a type of fully convolutional network designed specifically for doing medical image segmentation tasks. The network gets its name from its symmetrical shape, one downwards contracting path and one upwards expansive path, forming a "U" (see figure 2.13).

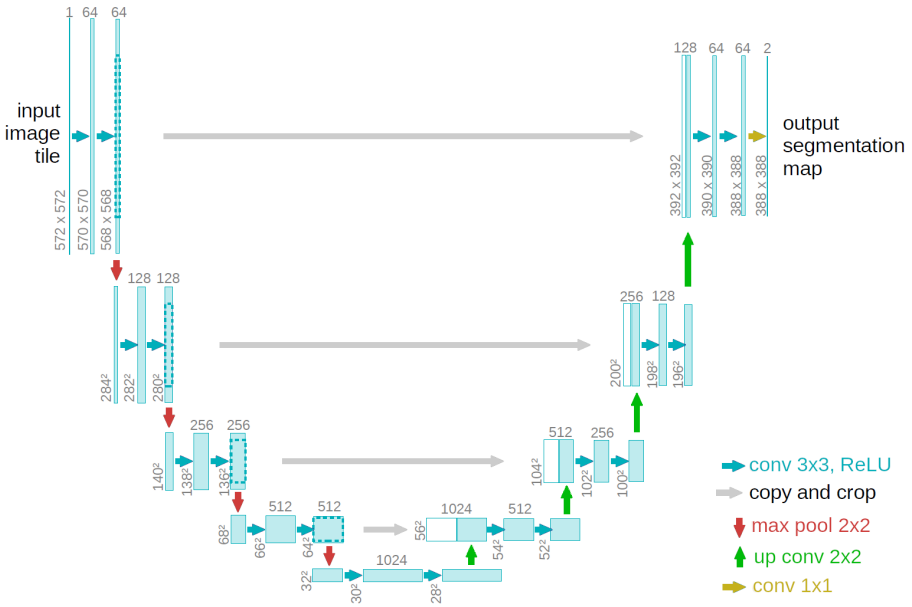

Figure 2.13: The U-Net architecture. The arrows denote the different operations. Figure from [32], with permission from the author.

The network has some additional operations in its architecture: *up-convolutions* and *skip connections*. Up-convolution is a technique to upsample or expand an image. In contrast to the downsampling max-pooling operation, the up-convolution operation is trainable. The feature mapping is upsampled to a higher dimension and to a shape that lets it concatenate with the previous feature mapping from the corresponding layer in the contracting path. This

concatenation is done through a skip-connection. The concatenated feature maps from the opposite layer are reused, recovering information from previous layers before it was downsampled, allowing feature maps to be preserved even in a deep network. This mitigates the problem of *vanishing gradients* caused by backpropagation in deeper networks.

Many newer networks designed for segmentation tasks have used U-Net's architecture as a base. See [33] for a review of its variants and their development. A 3D U-Net architecture was introduced in [34], extending Ronneberger's previous U-Net by replacing all 2D operations with their 3D counterparts, as in 3D convolutions, 3D max pooling, and 3D up-convolutions. U-Net architecture has later implemented *residual blocks* within the encoding layers to improve performance [35]. These blocks contain additional residual connections, which are a type of skip-connections. Recent *transformer*-based variants of 3D U-Net have also achieved good results, for instance UNETR in [36]. Transformers are a type of neural network architecture that has seen a lot of success in ML tasks based on language, and also more recently computer vision. In short, these are based on a technique called *attention*, which weights different parts of the input data differently, as well as divides the input into sequential image patches.

2.3 Deployment of machine learning models

Deep learning models have shown excellent results on a wide variety of problems, but to apply the models to real-world tasks, they need to be deployed.

A well-performing deep learning model getting great results on a test set and trained with a refined architecture can be powerful. But this model will be useless in practice if its domain is not fully understood and you don't account for the many challenges associated with model deployment. The challenges are many, ranging from technical infrastructure challenges to questions related to societal impact, including ethical implications relating to biases, privacy concerns with data collection, gaining the trust of end users, model performance and robustness in practice, and overall usefulness and effectiveness of the model. These challenges get amplified when deploying to clinical settings, where consequences for neglecting these issues are severe.

This chapter will present different challenges with machine learning deployment in general, in the medical domain, and look at the current progress of AI in health and medicine.

2.3.1 Challenges with deployment of machine learning models

Before delving into the ethical issues of medical AI, we will look at the basics of model deployment and some of the typical problems that arise when deploying all types of machine learning models.

What is deployment?

Model deployment is a process for publishing a model and making it available to perform predictions in a production setting on real data. The model is made

available to an end user through a system. Deploying a machine learning model and making it available for use does not mean it is complete, and a model scoring well in metrics during development does not necessarily perform well in production since there are many new factors to consider. Machine learning development is a cycle and needs to be continually updated throughout the application's lifetime, as seen in figure 2.14, where deployment is an important part.

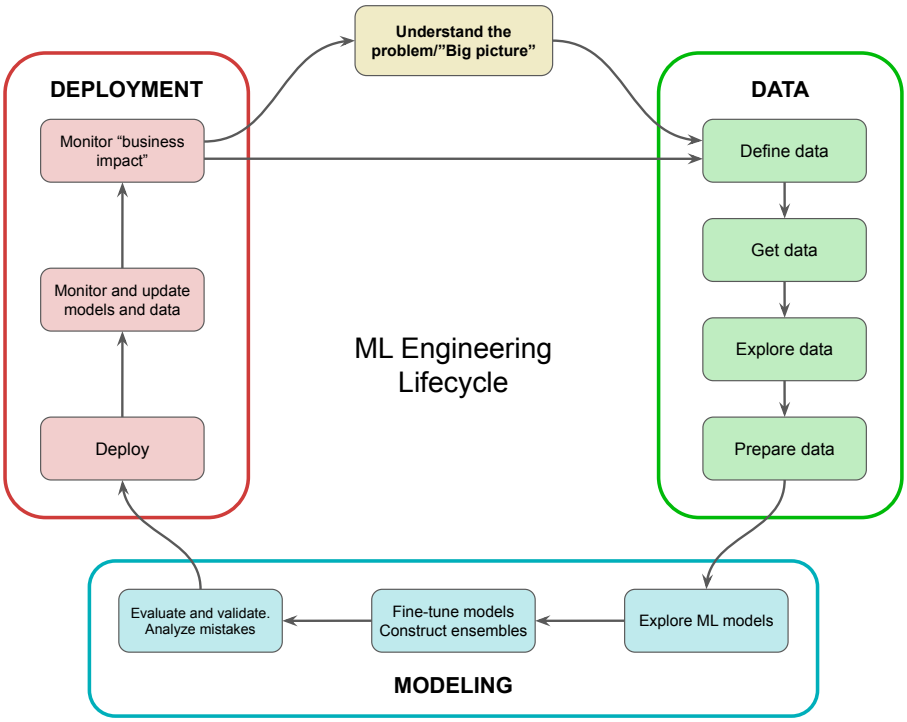

Figure 2.14: The machine learning engineering lifecycle. Model engineering is an ever-developing cycle that does not stop after deployment but instead starts the development process again. The three parts of the lifecycle are data, modeling, and deployment. However, starting the development by first understanding the big picture is a good practice. Examples of questions to ask at this stage are: "What problem should be solved?", "How is the problem solved today?", and "Is it feasible to solve this problem with machine learning?". Additionally, metrics to evaluate the model and the business impact should be defined early.
The modeling part of the machine learning engineering cycle is covered earlier in section 2.2. However, deployment and monitoring are some of the most challenging stages of the lifecycle, both from a technical, practical, and ethical point of view.

Monitoring

Deploying a model and monitoring it is one of the most important and challenging parts of model engineering. The goal of monitoring is to track the health and value of a deployed system, more precisely to mitigate *model decay*. Model

decay is when predictions of a model get less reliable over time. This typically happens when the model is deployed in a highly dynamic environment, where the model performance deteriorates over time. We will look at what causes model decay and how one can mitigate it by monitoring, which is a crucial part of the model engineering workflow for ensuring model value in practice.

If data and the data distribution in production differs from training data, model predictions will falter. This difference between training and test data and production data is called *data drift* and can happen if there is a significant time delay between training the model and integrating the model [37]. Other factors can affect data drift, such as errors relating to data collection or seasons. An example is a data set meant for predicting human behavior collected before the COVID-19 pandemic but deployed after. *Concept drift* is a different source of model decay where it's not the data that changes over time but instead our underlying understanding of it. In a classification problem, the same data point might change its class due to concept drift, even though it is still the same data point. Figure 2.15 visualizes the difference between these.

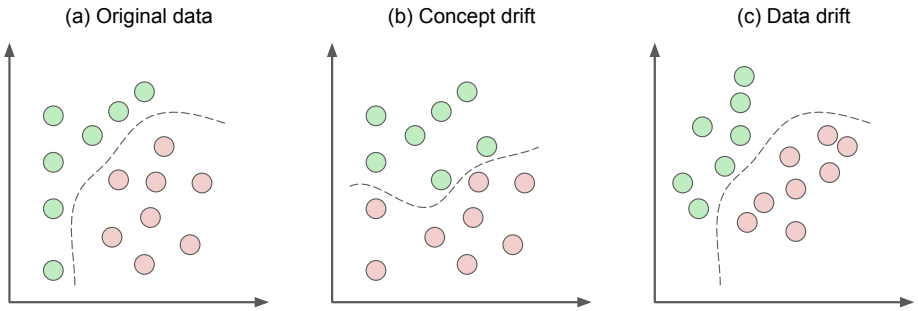

Figure 2.15: Illustrating the difference between concept drift and data drift in a classification problem. In concept drift the data doesn't change, but rather the problem context. Data drift is a shift in the data distribution, sometimes called virtual concept drift or sampling shift.

Comprehensive live monitoring is important for the long-term reliability of a system [6]. But detecting decay is not an easy task, as well as knowing what to monitor. A typical solution is a dedicated monitoring approach, either automatic, manual, or both [38]. Continuous evaluation in addition to well-defined metrics is key for monitoring models in production. Input data can be tested for consistency and distribution over time, while predictions can be monitored by domain experts routinely examinating.

If a slight drop in model performance over time is detected, this is a sign of model decay. Action is then needed to bring the model back to expected performance, either by fine-tuning the model or building a new model from scratch. Experiences gained from deployment and monitoring are valuable when redesigning the model, and data collected while deployed should be used for re-training. The extent of the re-training will depend on the situation, with some factors being how much the model performance has drifted and how much new labeled data is available.

In addition to the technical challenges with deploying machine learning models to production as described above, there are also many practical challenges involved. Similar to how model performance is monitored, the business impact of a deployed system also needs to be monitored. The cost of developing and running a well-performing model has trade-offs. A more complex model can achieve better results, but it also might require more resources in both training and serving, which would increase expenses. Finding the optimal balance for the relevant use case is a challenge, how to best turn accuracy into revenue.

Increased model complexity can also affect the end user's experience, by affecting latency and throughput. Latency is the delay between the user performing an action and observing its result, while throughput refers to how many requests a system can handle and process over a set period. Finding the optimal balance between these metrics will depend on the required service. Some systems might need low latency to keep user engagement, while others require high throughput. If a system requires both, it will likely come with a cost expense.

2.3.2 Challenges with deployment of machine learning models in health and medicine

As mentioned earlier, AI has vast potential in medicine, and a plethora of research has gone into developing high-performance deep learning algorithms. Image algorithms have demonstrated the ability to detect diseases from images at the level of experts [39, 40]. But research has not been as comprehensive in integrating these machine learning models with clinical practice due to the many challenges and limitations associated with deep learning. The challenges are both technical and ethical, in addition to the practical implications of adding AI to clinical settings and changing already well-established workflows. An example is a study from Google Health in 2020 [41]. They had developed an algorithm that identified diabetic retinopathy at a specialist level accuracy, and could potentially reduce waiting time from the eye screening process from weeks to minutes. But when the algorithm was used in a real-world clinical setting it sometimes failed to give a result at all, and occasionally caused frustration from nurses and patients. The study shows that the benefits of AI can be large, but real-world unexpected real-world situations need to be considered, and how a model's usefulness in practice needs to be monitored.

Further in this section, we want to cover some of the ethical challenges that are slowing down the medical integration of AI.

Trust

One of the biggest hurdles AI is facing before integration in medicine is building trust with the end users. Several aspects are needed to gain this trust. Models have to demonstrate robustness and reliability, not only in the lab on testing data during the development process, but also in rigorous tests on heterogeneous incoming clinical data. Explainability is another factor that affects trust. AI, in particular deep learning models, function essentially as *black boxes*, with no reliable way to explain how an algorithm lands on a certain prediction. The weights and parameters in the hidden network layers do not provide intuitive explanations on how it comes to a decision. Explainability is especially important

in health, as a medical doctor needs to be able to explain treatment decisions to a patient [42]. The problem with explainability has led to the emergence of a sub-field called explainable AI (XAI). Techniques exist that partially uncover a model's reasoning by highlighting the most important regions of an image being used to come to a certain prediction. These highlights can be useful for both end users as insights into the model prediction, and the developers when optimizing model performance. However, these techniques still have limitations that may reduce their reliability [43].

Involving clinicians in the different steps of the development cycle (see figure 2.14) may also increase their trust in the system, as they can evaluate and give feedback during development. However, even if the end user's trust is won, and they are interested in using prototype applications for medical AI, there are still many hindrances in the executive parts of the clinic with strict regulations and further ethical concerns.

Model fairness

The factors presented in the last section are not the only ones that cause uncertainty towards AI, model fairness and learned biases are other examples. If the data and labels are not carefully selected a model can learn discriminative biases towards marginalized groups. It is a huge ethical problem in itself and needs to be carefully checked before deploying any system into use [44].

Responsibility and accountability

The emergence of AI has also raised concerns regarding accountability. It is currently uncertain who should be held accountable if a model makes mistakes, even after being approved clinically [45], whether it is the developers, regulators, vendors, or the clinician who is liable.

The questions regarding accountability and the uncertainty around the model's unknown biases both affect the end user's trust, adding to the challenges that need to be overcome before ML-workflow integration becomes routine practice.

Data

Large sets of training data are important for training a robust deep learning model, as neural networks have large appetites for data. Lack of training data is a common obstacle in medical image analysis. The bottleneck is however not the amount of available imaging data in medical archives, which are filled with large magnitudes of images. The challenge lies however in obtaining task-relevant labels/ground truths for these images. Turning images from PACS-like systems into suitable images with precise labels for a specific task will require domain experts (e.g. radiologists). For example, in 3D brain tumor segmentation, each slice of a three-dimensional image needs to be labeled, which is a time-consuming process [46].

Perhaps even more important are the challenges related to privacy issues, information protection, and data access. If the training data is not anonymized, it is left vulnerable to privacy attacks. Therefore deep learning work on medical images is most often done with data sets that have already been anonymized.

Anonymizing the data adds another step of data preparation. Other machine learning techniques ensure better data privacy. For instance, in federated learning, an algorithm is trained across decentralized systems, using local data without further sharing it.

Chapter 3

Workflow-integrated machine learning in radiology

This chapter presents the design of a system for integrating machine learning models in the radiology workflow in Helse Vest and some of its key software technologies. Figure 3.1 shows an abstract illustration of the project.

Radiology is a branch of medicine that uses medical imaging to diagnose and treat diseases. Machine learning and deep learning can make a significant impact in the field of radiology, as radiologists analyze large amounts of images every day. This process could be sped up and otherwise assisted by using machine learning-based tools. The treatment delay in image processing could decrease when integrating deep learning tools with radiology (augmented radiology) and would cause the medical costs to be greatly reduced [47].

The following sections will introduce some important technologies and infrastructure the project is working with for workflow-integration.

3.1 Medical image file formats

Image file formats standardize how image information is stored in a computer file. Some well-known image file formats today are JPG, PNG, and GIF, however, these files do not contain enough necessary information to be used for medical images. Medical image file formats need to describe how the image data is organized and how it should be interpreted by software for loading and visualizing [48]. The two most used file formats for medical scans in this project have been DICOM and NIfTI.

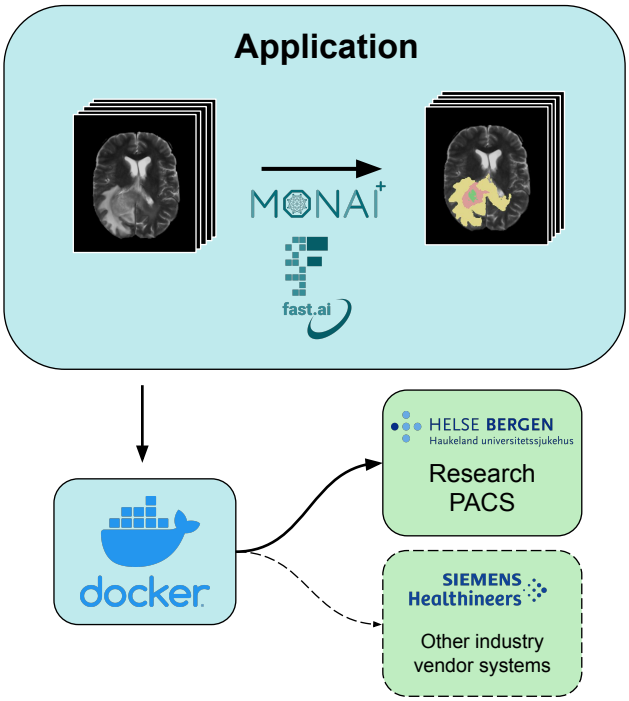

Figure 3.1: Illustration of the project's application pipeline and design. The main application performs an image segmentation on incoming DICOM images using `fastai` and `MONAI` with a U-Net model [32] or similar. A Docker image containing the application is then created, which can be integrated into research PACS and other potentially other industry vendor systems.

3.1.1 Digital Imaging and Communications in Medicine (DICOM)

DICOM is both a medical file format for images and a data exchange format that allows systems to send and receive image data from other systems in the hospital.

Tags in the DICOM file format are structured in groups. There are groups describing the last transfer of a file, groups that describe the patient, and groups that describe the parameters used in the data acquisition. The header is flexible so that vendors can add their own private tags. Some subset of the tags is made mandatory by the DICOM standard[49].

Each DICOM file is standalone, so you can identify it by itself. They are divided into four information levels, patient, study, series, and instance. DICOM files store one slice per file, so a single 3D scan may be hundreds of files, all with unique instance IDs, but the same series ID.

3.1.2 The Neuroimaging Informatics Technology Initiative (NIfTI)

NIfTI is based on an old volumetric file format (Analyze), and the newer NIfTI file format is used to store a collection of DICOM images as a volume or time series. [50] Appending the individual images and reducing the header information the file format is sufficient to store the image pixel information as well as the volume element (voxel) size and the distance between consecutive slices. Some tags present in the DICOM file format are removed when translating the files. Due to its simplicity, the file format is used by many post-processing software packages.

3.2 The WIML pipeline at Helse Vest RHF

WIML is an ongoing research project at Helse Vest RHF, funded by the Norwegian Research Council[1]. WIML aims to address a missing part of the successful integration of AI in clinical practice: integrating computational imaging methods for research purposes with the existing imaging solutions in a hospital setting. The primary objective of the research project is to create a functioning prototype system that can receive medical images from the clinical PACS archive (discussed in Section 3.2.2 below), perform a pseudonymization of the DICOM data, forward them to the Research PACS and perform a prediction with a deep learning model, returning and storing the derived information (segmentation, measurements, report) in the PACS archive ready for clinical reads in the PACS viewer.

In this section different components of the image data analysis pipeline of WIML are presented, as well as the software tools related to development and use.

3.2.1 Picture Archiving and Communication Systems (PACS)

Picture Archiving and Communication Systems (or PACS) is a medical technology that provides storage and transmission of different types of medical images for healthcare institutions. The images in PACS can be of different types of modalities such as CT, X-ray, ultrasound, or MRI, and they are typically stored as DICOM image files in the system. PACS has replaced the need for hard copy image storage in the clinic and, it allows instant access to data and remote access for different clinicians to study the same data concurrently at different locations (see figure 3.2). The PACS system is usually integrated into the hospital to receive study planning information, requests for specific imaging procedures, and information on performed procedure steps.

3.2.2 Research PACS

This thesis project works with the research PACS system at Helse Vest RHF. As the name implies, this system is purely for research purposes, in contrast to the clinical system which concerns patient care and clinical workflow. Inside

[1] `https://prosjektbanken.forskningsradet.no/en/project/FORISS/309755`

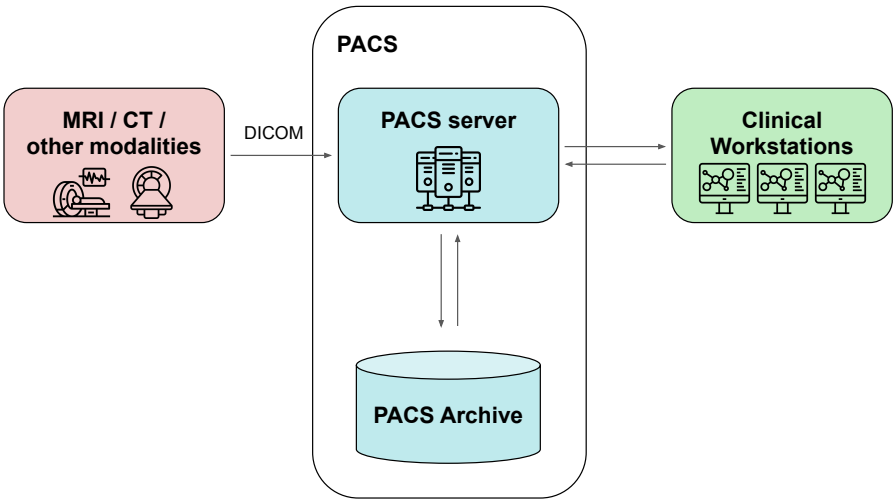

Figure 3.2: Illustration of the architecture of a clinical PACS system. DICOM files from image scanners are sent to PACS and stored in the archive, where clinicians can remotely and concurrently access it.

the research PACS, one has access to copies of the data from clinical PACS, transferred and pseudonymized from hospital personnel. Importantly, similar to the clinical PACS the Research PACS can also receive data directly from hospital scanners. Additionally, the research PACS can import imaging data from external third parties, such as online image repositories and PACS-to-PACS connections across Norway.

The DICOM files of medical images typically store identifying information about the patient in their file headers, such as the patient's name, address, phone number, and other sensitive data. Crucially, all personal information is anonymized as a privacy and security measure before the files are stored in research PACS. [2] This de-identification process is performed during the transfer process from clinical PACS. Additionally, DICOM files can sometimes have sensitive information burned directly onto the image data as pixel information. Aasen and Mathisen investigated ways to detect and remove this type of data using deep learning methods in a previous master thesis at Western Norway University of Applied Sciences [51].

Research PACS is one of the components of the research information system at Helse Vest. Research PACS is responsible for the storage and review of imaging data, while another component called the *research electronic record system* collects tabulated data such as questionnaires and diagnostic information and saves it. These two components are connected by a *Flash-based Input Output Network Appliance* (FIONA), a virtual machine acting as a gatekeeper and quarantine station for data entering the research PACS, illustrated in figure 3.3.

[2] A complete list of the DICOM tags containing sensitive information can be found here: `https://github.com/mmiv-center/DICOMAnonymizer/blob/f0762643caab3d84e522b99cdec4b8d271b12039/anonymize.cxx#L62`. Only the tags marked *keep* are unchanged before saving the data to research PACS, the rest are either removed, pseudonymized, or similar.

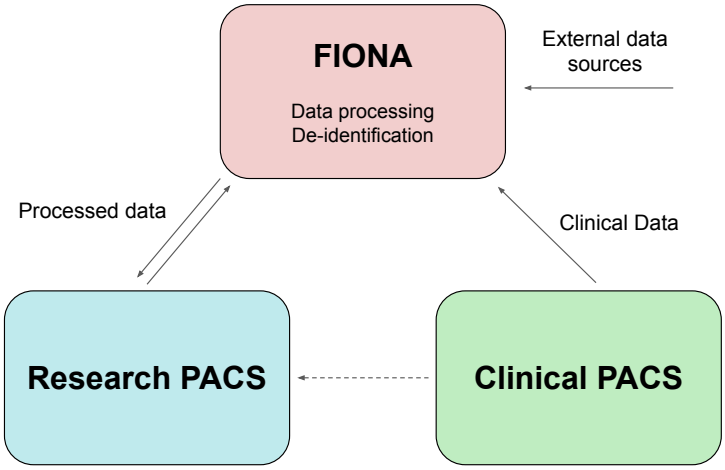

Figure 3.3: An illustration of the Helse Vest research PACS. Data transfers between the systems go through the FIONA gatekeeper.

FIONA is a web application acting as an edge system of research PACS responsible for all data transfers to research PACS, either from clinical PACS or external data sources. FIONA is also responsible for performing the de-identification process between clinical PACS and research PACS, as described in the previous section. In addition to de-identification, and more important to our project, FIONA supports uploading custom data processes called workflows. Uploaded workflows can perform predefined processing tasks, and when uploaded they are connected to projects inside research PACS. The workflows are triggered either explicitly or at the arrival of new data in the linked research PACS projects. When uploading workflows to FIONA, the system expects the application to come packaged in a *Docker image*. Docker images are files that contain application code and all its dependencies and they act as a template to create *Docker containers*, which are the actual environment that lets you run the code. Docker makes it possible to easily deploy and run code in different systems.

Section 4.4 describes the experiment regarding research PACS, going into detail about procedures and methods used to integrate a new ML workflow into the system.

Chapter 4

Experiments

In this chapter we document the three experiments done in our thesis:

1. Creating a simple and interactive web application for easy model deployment.
2. Integrating trained models with the existing research PACS at Helse Vest.
3. Comparing the `fastai` and `MONAI` deep learning libraries combined with another cutting edge deep learning framework.

These three experiments are carried out to help answer our research questions presented in section 1.1. The respective experiments correspond to the research questions of the same numbers. Before describing the different experiments, we need to first present the data set and deep learning libraries used in all studies.

4.1 BraTS 2021: the data set used in our experiments

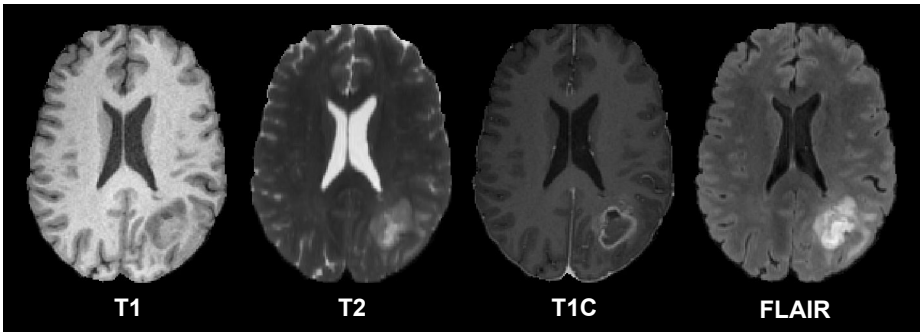

Figure 4.1: The different types of MRI sequences from the BraTS 2021 data set used in all experiments. T1-weighted, T2-weighted, contrast-enhanced T1-weighted, and T2-weighted FLAIR. The different images reveal unique details: see how the tumor in the lower right of the brain differs between the sequences.

All experiments in the thesis are done using the BraTS 2021 data set. [7, 52, 53] The Brain Tumor Segmentation (BraTS) challenge is a yearly public challenge that aims at evaluating state-of-the-art segmentation methods. The BraTS data set consists of 3D MRI brain scans with tumors, more specifically gliomas. Gliomas are the most common and aggressive primary tumor of the brain. They are intrinsically heterogeneous in appearance, size, and shape. Glioma tumors are considered to be among the deadliest human cancers [54]. Early detection, accurate segmentation, and volume estimation are vital for survival prediction, treatment, and planning of surgery. As mentioned in section 2.1.2 segmentation in practice is a tedious, manual, time-consuming process and requires close supervision from an expert [55].

As mentioned, the image scans in the data set are all MRI scans. An MRI scanner uses a strong magnetic field and radio waves to generate images of organs inside the human body, in this case, the brain. The images are multi-modal, meaning each image consists of four *image sequences* (T1, T2, contrast-enhanced T1, FLAIR). These different sequences are captured differently in the scanner and reveal different details for different types of tissue, as shown in figure 4.1.

The data labels are multi-label, meaning the goal of segmentation is not only to detect the tumor but also its different sub-regions. The BraTS 2021 data set is split into three sub-regions: GD-enhancing tumor (ET), peritumoral edematous (ED), and necrotic tumor core (NCR). See figure 4.2. These three-dimensional labels were manually segmented by raters based on the same protocols and approved by experienced radiologists.

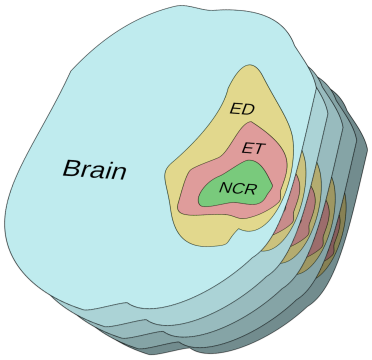

Figure 4.2: Simple illustration of two-dimensional image slices stacked, composing a three-dimensional image of the brain. The different sub-regions labeled in the BraTS 2021 data set are illustrated: GD-enhancing tumor (ET), peritumoral edematous (ED), and necrotic tumor core (NCR). The light blue section is the healthy brain tissue.

The whole data set consists of 2000 studies, the most extensive data set for the BraTS challenge since its start in 2012. It is a significant size upgrade from the 2020 data set, which was the previous largest BraTS challenge with 660 studies. All image scans were pre-processed, which involved co-registering a template shape, interpolating to voxel resolution (1x1x1 mm), and skull-stripping. The scans and labels for the segmentation task were made available as NIfTI files

only.

This data set was chosen for our project for mainly two reasons. It is a crucial topic and a serious, important medical problem by virtue of the glioma cancer's often grave prognosis. Secondly, it was chosen for our project because of its high quality. All 2000 data points are co-registered multi-channel images and it is one of the most extensive publically available data sets of its type, saving us from the much pre-processing needed.

4.2 Deep learning libraries

All our experiments have been conducted using an existing extension combining the two deep learning frameworks, `fastai` and `MONAI`. Both of these frameworks are based on PyTorch, which is a commonly used open source DL framework that acts as a continuation of the deprecated ML library `Torch`. PyTorch is implemented in Python, while Torch again is written in the programming languages C and Lua.

4.2.1 `fastai`

`Fastai` is a deep learning library built on top of PyTorch that is organized around two main goals: to be rapidly productive while being *hackable and configurable* [56]. It enables a streamlined production of state-of-the-art performing deep learning models. Additionally, by being structured around a layered API, `fastai` separates low-level components from data scientists, and production is achieved in few lines of code compared to similar libraries. This enables data scientists to build and test many different models fast in reproducible, reusable code explainable from a theoretical standpoint. An example of a powerful `fastai` method is learning rate schedulers, which implement the two variants fit_one_cycle and fit_flat_cos (see figure 4.3). These make specific changes to the learning rate during training, associated with the concept of super-convergence [57]. `Fastai` encourages scientists to modify and intertwine the library with added code to explore solutions to new domains. The extension explained in 4.2.3 exemplifies such a library.

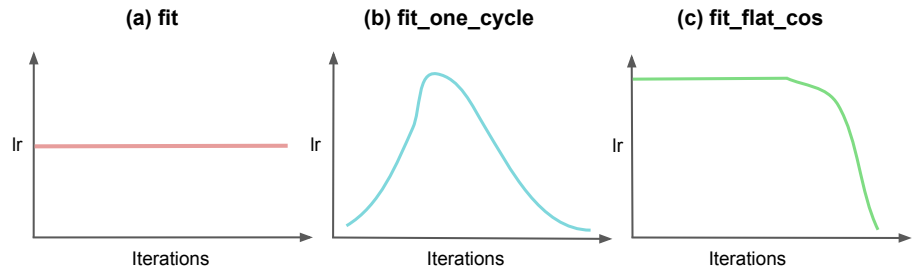

Figure 4.3: Illustration of the learning rate schedulers `fit_one_cycle` with the super-convergence phenomenon [57], and `fit_flat_cos` which is based on the same theorem - except it is built for the *ranger* optimizer. In short, the paper makes a case for varying the learning rate between two bound values in a cyclical pattern in order to optimize the learning rates and prevent getting stuck at a sub-optimal, local minima (see section 2.2.1).

4.2.2 `MONAI`

`MONAI` is a community-supported deep learning framework optimized for medical imaging tasks. It originally began as a collaboration between Nvidia and Kings College London [58] with the goal to accelerate the pace of research and development for deep learning in medical imaging. Medical data formats such as DICOM and NIfTI (see section 3.1) require specific support to be handled properly along with their meta-data (e.g., voxel spacing, orientation). Data augmentation, sample size limitations, and data formats are domain specific relating to medical imaging and based on PyTorch, `MONAI` provides the functionality and support methods needed to address these problems. Certain ANN architectures are highly suitable for biomedical applications, and `MONAI` provides a range of such architectures e.g. UNet [32], and variations of it.

4.2.3 Library extension combining `fastai` and `MONAI`

At MMIV, there exists an in-house library combining the `MONAI` framework (section 4.2.2) with the `fastai` library (section 4.2.1). The aim is to provide the advantages of `fastai` with functionality supporting biomedical applications (see figure 4.4). In short, the library utilizes multiple state-of-the-art techniques in the area of 3D medical image analysis like the powerful `fastai` functions learning rate finder, learning rate schedulers, and the benefit of low code through a layered API. This is then combined with the functionality required to tackle challenges in the domain of medical imaging.

A paper is scheduled to be released detailing this exciting library later this year.

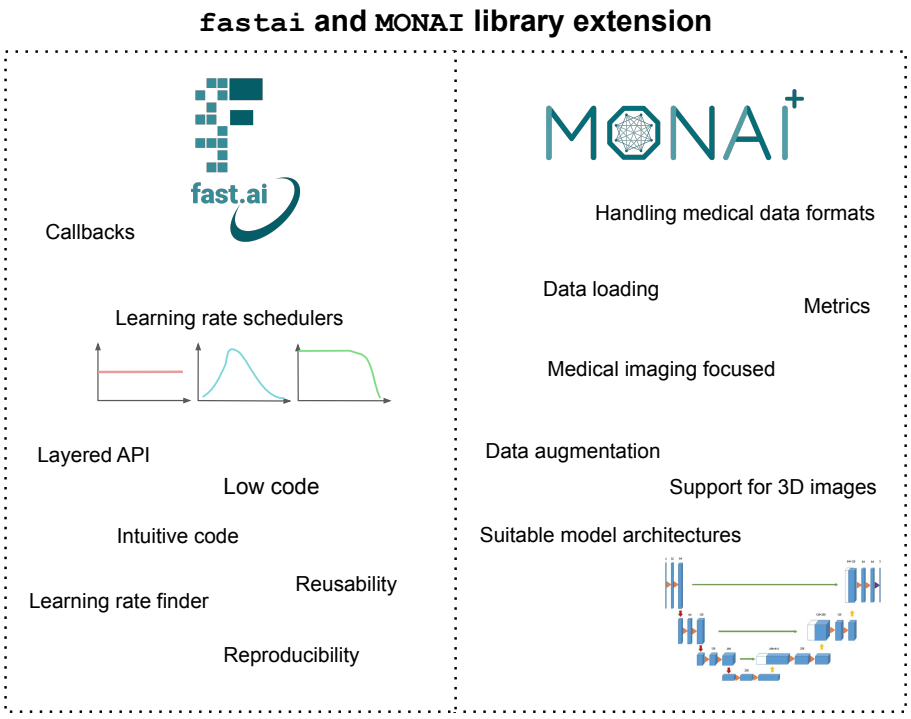

Figure 4.4: Overview of the main components that has motivated creatition of the library used in all three experiments that extends `fastai` and `MONAI`. It is important to note that highlighted components are case-specific. E.g., fastai provides great functionality for data loading and data augmentation, but not in the case of 3D medical images. Figure of U-Net is from [34], with permission from the author.

4.2.4 nnU-Net

nnU-Net (short for "no-new-net") is a segmentation framework intended for the medical domain that dynamically adapts itself to generalize on any given data set, and is used solely in experiment three for comparison with the `fastai` and `MONAI` library. It builds on the original U-Net architecture, claiming that "vanilla" U-Nets consistently underperform on benchmarks due to sub-optimal design choices and that a fully optimized U-Net can still yield the best result. Since its release and dominating display regarding performance at the Medical Segmentation Decathlon challenge in 2018, the nnU-Net framework has improved and continues to demonstrate its robustness. The framework won the test phase of the BraTS2020 competition, and a slightly modified version of nnU-Net also won the same competition in 2021. A general overview of nnU-Net's automated configuration is provided in figure 4.5.

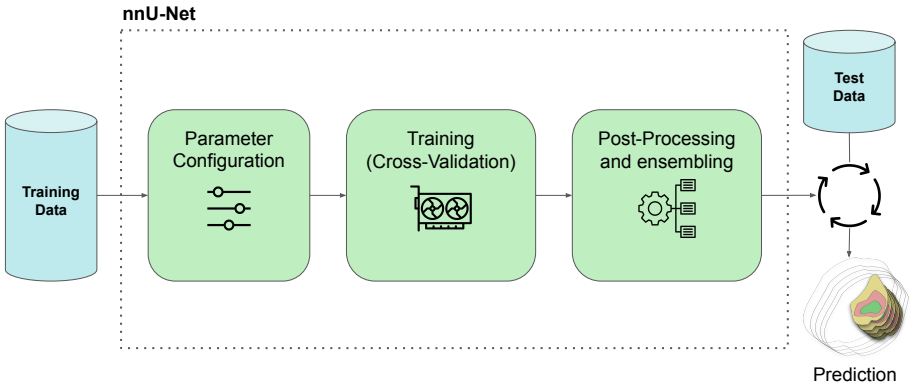

Figure 4.5: Simplified figure inspired by figure from Isensee et al. [59], illustrating the basic components of the nnU-Net framework. During parameter configuration, some *blueprint parameters* are initially set, these are *data-independent* design choices including the loss function, architecture template and some basic data augmentation. Then, another set of parameters gets inferred automatically based on the provided data set, called *data-dependant* hyperparameters. This includes network architecture configuration and batch size. After this, three different type of architectures (one 2D, two 3D) are trained in five-fold cross-validation (see figure 2.5). Finally, the best performing model ensemble is selected, and a check to observe if post-processing provides increased performance is run.

4.3 Experiment 1: Creating a simple and interactive web application

The focus of our first experiment was on putting the framework extension to use and testing it in a simple deployed application, eliminating much of the complexity related to ML infrastructure (shown in figure 1.1). Figure 4.6 illustrates the simplification of this experiment's system design and the elimination of specific infrastructural components. In software development terms, the first experiment is comparable to a component test, while the second experiment (described later in section 4.4) is comparable to an integration test. Additionally, experiments such as this let developers become more familiar with development processes in medical imaging. By utilizing the library combination from 4.2, in complement with other deployment-related frameworks described below, we hoped to answer research question 1 in section 1.1.

Another secondary point of interest was to see how a segmentation model trained with the `fastai` and `MONAI` framework would run on a CPU. When a model is deployed, it is often limited to running on the CPU instead of a GPU based on the target system's available resources (as was the case in this experiment). Limited computational power as a consequence of running on the CPU can increase latency, which affects a clinician's user experience and may reduce the usefulness of the application.

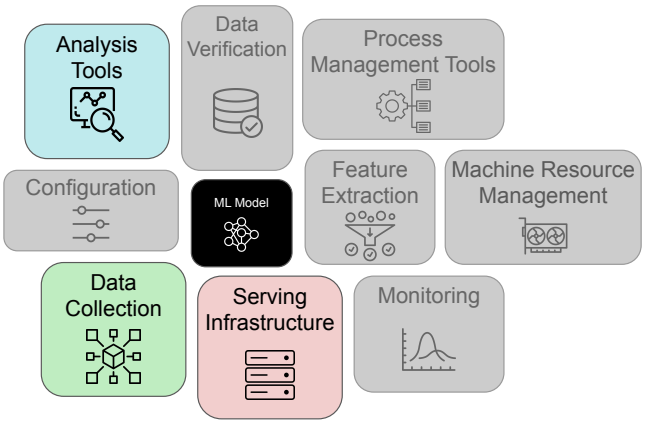

Figure 4.6: The same illustration as figure 1.1 in the introduction, but here showing the simplification of experiment 1's ML application. Only the colored boxes and the model are left in the system. Data collection represents data uploaded through a widget, analysis tools reflect the user interacting with the application, and serving infrastructure stands for Voilà.

4.3.1 Method

The result was a proof-of-concept application, which could be used to facilitate communication among people interested in practical brain tumor applications.

This experiment used the python library Voilà, which is built on Jupyter Notebooks and IPython widgets as technologies to create an application for easy model deployment. Jupyter Notebook is a web-based environment for creating Notebooks, which are essentially lists of cells that can contain code (most commonly python), text, or interactive media. Jupyter's purpose is to "support interactive data science and scientific computing across all programming languages" [60], and it is used by large companies like Netflix for working with data [61].

Voilà

Voilà is a library that converts Jupyter Notebooks with widgets (see next section about `ipywidgets`) into an interactive, stand-alone application. Voilà also allows deployment of these applications, and can further be hosted on a cloud service provider such as Binder[1] or Heroku[2], making the application easy to share and test.

`ipywidgets`

Ipywidgets are HTML-based widgets for Jupyter Notebooks, making the Notebooks interactive. The core widgets include sliders, buttons, text fields, and progress bars.

[1] `https://mybinder.org/`
[2] `https://www.heroku.com/`

4.3.2 Results

As explained in section 4.2, all experiments, including this one worked with the `fastai` and `MONAI` combination library. A brain tumor segmentation algorithm was trained with this library and the resulting model was exported.

The interactive application was based on a Jupyter Notebook that handled loading data, calling the trained model for prediction and visualizing the predicted mask on top of the brain.

Figure 4.7 shows the proof-of-concept application. As a prototype, the user is prompted with an upload button. The user chooses a directory containing four image sequences, which the model needs to predict and produce the mask output. When the user clicks the button labeled *Predict*, the application locates these files within the directory and feeds them as input to the deep learning model.

When finished predicting, the application produces a visualization of the input brain with the predicted mask overlayed. Since it was a three-dimensional image, a slider was added to browse through the image.

Initially, the application only showed the axial slice, but after feedback from researchers looking at the application, we added the coronal and sagittal slices.

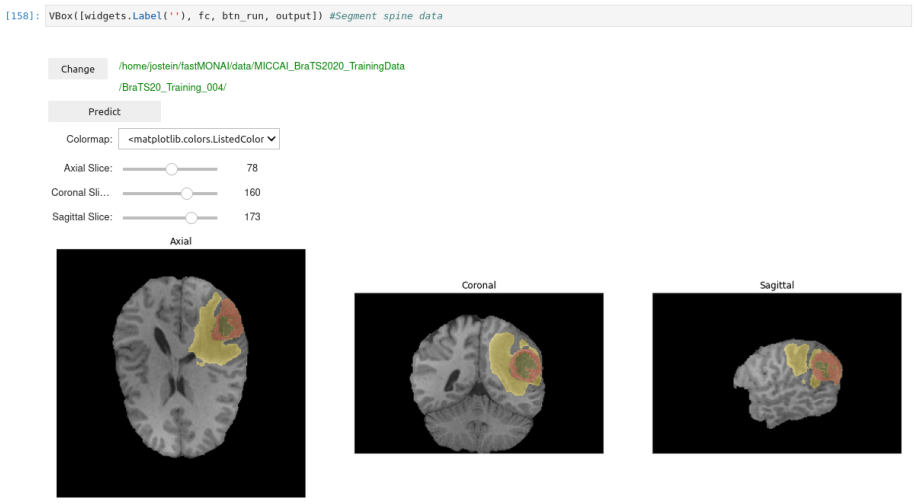

Figure 4.7: Screenshot of deployed Voilà application. Above the displayed brain scans and segmented mask are the Notebook widgets: Buttons for uploading and predicting, and sliders to look through the displayed 3D image.

4.4 Experiment 2: Research PACS integration

Our experiment regarding machine learning deployment in research PACS is the most important experiment of our thesis, aimed at providing major contributions to the development of the workflow-integrated machine learning infrastructure in Helse Vest. Our approach for this experiment is to integrate a

deep learning segmentation model into the existing research PACS platform at Helse Vest, hoping to uncover its limits and possibilities. The focus was not on developing the most robust deep learning algorithm but on deploying the algorithm and testing the infrastructure.

The research PACS system, described in section 3.2.2, is the target environment for the segmentation application (figure 4.8 shows how the application fits into research PACS). Research PACS is currently used by researchers at Helse Vest RHF, parallel to being further developed and tested. By deploying the application in this system, we wanted to help the testing of the research PACS-machine learning pipeline. As a pilot project inside WIML, the project hopes to find what works, what does not work, and if any necessary features are missing. The deployment let us build a list of requirements, as the aim of research question 3 previously described in section 1.1.

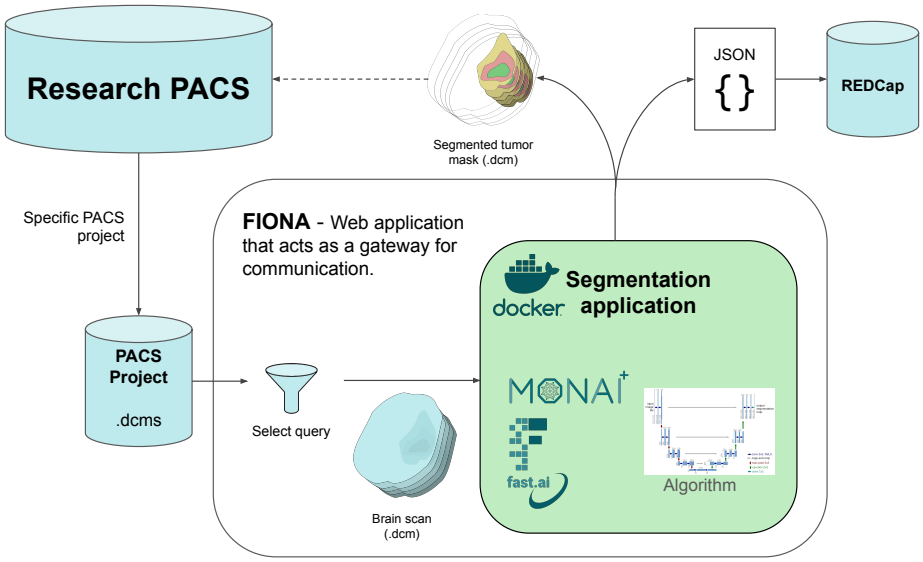

Figure 4.8: Visualization of how the project's segmentation application (the green box) fits into the WIML image data analysis pipeline. FIONA acts as a gateway for data query and processing. DICOM files of brain scans from a specific project within research PACS are fetched and sent to the deployed application. The resulting output after executing is a segmented tumor mask and a JSON file information related to the task. These are stored back into research PACS and REDCap respectively.

The task of brain tumor segmentation is a real problem and thus a good case for testing the system. Additionally, the fact that the data is multimodal and multilabel introduces extra factors that make them not as easy to handle.

4.4.1 Method

Like the other experiments this experiment used the `fastai` and `MONAI` libraries described in section 4.2 to train the segmentation model. However, in this experiment, the model performance was not the focus. The model was trained

for just a single epoch, but the importance was that it acted as a placeholder for a better model trained in the same framework. This unoptimized model was exported with `fastai` and saved just like how a more optimized model would be.

For the experiment, some image files had to be uploaded to the research PACS through FIONA for testing, and this system required these to be DICOM files. The DICOM files were generated from NIfTI-files of the BraTS 2021 data set, the same data the model was trained on. The conversion from NIfTI to DICOM was done using a medical image conversion toolkit called XMedCon [62]. By generating files using the data from the training set, the data would be in the expected form for the model and therefore could perform predictions on the incoming it without doing image registration. The data conversion step is specific to the data sharing format used in the BraTS 2021 data set and is not a step usually required during the application of machine learning models on medical images, which are always accessible in DICOM format.

4.4.2 Results

Using the ror tool

For turning the segmentation application into a workflow suitable for research PACS, we used a software development tool called *ror*, created by Hauke Bartsch.[3] This tool is designed for developing workflow applications fit for research PACS, written in Go [63]. It provides several features for creating and integrating applications as workflows in the research system, made available through these commands:

- `ror init`: Initializing workflow application structure.
- `ror config`: Find suitable local data.
- `ror trigger`: Triggering and running the workflow. This is simulating how arrival of new data triggers the workflow to run in research PACS.
- `ror build`: Build a container with the application and its dependencies.

We used ror to initialize the project with a python template for project structure. We added a script that imports our segmentation model to this template. For the application to work with our DICOM data, we used `ror config --data` to specify from which directory to fetch them. Additionally, we used `ror config` to write a request in the research system's *select* query language. The select language is a domain-specific, SQL-like query language designed in-house for the research PACS system, and it can filter data based on various DICOM tags. A typical use case is to select a DICOM series based on an identifier or select all series of the same modality. In our case, we need a somewhat complex select string that returns the four different image sequences together and separates them from any other image sequence in the same DICOM study. In the data generated for this experiment, each image sequence is identifiable by its `SeriesDescription` DICOM tag. This select string is shown in the code below, using the `SeriesDescription` tag.

[3]The tool is documented in this GitHub repository: `https://github.com/mmiv-center/Research-Information-System/tree/master/components/Workflow-Image-AI`.

```
Select study
    from patient
        where series named "T1w" has
            SeriesDescription = "T1w"
    also
        where series named "T2w" has
            SeriesDescription = "T2w"
    also
        where series named "FLAIR" has
            SeriesDescription = "FLAIR"
    also
        where series named "T1wCE" has
            SeriesDescription = "T1wCE"
```

Above is the ror select string used for querying a study of four image series by their `SeriesDescription`. The sequences are returned together.

Deploying into the research PACS

Uploading the containerized application was done through FIONAs existing web interface for workflows, see figure 4.9. This page lets developers at Helse Vest RHF interact with FIONA by uploading containerized applications made with ror and running them with data from a specific research PACS project. To gain access for uploading, the user needs to generate a token connected to a research PACS project, which lets them query data from the specific project.

We uploaded the container successfully through this page (shown in figure 4.10). Using the same select string from the code snippet earlier, we queried for data in the research PACS project and ran the workflow on returned data. The segmentation output from the application was sent back to the research PACS ready for evaluation (shown in figure 4.14). A JSON file with values related to the tumor volume was also outputted, and sent for storage to REDCap, the electronic medical record software [64] (see figure 4.12).

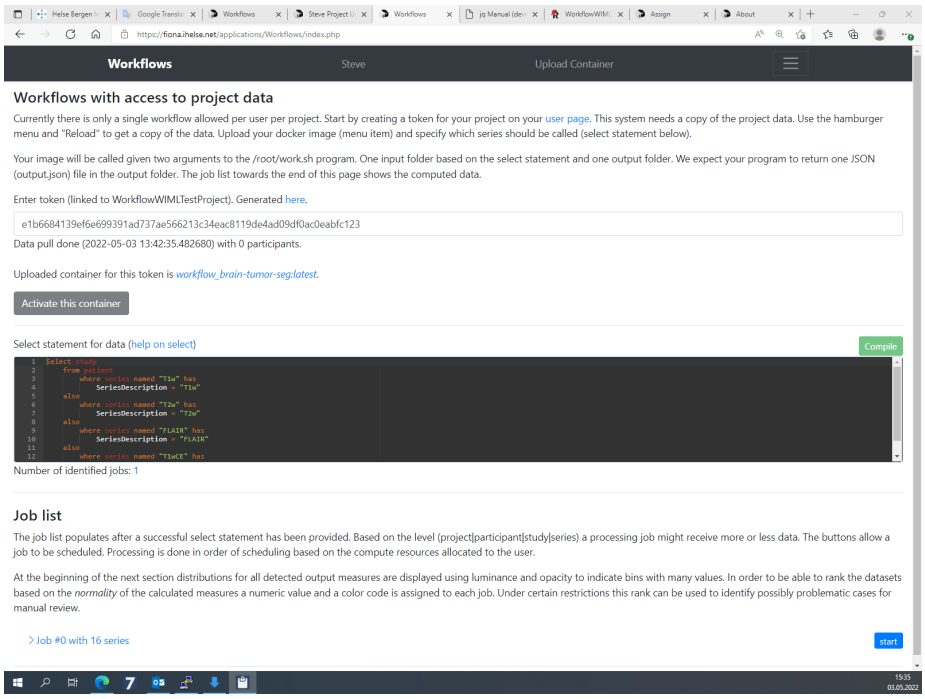

Figure 4.9: Screenshot of FIONA's main web page for uploading and using workflows. Before uploading a container, the user needs to be authenticated, so for first-time use, a token is requested and generated, as explained on the page. This token needs to be entered in the input section every time before getting access to the system. Once logged in you can upload a new workflow (a Docker image). We uploaded it through the "Upload Container"-button, which prompts with a new interface (see figure 4.10).

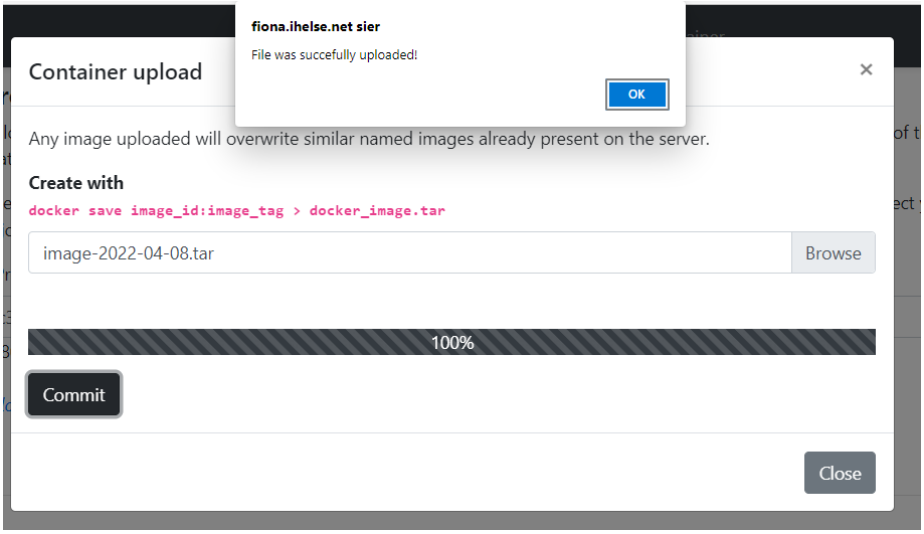

Figure 4.10: Screenshot of the "Container upload" prompt. Shown is a successful upload of our Docker image, made by compressing it into a .tar-file and saving it with `Docker save` before being uploaded to FIONA as a workflow. The next step in testing the deployed application is to retrieve data from research PACS to be processed. See figure 4.11.

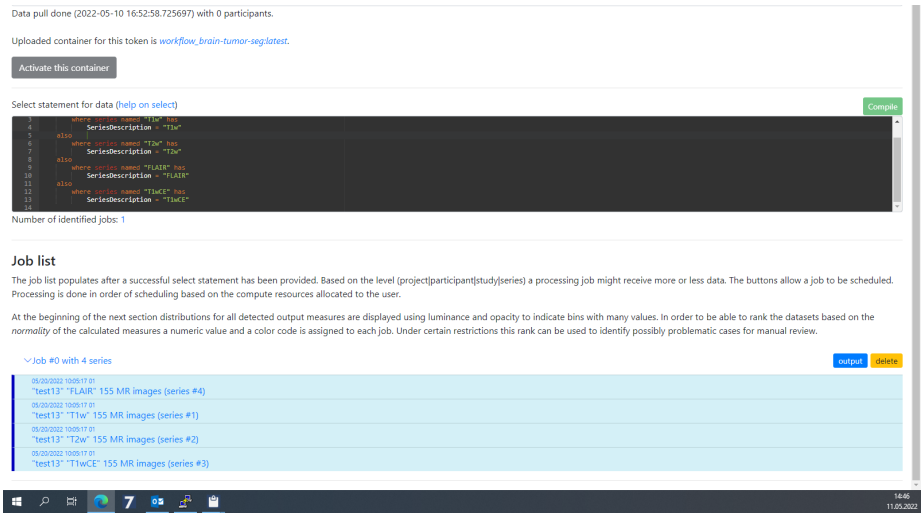

Figure 4.11: Screenshot of returned jobs to be processed in FIONA. For FIONA to know which data to retrieve, a select query is entered (see code snippet earlier in section 4.4.2 for our multi-modal select query). It is compiled and run by clicking the green "Compile" button. This screenshot shows the returned data matching the select query string, retrieved from the research PACS project linked to this workflow. The workflow is executed on the fetched data when clicking the "output" button.

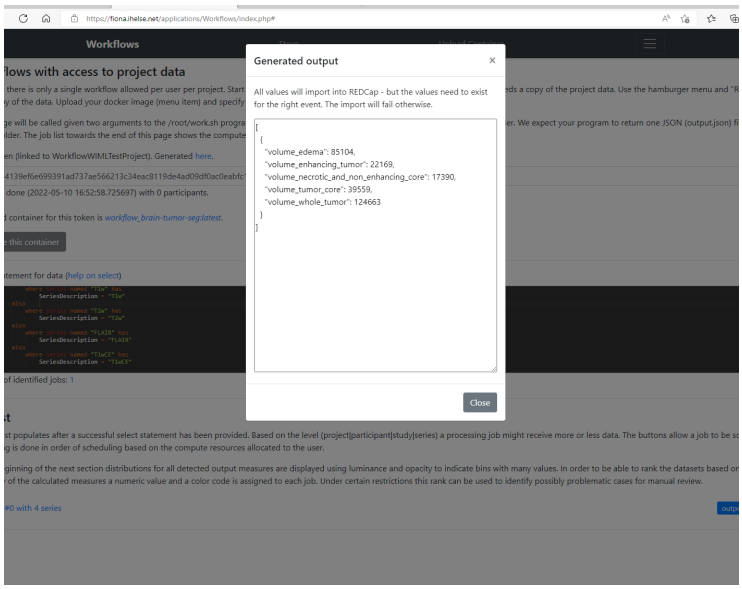

Figure 4.12: The user is prompted with some numerical values, which FIONA expects as output from a workflow. These are sent to REDCap and stored there. As shown in the screenshot, our application outputs the tumor volume of the predicted mask, calculated in cubic millimeters. This information is valuable for a radiologist, as tumor volume plays a part in diagnostics (as mentioned in section 2.1.2).

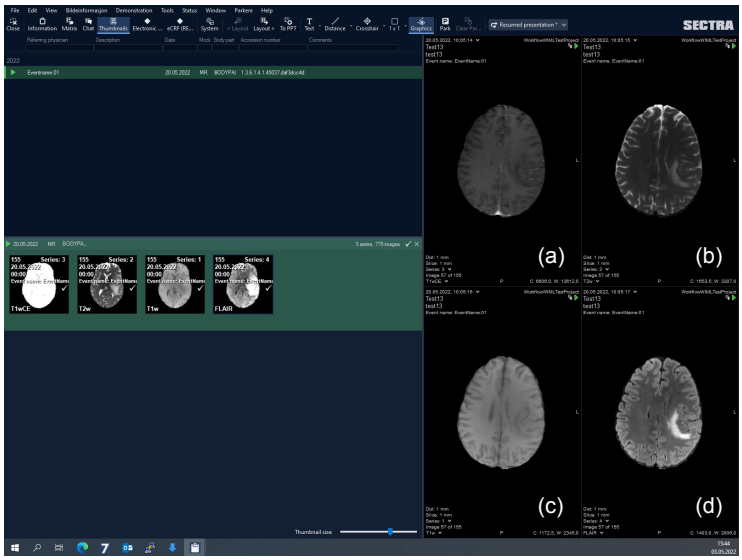

Figure 4.13: Screenshot of scans in the Sectra research PACS software, the same software as clinical PACS uses. Sectra is a secure medical software program, and due to its closed API, FIONA is needed to communicate with research PACS. The different image scans are: a) contrast-enhancing T1, b) T2, c) T1, and d) FLAIR.

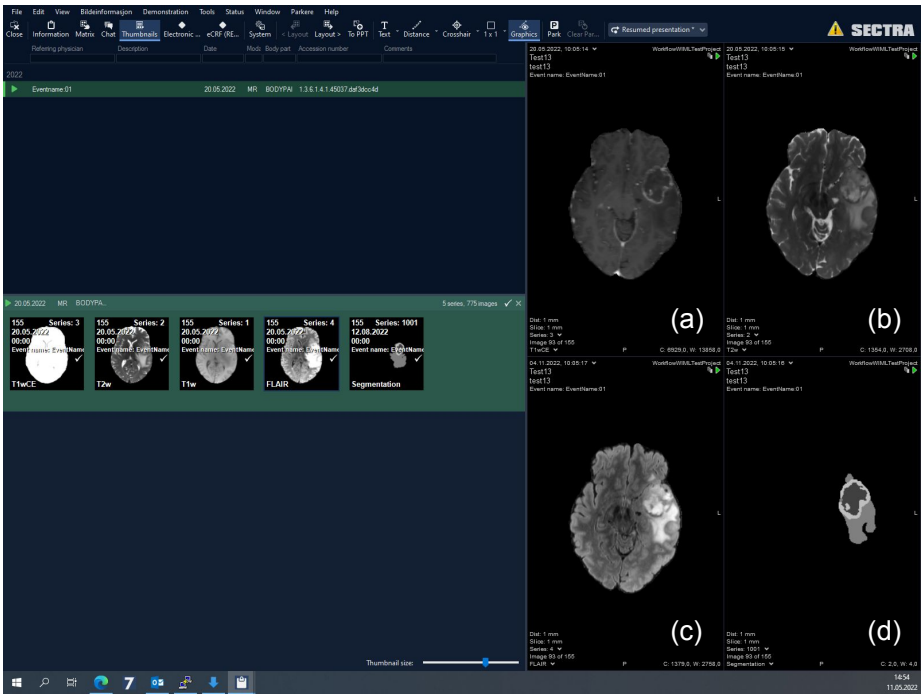

Figure 4.14: Screenshot of scans in the Sectra research PACS software, after the deployed application had been executed through FIONA. You can see the predicted tumor mask next to the original brain scans. a) Is contrast-enhancing T1, b) is T2, c) is FLAIR, and d) is the predicted mask. (This is the same brain scan as in figure 4.13, but a different axial slice). This screenshot shows the result of a fully functional machine learning workflow integrated with a PACS system.

Generating valid DICOM files

The task of creating valid DICOM files from NIfTI files is not as trivial as implied in the introduction of this experiment. Research PACS and FIONA have several metadata requirements for what information needs to be stored in the DICOM tags. The information in our NIfTI files is not sufficient to automatically create DICOM files with all required tags. As described in section 3.1 about medical image formats, NIfTI files, more specifically NIfTI headers, do not contain the same volume of information as DICOM files do. When generating DICOM files from NIfTI with XMedCon, it has to somehow come up with values for the DICOM tags which aren't present in the NIfTI files. It does an OK job at coming up with these values, but we found that it was not enough for the generated files to be compatible with research PACS.

This meant that additional information needed to be added to the generated files. To modify the generated DICOM files, we used the python library pydicom [65].

Below are some examples of modified tags:

- `StudyInstanceUID`: The four sequences from the same scan needed the

same study ID, preferably following DICOM standards for ID.

- `SeriesInstanceUID`: Each series needed a unique series ID.
- `SOPInstanceUID`: Each slice in a series needed a unique ID. `MediaStorageSOPInstanceUID` also needed to be changed to match it.
- `StudyDate` and `StudyTime`: These tags were missing, which caused an error within research PACS.
- `SeriesDescription`: There was no way to identify the sequence type of each image series. This was added in the series description tag.

After adding these tags, the DICOM files were fit for upload to research PACS. The above additions are mostly required because the four image series were treated by XMedCon as individual image series. For the target workflow, they needed to be added to a single DICOM study.

4.5 Experiment 3: Comparing `fastai` and `MONAI` with nnU-Net

The focus of our third experiment was to compare the soon-to-be-published library extending `fastai` and `MONAI` with an existing framework like nnU-Net, already established as state-of-the-art.

By developing this segmentation model, we wanted to illustrate something on par with what is considered the best to date, especially in the context of having such a deployable model in the research PACS system, as described in section 3.2.2.

Our approach to this experiment was to utilize the same composition of training and validation data for comparison across both frameworks. Although we did not expect our `fastai` and `MONAI` model to outperform the one that just won BraTS2021 (see section 4.1) at that particular data set, we hoped to see a comparable model. If the model, especially with less training compared to the computationally heavy nnU-Net, is comparable, the outcome would serve as a motivating factor for the result itself and the possibilities of the potential usefulness regarding the `fastai` and `MONAI` library moving forward.

Brain tumor segmentation makes a particularly good case for comparison. The reason for this is that in addition to being a complex, real problem, there is a vast amount of available, high-quality labeled data.

4.5.1 Method

As an extension of this experiment, the library combining `fastai` and `MONAI` has been applied in all experiments, however, here the focus is solely on the use of this library, and its comparison to the nnU-Net framework.

In this experiment, we used the BraTS 2021 data set (see section 4.1), Python within the Jupyter Notebook environment for development (see 4.3.1), the library combining `fastai` and `MONAI` and the powerful nnU-Net framework (see 4.2.4).

Network architecture

Following the nnU-Net paper's strong case for "taking away superfluous bells and whistles of many proposed network designs" by fully focusing on optimizing a "vanilla" U-Net, we used the simple MONAI UNet architecture. Similar to the nnU-Net architecture, it is close to a "standard" U-Net with slight modifications, such as implemented residual units (see section 2.2.3).

Experimental settings

Table 4.1: Comparison of `fastai` and `MONAI` library combination's and nnU-Net's training setup

	`fastai`+`MONAI`	nnU-Net
Epochs	100	1000
Model Architecture	`MONAI` U-Net	nnU-Net Generic U-Net
Layers	5 Layers	2 Layers
Activation function	Parametric ReLU	Leaky ReLU
Data augmentation	random flip, random zoom, random affine	random scaling, rotation, gamma, mirror, elastic transform
Optimizer	Ranger	SGD with momentum
Postprocessing	None	None
Batch size	2	2
GPU	Nvidia Tesla V100	Nvidia Tesla V100

4.5.2 Results

For training, we used k-fold cross-validation with k set to five (see figure 2.5). Although the BraTS2021 dataset contains comparatively many labeled data point, there is still only 2000 case studies, of which 1251 are reserved for training purposes. Therefore, it is crucial to get the most out of the available data, and k-fold cross-validation is a technique assisting that. Moreover, within the nnU-Net framework, five-fold cross-validation is a hard-coded feature making it necessary to implement it for comparison as well.

During training, when the model has completed an iteration through all training elements, we say that it has run through one *epoch*. As shown in table 4.1, our nnU-Net model was trained with 1000 epochs, compared to our `fastai`+`MONAI` model trained with 100.

Fabian Isensee et al. [59] use an optimized learning rate tailored for 1000 epochs in nnU-Net, something hard to match in our case of training with 100 epochs because it requires quite extensive research in itself to find such a learning rate. Usually, one does not already have access to an optimized learning rate before training, this is why, as a replacement, we utilized `fastai`'s learning rate finder.

After using the suggested learning rate from `fastai`, the model was trained using the state-of-the-art deep learning optimizer ranger, on top of `fastai`'s

learning rate scheduler `fit_flat_cos` (see figure 4.3) based on Leslie Smiths paper on super convergence [57].

Table 4.2: Validation results for nnU-Net folds. Dice score

	NCR	ED	ET
nnU-Net folds			
Fold 0	**0.8127**	0.8598	0.8673
Fold 1	0.8119	0.8763	0.8733
Fold 2	0.7948	**0.8808**	0.8867
Fold 3	0.7792	0.8710	**0.8887**
Fold 4	0.7713	0.8740	0.8606
Ensemble	0.7940	0.8724	0.8753

Table 4.3: Validation results for `fastai` + `MONAI` folds. Dice score

	NCR	ED	ET
`fastai` + `MONAI` folds			
Fold 0	0.7932	0.8325	0.8600
Fold 1	**0.8041**	0.8534	**0.8782**
Fold 2	0.7760	**0.8575**	0.8701
Fold 3	0.7702	0.8486	0.8772
Fold 4	0.7426	0.8490	0.8687
Ensemble	0.7772	0.8482	0.8708

We have evaluated the models based on the common segmentation metric dice score (see section 2.2.3). As seen in table 4.3 and 4.2, the nnU-Net models does outperform the `fastai` and `MONAI` model slightly. Our evaluation behind the discrepancy in performance is that it lies mainly in the difference between the number of epochs (with an optimized learning rate), and perhaps data augmentation methods. Figure 4.15 shows some visual examples of the segmentation results from both models compared with the ground truth.

The time spent training these models differed greatly from the total training time for the nnU-Net model clocking 225 hours (45 per fold) as opposed to 90 hours (18 per fold) for `fastai`+`MONAI`.

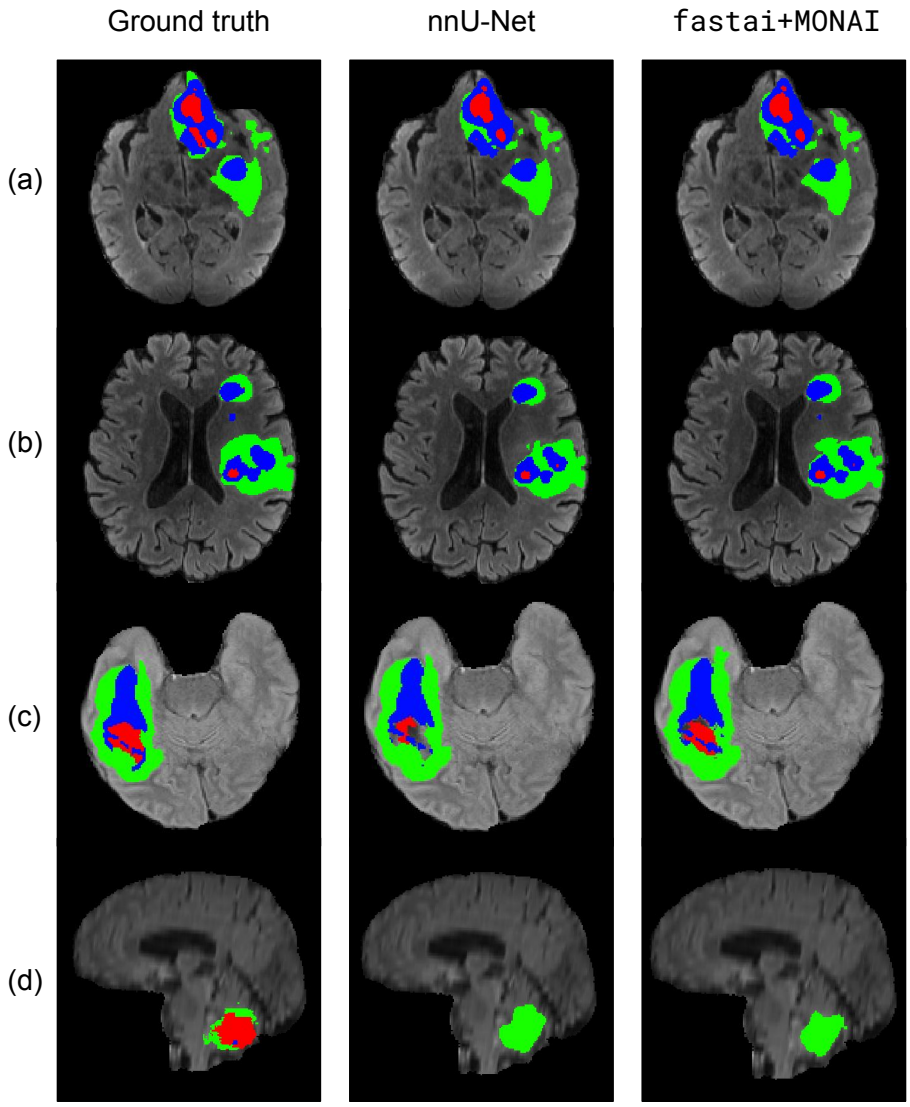

Figure 4.15: Ground truth and segmentation results from the nnU-Net model and the `fastai+MONAI` model for training subjects. The nnU-Net performed better at a) and b), while c) is an example where the `fastai+MONAI` performed best. Both models failed at predicting d) correctly. As shown in the section above, nnU-Net performed best overall. (Subject numbers in BraTS 2021 are: a) 00477, b) 00026, c) 01397, d) 01530).

Chapter 5

Discussion, conclusion, and further work

5.1 Discussion

This chapter consists of sections discussing the research questions established in section 1.1 and their corresponding experiments.

5.1.1 Discussion experiment 1

Research question 1 from section 1.1 was: "Is it possible to create and integrate a generic setup for easy model deployment within the existing `fastai` and `MONAI` framework extension?".

In this study, we demonstrated our development of a simple application, based on Jupyter Notebooks, ipywidgets, and the Python library Voilà. We trained a model with the library combining `fastai` and `MONAI`, exported it with `fastai`'s built-in methods, before integrating it in a Notebook with ipywidgets. We found the chosen deployment method to be a fairly uncomplicated process and the resulting application was simple, fast, and practical.

The first experiment regarding Voilà and model deployment gave us as developers practical insight into working with medical images and our selected technologies. The development also gave us a frame of reference for how a machine learning model might fit into an actual application.

We presented the Voilà application to an interdisciplinary group of researchers with expertise in the fields of machine learning, medical imaging, and medicine, among others. We felt that the application generated enthusiasm, both for us having realized the concept and for potential additions to it. Although it was not a focus of our research, the reactions from the presentation brought back some points from section 2.3.2 about trust. Building trust between clinicians and developers is crucial to a successful partnership between the two. For that to happen, clinicians and developers must establish a healthy and open way of communicating when introducing new AI tools. Our experience was that a

simple, intuitive application meant as a helping tool for radiologists sparked positivity among members of our group. This feedback may indicate the importance of a continuous and productive dialog in the context of ML-workflow integration in medicine, and a generic environment for model deployment as an arena where developers can gain trust from radiologists.

5.1.2 Discussion experiment 2

Research question 2 from section 1.1 was: "How can we integrate our trained models directly with the existing research PACS in Helse Vest? Are additional software development efforts and hardware for the research PACS needed?".

In this study, we have also demonstrated our development process and integration of an ML model in a clinical-like system: the research PACS infrastructure. Looking back to the research question, we wanted to investigate how to implement a model and if there were any further software development efforts required. Our results showed that it was possible to integrate it as a containerized application without much additional work needed to research PACS. The application executed a prediction while saving the results back to research PACS.

Using the ror-tool in development proved valuable for local testing and saved us time by checking the application for errors before deployment. A few bugs related to ror were discovered at this stage, some of them regarding the image data being multi-modal instead of a single channel. One of these bugs was related to the select query, which was meant for grabbing images by SeriesDescription. At the start, querying ignored the SeriesDescription tag and seemed to fetch images in an arbitrary order. This was a problem for our model, as it required the different image scans to come in a fixed channel order. Another bug caused errors in the loading of images, as only some of the slices in the DICOM image were loaded, causing the model to fail due to the 3D brain scan being incomplete (see figure 5.1). These are some examples of errors that were fixed before the system was functional and ready for testing.

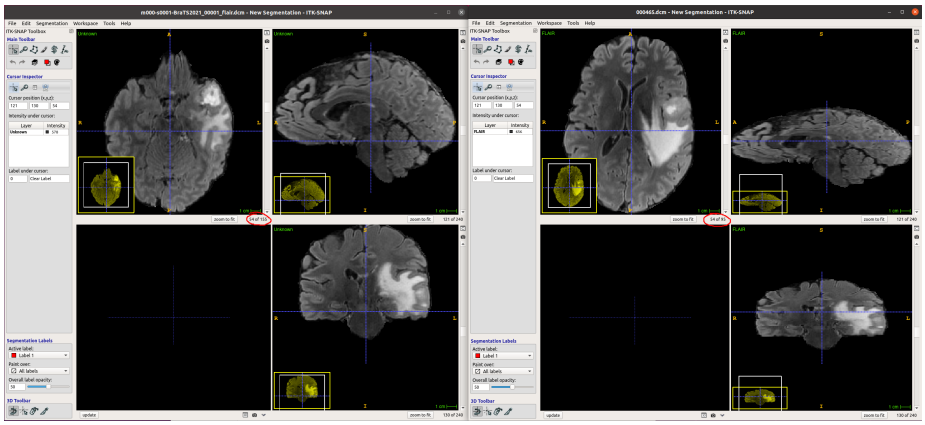

Figure 5.1: Screenshot of the original brain to the left and the “squished” brain to the right. This error occurred after ror loaded the image into our application, but this bug was fixed after we discovered it. Image visualized with ITK-snap [66]

Some bugs appeared first after the application was deployed for testing in FIONA. One of these bugs was related to how FIONA handled storage and cache when data was re-uploaded. The select query would not fetch the expected data from research PACS, due to the cache containing old versions of the same data. The cache would not update when re-uploading data. After this was fixed, a new bug related to the cache appeared. The cache did not get cleared when new data was uploaded, causing image duplicates in the archive (see figure 5.2), which again caused unexpected query results in FIONA. While discovering and fixing bugs, it became apparent that FIONA’s software architecture made it hard to maintain. Modernizing its architecture and making improvements regarding software architecture and technology choice could make the system more maintainable for future additions.

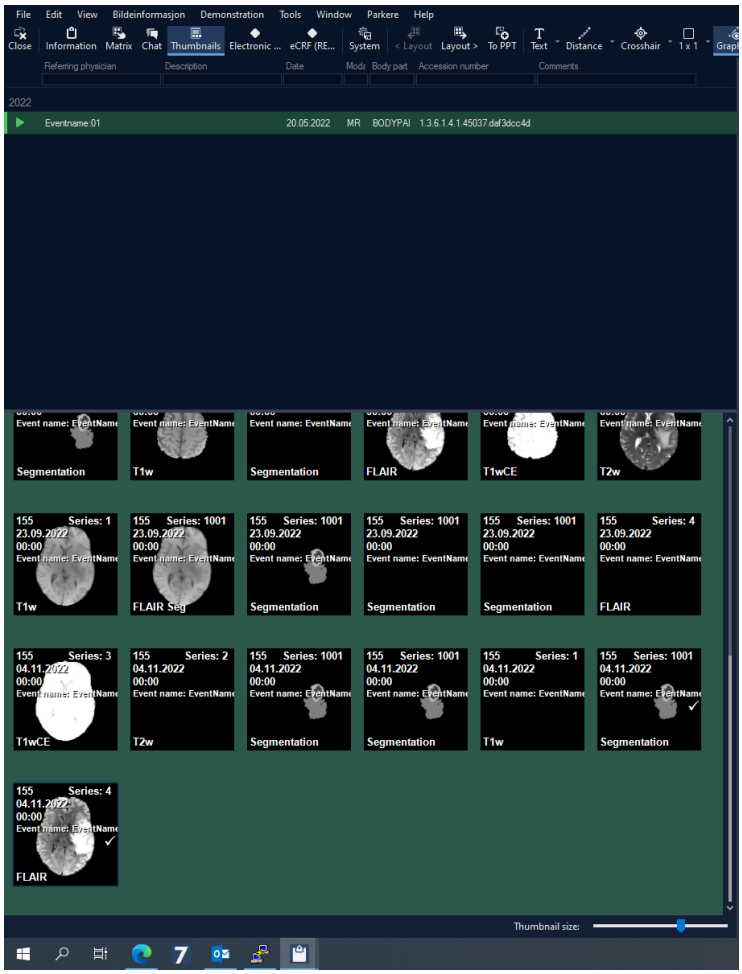

Figure 5.2: Screenshot showing image duplicates appearing in the research PACS' Sectra front-end. These image duplicates also caused unexpected query results in FIONA when triggering the workflow.

In addition to fixing bugs, new features were added to the workflow pipeline. In FIONA, saving DICOM images back to research PACS was not yet implemented as a feature. Our application segments the mask and creates a new DICOM file containing this mask. FIONA was adjusted so it could store images back.

By testing the research PACS pipeline and deploying a multi-modal deep learning algorithm specifically, we clear the way for future projects that want to deploy a model inside research PACS with not only similarly composite multi-modal data but also with single modality data. Our project makes progress toward clinicians using ML-based tools, but this pilot project also facilitates future developers to upload and use their ML models in PACS.

The program in its current state still lacks features that need to be properly implemented before it is production-ready. One of these missing features is pre-processing of new incoming data. The difference between how the data looks in

the experiment and how it will look in practice is contrasting. As described in section 4.4.2 we used the NIfTI files from the BraTS data set to generate the data used for our experiment. These files are already co-registered, interpolated to the same voxel resolution, and skull-stripped, saving our pipeline from needing these steps. However, when running on local data directly from the hospital scanner, our model would require pre-processing of the data for the input to be of the expected format. Although a pre-processing step would be necessary for a production environment, we chose to avoid this since our focus was to get the system up and running. Using the already processed data let us focus on the FIONA pipeline.

In conclusion, the deployment of our application uncovered a few bugs of different sizes, and a new feature involving sending back image DICOM files to the research PACS system was added.

5.1.3 Discussion experiment 3

Research question 3 from section 1.1 was: "How does `fastai` combined with `MONAI` compare to another cutting-edge deep learning framework (nnU-Net) regarding performance metrics?".

In this experiment, we have presented a segmentation algorithm based on a state-of-the-art architecture [32] with comparable results to the nnU-Net framework [59]. This was done by training a network with the locally developed, soon to be published library extending `fastai` and `MONAI` (see section 4.2.3).

Experiment 3 demonstrates that although nnU-Net is shown to slightly outperform the `fastai+MONAI` library regarding dice score, the metrics are comparable. As a part of a production-grade ML integration, although outside the scope of this thesis, further investigation into the `fastai+MONAI` library is required to get a clear picture of its potential in the clinic. We expect our dice score to increase with improved techniques regarding data augmentation, model architecture optimization, and possibly more epochs. We further believe that evaluating the models through different metrics such as Hausdorff distance, which is also used to evaluate BraTS 2021 submissions alongside dice score, would give more insight into the performance evaluation.[1]

Bugs were encountered within the `fastai+MONAI` library during model training. One of these bugs was related to printing metrics where the result regarding one entire tumor class was printed as NaN-values across all epochs. Some other bugs were caused by incompatibility issues due to ongoing refactoring within the library. We contributed to the development of the library by addressing these bugs through open discussion and cooperation with the library's author.

Setting metrics aside, the `fastai+MONAI` library is much easier to integrate new models in due to the layered API structure. The layered API enables intuitive low code with the possibility of advanced customization in deeper layers. In the context of experiments 1 and 2, we have demonstrated implementations of this promising library in addition to the comparisons made in experiment

[1] Note that by using metrics we can rank which algorithms perform better than others, but new research claims that some of the most common metrics used in training machine learning models might cause errors after deployment [67].

3. Conclusively, it has demonstrated the production and the result of a state-of-the-art comparable model intended for clinical-like systems. The resulting segmentation from the clinical PACS in experiment 2 highlights what such a model is capable of bringing to such a system.

5.2 Conclusion

In this thesis, we have integrated a model comparable to the state-of-the-art into Helse Vest's research PACS. Figure 5.3 illustrates the result and the system behind the integration. Our work opens up possibilities for future developers to continue workflow integration machine learning in PACS. We also found that the model is comparable, although not outperforming the state-of-the-art model made with nnU-Net in performance. However, the *fastai* and *MONAI* library combination we used trained faster and is more flexible for fine-tuning.

However, there are still many hurdles left before our deployed application is ready for practical use. It does not handle the pre-processing necessary for predicting raw clinical data. Furthermore, even after these technical hindrances are surmounted, there is still a lot of validation to be done before such applications could see clinical use. Additionally, one would have to consider various ethical challenges.

Additionally, our project found it to be possible to easily construct a simple application based on a `fastai` and `MONAI` model with the Python framework Voilà.

Our project cleared the way for future work with research PACS, and we see a lot of possible ideas for future work, discussed below.

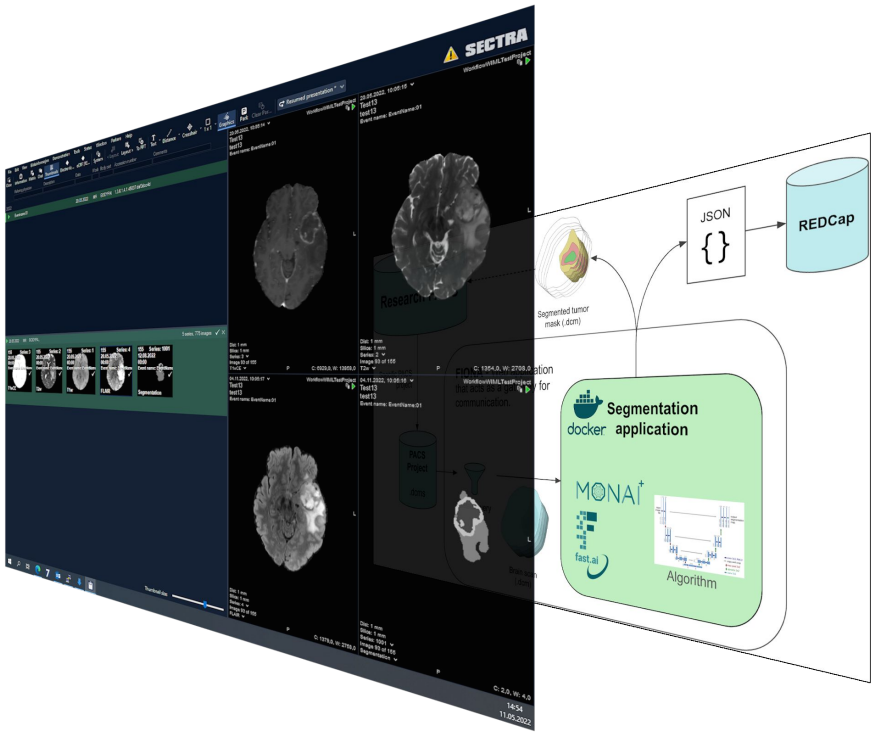

Figure 5.3: Figure showing the our application front-end results (from figure 4.14) and back-end structure (from figure 4.8). The figure summarizes the results of our work regarding research PACS integration.

5.3 Further work

Human-in-the-loop: A logical next step for the application is to create a *human-in-the-loop* system. When a radiologist uses the designed application, they could use the predicted mask. But if they disagree with the computer's prediction, they need to be able to redraw the mask or parts of the mask. In the current integration, this is not possible, but it is an idea for future work. If this feature is added, the new redrawn mask could be used for updating and retraining the model as in *online machine learning* and *active learning*. Figure 5.4 shows a possible human-in-the-loop cycle for workflow-integrated machine learning. Another point is the pre-processing and co-registering as mentioned earlier in 5.1.2. This is a requirement for loading and predicting new data. Possible solutions to solve registration could use FSL [68] or BraTS Toolkit [69].

Having first integrated a model into research PACS, the step to substitute this model for a better performing model, like one trained with nnU-Net, is not difficult. More work can be put into making the deployed model as robust as possible when facing new, unusual data.

Furthermore, since the application is in a Docker container, it comes with the advantage of being runnable in different environments. We think our application could be run in different vendor systems without much change needed. During our work, we have had some contact with the Healthineers group from Siemens and their syngo.via Frontier system can run Docker containers as prototype healthcare applications. This could be a possible next step for our work.

MONAI Deploy App SDK is another framework for designing and developing AI-driven applications in healthcare. It is an interesting project by MONAI under development and would support the deployment of our model trained with `MONAI`. This was a route we started working with, but due to MONAI Deploy App SDK still being in development and a few issues we met along the way, we did not continue down this path.

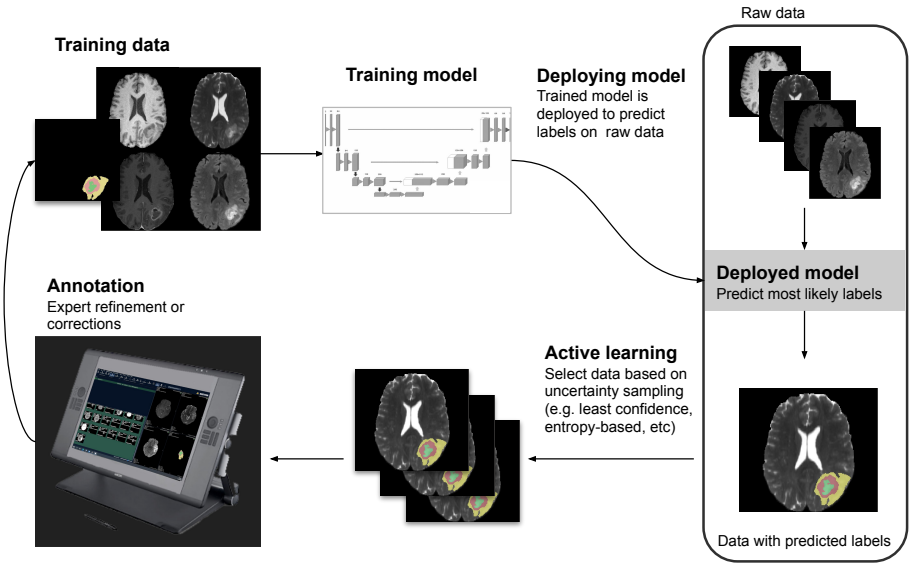

Figure 5.4: Illustrating a human-in-the-loop cycle, showing how a segmentation application could work in the future. After a model is deployed, it predicts on new raw data. An expert using the application would correct the predictions the model had the least confidence on. This new corrected data would be saved and used to retrain a model which is deployed. (Figure of training model is from [34], used with permission from the author).

Bibliography

[1] MMIV. *Workflow-integrated machine learning*. https://mmiv.no/wiml/. Accessed: 2021-09-01. 2021.

[2] Alexander Selvikvåg Lundervold and Arvid Lundervold. "An overview of deep learning in medical imaging focusing on MRI." en. In: *Zeitschrift für Medizinische Physik*. Special Issue: Deep Learning in Medical Physics 29.2 (May 2019), pp. 102–127. ISSN: 0939-3889. DOI: 10.1016/j.zemedi.2018.11.002. URL: https://www.sciencedirect.com/science/article/pii/S0939388918301181 (visited on Mar. 23, 2022).

[3] David Ben-Israel et al. "The impact of machine learning on patient care: A systematic review." en. In: *Artificial Intelligence in Medicine* 103 (Mar. 2020), p. 101785. ISSN: 0933-3657. DOI: 10.1016/j.artmed.2019.101785. URL: https://www.sciencedirect.com/science/article/pii/S0933365719303951 (visited on Apr. 7, 2022).

[4] Pranav Rajpurkar et al. "AI in health and medicine." In: *Nature Medicine* 28.1 (Jan. 2022), pp. 31–38. ISSN: 1546-170X. DOI: 10.1038/s41591-021-01614-0.

[5] Jack Wilkinson et al. "Time to reality check the promises of machine learning-powered precision medicine." en. In: *The Lancet Digital Health* 2.12 (Dec. 2020), e677–e680. ISSN: 2589-7500. DOI: 10.1016/S2589-7500(20)30200-4. URL: https://www.sciencedirect.com/science/article/pii/S2589750020302004 (visited on Apr. 7, 2022).

[6] D. Sculley et al. "Hidden Technical Debt in Machine Learning Systems." In: 28 (2015). Ed. by C. Cortes et al. URL: https://proceedings.neurips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf.

[7] Ujjwal Baid et al. "The RSNA-ASNR-MICCAI BraTS 2021 Benchmark on Brain Tumor Segmentation and Radiogenomic Classification." In: *CoRR* abs/2107.02314 (2021). arXiv: 2107.02314. URL: https://arxiv.org/abs/2107.02314.

[8] fast.ai. *fast.ai. Making neural nets uncool again*. https://www.fast.ai/. Accessed: 2021-09-01. 2021.

[9] MONAI. *Project MONAI. Medical Open Network For AI.* https://www.monai.io/. Accessed: 2021-09-01. 2021.

[10] Sathiesh Kumar Kaliyugarasan, Arvid Lundervold, and Alexander Selvikvåg Lundervold. "Pulmonary Nodule Classification in Lung Cancer from 3D Thoracic CT Scans Using fastai and MONAI." In: *IJIMAI* (2021).

[11] Satheshkumar Kaliyugarasan et al. "2D and 3D U-Nets for skull stripping in a large and heterogeneous set of head MRI using fastai." In: *NIK2020* (2020).

[12] S. Kaliyugarasan et al. "Brain Age versus Chronological Age: A Large Scale MRI and Deep Learning Investigation." In: ECR2020. 2020.

[13] Erlend Hodneland et al. "Fully Automatic Whole-Volume Tumor Segmentation in Cervical Cancer." In: *Cancers* 14.10 (2022). ISSN: 2072-6694. DOI: 10.3390/cancers14102372. URL: https://www.mdpi.com/2072-6694/14/10/2372.

[14] docker. *Docker overview.* https://docs.docker.com/get-started/overview/. Accessed: 2021-09-21. Oct. 2021.

[15] Michele Larobina and Loredana Murino. "Medical Image File Formats." In: *Journal of Digital Imaging* 27.2 (Dec. 2013), pp. 200–206. DOI: 10.1007/s10278-013-9657-9.

[16] Xiangrui Li et al. "The first step for neuroimaging data analysis: DICOM to NIfTI conversion." In: *Journal of neuroscience methods* 264 (2016), pp. 47–56.

[17] Seymour A. Papert. *The Summer Vision Project.* en_US. MIT AI Lab. July 1966. URL: https://dspace.mit.edu/handle/1721.1/6125.

[18] Geremy Heitz and Daphne Koller. "Learning spatial context: Using stuff to find things." In: *European conference on computer vision.* 2008, pp. 30–43.

[19] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context." In: *European conference on computer vision.* Springer. 2014, pp. 740–755. DOI: 10.48550/arXiv.1405.0312.

[20] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. "Coco-stuff: Thing and stuff classes in context." In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2018, pp. 1209–1218.

[21] Yogesh Kumar et al. "Artificial intelligence in disease diagnosis: a systematic literature review, synthesizing framework and future research agenda." en. In: *Journal of Ambient Intelligence and Humanized Computing* (Jan. 2022). ISSN: 1868-5137, 1868-5145. DOI: 10.1007/s12652-021-03612-z. (Visited on May 26, 2022).

[22] Abhimanyu S. Ahuja. "The impact of artificial intelligence in medicine on the future role of the physician." en. In: *PeerJ* 7 (Oct. 2019), e7702. ISSN: 2167-8359. DOI: 10.7717/peerj.7702. URL: https://peerj.com/articles/7702.

[23] Bo Wang et al. "AI-assisted CT imaging analysis for COVID-19 screening: Building and deploying a medical AI system." In: *Applied Soft Computing* 98 (Jan. 2021), p. 106897. DOI: 10.1016/j.asoc.2020.106897.

[24] Gregory Sharp et al. "Vision 20/20: Perspectives on automated image segmentation for radiotherapy." In: *Medical Physics* 41.5 (May 2014), p. 050902. ISSN: 0094-2405. DOI: 10.1118/1.4871620. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4000389/.

[25] Nikhil R. Pal and Sankar K. Pal. "A review on image segmentation techniques." In: *Pattern Recognition* 26.9 (1993), pp. 1277–1294. ISSN: 0031-3203. DOI: https://doi.org/10.1016/0031-3203(93)90135-J. URL: https://www.sciencedirect.com/science/article/pii/003132039390135J.

[26] K.K.D. Ramesh et al. "A Review of Medical Image Segmentation Algorithms." en. In: *EAI Endorsed Transactions on Pervasive Health and Technology* (July 2018), p. 169184. ISSN: 2411-7145. DOI: 10.4108/eai.12-4-2021.169184. (Visited on Feb. 15, 2022).

[27] Salwa Abdulateef and Mohanad Salman. "A Comprehensive Review of Image Segmentation Techniques." en. In: *Iraqi Journal for Electrical and Electronic Engineering* 17.2 (Dec. 2021), pp. 166–175. ISSN: 2078-6069, 1814-5892. DOI: 10.37917/ijeee.17.2.18. URL: http://ijeee.edu.iq/Papers/Vol17-Issue2/1570757842.pdf (visited on May 5, 2022).

[28] Niall O'Mahony et al. "Deep Learning vs. Traditional Computer Vision." In: *Advances in Computer Vision*. Ed. by Kohei Arai and Supriya Kapoor. Cham: Springer International Publishing, 2020, pp. 128–144. ISBN: 978-3-030-17795-9.

[29] Matthew D. Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks." en. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 818–833. ISBN: 978-3-319-10590-1. DOI: 10.1007/978-3-319-10590-1_53.

[30] Alexander LeNail. "NN-SVG: Publication-Ready Neural Network Architecture Schematics." In: *Journal of Open Source Software* 4.33 (2019), p. 747. DOI: 10.21105/joss.00747.

[31] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation." In: *Medical Im-*

age Computing and Computer-Assisted Intervention – MICCAI 2015. Ed. by Nassir Navab et al. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4.

[33] Nahian Siddique et al. "U-Net and Its Variants for Medical Image Segmentation: A Review of Theory and Applications." In: *IEEE Access* 9 (2021), pp. 82031–82057. DOI: 10.1109/ACCESS.2021.3086020.

[34] Özgün Çiçek et al. "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation." In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*. Ed. by Sebastien Ourselin et al. Cham: Springer International Publishing, 2016, pp. 424–432. ISBN: 978-3-319-46723-8.

[35] Eric Kerfoot et al. "Left-ventricle quantification using residual U-Net." In: *International Workshop on Statistical Atlases and Computational Models of the Heart*. Springer. 2018, pp. 371–380.

[36] Ali Hatamizadeh et al. "UNETR: Transformers for 3D Medical Image Segmentation." In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. Jan. 2022, pp. 574–584.

[37] Alexey Tsymbal. "The problem of concept drift: definitions and related work." In: *Computer Science Department, Trinity College Dublin* 106.2 (2004), p. 58.

[38] Lucas Baier, Fabian Jöhren, and Stefan Seebacher. "Challenges in the Deployment and Operation of Machine Learning in Practice." In: *ECIS*. 2019.

[39] Andre Esteva et al. "Dermatologist–level classification of skin cancer with deep neural networks." In: *Nature* 542.7639 (Feb. 2017), pp. 115–118. ISSN: 0028-0836. DOI: 10.1038/nature21056. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8382232/ (visited on May 10, 2022).

[40] Awni Y. Hannun et al. "Cardiologist-Level Arrhythmia Detection and Classification in Ambulatory Electrocardiograms Using a Deep Neural Network." In: *Nature medicine* 25.1 (Jan. 2019), pp. 65–69. ISSN: 1078-8956. DOI: 10.1038/s41591-018-0268-3. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6784839/ (visited on May 10, 2022).

[41] Emma Beede et al. "A human-centered evaluation of a deep learning system deployed in clinics for the detection of diabetic retinopathy." In: *Proceedings of the 2020 CHI conference on human factors in computing systems*. 2020, pp. 1–12.

[42] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. "Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models." In: *arXiv preprint arXiv:1708.08296* (2017).

[43] Adriel Saporta et al. “Deep learning saliency maps do not accurately highlight diagnostically relevant regions for medical image interpretation.” In: *medRxiv* (Oct. 2021). DOI: 10.1101/2021.02.28.21252634.

[44] Danton S. Char, Nigam H. Shah, and David Magnus. “Implementing Machine Learning in Health Care — Addressing Ethical Challenges.” In: *The New England journal of medicine* 378.11 (Mar. 2018), pp. 981–983. ISSN: 0028-4793. DOI: 10.1056/NEJMp1714229. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5962261/.

[45] W. Nicholson Price II, Sara Gerke, and I. Glenn Cohen. “Potential Liability for Physicians Using Artificial Intelligence.” In: *JAMA* 322.18 (Nov. 2019), pp. 1765–1766. ISSN: 0098-7484. DOI: 10.1001/jama.2019.15064. URL: https://doi.org/10.1001/jama.2019.15064 (visited on May 23, 2022).

[46] Geert Litjens et al. “A survey on deep learning in medical image analysis.” en. In: *Medical Image Analysis* 42 (Dec. 2017), pp. 60–88. ISSN: 1361-8415. DOI: 10.1016/j.media.2017.07.005. URL: https://www.sciencedirect.com/science/article/pii/S1361841517301135 (visited on Mar. 23, 2022).

[47] Luca Saba et al. “The present and future of deep learning in radiology.” In: *European Journal of Radiology* 114 (2019), pp. 14–24. ISSN: 0720-048X. DOI: https://doi.org/10.1016/j.ejrad.2019.02.038. URL: https://www.sciencedirect.com/science/article/pii/S0720048X19300919.

[48] Michele Larobina and Loredana Murino. “Medical Image File Formats.” In: *Journal of Digital Imaging* 27.2 (Apr. 2014), pp. 200–206. ISSN: 0897-1889. DOI: 10.1007/s10278-013-9657-9. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3948928/ (visited on May 18, 2022).

[49] Peter Mildenberger, Marco Eichelberg, and Eric Martin. “Introduction to the DICOM standard.” en. In: *European Radiology* 12.4 (Apr. 2002), pp. 920–927. ISSN: 1432-1084. DOI: 10.1007/s003300101100. (Visited on May 16, 2022).

[50] R.W. Cox et al. *A (sort of) new image data format standard: NiFTI-1.* Presented at the 10th Annual Meeting of the Organization for Human Brain Mapping. Jan. 2004.

[51] Malik Aasen and Fredrik Fidjestøl Mathisen. “De-identification of medical images using object-detection models, generative adversarial networks and perceptual loss.” eng. MA thesis. The University of Bergen, June 2021. URL: https://bora.uib.no/bora-xmlui/handle/11250/2770435.

[52] Bjoern H. Menze et al. “The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS).” eng. In: *IEEE transactions on medical imaging* 34.10 (Oct. 2015), pp. 1993–2024. ISSN: 1558-254X. DOI: 10.1109/TMI.2014.2377694.

[53] Spyridon Bakas et al. "Advancing The Cancer Genome Atlas glioma MRI collections with expert segmentation labels and radiomic features." eng. In: *Scientific Data* 4 (Sept. 2017), p. 170117. ISSN: 2052-4463. DOI: 10.1038/sdata.2017.117.

[54] Elizabeth A Maher et al. "Malignant glioma: genetics and biology of a grave matter." In: *Genes & development* 15.11 (2001), pp. 1311–1333. DOI: 10.1101/gad.891601. URL: http://genesdev.cshlp.org/content/15/11/1311.

[55] Mobarakol Islam, V Jose, and Hongliang Ren. "Glioma prognosis: Segmentation of the tumor and survival prediction using shape, geometric and clinical information." In: *International MICCAI Brainlesion Workshop*. Springer. 2018, pp. 142–153. DOI: 10.1007/978-3-030-11726-9_13.

[56] Jeremy Howard and Sylvain Gugger. "Fastai: a layered API for deep learning." In: *Information* 11.2 (2020), p. 108. DOI: 10.3390/info11020108.

[57] Leslie N Smith and Nicholay Topin. *Super-convergence: Very fast training of neural networks using large learning rates*. International Society for Optics and Photonics, 2019. DOI: 10.1117/12.2520589.

[58] *NVIDIA Blogs: MONAI Open Source AI Framework for Healthcare Research*. en-US. Apr. 2020. URL: https://blogs.nvidia.com/blog/2020/04/21/monai-open-source-framework-ai-healthcare/ (visited on May 24, 2022).

[59] Fabian Isensee et al. "nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation." In: *Nature methods* 18.2 (2021), pp. 203–211. DOI: 10.1038/s41592-020-01008-z.

[60] Project Jupyter. *Project Jupyter — about us*. en. URL: https://jupyter.org (visited on May 27, 2022).

[61] Ufford M. et al. *Beyond Interactive: Notebook Innovation at Netflix*. en. Feb. 2019. URL: https://netflixtechblog.com/notebook-innovation-591ee3221233 (visited on May 27, 2022).

[62] E Nolf et al. "XMedCon: an open-source medical image conversion toolkit." In: *2003 European Association of Nuclear Medicine annual congress*. Vol. 30. suppl. 2. 2003, S246–S246.

[63] *The Go Programming Language*. en. URL: https://go.dev/ (visited on May 23, 2022).

[64] Paul A. Harris et al. "Research electronic data capture (REDCap)—A metadata-driven methodology and workflow process for providing translational research informatics support." en. In: *Journal of Biomedical Informatics* 42.2 (Apr. 2009), pp. 377–381. ISSN: 1532-0464. DOI: 10.1016/

j.jbi.2008.08.010. URL: https://www.sciencedirect.com/science/article/pii/S1532046408001226 (visited on May 16, 2022).

[65] pydicom. *Pydicom*. URL: https://pydicom.github.io/ (visited on May 4, 2022).

[66] Paul A. Yushkevich et al. "User-Guided 3D Active Contour Segmentation of Anatomical Structures: Significantly Improved Efficiency and Reliability." In: *Neuroimage* 31.3 (2006), pp. 1116–1128.

[67] Kjetil Dyrland, Alexander Selvikvåg Lundervold, and PierGianLuca Porta Mana. "Does the evaluation stand up to evaluation?: A first-principle approach to the evaluation of classifiers." en-us. type: article. May 2022. URL: https://osf.io/7rz8t/ (visited on May 27, 2022).

[68] Stephen M. Smith et al. "Advances in functional and structural MR image analysis and implementation as FSL." In: *NeuroImage* 23 (2004). Mathematics in Brain Imaging, S208–S219. ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2004.07.051. URL: https://www.sciencedirect.com/science/article/pii/S1053811904003933.

[69] Florian Kofler et al. "BraTS Toolkit: Translating BraTS Brain Tumor Segmentation Algorithms Into Clinical and Scientific Practice." In: *Frontiers in Neuroscience* 14 (2020). ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00125.

Credits

Icons used in Figure 1.1 were made by Dmytro Vyshnevskyi, Elias Bikbulatov, twentyfour, orvipixel, phatplus, Smartline, Becris, lakonicon, and kmg design from www.flaticon.com.

Icons used in Figure 2.1 were made by Freepik, and kornkun from www.flaticon.com.

Icons used in Figure 3.2 were made by Smashicons, RaftelDesign, and FBJan from www.flaticon.com.

Icons used in Figure 4.5 were made by twentyfour, orvipixel, and lakonicon from www.flaticon.com.