FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Tumor detection in CT images using Neural Networks
Student:	Tomáš Detko
Supervisor:	Ing. Jakub Žitný
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2020/21

Instructions

Research current state-of-the-art techniques that are used for detection and segmentation tasks in the medical imaging domain, focus on CT images. Implement your own prototype model that will work on one of the datasets provided by the supervisor. Compare the performance of your model with reference results from literature or existing models and discuss the pros and cons. Publish your prototype code and make sure your results are reproducible.

References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague January 5, 2020

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Tumor detection in CT images using Neural Networks

Tomáš Detko

Department of Applied Mathematics
Supervisor: Ing. Jakub Žitný

June 4, 2020

Acknowledgements

Thanks firstly go to my supervisor Ing. Jakub Žitný for his valuable insights and suggestion in this topic, as well as, for his kindness and willingness to help, whenever help was needed. Next, my thanks go to my family, for their endless amount of support during my studies. Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on June 4, 2020 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Detko, Tomáš. *Tumor detection in CT images using Neural Networks*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Zlepšovanie a vyvíjanie nových algoritmov v oblasti machine learningu zasahuje do každodenného života. So zvyšovaním výkonu a veľkosti datasetov sa úlohy, ktoré boli v minulosti pokladané za nedosiahnuteľné méty, stávajú uchopiteľné a zvládnuteľné. Segmentácia objektov patrí v dnešnej dobe medzi bežné úlohy strojového videnia. Má veľké využitie v technológií samojazdiacích aút, priemysle, obchode a zdravotníctve.

Táto práca si kladie za cieľ navrhnutie, natrénovanie a porovnanie výkonnosti modelu s ostatnými State-of-the-art modelmi vrámci segmentácie CT skenov pacientov. Podáva teoretický základ a oboznamuje čitateľa. V práci je veľký dôraz kladený na celkovým procesom vývoja takýchto modelov. Ukazuje sa, že prototyp navrhnutej neurónovej siete dosahuje vepšie výsledky výsledky ako neurónové siete o väčšom počte parametrov.

Klíčová slova Segmentácia snímkov v medicíne, convolučná neurónová sieť, neurónová sieť, hlboké učenie, strojové učenie

Abstract

Improving and developing new algorithms in the field of machine learning has impact on everyday life. As the performance and size of datasets increase, tasks that were previously considered unattainable metas become comprehensible and manageable. Object segmentation is one of the most common tasks of computer vision today. It is widely used in car technology, industry, trade and healthcare.

This work aims to design, train and compare the performance of the model with other State-of-the-art models within the segmentation of CT scans of patients. It provides a theoretical basis and acquaints the reader. In this work, great emphasis is placed on the overall process of development of machine learning models. It turns out that the prototype of the proposed neural network achieves better results than neural networks with a larger number of parameters.

Keywords Medical image segmentation, convolution neural network, neural network, deep learning, machine learning

Contents

List of Figures

List of Tables

Introduction

Medicine has a huge impact on improving people's quality of life. In anient times, it was closely associated with the formation of early civilizations. Initially, the treatments and tools were very simple. At that time, doctors treated patients with herbs and the knowledge was passed down from generation to generation. However, such treatment has often not been successful. People lacked the knowledge and technology to make drugs and to understand the processes in human body. The knowledge of physicians from Arabia, ancient India, China, ancient Greece, Rome, Egypt served as the basis for the emergence of medicine that we know today.

Recently, we are increasingly encountering machine learning (ML) and deep learning (DL) technologies in industries where this would not have been possible a few years ago. The main obstacles to the extension of these algorithms were the insufficient computing power of personal computers and the absence of datasets. That is why primarily only universities and international companies have been involved in research.

Many things have changed in the last few years in the field of ML / DL. Algorithms perform better in some tasks than humans themselves. Cloud Computing Services (Google, AWS, Azure,…) are available, what provide an enormous amount of computing power needed to debug the model at an affordable price. Pre-trained models considered State-of-the-art can be found on the Internet. Some people consider it to be another revolution [1] that will hit all sectors of our daily lives.

Chapters 1 and 2 deal with the Medical Background and the theory of machine learning and deep learning itself. Segmentation and model architectures are discussed in Chapter 3. Chapter 4 deals with working with the dataset, its modification, processing and augmentation. Implementation, implementation details, measurements and the results are captured in Chapter 5.

Thesis's Objective

In this work DL algorithms are used to process CT scans of patients with kidney cancer. Then we will compare several approaches and choose the most suitable ones for this domain of problems.

There are used segmentation techniques for the detection of kidneys and tumors from CT scans in this work. Such a solution can facilitate and speed up the work of doctors in diagnosing diseases. The work aim to detect tumors in the kidneys, but it would be very similar to detect tumors in other organs, what creates a possibility for further improvement or potential for another project.

The aim of the theoretical part is to acquaint the reader with the basic concepts of machine learning, deep learning and selected architectures of models designed for image segmentation. Another goal of the theoretical part is to acquaint the reader with the method of creating diagnostic images in medicine.

The aim of the practical part is to compare State-of-the-art techniques used in the detection and segmentation in the domain of medical images with a focus on CT scans. The aim is also to create a prototype model for cancer segmentation and compare it with other State-of-the-art models, focusing on performance and discussing the advantages and disadvantages of specific models. The results of the thesis should be as reproducible as possible.

Chapter 1

Medical background

The chapter provides an overview of basic diagnostic devices and approaches in today's medicine.

1.1 Introduction

If the doctor wanted to find out the cause of the disease in the past, he had no choice but to take a scalpel and cut. However, there are places in the body where it is very difficult to get a knife. Such an procedure increases the risk of tissue damage, which could have been prevented already during the diagnosis. Modern medicine tries to prefer a non-invasive method of treatment, ie. interfere with the patient's tissues as little as possible. The doctor opts for surgery only in case other treatments have failed. Thanks to technology, doctors today are able to diagnose the disease with great accuracy, determine the location of damaged tissue, and make the right decisions for further treatment. Specialized medical devices open a window into the human body and the doctor no longer depends only on his judgment and the knife.

Standard imaging methods used in diagnostics include:

- X-ray
- CT (computed tomography)
- MR (magnetic resonance)
- Ultrasound
- Osteodensitometry

1.2 X-ray

It is a device that uses the properties of high-frequency electromagnetic radiation. This radiation is in the range between (3×10^{16} Hz to 3×10^{19} Hz), ie outside the visible spectrum. High-frequency X-rays are found in the ultraviolet spectrum but still have a lower frequency (and therefore energy) than gamma radiation. The following relation applies to electromagnetic radiation and its frequency ($f = c/\lambda$), where c is the speed of light and λ is the wavelength.

Medical examination uses a principle analogous to casting a shadow during an evening walk. The doctor sets up the device and it begins to emit electromagnetic radiation into the patient's tissues. An exposure plate is placed behind the observed part of the patient's body.

X-rays have a high energy ($e = \hbar f$) and pass through soft tissues almost without attenuation. If the radiation hits the bone, it is partially or completely attenuated and does not reach the exposure plate. This creates a negative (photo) of a given part of the patient's body.

X-rays [2] are used primarily to detect fractures of bones and denser tissues, because the soft tissue is shone through and its damage will not be captured in the resulting image. Another limitation is when there is a bone in front of the subject (such as the heart). Then we will not know about the soft tissue disorder because the radiation was shaded by the bone.

X-rays also have adverse effects on the bodies of living organisms. Gamma radiation is used to create images. This is very dangerous because it can cause changes in DNA.

1.3 Computed tomography

It is an imaging technique used to make 3D X-rays. The patient lies still during the examination and the examination is absolutely painless.

The device looks like a tunnel [3]. The patient gradually moves through it and the device takes pictures a few millimeters apart thanks to X-rays. Each image captures a section of the patient's body.

The resulting image is a mixture of grayscale pixels. The gray saturation is proportional to the amount of radiation that has passed through the organs. In the resulting image, the soft tissues are shown in a lighter color. CT can be used to detect tissue damage that is only a few millimeters in size. The device is most often used in examinations of the abdominal cavity and skull [3].

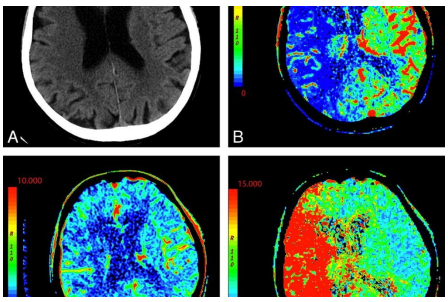
As with X-ray examination, the body is exposed to dangerous gamma radiation during CT examination. Complications can also be caused by the contrast agent itself. It is a fluid given intravenously into a patient's body. It is intended to ensure better visibility of the contours of the bloodstream.

1.4 Magnetic resonance

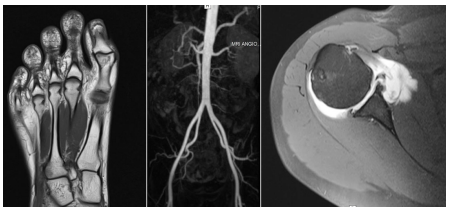
It is a diagnostic method used to take pictures of soft tissues in the body. It is especially suitable for examining blood vessels, abdominal cavity, joints, spine and brain [4].

Magnetic resonance imaging works on a different principle than X-rays. The device measures the energy intensity of the emission of hydrogen protons, which are excited by a magnetic field. Different parts of the body contain different amounts of water, which is reflected in the resulting image with different radiation intensity. They then process the images and assign different colors to different parts of the body. This method is capable of higher resolution and contrast than X-rays [4].

The disadvantage of resonance is that a patient who has metal objects in his body (joints, clamps, artificial heart valve) cannot undergo such an examination.



(a) CT scan of brain [5].



(b) MRI imaging of patient [6].

Figure 1.1: The images show two of the main imaging methods.

Chapter 2

Machine learning background

This chapter deals with information from the field of machine learning / deep learning with emphasis on explaining the operation of the basic principles of convolutional neural networks.

2.1 Machine learning

Machine learning is a way of programming that relies on knowledge stored in data. Machine learning algorithms are largely non-deterministic and their results depend on the qualitative and quantitative nature of the data. Machine learning is divided into 2 main fields based on type of acquired knowledge:

- Supervised learning
- Unsupervised learning

2.1.1 Supervised learning

We speak of supervised learning [7] when the dataset has shaped like $\{(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i)\}$, where $x_k \in \mathbb{R}^n$ and $y_k \in \mathbb{R}^m$. The machine learning algorithm tries to learn the mapping function $\hat{y_k} = f(x_k, \theta)$ between these corresponding pairs. θ represents a set of learning parameters. In order to be able to teach the model, we need a metric of accuracy of our algorithm. For ML / DL, this metric is called the loss function. $L : (\hat{y}, y) \in \mathbb{R} \times Y \longmapsto L(\hat{y}, y) \in \mathbb{R}$, where y is ground truth label from dataset and $\hat{y}$ is target value, which the algorithm returns. The loss function returns information in the form of a number, which expresses how much $\hat{y}$ differs from y.

Learning takes place as an iterative process based on the optimization of the parameters of the loss function $L(\hat{y}, y)$. The algorithm tries to find such θ where the mapping function will be as accurate as possible so the value of $L(\hat{y}, y)$ will be as small as possible.

There are a few criteria according to which we can divide ML algorithms into larger groups.

Types of algorithms based on value of target variable:

- Regression ($y \in \mathbb{R}^n$) (Linear regression, etc.)
- Classification ($y \in C$ where $C = \{c_1, c_2, \ldots, c_i\}$) (Logistic regression, SVM, etc.)

Classification of algorithms based on the type of models [7]:

- Discriminative – estimate $p(y|x)$
- Generative - calculates $p(x|y)$ and estimates the most probable y for the unknown x using Bayesian theorem

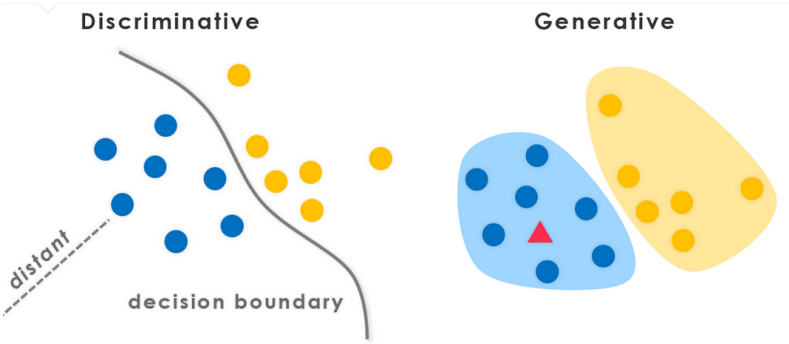


Figure 2.1: The discriminant model learns decision boundry. The generative model learns the probability distribution of data[7].

2.1.2 Unsupervised learning

Unsupervised learning [8] algorithms work only with the data itself in the form $\{x_1, x_2, , x_k\}$ without the label y. The algorithm reveals dependencies and complex relationships in data without a teacher. Unsupervised learning is mainly used for data clustering.

Such an approach is often used to preprocess the dataset, possibly generating some missing data features.

Some of commonly used algorithms of this type include:

- Singular value decomposition (SVD)
- K-Means
- Principal component analysis (PCA)

- Density-Based Spatial Clustering of Applications with Noise (DBSCAN)
- Agglomerative Hierarchical Clustering

2.1.3 Metrics

Measurement is a very important part of any process and ML is no exception.

Metrics are a functions that evaluates the quality of a model. It is very similar to the loss function with the difference that it is not used as an optimization criterion during model learning. To further understand and fine-tune the performance of the model appropriate metrics are used. There are metrics used for classification problems as well as regression problems.

2.1.4 Classification metrics

Confusion matrix is the most common way to evaluate the accuracy of a classification model.

It is defined as follows:

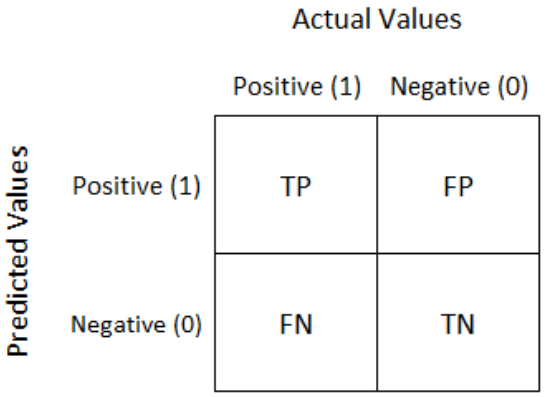


Figure 2.2: Confusion matrix for a model classifying 2 classes[9].

The most commonly used metrics are [10]:

Accuracy – overall performance of a model

$$\frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

Precision – how accurate the positive predictions are

$$\frac{TP + FP}{TP} \tag{2.2}$$

Recall Sensitivity – what fraction of true positive we can get

$$\frac{TP}{TP + FN} \tag{2.3}$$

Specificity – fraction of true negative we can get

$$\frac{TN}{TN + FP} \tag{2.4}$$

F1 score – consider both the precision and the recall

$$\frac{2TP}{2TP + FP + FN} \tag{2.5}$$

- True positives (TP) – cases when the actual class of the data point was 1 and the predicted class is also 1
- True negatives (TN) – cases when the actual class of the data point was 0 and the predicted class is also 0
- False positives (FP) – cases when the actual class of the data point was 0 and the predicted class is 1
- False negatives (FN) – cases when the actual class of the data point was 1 and the predicted class is 0

2.1.5 ROC Curve and AUC

AUC (area under curve) and ROC (Receiver Operating Characteristics) are among the most basic metrics of classification models [11].

TPR (True Positive Rate)

$$\frac{TP}{TP + FN} \tag{2.6}$$

FPR (False Positive Rate)

$$\frac{FP}{TN + FP} \tag{2.7}$$

ROC represents the probability distribution of the TP and TN classifications [11]. AUC is the area under the curve that originates from the interconnection of ROC points. Individual points are created by changing the threshold for classification, which changes the TPR and FPR. This classification method measures the numbers of correctly estimated TPs and TNs. ROC allows us to choose a threshold at which data separation is the best possible [11].

The AUC is used to compare the classification accuracy of individual models. Its area can have a value from 0 to 1.

- If the AUC = 1 – model has all TN and TP classified correctly
- If the AUC = 0.5 – model has classified the same number of TP cases as FP cases
- If the AUC = 0 – this is the state when the model classified everything wrongly

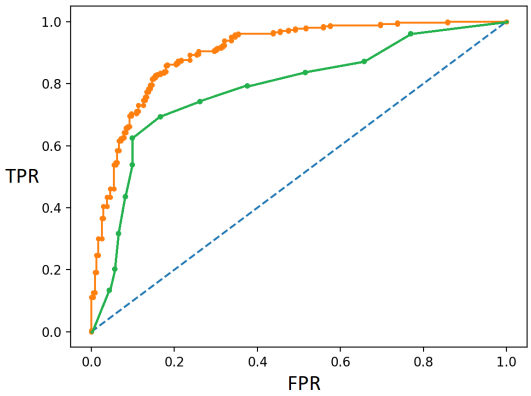


Figure 2.3: The picture shows comparison of performance for 2 models. The model represented by green color is less accurate than the orange one.

2.1.6 Regression metrics

The following metrics are used to evaluate models on regression type tasks:

Mean Square Error (MSE)

$$\frac{1}{N}\sum_{i=1}^{N}(y-\hat{y})^2 \tag{2.8}$$

Mean Absolute Error (MAE)

$$\frac{1}{N}\sum_{i=1}^{N}\mid y-\hat{y}\mid \tag{2.9}$$

2.2 Neural networks

An artificial neural network [12] is an information model inspired by biological clusters of neurons in the brains of living organisms. Its basic building block is the perceptron (artificial neuron). Task of the perceptron is to sum up the

potential of all input variables and decide whether to send the signal to other layers. An activation function $a(x)$ is used for this decision.

Every perceptron is mathematically by a function: $f(x) = a(W^T x + b)$, where $W \in \mathbb{R}^{n \times m}$ is a matrix of weights, $x \in \mathbb{R}^{n \times 1}$ is input vector, $b \in \mathbb{R}^{m \times 1}$ is bias term and $a(x)$ is an activation function. An important assumption is that the activation function is nonlinear and differentiable on the whole domain (due to the learning algorithm).

A neural network consists of one or more layers. The output of the layer h_i is represented by the vector a_i which represents the input for the layer h_{i+1}. The output of the last layer is the value $\hat{y}$. The value of $\hat{y}$ is further used in loss function so NN can optimize weights.

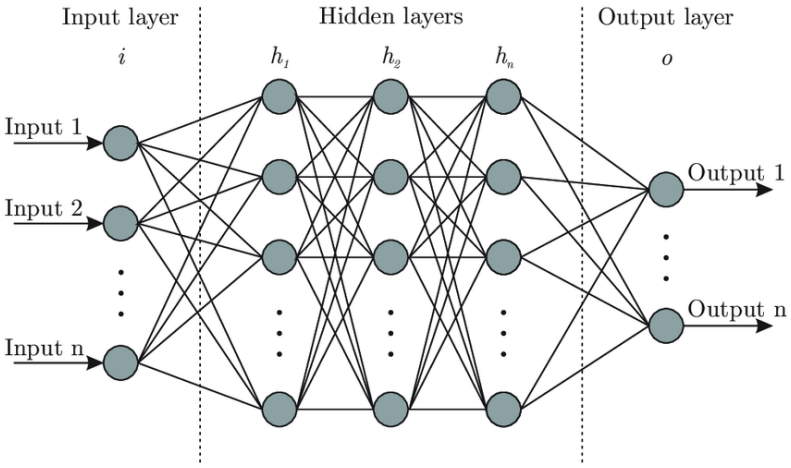


Figure 2.4: Deep neural network [13].

Deep neural network (DNN [13]) is called any architecture that contains more than one hidden layer.

2.2.1 Convolution neural network

CNN [14] is a type of neural network that is used primarily to solve Computer Vision problems. CV has a great impact on the development of new DL algorithms. Thoughts and ideas in one field are often applicable between different fields of the DL community (Attention is all you need [15], 1x1 convolutions [16], etc.).

CNN works on the input with images that are represented by 3D tensors. Each such 3D tensor has dimensions $(h \times w \times n_c)$, where h is height, w is width and n_c represents number of channels. The network consists of layers that have different functionalities (convolution [17], maxpooling [18], upsampling [18]).

The most common cases of CNN use in practice include self-driving cars, face recognition, image classification, object detection, object segmentation, neural style transfer.

2.2.2 Convolution layer

Assume that the image is represented by a 3D tensor of size $(H_{in} \times W_{in} \times D_{in})$. We can work with the image itself as if it was a vector $x \in \mathbb{R}^{(H_{in} \times W_{in} \times D_{in}) \times 1}$ and use a fully connected layer as in the case of a simple NN. The input vector x is transformed using the equation $f(x) = a(W^T x + b)$. That creates new vector $a_i \in \mathbb{R}^{k \times 1}$. Matrix W^T has the dimension $k \times (H_{in} \times W_{in} \times D_{in})$. This is a very inefficient approach as with the dimensions of commonly used images (1000x1000x3) and the size of the output vector $a_i \in \mathbb{R}^{k \times 1}$, where k = 1000, the matrix of weights has a size of almost 3GB. Another problem is that such a representation of the image (flatten vector) is not able to actually capture local information (receptive field) within a smaller part of the image. The convolutional approach solves all these problems.

The convolution filter is represented by a $(h \times w \times D_{in})$ tensor. This filter gradually convolves through a 3D tensor representing the image. In convolution, the dot product is calculated between the filter weights and the pixel values of the image. A bias is added to the output value of the filter and the result is advanced as an input parameter of the activation function (most often ReLU). By applying D_{out} filters to the input tensor we get the output tensor with dimensions $(H_{out} \times W_{out} \times D_{out})$.

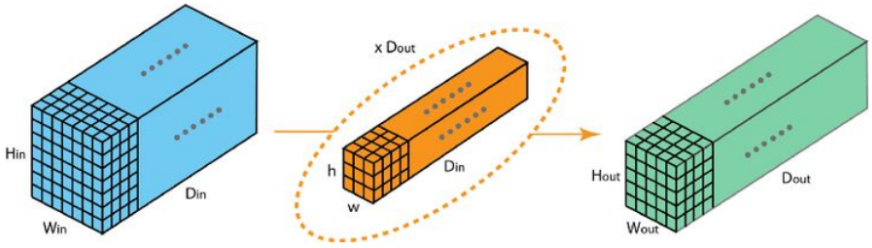


Figure 2.5: Convolution filter [19].

Convolution filters in the first layers are responsible for capturing simple geometric shapes. In the deeper layers, the filters learn to capture more complex geometric shapes (house, eye, bicycle, etc.) [14].

The pictures 2.7 and 2.6 show the shapes which make the neurons within layer most active - the convolution filter searches a specific shape on each picture.

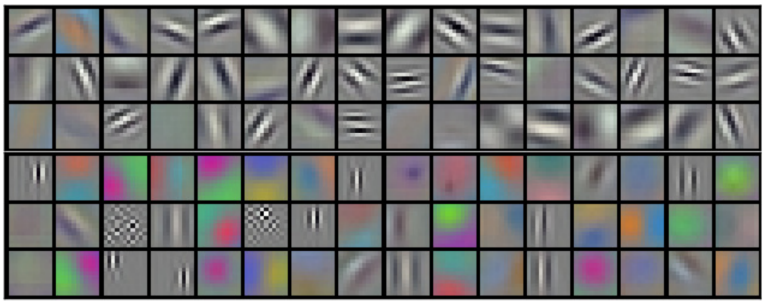


Figure 2.6: Filter activations in first layers [20].

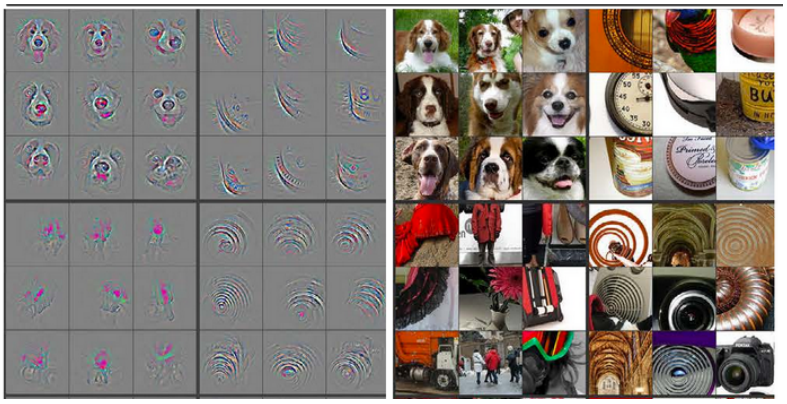


Figure 2.7: Filter activations in last layers [20].

2.2.3 Why CNNs have a relatively low number of learning parameters

There are 2 primary reasons why CNNs have a relatively small number of parameters (compared to fully connected NNs):

- **Parameter sharing** – detection of a feature (such as a horizontal edge) is useful in several parts of the image. The same filter responsible for detecting a specific object is applied to the entire image without the need to increase the number of training parameters.

- **Sparsity of connections** – in each layer the output value depends only on a small number of inputs (receptive field).

Because of these 2 features, CNN can be trained on a smaller dataset and it is less prone to overfit.

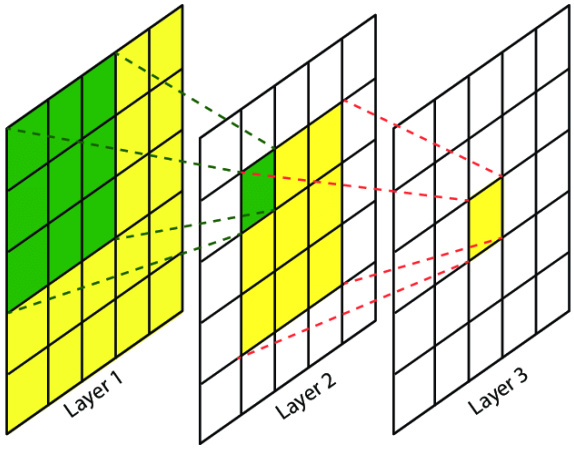


Figure 2.8: Receptive field in CNN.

2.2.4 Calculation of output tensor dimensions, padding, stride

Suppose we have an image with dimensions ($H_{in} \times W_{in} \times D_{in}$) and a convolution filter ($h \times w \times D_{in}$) (3rd dimension needs to be the same in both). After application of convolution, addition of bias and application of nonlinearity, the resulting tensor ($H_{out} \times W_{out} \times 1$) is created. In our case, we used D_{out} convolution filters. A 3D tensor with dimensions ($H_{out} \times W_{out} \times D_{out}$) is created.
The dimension of H_{out} is

$$\lfloor \frac{H_{in} + 2p - f_h}{s_h} + 1 \rfloor \tag{2.10}$$

The width W_{out} of the output is computed analogically. In equation p means padding, $f_{h/w}$ is the vertical/horizontal dimension of the kernel used and $s_{h/w}$ is the vertical/horizontal size of the convolution kernel displacement step above the image.

Padding can have 'same' value, then the image is padded with zeros so that $H_{in} = H_{out}$ and $W_{in} = W_{out}$, or valid. When value is 'valid' then padding is not used. Same padding has the advantage of maintaining the dimension of the input and output image. Another advantage is that it makes better use of the information stored along the edge of the image.

2.2.5 Number of learning parameters of the convolution layer

The number of convolution layer parameters itself depends on the filter size, the number of filters and the bias parameter. Suppose we have a tensor at the input ($H_{in} \times W_{in} \times D_{in}$). We use N convolution filters each with dimension

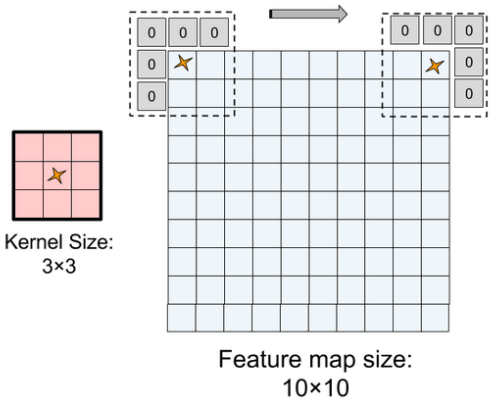


Figure 2.9: Padding [21].

$(f_h \times f_w \times D_{in})$. Then the resulting number of learning parameters of the given layer is: $(f_h * f_w * D_{in} * N) + N$

2.2.6 Pooling layer

The pooling layer is most often inserted after the convolution layer just after the application of the activation function. The filter is represented as a tensor of 2D dimensions $p_h \times p_w$. Information from the region $p_h \times p_w$ is reduced to one number, which best describes the region. Such a filter is applied horizontally and vertically to the whole image. The pooling layer itself has no learning parameters. There are many ways in which the pooling filter decides which information it considers important.

- AveragePooling – the output is the value calculated as the average of the values from the region under the filter
- MaxPooling – the output is the largest value from the region

2.2.7 Upsampling

The opposite operation to pooling layers is the layers responsible for upsampling. The easiest way to upsampling is to copy the necessary pixels (basically a simple image magnification). There is a more advanced method called unpooling. Here, it is assumed that the encoder-decoder architecture contains corresponding pooling and unpooling layer pairs. Unpooling layer remembers which cell had the largest value during the maxpooling operation and fills its position with the appropriate value [22].

2.3 Ways to improve training

2.3.1 Vanishing and exploding gradients

We have to solve this problem especially when working with deep neural networks. By gradually passing through the individual layers, the information is lost or acquires extremely large values [23]. Let $f(x)$ be a linear activation function that returns the identity. Then we calculate the output value of the first layer of the neural network as $f(x)_i = W_i^T x_i$ where W_i is the weight matrix of the layer L_i and x_i is the input vector. This value travels as an input parameter to the following layer of the network where the output value $f(x)_{i+1} = W_{i+1}^T * f(x)_i = W_{i+1}^T W_i^T x_i$ is calculated again. If the values in the weight matrix were initialized with weights greater than 1, then the value will gradually increase in the following layers. For values less than one, the value will decrease to 0.

2.3.2 Correct weight initialization

To partially avoid the problem of vanishing / exploding gradients, we need to initialize the weight matrices intelligently. When initializing all weights with zeros, we encounter the problem of symmetry - all neurons of one layer learn the same thing each backpropagation. Thus, the strength of one layer is the same as the strength of any neuron. Therefore, we need to initialize with small random numbers. Linear models generally work best when their input is normalized and standardized by $\mathcal{N}(0, 1)$.

Assume that the output value of a neuron before activation looks like this, where x represents data sample and w represents weights for one neuron in layer.

$$\sum_{i=1}^{m} x_i w_i \tag{2.11}$$

If $\mathbb{E}(x_i) = \mathbb{E}(w_i) = 0$, then $\mathbb{E}(\sum_{i=1}^{m} x_i w_i) = 0$. This means that expected value will be 0, but we cannot say it about variance.

$$Var(\sum_{i=1}^{m} x_i w_i) = \sum_{i=1}^{m} Var(x_i w_i) \tag{2.12}$$

where w_i are i.i.d. and we suppose that x_i are also not correlated.

$$\sum_{i=1}^{m} ([\mathbb{E}(x_i)]^2 Var(w_i) + [\mathbb{E}(w_i)]^2 Var(x_i) + Var(x_i)Var(w_i)) \tag{2.13}$$

x_i, w_i are standardized so $\mathbb{E}(x_i) = \mathbb{E}(w_i) = 0$

$$\sum_{i=1}^{m}(Var(x_i)Var(w_i)) \tag{2.14}$$

we suppose that x and w have same variance.

$$Var(x)[mVar(w)] \tag{2.15}$$

If m is greater than 1, then the variance for subsequent layers will increase.

In order for $[mVar(w)]$ to be 1, we need a to have the value $1/\sqrt{m}$. Used fact $Var(aw) = a^2Var(w)$.

$$mVar(aw) = 1 \tag{2.16}$$

$$ma^2Var(w) = 1 \tag{2.17}$$

$$m(\frac{1}{\sqrt{m}})^2Var(w) = 1 \tag{2.18}$$

$$Var(w) = 1 \tag{2.19}$$

In Xavier [24] initializations, the weights are multiplied by $\frac{\sqrt{2}}{\sqrt{m_{in}}}$.
In Glorot [25] initializations, the weights are multiplied by $\frac{\sqrt{2}}{\sqrt{m_{in}+m_{out}}}$.

2.3.3 Batch normalization

Batch normalization (BN) makes finding hyperparameters much easier and the neural network becomes much more robust [26]. After applying the BN, we can use a higher learning rate while the network weights converge faster. Thanks to this technique, even very deep neural networks can be trained. BN controls the mean and variance of the variables entering the activation function.

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.20}$$

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2 \tag{2.21}$$

$$x_{inorm} = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{2.22}$$

Information in the data may have a different distribution than the normal distribution. Therefore, for individual layers, the data distribution is adjusted as $x_{new} = \gamma * x_{inorm} + \beta$.

2.3.4 The reason why batch normalization works

The problem that is addressed by BN is called covariance shift [26]. Assume a model that learns from data. The training data have a distribution of p, while the test data have a distribution of k. The fact that the training and test set have different distributions will cause the model not to have high accuracy.

With the forward propagation of information through layers, their variance increases as calculated in 2.3.2 and a small change in the information at the input will greatly affect the calculation of the activation function itself somewhere deep in the neural network. Thanks to the application of BN, the input data are standardized to $\mathcal{N}(0, 1)$. The layer itself learns the β (new mean) and γ (new variance) parameters.

In this way, the variance of the output variable from the layer and thus the distribution of the values that represent the input values of the next layer is limited. As a result, the deeper values in the network are more stable and the individual layers can learn more independently.

2.3.5 Residual Connections

Despite the techniques already mentioned, very deep neural networks have still problem with vanishing / exploding gradients. Another way to adress it is to add residual connections [27]. The front layers merge with the layers located deeper in the net. This will ensure better propagation of information through the network.

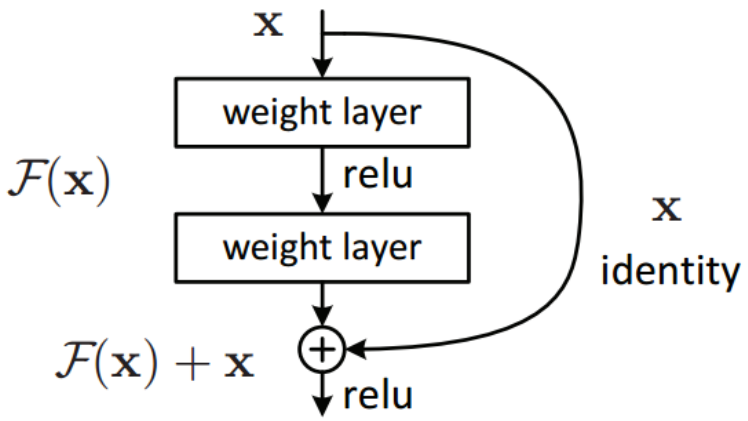


Figure 2.10: Residual connection in DNN [27].

$$z_l = W_l^T a_{l-1} + b_l \tag{2.23}$$

$$a_l = ReLU(z_l) \tag{2.24}$$

$$a_{l+2} = ReLU(z_{l+2}) + a_l \tag{2.25}$$

For such a block, the identity function is very easy to learn. If the network did not need layer l and $l+1$ (the weight matrix contains small values), then the output $l+2$ of the layer looks like a $a_{l+2} = 0 + a_l$ (identity). Such a block can only help the performance of the neural network (learning parameters within the residual block are used) but not harm (identity function - the residual block itself is skipped).

2.3.6 Convolution 1x1

Convolution filter 1x1 is filters with dimensions $(1 \times 1 \times D_{in})$. The most common use of 1x1 convolution in CNN is dimension reduction.

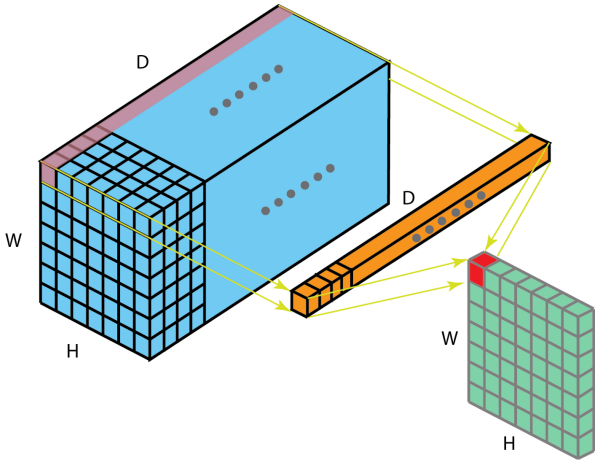


Figure 2.11: 1x1 convolution filter [19].

Such filters serve to reduce the number of mathematical calculations during convolution while maintaining the dimensions of the input and output tensors. In this case, several 1x1 convolution filters are applied to the input tensor. This creates a temporary tensor with denser information. Such temporary tensor can be a bottleneck for maintaining the information. If the number of 1x1 filters is reasonably large then there is no loss of information.

The 1x1 filter is used to replace the dense layer [16]. When using a dense layer, the dimensions of the tensors of the previous layers must be fixed. When creating a network, weight matrix is created in the dense layer, which has precisely defined dimensions. This means that the images at the network input must have a fixed width and height. Replacing the dense layer with a 1x1 convolution filter will ensure that the neural network input does not have to be fixed. In that case, we're talking about the Fully Convolutional Network(FCN).

2.3.7 Batch size utilization

Another important parameter determining the training process of the model is the batch size [28].

- Batch gradient descent – backpropagation is performed only once in the whole epoch. Here the network converges to the result very slowly.
- Stochastic gradient descent – batch has a size of 1. Such an approach is prone to entrapment at the local minimum. The network adjusts the weights based on one sample data.
- Mini-batch gradient descent – batch size is smaller than dataset size but larger than 1. The network calculates backpropagation after the end of each batch. Here, the network is still taking informative steps because it has seen several images and at the same time its training does not take as long as with Batch gradient descent. The method is a reasonable trade off between batch gradient descent and stochastic gradient descent.

It is good to keep the batch size in powers of 2 [29]. The pages of memory are always divided into blocks of size 2^n. Such an approach ensures that a batch or several batches are aligned on a single memory page. This makes training more efficient.

Chapter 3

Image segmentation

The chapter contains details about approaches used in the past and the State-of-the-art approachs. It describes architectures and discusses their strengths and weaknesses. It introduces segmentation tools used in medicine and talks about creating a medical dataset.

3.1 Segmentation

It is the process of classifying objects captured in an image. Such segmentation works at the pixel level. The algorithm assigns a value from a set of possible classes to each pixel. Some industries where image segmentation is used:

- Self-driving cars.
- Evaluation of satellite images region.
- Evaluation of medical images.

3.2 Image segmentation in medicine

Segmentation in medicine serves to better detect damage to organs and tissues in the human body. It very often happens that the hospital does not have a sufficient number of specialists who would evaluate the image and prescribe treatment. These images are often difficult to read and only an experienced doctor can determine the exact degree of damage to the body. Another reason to use a segmentation program is the fact that the doctor himself loses concentration during the day and gets tired. This is reflected in the results and thus can affect health of patients.

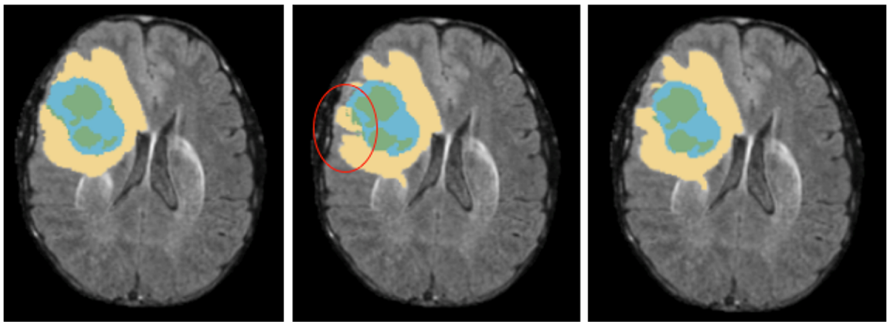


Figure 3.1: These are pictures from the BraTS 2016 challenge dataset, images contain brain tumors.

3.3 Medical datasets

Segmentation of medical images is a very interesting topic, but it is certainly not one of the most popular. The main reasons is the fact that a quality dataset is very difficult to obtain. Unless a competition is announced on one of the datascience sites, it is almost impossible to obtain (if any) data.

Experienced physicians who understand the issue must be involved in data collection process. There is a huge difference whether data is collected from one device or from multiple devices. For medical data, it depends on whether data is obtained locally or if we work with a dataset obtained on a global level.

The very nature of medical data is difficult to grasp. Individual segmentation classes are represented very unevenly, which must be taken into account when training the model. The way the dataset is created is one of the decisive factor influencing the quality of the final model.

3.4 Traditional segmentation approaches

There have been many ways in which segmentation problems have been solved in the past [30]. All methods that originated before the dawn of DNN (deep neural network) era fall into the category of traditional. Such approaches include, for example:

- Histogram of oriented gradients (HOG).
- Scale-invariant feature transform (SIFT).
- Local Binary Pattern (LBP)
- Bag-of-visual-words (BOVW)

A common feature of these approaches is that they extract information from the image into a latent vector. The information stored in this vector can

be presented as a histogram. If the images are similar then they have similar histograms. Latent vectors may provide input for other models (SVM, etc.).

3.5 Deep neural network segmentaion models

In recent years, the computer vision field has become more reliant on the concept of neural networks than handcrafted algorithms. The big breakthrough came with the paper Krizevsky 2012, ImageNet [31]. At that time, many computer vision experts recognized that a neural network with the right architecture and dataset had more potential than manually created filters.

Transition to neural networks came gradually. Huge datasets were once not available, which meant handcrafting and defining algorithms for problem domains.

The neural network works differently. It tries to extract the information contained in the dataset in the form of parameter weights. We rely on new architectures, network sizes and learning parameters rather then on specific algorithm. Thus, the very definition of algorithms, which was essential for computer vision, is less important.

3.5.1 Unet architecture

This architecture [32] was created to segment medical images. Today, we encounter it in segmentation tasks of all types.

It is one of the FCN architectures, thanks to which it can work with data of various sizes. The name itself suggests a lot about this architecture. The network has an encoder-decoder architecture. Information is extracted using convolution filters and maxpooling layers. The convolution filters have size of 3x3. The values from the convolution filters are further fed to the activation function ReLU. Gradual reduction of image dimensions is ensured by pooling of layers. This creates a phenomenon when the dimension of height and width of processed data decreases and the dimension of the number of filters increases (natural for encoder-decoder architecture). After extracting the information, a gradual expansion to the original size of the input data occurs. This is taken care of by upsampling layers analogously to the case of pooling layers. Connections are established between the individual levels of the network. These shortcuts serve to better propagate information over the network. The Unet architecture on the decoder side chains a feature map from the corresponding encoder layers with upsampled data from the previous layer. These links help capture the local structure (important for detail segmentation) as well as the overall information from a larger piece of the image.

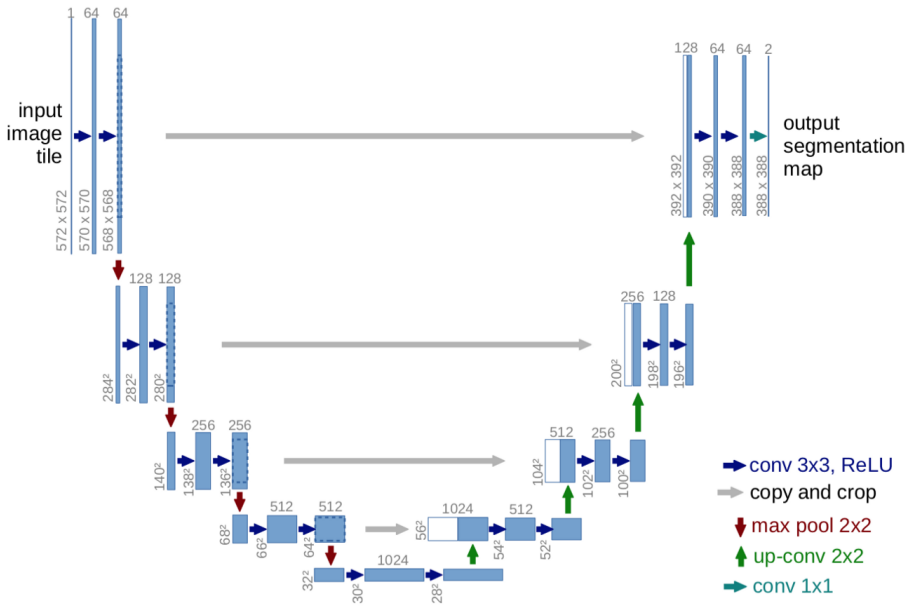


Figure 3.2: Structural architecture of Unet network [32].

3.5.2 Pyramid Scene Parsing Network (PSPNet) architecture

The PSPNet [33] architecture, like Unet, is used to solve segmentation tasks. PSPNet, it is built as an encoder-decoder architecture. The advantage of PSPNet over Unet is that it uses global context information for accurate prediction of images.

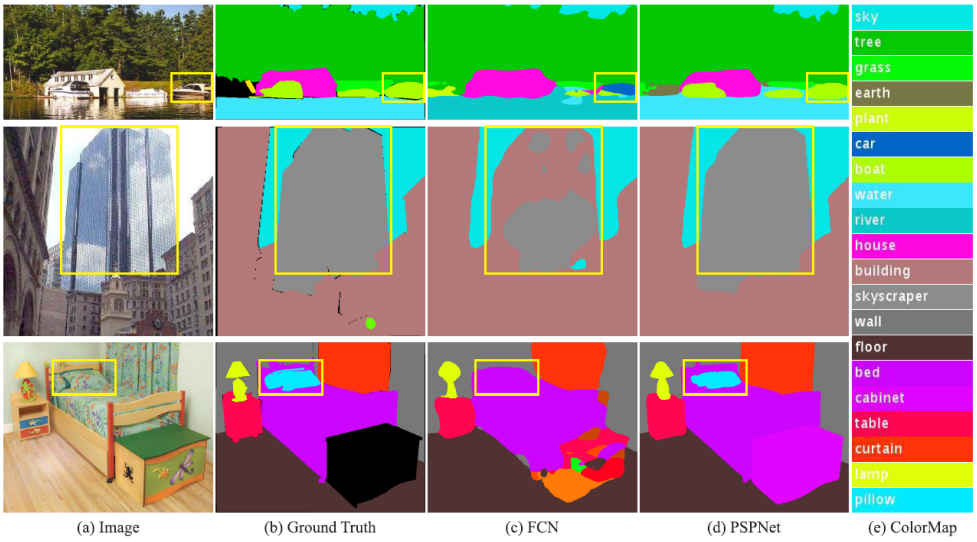


Figure 3.3: PSPNet segmentation comparison [33].

In the first row of 3.3 , the FCN network predicts a motorboat as a car, even though the object is on the water. Thus, most likely, the object should be categorized as a boat. An ordinary network without understanding the context has no chance to know.

In the second row of 3.3, the FCN divides the object (skyscraper) into part of the neighboring building and part of the skyscraper. PSPNet with context information classifies a skyscraper correctly as a whole.

The third row 3.3 shows the segmentation of the children's toy box. FCN considers individual parts as separate objects and classifies them incorrectly. Thanks to the context, PSPNet knows that the box consists of several boards, but it is still one object.

The most important part of PSPNet is the so-called pyramide pooling module. As with Unet, we get a representation of the image from the encoder. We reduce this part in different proportions by using the pooling layer to obtain different densities of captured information. The blocks prepared in this way serve as an input for 1x1, 2x2, 3x3, 6x6 convolution filters. These are responsible for obtaining the context from the image at various levels (from the local to the global context). The output from the convolution filters is enlarged to the required size and concatenated together with the output from

the encoder. The tensor prepared in this way serves for the final creation of a segmentation image.

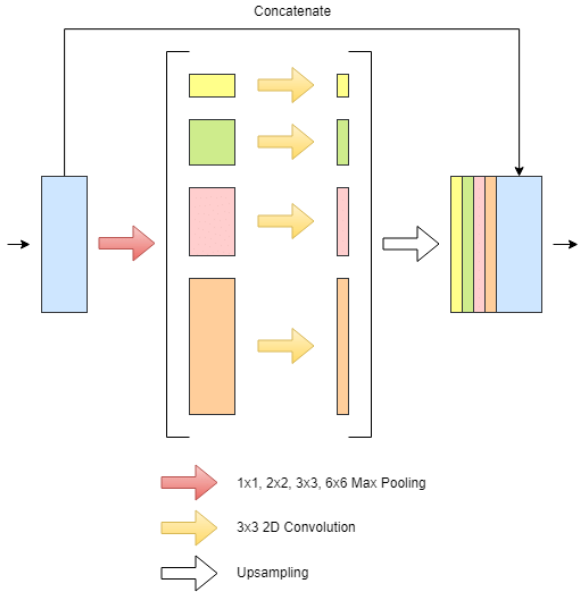


Figure 3.4: Pyramid pooling module (heart of PSPNet) [33].

3.5.3 DeepLabv3+ architecture

DeepLabv3+ [34], like the 2 previous architectures, is used for segmentation tasks. The problem it tries to solve compared to FCN networks is that feature maps become smaller and smaller as information flows through the convolutional and pooling layers. This causes a reduced quality of the predicted results. This issue is addressed by using Atrous Spatial Pyramid Pooling (ASPP). The information from the image is extracted using one of the backbone networks (VGG, DenseNet, ResNet). Context information is obtained using an ASPP block connected behind the backbone network. We concatenate the output from the ASPP block and it then serves as an input for a 1x1 convolution filter which creates a segmentation mask of the image. The DeepLabv3 + architecture adds a section to the decoder that provides better results of segmentation along object boundaries.

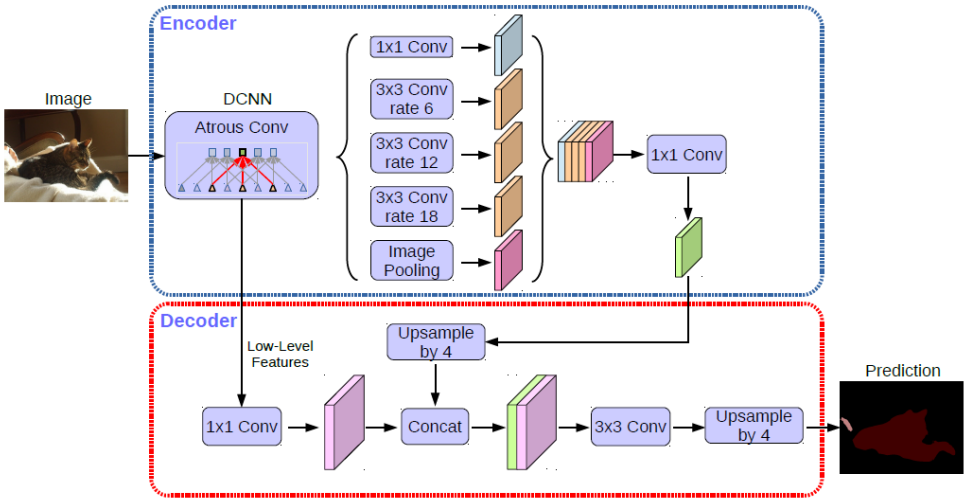


Figure 3.5: Deeplabv3+ [34].

3.5.4 DenseUnet architecture

This architecture [35] uses an idea that worked very well in a classification task using DenseNet. When creating the architecture, the authors realized that layers that are directly concatenated, transmit information better and the whole network is easier to train. The set of such directly chained layers forms a dense block. Within the concatenated layers in the dense block, we obtain excess information that has a positive effect on the accuracy of the model.

The architecture of DenseUnet is very similar to the architecture of Unet. However, convolution blocks are replaced by dense blocks. The network therefore takes advantage of Unet (architecture of encoder-decoder) and dense blocks (a significant feature of DenseNet).

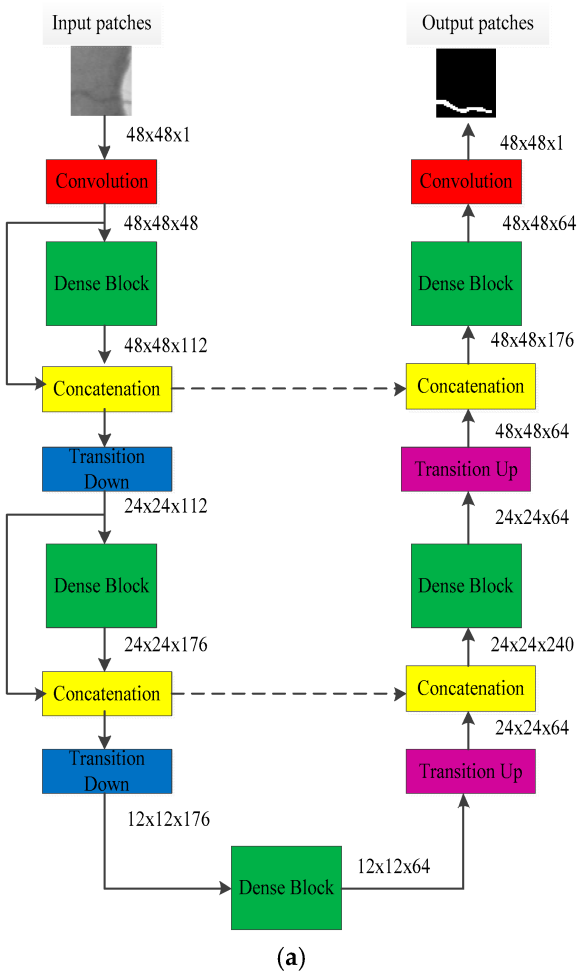


Figure 3.6: DenseUnet [35].

3.6 Loss functions used for segmentation

A large number of metrics are used for segmentation tasks[36]. Some of them are:

Cross Entropy (CE)

$$-(y \log(\hat{y}) + (1-y) \log(1-\hat{y})) \tag{3.1}$$

Balanced cross entropy (BCE)

$$-(\beta y \log(\hat{y}) + (1-\beta)(1-y) \log(1-\hat{y})) \tag{3.2}$$

This loss is used when the representation of classes in a dataset is not balanced.

Dice coefficient (DC)

$$\frac{y\hat{y}}{y\hat{y} + \frac{1}{2}(1-y)\hat{y} + \frac{1}{2}y(1-\hat{y})} \tag{3.3}$$

Jaccard coefficient (JC)

$$\frac{y\hat{y}}{y\hat{y} + (1-y)\hat{y} + y(1-\hat{y})} \tag{3.4}$$

DC, JC metrics are used to estimate the similarity and overlap of the samples.

Dice coefficient loss (DCL)

$$1 - DC(y, \hat{y}) \tag{3.5}$$

Jaccard coefficient loss (JCL)

$$1 - JC(y, \hat{y}) \tag{3.6}$$

Softmax dice loss (SDL)

$$\alpha CE(y, \hat{y}) + \beta DCL(y, \hat{y}) \tag{3.7}$$

Tversky index (TI)

$$\frac{y\hat{y}}{y\hat{y} + \beta(1-y)\hat{y} + (1-\beta)y(1-\hat{y})} \tag{3.8}$$

Tversky index is generalization of the Dice coefficient. TI adds the alpha, beta parameter to balance FP and FN samples.

- If $\alpha = \beta = 0.5$ then tversky index is equal to dice coefficient
- If $\alpha = \beta = 1.0$ then the tversky index is equal to the jaccard coefficient
- If $\alpha + \beta = 1.0$ then tversky index is equal to F1 score

Chapter 4

Dataset

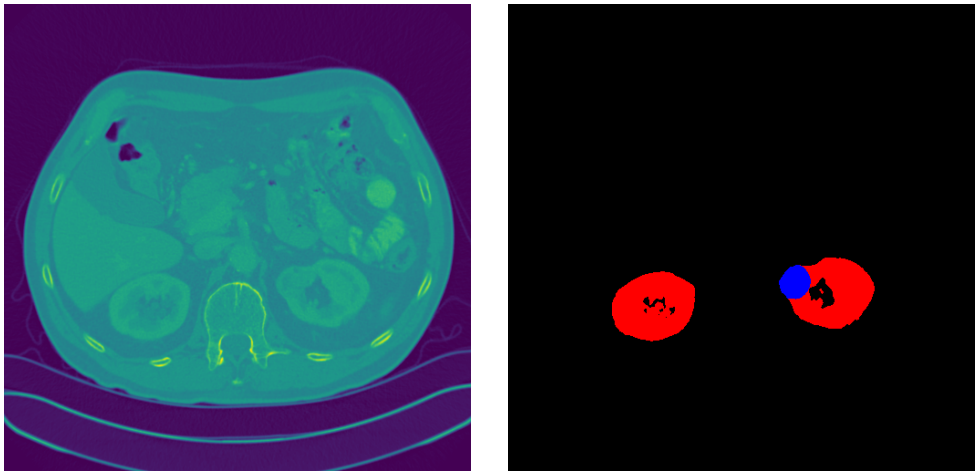
This chapter is dedicated to the dataset. There will described basic information about the dataset, the data format it contains, data preprocessing, data augmentation and the creation of an advanced data generator adapted for distributed training.

4.1 Dataset Kits19

All data images [37] used for training were obtained from the KiTS19 Challenge [38]. It is one of many machine learning competitions involving medical segmentation enthusiasts from many countries around the world.

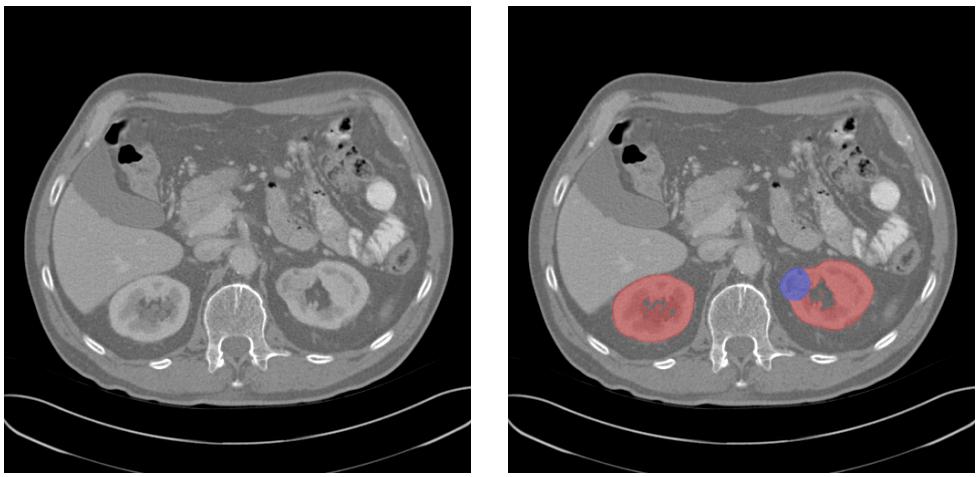
The dataset [39] provided by the organizers contains information on individual patients suffering from kidney cancer. Such a set consists of several CT images on which transverse cuts of the patient's abdominal cavity are captured. The thickness of these cuts is in the range of 1 mm - 5 mm, while within one patient this thickness is constant. In addition, the file contains segmentation data mask. These are pictures on which the kidneys and tumors are marked. All other organs in the images are painted black and represent background. Data were provided by the University of Minnesota Medical Center.

All files that contain images and ground truth labels are anonymized and saved in NIFTI format. The data file for one patient contains data in the format $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where $x_k \in \mathbb{R}^{512\times512\times3}$ and $y_k \in \mathbb{R}^{512\times512\times3}$, $n = 45965$, x represents patient images and y represents ground truth segmentation masks of the background, kidneys and tumor. The whole dataset consists of 210 patients.



(a) Original picture from CT. (b) Segmentation mask.

Figure 4.1: CT scan of a patient and the segmentation of the background (black), kidney (red), tumor (blue).



(a) CT scan in gray hue. (b) After application of segmentation mask.

Figure 4.2: Data sample in gray hue after applying the segmentation mask.

4.2 Data preprocessing

When training and testing new models, iteration speed through possible architectures and parameters is very important. That is why all data samples have been reduced from 512x512x3 to 128x128x3. Subsequently, all images were saved in .npy format.

The OpenCV library was used to reduce dimension of the images. Several interpolation methods were tried and compared their results and adjusted them to correspond to the values in the original dataset. The reason we focused on different interpolation methods is the possible loss of information during transformations. Looking at the picture 4.1b, we can see that the tumor itself is much smaller compared to other objects. Such interpolation can negatively affect the quality of the CT scan dataset itself. If a tumor has specific features that distinguish it from other objects, then interpolation can destroy this information.

All images that do not contain a kidney or tumor are removed from the dataset during training. Models that were trained on data without this adjustment very often converged to black, because everything else (kidneys and tumors) was considered an anomaly and the weights of the models did not adapt to such an object.

The dataset undergoes further modification if we decide to train a 3D model. Then, after removing the irrelevant images, 32-image blocks are created for each patient. Each such block has an overlap of 12 frames with the previous data block. The overlay ensures that the network can work with information that was processed in the previous cycle of the forward propagation algorithm. Thus, the overlay represents something like a context. After this processing, the blocks from all patients are concatenated to form a new dataset $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where $x_k \in \mathbb{R}^{32\times128\times128\times3}$ and $y_k \in \mathbb{R}^{32\times128\times128\times3}$, without any other preprocessing n = 691.

4.3 Data augmentation

In tasks where we work with a large dataset, the DataGenerator class, which inherits properties from keras.utils.Sequence, is useful. Such generator ensures parallel processing. The data is read from the disk, augmented and the CPU sends it directly to the GPU. This principle speeds up training when more graphics cards and more CPU cores are available. DataGenerator[40] class provides this parallel data feeding to the model.

DataGenerator also takes care of the augmentation itself. Data images are horizontally flipped, they are rotated in the range $(-5°, 5°)$, zoom is used in the range (-5%, 5%) of the image size, the image is shifted in the range (-10px, 10px) in the horizontal and in the vertical direction. Such postprocessing of input data will ensure greater robustness of training process and the models.

Another method used for dataset enrichment and class balancing is GAN (generative adversarial network). This is different augmentation method. There is no modification to an image that is already in the dataset. Conventional generators use rotation, scaling or flipping which helps very little if the dataset contains very few samples or the classes are unbalanced. The GAN approach is based on the random selection of an image from a distribution that

the generative part of the GAN learnt during training. A discriminator is a network that is pre-trained on a dataset with a similar problem domain and fine-tuned on our dataset.

GAN augments the dataset by creating new unique samples. An ordinary generator augments already created samples and partially changes them.

Chapter 5

Image segmentation in practice

The chapter provides information about the implementation process.

5.1 Path of implementation

Myna development environments were used and the technologies were adapted to the requirements of the task. A base machine learning pipeline was created. It is the easiest way to create a working solution and gradually extend modules. Such an approach is very common in prototyping.

Google Colab was the first prototyping environment. It contains all the necessary libraries and in case some is missing, it can be easily installed. This was the place where the dataset was preprocessed and images were reduced to a reasonable size. Such resized dataset was saved into multiple .npy files.

After working with the dataset, work continued with prototying model, creating data generator. The first implemented architecture was Unet. It is a proven approach to solving segmentation tasks. The advantage of such a model is its simplicity. It can be very easily prototyped and modified.

After creating the model, it is very important to test whether specific parameters of the model (number of filters, filter size, regularization,…) will work. It was necessary to work towards a model that can reasonably capture the information contained in the dataset but will not contain unnecessarily many learning parameters.

We used data of 5 patients for first prototyping. There were a reasonable number of images (about 5% of the dataset) which helped to experiment much faster. It was necessary to create an architecture that would learn structures from the data. This served as proof that a particular model could extract information from the dataset. After a while, we get to a model that was satisfactory.

However, it was not the task to train only one model but compare a set of approaches and choose the best one. This was the main reason we had

to find replacement for Google Colaboratory. The environment had a few disadvantages in this regard. The first was the fact that free account was used and the training time for the model was limited. Another disadvantage is the number of GPUs available for a free user.

There were 2 paths we could take in next development. Use Tensorflow (MirroredStrategy) or Horovod. Both of these technologies serve as a tool for parallelizing calculations on a CPU or GPU. Horovod was chosen as the final technology, see 5.1. Horovod [41] is a project built on top of OpenMPI. OpenMPI is a technology that solves High Performance Computing problems. Implements distributed reduce function *allreduce*. This function is used to aggregate tensors during the calculation and effectively provide them to other processes.

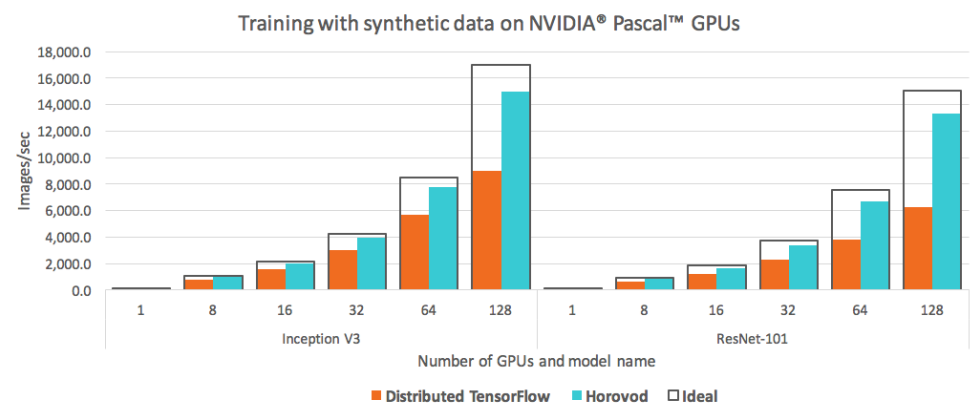


Figure 5.1: Efficency of parallel training with Horovod vs Tensorflow [41].

We adapted the development environment just for the needs of HPC. NCCL (NVIDIA Collective Communications Library) has been installed. This library implements the multi-GPU and multi-CPU messaging needed to train distributed models. The instalation instructions are very well described on the Horovod official website.

During the transition to Horovod technology, a few adjustments had to be made to the training script. The most important thing to realize is that training and overall processing run in parallel, and any bottleneck can cause desynchronization of the entire process.

One of the problems was the uneven distribution of the dataset among workers. If one of the workers ends too soon or too late, the training process is desynchronized. This problem is solved by writing a custom data generator in the chapter 4.3. The desynchronization of the distributed optimizer was also caused by the batch size being too large.

This whole process led to the creation of a deep learning pipeline, which provided the necessary speed in developing and experimenting with new models. There were implemented and trained 6 other models via this pipeline.

5.2 Implementation of models

In thesis we experimented and compared results of architectures Unet2D, Unet3D, PSPNnet, DenseUnet, Deeplabv3+.

When creating models, we divided them into two categories according to the number of learning parameters. If we compare the performance of 2 models that have same architecture and differ only in the number of learning parameters, then a larger model should have better results. That was exactly the reason why we decided to create 2 versions for PSPNet and DenseUnet networks. The input of all neural networks is modified to be able to receive images with a size of 128x128x3.

5.2.1 Unet2D

This network has been optimized the most. The total number of parameters is 2,465,619. The network has a very similar architecture to the network in the picture 3.2. However, it has only 2 levels of maxpooling and upsampling (the network in the picture has 4). The network uses a set of convolutional blocks connected by maxpooling layers (used in the encoder part) and upsampling layers (used in the decoder part). Maxpooling / upsampling use pooling_size = (2, 2). The number of convolution filters at the i_{th} level of the convolution network is always $64 * 2^i$, where $i \in [0, 1, 2]$. Convolution filters in convolution blocks have dimensions (3×3). In the last layer there is a convolution filter (1×1), which serves as a replacement for the dense layer. Thanks to that, this network is FCN. The outputs of the convolution blocks from the encoder are concatenated with the inputs of the convolution blocks of the decoder. This ensures better spread of information and better contextual understanding.

The convolutional block consists of two consecutive pairs of convolutional filter and applied batch normalization . The deeper the block is in the network, the larger its receptive field.

The activation function of convolutional layers is ReLU. LeakyReLU was tested on several networks, but its effect, even though it has a gradient other than 0 in negative numbers, was negligible. In the last layer is the activation function softmax, which is used for multiclass classification.

5.2.2 Unet3D

The total number of parameters is 15,296,451. A network that has almost the same architecture as the previous one 5.2.1. The only difference is in the dimensions of the input data and the dimensions of the filters. The input data is in the form $\mathbb{R}^{patchsize \times 128 \times 128 \times 3}$ where patch size is the size of a 3D block of several images on top of each other. An overlay of several images from the previous patch was used to create patches. This provided the context information

for the 3D network, which led to better results. The network consists of the same number of convolutional blocks and uses the same activation function as in 5.2.1.

The only difference is in the size of the convolution and pooling filters. Unet3D uses convolution filters that have a size $(3 \times 3 \times 3)$ and pooling filters that have a size $(2 \times 2 \times 2)$. The number of convolution filters at the i_{th} level of the convolution network is always $48 * 2^i$, where $i \in [0, 1, 2]$. This size is limited by the size of the RAM.

5.2.3 PSPNet

There were implemented two networks (PSPNet_big, PSPNet_small). The first network contains 46,765,763. The smaller network contains 1,651,331 parameters.

In PSPNet_b, ResNet50 serves as an encoder. In PSPNet_s, a customized block serves as an encoder in such a way that the whole network has a comparatively large number of parameters as Unet2D. For a smaller network, the output tensor from the encoder has dimensions $(32 \times 32 \times 256)$. For a larger network, the output from the ResNet encoder has dimensions $(60 \times 60 \times 4096)$. The decoder consists of a pyramid pooling module, see 3.5.2. In pyramid pooling, we have empirically opted for the sizes of the average pooling layers. 4 such blocks are created. The i_{th} block contains the average pooling layer where kernel size $= (2^{4-i}, 2^{4-i})$ and stride $= (2^{4-i}, 2^{4-i})$ where $i \in [0, 1, 2, 3]$.

After applying convolution filters, these tensors are enlarged to the size of the input tensor. Such a tensor is the input from the last layer. The last layer is responsible for the multiclass classification.

5.2.4 DenseUnet

There were implemented two networks (DenseUnet_big, DenseUnet_small). The first network contains 46,765,763. The smaller network contains 1,651,331 parameters. Both networks use the Unet 5.2.1 architecture supplemented by dense blocks. They take care of the gradual extraction of information from the input and then concatenate it with the input to the next part of the dense block.

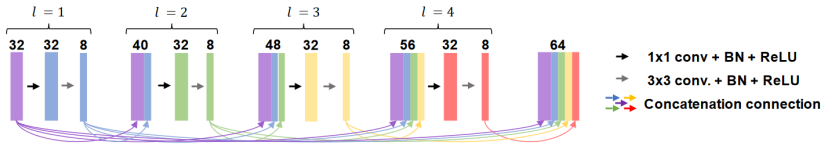


Fig. 4. Four layered dense block with $k_1 = 8$ and $F = 32$. Feature-maps from previous layers are concatenated together as the input to following layers.

Figure 5.2: DenseUnet connections [35].

In our case, DenseUnet_s works with 4 subblocks within one dense block as in the picture (l = 4). However, the difference is in the size of the number of output filters. In the models, the size that each additional dense block works with changes. Let D_i be a certain dense block used in the network. Then the size of the outputs is $4(2^i)$ where $i \in [0, 1, 2, 3, 4]$, i is the order of the dense block in the architecture from the input layer. Transition up blocks are responsible for retrieving information from the previous block and increasing the image size using the upsampling layer. Transition down only use maxpooling instead of upsampling layers and reduce the image dimensions. My classmate Uladzislav Yorsh implemented DenseUnet_s.

5.2.5 Deeplabv3+

This network contains 2,142,019 learning parameters. The network uses Atrous Convolution approach [34]. It is a special kind of convolution when the filter does not necessarily replicate pixels next to each other but sets atrous rate. If we concatenate the output of such filters, then ASPP arises. Filters thus capture information at the local and global level. In my work, atrous rates has values of 6, 12, 18. The architecture of the MobileNetv2 network was used as a backbone. The reason is its very good compactness and reasonably small number of parameters comparable to other architectures.

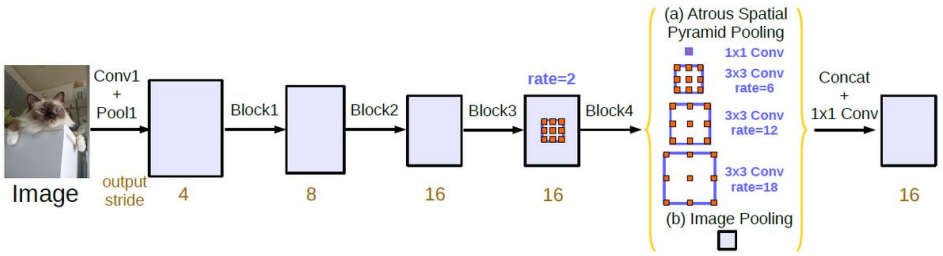


Figure 5.3: Deeper insights about Deeplabv3+ architecture[34].

5.3 Loss functions

During the training of neural networks, we experimented with all the loss functions mentioned in 3.6. First, we began experimenting with functions based on the CCE approach. Such a loss function helps model if the model is too confident in predicting incorrect values. Another reason is the fact that the gradients calculated during backpropagation for CCE are much more stable compared to DCL [42]. CCE did not appear to be beneficial after a long period of experimentation. All networks converged to a background color (black) with no visible signs of kidney or tumor.

Another used loss function was SDL. In this loss function, we combined the CCE approach, which was to ensure fast and smooth convergence, especially from the beginning of learning, and DCL will take care of the overall result and training in the last epochs. In this loss function, we looked for α and β parameters that would ensure convergence as quickly as possible with the best possible result. When using this loss function, some models converged to the background color but small red islands representing the kidney began to form.

The next used loss function was pure DCL. The results were very similar, most models returned black images with a rarely occuring red color. DCL_w with balanced classes was used as the final loss function.

5.4 Training

The whole training process had several levels. Each level brought some improvements and a more accurate model. The dataset was divided into training, validation and testing data. The training set contained 180 patients. The validation set contained 20 patients and the test set 10 patients.

The first step was to train the models on 5% of the data and evaluate the results. If the network was able to capture information, we did not utilize the architecture any further. If architecture could not even converge to the color of the background itself, it was clear that such an architecture could not

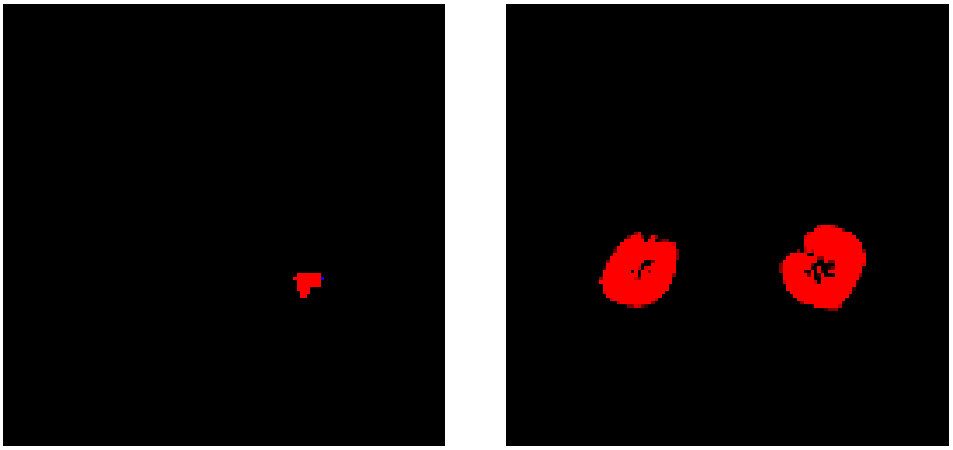
capture enough information and the model had to be modified. For example, by adding the number of filters in the convolution layer.

Another important factor during network training was the presence of batch normalization. Initially, the models were trained without this layer. At that time, networks were very sensitive to changes in the learning rate. Immense sensitivity to small changes in learning rate made it very difficult to find the right hyperparameters of the models and slowed down the whole development process.

After applying batch normalization, a higher learning rate value can be used. The network converges faster and is less sensitive to small changes in this parameter. Learning is more stable and the network becomes more resilient.

After these adjustments, the networks converged to good results on a 5% dataset. It was possible to recognize the kidney. However, after training the networks on the entire dataset, the results were very weak. They all converged to a black background color.

Here we started experiments with various loss functions. From CCE loss, weighted CCE loss, focal loss, SDL, to weighted DCL. The optimization of loss functions at this stage of development had almost no impact on the resulting accuracy of the model. The resulting models predicted only images containing background and only very rarely a piece of kidney.



(a) Models in this stage converged very poorly.

(b) Real segmentation mask.

Figure 5.4: The images show difference between prediction of models and real segmentation mask.

The reason why the models had a problem regardless of used architecture and the loss function was in the dataset.

5.4.1 Dataset balancing

The Kits19 dataset contains sequences of images of the patient's abdominal cavity. The image distribution is captured in the image 5.5.

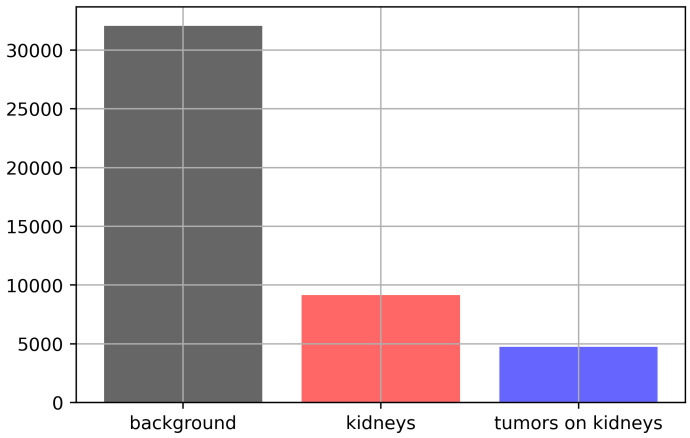


Figure 5.5: Distribution of classes in dataset.

Background images are data samples that contain only black color (background). No relevant data containing kidneys or tumors. In images where is a background as well as kidney or tumor, the distribution of information is different.

In such images, the kidneys occupy about 3% -6% of the entire image. The image contains about 95% black (background) and about 5% useful information (kidney). The tumors in the images are much smaller than the kidneys (about 0.5% - 1% of the information in the picture). Another important thing is that tumors occur much less frequently than the whole kidney.

In the case of such datasets, it is extremely important to balance classes. If the original distribution is kept, the model will be optimized for color of background because this is the best way to minimize the loss function. This is why all loss functions and all models have converged to backgrounud color. No model learnt anything about kidney or tumor because such data was so scarce that they were considered to be outlayers and anomalies to which the models did not optimized.

All images that contained only the background were removed from the datadataset. Only part of the data containing only kidneys and kidneys with tumors was left. The combination of balancing the dataset and adding class weights to the loss function resulted in models starting to segment the kidneys.

Tumors have not appeared on any of the resulting segmentation images yet. Beautiful kidney segmentations could be seen. To obtain a tumor, we decided

to weight classes in the newly formed dataset. We created 3 new parameters $\alpha_{background}, \beta_{kidney}, \gamma_{tumor}$. We multiplied the weights obtained from the sklear.utils.class_weights() function with these parameters.

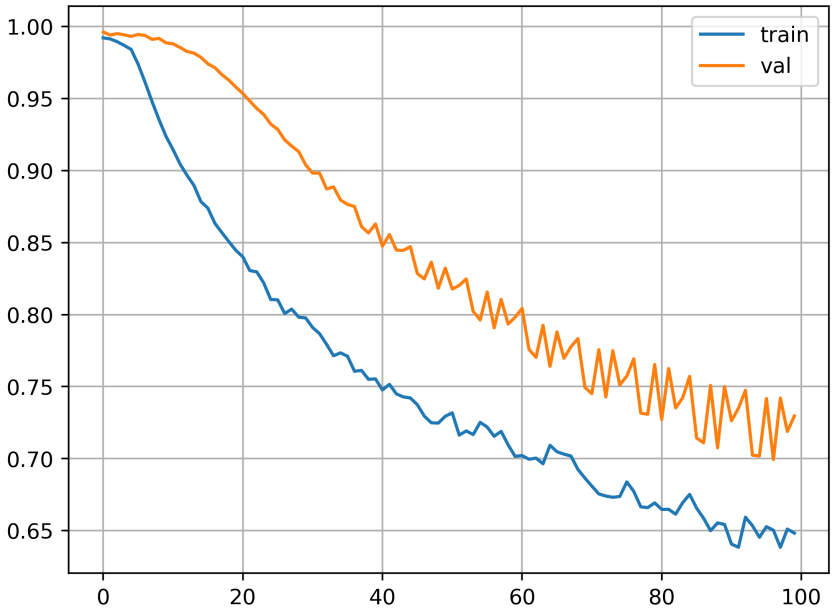


Figure 5.6: Training and validation loss when using only weights.

Such a multiplication brought us to another problem. It caused the training and validation losses to oscillate. The oscillations were greater when the model began to learn features of the kidney and tumor.

Background is still the most represented component. The model converges to black very easily. It has to make small changes when optimizing weights for a kidney or tumor, but it is very difficult to do so because class weights are used. Due to the large weights of classes, the steps during loss function optimization are too large and the model overshoots the local minimum. The principle is very similar to models where L2 regularization is not used. Then weights with huge values can occure and training is not stable. In the models where L2 is used, the weights are reasonably large and a small change will not cause a huge instability in training. It is therefore not enough just to outweigh the representation of individual classes but to rebalance the dataset classes again.

The dataset had to be modified again. From the dataset, which contained only kidneys and kidneys with tumors, images containing only tumors were selected, duplicated several times and added to the dataset. Such a balanced

dataset was used to train the latest models.

Very interesting phenomenon appeared during training of some models. Sometimes training error was lower then validation error. We know from theory that a well-trained model has a slightly lower loss on training than on validation data. If this error between the training loss and the validation loss is huge then the model overfits the data (solution is to use Dropout, L2, EarlyStopping, etc.). If both the training and validation error is high, then the model cannot capture information from the data (the solution is a larger model, learning rate utilization, etc.). However, there was a case when the loss on the training data was greater than on the validation data (even if no regularization or dropout was used in the models). Such behavior can be caused by augmentation of the training set. During training, the images from the training set are augmented. Thanks to augmentations, data that are similar to the validation ones are created. The validation set thus becomes a subset of the training set. This fact can cause a lower validation loss than a training loss.

5.5 Results

Weighted dice coefficient loss (DCL_w) was used to train the final models. Such a loss function converged reasonably fast and with the same parameters gave the best results compared to other loss functions from 3.6. Weighted jaccard coefficient (JC_w) was chosen as a metric determining the final quality of the model. The best model was considered to be the one with the lowest error on the validation set when using JC_w.

Network	JC_w
PSPNet_s	0.8566
Unet2D	**0.8633**
DenseUnet_s	0.8522
Deeplabv3+	0.8281

Table 5.1: Performance of smaller models on testing data.

Network	JC_w
PSPNet_b	0.8654
Unet3D	**0.9113**
DenseUnet_b	0.8639

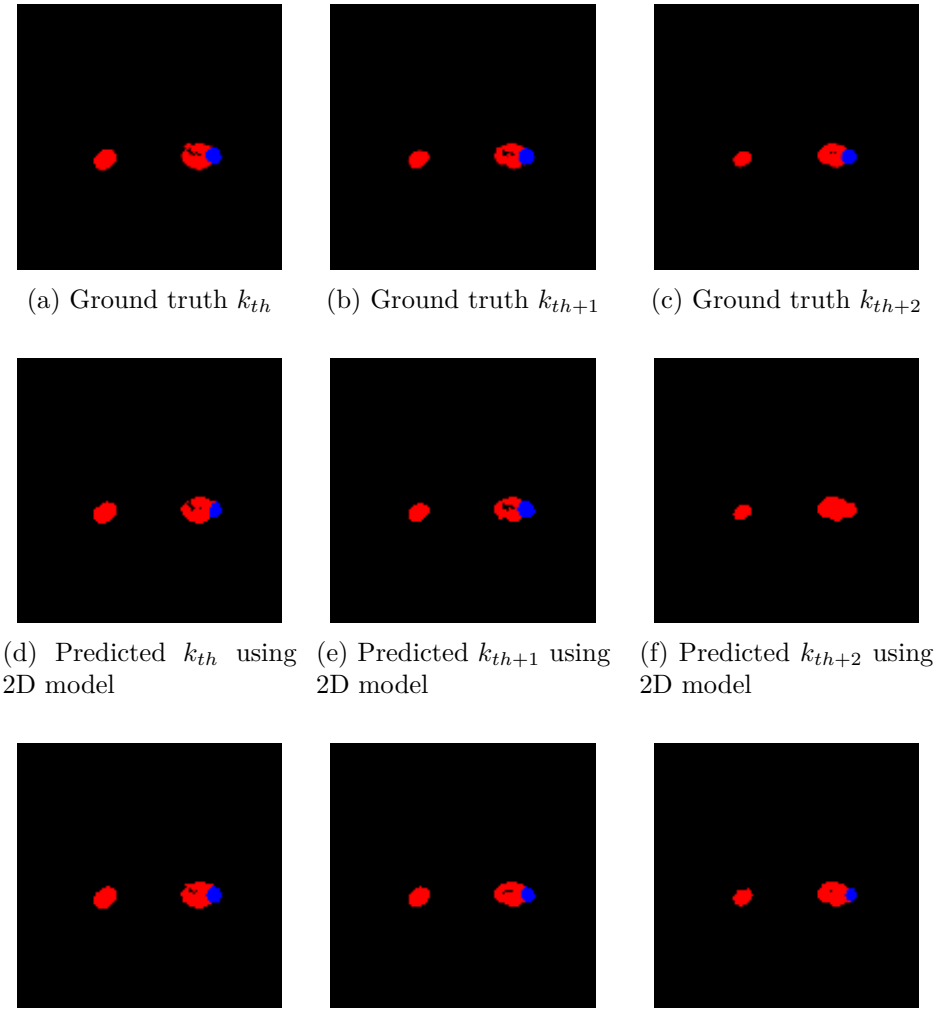
Table 5.2: Performance of bigger models on testing data.

The fact that the Unet3D and Unet2D networks were optimized to the greatest extent may have contributed to the result. Unet2D network had the highest JC_w within the smaller models 5.1. Within models with a larger number of parameters, Unet3D won 5.2. This was the expected result for Unet3D. The architecture was designed to capture spatial information. Such a model can estimate the required tumor shape when processing image sequences, even if some data in the sequence are ambiguous and cannot be accurately determined using information from a single image. Thanks to 3D filters, the network was able to capture context that other networks could not.

Figure 5.7 shows tumor segmentation using Unet2D and Unet3D. The network has converged to a state where it is able to capture the tiny structures of the kidneys and tumor. Unet2D is not able to capture information because it works with individual images. Segmentation inaccuracy can be seen in figure 5.7f where 2 previous photos was segmented correctly. The last picture of sequence with the tumor was not captured. In contrast, Unet3D on 5.7i recognized everything correctly.

Unet3D had a problem recognizing smaller tumors. This can be due to the higher density of information or very specific shape of such tumors.

Additional patient segmentations are created and stored in the appendix using 2D and 3D models on test dataset.



(a) Ground truth k_{th} (b) Ground truth k_{th+1} (c) Ground truth k_{th+2}

(d) Predicted k_{th} using 2D model (e) Predicted k_{th+1} using 2D model (f) Predicted k_{th+2} using 2D model

(g) Predicted k_{th} using 3D model (h) Predicted k_{th+1} using 3D model (i) Predicted k_{th+2} using 3D model

Figure 5.7: Segmentation with 2D model.

5.5.1 Implementation details

All trained networks are stored in separate files corresponding to their names. When creating scripts, emphasis was placed on the possibility of reproducibility of results. In the development environment, all used libraries were initialized with a seed value, which can be clearly seen at the beginning of training script. The results of the training process are reproducible. The absolute determinism of the training process was not ensured as the pipeline works and initializes several GPUs. The allocation of such resources within a cluster is not deterministic.

Conclusion

In this work, we focused on creating, training and comparing the performance of models designed for segmentation of organs and cancer. We created and trained models that are used when working with medical diagnostic scans.

The aim of the theoretical part was to introduce the reader to the types of diagnostic images used in medicine nowadays. This goal is fulfilled in the chapter Medical background 1.

Another objective was to describe the basic concepts and principles within neural networks, to address the issue of segmentation and to describe the architecture of models designed for this type of task. The goal is met in the chapter Machine learning background 2 and Image Segmentation 3.

Chapter 4 was devoted to the preparation of the dataset.

In the practical part 5, there was described the design of solutions with a discussion of advantages and disadvantages of each model. In this section, segmentation models were implemented and trained. From all of the models, Unet3D came out as the most accurate model. The results of segmentation on the test data were above expectations. This model was able to segment tumors and kidneys with great accuracy. However, during segmentation, images that contained very small tumors were segmented as kidneys. This could be improved by training the model on the original dataset that has a higher resolution than the dataset on which these models were trained. Another possibility for improvement is the use of GAN for dataset augmentation.

Another goal was to ensure the reproducibility of the solution. This requirement was met thanks to a modification of the training script. All parameters of the dataset and the trained network are stored in a file and the imported libraries in which randomness is used are initialized by a predefined seed. The next step was to publish the source code as an open source https://github.com/HypnoOcio/kits19.

Outline of future work

In the future, I would like to focus on training networks on the original image size (samples x 512 x 512 x 3). This will include modifying preprocessing scripts and modifying neural network architectures. To achieve even better results, it is worth considering combining a 3D network with elements of other State-of-the-art architectures.

Another important factor is the dataset. For augmentations, it would be interesting to use the CycleGAN network together with the Keras generator and compare the results on such trained models. Such an approach could again provide improved model performance and provide a base line for segmentation in other medical applications where the dataset is small.

Bibliography

1. LI, Oscar. Artificial Intelligence is the New Electricity — Andrew Ng. *Medium* [online] [visited on 2020-05-25]. Available from: https://medium.com/syncedreview/artificial-intelligence-is-the-new-electricity-andrew-ng-cc132ea6264.
2. X-ray. *MedicalNewsToday* [online] [visited on 2020-04-14]. Available from: https://www.nhs.uk/conditions/x-ray/.
3. How does a CT or CAT scan work? *MedicalNewsToday* [online] [visited on 2020-04-14]. Available from: https://www.medicalnewstoday.com/articles/153201.
4. MRI scan. *NHS* [online] [visited on 2020-04-14]. Available from: https://www.nhs.uk/conditions/mri-scan/.
5. BITTENCOURT, Paulo. CT perfusion imaging takes over the scene in acute stroke. *Dimpna* [online] [visited on 2020-05-26]. Available from: http://www.dimpna.com/2020/01/14/ct-perfusion-in-stroke/.
6. High-Field Wide Bore MRI 1.5T. *Forest City* [online] [visited on 2020-05-26]. Available from: http://www.forestcitydi.com/services-mri.html.
7. AMIDI, Afshine; AMIDI, Shervine. Supervised Learning cheatsheet. *Standford* [online] [visited on 2020-04-14]. Available from: https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-supervised-learning.
8. AMIDI, Afshine; AMIDI, Shervine. Unsupervised Learning cheatsheet. *Standford* [online] [visited on 2020-04-14]. Available from: https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-unsupervised-learning.
9. NARKHEDE, Sarang. Understanding Confusion Matrix. *Towards Data Science* [online] [visited on 2020-05-26]. Available from: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

10. AGARWAL, Rahul. The 5 Classification Evaluation metrics every Data Scientist must know. *Towards Data Science* [online] [visited on 2020-04-14]. Available from: https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226.

11. NARKHEDE, Sarang. Understanding AUC - ROC Curve. *Towards Data Science* [online] [visited on 2020-05-31]. Available from: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

12. FUMO, David. A Gentle Introduction To Neural Networks Series — Part 1. *Medium* [online] [visited on 2020-04-14]. Available from: https://towardsdatascience.com/a-gentle-introduction-to-neural-networks-series-part-1-2b90b87795bc.

13. BRE, Facundo; GIMENEZ, Juan; FACHINOTTI, Víctor. Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks. *Energy and Buildings*. 2017, vol. 158. Available from DOI: 10.1016/j.enbuild.2017.11.045.

14. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. In: *Deep learning: Convolutional Networks*. Cambridge, Massachusetts: The MIT Press, 2017, pp. 105–116. ISBN 978-0262035613.

15. VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N; KAISER, Łukasz; POLOSUKHIN, Illia. Attention is all you need. In: [online] [visited on 2020-05-26]. Available from: https://arxiv.org/abs/1706.03762.

16. SZEGEDY, Christian; LIU, Wei; JIA, Yangqing; SERMANET, Pierre; REED, Scott; ANGUELOV, Dragomir; ERHAN, Dumitru; VANHOUCKE, Vincent; RABONOVICH, Andrew. Going deeper with convolutions. In: [online] [visited on 2020-05-26]. Available from: https://arxiv.org/abs/1409.4842.

17. BROWNLEE, Jason. How Do Convolutional Layers Work in Deep Learning Neural Networks? *Machine Learning Mastery* [online] [visited on 2020-05-30]. Available from: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/.

18. POKHARNA, Harsh. The best explanation of Convolutional Neural Networks on the Internet! *Medium* [online] [visited on 2020-06-04]. Available from: https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8.

19. AKSA, Neo. Types of Convolution. *Doing Interesting Things in Life* [online] [visited on 2020-05-26]. Available from: https://www.jie-tao.com/types-of-convolutionstranslation/.

20. HUI, Jonathan. Types of Convolution. *Jonathan Hui blog* [online] [visited on 2020-05-26]. Available from: https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/.

21. MIRJALILI, Vahid. What does the "same" padding parameter in convolution mean in TensorFlow? *Quora* [online] [visited on 2020-05-26]. Available from: https://www.quora.com/What-does-the-same-padding-parameter-in-convolution-mean-in-TensorFlow.

22. ZEILER, Matthew; FERGUS, Rob. Visualizing and Understanding Convolutional Networks. In: [online] [visited on 2020-05-26]. Available from: https://arxiv.org/abs/1311.2901v3.

23. GROSSE, Roger. Attention is all you need. In: [online] [visited on 2020-05-26]. Available from: http://www.cs.toronto.edu/~rgrosse/courses/csc321_2017/readings/L15%20Exploding%20and%20Vanishing%20Gradients.pdf.

24. SIRIGNANO, Justin; SPILIOPOULOS, Konstantinos. Scaling Limit of Neural Networks with the Xavier Initialization and Convergence to a Global Minimum. In: [online] [visited on 2020-05-27]. Available from: https://arxiv.org/abs/arXiv:1907.04108v2.

25. SKORSKI, Maciej. Revisiting Initialization of Neural Networks. In: [online] [visited on 2020-05-26]. Available from: https://arxiv.org/abs/2004.09506.

26. SZEGEDY, Christian; IOFFE, Sergey. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: [online] [visited on 2020-05-29]. Available from: https://arxiv.org/abs/1502.03167v3.

27. HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Deep residual learning for image recognition. In: 2016. Available also from: https://arxiv.org/abs/1512.03385.

28. BROWNLEE, Jason. How to Control the Stability of Training Neural Networks With the Batch Size. *Machine Learning Mastery* [online] [visited on 2020-05-30]. Available from: https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/.

29. MRK. Is using batch size as 'powers of 2' faster on tensorflow? *stackoverflow* [online] [visited on 2020-06-01]. Available from: https://stackoverflow.com/questions/44483233/is-using-batch-size-as-powers-of-2-faster-on-tensorflow.

30. LIU, Xiaolong; DENG, Zhidong; YANG, Yuhan. Recent progress in semantic image segmentation. In: 2018. Available also from: https://arxiv.org/abs/1809.10198.

31. KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. In: PEREIRA, F.; BURGES, C. J. C.; BOTTOU, L.; WEINBERGER, K. Q. (eds.). *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, pp. 1097–1105. Available also from: `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

32. RONNEBERGER, Olaf; FISCHER, Philipp; BROX, Thomas. U-Net: Convolutional Networks for Biomedical Image Segmentation. In: [online] [visited on 2020-05-27]. Available from: `https://arxiv.org/abs/1505.04597`.

33. ZHAO, Hengshuang; SHI, Jianping; QI, Xiaojuan; WANG, Xiaogang; JIA, Jiaya. Pyramid Scene Parsing Network. In: [online] [visited on 2020-05-27]. Available from: `https://arxiv.org/abs/1612.01105`.

34. LIANG-CHIEH, Chen; YUKUN, Zhu; GEORGE, Papandreou; FLORIAN, Schroff; HARTWIG, Adam. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. In: [online] [visited on 2020-05-27]. Available from: `https://arxiv.org/abs/1802.02611`.

35. SIMON, Jegou; MICHAL, Drozdzal; DAVID, Vazquez; ADRIANA, Romero; YOSHUA, Bengio. The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation. In: [online] [visited on 2020-05-27]. Available from: `https://arxiv.org/abs/1611.09326`.

36. JUN, Ma. Loss Functions for Medical Image Segmentation: A Taxonomy. *Medium* [online] [visited on 2020-05-25]. Available from: `https://medium.com/@junma11/loss-functions-for-medical-image-segmentation-a-taxonomy-cefa5292eec0`.

37. HELLER, N. et al. *Data from C4KC-KiTS [Data set]. The Cancer Imaging Archive* [=https://doi.org/10.7937/TCIA.2019.IX49E8NX]. 2019. Available from DOI: `10.7937/TCIA.2019.IX49E8NX`.

38. Challenge Data. *Grand Challenge* [online] [visited on 2020-05-27]. Available from: `https://kits19.grand-challenge.org/data/`.

39. K, Clark et al. The Cancer Imaging Archive (TCIA): Maintaining and Operating a Public Information Repository. *Journal of Digital Imaging*. 2013, vol. 26, no. 6, pp. 1045–1057. Available from DOI: `10.1007/s10278-013-9622-7`.

40. AMIDI, Shervine; AMIDI, Afshine. A detailed example of how to use data generators with Keras. *Standford Education* [online] [visited on 2020-05-27]. Available from: `https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly`.

41. JOUET, Gregor. Avoid Bottlenecks in distributed Deep Learning pipelines with Horovod. *Adaltas* [online] [visited on 2020-05-27]. Available from: `https://www.adaltas.com/en/2019/11/15/avoid-deep-learning-bottlenecks/`.

42. HUANG, Lei. Dice-coefficient loss function vs cross-entropy. *StackExchange* [online] [visited on 2020-05-27]. Available from: `https://stats.stackexchange.com/questions/321460/dice-coefficient-loss-function-vs-cross-entropy`.

Appendix A

Acronyms

CT Computed Tomography

MRI Magnetic resonance imaging

NN Neural network

ML Machine learning

DL Deep learning

DCL Dice coefficient loss

JCL Jaccard coefficient loss

CNN Convolutional Neural Network

FCN Fully Convolutional Network

GPU Graphics Processing Unit

RAM Random Access Memory

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with USB contents description
- `src` ..... the directory of source codes
  - `model_weights` ..... the directory with pretrained weights
    - `basic_info.txt` ..... info about training parameters
    - `events.out.tf` ..... log from Tensorflow
    - `weights_2D/3D.h5` ..... model weights
  - `testing_model_weights.ipynb` ..... notebook for testing models
  - `python_source_codes` ..... directory with *.py files needed for preprosessing, data augmentation and train support
- `script_Horovod.txt` ..... how to install horovod on Metacentrum
- `segmentation.zip` ..... ground truth vs predicted segmentations
- `dataset.tar.gz` ..... smaller resolution dataset
- `text` ..... the thesis text directory
  - `fig` ..... the directory with figures
  - `ref.bib` ..... the bibliography resource
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis.tex` ..... the thesis text in LATEX format