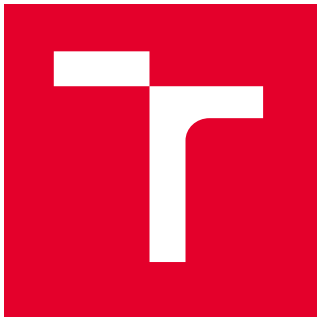


BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF MECHANICAL ENGINEERING

FAKULTA STROJNÍHO INŽENÝRSTVÍ

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

ÚSTAV AUTOMATIZACE A INFORMATIKY

SEGMENTATION OF CARDIAC MUSCLE IMAGES ACQUIRED USING CONFOCAL MICROSCOPY

SEGMENTACE SNÍMKŮ SRDEČNÍ SVALOVINY ZACHYCENÝCH POMOCÍ MIKROSKOPIE

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR

AUTOR PRÁCE

Bc. Filip Kadlec

SUPERVISOR

VEDOUCÍ PRÁCE

Ing. Pavel Škrabánek, Ph.D.

BRNO 2022

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF MECHANICAL ENGINEERING

Assignment Master's Thesis

Institut:	Institute of Automation and Computer Science
Student:	**Bc. Filip Kadlec**
Degree programm:	Applied Computer Science and Control
Branch:	no specialisation
Supervisor:	**Ing. Pavel Škrabánek, Ph.D.**
Academic year:	2021/22

As provided for by the Act No. 111/98 Coll. on higher education institutions and the BUT Study and Examination Regulations, the director of the Institute hereby assigns the following topic of Master's Thesis:

Segmentation of cardiac muscle images acquired using confocal microscopy

Brief Description:

Modern optical microscopy allows displaying areas of interest in cells without the need to separate the cells from tissue. The cells in the microscopy images must be identified and localized to allow an extraction of a relevant information from the images.

Master's Thesis goals:

The student will develop a survey of image segmentation methods. He/she will propose, implement, and verify an image segmentation system aimed at segmentation of heart muscle images captured using the confocal microscopy. The system will delimit the tissue–capturing areas and determine the boundaries of individual cardiomyocytes. The student will implement the system into a software which will allow training of the system on labelled images as well as automated processing of unlabelled images. The software interface will be created according to the requirements of researchers in the field of biology.

Recommended bibliography:

GONZALEZ, Rafael C. a Richard E. WOODS. Digital image processing. New York, NY: Pearson, [2018]. ISBN 978-0133356724.

PAWLEY, James, ed. Handbook of Biological Confocal Microscopy [online]. 3. US: Springer, 2006 [cit. 2019-09-03]. ISBN 978-0-387-45524-2. Dostupné z: https://www.springer.com/gp/book/9780387259215

LATEEF, Fahad a Yassine RUICHEK. Survey on semantic segmentation using deep learning techniques. Neurocomputing [online]. 2019, 338, 321-348 [cit. 2019-09-03]. DOI: 10.1016/j.neucom.2019.02.003. ISSN 09252312. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S092523121930181X.

Deadline for submission Master's Thesis is given by the Schedule of the Academic year 2021/22

In Brno,

L. S.

doc. Ing. Radomil Matoušek, Ph.D. Director of the Institute	doc. Ing. Jiří Hlinka, Ph.D. FME dean

ABSTRACT

Automating data acquisition and processing is common practice in both microscopy and computer vision fields. To classify and localize objects of interest (cardiomyocytes in this case) in microscopy images, segmentation can be performed. In this particular case, semantic segmentation by using deep neural networks was used as the core mean to perform mentioned task and software providing possibility of processing unlabeled data or training neural network architectures on labeled data was implemented. This work does a brief introduction to optical microscopy, inspects segmentation and deep learning in detail and finally describes the process from preparing data, implementing and training neural networks, to design of the final software. This software will ease the work of researchers by providing them with only relevant data, automate microscopy data acquisition, and with minor changes it can be applied to any similar segmentation task.

ABSTRAKT

Automatizace získávání a zpracování dat je dnes běžnou záležitostí jak v mikroskopii tak v počítačovém vidění. Ke klasifikaci a lokalizaci objektů zájmu (v tomto případě kardiomyocytů) lze užít segmentaci. V tomto konkrétním případě byla aplikována sémantická segmentace za užití hlubokých neuronových sítí jakožto hlavního prostředku k provedení zmíněné úlohy a byl vytvořen software umožňující jak zpracování neoznačených dat, tak trénování modelů neuronových sítí na označených datech. Tato práce krátce hovoří o optické mikroskopi, detailně popisuje segmentaci a hluboké učení a na závěr poskytuje popis procesu od přípravy dat, přes implementaci a trénování neuronových sítí, k vytvoření konečného softwaru. Tento software usnadní a zefektivní práci výzkumníků poskytnutím pouze relevantních dat pro výzkum, pomůže automatizovat sběr mikroskopických snímků a s menším upravením může být aplikován na další obdobné segmentační úlohy.

KEYWORDS

segmentation, deep learning, Keras, neural networks, microscopy images

KLÍČOVÁ SLOVA

segmentace, hluboké učení, Keras, neuronové sítě, mikroskopické obrázky

2022

BIBLIOGRAPHIC CITATION

KADLEC, Filip. *Segmentation of cardiac muscle images acquired using confocal microscopy*. Brno: Brno University of Technology, Faculty of Mechanical Engineering, Institute of Automation and Computer Science, 2022, 133 p. Master's Thesis. Supervised by Ing. Pavel Škrabánek, Ph.D.

ROZŠÍŘENÝ ABSTRAKT

Úvod

Kardiomyocyty jsou podlouhlé kontraktilní buňky o velikosti v rámci mikrometrů, které tvoří srdeční svalovou tkáň. Pro funkci srdce, pumpování a rozmístění krve po těle, jsou nezbytné. Pro zobrazení a studium buněk zmíněných proporcí se musí použít mikroskopie.

Dnes je poměrně běžné automatizovat sběr a zpracování dat jak v počítačovém vidění, tak v mikroskopii pro zefektivnění práce výzkumníků. Přesně o to se pokouší tato diplomová práce- implementovat automatizovaný proces, který identifikuje a lokalizuje kardiomyocyty zájmu za použití konvolučních neuronových sítí aplikujících sémantickou segmentaci. Cílem této práce je tedy implementace algoritmu, který bude schopen segmentovat jednotlivé kardiomyocyty a definovat oblasti zájmu a jejich ohraničení. Tento software umožní automaticky zpracovat neoznačená data, ale také trénování segmentačních modelů na již vytvořené datové sadě s označenými obrázky.

První část této práce pokryje rešerši a teorii, které stojí za koncepty, kterým je potřeba porozumět pro úspěšné dokončení praktické části. Optická mikroskope je popsána jako první a je to prostředek, kterým byla nasbírána data pro tuto práci (mikroskopické obrázky obsahující buňky srdečního svalu). Následně jsou popsána obě hlavní témata, a to segmentace a hluboké učení. Segmentace obrazu je hlavním úkolem praktické části této práce, která má být aplikována na nasbírané snímky a hluboké učení je nástroj, jenž byl k segmentaci použit. Nejprve jsou v obou částech popsány obecné koncepty a následně splynou dohromady v popis segmentace v rámci hlubokého učení s důrazem na prvky a architektury standardních, i nových architektur neuronových sítí vhodných pro segmentaci a malé datové sady.

V druhé, praktické, části této práce je popsán postup od vytvoření datové sady, trénování a vyhodnocení architektur neuronových sítí, po zvolení těch nejvhodnějších architektur. Nakonec je popsán a vyhodnocen navržený algoritmus pro identifikaci a lokalizaci kardiomyocytů. Do tohoto algoritmu vstupují obrázky o rozměrech 4096×4096px a výstupem jsou segmentační mapy a souřadnice oblastí zájmu. Tento úkol je výzvou s ohledem na rozměry obrázků, relativně malé množství dat a limitace hardwaru. Navržená metoda může být použita výzkumníky pro automatický sběr oblastí zájmu z mikroskopických obrázků, které obsahují pouze zachované, "zdravé" kardiomyocyty, nebo pro mikroskop, který po získání souřadnic oblastí zájmu může automaticky změnit souřadnice sběru dat na vzorku a následně zaostřit a nasbírat pouze zajímavé části kardiomyocytů pro jejich pozorování a výzkum.

Popis řešení

Implementaci zmíněného softwaru předchází vytvoření datové sady a natrénování a výběr architektur neuronové sítě. Bylo obdrženo 33 obrázků o rozměrech 4096×4096px ze Slovenské akademie věd od paní Ing. Alexandry Zahradníkové, DrSc. Po rešerši byl zvolen nástroj pro označení dat a tyto obrázky byly označeny, tedy byly na nich vyznačeny oblasti tří tříd kardiomyocytů: zachovalé kardiomyocyty se strukturou, zničené, zhroucené kardiomyocyty a třída, jenž neodpovídá ani jedné ze zmíněných definic, tj. kardiomyocyty polorozpadlé, či nezaostřené, rozmazané. Výstupem z tohoto procesu byly mapy mající pixely o hodnotách 1-3 dle zmíněných tříd a 0 tam, kde je zbylé pozadí. Originální obrázky následně byly rozděleny do regionů o rozměrech 512×512px, jelikož originální rozměr by vzhledem k RAM a GPU omezením neuronovou sítí při trénování neprošel. Nasekáním obrázků vzniklo určité množství dat, což se následně rozšířilo o tato data augmentovaná za běhu trénování neuronové sítě.

Dalším krokem byla implementace, natrénování a vyhodnocení určitých architektur hlubokých neuronových sítí vhodných pro segmentační úlohy. Standardně se jedná o konvoluční neuronové sítě dělící se na enkodér a dekodér, jejichž vstupem je obrázek a výstupem segmentační mapa. Prvotně bylo v plánu otestovat standardní modely jako U-Net a postupně vyzkoušet nové, pokročilé architektury jako R2U-Net, DeepLabv3+, či Attention U-Net. Vzhledem k poklesu úspěšnosti architektur mající komplexní vlastnosti a velký počet parametrů se postup změnil a přiklonil se k otestování dalších jednodušších, méně komplexních modelů. Z natrénovaných modelů byly na této úloze nejúspěšnější architektury *Segnet* a *Xception U-Net s residuálními bloky*.

Po volbě natrénovaných architektur byl implementován finální algoritmus, jenž bere obrázky o originální velikosti 4096×4096px jako vstup. Tento software lze rozčlenit na několik částí. Hlavičkový soubor, kde si uživatel může nadefinovat dle svých preferencí možnosti a variace výstupů. Po importování a definování vlastních funkcí jsou data načtena a nasekána na zmíněné rozměry pro zpracování neuronovou sítí. Těchto 64 nasekaných obrázků je segmentováno natrénovaným modelem a následně jsou tyto segmentační mapy "sešity" do obrázku o originální velikosti- tedy v tento moment je obdržena segmentační mapa originálního obrázku. Tato mapa je binarizována. Tam, kde se nachází segmentovaný kardiomyocyt zájmu, se nachází 1, zbytek 0. Po provedení binárního uzavření (binary closing) se detekují objekty nad určitým prahem velikosti těchto objektů a ohraničující obdelníky těchto objektů jsou nalezeny. Výstupem z tohoto programu jsou vystřižené oblasti zájmu jako obrázky ve vlastní složce, jejich souřadnice, segmentační mapa originálního obrázku, popřípadě mohou být výstupem stejné objekty pro polorozpadlé kardiomyocyty.

Výsledky a závěr

Segmentace srdeční svaloviny pro přesnou a automatizovanou lokalizaci kardiomyocytů zájmu byla implementována a popsána. Vstupem do tohoto softwaru jsou mikroskopické obrázky obsahující srdeční svalovinu a výstupem, mimo jiné, segmentační mapy vstupních obrázků či vyřezané oblasti zájmu. Implementace tohoto softwaru byla provedena v prostředi Google Colab (Jupyter notebooky) a umožňuje jak trénování na označených datech, tak zpracování neoznačených obrázků.

Vyhodnocení úspěšnosti navrženého softwaru není jediným měřítkem kvality softwaru. Nejprve může být vyhodnocena samotná segmentace. Překvapivě největší úspěšnost měly natrénované modely SegNet a Xception Unet s residuálními bloky. Tento výsledek neguje předpoklad, že aplikování nových technik, *attention gates, dilated convolutions,* nebo kombinace *reccurent, residual blocks*, či nových modeů jako DeepLabv3+ nebo R2U-Net, zvětší přesnost segmentace. Zvětšení úspěšnosti segmentace by mohlo být dosaženo pomocí implementování vhodnější účelové funkce (Dice, Jaccard, Focal), která by potlačila zřejmý charakter těchto obrázků- nevyváženost tříd. Zlepšení by mohlo být dosaženo také přesnějším označením dat, jelikož ve stávající verzi datové sady jsou objekty označeny co nejpřesněji po jejich obrysech včetně rozmazaných, nezaostřených částí kardiomyocytů.

Finální program byl otestován na deseti obrázcích, které obsahovali 84 objektů zájmu k segmentaci, při čemž žádný z nich nebyl programem přehlédnut. Na druhou stranu 14 objektů patřící do třídy polozhroucených kardiomyocytů bylo identifikováno jako kardiomyocyty zájmu. Většina z těchto objektů je ovšem na hranici mezi těmito třídami a v případě, že by tyto buňky klasifikoval člověk, tak by k rozhodnutí nejspíše došlo na základě vlastní preference. Při pohlédnutí do segmentačních map na tyto objekty je zjevné, že i síť tyto buňky označuje dvoubarevně (tedy rozhoduje se mezi zmíněnými třídami, mezi kterými je označení na hraně). Tento program byl také otestován na zmíněných deseti obrázcích, kde se vzali predikované a očekávané oblasti zájmu a byl z nich vypočítán Mean IoU roven 0.6443. V momentě výpočtu Mean IoU s vynecháním misklasifikovaných kardiomyocytů byl výsledek roven 0.8209. Tyto výsledky byly následně i ilustrovány, co si pod tím čtenář může představit, je zobrazeno v sekci 5.4. Je ovšem důležité zmínit, že toto vyhodnocení je validní pouze v případě, že prvotní segmentace je spolehlivá. Toto vyhodnocení proběhlo na obrázcích z trénovací sady a tedy segmentace na těchto obrázcích je zkreslena a pravděpodobně jsou výsledky lepší, než by byly na nových datech (ta ovšem dostupná nejsou a vyhrazení dostatečného množství obrázků na toto vyhodnocení by znamenalo ztrátu významného množství dat z už tak malé datové sady).

Tento software by šlo vylepšit i z pohledu uživatelského rozhraní. Volba Jupyter notebooků byla kompromisem mezi spustitelnými Python soubory a imple-

mentováním grafického rozhraní, jelikož se zdá uživatelsky přívětivější, než spustitelné soubory a program je zároveň nezávislý na výpočetním prostředí. Dalším krokem by bylo posunutí softwaru směrem ke grafickému rozhraní po poskytnutí časových prostředků.

Navržený program může ulehčit práci výzkumníkům či automatizovat sběr relevantních dat pomocí mikroskopu. Software nemusí být limitován pouze na řešenou úlohu, mohl by být použit na jakoukoliv segmentační úlohu po upravení architektur neuronových sítí.

AUTHOR'S DECLARATION

I hereby declare that this master's thesis was prepared as an original author's work under the supervision of Ing. Pavel Škrabánek, Ph.D. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

In Brno, 20. 5. 2022 .....

Filip Kadlec

ACKNOWLEDGEMENT

I wish to express my thanks to my supervisor Ing. Pavel Škrabánek, Ph.D. for his guidance and goodwill, that helped me to create this work more professional. My gratitude also belongs to Ing. Alexandra Zahradníková, DrSc., without whose data and cooperation it would not be possible to create this thesis.

CONTENTS

1 INTRODUCTION

Cardiomyocytes are oblong contractile cells with the size within micrometers that form cardiac muscle tissue. They are essential for the heart function, that is to pump and spread blood over the body. To be able to observe and study cells of such proportions, microscopy must be used to display them.

It is common practice nowadays to automate data acquisition and processing in computer vision and microscopy domain to make the work of researchers more efficient and convenient. That is exactly the type of problem that is being dealt with in this thesis: to implement an automated process, that will identify and localize cardiomyocytes of interest using convolutional neural network performing semantic segmentation. The ultimate goal is to implement a software for the segmentation of cardiac muscle cells and define areas of interest and their boundaries. This software should allow to automatically process unlabeled data and to train segmentation models on created dataset with labeled images.

The first part of this work will cover the research and theory behind concepts that were used to succesfully complete the practical part of this thesis. The optical microscopy, which is the mean of acquiring the data displaying cardiac muscle cells, is briefly introduced at first. Then both major fields of study are dealt with: image segmentation and deep learning. Image segmentation is the main task to be performed on the data and deep learning is the instrument used for the task. General concepts of both areas are presented at first, including classical approaches as well as current advanced methodologies. Afterwads, they merge into a description of segmentation using deep learning with emphasis on both standard and state-of-the-art segmentation architectures.

In the second, practical, part of this work, the procedure from dataset creation, neural network training and evaulation, to the neural network architecture selection is described. The proposed segmentation method is explained and evaluated in the end, with 4096×4096px images as the inputs and segmentation maps and coordinates of areas of interest as the outputs of the algorithm. The task is challenging with respect to the size of images, low amount of data available and hardware limitations. The proposed software is aimed at researchers for them to use to automatically obtain areas of interest containing healthy cardiomyocytes in images. It can be applied for the microscope to change focus coordinates and later acquisition of interesting parts of images for cardiac muscle cell observation as well, or to provide the researcher only with areas of interest from the microscopy images.

2 OPTICAL MICROSCOPY

Microscopy, in general, is a collection of optics applications used for displaying structures that are not visible to the human eye. The ability to differentiate two points, details, of the human eye is around 0,2 - 0,3 milimeters. To be able to differentiate smaller structures, the image of a sample must be magnified. That is where microscope is used to accomplish such task.[40]

Methods of microscopy may be divided by numerous attributes. Depending on the type of energy transfer, the microscopy may be divided into electron microscopy and optical microscopy. This brief introduction into microscopy consists only of optical microscopy, since the method used to collect data for this thesis was one of the optical ones and there is no other relation between this thesis and microscopy apart from the method used to collect the desired data.

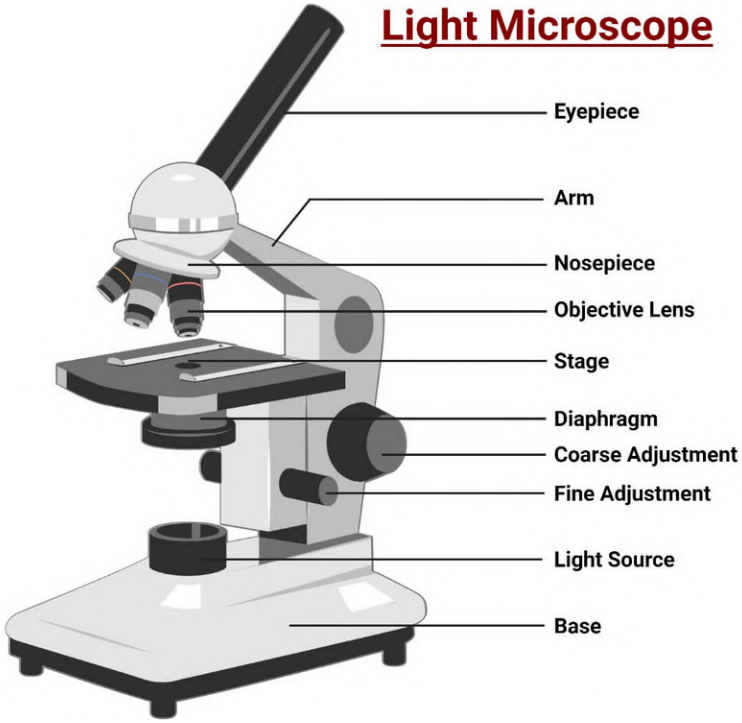


Fig. 1: Example of optical (light) microscope with annotated parts [31]

Apart from optical microscope lens and parameters, the resolution of these microscopes is limited also by the properties of light and can be pushed down to 0,25μm. This is caused by the wavelength of light (photons). Objects smaller than half the wavelength of light will not be seen under an optical microscope. When the abilities of optical microscopes regarding resolution and magnification (the image size divided by the object size) are not enough, electron or atomic force microscopy is used instead.[40, 31]

The construction of optical microscopes can be divided into mechanical and optical parts (in some literature the illuminator is viewed as a standalone *light part* of the microscope). The optical part consists of three optical systems (light part included):

- Illuminator, the source of light of the microscope
  Typically located at the base of the microscope with light beam coming out if it, facing against the observation direction providing back-illumination.
- Objective lenses, the system of (mainly) optical lenses
  This part of microscope usually consists of 3-5 lenses and displays the inverted, magnified and real view of the sample. The final view is projected between the ocular and its focus.
- Ocular, the part of microscope that human eye looks into
  It is a system of lenses as well and fulfills the reversal of the view into virtual, upright and magnified view and fullfills functions as a magnifier as well. [32]

2.1 Optical microscope imaging concept

2.1.1 Introduction into microscopy

Basically, the goal of microscopes is to enlarge the angle of the field of view (FOV) of created image of the sample. The angle of the field of view is an angle created by the edge rays of the object displayed on retina of the human eye (or displayed on pixels created by camera). If the mentioned angle is enlarged, the ability to differentiate between two points is enlarged as well. The two points are differentiable, if the rays of light of these points are apart from each other at least one cell of the retina, where they land.[40]

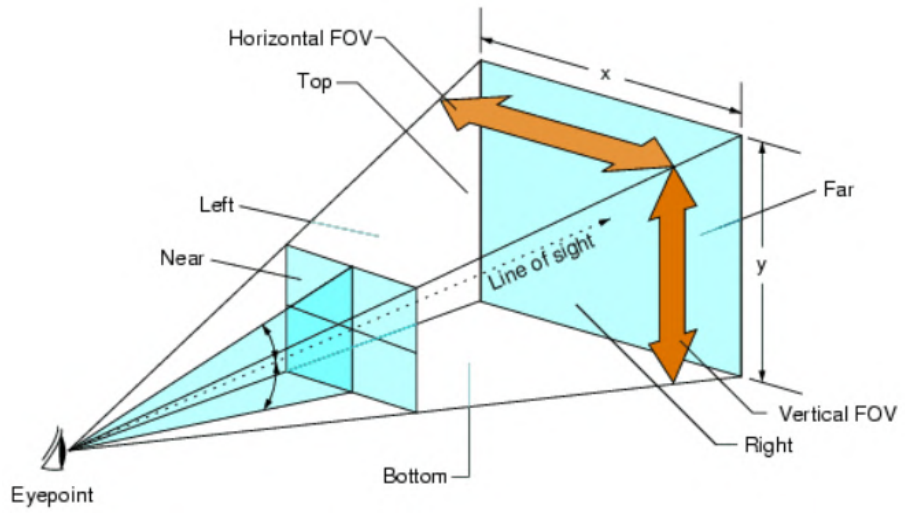


Fig. 2: Scheme of field of view and angle of view. Taken and modified from [14]

In the Fig. 2 the horizontal and vertical components of the field of view is displayed. The angle of view is defined as an angle from the axis of the machine (camera), or human eye, to the outmost observable point in given direction.

The basic principle of (compound) microscope imaging may be described as follows. The beam of light is generated and focused on the object/sample. The microscope uses the objective lens close to the sample to collect the light transmitted by the sample. That focuses and creates the magnified image of the real object in the body tube of the microscope. This image is called *the primary image*. The primary image is then magnified by the ocular lenses. The use of combinations of ocular and objective lens results in various magnifications. If the change of magnification is needed, the microscope usually has the rotating mechanical part called *nose piece*, that, when rotated, interchanges the lenses. This process from the optics point of view is displayed in the Fig. 3.[4]

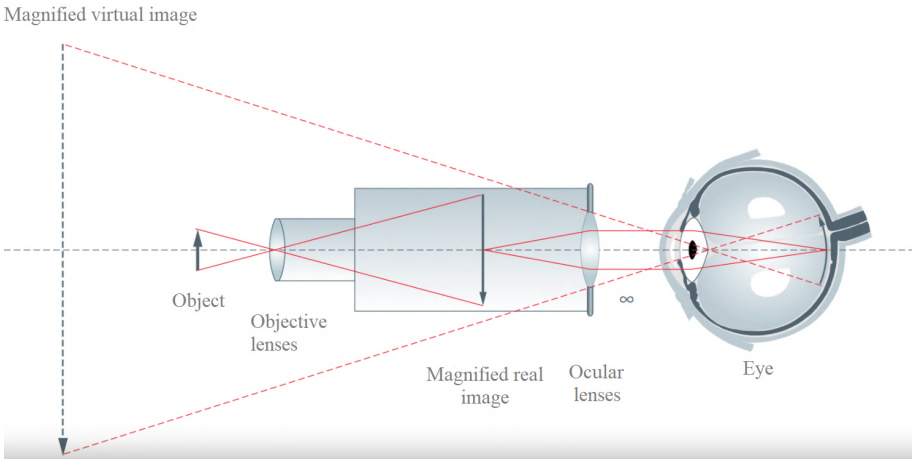


Fig. 3: Scheme of the principle of compound light microscopes [41]

The total magnification of a light microscope may be usually computed from the magnification of the objective lenses and ocular by simple multiplication. So if the power of objective lenses is 40x and if the power of the ocular is 10x, the resulting total magnification is 400x.

2.1.2 Light microscopy techniques

The microscopes are not only designed and used in the basic form described above. There are several types of optical microscopes. Most of them are advanced or modified versions of the basic one. They can be divided into:

- Bright field Microscopes
  Most basic, already described microscopes that produce dark image against the bright background with high contrast. Typically used to view plant or animal cells

- Phase Contrast Microscopes
  Optical microscopes that use phase shifts during light penetration into the sample. When photons pass through the opaque sample, the phase shifts brighten the sample. This results in bright image in the background.
- Dark-Field Microscopes
  Bright field microscopes that are similar to phase contrast microscopes. To create such microscope, a darkfield stop must be placed underneath the microscope with a condenser lens. That produces a hollow cone beam of light that enters the objective only. So when beam is passed through the sample, unreflected light rays do not pass through the objective lens, but the reflected light does pass through the objective lens, which forms an image. The image then has black surroundings of the sample and the sample itself appears illuminated. In other words, direct light is blocked in the darkfield stop and only light that is refracted by structures of the sample can enter the objective. This can be used to display live unsustained cells.
- Differential interference contrast (DIC) Microscopes
  DIC microscopy creates sharp images with 3D effect. This is done by two parallel light rays very close to each other. Those rays interfere after passing through an unstained sample. That results in the background being dark while the interference pattern being very crisp at boundaries.
- Fluorescence Microscopes
  Fluorescencce microscopy uses high intensity light. The sample is marked with a fluorescent substance and illuminated with higher energy source. The illumination light is absorbed by the fluorescent substance attached on the sample and causes them to emit a lower energy wavelength light. This fluorescent light can be distinguished from the surrounding radiation with filters set up for this specific wavelength. That allows to display only the light of the wavelength which is fluorescing. For more detail see section 2.2
- Confocal microscopes
  The confocal microscope is a certain type of fluorescent microscope. It usually displays high resolution 3D images. Unlike in fluorescent microscopes, they do not expose the samples to high intensity UV light so the images of samples are not bleachy or blurry. The laser light is used instead. Images are taken with digital camera with a pinhole. The pinhole is allowing the light of only one focal plane to be focused on the digital camera. A laser beam focused and scanned over the sample produces 3D or 2D images.
- Polarization microscopes
  Some biological samples contain highly organized structures and they show certain properties and diffraction patterns under polarized light. This is used

in polarization microscopes that use polarizing filters through which light passes in the microscope (placed under the sample). Like in the dark field microscopes, only light that has been reflected by the sample enters objective lens.[31, 26, 44]

2.2 Fluorescence Light Microscopy

Since the fluorescence microscopy was used to collect data for this thesis, the technique will be described in more detail. Fluorescence is a spontaneous emition of light by a substance that is not resulting from heat. It is caused by the absorption of photons (or electrons) by the iluminated system and the system becoming excited. The energy of the system is increased. The system getting into higher energetical state becomes unstable and starts emitting photons (or electrons) trying to get back into the original equilibrium state. Doing this, superfluous energy is emitted in the form of light.

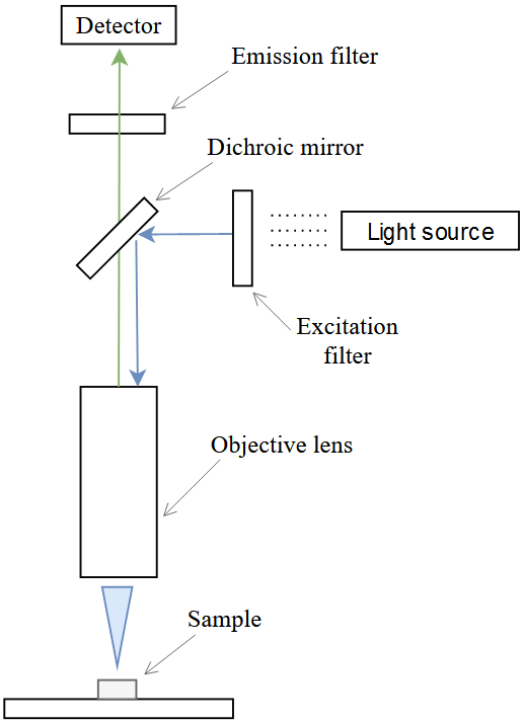


Fig. 4: Scheme of the fluorescence microscope [27]

This effect is used in fluorescence microscopy directly on observed samples. The sample is exposed (usually) to UV light. The sample, as said, absorbs the light and emits visible light having higher wavelenghts, which is observable by optical microscope. [45] The scheme of fluorescence microscope is shown in the Fig. 4 and examples of captured images of the described method in the Fig. 5.

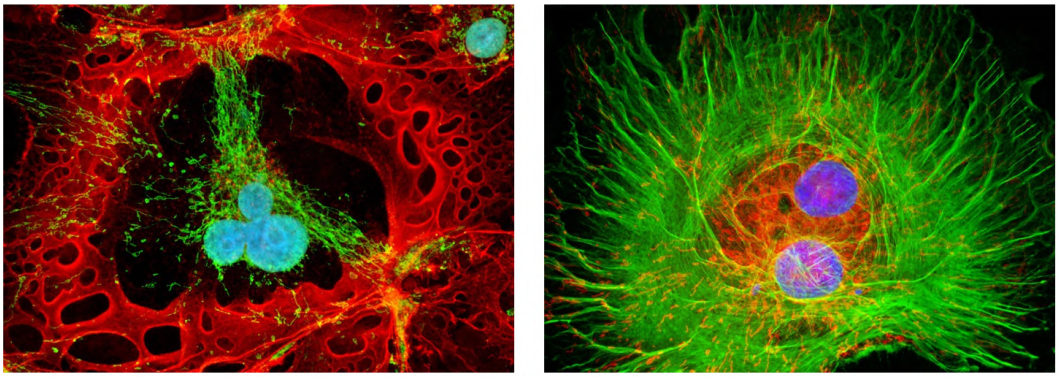


Fig. 5: Examples of fluorescence microscopy images [30]

3 IMAGE SEGMENTATION

Image segmentation is an image processing method in which a digital image is partitioned into objects (regions, segments). That helps in reducing the image complexity in order to make further processing, like feature extraction or analysis, more straightforward. From the practical point of view, image segmentation is nothing but assigning labels to pixels of an image. All pixels of an image belonging to the same class are labeled with the same label. [19, 48]

Basic segmentation algorithms can be generally divided into two groups: algorithms that are based on **discontinuity** or **similarity** of image intensity values. The first mentioned group of algorithms relies on pixel intensity values of the image, such as lines, points or edges. The second group is based on partitioning an image into regions, that can be characterized by similar predefined properties. Thresholding, region growing, or clustering are few of many examples from the second mentioned group. [19]

Partition into concrete types of segmentation methods that will be mentioned in this chapter is displayed on the following picture:

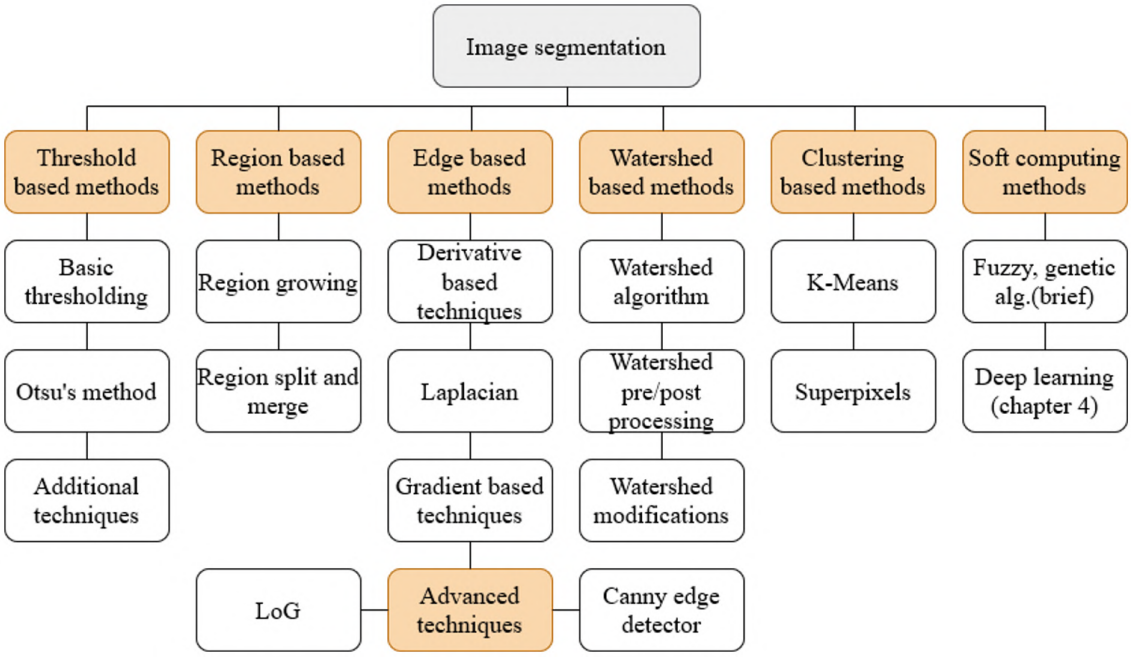


Fig. 6: Types of mentioned segmentation techniques in this chapter

3.1 Threshold methods

Thresholding itself is probably the simplest technique of image segmentation, which may be used for separating objects from the background. Typically greyscale images are thresholded (or RGB images converted to greyscale and then processed). The process of thresholding an image is basically comparing a value of each pixel to a given threshold. That divides all pixels of an image into two groups: pixels having intensity value lower and greater than mentioned threshold. That usually results in a binary image. The pixel value of the binary image at the spatial coordinates (x,y) is given as

$$g(x,y) = \begin{cases} 0 & \text{if } f(x,y) \leq T \\ 1 & \text{if } f(x,y) > T \end{cases}, \tag{1}$$

where $f(x,y)$ is intensity value of a pixel in the original image at the same spatial coordinates. Thresholding naturally is not used in this basic form, but with some adjustments. An efficient thresholding segmentation cannot happen without an algorithm, that chooses the threshold value automatically based on each input image. Value of T may also change with time, if local/neighbooring thresholding is being used. It is noteworthy what *multiple thresholding* is as well. That is when more than one threshold is selected and therefore the set of pixels is divided into more categories than two (number of thresholds + 1). An advanced technique among threshold methods is an *adaptive thresholding*, which does not have only one global value of the threshold, but computes thershold for small regions in images. Thanks to that the method is more robust for example against varying lightning conditions in an image.[19]

Basic global thresholding algorithm

The typical way of choosing a threshold T is with the help of peaks and valleys in the image histogram, where algorithms usually converge to the solution of choosing T in the valleys. The basic global thresholding algorithm could be described as follows[19]:

- Initial estimate for the global threshold, T
- Segment the image using T. This will produce two groups of pixels $G1$ and $G2$ (pixels with value greater or lower than T)
- Compute the mean intensity values $m1$ and $m2$ for the pixels in $G1$ and $G2$
- Compute the new threshold

$$T = \frac{m_1 + m_2}{2} \tag{2}$$

- Repeat Steps 2 through 4 until the difference between found thresholds from two consecutive iterations ΔT is smaller than a predefined value or a maximum number of iterations is reached

There is an example on the Fig. 7, where coins are displayed on the grayscale image.

Fig. 7: Image of coins, pre-packaged demo image in Matlab

A built-in Matlab function was used to find the optimal threshold of this image. This function uses a shape of a given image histogram to determine it. More concretely, it does not use basic global thresholding, but *Otsu's method* was used as the mean of computing desired threshold (more on that in the next paragraph).[46] The histogram of the image is shown on Fig. 7:

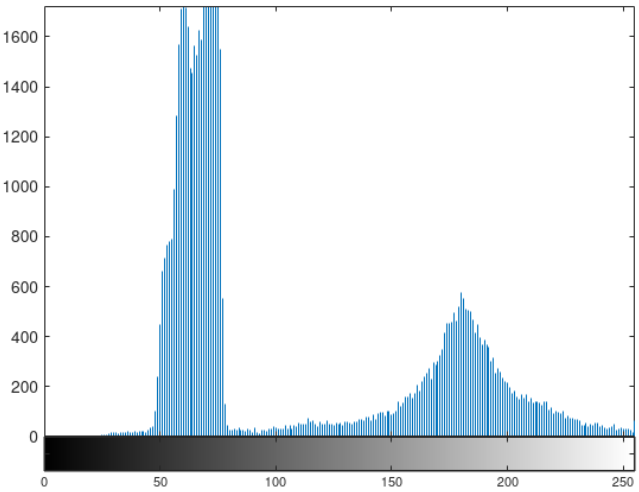


Fig. 8: Histogram of previous coins image

As expected, the found value of threshold is right between the two peaks in the valley. More concretely, the found value is (when recomputed from returned value of the function to the 0-255 scale) 125.99. If the image is binarized with the found threshold, it is clear on the Fig. 9, that objects are succesfully distinguished from

the background and with some following operations, precise segmentation could be accomplished.

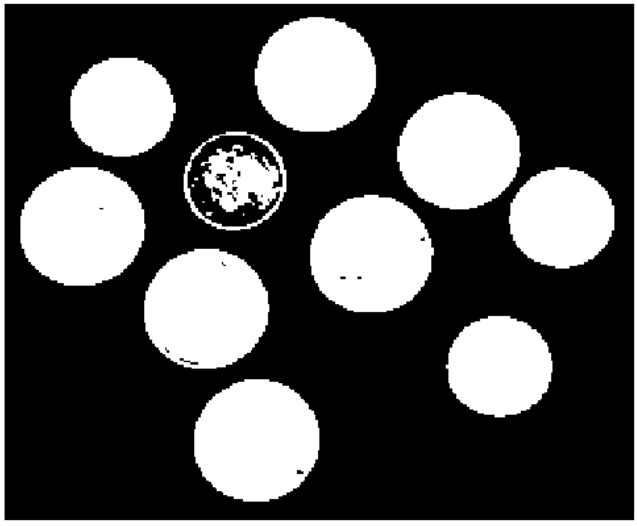


Fig. 9: Segmented coins image

Otsu's method

In Otsu's method, that was used in the previous example and that is an instance of methods that use a shape of a histogram to determine the threshold, the image is basically seen as two groups of points with different intervals of pixel intensity (foreground and background). It is often the case for these two groups to overlay. The goal of the method is then minimizing the misidentification of those two groups. In other words between-class variance is being maximized. An important property of Otsu's method is the fact that it is completely based on computations on the image histogram, which is easily obtainable and is a one dimensional array of numbers. That, compared to other approaches, such as Bayes decision function, is a convenient fact regarding computational load.[19]

Without going into too much detail, Otsu's algorithm may be summarized as follows[19]:

- Compute the normalized histogram of the input image
- Compute the cumulative sums, cumulative means and the global mean from the normalized histogram components
- Compute the **between-class variance**
- Obtain the **Otsu threshold**, for which between-class variance is maximum
- Compute the global variance and obtain the separability measure

Since the original paper was released (year 1979), other improvements and additions were added to the method. Some concerning computational efficiency, some expanding the method, for example to solve multi level thresholding.

Additional techniques

Nevertheless, images are not always as easy to process by threshold segmentation as it is shown on the coins image. It was already mentioned that using threshold segmentation technique may be considered simple and sometimes inefficient. Especially when an image contains a considerable amount of noise or histogram peaks are not properly separated. Threshold methods are sensitive to monotony created by illumination or reflactance as well. [18]

Image histogram is not the only thing that may be used for determining a threshold. A clustering algorithm may be used to cluster pixels into two groups (background, foreground) and then being able to set a threshold between them. An entropy may be used as well, since entropy values of the foreground and background differ and therefore it is possible to distinguish between them and afterwards optimize the selection of threshold by looking at crossentropy between the original image and resulting binarized image. Another interesting approach might be methods that track probability distribution and correlation between pixels in the image. [19, 18]

Additional techniques may be applied to improve the segmentation result. This topic could cover another whole chapter, so only few of the possible improvements will be mentioned. Since noise can make this task difficult, smoothing an image may be very helpful. Since it is also convenient for the histogram peaks to be narrow, tall and far from each other, it is useful to combine thresholding with finding object edges. The edges may be found by computing the magnitude of the gradient, or the value of the Laplacian (more on edge based methods in section 3.3). Another helper technique might be already mentioned variable thresholding (based either on local image properties or on moving averages. Second mentioned is useful especially in document processing).[19]

3.2 Region based methods

A region may be categorized as a set of neighboring, connected pixels possessing the same properties. The properties could be, for example, color or pixel intensity. Region based methods try to find and segment these regions. In the case of noisy images, region based methods are usually prefered against edge based methods. On the other hand, they usually are not a good choice on imgaes with significant textures. Region based methods may be classified into two types of methods: *region growing* and *region split and merge.*[19, 48]

Region growing

This technique segments the regions using a chosen pixel (later group of pixels), a seed (later region), and an image property. It groups pixels or subregions into larger and larger regions. From the seed, adjacent pixels are checked and if they follow

predefined rules and criteria to grow, they are added to the region of the seed pixel. This process continues until there are no neighboring pixels around given region that could be characterized by similarity to the region (for example if the difference between the gray levels is greater than a threshold).

The choice of the initial seeds strongly depends on particular problems and aplications. It may be chosen at random, it may be automatically chosen with the use of helper techniques like mathematical morphology (opening, closing), using clustering algorithms before placing the seed. The seed placement may also be semi-automated or even manual in some cases.

A basic region growing algorithm consist of a seed placement and then checking all the adjacent pixels of the seed/region if they satisfy a condition for the given pixel to be added to the seed's region. The condition for a greyscale picture checking values of pixels intensities $f(x, y)$ may be defined as [48]

$$f(x,y) = \begin{cases} z_i, & \text{if } |f(x+r, y+s) - v_{z_i}| < T \\ \not\Delta, & \text{otherwise} \end{cases}, \tag{3}$$

where z_i is an affiliation to the region Z_i from the seed z_i, r and s may have values -1,0,1, x and y are coordinates of current outer pixel being inspected, v_{z_i} is a value of seed z_i, T is predefined threshold and $\not\Delta$ signifies no change.

In other words, this equation is applied iteratively on all outer pixels of a region until there is no change after the application. There is as many regions $z_1, z_2, ...$ as there is initial seeds, if merging the regions or additional techniques are not applied

An example in a form of matrix, that may represent pixel intensity values, follows.

$$\begin{pmatrix} 0 & 0 & 5 & 6 & 7 \\ 1 & 1 & 5 & 8 & 6 \\ 0 & 1 & 6 & 7 & 7 \\ 2 & 0 & 7 & 6 & 6 \\ 0 & 1 & 5 & 6 & 5 \end{pmatrix}$$

In this example, the seeds were chosen at the coordinates [1,2] and [3,3], where first coordinate represent horizontal axis from left to right beginning at 1 and second coordinate represents vertical axis from top to bottom starting with 1 as well. If the previous equation (3) is applied iteratively on all outer pixels of defined regions (seeds in the first iteration), the results are following with respect to the choice of T.

For T=3, the resulting regions would be:

$$\begin{pmatrix} a & a & b & b & b \\ a & a & b & b & b \\ a & a & b & b & b \\ a & a & b & b & b \\ a & a & b & b & b \end{pmatrix}$$

For T=5, the resulting regions would be:

$$\begin{pmatrix} a & a & a & a & a \\ a & a & a & a & a \\ a & a & a & a & a \\ a & a & a & a & a \\ a & a & a & a & a \end{pmatrix}$$

Region split and merge

Split and merge method relies on dividing the image into regions (usually four, quadrants). If any of the regions cannot be characterized by a certain unanimity (could be the same level/interval of pixel intensity value), then the region has to be divided into additional (same number as before, mostly four) regions. The algorithm continues until all the created regions have their own individual property, with which they differ from the other, neighboring ones. After the splitting part, the split regions can be merged using a similar criteria that was used for regions division, because when only the splitting is used, usually there is a number of regions in an image that are adjacent to each other and have similar/same properties (i.e. oversegmentation happens). Then there is nothing easier to do than create an union of these regions into one[19].An ilustration of the splitting process is shown in the Fig. 10.

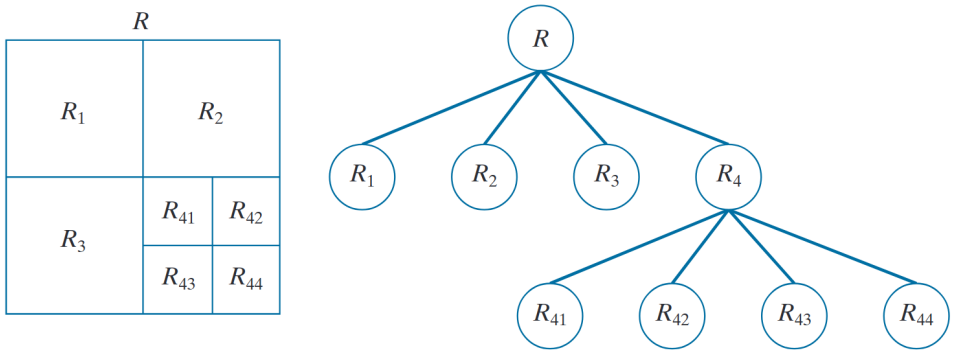


Fig. 10: Spliting an image and regions into subregions [19]

On the ilustration of the splitting process in the Fig. 10 the R represents the whole image that is then divided into quadrants $R_1 - R_4$. The quadrants R_1, R_2 and

R_3 satisfy the homogenity condition. The quadrant R_4, however, does not. That is why it is divided into another four subregions (quadrants). Those quadrants now seem to satisfy the homogenity condition as well. At this point there is no need for further region division, since all the regions satisfy a certain condition of unanimity and theoretically the merging process may begin to check, whether some of the adjacent regions are not supposed to be combined into one.

3.3 Edge based methods

Not only in segmentation, but in image processing in general, edge detection is widely studied set of methods. The edges are closely related to the segmentation, since at region boundaries, as well as at edges, there are often rapid changes of pixel intensity values. With edge detection there are usually other methods combined. The standard way of detecting edges is using filters and convolution.

Techniques based on differentiation

Derivative is one of the very basic concepts in edge detection. The derivative is simply sensitivity to any change of the function value with respect to a change in its argument (e.g. pixel intensity values). To be able to use such a function in a discrete manner, as computers work and as images are, the *difference* is created by using a Taylor approximation.[19]

Taylor approximation of the first order derivative at arbitrary point x of a one dimensional function is

$$\frac{\delta f}{\delta x} = f(x+1) - f(x), \tag{4}$$

and is called difference. For completeness, it is important to mention that the equation (4) is an expression of a forward difference and backward and central differences are derivable from the taylor approxiamtion of derivative as well. It is clear that difference (also in an image) is really simply the difference between two subsequent values and it can be said that it is a measure of change of the function. Before describing the effect of the difference on an image, second derivative will be defined first. Or rephrased, second order difference (Taylor approximation of the second order derivative at arbitrary point of a one dimensional function)

$$\frac{\delta^2 f}{\delta^2 x} = f(x+1) + f(x+1) - 2f(x). \tag{5}$$

Key note about this function is that it takes into account values both before and after the current one.[19]

With differences defined, the effect of applying such functions on a series of numbers (could be pixels) will be shown.

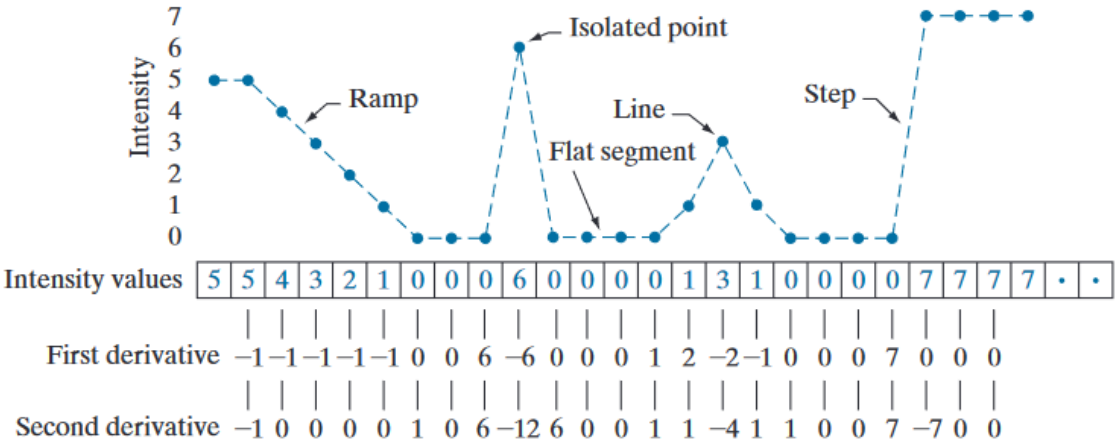


Fig. 11: First and second order difference applied on a series of numbers [19]

From the Fig. 11 it is clear that both differences are able to detect extreme points such as points and edges in the image. As can be seen, the second derivative is even more sensitive to the outermost points and fine details in the image. The second derivative produces thinner lines as well when compared to the first order derivative due to the zero crossing point. Exactly something, that could be useful in detecting edges (boundaries of regions to be segmented).

Laplacian

As was demonstrated in previous paragraphs, derivation (difference) is useful in finding places on a line, that have a specific property, that is a significant sudden change. Images, however, are at least two dimensional. That leads to the *Laplacian*. Laplacian is, explicitly described, the sum of all the unmixed second partial derivatives. In the particular case of two dimensions, the equation of Laplacian is

$$\nabla^2 f = \frac{\delta^2 f}{\delta x^2} + \frac{\delta^2 f}{\delta y^2}. \tag{6}$$

The follow up expression of Laplacian in an eight neighborhood is

$$\frac{\delta^2 f}{\delta x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y) \tag{7}$$

$$\frac{\delta^2 f}{\delta y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y), \tag{8}$$

that can be rewritten into

$$\nabla^2 f = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y). \tag{9}$$

Instead of using this as an operation on each pixel, it is convenient to imlpement a mask/filter with which it is possible to perform convolution afterwards. The original Laplacian filter (9) and its inverted variant is shown in the Fig. 12.[19]

0	1	0
1	-4	1
0	1	0

0	-1	0
-1	4	-1
0	-1	0

Fig. 12: Laplacian filter

Extension of the previous equation and filters that includes the diagonal neighbors may be found in the Fig. 13.

1	1	1
1	-8	1
1	1	1

-1	-1	-1
-1	8	-1
-1	-1	-1

Fig. 13: Laplacian filter with diagonal neighbors extension

What effect does a Laplacian filter actually has on an image will be inspected. In the Fig. 14 there is shown the original image (pre-packaged demo image in Matlab, *moon.tif*) on the left, applied original Laplacian filter in the center and applied Laplacian filter with diagonal neighbors on the right.

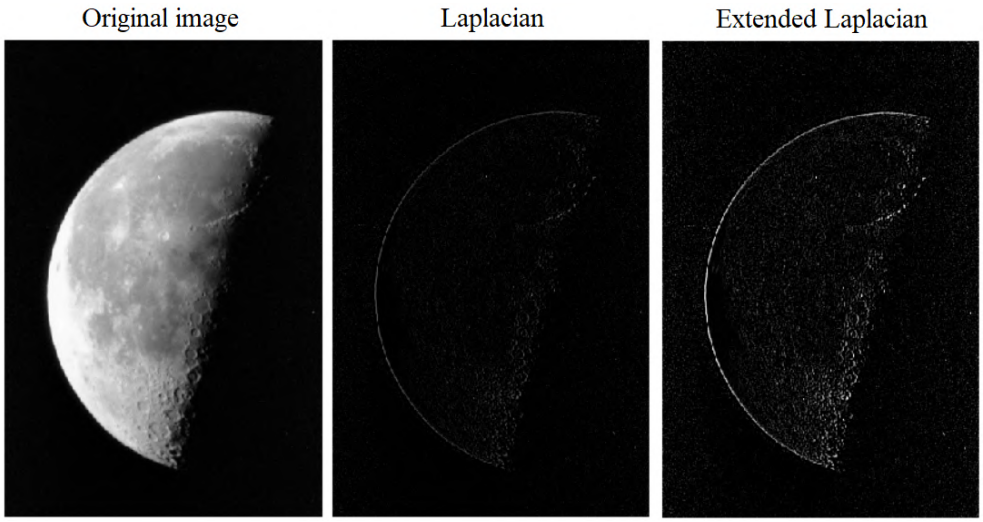


Fig. 14: Effect of Laplacian

It is clear that in this case, with some additional technique (e.g. math. morphology), it would be possible to acquire a complete boundary of the object. That would make segmenting it significantly easier.
Laplacian may be used to enhance/sharpen the edges of an original image as well, easily just by adding the Laplacian image after filtering to the original one.

Gradient based techniques

Not only second order derivative methods are useful for finding edges in an image. As was shown in the beginning of this chapter, first derivatives are able to perform edge detection as well, and when it is desirable to use such a tool on an image (in horizontal and vertical manner), it is possible to use image *gradient*.

Image gradient expresses not only by how much does the pixel intensity value change on a certain place in an image, but its direction as well. Image gradient is a vector at a certain point of an image (two dimensional, *x,y)*, whose components are partial derivatives of function *f*. This partial derivation is given as

$$\nabla f(x,y) = \begin{bmatrix} \frac{\delta f(x,y)}{\delta x} \\ \frac{\delta f(x,y)}{\delta y} \end{bmatrix}. \tag{10}$$

This vector has a useful property: it points in a direction of maximum rate of change of function f. To get a value of a rate of change in an image at a certain point, obtaining gradient is not enough. The *magnitude* of this gradient must be computed

$$M(x,y) = \sqrt{\frac{\delta f(x,y)}{\delta x} + \frac{\delta f(x,y)}{\delta y}}, \tag{11}$$

and the direction of gradient at certain point as well

$$\alpha(x,y) = \tan^{-1} \begin{bmatrix} \frac{\delta f(x,y)}{\delta x} \\ \frac{\delta f(x,y)}{\delta y} \end{bmatrix}. \tag{12}$$

As before, to compute derivatives needed for the gradient, the difference is used. Typically forward or central finite difference of the first (4) and second (5) order are used in the case of gradients. However, if not only horizontal and vertical edges are of interest (but diagonal as well), which is always the case in an image, 2D kernels must be used again. The first attempt of a 2D mask with a diagonal preference was the Robert's mask.[19]

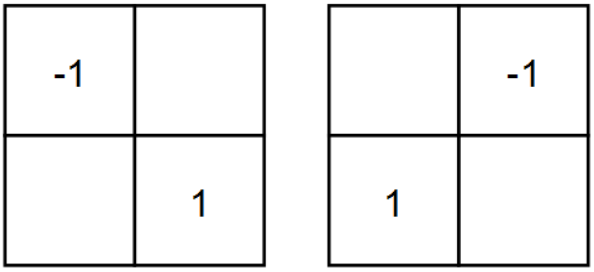


Fig. 15: Robert's masks

As mentioned, Robert's kernels were the first attempt and they were not without flaws. Noteworthy advanced version of Robert's attempt were Prewitt's masks.

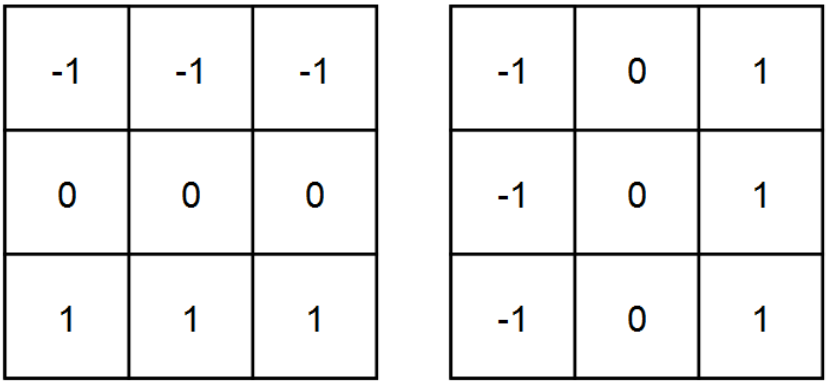


Fig. 16: Prewitt's masks

However, the preffered version of masks for gradient technique using convolution/filtering are Sobel's masks. Prewitt's masks may be easier to implement, but the computational difference is not significant and Sobel's masks seem to perform better regarding edge detection. One of the reasons may be the fact, that Sobel's masks have better noise surpression properties. These masks are shown in the Fig. 17.

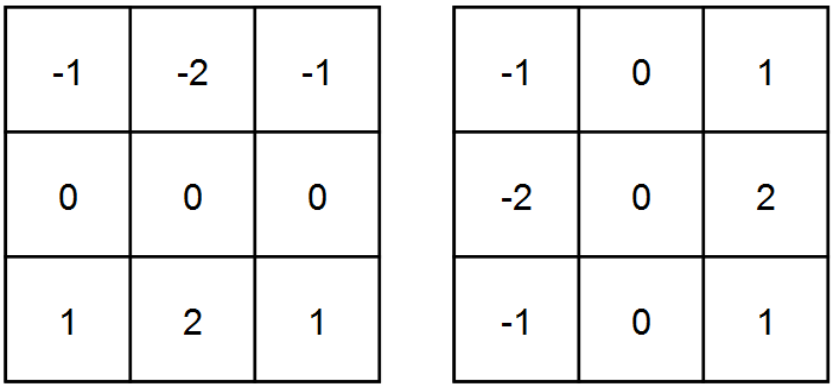


Fig. 17: Sobel's masks

The effect of filtering an image with all three mentioned masks based on gradient is shown in the Fig. 18.

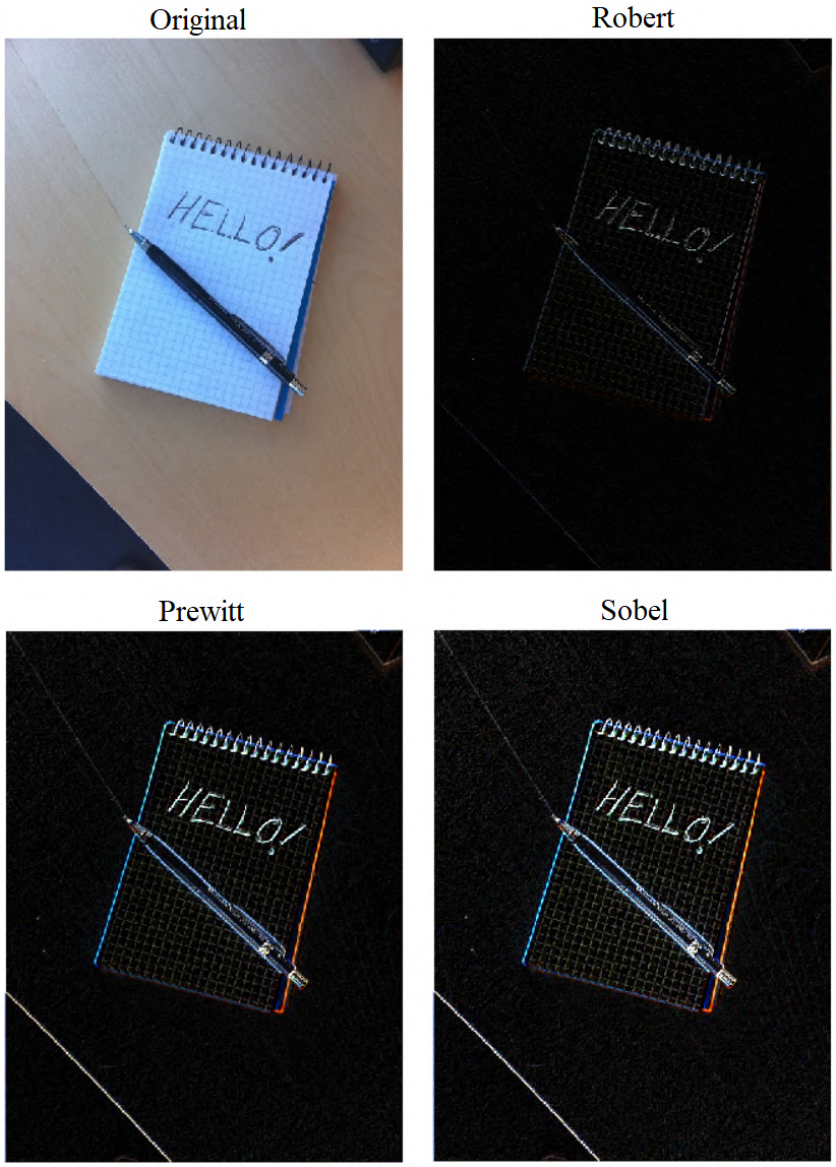


Fig. 18: Robret's, Prewitt's and Sobel's masks used to filter an image (pre-packaged demo image in Matlab, *hello.jpg*)

From the Fig. 18 it is clear why would one use Prewitt's or Sobel's mask for edge detection. It is also visible that Robert's mask has trouble with diagonal edges. The figure is clean without a significant noise, but even in this case Sobel's mask makes a slightly better job than Prewitt's.[19]

Advanced edge detection techniques

One of the techniques widely used to detect edges/contours/boundaries is *LoG*, Laplacian of a Gaussian. It is a filter that combines a Gaussian filter (image blurring,

smoothing) and already mentioned Laplacian. Its big advantage is that with the use of a Gaussian it is possible to surpress noise to some extent. That can be a key aspect in this process, since noise has a significant negative effect on edge detection.

The Gaussian function in 2D may be defined as

$$G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{13}$$

where σ is a standard deviation (space constant in this context) and x, y are coordinates. Since this computationally expensive operation needs to be effectively used on images, again, implementation of masks to be used for filtering takes places. Two examples of such filters are shown in the Fig. 19.[19]

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Fig. 19: Two examples of a Gaussian mask

As mentioned, Gaussian filter blurres/smooths the image when applied. Examples of image smoothing using Gaussian filter with varying standard deviations are shown in the Fig. 20.

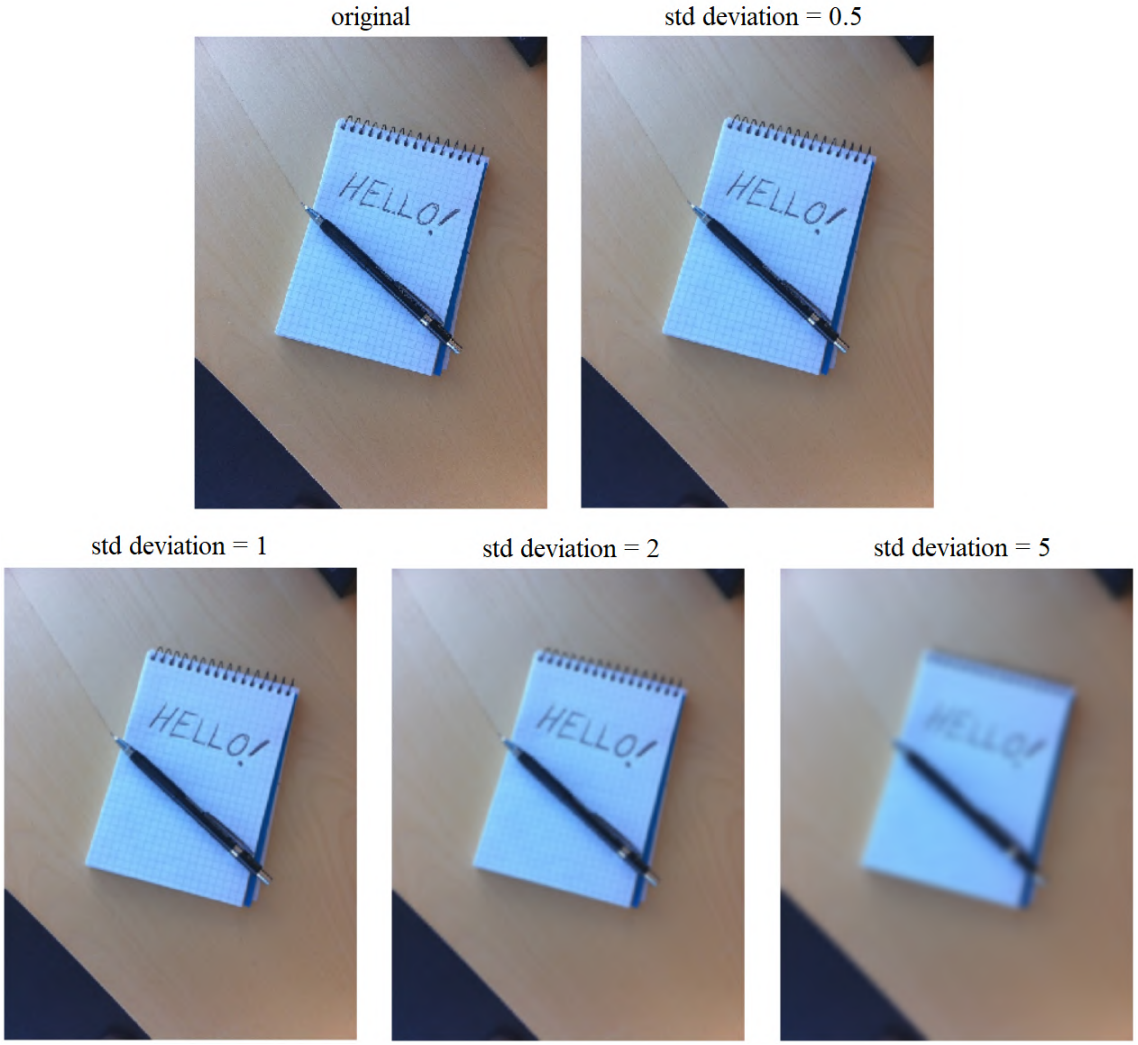


Fig. 20: Applied gaussian filter with different standard deviations

The LoG is, as mentioned, a combination of gaussian filtering and laplacian. It may be obtained by computing the Laplacian of the Gaussian function. That results in the equation:

$$\nabla^2 G(x,y) = \left(\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right) e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{14}$$

The result of LoG with the comparison to Laplacian is shown in the Fig. 21. The original is again pre-packaged demo Matlab image.[19]

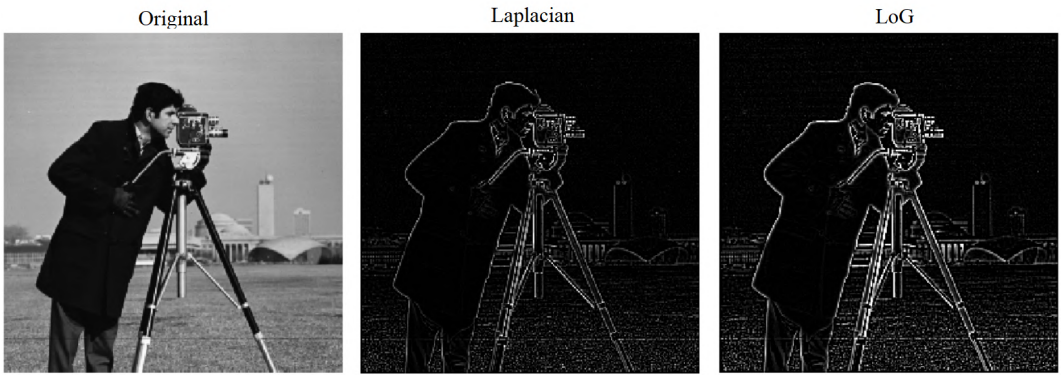


Fig. 21: Effect of LoG and Laplacian

The LoG is then used, for example, in a **Marr-Hildreth edge-detection algorithm**. That is basically applying LoG alongside zero crossing. This algorithm may be summarized as:

1. Filter the input image with a Gaussian lowpass mask
2. Compute the Laplacian of the image from the first step
3. Find the zero crossings of the image from second step

Another well known technique is the *Canny edge detector*. Due to its complexity, it may be computationally expensive, however this technique outperforms all techniques mentioned so far. Without diving into too much detail, the Canny's detector is based on formulation of the following criteria mathematically and then attempting to find optimal solutions to them. The criteria are:

- Low error rate
  As many real edges in the image should be marked as possible without falsely marking spaces that are not an edge.
- Good edge points localization
  Located edges should be marked as close to the real edges as possible. Meaning, the distance between a point marked as an edge and the true edge should be minimal.
- Single edge point response
  Only one point should be returned for each edge point (the number of maxima around the edge should be minimal). That should result in the detector not identifying multiple edge pixels for a single edge point.

It is generally difficult (impossible) to find a closed form solution to the mentioned criteria, numerical approach must be used. In addition it was found that one dimensional edges corrupted with white noise are detected well with first derivative of a Gaussian. So well, it performed around 20% worse than the numerical solution (in most cases it is not possible to see the difference with naked eye). In the two dimensional problem, however, it is not known, which direction the edge will be and

the 1D solution would have to be used in all possible direcitions. That is why the image is at first smoothed (with Gaussian filter), the gradient is computed and its magnitude and direction may be used to estimate edge strenght and direction at every point. The gradient magnitudes and directions have same dimensions as an input image.[19]

Gradient image generally contains wide ridges around maxima. That needs to be thinned by, for example, using *nonmaxima surpression*, where the algorithm goes through all the points on the gradient intensity matrix and finds the pixels with the maximum value in the edge direction. The results contain only thin edges at this point.[43, 19]

Finally, the false edge points are eliminated by *hysteresis thresholding* (which uses a high and low threshold). High threshold identifies strong pixels, low threshold indentifies non-relevant pixels and pixels between are marked as weak pixels. Then hystheresis mechanism helps to identify the last mentioned pixels as strong or weak. This process identifies a weak pixel as a strong one only if at least one of the pixels around the one being processed is a strong one.

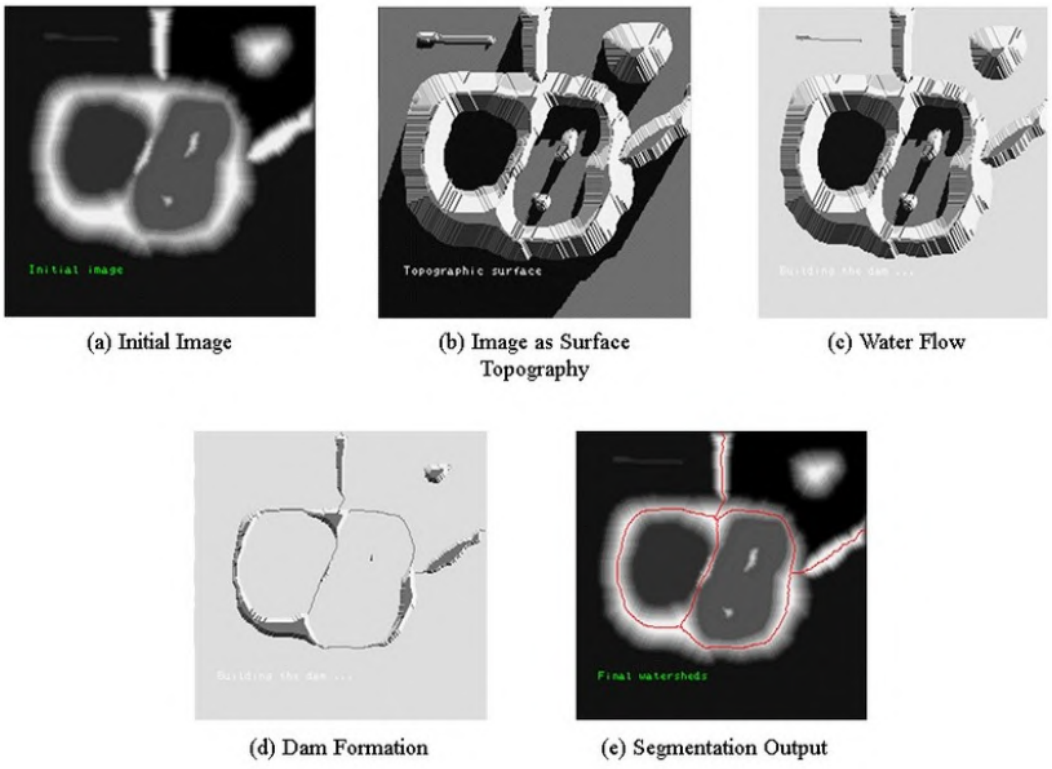
To end this chapter, it is mention-worthy that in practice, these teqchniques are usually used alongside with each other or with another ones (combination of techniques can already be seen in this subsection). It is convenient to combine gradient edge detection techniques with thresholding for example. From the basic concepts, use of combinations of mathematical morphology operations can be a help as well. With a good edge detection it is easier to perform a good segmentation, since with such detection it is more likely that a boundary of a region to be segmented can be found.

3.4 Watershed based methods

The watershed methods are based on the idea of finding the watershed lines (dam boundaries), that prevent the flooding when water rises in each valley (regional minimum). While the water rises, it happens that the water from one valley gets mixed with another over a common peak. At this point a dam/barrier is made to prevent that. The procedure of water filling and barriers building continues until all the peaks are under water. The barriers denote segmentation results of found regions. Images in watershed approach may be pictured as a topographic map with pixel intensity to be represented as height. There are three types of points in this interpretation.

- points that belong to a regional minimum
- points at which a drop of water would travel to a single minimum
- points at which the water could fall into more than one minimum

This concept of finding watershed lines is ilustrated in the Fig. 22.



(a) Initial Image (b) Image as Surface Topography (c) Water Flow

(d) Dam Formation (e) Segmentation Output

Fig. 22: Ilustration of watershed algorithm [8]

As can be seen in the Fig.22, from a made up hole, a minimum of a valley, water rises and the dams are built to prevent the overflow of water from one region to another.[19]

The basic watershed approach tends to give oversegmented results because of any irregularities and noise in images. To make the algorithm more efficient, some pre-processing, post-processing is usually used with an additional functionality or extension of the basic approach.

Image pre-processing

As for image pre-processing, that happening before the segmentation algorithm, it is typical to use *median filter*, followed by finding image *gradient* and *thresholding*. Most of the images contain some kind of noise (typically impulse noise, that is either salt and pepper noise or random value shot noise) due to camera imperfections. It is possible to reduce negative effects of this noise with median filter. Median filter is a non-linear filter that is able to not only surpress noise but also preserve edges under certain conditions. The median filter takes each pixel (when stride is 1) and computes median of the pixel neighbourhood (in this case median is the value separating the higher half from the lower half of pixel intensity values from

chosen neighbourhood). The effect of such filter of 3x3 dimensions applied on a pre-packaged demo image in Matlab is shown in the Fig. 23.

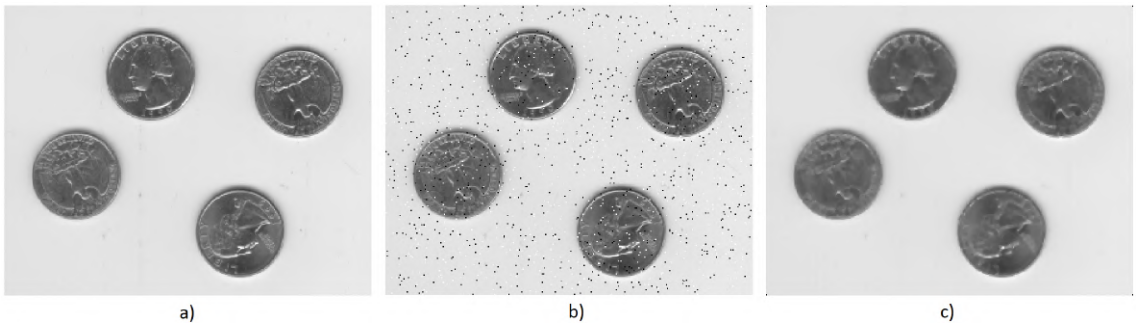


Fig. 23: Effect of median filter

In the Fig. 23 there is original image of coins on a), image with salt and pepper noise applied on b) and the result of applying median filter on b) may be seen on c). It is clear that median filter tends to blur the image. That, however, seems like a feasible trade off with the noise elimination, especially if following techniques are to be applied, and if the output of the process will be input into the segmetnation algorithm.

As mentioned before, after median filter in this context usually another techniques are applied on the processed image. Specifically finding gradients in the image and applying thresholding. Those techniques were already mentioned. Gradients were mentioned in section 3.3 and thresholding was explained in its own section 3.1.[19]

Image post-processing

Even after image pre-processing and watershed segmentation algorithm oversegmentation is likely to appear. To resolve this issue, another image processing may be done. This problem is usually solved by merging smaller regions into bigger ones. One of the most well known and suitable techniques is the *region merging* method. Region merging is based on merging the most similar pairs of ajdacent regions. This method was discussed in section 3.2.

Watershed method modifications

Since the basic watershed tends to oversegment images and the pre/post-processing does not completely resolve this issue, various modifications and extensions to this method were made.

Useful extension to the basic method are **markers**. Marker controlled watershed does not use local minima of the gradient of the image to determine the initial points of the water rising (choosing a region to grow and be segmented). It uses marker positions either explicitly set by user (so an interactive segmentation

with user is required) or automatically determined by a certain approach, typically accompanied with morphological operations. Finding marker positions can vary from basic approaches based on intensity values, connectivity, to more complex approaches considering size, shape, location, relative distances, texture content,... A basic idea behind automatically determining marker positions could be as follows:

- If the image has more than one color channels, convert the image to greyscale
- Find segmentation function (gradient magnitude of the input image)
- Mark the foreground objects
  - multiple procedures could be applied, such as opening and closing followed by a procedure to find a regional maxima
- Mark the background
- Modify the gradient magnitude image so that its only regional minima occure at foreground and background marker pixels
- Apply watershed algorithm[29, 19]

Other modifications to the basic algorithm may be derived from the background idea of building watersheds. For example, the watershed algorithm discussed so far is based on *flooding* (water rises from each regional minimum). Another way to look at the watershed building principle is *watershed by the drop of water principle*. In this case the edge point, where a dam shall be built is not viewed as a place, where water overflows to another valley (or two waters mix), but as a place where a drop of water can descend to different minima. In other words, rain water drops fall on the image seen as topographic surface and move in the descending manner (by the effect of gravity) reaching local minimum in the end. The path of the water drop is followed and saved. This may be done for each point of the image. Segmented regions are created of all initial poins from which rain drops descended to the same local minimum. A threshold of minimal flooding is implemented as well for distinguishing bigger valleys from smaller local hills.[15]

3.5 Clustering based methods

Clustering methods are rather classical approach, however their impact, usage and basis for modern approaches is significant. As the name hints, these algorithms seek clusters in data (images in this case), which results in finding a number of groups in data points, such that data points in the same groups are more similar to each other, than to those in other groups.

Segmentation using K-means clustering

A famous clustering method is K-means. The aim is to divide a set of data points into a specified number of clusters. In this procedure, each data point is assigned to the cluster set with the nearest mean. Typically Euclidean distance is used to

decide, which cluster is closest to each point. The goal of this process is basically finding the cluster sets, such that for all sets the sum of the distances from each point in a set to the mean of that set is minimum. The standard K-means algorithm may be summarized as follows:

- Initialize chosen number (k) of clusters and their means (m_i, $i = 1, 2, .., k$)
- Assign each data point (pixel of the image) to certain cluster (whose mean is the closest)
- Update centroids (cluster centers, means):

$$m_i = \frac{1}{|C_i|} \sum_{z \in C_i} z, \tag{15}$$

  where $|C_i|$ is the number of data points in the cluster C_i and z is a set of data points (belonging to cluster C_i)
- Terminating condition test
  The algorithm ends if the maximum number of iterations is reached, or other ending condition is met. That may be found out by computing Euclidean norm between the means in the current and previous steps. Then error E may be computed by summing up the computed norms. Algorithm stops if $E < T$, where T is a specified threshold.

Finding a minimal solution to mentioned goal of defined problem of K-means clustering is a NP-hard problem. That is why multiple heuristic methods were implemented over time that were able to approximate solution to the mentioned problem.[19]

Segmentation using superpixels

In most image processing tasks, including processes regarding image segmentaion, the base units are pixels. The superpixels are created by perceptual grouping of pixels. These primitive regions are intuitively more meaningful than individual pixels. With superpixels lower computational load and reducing irrelevant detail is achieved. The downsides of using superpixels are an additional computational effort for computing the superpixels as a preprocessing segmentation and a possibility of loosing meaningful edges by placing them into the superpixel. That leads to one of the most important requirements: in superpixel image, boundaries between regions of interest must be preserved. To get the idea of what superpixels are and what the goal of the process is, superpixel segmentation of an input image is shown in the Fig. 24.

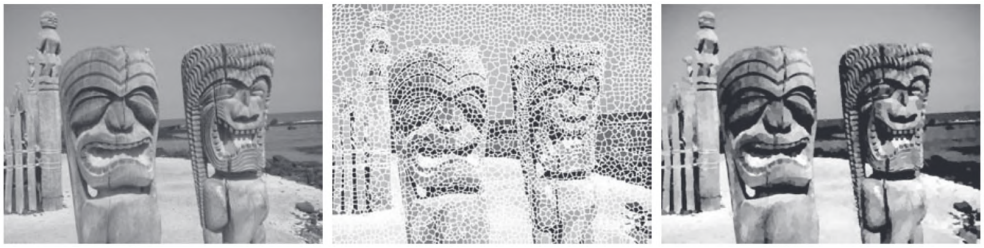


Fig. 24: Example of superpixel segmentation [19]

In the Fig. 24, the first picture on the left displays an input image of the size 600x480 (288,000) pixels. In the middle picture, boundaries between superpixels are shown. They, however, are not part of the data. Third picture on the right displays the superpixel image. The input picture and superpixel picture may seem similar. But the superpixel image contains only 4000 superpixels to be processed. That means lower computational load after the superpixel segmentation is needed and lower number of regions to be processed in the segmentation processed, hence lower space for oversegmentation and segmentation mistakes. The quality of superpixel segmentation also depends on a number of superpixels. If it is set too low, the data loss may be too high and it may result in loosing important region boundaries.[10, 19]

To compute superpixels, an algorithm must be used. The number of options is high, however one of the well known algorithms, that may be explained along already mentioned K-means, is called *SLIC*. SLIC stands for *simple linear iterative clustering*. SLIC is basically a modified K-means clustering. It generates superpixels by clustering pixels based on color and proximity. SLIC observations usually use 5-dimensional vectors, that contain colors (three color channels) and spatial coordinates. If RGB color system is being used, mentioned vector associated with a pixel could look like

$$z = \begin{bmatrix} r \\ g \\ b \\ x \\ y \end{bmatrix}.$$

It is useful to define number of pixels n_p and a desired number of superpixels n_{sp}. Initial superpixel centers m_i are obtained by sampling the image always s units apart. The initial superpixels are approximately of the same area, that is done by selecting $s = (\frac{n_p}{n_{sp}})^{1/2}$. To reduce the chances of centering a superpixel on the edge or starting at the noisy point, the initial cluster centers are moved to the lowest gradient position in the 3x3 neighborhood around each center. The SLIC algorithm may be divided into the following steps:

1. Initialize cluster centers $m_i = [r_i \;\; g_i \;\; b_i \;\; x_i \;\; y_i]^T$ for $i = 1, 2, .., n_{sp}$

2. Move cluster center points to the lowest local gradient position
3. Assign pixels to cluster centers: For each cluster center with m_i compute distance D_ip between m_i and each pixel in $2s \times 2s$ neighborhood around m_i. Then for each pixel and cluster if D_i is lower than current distance d_i on pixel, $D_i = d_i$ and assign current pixel to the cluster with found lower distance.
4. Update the cluster centers: Let C_i be a set of pixels with the same label (belonging to cluster with cluster center m_i). Update m_i by using equation (15)
   for $i = 1, 2, ..., n_{sp}$, where $|C_i|$ is a number of pixels in set C_i and z.
5. Repeat steps 3 and 4 until terminating condition is met:
   The algorithm ends if the maximum number of iterations is reached, or other ending condition is met. That may be found out by computing Euclidean norm between the means in the current and previous steps. Then error E may be computed by summing up the computed norms. Algorithm stops if $E < T$, where T is a specified threshold.
6. Replace all the superpixels in each region C_i by their average value m_i

So far the computation of the distance was not mentioned regarding this method. SLIC superpixels are basically regions in a space whose cooridantes are colors and spatial variables. The spatial and color distances are treated separately. Let the d_c be the color Euclidean distance between points i and j in the image defined by the equation

$$d_c = [(r_i - r_j)^2 + (g_i - g_j)^2 + (b_i - b_j]^2)^{1/2}. \tag{16}$$

Let the d_s be the spatial Euclidean distance between two points i and j in the image

$$d_s = [(x_i - x_j)^2 + (y_i - y_j)^2]^{1/2}. \tag{17}$$

With d_c and d_s a composite distance D may be defined:

$$D = \left[\left(\frac{d_c}{d_{cm}} \right)^2 + \left(\frac{d_s}{d_{sm}} \right)^2 \right]^{1/2}, \tag{18}$$

where d_{cm} and d_{sm} are maximum expected values of d_c and d_s. The maximum spatial distance corresponds to the sampling interval $d_{sm} = s = (\frac{n_p}{n_{sp}})^{1/2}$. Maximum color distance is not simple to define, since it varies a lot from cluster to cluster and image to image. Before algorithm begins, it usually is set to constant c. After simple substitution, the resulting equation for the distance is

$$D = \left[\left(\frac{d_c}{c} \right)^2 + \left(\frac{d_s}{s} \right)^2 \right]^{1/2}, \tag{19}$$

which can be rewritten as

$$D = \left[d_c^2 + \left(\frac{d_s}{s} \right)^2 c^2 \right]^{1/2}, \tag{20}$$

where c can be a weight between the relative importance between color similarity and spatial proximity. When c is large, spatial proximity is more important and superpixels are in the end more compact. If c is small, the resulting superpixels have les regural size and shape, but they tend to follow image boundaries mroe tightly.[19]

3.6 Soft computing methods

So far most of the mentioned segmentation methods were exact, iterative algorithms or their extensions and accompanying procedures. In some specific use cases or useful researches the implementation of soft computing methods for image segmentation is being used. Soft computing is a set of techniques, algoritmhs that are the opposite of conventional computing, they tolerate uncertainty and approximation. They are adaptive and usually learn from experimental data. Soft computing algorithms are often based on artificial inteligence methods. Examples of soft computing methods might be categorized into subgroups like fuzzy logic, genetic algorithms, machine learning or deep learning.

For example, some traditional segmentation methods might be adjusted from hard computing to soft computing, such as fuzzy clustering. Clustering methods were introduced in the section 3.5 and the word fuzzy refers to fuzzy logic. That is a logic accepting more than one value for an observation, in which values of variables can be represented by any real number between 0 and 1. Meaning concepts like (complete) truth and false, but also partial truth are accepted. This logic is able to deal with imprecise and non-numerical information. In fuzzy clustering, the principle of the algorithm stays the same, however each element may belong to more than one cluster, but the value of its "overall membership" is equal to one. Fuzzy logic may be used to "soften" computations on various traditional algorithms including segmentation algorithms.[36]

Genetic algorithms (GA) is another group under soft computing and machine learning methods. GA take inspirtion in the idea behind the process of evolution in nature. They search a global space of the problem to find an optimal solution by miming the evolution process, seeing the task as an optimisation problem, divided into:

- Initialization
  Generation of a population is the first step (number of individuals depends on the size of the problem).

- Selection
  A portion of individuals are chosen using various criterias to breed to make new generation.
- Crossover
  Two (or more) individuals are used to generate new individual(s) combining their properties.
- Mutation
  Mutation is used to maintain diversity of population bringing "randomness" into the computation, which allows the generations to evolve and to not get stuck in local minima.
- Heuristics
  All kinds of additional heuristics in a form of reward or penalty for certain situations during computation are applied to optimize it.
- Termination
  Once the maximum number of generations (iterations) of the process is reached or the termination condition is met, the process is terminated.

Genetic algorithms may be used for pixel level segmentation. As mentione in the [20], Optimization of "*various tasks from basic image contrast and level of detail enhancement, to complex filters and deformable models parameters are solved using this paradigm*". GA are robust algorithms that avoid local extremes.[20, 9]

Another soft computing area the segmentation might be performed by is deep learning. There is a whole chapter dedicated to deep learning and artificial neural networks in this thesis, see chapter 4.

4 DEEP LEARNING

4.1 Artificial neural networks introduction

Deep learning and its neural networks are briefly described in this chapter. The attention is put on convolutional neural networks and architectures for semantic segmentation. Deep learning is part of the machine learning field that uses artificial neural networks as the instrument for data description. For example, neural networks are able to process images, videos, texts, sounds, signals and the standard tasks performed on input data might be classification, prediction, clustering. Examples of real life applications might be object detection in videos, media recommendation, weather forecast or segmentation of medical images.

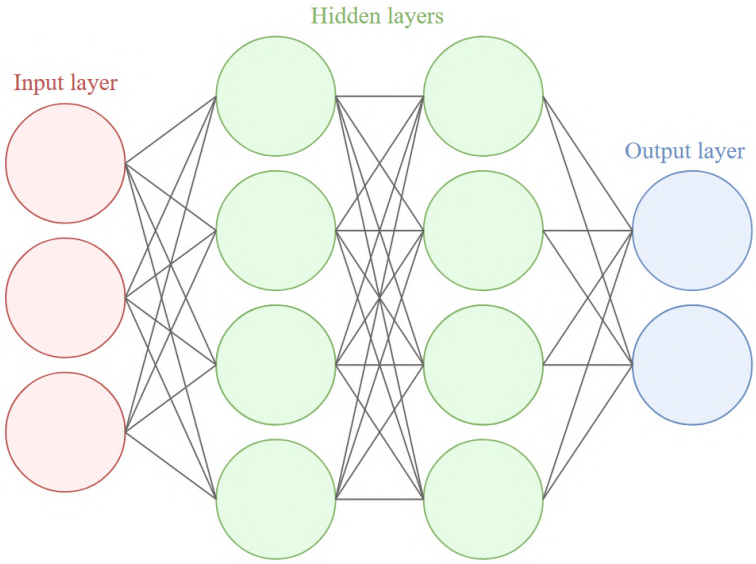


Fig. 25: Illustration of artificial neural network

Artificial neural networks are mathematical models that are able to process input data in parallel manner. These models consist of artificial neurons. The neurons are connected to each other, forming individual layers and each one of them processing the signals they receive and sending them forward. The neurons are defined by their activation function and may standardly have multiple inputs and one output. Standard neural networks consist of mentioned layers, specifically from input, output and hidden layers. As probably is clear from the names, input layer takes data into neural network, output layer takes data from the last hidden layer and outputs the results. The training process of parameters is happening in the hidden layers of neural networks.

4.2 Biological neuron vs artificial neuron

Artificial neural networks were highly influenced by the human brain. The basic unit of a human brain is neuron. The neurons in a brain are interconnected and the principle of the signal flow there is similar to the idea behind neural networks data processing. An illustration of biological neuron is shown in the Fig. 26.

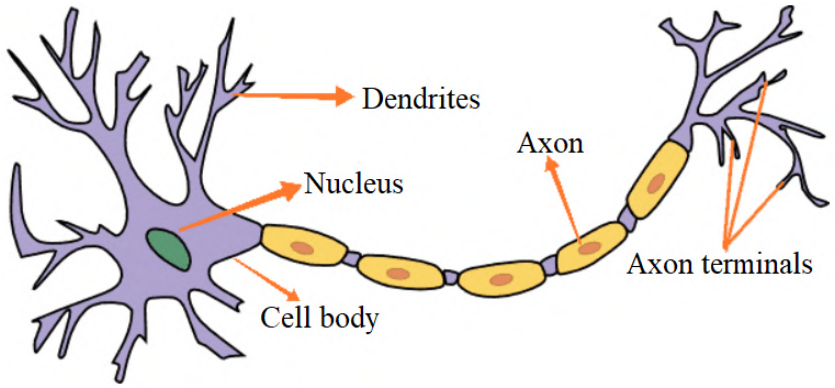


Fig. 26: Scheme of biological neuron[6]

The biological neuron may be divided into the following parts. The first visible partition is into *cell body* and *axon*. In the cell body, there are *dendrites* and *nucleus*. Dendrites are responsible for the intake of signals. Nucleus is responsible for processing the signals. The axon is responsible for transmitting information to different neurons (or muscles, glands). It ends with *axon terminals*, there is usually more of those and they might be connected to other neurons' dendrites. For comparison, artificial neuron is shown in the Fig. 27.

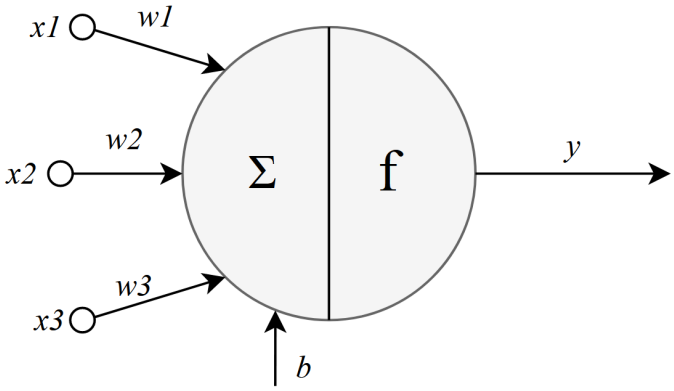


Fig. 27: Scheme of artificial neuron. Taken and modified from [25]

Before describing artificial neuron, it is important to note that the inspiration in biological neuron and brain is very loose. There are various types of neurons having different purposes and morphologies, synapses are complex non-linear systems,

the timing of signals fired by neurons varies and is an important information, and so on... As is shown in the Fig. 27, the artificial neuron resembles the biological one not only by shape, but by mentioned function as well. The x_1, x_2, x_3 are inputs to the neuron (might be outputs from the neurons of the previous layer). The w_1, w_2, w_3 are weights (the tunable parameters of networks) of connections between two neurons or an input and a neuron, their tuning is the goal of neural network training. The weights with the input may be liken to dendrites of the biological neuron. Multiplications of $x_i w_i$ are then summed with bias b and processed by the activation function f. This process might be liken to what nucleus of cell body of the biological neuron does. The output y might be liken to the axon, and may work as an input to other neurons. From the artificial neuron and its variables description, the output (function of a neuron) may be defined (21), as well as the note that this described model is known as **perceptron**.[25]

$$y = f\left(\sum_i x_i w_i + b\right). \tag{21}$$

4.3 Neural networks training

To introduce concepts of feedforward and backpropagation in neural networks, the basic type of neural networks might be used for clarity, the *multilayer perceptron*. It is a feedforward artificial neural network whose basic unit is perceptron. These basic units form layers. This idea was already illustrated on the Fig. 25. The input layer standardly corresponds to the data, output layer is dependant on the task and specified problem and hidden layers usually consists of mentioned perceptrons, that are defined by a chosen activacion function (a non-linearity that takes a single value and performs fixed mathematical operation, more on that in section 4.4). Usually, each neuron in hidden and output layers takes all the outputs from the previous layer as inputs (multiplied by the connection weight). The equation for an output of a neuron may be defined as

$$a_x^i = f(w_{1,x}^{i-1} a_1^{i-1} + w_{2,x}^{i-1} a_2^{i-1} + ... + w_{n,x}^{i-1} a_n^{i-1} + b), \tag{22}$$

where a_x^i is a neuron x from $i-th$ layer, $w_{k,x}^{i-1}$ are k weights from 1 to n going from the $(i-1)th$ layer, a_k^{i-1} are k output values of neurons of the $(i-1)th$ layer going from 1 to n, b is bias and f is chosen activation function. Since the output of one neuron was just described and dimensions of layers of designed neural networks are always known, it is possible to describe the feedforward concept using matrices

$$\mathbf{A^{(i)}} = f\left(\mathbf{W^{(i-1)}} \mathbf{A^{(i-1)}} + \mathbf{B}\right), \tag{23}$$

wich can be rewritten as:

$$\mathbf{A^{(i)}} = f\left(\begin{bmatrix} w_{1,1}^{(i-1)} & w_{1,2}^{(i-1)} & w_{1,3}^{(i-1)} & \dots & w_{1,n}^{(i-1)} \\ w_{2,1}^{(i-1)} & w_{2,2}^{(i-1)} & w_{2,3}^{(i-1)} & \dots & w_{2,n}^{(i-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{m,1}^{(i-1)} & w_{m,2}^{(i-1)} & m_{m,3}^{(i-1)} & \dots & w_{m,n}^{(i-1)} \end{bmatrix} \begin{bmatrix} a_1^{(i-1)} \\ a_2^{(i-1)} \\ \vdots \\ a_m^{(i-1)} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}\right), \tag{24}$$

where n represents the number of outputs from previous layers and m represents the number of neurons in the current layer, the rest of variables corresponds to the description of the output of a neuron (22).[25]

Understanding the inner process of the data flow from input to output in neural networks is not enough to understand how neural networks are trained. The basic concept of how neural networks learn may be summarized as optimizing parameters of the network (weights and biases) to converge to certain values. That is done by choosing the parameter values so that it performs the given task as well as possible, while maximizing the results accuracy. The process may be divided into:

1. Initialize weights and biases randomly
2. Process input(s) using feedforward and obtain result
3. Evaluate result using loss function suitable for the task
4. Depending on previous step, change parameters of neural network
5. Repeat steps 2-4 until convergence is achieved or a maximum number of iterations is reached

In the terminology of the previous process a *loss function* was mentiond. Loss function plays an important part in the evaluation of neural network's performance. It is a way of passing an information to the network about how accurately was the given task performed. Loss function usually takes real output of neural network and a target value into consideration and decides how far the real output is from the target value. There is a large amount of loss functions whose function is based on the task being performed. They differ from simple mean squared error (regression loss) to probabilistic losses like crossentropy. The loss function used in this thesis is mentiond in section 5.2.

Once the data is processed using feedforward and the performance is evaluated by loss function, the parameters of neural network shall be changed according to it. For example, connections going to the output neuron, that should have been, for example in classification task, the correct output should be adjusted to more positive values, if the previous outputs of neurons are positive as well and lowering the values of weights connecting the same output with previous negative neurons. The reverse is applied for incorrect output neuron.

This seems to be a simple rule, however, since all the parameters in the network need to be adjusted accordingly, this rule is not enough and a derivation of the loss function must be used. This can be used to express the problem as

minimizing the loss and loss function by looking for a minimum of loss function with respect to all the parameters the neural network consists of. It is important to add that this process usually does not take only one input before evaluating performance and adjusting parameters, but whole batches of inputs. The reason is fastening the computation. Since the network may consist of milions of parameters, the data may be large and training on single data samples may be precise, but slow. The results during the training process are not as precise, however it is possible to reach convergence of parameters way sooner. The process described in this paragraph is called *stochastic gradient descend*.[25, 35]

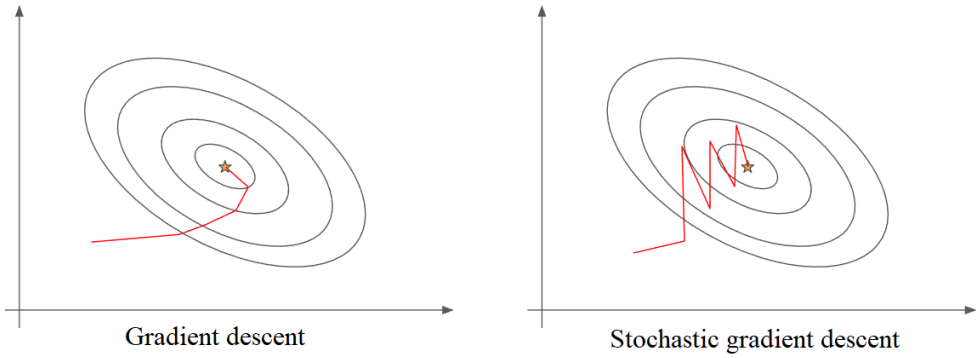


Fig. 28: Stochastic gradient descent illustration [16]

The process of neural network learning that is usually used is called *back-propagation*. It is an algorithm that may be classified as supervised learning. The process goes in the opposite direction than the feedforwad: from the output layer to the input layer. The principle is basically what was described so far in this chapter regarding neural networks learning (it tries to minimize the loss function to reach convergence of parameters). As was already said, parameters of the network are adjusted in the direction of the fastest descend of the loss function (in the opposite direction of the gradient). That means it is desired to find partial derivatives of the loss function by the trainable parameters. To better understand backpropagation, definition of needed variables comes before explanation. Let the a^i be the output of a certain neuron of the ith layer, o_t target output of the network on a certain output neuron, o_r real output of the network on a certain output neuron, w^i a weight of a connection coming out of the ith layer between neurons in this imaginary neurons thread and b^i be the bias of the certain neuron in ith layer. The MSE was taken as the loss function for simplicity

$$L = \frac{1}{2N}\sum_{i=1}^{N}(o_r - o_t)^2, \tag{25}$$

where o_r may be expressed as an output of the neuron (22). To clarify the explanation, the insides of the brackets (function parameters) are substituded in this

equation by helper variable z. To find a partial derivative of the weight of interest (example of one weight will follow, could be used for any weight in neural network), we need to use a *chain rule*, since the parameters to be derived (the weights) by derivation are not explicit in this expression. This rule may be applied to formulas, that express the derivative as a combination of two functions that are possible to derivate. If the two functions f and g are in the relation

$$F(x) = f(g(x)), \tag{26}$$

then the chain rule applied looks like

$$\frac{dF}{dx} = \frac{df}{dg}\frac{dg}{dx}. \tag{27}$$

This then may be applied to the problem to be resolved, that is partially differentiating loss functions by weights

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w}, \tag{28}$$

where L is mentioned loss function, w is weight to be optimized, a is the equation for the neuron output and z is the already mentioned helper variable. The inputs to backpropagation algorithm are the outputs of the neural network, which enter the loss function and the output of this algorihm are the values of changes of the weights. Those changes are then applied to the parameters. To get back to the bigger picture, after backprogapation, another data come to the neural network's input layer and feedforward computation may begin.[25, 35]

4.4 Activation functions

Activation functions define individual neurons. The activation function takes an input (usually more summed inputs) and performs mathematical operation. The purpose of the use of activtion functions is to add non-linearities to the neural network, since without any non-linearity, neural networks would be just linear mathematical models, each neuron performing linear transformation on the inputs using weights and biases (and not being able to learn). In this case, it would not matter how many layers are used in models, since all of them would behave in the same manner. The reason is that the combination of two linear functions is again just a linear function.

Several activation functions may be found in deep learning. A well known example is *sigmoid function*

$$S(x) = \frac{1}{1 + e^{-x}}. \tag{29}$$

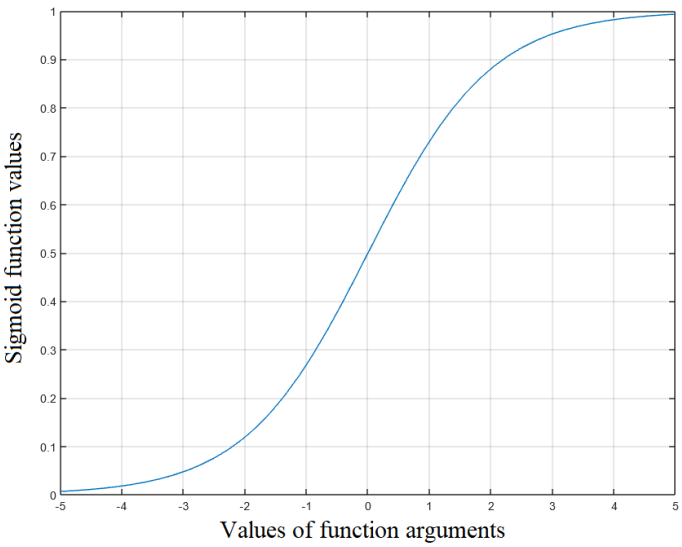


Fig. 29: Sigmoid function

This function is able to squeeze the values to the [0,1] interval, while values further from zero are enclosing towards the values 0 and 1. This function is not really used in practice anymore. Even though it is well known, its simplicity and properties have been overcome by other functions. One of the problems is that this function encloses to 0 and 1 very close to the origin and thus there is a high probability of killing an output of a neuron (values close to zero) or saturation appears. Another issue with sigmoid function is that the output values are always bigger or equal to zero. In other words, the output values are not zero centered. That can result in gradient during backpropagation being only positive (or negative), which makes the training task more difficult.[25]

Another activation function is called *hyperbolic tangent*

$$\tanh(x) = \frac{sinh(x)}{cosh(x)}. \tag{30}$$

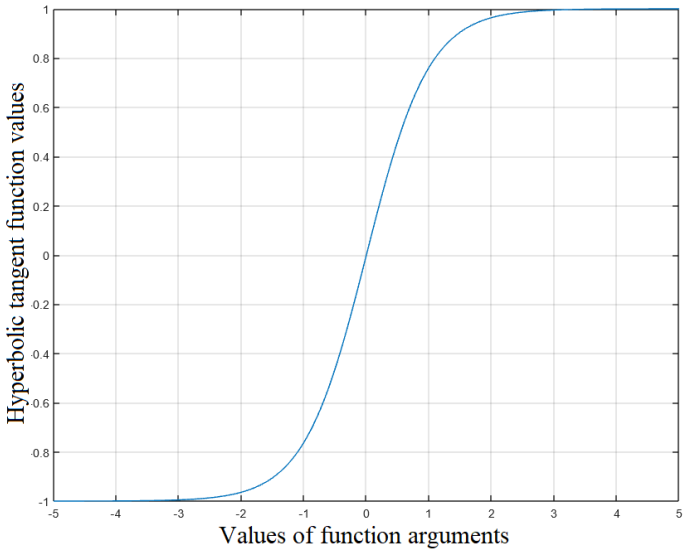


Fig. 30: Hyperbolic tangent

As can be seen, hyperbolic tangent is nothing but sigmoid differently scaled and shifted. The equation for this function then might be rewritten as

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}. \tag{31}$$

The effect of hyperbolic tangent may be seen on the previous picture as well. It squeezes values into [-1,1] interval, so the outputs are zero centered at this point, however the killing and saturation of neurons persist. For the mentioned reasons, hyperbolic tangent is usually preferred against the sigmoid.[25, 35]

The famous and probably one of the most used activation functions nowadays is $ReLU$. That stands for Rectified Linear Unit

$$f(x) = \max(0, x). \tag{32}$$

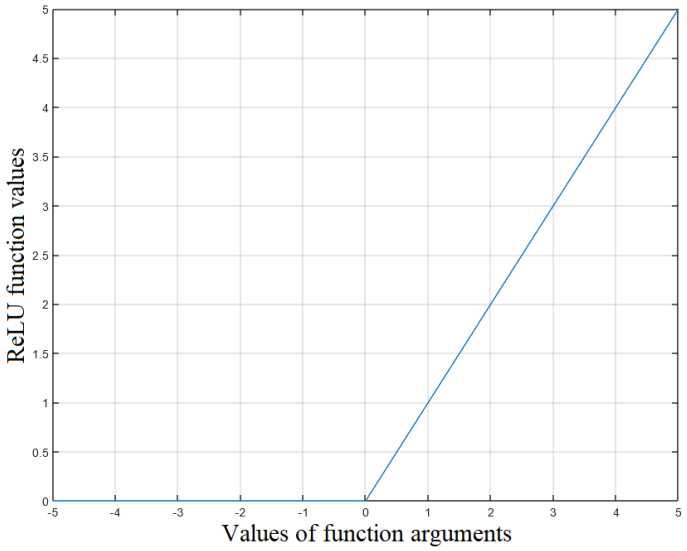


Fig. 31: ReLU function

To summarize this function, all the values lower or equal to zero change to zero and all the values greater than zero keep their original value. The big advantage of ReLU is that its computational complexity is way smaller than previous mentioned functions and it fastens up the stochastic gradient descent process. Another advantage is that neurons are not saturated. On the other hand, all the neurons with output value lower or equal to zero are killed. Alternatives trying to solve mentioned issue are*Leaky ReLU* and *Parametric ReLU*.[25]

Leaky ReLU has the same part as normal ReLU for the values greater than zero. A small slope (a) is given to the values that are lower than zero to avoid killing neurons

$$f(x) = \max(ax, x). \tag{33}$$

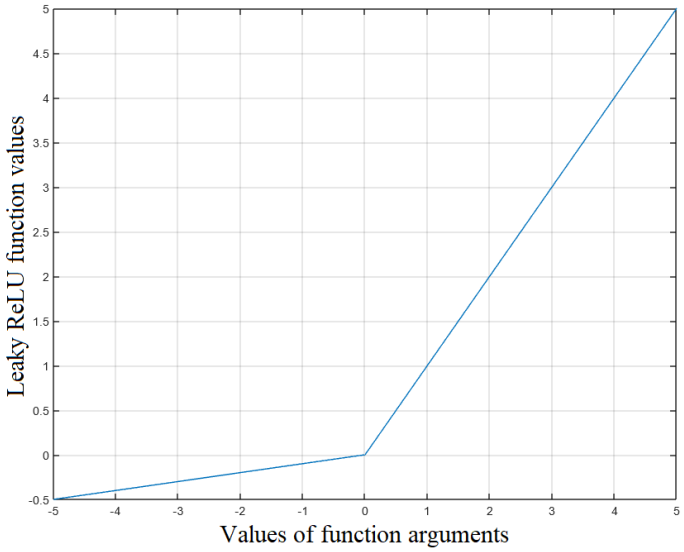


Fig. 32: Leaky ReLU function

Parametric ReLU takes it even further. It does not give a fixed slope (a) to the values lower than zero, but this coefficient of a slope is made into a parameter that is being learned by the network itself while training.

Softmax is another important activation function. Thanks to softmax, the neural network does not output seemingly random numbers (real-valued numbers positive or negative of various distances from zero), but the function recomputes the outputs into the probabilistic distribution, where the summed outputs are equal to one. This activation function is suitable for multi-class classification. However, all kinds of problems may be transfered to multi-class classification problem, for example, this activation function is used in the most segmentation models as well

$$\sigma(Z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}. \tag{34}$$

In the Softmax (34) the K is the number of classes, z_i are individual elements of the input (vector Z), to each input element exponentional function is applied (e^{z_i}), which will put all the values above 0 having negative values smaller than positive values. With the $\sum_{j=1}^{K} e^{z_j}$ term the normalization is performed, that will ensure that after summing all the final output values, the sum is equal to one. That results in a probability distribution of the neural network output.[51]

4.5 Types of neural network architectures

In this section the most used types of neural network architectures in practice are explained. The first type was already partially explained in this thesis, the *multilayer perceptron*.

Multilayer perceptron

Also known as fully-connected neural network. This type of architecture is the simplest one that will be explained. Since neural network training in section 4.3 was already described (feedforward processing, backprogapation, SGD..), there is not a lot more to describe. Multilayer perceptron standardly consists of the input and output layer and a certain amount of hidden layers. It may be compared to the Fig. 25. To put it to words: between layers, all the neurons are interconnected, neurons are defined only by their activation function and may be represented as perceptrons.

Recurrent neural networks

Recurrent neural networks (RNNs) is a type of neural networks, that is typically used on sequential data. Therefore typical use cases of these models are natural language processing, speech recognition or various prediction problems (e.g. next word in a sentence, weather forecast). The typical feature of RNNs is their memory, which allows the network to put more weight to the current inputs than to the prior ones for example. This is important, since unlike traditional deep neural networks, the outputs of RNNs and the data themselves are dependent on each other.[25, 17] The RNN scheme is displayed in the Fig. 33.

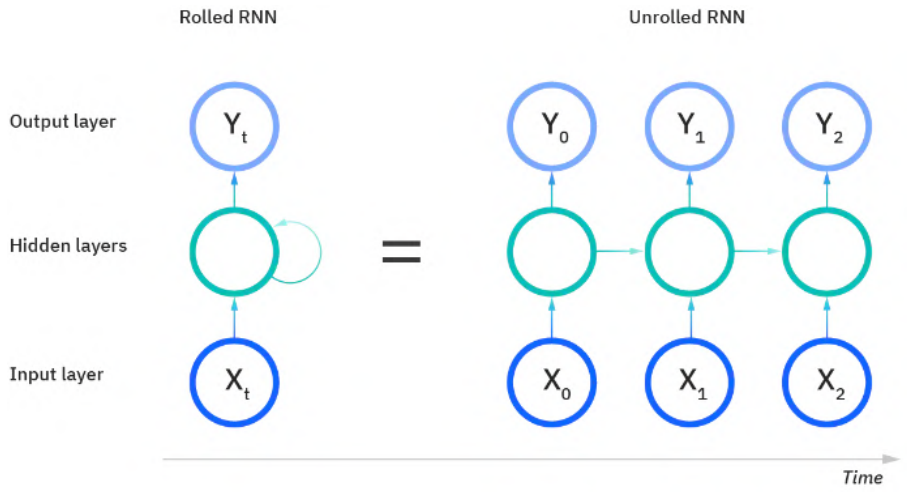


Fig. 33: Recurrent neural network scheme [17]

The "rolled" scheme of RNN in the Fig. 33 represents the whole neural network, whereas on the "unrolled" scheme individual layers (time steps) of RNN may be seen. The function of general RNN may be described as follows. The model takes in the input and computes the output in the feed forward manner. This output may be compared to the target output and evaluated by the loss function. The weight adjustment may happen by backpropagation algorithm. The data, however, do not continue to another layer, but the output of this model, layer, is now an input again to the model itself. Each state of the model (each output) Y_t is a function of input

X_t and previous state Y_{t-1}. That is represented in the Fig. 33. In other words, unlike feedforward-like networks, RNNs share the same weights within each layer throughout the model. These weights are still updated by backpropagation and SGD.

RNNs do not use the typical version of backpropagation, they use backpropagation through time (BPTT) algorithm. The idea is the same (calculating errors from its output to input layer), but it is tailored to sequence data and this type of neural networks. The difference is that BPTT sums errors at each time step, (feedforward networks do not need to sum errors as they do not share parameters across multiple layers).[25, 17]

Because of the use of this BPTT, RNNs tend to face exploding and vanishing gradients problem. Meaning when the gradient is too small, it tends to become even smaller while adjusting weights until they become insignificant (and when this occurs, the model stops learning). The other problem happens when the gradient is too large, that results in an unstable model with weights growing too large as well. This issue is usually resolved by lowering the number of layers in the model making it less complex.

Unlike multilayer perceptron or convolutional neural networks, RNNs do not have fixed size. That can result in a significant computational load. That is why in practice the network is trained "part by part" while using SGD as well. But one of the biggest problem of RNNs is memory loss. The result of memory loss is the fact, that, for example, process that happened thousands time steps before the current one will have significantly less importance. That is why RNNs are being implemented with Long Short-Term Memory, where another connections with computations is added to the model, where the data are filtered and saved after evaluation with respect to given priorities given by the model implementation (mentioned later on). Another fact is that RNNs can also be multilayered. The number of layers is significantly lower than in other neural networks types because of the mentioned computational expensiveness. [25]

RNNs do not share with feedforward neural networks the constraint about mapping one input to one output layer. Their inputs and outputs can vary in length, each type used for different use cases. The typical types are shown in the Fig. 34.

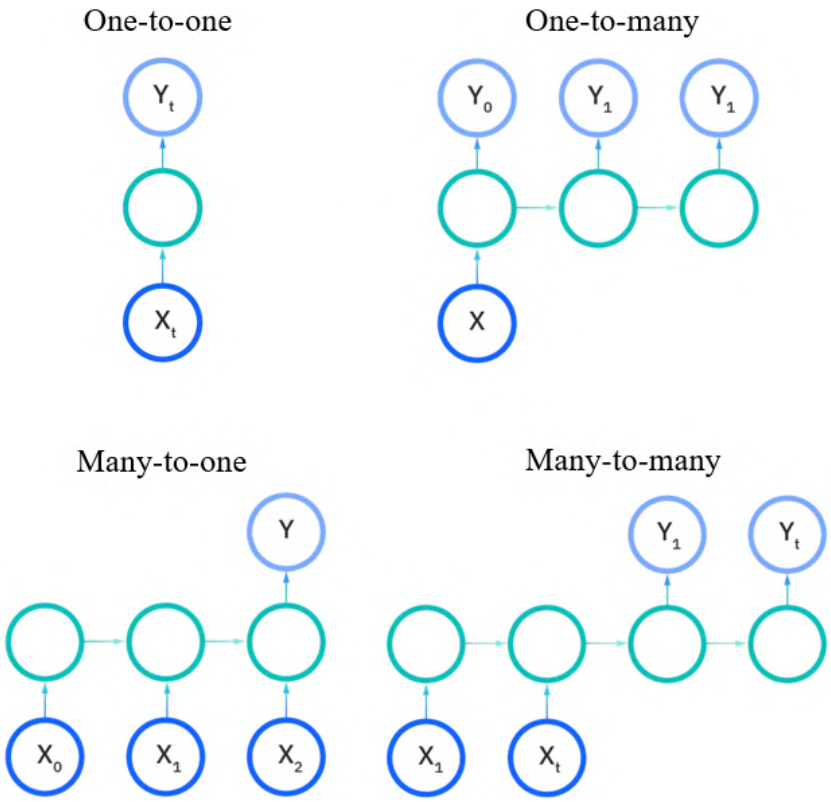


Fig. 34: Types of recurrent neural networks[17]

To conclude this brief overview of recurrent neural networks, the variants and advanced extensions will be outlined.

- Bidirectional recurrent neural networks
  Unlike original RNNs, not only previous inputs are considered to make predictions, the future data are used as well to predict the current state/output.
- Models with long short-term memory
  This is widely used variant of RNNs, that adresses the long-term dependencies problem (and its exploding and vanishing gradients). The problem occurs when the prior state affects the current prediction and is in distant past, the RNN model will not take it into consideration and will not predict accurately. To surpress this problem, LSTMs add "cells" into the model, that consist of three gates (an input gate, an output gate, and a forget gate). The flow of information and model's memory is controlled by these cells.
- Models with gated recurrent units
  Similarly to the LSTMs, GRUs also try to solve the short-term memory problem of RNN models. However, they do not use a cell states, but hidden states with reset and update gates. These states also control how much and what information is gonna be kept for future use as a memory. [17, 3]

Convolutional neural networks

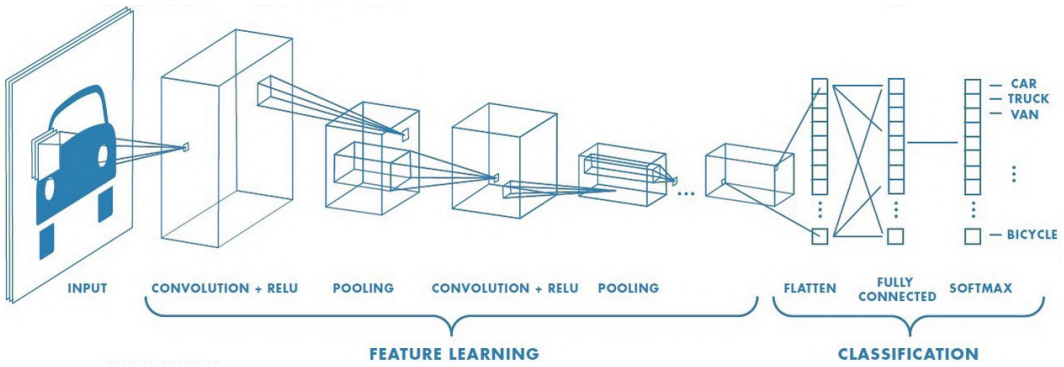


Fig. 35: Convolutional neural network illustration [42]

Convolutional neural networks (CNNs) process data, like multilayer perceptron, in the feedforward manner, unlike RNNs. All the introduced concepts from loss functions, to backpropagation or stochastic gradient descent are used in them as well. The biggest deviation in these models from multilayer perceptron is the assumption, that the input is going to be an image (or images). With this assumption, several specific techniques may be used while implementing CNN architectures.

As was already pointed out, CNNs are suitable for operations done on images related data. A few of many use cases might be image recognition, classification, object detection (in cameras, self-driving cars), face recognition (social media, videos), or image analysis in healthcare (such as segmentation of medical images).

Convolutional layer

Convolutional layer is the place where CNNs make most of the computations and parameter tuning. The learnable parameters are placed in convolutional layers in filters/kernels. Every filter is usually spatially small compared to the layer input. Typically they have dimensions of 5x5 or 3x3 with depth equal to the input depth (so total dimensions of convolutional filter might be for example 5x5x3 for RGB image in the sense of (height)×(width)×(depth)). On each layer there is a certain number of filters, usually the deeper in the network the layer is, the more filters it consists of.[25]

During the forward data flow in the CNN, the filter convolves across the input width and height with a specified stride. On each position of the filter the dot product is computed between the values of the filter and parts of the input that overlap with the filter. After the whole input is convolved by one filter, a 2-dimensional activation map is produced. This convolution process is illustrated in the Fig. 36.

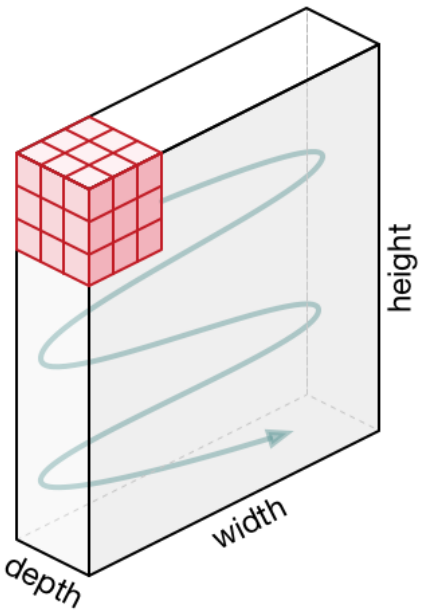


Fig. 36: Convolutional kernel movement [47]

The dimensions of the output of any convolutional layer are directly affected by three parameters. The depth is affected by the number of filters in the convolutional layer since the depth is equal to the number of filters. The height and width are affected by the kernel size, stride (the bigger stride, the lower dimensions, stride is usually equal to two in practice), and by zero padding. Zero padding means that the input to the convolutional layer is padded with zeros (it allow us to control the spatial size of the output volumes).[25]

0	0	0	0	0	0	0
0						0
0						0
0						0
0						0
0						0
0	0	0	0	0	0	0

Fig. 37: Zero padding illustration

Quite recent development [53] introduced a new feature in convolutional layers- **dilation**. Dilated convolutions do not implement contiguous filters (filters with dilation equal to zero). Dilated filters with spaces between each cell are used (see Fig. 38).

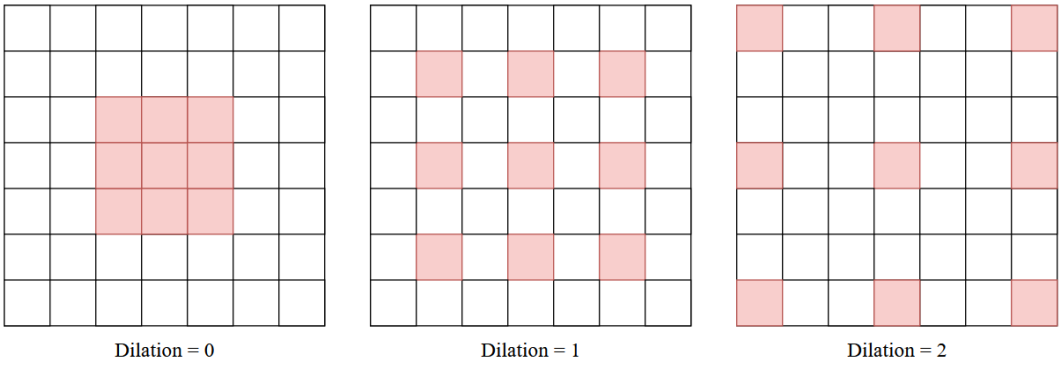


Fig. 38: Dilated convolution

Thanks to this technique, spatial information is merged across the inputs more agressively and the receptive field grows much quicker.

ReLU layler

Hidden layers of CNNs cannot be made only of convolutional layers, since they do not perform anything else but linear operations (multiplications and summing). The neural network needs non-linearities to be able to learn and that is when ReLU layer comes in. Typically in CNNs architectures, after convolutional layer comes a layer with activation function. Activation functions were already described in section 4.4, where more detail may be found. A layer with ReLU activation function is widely used in CNNs.

Pooling layler

Pooling layer's purpose is to reduce the amount of data that is being processed by the neural network. As a result, the spatial size of activation maps is reduced and hence the following computation in the network is reduced as well. With the amount of pooling layers and the size of their filters the data flow may be controlled and thanks to that, a potential overfitting may be surpressed as well. Balancing the pooling in the network is important, because without it the amount of computations grows with the number of filters and it might not even be able to process the inputs without it. Too large pooling may result in an important information loss.

The pooling may be done, for example, by *max pooling* or *average pooling*. The average pooling was used historically and is not used anymore, since max pooling is standardly showing better performance in practice. It is important to notice

that the pooling process downsamples the volume spatially and each depth slice independently. The max and average pooling principle is shown in the Fig. 39.

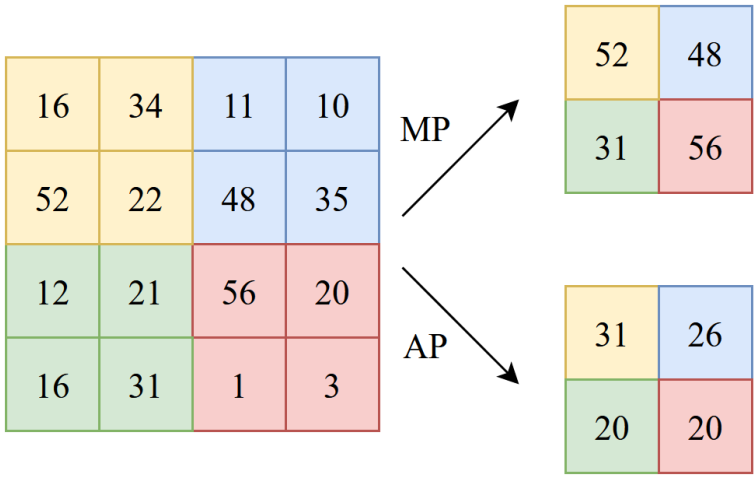


Fig. 39: Average (AP) and Max (MP) pooling with 2x2 filter

Current development is heading in the direction without pooling layers. In some architectures (generative adversarial networks, variational encoders) it seems it is important to discard pooling layers and the tendencies are to come up with models without them. The solution for handeling the volume of data in the network and hence the computations in these cases may be solved by enlarging stride in convolutional layers or using dilated convolutions.[25]

Fully connected layler

These layers are connected to all the outputs of the previous layer. They were already partially introduced, since they may be found in regular neural networks, such as multilayer perceptron. Each neuron then takes all the inputs and creates one output. The usual use case is at the end of CNNs before the output layer. Before each fully connected layer there is a method (flatten layer) to transform the data of two, three dimensions into a vector, that the fully connected layer is able to take as an input.

4.6 Semantic segmentation

Since semantic segmentation, a specific use case of neural networks, was done in this thesis, it will be inspected in more detail. Convolutional neural networks (CNNs) proved to be efficient in performing computer vision tasks on images and nowadays they outperform the traditional methods or even humans in some cases. The typical computer vision tasks solved by neural networks are classification, detection and segmentation. In image classification, the task is to give labels (of classes) to the

objects present in the input image. The initial task of classification classifies the whole image. For example, if there is specific class to be classified from the set of classes, pictures containing and representing this class are labeled so and that is the target output of the neural network, the label corresponding to the correct class.[28]

The object detection extends the image classification task by looking for all the objects (that are classes of the given task) in the image while figuring out the objects' location in the meantime. The localisation is usually represented by bounding boxes and may be seen in the Fig. 40.

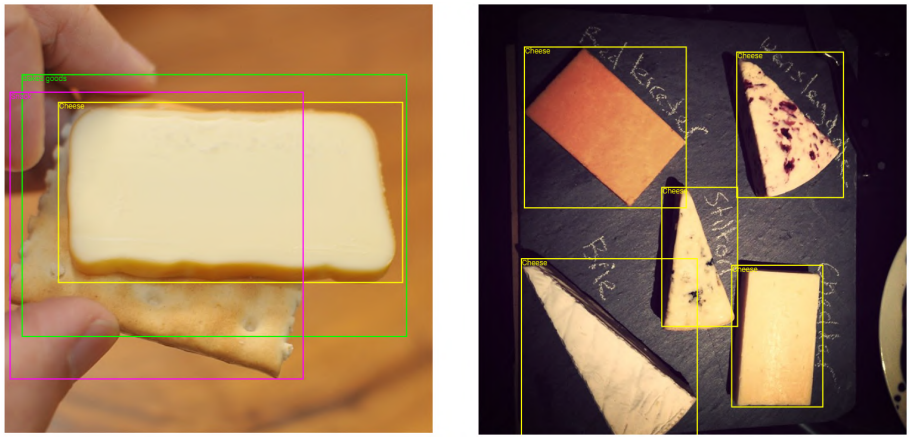


Fig. 40: Examples of object detection [1]

Segmentation (the segmentation itself is described in detail in the chapter 3) takes it even further and tries to find boundaries of the objects in the image. In other words, segmentation is basically classification of each pixel, whether it belongs to a class of a class set or the background. The segmentations done by neural networks may be divided into two types. The first one is **semantic segmentation**, in which the task is basically classification of each pixel into classes by labels, while the instances of objects are not differentiated. That means that if there are more objects of the same class in one image, they are all labeled by the same label. The second type of segmentation using neural networks is **instance segmentation**. This segmentation differs from semantic segmentation with giving unique labels to every instance of an object of given class. Examples of image segmentation are shown in the Fig. 41.[33, 28]

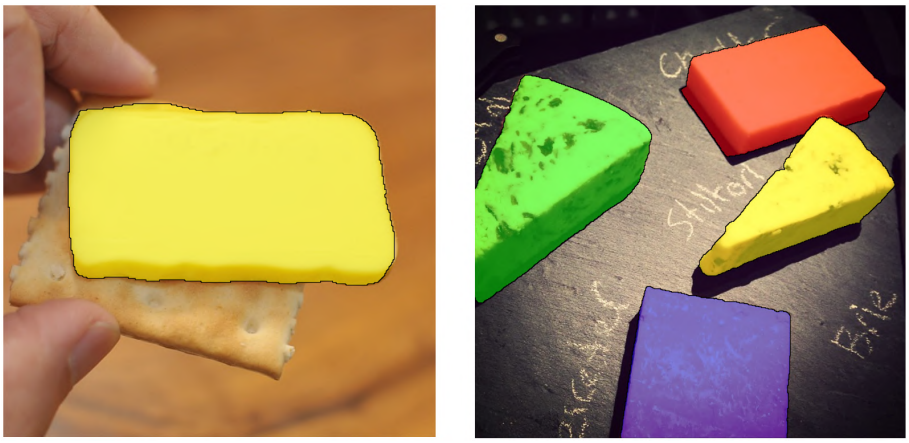


Fig. 41: Examples of image segmentation [1]

Some of the interesting use cases of mentioned segmentations might be recognition of handwriting in documents, image enhancements of photographs, portraits, in self driving cars and a significant number of use cases in medical or microscopy images on data like images of brain, tissues, organs containing interesting attributes from the medical perspective like tumors or other deffects or diseases.

4.7 Segmentation neural network architectures

Since segmentation task is usually performed on images, from the variety of neural network models it is suitable to choose convolutional ones. CNNs usually consist of combination of convolutional (with activation layers) and pooling layers accompanied with additional techniques, and might be finalized with one or more fully connected layers. Convolutional networks must be adjusted for the segmentation task. It usually uses convolutional and pooling layers as well to decompress an input image. At this point a prediction of each "pixel" in activation map is made and then up-sampling/deconvolution layers are used to resize the image to the original size. These models usually do not have fully connected layers. The main thought behind down-sampling followed by up-sampling is that as like in traditional CNNs, while down-sampling, the network is trying to learn semantic information from the input. The up-sampling recovers spatial information. For the recovery of lost information due to down-sampling, skip connection are used. Skip connections are connections thay skip a certain number of layers, usually connecting layers on the corresponding level of down-sampling and up-sampling layers. It is used to pass information from down-sampling step to up-sampling step to merge features on all resolution levels to combine contextual/semantic information with spatial information.[25, 39]
It is custom to call the down-sampling part of these CNNs, that is extracting image features by convolutional kernels, *encoder* and the up-sampling part, that generates

the final output, *decoder*. The goal of the decoder, in other words, is to semantically project the features learnt by the encoder onto the pixel space. The mean to learn contextual information are weights of the CNNs, in this case parameters of convolutional filters. The output of such architecture usually is segmentation mask containing pixel-wise labeling of the objects in the image. Most of the architectures that are used for semantic segmentation form this U (V) shape divisible into encoder and decoder. This typical shape of neural network architecture consisting of mentioned features is also shown in the Fig. 43. [34, 24]

One of the first architectures developed in this manner that laid basis for plenty of segmentation models used nowadays is called *U-Net*.

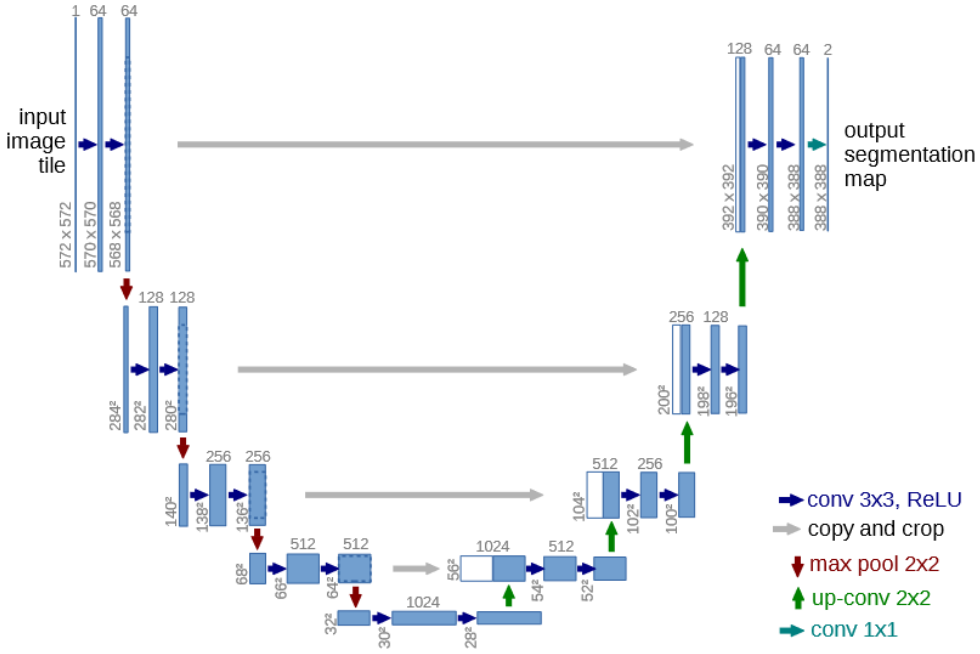


Fig. 42: U-Net architecture [39]

In the Fig. 42 the U-Net architecture from the original paper is displayed and the blue boxes represent feature maps with the number of channels noted above those boxes. The spatial size of feature maps are noted at the lower left edge of the feature maps. Copied feature maps are denoted in white color. Various operations between feature maps are denoted with arrows, legend is provided on the bottom right of the image. Important note: the shape of U-Net used in the practical part of this thesis was different than the one shown in the Fig. 42, so that the model outputs the segmentation map of the same dimensions as the input, just as all the trained models.

U-Net (displayed in te Fig. 42) was originally built for biomedical images. It uses the described encoder, decoder architecture with skip connections. What is important to mention, it was U-Net that was the first architecture with skip

connections. On each level in the encoder, two convolutional layers followed by ReLU are used and then pooled, and decoder mirrors this first part of the model.[28, 33]

A similar architecture to already described U-Net is *SegNet*. SegNet was introduced in 2015 right after U-Net. The mention is important not due to new technical advancements it brought with, but due to the performance on the task of this thesis. To inspect this architecture in more detail, see Fig. 43. SegNet differs from U-Net with its depth (it consists of 5 convolutional blocks, unlike U-Net that consists of 3). The second distinction is the implementation of skip connections, where they transfer only pooling indices from encoder to decoder, unlike U-Net which transfers whole feature maps from encoder's convolutional layers. This results in less demanding computations.[5]

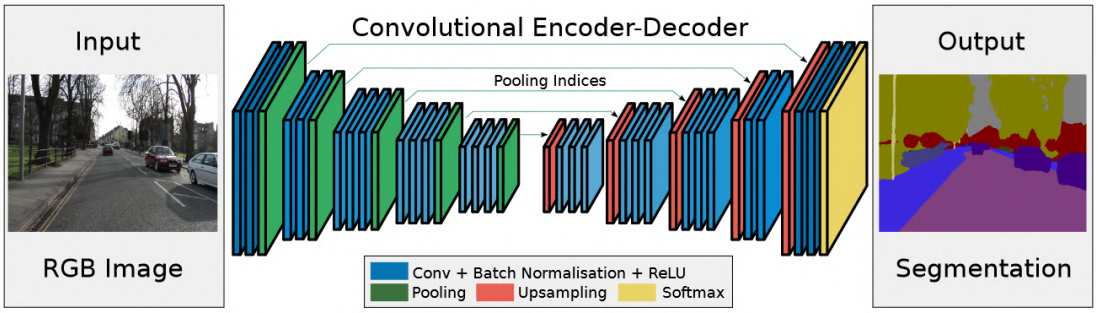


Fig. 43: SegNet architecture [5]

As can be seen in the Fig. 43, this architecture can be broken down into three parts. Encoder, corresponding decoder and final (pixel-wise) classification layer. The encoder of the original model consists of 13 convolutional layers which correspond to the first 13 convolutional layers in the VGG16 network. After each convolutional layer there is a bach normalization applied on a feature map created by precedent convolutional filters. The layers denoted in green in the Fig. 43 are pooling layers that compress the data flow inside the net. The decoder consists of the same number of convolutional layers that the encoder has and consist of as many upsampling layers as pooling layers in encoder.

Listing and describing segmentation neural network architectures might be enough for its own work. That is why only the architectures and their features used in the practical part of this thesis will be described. The initial research regarding the choice of architectures, that might be suitable for this use case was driven by the data. The data are very specific and it was obvious that no pre-trained neural network will be available for transfer learning. The data, as mentioned in the subsection 5.1.1, are microscopic images of heart tissue cells- cardiomyocytes. The significant constraint of this problem is that the number of images is relatively small (thirty three). That is why architectures, that have proven to be effective

on medical, microscopy images and small datasets, were chosen to be trained and evaluated.

The first type of modifications used is different *backbones* for the U-Net architecture. The backbones are variations of convolutional neural networks that might be used in the chosen model as the feature extracting part (encoder). Its main function is to encode the input into a feature representation and the model (U-Net) is adjusted and wrapped around this backbone. In other words, instead of using the basic U-Net implementation, architectures like VGG, ResNet, Xception might be used as the encoder and the decoder is implemented in the same manner like in the case of U-Net, but adjusted to the given backbone. Two of the mentioned backbones (Xception and VGG) were actually used in the practical part of this thesis.

Other models that are proven to be effective on segmentation tasks usually took the idea from U-Net (with possible backbone) and additional features or techniques were implemented and added to the model. For example *R2U-Net* (recurrent residual network based on U-Net) utilizes the residual units to help with training deep architectures. The residual units (skip connections) jumps over few layers and interconnects non neighboring layers. These skip connections usually skip 2-3 connections between layers and they might contain ReLU and possibly normalization layer. The residual units are added to models to reduce the vanishing gradient problem. Some architectures result in worse training performance when adding deeper and deeper layers, that can be minimized by using these units as well. The second feature added to this architecture is recurrent blocks on each layer. Recurrent blocks were described in more detail in the section 4.5. The implementation of recurrent blocks allows the network feature accumulation, which ensures better feature representation. That is a major improvement in segmentation tasks. To summarize, using these features makes it possible to implement improved U-Net model with the same number of parameters while upgrading performance on segmentation tasks (speficially tested on medical images).[2]

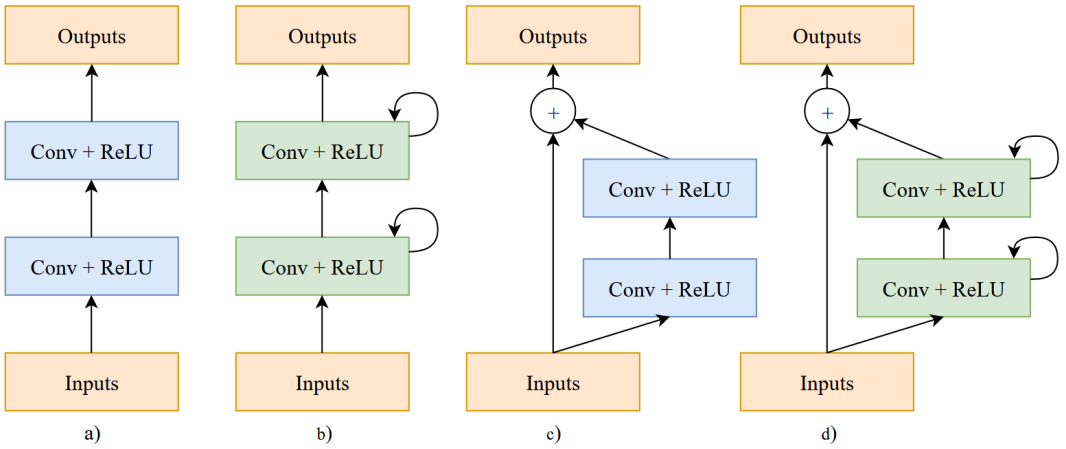


Fig. 44: Variants of convolutional and recurrent convolutional units [2]

In the Fig. 44 variants of convolutional/recurrent convolutional units are shown. The inputs in this figure are activations maps from preceding layers and the outputs are activation maps produced by the structures shown. The subfigure 44 a) represents a conolutional block present in U-Net (shown on Fig. 42). Exchanging the original convolutional block in U-Net for the recurrent convolutional block shown in the subfigure 44 b) it is possible to obtain RU-Net (Recurrent U-Net architecture). On the subfigure 44 c) a residual, skipping block is introduced, which is essential for R2-Unet architecture (Recurrent residual U-Net) using the block shown in the subfigure 44 d), where, as the title of the model says, both recurrent and residual blocks are implemented on top of the basic convolutional block.

Attention U-Net is another major modification of U-Net architecture that was used in the practical part of this thesis. As the source says, an attention gate model is introduced with this architecture that should be able to focus on structures of various shapes and sizes specifically tuned for medical images. These attention gates should be possible to implement on any convolutional model suitable for segmentation task as well. The attention gates are taken from and already used in natural language processing or natural image analysis.[37]

Attention allows the model to amplify the important parts of images and provides more robust pixel classification in the segmentation case. This fact also results in surpressing the feature responses of the irrelevant pixels in the image, mostly the background. By implementing attention gates the localisation and segmentation steps are separated. The attention mechanism was improved by using grid-attention on the U-Net architecture. That means that instead of the gating signal as a global vector, more of them are positioned in the mentioned grid, which allows the process to pay more attention to local regions. Attention gates essentially filter the features that are being transported by skip connections. The attention block and its placement in the U-Net model is shown in the Fig. 45.[37]

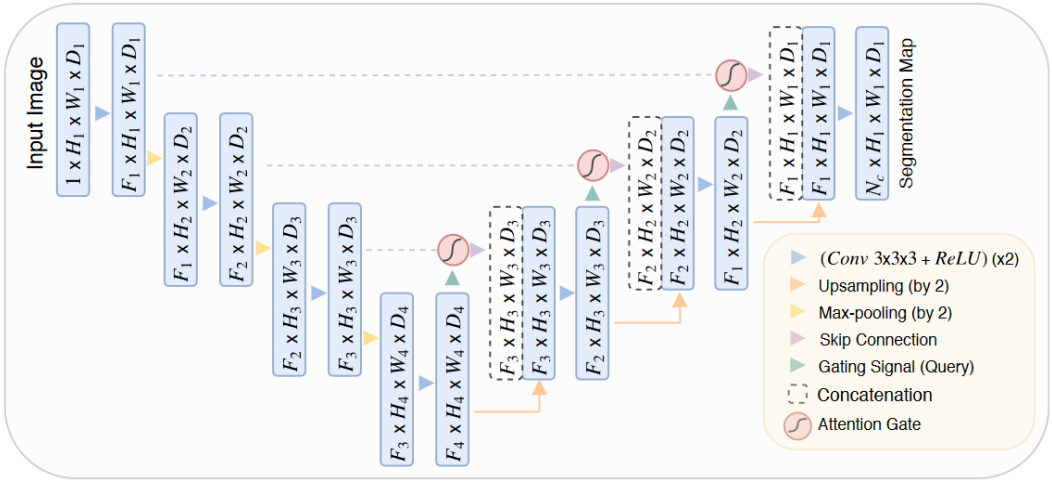


Fig. 45: Attention U-Net architecture [37]

AGs are placed right before concatenation to keep only relevant information. Information coming from irrelevant areas in the image are down-weighted, which supports the learning that concentrates on the the relevant areas in the image in the following layers. The attention block is shown in more detail in the Fig. 46.

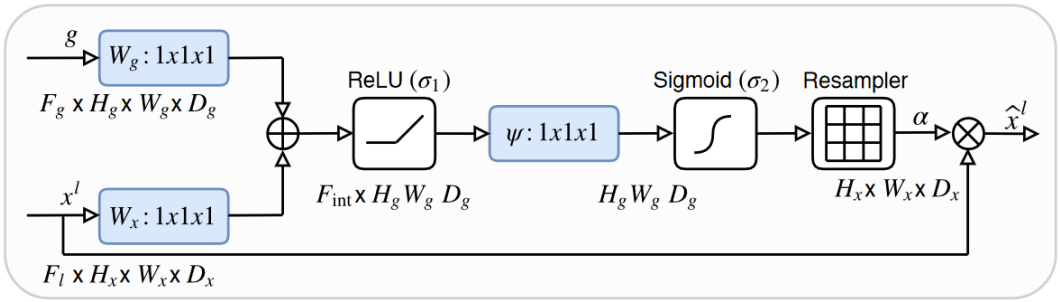


Fig. 46: Attention block[37]

The input g entering the attention block is the gating signal from current layer in decoder and x is the skip connection from the corresponding layer from encoder (one above). In other words, gating signal comes deeper from the network and skip connection higher from the network, that is why the gating signal has better feature representation and the skip connection better spatial representation. Taking both features into account at once might result in more promising results. These input signals are convolved with filters 1×1. Due to dimension difference between the inputs, the input x is convolved with stride equal to 2 to equalize the dimensions of both inputs (gating signal should be two times smaller). Those two signals are then summed, meaning the weights of activations that are alligned get larger and unalligned weights get smaller during this process. This signal passes through ReLU activation function. The following ψ operation is 1x1 operation as well, this time

with only one filter (so far the dimensinos were (height)×(width)×(number of filters)), which will create an activation map with depth equal to one. Since now the values after summing and ReLU function might vary a lot, to get this signal under control, it is processed by sigmoid function to get the values into reasonable interval between 0 and 1. The resampler then upsamples this feature map to the original dimension of x so it is compatible with the following layer and operations of U-Net, named α. The final signal is then given by element-wise multiplication of the input signal x and signal α.[37]

Another significant architecture in the segmentation using deep learning domain is *DeepLab*.

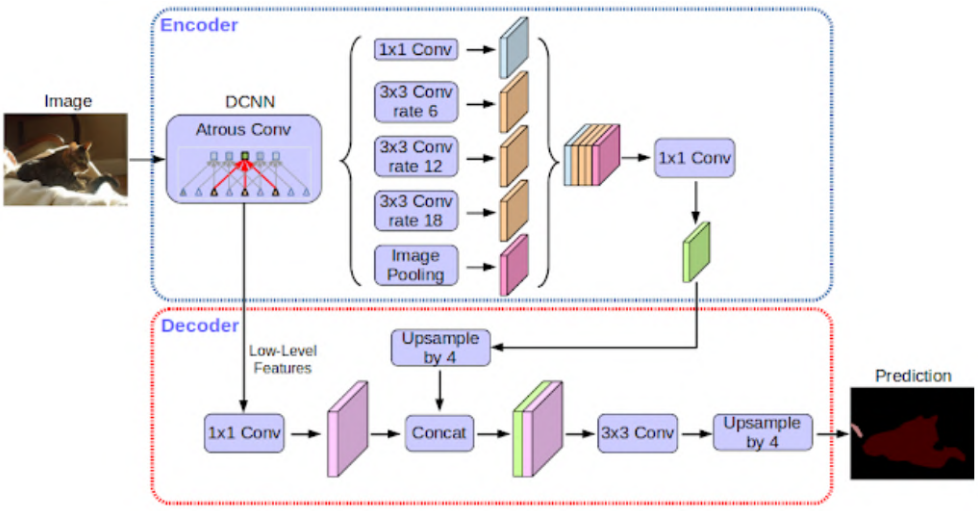


Fig. 47: DeepLabV3+ architecture [13]

DeepLab introduced three main features that proved to improve performance of neural networks. First is *atrous convolution* (dilated convolutions, already described in section 4.5), which is a way of down sampling the flow in neural network without using pooling layers. It increases the receptive field while reducing the spatial size of activation maps for the next layer. It allows the network to use larger context while keeping the number of parameters of given neural network same. The second feature is *atrous spatial pyramid pooling* (ASPP), which allows the net to perform more robust segmentation in the sense of ignoring multiple scales of objects in images. ASPP takes the activation map and performs multiple convolution (with multiple sampling rates for the filters), that ensures to capture the objects and context around them at multiple scales. These branches of filters with different size result in different dimensions, that is why their activation maps are interpolated to the original image resolution and joined afterwards. They are joined by taking the maximum response at each position across different scales. The third feature is the improvement of the

localization of object boundaries. It tries to surpress the problem of pooling/down-sampling, which usually results in loss of information by combining the final layer of given neural network with fully connected *conditional random field* (CRF). CRFs were already designed to smooth noisy segmentation maps in the past. In CRFs, a given pixel is influenced by neighboring pixels (this might be seen as a graph problem as well), but in fully connected CRF used in DeepLab, a pixel is connected to all the other pixels. What happens is that in DeepLab the loss function is being minimized along with the relation between pixels, mentioned fully connected CRF. [11] In the practical part of this work, the state-of-the-art DeepLabV3+ was used with advanced mentioned features as well as with new propositions.

4.8 Selection of accompanying techniques for deep learning and semantic segmentation

Some of the specific widely used techniques and techniques used in the practical part of this thesis, that are typical for segmentation models and tasks, or problems resembling the task in the practical part of this thesis, are yet to be described in this chapter.

Data augmentation

Data augmentation is a set of techniques used to increase the amount of data. That is done by slightly changing original images and then adding them to dataset. This set of techniques is used to prevent overfitting when training a neural network (or other machine learning models) and is especially useful, when only a small amount of data is available. Since this thesis is dealing with images as data, only augmentation regarding images will be presented.

The number of augmentations that might be performed on images is enormous and only a brief enumeration will be done. *Arithmetic* operations on pixels might be done to augment an image, such as directly changing the pixel intensity values by addition or multiplying, applying salt and pepper noise, pixels dropout, compression or cutting pieces of images out by filling one or more rectangular areas in an image by specified color. Selection of this type of augmentations is shown in the Fig. 48.

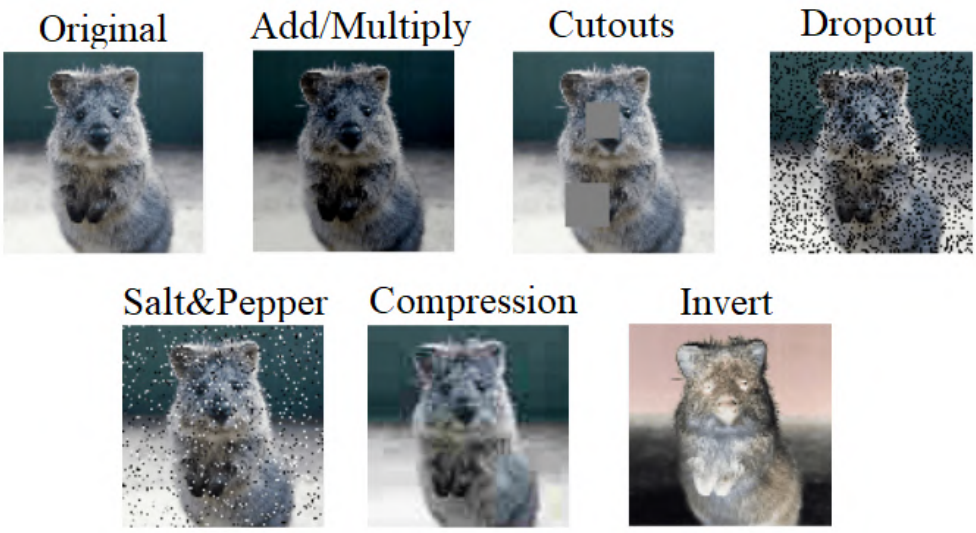


Fig. 48: Examples of arithmetic augmentations [21]

A certain type of *blur* augmentation might be applied on images as well, using various filters from gaussian, average, to median, or imitating a motion blur. That may be seen in the Fig. 49.

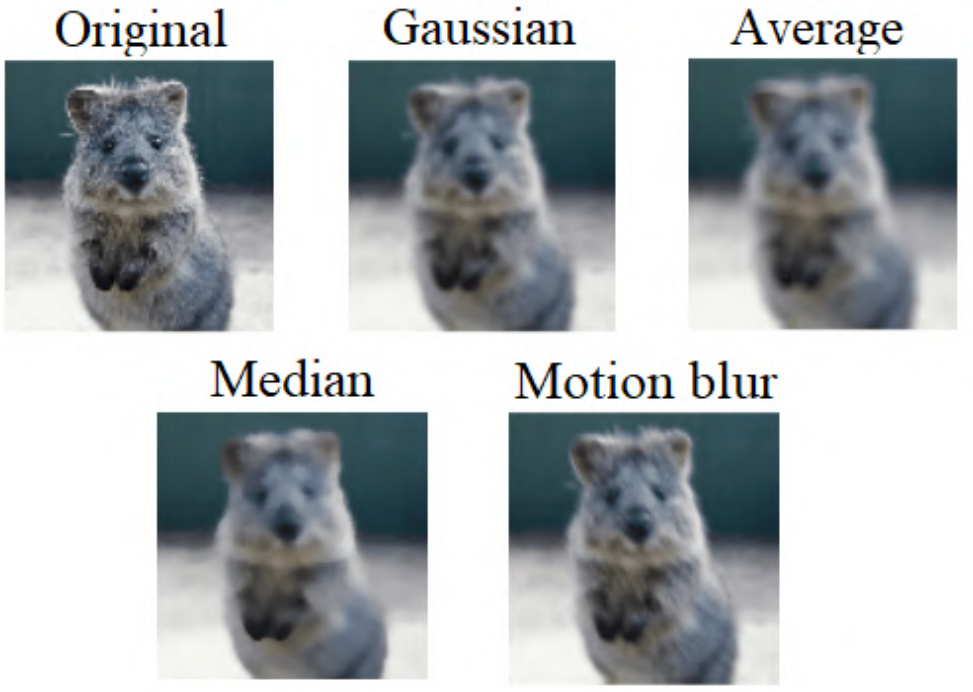


Fig. 49: Examples of blur augmentations [21]

Various *color* augmentation in color spaces can be applied to change the original data as well, using parameters from HSV, RGB color spaces for example, or adjusting image *contrast* by different functions. These types of augmentations are displayed in the Fig. 50.

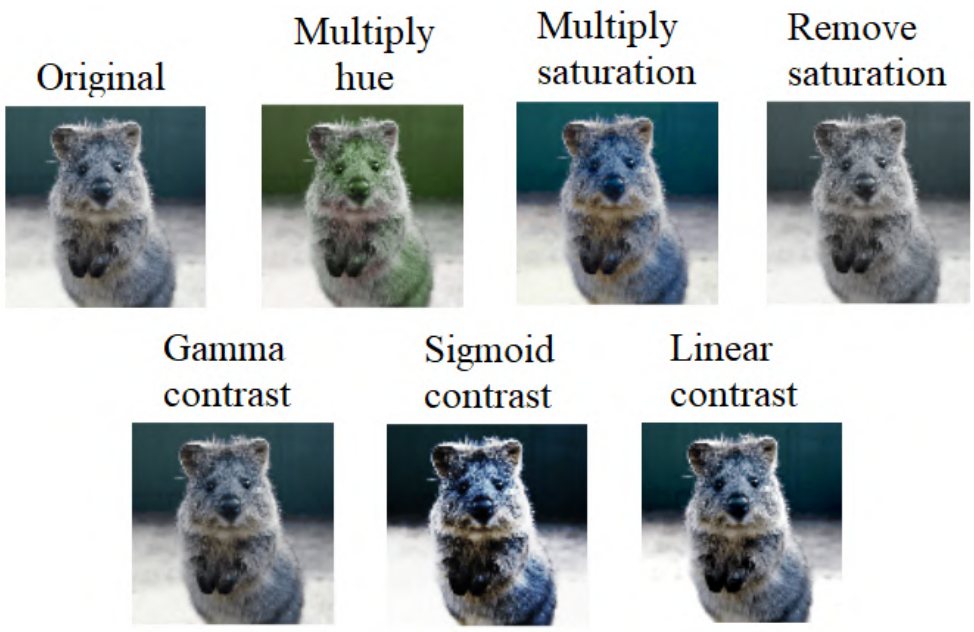


Fig. 50: Examples of color, contrast augmentations [21]

Some of the useful augmentations from the category of *geometric* augmentations might be simple image rotation, horizontal and vertical flip and deforming the original images in various ways. For the reasons that some of the following augmentations were used in the practical part of thies thesis as well, an image and its corresponding segmentation map (ground truth) are transformed at once. Augmented images with keypoints and localized objects are shown along with the segmentation mask in the Fig. 51.

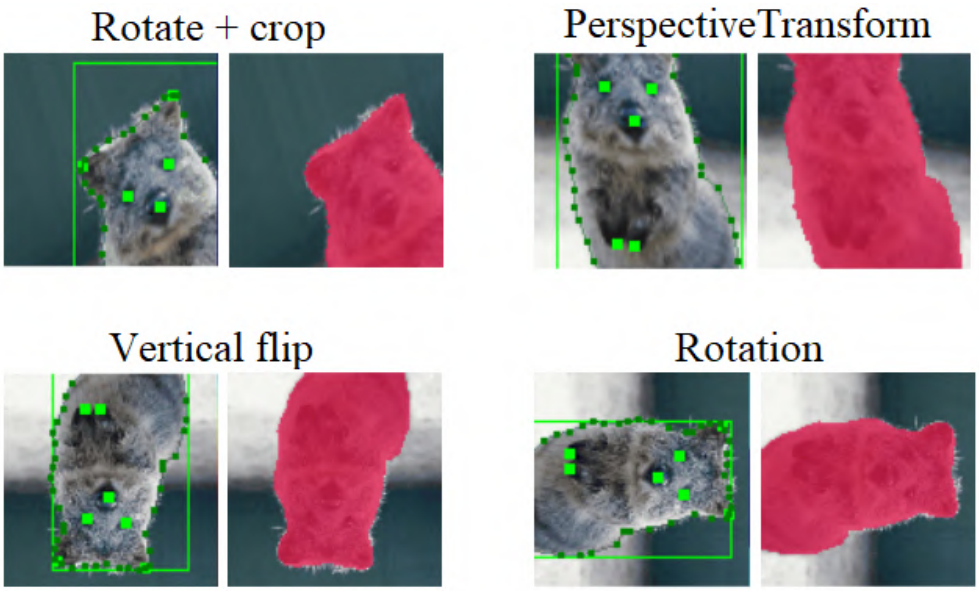


Fig. 51: Examples of geometric augmentations [21]

Another category of augmentations is *size* augmentations. For example techniques like cropping and padding may be used.

Fig. 52: Examples of size augmentations [21]

There are many more augmentations like advanced image enhancing, filtering, finding edges, image pooling, performing segmentations like superpixels and so on. These augmentations can be naturally combined creating countless variations of augmentations that may be applied on image dataset.

When choosing augmentations to be performed, it is important to consider the data and the task. In this thesis, the data were greyscale images, so no color augmentations may be done. Also, the dataset consists of images and their ground truths (segmentation maps), meaning that the augmentations must be applied also on the ground truths (if they should be applied, there are cases where only the image must be augmented). Considering the task, the choice of augmenting technique is crucial, since in the case of this thesis, it is being distinguished between classes, where their detailed structure is the classifying criteria, so any augmentations damaging these details might influence the training in a negative way. For more information about the choice of augmentations see subsection 5.1.3.

Loss functions and metrics for semantic segmentation

Choice of loss functions and metrics used in neural network training and evaluation strongly depends on the type of task that the neural network is performing or being trained for. In this subsection only problem specific metrics and loss functions are going to be described. One of the most used loss functions for semantic segmentation is called *Categorical crossentropy*. This loss function is used for multi class classification as well, where an example can belong to only one class out of many possibilities. In this case, when it is being used for segmentation, each pixel is basically being categorized into one of the defined classes. Categorical crossentropy CC may be described as

$$CC = -\sum_{i=1}^{N} y_i \log \hat{y}_i, \tag{35}$$

where N is the output size, y_i is *i-th* target value and $\hat{y}_i$ is corresponding value of the model output. This loss function basically says how distinguishable two discrete probability distributions are from each other.[38]

Another loss function that might be used on semantic segmentation task is *Dice loss* (DL). Before describing equation for Dice loss, its parameters precision and recall must be defined. Precision p may be expressed as

$$p = \frac{|TP|}{|TP| + |FP|}, \tag{36}$$

where TP means true positive (data samples that were correctly identified as a positive class) and FP means false positive (data samples that were incorrectly identified as a positive class). The placement of both variables into brackets as $|TP|, |FP|$ indicates the number of true positive and false positive respectively. The recall r may be defined as

$$r = \frac{|TP|}{|TP| + |FN|}, \tag{37}$$

where TP means again true positive and FN means false negative (data samples incorrectly identified as a negative classs). Finally, the Dice loss equation may be expressed as

$$DL(p, r) = 1 - (1 + \phi^2)\frac{p \cdot r}{\phi^2 \cdot p + r}, \tag{38}$$

where p, r are precision (36) and recall (37) respectively and ϕ is a coefficient for precision and recall balance.

Dice loss is basically function of intersection and union over foreground pixels. This idea results in the loss giving low error when the network focuses on maximising intersection area over foreground and when the union over background is being minimised. For example, when the model predicts all the pixels as background, the intersection would be 0 and hence the loss would reach a value 1 (maximum value, dice loss is between 0-1). If the model predicts all the pixels as the foreground, the union will become so big the loss would enclose to the value equal to one again. This mechanism is explained mainly because it surpress the problem of unbalanced datasets, which are very common in semantic segmentation tasks.

Another interesting and considerable loss function suitable for semantic segmentation uses *tversky* coefficient and is called Tversky loss (TL). It is basically generalization of Dice loss adding parameters α and β to adjust weight of penalization between false negatives and false positives. Emphasis on penalizing false negatives is achieved by setting $\alpha > \beta$ and emphasis on penalizing false positives is achieved by the opposite in the Tversky loss equation [49]

$$TL = 1 - \frac{|TP|}{|TP| + \alpha|FN| + \beta|FP|}, \tag{39}$$

where $\alpha + \beta = 1$. Coincidentally, when $\alpha = \beta = 0.5$, the tversky coefficient (loss) is equal to dice coefficient (loss). The tversky loss itself is just a minor improvement useful in fine tuning of the training process. But it's improved variant *Focal-Tversky loss* (FTL) provides another functionality. It adds a non-linearity to Tversky loss that changes the behaviour of the loss at different values of tversky coefficient. The Focal-Tversky loss may be computed as

$$FTL = \left(1 - \frac{|TP|}{|TP| + \alpha|FN| + \beta|FP|}\right)^{\gamma}, \tag{40}$$

where γ may have values lower and higher than one, above zero.

Another concept to be explained is the metric(s) that may be used to evaluate the networks on testing and validation data when the semantic segmentation task is being performed. Using the typical *accuracy* metric is no good in this case, since the accuracy of majority of segmented pixels is very easily achievable by the network (for example by labeling all pixels of as the background in most cases and even in the case of this thesis' dataset the accuracy could reach more than 90 percent). That is why *Mean IOU* (intersection over union), also known as Jaccard index, and *F-score* may be considered as a valid mean of networks' training evaluation. Mean IOU returns a number between 0 and 1 (1 indicating the best result). This number represents the amount of overlapping pixels between the ground truth and predicted result by the neural network. On the segmentation task, IOU is computed for each class and their mean is computed and returned as the metrics. Idea of IOU (41) is displayed in the Fig. 53.[12]

$$IOU(A, B) = \frac{A \cap B}{A \cup B} \tag{41}$$

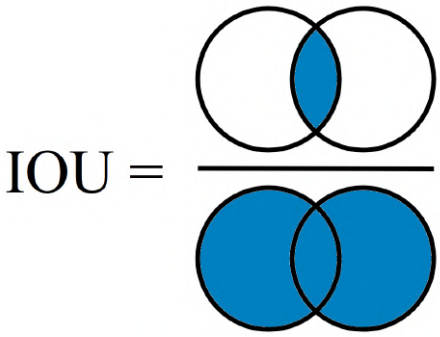


Fig. 53: Intersection over union

F-score may be seen as a weighted average of the precision (36) and recall (37). Again, the best result that can be given by this metric is equal to 1, the worst to 0. The relative contribution of precision and recall to this metric are equal by default,

but more weight may be put on either of those by changing the ϕ parameter. The equation of described metric as the function of precision and recall [52, 50] is

$$F_\phi(p, r) = (1 + \phi^2)\frac{p \cdot r}{\phi^2 \cdot p + r}, \tag{42}$$

and that may be denoted as the function of the two sets being compared A and B as well

$$F_\phi(A, B) = \frac{(1 + \phi^2)TP}{(1 + \phi^2)TP + \phi^2 FN + FP}. \tag{43}$$

Not coincidentally the shape of the equation of the F-score metric looks very similarly to the Dice loss. This metric may be called Dice coefficient as well and is substracted from 1 to form the Dice loss (38).

5 SEGMENTATION OF CARDIAC MUSCLE IMAGES

The goal of this thesis is to design an algorithm that will automatically identify and localize cardiac muscle cells of interest. Specifically, the goal is to find areas in the microscopy images, that will contain all the cardiomyocytes of interest. Due to the complexity of this problem, deep learning will be used as the core mean for the segmentation task. The problem is then solved by firstly implementing and training several deep convolutional neural network architectures suitable for image segmentation and then, based on their performance, selecting the most suitable architectures for this task. In this case, SegNet and Xception U-Net with residual blocks seem to be having the highest performance. The chosen trained segmentation models are then used in the created software.

This software's specifications were to be able to train segmentation models on labeled data and to process unlabeled microscopy images. That is exactly what was accomplished. It is implemented in Jupyter notebooks. It takes original microscopy images as inputs, cuts the inputs to the size that is compatible with neural network models and that is computer-processable. Then, each cut fragment of the original image is segmented and all these fragments and their corresponding segmentation maps are stitched back together. The segmentation map is then transformed into a binary image with 1s, where the objects of interest were found and 0s, where the background and remaining objects are. Morphological operations are made on this binary image to make found objects more robust and then connected components are found along with coordinates of their ares of interest. After the discussion with the researcher, the decision was made to represent the output areas of interest as bounding boxes around interesting objects in the image with a small margin.

To be able to perform the steps described in the previous paragraph, the dataset must have been created from scratch in the first place. Suitable tools for data annotation and augmentation had to be chosen and applied. For data annontation the tool *labelme* was selected. Using this tool, pixel-wise annotation was performed on the original images resulting in segmentation maps (ground truths). The images and their corresponding segmentation maps were cut into smaller tiles afterwards, since the images are too large to be processed by chosen neural network architectures with given hardware limitations. Then, using *imgaug* python library, the cut images with corresponding segmentation maps were rotated by 90, 180 and 270 degrees and added to the original images and segmentation maps. The final step was division of data into train, test and valiation subsets. Any possible additional augmentation of images and their segmentation maps were done on-the-fly during training process using imgaug library as well.

As was mentioned, the chosen neural network architectures were trained with some hardware limitations. More concretely, a computing workstation based at the Institute of Automation and Computer Science, Brno University of Technology was used. This workstation had the following hardware configuration:

- CPU: AMD FX-8350, 4GHz
- RAM: 32 GB, DDR3
- GPU: NVIDIA GeForce GTX TITAN X, 12GB

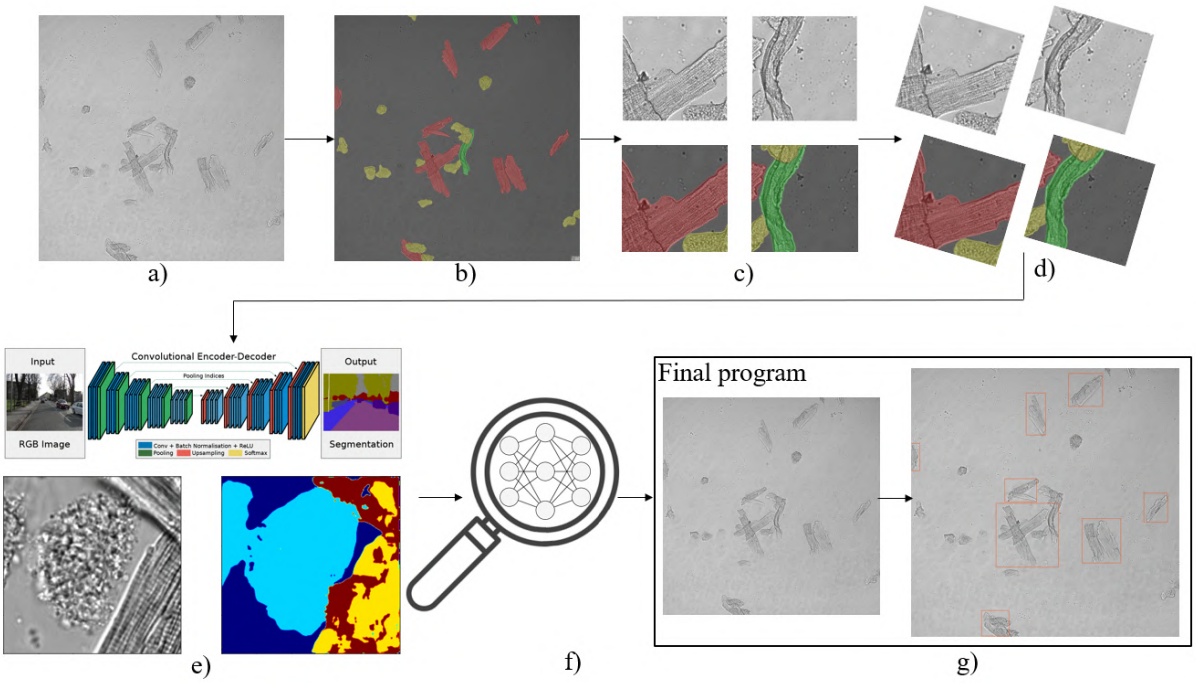


Fig. 54: Thesis workflow

The whole workflow is ilustrated in the Fig. 54, where on subfigure a) the example of original image is displayed. The process continues with b), where the depiction of labeling process may be found, on c) and d) the cutting and augmentation are shown respectively. In subfigure e) the implementation and training of deep convolutional neural networks suitable for segmentation is shown, that is followed by f), where the evaluation of models and selection of the best performing one is ilustrated. The whole process ends with the final program, which takes microscopy images as input and returns the areas of interest in the image represented by bounding boxes around interesting types of cardiac muscle cells.

5.1 Dataset creation

Due to the specific type of images, no similar data may be found, which means no pre-trained segmentation models may be found as well for the purpose of transfer

learning. That is why the models must be trained from scratch and new dataset must be created. Thirty three images (4096×4096px) were obtained from the Slovak Academy of Sciences from Ing. Alexandra Zahradníková, DrSc. This directly affects the choice of neural network architectures, training approach and the final algorithm for segmenting these images.

5.1.1 Images presentation

As mentioned above, thirty three images of 4096×4096 dimensions are available. These images were obtained by using fluorescence (confocal) microscopy. Cardiomyocytes, cardiac muscle cells, are on the samples from which the images were taken. When viewed under the microscope, it is visible that cardiac muscle cells are more or less rectangular. In the heart, these cells are joined together to form long strands. On the dataset to be processed, however, the cells are cut to individual fragments of these strands, or to a group of fragments. A demonstration of such cells from the dataset is shown in the Fig. 55.

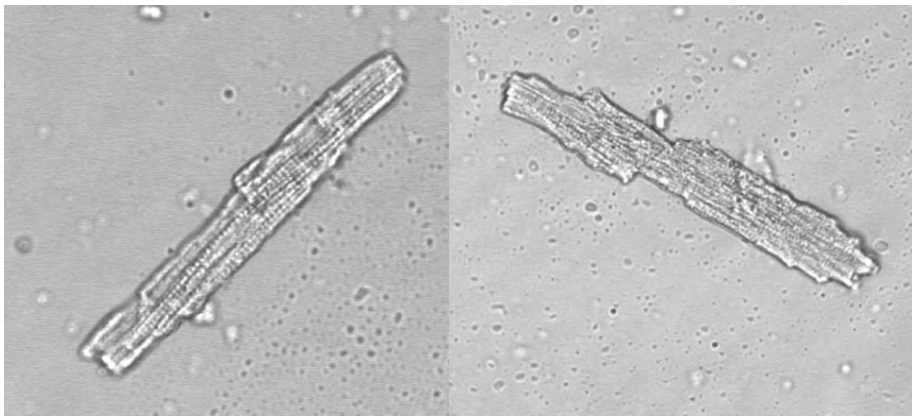


Fig. 55: Cardiomyocyte samples from dataset

On the obtained images there are not only cardiac muscle cells in a form as is displayed on the Fig. 55. Some of the cardiomyocytes are captured collapsed due to the preparation and preserving the samples (while preparing and storing samples, the enzymes are used to "poison" the intercellular areas for cutting the cardiomyocytes strands to smaller pieces. The quality of the final sample is directly dependent on the amount and the type of an enzyme used). And some are right between those two states, one could say half collapsed. These three types of cells make three classes that will be distinguished and segmented later on. To rephrase previous statement, the first class to be segmented represents "healthy" cardiomyocytes showing roughly rectangular shape with clear structure and may be seen in the Fig. 55. The second class represents collapsed cardiomyocytes without any structure and may be seen in the Fig. 56.

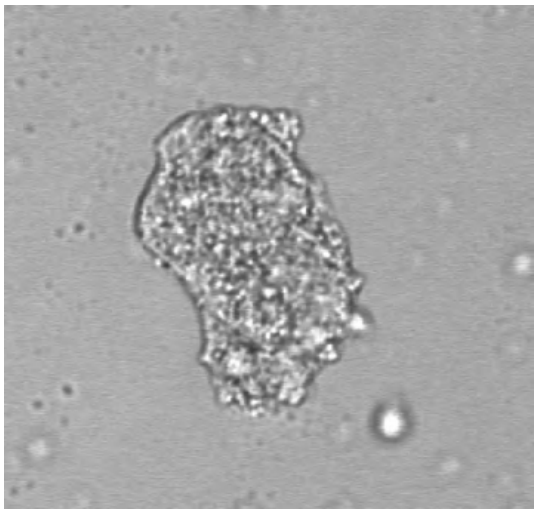


Fig. 56: Example of collabsed cardiomyocyte

The third class contains cells that are not yet crumbled, but it also cannot be said they posses properties to be in the first class. Into this class are also placed cells that appear to be blurry or illegible. Example of this class is shown in the Fig. 57.

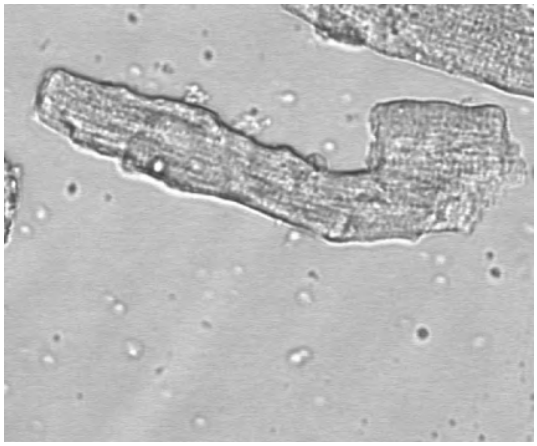


Fig. 57: Example of half-collabsed cardiomyocyte

It is worth to mention that classes of preserved cells with structure and half-collapesd cells in some cases seem to be very close to each other. The difference between them is subtle and that might cause a problem when labeling the data. Potential mislabels might have a significant impact on a neural network training quality.

5.1.2 Data annotation

As mentioned before, the dataset needs to be made from the basis. With respect to the task (semantic segmentation), the ground truth must be created for images. That practically means that another corresponding image with same dimensions as

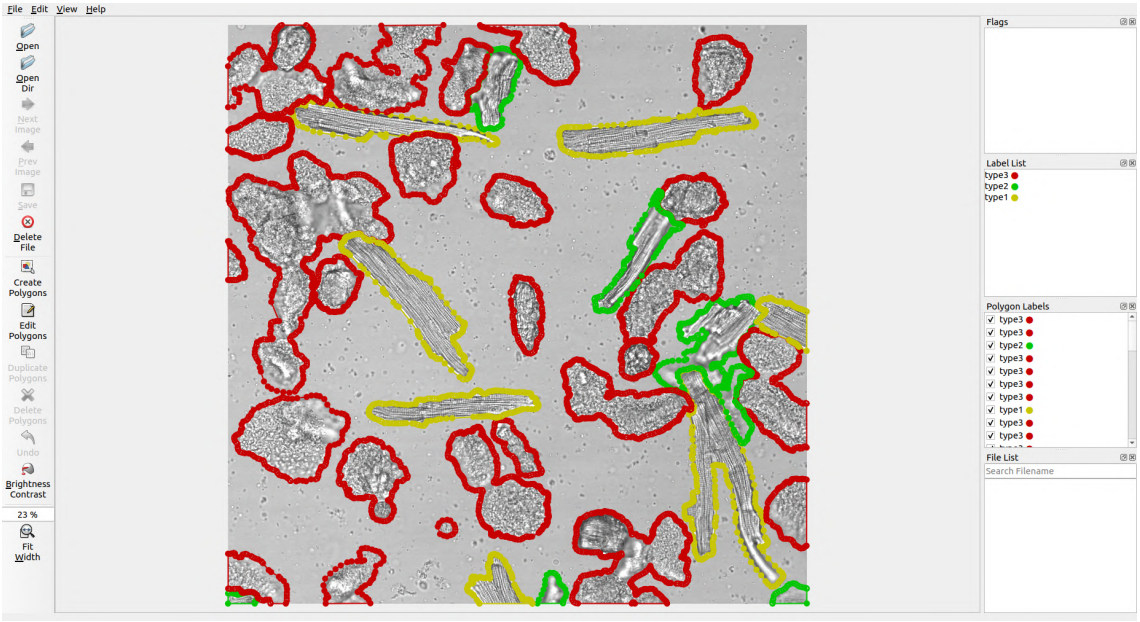


Fig. 58: labelme GUI and annotation example

the input image must be created with labeled objects, that are to be segmented and distinguished from the background.

For such task a tool called *labelme* was chosen. Labelme is a graphical image annotation tool with which it is possible to create a pixel-wise labels for input images. It was chosen because it was written in python (and as everything else was implemented in python in this project, unity was a helpful factor when choosing it), because it is fairly easy to use and install (only Python itself and PyQt are required) and because it fullfils its purpose for this annotation task.[22]

Labelme has multiple annotation options and techniques, but what was interesting for obtained images was polygonial annotation. With this technique, once polygon of a certain object being labeled is enclosed, all the inner pixels are labeled with corresponding class as the outline. An example of image annotation with user interface of this tool is shown in the Fig. 58. The approach for labeling was to zoom in the working environment as much as possible and to follow the outline of given object as precise as possible. In the Fig. 58 all three types of cardiomyocytes are visible, which is clear from *Label List* on the right and on the displayed picture as well.

Once the annotation process is finished, it is possible to use labelme to generate a .json file with saved coordinates of labeled objects (all of its pixels). Then the tool may be used to generate a label file, which is a file containing names of labels/classes and visualisation file. Label file at this point is shown on the Fig. 59.

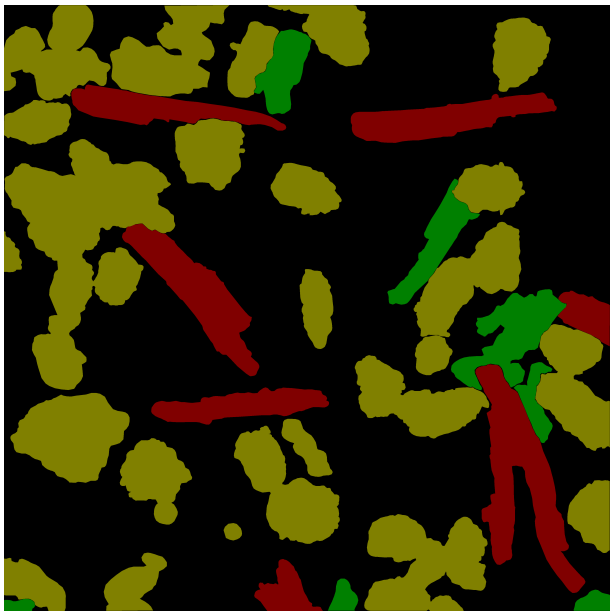


Fig. 59: Label file corresponding to Fig. 58

Note: colors of classes in the labelme tool and on label files do not correspond, however when creating a dataset, the tool generates reliably all labels into each group correctly.
Selection of few visualisations (original images overlayed by label file) are shown in the Fig. 60 and 61.

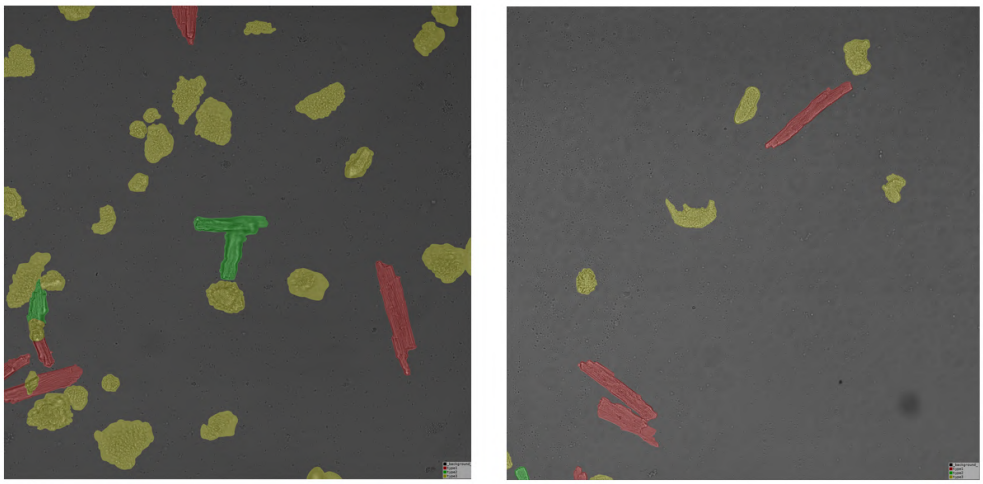


Fig. 60: Examples of labeled images 1

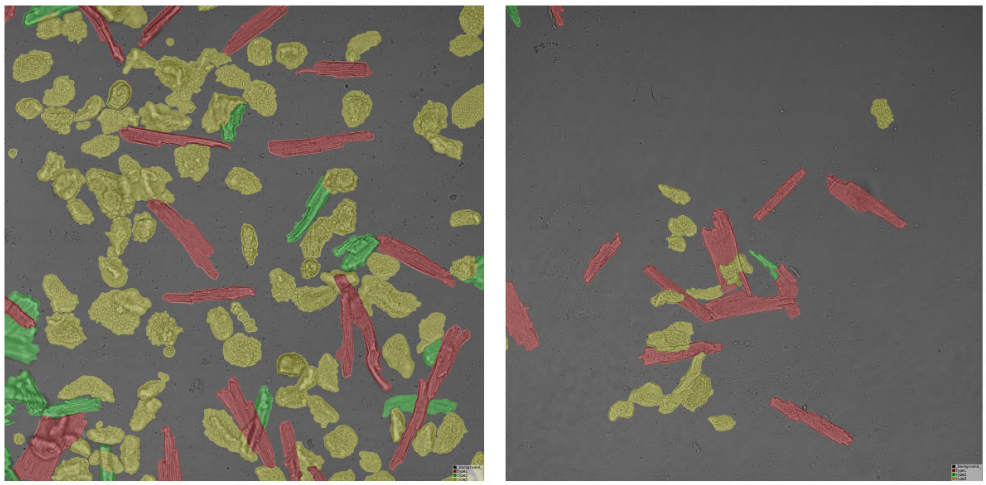


Fig. 61: Examples of labeled images 2

It is clear that yellow colour represents a class constisting of collapsed cardiomyocytes, green color the half-collapsed class and the red color the class consisting of preserved cardiomyocytes.

The label files (or ground truth, segmentation maps) are currently in 4096×4096×3 dimensions, displaying colors. These segmentation maps will figure as an input to the neural network models and in fact, these dimensions are not compatible with input layers of designed segmentation models. So in reality, when the images were cut to smaller tiles, label files were also changed to dimensions 4096×4096×1 (512×512×1 after cutting), having only values 0-3 (0 for background, 1-3 for corresponding classes).

5.1.3 Images augmentation

Cutting

Since the original images have 4096×4096 dimensions, it is not possible to process them through a neural network due to hardware limitations. Both RAM and GPU that were used as the means of computing would not be sufficient.

Image compression does not seem like a proper approach, since significant amount of information would have been lost. Especially in this case, when it is being distinguished between cardiac muscle cells that have visible clear structure and those who do not. That is why another approach was selected: cutting images into smaller tiles.

The next question to be answered is the size of tiles that the original image is going to be cut into. The idea was to leave the tiles be as large as possible for the neural network to register the context and surroundings of the cells properly and to have the least number of slices as possible. However the hardware limitations must be considered, that is why size 512×512px was chosen for the tiles. The original

4096×4096px images were then divided into 8×8 parts, one part having exactly 512 by 512 pixels. Then another cutting took place, the original image was divided into the tiles of the same size as before, but 256 pixels from each side were ommited and the rest of the image was divided into 7×7 parts. This, maybe confusing at first, description of cutting original images is displayed on the Fig. 62.

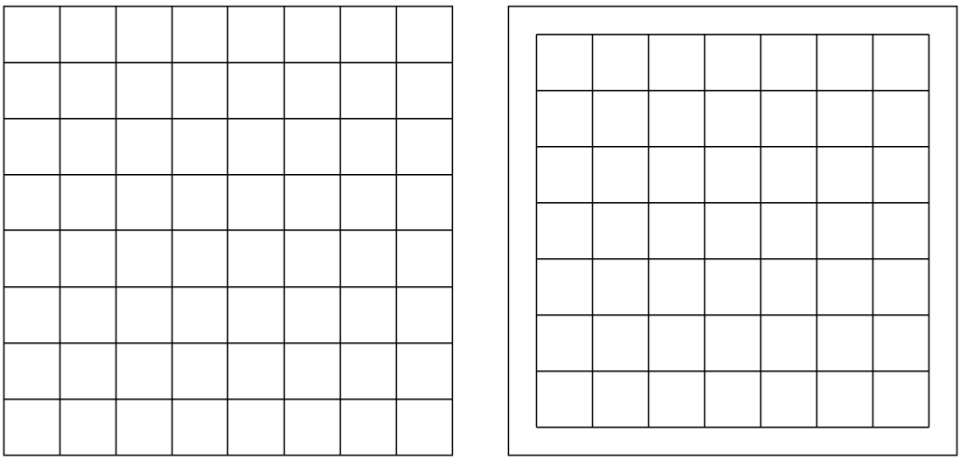


Fig. 62: 8×8 and 7×7 cuts of original images

The second 7×7 cutting was done to compensate for the loss of local information by the cuts. It is impossible to implement the image partition without splitting the cardiac muscle cells. The side effect of additional cutting was also generation of more data. That is desirable and since image shifting for example is a standard augmentation technique for creating additional data, it is a welcome side effect. An illustration of two types of cuts overlaying is shown in the Fig. 63.

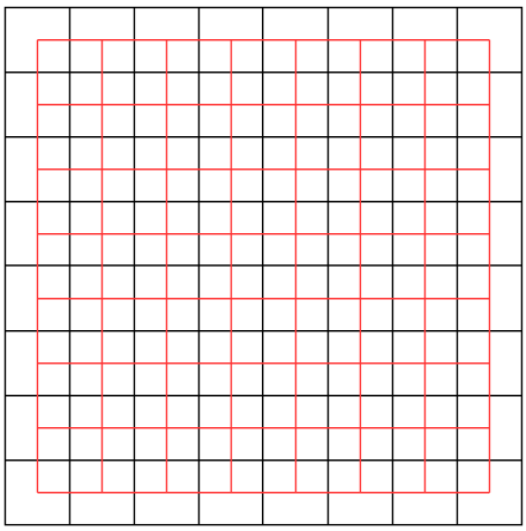


Fig. 63: Cuts overlay

Cutting 33 original images into 7×7 (red squares) and 8×8 (black squares) pieces as described above results in 3279 512×512 data samples.

Additional augmentations

As already mentioned, one of the challenges of the practical part of this thesis is the small amount of data. That is why the idea of using augmentation, apart from the mentioned cutting, is considered. Data augmentation is a set of techniques that are used to expand dataset by taking original images, slightly changing them and adding these altered versions of originals to the dataset. This is described in more detail in section 4.8.

From various options a libray called *imgaug* was chosen for this problem. Imgaug is a python library that provides plenty of augmentation techniques. It was chosen for the fact that it is written in python again for unity, it is well documented and updated and last, but not least, it supports simultaneous augmentation of both images and segmentation masks (and automatically ignores alterations on masks where they are not applicable).[21]

The amount of augmentations that may be done with images is basically countless. For detailed description of augmentation techniques for images see 4.8. All the operations with color channels (intensifying colors, random pixel channels dropout, alterating hue, saturation, brightness, contrast,...) are negligible, since the original images are greyscale. Also details wrenching, image pooling or image twisting and changing in significant manner is not used, since it is desirable for the network to be able to learn to identify detailed structures of the cardiac muscle cells. However, geometric augmentations, like rotation, cropping, image flipping are used in the actual augmentation of the dataset, along with slight blurring as well.

An augmentation session was predefined as follows. On each image in random order:

- Either *Crop and pad* or slight *Gaussian blur* is applied with a chance of 60%
- A rotation from -180 to 180 degrees with a chance of 80%
- Flip the image in the horizontal manner with a chance of 25%
- Flip the image in the vertical manner with a chance of 25%

A few examples of such augmentation pipeline applied on the cut data is shown in the Fig. 64.

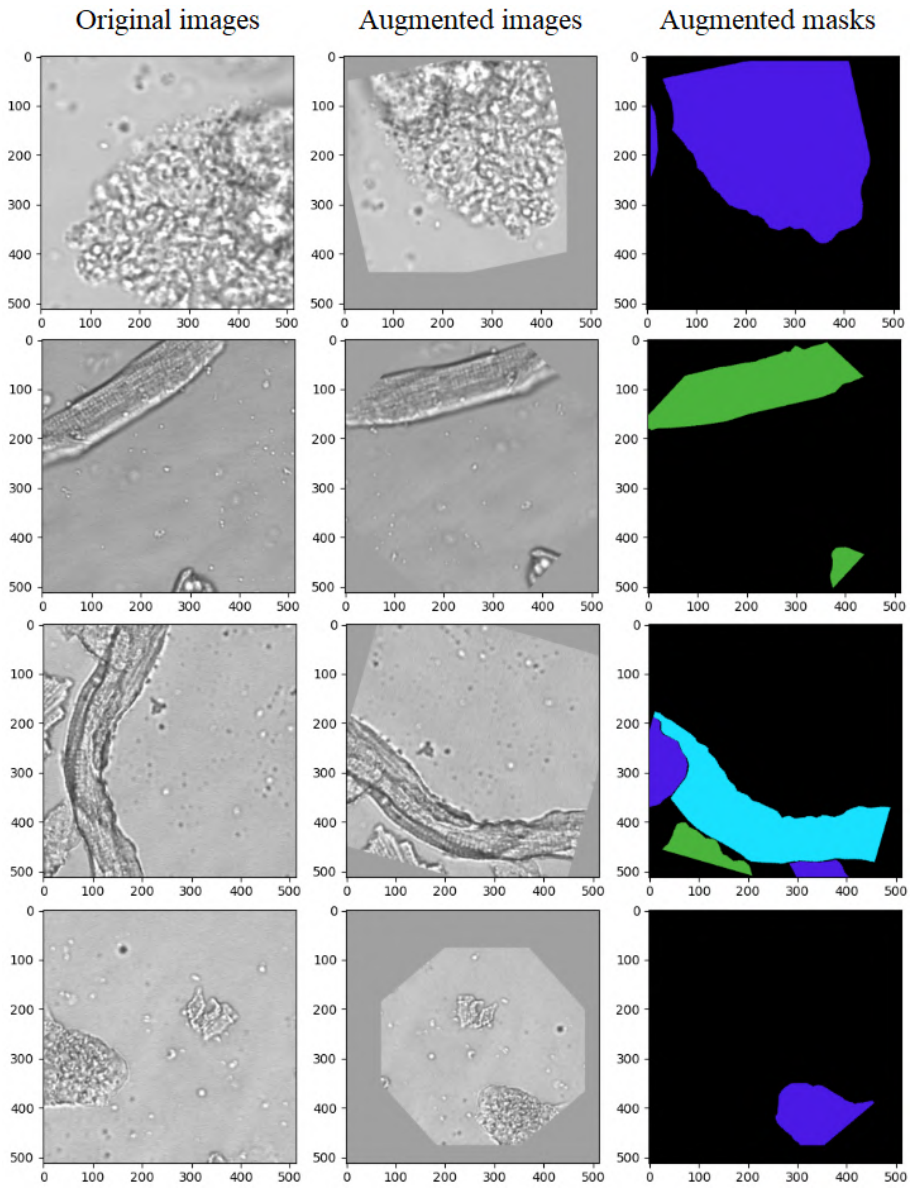


Fig. 64: Augmentation examples from defined augmentation pipeline

The only augmenting techniques used on the dataset before training were the initial cutting and rotating by 90,180 and 270 degrees for the purpose of creating a second, larger dataset. The reason for this is some of the hypothesis about augmentation improving the training quality were negated (as discussed later on). The segmentation models were train on both larger dataset without further augmentations and the original, only cut, dataset, on which the rest of the augmentation techniques such as the small rotations, image flipping or cropping were done to each image (or batches of images) on the fly during the training process. That means that once the image from dataset was chosen, it was augmented before acting as an input to the neural network along with corresponding segmentation map. This makes the training process slower, but since a custom image generator was needed to

be done anyway (more on that in the section 5.2), it seemed like a great opportunity to create varying dataset, preventing overfitting, solving a low number of data issue and thus improving the neural network training.

5.2 Implementation and analysis of neural networks and their training

Before describing how the neural networks were implemented, the description of the tool used for deep learning will be provided first. *Keras* is a well known application programming interface written in Python, that is used for deep learning. It is well maintained and documented with large user base. Since 2019, Keras was officially built on top of the TensorFlow 2, which is an open source platform for machine learning. It is relatively easy to use and Keras documentation provides examples for the typical use cases. Since Keras' main specialization is deep learning, all kinds of neural networks layers, optimizers, loss functions, metrics, or training/evaluating algorithms are already provided by this framework.

The typical shape and properties of segmentation architectures is described in a thorough detail in section 4.7, so it will not be mentioned again and a concrete implementation example is going to be shown. SegNet, the architecture that is implemented in Keras on the following example is described in section 4.7 as well. The implementation of SegNet achitecture was chosen because of it's relative simplicity and a good performance on the segmentation task of this thesis. Detailed description of implementation of SegNet architecture will be available after the code example. It is a smaller version of SegNet with 4 encoding/decoding layers consisting of 2 convolutional layers each. This function returns the actual model that was used for the task of this thesis. [5]

```
def Segnet(nClasses=4, input_height=512, input_width=512):
    inputs = Input(shape=(input_height, input_width, 1))

    #Encoder
    conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
    conv1 = BatchNormalization()(conv1)
    conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv1)
    conv1 = BatchNormalization()(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool1)
    conv2 = BatchNormalization()(conv2)
    conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv2)
```

```
conv2 = BatchNormalization()(conv2)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(pool2)
conv3 = BatchNormalization()(conv3)
conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv3)
conv3 = BatchNormalization()(conv3)
pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

conv4 = Conv2D(512, (3, 3), activation='relu', padding='same')(pool3)
conv4 = BatchNormalization()(conv4)
conv4 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv4)
conv4 = BatchNormalization()(conv4)
pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

# Decoder
up7 = UpSampling2D(size=(2, 2))(pool4)
conv7 = Conv2D(512, (3, 3), activation='relu', padding='same')(up7)
conv7 = BatchNormalization()(conv7)
conv7 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv7)
conv7 = BatchNormalization()(conv7)

up8 = UpSampling2D(size=(2, 2))(conv7)
conv8 = Conv2D(256, (3, 3), activation='relu', padding='same')(up8)
conv8 = BatchNormalization()(conv8)
conv8 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv8)
conv8 = BatchNormalization()(conv8)

up9 = UpSampling2D(size=(2, 2))(conv8)
conv9 = Conv2D(128, (3, 3), activation='relu', padding='same')(up9)
conv9 = BatchNormalization()(conv9)
conv9 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv9)
conv9 = BatchNormalization()(conv9)

up10 = UpSampling2D(size=(2, 2))(conv9)
conv10 = Conv2D(64, (3, 3), activation='relu', padding='same')(up10)
conv10 = BatchNormalization()(conv10)
conv10 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv10)
conv10 = BatchNormalization()(conv10)
```

```
    outputs = Conv2D(nClasses, (1, 1), padding='same',\\
              activation='softmax')(conv10)

    model = Model(inputs, outputs)

    return model
```

As can be seen on the enclosed code, all convolutional layers use 3x3 convolutional filters, and ReLU as activation function. All pooling or upsampling layers use filter of size 2x2. The last part of the network, classification layer, uses softmax. The number of filters on each encoder/decoder layers varies from the input as: 64 - 128 - 256 - 512 - 512 - 256 - 128 - 64, which certainly resembles the typical U (or V) shape of segmentation neural network architectures.

As was already mentioned in the previous chapter, the processed images are quite large (4096×4096px) and when the dataset was created, as was described in section 5.1, it was found out it is not possible to simply create keras dataset and feed it to the training process, since the amount of data is too spacious for pre-allocating that much space in RAM of the workstation used. After a short research *tf.keras.preprocessing.image.ImageDataGenerator* class was found. This class provides implementation of python generator for feeding data into neural network in the training process by batches of images. Specifically method *flow_from_directory()* seemed to be providing the solution to the problem that had risen. Unfortunately, as was found out, this method did not handle the data for *multiclass* semantic segmentation. It seemed to be handeling data for most of the known use cases such as classification, or even binary segmentation. This led to implementation of custom image data generator. That probably results in a solution only for this specific use case and most likely is not as robust and fast as keras built-in implementations. However, it did not only made the training process work, it also made the on the fly augmentation possible. Due to the custom implementation it was possible to have the data flow from directories to neural network more under control and changes, like augmentation, were easily made. A demonstration of the custom generator (and data preparation mentioned in previous paragraph) will be shown later on in a compact example.

The following example of training process will include also the custom image generator and will be divided into parts interspersed with explanations. The first significant part of the example is a definition of image generator used by the training process. In the two parts of the code wrapped in hash lines is the implementation of data augmentation, that was already described in more detail in section 5.1.3.

```
def img_generator(img_dir, label_dir, batch_size):
    list_images = os.listdir(img_dir)
```

```
# random.shuffle(list_images)  # Randomize the choice of batches
ids_train_split = range(len(list_images))

############ Definition of augmentation pipeline ############
sometimes7 = lambda aug: iaa.Sometimes(0.7, aug)
sometimes2 = lambda aug: iaa.Sometimes(0.2, aug)

seq = iaa.Sequential([
    iaa.OneOf([
        sometimes2(iaa.CropAndPad(percent=(0, 0.2), pad_mode="constant",\
        pad_cval=160))
    ]),
    sometimes7(iaa.Affine(rotate=(-180, 180), mode='constant', cval=160)),
    iaa.Fliplr(0.4),
    iaa.Flipud(0.4)
], random_order=True)
#############################################################

while True:
    for start in range(0, len(ids_train_split), batch_size):
        x_batch = []
        y_batch = []
        end = min(start + batch_size, len(ids_train_split))
        ids_train_batch = ids_train_split[start:end]
        for id in ids_train_batch:
            img = cv2.imread(os.path.join(img_dir, list_images[id]), 0)
            mask = imageio.imread(os.path.join(label_dir, \
            list_images[id].replace('jpg', 'png')))

            ############ Augmentation applied ############
            segmap = SegmentationMapsOnImage(mask, shape=img.shape)
            images_aug_i, segmaps_aug_i = seq(image=img, \
            segmentation_maps=segmap)
            segmaps_aug_i = segmaps_aug_i.get_arr()
            ##############################################

            x_batch.append(images_aug_i)
            y_batch.append(segmaps_aug_i)
```

```
            # Preparation of data to be used as an input of neural network
            x_batch = np.array(x_batch, np.float32) / 255.
            y_batch = np.array(y_batch, np.float32)

            x_batch = np.expand_dims(x_batch, axis=3)
            x_batch = normalize(x_batch, axis=1)

            y_batch = np.expand_dims(y_batch, axis=3)
            y_batch = to_categorical(y_batch, num_classes=4)

            yield x_batch, y_batch
```

After definition of data generator that yields batches of augmented data prepared to be processed by the network, the training process is implemented. First the implemented architecture of desired neural network is called and compiled.

```
model = get_model()  # could be implementation of SegNet for example
model.compile(optimizer='adam', loss='categorical_crossentropy', \
              metrics=[tf.keras.metrics.MeanIoU(num_classes=4)])
```

As can be seen categorical crossentropy was chosen as a loss function in the training process. This is the case for all the trained neural networks in this thesis. The reason for not using more advanced and maybe more suitable loss functions for this task (loss functions such as focal tversky loss or dice loss), that would surpress the imbalanced dataset problem, is that it is not built in the Keras API. With some parts of the program being already implemented customly and these functions not being in Keras resulted in the fact, that it was not possible to implement these loss functions and make them compatible with the training process as well. As a metric here is shown Mean IoU from Keras API. This metrics was used for all the neural network architectures training as well, but in this case with some parts of the program being customly implemented, the built in metric did not work. That is why in the end a custom metrics Mean IoU and F1-score were created and used in the trainings of all models.

The next step is initialization of data generators using correct folders that will be used for yielding training and testing data during the training process.

```
batchsize = 4
img_traindir = '/path/to/train/images/'
seg_traindir = '/path/to/train/masks/'
train_generator = img_generator(img_traindir, seg_traindir, batchsize)

img_testdir = '/path/to/test/images/'
```

```
seg_testdir = '/path/to/test/masks/'
test_generator = img_generator(img_testdir, seg_testdir, batchsize)
```

Callbacks are defined as well. A callback is an object that can perform actions at various stages of training. Here specifically three callbacks were used. First, *ModelCheckpoint*, used to define a place where the trained weights will be stored and under what condition. In this case, only the weights of the epoch that gave the best result on testing data will be saved. Second callback is *EarlyStopping*. Since a significant amount of models were to be trained and the time and computing resources were not unlimited, the training did not go for a high number of epochs and then from graphs of validation and training loss was not read what was the optimal epoch from which the weights should be used/saved. Instead by using EarlyStopping callback, if the validation loss did not improve for 15 epochs in this case, the training would stop. Not coincidentally after this amount of epochs training loss kept lowering and validation loss started to go up, the models might have started overfitting at this point.

```
callbacks = [
    ModelCheckpoint('/path/to/save/trained/weights/,  \
    verbose=1, save_best_only=True),                  \
    EarlyStopping(patience=15, monitor='val_loss'),
    TensorBoard(log_dir='logs_NAME_of_net_with_parameters.h5')]
```

And last, but not least, is the mention of the *fit()* function that keras provides. Since the data are ready to enter the training process and all required processed were made, the training of the network may begin.

```
numof_train_images = 2796
numof_test_images = 559
model.fit(train_generator,
          verbose=1,
          epochs=500,
          callbacks=callbacks,
          validation_data=test_generator,
          steps_per_epoch=numof_train_images//batchsize,
          validation_steps=numof_test_images//batchsize,
          shuffle=False)
```

During the training of models two phenomena appeared. Some models performed worse with augmentation. Unfortunately without any apparent pattern. The augmentation was made so it would not damage the clarity of the cells or making the structure ambiguous. Only spatial augmentations were made like rotation, crop

and pad, flipping in the end. One could expect a training of especially a deeper model would be improved by data augmentation, however there seems to be no rule for that in this case. The second phenomena that appeared was that more complex models with more advanced techniques and more parameters performed worse than simpler models. One of the possible explanations for the second observation could be that there was not nearly enough data in the training dataset and the complexity and high number of parameters of some models were too large for the parameters to converge close to optimal values. Apart from the fact that there was not enough data, the second explanation may be that for this particular task it might have been better for the neural network to concentrate more on the bigger contexts, meaning whole/significant parts of the cells and not the details.

These observations led to change of the selection of architectures to be trained. First architecture that was trained was *U-Net* and after that more complex models (*Attention U-Net* and *DeepLabv3+*). After seeing that these models performed significantly worse than a basic U-Net architecture, a re-evaluation of the architectures selection was made and more U-Net like models were chosen along SegNet for reasons described above. Another thing that was tried out was lowering the number of filters on certain architectures for already mentioned reasons.

5.3 Selection and comparison of neural network achitectures

In this section more clarification about modifications to selected models is provided, as well as the final comparison of trained models.

Apart from the change of selection due to the observations from the last section (with the augmentation not being an improvement in some cases and simpler models performing better), a variation of trainings of selected models was made. The models were trained on either:

- Default dataset with on-the-fly augmentation
- "Less augmented" dataset (due to low amount of data available, images of the default dataset were just rotated by 90, 180 and 270 degrees and added to default dataset)

Another modification of trained models was lowering the number of filters in chosen models to make them less complex. In other words, in, for example, U-Net the number of filters on each convolutional block goes as: 64 - 128 - 256 - 512 - 1024 - 512 - 256 - 128 - 64 and to make it less complex, the number of filters was lowered to half on each convolutional layer of each convolutional block. So another variation of trained models is:

- Architectures with original number of filters, on initial layer 64
- Architectures with lowered number of filters to half, 32

Apart from documenting the approach of selection and modification models that were trained in this thesis, the previous information is required to understand the Table 1. In this table, all the trained models are evaluated on validation data (the original created dataset was divided into train, test and validation parts). The models performance with respect to validation loss (Categorical crossentropy) and especially with respect to validation metrics (Mean IOU and F1 score) is displayed there. To provide a legend, after the model names, the *(n)* stands for trained model on "not/less augmented" dataset, the *(a)* stands for model trained on augmented dataset, the *(i32)* stands for the initial number of filters being equal to 32, and the *(i64)* stands for the initial number of filters being equal to 64. The models without information about initial number of filters consist of default number of filters for the specific architecture.

Model	Cat. Crossentropy	Mean IOU	F1 score
U-Net (n)(i64)	0.0392	0.4402	0.4818
U-Net (a)(i64)	0.1270	0.3018	0.3439
U-Net (n)(i32)	0.0579	0.3983	0.4417
U-Net (a)(i32)	0.0624	0.3845	0.4298
R2U-Net (n)(i64)	0.1305	0.3095	0.3503
R2U-Net (a)(i64)	0.2372	0.2648	0.2905
R2U-Net (n)(i32)	0.2011	0.2686	0.3112
R2U-Net (a)(i32)	0.2663	0.2376	0.2716
VGG U-Net (n)	0.3349	0.2100	0.2502
VGG U-Net (a)	0.1915	0.2502	0.2759
Attention U-Net (n)(i64)	0.1868	0.2867	0.3225
Attention U-Net (a)(i64)	0.2621	0.2304	0.2570
Attention U-Net (n)(i32)	0.4600	0.2345	0.2426
Attention U-Net (a)(i32)	0.1726	0.2662	0.2979
SegNet (n)	0.1045	0.3058	0.3451
SegNet (a)	0.0604	0.4100	0.4495
DeepLabv3+ (n)	0.6290	0.1659	0.2182
DeepLabv3+ (a)	0.3479	0.2499	0.2677
Xception U-Net + res.(n)	0.0409	0.4144	0.4566
Xception U-Net + res.(a)	0.0561	0.3811	0.4252

Tab. 1: Comparison of neural networks performance

The last thing to notice about the comparison table is the highlighted lines. Models on these lines are chosen to be considered as candidates for the model to be used in the final segmentation algorithm. The selection was made based on the displayed validation performance: the loss function and more importantly the metrics.

The actual segmentation done by the selected models is soon to be presented. Four validation samples were chosen (so that there is not only a backgroung, but objects to segment as well) and four selected models performed a segmentation on these. On the Fig. 65 and Fig. 66, the predictions are displayed. It is split into two images for the images to fit into the paper format in a legible form. The first column displays original validation images, the second column displays the masks (ground truths) corresponding to the original validation images and the third column displays actual predictions done by U-Net(n)(i64) in the Fig. 65. In the Fig. 66, at the first column the ground truths for direct comparison are shown again and then the columns show predictions done by actual models as follows: U-Net(n)(i32), SegNet, Xception U-Net with resudal blocks.

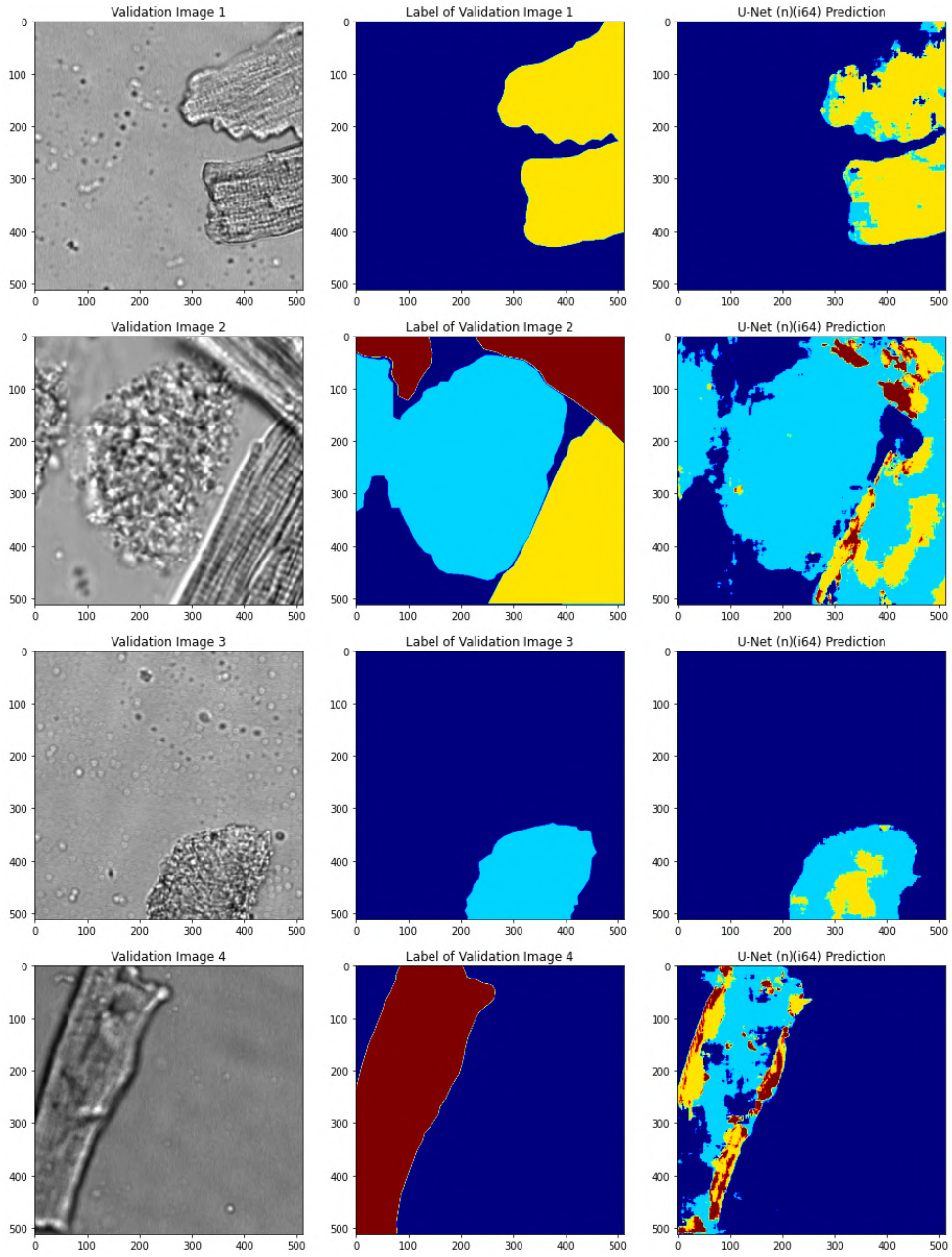


Fig. 65: Comparison of selected models, part 1

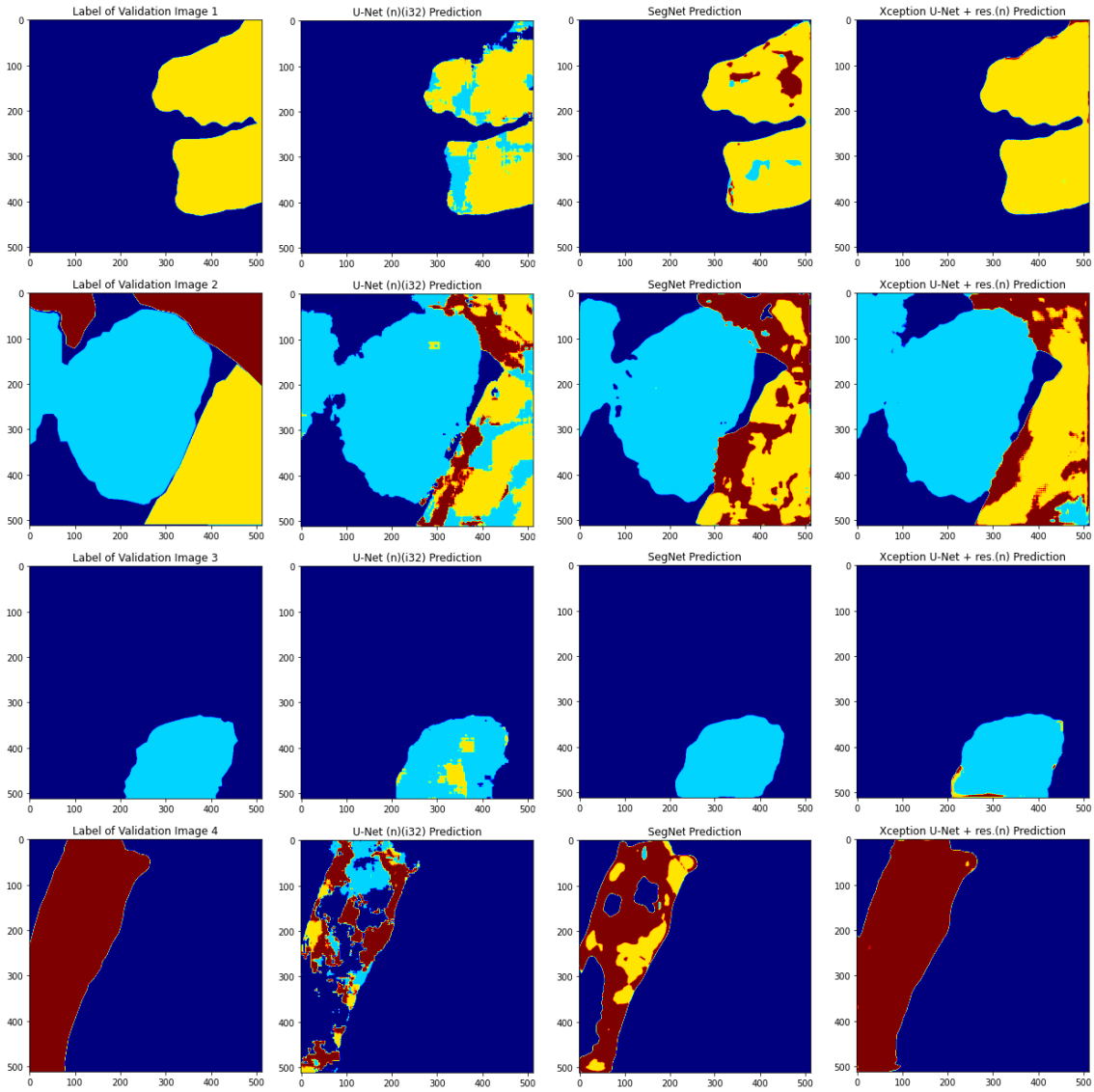


Fig. 66: Comparison of selected models, part 2

Even though the model *U-Net (n)(i64)* performed better than other models according to the table of neural network performances, the segmentation itself does not seem comprehensive and the groups of class pixels are fragmented. The misclassification may be the smallest out of all models, but the following processing of localization of the desired cardiomyocytes might have to be more complicated having a significant space for other mistakes during the process. On the other hand SegNet and Xception U-Net with residual blocks seems to be segmenting compact objects. And even with bigger overal misclassification they might achieve more suitable results for the final program in the end. That is why a user is able to choose between one of the two models when predicting unlabeled images with the final program.

5.4 Final program

In the end it was decided to deliver the program in Google Colab environment (a free Jupyter notebook environment from Google). It runs on the cloud and may be adjusted to use local computational resources. This environment was chosen for multiple reasons. With respect to user experience, it seems like an intermediate step between python executable files and graphical user interface. Since there were not enough time resources to implement fully functional GUI and since python programs require advanced user handeling, Jupyter notebook provided a nice compromise. Another reason for choosing this interface is the fact, that the usage of this software is not imminent and the environment is unclear. In this case, no future problems with respect to the local environment on computing machines regarding operating systems, installing needed libraries and so on. The only thing the user will have to do here apart from running the program is filling the header on top of the Jupyter notebook specifying location of files to be processed and parameters regarding user preferences.

The delivered program consist of two Jupyter notebooks. One notebook is ready to process unlabeled data. In other words, it is able to segment input images with respect to specifications described in previous sections by using already trained segmentation model. The second notebook allows the user to train a model on labeled data, on already created dataset.

The second mentioned notebook regarding neural network training was basically described in the subsection 5.2. All the functionalities were kept, it was just extended with few user specifications like location of dataset, selection of the model, option to train from scratch or starting with already pre-trained model. The output of this process is trained neural network model.

The notebook that performs segmentation on unlabeled data may be divided into the steps shown in the Fig. 67.

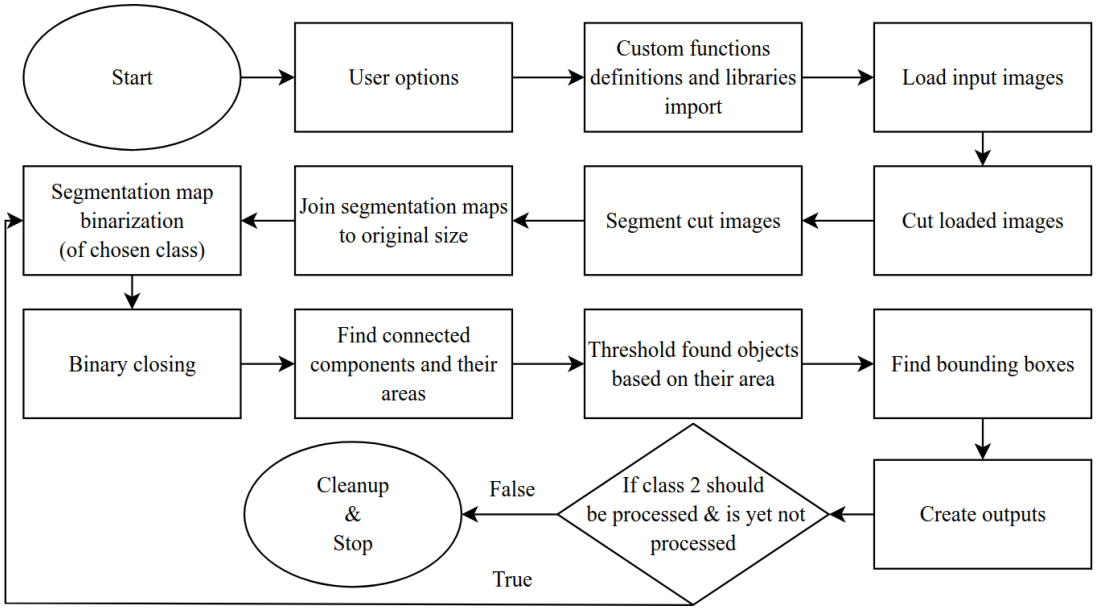


Fig. 67: Program flowchart

Some of the steps from Fig. 67 are described in more detail in the following enumeration.

- User options: user header file where user may define options about outputs, neural network architecture, location of input data,..
- Importing libraries and defining custom functions: definition of function for loading images or for neural network models implementation
- Cut loaded images: the images are cut to dimensions to fit to neural network and to be computer-processable
- Segment cut images: making predictions on cut images resulting in 64 512^2px segmentation maps
- Join segmentation maps to original size: stiching smaller segmentation maps back to the original size of 4096^2px
- Segmentation map binarization: Transforming segmentation map into binary image

  - 1s where the observed class is, 0s the rest. The observed class is the cardiomyocytes of interest by default, if half-collapsed shall be processed as well, it may be processed after processing the first class
- Binary closing: morphological erosion of the dilation is performed to make objects more compact and robust
- Find connected componnents and their areas: an advantage was taken of already used opencv library in the software and its function for finding connected components. Spaghetti algorithm [7] with 8-way connectivity was applied in this process

- Outputs:
  - cutouts of bounding boxes with small margin as images into separate folder
  - bounding boxes coordinates
  - segmentation maps
  - image of colored segmentation map and original image next to each other for clear comparison
  - areas ratio between "healthy" cadiac muscle cell and other ones
- If statement: if the user chooses to process half-collapsed cardiomyocytes (class 2) as well, the process is repeated from binarization step until outputs again

Out of the two highest performing trained segmentation models (SegNet and Xception U-net with residual blocks) the SegNet was chosen for later use for the demonstration of the final program and for its evaluation as well. The reason is time efficiency. The SegNet has 11,742,788 parameters and Xception U-net with residual blocks has 38,430,924 parameters, while both performing similarly for the needs of the final program. This fact however does not exclude the Xception U-net of using it in the final program, the user is able to choose to train this or SegNet model for both training or processing unlabeled data.

Inputs to this process, as mentioned, are microscopy images of 4096^2 dimensions (see an example on Fig. 68).

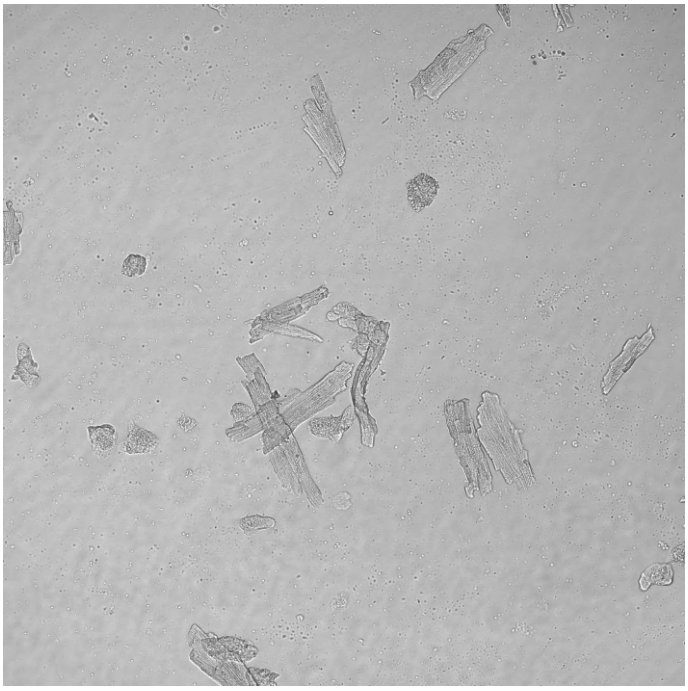


Fig. 68: Example of input to proposed algorithm

After the user fills options about the segmentation process and all the libraries are imported and custom functions are defined, the original images are cut to 64 tiles of 512^2 pixels.

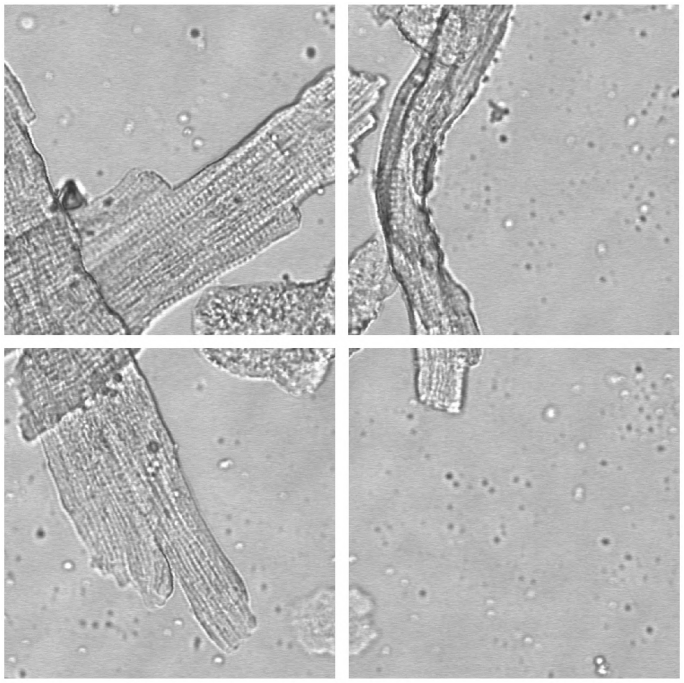


Fig. 69: Example of cropped tiles of input

Predictions are made by chosen trained neural network on the cut images,

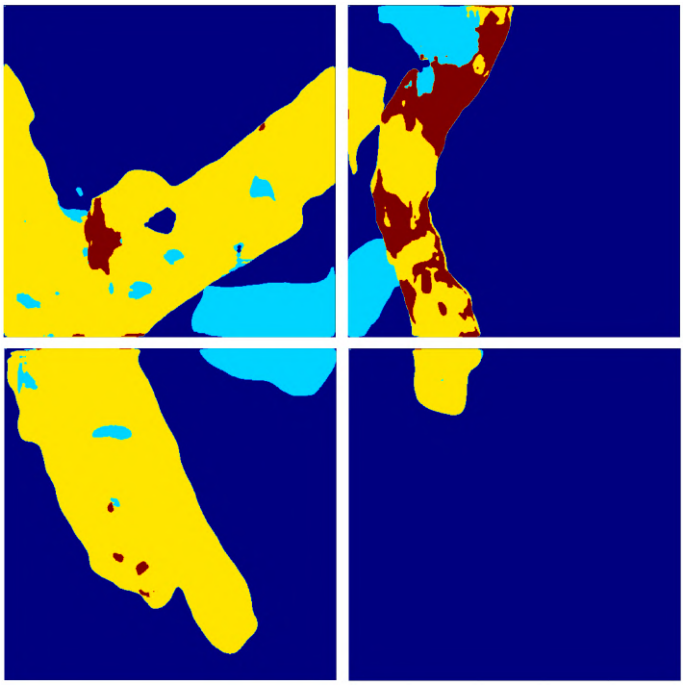


Fig. 70: Predictions of images corresponding to Fig. 69

and the 64 predicted segmentation maps corresponding to cut tiles are then joined to form the segmentation mask of the original image of the dimensions 4096^2.

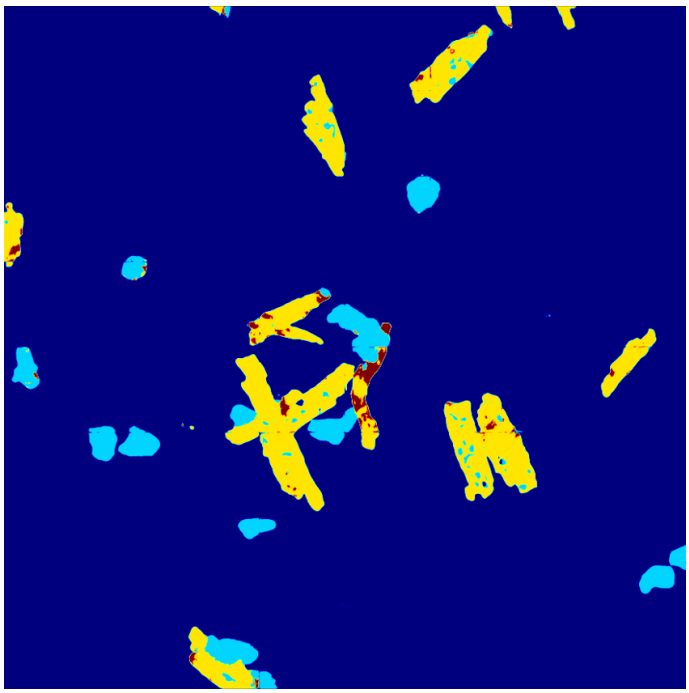


Fig. 71: Segmentation map of input image in the Fig. 68

Then, depending on user specifications, segmentation maps are converted into binary images having 0s everywhere apart from segmented classes of interest, where are 1s. On this binary image binary closing is performed, both is shown in the Fig. 72.

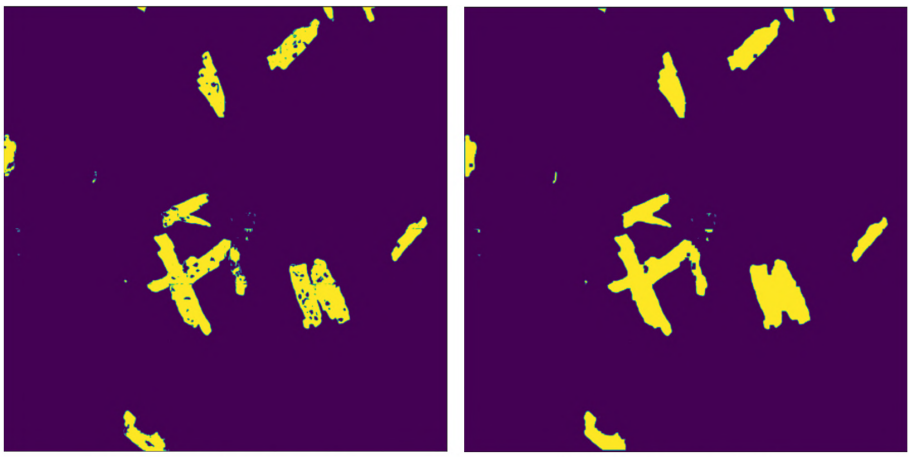


Fig. 72: Binary images containing 1s only where cells of interest were located

After all these steps, connected components are found in the image using Spaghetti algorithm [7] with 8-way connectivity, small objects are ignored (based on areas of

found objects) and bounding boxes around objects of interest may be found and displayed.

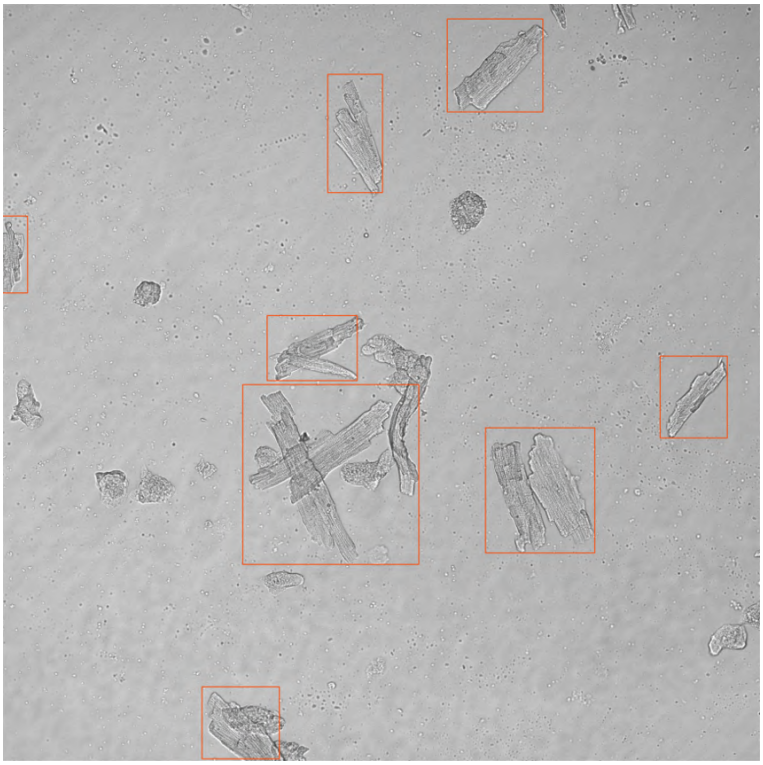


Fig. 73: Original image with located cardiomyocytes of interest

Once having the bounding boxes and segmentation masks, it is straightforward to provide other mentioned outputs.

6 EVALULATION AND RESULTS

The created algorithm was applied to 10 randomly chosen microscopy images while using SegNet trained model and the evaluation was done in the following manner. Firstly, ground truth bounding boxes were created on mentioned images and their coordinates were noted (tight bounding boxes with 30px margin on each side). Secondly, mentioned images were processed by the created program and coordinates of found bounding boxes (with the same margin, 30px) by the program were noted as well. All the corresponding bounding boxes (from ground truth bounding boxes and predicted ones) were paired. The bounding boxes without a pair were noted as well for the following computation. Lastly, the Mean IoU was computed by computing the mean of IOU (41) results. The value of computed Mean IoU is **0.6443**. To get the idea about what this number represents, an object with 0.6338 IoU of ground truth and predicted bounding box (the closest one to the found overal result) is shown on the Fig. 74. The red lines represent ground truth bounding box, the blue lines represent the predicted bounding box.

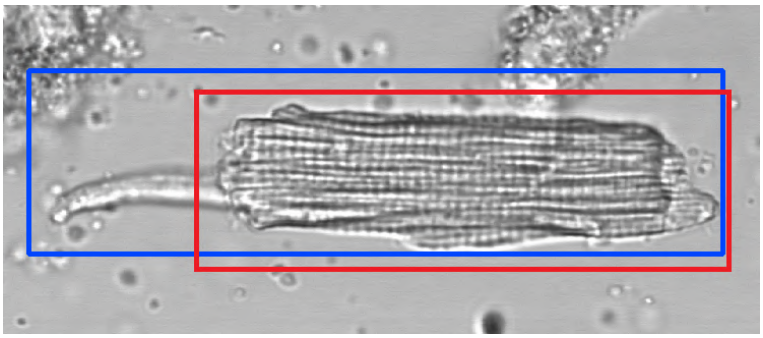


Fig. 74: Cardiomyocyte with 0.6338 IoU

The overal value of Mean IoU is brought down significantly due to misclassified objects (discussed in the following paragraph). Once only the mean of correctly classified objects is computed, the result is **0.8209**. A close example to this value (with 0.8006 IoU) is shown on the Fig. 75.

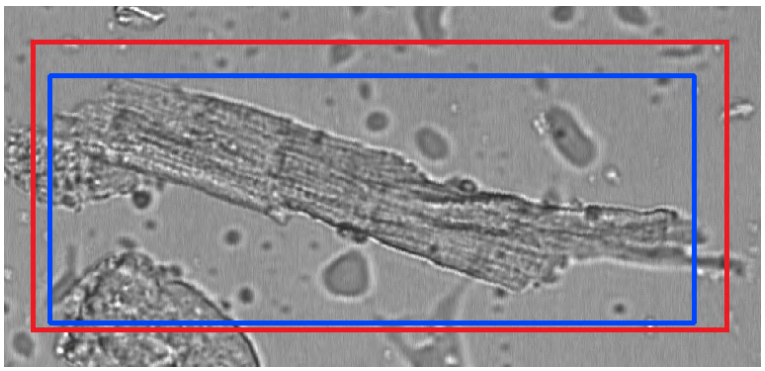


Fig. 75: Cardiomyocyte with 0.8006 IoU

The evaluation problem may be also seen from the perspective of finding desired objects. In this case, 84 objects of interest were to be identified and localized. A succesfully found object of interest is considered the one, that is captured in bounding boxes in the final output. A mislabeled object is considered the one, where it is clear it was purposefully and mistakenly identified as object of interest, meaning a half-collapsed cardiomyocyte in the output bounding box in which it is overlayed under a healthy cardiomycyte because of which this bounding box was created is not a mislabeled object. Results of the evaluation experiment from this point of view are shown in the Table 2.

Total	Found	Missed	Mislabeled
84	84	0	14

Tab. 2: Results of the evaluation experiment

As can be seen in the previous table, the program managed to identify and localize all the objects of interest. There was, however, 14 mislabeled objects. To get the mislabeled number to perspective, some of the cells is very hard to distinguish, whether they belong to the class of interest or to the class of half-collapsed cardiomyocytes. In some cases even a person labeling the data or person evaluating given cells would hesitate between classifying them and the decision would probably be decided based on a personal opinion. Some of these cells are shown in the Fig. 76.

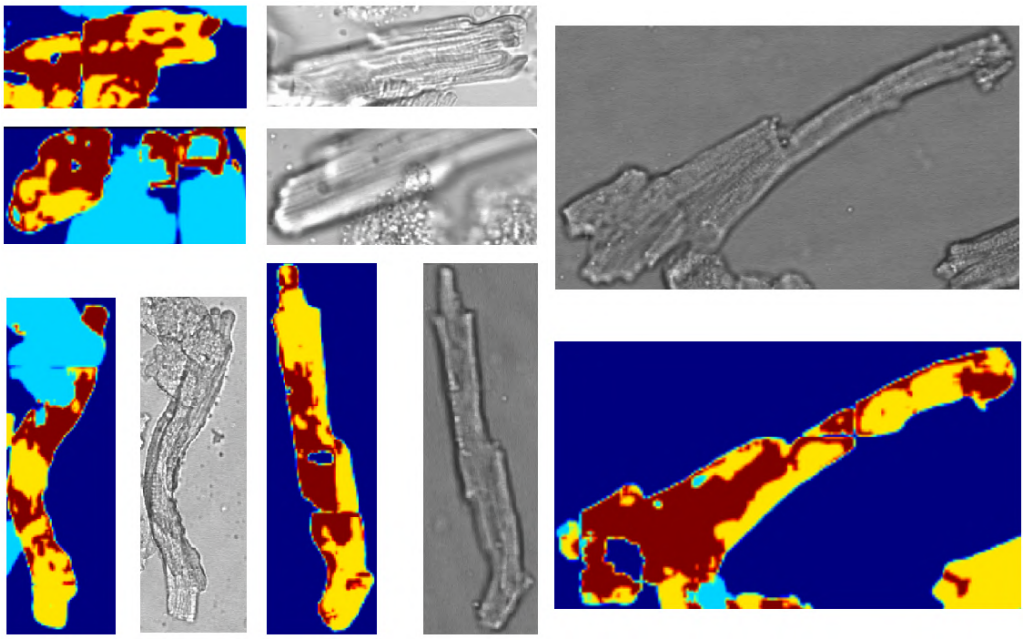


Fig. 76: Examples of mislabeled cardiomyocytes

From the Fig. 76 it is clear, that deciding a class of certain objects is not easy. As can be seen, even the neural network labels these cells usually densely with two colors: two corresponding classes between which the network is deciding, healthy objects of interest (yellow in the Fig. 76) and half collapsed objects (red). The neural network is performing above expectations and is copying the decision making of a human, and how the data were initially labeled as well.

It is important to note that this evaluation is valid only for the performance excluding segmentation part. Segmentation of the models was evaluated separately in the section 5.3. The validation images for this evaluation were taken from the initial training data, since there were no more images available and it was not possible to acquire more of them. Also reserving enough images to make this evaluation valid would result in significant loss of training images and as mentioned, the amount of data was already relatively low. This evaluation may be seen valid only with the assmuption that the segmentation accuracy will be high. Based on the observations in the section 5.3, where the models were tested on validation data (data, which the models were not trained or tested on) it can be presumed, that high level of segmentation accuracy may be ensured with chosen neural network architectures. Naturally the optimal scenario would be trying the software in practice or on new data, if it was possible.

7 DISCUSSION

The main goal of this experiment was to design a program that would allow the user to automatically identify and localize cardiac muscle cells in microscopy images with pre-defined outputs. Apart from the evaluation of the final program performance, the core of this program may be evaluated as well, the semantic segmentation. According to the section 5.3, the SegNet and Xception U-Net with residual blocks showed the highest performance on the segmentation task. These results negate some of the hypothesis about the architecture selection. Models with advanced and state-of-the-art techniques consisting of features like residual and recurrent blocks in the network, attention gates or dilated convolutions [11, 2, 37] seem to have lower performance on this task than smaller, standard and less complex models. This is probably caused by the significantly low amount of training data and these techniques rapidly increasing complexity of neural network models. Another explanation might be that the initial assumption of the model perceiving cardiomyocytes based on local structures was not valid and models that did not have tools to learn detailed structures more than actual whole objects performed better in the end. Second hypothesis, that was negated, was that some models' training performance was not improved by augmentation. What is more, some of the models' performance was even degraded while using augmentation. With the augmentations done so that it does not damage the clarity of cardiomyocytes or so that it makes the structure ambiguous, one would expect the training to be improved, especially in the case of deeper models, but it is not the case.

Possible improvements on the training process and hence on the final segmentation capabilities of the model might be done by implementing custom loss functions, which are not included in the Keras framework (Dice, Jaccard, Focal loss), that would surpress the unbalanced classes issue during training. Adding more data could be a major improvement to the training of chosen models as well. Regarding training data, experimenting with labeling ground truths might also result in better performance. In this case, the objects were labeled as precisely as possible, including blurry or out of focus parts of cardiac muscle cells.

As for the final localization of cardiac muscle cells of interest, it seems to be performing well once the core segmentation has high accuracy. It is important to note that this is the only case it was tested on. The high accuracy is partially caused by testing the final program on images, that were already used on training, testing and validation of neural network models. The reason behind this is that leaving a sufficient amount of images to test the final application would result in significant loss of training data, that was already barely sufficient and new data was not possible to acquire. Nevertheless, based on the observations in section

5.3, where the networks were tested on validation data (that the models did not encounter before), it can be assumpted that a high accuracy of the segmentation task can be ensured by the SegNet or Xception U-Net models and hence the high accuracy of the final localization and outputs as well.

The solution was implemented in Google Colab (Jupyter notebooks) and allows a user to make certain choices and specifications on both main functions provided. The software could be improved from the functional point of view (see previous paragraphs), but from the user interface point of view as well. The first thing that could be done for the user is to migrate the code from Jupyter notebook back to python files and create a graphical user interface.

8 CONCLUSION

This thesis aimed to implement a software for automated and precise localization of cardiomyocytes of interest in microscopy images using segmentation technique based on deep convolutional neural networks. And exactly that was accomplished. It can be said that the software copies the decision making of a human, based on the way the dataset was created (labeled). While having a small amount of very specific and large data with hardware limitations, this thesis was automatically directed in a specific way both in theory and practical implementation. As discussed in chapter 7, there are still some obstacles to be tackled down, options to be examined and improvements to be implemented to make the software more precise, robust and user friendly. However, with the results presented, it can be said, that the assignment was successfully fulfilled.

Automating data acquisition and processing in computer vision and microscopy domain is widely spread. Software like this will ease the work of researchers by providing them with only relevant parts of the image and/or obtains coordinates of areas of interest in the current image for the microscope to focus on and obtain detailed images on these coordinates. This software does not have to be limited to this problem only, it could be easily applied to any other semantic segmentation task by only adjusting neural network architecture accordingly.

9 LITERATURE

[1] *Open Images Dataset V6*. [online], 2020.
URL https://storage.googleapis.com/openimages/web/index.html

[2] Alom, M. Z.; Hasan, M.; Yakopcic, C.; aj.: *Recurrent Residual Convolutional Neural Network based on U-Net (R2U-Net) for Medical Image Segmentation*. [online], 2018.
URL https://arxiv.org/ftp/arxiv/papers/1802/1802.06955.pdf

[3] Amidi, A.; Amidi, S.: *CS 230 - Deep Learning*. [online].
URL https://stanford.edu/~shervine/teaching/cs-230/cheatsheet\-recurrent-neural-networks

[4] Aryal, S.: *Compound Microscope*. [online], 2021.
URL https://microbenotes.com/compound-microscope-principle\-instrumentation-and-applications/

[5] Badrinarayanan, V.; Kendall, A.; Cipolla, R.: *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*. [online].
URL https://arxiv.org/pdf/1511.00561.pdf

[6] Bhagwat, P.: *Introduction to Artificial Neural Networks in Python*. [online], 2019.
URL https://aibusiness.com/document.asp?doc_id=761027

[7] Bolelli, F.; Allegretti, S.; Baraldi, L.; aj.: *Spaghetti Labeling: Directed Acyclic Graphs for Block-Based Connected Components Labeling*. IEEE Transactions on Image Processing, 29(1):1999–2012, 2019.

[8] Bustamam, A.; Abdillah, B.; Sarwinda, D.: *Image processing based detection of lung cancer on CT scan images*. [online].
URL https://www.researchgate.net/publication/320687993_Image_processing_based_detection_of_lung_cancer_on_CT_scan_images

[9] Chabrier, S.; Rosenberger, C.; Emile, B.; aj.: *Optimization Based Image Segmentation by Genetic Algorithms*. [online], 2008.
URL https://hal.archives-ouvertes.fr/hal-00255987/document

[10] Chemnitz, T. U.: *Superpixel*. [online], 2018.
URL https://www.tu-chemnitz.de/etit/proaut/en/research/superpixel.html

[11] Chen, L.-C.; Papandreou, G.; Kokkinos, I.; aj.: *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*. [online], 2017.
URL https://arxiv.org/abs/1606.00915

[12] Chen, L.-C.; Zhu, Y.: *Image segmentation metrics*. [online].
URL https://keras.io/api/metrics/segmentation_metrics/

[13] Chen, L.-C.; Zhu, Y.: *Semantic Image Segmentation with DeepLab in TensorFlow*. [online], 2018.
URL https://ai.googleblog.com/2018/03/semantic-image-segmentation-with.html

[14] Coleman, A.: *Angle of view*. [online], 2013.
URL https://stackoverflow.com/questions/19720918/angle-of-view-as-one-float

[15] Cousty, J.; Bertrand, G.; Najman, L.; aj.: *Watershed Cuts: Minimum Spanning Forests and the Drop of Water Principle*. [online], 2009.
URL https://hal-upec-upem.archives-ouvertes.fr/hal-00622410

[16] Deshpande, M.: *Complete Guide to Deep Neural Networks – Part 2*. [online], 2017.
URL https://pythonmachinelearning.pro/complete-guide-to\-deep-neural-networks-part-2/

[17] Education, I. C.: *Recurrent Neural Networks*. [online], 2020.
URL https://www.ibm.com/cloud/learn/recurrent-neural-networks

[18] Ferrari, S.: *Image Processing I*. [online].
URL http://homes.di.unimi.it/ferrari/ElabImm2011_12/index.html

[19] Gonzalez, R. C.; Woods, R. E.: *Digital image processing*. New York, NY: Pearson, 2018, ISBN 978-0133356724.

[20] Hole, K.; Gulhane, V.; Shellokar, N.: *Application of Genetic Algorithm for Image Enhancement and Segmentation*. [online], 2013.
URL http://ijarcet.org/wp-content/uploads/IJARCET-VOL-2-ISSUE-4-1342-1346.pdf

[21] Jung, A. B.; Wada, K.; Crall, J.; aj.: *imgaug*. https://github.com/aleju/imgaug, 2020.

[22] Kentaro, W.: *labelme*. [online], 2016.
URL https://github.com/wkentaro/labelme

[23] Kon, K.: *Human heart and close-up view of cardiac muscle structure, 3D illustration*. [online].
URL https://www.shutterstock.com/image-illustration/human-heart-close-view-cardiac-muscle-769633708

[24] Le, J.: *How to do Semantic Segmentation using Deep learning*. [online], 2021.
URL https://nanonets.com/blog/how-to-do-semantic-segmentation-using\-deep-learning/

[25] Li, F.-F.; Krishna, R.; Xu, D.: *Stanford Vision and Learning Lab: CS231n: Convolutional Neural Networks for Visual Recognition*. [online], 2021.
URL http://cs231n.stanford.edu

[26] ioLight Limited: *Different Types of Light Microscopy*. [online].
URL https://iolight.co.uk/different-types-of-light-microscopy/

[27] Majtner, T.: *Texture-Based Image Description in Fluorescence Microscopy*. [online], 2015.
URL https://www.researchgate.net/publication/320757560_Texture-Based_Image_Description_in_Fluorescence_Microscopy

[28] Matcha, A. C. N.: *A 2021 guide to Semantic Segmentation*. [online], 2021.
URL https://nanonets.com/blog/semantic-image-segmentation-2020/

[29] MathWorks: *Marker controlled watershed segmentation*. [online].
URL https://fr.mathworks.com/help/images/marker-controlled-watershed-segmentation.html

[30] MicroscopyU: *Animal and Human Cells in Culture*. [online].
URL https://www.microscopyu.com/galleries/fluorescence/cells

[31] Mokobi, F.: *Light Microscope*. [online], 2017.
URL https://microbenotes.com/light-microscope/

[32] Murphy, D.: *Fundamentals of light microscopy and electronic imaging*. A JOHN WILEY & SONS, INC, 2001, ISBN ISBN 0-471-25391-X.

[33] Mwiti, D.; Li, K.: *Image Segmentation in 2021: Architectures, Losses, Datasets, and Frameworks*. [online], 2021.
URL https://neptune.ai/blog/image-segmentation

[34] Mwiti, D.; Li, K.: *Image Segmentation in 2021: Architectures, Losses, Datasets, and Frameworks*. [online], 2021.
URL https://neptune.ai/blog/image-segmentation

[35] Nielsen, M.: *Neural Networks and Deep Learning*. [online], 2019.
URL http://neuralnetworksanddeeplearning.com/chap2.html

[36] Nyúl, L. G.: *Fuzzy Techniques for Image Segmentation*. [online], 2008.
URL https://www.inf.u-szeged.hu/~ssip/2008/presentations2/Nyul_fuzzy_segmentation.pdf

[37] Oktay, O.; Schlemper, J.; Folgoc, L. L.; aj.: *Attention U-Net: Learning Where to Look for the Pancreas*. [online], 2018.
URL https://arxiv.org/abs/1804.03999

[38] Peltarion: *Categorical crossentropy*. [online].
URL https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy

[39] Ronneberger, O.; Fischer, P.; Brox, T.: *U-Net: Convolutional Networks for Biomedical Image Segmentation*. [online], 2015.
URL https://arxiv.org/abs/1505.04597

[40] Rosina, L.; Rosina, J.; aj.: *Medicínská biofyzika*. Grada, 2005, ISBN ISBN 80-247-1152-4.

[41] Rühl, H.: *Optical Microscopes – Some Basics*. [online], 2012.
URL https://www.leica-microsystems.com/science-lab/optical-microscopes-some-basics/

[42] Saha, S.: *A Comprehensive Guide to Convolutional Neural Networks*. [online], 2018.
URL https://towardsdatascience.com/a-comprehensive-guide-to-convolutional\-neural-networks-the-eli5-way-3bd2b1164a53

[43] Sahir, S.: *Canny Edge Detection Step by Step in Python — Computer Vision*. [online].
URL https://towardsdatascience.com/canny-edge-detection-step-by-step\-in-python-computer-vision-b49c3a2d8123

[44] Siegfried Reipert, U. W.: *Light microscopy techniques*. [online].
URL https://cius.univie.ac.at/techniques/light-microscopy/light-microscopy-techniques/

[45] Spring, K.; Davidson, M.: *Introduction to Fluorescence Microscopy*. [online].
URL https://www.microscopyu.com/techniques/fluorescence/introduction-to-fluorescence-microscopy

[46] The MathWorks, I.: *graythresh (function documentation)*. [online].
URL https://www.mathworks.com/help/images/ref/graythresh.html

[47] Thomasa, C.: *An introduction to Convolutional Neural Networks*. [online], 2019.
URL https://towardsdatascience.com/an-introduction-to\-convolutional-neural-networks-eb0b60b58fd7

[48] Tyagi, M.: *Mathematical and practical implementation of various image segmentation techniques*. [online], 2018, uRL https://towardsdatascience.com/image-segmentation-part-1-9f3db1ac1c50 and https://towardsdatascience.com/image-segmentation-part-2-8959b609d268.

[49] Vinod, R.: *Dealing with class imbalanced image datasets using the Focal Tversky Loss*. [online], 2020.
URL https://towardsdatascience.com/dealing-with-class-imbalanced-image-datas

[50] Wood, T.: *DWhat is the F-score*. [online].
URL https://deepai.org/machine-learning-glossary-and-terms/f-score

[51] Wood, T.: *What is the Softmax Function?* [online].
URL https://deepai.org/machine-learning-glossary-and-terms/softmax-layer

[52] Yakubovskiy, P.: *Segmentation Models*. https://github.com/qubvel/segmentation_models, 2019.

[53] Yu, F.; Koltun, V.: *Multi-Scale Context Aggregation by Dilated Convolutions*. [online], 2015.
URL https://arxiv.org/abs/1511.07122

10 LIST OF FIGURES

11 LIST OF APPENDICES

A Attachment

Contents of the enclosed .zip file

- Models.py - file with implemented segmentation model architectures, that were trained and evaluated, as callable functions
- DataPrep.py - python script for data preparation
- TrainMe.ipynb - jupyter notebook providing the program for training segmentation model on labeled data
- SemSeg.ipynb - juptyer notebook providing the final program for processing unlabeled images