Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies

FIIT-100241-116291

Martin Sivák

Histological Image Data Processing Using Methods of Computer Vision and Deep Neural Networks

Bachelor's thesis

Supervisor: prof. Ing. Vanda Benešová, PhD.

May 2025

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies

FIIT-100241-116291

Martin Sivák

Histological Image Data Processing Using Methods of Computer Vision and Deep Neural Networks

Bachelor's thesis

Study program: Informatics

Field of study: Computer Science

Training workspace: Institute of Computer Engineering and Applied Informatics, FIIT STU, Bratislava

Supervisor: prof. Ing. Vanda Benešová, PhD.

May 2025

Slovak University of Technology in Bratislava
Institute of Computer Engineering and Applied Informatics

Faculty of Informatics and Information Technologies
Academic year: 2024/2025
Reg. No.: FIIT-100241-116291

BACHELOR THESIS TOPIC

Student: **Martin Sivák**
Student's ID: 116291
Study programme: Informatics
Study field: Computer Science
Thesis supervisor: prof. Ing. Vanda Benešová, PhD.
Head of department: Ing. Katarína Jelemenská, PhD.

Topic: **Histological Image Data Processing Using Methods of Computer Vision and Deep Neural Networks**

Language of thesis: English

Specification of Assignment:

Pri spracovaní medicínskych obrazových dát, napr. histologických mikroskopických skenov, nachádzajú metódy počítačového videnia stále významnejšie uplatnenie. Moderné prístupy s využitím hlbokého učenia umožňujú napr. analýzu na úrovni buniek, vyšších štruktúr alebo získavanie ďalších, histologicky relevantných informácií. Analyzujte súčasný stav problematiky využitia počítačového videnia pri spracovaní digitálnych histologických snímok. Zamerajte sa predovšetkým na metódy využívajúce hlboké neurónové siete. Navrhnite vlastnú metódu na podporu diagnostiky s využitím moderných metód umelej inteligencie. Navrhnutú metódu realizujte softvérovým prototypom s využitím knižníc a rámcov vhodných na spracovanie medicínskych dát. Riešenie overte experimentom s reálnymi histologickými dátami. Vyhodnoťte presnosť, robustnosť a časovú efektívnosť spracovania. Výsledky porovnajte s inými publikovanými riešeniami.

Length of thesis: 40

Deadline for submission of Bachelor thesis: 12. 05. 2025
Approval of assignment of Bachelor thesis: 15. 04. 2025
Assignment of Bachelor thesis approved by: doc. Ing. Ján Lang, PhD. – Study programme supervisor

Declaration of Honour

I, the undersigned, hereby declare that the work presented in this thesis is my own and has been completed based on consultation with my supervisor and the referenced literature.

Date: May 12, 2025

Martin Sivák

Acknowledgment

I sincerely thank my thesis supervisor, Prof. Ing. Vanda Benešová, PhD, for her help, guidance, and inspiration during the whole period of working on this Bachelor's thesis. She was always willing to give me expert advice and constantly motivated me to improve. I would also like to thank my family and friends for their support and motivation.

Annotation

Slovak University of Technology, Bratislava
Faculty of Informatics and Information Technologies
Degree Course: Informatics

Author: Martin Sivák
Bachelor's Thesis: Histological Image Data Processing Using Methods of Computer Vision and Deep Neural Networks
Supervisor: prof. Ing. Vanda Benešová, PhD.
May 2025

In this thesis, we look at state-of-the-art weak segmentation techniques in digital pathology, focusing on segmenting lymphocyte cell nuclei in breast cancer patients. The main challenge stems from the weak annotations of nuclei in the form of bounding boxes instead of exact pixel-level masks. To tackle this challenge, we introduce a hybrid approach, where we use traditional computer vision techniques, such as Otsu and adaptive thresholding, and marked watershed, to create pixel-level pseudo-masks that are used to train a U-Net model. We show that using a small, although fully annotated, dataset is insufficient to train the model. Next, we try training the model on the pseudo-masks created by different computer vision pipelines, on the large weakly annotated dataset. To use the combined strength of different pseudo-masks, we then try making a second generation of them by trying various fusion strategies. Finally, we tried a transfer learning approach, where a model pretrained on a large dataset with pseudo-masks is fine-tuned on a small, fully annotated dataset, and this model achieved the best results. We evaluate each model using quantitative metrics such as the Dice coefficient and the intersection over the union, and qualitative metrics by visualizing its predictions.

Anotácia

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Študijný program: Informatika

Autor: Martin Sivák
Bakalárska práca: Histological Image Data Processing Using Methods of Computer Vision and Deep Neural Networks
Vedúci bakalárskeho projektu: prof. Ing. Vanda Benešová, PhD.
May 2025

V tejto práci sa zaoberáme technikami slabej segmentácie v digitálnej patológii so zameraním na segmentáciu jadier lymfocytov u pacientov s rakovinou prsníka. Hlavná výzva vyplýva zo slabých anotácií jadier vo forme ohraničujúcich rámčekov namiesto presných masiek na úrovni pixelov. Na riešenie tejto výzvy zavádzame hybridný prístup, kde používame tradičné techniky počítačového videnia, ako sú Otsu a adaptívne prahovanie, a značkami-riadený algoritmus watershed, na vytvorenie pseudo-masiek, ktoré sa používajú na trénovanie modelu U-Net. Ukazujeme, že použitie malého, plne anotovaného datasetu je na trénovanie modelu nedostatočné. Ďalej vyskúšame trénovať model na pseudo-maskách vytvorených rôznymi metódami počítačového videnia na veľkom, slabo anotovanom datasete. Aby sme využili kombinovanú silu rôznych pseudo-masiek, vytvoríme ich druhú generáciu vyskúšaním rôznych stratégií zlúčenia. Nakoniec použijeme prístup učenia s prenosom, kde sa model predtrénovaný na veľkom datasete s pseudo-maskami dotrénuje na malom, plne anotovanom datasete, pričom tento model dosiahol najlepšie výsledky. Každý model hodnotíme pomocou kvantitatívnych metrík, ako sú Dice koeficient a IoU, a kvalitatívnych metrík vizualizáciou jeho predpovedí.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence

ML Machine Learning

AI/ML Artificial Intelligence and Machine Learning

NLP Natural Language Processing

CT Computed Tomography

MRI Magnetic Resonance Imaging

H&E Hematoxylin and Eosin

HER2 Human Epidermal Growth Factor Receptor 2

HR Hormonal Receptor

TNBC Triple Negative Breast Cancer

TIL tumor-infiltrating lymphocyte

WSI whole-slide image

IoU Intersection over Union

ViT Vision Transformer

2D Two-dimensional

3D Three-dimensional

CLAHE Contract-Limited Adaptive Histogram Equalization

ANN artificial neural network

DNN deep neural network

ReLU Rectified Linear Unit

ELU Exponential Linear Unit

GELU Gaussian Error Linear Unit

MSE Mean squared error

BCE binary cross-entropy error

CCE categorical cross-entropy error

DSC Dice similarity coefficient

TP True Positive

FP False Positive

TN True Negatives

FN False Negatives

ROC Receiver Operating Characteristics

AUC Area Under the ROC Curve

TPR True Positive Rate

FPR False Positive Rate

CNN Convolutional Neural Network

RGB red-green-blue

R-CNN Region-based Convolutional Neural Network

YOLO You Only Look Once

RNN Recurrent Neural Network

DNA Deoxyribonucleic acid

SAM Segment Anything Model

ILP Integer Linear Programming

ROI region of interest

FSD Fourier Shape Descriptors

SMOTE synthetic minority oversampling technique

DDTN dense dual-task network

TCGA The Cancer Genome Atlas

EMA exponential moving average

DQ Detection Quality

SQ Segmentation Quality

PG Panoptic Quality

JSON JavaScript Object Notation

PNG Portable Network Graphics

PIL Python Imaging Library

CPU Central Processing Unit

GPU Graphical Processing Unit

HE Histogram Equalization

Chapter 1

Introduction

In the past few years, algorithms of computer vision and especially artificial intelligence and deep neural networks brought promising results in image data processing, mostly in the tasks of object detection, semantic segmentation, and classification [1]. These advancements may have a significant impact in a vast number of fields, one of them being medicine [2, 3, 4].

In medicine, different types of imaging techniques are being used to provide both non-invasive and invasive visualizations of internal organs, tissues, and other structures. Among non-invasive techniques, we can count, for example, X-ray radiography, ultrasound imaging, magnetic resonance imaging (MRI), and computed tomography (CT). Apart from them, we also mentioned invasive techniques - these are necessary when doctors need to examine a microscopic piece of tissue, e.g., potential tumor tissue or tissue that is known to be a tumor. This is a discipline called histology or histopathology. Doctors can obtain the tissue either by performing a biopsy or surgical resection. Biopsy is a less invasive method - it involves inserting a needle into the patient's body tissue and taking out a small sample. On the other hand, surgical resection is much more invasive and involves some sort of

surgical procedure during which the desired piece of tissue is removed. Depending on what doctors want to examine, these samples are then processed further. In the histopathology domain, staining of these images with chemicals is a common practice. This staining helps to create visual contrast between cells, tissues, and other objects on the image slide. Hematoxylin and eosin staining (H&E staining) is the most widely used staining method for histopathology slides [5]. Both its components are used to stain different regions of the image. Hematoxylin is responsible for colorizing cell nuclei into shades of deep blue and purple, while eosin is used for staining the extracellular matrix, cytoplasm, and connective tissues in shades of pale red and pink [5]. An example of this staining on a histopathology image can be seen in Figure 1.1.

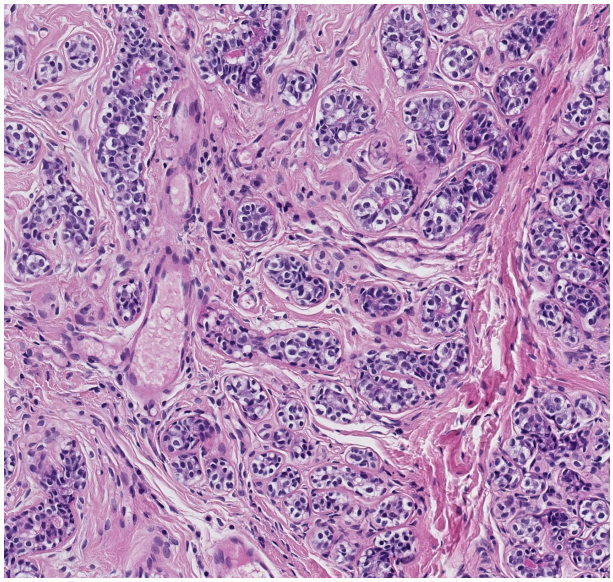


Figure 1.1: Example of histology image stained with hematoxylin and eosin

Slides stained by these chemicals are then examined by histopathology experts who try to identify key features that would determine a diagnosis, future treatment plan, or other subsequent steps. A very good example of this whole process can serve as a method of adjusting treatment for patients who suffer from breast

cancer.

1.1 Motivation

In recent decades, breast cancer has been one of the leading causes of death among women and the second most commonly diagnosed type of cancer worldwide [6, 7]. According to [6] in 2022, breast cancer was attributed to approximately 2.3 million newly diagnosed patients - this represents 11.6% of all diagnosed cancer patients in that year and 666,000 deaths, comprising 6.9% of all cancer deaths. [7] informs that in the USA in the year of 2023, breast cancer among women accounted for more than 297,000 new cases - 31% of all new female cancer cases and more than 43,000 deaths - 15% of all female cancer deaths.

When dealing with breast cancer, one needs to keep in mind that there are also different subtypes of breast cancer. Firstly introduced in [8], we now know four breast cancer molecular subtypes, based on the positivity or negativity of several receptors. These receptors are Human Epidermal Growth Factor Receptor 2 (HER2) and Hormonal Receptor (HR), which are positive if either Estrogen or Progesterone receptors are positive; otherwise, it is negative. These four classes, along with respective receptor statuses, can be seen in Table 1.1.

Table 1.1: Breast Cancer Molecular Subtypes and Receptor Statuses

Subtype Class	**Hormone Receptor (HR)**	**HER2**
Luminal A	Positive	Negative
Luminal B	Positive	Positive
HER2-enriched	Negative	Positive
Triple Negative (TNBC)	Negative	Negative

From the aforementioned subtypes, the last three are the ones that currently have the worst prognosis [9, 10]. Identifying and using certain biomarkers could po-

tentially improve the prognosis of patients with these subtypes of breast cancer. Tumor-infiltrating lymphocytes (TILs) appear as such biomarkers, especially their presence, number, and spatial organization within the tumor and tumor-related tissue [11, 12, 13]. However, manual identification and visual recognition of TILs from H&E-stained slides is a difficult, time-consuming, and error-prone task even when performed by experienced histopathology experts [11, 13].

1.2 Objectives

Manual analysis of histopathology slides is expensive, takes a long time to complete, and requires highly trained professionals and quality assurance by performing peer reviews [4]. With the invention of virtual microscopy, which enables H&E-stained glass slides to be converted into digital slides, and the introduction of Whole-slide Images (WSIs), the field is entering a new era. The term Digital Pathology or Digital Histopathology is often used. In Digital Pathology, much effort is put into developing tools that would help medical experts to semi- or fully automate the visual analysis of the digital slides. Entities such as different tissue types and cells can be identified and classified.

Deep learning has shown extreme potential in many areas, including medicine and processing of medical image data [1]. The usage of deep learning models also introduces a new challenge: for them to produce reasonably good results, they need a huge amount of high-quality data [14]. Precise manual annotation of histology slides is not an easy nor a cheap task, as we have mentioned earlier. Therefore, our aim in this work is to develop and implement a pipeline for automated segmentation of tumor-infiltrating lymphocytes from breast cancer histology image slides using two sources of data:

- Tumor Infiltrating Lymphocytes in Breast Cancer - TIGER - a large dataset

with weak annotations of lymphocyte nuclei, in the form of bounding boxes, which is publicly available via the Grand Challenge platform [15] and

- Triple Negative Breast Cancer Nuclei Segmentation - TNBC - a small dataset with full pixel-level annotations of lymphocyte nuclei, which is also publicly available [16].

Since the provided weak annotations of the TILs are in the form of bounding boxes, our goal is twofold:

1. Develop, implement, and compare different strategies for creating pseudo-masks by converting bounding box annotations into pixel mask annotations by utilizing methods of traditional computer vision and
2. Train a deep learning segmentation model, using different combinations of pseudo-masks, and evaluate it on the evaluation metrics such as Intersection over Union (IoU) and Dice coefficient.

In the end, we want to compare models trained on various pseudo-mask creation strategies in the semantic segmentation task of individual lymphocyte nuclei, utilizing both a weakly annotated large dataset and a small, fully annotated dataset of H&E-stained histology images of breast cancer patients. We also want to utilize transfer learning, where we pre-train the model on the TIGER dataset and then fine-tune it using the TNBC dataset.

This work is structured in the following way: Chapter 2 describes the concept of computer vision and machine learning. Chapter 3 looks at the history and current trends in deep learning. Chapter 4 describes the state-of-the-art works in our field of interest. In Chapter 5 we present our work, and in Chapter 6 we conclude the work. In Appendix A we show the plan of work, and in Appendix B we include the technical documentation for our work.

Chapter 2

Computer Vision and Machine Learning

Vision is one of our primary senses. Therefore, it is understandable that we seek methods for capturing, storing, analyzing, and processing this kind of data. Digital image processing is a vast area of different disciplines, ranging from low-level operations such as noise reduction, image sharpening, and contrast adjustment through mid-level operations like classification and segmentation to high-level operations which involve making higher sense of the images and resembling human visual perception and intelligence [17].

Computer Vision, a subfield of computer science and an extension of digital image processing, focuses on using computers to extract meaningful knowledge from images in various ways, thereby emulating the capabilities of the human brain and visual cortex [17]. As part of machine learning and artificial intelligence, computer vision uses automation algorithms to analyze and process visual data, including 2D and 3D images as well as videos [18, 19].

Machine learning is a subset of artificial intelligence that includes both statistical learning and deep learning algorithms to make intelligent decisions based on data. Modern computer vision primarily utilizes deep learning techniques, as illustrated in Figure 2.1. In classical programming, we are designing an explicit program that produces desired outputs for specific inputs. In machine learning, however, we let the machine design an appropriate program, given the specified set of inputs and outputs (labels) by analyzing the features and patterns of the input with relation to the output [20].

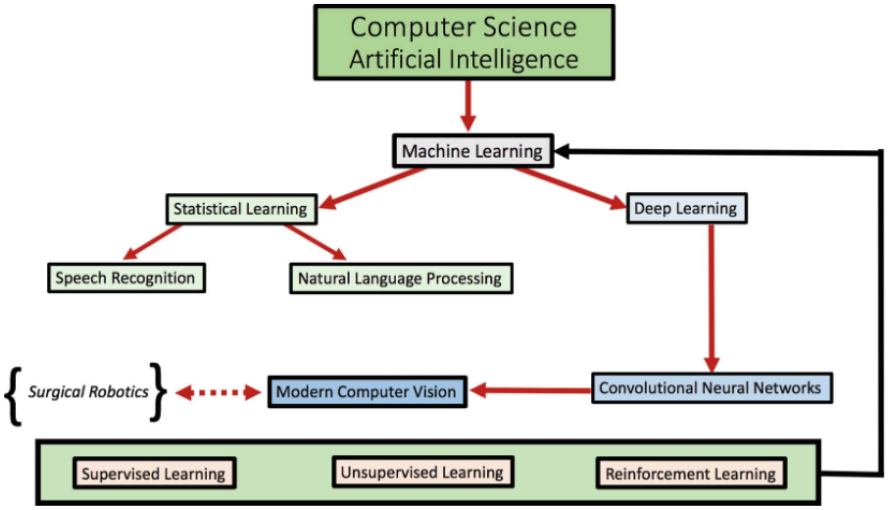


Figure 2.1: Division of AI/ML [19].

With the advent of deep learning [1] and especially convolutional neural networks [21], computer vision is now a field of huge interest.

2.1 Preprocessing

Since the machine learning algorithms try to examine the relationship between input and output, we need to ensure an appropriate quality of the input data. Especially in medical imaging and digital histopathology, where the different staining

techniques, scanning tools, or position of the tissue can vary widely, and this can affect the further analysis [22].

In the domain of digital histopathology, a common issue is the varying intensities of purple, red, and pink tones of H&E-stained slides [22]. For this purpose, different stain normalization techniques were created. Among the examples, we can list the Macenko, Reinhart, or Zheng normalization techniques, which try to normalize the dataset of input images [22]. Among some other techniques, we can list the histogram equalization, Contrast-Limited Adaptive Histogram Equalization (CLAHE), and the power law (gamma) transformation [23].

2.2 Core Computer Vision Tasks

When analyzing an image, we can come across the three main tasks [20]. The example of each of these tasks can be seen in Figure 2.2.

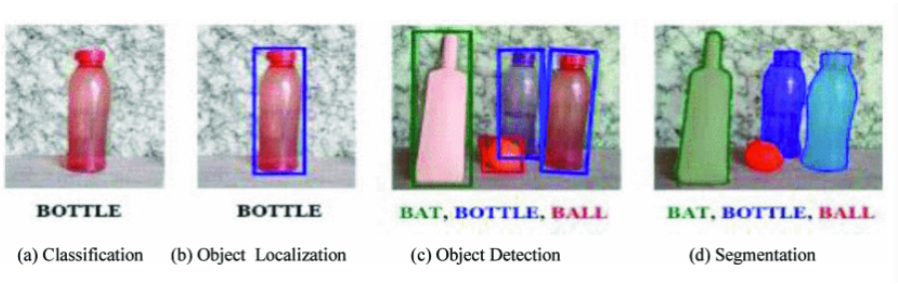


Figure 2.2: Different Computer Vision tasks [20].

2.2.1 Image Classification

Image classification is used when we have a label categorizing the image into one of the classes (or multiple classes) in the set of classes [20]. For example, in the medical imaging domain, we could label an image with the "disease" or "non-disease" class.

2.2.2 Object Localization and Object Detection

Object localization and object detection are very similar tasks. While the former is a task of localizing a single object instance in the image, the latter is a task where multiple instances of one or many objects should be detected and bounded [20].

2.2.3 Segmentation

Sometimes we want to get a more detailed label than just an approximate object location (bounding rectangle). Segmentation utilizes pixel-level classification, where pixels can be labeled based on their relationship to various classes. According to [20], we know two main types of segmentation:

- Semantic segmentation, where each pixel of a certain class gets the same label, no matter the number of instances, and
- Instance segmentation, where the pixels of different instances of the same class are distinguished as well.

Apart from the currently most popular and interesting segmentation algorithms using deep learning, we also know some traditional segmentation techniques, like the Otsu thresholding, adaptive thresholding, and watershed algorithm.

2.3 Learning Paradigms

Computer vision algorithms can be further divided by how they can learn from the data [20].

Supervised Learning In the supervised learning tasks, both the data and their respective labels are known and are available to the model during the training.

Typical supervised learning tasks include classification, detection, and segmentation. By the quality and precision of the labels and the task goal, we can split supervised learning into three categories:

- Standard supervised learning, when available labels are of the same quality as the labels we want to predict, e.g., bounding box to bounding box.
- Strong supervised learning, when the training labels contain richer information than the labels we want to predict, e.g., bounding box from pixel-level annotations.
- Weak supervised learning, when the training labels contain less precise information than the labels we want to predict, e.g., a bounding box from image image-level annotation.

Unsupervised Learning In unsupervised learning, on the other hand, the data labels are not available to the model during the training. The model itself must discover the patterns directly from the raw input, for example by grouping similar samples into clusters (e.g., k-means or hierarchical clustering), reducing dimensionality to capture the most informative features (e.g., principal component analysis or autoencoders), or learning a compact representation through self-organizing maps or generative models. Examples of tasks include: anomaly detection, where outliers stand out from the norm, and density estimation, where the goal is to model the underlying data distribution. Because no true labels guide the process, evaluation often relies on intrinsic measures (such as silhouette score for clusters) or downstream performance when those representations are fed into supervised tasks.

Chapter 3

Deep Neural Networks

The history of artificial neural networks (ANN) dates back to 1943. In [24] authors tried to mathematically describe the activity of biological neurons in the human brain. Using these principles, they built the first artificial neuron and artificial neural network. In 1974, a PhD student, Paul Werbos, introduced in [25] the idea of backpropagation of errors by which ANNs can learn other than linearly separable problems, and this idea was further expanded in [26]. Artificial neural networks that contain many hidden layers are also called deep neural networks (DNN), and the process of training this network is called deep learning [1]. Over the years, deep learning and one of its variants - a convolutional neural network that was proposed in [27] - were found to be very effective and precise in domains that were found unreachable by the classical AI and ML algorithms [1]. This was caused by their ability to capture abstract and complex patterns that simpler models found impossible to catch. Such examples include analysis of image data [28, 29] and recent advancements in natural language processing (NLP) [30].

3.1 Structure

The fundamental part of every artificial neural network is the neuron. A neuron is basically a function that has one or more inputs and one output. Inside this neuron, a mathematical computation is being done to transform input into output. Input can also be referred to as an input vector or a vector of input features. Each input feature has its own weight by which it is multiplied. Next, a bias is added to the multiplied and summed features and weights. This calculation is still linear, so for it to be able to capture more complex patterns, we need to apply a non-linear activation function to its output. The mathematical representation of an artificial neuron can be seen in the Equations 3.1 and 3.2 [31].

$$z = b + \sum_{i=1}^{n} (w_i x_i) \tag{3.1}$$

$$a = \varphi(z) \tag{3.2}$$

Where z is the output produced by the linear unit, b is the bias, n is the number of input features, x_i is the i-th input feature, w_i is the weight associated with the i-th input feature, a is the actual output, and φ is the activation function.

A visual example of the artificial neuron can be seen in Figure 3.1.

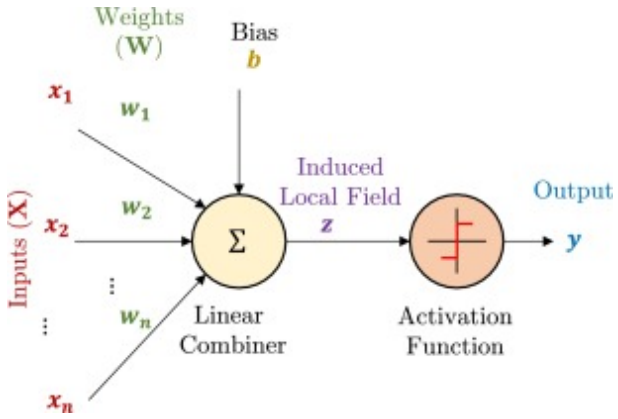


Figure 3.1: Artificial neuron [32].

3.1.1 Activation Functions

Activation functions are used to break linearity in neural networks - this enables them to capture more complex patterns, which are not linearly separable. Activation functions are used in combination with linear functions inside neurons. Different activation functions can be used, such as Sigmoid, Tanh, ReLU, ELU, GELU, and many more [33, 34]. The important part of an activation function is also its gradient, which is computed during backpropagation.

Sigmoid Sigmoid is computed by the Equation 3.3 and its derivative by the Equation 3.4. As we can see in Figure 3.2 has a steep gradient around zero and it gradually flattens on both sides.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.3}$$

$$\sigma^{'}(z) = \sigma(z)(1 - \sigma(z)) \tag{3.4}$$

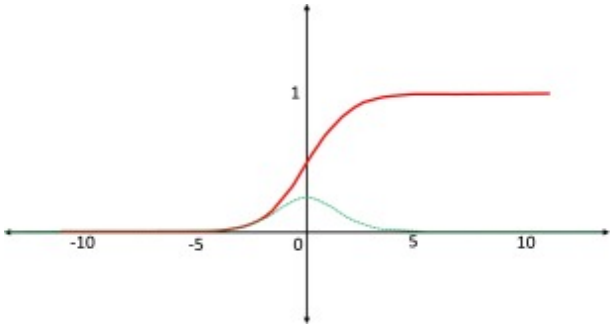


Figure 3.2: Sigmoid activation function (red) and its derivative (green) [35].

The output of the sigmoid is bound between zero and one, and its gradient can be used to push the output either closer to one or closer to zero [35]. It is often used for the output unit for the binary classification task, where the output is desired to be between zero and one [35, 31].

Tanh Next function is the *tanh* activation function, given by the Equation 3.5 and its respective derivative displayed on the equation 3.6.

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.5}$$

$$\tanh'(z) = 1 - tanh^2(z) \tag{3.6}$$

Like the *sigmoid* function, it compresses the input; however, unlike the *sigmoid*, its output is constrained to the range of -1 to 1, as shown in Figure 3.3.

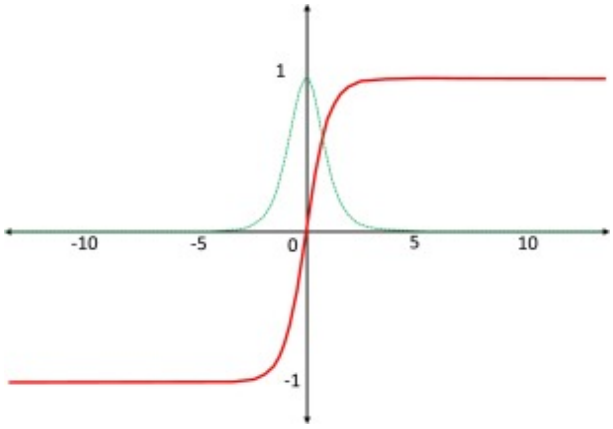


Figure 3.3: Tanh activation function (red) and its derivative (green) [35].

ReLU The problem with *sigmoid* and *tanh* functions is the vanishing gradient and computational complexity. Vanishing gradient means that the gradient of a function is almost flat, hence close to zero, which leads to no or very little update in the network's learnable parameters (weights and biases) during the training [33, 34].

As a possible solution to these problems, a rectified linear unit, also known as ReLU, was introduced [36]. ReLU is a simple function; its Equation 3.7 and derivative Equation 3.8 are straightforward.

$$\mathrm{ReLU}(z) = \begin{cases} z, & \text{if } z > 0, \\ 0, & \text{if } z \leq 0. \end{cases} \tag{3.7}$$

$$\mathrm{ReLU}'(z) = \begin{cases} 1, & \text{if } z > 0, \\ 0, & \text{if } z \leq 0. \end{cases} \tag{3.8}$$

The ReLU function can be seen in Figure 3.4. It basically returns its input if the input is positive otherwise, it returns zero. Since the derivative of x is always one, the problem with vanishing gradient is solved.

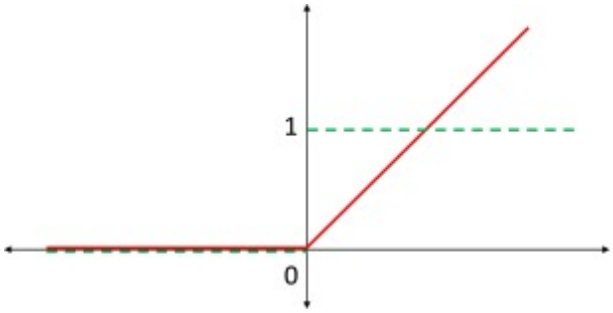


Figure 3.4: ReLU activation function (red) and its derivative (green) [35].

ReLU also introduces some potential drawbacks, i.e., the output for negative input is always zero. The problem called dying ReLU [33, 35, 34] is when a negative input causes no updates in weights during training, and neurons in this state do not respond to error variations [35]. To fix this problem, we can multiply the negative input value by a very small constant, which will allow the weights to be updated if it is needed. This modified ReLU is called Leaky ReLU [37] and its formula and formula of its gradient are displayed in Equations 3.9 and 3.10 respectively.

$$\text{LeakyReLU}(z) = \begin{cases} z, & \text{if } z > 0, \\ \alpha z, & \text{if } z \leq 0. \end{cases} \tag{3.9}$$

$$\text{LeakyReLU}'(z) = \begin{cases} 1, & \text{if } z > 0, \\ \alpha, & \text{if } z \leq 0. \end{cases} \tag{3.10}$$

In addition to the Leaky ReLU, many other ReLU variants were introduced over the years, each bringing its own advantages, disadvantages, and challenges [33, 34].

Nowadays, the most commonly used activation function for hidden units is the ReLU activation function [33, 31, 1].

3.1.2 Layers

Similarly to biological neural networks, when artificial neurons are chained together, meaning the output from one neuron is passed to another neuron, they create an artificial neural network.

This network is organized in layers. Neurons in each layer are not connected together, but rather every neuron from layer L is connected with every neuron from layer $L+1$, except neurons in the first (input) layer. For better understanding, we will refer to the Figure 3.5, where we can see an example of a neural network.

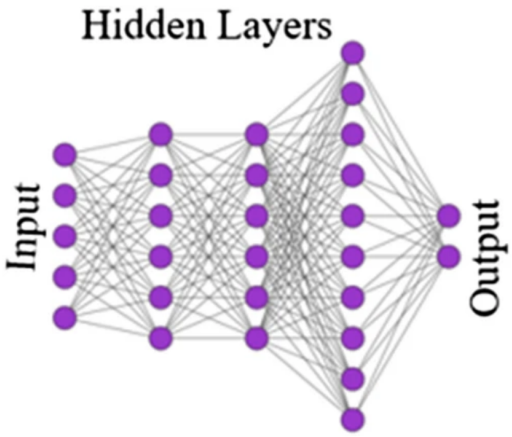


Figure 3.5: Example of deep artificial neural network [38].

A neural network can be divided into three main parts:

- Input layer
- Hidden layers
- Output layer

Input layer is the initial layer and the only layer that does not contain neurons which perform calculations but rather consists of N input features $x_1, x_2, \ldots, x_N$ also referred to as a vector $\vec{x}$ of input features displayed in equation 3.11.

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \tag{3.11}$$

The subsequent layers between the input layer and output layer are called hidden layers. The name comes from the fact that their outputs are not directly observable, nor are they provided by the external environment - they are internal to the network's architecture. Neurons inside these layers perform calculations on the input and produce output, which is then fed forward to the next layer [1].

The final output layer produces the output of the network. Output and number of neurons depend on the task the network is being trained for. For regression tasks, one neuron is often suitable - it predicts a continuous variable [31]. During classification tasks, it can further depend on the nature of the classification. In binary classification, again, a single neuron can suffice. It will display a probability of the input belonging to one of the classes - if the probability is high, it will assign that class to it, and if the probability is low, it will assign the other class to it [31]. In multi-class classification, the number of neurons is the same as the number of classes, and each neuron predicts a probability of the input belonging to one specific class [31].

3.2 Loss Functions

Loss function, sometimes also referred to as cost function, is a function that computes the difference between the result predicted by the model and the ground truth. This difference is called an error. The error guides the model during training and is responsible for parameter updates. We are trying to find the local

minimum of the cost function - a point where the error value is as low as possible, because this means that the model is making good predictions. Hence, we are trying to find a global minimum of the cost function. Similarly to different activation functions, there is also a variety of cost functions.

Mean Squared Error Mean squared error (MSE) is computed as the sum of all differences between predicted output and real values (ground truth) raised to the power of two. The Equation 3.12 displays this computation, where m is the number of input samples, y is the ground truth, and $\hat{y}$ is the output predicted by the model. Despite being effective for regression problems, MSE is not suitable for classification problems [35].

$$E_{\text{MSE}} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{3.12}$$

Cross-entropy Loss Much more efficient loss functions for classification problems are the entropy-based ones. For example, for binary classification, a logistic loss function, by which a binary cross-entropy error (BCE) is measured, is suitable [35]. It is given by the Equation 3.13, where m is the number of input samples, y is the ground truth, and $\hat{y}$ is the predicted output.

$$E_{\text{BCE}} = -\frac{1}{m}\sum_{i=1}^{m}(\hat{y}\log y + (1-\hat{y})(\log(1-y))) \tag{3.13}$$

This function can be modified to compute error for multiclass classification as well - this is also called categorical cross-entropy error (CCE). If we assume we have C distinct classes we want to assign input into (and input can belong to exactly one

class), then the Equation 3.14 computes the error. Here m is the number of input samples, C is a set of classes, $y_{i,c}$ is the ground truth for the i-th sample and c-th class, usually represented as a one-hot encoded vector where $y_{i,c} = 1$ if the i-th input belongs to the c-th class, and $y_{i,c} = 0$ and $\hat{y}_{i,c}$ is the predicted probability for the i-th sample belonging to the c-th class.

$$E_{\text{CCE}} = -\frac{1}{m}\sum_{i=1}^{m}\sum_{c=1}^{C}(y_{i,c}\log\hat{y}_{i,c}) \tag{3.14}$$

Dice Loss The most popular choice for the object segmentation task, and especially in the medical imaging domain, is the Dice loss function [39]. It uses the Dice similarity coefficient (DSC) to compute the difference between predicted map p and ground truth map y for each class j of C classes. A slight problem exists with the DSC - it is not differentiable, therefore it cannot be used directly in training. To overcome this obstacle, neural networks use a probabilistic version of DSC to the discrete DSC in training [39]. Its computation is displayed in Equation 3.15 where N depicts the number of pixels and ϵ is a small constant used to avoid division by zero. The computation of overall dice loss is displayed in Equation 3.16, where D_i is the DSC computed for the i-th training sample.

$$\text{DSC}_i = D_i = \frac{2\sum_{n=1}^{N}\sum_{j=1}^{C}(y_{n,j}p_{n,j}) + \epsilon}{\sum_{n=1}^{N}\sum_{j=1}^{C}(y_{n,j} + p_{n,j}) + \epsilon} \tag{3.15}$$

$$E_{\text{DiceLoss}} = \frac{1}{m}\sum_{i=1}^{m}(1 - D_i) \tag{3.16}$$

3.3 Training

During the training phase, a neural network tries to minimize the cost function by adjusting its parameters - weights and biases. This process is often called learning, and we can say that the neural network learns to map input features onto the desired output. Before the training process, we need to ensure that the data is of the desired quality and quantity; otherwise, the training will not be effective, and the performance of the resulting model will be poor. Methods such as data preprocessing are typically used [31, 1].

The training itself consists of multiple steps:

1. Parameter initialization
2. Forward propagation
3. Cost function computation
4. Backpropagation and parameter updates

Parameter initialization During parameter initialization, we set the initial values of all learnable parameters of the network (parameters that can be updated during training). Different weight initialization strategies were developed; their overview can be seen in Figure 3.6.

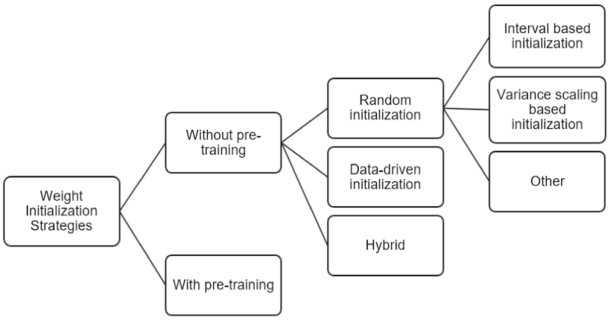


Figure 3.6: Different weight initialization strategies [40].

Methods such as Xavier/Glorot or He initialization are also popular [41].

Forward propagation During forward propagation, the input sample data is fed forward through the layers of the network. For a hidden layer L with N neurons, and activation function φ:

- The input is output from previous layer *L-1* (the activations), denoted: $A_{L-1} \in \mathbb{R}^{m \times d}$, where m is the number of input samples, and d is the number of input features of each sample. Number d is also equal to the number of neurons present in layer *L-1*.
- Each neuron needs to have d weights, one for each input feature. A weight matrix holding weights of all neurons in layer L can then be denoted as: $W_L \in \mathbb{R}^{d \times N}$.
- Each neuron also holds a bias term, all biases in layer L can be represented with vector $b_L \in \mathbb{R}^N$.
- Output matrix (the activations) returned by this layer can be denoted as: $A_L \in \mathbb{R}^{m \times N}$. This becomes input for the layer *L+1*.

A computation performed by an arbitrary hidden layer L with N neurons can be calculated with Equations 3.17 to compute $Z_L \in \mathbb{R}^{m \times N}$ pre-activation values, and 3.18 - to compute output of layer L. Function φ is applied element-wise on all elements of the input matrix.

$$Z_L = XW + b \tag{3.17}$$

$$A_L = \varphi(Z) \tag{3.18}$$

The final output layer will then return the predicted output value for each sample. At the beginning of the training, the output values will be almost random, but as the training continues, the predicted values should converge towards the ground truth values - this is the desired behavior [31, 1].

Cost function computation After the sample (or samples) are fed forward through the network, we get either a matrix or vector of output values or a single output value. The next step is to compute the error of the network using one of the aforementioned cost functions, e.g., dice loss in the case of image data. The error is then propagated backwards through the network, and weights and biases are adjusted in a way that will minimize the cost function. This algorithm is called backpropagation [31].

Backpropagation For a network to learn, it should implement some kind of algorithm that will adjust its learnable parameters (weights and biases) in a way that the overall error will be lower next time the input samples are fed forward through the network. Backpropagation computes the gradients for each layer starting with the output layer, and by utilizing the chain rule of calculus, it propagates the error back through the network to the first hidden layer [1].

After the gradients are computed, the parameters are updated accordingly by optimization algorithms such as stochastic gradient descent [35]. Formulas for updating weights and biases are shown in Equations 3.19 and 3.20, where α is a learning rate (which controls the speed of learning - if changes to the parameters are too great, the minimum can be missed, if changes are too small, the minimum will not be reached in a reasonable time) and E is the cost function [31, 1].

$$w = w - \alpha \frac{\partial E}{\partial w} \tag{3.19}$$

$$b = b - \alpha \frac{\partial E}{\partial b} \tag{3.20}$$

There is a strict rule for backpropagation to work - all functions used in the network must be differentiable at all points [26, 35].

3.3.1 Optimization and Regularization

Optimization Many optimization techniques are capable of further enhancing the model training and performance. Examples include:

- using small batches of input samples, and after each batch passes, perform parameter updates,
- utilizing momentum [42] to have more control over the learning speed based on the previous gradients [35],
- using adaptive learning rates, where the learning rate α is usually great at the beginning, and as the training progresses, it is gradually reduced [35]. Updates to α can be done after some number of iterations by some preset factor or automatically by utilizing methods such as Adam (adaptive moments) [43] or RMSProp [44].

Regularization Another set of techniques that can improve model performance is regularization. Usually, a model's performance and prediction capabilities improve during training. Available data are often split into three subsets for training, validation, and testing. The model is trained using the training subset. The validation subset is used to check model performance during training, and the test

subset is used for the final evaluation of the model. It is important that both validation and test subsets contain samples the model has not yet seen during training - otherwise, the results would be biased. A good model should be robust and generalize well, not only learn patterns that are specific to training data. Sometimes, especially in more complex models, we can observe an effect when, at the beginning of the training, both training and validation performance (such as error value) improve, but later the validation performance plateaus or worsens - this effect is called overfitting [45] and is displayed in Figure 3.7.

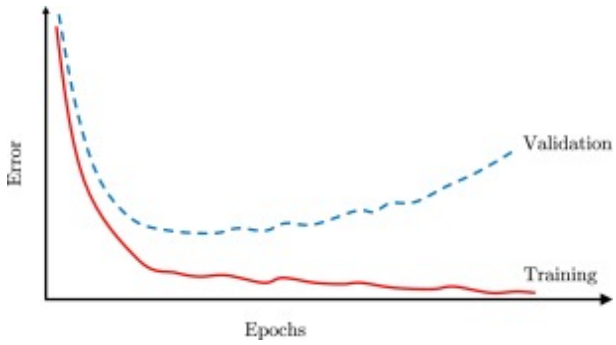


Figure 3.7: Error curve during training - overfitting happens [35].

This effect is not desired because it means that the model cannot generalize well on previously unseen samples - it is learning "by heart" from the training data. To overcome this problem, we can implement several regularization mechanisms that will improve the model's robustness and ability to generalize. Examples include dropout [46], transfer learning [47], early stopping [48], parameter norm penalties such as L1 regularization (lasso regression) and L2 regularization (ridge regression) [49], and more [35].

3.4 Evaluation Metrics

When evaluating a model's performance, different metrics exist that can be used. The evaluation metrics also depend on the task the model was trained for. During classification tasks, depending on the predicted and real values for each sample, we can differentiate four groups of results:

- True Positives (TP) - a model assigns a sample to class c when a sample belongs to class c,
- True Negatives (TN) - a model does not assign a sample to class c when a sample does not belong to class c
- False Positives (TP) - a model assigns a sample to class c when a sample does not belong to class c
- False Negatives (TN) - a model does not assign a sample to class c when a sample does belong to class c

We will briefly describe some of the evaluation metrics in the following paragraphs.

Accuracy, Precision, Recall, and F1-score Calculation of these basic metrics is displayed in Equations 3.21, 3.22, 3.23, and 3.24. They describe the relationships between the number of samples belonging to either the true positive (TP), true negative (TN), false positive (FP), or false negative (FN) groups.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.21}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.22}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.23}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.24}$$

Area Under the ROC Curve Receiver Operating Characteristics (ROC) Curve and area under it can be used as another evaluation metric for classification and is superior when compared to overall accuracy [50]. The ROC Curve is drawn in the ROC Space as a relationship between the True Positive Rate (TPR, Recall or Sensitivity) and False Positive Rate (FPR) at the different threshold levels [50, 51, 52]; the calculation of TPR and FPR is shown in the Equations 3.25 and 3.26.

$$\text{FPR} = \text{Specificity} = \text{Recall} = \frac{TP}{TP + FN} \tag{3.25}$$

$$\text{FPR} = 1 - \text{Specificity} = \frac{FP}{FP + TN} \tag{3.26}$$

The ROC Space and an example of ROC Curve are displayed in Figure 3.8. Area under this curve (AUC - Area Under the ROC Curve) is then computed and interpreted:

- if AUC = 1, this is the perfect model
- if AUC = 0.5, model capability is equal to random guess

- if AUC < 0, performance of the model is worse than the random guess

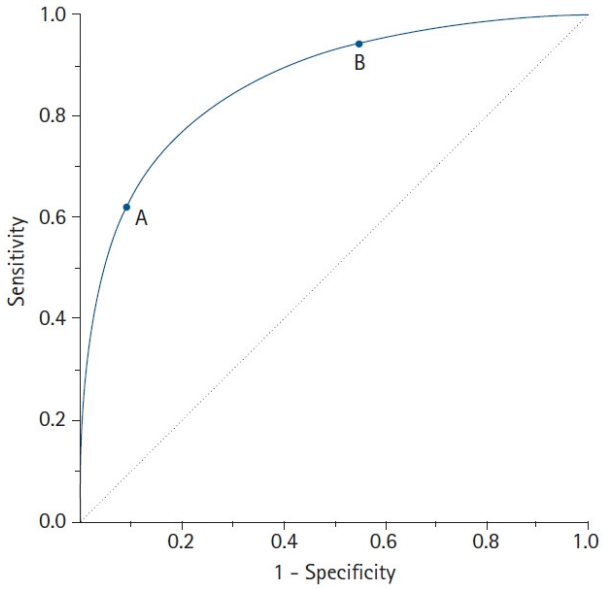


Figure 3.8: ROC Space with ROC Curve [51].

Intersection over Union Intersection over Union (IoU), also known as the Jaccard Index, is a widely used evaluation metric in image segmentation and object detection [53]. It is computed as the area of overlap between the predicted and ground truth regions divided by the area of their union. The closer the resulting value is to 1, the better the model predictions are [53]. Its computation is shown in Equation 3.27, where y is the true area and $\hat{y}$ is the area predicted by the model.

$$\text{IoU} = \frac{y \cap \hat{y}}{y \cup \hat{y}} \tag{3.27}$$

Dice Coefficient Similarly to the IoU, the Dice Coefficient is used for image segmentation and detection tasks [2], and the closer the resulting value is to 1, the

better the model predictions are. Its computation is given by the Equation 3.28, where y is the true area and $\hat{y}$ is the area predicted by the model.

$$\text{DiceCoefficient} = \frac{2|y \cap \hat{y}|}{|y| + |\hat{y}|} \tag{3.28}$$

3.5 Architectures

Neural network architectures like Convolutional Neural Networks [27] and U-Net [21] have proven to be effective in medical image analysis [35]. In recent years, a concept of Vision Transformers [54, 2] used in medical imaging shows promising results [55, 2, 3]. In further sections, we describe each architecture and its contribution to the analysis of medical images and Digital Pathology.

3.5.1 Convolutional Neural Networks

In 1959, Hubel and Wiesel conducted experiments that inspired the advent of the Convolutional Neural Networks (CNN). In their experiments, they put a microelectrode into a cat's brain (into the part called the primary visual cortex), while it was under partial anesthesia. While showing various images to it, they measured the neurological activity of the cortex [56]. According to the results, a hierarchical pattern can be observed in the activity of the visual cortex, where the neurons close to the retina captured the simplest patterns (like different illuminations and lines under various angles) and the farther layers captured more complex patterns (like geometric shapes and other complex visual patterns) [56].

CNN took advantage of these findings and rebuilt the classical neural network layers to be able to capture more complex features with increasing depth. They

utilize so-called convolution layers along with the ReLU activation function to learn to extract relevant features from the image. The deeper the convolution layer, the more complicated features it can learn [35].

Similarly to the Hubel and Wiesel cat's visual cortex, the first layers can learn to identify basic shapes like lines and simple geometric shapes, and the deeper layers can learn to identify more complex ones. Such an example of CNN is shown in Figure 3.9.

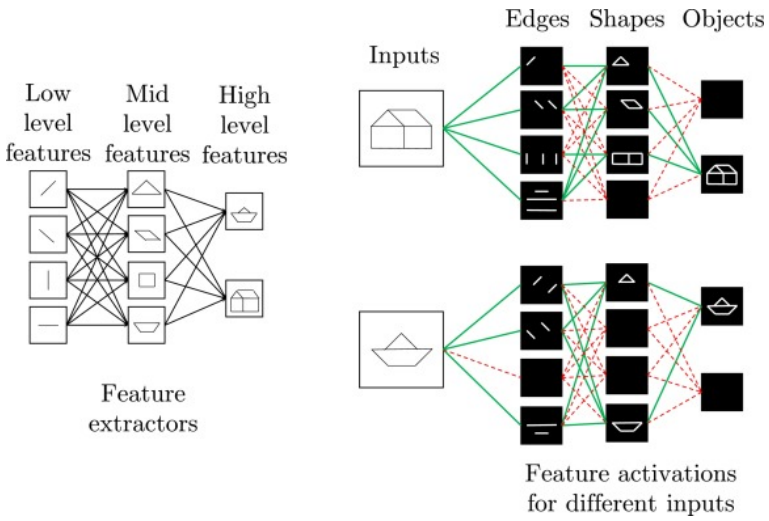


Figure 3.9: Example of CNN learning strategy [35].

A convolution layer applies a series of operations on its input and produces an output map. The most important part is applying a kernel, which is a tensor of fixed width and height, over the input image or output map from the previous convolution layer. This fundamental operation serves as a feature-extracting technique. The kernel slides over its input across its height and width, and at each step, it performs element-wise multiplication of the pixel values it currently overlaps at each layer of depth and then sums them together to produce a single value. So, for example, if the input is of size $100 \times 100 \times 3$ (standard RGB image with 3 channels for red, green, and blue color), then the kernel is applied to all

three channels simultaneously. After the kernel is applied to the whole image, the resulting output map will be of size $100 \times 100 \times 1$ (assuming that other hyperparameters of convolution are configured in such a manner that the original width and height remain unchanged for the output map - we will cover them later in this chapter), because the kernel will collapse its depth.

The kernel filters can be handcrafted to multiply and intensify certain properties of the image. Examples include the Prewitt, Gabor, Sobel, Laplacian, and Roberts filters for edge and gradient detection. In standard image processing, the weights inside the kernel are preset. However, in the CNNs, these weights are learned during training, so the network determines what features of the image the output maps will be focused on, and hence the network can be more effective [35, 3]. In the AlexNet [57], the first deep CNN which outlined the original structure, a ReLU activation function was applied to the value obtained from the convolution operation to break the linearity of the operation. Since then, using an activation function after the convolution operation has become a standard practice [35, 3], and the name of the output produced by the convolution + ReLU is also called an activation map.

According to the [35], convolution operation can be expressed as a function with hyperparameters: $\varphi_{\text{conv}}(C_{\text{in}}, C_{\text{out}}, K, S, P, D)$. The definition of these hyperparameters is:

- C_{in} is the number of channels of the input map - its depth.
- C_{out} is the number of channels of the output produced by the layer - it is also the number of filters that will be applied to the input map since one filter produces an activation map with one channel.
- K is a tuple that defines the size of the kernel - its width (k_w) and height

(k_h)

- S is also a tuple, which defines the stride - the number of pixels the kernel will slide along, both in terms of width and height.
- P can also be a tuple and it defines the number of added dummy pixels to artificially increase the input map size for the output map to keep the same size as the input map (otherwise the output map would be smaller since the kernels cannot slide outside of the boundary of the input map).
- D (a tuple as well) is the dilation, and it serves the purpose of increasing the field of view of the kernel (the area of the image it can cover) without adding more weights to it. Dilation defines the gap that is added both horizontally and vertically between the weights of the kernel.

We can see an example of the convolution in the Figure 3.10.

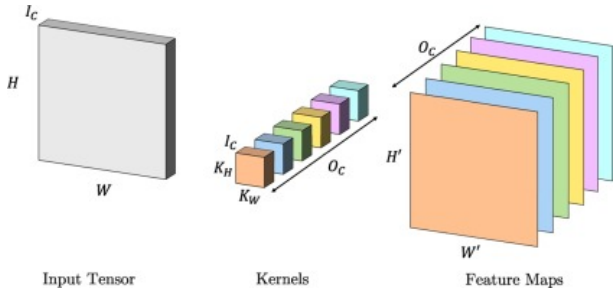


Figure 3.10: Example of the convolution operation [35].

The field of view of a single kernel is small, as typically kernels of size 3×3, 5×5, or 7×7 are used [35]. To address this issue and to be able to build up and capture more complex features in the subsequent layers, the pooling layer is often added. The pooling layer effectively downsamples the feature maps, commonly by a factor of two. This allows the next convolution layers to learn more abstract features that were further apart in the previous feature map. To compensate for

the information lost during the downsampling, usually a number of independent kernels is increased for the next convolutions after each pooling layer. During pooling, a small array slides over the input map and always selects only a single value from the area it covers, hence decreasing the size of the map. Two pooling methods are common:

1. Max pooling selects the maximal value from the area it covers, and
2. Average pooling computes the mean from the values it covers.

Example of pooling, both max and average, can be seen in the Figure 3.11.

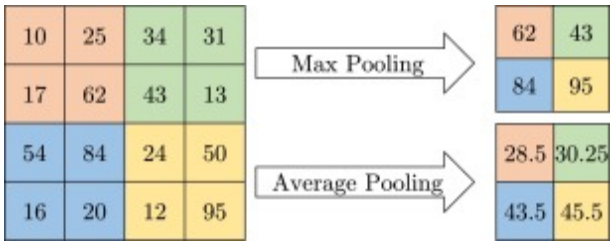


Figure 3.11: Example of max and average pooling operations [35].

CNNs are usually organized as repeating layers of convolutions, followed by the ReLU activation function, and then the pooling layer. The output can then be fed into another convolution, and a deep CNN can be built using this approach. There is a rule of thumb [14], where we start with a small number of independent filters with a small field of view, and then we downsample the image by the factor of two (to increase the field of view) and double the subsequent number of independent filters as can be seen in Figure 3.12.

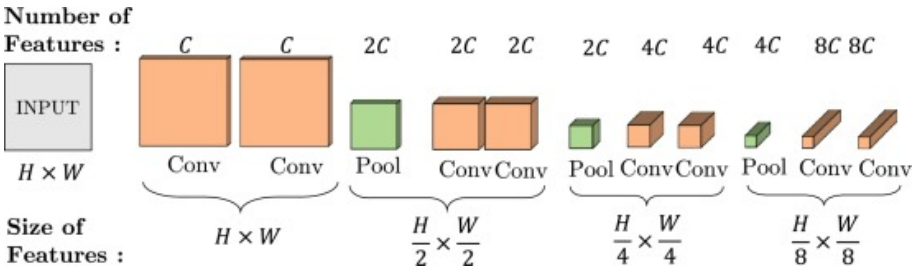


Figure 3.12: Scale space pyramid of a CNN [14].

In CNNs with very large depth, a concept of a skip connection, first introduced in the [58], can be used. A skip connection allows the input of a layer to bypass the convolutions and then be added to their result, as shown in figure 3.13. Also, if we can represent a series of convolutions with a function $\varphi(x)$, then the output with skip connection included would be given by $y = \varphi(x) + x$. This residual learning architecture can mitigate the problem of vanishing gradient in very deep CNNs [35].

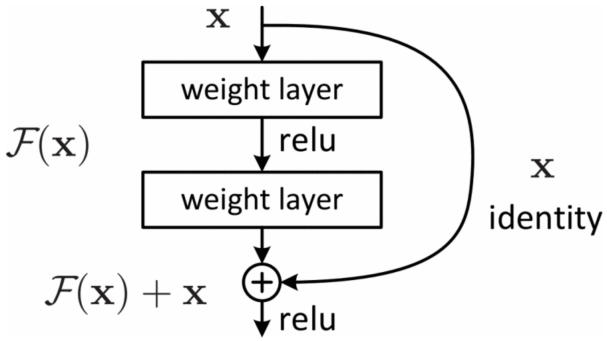


Figure 3.13: A building block of residual network [58]

Other strategies that can improve the performance of CNNs include batch-normalization layers [59](which are inserted after the convolution layers) and the utilization of parallel branches [60] (which use different kernel sizes to compute multi-scale feature maps by concatenating their output maps).

After the series of convolutions and downsampling, the final compressed feature representation of the image needs to be flattened (converted into a vector) so it can serve as an input for the fully connected layer(s), which will then make the appropriate decision based on the task the model should do. Two methods exist [35], which we can use to convert the image descriptor into a vector:

1. Reshape the activation maps to form a one-dimensional tensor [57, 27].
2. Use average pooling on the entire activation map, which will collapse the entire information into a single value and then concatenate these values to form a vector. For N activation maps, we will get a vector with N elements [58, 60].

We can see their visual representation in Figure 3.14.

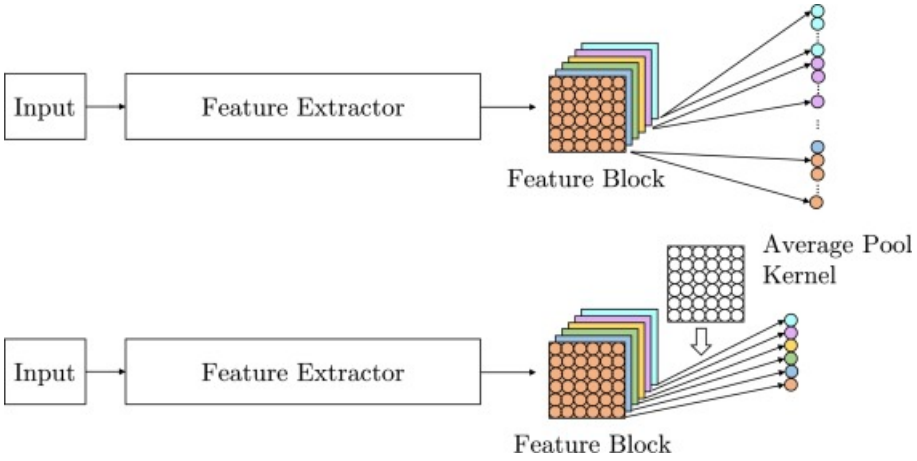


Figure 3.14: Different flattening strategies [35].

An example of a CNN with all layers is in Figure 3.15. The input image features are extracted by the convolution layer with four independent 5×5 kernel filters, followed by the ReLU activation function and max pooling layer. Then the extracted image descriptor is flattened into a vector and fed into fully connected layers, which serve as the classification output head.

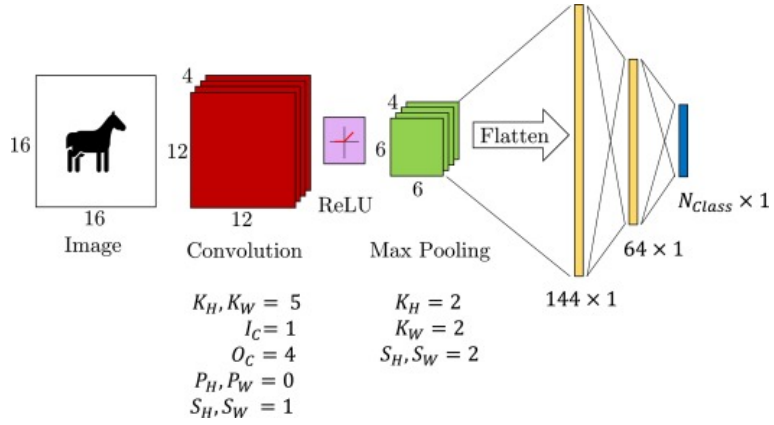


Figure 3.15: Example of CNN architecture [35].

Many CNN architectures introduced new concepts in the field of computer vision and image analysis when they were presented. Considering today's knowledge and advancements, some of them might look trivial, but their contribution should not be overlooked. In the classification task, we can mention:

1. LeNet5 [27]: uses a series of convolution and pooling operations to extract the image features and fully connected layers to classify the input. It was introduced as a model that should recognize handwritten digits.

2. AlexNet [57]: built on top of the LeNet, utilized a deeper architecture along with ReLU activation functions, and had immense success at the ImageNet Large-Scale Visual Recognition Competition. This success brought huge attention to the CNNs and their potential in image analysis-related tasks.

3. VGGNet [61]: introduced a strategy of expandable convolutional blocks, where in each block a varying number of convolutions can be used. Each block is then followed by the max pooling layer. This allowed for tailoring the network to reflect the complexity of the problem it was assigned to solve.

4. GoogleLeNet [60]: used kernels of varying sizes to extract features from the

input maps and then concatenated their output maps to create a depth-wise combined feature. It also used the full-scale average pooling in the final feature extraction layer to create a one-dimensional tensor. Furthermore, it utilized auxiliary classifiers in the intermediate layers to boost the gradient flow. The later version, called Inception Net, introduced more concepts, like kernel factorization and batch normalization [62].

5. ResNet [58]: introduced a strategy of skip connections to solve the problem of vanishing gradient in very deep CNNs.

In the detection task, two main architectures were created, namely the Region-based CNN [63] (further upgraded to the Faster R-CNN and Mask R-CNN [64]) and the You Only Look Once (YOLO) [65].

Both R-CNN and YOLO can localize objects in an image with a bounding box label. However, sometimes we want to obtain a more precise location of the object, and this is where segmentation comes into play.

3.5.2 U-Net and Its Variants

To perfectly localize an object in an image, we would like to construct a pixel-level mask that would mark the pixels where the object is present. We can use classification for this - each pixel can either be classified as one that does or does not display a part of the object. Segmentation algorithms have a deep impact and huge potential for medical imaging and digital pathology, where they can be used to mark different tissue types, organs, cells, and tumor regions [14].

The limitation of full CNN architectures is the inability to preserve the spatial information after the initial feature-extracting layers [14]. This issue is addressed in a new type of architecture, the encoder-decoder architecture. This architec-

tural type, also known as the autoencoders or auto-associative networks, was first introduced in the [66]. Its visual representation is shown in Figure 3.16.

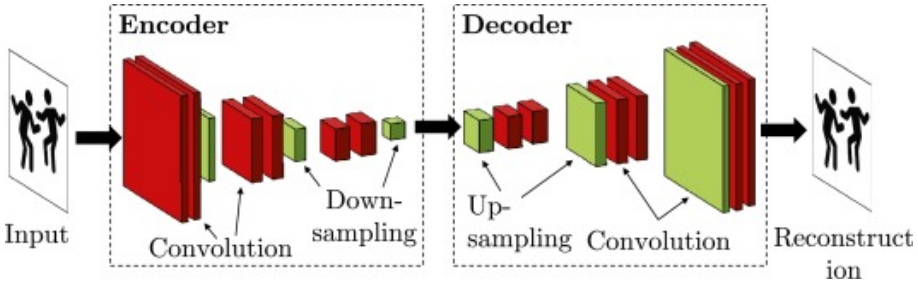


Figure 3.16: Example of the autoencoder architecture [35].

It consists of two main parts:

1. The encoder part, which is responsible for encoding the input image and extracting the most descriptive features, compressing them, and reducing the redundancy, and
2. The decoder part is responsible for reconstructing the original input image from the compressed image descriptor.

The loss function computes the difference between the original input image and the image constructed by the decoder. If the decoder is able to create an image that looks very similar to the original, it means that the hidden representation of the image extracted by the encoder is credible enough. Then the decoder part can be removed and instead, a classification, segmentation, or localization module can be attached and exploit the features learned by the encoder [35].

In segmentation tasks, a simple change is added to the decoder part. Instead of generating the original image, it is trained to create the segmentation mask, where each pixel has a probability of belonging to a certain class. This computed probability distribution mask is used along with the original mask in the loss function to compute their difference and guide the training.

When the activation maps are downscaled in the encoder using operations such as max pooling, where only a single value is picked, we need a correct mechanism to reconstruct the original spatial position of that value during the upscaling of the maps. This can be achieved by remembering and forwarding the indices of the chosen value, as it was first introduced in the SegNet architecture [67].

One of the most widely adopted and impacting architectures is the U-Net model [21]. U-Net was designed as a model for medical image segmentation tasks and achieved great success in doing so [14, 68]. Its architecture can be seen in Figure 3.17.

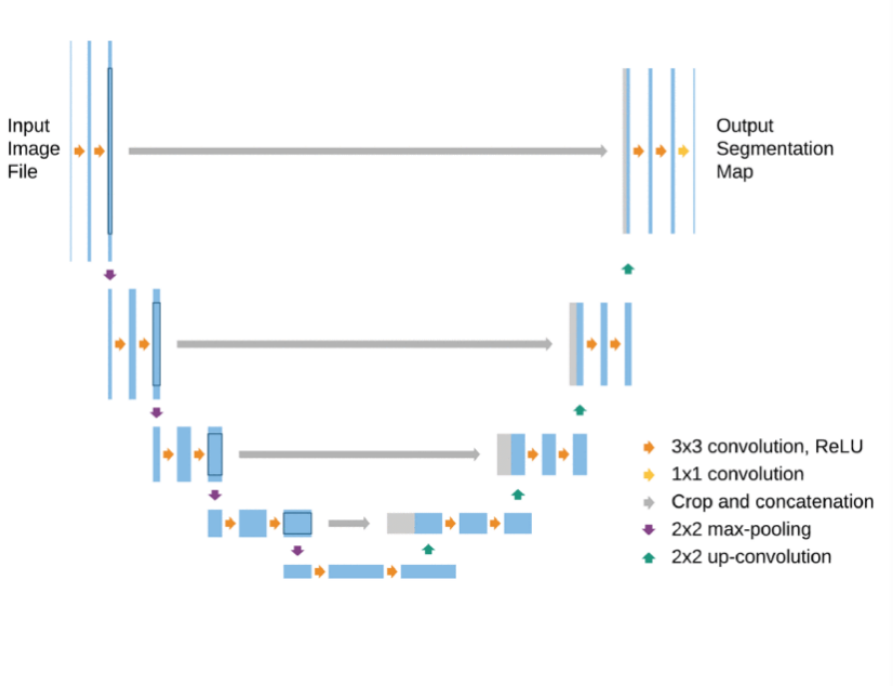


Figure 3.17: U-Net architecture [68].

Similarly to autoencoders, it has two main parts, the encoder and decoder. The encoder works as a classical CNN feature extractor, with convolutions, ReLU activation functions, and max pooling. In each layer, there are two convolutions (kernel size is 3×3), followed by the ReLU, and the resulting activation maps are

downsampled by a factor of two using the max pooling layer (with 2×2 matrix). Before the downsampling happens, the copy of the activation maps is sent to the decoder (skip connections). Every next block doubles the number of channels. This is repeated four times until we reach the bottleneck, where we again perform the convolutions with ReLU, but omit the downscaling. At this point, the features of the input image are in their most compressed form. Output from the bottleneck is then sent to the decoder part. Next, the decoder part uses up-convolutions (kernel size is 2×2), to expand the feature maps (double their size) and halve the number of channels. Particularly, these up-convolutions are distinctive when comparing U-Net to other architectures [68]. The activation map is upscaled by up-convolution, and then the activation maps, previously sent from the encoder, are concatenated with it. The concatenation is a required step since the border pixels are lost in every convolution (the convolutions do not use padding), and also to reintroduce some information that might be lost during the downsampling. Then again, two 3×3 convolutions with ReLU are applied. This is also repeated four times, to reflect the encoder blocks. This approach can be visually drawn into a U-shape-like architecture, from which the U-Net derived its name. Finally, after the last decoder block, the 1×1 convolution is used to get the desired number of channels.

Since the original U-Net, many different variants of it have been introduced [68]. To list a few examples:

- 3D U-Net [69]: works as a classical U-Net but was modified in a way that it can segment 3-dimensional data. Every 2D operation (2D convolution, 2D pooling, 2D up-convolution) was replaced by its corresponding 3D equivalents. It can be useful in medical images that utilize 3D space, like MRI and CT.

- Attention U-Net [70]: utilizes the attention gate to draw the attention of the network to the important parts of the image. These attention gates are inserted in a place where the concatenation of encoder feature maps and decoder feature maps should occur, as can be seen in Figure 3.18. Before this concatenation happens, both sets of feature maps are run through the attention gate, where a series of operations is performed. These operations are visualized in Figure 3.19. Firstly both sets are run through a $1 \times 1 \times 1$ convolution to align their dimensions and then are added together. Then they pass through the ReLU activation function, $1 \times 1 \times 1$ convolution layer to reduce their depth to 1, the sigmoid activation function to squeeze the values between 0 and 1, and an optional resampler to correctly align the spatial dimensions. This results in an attention map containing values between 0 and 1 and with a depth of 1. This attention map is then broadcast and multiplied by the feature maps from the encoder - this produces the final output of the attention gate, which is then concatenated with the feature maps upsampled by the decoder.

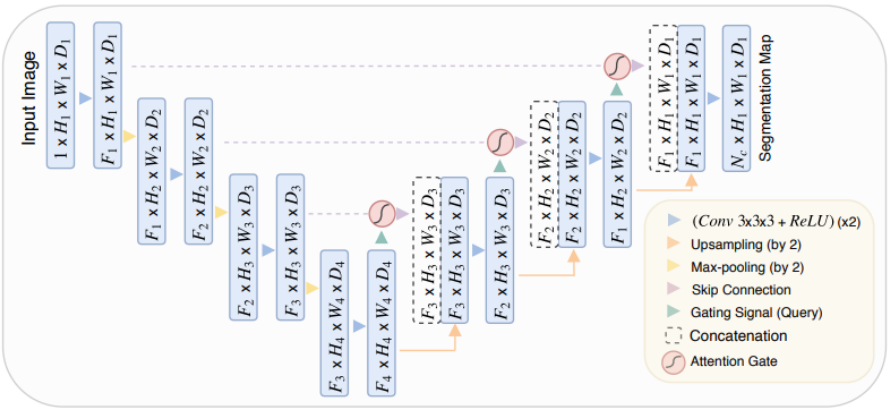


Figure 3.18: U-Net architecture with added attention gates [70].

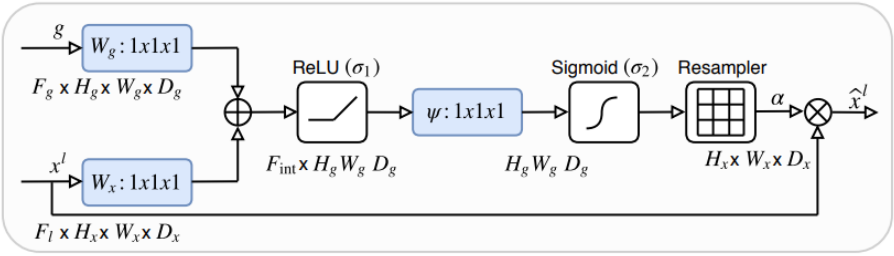


Figure 3.19: Additive attention gates [70].

- Residual U-Net is inspired by the [58] and uses the skip connections within each block to help the gradient flow and address the problem of vanishing gradient.

Many more U-Net variants exist that we do not explain further in this work, for example, the Inception U-Net, Recurrent U-Net, Dense U-Net, Adversarial U-Net, Ensemble U-Net, and U-Net^{++}, each showing potential in various medical imaging domains, from CT and MRI scans to radiology, cytology, and histology [68].

3.5.3 Vision Transformers

CNNs were for a long time considered the dominating architectural pattern in deep learning tasks related to visual data, similar to the RNNs being dominant in sequential data processing, such as natural language processing (NLP) [71]. When the Transformer architecture was proposed in [71], things began to change. Nowadays, transformer-based architecture is prevalent in the NLP field [55], and with the introduction of Vision Transformer in [54], it seems that transformers can be applied in computer vision as well and potentially compete with CNNs [2].

Vision Transformers (ViTs) build on the success of Transformers in NLP tasks [54, 55]. Transformers utilize the attention mechanism that is able to capture a

global context of the input data and is not limited by the distance of the pixels, as was the case with CNNs [55]. ViTs took advantage of the standard transformer encoder part, which can be seen in Figure 3.20. The encoder connects multiple attention blocks to make use of the global image context.

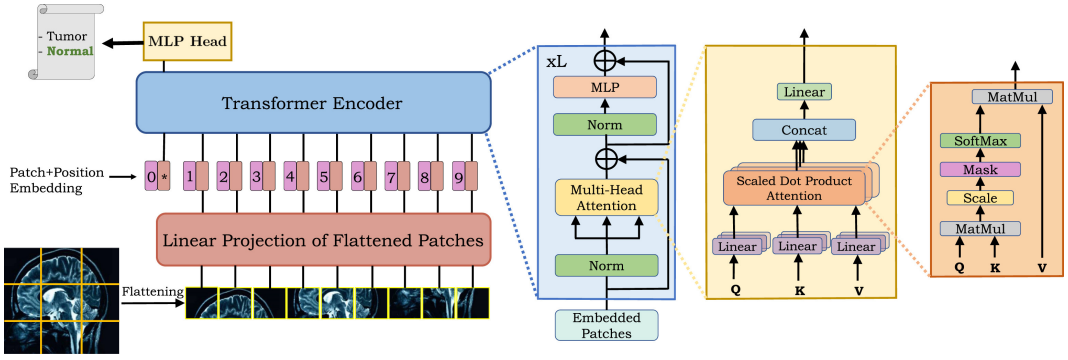


Figure 3.20: Vision transformer architecture for classification [55].

Below, we briefly mention the ViT algorithm as described in [55]:

1. The input image is split into patches with fixed sizes, for example, authors in [54] used a patch size of 16×16 pixels
2. Image patches are converted by flattening into the vector space
3. To reduce the dimensions of the resulting embeddings, vectorized patches are run through a trainable linear layer
4. Since transformers are not aware of the spatial information, positional information is added to each vectorized embedding
5. This sequence is then fed into the encoder
6. Since ViTs require a huge amount of data to perform well, it is a great idea to pre-train the ViT on a large dataset and then fine-tune it to the specific task.

7. In classification tasks, an extra embedding is added, which will be learned during training - the class embedding.

The self-attention mechanism and multi-head self-attention are the key components of ViT's success. The self-attention mechanism allows the model to figure out the importance of a patch embedding with respect to all the other patch embeddings of the image. The multi-head self-attention is composed of multiple self-attention units (also called heads), where each head is independent of the other heads. In the end, their outputs are stacked onto one another and passed through another linear layer. Skip connections facilitate better gradient flow and are added after the multi-head attention unit and before the final output. The produced output can serve as input to another attention block, hence a deep network can be constructed. This allows the model to capture complex relationships and dependencies across the input embeddings [55].

Vision Transformers have achieved great success along with CNNs in many applications of the medical imaging domain in tasks such as medical image classification, segmentation, restoration, synthesis, medical object detection, and more [55].

Chapter 4

Related Work

Precise manual cell annotation on huge WSI slides is a laborious task that needs to be performed by skilled expert pathologists. There exists a large number of models trained on the pixel-level masks for cell segmentation, which perform remarkably well. In the field of weak supervision for cell segmentation, a number of studies focus either on weak supervision in the form of point annotations in H&E slides or weak supervision with bounding box annotations of cells in microscopic imaging or DNA cytometry. However, we did not find many studies focusing on weakly annotated cell segmentation, especially from the histopathological H&E-stained slides, when annotations were presented in the form of bounding boxes. Furthermore, our task is specific in the fact that we are not interested in the segmentation of all cell nuclei, but only in segmenting nuclei of lymphocytes. Therefore, to comprehensively review the current state of research, we will first examine studies that utilize bounding box cell annotations in histology. Subsequently, we will explore selected papers focusing on cell annotations using bounding boxes in modalities other than histology, as well as those addressing weakly supervised cell segmentation in histology employing point annotations of cell nuclei.

4.1 Guided Prompting in SAM for Weakly Supervised Cell Segmentation in Histopathological Images [72]

The authors of this work explore the applicability of the Segment Anything Model (SAM), using guided prompting, to the cell segmentation task from histology image slides, where the cells are only annotated using bounding box labels. Their results outperformed other models for weakly supervised segmentation by a huge margin.

Three different datasets were used, and since each dataset was annotated with pixel-level masks, these were converted into the bounding boxes for the purpose of this study and the segmentation mask labels were not used during the training. If the dataset also contained class labels for individual cell nuclei, these labels were not used as this study is not concerned with cell classification. The datasets used were:

1. ConSep dataset, containing 41 H&E stained WSIs, each having 1000×1000 pixels. The images are of single cancer and colorectal adenocarcinoma. Together, there are 24,319 annotated cells, split into three different categories (inflammatory, epithelial, spindle). For this work, each image was split into four patches, each patch having 500×500 pixels. Then 98 of these patches were used for training, 10 for validation, and 56 for testing.

2. MoNuSeg is a multi-organ cell segmentation dataset containing 51 H&E-stained images of different organ tissue (stomach, bladder, breast, liver, kidney, prostate, colon). Together, the images have 28,846 annotated cell nuclei. Similarly to the ConSep, each 1000×1000 image is split into four 500×500 patches, 133 of them used for training, 15 for validation, and 56 for testing.

3. TNBC dataset of 50 512×512 WSIs of triple-negative breast cancer tissue. In total, it contains 4,022 annotated cell nuclei. 34 images were used for training, 5 for validation, and 11 for testing.

Two main approaches were used. During the first, called D-SAM, a YOLO object detector was trained to generate the bounding boxes of cells, using a sum of three losses (objectness loss - $\mathcal{L}_{obj}$, classification loss - $\mathcal{L}_{cls}$, and localization loss - $\mathcal{L}_{loc}$). During the inference (the testing, at the time of prediction) the image along with bounding boxes predicted by the detector model were embedded and fed to the SAM to predict the segmentation masks (with bounding box predicted labels as guiding prompts). The second approach, called SAM-S, used SAM as a pseudo-label generator. Both images and corresponding ground truth labels were embedded and fed to the SAM and its output segmentation masks were then used as pseudo-labels to train a separate segmentation model with combined Dice loss ($\mathcal{L}_{dice}$) and binary cross entropy ($\mathcal{L}_{BCE}$). Both approaches can be seen in Figure 4.1.

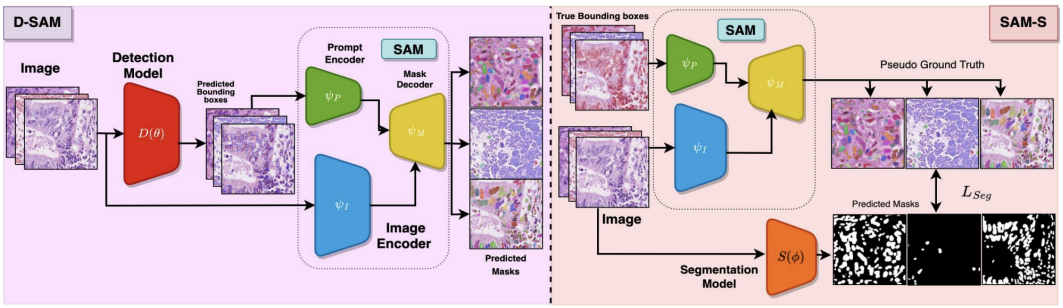


Figure 4.1: Workflows with SAM [55].

In addition to these, three more strategies were used, namely:

1. SAM-W, where SAM was fine-tuned using weakly supervised losses
2. SAM-M, where the mask generated by the SAM-S approach is used as a

guiding prompt for another SAM prediction

3. SAM-ILP, where an Integer Linear Programming is used as a post-processing technique to align the results obtained from both the D-SAM and SAM-S approaches

Different prompting methods were used, and also a no-prompting case, where only an image was provided, was used. In the 1P-kN scenarios, one positive and k negative points were used for each bounding box, where the positive point was the center of the bounding box, and negative points were outside of the bounding box and were not part of any other bounding box. All of the prompting methods are summarized in Figure 4.2, where we can see that the bounding box prompts achieved the best Dice scores in most cases.

Prompt Type	ConSep	MoNuSeg	TNBC
No prompt	31.13	38.96	37.67
1P-0N	37.52	74.16	73.23
1P-4N	56.62	78.54	81.52
1P-8N	61.81	78.00	81.65
1P-16N	59.18	72.97	79.11
Box	**80.00**	79.87	82.80
Box-1P	79.31	79.72	**83.24**
Box-1P-8N	73.39	77.73	81.89
Mask	31.65	35.16	32.78
Mask-Box	80.00	**79.89**	82.12

Figure 4.2: Different prompting methods used for SAM.

Mean average precision (mAP), precision, and recall were used as evaluation metrics for object detection, and the Dice coefficient was used as an evaluation metric for segmentation.

Two non-SAM models were used as baselines for comparison, the BBTP and BB-WSIS, both based on the Residual U-Net architecture, trained for 50 epochs and a learning rate of 0.0001. Yolov8x was used as the object detector, trained for 300 epochs with early stopping, batch size 32, and decreasing learning rate (starting at

0.01 and decreasing by a factor of 10). The CaraNet was used as the segmentation model. It uses reverse axial attention and has great performance for small objects. It was trained for 200 epochs, with Adam optimizer, learning rate 0.0001, and early stopping. The experiments were carried out using NVIDIA-RTX 5000 and Tesla A100 GPUs.

From the results displaying the Dice scores shown in Figure 4.3, we can see that both SAM-S and D-SAM outperformed the baseline models, and the overall best results were achieved by the SAM-ILP model.

Model	ConSep	MoNuSeg	TNBC
BBTP [14]	62.40	72.34	68.53
BB-WSIS[43]	66.00	72.81	70.37
SAM-S	79.86	79.38	80.66
D-SAM	80.00	79.87	82.86
SAM-M	80.50	80.07	82.26
SAM-W (MIL) [14]	72.27	73.58	79.64
SAM-W (BoxInst) [38]	70.25	73.05	79.43
SAM-ILP	**81.39**	**81.45**	**83.58**

Figure 4.3: Comparison of used models with Dice scores.

4.2 A pathomic approach for tumor-infiltrating lymphocytes classification on breast cancer digital pathology images [73]

The second study aimed to classify TILs in H&E-stained images of breast cancer based on a handcrafted set of features, to achieve a better model explainability. Even though the main focus of this study is different from our goals, it is relevant and interesting for us for two reasons:

1. it uses the same TIGER dataset [15] as we do, and

2. it uses a watershed-based method to segment cell nuclei within the tissue as a preprocessing step.

The dataset contains 195 WSIs scanned at three different institutes. They contain region of interest (ROI) annotations of both tissue types and TILs. TILs were annotated using point annotations and a bounding box of 8×8 µm was constructed and centered on the point.

In the preprocessing step, the authors applied a stain normalization proposed in [74] and watershed-based cell nuclei segmentation. The authors decided to use this method for its simplicity, speed, and easy parameter adjustments and fine-tuning. The method used mathematical operations. They used the implementation from the QuPath digital pathology tool with the following set of parameters:

- The setup parameter: hematoxylin OD for the detection image, pixel size of 0.5µm
- Nucleus parameters: background radius 8µm; median filter radius 0µm; σ = 1.5µm; minimum cell area 10µm^2; maximum cell area 400µm^2
- Intensity parameters: threshold 0.1; maximum background intensity 2

The resulting segmentation masks were verified by an expert microscopist. This method was applied to 1037 ROIs, where 92,141 cell nuclei were segmented; 20,111 of them were TILs.

The study further worked on the TIL/non-TIL classification task and identifying the relevant features, but since this is not our primary interest, we only briefly describe the methodology and results. The study analyzed 71 features split into five groups (6 Fourier Shape Descriptors features - FSD, 8 gradient features, 26 Haralick features, 12 intensity-based features, 19 morphometry features). Out of them, 21 were selected as pathomic features. These should best describe the

properties of TILs. Then, five different classification models (Random-Forest, Decision Tree, Linear Discriminant Analysis, K-Nearest Neighbors, Multi-layer Perceptron) with three different resampling techniques (none, synthetic minority oversampling technique - SMOTE, Down) were trained using these features. The AUC (area under the ROC curve), accuracy, precision, sensitivity, specificity, and F1-score were used as evaluation metrics. The Random-Forest classifier achieved the best result with an AUC of 0.86, where the resampling technique did not make a significant difference.

4.3 DDTNet: A dense dual-task network for tumor-infiltrating lymphocyte detection and segmentation in histopathological images of breast cancer [75]

The third work introduces a dense dual-task network (DDTN), which is used both for TIL detection and segmentation in breast cancer H&E-stained images using only point annotations. These two modules share the same backbone, which allows them to learn from one another and promote each other. The ultimate goal of this network is to perform a precise TIL instance segmentation.

The training and testing workflows of the network can be seen in Figures 4.4 and 4.5. During the training phase, the network produces three output types for cells - bounding boxes, cell contours, and cell masks. The separate semi-automatic tool is used to create segmentation masks, from which bounding boxes and cell contours are derived. These are then used to guide the training of the model. During the inference phase, a network is used to produce the aforementioned three types of

output again. Cell masks and contours are then unified to create cell segmentation masks, and these are further merged with the detection bounding boxes to provide the final output.

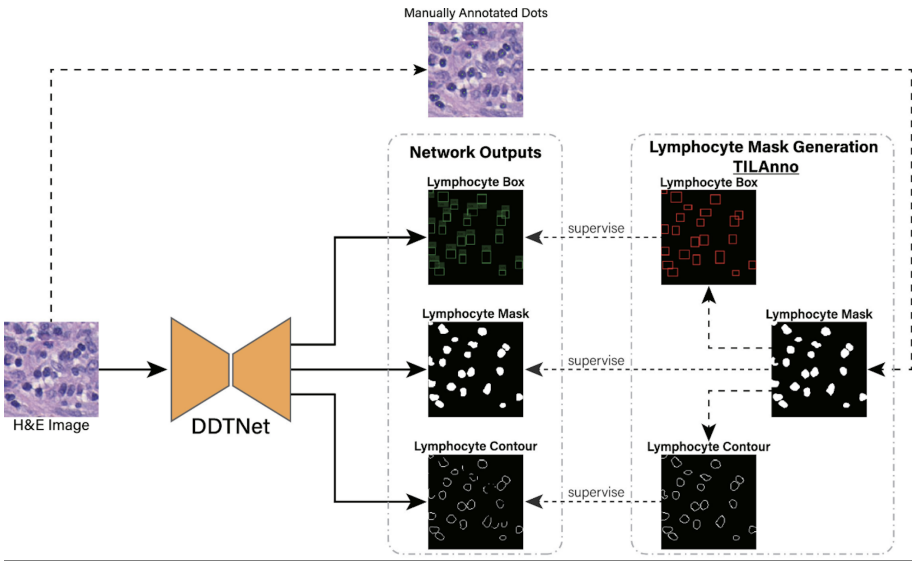


Figure 4.4: DDTN workflow during training.

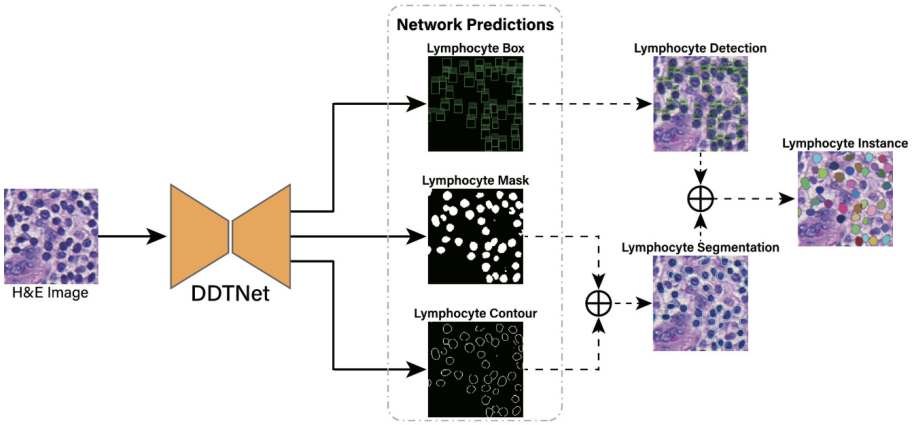


Figure 4.5: DDTN workflow during inference.

A detailed architecture of the DDTN model and each of its key components, like the backbone model, segmentation and detection modules, and feature fusion module,

can be seen in Figure 4.6.

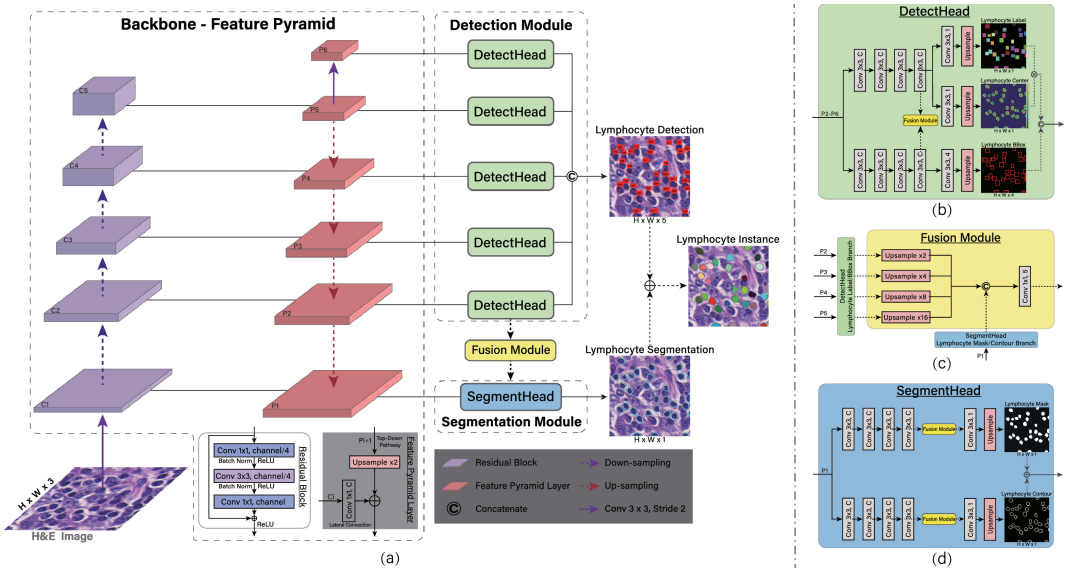


Figure 4.6: DDTN architecture.

The study uses two publicly available datasets and creates a new one with the use of their semi-automatic mask generator, which the authors called TILAnno. The datasets used were:

- BCa-lym dataset: containing 100 H&E stained ROIs of size 100×100 pixels. 3,064 cells with point annotations are present on them.
- Post-NAT-BRCA dataset: contains H&E-stained WSIs with manual annotations of different cell types. For this study, 29 WSIs were selected and 740 ROI patches of size 100×100 pixels were used, together containing 4,488 dot-annotated lymphocytes.
- TCGA-lym introduced dataset: authors used 15 H&E stained WSIs from The Cancer Genome Atlas (TCGA), extracted ROIs of size 1600×1600 and let two junior pathologists annotate the lymphocyte centers and then a single expert refined them. In total, 5,029 cells were annotated. For the training,

each ROI was divided into 64 200×200 patches.

Each dataset originally contained dot annotations of lymphocytes. The authors used the TILAnno tool to generate pixel-level masks, contours, and bounding boxes.

During the training, the input images were first resized to 320×320 pixels, and different augmentation techniques were used (mirror, flip, light noise, brightness, and color conversion). ResNet101 was used as a backbone network, and it was pre-trained on the ImageNet dataset. The training hyperparameters were set in the following fashion: 1000 epochs with stochastic gradient descent, batch size of 4, an initial learning rate of 0.0001, and decreased by a factor of 10 at the 500th and 750th epochs; weight decay was set to 0.01 and momentum to 0.9.

Evaluation metrics were split for detection and segmentation tasks. In the lymphocyte segmentation task, the Dice score, Aggregated Jaccard Index, and panoptic quality were used. In the lymphocyte detection task, the precision, recall, and F1-score were used, where the truthfulness of the positivity of a sample was determined by the IoU threshold set to 0.5 in case of TP/FP and FN if the bounding box does not intersect any ground truth bounding box.

For the evaluation of the TILAnno tool, the authors compared it to two other tools used for weak cell segmentation, namely QuPath and Cell Profiler. They ran all three tools on all three datasets, then let two experts manually label lymphocyte boundaries on 20 randomly selected images, which were used for evaluation with expert labels as ground truth. The TILAnno tool (Ours) seemed to outperform the baseline tools by a great margin in the Dice score, as can be seen in Figure 4.7. The study further analyzed the model performance of their proposed solution with other baseline models during inference, and from Figure 4.8 we can see that their model outperformed the in all metrics except time.

Method	BCa-lym	Post-NAT-BRCA	TCGA-lym
Cell Profiler (Carpenter etal., 2006)	0.617	0.566	0.747
QuPath (Bankhead etal., 2017)	0.640	0.659	0.822
Ours	**0.982**	**0.937**	**0.926**

Figure 4.7: Comparison of TILAnno and baseline tools.

Method	BCa-lym				Post-NAT-BRCA				TCGA-lym			
	F1	Dice	PQ	Time	F1	Dice	PQ	Time	F1	Dice	PQ	Time
DCAN (Chen etal., 2016)	0.573	0.753	0.437	**0.0052**	0.694	0.737	0.539	**0.0046**	0.501	0.666	0.381	**0.0059**
U-Net (Ronneberger etal., 2015)	0.656	0.831	0.550	0.0081	0.646	0.779	0.561	0.0076	0.470	0.701	0.391	0.0065
HoVer-Net Graham etal. (2019)	0.812	0.845	0.711	0.7669	0.793	0.812	0.703	0.7617	0.691	0.768	0.617	0.7611
ANCIS (Yi etal., 2019b)	0.854	0.807	0.681	0.2159	0.845	0.787	0.690	0.0907	0.720	0.702	0.518	0.0847
Ours	**0.885**	**0.845**	**0.731**	0.0674	**0.892**	**0.846**	**0.782**	0.0662	**0.793**	**0.788**	**0.635**	0.0647

Figure 4.8: Comparison of DDTN and baseline models.

Moreover, the study also evaluated the model's ability to generalize by training

it only on the BCA-lym and Post-NAT-BRCA datasets and evaluating it on the TCGA-lym dataset. The results are summarized in Figure 4.9, where we can see that their model again outperformed the existing baselines in all used metrics.

Method	**BCa-lym♯**			**Post-NAT-BRCA♯**			**BCa-lym♯ + Post-NAT-BRCA♯**		
	F1	**Dice**	**PQ**	**F1**	**Dice**	**PQ**	**F1**	**Dice**	**PQ**
DCAN (Chen etal., 2016)	0.301	0.485	0.203	0.301	0.581	0.226	0.457	0.549	0.341
U-Net (Ronneberger etal., 2015)	0.309	0.446	0.251	0.380	0.605	0.313	0.443	0.615	0.385
HoVer-Net Graham etal. (2019)	0.517	0.551	0.427	0.573	0.633	0.489	0.592	0.643	0.513
ANCIS (Yi etal., 2019b)	0.551	0.470	0.284	0.623	0.582	0.405	0.634	0.562	0.421
Ours	**0.600**	**0.568**	**0.459**	**0.695**	**0.691**	**0.514**	**0.739**	**0.717**	**0.577**

Figure 4.9: Comparison of DDTN and baseline models in generalization.

4.4 Nuclei segmentation with point annotations from pathology images via self-supervised learning and co-training [76]

In the fourth work. The authors present a self-supervised approach that generates segmentation masks from point annotations of cell nuclei in H&E-stained images.

Two datasets were used, and since each dataset was annotated using pixel-level annotations, these were converted to point annotations that were set approximately to the center of each mask. The datasets were:

- MoNuSeg dataset, which we already mentioned earlier in this chapter. 24 images were used for training, 6 for validation, and 14 for testing.
- CPM dataset, containing 32 500×500 or 600×600 H&E stained images of four tumor types. 20 images were used for training, 4 for validation, and 8 for testing.

All the images for training were cropped to 250×250 patches with 125-pixel overlap for training - these are then randomly cropped further into 224×224 sub-patches, rotated, flipped, and zoomed. The images used for testing are cropped to 224×224 patches with 80-pixel overlap.

Their method contains three modules:

1. Segmentation of nuclei with rough (not very precise) labels. Initial pixel-level masks are generated as follows: From point annotations using a Voronoi diagram and k-means clustering the Voronoi labels (a division of the image into convex polygons) and cluster labels (3 clusters in total - nuclei, background, ignored area) are generated. The H-component image is separated from the original H&E-stained image. Then, the Residual U-Net network is trained using the Voronoi and cluster labels with cross-entropy loss to generate coarse pixel-level masks.
2. Next in the co-training strategy, two segmentation networks are trained, where they supervise each other. The training data is split into two parts, and each network is trained with one part. Apart from the two mentioned labels, each of them also uses pseudo-labels generated by the other network, which are stabilized using exponential moving average (EMA), where the averaged predicted labels are used to label the ignored area of the cluster label. The co-training loss is given by the Kullback-Leibler divergence.

3. The self-supervised representation learning employs two U-Nets in sequential order, where the first U-Net computes the nuclei probability map (using the H-component images) and the second then reconstructs the colorized image from these maps.

To integrate all of these modules, a final model is proposed. It has two networks, which are co-trained using Voronoi, cluster, and each other's labels (with the EMA stabilization) and colorization loss. Each network consists of two U-Nets, the segmentation U-Net and the colorization U-Net. The ResNet-34 is used as a backbone network, pre-trained on the ImageNet dataset. The training hyperparameters were set as follows: initial learning rate of 0.001 reduced by a factor of 10 every 30 epochs, Adam optimizer, and weight decay set to 0.0005. The colorizing network part is discarded during inference, and only the segmentation part is used. The full architecture can be seen in Figure 4.10.

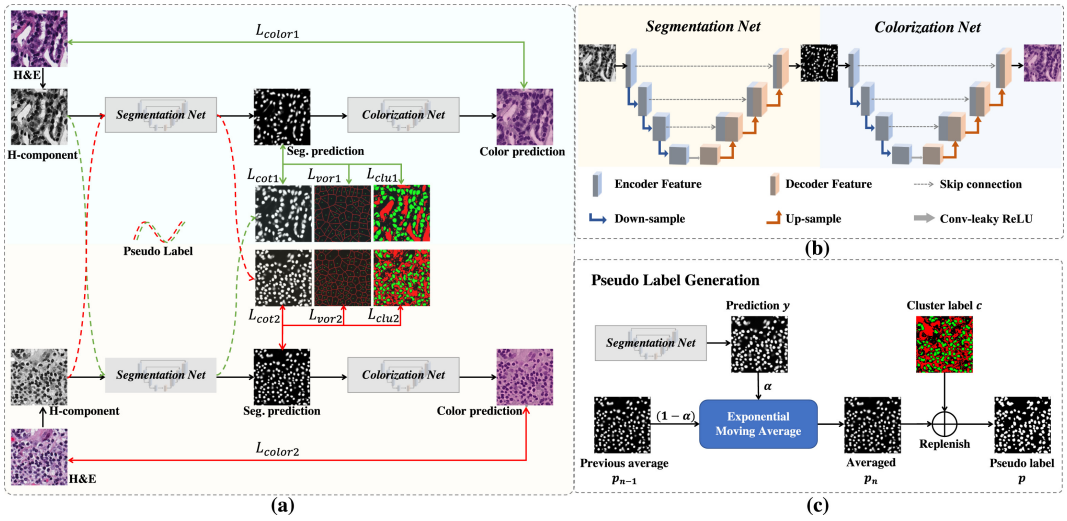


Figure 4.10: The architecture of the proposed model.

Pixel accuracy, F1-score, Dice coefficient, Aggregated Jaccard Index, Detection Quality (DQ), Segmentation Quality (SQ), and Panoptic Quality (PQ) are used

as the evaluation metrics. From Figure 4.11 we can see that the proposed network achieved better results on both datasets in almost all metrics when compared to other state-of-the-art models trained for weakly supervised nuclei segmentation with the same set of hyperparameters.

Model	MoNuSeg							CPM						
	Acc (%)	F1 (%)	$Dice_{obj}$ (%)	AJI (%)	DQ (%)	SQ (%)	PQ (%)	Acc (%)	F1 (%)	$Dice_{obj}$ (%)	AJI (%)	DQ (%)	SQ (%)	PQ (%)
Yoo et al. (2019)	86.51	72.11	56.68	29.03	43.97	72.21	31.86	90.50	79.81	72.46	49.42	61.04	**73.63**	45.10
Tian et al. (2020)	88.01	71.47	63.96	40.51	50.67	67.19	34.12	87.87	71.74	64.39	42.11	42.86	63.03	27.06
Xie et al. (2020)	91.19	77.56	72.51	51.69	68.82	72.50	49.97	89.96	75.87	70.28	49.40	59.49	70.87	42.68
Qu et al. (2020)	**91.52**	76.76	73.24	54.32	69.72	71.28	49.84	89.90	76.56	71.17	50.91	64.10	70.66	45.69
Chamanzar and Nie (2020)	91.04	74.18	71.70	53.69	69.40	69.84	48.75	88.57	70.69	66.44	45.92	57.06	67.80	39.20
Lee and Jeong (2020)	91.13	77.05	73.44	54.20	72.03	71.80	51.78	89.85	75.80	70.82	50.26	65.37	69.75	45.99
Ours	91.44	**77.64**	**74.41**	**56.20**	**73.27**	**72.48**	**53.19**	**91.01**	**79.97**	**73.73**	**51.69**	**68.42**	72.18	**49.66**

Figure 4.11: The architecture of the proposed model.

4.5 Weakly Supervised Deep Nuclei Segmentation With Sparsely Annotated Bounding Boxes for DNA Image Cytometry [77]

The last work focuses on segmenting cell nuclei in DNA image cytometry from bounding box annotations using a teacher-student network setup.

Two datasets were used:

- DNA-ICM database, contains 23,485 images of cervical cancer screening stained with feulgen and eosin. Each image has 4096×2816 pixels. Together, the dataset contains more than 1M cell nuclei. 18,266 images were selected for training and validation, and 5,219 for testing. The authors used a semi-automated approach to get pixel-level masks for the test set. For the training and validation sets, they initially generated the pixel-level masks with traditional methods and then let experts refine them.
- ISBI14 dataset, containing 16 real and 945 synthetic images of cervical cy-

tology. 8 real and 45 synthetic images were used for training, while the rest were used for testing.

Firstly, pseudo-masks are generated for each available bounding box by cropping out the box area and applying traditional segmentation methods, namely Otsu, K-means, and GrabCut. These initial pseudo-labels, along with the bounding boxes, are then used to train the teacher model. It produces pseudo-labels in the form of refined masks for ground truth nuclei labels (bounding boxes), and bounding boxes and masks for unlabeled nuclei. The student model then uses the ground truth labels and teacher-generated pseudo labels to further optimize the loss. The loss, combining the supervised, weakly supervised, and unsupervised losses, is used for the training of the student model.

Both the teacher and student models share the ResNet-50 with a feature pyramid network (with discarded level P6) as a backbone. The backbone is initialized with weights pre-trained on the ImageNet dataset. It is used to extract ROI features. Then both have the same architecture, the Mask R-CNN, which is, in total, trained for 32,000 iterations. The first 16,000 iterations are used to train the teacher model. Then the pseudo-labels generated by it are used for training the student model. The student model is initialized with the weights of the teacher model. The architecture can be seen in Figure 4.12. The training hyperparameters were set as follows: weight decay of 0.0001; momentum of 0.9; initial learning rate of 0.01, and decreased by a factor of 10 after the 20000th and 27000th iterations.

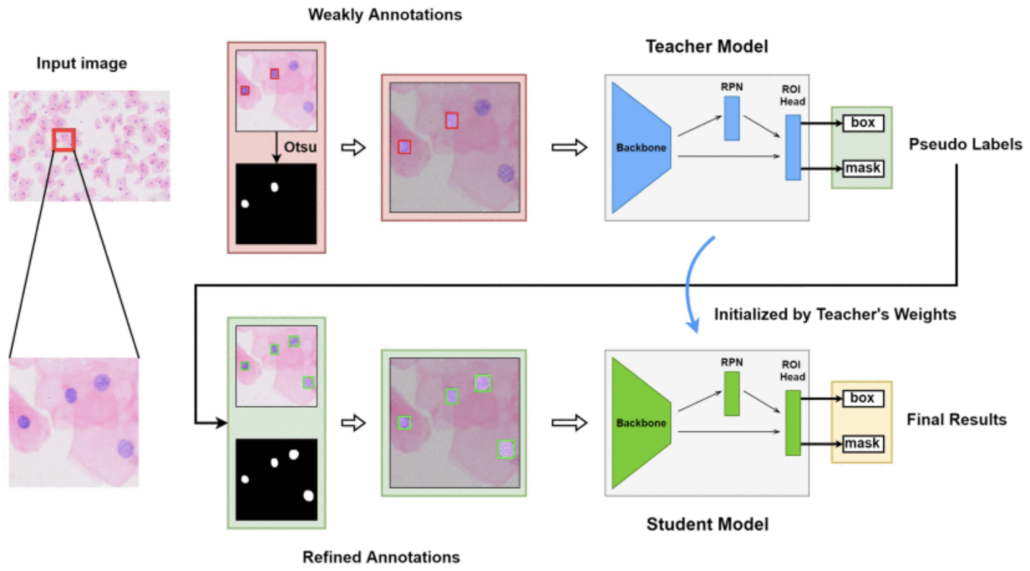


Figure 4.12: The architecture of the teacher-student model.

Precision, recall, pixel-level IoU, and Dice coefficient were used as evaluation metrics for nuclei segmentation. For cell detection, the average precision and recall over different IoU thresholds were used.

The comparison of results compared to other state-of-the-art weakly supervised methods can be seen in Figures 4.13 for segmentation and 4.14 for detection on the DNA-ICM dataset, where it achieved the best results compared to the other methods in all metrics but recall. Figure 4.15 displays results comparison for segmentation on the ISBI14 dataset. Since this dataset is fully annotated with pixel-level masks, the authors used these (either 100% of them or 50% of them) to generate the pseudo-labels. The model trained on this dataset was trained only for 100 epochs in total, 50 of which were for the training of the teacher model. Again, their solution seemed superior in most of the metrics when compared to the other models.

method	*IoU*	*Dice*	*Precison*	*Recall*
WeaklySegPartialPoints [17]	6.7	12.6	10.4	15.9
C2FNet [18]	3.4	6.6	3.4	**100.0**
Scribble2Label [20]	53.2	69.4	61.3	80.0
Ours	**57.0**	**72.6**	**73.8**	71.4

Figure 4.13: The comparison of results for segmentation on the DNA-ICM dataset.

method	AP	AP_{50}	AP_{75}	AR^{100}
RetinaNet [40]	61.8	81.8	83.5	70.2
YOLOv3 (DarkNet-53) [41]	63.1	84.6	84.4	70.6
FCOS [44]	60.8	80.3	82.2	70.6
Reppoints [45]	**68.1**	**86.4**	**89.5**	**73.8**
ATSS [46]	64.2	81.8	84.8	73.2
Ours	67.5	86.2	88.5	73.3

Figure 4.14: The comparison of results for detection on the DNA-ICM dataset.

method	annotation	*IoU*	*Dice*	*P*	*R*
WeaklySegPartialPoints [17]	100%	14.1	24.8	14.2	**98.3**
C2FNet [18]	100%	55.6	71.5	58.3	92.5
Scribble2Label [20]	100%	69.8	82.2	70.9	97.9
Ours	100%	**72.7**	**84.2**	**95.5**	75.2
WeaklySegPartialPoints [17]	50%	6.7	12.6	6.7	**99.9**
C2FNet [18]	50%	7.4	13.8	**99.9**	7.4
Scribble2Label [20]	50%	69.5	82.0	72.2	95.0
Ours	50%	**70.0**	**82.4**	72.0	96.2

Figure 4.15: The comparison of results for segmentation on the ISBI14 dataset.

Chapter 5

Our Work

5.1 Overview

This work presents a method dealing with challenges where we have a fully annotated dataset, however, very small in size (TNBC dataset [78]), and a large dataset but weakly annotated (TIGER dataset [15]). We aim to overcome these challenges by implementing a hybrid approach for the semantic segmentation of lymphocyte cells. The overview of the whole approach is displayed in Figure 5.1. The hybrid method consists of a preprocessing module, which prepares data for training and evaluation; then the pseudo-mask creating module, which creates the pseudo-masks for the TIGER dataset images. The resulting image patches are then used for the training of the deep learning segmentation model, which we describe in Section 5.3.

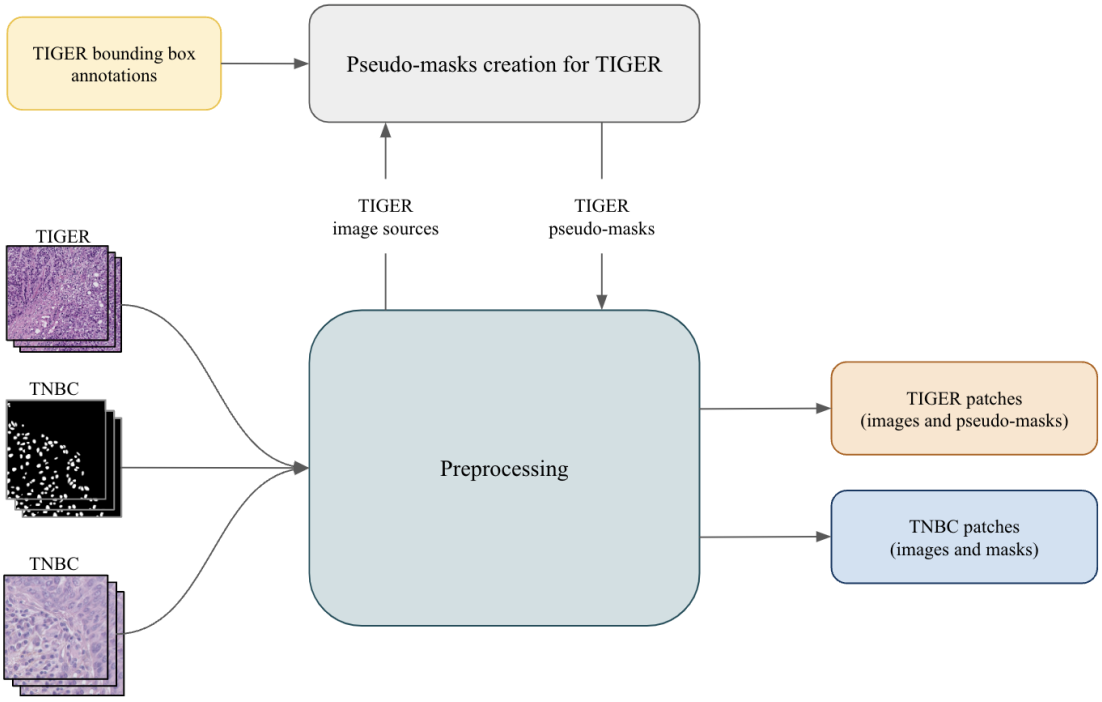


Figure 5.1: The overview of our hybrid approach for semantic segmentation.

Firstly, we try to train and validate a model on the small dataset itself. Then we use preprocessing and computer vision techniques to generate various pseudo-mask sets out of bounding-box annotations for the weakly annotated dataset and train a model on it, which is again validated on the small, fully annotated dataset. Then we try to identify and select the best fusing strategy for the mask sets, to utilize different abilities of the sets to capture the cell region. Next, we select the best model (with the most successful mask-fusing strategy) and fine-tune it using a portion of the data in the fully annotated dataset. We evaluate each of these experiments with the Dice coefficient and IoU. We start with the fundamental part - the description of the datasets used.

5.2 Datasets

TIGER In our work, we use the Tumor Infiltrating Lymphocytes in Breast Cancer - TIGER - dataset, which was released with the challenge under the same name on the Grand Challenge platform [15]. It contains H&E-stained WSIs of HER2-positive and TNBC breast cancer tissues obtained by core needle biopsies or surgical resections. The images were scanned using 20x magnification. The dataset is released in three formats. We work with the one called WSIROIS. The WSIs come from three different institutions:

1. TCGA (151 WSIs) dataset, which contains images of TNBC from the TCGA-BRCA archive, annotations, and magnification, was adopted to be in line with those used further.

2. RUMC (26 WSIs) images of both TNBC and HER2-positive breast cancer obtained from Radboud University Medical Center in the Netherlands, annotated by a panel of board-certified pathologists.

3. JB (18 WSIs) images of both TNBC and HER2-positive breast cancer obtained from Jules Bordet Institute in Belgium, annotated by a panel of board-certified pathologists.

The RUMC and JB WSIs contain 3 annotated ROIs with a size of approximately 500×500 µm. The WSIs obtained from TCGA are more specific. This dataset was created by merging two other datasets: the BCSS (151 WSIs) and the NuCLS (124 WSIs). The NuCLS is a subset of the BCSS dataset. In the BCSS dataset, the tissue in a single large ROI is annotated, but no cells are annotated. In the NuCLS, a variable number of smaller ROIs are selected within the large ROI (same large ROI as in the BCSS), and these are densely annotated for multiple cell types. Annotations are adapted to match the other used annotations, as was

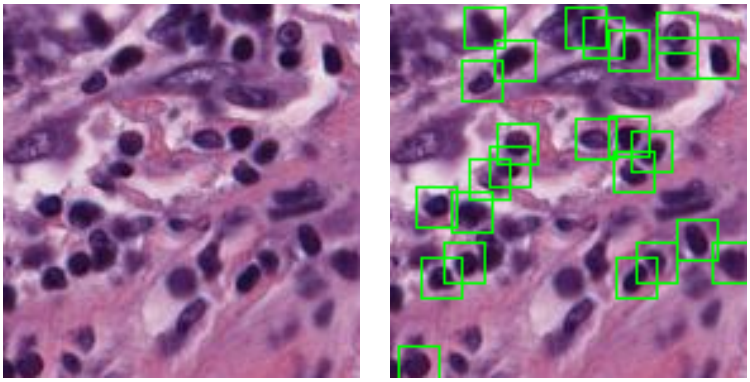
mentioned.

The WSIROIS format contains:

- WSI level annotations, wherein each WSI contains manual annotations of ROIs. Different tissue types are annotated with polygons, namely: invasive tumor, tumor-associated stroma, in-situ tumor, healthy glands, necrosis not in-situ, inflamed stroma, and rest. Most ROIs have also annotated plasma cells and lymphocytes. These were annotated using point annotations and then a bounding box was constructed and centered on the point of annotation with the size of 6×6 µm, 8×8 µm, or 9×9 µm. Annotations for WSIs are released in XML format and also as a multi-resolution TIF image.
- ROI level annotations, where authors cropped the ROIs from WSIs and stored them as PNG files. Tissue type annotations are released as PNG images, containing pixel-level masks, and cell annotations are released in the COCO format - a JSON file containing file paths (the PNG images of ROIs) with IDs and metadata and corresponding annotations of bounding box position and size.

We further work with the part of the dataset that has ROI-level annotations. This part of the dataset consists of 1,879 (1,744 from TCGA, 81 from RUMC, 54 from JB) ROIs cropped from 44 (124 from TCGA, 26 from RUMC, 18 from JB). Together, they contain 30,524 annotated cell nuclei.

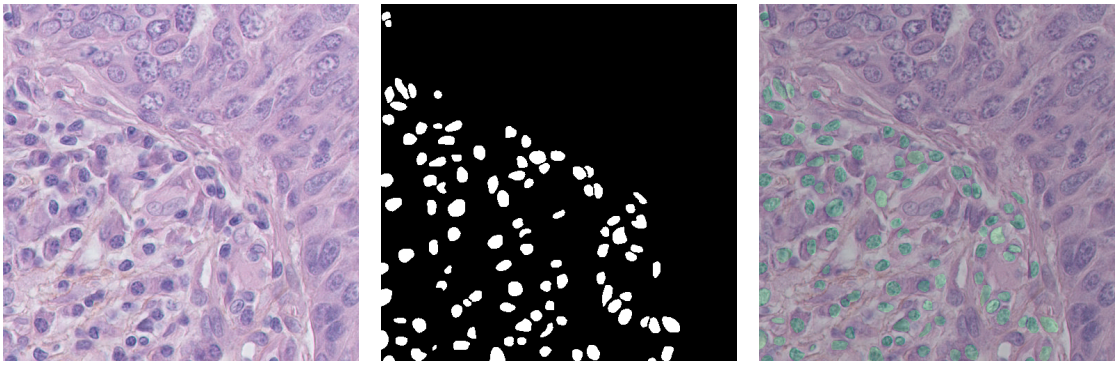
An example of an image and its bounding box labels can be seen in Figure 5.2.



(a) Without annotations (b) With annotations

Figure 5.2: Example of TIGER image without and with annotations of TILs [15].

TNBC Triple Negative Breast Cancer Nuclei Segmentation dataset [16], is an open dataset consisting of 11 patients with breast cancer, with varying numbers of images for each patient, provided regions of interest (ROIs) in PNGs. Together, it has 50 annotated ROIs of size 512×512, scanned with 40x magnification. Specifically, we use the extended version of this dataset [78], where annotations of cell classes were added. Each image has a corresponding pixel mask, where each pixel is labeled by the class it represents. There are 11 different cell classes, plus background and unknown classes. We also note that this extended version of the dataset provides images of brain tissue, but since it is not part of our work, we only work with the images of breast cancer. An example image with its ground truth mask, already relabeled so only lymphocytes are annotated, can be seen in Figure 5.3.



(a) Image (b) Mask (c) Image with mask overlay

Figure 5.3: Example of TNBC image, mask, and overlay, where TILs are annotated [78].

5.3 Deep Learning Model

5.3.1 Architecture

As a deep learning model for semantic segmentation, we employ the U-Net architecture. U-Net is a powerful architecture, and as we mentioned in Chapter 3, in Section 3.5, it is also widely used in the medical imaging domain. Specifically, we use the ResNet-34 encoder as the U-Net's backbone, which is already pretrained on the ImageNet dataset. This choice was based on the fact that residual blocks further improve the U-Net's ability to learn, as we continue to write in Section 3.5 of Chapter 3. In the state-of-the-art, which we present in Chapter 4, authors in [75, 77] also use ResNet architectures, and specifically, ResNet-34 is used in [76]. The inspiration to initialize the encoder with weights pretrained on the ImageNet dataset came from the state-of-the-art works as well, where a similar approach was used in [75, 76, 77]. The full architecture of the model can be seen in Figure 5.4.

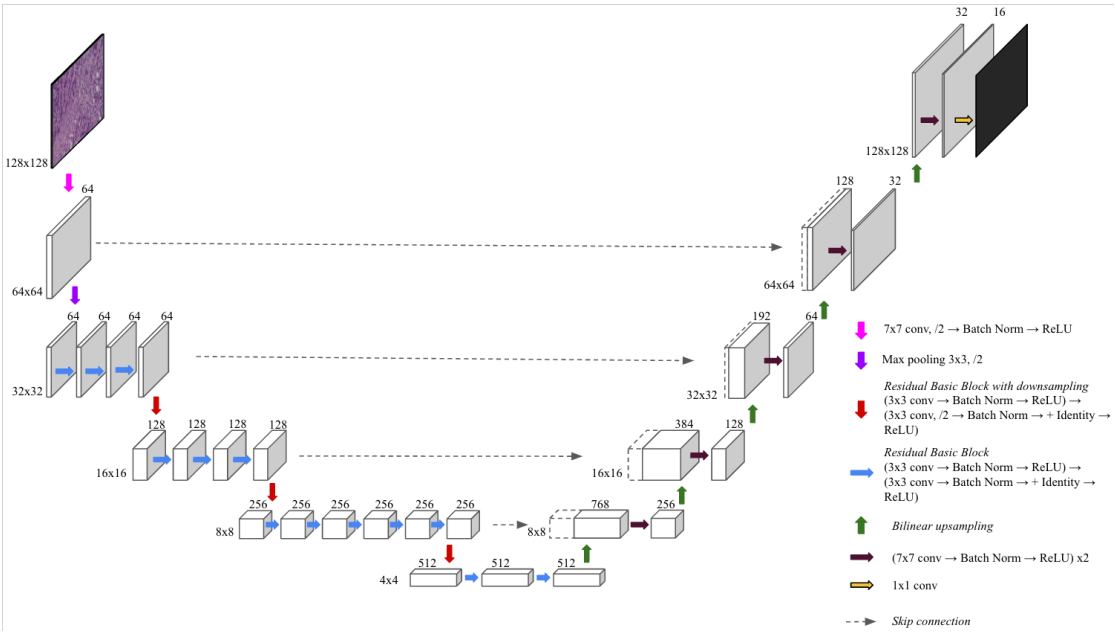


Figure 5.4: The architecture of our model.

This model is being used in all our experiments as the segmentation model. The training setup and model's hyperparameters remain the same in every experiment as well.

5.3.2 Input and Output Specifications

The input for the model is an image of size 128×128 pixels in a 3 - RGB - channel space. The model's segmentation head produces a binary mask of size 128×128 pixels and a depth of 1. The output mask is then run through the sigmoid function to squeeze the values between 0 and 1. A threshold of 0.5 is applied to this mask as pixels with a value less than 0.5 are predicted background and labeled with a number 0, and pixels with a value greater than or equal to 0.5 are predicted TILs and labeled with a number 1.

5.3.3 Loss Function

The loss function used during training is the Dice Loss function. This loss function is the preferred function to be used in the segmentation of objects and is also frequently used in the medical imaging domain, as described in [39].

5.3.4 Optimization

We used the Adam optimizer, and the initial learning rate was set to 0.001, and it was reduced every 5 epochs by a factor of 0.1. Early stopping was also used, where the patience was set to 10 checks - if the validation loss was not improved during the training and it got worse at least on 10 checks, the training stopped to prevent overfitting. Checks were performed during every validation run, after every epoch.

5.3.5 Training

Every training was set to run for 100 epochs, but it could be stopped earlier. The batch size was set to 16 samples. Checkpoints of the model were periodically saved every epoch, always the best checkpoint (in terms of validation loss). If the CUDA framework is available, the training runs on the GPU; if not, then on the CPU. During the training stage, we monitored the model's performance on various variables. These included accuracy, recall, precision, Dice coefficient, IoU, and the running loss.

For training purposes, we further split the training data into the training subset (80% of the whole training dataset) and the validation subset (20% of the whole training dataset). Then, in each epoch, we let the model process the whole training subset (in the form of batches) while monitoring and logging the aforementioned variables. After each epoch, we set the model into validation mode. In this mode,

the model does not update its parameters. We let it process the validation subset and also monitor and log the variables, out of which the most interesting for us is the validation loss, since this is used both for the early stopping and for saving the checkpoints. After the validation was completed, we set the model into the training mode again, where it could further update its parameters. This whole process was repeated until the training was finished, and the model could be evaluated on the testing dataset.

5.4 Evaluation Methods

Upon completing the training process, we initiated the final evaluation of the segmentation model. The best-performing model, determined by the lowest validation loss, was loaded from the saved checkpoint and set to evaluation mode to prevent any parameter updates. Subsequently, the model processed the entire testing dataset, and the final testing metrics were computed.

We assessed the model using both quantitative and qualitative methods. The quantitative evaluation included the Dice coefficient and IoU metrics - we describe these in more detail and why they are most suitable for segmentation tasks in Chapter 3, in Section 3.4. For qualitative assessment, we visualized the predicted binary masks alongside the ground truth masks to facilitate direct comparison.

5.5 Data Preprocessing

Since we work with two very distinct datasets, and furthermore, the TIGER dataset is composed of three other datasets, we need to employ a robust preprocessing framework to align all datasets on the same level. In Figure 5.5, we can see how the images and masks from each dataset are preprocessed. Below, we

describe each preprocessing step for each image and mask set.

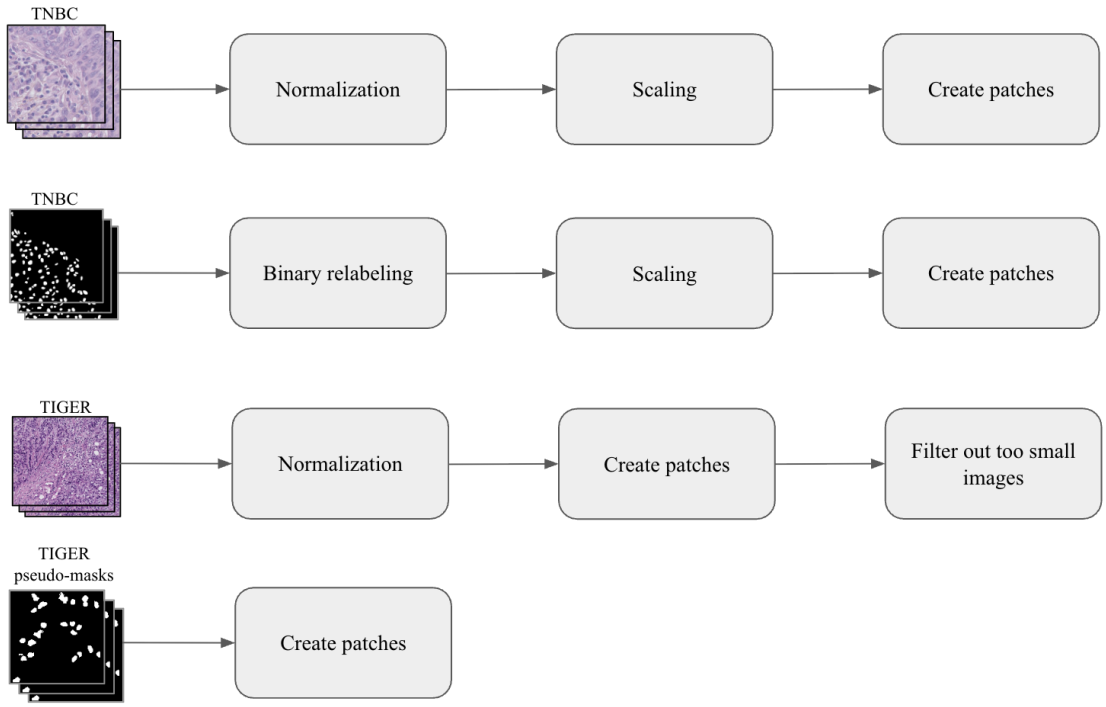
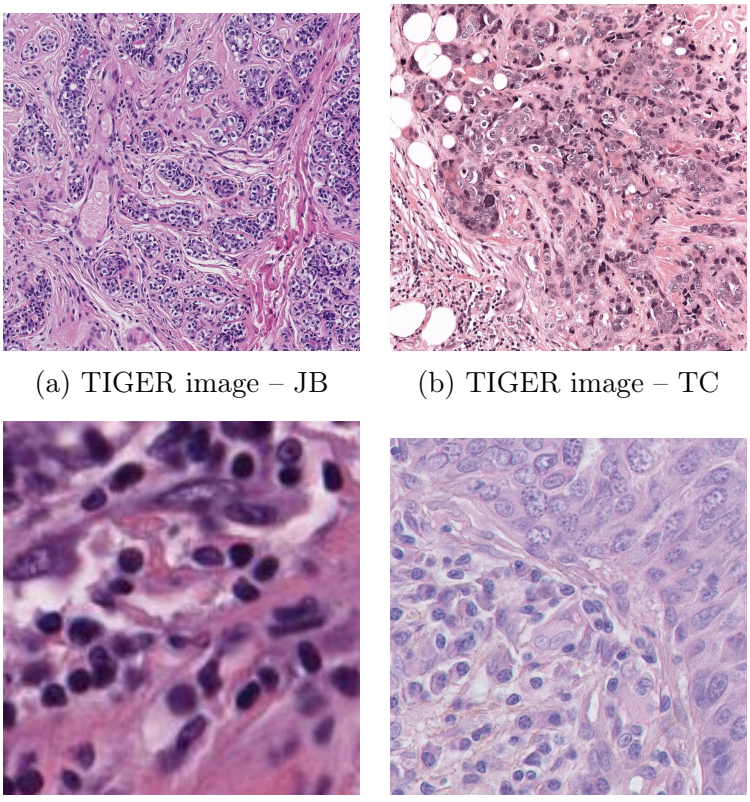


Figure 5.5: The preprocessing pipelines of both TIGER and TNBC datasets.

5.5.1 Normalization

TIGER and TNBC datasets pose several challenges to us. As we described in section 5.2, data come from four distinct sources (three for TIGER and one for TNBC) - this means that the staining is very different, which we can see in Figure 5.6.



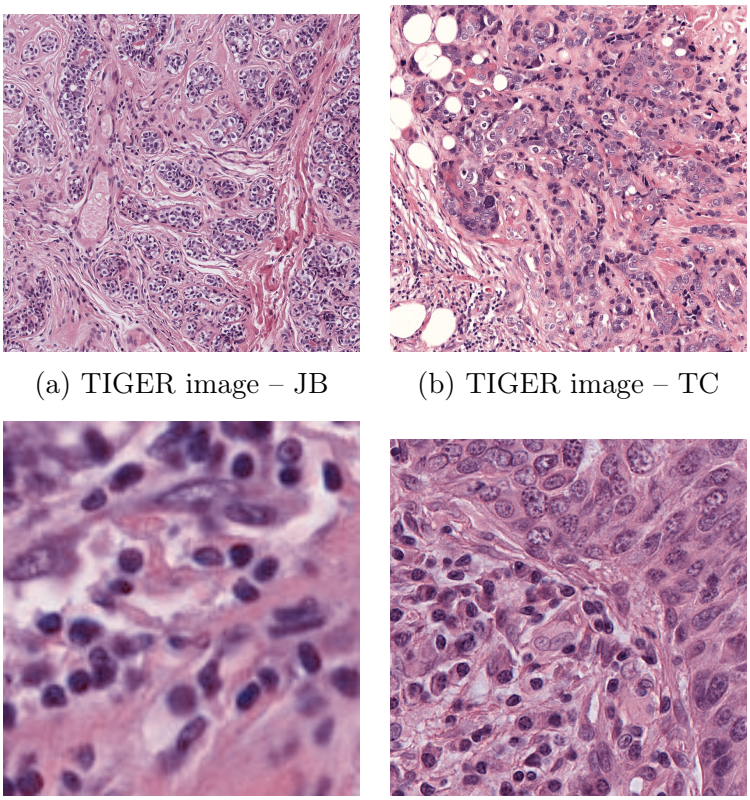
(a) TIGER image – JB (b) TIGER image – TC

(c) TIGER image – TCGA (d) TNBC image

Figure 5.6: Example images from TIGER datasets [15] and TNBC [78] before normalization.

Firstly, we tried a randomly selected single image as a reference image for Macenko normalization, which yielded suboptimal results when assessed visually. We have considered other approaches to selecting a reference image, but as a recent study [79] showed, when selecting only a single reference image for Macenko normalization, the results will always be biased and suboptimal. Therefore, we employ the multi-target Macenko stain normalization technique as described in [79], where we select 8 reference images from the TIGER and 2 reference images from the TNBC dataset. This number is not arbitrary; in [79], authors experimented with a different number of reference images (2-20) and showed that the higher number has slightly better results, but if the number is too high, there are no signifi-

cant improvements. They also experimented with different ways of computing the stain matrix, and the best results were achieved by the *avg-post* method, and this method peaked when 10 reference images were selected [79]. Given the sizes of our respective datasets, we decided to go with the 8 and 2 images and also with the *avg-post* method. Then we used the same 10 reference images to normalize both datasets. The normalization technique improved the color inconsistencies, as we can see in Figure 5.7.



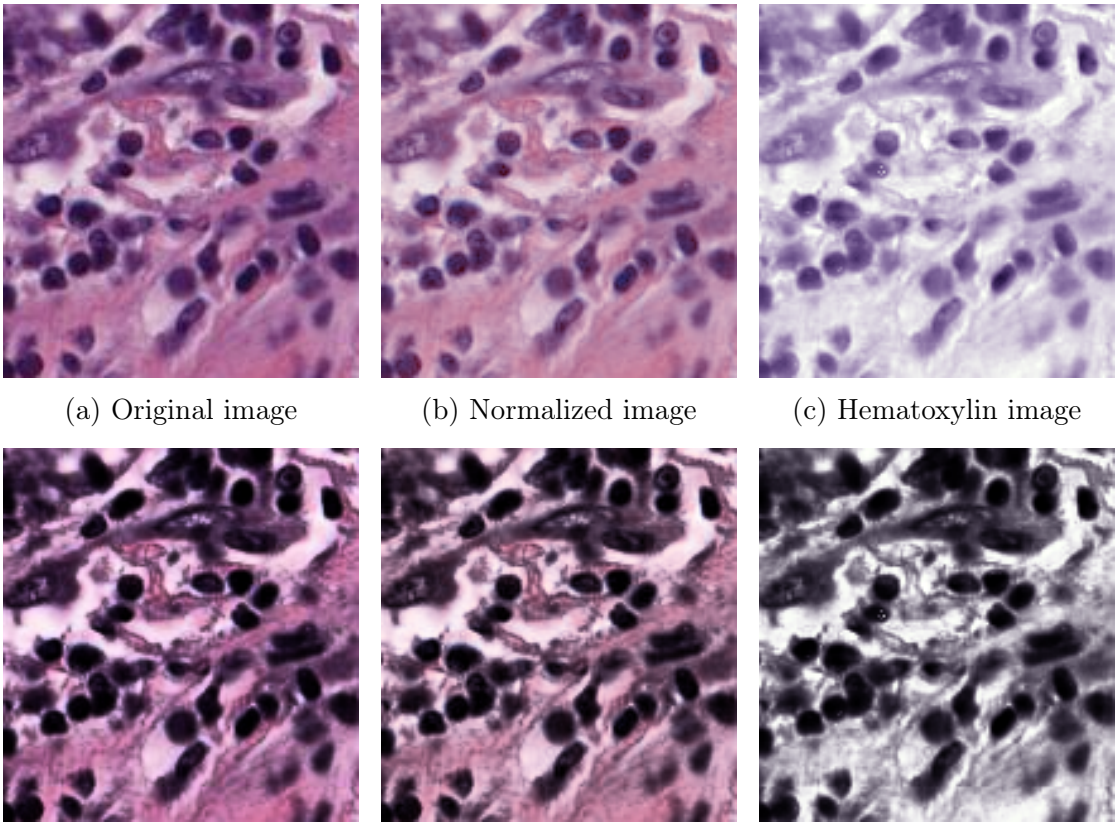
(a) TIGER image – JB (b) TIGER image – TC

(c) TIGER image – TCGA (d) TNBC image

Figure 5.7: Example images from TIGER datasets [15] and TNBC [78] after normalization.

5.5.2 Pseudo-mask Sources

The next step was to generate the pseudo-masks. For this, we created different variations of the same image. We named the images that were created as a part of one variation *image source*. In total, six different image sources were created for the experiments. We used the original (raw) image as one source, then the normalized image as another source. Furthermore, we extracted the hematoxylin image out of the original image (Macenko normalization does this internally). This was done based on the fact that hematoxylin highlights the cell nuclei, as we described in Chapter 1. This hematoxylin image became our third image source. Lastly, by applying histogram equalization to all of the aforementioned image sources (to increase the overall contrast of the image and shift dark colors into darker ones and light into lighter ones), we obtained another three image sources. The difference in the image sources can be seen in Figure 5.8.



(a) Original image (b) Normalized image (c) Hematoxylin image

(d) Original image with histogram equalization (e) Normalized image with histogram equalization (f) Hematoxylin image with histogram equalization

Figure 5.8: Examples of image sources.

We then operated on each image source with different computer vision techniques to generate the final pseudo-mask PNGs. We describe each of these techniques in Section 5.6.

5.5.3 Aligning the TNBC Dataset

To use the TNBC dataset, we needed to align its scale and the ground truth masks. This dataset was scanned with the 40× magnification; however, the TIGER dataset was scanned using the 20× magnification. To align them, we down-scaled the TNBC dataset images and masks by a factor of 2. Moreover, the TNBC ground

truth masks were relabeled to binary masks by setting all of the other labels except for the lymphocyte cell labels as background.

5.5.4 Patching Strategy

To be able to feed our data to the deep learning UNet model, we created patches of fixed size 128×128 pixels. The TIGER dataset contained images of varying sizes. Therefore, we created overlapping patches with a dynamic stride in such a way that no patch was shifted outside of the original image. We created 19,386 patches of the TIGER dataset images. The TNBC dataset was nicer since the original images were of 512×512 pixels in size, and after down-scaling by a factor of 2, they became 256×256 pixels. Each image was then split into 4 non-overlapping patches. This got us exactly 200 patches of TNBC dataset images. After this stage, the data is ready for the training and evaluation process.

5.5.5 Images to Tensors

For the PyTorch framework to work with the PNG image patches, both original images and masks, we needed to convert them from NumPy arrays into tensors. This was done before the training and evaluation of each trained model.

5.6 Pseudo-masks Generation

To be able to start training the segmentation model on the TIGER dataset, we needed to convert the bounding box annotations of lymphocyte nuclei into pixel-level pseudo-masks. We used a series of computer vision methods, which were chained in different ways to better identify the region where a cell nucleus is present within the bounding box. This task was challenging because of the lower contrast between the nuclei and the surrounding tissue, and also because some nuclei were

too close to each other, meaning that there was an overlap between the bounding boxes. This inspired us to create different versions of a single image, to promote some properties of the image, such as increasing the contrast or isolating only the hematoxylin staining. We called these versions image sources, and the process of their creation is described in Subsection 5.5.2 as a part of image preprocessing. The whole process of pseudo-mask generation can be seen in Figure 5.9. To better understand the chain of operations applied, we will illustrate it on a single image example of one image source:

1. Firstly, the image is loaded together with its corresponding bounding box annotations.
2. Next, the individual labeled nuclei are cropped out of the image, using the bounding box values.
3. Then a four combinations of operations are applied on the cropped region, which means that from the single cropped region, four new versions of it are created, based on which combination of operations was applied. It was either:
    - The Otsu thresholding
    - The Adaptive thresholding
    - The median blur with Otsu thresholding
    - The median blur Adaptive thresholding
4. In the next step, the morphological opening is applied to remove small artifacts left after the thresholding.
5. The previous operations created so far a 'prototype' of the pseudo-mask. In the subsequent step, the marked watershed is applied, which uses this

prototype together with the initial cropped image region to create a final pseudo-mask.

6. Finally, the small pseudo-masks for individual cells are combined into a full-image pseudo-mask

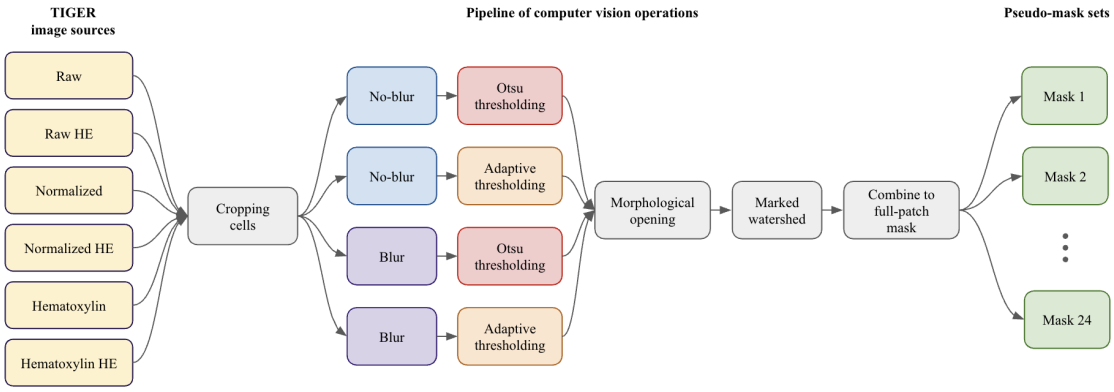


Figure 5.9: The process of pseudo-masks generation.

Given that we work with 6 image sources and 4 pipelines of computer vision operations, where each pipeline is applied to every image source, this gives us in total of 24 pseudo-masks for any given image. In the following paragraphs, we describe each computer vision operation in more detail.

Median Blur We use the 3×3 median blur with the intuition that it could remove small noises around the cell nuclei. This filter replaces each pixel with the median of its neighborhood, which is determined by the size of the kernel. We use the kernel of size 3×3 - this is reasonable for us, since the bounding boxes are of size 12×12, 16×16, or 18×18 and we do want to remove possible small noise but still preserve the shape of the nuclei.

Otsu Thresholding The Otsu thresholding finds a threshold that minimizes intra-class variance and then binarizes the pixel values based on this threshold. Since this operation needs a grayscale image, we first do this conversion. Then we apply the Otsu thresholding method and inversion - this automatically computes the optimal Otsu threshold and also inverts the binarization so that dark nuclei appear as white foreground and background pixels become black.

Adaptive Thresholding In contrast to Otsu thresholding, adaptive thresholding computes the threshold for each pixel based on its neighborhood. We use the 11×11 adaptive thresholding with a constant of 2 - this ensures that we cover the small nuclei diameter but still preserve fine details. In the Equation 5.1 we can see how the threshold $T(x, y)$ is computed for each pixel with coordinates x and y. Firstly, a window $\mathcal{W}$ of size $B \times B$ is centered around the pixel. Then each pixel's grayscale intensity $I(u, v)$ within this window with pixel coordinates u and v is summed and then averaged. Finally, a constant is subtracted from this mean to bias the threshold below the local mean. Again, we use it with inversion, as in Otsu thresholding.

$$T(x, y) = \frac{1}{B^2} \sum_{u=x-\frac{B-1}{2}}^{x+\frac{B-1}{2}} \sum_{v=y-\frac{B-1}{2}}^{y+\frac{B-1}{2}} I(u, v) \; - \; C \tag{5.1}$$

Morphological Opening We use the elliptical 3×3 morphological opening to remove any small artifacts left after the thresholding operations and to preserve the shape of the nuclei.

Marked Watershed To obtain the final pseudo-mask, we employ the mark-controlled watershed. This includes a series of steps:

1. Firstly, the distance transform operation computes, for each foreground pixel in the so-far-created binary mask, the Euclidean distance to the nearest background pixel, producing a map whose peaks lie at object centers.

2. Secondly, we threshold the distance map, which keeps only the central 30% of each cell nucleus - this helps to separate the touching nuclei and gives us the pseudo-mask of sure foreground area (the sure nuclei area).

3. Then we dilate the sure foreground with a 3×3 elliptical kernel to expand the region. Now we consider all pixels lying outside of these regions as sure background.

4. After that, we subtract the sure foreground from the sure background mask. This gives us the regions that should have the shape of a ring and are 'unknown' - either background or foreground. Those are the pixels that lie on the boundaries of each cell.

5. Next, we do marker labeling - we mark each connected component of the sure foreground mask with a different mark (integer), starting with number 1. Then we add number 1 to each pixel value. Lastly, we set those pixels that are marked as an unknown region to zero. This step ensures that:

   - The sure foreground areas start from number 2 onward,
   - The sure background areas are marked with number 1, and
   - The unknown regions are marked with number 0.

6. Finally, the watershed algorithm will flood and try to segment the unknown regions (marked with 0). It treats the original image provided to it (the colorful crop of cell nuclei) as a height map, where the brighter pixels represent high elevations and the darker pixels represent low elevations. It also uses

the marker image to seed the foreground and background regions. During the segmentation process, it floods the image starting at each marker label, and when the floods meet, the lines separating the cell nuclei are created. Pixels on these delineating lines have a value set to -1.

After the mark-controlled watershed produces the delineated nuclei mask, we set all pixel values that are lower than or equal to 1 to 0 (sure background and delineation lines) and those that are greater than 1 (all nuclei components) to 1 to create a binary pseudo-mask.

Full-patch Combining After we obtain the small-sized pseudo-masks for each cell nucleus bounding box, we reconstruct the full-patch mask by firstly creating an image where all pixels have 0 values and then applying the binary OR operation with the small-sized pseudo-masks on the position from which they were cropped. By this, we get the final pseudo-mask for the original image, where pixels of nuclei regions have values of 1 and background pixels have values of 0.

5.7 Pseudo-masks Fusion

During the generation of pseudo-masks, we obtained 24 different masks per single image. We decided to fuse them based on the results of the experiment, which we present in Subsection 5.8.2 and based on the visualization of the combined masks, which we can see in Figure 5.10. These combined masks were created using the pixel-wise addition of all 24 pseudo-masks, which gave us a single pseudo-mask per image, where pixel values ranged between 0 (all pseudo-masks labeled the pixel as background) and 24 (all masks labeled the pixel as foreground). For visualization, a pixel-wise multiplication by 10 was applied to the resulting pseudo-masks to better see the combined power of the pseudo-masks.

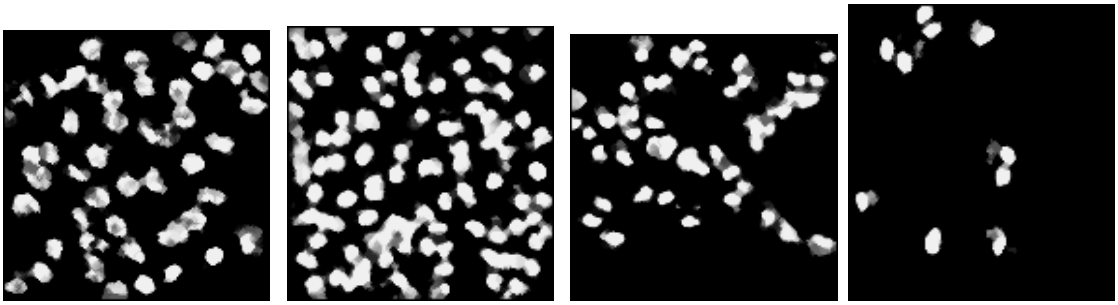


Figure 5.10: Examples of combined pseudo-masks.

We decided to try two different fusion approaches:

1. Fuse the pseudo-masks via pixel-wise voting at quartile agreement levels - 100%, 75%, 50%, and 25% - keeping only those pixels declared foreground by at least that percentage of the masks which achieved the highest Dice scores in the experiment of Subsection 5.8.2. This gave us four sets of fused masks. The overview of this fusing approach can be seen in Figure 5.11. Firstly, the masks are summed together (pixel-wise) and then all pixels that are greater than 0 are set to 1 (foreground - nuclei) to maintain the binary mask.

2. Fuse the pseudo-masks via pixel-wise voting consensus by all involved masks - a pixel was labeled as foreground (cell nuclei) if either:

   - 24 out of 24 masks declared the pixel as foreground,
   - 23 out of 24 masks declared the pixel as foreground,
   - 22 out of 24 masks declared the pixel as foreground, or
   - 21 out of 24 masks declared the pixel as foreground.

   This approach also gave us another four sets of fused masks. The whole process can be seen in Figure 5.12. We always use all 24 pseudo-mask sets, sum the pseudo-masks (pixel-wise), and then the threshold is used based on

the voting strategy. If the pixel value is greater than or equal to the number of masks that need to agree on it (either 24, 23, 22, or 21), it is set to 1 (foreground - nuclei); otherwise, it is set to 0 (background).

Together, we obtained 8 sets of fused masks.

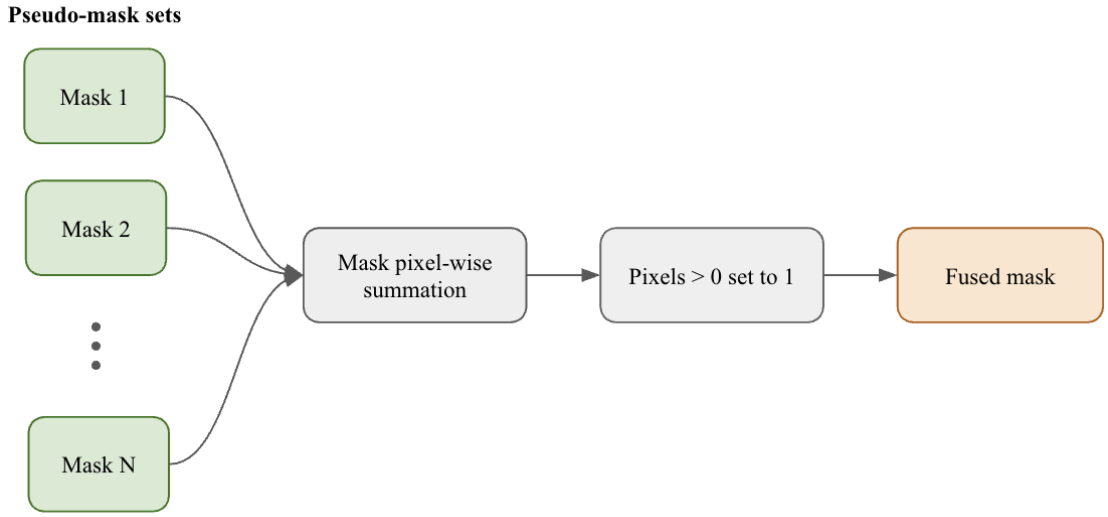


Figure 5.11: The process of pseudo-masks fusion using quartile agreement levels.

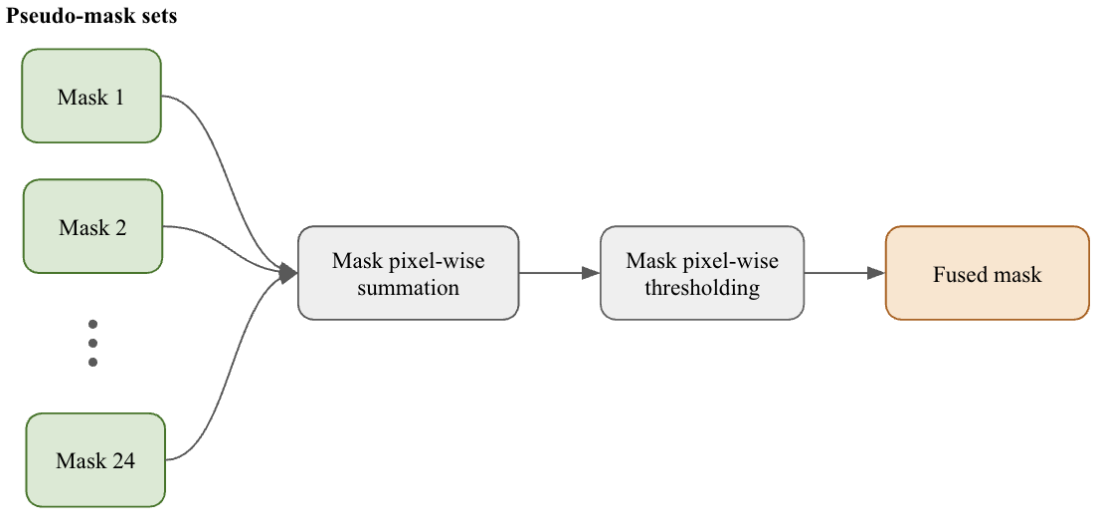


Figure 5.12: The process of pseudo-masks fusing using voting consensus.

5.8 Experiments

Our proposed system evolved with each performed experiment, since each experiment built on the previous one. We will describe each experiment in this section, with a focus on the data used for training and testing, the experiment setup and workflow, and the results of the experiment. Every experiment was run three times to reduce the risk that random initialization of the model's parameters, or the split of training data into training and validation subsets, would significantly improve or significantly worsen the final results.

For the first experiment, we utilized just the U-Net model itself as a deep learning module. The fully annotated TNBC dataset was used for this task.

The second experiment used the pseudo-masks generated by a combination of image sources and different computer vision techniques. The pseudo-masks were generated for the weakly annotated TIGER dataset from the provided bounding box annotations. These were then used as ground truth masks for subsequent training of the U-Net model, which was trained on the TIGER dataset. For the final evaluation, the TNBC dataset was used.

The third experiment builds on the second. It compared the different fusing strategies of mask sets. Again, we used these fused masks to train the U-Net model on the TIGER dataset, and the TNBC dataset was used for the final evaluation.

In the fourth experiment, we utilized a transfer learning strategy, where we took the best model trained during the third experiment and fine-tuned it using part of the fully annotated TNBC dataset.

5.8.1 Experiment 1 - Training on TNBC Dataset

In the first experiment, we wanted to check whether the small, fully annotated TNBC dataset would be sufficient for training the segmentation model.

Data We use only the fully annotated TNBC dataset, both for the training and for the final evaluation. This dataset is split into three folds, where the folds contain the following number of image patches:

- Fold 1 contains 72 image patches, from 4 patients,
- Fold 2 contains 68 image patches, from 4 patients, and
- Fold 3 contains 60 image patches, from 3 patients

Always, two folds were used as the training set, and one fold was used as the testing set. Together, this gave us three rounds of training and evaluation for one experiment run. To calculate the final result for each reported metric, the weighted average of the metrics logged by every round was calculated, given the Equation 5.2, where $\bar{M}$ is the final reported metric (Dice coefficient and IoU), n is the number of the fold that was used for evaluation, i is the i-th fold used for model evaluation, M_i is the evaluation metric calculated when evaluating the model on the i-th fold, and W_i is the size of the i-th fold.

$$\bar{M} = \frac{\sum_{i=1}^{n} M_i \cdot W_i}{\sum_{i=1}^{n} W_i} \tag{5.2}$$

The run of this experiment is simple. We train the U-Net model using the TNBC dataset, with the provided ground truth masks, and evaluate it on the same dataset.

Results The experiment ran on average around 35 epochs due to early stopping. From the training and validation loss in Figure 5.13, we can observe that the model was not able to learn when trained only on the small, although fully annotated, dataset. This can be seen both in the graph spikes on the training loss as well as on the validation loss, which was not able to improve significantly after the first 10 epochs. We see this also on the evaluation metrics, which are summarized in Table 5.1, where the Dice coefficient was 18.91% and the IoU was 10.64%. In the Figure 5.14, we can see that the model was not able to differentiate between different cell nuclei types, nor was it able to distinguish the cell nuclei and the background area.

Table 5.1: Results of the model trained on the TNBC dataset.

Metric	**Value (%)**
Dice coefficient	18.91
IoU	10.64

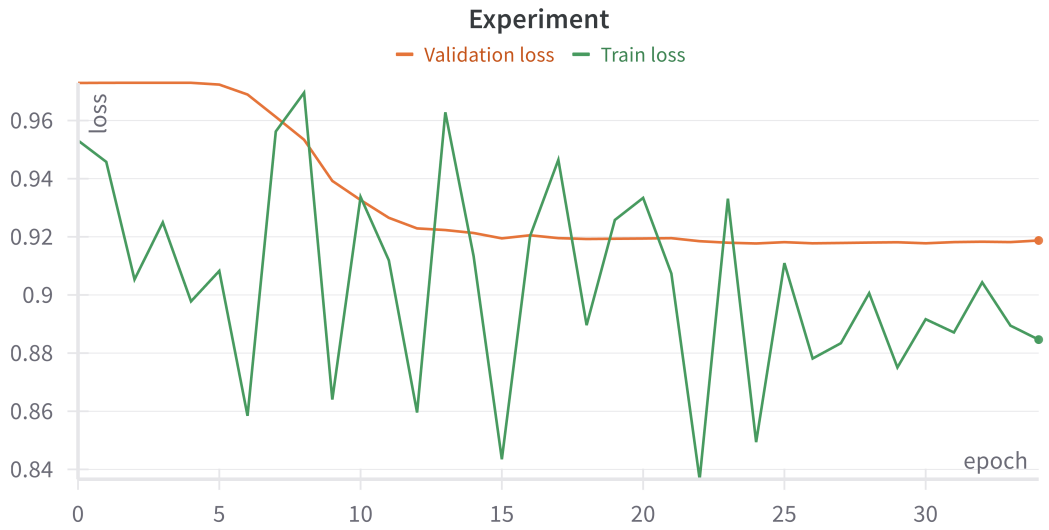
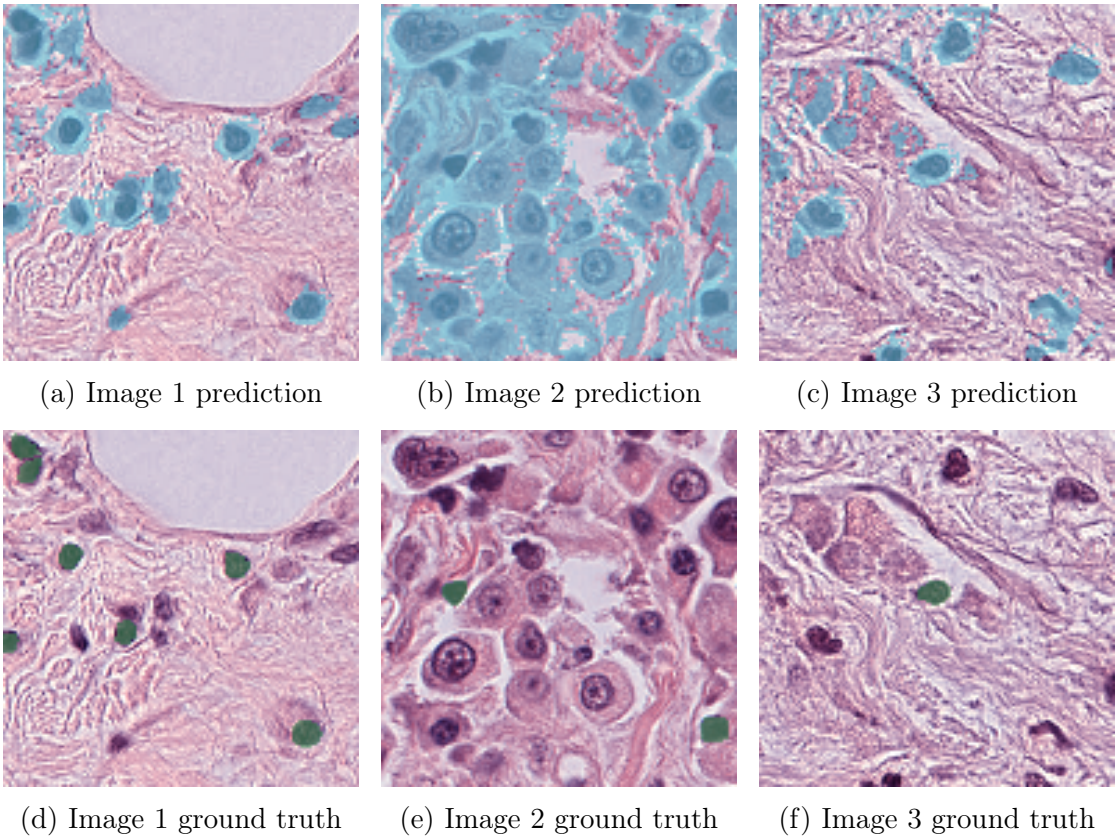


Figure 5.13: The loss function during training (green) and validation (orange) of the model trained on the TNBC dataset.



(a) Image 1 prediction (b) Image 2 prediction (c) Image 3 prediction

(d) Image 1 ground truth (e) Image 2 ground truth (f) Image 3 ground truth

Figure 5.14: Visual evaluation of model trained on the TNBC dataset. Predicted lymphocytes (cyan) and ground truth lymphocytes (green).

5.8.2 Experiment 2 - Pseudo-mask Generating Strategies

In the second experiment, we used the 24 different pseudo-masks to train 24 different models. We wanted to compare the different models based on the pseudo-mask sets they were trained on.

Data As training data, we used the TIGER image patches. Together, 19,386 image patches were used in the training set. Pseudo-mask sets were used as ground truth labels for the training of the segmentation model. We explain the generation of these sets in detail in Section 5.6. Since these image patches come with weak

annotations, we needed to evaluate the model on the TNBC dataset. For the evaluation, we used all 200 patches present in the TNBC datasets. The Dice coefficient and IoU were used as the evaluation metrics.

Results We summarized all results from this experiment in the Table 5.2. We can see that when compared to the first experiment, where we trained the same model only on the small fully annotated TNBC dataset, both the Dice coefficient and IoU improved significantly. We can also see in Figure 5.15 that the model was able to learn better and converge faster (14 epochs on average - more than 2 times faster than the model trained solely on the TNBC dataset)

From the results, we can see that the highest Dice coefficient (52.57%) and IoU (36.01%) were achieved by the model that was trained on the mask set, which was created from the histogram-equalized hematoxylin image source, where the blur was not applied and where the Otsu thresholding was used.

Next, we noticed that the histogram equalization (HE) boosts the model's performance. It consistently improved segmentation metrics across all image sources, except the ones that were using the normalized image. For example, a histogram-equalized and normalized image source with blur and adaptive thresholding reached 51.09% Dice and 34.83% IoU - over 11.5% improvement in Dice coefficient and over 8% improvement in IoU over the same non-equalized normalized pipeline (39.57% Dice, 26.78% IoU).

We also observed that pseudo-mask sets that used median blur were greatly outperformed by the ones that did not use it. This might be because the blur reduces noise on one hand, but on the other hand, it could distort cell boundaries and 'blend' the cell nuclei with its surroundings.

Overall, there was no universally dominant strategy - when deciding whether to

use only one image source, or only blur/no-blur, or a single thresholding technique, we always observed that a different strategy could overrule it. This could also be demonstrated by the following examples:

- On raw images, the best result is with no blur + Otsu thresholding (49.94% Dice), but on normalized HE images, the best is no blur + adaptive thresholding (51.09% Dice), and on hematoxylin HE, the top is no blur + Otsu thresholding (52.57% Dice).
- In the raw setting, disabling blur (49.94% Dice) beats enabling it (46.92% Dice) for Otsu thresholding, but in the hematoxylin case, enabling blur (50.39%) actually outperforms no-blur for Otsu thresholding when equalization is applied.
- For raw HE images with blur applied, adaptive thresholding (47.50% Dice) beats Otsu thresholding (44.86% Dice), whereas for normalized images with blur applied, Otsu thresholding (49.45% Dice) outperforms adaptive thresholding (39.57% Dice) without equalization.

These observations inspired us to design another generation of sets of pseudo-masks, where we fuse the original 24 masks into one - we explain the process of pseudo-masks fusion in Section 5.7 and the experiment itself in Subsection 5.8.3.

On the qualitative side, we can see in Figure 5.16 that the best model, trained on the hematoxylin image with HE, without blur and with Otsu thresholding, was able to segment cell nuclei much better than the model trained only on the small TNBC dataset. The issue here is that, although it can successfully determine whether a pixel belongs to nuclei or tissue, it cannot determine very well if the pixel should belong to the lymphocyte nuclei or some other cell type.

Table 5.2: Dice and IoU percentages for the models trained on different pseudo-mask generation strategies.

Image Source[1]	**Blurred**	**Threshold Type**	**Dice (%)**	**IoU (%)**
raw	yes	adaptive	44.18	29.28
raw	no	adaptive	46.53	31.45
raw	yes	otsu	46.92	31.60
raw	no	otsu	**49.94**	**33.77**
raw HE	yes	otsu	44.86	29.42
raw HE	no	otsu	47.28	31.91
raw HE	yes	adaptive	47.50	31.98
raw HE	no	adaptive	49.30	33.33
normalized	yes	adaptive	39.57	26.78
normalized	no	otsu	46.33	31.24
normalized	no	adaptive	48.30	32.48
normalized	yes	otsu	49.45	33.35
normalized HE	no	otsu	38.30	25.96
normalized HE	no	adaptive	47.70	32.05
normalized HE	yes	otsu	48.66	32.66
normalized HE	yes	adaptive	**51.09**	**34.83**
hematoxylin	no	otsu	41.61	27.46
hematoxylin	no	adaptive	45.18	29.77
hematoxylin	yes	adaptive	47.94	32.16
hematoxylin	yes	otsu	49.19	33.10
hematoxylin HE	yes	adaptive	48.06	32.20
hematoxylin HE	yes	otsu	50.39	34.22
hematoxylin HE	no	adaptive	52.35	35.97
hematoxylin HE	no	otsu	**52.57**	**36.01**

[1]HE: histogram-equalized

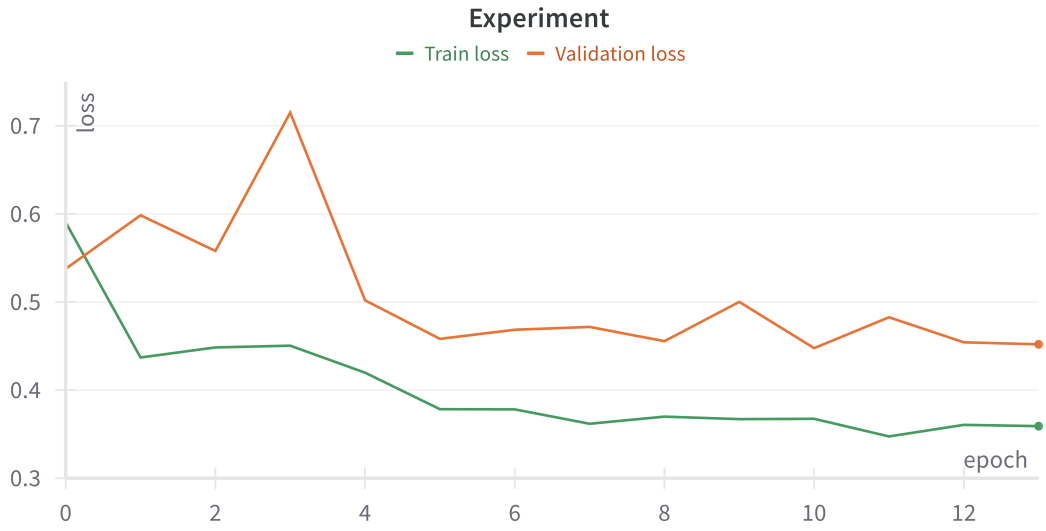
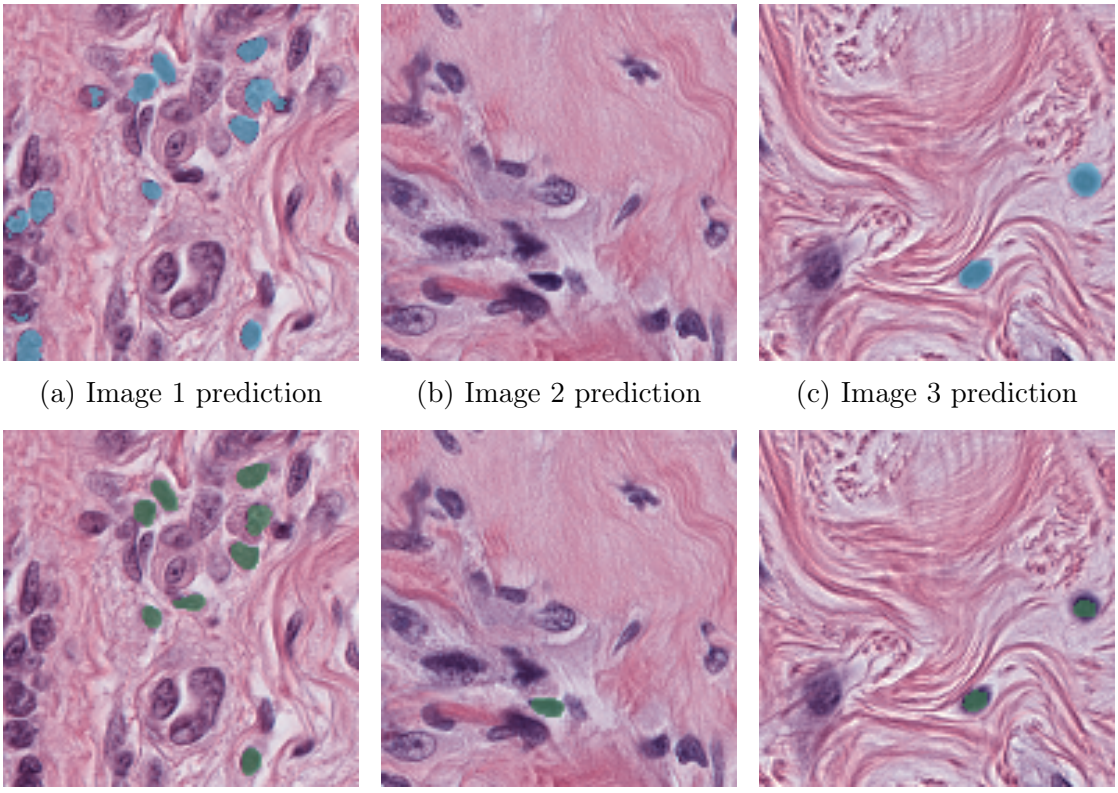


Figure 5.15: The loss function during training (green) and validation (orange) of the best model trained with the hematoxylin histogram-equalized pseudo-mask, created without blur applied and with Otsu thresholding.



(a) Image 1 prediction (b) Image 2 prediction (c) Image 3 prediction

(d) Image 1 ground truth (e) Image 2 ground truth (f) Image 3 ground truth

Figure 5.16: Visual evaluation of the best model trained with the hematoxylin HE pseudo-mask, created without blur applied and with Otsu thresholding. Predicted lymphocytes (cyan) and ground truth lymphocytes (green).

5.8.3 Experiment 3 - Pseudo-mask Fusing Strategies

The third experiment expanded further on the idea that different pseudo-mask sets can sometimes better capture the nuclei under varying conditions, as we saw in experiment 2 in Subsection 5.8.2. For this purpose, we try to combine the power of different pseudo-mask sets to create one pseudo-mask that will be used for the training. In this experiment, we want to compare different fusing strategies for the resulting pseudo-mask.

Data Similarly to Experiment 2, we work with the TIGER dataset as the training set for the model, and evaluate it on the TNBC dataset. The major change is the usage of fused masks. We present the different fusing strategies in Section 5.7. Together, there are 8 fused pseudo-mask sets, created with both best quartile agreement levels and voting consensus. The rest of the experiment setting is the same as it was in Experiment 2.

Results From the results summary presented in Table 5.3, we can clearly state that the consensus method - where either 24 out of 24, 23 out of 24, 22 out of 24, or 21 out of 24 masks voted for a pixel in a fused pseudo-mask - heavily outperformed the quartile agreement strategy, where the best 100%, 75%, 50%, or 25% of the pseudo-masks voted for a pixel (if at least one mask voted for the pixel, the pixel was set as foreground - nuclei). The order of the best pseudo-masks was determined by the previous experiment, which we describe in Subsection 5.8.2, specifically the Dice coefficient values.

The best model trained on the consensus strategy achieved a Dice coefficient of 53.53%, while the best model trained on the quartile strategy achieved a Dice of 42.93%, by 10.6% worse than the best consensus strategy model.

We also compare these two best models in terms of train and validation loss, which we can see in the Figure 5.17. There, we see that the consensus model was able to learn better during training as well. The quartile model was not able to learn that good, and the training even stopped earlier.

Finally, we compare these two models visually, in the Figure 5.18. From the figure, it is clear that the quartile model also captures cell nuclei surroundings, and is more prone to mark non-lymphocyte nuclei as lymphocyte nuclei. The consensus model is much better at segmenting only the cell nuclei, however, the same issue as in

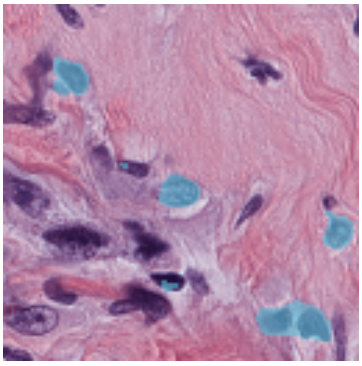
experiment 2 prevails - the model can identify cell nuclei, but it is harder for it to tell if it is a lymphocyte or non-lymphocyte nucleus.

When we compare the best model from this experiment - the consensus model (53.53% Dice) with the best model from experiment 2 - the model trained on the HE hematoxylin image, without blur and Otsu thresholding (52.57%), we see that by fusing the masks and selecting the correct fusion strategy, we were able to improve the model's performance by 0.96%.

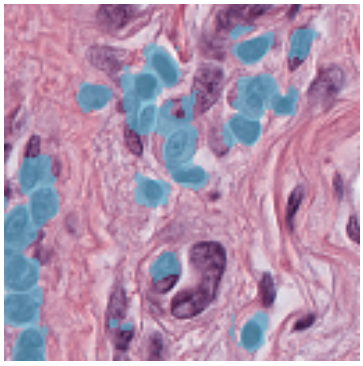
Table 5.3: Dice and IoU percentages for the models trained on different pseudo-mask fusion strategies.

Masks Set Type	**Mask Set**	**Dice (%)**	**IoU (%)**
consensus	leave 0 out	50.88	34.96
consensus	leave 1 out	**53.53**	**37.42**
consensus	leave 2 out	52.31	36.35
consensus	leave 3 out	52.58	36.33
quartile	top 100%	41.11	26.40
quartile	top 75%	40.35	25.72
quartile	top 50%	**42.93**	**27.89**
quartile	top 25%	42.59	27.61

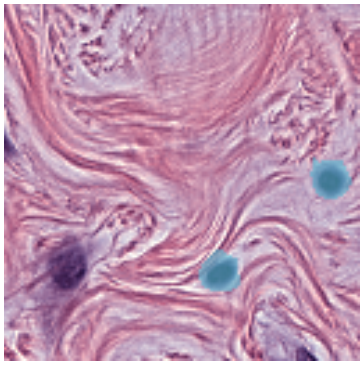
Figure 5.17: The loss function during training (green dashed) and validation (orange dashed) of the best model trained with the quartile strategy of fusing pseudo-masks, and the loss function during training (green solid) and validation (orange solid) of the best model trained with the consensus strategy of fusing pseudo-masks.



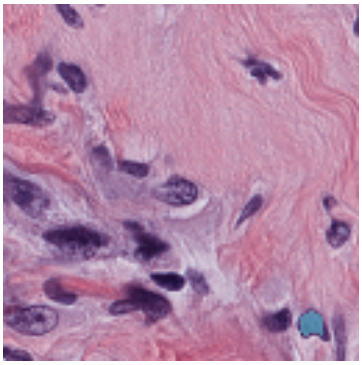
(a) Image 1 prediction - Top 50%



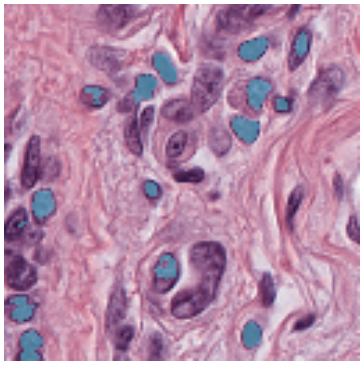
(b) Image 2 prediction - Top 50%



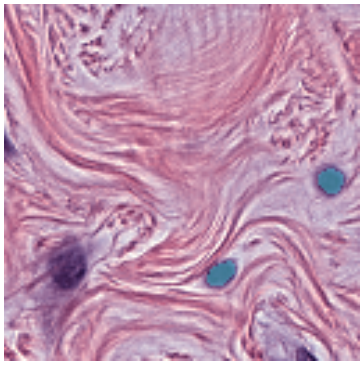
(c) Image 3 prediction - Top 50%



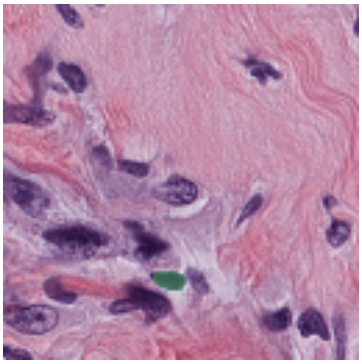
(d) Image 1 prediction - Leave 1 out



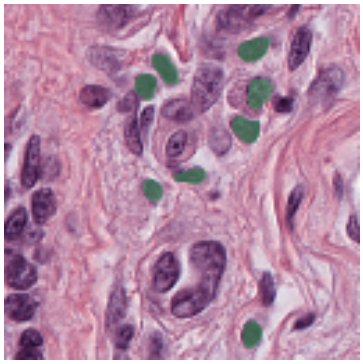
(e) Image 2 prediction - Leave 1 out



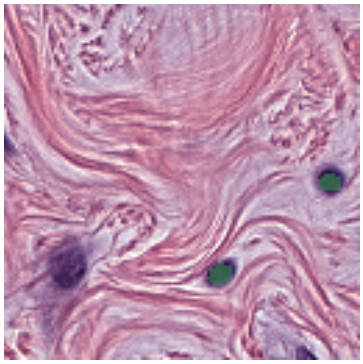
(f) Image 3 prediction - Leave 1 out



(g) Image 1 ground truth



(h) Image 2 ground truth



(i) Image 3 ground truth

Figure 5.18: Visual comparison of the best models trained with the quartile (top row) and consensus strategies (middle row), and ground truth (bottom row). Predicted lymphocytes (cyan) and ground truth lymphocytes (green).

5.8.4 Experiment 4 - Transfer Learning

In the last experiment, we wanted to build on the results of experiment 3. We decided to take the best model from that experiment and fine-tune it using a portion of data from the fully annotated TNBC dataset. We also experimented with the encoder freezing during the training to see if we could further improve the model's performance.

Data We use the model pretrained on the TIGER dataset. For the fine-tuning, we use the TNBC dataset, with the exact same 3-fold split as we used in experiment 1, which we describe in Subsection 5.8.1. We always used 2 folds for fine-tuning and 1 fold for evaluation of the model. The final evaluation metrics were computed as a weighted average of the metrics reported in each fold evaluation. We fine-tuned the model under two conditions. We tried fine-tuning it with a frozen encoder, which means that the weights of the encoder were not updated during the training, only the weights of the decoder. In the second approach, we trained the whole model, both the encoder and the decoder.

Results In the Table 5.4, we have summarized the final evaluation metrics of this experiment. We can see that when compared to the best model from experiment 3 (53.53% Dice) - the consensus model, where 23 out of 24 masks voted for a pixel - both the model with unfrozen encoder (55.01% Dice) and the one with frozen encoder (57.59% Dice) were able to slightly improve. Overall, the best model was the model with frozen encoder during the fine-tuning - it achieved a Dice coefficient of 57.59% and IoU of 41.25%.

On the graph showing the training and validation loss of the model with frozen encoder in Figure 5.19 we can see that the fine-tuning enabled the model to learn slightly more, but also that the training was very short (13 epochs) since the model

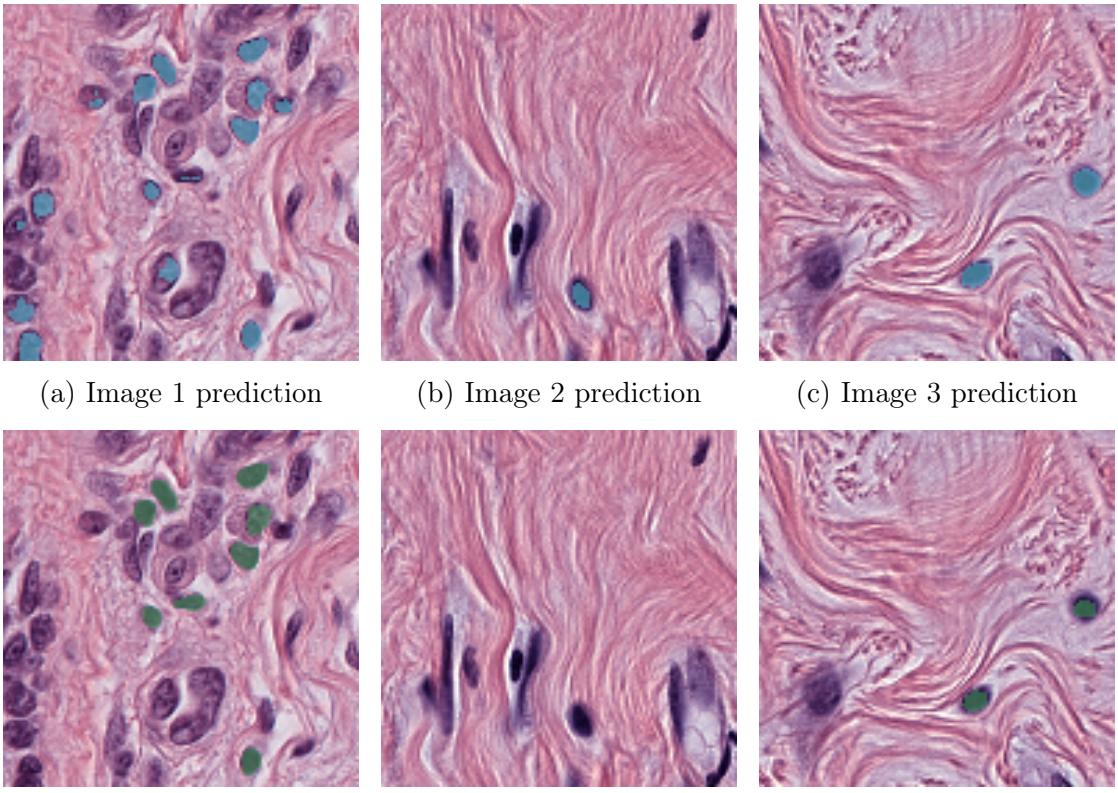
could not improve further on the validation loss.

Finally, in Figure 5.20 we observe a similar behavior to the best models from experiments 2 and 3 - that the model can recognize the cell nuclei pixels, but has a problem of differentiating between the pixels that belong to lymphocyte nuclei and non-lymphocyte nuclei.

Table 5.4: Dice and IoU percentages for the models fine-tuned on the TNBC dataset.

Encoder status	**Dice (%)**	**IoU (%)**
Unfrozen	55.01	38.6
Frozen	**57.59**	**41.25**

Figure 5.19: The loss function during training (green) and validation (orange) of the model fine-tuned with the frozen encoder.



(a) Image 1 prediction (b) Image 2 prediction (c) Image 3 prediction

(d) Image 1 ground truth (e) Image 2 ground truth (f) Image 3 ground truth

Figure 5.20: Visual evaluation of the model fine-tuned with the frozen encoder. Predicted lymphocytes (cyan) and ground truth lymphocytes (green).

5.8.5 Experiments Summary

To summarize the experiments, we decided to put the evaluation metrics of the best model from each experiment into a single Table 5.5. There, we can see that in each subsequent experiment, we were able to improve the overall performance of the corresponding model. The best model overall is the one trained on the TIGER dataset and the consensus fusion strategy, where 23 out of 24 masks voted for a pixel to be foreground (lymphocyte nuclei) and then fine-tuned on the TNBC dataset, with the encoder frozen during the training.

Table 5.5: Comparison of best Dice and IoU across experiments.

Experiment	**Best Dice (%)**	**Best IoU (%)**
1 – Training with full annotations	18.91	10.64
2 – Mask generating strategies	52.57	36.01
3 – Mask fusing strategies	53.53	37.42
4 – Transfer Learning	**57.59**	**41.45**

5.9 Tools

The whole project was written in the Python programming language version 3.12 [80]. We used Python libraries such as NumPy [81] for efficient numerical operations, and Matplotlib [82] for the image, masks, and overlay visualizations. Furthermore, we utilized the power of Jupyter Notebooks [83] to be able to run parts of code and easily explore the data during the preprocessing and pseudo-mask creation stages, and also to be able to upload data and submit training on the remote clusters.

For the image manipulation, transformation, and other computer vision operations during the preprocessing, pseudo-masks generation, and pseudo-masks fusion stages, we relied on the OpenCV Python library [84]. For the multi-target Macenko stain normalization, we used the existing implementation from [85]. The whole preprocessing and pseudo-masks creation process was executed locally on a MacBook Air with an M1 Silicon chip, with 16 GB of RAM.

Libraries and frameworks such as scikit-learn [86], PyTorch [87], PyTorch Lightning [88], and Segmentation Models PyTorch [89] were used for implementation of the classical machine learning baselines as well as for building the deep learning segmentation model to reduce the boilerplate code and make use of trusted and validated approaches implemented in those modules.

For the development and debugging, we used the PyCharm [90] integrated development environment. We also used Git [91] for version management and GitHub [92] for remote control of our project.

The training of all models was done remotely in the cloud environment, since this was the fastest and most feasible option. We utilized the Azure Machine Learning Studio [93] for this purpose. As a compute device, we used the virtual machine, which provides 24 CPU cores, 448 GB of RAM, a 2.9 TB disk, and 4 NVIDIA Tesla V100 GPUs. The computationally expensive tasks used CUDA [94] for GPU acceleration, so that the training could be completed in a shorter amount of time compared to the CPU. For easier monitoring, logging, data visualization, and the overall improved management of the whole training process and evaluation process, we used the Weights and Biases [95], where we could save, compare, and plot different trainings and runs.

Chapter 6

Conclusion

The goal of this work was to develop a hybrid approach that would be able to deal with common challenges in the automated semantic segmentation of medical images. These were, namely, a low volume of fully annotated data and a large volume of data that is only weakly annotated. The task was to segment the lymphocyte nuclei. This was also a great challenge, since many state-of-the-art works work with weak annotations in the form of bounding boxes, but their goal is to perform the nuclei segmentation, no matter the cell nuclei class, while in our work, we try to segment a specific class of nuclei - the lymphocyte nuclei. To our best knowledge at the time of writing, there is little to no available research papers on the exact specific setting we have. Two publicly available datasets were used for this task: the TIGER dataset [15] with bounding box annotations of lymphocyte nuclei, and the TNBC [78] dataset, which provided full pixel-level mask annotations of lymphocyte nuclei.

We selected the deep learning model based on the state-of-the-art work and employed known methods of traditional computer vision, such as Otsu and adaptive thresholding, and mark-controlled watershed, to prepare different sets of pseudo-

masks out of bounding box annotations. We then built further on the idea that each set could perform better on different images, so we developed different pseudo-mask fusion strategies.

In the experiments, we proved that training the model only on a very small TNBC dataset, although fully annotated, is insufficient. The same model, when trained on the larger TIGER dataset, even though with pseudo-masks used during the training, achieved superior results compared to the model trained solely on the TNBC dataset, both in terms of Dice coefficient and IoU. The model that used the best fused pseudo-mask showed improvements on both evaluation metrics as well. The final model that was pretrained on the TIGER dataset with pseudo-masks and then fine-tuned on the TNBC dataset was able to achieve the best results in Dice coefficient and IoU.

Possible future improvements may include a fully automated pipeline with an iterative self-training loop, where in the first iteration we train the model using pseudo-masks generated via the traditional computer vision pipeline, then let it predict the masks on the same dataset it was trained on, effectively creating a second generation of pseudo-labels. Then it would be retrained using the first generation of predictions, and so on, with iterative improvements of the pseudo-masks.

Chapter 7

Resumé

7.1 Úvod

V posledných rokoch ukazujú algoritmy počítačového videnia a hlbokých neurónových sietí výborné výsledky v mnohých oblastiach spracovania obrazu, napríklad detekcie, segmentácie, či klasifikácie. To má výrazný dopad na mnohé oblasti a jednou z nich je aj medicína. V medicíne sa používajú rôzne metódy, ktoré pomáhajú pri diagnostike. Jednou z takýchto metód je aj histológia, kedy je odobraný kúsok tkaniva, ktorý je následne ďalej spracovaný. Vzorky sa často zafarbujú, pre zvýšenie kontrastu medzi rôznymi štruktúrami, ako sú jadrá buniek a tkanivá. Jedným z najpoužívanejších zafarbovacích protokolov je zafarbovanie pomocou hemaoxylínu a eozínu, kde hematoxylín zafarbuje jadrá buniek do odtieňov fialovej a eozín zafarbuje okolité tkanivo do odtieňov ružovej. Takto ofarbené vzorky sa potom ďalej analyzujú.

V našej práci sa zameriavame na automatizovanú segmentáciu lymfocytov z nádorového tkaniva prsníkov, ktoré je ofarbené hematoxylínom a eozínom. Tieto lymfocyty môžu totiž slúžiť ako potenciálne biomarkery pri prognóze nádorového

ochorenia, akým je aj rakovina prsníkov. V bežnej praxi, musí kvalifikovaný patológ manuálne počítať a odhadovať množstvo a priestorové usporiadanie týchto lymfocytov, čo je časovo náročné a náchylné na chyby a nepresnosti. Algoritmy umelej inteligencie a hlbokých neurónových sietí ukázali v posledných rokoch vysoký potenciál pri spracovaní medicínskych obrazových dát. Tieto algoritmy však vyžadujú veľké množstvo presne anotovaných dát, pričom príprava týchto presných anotácií opäť spočíva na medicínskych expertoch. Preto v našej práci navrhujeme riešenie, ktoré využíva dva verejné zdroje dát:

- veľký dataset TIGER, ktorý je slabo anotovaný a poskytuje anotácie jadier lymfocytov v tvare ohraničujúcich rámčekov.
- malý dataset TNBC, ktorý je úplne anototvaný a poskytuje presné anotácie jadier lymfocytov na pixelovej úrovni.

Keďže anotácie datasetu TIGER sú vo forme ohraničujúcich rámčekov, cieľ tejto práce má dve časti:

1. Vyvinúť, implementovať a porovnať rôzne stratégie vytvárania pseudo-masiek pre dataset TIGER, za použitia metód počítačového videnia.
2. Natrénovanie segmentačného modelu hlbokého učenia, pričom pri tréningu budú použité pseudo-masky pripravené rôznymi spôsobmi a tento model bude vyhodnotený metrikami ako sú Dice koeficient a IoU.

7.2 Analýza problému

7.2.1 Počítačové videnie

Videnie je jedným z našich primárnych zmyslov, a preto je pochopiteľné, že hľadáme a vyvíjame rôzne metódy na zachytenie, uskladnenie a spracovanie obrazu.

Počítačové videnie, ako podmnožina počítačovej vedy, sa zameriava na využitie počítačov pre extrakciu zmysluplných informácií z obrazových dát, čím sa snaží napodobniť schopnosti ľudského mozgu. Medzi hlavné úlohy patrí napríklad klasifikácia, detekcia a segmentácia objektov na obrazových dátach. Tieto úlohy vieme riešiť buď tzv. tradičnými metódami počítačového videnia, s vopred zadefinovaný postupom, alebo algoritmami strojového učenia a umelej inteligencie. Medzi tradičné metódy segmetácie obrazu vieme zaradiť napríklad Otsu funkciu prahovania (ang. *thresholding*), adaptívnu funkciu prahovania, alebo watershed algoritmus.

Strojové učenie a umelá inteligencia tiež tvoria súčasť počítačového videnia. Pri klasickom programovaní sú to ľudia, kto tvorí počítačový program, zatiaľ čo pri strojovom učení necháme stroj, aby vytvoril optimálny program, pokiaľ sú mu známe vstupy a výstupy. Stroj sa učí toto mapovanie medzi vstupmi a výstupmi analýzou príznakov a vzorov vo vstupe. Hlboké učenie a špeciálne konvolučné neurónové siete sú známe veľmi dobrou schopnosťou identifikácie týchto vzorov aj v obrazových dátach.

Kvalita vstupných dát pre tieto algoritmy je veľmi dôležitá. Preto sa využívajú rôzne techniky predspracovania dát, ako prvý krok prípravy dát. V doméne digitálnych histologických snímkov sa často používa farebná normalizácia, ktorá má zmierniť výkyvy v ofarbení snímok hematoxylínom a eozínom. Medzi najpoužívanejšie normalizačné techniky patrí napríklad tzv. Macenko normalizácia.

7.2.2 Hlboké neurónové siete

Umelé neurónové siete sú inšpirované biologickými neurónmi, kedy jeden neurón prijíma vstupy zo svojho okolia a následne sa buď "aktivuje", teda vyšle signál pre ďalšie neuróny, alebo ostáva neaktívny. Základnou stavebnou jednotkou ne-

urónových sietí je teda neurón. Jedná sa o funkciu, ktorá má vstupy, a každý vstup násobí jeho prislúchajúcou váhou. Takto vynásobené vstupy sú sčítané a pripočíta sa k nim ešte prahová hodnota. Takáto funkcia je stále lineárna, preto, aby sme boli schopný riešiť aj nelineárne problémy, prechádza výsledná hodnota do tzv. aktivačnej funkcie, ktorá je nelineárna. Medzi príklady aktivačných funkcií patrí napríklad funkcia sigmoid, tanh, alebo ReLU. Neróny vedia byť následne usporiadané do vrstiev, kde neuróny z jednej vrstvy prijímajú vstupy od neurónov z prechádzajúcej vrstvy a posielajú výstupy z aktivačných funkcií ako vstup pre neuróny z nasledujúcej vrstvy. Týmto spôsobom vzniká neurónová sieť. Výstup neurónov na poslednej vrstve je predikcia siete. Táto predikcia sa potom porovnáva so skutočnou hodnotou výstupu a vyrátava sa tzv. chybová funkcia. Chyba siete je následne spätne propagovaná cez všetky vrstvy a neuróny, ktoré si následne upravujú svoje parametre (váhy a prahy) tak, aby minimalizovali chybovú funkciu. Poznáme rôzne chybové funkcie, napríklad MSE chybová funkcia alebo Dice chybová funkcia.

V priebehu rokov vznikli rôzne architektúry neurónových sietí. Medzi najznámejšie patria konvolučné neurónové siete, ktoré sú schopné zachytiť komplexné vzory v dátach s narastajúcou hĺbkou siete. Konvolučné siete využívajú konvolučné bloky spolu s ReLU aktivačnou funkciou na extrakciu relevantných čŕt z obrázku.

Jednou z architektúr konvolučných neurónových sietí je aj U-Net architektúra, ktorá dosahuje výborné výsledky pri segmentácii objektov z obrazových dát. Pri segmentácii chceme vytvoriť pixelovú masku, kde budú vyznačené pixely, na ktorých sa nachádza hľadaný objekt. Obmedzením klasických konvolučných architektúr je neschopnosť zachovať priestorové informácie po počiatočných vrstvách na extrakciu príznakov. Tento problém rieši nový typ architektúry – enkóder-dekóder architektúra. Enkóder extrahuje najviac opisné črty obrázka a komprimuje ich,

pričom znižuje redundanciu informácii a dekóder sa následne snaží z tejto zakódovanej reprezentácie obrázka naspäť poskladať pôvodný obrázok. Stratová funkcia pritom počíta rozdiel medzi pôvodným obrázkom a obrázkom vytvoreným dekóderom. Ak je dekodér schopný vytvoriť obraz veľmi podobný pôvodnému, znamená to, že skrytá reprezentácia obrazu, ktorú enkóder extrahoval, je dostatočne kvalitná. Následne je dekóder možné odstrániť a namiesto neho pripojiť modul pre klasifikáciu, segmentáciu alebo lokalizáciu, ktorý využije príznaky naučené enkóderom. V segmentačných úlohách sa do časti dekódera zavádza jednoduchá modifikácia. Namiesto generovania pôvodného obrazu sa dekóder trénuje na vytváranie segmentačnej masky, kde každý pixel nesie pravdepodobnosť príslušnosti k určitej triede. Táto vypočítaná maska pravdepodobnostnej distribúcie sa následne použije spolu s originálnou maskou v stratovej funkcii na výpočet ich rozdielu a usmernenie tréningu. U-Net architektúra tiež pozostáva z enkódera a dekódera, pričom ešte pridáva koncept tzv. preskočených spojení, kedy posledný výstup z každej vrstvy enkódera je zreťazený s prvým vstupom do každej vrstvy dekódera, ako kompenzáciu za možnú stratu čŕt, ktorá mohla nastať pri zmenšovaní aktivačnej mapy obrázka.

Medzi ďalšie varianty modelu U-Net patrí napríklad 3D U-Net, sieť, ktorá je podobná U-Net architektúre ale upravená pre prácu s 3D dátami, alebo Attention U-Net, ktorý využíva bránu pozornosti na upriamenie pozornosti siete na najdôležitejšie časti obrázka.

Ďalším typom architektúry sú tzv. Vision Transformery, ktoré využívajú mnoho blokov pozornosti na zachytenie globálneho kontextu z obrázku.

7.3 Naša práca

Táto práca predstavuje metódu, ktorá sa zaoberá výzvami, keď máme plne anotovaný súbor údajov, avšak veľmi malej veľkosti (dataset TNBC), a veľký dataset, ale slabo anotovaný (dataset TIGER). Naším cieľom je prekonať tieto výzvy implementáciou hybridného prístupu na sémantickú segmentáciu lymfocytov. Hybridný prístup sa skladá z predspracovania, ktoré pripravuje údaje na trénovanie a vyhodnotenie a zo samotného vytvárania pseudo-masiek. Výsledné výrezy obrázkov sa potom použijú na trénovanie segmentačného modelu hlbokého učenia - architektúry U-Net, s ResNet-34 enkóderom, ktorá bola predntrénovaná na ImageNet datasete. Hyperparametre modela sú nasledovné:

- Dice stratová funkcia
- Adam optimizér
- Počiatočná rýchlosť učenia 0.001, znížená faktorom 0.1 každých 5 epoch
- Tréning je nastavený na 100 epoch s predčasným zastavením, ak sa validačná chyba zhorší počas 10 kontrol, kontrola sa vykonáva po každej epoche

Najskôr sa pokúsime natrénovať a vyhodnotiť model na samotnom malom datasete. Potom použijeme techniky predspracovania a počítačového videnia na generovanie rôznych súborov pseudo-masiek z anotácií vo forme ohraničujúcich rámčekov pre slabo anotovaný dataset a natrénujeme na ňom model, ktorý opäť vyhodnotíme na malom, plne anotovanom datasete. Následne sa pokúsime identifikovať a vybrať najlepšiu stratégiu zlučovania súborov masiek, aby sme využili rôzne schopnosti súborov zachytiť oblasť bunkových jadier. Nakoniec vyberieme najlepší model (s najúspešnejšou stratégiou zlučovania masiek) a dotrénujeme ho pomocou časti údajov z plne anotovaného datasetu. Každý z týchto experimentov vyhodnotíme pomocou metrík ako Dice koeficient a IoU.

7.3.1 Datasety

Pracujeme s dvoma datasetmi. Prvým datasetom je TIGER dataset, ktorý obsahuje 1879 PNG obrázkov tkaniva rakoviny prsníkov, pod 20x zblížením, zafarbených hematoxylínom a eozínom. Obrázky sú rôznej veľkosti a pochádzajú z troch rôznych inštitútov. Dataset je anotovaný slabo, anotácie sú vo forme ohraničujúcich rámčekov.

Druhým datasetom je dataset TNBC. Obsahuje 50 PNG obrázkov (od 11 pacientov) tkaniva rakoviny prsníkov, pod 40x zblížením, zafarbených hematoxylínom a eozínom. Všetky obrázky sú veľké 512×512 pixelov a sú k nim poskytnuté anotácie vo forme pixelových masiek, kde sú anotované rôzne triedy buniek, spolu 11 tried.

7.3.2 Predspracovanie

Keďže obrazové dáta pochádzajú z rôznych zdrojov, využívame multi-cieľovú Macenko normalizáciu, pričom používame 8 obrázkov z datasetu TIGER a 2 obrázky z datasetu TNBC. Následne normalizujeme obrázky v oboch datasetoch. Obrázky v TNBC datasete ešte preškálujeme faktorom 0.5, aby sme zmenili zblíženie z 40x na 20x. Následne vytvoríme z obrázkov z oboch datasetov výrezy o veľkosti 128×128. Pokiaľ sú obrázky z TIGER datasetu príliš malé, nepoužijeme ich. Spolu nám takto vznike 19 386 výrezov z TIGER datasetu a 200 výrezov z TNBC datasetu. Na maskách z TNBC zmeníme označenie pre triedy iné ako lymfocyty na pozadie a vytvoríme takto binárnu masku. Následne aj tieto masky preškálujeme faktorom 0.5 a vytvoríme z nich výrezy.

Počas predspracovania vytvoríme aj 6 rôznych verzii obrázka, z obrázkov v TIGER datasete, konkrétne:

- Pôvodný obrázok
- Pôvodný obrázok s vyrovnaním histogramu (ang. *histogram equalization*)
- Normalizovaný obrázok
- Normalizovaný obrázok s vyrovnaním histogramu
- Hematoxylínový obrázok
- Hematoxylínový obrázok s vyrovnaním histogramu

Tieto verzie potom použijeme pri vytváraní pseudo-masiek.

7.3.3 Tvorba pseudo-masiek

Pre vytváranie pseudo-masiek používame sekvenciu metód počítačového videnia. Spolu používame 4 rôzne sekvencie, pričom každá je aplikovaná na každú verziu obrázka spomínanú vyššie. Týmto spôsobom dostaneme 24 rôznych pseudo-masiek pre jeden pôvodný obrázok. Tento proces vytvárania pseudo-masky prebieha nasledovne:

1. Obrázok je načítaný spolu s jeho anotáciami vo forme ohraničujúcich rámčekov.
2. Následne sa z obrázka vyrežú jednotlivé lymfocyty podľa ohraničujúcich rámčekov.
3. Ďalej sa aplikujú 4 rôzne sekvencie metód počítačového videnia, teda z jedného pôvodného výrezu vzniknú 4 ďalšie. Tieto sekvencie sa líšia podľa aplikovaných funkcií:
    - Otsu funkcia prahovania (ang. *thresholding*)
    - Adaptívna funkcia prahovania

- Otsu funkcia prahovania spolu s rozmazaním obrázka
- Adaptívna funkcia prahovania spolu s rozmazaním obrázka

4. V ďalšom kroku sa použije morfologické otváranie, ktoré odstráni malé artefakty ponechané po funkcii prahovania.
5. Predchádzajúce kroky nám vytvorili "prototyp" pseudo-masky. V ďalšom kroku použijeme tento prototyp pre algoritmus značkami-riadeného watershedu (ang. *mark-controlled watershed*) spolu s pôvodnou verziou obrázku. Výsledkom je hotová pseudo-maska pre oblasť jedného bukového jadra.
6. V poslednom kroku spojíme všetky pseudo-masky jednotlivých lymfocytov a vytvoríme tak veľkú pseudo-masku veľkosti pôvodného obrázku.

Takto sme spolu dostali 24 rôznych pseudo-masiek pre jeden obrázok. Ďalší spôsob vytvárania pseudo-masiek spočíval v zlučovaní týchto 24 do jednej. Tu sme použili 2 odlišné prístupy:

1. Zoradili sme masky podľa úspešnosti príslušného modelu (podľa Dice koeficientu), a následne sme spojili 100%, 75%, 50% a 25% najlepších masiek do jednej.
2. Nechali sme hlasovať všetky masky pre každý pixel. Aby mohol byť pixel označený ako popredie (jadro lymfocytu), muselo zaň hlasovať 24/24, 23/24, 22/24 alebo 21/24 masiek.

Týmito prístupmi sme získali ďalších 8 sád masiek.

7.3.4 Experimenty

Náš navrhovaný systém sa vyvíjal s každým vykonaným experimentom, pretože každý experiment nadväzoval na predchádzajúci. Každý experiment bol vykonaný

trikrát, aby sa znížilo riziko, že náhodná inicializácia parametrov modelu alebo rozdelenie trénovacích údajov na trénovaciu a validačnú sadu výrazne zlepší alebo výrazne zhorší konečné výsledky.

Pri prvom experimente sme použili len samotný model U-Net ako model hlbokého učenia. Na túto úlohu sa použil plne anotovaný TNBC dataset. Vo výsledku tohto experimentu môžeme vidieť, že trénovanie modelu len na malej dátovej vzorke, hoci plne anotovanej, je nepostačujúce. Model dosiahol Dice koeficient 18,91% a IoU 10,64%.

V druhom experimente sa použili pseudo-masky vytvorené kombináciou rôznych verzii obrázka a rôznych techník počítačového videnia. Pseudo-masky boli generované pre slabo anotovaný súbor údajov TIGER z poskytnutých anotácií ohraničujúcich rámčekov. Tie sa potom použili na následné trénovanie modelu U-Net, ktorý bol trénovaný na datasete TIGER. Na konečnú evaluáciu modelu sa použil dataset TNBC. Model natrénovaný na každej sade pseudo-masiek dosiahol výrazne lepšie výsledky oproti modelu trénovanému len na TNBC datasete. Najlepší výsledok dosiahol model trénovaný s hemtoxylínovou pseudo-maskou s vyrovnaním histogramu (ang. *histogram equalization*), kde sa nepoužilo rozmazanie a bola použitá Otsu funkcia prahovania. Dosiahol Dice koeficient 52,57% a IoU 36,01%.

Tretí experiment nadväzuje na druhý. Porovnávali sme v ňom rôzne stratégie zlučovania súborov masiek. Opäť sme tieto zlučované masky použili na trénovanie modelu U-Net na datasete TIGER a na konečné vyhodnotenie sa použil dataset TNBC. Jednoznačne lepšie výsledky dosiahol prístup zlučovania masiek prostredníctvom hlasovania, pričom najlepší spôsob hlasovania bol ten, v ktorom za každý pixel hlasovalo 23/24 masiek. Takto trénovaný model dosiahol Dice koeficient 53,53% a IoU 37,42%.

Vo štvrtom experimente sme využili stratégiu učenia s prenosom (ang. *transfer*

learning), pri ktorej sme použili najlepší model natrénovaný počas tretieho experimentu a dotrénovali ho na TNBC datasete. Experimentovali sme aj so zamrazením enkóderu počas dotrénovania. Model som zamrazeným enkóderom dosiahol najlepšie výsledky, Dice 57.59% a IoU 41.25%.

7.4 Záver

Cieľom tejto práce bolo vyvinúť hybridný prístup, ktorý by sa dokázal vysporiadať s bežnými výzvami pri automatizovanej sémantickej segmentácii lekárskych snímok. Konkrétne, výzvy predstavovali malý objem plne anotovaných dát a veľký objem dát, ktoré sú len slabo anotované. Úlohou bolo segmentovať jadrá lymfocytov. Aj to bola veľká výzva, pretože mnohé najmodernejšie práce pracujú so slabými anotáciami vo forme ohraničujúcich boxov, ale ich cieľom je vykonať segmentáciu jadier bez ohľadu na triedu bunkových jadier, zatiaľ čo v našej práci sa snažíme segmentovať špecifickú triedu jadier - jadrá lymfocytov. Podľa našich najlepších vedomostí v čase písania tejto práce nie je k dispozícii takmer žiadna výskumná práca týkajúca sa presne tohto nášho špecifického nastavenia. Na túto úlohu sme použili dva verejne dostupné datasety: TIGER [15] dataset s anotáciami ohraničujúcich rámčekov jadier lymfocytov a dataset TNBC [78], ktorý poskytoval úplné anotácie vo forme masiek jadier lymfocytov na úrovni pixelov.

Vybrali sme model hlbokého učenia na základe najnovších poznatkov a použili sme známe metódy tradičného počítačového videnia, ako sú Otsu a adaptívne prahovanie a algoritmus značkami-riadeného watershedu, na prípravu rôznych súborov pseudo-masiek z anotácií ohraničujúcich polí. Potom sme ďalej vychádzali z myšlienky, že každá sada by mohla fungovať lepšie na rôznych obrázkoch, takže sme vyvinuli rôzne stratégie zlúčenia pseudo-masiek.

V experimentoch sme dokázali, že trénovať model len na veľmi malom, hoci plne

anotovanom, datasete TNBC je nedostatočné. Ten istý model, keď bol natrénovaný na väčšom datasete TIGER, aj keď s pseudo-maskami použitými počas trénovania, dosiahol lepšie výsledky v porovnaní s modelom natrénovaným výlučne na súbore údajov TNBC, a to z hľadiska koeficientu Dice aj IoU. Model, ktorý používal najlepšie zlúčenú pseudo-masku, sa dokázal ešte viac zlepšiť v metrikách Dice koeficientu aj IoU. Konečný model, ktorý bol predtrénovaný na datasete TIGER s pseudo-maskami a potom dotrénovaný na datasete TNBC, dokázal dosiahnuť najlepšie výsledky v koeficiente Dice a IoU spomedzi všetkých modelov.

Medzi možné budúce zlepšenia môže patriť plne automatizovaný proces so samoučiacou sa slučkou, kde v prvej iterácii natrénujeme model pomocou pseudo-masiek vytvorených prostredníctvom sekvencie tradičných metód počítačového videnia, potom ho necháme predpovedať masky na tom istom súbore údajov, na ktorom bol natrénovaný, vďaka čomu vytvoríme "druhú generáciu" pseudo-masiek. Následne by sa model natrénoval na tejto druhej generácii pseudo-masiek a celý tento proces by sa iteratívne opakoval.

Bibliography

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539.

[2] Weiming Hu et al. "A state-of-the-art survey of artificial neural networks for Whole-slide Image analysis: From popular Convolutional Neural Networks to potential visual transformers". In: *Computers in Biology and Medicine* 161 (July 2023), p. 107034. ISSN: 0010-4825. DOI: 10.1016/j.compbiomed.2023.107034.

[3] Kelei He et al. "Transformers in medical image analysis". In: *Intelligent Medicine* 3.1 (Feb. 2023), pp. 59–78. ISSN: 2667-1026. DOI: 10.1016/j.imed.2022.07.002.

[4] Cédric Wemmert et al. "Deep Learning for Histopathological Image Analysis". In: *Deep Learning for Biomedical Data Analysis*. Springer International Publishing, 2021, pp. 153–169. ISBN: 9783030716769. DOI: 10.1007/978-3-030-71676-9_7.

[5] Pranab Dey. "Haematoxylin and Eosin Stain of the Tissue Section". In: *Basic and Advanced Laboratory Techniques in Histopathology and Cytology*. Singapore: Springer Nature Singapore, 2022, pp. 71–82. ISBN: 978-981-19-6616-3. DOI: 10.1007/978-981-19-6616-3_8.

[6] Freddie Bray et al. "Global Cancer Statistics 2022: GLOBOCAN Estimates of Incidence and Mortality Worldwide for 36 Cancers in 185 Countries". In: *CA: A Cancer Journal for Clinicians* 74.3 (2024), pp. 229–263. DOI: 10.3322/caac.21834.

[7] Rebecca L. Siegel et al. "Cancer Statistics, 2023". In: *CA: A Cancer Journal for Clinicians* 73.1 (2023), pp. 17–48. DOI: 10.3322/caac.21763.

[8] Charles M. Perou et al. "Molecular portraits of human breast tumours". In: *Nature* 406.6797 (2000), pp. 747–752. DOI: 10.1038/35021093.

[9] Kurt A. Schalper et al. "Influence of HER2 Expression on Prognosis in Metastatic Triple-Negative Breast Cancer". In: *ESMO Open* 7.6 (2022), p. 100381. DOI: 10.1016/j.esmoop.2022.100381.

[10] Wei Zhang et al. "Clinicopathological characteristics, treatment patterns and outcomes in HER2-positive breast cancer associated with hormone receptor status". In: *BMC Cancer* 24.1 (2024), p. 197. DOI: 10.1186/s12885-024-12974-4.

[11] R. Salgado et al. "The evaluation of tumor-infiltrating lymphocytes (TILs) in breast cancer: recommendations by an International TILs Working Group 2014". In: *Annals of Oncology* 26.2 (2015), pp. 259–271. ISSN: 0923-7534. DOI: https://doi.org/10.1093/annonc/mdu450.

[12] Carsten Denkert et al. "Tumour-infiltrating lymphocytes and prognosis in different subtypes of breast cancer: a pooled analysis of 3771 patients treated with neoadjuvant therapy". In: *The Lancet Oncology* 19.1 (2018), pp. 40–50. ISSN: 1470-2045. DOI: https://doi.org/10.1016/S1470-2045(17)30904-X.

[13] Mohamed Amgad et al. "Joint Region and Nucleus Segmentation for Characterization of Tumor Infiltrating Lymphocytes in Breast Cancer". In: *Medical*

Imaging 2019: Digital Pathology. Vol. 10956. Proceedings of SPIE. 2019, p. 109560M. DOI: 10.1117/12.2512892.

[14] KC Santosh, Nibaran Das, and Swarnendu Ghosh. "Chapter 3 - Deep learning models". In: *Deep Learning Models for Medical Imaging*. Ed. by KC Santosh, Nibaran Das, and Swarnendu Ghosh. Primers in Biomedical Imaging Devices and Systems. Academic Press, 2022, pp. 65–97.

[15] Diagnostic Image Analysis Group and International Immuno-Oncology Biomarker Working Group. *TIGER Challenge Dataset: Automated Assessment of Tumor-Infiltrating Lymphocytes in Breast Cancer*. CC BY-NC 4.0 License. 2021. URL: https://tiger.grand-challenge.org/Data/ (visited on 12/30/2024).

[16] Naylor Peter Jack et al. *Segmentation Of Nuclei In Histopathology Images By Deep Regression Of The Distance Map*. en. 2018. DOI: 10.5281/ZENODO.1175282.

[17] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. 4th. Pearson, 2018. ISBN: 9781292223049.

[18] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer International Publishing, 2022. ISBN: 9783030343729. DOI: 10.1007/978-3-030-34372-9.

[19] Sam Atallah. "Artificial Intelligence and Computer Vision". In: *Digital Surgery*. Springer International Publishing, Aug. 2020, pp. 407–417. ISBN: 9783030491000. DOI: 10.1007/978-3-030-49100-0_31.

[20] Ekram Alam et al. "Leveraging Deep Learning for Computer Vision: A Review". In: *2021 22nd International Arab Conference on Information Technology (ACIT)*. IEEE, Dec. 2021, pp. 1–8. DOI: 10.1109/acit53391.2021.9677361.

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. DOI: 10.48550/ARXIV.1505.04597.

[22] Md. Ziaul Hoque et al. "Stain normalization methods for histopathology image analysis: A comprehensive review and experimental comparison". In: *Information Fusion* 102 (Feb. 2024), p. 101997. ISSN: 1566-2535. DOI: 10.1016/j.inffus.2023.101997.

[23] Manju Dabass and Jyoti Dabass. "Preprocessing Techniques for Colon Histopathology Images". In: *Advances in Communication and Computational Technology*. Springer Nature Singapore, Aug. 2020, pp. 1121–1138. ISBN: 9789811553417. DOI: 10.1007/978-981-15-5341-7_85.

[24] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/bf02478259.

[25] Paul Werbos and Paul John. "Beyond regression : new tools for prediction and analysis in the behavioral sciences". PhD thesis. Harvard University, Jan. 1974.

[26] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0.

[27] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[28] Clement Farabet et al. "Learning Hierarchical Features for Scene Labeling". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (Aug. 2013), pp. 1915–1929. ISSN: 2160-9292. DOI: 10.1109/tpami.2012.231.

[29] Laith Alzubaidi et al. "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions". In: *Journal of Big Data* 8.1 (Mar. 2021). ISSN: 2196-1115. DOI: 10.1186/s40537-021-00444-8.

[30] Li Deng and Yang Liu, eds. *Deep Learning in Natural Language Processing*. Springer Singapore, 2018. ISBN: 9789811052095. DOI: 10.1007/978-981-10-5209-5.

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org (visited on 12/28/2024).

[32] KC Santosh, Nibaran Das, and Swarnendu Ghosh. "Chapter 1 - Introduction". In: *Deep Learning Models for Medical Imaging*. Ed. by KC Santosh, Nibaran Das, and Swarnendu Ghosh. Primers in Biomedical Imaging Devices and Systems. Academic Press, 2022, pp. 1–27. ISBN: 978-0-12-823504-1. DOI: https://doi.org/10.1016/B978-0-12-823504-1.00011-8.

[33] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. "Activation functions in deep learning: A comprehensive survey and benchmark". In: *Neurocomputing* 503 (2022), pp. 92–108.

[34] Aswathy Elma Aby et al. "Classification of acute myeloid leukemia by pretrained deep neural networks: A comparison with different activation functions". In: *Medical Engineering & Physics* 135 (2025), p. 104277.

[35] KC Santosh, Nibaran Das, and Swarnendu Ghosh. "Chapter 2 - Deep learning: a review". In: *Deep Learning Models for Medical Imaging*. Ed. by KC Santosh, Nibaran Das, and Swarnendu Ghosh. Primers in Biomedical Imaging Devices and Systems. Academic Press, 2022, pp. 29–63.

[36] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[37] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1. Atlanta, GA. 2013, p. 3.

[38] Tala Talaei Khoei, Hadjar Ould Slimane, and Naima Kaabouch. "Deep learning: systematic review, models, challenges, and research directions". In: *Neural Computing and Applications* 35.31 (Sept. 2023), pp. 23103–23124. ISSN: 1433-3058. DOI: 10.1007/s00521-023-08957-4.

[39] Yue Zhang et al. "Rethinking the Dice Loss for Deep Learning Lesion Segmentation in Medical Images". In: *Journal of Shanghai Jiaotong University (Science)* 26.1 (Jan. 2021), pp. 93–102. ISSN: 1995-8188. DOI: 10.1007/s12204-021-2264-x.

[40] Meenal V. Narkhede, Prashant P. Bartakke, and Mukul S. Sutaone. "A review on weight initialization strategies for neural networks". In: *Artificial Intelligence Review* 55.1 (June 2021), pp. 291–322. ISSN: 1573-7462. DOI: 10.1007/s10462-021-10033-z.

[41] Akibu Mahmoud Abdullahi et al. "A Comparison of Weight Initializers in Deep Learning". In: *2023 IEEE 21st Student Conference on Research and Development (SCOReD)*. 2023, pp. 97–101. DOI: 10.1109/SCOReD60679.2023.10563215.

[42] Boris T Polyak. "Some methods of speeding up the convergence of iteration methods". In: *Ussr computational mathematics and mathematical physics* 4.5 (1964), pp. 1–17.

[43] Diederik P Kingma. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[44] Fangyu Zou et al. "A sufficient condition for convergences of adam and rmsprop". In: *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*. 2019, pp. 11127–11135.

[45] Cullen Schaffer. "Overfitting Avoidance as Bias". In: *Machine Learning* 10.2 (1993), pp. 153–178. ISSN: 0885-6125. DOI: 10.1023/a:1022653209073.

[46] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[47] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning". In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359.

[48] Warren S Sarle et al. "Stopped training and other remedies for overfitting". In: *Computing science and statistics* (1996), pp. 352–360.

[49] Anders Krogh and John Hertz. "A simple weight decay can improve generalization". In: *Advances in neural information processing systems* 4 (1991).

[50] Andrew P. Bradley. "The use of the area under the ROC curve in the evaluation of machine learning algorithms". In: *Pattern Recognition* 30.7 (1997), pp. 1145–1159.

[51] Francis Sahngun Nahm. "Receiver operating characteristic curve: overview and practical use for clinicians". In: *Korean Journal of Anesthesiology* 75.1 (Feb. 2022), pp. 25–36. ISSN: 2005-7563. DOI: 10.4097/kja.21209.

[52] Tom Fawcett. "An introduction to ROC analysis". In: *Pattern Recognition Letters* 27.8 (2006), pp. 861–874.

[53] Hamid Rezatofighi et al. *Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression*. 2019. DOI: 10.48550/ARXIV.1902.09630.

[54] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020. DOI: 10.48550/ARXIV.2010.11929.

[55] Fahad Shamshad et al. "Transformers in medical imaging: A survey". In: *Medical Image Analysis* 88 (Aug. 2023), p. 102802. ISSN: 1361-8415. DOI: 10.1016/j.media.2023.102802.

[56] D. H. Hubel and T. N. Wiesel. "Receptive fields of single neurones in the cat's striate cortex". In: *The Journal of Physiology* 148.3 (Oct. 1959), pp. 574–591. ISSN: 1469-7793. DOI: 10.1113/jphysiol.1959.sp006308.

[57] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[58] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2016. DOI: 10.1109/cvpr.2016.90.

[59] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. DOI: 10.48550/ARXIV.1502.03167.

[60] Christian Szegedy et al. "Going deeper with convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015, pp. 1–9. DOI: 10.1109/cvpr.2015.7298594.

[61] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. DOI: 10.48550/ARXIV.1409.1556.

[62] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[63] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[64] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (June 2017), pp. 1137–1149. ISSN: 2160-9292. DOI: 10.1109/tpami.2016.2577031.

[65] Kaiming He et al. “Mask R-CNN”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2017. DOI: 10.1109/iccv.2017.322.

[66] M.A. Kramer. “Autoassociative neural networks”. In: *Computers & Chemical Engineering* 16.4 (Apr. 1992), pp. 313–328. ISSN: 0098-1354. DOI: 10.1016/0098-1354(92)80051-a.

[67] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.12 (Dec. 2017), pp. 2481–2495. ISSN: 1939-3539. DOI: 10.1109/tpami.2016.2644615.

[68] Nahian Siddique et al. “U-Net and Its Variants for Medical Image Segmentation: A Review of Theory and Applications”. In: *IEEE Access* 9 (2021), pp. 82031–82057. ISSN: 2169-3536. DOI: 10.1109/access.2021.3086020.

[69] Özgün Çiçek et al. *3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation*. 2016. DOI: 10.48550/ARXIV.1606.06650.

[70] Ozan Oktay et al. *Attention U-Net: Learning Where to Look for the Pancreas*. 2018. DOI: 10.48550/ARXIV.1804.03999.

[71] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762.

[72] Aayush Kumar Tyagi et al. *Guided Prompting in SAM for Weakly Supervised Cell Segmentation in Histopathological Images*. 2023. DOI: 10.48550/ARXIV.2311.17960.

[73] Mario Verdicchio et al. "A pathomic approach for tumor-infiltrating lymphocytes classification on breast cancer digital pathology images". In: *Heliyon* 9.3 (Mar. 2023), e14371. ISSN: 2405-8440. DOI: 10.1016/j.heliyon.2023.e14371.

[74] Abhishek Vahadane et al. "Structure-preserved color normalization for histological images". In: *2015 IEEE 12th International Symposium on Biomedical Imaging (ISBI)*. IEEE, Apr. 2015, pp. 1012–1015. DOI: 10.1109/isbi.2015.7164042.

[75] Xiaoxuan Zhang et al. "DDTNet: A dense dual-task network for tumor-infiltrating lymphocyte detection and segmentation in histopathological images of breast cancer". In: *Medical Image Analysis* 78 (May 2022), p. 102415. ISSN: 1361-8415. DOI: 10.1016/j.media.2022.102415.

[76] Yi Lin et al. "Nuclei segmentation with point annotations from pathology images via self-supervised learning and co-training". In: *Medical Image Analysis* 89 (Oct. 2023), p. 102933. ISSN: 1361-8415. DOI: 10.1016/j.media.2023.102933.

[77] Yixiong Liang et al. "Weakly Supervised Deep Nuclei Segmentation With Sparsely Annotated Bounding Boxes for DNA Image Cytometry". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 20.1 (Jan. 2023), pp. 785–795. ISSN: 2374-0043. DOI: 10.1109/tcbb.2021.3138189.

[78] Naylor Peter Jack et al. *Extention to the TNBC dataset, brain section and cell type*. en. 2021. DOI: 10.5281/ZENODO.3552674.

[79] Desislav Ivanov, Carlo Alberto Barbano, and Marco Grangetto. *Multi-target stain normalization for histology slides*. 2024. DOI: 10.48550/ARXIV.2406.02077.

[80] Python Software Foundation. *Python Language Reference, version 3.12*. 2024. URL: https://docs.python.org/3.12/reference/.

[81] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. ISSN: 1476-4687. DOI: 10.1038/s41586-020-2649-2.

[82] John D. Hunter. "Matplotlib: A 2D Graphics Environment". In: *Computing in Science &; Engineering* 9.3 (2007), pp. 90–95. ISSN: 1521-9615. DOI: 10.1109/mcse.2007.55.

[83] Kluyver Thomas et al. "Jupyter Notebooks - a publishing format for reproducible computational workflows". In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. IOS Press, 2016. DOI: 10.3233/978-1-61499-649-1-87.

[84] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[85] Carlo Alberto Barbano and André Pedersen. *EIDOSLAB/torchstain: v1.2.0-stable*. 2022. DOI: 10.5281/ZENODO.6979540.

[86] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[87] Jason Ansel et al. "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation". In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. ASPLOS '24. ACM, Apr. 2024, pp. 929–947. DOI: 10.1145/3620665.3640366.

[88] William Falcon et al. *PyTorchLightning/pytorch-lightning: 0.7.6 release*. 2020. DOI: 10.5281/ZENODO.3828935.

[89] Pavel Iakubovskii. *Segmentation Models Pytorch*. https://github.com/qubvel/segmentation_models.pytorch. 2019.

[90] JetBrains. *PyCharm*. 2010. URL: https://www.jetbrains.com/pycharm/ (visited on 04/10/2025).

[91] *Git*. 2005. URL: https://git-scm.com/ (visited on 04/10/2025).

[92] *GitHub*. 2008. URL: https://github.com/ (visited on 04/10/2025).

[93] *Azure Machine Learning Studio*. 2020. URL: https://ml.azure.com/ (visited on 05/06/2025).

[94] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. *CUDA, release: 10.2.89*. 2020. URL: https://developer.nvidia.com/cuda-toolkit.

[95] L. Biewald, C. Van Pelt, and S. Lewis. *Experiment Tracking with Weights and Biases*. 2017. URL: https://www.wandb.com/ (visited on 04/10/2025).

Appendix A

Plan of Work

Bachelor's thesis evidence number: FIIT-100241-116291

A.1 Winter Semester

In Table A.1, we can see a summarized plan of work for the winter semester. During this time, we familiarizing ourselves with the whole topic of weak segmentation and traditional methods of computer vision. We also explored the available datasets, and after we selected the TIGER dataset, we explored it in more depth. We were studying the state-of-the-art work, gathering information, and running preliminary experiments. Later, we constructed an initial concept of experiments that should be performed. The experiments were performed more slowly than we anticipated because of the complicated dataset features, which caused us a slight delay, but on the other hand, we were now very confident in the dataset usage and knew better what we should do next to be successful.

Table A.1: Plan of Work for Winter Semester

Week	**Planned Work**
1-2	Literature review on digital pathology and TIL detection.
3-6	Study of deep learning techniques and architectures (CNNs, U-Net, Vision Transformers) and different pseudo-label generation techniques (GrabCut, watershed, Otsu).
7	Familiarization with the TIGER dataset.
8-10	Initial development of the pipeline to convert bounding box annotations to pixel masks. Preparing the mid-term report.
11-13	Preparing mid-term report, finishing analysis, summarizing progress and findings.

A.2 Summer Semester

The Table A.2 displays the plan for the summer semester. During this time, we had a large portion of work to do. We selected a TNBC dataset, which was fully annotated, as another dataset to be included in this work. We had to implement 24 different strategies for pseudo-mask generation and then perform the model training on all of them, and select the best one. We also came up with different methods of generating pseudo-masks (fusing the original 24 pseudo-masks), meaning that we had another large portion of experiments to perform. Then, a transfer learning approach idea came for the final model, which was the most successful. During the whole semester, we performed almost 600 trainings, and in comparison to the winter semester, the whole work has picked up speed. As we mentioned, we were adding more experiments as the work progressed. We were able to successfully build the pipeline of computer vision operations that generated different pseudo-masks, prepare the U-Net segmentation model, and were able to train it in the Azure cloud environment.

Given the many challenges we had to overcome, like the weakly annotated TIGER dataset, a very small TNBC dataset, inconsistencies across the datasets, and the

challenging task of segmenting only a specific class of cell nuclei (not all cell nuclei), we conclude that this plan was fully adhered to and completed.

Table A.2: Plan of Work for Summer Semester

Week	**Planned Work**
1-2	Continuing with the computer vision pipeline development.
3	Testing and refining the pseudo-label generation algorithm.
4-8	Implementing, training, and evaluation of the U-Net segmentation model trained with various pseudo-masks. Analysis of the model using evaluation metrics (IoU, Dice coefficient).
9	Final experiments, preparation of the final thesis outline.
10-12	Writing and presenting the final report with results and conclusions.
13	Submission of final thesis.

Appendix B

Technical Documentation

B.1 Project's folder structure

```
.
├── configs/
┐   ├── azure_connect_example.json
┐   ├── azure_job.yaml
┐   ├── azure_upload_data.yaml
┐   ├── model_train_base.yaml
┐   └── paths.yaml
├── data/
┐   ├── processed_tiger/
┐   ├── processed_tnbc/
┐   ├── raw_tiger/
┐   ┐   ├── images/
┐   ┐   └── coco_annotations_placeholder.json
┐   └── raw_tnbc/
┐
```

```
        ├── images/
        └── masks/
├── example_data/
    ├── processed_tiger/
    ├── processed_tnbc/
    ├── raw_tiger/
        ├── images/
        └── tiger-coco.json
    └── raw_tnbc/
        ├── images/
        └── masks/
├── models/
├── src/
    ├── azure/
        ├── azure_conda.yaml
        ├── azure_train.ipynb
        └── azure_upload_data.ipynb
    ├── models/
        ├── inference.ipynb
        ├── model_factory.py
        └── til_dataset.py
    ├── image_preprocessor.py
    ├── image_stats.py
    ├── mask_generator.py
    ├── sample.py
    └── utils.py
├── .amlignore
```

```
├── .gitignore
├── main.ipynb
├── main.py
├── README.md
└── requirements.txt
```

B.2 Description of folders and files

B.2.1 `root` directory

.amlignore Here is a list of folders and files that are ignored when a job is submitted to the Azure ML platform. When a job is submitted, Azure takes a snapshot of the directory it is given as the source directory. The files listed in `.amlignore` will be ignored by this operation.

.gitignore Files to be ignored by the Git versioning system.

main.ipynb In this Jupyter notebook, the whole preprocessing, pseudo-mask generating, and pseudo-mask fusing pipeline can be run. It also provides the visualizations of preprocessed images and pseudo-masks. This notebook has already been executed, so you can also see the outputs of each cell. Open the `main.ipynb` to see it.

main.py This is the main training and evaluation script. It can be run both locally and on the Azure ML platform. See Section B.4 to see both possible options.

README.md In this file, the document with the same content as in this Chapter is.

requirements.txt Contains Python dependencies that need to be installed to run the project.

B.2.2 `config` directory files

azure_connect_example.json Contains the information required to authenticate and connect to your Azure ML Workspace. You will need to fill out this configuration file, otherwise, the connection will not be successful. Keep the structure, just change the values to match your account.

azure_job.yaml Contains all configuration values that are used to submit the job run, which will train the model. These include the data asset information, the environment information (environment where the job will run), the job information (like source directory to push to Azure ML, compute target, etc.), and the arguments to be passed to the `main.py` function once it is executed.

azure_upload_data.yaml Here is the information about the folder you want to upload to the Azure storage, the destination folder on Azure ML, and options to overwrite already existing files and see the progress of the whole process.

model_train_base.yaml Contains the hyperparameters that are used by the model during the training and evaluation. It also contains the option to load the pre-trained model.

paths.yaml This file contains all paths, or parts of paths, where the images are being stored, created, modified, and updated, and from which are loaded during the preprocessing. The whole folder structure for the preprocessing and pseudo-mask creation is created in the `./main.ipynb` notebook, where the full paths are built. Note that by default, all file manipulations are performed under the

`/example_data` directory (unless changed in this config). For the demo, we recommend keeping this configuration file as is. For the real preprocessing, we advise changing the `root_data_dir` value to point to the `/data` directory.

B.2.3 `data` and `example_data` directories

The `data` directory contains four main subdirectories. Here the images and annotations of the respective datasets should reside (TIGER[1] and TNBC[2] datasets). We do not include the actual images and masks here, because of their large size, but when running a real preprocessing, you should place them here and change the `root_data_dir` value in the `/configs/paths.yaml` file to point to the `/data` directory. The `/data/raw_*` folders contain the raw images and annotations (bounding box for TIGER - in the COCO JSON format[3], PNG masks for TNBC). The `/data/preprocessed_*` directories contain more subdirectories that are created during the run of the `./main.ipynb` notebook. The most important ones are:

- `/data/preprocessed_*/patches/images` which contains the 128×128 normalized image patches
- `/data/preprocessed_*/patches/masks` which contains the 128×128 mask (or pseudo-mask) patches
- `/data/preprocessed_tnbc/patches/folds` which contains TNBC image and mask patches, but split into folds, where each fold directory has `/data/preprocessed_tnbc/patches/folds/fold_*/images` and `/data/preprocessed_tnbc/patches/folds/fold_*/masks` folder

Note that this directory is meant to be used for real preprocessing, and you need

[1] `https://tiger.grand-challenge.org/Data/`
[2] `https://zenodo.org/records/3552674`
[3] `https://roboflow.com/formats/coco-json`

to put here the correct images and annotations yourself.

The `/example_data` directory follows the same structure, but already contains 10 example images from the TIGER dataset in the `/data/raw_tiger/images` subdirectory, the `tiger-coco.json` file with the TIGER bounding box annotations in the `/data/raw_tiger` subdirectory, and 4 images from the TNBC dataset in the `/data/raw_tnbc/images` subdirectory and their corresponding masks in the `/data/raw_tnbc/masks` subdirectory. This directory is by default listed as the `root_data_dir` in the `/configs/path.yaml` file, so in order to run the Demo you do not need to change anything in the `/configs/path.yaml` file.

B.2.4 `models` directory

Here, the models that you wish to save and use for future fine-tuning or reference should be placed. We do not include any pre-trained model here, since the `.ckpt` files are around 300MB in size.

B.2.5 `src/azure` directory

azure_conda.yaml Defines the dependencies that will be installed within Azure ML environment.

azure_train.ipynb From this Jupyter notebook, the training is managed. This involves authenticating, pulling the correct data asset path, creating the environment, and submitting the job to Azure ML. Use the `/configs/azure_connect_example.json`, `/configs/azure_job.yaml` and `/configs/model_train_base.yaml` to manage the configuration of parameters.

azure_upload_data.ipynb This Jupyter notebook is used to upload a locally stored folder to the remote Azure ML data storage. Use the

`/configs/azure_upload_data.yaml` to manage the configuration of parameters.

B.2.6 `src/models` directory

inference.ipynb In this Jupyter notebook, you run the inference of the model. A pretrained model is required for this stage. The predictions are visualized. The inference is run on the `tnbc_sample_img_patch` image from the `/configs/paths.yaml` configuration file.

model_factory.py Here we define the architecture of the model.

til_dataset.py This file defines a utility class that is used to output the image and mask pairs that are further used during the training and evaluation by the PyTorch `DataLoaders`.

B.2.7 `src/*.py` files

image_preprocessor.py Contains the `ImageProcessor` class that groups all the preprocessing functionalities.

image_stats.py Contains the `ImageStats` class that prints the statistics of the folder that contains images, like average image height, width, area, etc.

mask_generator.py Contains the `MaskGenerator` class that groups all the mask-generating and mask-fusing functionalities.

sample.py Contains the `Sample` class that is used for visualization of the image and its mask.

utils.py Contains all other utility functionalities, for example, for opening and loading `.json` and `.yaml` files.

B.3 Installation guide

B.3.1 Prerequisites

Below, we list the necessary software requirements:

- Python version 3.12+
- `pip` version 23.2+
- Internet access to download packages
- Weights and Biases account for model logging
- Azure ML access (if you wish to train models there)

B.3.2 Clone the repository

Clone this repository and navigate into it:

```
git clone <repository-name>
cd <cloned-repository-name>
```

Alternatively, you can download the `.zip` file of this project, unpack it, and open a terminal within it.

B.3.3 Set up the Python environment

Set up the virtual environment using `pip` (or create Conda environment, but we will be using `pip`).

Using MacOS:

```
python3 -m venv .venv
source .venv/bin/activate
```

or Windows (from PowerShell):

```
python -m venv .venv
.\.venv\Scripts\Activate.ps1
```

B.3.4 Install dependencies

Dependencies are listed in the `requirements.txt` file. To install them all, use:

```
pip install -r requirements.txt
```

B.4 How to run the Demo

Here we present a way to run the demo version (using the demo data placed in the `/example_data` folder). Be aware of the fact that since we only have 10 training images and 4 testing images in this demo, the model's performance will not be representative of real-world results. This is just to showcase how the project works. Also, note that our project works with the PNG images only.

B.4.1 Preprocessing and pseudo-mask creation

1. Navigate to the `./main.ipynb` Jupyter notebook. You will notice that the notebook has already been executed (for the demonstration). Feel free to examine it before trying to run anything.

2. Next, make sure that you clear all outputs (to avoid any confusion) and start running it cell after cell (or all at once). You will notice that under the `/example_data/processed_*` directories, different subdirectories will appear. Those will be populated with different images or versions of images and masks during the preprocessing and pseudo-mask creation. During the execution of the cells, you will also see the textual and visual output responses.

3. After the whole notebook is executed, feel free to examine the different subdirectories that were created - but be careful not to delete, move, or rename any of them or their contents.

4. The data is now prepared for the training.

B.4.2 Training on Azure

Here we describe the necessary steps that are required to be able to train the model on the Azure ML platform.

1. Ensure you have access to an Azure ML workspace and all the required information. Fill them in the `/configs/azure_connect_example.json` configuration file.

2. Ensure that the information in the `/configs/azure_upload_data.yaml` configuration file is correct. You will need to input the correct `target_path` as this is not provided by us!

3. Then navigate into the `./src/azure/azure_upload_data.ipynb` and run it cell by cell. Be especially careful with the local and remote directory paths. The contents of the local directory will be copied into the remote directory.

4. After the data has been uploaded, you will need to create the Azure Data

Asset. See the official Azure documentation[4] how to do it.

5. Then you will need to create an Azure compute instance. See this official Azure documentation[5] for precise instructions.

6. Next, you will need to modify `/configs/azure_job.yaml` file, as we cannot provide defaults for certain variables:

   - See the `dataset` top-level key. You need to input the `name` of the Data Asset and its `version` you created in Step 4.
   - See the `job` top-level key. You need to change the `job.compute` to have the name of the compute instance target you created in Step 5.
   - See the `jobs.args.wandb` key. You will need to input your Weights and Biases key, so the training and evaluation process can be monitored. See the official guide[6] on how to get the key.

7. *(Optional)* If you wish, you can try to change the model parameters; you can do so in the `/configs/model_train_base.yaml` file, but this step is optional.

8. Now navigate to the `./src/azure/azure_train.ipynb` and follow the instructions within it to submit the training and evaluation job to the Azure ML platform.

9. During the training, you can see and monitor the whole process in your Weights and Biases account.

10. After the training and evaluation are done, look for the `outputs` folder in

[4] `https://learn.microsoft.com/en-us/azure/machine-learning/how-to-create-data-assets`

[5] `https://learn.microsoft.com/en-us/azure/machine-learning/how-to-create-compute-instance`

[6] `https://docs.wandb.ai/support/find_api_key/`

the job details on the Azure ML platform. It should be in the *Outputs + logs* tab, but the Azure ML platform UI changes constantly.

11. You can download the trained model from the `outputs/checkpoints/best.ckpt`. Be aware that the checkpoint file is around 300MB in size.

B.4.3 Training locally

This option presents a way to run the training and evaluation locally. Note that the Demo will work just fine, since there is only a fraction of the size of a real dataset, but when training with a large dataset, the time to train the model locally can be significantly longer.

Follow these steps:

1. *(Optional)* If you wish, you can try to change the model parameters in the `/configs/model_train_base.yaml` file, but this step is optional.
2. Run the `main.py` script. Be sure to input your correct Weights and Biases key. See the official guide[7] on how to get the key.

```
python3 main.py \
  --data_path './example_data' \
  --wandb '<your-wandb-key>' \
  --train_images_path 'processed_tiger/patches/images' \
  --train_masks_path
  ↪  'processed_tiger/patches/masks/fused_leave_1_out' \
  --test_images_path 'processed_tnbc/patches/images' \
  --test_masks_path 'processed_tnbc/patches/masks'
```

[7] https://docs.wandb.ai/support/find_api_key/

3. During the training, you can see and monitor the whole process in your Weights and Biases account.

4. Once the training finished, you will notice that a new `/outputs` directory was created. This contains both the trained model in the `/outputs/checkpoints/best.ckpt` file and the raw Weights and Biases logs in the `outputs/wandb` folder. Furthermore, it contains a `outputs/test_results.json` with the evaluation metrics from the evaluation phase.

B.4.4 Inference

To see how the model works during inference, navigate to the `./src/models/inference.ipynb`. Notice that this notebook has already been executed as well; feel free to examine it and then clear the outputs (to avoid any confusion). You will need to input the path to the trained model `.ckpt` file, as we do not provide a trained model in the demo. Run the notebook and see the results!