# Subtyping Lymphoma Whole Slide Images with Deep Learning

Master's Thesis by

Ferdinand Tölkes

University of Regensburg
Institute for Theoretical Physics

Supervision by

Prof. Dr. Andreas Schäfer
Prof. Dr. Michael Altenbuchinger

March 27, 2025

**Abstract**

Automating the classification of lymphoma subtypes from Whole Slide Images (WSIs) is a key challenge in Computational Pathology (CPath). In this work, I present a data preprocessing pipeline for extracting image patches from WSIs stored in a standardized .sqlite format. My Python package, Patchcraft (PC), facilitates the sampling of arbitrarily sized square patches as individual files while enabling direct application of data augmentations. To generate feature embeddings, I employ a pre-trained Vision Transformer (ViT), specifically the UNI model introduced by Chen et al., 2023, trained on over 100 million tissue patches. These embeddings serve as input to a lightweight Residual Network (ResNet) classifier, trained at the patch level with weighted loss and label smoothing regularization. I systematically analyze the impact of image resizing and augmentations on both embeddings and attention maps generated by the UNI model. For the classification of WSIs, I adopt an ensemble-based majority voting approach using five-fold cross-validation. My study focuses on six lymphoma subtypes: Hodgkin's Lymphoma (HL), Diffuse Large B-Cell Lymphoma (DLBCL), Chronic Lymphocytic Leukaemia (CLL), Follicular Lymphoma (FL), Mantle Cell Lymphoma (MCL), and Lymphadenitis (LTDS). The model achieves 93% accuracy when trained and evaluated on a dataset from a pathology lab in Kiel. However, its performance drops to approximately 48% on an Out-Of-Distribution (OOD) dataset from another pathology lab in Munich, highlighting challenges possibly related to systematic differences between the datasets and the need for improved generalization strategies.

# Contents

# List of Figures



# List of Tables

# List of Acronyms

# 1 Introduction

Pathologists diagnose diseases based on tissue sections, but this task is becoming increasingly challenging due to a rising workload and a shortage of specialists. In Germany, there are relatively few histopathology experts responsible for annotating cancerous tissues. Compared to other European countries, Germany has the second lowest density of pathologists relative to the total population, and the proportion of physicians specializing in pathology is the lowest in Europe and significantly lower than in the United States or Canada (Märkl et al., 2021). This shortage is part of a broader global trend: histopathology faces increasing workload demands, rising complexity, and a declining number of professionals (Walsh and Orsi, 2024).

Advancements in whole slide imaging technology over the past decade have significantly expanded the availability of digital pathology data. However, this also increases the need for efficient annotation and analysis. This raises an important question: can computational methods help alleviate the growing burden on histopathologists? Deep learning has demonstrated remarkable success in image classification tasks, starting with AlexNet (Krizhevsky et al., 2012), which achieved groundbreaking performance in the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al., 2015). Given these advancements, applying deep learning to Whole Slide Image (WSI)-based lymphoma classification is a promising research direction.

WSIs are high-resolution, digitized versions of histopathology glass slides used for disease diagnosis, particularly in biopsy-based tissue analysis. Traditionally, pathologists examined thin tissue slices under a microscope. With whole slide imaging, these slides are now digitized at high magnification, preserving microscopic details at multiple scales (Kumar et al., 2020). This digitization enables the application of deep learning models to assist in disease classification. In this work, I focus on classifying six lymphoma subtypes: Hodgkin's Lymphoma (HL), Diffuse Large B-Cell Lymphoma (DLBCL), Chronic Lymphocytic Leukaemia (CLL), Follicular Lymphoma (FL), Mantle Cell Lymphoma (MCL), and Lymphadenitis (LTDS). The high prevalence of these subtypes and the clinical importance of their classification are well documented (Rozman and Montserrat, 1995; Smith et al., 2015).

Despite its promise, lymphoma classification using WSIs presents significant challenges at both the human and algorithmic level. WSIs are gigapixel-scale images, making direct processing at reasonable resolution infeasible on most modern high-performance GPUs. In addition, artifacts and color variability from experimental staining of tissues required to visualize and distinguish cells challenge model development. Other challenges include different image formats due to a variety of scanners, low data availability due to the proprietary nature of many datasets, and the high effort required to obtain detailed annotations (Dimitriou et al., 2019; Wang et al., 2019). The scarcity of publicly available datasets further complicates model development. Moreover, to achieve broader accessibility, it is critical to explore classification approaches that do not rely on the highest resolution slide data, potentially enabling diagnosis from images of tissue samples captured with smartphones. Such advances could have a significant impact on democratizing healthcare as well as healthcare accessibility in developing regions (Brinkel et al., 2014; Hernández-Neuta et al., 2019).

This thesis contributes to the larger research project 'Federated Learning in Lymphoma Pathology', funded by the German Ministry of Education and Research. One of the goals of this project is to support pathology labs all over Germany with a deep learning model that learns from all of these labs in a

federated way, i.e. without violating German data protection laws. As part of this effort, I focus on solving the classification problem using data from a single pathology lab. My work addresses the technical challenges of classification, while medical expertise in histopathology remains outside the scope of this thesis.

The structure of this thesis is as follows: In section 2, I describe the data preprocessing pipeline, including patch extraction and an overview of the datasets. Section 3 introduces fundamental deep learning concepts, the Vision Transformer (ViT) architecture, and the specific techniques used for classification. In section 4 I first analyze the pre-trained ViT model I will use in section 4.1 before introducing my data processing pipeline and ensemble classifier. I then evaluate its performance and assess the impact of different processing choices in section 4.2. Finally, section 5 discusses my findings and their implications.

# 2 Preprocessing

The dataset used in this study consists of WSIs of lymph nodes, obtained either through a whole lymph node biopsy or a needle biopsy. A whole lymph node biopsy is an excisional procedure that requires an invasive approach, whereas needle biopsies are less invasive and more cost-effective. However, needle biopsies yield significantly less tissue, making diagnoses more challenging (Frederiksen et al., 2015). After extraction, the tissue is sectioned into thin slices and stained with various immunohistochemical markers, which highlight distinct slide characteristics. Throughout this thesis, I focus on the Hematoxylin and Eosin (HE) stain, which enhances the fine structures of cells and tissues. It is the most widely used stain in histopathology, as pathologists typically begin their analysis with it. Other stains are applied depending on the specific features a pathologist needs to highlight for diagnosis. My emphasis on the HE stain is due to its fundamental role in histopathology. For a more detailed overview of HE staining and its process, see Gurcan et al., 2009, Chan, 2014, and Feldman and Wolfe, 2014.

Making the digitized versions of the WSIs accessible and ready for neural network training is a two-step process. First, the Pamly preprocessing tool is used to bring the scanned slides into a uniform data format (Huttner, 2023). Second, I developed the Patchcraft (PC) package, which enables the sampling of patches and provides PyTorch[1] tensors ready for neural network training. These steps are described in detail in the following sections.

## 2.1 Preprocessing Whole Slide Images with Pamly

I employ Pamly to transfrom WSIs scanned with a variety of proprietary scanners into a standardized data format based on SQLite databases.

Pamly partitions the slide into tiles of fixed size $s$ and pads the slide to create a square with a side length of $s \cdot 2^n$, where $n$ is a positive integer. This ensures that recursively merging four smaller tiles into a larger one ultimately produces a tile that matches the padded square. This structure is crucial for the file format in which the slides will be saved. Pamly retains only tiles of size $s$ at full resolution, while tiles representing larger regions are stored at progressively lower resolutions. Next, Pamly applies edge and brightness detection to filter out unsuitable tiles, such as those containing excess fat. It also removes small, isolated tile groups (referred to as 'islands') that are disconnected from larger, contiguous regions. The left side of fig. 1 illustrates an example of a

[1] Paszke et al., 2019.

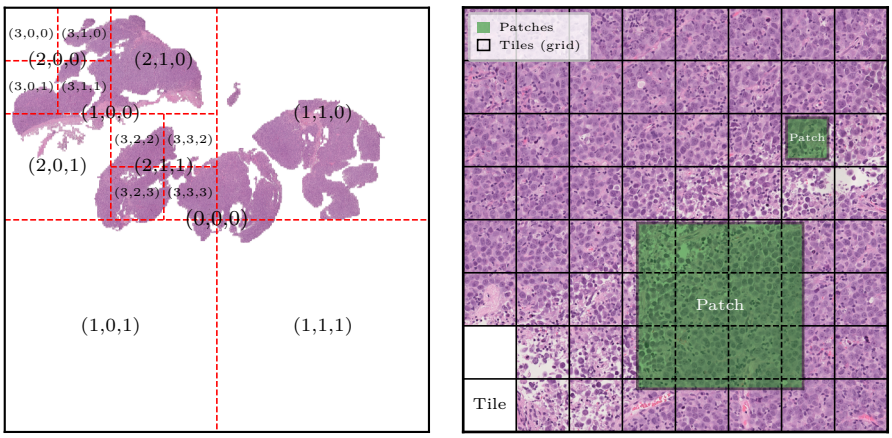



Figure 1: Visualization of a quadtree structure used to recursively divide WSIs. The slide shown has been preprocessed using Pamly. The tuples represent the zoom level and the x- and y-coordinates of the corresponding tile. Each tile is stored with a fixed side length of 512 pixels, resulting in different resolutions at each zoom level (left). Visualization of tiles and patches: Tiles have a fixed size, while patches are arbitrary square subimages that can span multiple tiles. Only the highest-resolution tiles are displayed or labeled here (right).

preprocessed WSIs, highlighting the quadtree-based tile structure.

The WSI is finally stored as an SQLite file consisting of multiple tables. The two most relevant tables for my analysis are the metadata table and the tiles table. The metadata table contains the diagnosis and stain of the slide, as well as technical details such as slide dimensions, tile size $s$, and pixel resolution. The tiles table on the other hand, contains the tiles of the slide at different magnification levels to speed up data loading and interactive viewing. Each tile is indexed with its $x, y$ coordinates as well as its zoom level, as depicted on the left side of fig. 1. The highest level of magnification contains the original tiles into which the slide was divided at the beginning of Pamly's preprocessing. Lower level tiles are obtained by concatenating four adjacent tiles from the next higher level and downsampling the resulting tile to match the dimensions of higher-level tiles. Consequently, all tiles remain squares of size $s$, which in our case is 512 pixels. The side length of a tile at the highest resolution is 128 µm for the Kiel data and approximately 129.23 µm for the Munich data. The lowest level, level zero, retains a scaled-down image of the entire slide (see again left side of fig. 1).

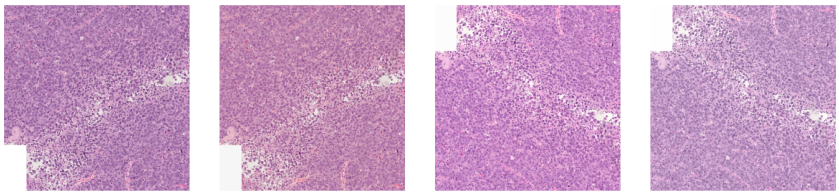
This standardized WSI format is now used to sample data for the neural networks, which will be discussed below. For a comprehensive description of all of Pamly's features, see Huttner, 2023.

## 2.2 Patch Extraction using Patchcraft

Processing an entire WSI, which can occupy several gigabytes of memory, is computationally infeasible for deep learning models. Instead, one extracts smaller 'patches', which are square subimages of the slide, as visualized on the right side of fig. 1. To facilitate this process, I developed the PC package, which I introduce below.

The core functionality of PC is to sample patches for training and testing neural networks. Training data is generated with augmentations, repeated sam-

pling, and overlapping patch selection, whereas test data is extracted without modifications to ensure unbiased evaluation. PC performs data augmentation directly during the sampling process, i.e. before saving the patches, to avoid computational bottlenecks during training. Each of these sampling strategies



Original Patch Examples for Augmentated Patches

Figure 2: An unaugmented patch and three examples of augmented versions of this patch (from left to right). Random horizontal and vertical flips and random color jitter were used as augmentations. The color jitter is a PyTorch transform and was set with brightness, contrast and saturation equal to 0.2 and a hue value of 0.05.

plays a crucial role in training data generation and I continue by explaining them in more detail.

In my experiments, I use horizontal and vertical flips as well as random color jitter, which alters brightness, contrast, saturation, and hue. The flipping transformations encourage the model to classify patches independently of their orientation. Additionally, color jitter enhances generalization across different scanners, as a slide's optical properties (e.g., contrast) depend on the digitization process and can therefore vary randomly (Duenweg et al., 2023). The effects of these augmentations are shown in fig. 2, where brightness, contrast, and saturation are set to 0.2 and hue to 0.05. However, I use smaller values in practice, as little prior work has been conducted on the datasets I use. PC also provides additional augmentation options, such as random rotation and rescaling, though, I do not use these in my setup. To further artificially expand the dataset, PC enables repeated sampling, where multiple patches are extracted from the same positions within the slide, which I will refer to as slide coordinates. Additionally, PC can apply coordinate perturbation, shifting the sampled patches slightly within a small region instead of extracting them from the exact same point. This is done during repeated sampling, to increase the variablity. Another augmentation strategy, which is called overlapping sampling, ensures that neighboring patches partially cover the same regions. In contrast, test data is sampled without applying any augmentation, repeated sampling, or coordinate perturbations. To avoid sampling patches that contain more white pixels than actual tissue, PC allows you to set a threshold for the percentage of white pixels allowed per patch, which I set to 25% for the training and test data.

PC provides two primary sampling methods. The `sample_tiles` function extracts patches strictly from tiles stored in the slide's SQLite file, ensuring a structured sampling process and it is the one I use for sampling from our data. Alternatively, the `sample_patches` function offers greater flexibility, allowing advanced augmentations such as perturbed repeated sampling, rescaling, and random rotation. This method also allows sampling from lower resolution tiles, which can significantly reduce execution time. When not sampling from the highest resolution, PC selects the highest magnification that allows a patch to be assembled from at most four tiles. The case where sampling occurs at the highest resolution is shown on the right side of fig. 1. The extracted patches are finally saved as individual files as PyTorch tensors in separate

directories corresponding to the slides from which they were sampled. This is very convenient for writing custom data loaders based on annotation files containing the paths to the different patches. Each filename encodes the slide coordinates and zoom level, ensuring traceability.

Beyond patch extraction, PC provides additional utilities to simplify dataset management. It can generate a dataset summary file listing filenames, diagnoses, and stains, allowing for a quick overview of dataset composition. From this information file, an overview of how many slides exist for a particular stain and diagnosis can be generated, which is especially helpful when the underlying dataset contains slides with different stains. Additionally, it supports patch visualization by displaying up to 100 sampled patches in a ten by ten grid, annotated with stain and diagnosis labels. PC is implemented as a Python package that can be installed via GitHub and Pip, with support for command-line execution in Docker containers. For details on the parameters that I used, see appendix A. Further implementation details can be found in the corresponding GitHub repository.[2]

## 2.3 Overview of the Datasets

The main dataset on which I train the majority of my networks originates from a pathology lab in Kiel, Germany. It comprises all six diagnostic classes and contains a total of 253 whole slide images, specifically 30 CLL, 53 DLBCL, 26 FL, 83 HL, 35 LTDS and 26 MCL slides. Each slide is represented by at least one 1024 µm patch with less than 25% white pixels. However, the

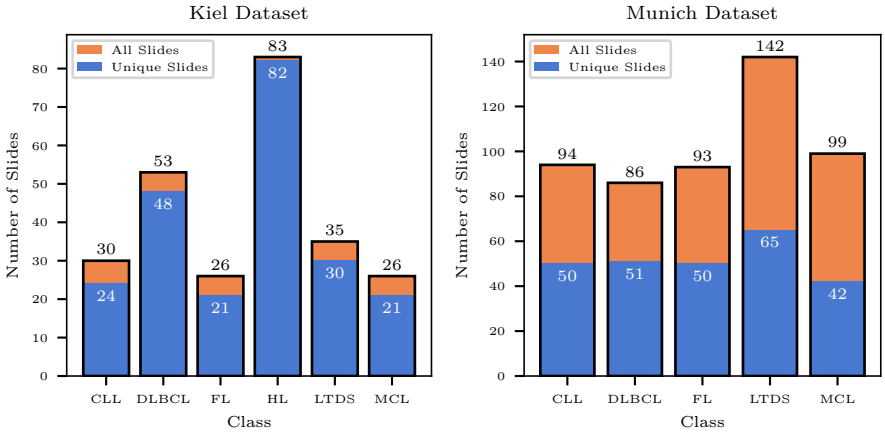



Figure 3: The number of total and unique slides per class from which it was possible to sample 4096 pixel patches for the Kiel dataset on which I will train the majority of my models (left) and a dataset from Munich that I will primarily use to investigate the Out-Of-Distribution (OOD) performance of my model (right).

number of slides per class is unevenly distributed, and multiple slides from the same patient are present in the dataset. To prevent data leakage, I ensure that all slides from a single patient remain in the same subset when splitting the data for training, validation, and testing. Taking this into account, the number of unique slides per class is 24 CLL, 48 DLBCL, 21 FL, 82 HL, 30 LTDS and 21 MCL, leading to a total of 226 unique slides. The distribution of slides in this dataset is summarized in the left panel of fig. 3.

[2] https://github.com/FerdinandToelkes/patchcraft

The second dataset, provided by pathologists in Munich, Germany, differs in several aspects. It does not include the HL class, and due to a slightly different scanning resolution, the extracted patches have a side length of approximately 1034 µm. This dataset consists of 94 CLL, 86 DLBCL, 93 FL, 142 LTDS and

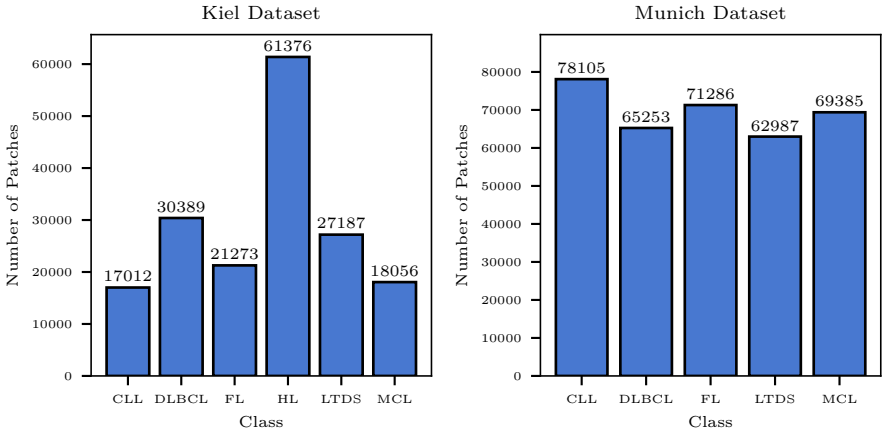



Figure 4: The total number of patches per class from which it was possible to sample 4096 pixel patches for the Kiel dataset (left) and the Munich dataset (right).

99 MCL slides, totaling 514 slides. Compared to the Kiel dataset, a much larger fraction of these are not unique in the previously defined sense. The number of unique slides per class is 50 CLL, 51 DLBCL, 50 FL, 65 LTDS, and 42 MCL, amounting to 258 unique slides. The right panel of fig. 3 shows this distribution graphically.

The corresponding distributions of sampled patches for both datasets are illustrated in fig. 4. While the patch distribution remains largely unchanged for the Kiel dataset, it becomes even more balanced for the Munich dataset. In order to estimate the performance not only in the OOD setting, I set aside five slides per class for the Kiel dataset, resulting in a balanced test set of 30 slides. The remaining slides will be used as training data. Since I use the Munich dataset exclusively for testing, no further splitting is necessary.

# 3 Methods

In general, a classification problem can be formulated as the task of finding a function

$$\begin{aligned} f : \mathbb{X} &\longrightarrow \mathbb{Y} \\ x &\longmapsto y, \end{aligned}$$

where the input and output data, $\mathbb{X}$ and $\mathbb{Y}$, are given and the function $f$ itself is unknown. The input data $\mathbb{X}$ consists of potentially high-dimensional data points, such as patches extracted from a WSI. Unlike in regression, the output space $\mathbb{Y}$ in classification tasks is finite and consists of discrete categories, such as lymphoma subtypes present in a given tissue sample. One expects that finding this function for our task of classifying WSIs will be very challenging due to the high complexity associated with the diagnostic process of pathologists (see section 1). A major motivation for applying neural networks to this problem is their ability to approximate complex functions. Even a relatively simple neural network, such as a Multilayer Perceptron (MLP) with a single hidden layer and an appropriate activation function, can approximate any continuous

function to an arbitrary degree of accuracy, provided that a sufficient number of hidden units is available (Cybenko, 1989). However, designing a practical parametrization of such a function, especially for image-based data, requires additional architectural refinements. Building on the structure of MLPs, Convolutional Neural Networks (CNNs) were introduced, incorporating the convolution operation as a key feature (Krizhevsky et al., 2012; LeCun et al., 1998). As the need for more expressive neural networks grew, deeper CNN architectures became popular due to their superior performance. However, training deep networks introduced numerical challenges, such as vanishing and exploding gradients during gradient descent optimization (Glorot and Bengio, 2010). These issues were mitigated through the introduction of residual blocks, leading to the development of Residual Network (ResNet) architectures (K. He et al., 2016). A further breakthrough came with the introduction of the attention mechanism, which led to the Transformer architecture in Natural Language Processing (NLP) and the ViT in Computer Vision (CV). These models enabled more effective contextual learning while improving training efficiency, making them the preferred choice for training on very large datasets (Dosovitskiy et al., 2020; Vaswani, 2017).

This section provides an overview of these deep learning methodologies, beginning with MLPs, convolutional layers, and residual blocks. It then introduces ViTs and their application in Computational Pathology (CPath). Finally, it discusses the loss function, which serves as a fundamental component across different network architectures, as well as a potential regularization technique to improve model generalization.

## 3.1 Deep Learning Fundamentals

After the first drastic decrease in funding for artificial intelligence, also known as the first 'AI winter', the MLP combined with the backpropagation algorithm reignited interest in neural networks in the mid-1980s (Muthukrishnan et al., 2020; Rumelhart et al., 1986). As one of the most fundamental network structures, the MLP serves as a foundation or key component of more complex architectures. It consists of multiple fully connected layers, where input data is sequentially processed through weighted sums and nonlinear activation functions, a process known as the forward pass. The model parameters, or weights, are then optimized using the backpropagation algorithm (Popescu et al., 2009; Rumelhart et al., 1986). In my classifier, I use an MLP in the final layers to map high-dimensional representations to an output vector with $n + 1$ entries, each representing the probability that the input belongs to one of the $n$ predefined classes or an additional 'unknown' class.

A classic example of classification is recognizing handwritten digits, where the network must identify specific patterns such as straight, horizontal, or circular lines. To learn these patterns more effectively, convolutional layers were introduced, forming the basis of CNNs when combined with MLPs. This network architecture demonstrated its power when it won the ILSVRC in 2012, marking a turning point for deep learning applications in CV (Russakovsky et al., 2015). The key operation in these layers is called convolution, where a small matrix, known as a kernel or filter, slides across the input image in a stepwise fashion. At each step, the Hadamard product (i.e., the elementwise product) is applied, followed by summation, producing a single value in the output feature map. This process is visualized below with a simple example of a kernel

designed to detect vertical lines:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 1 & 1 \\ 3 & 1 & 1 \\ 3 & 0 & 0 \end{bmatrix}$$

Input Kernel Output

The highlighted red and blue areas in the input matrix are transformed by this convolution process into the correspondingly highlighted values in the output matrix. Unlike fully connected layers, where weights correspond to individual summation terms, convolutional layers learn patterns by optimizing the entries of these convolutional filters. For a more detailed discussion of CNNs, including their fundamental principles and applications, the works of LeCun et al., 1998 and Krizhevsky et al., 2012 provide extensive insights beyond the brief summary given here.

To further improve the performance of deep learning models, both larger training datasets and deeper architectures are required. However, increasing the number of layers can lead to optimization challenges, particularly due to exploding or vanishing gradients (Glorot and Bengio, 2010). A notable symptom of these issues is the degradation problem, where a shallower CNN can empirically outperform a deeper one on both training and test data (K. He and Sun, 2015). This effect is illustrated on the left side of fig. 5, which compares the performance of a deep and a shallow model during training on the CIFAR-10 dataset.[3] This occurs even though the deeper model shares the same architecture as the shallower one, with additional layers appended. Intuitively, the deeper model should at least match the shallower model's performance, as the extra layers could learn an identity mapping. The introduction of 'residual blocks' helped to mitigate these difficulties. To facilitate identity mappings, K. He et al., 2016 proposed the residual block structure, illustrated on the right in fig. 5. The residual block encourages the network to learn the residual mapping

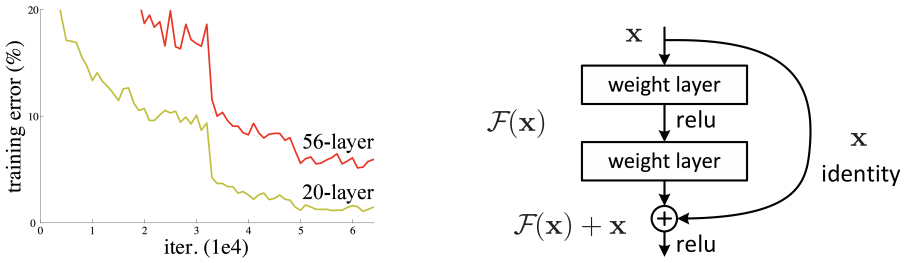



Figure 5: Training statistics illustrating the degradation problem in deep networks. The plot shows that a 56-layer model performs worse on the training data than a 20-layer model, despite having more parameters (left). The diagram on the right depicts a residual block, where $\mathcal{F}(x)$ represents the residual mapping. A weight layer can be a fully connected or convolutional layer, while relu (short for rectified linear unit) is an example activation function. Both figures are adapted from K. He et al., 2016, p.1-2.

$\mathcal{F}(x) \coloneqq \mathcal{H}(x) - x$, where $\mathcal{H}(x)$ represents the desired underlying mapping, e.g. the output of multiple stacked layers, and $x$ is the input to those layers. This reformulation makes identity mappings easier to learn, alleviating the degradation problem in deeper networks. Comprehensive implementations of CNNs

[3]The CIFAR-10 dataset consists of 32 by 32 pixel color images belonging to ten different classes and is widely used as a benchmark dataset (Krizhevsky, Hinton, et al., 2009).

with residual blocks, called ResNets, which won the 2015 ILSVRC, along with a deeper exploration of their architecture, are discussed in detail in K. He et al., 2016.

## 3.2 The Attention Mechanism and its Application to Vision Transformers

The Transformer architecture, introduced in the seminal paper Vaswani, 2017, first gained popularity in NLP. Its most widely known application is the large language model ChatGPT, where the 'T' stands for 'Transformer' (Achiam et al., 2023). The key component of the Transformer that we focus on here is its self-attention mechanism. To develop an intuition for how this mechanism works, we first examine its application in NLP before extending the concept to images. This leads to the ViT, which is the CV analog of the Transformer. The self-attention mechanism allows for rich context representation over long distances while allowing for more efficient parallelization, resulting in shorter training times compared to CNNs which has been observed in NLP (Vaswani, 2017) as well as in CV (Dosovitskiy et al., 2020).

Let us use the following simple example sentence to illustrate the self-attention process[4]

*'Dark hot coffee is my absolute favorite'.*

To process the input, we first convert each word to a numerical representation by mapping each word to a vector of fixed dimension $d_{\text{model}}$, resulting in the word embedding vectors:

$$\boldsymbol{E}_1 \quad \boldsymbol{E}_2 \quad \boldsymbol{E}_3 \quad \boldsymbol{E}_4 \quad \boldsymbol{E}_5 \quad \boldsymbol{E}_6 \quad \boldsymbol{E}_7,$$

where $\boldsymbol{E}_i \in \mathbb{R}^{d_{\text{model}}}$ for $i = 1, \dots, 7$ and $d_{\text{model}} \in \mathbb{N}$. To help the model understand word order, a positional encoding is also incorporated at this stage. The goal of the self-attention process is to update each of these embedding vectors based on contextual information. For instance, the embedding $\boldsymbol{E}_3$ corresponding to the word coffee should also incorporate the context that the coffee is described as dark and hot. For this, the model generates a query vector $\boldsymbol{Q}_i$ from each word embedding using a learned parameter matrix $W_Q$ (see sections 3.3 and 3.4 for details on the underlying loss function). In our example, nouns may be associated with the query

$$\textit{'Are there any adjectives in front of me?'} \quad \widehat{=} \quad W_Q.$$

It produces query vectors defined as

$$\boldsymbol{Q}_i = W_Q \boldsymbol{E}_i \in \mathbb{R}^{d_k}, \quad \text{where } W_Q \in \mathbb{R}^{d_k \times d_{\text{model}}}, d_k \in \mathbb{N}.$$

In this abstract $d_k$-dimensional vector space, query vectors associated with nouns will cluster in similar directions. The values for $d_{\text{model}}$ and $d_k$ are hyperparameters that must be set manually. Since the vector space $\mathbb{R}^{d_{\text{model}}}$ is used to encode words, while $\mathbb{R}^{d_k}$ encodes concepts, $d_{\text{model}}$ is typically chosen to be larger than $d_k$. For instance, in Vaswani, 2017, they were chosen as $d_{\text{model}} = 512$ and $d_k = 64$. The query is paired with a corresponding key that conceptually serves as the answer, which in our example could be interpreted as

$$\textit{'Yes (No), I am (not) an adjective!'} \quad \widehat{=} \quad W_K.$$

[4]The original Transformer paper Vaswani, 2017 and other related works mostly provide a rather high-level explanation of the general structure of attention. Therefore, I have adapted this very simple example from Grant Sanderson's YouTube video (see Sanderson, 2024) on the attention mechanism, which uses a slightly different notation than the original paper.

The model also learns a corresponding key matrix $W_K$ (see sections 3.3 and 3.4), which generates key vectors. These key vectors

$$\boldsymbol{K}_i = W_K \boldsymbol{E}_i \in \mathbb{R}^{d_k}, \quad \text{where } W_K \in \mathbb{R}^{d_k \times d_{\text{model}}},$$

determine how relevant each word is in response to a given query. The key vectors are grouped in the abstract $d_k$-dimensional space based on whether they represent adjectives. If they do, they align closely with the direction of query vectors from nouns. For convenience, the vectors are collected as follows:

$$\begin{aligned} Q^T &\coloneqq [\boldsymbol{Q}_1 \dots \boldsymbol{Q}_7] \\ K^T &\coloneqq [\boldsymbol{K}_1 \dots \boldsymbol{K}_7]\,. \end{aligned}$$

The degree of alignment between a query and a key is measured by taking the dot product of each possible pair, $\boldsymbol{Q}_i \cdot \boldsymbol{K}_j$. To ensure that the similarity scores in each row form a probability distribution summing to one, we apply the softmax function (see eq. (9)). Using matrix notation, we express this operation compactly as

$$\begin{aligned} W &= \text{softmax}\left(\begin{pmatrix} \boldsymbol{Q}_1\boldsymbol{K}_1 & \cdots & \boldsymbol{Q}_1\boldsymbol{K}_n \\ \vdots & \ddots & \vdots \\ \boldsymbol{Q}_n\boldsymbol{K}_1 & \cdots & \boldsymbol{Q}_n\boldsymbol{K}_n \end{pmatrix}\right) \equiv \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nn} \end{pmatrix} \\ &= \text{softmax}\left(QK^T\right) \in \mathbb{R}^{n \times n}, \end{aligned} \tag{1}$$

where $n$ is the sequence length. For our example, the weight matrix $W$ might look like this:

| Q\K | Dark | hot | coffee | is | my | absolute | favorite |
|---|---|---|---|---|---|---|---|
| Dark | 1/7 | 1/7 | 0.39 | 1/7 | 1/7 | 1/7 | 0.04 |
| hot | 1/7 | 1/7 | 0.53 | 1/7 | 1/7 | 1/7 | 0.06 |
| coffee | 1/7 | 1/7 | 0 | 1/7 | 1/7 | 1/7 | 0 |
| is | 1/7 | 1/7 | 0 | 1/7 | 1/7 | 1/7 | 0 |
| my | 1/7 | 1/7 | 0 | 1/7 | 1/7 | 1/7 | 0 |
| absolute | 1/7 | 1/7 | 0.01 | 1/7 | 1/7 | 1/7 | 0.8 |
| favorite | 1/7 | 1/7 | 0.07 | 1/7 | 1/7 | 1/7 | 0.1 |

Here, I present $W^T$ instead of $W$ for readability, with row and column labels indicating which query-key interactions are captured in the weight matrix. The weight matrix $W$ is often referred to as an attention map or attention weights. This attention map highlights which words are most relevant for each query. For instance, the word coffee strongly attends to dark and hot, reinforcing their descriptive relationship. Because of their nice interpretability, these attention maps are usually plotted to gain insight into how the corresponding model works.

To incorporate the contextual information into word representations, one introduces the value matrix $W_V$. This matrix determines how each embedding should be adjusted based on the attention scores from the query-key interactions. For example, after applying attention, the embedding $\boldsymbol{E}_3$ for coffee is updated to incorporate information that it is dark and hot:

$$\begin{aligned} \boldsymbol{E}_3' &= \boldsymbol{E}_3 + \sum_{i=1}^{n} w_{3i} \left(W_V \boldsymbol{E}_i\right) \\ &\equiv \boldsymbol{E}_3 + \sum_{i=1}^{n} w_{3i} \boldsymbol{V}_i, \end{aligned}$$

where $W_V \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ and $V_i \in \mathbb{R}^{d_{\text{model}}}$. This means that the network must learn $W_V$ (see sections 3.3 and 3.4) such that it maps an embedding $\boldsymbol{E}_i$

to the change that would be added to another embedding $\boldsymbol{E}_j$ if $\boldsymbol{E}_j$ attends to $\boldsymbol{E}_i$. A simple example of a value matrix is the identity matrix, which results in important embeddings being directly added, unchanged, to the embedding being updated. The weighted update to the embedding, given by the second term in eq. (2), is what is defined in Vaswani, 2017 as attention. This can be expressed compactly as:

$$\text{Attention(Q, K, V)} = \text{softmax}\left(QK^T\right)V, \tag{2}$$

where $V^T \coloneqq [\boldsymbol{V}_1 \dots \boldsymbol{V}_7]$. The equation above describes a single attention head. However, Transformers typically employ multiple attention heads, each learning to capture different aspects of the input representations.

Our simplified example is designed to provide intuition about the workings of a single attention head, omitting certain complexities for clarity. Given the flexibility of the model to learn and adjust its parameters in various ways, the exact behavior of the network may differ from this description. However, studies analyzing attention heads in Transformers, such as in the appendix of Vaswani, 2017, indicate that this intuition aligns with many observed patterns. For a deeper exploration of how these attention blocks form a Transformer, along with more detailed explanations, see Brauwers and Frasincar, 2021; Niu et al., 2021; Vaswani, 2017.

The conceptual link between Transformer networks and ViTs is well captured by the title of the paper from Dosovitskiy et al., 2020, where ViTs were first prominently introduced: 'An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.' To adapt the attention mechanism to images, the input is first partitioned into smaller image patches. This results in a sequence of images that can now be treated very similarly to a sequence of words as described above. This idea is illustrated in fig. 6, where the process is demonstrated with $3 \times 3$ patches. In NLP, a Transformer

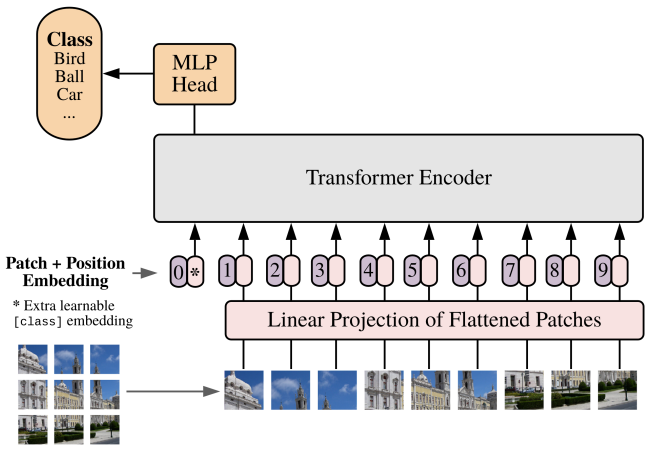



Figure 6: Depiction of a ViT adapted from Dosovitskiy et al., 2020, p.3. The image is split into equal-sized patches, each of which is transformed into a patch embedding vector. A learnable [`class`] embedding is added to enable the aggregation of information from all patches. Additionally, position encodings are added to these embeddings. The resulting sequence is then processed by a Transformer model, followed by an MLP that classifies the image based solely on the [`class`] embedding.

naturally incorporates the context of an entire text into its final word. However, a sequence of images does not have a well-defined last image. To address this, the ViT architecture introduces a learnable [`class`] embedding,

also referred to as the [`class`] token or [`class`] label. This embedding is simply a high-dimensional vector of fixed size. It serves as a central point where the network aggregates information extracted from the various patches, enabling downstream tasks such as classification to be based on it. Consequently, we will later use this embedding as a lower-dimensional representation of our image patches. For further details, please refer to Dosovitskiy et al., 2020.

There is no overall best model architecture when comparing attention and convolutional approaches, but there are key differences that help in choosing the right architecture for a given problem. The Transformer architecture is better at capturing global correlations with its attention mechanism, but struggles to capture local dependencies as effectively as CNNs. Unlike CNNs, which inherently possess translational equivariance due to convolution (Han et al., 2022), ViTs must be explicitly trained to learn this property. Consequently, ViTs typically require more data to match the performance of CNNs, but can outperform CNNs on large datasets (Dosovitskiy et al., 2020). The next section explores how a large CPath dataset is used to train a Transformer-based feature extractor.

### 3.3 The UNI Model: A Foundational Model for Computational Pathology

The ViT has proven particularly effective when applied to very large datasets (Caron et al., 2021; Dosovitskiy et al., 2020; Oquab et al., 2023; Zhou et al., 2021). This trend in CV parallels developments in NLP, where models are trained on vast amounts of data using self-supervised learning to obtain improved representations of input data. These learned representations can be used to train models for downstream tasks, such as classification (Devlin, 2018). Such large-scale models are often referred to as foundation models, and their training process is known as self-supervised pre-training. For example, the large language model ChatGPT employs a similar approach, so the 'P' in ChatGPT stands for 'Pre-training' (Achiam et al., 2023). One advantage of self-supervised training is that the resulting representations are richer and less restricted compared to those derived from supervised learning with labeled data (Caron et al., 2021). In addition, self-supervised learning helps circumvent the label scarcity that can limit the development of deep learning algorithms because it does not require labeled data. A well-known example of a foundation model in CV is DINOv2 (Oquab et al., 2023). The UNI model (Chen et al., 2023) serves as a foundation model for CPath, building on the architecture of DINOv2. In the following section, I will provide an overview of the key ideas underlying the UNI model.

To overcome potential problems associated with the difficulty of retrieving large annotated datasets due to the high complexity of the diagnostic process in CPath (Wang et al., 2019), self-supervised pre-trained models offer a promising approach (Chen et al., 2023). One of the key challenges in self-supervised learning is to determine an appropriate loss function without relying on labeled data. The first key part of this loss function for our case is called knowledge **di**stillation with **no** labels, which is the origin of the DINO model's name. In this approach, knowledge is distilled from an image $\mathcal{J}$ by introducing two networks: a student network $f_s$ and a teacher network $f_t$. These networks process different parts $u, v$ of the image, referred to as global views, and the training objective is to minimize the cross-entropy loss

$$\mathcal{L}_{[\mathrm{CLS}]} = \hat{v}_s^{[\mathrm{CLS}]} \log\left(u_t^{[\mathrm{CLS}]}\right) + \hat{u}_s^{[\mathrm{CLS}]} \log\left(v_t^{[\mathrm{CLS}]}\right) \tag{3}$$

between the different [`class`] labels predicted by the student (marked with a hat and index $s$) and teacher (marked with index $t$) networks. This can be seen

in fig. 7, which illustrates the network structure underlying the UNI model. In contrast to the teacher model, the student network receives additional local views as input, which are created by cropping global views. Including these local views in the loss minimization encourages the model to learn 'local-to-global' correspondences. Both networks are ViTs with identical architectures, where the teacher's parameters are updated using an exponentially moving average of the student's parameters. To avoid trivial solutions, such as always predicting a zero vector, output centering and softmax normalization in the teacher network is applied, while the student network only applies softmax normalization. This simple approach proved to be sufficient to avoid trivial solutions. For further information, see Caron et al., 2021.

The second key component of the loss is Masked Image Modeling (MIM), which is illustrated by the following example. Consider an image of an

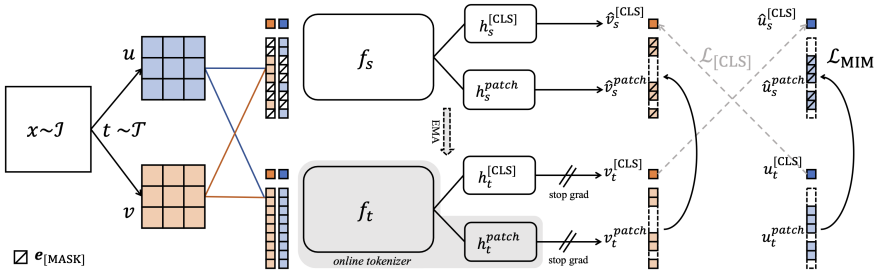



Figure 7: Visualization of the network structure on which the UNI model is based, adapted from Zhou et al., 2021, p.4. Two different views $u$ and $v$ are sampled from an input image $\mathcal{J}$ and passed to student and teacher networks $f_s$ and $f_t$, with the views passed to the student network being masked. The teacher network is derived from previous versions of the student network by transferring network weights with an exponentially moving average (abbreviated with EMA in the figure). The student network aims to minimize the cross-entropy between [`class`] labels of different views, denoted by $\mathcal{L}_{[\text{CLS}]}$, and the cross-entropy between the generated patch tokens of masked patches of the same view, denoted by $\mathcal{L}_{[\text{MIM}]}$.

elephant. Even if the bottom half of the image is masked, it is still easy to identify the animal as an elephant based on context clues, such as its large gray body, four legs, big ears, and long trunk, along with a natural environment in the background. Humans infer missing information from context, and MIM aims to teach models a similar contextual understanding to enhance input representations. Thus following a MIM objective, the student network processes masked views of the input image, and a second loss $\mathcal{L}_{[\text{MIM}]}$ is introduced. This loss is defined as the cross-entropy between the patch representations $\hat{u}_s^{patch}$ and $u_t^{patch}$ generated by the student and teacher networks for the same view $u$. This loss is applied only to the representations of masked patches. The resulting model architecture initially explored in Zhou et al., 2021, is depicted in fig. 7.

The DINOv2 paper (Oquab et al., 2023) primarily refines these ideas and presents a new data processing pipeline that enhances the quality of large datasets. Using the DINOv2 framework, the UNI model was trained on 'over 100 million image patches from over 100,000 diagnostic haematoxylin and eosin-stained WSIs' (Chen et al., 2023, p.1). This section provided a high-level summary of the UNI model's key concepts. For a more detailed discussion, I refer to the original works (Caron et al., 2021; Chen et al., 2023; Oquab et al., 2023; Zhou et al., 2021). The DINOv2 model has been applied directly to medical image data from various domains, including MRI, CT, MR, Ultrasound, and X-Ray, for tasks such as segmentation and classification (Anand et al., 2023;

Ayzenberg et al., 2024; Baharoon et al., 2023; Huang et al., 2024; Kundu et al., 2024). There have also been efforts to train other foundational models for CPath using the DINOv2 framework, such as Prov-Gigapath, CONCH, or Virchow, among others, with the UNI network being the most prominent to date (Aben et al., 2024; Alfasly et al., 2024; Dippel et al., 2024; Lu et al., 2024; Nechaev et al., 2024; Vorontsov et al., 2023; Xu et al., 2024).

### 3.4 Cross-Entropy Loss for Classification

The loss function is a crucial component of a neural network, as it is used to update the model parameters. It provides many ways to address the unique challenges posed by different problem settings. When explaining the structure of the UNI network, I briefly mentioned the cross-entropy loss between different [`class`] labels (see eq. (3)). In this section, I formally introduce cross-entropy as a standard loss function for supervised classification tasks. I also discuss potential modifications to cross-entropy that could be useful for our specific problem of classifying WSIs from the datasets introduced in section 2.3.

Cross-entropy is derived using maximum likelihood estimation. For a general supervised learning task, let $\mathbb{X} = \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ be the training data and $\mathbb{Y} = \{\boldsymbol{y}^{(1)}, \dots, \boldsymbol{y}^{(m)}\}$ be the labels associated with the training data. Both contain $m$ samples and we assume that the samples are independent and identically distributed. The task is to predict, given a sample $\boldsymbol{x} \in \mathbb{X}$, the underlying label $\boldsymbol{y} \in \mathbb{Y}$. In our case, $\boldsymbol{x}$ is an image and $\boldsymbol{y}$ is the one-hot encoded diagnosis, i.e. $\boldsymbol{y} \in \{0,1\}^7$. According to maximum likelihood estimation, the parameters $\boldsymbol{\theta}_{\text{model}}$ that define our model are now the parameters that maximize the probability of reproducing $\mathbb{Y}$ given $\mathbb{X}$ and $\boldsymbol{\theta}$, i.e.,

$$\boldsymbol{\theta}_{\text{model}} = \arg\max_{\boldsymbol{\theta}} P(\mathbb{Y}|\mathbb{X};\boldsymbol{\theta}). \tag{4}$$

In the case of binary classification ($y \in \{0,1\}$), this can be written as

$$\boldsymbol{\theta}_{\text{model}} = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p\left(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right). \tag{5}$$

Generalizing this to more than two classes by using one-hot encoding then yields

$$\boldsymbol{\theta}_{\text{model}} = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} \prod_{c=1}^{C} p\left(\boldsymbol{y}_c^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right)^{\boldsymbol{y}_c^{(i)}}, \tag{6}$$

where $C$ represents the number of classes. For $C = 2$, the above equation reduces to eq. (5), since any term raised to the power of zero is equal to one. Since this potentially very long product of probabilities is prone to numerical instability caused by floating-point arithmetic, the logarithm of eq. (6) is taken

$$\boldsymbol{\theta}_{\text{model}} = \arg\min_{\boldsymbol{\theta}} \left[ -\sum_{i=1}^{m} \sum_{c=1}^{C} \boldsymbol{y}_c^{(i)} \log\left(p\left(\boldsymbol{y}_c^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right)\right)\right], \tag{7}$$

where we also multiplied the expression by minus one, in order to make it a minimization problem. Minimizing eq. (7) is equivalent to maximizing eq. (6), as the logarithm is monotonically increasing. The argument in eq. (7) is referred to as cross-entropy between the ground truth $\boldsymbol{y}^{(i)}$ and the models prediction $p(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta})$. For more details on maximum likelihood estimation and cross-entropy, see Bengio et al., 2017, on which this brief derivation is based, or refer to Bishop and Nasrabadi, 2006.

In our case, there are two potential challenges: a significant class imbalance within the Kiel dataset (see section 2.3) and the need to train on image patches

rather than entire WSIs, where each patch may not contain sufficient information for labeling the entire data point. In the following sections, I present two modifications to the loss function that aim to address these specific challenges.

### 3.4.1 Class Imbalance and Weighted Cross-Entropy

In an attempt to counteract the class imbalance shown in figs. 3 and 4, we introduce weights into the loss function. The cross-entropy loss in PyTorch is defined as follows:

$$\begin{aligned} l(x, y) &= -\sum_{n=1}^{N} l_n(x, y) \\ &= -\sum_{n=1}^{N} \sum_{c=1}^{C} w_c \log \left( \frac{\exp(x_{n,c})}{\sum_{i=1}^{C} \exp(x_{n,i})} \right) y_{n,c}, \end{aligned} \tag{8}$$

where $\boldsymbol{x}$ are the output vectors of the neural network for a batch, with dimensions of batch size $N$ times number of classes $C$ (see Contributors, 2023). The target vectors $\boldsymbol{y}$ have the same dimensions, but consist of one-hot coded vectors. The vector $\boldsymbol{w}$ corresponds to user-specified weights. Comparing PyTorch's definition of cross-entropy loss, shown in eq. (8) for weights equal to one, with eq. (7), we see that PyTorch uses a softmax function

$$\sigma(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{C} \exp(x_j)}, \tag{9}$$

which ensures that the inputs to the cross-entropy are normalized to one, making them interpretable as probabilities. We refer to these as the model's predictions, $\hat{\boldsymbol{y}}$, such that

$$\ell_n(x, y) = -\sum_{c=1}^{C} w_c \log \left( \hat{y}_{n,c} \right) y_{n,c}. \tag{10}$$

To reduce the effects of class imbalances, we assign less weight to more frequently represented classes in the loss. To achieve this, we set the weights as follows, following the approach proposed in Aurelio et al., 2019:

$$w_c = \frac{\text{length of the training set}}{\text{number of samples of class c in the training set}}. \tag{11}$$

With this weighting scheme, the contribution of samples from overrepresented classes to the cross-entropy loss is reduced, balancing their influence during optimization. This adjustment aims to mitigate the impact of class imbalance on the training process.

### 3.4.2 Regularization via Label Smoothing

We face a fundamental problem when training a neural network on our dataset of WSIs because a single WSI is too large to pass through the network (see section 2.2). To address this, we divide the WSIs into smaller patches for training, which introduces a new difficulty. Training on these image patches using only the labels corresponding to the diagnosis of the underlying WSI implicitly assumes that each patch contains enough information to predict the WSI. As far as we know from pathologists, this is not the case. Simply put, we expect models trained on patches to be overly confident in their predictions and prone to overfitting. A regularization method proposed to address this challenge, called Label Smoothing (LS), was introduced in Szegedy et al., 2016. When computing the weighted cross-entropy, the one-hot encoded ground truth labels are modified to

$$y'_{n,c} = (1 - \text{ls}) y_{n,c} + \frac{\text{ls}}{C}, \tag{12}$$

where ls $\in [0, 1]$ is the hyperparameter specifying the amount of smoothing applied and $C$ is the number of classes as before. Applying this to four classes with ls equal to 0.1, a label $y$ representing the second class would be transformed like this

$$y = 2 \xrightarrow{\text{OHE}} \boldsymbol{y} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \xrightarrow{\text{LS}} \boldsymbol{y}' = \begin{pmatrix} 0.025 \\ 0.925 \\ 0.025 \\ 0.025 \end{pmatrix}, \tag{13}$$

where OHE is short for one-hot encoding and LS stands for label smoothing. For a thorough examination of the effects of label smoothing, see, for example Müller et al., 2019.

# 4 Results

Addressing the classification problem involves two key steps: data processing and model construction. Data processing encompasses both preparing the data for use (see section 2) and possibly transforming it into a representation that optimizes computational efficiency while retaining relevant information. For instance, consider designing a model to predict whether a jacket is needed on a given day. Suppose each data point consists of the mean temperatures from the past 10,000 days, along with a label indicating whether a jacket was worn. A reasonable preprocessing step might be to retain only the temperatures from the last 365 days, reducing input dimensionality and improving computational efficiency, but potentially at the expense of model performance.

In our case, I use the UNI model to generate embeddings that serve as a compact representation of image patches. Since extracting these embeddings is a computationally expensive step, I investigate how resizing the input images affects the resulting embeddings. Specifically, I compare how embeddings and attention maps change when computed from patches at different resolutions, assessing the degree to which lower-resolution embeddings preserve the information contained in higher-resolution ones. This analysis informs my choice of preprocessing parameters, allowing me to balance computational efficiency with representation quality.

After establishing these insights, I present the complete data processing pipeline, followed by the details of my classifier training strategy. I describe how individual classifiers are trained within a cross-validation framework and how I construct an ensemble model from them. Finally, I evaluate this ensemble and analyze the contributions of different processing steps and architectural choices.

By structuring the results section in this way, I ensure a logical progression: first analyzing the effect of image resolution on the embeddings and attentions themselves, then integrating these findings into my preprocessing strategy, and finally constructing and evaluating the classification model.

## 4.1 UNI Analysis

The UNI network is designed to generate rich embeddings from tissue patches, which we expect to be highly informative. Training networks on these embeddings instead of directly on patches significantly reduces training time and allows training on larger input images. A minor drawback of using the UNI network is the added model complexity and our limited understanding of the resulting

embeddings. In order to make the model's predictions more comprehensible to us, I want to store the attention maps that ultimately led to the patch embeddings. This allows us to track which regions of a given patch were particularly important for the final classification, and one can design a network that only trains on regions that were highly attended by the network (see section 4.2). A somewhat related approach without using pre-trained ViTs can be found in Qaiser and Rajpoot, 2019, from which I took some inspiration, such as focusing on rather large input patches. I will sample 4096 by 4096 pixel images and use their embeddings and attention maps generated by the UNI model. In the following sections, I explore efficient memory storage for attention maps, examine how downsizing 4096 by 4096 input patches impacts UNI embeddings and their corresponding weight matrices and visualize the patch embeddings of our two datasets.

### 4.1.1 Extracting Attention Maps from the UNI Model

Working with large image sizes introduces challenges, particularly regarding memory efficiency and computational feasibility. The UNI model is designed to accommodate input images with relatively arbitrary dimensions, with the primary limitation being the available memory (see Chen et al., 2023 for more details).

To better understand the computational demands, let us examine the memory requirements. Computing attention maps involves the operation

$$W = \text{softmax}\left(QK^T\right) \in \mathbb{R}^{n\times n},$$

as introduced in eq. (1) where $Q$ and $K$ are the query and key vectors of an attention head and $n$ is the sequence length. We are particularly interested in the attention maps from the final layer of the UNI model, which uses 16 by 16 pixel patches (see Chen et al., 2023, p.56). For a 4096 by 4096 pixel image, this results in a sequence length of

$$n = \left(\frac{4096}{16}\right)^2 + 1 = 65537,$$

with the additional token corresponding to the [`class`] token introduced in ViTs (see fig. 6). In the final layer, the UNI model uses 16 attention heads, leading to a combined attention map or weight matrix

$$W \in \mathbb{R}^{65537\times 65537\times 16}.$$

Since the attention weights are stored as float32 values, which require four bytes per number, the total memory $M$ required to store the full attention maps can be calculated as

$$\begin{aligned} M &= 65537 \times 65537 \times 16 \times 4 \times 10^{-9}\,\text{GB} \\ &\approx 275\,\text{GB}, \end{aligned}$$

which represents a prohibitively large memory requirement. This raises a key question: do we need the full weight matrices, or can a more memory-efficient approach suffice?

Let us carefully consider our use case. We aim to utilize the attention map to identify contextually important regions of the input image that were important for generating the final [`class`] label, i.e., the patch embedding. This can be compared to how a pathologist surveys a large slide but focuses on a few specific regions for detailed analysis. In our case, these regions correspond to image patches with the highest attention values. Thus, we expect the attention maps

to be visually meaningful, i.e., they should highlight specific regions of the input image that were particularly important for generating the embedding. However, subdividing the input into fixed-size image patches causes the sequence length to lose some interpretability, particularly when compared to NLP applications. In NLP, each sequence element typically represents a whole word, whereas in our case, a sequence element represents a patch of the image. For simplicity, I assume that each word is encoded as a whole and not further tokenized, as is often the case in NLP.[5] Since the full weight matrices lack direct interpretability, we focus on the [`class`] token, which aggregates contextual information from the entire image into a single designated embedding. Therefore, the attention maps describing how the [`class`] label attends to each image patch are precisely what we are interested in. Consequently, instead of computing the full matrix multiplication of the query and key vectors for all patches, we only need to compute the attention of the [`class`] label's query vector with the key vectors corresponding to the patches. This reduces the weight matrix to

$$W \in \mathbb{R}^{1\times 65536\times 16},$$

which translates to a memory requirement of

$$\begin{aligned} M &= 1 \times 65536 \times 16 \times 4 \times 10^{-6}\,\text{MB} \\ &\approx 4\,\text{MB}. \end{aligned}$$

This reduction in memory requirements makes the calculation much more feasible. Beyond that, further optimizations can be achieved by reducing numerical precision, e.g., switching from float32 to float16, and downsampling the original images before passing them through the UNI model. The effects on the embeddings of not computing the full attention maps in the last layer are insignificant compared to the effect of augmentations, as one can see in table 1.

The original UNI model tackles memory limitations by employing fused attention, a modified version of attention designed to improve memory efficiency by combining certain operations and avoiding full matrix multiplications. For more details, consult the source code of PyTorch's `scaled_dot_product_attention` function, as well as parts of Dao et al., 2022. The UNI model, built on PyTorch's ViT architecture, employs fused attention, which optimizes memory but does not store attention maps by default. To circumvent this, I utilize a custom wrapper function (see appendix B), which modifies the final layer of the model to compute and store the attention weights for the class token as described previously. More recently, PyTorch has introduced 'FlexAttention', which is an official mechanism for this purpose. However, at the time of my analysis, this feature had not yet been included in the latest stable release (see He et al., 2024 for more information). Even though I successfully implemented a custom version of the UNI model to store attention maps efficiently, there may still be valid reasons not to use the highest image resolutions. These include reducing code execution time and further decreasing memory costs associated with saving attention weights. The effects of using lower-resolution images as inputs will be explored below.

#### 4.1.2 Comparison of UNI Embeddings Across Different Image Resolutions

This section and the one after next investigate how data augmentation and dimension reduction impact embedding quality and attention maps. I quantify

[5] For instance, subword tokenization may result in sequence elements representing parts of words, complicating interpretability.

these effects using two metrics, the Mean Squared Error (MSE)

$$\mathrm{MSE}(\boldsymbol{u}, \boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^{n} (u_i - w_i)^2 \tag{14}$$

and the cosine similarity

$$\begin{aligned} S_C(\boldsymbol{u}, \boldsymbol{w}) = \cos(\theta) &= \frac{\boldsymbol{u} \cdot \boldsymbol{w}}{|\boldsymbol{u}||\boldsymbol{w}|} \\ &= \frac{\sum_{i=1}^{n} u_i w_i}{\sqrt{\sum_{i=1}^{n} u_i^2} \cdot \sqrt{\sum_{i=1}^{n} w_i^2}}, \end{aligned} \tag{15}$$

where $\boldsymbol{u}, \boldsymbol{w} \in \mathbb{R}^n$ are two arbitrary embeddings, and $n = 1024$ is the embedding dimension in our case (see Chen et al., 2023, p.56). To build intuition for these metrics in our specific context, I begin by examining the impact of data augmentations, such as horizontal flips, vertical flips, and color jitter, on the embeddings. Throughout the embedding analysis, the embeddings from the

| Augmentation | Cosine Similarity | Mean Squared Error |
|---|---|---|
| No Augmentation | 0.9975 ± 0.0002 | 0.0076 ± 0.0005 |
| Color Jitter | 0.9725 ± 0.0022 | 0.0826 ± 0.0065 |
| Horizontal Flip | 0.9648 ± 0.0012 | 0.1058 ± 0.0036 |
| Horizontal and Vertical Flip | 0.9604 ± 0.0013 | 0.1190 ± 0.0039 |
| All Three Combined | 0.9366 ± 0.0025 | 0.1898 ± 0.0074 |

Table 1: The average cosine similarity and MSE with their standard errors for different combinations of augmentations at full resolution, i.e. 4096 by 4096 pixels. All values are relative to UNI's original embeddings, meaning the 'No Augmentation' row reflects the comparison between the untouched UNI model and my modified version.

original UNI model serve as the baseline for comparison and I setup PyTorch's color jitter transform with brightness, contrast and saturation equal to 0.1 and a hue value of 0.025. I also include the results of comparing the baseline embeddings to those generated by my slightly modified model. Our dataset consists of 90 randomly selected patches from six slides, each representing a different class (15 patches per slide). All these patches are taken from the Kiel dataset. Table 1 shows that data augmentations alone have minimal impact on embeddings. This observation becomes even clearer when I investigate simple dimension reductions in the following.

I discuss two methods for decreasing image size: Pytorch's `resize` function, which uses bilinear interpolation, and Pytorch's `random_crop` function, which crops the image at mostly arbitrary positions. Starting with the full resolution of 4096 pixels, I gradually reduce the width in steps of 512, eventually reaching 512 pixels. Since the width is decreased linearly, the area shrinks quadratically. The standard errors of the averaged metrics are shown as shaded regions. Additionally, the MSE and cosine similarity values for the combined data augmentations (see the last row of table 1) are plotted to help interpret the results. The results are displayed in fig. 8, from which one can observe that random cropping is by far the better choice for simple dimension reduction according to the chosen metrics. This agrees nicely with encouraged local-to-global correspondences briefly mentioned in section 3.3. Furthermore, the effects of data augmentation are most pronounced when the input dimensions remain above 3584 pixels (for resizing) or 2560 pixels (for cropping). However, since my aim is to visualize attention weights of full patches rather than parts of patches, I rely on resizing for downsampling. Before continuing my analysis by exploring

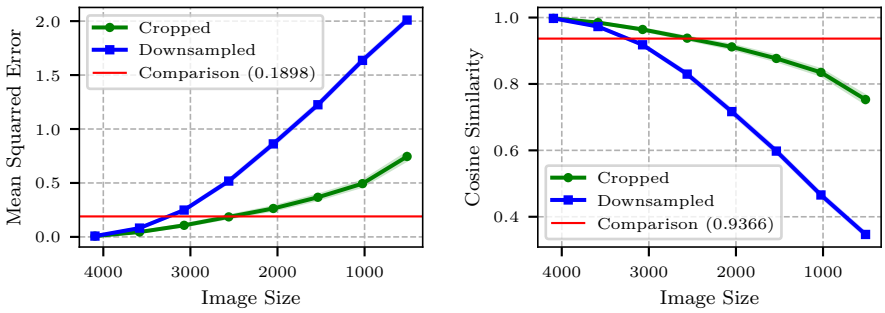



Figure 8: Visualization of the effects on the embeddings generated by the UNI model when reducing the dimensions of the input data. The MSE (left) and cosine similarity (right) are used as metrics. To reduce dimensionality, images are either randomly cropped (green lines) or resized using bilinear interpolation (blue lines). The shaded regions, primarily in the green curves, represent the standard errors of the metrics. The red lines show the corresponding values for all augmentations combined, as listed in table 1.

how the reduction of the input data affects the weight matrices, I use dimension reduction to visualize the UNI embeddings for both of our datasets.

### 4.1.3 Visualization of UNI Embeddings with the UMAP Algorithm

To visualize the 1024-dimensional UNI embeddings, I apply dimensionality reduction. I use the Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) algorithm, which achieves results comparable to the state-of-the-art t-SNE algorithm and is one of many available methods. A key advantage of UMAP is its scalability and efficiency, making

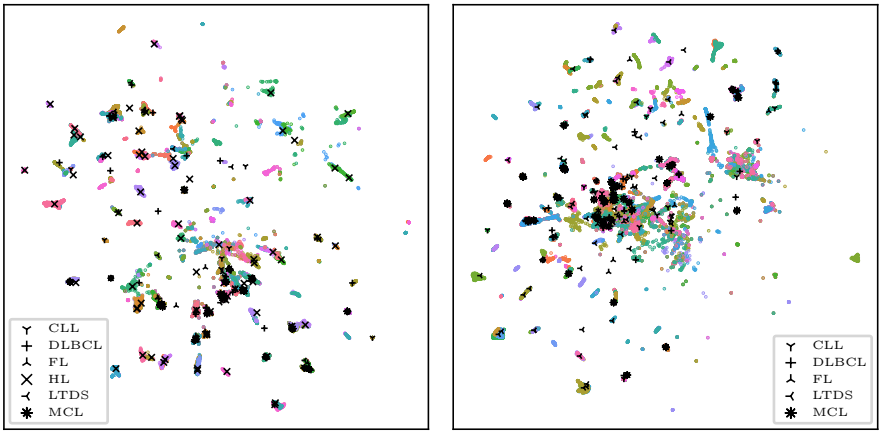



Figure 9: Embeddings resulting from patches sampled in test mode for the Kiel (left) and Munich (right) datasets, visualized in two dimensions using UMAP with 15 nearest neighbors and a minimum distance of 0.1. Embeddings belonging to the same slide are shown in the same color, and a marker corresponding to the respective class is shown at the median x-y coordinates for each slide.

it well-suited for large, high-dimensional datasets like ours. The goal of UMAP is to reduce dimensionality while preserving both local and global structures as much as possible. Its approach is rooted in Riemannian geometry and algebraic topology, providing a strong mathematical foundation. For a detailed derivation, see McInnes et al., 2018. Throughout this analysis, I

use only one slide per patient to prevent biases arising from variations in the ratio of total to unique slides across the two datasets. Furthermore, all points within a single plot are derived from the same fitted UMAP to ensure interpretability. Notably, the Kiel and Munich datasets contain a comparable number of unique slides, with 225 and 258, respectively (see section 2.3).

To gain insights into the dataset structures, I applied UMAP separately to all patch embeddings from the Kiel data and Munich datasets. These embeddings were generated following the processing steps in section 4.2.1. To minimize the impact of preprocessing choices, I only included patches sampled in test mode, meaning no augmentations were applied (see section 2.2). I used Python's `UMAP` function from the `UMAP` package with default parameters, where the most important ones, the number of nearest neighbors and the minimum distance, were set to 15 and 0.1, respectively. The resulting two-dimensional

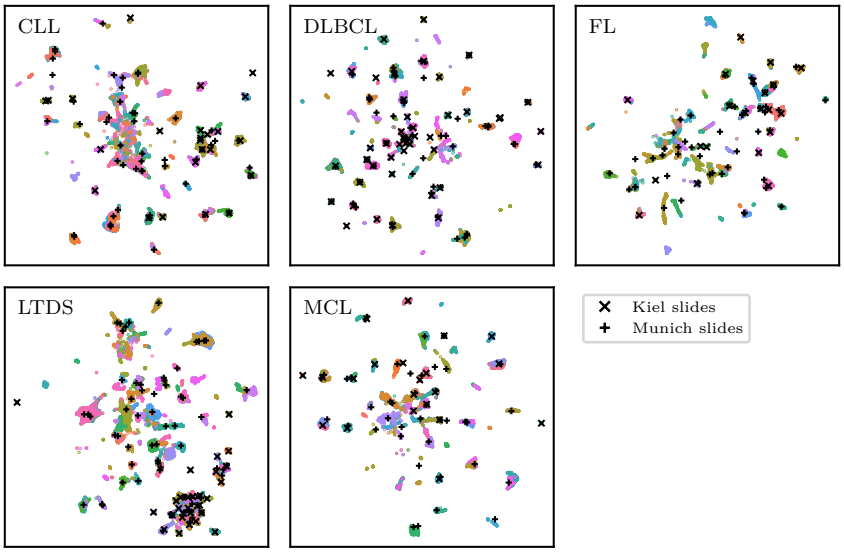



Figure 10: Embeddings resulting from patches sampled in test mode for the Kiel and Munich datasets for each class separately visualized in two dimensions using UMAP with 15 nearest neighbors and a minimum distance of 0.1. Embeddings belonging to the same slide are shown in the same color, and a marker corresponding to the respective class is shown at the median $x, y$ coordinates for each slide.

representations of these patch embeddings are shown in fig. 9, where each slide's patches share a color, and the slide's class is indicated by a marker at the median x-y patch coordinates. In the Kiel dataset, most HL clusters exhibit minimal overlap with other classes, suggesting that HL patches may be easier to classify. Furthermore, the Kiel data appears less clustered than the Munich data, which could indicate that classifying slides from Kiel is less complex than classifying the Munich data.

To further investigate class-specific differences between datasets, I applied UMAP to all patches from a single class across both datasets. These plots follow the same visualization scheme as before, but the median patch markers now represent dataset origin. As shown in fig. 10, DLBCL and MCL clusters frequently overlap across datasets, suggesting consistency between domains. In contrast, CLL and particularly LTDS show clear separation, implying structural differences. One possible explanation is that all Kiel LTDS samples were taken from full biopsies, whereas Munich LTDS cases were obtained from

needle biopsies. The FL class exhibits both overlapping and distinct clusters, warranting further investigation.

Due to its probabilistic nature, UMAP results may vary upon regeneration. Additionally, its hyperparameters influence the final projections. However, adjusting the number of nearest neighbors to 10 or 20 did not significantly alter the results, as shown in appendix C. This concludes my analysis of the UNI embeddings.

#### 4.1.4 Comparison of UNI Attention Maps Across Different Image Resolutions

To understand the impact of resizing operations, I measured execution times for generating embeddings and attention maps using 90 exemplary patches. This was done for full-resolution images with different augmentations applied,

| Augmentation | Time (s) |
|---|---|
| No Augmentation | 1766.72 |
| Color Jitter | 1878.68 |
| Horizontal Flip | 1758.81 |
| Horizontal and Vertical Flip | 1763.92 |
| All Three Combined | 1900.49 |

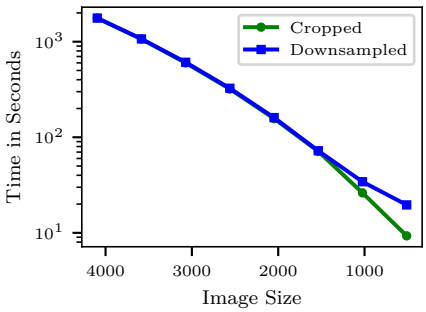



Figure 11: Code execution times for embedding generation and attention map extraction at 4096 resolution with different augmentations (left) and for cropped vs. downsampled images at varying resolutions (right).

as well as for resized and cropped images. Figure 11 reveals two key observations. First, there is a notable time difference between augmenting and not augmenting the images, as seen in the left side of the figure. Second, resizing images provides substantial performance gains, with execution times significantly reduced for both methods of dimension reduction (cropping and resizing), as shown in the right plot of fig. 11. As expected, the time savings are approximately equivalent between the two methods. With these performance gains in mind, I continue to investigate how reducing dimensionality impacts the resulting attention maps.

To this end, I again use the MSE to compare full attention maps. Since my goal is to infer positions in the original image from the weight matrices, I resize each map to match the original image size. This approach allows for a direct comparison of attention maps generated from inputs of different resolutions. To gain intuition about how the MSE behaves in this scenario, I first examine its values for various augmentations applied to high-resolution input images. Because the original UNI model does not store attention weights, our new baseline consists of attention maps extracted from full-resolution images using my wrapper function (see appendix B). To ensure a fair comparison, any flips must be reversed before comparing the weight matrices. The results are shown in fig. 12, and they clearly indicate that the chosen augmentations have a relatively minor effect on the MSE between attention maps compared to the impact of resizing the input to the UNI model. Starting at a resolution of 2560 by 2560 pixels, each further resizing step leads to significant increases in the MSE.

| Augmentation | MSE |
|---|---|
| Color Jitter | $(1.75 \pm 0.27) \times 10^{-4}$ |
| Horizontal Flip | $(3.70 \pm 0.50) \times 10^{-4}$ |
| Hor. & Vert. Flip | $(3.93 \pm 0.44) \times 10^{-4}$ |
| All Combined | $(4.64 \pm 0.44) \times 10^{-4}$ |

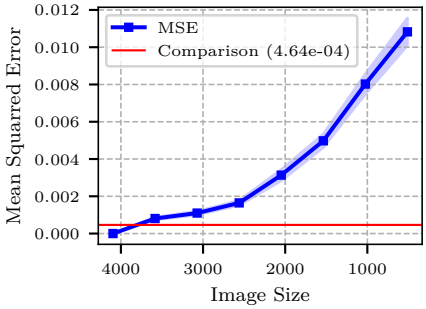



Figure 12: The MSE of attention maps resulting from 4096 by 4096 pixel augmented and non-augmented images (left) and the MSE of attention maps generated from images with different resolutions (right). For easier comparison, the MSE resulting from combining all augmentations is added as a vertical red line in the plot on the right. All results are averaged over 90 patches, with the corresponding standard errors displayed. The baseline for comparison are the attention maps from the full-resolution images, so the MSE is zero for the first x-value in the plot on the right.

Since I aim to utilize the attention maps for identifying key regions of the underlying image, I am particularly interested in determining the locations of the $n$ points with the highest attention weights. To achieve this, I first extract the coordinates of the $n$ highest values in the attention map, and then initialize a mask tensor, filled with zeros and matching the dimensions of the original image (excluding the color channels). To mark important regions, I place ones in the mask at positions corresponding the highest values from the attention map (step b). Note that the weight matrices correspond to 16 by 16 patches of the input image, as defined by the UNI model (see Chen et al., 2023, p.56). Therefore, when marking the highest-attended regions, the coordinates derived from the attention weights are resized to the original image size using bilinear interpolation (step a). To compare regions rather than individual points, I expand the points in the mask into squares around the already marked locations (step c). The final result is a tensor masked with ones at the $n$ highest-attended regions. A simplified example of this mask generation process for $n = 2$ top regions is illustrated below:

$$\begin{pmatrix} 0.3 & 0.8 \\ 0.7 & 0.2 \end{pmatrix} \xrightarrow{\text{a}} \begin{pmatrix} 0.3 & 0.425 & 0.675 & \boxed{0.8} \\ 0.4 & 0.4625 & 0.5875 & 0.65 \\ 0.6 & 0.5375 & 0.4125 & 0.35 \\ \boxed{0.7} & 0.575 & 0.325 & 0.2 \end{pmatrix} \xrightarrow{\text{b}} \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{\text{c}} \begin{pmatrix} 0 & 0 & \boxed{1} & \boxed{1} \\ 0 & 0 & \boxed{1} & \boxed{1} \\ \boxed{1} & \boxed{1} & 0 & 0 \\ \boxed{1} & \boxed{1} & 0 & 0 \end{pmatrix}$$

Attention Map | Resized Attention Map | Mask with 1's at Maxima | Completed Mask

To compare the top regions across attention maps, one can take the element-wise product, i.e., the Hadamard product, of two masks and sum over the resulting tensor. This provides a measure of the overlap between the top regions, normalized by the number of ones in the larger of the two masks. Since I will be comparing not only how this overlap behaves for different resolutions, but also for different numbers of top regions, one has to keep in mind that the more regions one compares, the more total area is covered by the image. If too many regions are compared, the overlap will inevitably reach 100%. For our case, I set the side length of the squares representing the top regions to 400 by 400 pixels, which corresponds to 100 µm sized patches. This results in

$$p = \frac{10 \cdot 400^2}{4096^2} \approx 9.537\%$$

being maximally covered of the total area of the underlying image for $n = 10$

top regions to be compared. As before, I compute the average overlaps between attention maps from images subjected to different augmentations for $n = 1, \ldots, 10$, before comparing this to the impact of resizing the input data. The results for different augmentations are shown in table 2. Here,

| Regions | Color Jitter | Hor. Flip | Hor. & Vert. Flips | All Combined |
|---|---|---|---|---|
| 1 | 95.56 ± 2.17 | 92.36 ± 2.63 | 92.00 ± 2.75 | 89.33 ± 3.21 |
| 2 | 96.94 ± 1.23 | 90.29 ± 2.11 | 87.02 ± 2.25 | 86.92 ± 2.43 |
| 3 | 93.11 ± 1.76 | 85.32 ± 2.31 | 86.45 ± 2.21 | 85.81 ± 2.37 |
| 4 | 90.39 ± 1.96 | 83.87 ± 2.25 | 83.97 ± 2.21 | 83.95 ± 2.30 |
| 5 | 92.97 ± 1.75 | 85.66 ± 2.20 | 81.54 ± 2.32 | 81.01 ± 2.31 |
| 6 | 89.48 ± 1.89 | 83.04 ± 2.26 | 80.19 ± 2.24 | 79.40 ± 2.27 |
| 7 | 90.82 ± 1.86 | 79.23 ± 2.31 | 78.45 ± 2.33 | 78.46 ± 2.39 |
| 8 | 90.92 ± 1.88 | 80.55 ± 2.23 | 78.84 ± 2.30 | 77.71 ± 2.42 |
| 9 | 90.58 ± 1.82 | 80.72 ± 2.21 | 78.35 ± 2.38 | 77.92 ± 2.44 |
| 10 | 88.86 ± 1.84 | 79.15 ± 2.20 | 75.79 ± 2.41 | 77.01 ± 2.32 |

Table 2: Overlap in percent of top attended regions for differently augmented input data, calculated for a number of top regions $n = 1, \ldots, 10$. The values represent averages over the 90 exemplary patches, with corresponding standard errors included.

we observe that flipping the original image has a more substantial impact on the overlap measure than color jitter. Additionally, these effects remain largely independent of the number of regions compared. Turning to the overlap measure across different resolutions, as depicted in fig. 13, we see that although the MSE increases significantly with decreasing resolution (see fig. 12), the overlap of the top attended regions remains relatively stable across resolutions ranging from 3584 to 1024 pixels. This behavior is consistent for up to six top attended regions, and the largest decrease in overlap across all numbers of attended regions is about 10%.

Consequently, if one decides not to feed the UNI model with images at their original resolution and focus solely on the resulting attention maps, substantial resizing can be applied without a notable loss of contextual information in the attention maps. However, if the embeddings produced by UNI are also of interest, resizing presents a trade-off between embedding similarity and execution time.

## 4.2 Proposed Method

This section integrates all presented methods and insights gained from the UNI analysis (section 4.1) to address the classification problem. I first outline how I process and structure WSIs using my PC package and the UNI model, incorporating findings from my analysis to inform key processing decisions. I then describe the construction and training of a deep neural network classifier, followed by a thorough evaluation of its performance, including an analysis of how different data processing choices affect the results.

### 4.2.1 The Data Processing Pipeline with the UNI Model

Here, I detail the specific steps involved in transforming raw WSIs into a structured representation for classification. I integrate preprocessing (section 2) with additional patch processing (section 3.3), guided by my findings from the UNI analysis (section 4.1). To preserve fine details in top-attended regions, I extract 4096-pixel image patches at full resolution. However, to balance computational efficiency with embedding quality, I downsample these patches to 2048 pixels

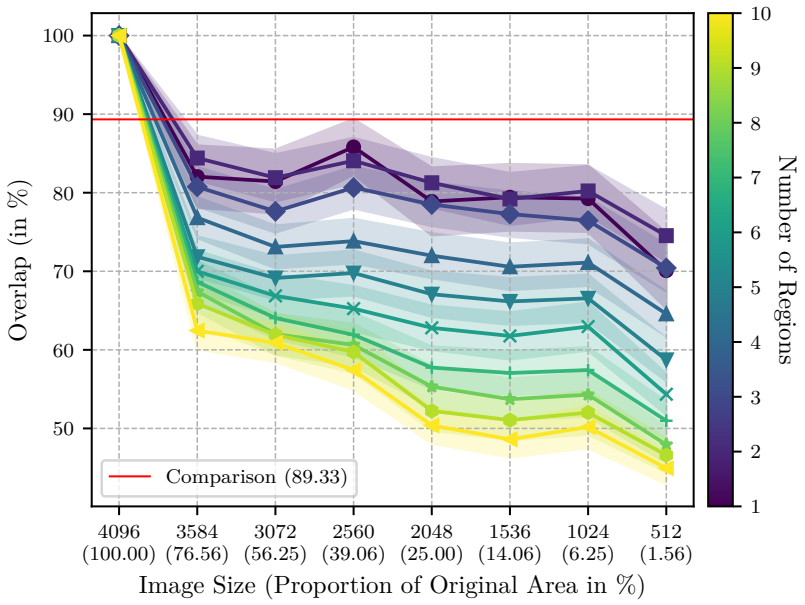



Figure 13: Overlap of top regions resulting from the attention maps plotted against different resolutions for a different number of regions to be compared. The value being displayed as red line is taken from table 2 for one region from the column 'All Combined'. The baseline of comparison are the attention maps resulting from full resolution, not augmented images and thus the overlap is equal to one hundred percent for the first x-value.

before feeding them into the UNI model. While this resizing introduces some differences in the embeddings, the most attended regions remain largely unaffected (section 4.1). Moreover, this approach aligns with the potential use of lower-resolution images, such as those captured by smartphones.

We use the PC package to sample $1024 \times 1024\,\mu m$ patches at the highest available resolution. In the Kiel dataset, a high-resolution tile represents $128 \times 128\,\mu m$ of tissue, corresponding to $512 \times 512$ pixels. Thus, one constructs $1024 \times 1024\,\mu m$ patches by combining eight adjacent tiles, resulting in $4096 \times 4096$-pixel images. To reduce computational overhead during training, I apply data augmentation directly during patch sampling. I sample patches that overlap by 50% to increase the dataset and ensure that each region appears in

```
slide_A/
    mean_std.pt
    patch_1.pt
    patch_2.pt
    ...
slide_B/
    mean_std.pt
    patch_1.pt
    patch_2.pt
    ...
```

Figure 14: Simplified visualization of the resulting file structure after sampling with PC. Each slide is stored in a dedicated directory containing its patches as individual files, along with the mean and standard deviation normalized by 255 in PyTorch tensor format.

multiple orientations. This is achieved by flipping and sampling the patches with overlapping regions. For details of this sampling process, see appendix A and my code.[6] The resulting file structure is shown in fig. 14.

To account for differences between classes, which might arise from differences in the data generating process of the WSIs, I compute the mean and standard deviation for each slide and color channel. I normalize these values by dividing by 255, the maximum pixel value, and store them for each slide (see fig. 14). Finally, I normalize pixel values by dividing by 255 and standardize each slide's patches to have zero mean and unit standard deviation using the precomputed values. This prevents any slide from dominating the training process due to systematic pixel intensity shifts for instance. After standardization, I resize patches to 2048 pixels and input them into the UNI model, which outputs the [`class`] token (one 1024-dimensional embedding for each patch, see fig. 6 and section 4.1.2). I also store the final-layer attention maps (see section 4.1.1) to analyze which regions the model focuses on during predictions. To ensure predictions are based on the most influential regions, I introduce an additional preparation step that focuses on identifying key areas for the model.

Pathologists begin diagnosis by examining WSIs at low resolution before zooming into regions of interest for final assessment (Ashman et al., 2022). To mimic this approach, I first process low-resolution patches with the UNI model and use the resulting attention maps to identify regions of interest. I then zoom into these regions by processing higher-resolution patches from the same areas. In practice, I use stored weight matrices to extract the coordinates of the $n$ most attended regions. These are extracted from a 1024 µm patch, represented at 2048 pixels, which corresponds to 25% of the original resolution. From these regions, I sample 100 µm and 200 µm patches at the highest resolution, both with a side length of 400 pixels (see section 4.1.1). The 200 µm patches therefore correspond to a lower resolution. We ensure that patches from different top regions do not

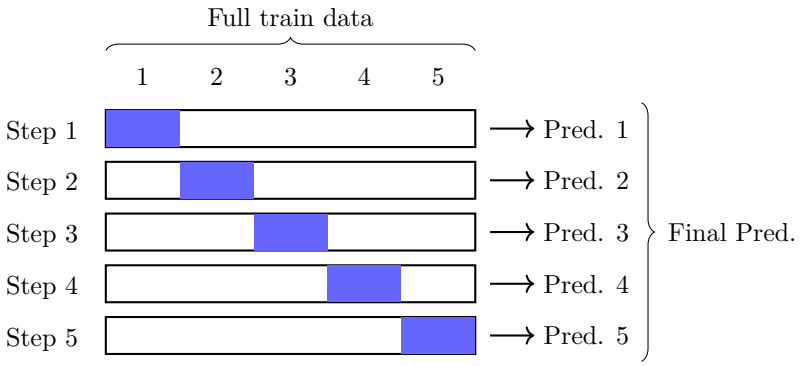



Figure 15: Visualization of the training process with five-fold cross-validation. The blue rectangles represent the evaluation data of each fold, while the rest of the data is used to train the model of the current fold. This results in five trained models, which are then used together as a model ensemble to arrive at a final prediction for a given patch. The figure is inspired by García et al., 2019, p.8.

overlap to prevent all high-resolution patches from coming from the same region of the low-resolution patch. I process these patches separately in the UNI network and concatenate their embeddings into a final vector of dimension $2n \cdot 1024$. This approach ensures that classification relies exclusively on highly attended regions while also incorporating embeddings from high-resolution patches. As

[6] https://github.com/FerdinandToelkes/patchcraft

shown in table 6 in appendix D, this approach does not improve classification performance but better aligns with the diagnostic workflow of pathologists.

#### 4.2.2 Building a Classifier as an Ensemble Model via Cross-Validation

Each WSI is transformed into a set of patch embeddings, generated from either whole low-resolution patches or zoomed-in, high-resolution top attended regions. These embeddings inherit the diagnosis label of the original WSI. My goal is to train a classifier that predicts the class of each patch embedding and then aggregate these predictions to obtain a slide-level diagnosis. Since our dataset contains a relatively small number of unique WSIs, I directly combine models from cross-validation into a final ensemble classifier, maximizing data efficiency. The following section explains this approach in detail.

Regardless of whether the embeddings were derived from full low-resolution patches or top-attended regions, I use a ResNet as classifier. A natural question arises: Why use a ResNet instead of a simple MLP to classify non-image data like embeddings? I chose ResNets to ensure a more realistic comparison in my ablation study, specifically, when comparing models trained on embeddings with those trained directly on images. Note that the number of parameters is

| | Patch | | Slide | |
|---|---|---|---|---|
| **Metric** | **SM** | **Ens** | **SM** | **Ens** |
| AUC | $0.9402 \pm 0.0071$ | 0.9708 | $0.9747 \pm 0.0031$ | 0.9787 |
| Accuracy | $0.7415 \pm 0.0175$ | 0.8493 | $0.8867 \pm 0.0179$ | 0.9333 |
| Balanced Acc. | $0.7503 \pm 0.0181$ | 0.8477 | $0.8867 \pm 0.0179$ | 0.9333 |
| Precision | $0.7424 \pm 0.0161$ | 0.8455 | $0.9021 \pm 0.0159$ | 0.9444 |
| Recall | $0.7503 \pm 0.0181$ | 0.8477 | $0.8867 \pm 0.0179$ | 0.9333 |
| F1-Score | $0.7400 \pm 0.0164$ | 0.8450 | $0.8841 \pm 0.0186$ | 0.9327 |
| Cross-Entropy | $0.7673 \pm 0.0342$ | 0.6662 | $0.6669 \pm 0.0220$ | 0.6440 |

Table 3: Comparison of performance metrics between single models (SM) and the ensemble model (Ens) for patch-level and slide-level predictions for models trained on embeddings of 1024 µm patches. Standard errors are shown only for the single model metrics, as they are obtained by averaging the results of each of the five single models trained in the cross-validation.

not significantly different between the two approaches. Since the embeddings are already quite low-dimensional, I use the smallest basic ResNet architecture, consisting of 18 layers, of which 17 are convolutional layers and the last layer is a fully connected layer (see K. He et al., 2016, p.5). The final layer transforms the network output into class probabilities. In our case, the network predicts the probability that a given patch belongs to one of the seven diagnostic categories: HL, DLBCL, CLL, FL, MCL, LTDS, and Unknown. Although ResNet18 is named after its 18 layers, I refer to it simply as ResNet, as this is the only variant used in my experiments. For further details on the architecture of the ResNet, see K. He et al., 2016 and my implementation.[7]

A key question remains: How does one aggregate patch predictions into a slide-level diagnosis? While conceptually simple, this step offers multiple possible approaches. I proceed as follows: My model processes all patches of a slide and counts the predicted class for each patch. The slide is then assigned the most frequently predicted class, following the majority voting approach. I use the AUC computed at the slide level for the evaluation data as a metric to

[7] https://github.com/FerdinandToelkes/lymphoma

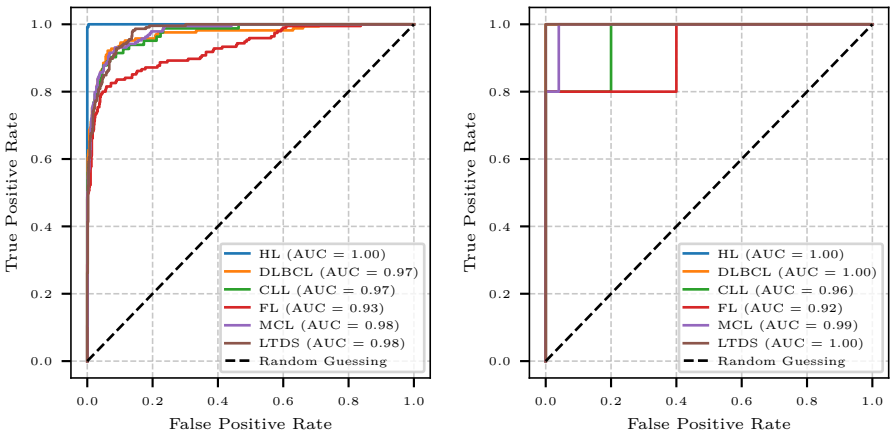



Figure 16: ROC curves per class for the one-versus-all setting of the model ensemble resulting from cross-validation on the test data, shown at both the patch (left) and slide (right) levels. The area under these curves gives the Area Under the Curve (AUC) values, which are also displayed within the plot.

track the performance of my model during training and to define a stopping criterion that ends training if the model does not improve its slide AUC over multiple epochs. Unlike majority voting, which assigns the most frequent class, the slide AUC is computed by averaging the model's probabilistic outputs across all patches in each evaluation slide.

To estimate the performance of my method for different hyperparameter sets (e.g., learning rate and LS), I perform a hyperparameter search using five-fold cross-validation, saving all models from each fold and hyperparameter set. Due to the limited number of unique WSIs, I will not retrain a model with the best hyperparameters found. Instead, I use the five models from the best hyperparameter set as an ensemble model, predicting a single patch by passing it through all five models and averaging their probabilistic outputs. The final prediction for the patch is the diagnosis corresponding to the highest output probability from the ensemble classifier. This approach is visualized in fig. 15. If one were to retrain a single model on the training data with the found hyperparameters, one would either need to fix the number of training epochs or use a portion of training data again as an evaluation set to allow the use of a stopping criterion. The advantage of this ensemble approach is that I can fully utilize the training data to construct a final predictor, since the five models together have seen the entire training data. Another possible advantage of this method is that the variance may be reduced by introducing more stochasticity due to the use of an ensemble model for a single prediction, increasing the robustness of the final model to fluctuations in the data. These effects are not rigorously studied here, but are observed, for example, when transitioning from decision trees to random forests as demonstrated by Breiman, 2001.

### 4.2.3 Evaluating the Ensemble Classifier

In this section, I evaluate ensemble classifiers trained on the Kiel dataset and analyze the impact of various model parameters and data processing steps. To establish baseline performance expectations, I first consider the class distributions of different test sets. When training on the Kiel dataset, the baseline accuracy is $1/6 = 16.67\%$ for its balanced test set, and $142/(94 + 86 + 93 + 142 + 99) = 27.63\%$ for the OOD setting when evaluating

on the Munich dataset (section 2.3).

The results of my hyperparameter search, conducted using five-fold cross-validation (section 4.2.2), are presented in table 6 in appendix D. This table compares two approaches: training on embeddings generated from the whole image patch versus concatenating embeddings from the top one, five, or ten attended regions. The classifier's performance appears relatively insensitive to the choice of hyperparameters. One of the best performances is achieved with a LS of 0.1 and a learning rate of 0.001 when training on embeddings from whole patches. I use this set of hyperparameters, along with the models trained on whole patch embeddings, to construct my model ensemble. For the final evaluation of this ensemble, I use an untouched test

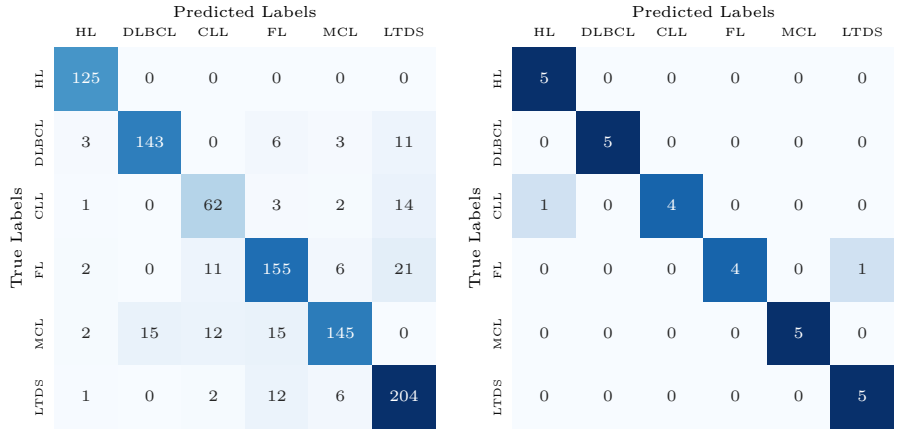



Figure 17: Confusion matrices for the ensemble model predictions on the test data on patch (left) and slide level (right). The ensemble classifier was taken from the cross-validation for training on 1024 µm patches from the Kiel dataset.

set consisting of randomly selected slides, with exactly five slides per class to ensure class balance. This test set represents approximately ten percent of the total available data from Kiel, given that we have 225 unique WSIs (section 2.3). The results of evaluating the ensemble and the individual models separately at both the patch and slide levels are summarized in table 3. These results show a consistent improvement when transitioning from individual cross-validation models to the ensemble. However, this improvement is slightly biased, as the ensemble has been exposed to more data than any individual model. However, even when comparing the performance of the ensemble classifier with that obtained during cross-validation (table 6), the ensemble outperforms single models. A comprehensive overview of the metrics used in table 3 can be found in Grandini et al., 2020, while the Receiver Operating Characteristic (ROC) and the AUC are discussed in Fawcett, 2006; Ling et al., 2003. The ROC curves that underlie the ensemble AUC values are provided in fig. 16. Additionally, the confusion matrices for the ensemble model at both the patch and slide levels are shown in fig. 17, with many of the metrics in table 3 derived from these matrices (see also Grandini et al., 2020). The patch-level confusion matrix confirms that classifying HL slides is easier than classifying the other five classes (section 4.1.3). I also experimented with learning rate warmup, as described in Kalra and Barkeshli, 2025, but found no significant improvements (see table 7 in appendix D). Figure 18 presents an example of an FL slide from the balanced test dataset, overlaid with predictions and attention maps. The slide's edges clearly reveal its patch composition. The majority of patches are correctly predicted as FL, leading to the correct diagnosis through majority voting. The attention maps do not exhibit specific patterns

| | Patch | | Slide | |
|---|---|---|---|---|
| **Metric** | **SM** | **Ens** | **SM** | **Ens** |
| AUC | $0.7514 \pm 0.0061$ | 0.7948 | $0.8182 \pm 0.0060$ | 0.8391 |
| Accuracy | $0.4157 \pm 0.0167$ | 0.4712 | $0.4770 \pm 0.0247$ | 0.4942 |
| Balanced Acc. | $0.4077 \pm 0.0103$ | 0.4542 | $0.4712 \pm 0.0218$ | 0.4849 |
| Precision | $0.3871 \pm 0.0072$ | 0.4501 | $0.4842 \pm 0.0110$ | 0.5252 |
| Recall | $0.3397 \pm 0.0086$ | 0.3785 | $0.3926 \pm 0.0182$ | 0.4041 |
| F1-Score | $0.3390 \pm 0.0139$ | 0.3798 | $0.3937 \pm 0.0241$ | 0.4011 |
| Cross-Entropy | $1.3699 \pm 0.0442$ | 1.2126 | $1.2200 \pm 0.0255$ | 1.1836 |

Table 4: Comparison of OOD performance metrics between single models (SM) and the ensemble model (Ens) for patch- and slide-level predictions. The corresponding models were trained on embeddings from 1024 µm patches. Standard errors are shown only for single model metrics since they originate from averaging over the results from each of the five single model which were trained in the cross-validation.

for patches with uniform tissue sections, mainly highlighting global features. A grid-like pattern is visible throughout the slide, an artifact of generating one attention map per patch rather than one for the entire slide. For better visualization, a larger version of the attention map is provided in appendix E in fig. 23.

Finally, I assess how my ensemble classifier performs on a dataset with a different distribution, namely the Munich dataset presented in section 2.3. This dataset does not contain the HL class. However, since my model was trained on data that includes it, false predictions of HL are possible. As shown in

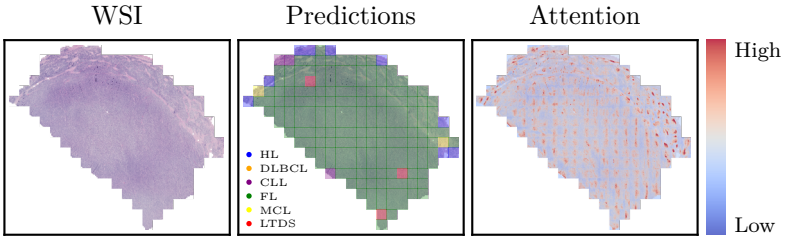



Figure 18: Visualization of a WSI displaying FL composed of 1024 µm patches (left), the ensemble model's color-coded predictions overlaid on the WSI (center), and the corresponding attention maps for each patch overlaid on the WSI (right).

table 4, performance drops significantly compared to the Kiel test set (table 3), indicating that further improvements are needed for OOD tasks. Results for the reverse setting, where the model is trained on Munich and tested on Kiel, are provided in appendix E, where it becomes evident that the Kiel dataset is easier to classify, in accordance to section 4.1.3.

Through my hyperparameter search, I found that using embeddings generated from full-resolution 100 and 200 µm patches does not improve performance (table 6 in appendix D). In the following, I further investigate different aspects of my method. Since training on the UNI embeddings is not very time consuming, one can easily rerun hyperparameter searches on this level. In contrast, training on the original image patches is much more time-intensive, so I restrict my search in this parameter space to one set of parameters.

We first extend the hyperparameter search for the embeddings generated from full patches (shown in the upper part of table 6 in appendix D) by including the possibilities of not using weighted loss as well as not using label smoothing regularization. These extended results are shown in table 5 and show the

| WL | LR | LS | Slide Accuracy (%) | Slide AUC |
|---|---|---|---|---|
| True | 0.001 | 0.0 | $81.78 \pm 1.88$ | $0.9503 \pm 0.0106$ |
| True | 0.001 | 0.1 | $84.09 \pm 1.43$ | $0.9488 \pm 0.0105$ |
| True | 0.001 | 0.2 | $83.64 \pm 1.59$ | $0.9460 \pm 0.0130$ |
| True | 0.001 | 0.3 | $81.39 \pm 1.89$ | $0.9367 \pm 0.0187$ |
| True | 0.0001 | 0.0 | $81.38 \pm 1.72$ | $0.9451 \pm 0.0108$ |
| True | 0.0001 | 0.1 | $81.37 \pm 2.12$ | $0.9425 \pm 0.0141$ |
| True | 0.0001 | 0.2 | $79.57 \pm 1.85$ | $0.9414 \pm 0.0110$ |
| True | 0.0001 | 0.3 | $76.83 \pm 2.03$ | $0.9346 \pm 0.0121$ |
| False | 0.001 | 0.0 | $80.87 \pm 2.23$ | $0.9530 \pm 0.0091$ |
| False | 0.001 | 0.1 | $81.76 \pm 2.13$ | $0.9579 \pm 0.0070$ |
| False | 0.001 | 0.2 | $80.89 \pm 1.83$ | $0.9540 \pm 0.0082$ |
| False | 0.001 | 0.3 | $81.32 \pm 2.61$ | $0.9520 \pm 0.0092$ |
| False | 0.0001 | 0.0 | $80.47 \pm 1.85$ | $0.9469 \pm 0.0086$ |
| False | 0.0001 | 0.1 | $80.47 \pm 2.04$ | $0.9476 \pm 0.0086$ |
| False | 0.0001 | 0.2 | $79.54 \pm 1.71$ | $0.9431 \pm 0.0100$ |
| False | 0.0001 | 0.3 | $78.63 \pm 2.00$ | $0.9331 \pm 0.0095$ |

Table 5: Results of the extended hyperparameter search using five-fold cross-validation. The models were trained on embeddings of 1024 µm patches. The AUC is an average of the one-versus-all AUC values for the different classes. The standard errors are computed over the five scores from each cross-validation. WL stands for weighted loss, LR for learning rate and LS for label smoothing.

indifference of using moderated LS and a weighted version of the cross-entropy loss or not. Next, I repeat the hyperparameter search without standardizing the patches using the means and standard deviations calculated for each slide. These results, presented in the top part of table 7 in appendix D, show that standardization is not critical for the model's performance, although pixel values are still normalized by dividing by 255. I conclude that when training models directly on the image patches, using a fixed set of hyperparameters, specifically, weighted loss, a learning rate of 0.001, and LS with a strength of 0.1, is sufficient.

Moreover, I only use normalization to speed up training, as training on image patches is time-consuming. As the 4096-pixel image patches are too large for the available GPUs, I have to resize them to a side length of 512 pixels. This is one of the advantages of the UNI model, as it allows for processing larger image patches. Training a ResNet with five-fold cross-validation under these settings results in a slide accuracy of $79.19\% \pm 2.53\%$ and a slide AUC of $0.9312 \pm 0.0154$. This performance aligns with the results from the hyperparameter search on patch embeddings (e.g., table 5), indicating no significant performance loss when training on the lower-dimensional embeddings generated by the UNI ViT.

# 5 Conclusion

In this work, I prepared our datasets for deep learning applications by introducing the PC package, which facilitates convenient sampling from WSIs stored in a standardized .sqlite format (section 2.2).

As part of my analysis of the UNI ViT, I addressed the prohibitively large memory requirements of storing full attention maps and introduced an efficient

method to save only the relevant portions, specifically, how embeddings attend to different regions of the input image (section 4.1.1). I then examined the impact of input resolution on embeddings, finding that differences between embeddings, measured using MSE and cosine similarity, increase with downsampling (section 4.1.2). Here, cropping proved to be a more effective downsampling method than resizing via bilinear interpolation. Additionally, I visualized the embeddings used for training through the UMAP algorithm, identifying differences between the Kiel and Munich datasets (section 4.1.3). These differences, among others, prompted further investigation by pathologists in our project group. Furthermore, I studied the effect of input resizing on attention maps by introducing an overlap measure to track changes in the most highly attended regions. I found that this overlap measure remained relatively stable across a wide range of resolutions (section 4.1.4).

Finally, I developed a data processing pipeline with the UNI network, incorporating insights from my analysis to reduce training time without compromising model performance (section 4.2.1). Building on this pipeline, I introduced an ensemble model based on five-fold cross-validation, which outperformed individual models (section 4.2.2). My evaluation demonstrated high classification performance (93%) on the Kiel dataset using low-resolution patches, validating the use of embeddings rather than raw images for training the final classifier while significantly accelerating training. I further evaluated the model's OOD performance on the Munich dataset, where classification accuracy dropped significantly to 45% (section 4.2.3).

These findings align with broader challenges in digital pathology, particularly within the 'Federated Learning in Lymphoma Pathology' initiative, a project funded by the German Ministry of Education and Research. Other members of our research group working on these datasets with different approaches have observed similar performance discrepancies between the Kiel and Munich datasets. This raises a key question: how do the two datasets differ, and is more standardization of the scanning process necessary to facilitate comparisons across research groups? To improve model robustness under such dataset differences, applying more aggressive data augmentation techniques might be beneficial. Additionally, a second version of the UNI model was recently released, trained on a dataset containing even more tissue samples, which may further enhance performance on the Munich dataset.

To advance the use of smartphone images for classifying lymphoma, further investigation into how downsampling affects the outputs of foundation models in CPath would be valuable. Beyond analyzing differences in embeddings, future work could assess the impact of resolution changes on downstream tasks such as classification. Additionally, examining the interplay between input resolution and patch size could provide deeper insights into model performance. A more rigorous comparison between models trained on embeddings and those trained directly on images should also be conducted, ensuring that both approaches start from images of the same resolution. Finally, leveraging attention maps from large, low-resolution patches to identify regions of interest requires careful validation against expert annotations from pathologists. I leave these explorations to future research.

# References


Aben, N., de Jong, E. D., Gatopoulos, I., Känzig, N., Karasikov, M., Lagré, A., Moser, R., van Doorn, J., Tang, F., et al. (2024). Towards large-scale training of pathology foundation models. *arXiv preprint arXiv:2404.15217*.

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Alfasly, S., Shafique, A., Nejat, P., Khan, J., Alsaafin, A., Alabtah, G., & Tizhoosh, H. R. (2024). Rotation-agnostic image representation learning for digital pathology. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11683–11693.

Anand, D., Singhal, V., Shanbhag, D. D., KS, S., Patil, U., Bhushan, C., Manickam, K., Gui, D., Mullick, R., Gopal, A., et al. (2023). One-shot localization and segmentation of medical images with foundation models. *arXiv preprint arXiv:2310.18642*.

Ashman, K., Zhuge, H., Shanley, E., Fox, S., Halat, S., Sholl, A., Summa, B., & Brown, J. Q. (2022). Whole slide image data utilization informed by digital diagnosis patterns. *Journal of Pathology Informatics*, *13*, 100113.

Aurelio, Y. S., De Almeida, G. M., de Castro, C. L., & Braga, A. P. (2019). Learning from imbalanced data sets with weighted cross-entropy function. *Neural processing letters*, *50*, 1937–1949.

Ayzenberg, L., Giryes, R., & Greenspan, H. (2024). Dinov2 based self supervised learning for few shot medical image segmentation. *arXiv preprint arXiv:2403.03273*.

Baharoon, M., Qureshi, W., Ouyang, J., Xu, Y., Phol, K., Aljouie, A., & Peng, W. (2023). Towards general purpose vision foundation models for medical image analysis: An experimental study of dinov2 on radiology benchmarks. *arXiv preprint arXiv:2312.02366*.

Bengio, Y., Goodfellow, I., & Courville, A. (2017). *Deep learning* (Vol. 1). MIT press Cambridge, MA, USA.

Bishop, C. M., & Nasrabadi, N. M. (2006). *Pattern recognition and machine learning* (Vol. 4). Springer.

Brauwers, G., & Frasincar, F. (2021). A general survey on attention mechanisms in deep learning. *IEEE Transactions on Knowledge and Data Engineering*, *35*(4), 3279–3298.

Breiman, L. (2001). Random forests. *Machine learning*, *45*, 5–32.

Brinkel, J., Krämer, A., Krumkamp, R., May, J., & Fobil, J. (2014). Mobile phone-based mhealth approaches for public health surveillance in sub-saharan africa: A systematic review. *International journal of environmental research and public health*, *11*(11), 11559–11582.

Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., & Joulin, A. (2021). Emerging properties in self-supervised vision transformers. *Proceedings of the IEEE/CVF international conference on computer vision*, 9650–9660.

Chan, J. K. (2014). The wonderful colors of the hematoxylin–eosin stain in diagnostic surgical pathology. *International journal of surgical pathology*, *22*(1), 12–32.

Chen, R. J., Ding, T., Lu, M. Y., Williamson, D. F., Jaume, G., Chen, B., Zhang, A., Shao, D., Song, A. H., Shaban, M., et al. (2023). A general-purpose self-supervised model for computational pathology.

Contributors, P. (2023). *Crossentropyloss*. Retrieved October 12, 2024, from https://pytorch.org/blog/flexattention/

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, *2*(4), 303–314.

Dao, T., Fu, D., Ermon, S., Rudra, A., & Ré, C. (2022). Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, *35*, 16344–16359.

Devlin, J. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dimitriou, N., Arandjelović, O., & Caie, P. D. (2019). Deep learning for whole slide image analysis: An overview. *Frontiers in medicine*, *6*, 264.

Dippel, J., Feulner, B., Winterhoff, T., Milbich, T., Tietz, S., Schallenberg, S., Dernbach, G., Kunft, A., Heinke, S., Eich, M.-L., et al. (2024). Rudolfv: A foundation model by pathologists for pathologists. *arXiv preprint arXiv:2401.04079*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale.

Duenweg, S. R., Bobholz, S. A., Lowman, A. K., Stebbins, M. A., Winiarz, A., Nath, B., Kyereme, F., Iczkowski, K. A., & LaViolette, P. S. (2023). Whole slide imaging (wsi) scanner differences influence optical and computed properties of digitized prostate cancer histology. *Journal of Pathology Informatics*, *14*, 100321.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, *27*(8), 861–874.

Feldman, A. T., & Wolfe, D. (2014). Tissue processing and hematoxylin and eosin staining. *Histopathology: methods and protocols*, 31–43.

Frederiksen, J. K., Sharma, M., Casulo, C., & Burack, W. R. (2015). Systematic review of the effectiveness of fine-needle aspiration and/or core needle biopsy for subclassifying lymphoma. *Archives of Pathology and Laboratory Medicine*, *139*(2), 245–251.

García, V., Sánchez, J. S., & Marqués, A. I. (2019). Synergetic application of multi-criteria decision-making models to credit granting decision problems. *Applied Sciences*, *9*(23), 5052.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 249–256.

Grandini, M., Bagli, E., & Visani, G. (2020). Metrics for multi-class classification: An overview. *arXiv preprint arXiv:2008.05756*.

Gurcan, M. N., Boucheron, L. E., Can, A., Madabhushi, A., Rajpoot, N. M., & Yener, B. (2009). Histopathological image analysis: A review. *IEEE reviews in biomedical engineering*, *2*, 147–171.

Han, K., Wang, Y., Chen, H., Chen, X., Guo, J., Liu, Z., Tang, Y., Xiao, A., Xu, C., Xu, Y., et al. (2022). A survey on vision transformer. *IEEE transactions on pattern analysis and machine intelligence*, *45*(1), 87–110.

He, Guessous, L., & Dong. (2024). *Flexattention: The flexibility of pytorch with the performance of flashattention*. Retrieved October 12, 2024, from https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

He, K., & Sun, J. (2015). Convolutional neural networks at constrained time cost. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 5353–5360.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hernández-Neuta, I., Neumann, F., Brightmeyer, J., Ba Tis, T., Madaboosi, N., Wei, Q., Ozcan, A., & Nilsson, M. (2019). Smartphone-based clinical diagnostics: Towards democratization of evidence-based health care. *Journal of internal medicine*, *285*(1), 19–39.

Huang, Y., Zou, J., Meng, L., Yue, X., Zhao, Q., Li, J., Song, C., Jimenez, G., Li, S., & Fu, G. (2024). Comparative analysis of imagenet pre-trained deep learning models and dinov2 in medical imaging classification. *arXiv preprint arXiv:2402.07595*.

Huttner, M. (2023). *Pamly-lib*. Retrieved February 10, 2025, from https://github.com/spang-lab/pamly-lib

Kalra, D. S., & Barkeshli, M. (2025). Why warmup the learning rate? underlying mechanisms and improvements. *Advances in Neural Information Processing Systems*, *37*, 111760–111801.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.

Kumar, N., Gupta, R., & Gupta, S. (2020). Whole slide imaging (wsi) in pathology: Current perspectives and future directions. *Journal of digital imaging*, *33*(4), 1034–1040.

Kundu, B., Khanal, B., Simon, R., & Linte, C. A. (2024). Assessing the performance of the dinov2 self-supervised learning vision transformer model for the segmentation of the left atrium from mri images. *arXiv preprint arXiv:2411.09598*.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Ling, C. X., Huang, J., Zhang, H., et al. (2003). Auc: A statistically consistent and more discriminating measure than accuracy. *Ijcai*, *3*, 519–524.

Lu, M. Y., Chen, B., Williamson, D. F., Chen, R. J., Liang, I., Ding, T., Jaume, G., Odintsov, I., Le, L. P., Gerber, G., et al. (2024). A visual-language foundation model for computational pathology. *Nature Medicine*, *30*(3), 863–874.

Märkl, B., Füzesi, L., Huss, R., Bauer, S., & Schaller, T. (2021). Number of pathologists in germany: Comparison with european countries, usa, and canada. *Virchows Archiv*, *478*, 335–341.

McInnes, L., Healy, J., & Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Müller, R., Kornblith, S., & Hinton, G. E. (2019). When does label smoothing help? *Advances in neural information processing systems*, *32*.

Muthukrishnan, N., Maleki, F., Ovens, K., Reinhold, C., Forghani, B., Forghani, R., et al. (2020). Brief history of artificial intelligence. *Neuroimaging Clinics of North America*, *30*(4), 393–399.

Nechaev, D., Pchelnikov, A., & Ivanova, E. (2024). Hibou: A family of foundational vision transformers for pathology. *arXiv preprint arXiv:2406.05074*.

Niu, Z., Zhong, G., & Yu, H. (2021). A review on the attention mechanism of deep learning. *Neurocomputing*, *452*, 48–62.

Oquab, M., Darcet, T., Moutakanni, T., Vo, H., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Massa, F., El-Nouby, A., et al. (2023). Dinov2: Learning robust visual features without supervision.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, *32*.

Popescu, M.-C., Balas, V. E., Perescu-Popescu, L., & Mastorakis, N. (2009). Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, *8*(7), 579–588.

Qaiser, T., & Rajpoot, N. M. (2019). Learning where to see: A novel attention model for automated immunohistochemical scoring. *IEEE transactions on medical imaging*, *38*(11), 2620–2631.

Rozman, C., & Montserrat, E. (1995). Chronic lymphocytic leukemia. *New England Journal of Medicine*, *333*(16), 1052–1057.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*, 211–252.

Sanderson, G. (2024). Visualizing attention, a transformer's heart [Accessed: 2024-08-20].

Smith, A., Crouch, S., Lax, S., Li, J., Painter, D., Howell, D., Patmore, R., Jack, A., & Roman, E. (2015). Lymphoma incidence, survival and prevalence 2004–2014: Sub-type analyses from the uk's haematological malignancy research network. *British journal of cancer*, *112*(9), 1575–1584.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2818–2826.

Vaswani, A. (2017). Attention is all you need.

Vorontsov, E., Bozkurt, A., Casson, A., Shaikovski, G., Zelechowski, M., Liu, S., Severson, K., Zimmermann, E., Hall, J., Tenenholtz, N., et al. (2023). Virchow: A million-slide digital pathology foundation model. *arXiv preprint arXiv:2309.07778*.

Walsh, E., & Orsi, N. M. (2024). The current troubled state of the global pathology workforce: A concise review. *Diagnostic Pathology*, *19*(1), 163.

Wang, X., Chen, H., Gan, C., Lin, H., Dou, Q., Tsougenis, E., Huang, Q., Cai, M., & Heng, P.-A. (2019). Weakly supervised deep learning for whole slide lung cancer image analysis. *IEEE transactions on cybernetics*, *50*(9), 3950–3962.

Xu, H., Usuyama, N., Bagga, J., Zhang, S., Rao, R., Naumann, T., Wong, C., Gero, Z., González, J., Gu, Y., et al. (2024). A whole-slide foundation model for digital pathology from real-world data. *Nature*, 1–8.

Zhou, J., Wei, C., Wang, H., Shen, W., Xie, C., Yuille, A., & Kong, T. (2021). Ibot: Image bert pre-training with online tokenizer.

# Appendices

## A Details on our Data Sampling

I have chosen to use the `sample_tiles` command for my data sampling, with the various options and arguments specified within the .yaml file shown below. This file is automatically saved during sampling to ensure reproducibility. To sample test data, the overlap is disabled and the mode as well as the prefix of the output path is set to 'test' instead of 'train'.

**The config.yaml file used for sampling training data**

```yaml
general_transforms:
  sampling:
    highest_zoom_level: true
    overlap: 0.5
    overlap_bool: true
    patch_size_um: 1024
    random_seed: 1024
    stain: 'HE'
    wsi_pixels_per_m: 4000000
input:
  info_filename: 'metadata'
  overview_filename: 'overview.yaml'
  path: '/data'
output:
  desired_metadata:
  - 'filename'
  - 'diagnosis'
  - 'stain'
  log_level: 'INFO'
  mode: 'train'
  number_of_patches_per_slide: 'all'
  number_of_repeated_patches: 1
  number_of_slides: '50'
  path: '../train_1024um_patches'
  start_slide: 0
  white_threshold: 0.25
training_transforms:
  color_jitter:
    brightness_jitter: 0.1
    contrast_jitter: 0.1
    hue_jitter: 0.025
    saturation_jitter: 0.1
  flips:
    enabled: true
  rotation:
    enabled: true
```

# B Efficient Saving of Attention Maps with a Custom Wrapper Function

To efficiently store the UNI model's attention maps, I modify its final layer. Since I use the [class] token from the UNI ViT as a lower-dimensional representation of our images, I am specifically interested in how this token attends to different parts of the input image. This allows me to discard the query vectors of all patches, significantly reducing computation and memory requirements. The corresponding Python function is shown below, where operations unnecessary for inference have been omitted.[8]

**A custom wrapper function to save attention maps**

```python
def forward_wrapper(attn_obj):
    def attn_saving_forward(x):
        B, N, C = x.shape # N: sequence length
        nh = attn_obj.num_heads
        # Compute q, k, v for all patches
        qkv = attn_obj.qkv(x).reshape(B, N, 3, nh, C // nh)
        qkv = qkv.permute(2, 0, 3, 1, 4)
        q, k, v = qkv.unbind(0)

        # Only use the query for the class token (index 0)
        q = q[:, :, 0, :]
        q = q.unsqueeze(2) # allows multiplication with k

        # Use the keys and values of all patches
        k = k[:, :, 1:, :] # exclude class token
        v = v[:, :, 1:, :] # exclude class token

        # Compute the class-to-patch attention
        attn = (q @ k.transpose(-2, -1)) * attn_obj.scale
        attn = attn.softmax(dim=-1)

        # Save the attention map for class token only
        attn_obj.attn_map = attn

        # Compute the output
        x = (attn @ v).transpose(1, 2).reshape(B, 1, C)
        # project back to the original dimension
        x = attn_obj.proj(x)
        return x
    return attn_saving_forward
```

[8] https://github.com/FerdinandToelkes/lymphoma

# C UMAPs for different Hyperparameters

Due to the probabilistic elements of the UMAP method, it is recommended to run the algorithm more than once and look at the results. I regenerated the visualization shown in section 4.1.3 for the number of nearest neighbors set to 10 and 20. The resulting plots are shown in figs. 19 and 20 and their overall structure is not significantly different from the plots generated with 15 nearest neighbors (see section 4.1.3).

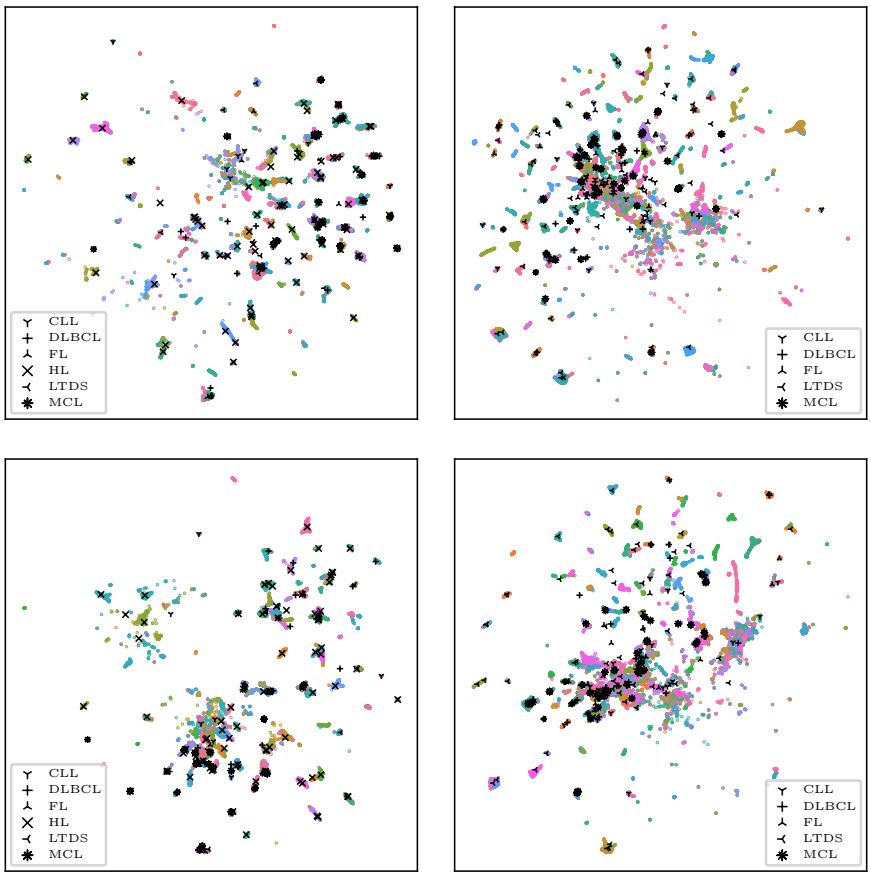



Figure 19: Embeddings resulting from patches sampled in test mode for the Kiel (left) and Munich (right) datasets, visualized in two dimensions using UMAP with a minimum distance of 0.1. The top row uses 10 nearest neighbors, while the bottom row uses 20 nearest neighbors. Embeddings belonging to the same slide are shown in the same color, and a marker corresponding to the respective class is shown at the median x-y coordinates for each slide.

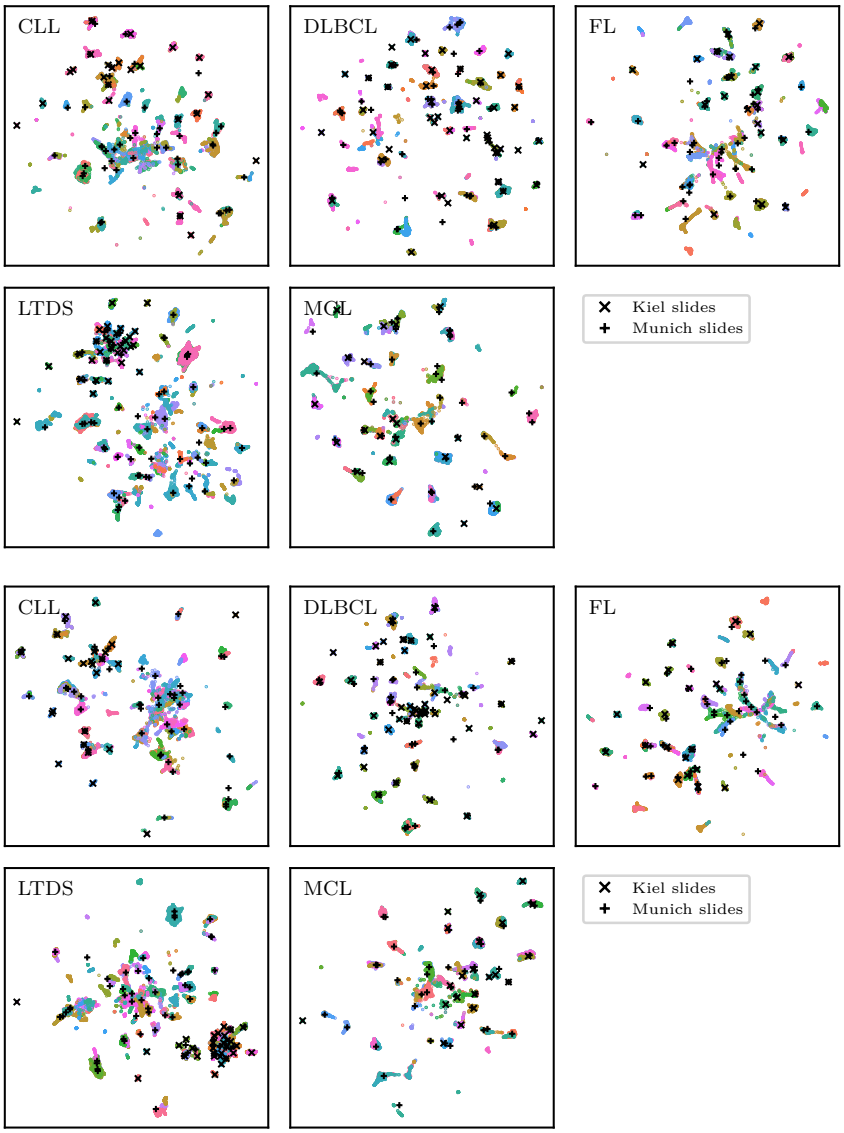



Figure 20: Embeddings resulting from patches sampled in test mode for the Kiel and Munich datasets for each class separately, visualized in two dimensions using UMAP with 10 (top) and 20 (bottom) nearest neighbors and a minimum distance of 0.1. Embeddings belonging to the same slide are shown in the same color, and a marker corresponding to the respective class is shown at the median x,y coordinates for each slide.

# D Extended Cross-Validation Results

To assess the sensitivity of my ResNet classifier to different hyperparameters, I conducted hyperparameter searches across various data processing methods. The results of these searches are presented below.

| Embedding Type | LR | LS | Slide Accuracy (%) | Slide AUC |
|---|---|---|---|---|
| Whole Image | 0.001 | 0.1 | $84.09 \pm 1.43$ | $0.9488 \pm 0.0105$ |
| Whole Image | 0.001 | 0.2 | $83.64 \pm 1.59$ | $0.9460 \pm 0.0130$ |
| Whole Image | 0.001 | 0.3 | $81.39 \pm 1.89$ | $0.9367 \pm 0.0187$ |
| Whole Image | 0.0001 | 0.1 | $81.37 \pm 2.12$ | $0.9425 \pm 0.0141$ |
| Whole Image | 0.0001 | 0.2 | $79.57 \pm 1.85$ | $0.9414 \pm 0.0110$ |
| Whole Image | 0.0001 | 0.3 | $76.83 \pm 2.03$ | $0.9346 \pm 0.0121$ |
| Top One Region | 0.001 | 0.1 | $83.21 \pm 0.94$ | $0.9392 \pm 0.0101$ |
| Top One Region | 0.001 | 0.2 | $81.38 \pm 1.45$ | $0.9295 \pm 0.0145$ |
| Top One Region | 0.001 | 0.3 | $79.55 \pm 1.79$ | $0.9280 \pm 0.0163$ |
| Top One Region | 0.0001 | 0.1 | $78.65 \pm 2.07$ | $0.9268 \pm 0.0119$ |
| Top One Region | 0.0001 | 0.2 | $76.87 \pm 1.85$ | $0.9114 \pm 0.0157$ |
| Top One Region | 0.0001 | 0.3 | $74.19 \pm 3.13$ | $0.9120 \pm 0.0156$ |
| Top Five Regions | 0.001 | 0.1 | $82.72 \pm 1.69$ | $0.9303 \pm 0.0149$ |
| Top Five Regions | 0.001 | 0.2 | $82.76 \pm 1.27$ | $0.9323 \pm 0.0154$ |
| Top Five Regions | 0.001 | 0.3 | $79.55 \pm 1.28$ | $0.9262 \pm 0.0167$ |
| Top Five Regions | 0.0001 | 0.1 | $80.48 \pm 1.84$ | $0.9285 \pm 0.0116$ |
| Top Five Regions | 0.0001 | 0.2 | $78.25 \pm 2.04$ | $0.9211 \pm 0.0094$ |
| Top Five Regions | 0.0001 | 0.3 | $73.76 \pm 3.81$ | $0.9080 \pm 0.0208$ |
| Top Ten Regions | 0.001 | 0.1 | $83.19 \pm 0.79$ | $0.9338 \pm 0.0135$ |
| Top Ten Regions | 0.001 | 0.2 | $82.72 \pm 1.07$ | $0.9328 \pm 0.0149$ |
| Top Ten Regions | 0.001 | 0.3 | $80.91 \pm 0.44$ | $0.9239 \pm 0.0182$ |
| Top Ten Regions | 0.0001 | 0.1 | $78.23 \pm 2.11$ | $0.9252 \pm 0.0130$ |
| Top Ten Regions | 0.0001 | 0.2 | $77.31 \pm 1.60$ | $0.9173 \pm 0.0114$ |
| Top Ten Regions | 0.0001 | 0.3 | $74.15 \pm 2.51$ | $0.9175 \pm 0.0178$ |

Table 6: Results of a hyperparameter search using five-fold cross-validation. The models were trained on embeddings of 1024 µm patches as well as concatenated embeddings corresponding to the top one, five, or ten attended regions of the underlying patch. The rows corresponding to the best-performing combinations of hyperparameters, as measured by the AUC and the slide accuracy, are highlighted. The AUC is an average of the one-versus-all AUC values for the different classes. The standard errors are computed over the five scores from each cross-validation. LR stands for learning rate and LS for label smoothing.

| WL | LR | LS | WU | Slide Acc. (%) | Slide AUC |
|---|---|---|---|---|---|
| **Hyperparameter search without standardization** | | | | | |
| True | 0.001 | 0.0 | 0 | $82.73 \pm 2.30$ | $0.9498 \pm 0.0096$ |
| True | 0.001 | 0.1 | 0 | $83.22 \pm 1.85$ | $0.9558 \pm 0.0114$ |
| True | 0.001 | 0.2 | 0 | $83.69 \pm 1.88$ | $0.9519 \pm 0.0133$ |
| True | 0.001 | 0.3 | 0 | $81.84 \pm 1.36$ | $0.9435 \pm 0.0166$ |
| True | 0.0001 | 0.0 | 0 | $83.18 \pm 1.65$ | $0.9480 \pm 0.0097$ |
| True | 0.0001 | 0.1 | 0 | $80.92 \pm 1.62$ | $0.9474 \pm 0.0106$ |
| True | 0.0001 | 0.2 | 0 | $79.57 \pm 1.35$ | $0.9405 \pm 0.0135$ |
| True | 0.0001 | 0.3 | 0 | $80.04 \pm 1.76$ | $0.9425 \pm 0.0131$ |
| False | 0.001 | 0.0 | 0 | $82.29 \pm 1.94$ | $0.9525 \pm 0.0102$ |
| False | 0.001 | 0.1 | 0 | $83.19 \pm 2.08$ | $0.9526 \pm 0.0107$ |
| False | 0.001 | 0.2 | 0 | $83.63 \pm 2.34$ | $0.9613 \pm 0.0080$ |
| False | 0.001 | 0.3 | 0 | $82.27 \pm 2.00$ | $0.9513 \pm 0.0117$ |
| False | 0.0001 | 0.0 | 0 | $81.86 \pm 2.16$ | $0.9437 \pm 0.0107$ |
| False | 0.0001 | 0.1 | 0 | $80.94 \pm 2.41$ | $0.9466 \pm 0.0080$ |
| False | 0.0001 | 0.2 | 0 | $80.45 \pm 2.61$ | $0.9459 \pm 0.0098$ |
| False | 0.0001 | 0.3 | 0 | $80.08 \pm 2.65$ | $0.9422 \pm 0.0114$ |
| **Hyperparameter search with learning rate warmup** | | | | | |
| True | 0.01 | 0.0 | 5 | $83.65 \pm 2.04$ | $0.9514 \pm 0.0085$ |
| True | 0.01 | 0.1 | 5 | $85.01 \pm 1.77$ | $0.9530 \pm 0.0086$ |
| True | 0.01 | 0.2 | 5 | $86.37 \pm 0.88$ | $0.9485 \pm 0.0083$ |
| True | 0.01 | 0.3 | 5 | $85.00 \pm 1.36$ | $0.9504 \pm 0.0100$ |
| True | 0.001 | 0.0 | 5 | $82.27 \pm 1.16$ | $0.9505 \pm 0.0077$ |
| True | 0.001 | 0.1 | 5 | $84.55 \pm 1.47$ | $0.9460 \pm 0.0115$ |
| True | 0.001 | 0.2 | 5 | $84.11 \pm 2.17$ | $0.9478 \pm 0.0122$ |
| True | 0.001 | 0.3 | 5 | $82.28 \pm 2.04$ | $0.9443 \pm 0.0123$ |
| True | 0.01 | 0.0 | 10 | $83.59 \pm 1.84$ | $0.9532 \pm 0.0063$ |
| True | 0.01 | 0.1 | 10 | $85.91 \pm 1.49$ | $0.9554 \pm 0.0080$ |
| True | 0.01 | 0.2 | 10 | $85.46 \pm 1.19$ | $0.9502 \pm 0.0100$ |
| True | 0.01 | 0.3 | 10 | $85.00 \pm 1.36$ | $0.9461 \pm 0.0125$ |
| True | 0.001 | 0.0 | 10 | $82.73 \pm 1.87$ | $0.9520 \pm 0.0098$ |
| True | 0.001 | 0.1 | 10 | $84.55 \pm 1.47$ | $0.9520 \pm 0.0091$ |
| True | 0.001 | 0.2 | 10 | $84.98 \pm 1.06$ | $0.9483 \pm 0.0109$ |
| True | 0.001 | 0.3 | 10 | $81.85 \pm 1.64$ | $0.9458 \pm 0.0114$ |

Table 7: Results of the hyperparameter search using five-fold cross-validation. The models were trained on embeddings of 1024 µm patches belonging to the Kiel dataset. The standard errors are computed over the five scores from each cross-validation. WL stands for weighted loss, LR for learning rate, LS for label smoothing, and WU for warmup epochs. The first part of the table shows results without standardization, and the second part shows results with learning rate warmup.

# E Extended Results on Test Data

To compare the classification difficulty between the two datasets, I also trained an ensemble classifier via cross-validation on the Munich data using a weighted loss, a learning rate of 0.001, and a LS of 0.1. The corresponding results are presented in table 8 and fig. 21. Additionally, fig. 22 shows the confusion matrices and ROC curves for the ensemble model trained on Kiel data and evaluated on the Munich dataset. Finally, a larger version of the right side of fig. 18 is provided for convenience.

| | **Patch** | | **Slide** | |
|---|---|---|---|---|
| **Metric** | **SM** | **Ens** | **SM** | **Ens** |
| **In-Distribution Performance (Munich Data)** | | | | |
| AUC | $0.8300 \pm 0.0080$ | 0.8642 | $0.8668 \pm 0.0042$ | 0.8750 |
| Accuracy | $0.5993 \pm 0.0070$ | 0.6466 | $0.6160 \pm 0.0119$ | 0.6400 |
| Balanced Acc. | $0.5543 \pm 0.0048$ | 0.6002 | $0.6160 \pm 0.0119$ | 0.6400 |
| Precision | $0.5541 \pm 0.0056$ | 0.6046 | $0.6469 \pm 0.0087$ | 0.6847 |
| Recall | $0.5543 \pm 0.0048$ | 0.6002 | $0.6160 \pm 0.0119$ | 0.6400 |
| F1-Score | $0.5484 \pm 0.0052$ | 0.5964 | $0.6089 \pm 0.0109$ | 0.6373 |
| Cross-Entropy | $1.1348 \pm 0.0322$ | 0.9827 | $1.0141 \pm 0.0140$ | 0.9901 |
| **Out-of-Distribution Performance (Kiel Data)** | | | | |
| AUC | $0.7788 \pm 0.0048$ | 0.8283 | $0.8819 \pm 0.0037$ | 0.9009 |
| Accuracy | $0.4924 \pm 0.0067$ | 0.5575 | $0.6365 \pm 0.0080$ | 0.6647 |
| Balanced Acc. | $0.4966 \pm 0.0058$ | 0.5634 | $0.6122 \pm 0.0068$ | 0.6356 |
| Precision | $0.5033 \pm 0.0031$ | 0.5682 | $0.6327 \pm 0.0057$ | 0.6531 |
| Recall | $0.4966 \pm 0.0058$ | 0.5634 | $0.6122 \pm 0.0068$ | 0.6356 |
| F1-Score | $0.4814 \pm 0.0069$ | 0.5449 | $0.5968 \pm 0.0100$ | 0.6229 |
| Cross-Entropy | $1.3558 \pm 0.0204$ | 1.1652 | $1.0429 \pm 0.0148$ | 1.0156 |

Table 8: Comparison of performance metrics between single models (SM) and model ensembles (Ens) for patch-level and slide-level predictions, both for in-distribution (Munich data) and out-of-distribution (Kiel data) settings. Standard errors are shown only for single model metrics, as they originate from averaging over results from each of the five single models trained in cross-validation. Note that the Munich dataset does not contain the HL class and it is thus excluded from the Kiel dataset to ensure fair evaluation.

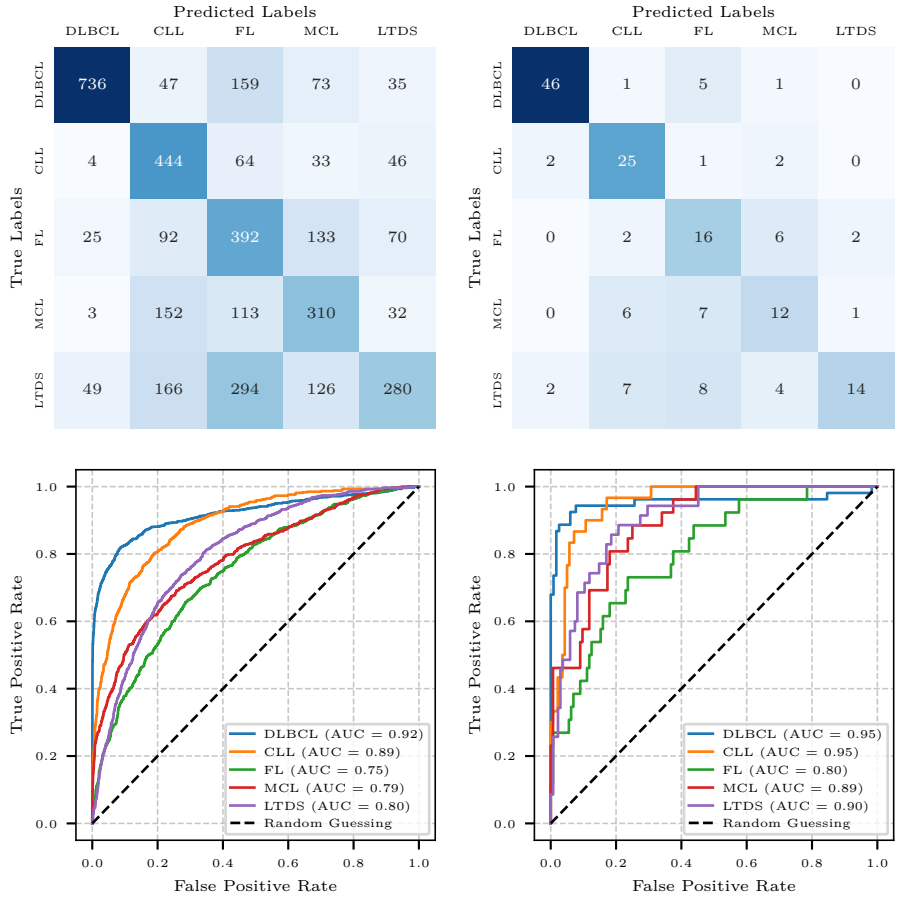



Figure 21: Confusion matrices and ROC curves for patch- and slide-level predictions using a model trained on Munich and evaluated on Kiel data. The top row shows confusion matrices, while the bottom row displays ROC curves for the model ensemble's predictions on both patch (left) and slide (right) levels. The model ensemble was obtained from the cross-validation for the case of training on 1024 µm patches. Note that the Munich data does not contain the HL class.

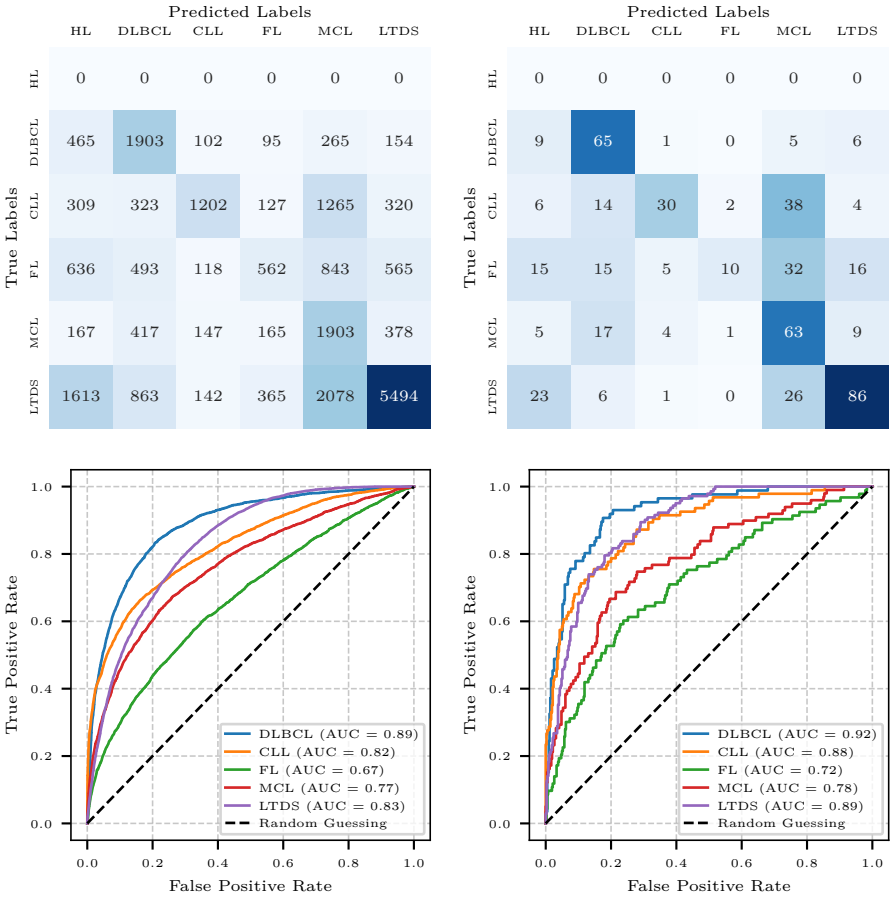



Figure 22: Confusion matrices and ROC curves for patch- and slide-level predictions using a model trained on Kiel and evaluated on Munich data. The top row shows confusion matrices, while the bottom row displays ROC curves for the model ensemble's predictions on both patch (left) and slide (right) levels. The model ensemble was obtained from the cross-validation for the case of training on 1024 µm patches. Note that the Munich data does not contain the HL class.

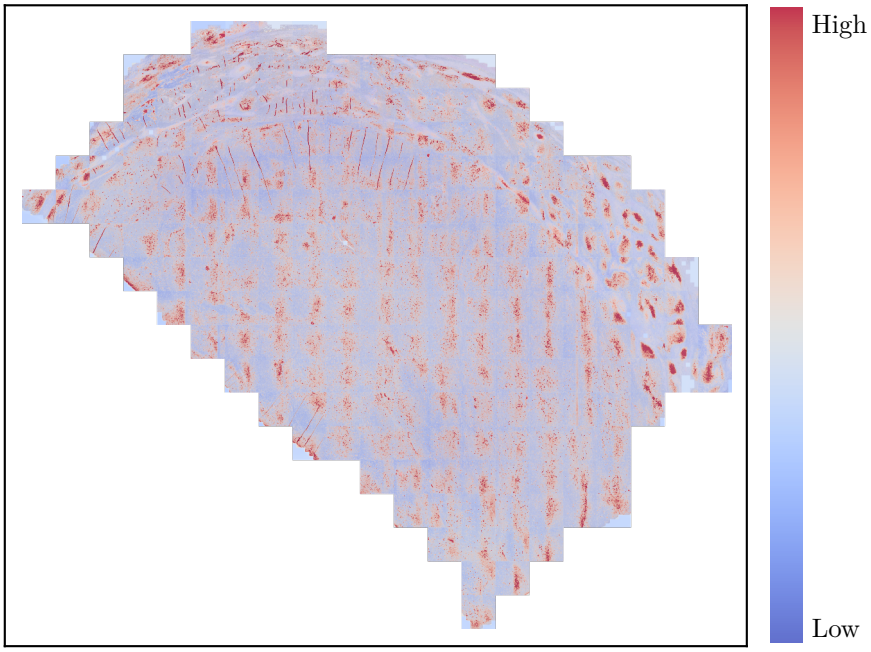



Figure 23: Visualization of a WSI displaying FL composed of 1024 µm patches with the corresponding attention maps for each patch overlaid. This figure is a larger version of the right plot in fig. 18.

## Acknowledgements

I would like to express my deepest gratitude to my supervisors, Andreas Schäfer, Michael Altenbuchinger, and Stefan Schrod, for their invaluable guidance, insightful feedback, and continuous support throughout this research. I am especially grateful to Stefan Schrod for his willingness to take the time for numerous Zoom meetings, carefully answering all my questions, and thoroughly proofreading this thesis. His expertise and encouragement have been instrumental in shaping this thesis.

I also extend my thanks to the joint research group of the 'Federated Learning in Lymphoma Pathology' project for their constructive discussions, helpful suggestions, and collaborative environment.

Special thanks to my dear friends Thomas Sterr and Matthias Holzammer for taking the time to proofread this thesis. I would also like to thank my dear friend Christian Kindler for his general support and his keen eye for all things latex-related.

Finally, I want to thank my family and friends for their support and encouragement, especially during the time of my hand injuries. I am particularly grateful to my roommates, whose patience and constant assistance with everyday tasks that I was unable to perform myself meant more to me than a Subway meal could possibly express. Their support made a challenging time much more manageable, and I deeply appreciate their kindness and generosity.


## Eigentständigkeitserklärung

Ich habe die Arbeit selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und bisher keiner anderen Prüfungsbehörde vorgelegt. Außerdem bestätige ich hiermit, dass die vorgelegten Druckexemplare und die vorgelegte elektronische Version der Arbeit identisch sind, dass ich über wissenschaftlich korrektes Arbeiten und Zitieren aufgeklärt wurde und dass ich von den in §24 Abs. 6 vorgesehenen Rechtsfolgen Kenntnis habe.

Regensburg, 27.03.25

Ort, Datum

Unterschrift