VILNIUS UNIVERSITY

FACULTY OF MATHEMATICS AND INFORMATICS

MODELLING AND DATA ANALYSIS STUDY PROGRAMME

Master's thesis

Multi-task learning for survival analysis using pathology images

Daugialypis mokymasis išgyvenamumo analizėje naudojantis patologijos vaizdais

Ovidijus Kuzminas

Supervisor : assoc. prof. Linas Petkevičius

Scientific advisors : prof. Arvydas Laurinavičius

res. Dovilė Žilėnaitė-Petrulaitienė

sr. res. Allan Rasmusson

Reviewer : assoc. prof. Tadas Žvirblis

Vilnius

2025

Acknowledgements

I am grateful to the National Center of Histopathology for the dataset that was key to this study. Their continuous mentoring and guidance during the course of research has helped this thesis get done.

I'm also so grateful for my family for all of their support and understanding. They were patient and encouraging so I could devote the time and energy required for my courses and research.

Summary

This thesis introduced a multi-task learning framework for survival analysis from a combination of histopathological images and clinical data. The integration of structured tabular data and unstructured whole-slide images (WSIs) improves the prediction of survival probability and time-to-event outcomes. The approach is based on usafe of pre-trained convolutional neural networks for feature extraction, watershed segmentation for preprocessing, parametric tile selection, and a multi-layer perceptron model for multi-task inference. The pipeline achieves a significant improvement in survivability estimation, reaching concordance index of 0.829 on the validation dataset and 0.823 on the test dataset, and outperforming baseline methods like Cox regression which was analyzed for the same dataset in the previous work. This research contributes to the field by presenting an efficient methodology for integrating clinical and histopathological data, proposing framework for image preprocessing and tile selection, and employing customized loss functions for survival analysis tasks.

Keywords: Multi-task learning, survival analysis, histopathology images, feature extraction, deep learning.

Santrauka

Magistro darbe pasiūlyta daugialypio mokymosi sistema, skirta histopatologijos vaizdų ir klinikinių duomenų išgyvenamumo analizei. Struktūrizuotų lentelių duomenų ir nestruktūrizuotų didelės raiškos vaizdų integravimas pagerina išgyvenamumo tikimybės ir laiko iki įvykio kintamųjų prognozavimą. Metodas grindžiamas iš anksto apmokytais konvoliuciniais neuroniniais tinklais, skirtais požymių išskyrimui, *watershed* segmentavimu, skirtu išankstiniam apdorojimui, parametrine vaizdo dalių atranka ir daugiasluoksniu perceptrono modeliu, skirtu daugialypėms prognozėms. Naudojant šią sistemą gerokai pagerinamas išgyvenamumo prognozavimas, kaip rodo 0.829 atitikimo indeksas (angl. *c-index*) validacijos duomenų rinkinyje ir 0.823 - testiniame duomenų rinkinyje, ir tai lenkia Cox regresiją, kuri analizuota to pačio duomenų rinkinio kontekste. Šiuo tyrimu prisidedama srities plėtojimo pateikiant veiksmingą klinikinių ir histopatologinių duomenų integravimo metodiką, pasiūlant vaizdų pirminio apdorojimo ir vaizdo dalių parinkimo sistemą ir taikant naujas nuostolių funkcijas išgyvenamumo analizės užduotims.

Raktiniai žodžiai: daugelio tikslų mokymasis, išgyvenamumo analizė, histopatologijos vaizdai, informatyvių požymių ištraukimas, gilusis mokymasis.

List of Figures

List of Tables

Contents

List of abbreviations

Abbreviation	Full Form
H&E	hematoxylin and eosin
WSI	whole-slide image
DL	deep learning
NN	neural network
CNN	convolutional neural network
MTL	multi-task learning
RSF	random survival forest
c-index	concordance index
ReLU	rectified linear unit
MLP	multi-layer perceptron
ResNet	residual networks
ViT	visual transformer
NLL	negative log likelihood
SGD	stochastic gradient descent
MSE	mean squared error
PCA	principal component analysis
MAE	mean absolute error
R^2	coefficient of determination
TTE	time-to-event

Introduction

Cancer is a major societal, public health, and economic problem in the 21st century, responsible for 16.8% deaths worldwide. Specifically, breast cancer is a leading cause of cancer-related mortality among women globally [4]. underscoring the pressing need for accurate prognostic tools to improve patient outcomes. Current approaches heavily rely on clinical and pathological markers, including tumor size, lymph node involvement, and receptor expression [43]. While the visually scored biomarkers hold significant prognostic power, the majority of prognostic features are still remain unused within the pathology images.

The survival analysis is an essential part of clinical research that provides tools to predict time-to-event data (e.g., disease recurrence or death) [37]. Survival analysis helps in clinical research for assessing treatment efficacy and understanding disease progression. Traditional survival analysis methods such as the Cox proportional hazards model are commonly used. Nevertheless, they have some limitations especially for non-linear relationships, high dimensionality or a large quantities of censored data [33]. Such drawbacks prevent exploiting rich structured and unstructured data which are available for each patient.

Histopathological images are gold standard in cancer diagnosis and prognosis because they provide insights about the tumor and its microenvironment. They include nuclear structure and other tissue components that can be related to tumor advancement, immune reaction, and development [28]. However, the manual assessment of such images is time-intensive and may introduce biases. Last decades advances in deep learning, and more specifically, computer vision techniques have greatly enhanced the process of examining histopathological images [40]. Despite the fact that histopathological images are high dimensional, the general hypothesis is that only a small portion of information inside the image is relevant while making the predictions.

Common tasks where deep learning models are employed in pathology are traditional computer vision tasks: classification (f.e. malignant vs benign), detection and segmentation. Neural networks are proficient in grasping the complex relationships and patterns, which enable to solve quite sophisticated medical problems. Problems that occur with traditional Cox proportional hazards model or manual feature extraction could be mitigated by employing deep learning techniques. However, generic problem that occurs with deep learning is the prediction's inexplicability [39].

Deep learning employment on survival analysis tasks' showed state of the art performance when combination of structured and unstructured is available for records [39]. In the analysis by Wiegrebe et al. customizability of loss functions and dealing with censoring type and ratio are considered advantegous points of deep learning frameworks in survival analysis.

Multi-task learning frameworks represent a transformative approach in survival analysis,

leveraging shared learning across related tasks to optimize generalization capability [42]. These frameworks have the ability to improve the accuracy and robustness of the model since different tasks such as estimation of survival probability and estimation of the time to the event can be trained together and make use of the shared representation. This is especially advantageous when different types of data like imaging and clinical data need to be combined in order to fully represent prognostic factors concerning a patient. Multi-task learning facilitates the need to make accurate predictions but also the need to understand the underlying model which can aid in making better clinical decisions in the future. Since its nature permits the use of less data to achieve more, it is a very effective method for the enhancement of precision medicine practices.

Thesis' aim: Propose a deep-learning based methodology to infer patient's survivability metrics from fusing histopathology image features and tabular data.

Objectives:

- Determine well-performing image preprocessing techniques;
- Review state-of-the-art methods and compare existing pre-trained models and their ability to extract low-level image features for survival analysis;
- Build parametric tile selection framework;
- Create multi-task model for survivability and time-to-event predictions.

Problem statement: There are limited number of researches which would specifically try to extract cumulative information from histopathological images and clinical variables in order to predict patient's survival metrics. Additionally, there are no open-sourced, robust frameworks for preprocessing, feature extraction and modelling of these histopathological imagery for survival analysis.

Structure of the thesis:

- Related work section will cover the researches that are similar in terms of their goals, methods and techniques.
- Methodology section will introduce theoretical side of techniques and methods that were used for this thesis.
- Experiments section will cover the proposed approach for integrating heterogeneous data sources in survival analysis predictions.
- Results and conclusions section will shortly summarize the experiment results.

Results achieved: Multitask learning model developed during this study significantly improved survival estimation and obtained a good concordance index (c-index: 0.829 for validation, 0.823 for test datasets) as well as beating baseline Cox Regression [43]. Major contributions are efficient integration of clinical and histopathological data, tile selection via cosine dissimilarity ranking, and a regularised shallow MLP architecture with customized loss functions for dual-task prediction of survival likelihood and time-to-event.

1 Related work

The area of computational pathology has advanced in the analysis of histopathological images, especially in the segments of detection and segmentation in which deep learning methods are involved. For instance, Ronneberger et al. came up with the UNet architecture [30], a widely used method for the purposes of segmentation in the presence of images such as cellular and tissue structures. Kumar et al. also achieved similar results as they created methods for the separation of cell nuclei in images [24] which prove useful in differentiating the heterogeneity within tumors.

More attention has been directed towards detection of targeted features from the tissue samples. Coudray et al. for example employed the use of Convoliutional neural networks (CNNs) to accurately recognize targeted lung cancer types from whole slide images [7], and their findings were similar to those of experts. The impact of automation in these images spans a broad range, including locating regions of interest and performing image classification tasks. But the main emphasis is more on the establishing the presence of a situation, disease, condition rather than regular assessment of individuals.

As much emphasis is placed on detection and segmentation which seem to be favored research avenues, there is limited work that has focused on survival analysis where prediction of life expectancy is done. The aforementioned features are always incorporated, but are very general to the extent of reliance on manually chosen features or other available data and are inefficient in harvesting the detail in patterns and structures available in WSIs. This gap emphasizes the need for new methods that explore Whole slide images (WSIs) for predictive tasks beyond diagnosis.

1.1 CNNs for Survival Analysis with Histopathological Images

The survival computations using CNNs in histopathology are a less explored domain than their use in classification, detection and segmentation applications. Simulations like DeepConvSurv [19] and Pathomic Fusion [5] have shown that histopathology can be used to model the prediction of survival. DeepConvSurv: CNNs are used to extract features from histopathological images directly which are then used in survival models to compute time-to-event response. Pathomic Fusion extends this by combining imaging data with clinical and genomic information to provide a unified cancer prognostic strategy. These results show the promise of CNNs in survival analysis and highlight that higher-level tile selection and feature extraction is required to achieve best-practice performance. Even though they seem like good ideas, it remains difficult for the industry to come up with generic benchmarks and datasets for survival analysis work. We would need better tests and standardised frameworks to compare and optimise methods for this in future work.

1.2 Tile Selection and Preprocessing in Histopathology

WSIs are normally partitioned into small image tiles for data analysis because of the resolution as presented in Figure 1. It is extremely difficult to choose tiles corresponding to the most effective prediction of survival. The earlier experiments have used heuristics, like tiles with the most tissue

content or tiles in well-defined tumour sites [9]. Recent studies have applied ranking to classify tiles on the basis of feature proximity to regions of interest [27]. Ranking tiles by cosine dissimilarity and pooling features using percentile-based selection offer powerful tool for noise reduction and identifying diagnostically relevant areas to coincide with these advances. Also, pre-processing methods such as color normalization and stain deconvolution can help to improve the extracted features due to minimizing variance among slides. These techniques are useful to build scalable and generalized pipelines for datasets.

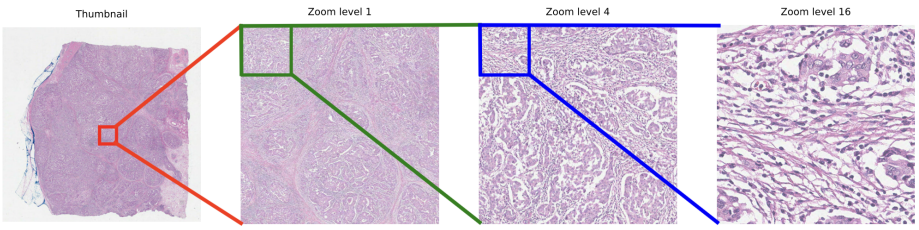


Figure 1. *Different WSI zoom levels*

1.3 Multi-Task Learning for Survival Analysis

MTL models have emerged in survival analyses because they can take advantage of common representations across tasks. For example, Zhu et al. had suggested an intertask survival model that simultaneously predicted survival probabilities and recurrence risks [11], making it more robust by using additional information from other tasks. Integration of prediction of survival probability with time-to-event can increase the interpretation and predictive power, especially when we combine features from various data sources like histological images and clinical parameters. This method allows the model to deal with other aspects of survival analysis, including the difference between long-term survivors and patients with advanced disease. Furthermore, MTL models innately prevent overfitting by training on common, task-neutral models. These are therefore best suited for limited datasets or very large inputs.

1.4 Feature Extraction with Pre-Trained CNNs

Pre-trained CNNs are capable of extracting relevant patterns employing stacked convolutional layers from the imagery as presented in the Figure 2. Moreover, CNNs have already been widely used to feature extract histopathological images for their efficacy and universality. Architectures like ResNet [17], DenseNet [18], EfficientNet [35] have proved effective for low-level and mid-level features. These features are then usually further enriched for survival analysis through pooling or tile selection methods to keep the prognostic data preserved. These approaches and new strategies for prioritizing diagnostically useful features include ranking and tiles selection methods on cosine dissimilarity. The pre-trained models saves the compute cost and time required to train networks on

a new set of data and utilizes rich feature representations learned on large-scale data. This transfer learning model has been successfully applied to general purpose capabilities in task specific applications, including survival analysis in histopathology. In future research, specialized pre-training could be added to further increase model performance and robustness.

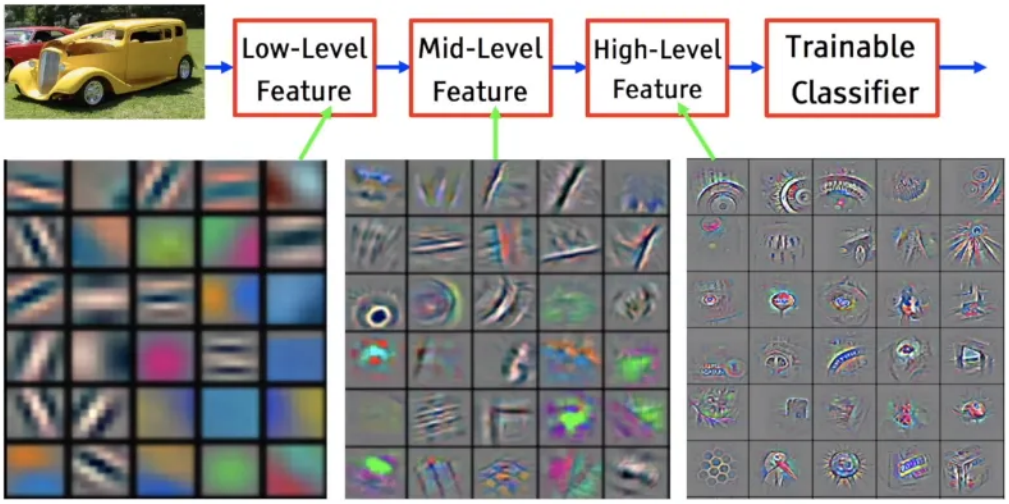


Figure 2. *CNN feature extraction visualization [38].*

1.5 Combining Imaging and Tabular Data for Prognostic Modeling

A combination of structured tabular data and unstructured image data is crucial to build robust survival models. For example, Mobadersany et al. [27] have shown that merging histopathological and clinical variables together substantially enhances the prediction of survival over either data format. Multi-modal neural networks or feature fusions across multiple inputs can help models detect complementary data. By integrating imaging and clinical data, the model incorporates macroscopic tissue structure and context variables of the patient, which enhances an overall picture of disease progression. Attention mechanisms or feature fusion can also highlight the most important information of each modality. These techniques make prognostic models both more precise and more useful for clinical decision making.

1.6 Custom Loss Functions for Survival Analysis

Loss functions in DL are not adapted to the specific problems of survival analysis, including censored data and unbalanced events. Custom loss functions, such as negative log-likelihood (NLL) for survival odds and mean squared error (MSE) for time-to-event prediction, have been used in the past [1]. Moreover, loss functions defined on the concordance index have also been reported to improve survival predictions ranking power [31]. If weights are added to loss function, it can further improve the class balancing by giving greater weight to low-representation events. These customised loss functions optimize the generalisation capabilities of the model, especially for datasets with large numbers of censored observations. Adaptive loss functions, with weights being changed dynamically according to model performance or distribution, may be a promising area of future work.

2 Methodology

2.1 Histopathology and Its Digital Transformation

Histopathology is the microscopical study of tissues or cells to diagnose and evaluate disease [25]. Histopathologists usually diagnose using biopsies – patches of skin, for example, or tissues from the liver or kidney – to pinpoint lesions, describe states of disease, and help clinicians care for patients. Particularly, for cancer diagnosis, histopathologists study the cell structures and tissue organisation to identify tumor type, severity and treatment response. The hematoxylin and eosin (H&E) stain is still the standard way to visualize tissue anatomy and help pathologists detect the most significant structures and pathologies that can make or break diagnosis.

The discipline has entered the midst of a huge digital transformation in recent years, what's sometimes called digital pathology. High-resolution whole slide imaging (WSI) lets pathologists transform glass slides into digital files to store, share and examine histological samples remotely. More than just practical ease of use, digital pathology opened new avenues for computational research. Combining AI and ML techniques, digital pathology systems can detect micromorphological changes quickly, measure biomarkers, and assist in complex diagnoses to make diagnosis more accurate, efficient and reproducible. This combination also supports large-scale, data-driven research programmes that lead to personalized medicine and the standardization of diagnostic tests [22, 29].

In all, integrating traditional histopathology and modern digital workflows enhances diagnostics, international partnerships and research pipelines. As histopathology takes shape in the digital age, it will become the foundation of modern pathology, for patients, clinicians and researchers alike.

2.2 Survival analysis

Survival analysis is a set of statistical methods for examining time-to-event outcomes — the timing and likelihood of death, disease relapse, or other endpoints of interest. Survival models have to contend with the unique issues that censored data presents, as well as time-varying risk dynamics, in contrast to regression techniques. These approaches are used in medicine, engineering, economics and other fields for a variety of purposes, providing a glimpse of how certain covariates affect survival and event probability over time [23, 36].

2.2.1 Censoring

The challenge in survival data is censored observation. Censorship arises if the time of the event cannot be known—either because the participant had not yet witnessed it by the end of the study, or because they were lost to follow-up. While censoring could take many forms (left-side, two-side), in this study only the right-side censoring will be discussed.

Suppose we observe n individuals. For the i-th individual, let T_i denote the event time, $\delta_i \in \{0,1\}$ indicate whether the event was observed or censored, and X_i represent predictor variables:

$$\delta_i = \begin{cases} 1 & \text{if the event is observed,} \\ 0 & \text{if the event is censored.} \end{cases}$$

The observed data can be represented as triplet:

$$\left\{ \left(Y_i, \delta_i, X_i \right) \right\}_{i=1}^{n},$$

where

$$Y_i = \min(T_i, C_i),$$

and $C_i > 0$ is the (strictly positive) censoring time.

2.2.2 Cox Regression

The Cox proportional hazards model is a semi-parametric model commonly used to assess the relationship between covariates and the hazard function [8]. Let $h(t|\vec{X})$ be the conditional hazard function given covariate vector $\vec{X} = (X_1, \ldots, X_p)^T$. The Cox model assumes:

$$h(t|\vec{X}) = h_0(t) \exp(\beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p),$$

where $h_0(t)$ is the baseline hazard function and β_j are regression coefficients estimating the log-risk increase associated with covariates X_j, for $j \in \{1, 2, \ldots, p\}$, where p represents the number of covariates. The estimation of $\vec{\beta}$ leverages partial likelihood methods, avoiding assumptions about the baseline hazard [23, 36].

2.2.3 Random Survival Forest

Random survival forests (RSF) are time-to-event variants of the random forest algorithm. Such approaches develop a set of survival trees each of which divides the feature space into leaves similar to the conditional survival function. The resulting prediction is often an estimated survival rate of all the trees in the forest. RSFs can account for high-dimensional interactions and nonlinearities without the limiting assumptions of parametric or semi-parametric models. They provide general non-linear modeling and robust performance across a broad range of scenarios [20].

2.2.4 Hazard Functions

$h(t)$ is the hazard function which describes how fast things are happening at time t, if nothing has happened before t. Formally:

$$h(t) = \lim_{\Delta t \to 0} \frac{P(t \leq T < t + \Delta t \mid T \geq t)}{\Delta t}.$$

This function can also tell you how risk evolves over time. These have the survival function $S(t) = P(T > t)$ that can be estimated with the Kaplan-Meier estimator or by modeling and the cumulative

hazard function $H(t)$, defined as:

$$H(t) = \int_0^t h(u)\, du.$$

2.2.5 Concordance Index

Concordance index (c-index): This indicates the prediction quality of a survival model with respect to the order of time of events. It scores how well the model can order subjects by risk correctly. For two people i and j, c-index would add the predicted risks/predicted survival times to the observed event times to see if the model is always placing greater risk on the person with the earlier event. It is between 0.5 (not much better than guessing) and 1.0 (absolute correct prediction) [14].

2.2.6 Brier Score

The Brier score is a standard scoring rule for probabilistic predictions. In terms of survival analysis, it could be stated at a time t like this::

$$f_{BS}(t) = \frac{1}{n} \sum_{i=1}^{n} \Big(I(T_i > t) - \hat{S}(t|X_i) \Big)^2,$$

where I is an indicator function and $\hat{S}(t|X_i)$ is estimated survival probability of i at time t. Lower Brier scores mean higher predictive accuracy [13].

2.3 Watershed segmentation

Watershed segmentation is the idea of image brightness as topographical elevations, where bright pixels are peaks and dark pixels are valleys. With water conjured up from lower intensities in mind, there naturally emerge catchment basins. Where these basins would meet, watershed lines appear, which help cut the image into useful chunks as presented in Figure 3. This is often used in medical images and is often used along with gradient-based filtering and morphological operations to reduce over-segmentation and increase the quality of partitions [12].

A marker-based version of the watershed transform streamlines this process with pre-generated seeds for buildings or sights of interest. These seeds control partitioning and keep the end result segmentation more inline with feature in the image. By using markers at the correct placement and pre-processing with smoothing or gradient processing, the algorithm will have better and stable partitions.

Image pre-processing like gradient filtering help to sharpen object boundaries, and morphological transformations smooth intensity changes resulting in more consistent and reliable segmentation results. For medical imaging, these advances make sure that the most important information – tumor margins or organ structures – is being segmented more consistently.

The pseudocode 1 algorithm shows a very reduced implementation of the watershed algorithm. It starts by computing gradient magnitude image with discontinuity of intensity and sort pixels on gradient value. Each pixel is then allocated a basin if it matches one specific basin, or to the

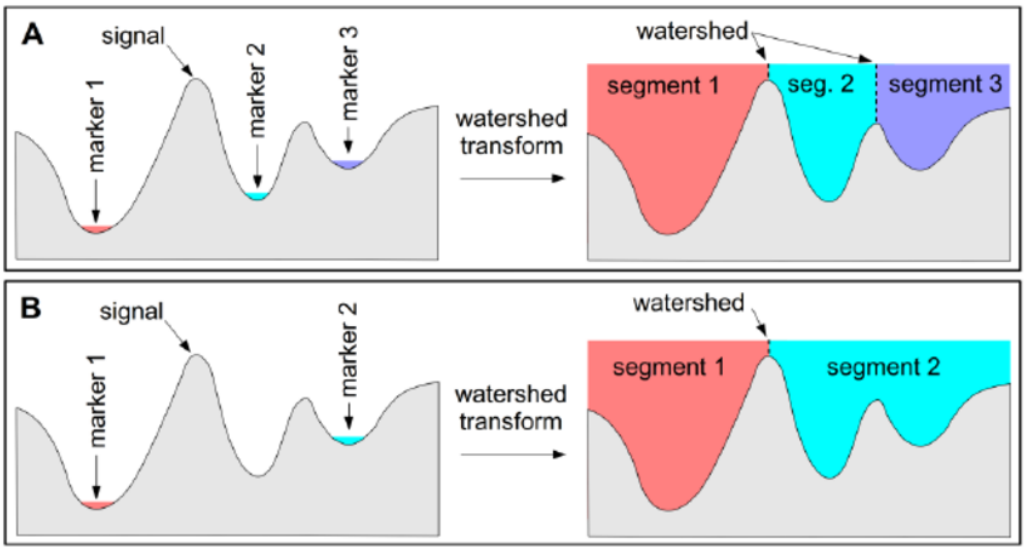


Figure 3. *Watershed segmentation visualization [10].*

watershed boundary if there are multiple basins that work equally well. This way we can ensure the final segmentation makes use of topographic information and pre-processing, and create partitions that can then be used for an analysis, diagnostic or decision making step.

1 algorithm Watershed Segmentation

```
# Inputs:
Input: I – input image
Input: M – initial markers (basins)
# Pre-processing: Compute gradient magnitude image G
G := GradientMagnitude(I)
Sort all pixels in G in ascending order by their gradient value
Initialize each marker in M as a separate basin label
for each pixel p in sorted order do
    Let N(p) be the neighbors of p that have assigned basins
    if N(p) contains pixels from exactly one basin then
        Assign p to that basin
    else if N(p) contains pixels from multiple distinct basins then
        Mark p as a watershed boundary pixel
    end if
end for
# Output:
Output: Final segmentation defined by basins and watershed boundaries
```

2.4 Deep Learning

Deep learning is the application of learning higher levels of representations through neural networks directly from raw data. Such systems identify complex patterns without manually composing features, and thus are suitable for image classification, segmentation and feature extraction. For the analysis of three-dimensional histopathological photographs, deep neural networks provide a robust way to image morphological and structural changes in tissue slides. With the incorpora-

tion of learned representations into the feature extraction, selection, and inference pipelines we can improve the performance and powerful generalization [41].

2.4.1 Activation Functions

Activation functions introduce nonlinearity to neural networks so that they approximate complex processes. If nonlinear activations weren't present, the result of layering several linear layers is simply to create a linear mapping. Some of the popular activation functions are the sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

and the hyperbolic tangent (tanh). But perhaps the most commonly used activation in deep learning today is the Rectified Linear Unit (ReLU):

$$\mathsf{ReLU}(x) = \mathsf{max}(0, x).$$

The ReLU doesn't get saturated with positive inputs, it is nonlinear, and it is also very computationally easy. Its widespread deployment has drastically increased training stability and performance on large image processing problems [41].

2.4.2 Multi-layer Perceptron

Multi-layer perceptron (MLP) is one of the most elementary neural network designs. It consists of one or more fully connected (dense) layers that are each translating an input vector into an output vector with a learned linear transformation and a nonlinear activation. Given an input vector $\mathbf{x} \in \mathbb{R}^d$, a single MLP layer can be expressed as:

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}),$$

where **W** and **b** are learnable parameters, and $\sigma(\cdot)$ is the chosen activation function. MLP, when aggregated with several of these layers, can model high-level functions and can be a malleable piece of an architecture. MLPs are typically layered over feature extractors (e.g., convolutional backbones) to do classification, regression or any downstream operation. [41].

2.4.3 Dropout

Dropout is regularization method that allows to minimize overfitting by randomly eliminating some neurons while training. To have the probability p of dropout, every neuron is zeroed with probability p:

$$\tilde{\mathbf{h}} = \mathbf{h} \odot \boldsymbol{\epsilon}, \quad \epsilon_i \sim \mathsf{Bernoulli}(1 - p),$$

where $\odot$ denotes element-wise multiplication. Training on such stochastically thin networks trains the model with stronger generalizations. This method reduces the ability of deep networks to learn training data trends and makes them better at absorbing new, invisible inputs [41].

2.4.4 Convolutional Neural Networks

Convolutional neural networks (CNNs) take advantage of the spatial representation of the images by convolutional layers using local, shared-weight filters. This dramatically decreases the number of input parameters and yields translation-invariant visual representations.

Convolution Layers: A 2D convolution operation over an input image **X** with a kernel **K** of size $M \times N$ is given by:

$$Y_{i,j} = \sum_{m=-\left\lfloor \frac{M}{2} \right\rfloor}^{\left\lfloor \frac{M}{2} \right\rfloor} \sum_{n=-\left\lfloor \frac{N}{2} \right\rfloor}^{\left\lfloor \frac{N}{2} \right\rfloor} X_{i+m,\, j+n}\, K_{m,n}.$$

This operation learns local spatial features (edges, textures, etc) and then iterates over the image to glean a hierarchy of features.

Pooling Layers: Pooling layers make feature maps spatially smaller, resulting in greater spatial consistency and reduced computation. Max pooling finds the maximum within a single area:

$$Y_{ij} = \max_{(u,v) \in R_{ij}} X_{u,v},$$

while average pooling takes the average. This downsampling aims the network towards the best features and gives a kind of translation invariance [41].

2.4.5 Transfer Learning

Transfer learning uses the information learned from a pretrained model (once trained on a big, rich dataset) to start training on a new task or dataset. Transfer learning does not initialize all parameters at random, but instead from model weights that already encode general-purpose features. This often means a decrease in labeled data and training time to perform competitively. Formally, let $\vec{\theta}_{\text{pre}}$ denote the pretrained parameters. The transfer learning process initializes the new model parameters $\vec{\theta}_0$ as:

$$\vec{\theta}_0 \leftarrow \vec{\theta}_{\text{pre}},$$

and then task-driven training works on the parameters in a new dataset. It is especially useful for domains with limited annotations (eg, in medical image applications) and allows models to skip over having to learn the low-level patterns (edges, textures) from the scratch [41].

2.4.6 Residual networks Architecture

Residual networks (ResNet) solves the issue of training very deep neural networks. As you increase the network depth gradient disappear and accuracy of training diminishes. ResNet adds skip connected residual blocks:

$$\mathbf{y} = F(\mathbf{x}) + \mathbf{x},$$

where **x** is the input and $F(\mathbf{x})$ is a sequence of convolutional and nonlinear layers. By letting gradients run through the skip links, deeper networks are stable and easier to train. ResNet architectures are

now the default platforms for many image tasks to extract feature from large and intricate image data sets efficiently. [41].

2.4.7 ViT Architecture

Vision Transformer (ViT) system applies self-attention to image patches instead of convolutions. The picture is separated into patches, linearly projected and fed to a Transformer encoder. The self-attention learns dependences between these patches so that the model detects global relationships in the image. Given queries Q, keys K, and values V:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V,$$

where d is the embedding dimension. Performing these operations again and again gives ViT the complete representations without the locality of convolution. This can be especially helpful in ad-hoc imaging fields, where the features of interest might be spread over many large pictures [41].

2.4.8 Loss Functions

Loss functions are the variance between output predictions and ground-truth labels, and they guide optimization. The cross-entropy loss is often used in classification. Letting y_i denote the true class indicator and $\hat{y}_i$ the predicted probability:

$$\mathcal{L}_{CE} = -\sum_i y_i \log(\hat{y}_i).$$

For regression tasks, the mean squared error (MSE) is a standard choice:

$$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2.$$

The other popular loss is Negative log-likelihood loss (NLLLoss) used with the log-softmax output layer for classification. Only for a single data point that actually is of class. For a single data point with true class c:

$$\mathcal{L}_{NLL} = -\log(\hat{y}_c),$$

where $\hat{y}_c$ is the likelihood of having the right class. NLLLoss directly guides the model to allocate a high probability for the correct category and is used for multiclass classification issues [41].

2.4.9 AdamW optimizer

Optimizers are algorithms that would update network parameters according to the gradients calculated and reduce the loss function chosen. Stochastic gradient descent (SGD) has been a default, but more advanced optimizers can vary the learning rate or update of parameters to accelerate convergence or generalization. One of those optimizers is AdamW which is popular due to its stability

and performance. It is an Adam optimizer version with a decoupled weight decay term which keeps weight decay free from gradient updates. Formally, let $\vec{\theta}$ denote parameters, α the learning rate, and λ the weight decay factor. AdamW updates parameters as:

$$\vec{\theta}_{t+1} = \vec{\theta}_t \; - \; \alpha_t \, \hat{g}_t(\vec{\theta}_t) \; - \; \alpha_t \, \lambda \, \vec{\theta}_t,$$

where $\hat{g}(\vec{\theta})$ is gradient update (adapted by the Adam algorithm with per-parameter adaptive learning rate) and (alpha lambda theta) direct L2 regularization. AdamW abstracts the weight decay from gradient update parameter synthesis for better generalization, and is typically used in large training data [41].

2.4.10 Multi-task Learning

Multi-task learning (MTL) is the strategy to generalize a model through co-training on different task pairs. Rather than tuning model parameters for one purpose, MTL is learning shared representations to support several purposes at once. Formally, let $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ represent a set of k tasks, each with its own loss function $\mathcal{L}_i$. The overall objective in a multi-task setting can be expressed as a weighted sum of individual task losses:

$$\mathcal{L}_{\text{multi-task}}(\theta) = \sum_{i=1}^{k} \lambda_i \, \mathcal{L}_i\big(y_i, \hat{y}_i(\theta)\big),$$

where λ_i are weights that balance the importance of each task, and θ denotes the model parameters. If tasks are shared across model components (e.g., early layer feature extractors), then the network can draw upon relationships and shared hierarchies found in the data. This will usually be a more effective, data-efficient and less overfitting model training method than training separate models for each task separately. Multitask learning has also been used in computer vision, natural language processing, and healthcare, where many related predictions (e.g., classification, segmentation, regression) can be processed together to gain benefits from the synergies and boost predictive power [41].

2.4.11 Imbalance handling

To compensate for an imbalanced dataset, a popular deep learning approach is to use class-dependent weights to correct for the over-representation of some classes. Cost-sensitive learning translates this to updating the loss function and adding higher weights to underrepresented classes so their associated errors exert a more significant impact on parameter changes. For instance, in a weighted cross-entropy loss:

$$\mathcal{L}_{WCE} = -\sum_{i=1}^{N} w_{y_i} \log(\hat{y}_i),$$

where y_i denotes the true class label for sample i, $\hat{y}_i$ is the predicted probability for the correct class, and w_{y_i} is the weight assigned to class y_i. By increasing w_{y_i} for underrepresented classes, the network

can learn more about them and less often make majority-class predictions. This weighting algorithm can be useful in medical imaging, anomaly detection or anywhere where minority class examples are important. Researchers have found that class-weighted training can greatly increase the classifier's performance on minority classes without significantly diminishing its accuracy [16].

2.5 Data splitting

The usual method in deep learning is to divide the dataset into three subsets: a training set, a validation set, and a test set. In medical contexts, each data point (i.e., each row or record) typically represents a single patient. The model is trained on the training set to learn the patterns in the data by fine-tuning its parameters. A validation set, shaved off from the training set, is used to tune hyperparameters and apply early stopping, ensuring that improvements are not merely due to overfitting on the training examples. Finally, the test set—completely unseen during training and validation—serves to provide a clear, unbiased evaluation of the model's generalization performance. This three-way split is particularly valuable in high-complexity domains like medical imaging or image classification, where overfitting is a major concern and rigorous generalization assessment is critical. Additional techniques such as balanced splits, class-proportion stratification, or domain-specific variables can further enhance the credibility of the analysis. Many deep learning studies regard these splitting protocols as foundational to experimental design, ensuring reproducibility and sound data management [2].

2.6 Min-max feature scaling

Min-Max scaling is a normalization method that transforms tabular features into a specified range, most often [0, 1]. For a given feature X with minimum X_{min} and maximum X_{max}, the transformed value X' is given by:

$$X' = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}.$$

It is useful in pipelines for deep learning to deal with tabular input that span large disparate scales or containing huge values. By smoothing out all features into a fixed range, Min-Max scaling allows for better numerical stability, faster convergence, and may also make gradient optimizers more efficient. It is commonly found in open-source packages such as scikit-learn that have a simple and documented Min-Max scaling implementation to fit it into pre-processing functions for deep neural network training [26].

2.7 Cosine dissimilarity

Cosine dissimilarity (or cosine distance) comes from cosine similarity, which is often used to determine how much 2 vectors are aligned in a high dimension space. If we are given two vectors **u** and **v**, then the cosine similarity is:

$$\text{CosineSimilarity}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}.$$

Cosine dissimilarity then can be expressed as:

$$\text{CosineDissimilarity}(\mathbf{u}, \mathbf{v}) = 1 - \text{CosineSimilarity}(\mathbf{u}, \mathbf{v}).$$

This is a function that takes into account the vectors' orientation rather than their size, which is why it is resistant to scale difference. Cosine dissimilarity is typically used in deep learning to compare representation vectors or embeddings because it highlights directionality in feature space [34].

2.8 Bayesian hyperparameter optimization

Bayesian hyperparameter optimization is a model-based search algorithm for finding optimal hyperparameter settings for machine learning models. In contrast to manual tuning or grid searches, Bayesian approaches build a probabilistic substitute model over the objective function based on observations. Formally:

$$p(f \mid \mathcal{D}),$$

where $\mathcal{D} = \{(\theta_i, f(\theta_i))\}$ denotes the set of previously evaluated hyperparameter configurations and their outcomes. A common choice is to assume a Gaussian Process prior over f, enabling the model to represent uncertainty about unexplored regions of the hyperparameter space Θ. At each iteration, an acquisition function $\alpha(\theta \mid p)$ is employed to select the next hyperparameter configuration to evaluate:

$$\theta_{\text{next}} = \arg\max_{\theta \in \Theta} \alpha(\theta \mid p).$$

Through iterative refinement of the surrogate model, and by conflating exploration with exploitation in the acquisition function, Bayesian optimization operates more fluidly in large and heterogeneous hyperparameter environments than simple methods do. Such a strategy is demonstrated to enhance the performance of machine learning models, while cutting the computational costs of running large hyperparameter searches [32].

2.9 Principal component analysis

Principal component analysis (PCA) is a dimension reduction method popular for identifying the structure in large data sets by transforming it to a smaller subspace. The algorithm converts a vector of possibly related variables into a vector of linearly unrelated components, or principal components. Formally, given a zero-mean data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, PCA computes the covariance matrix:

$$\mathbf{C} = \frac{1}{n-1} \mathbf{X}^\top \mathbf{X},$$

where $\mathbf{C} \in \mathbb{R}^{d \times d}$.

Next, an eigen-decomposition of $\mathbf{C}$ is performed:

$$\mathbf{C}\mathbf{V} = \mathbf{V}\mathbf{\Lambda},$$

where:

- $\mathbf{V} \in \mathbb{R}^{d \times d}$ is the matrix whose i-th column, $\mathbf{v}_i$, is an eigenvector (principal direction) of **C**.
- $\mathbf{\Lambda} \in \mathbb{R}^{d \times d}$ is the diagonal matrix of corresponding eigenvalues λ_i, often sorted in descending order $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$.

The first few principal components explain most of the variance of the data. These can be combined to get away with dimensional reduction without losing important information. PCA is generally better at generating training machines with fewer failures as it helps reduce the curse of dimensionality and generalize when data dimensions are large (small compared to sample size) [21].

2.10 Regression metrics

Regression metrics are statistical tools to measure predictive and generalization accuracy of regression models. Common metrics are the mean squared error (MSE), mean absolute error (MAE), and the coefficient of determination (R^2). Given true targets $\{y_i\}_{i=1}^N$ and predictions $\{\hat{y}_i\}_{i=1}^N$, the MSE is defined as:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2,$$

while the MAE measures the average magnitude of errors without considering their direction:

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|.$$

The coefficient of determination R^2 compares the explained variance of the model to the total variance in the data:

$$R^2 = 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2},$$

where $\bar{y}$ is the mean of the actual targets. The higher the R^2 score, the better the fit, the lower the MSE and MAE value indicates closer predictions to the true value. These metrics are known and popular, appearing in textbooks on statistical learning and freely available in open-source libraries to evaluate models [15].

3 Experiments

3.1 Dataset

The dataset includes WSIs (whole-slide histopathological images) and tabular information based on clinical and morphological measurements. There are 252 records and 58 columns that incorporate categorical and numerical variables. Every entry is associated with a patient and includes demographic details (age, age cohort), tumor-specific data (stage, grade, subtype), and biomarker metrics (ER, PR, HER2, Ki67). It contains extracted texture features and clinical results to allow prognosis for the patients survival analysis. Key attributes include:

- **Clinical Information:** Age, tumor stage (T), nodal status (N), tumor grade (G).
- **Biomarkers:** Hormone receptor expressions (ER, PR), Ki67 proliferation index, and HER2 status (HER2_IHC and HER2_01).
- **Texture Features:** parameters such as contrast, homogeneity, entropy, dissimilarity, and energy, especially relevant for Ki67 entropy and AshD for assessing heterogeneity.
- **Outcome Variables:** Disease-specific survival (DSS, DSS_10) and overall survival (KV_2023).

The tabular features were analyzed to summarize central tendencies and variability. The descriptive statistics are provided in Table 1.

Table 1. *Descriptive Statistics for Numerical Features*

Feature	Count	Mean	Std Dev	Min	25%	50%	Max
Age	252	60.72	12.52	36.0	50.0	62.0	88.0
DSS	252	0.15	0.35	0.0	0.0	0.0	1.0
Ki67_pat	252	21.89	15.27	3.0	10.0	19.5	95.0
ER_pat	252	90.63	16.91	10.0	90.0	100.0	100.0
PR_pat	252	60.98	37.92	0.0	25.0	80.0	100.0
HER2_IHC	252	0.94	0.49	0.0	1.0	1.0	2.0
T	252	1.42	0.50	1.0	1.0	1.0	2.0
N	252	0.54	0.74	0.0	0.0	0.0	3.0
G	252	2.10	0.65	1.0	2.0	2.0	3.0

There are no missing values in the main clinical features, but a few secondary biomarker scores (e.g., HER2 FISH) are sparse. The variables also have diverse distributions — some are binary or ordinal [43].

The tabular data are very heavily censored (87%) and many of the entries do not include full survival data, so the validity of some of these analyses is compromised. Also, since the dataset contains only 252 samples, the dataset will probably not be big enough to train a deep learning model without overfitting.

Dataset also holds 252 `svs` format files. Each patient could be matched to a single slide by a unique identifier appearing in the file name. The image dimensions at the base resolution vary, with the largest being `(31872, 30252)` pixels. The pyramid representations of the dataset are multi-resolution and at their lowest resolution, can be downsampled to `(1992, 1890)` pixels.

There are aspect ratios ranging from 0.47 to 1.86, and the majority of slides are near 1 in aspect ratio meaning near-square. The pixel size in microns (MPP) is consistent across the slides with a median of 0.5034 microns for horizontal (MPP X) and vertical (MPP Y) axes. This homogeneity means the physical scaling of features is constant in images. Pyramid plots of the dataset is derived from standard downsampling ratios `1.0`, `4.0`, and `16.0` that allow to analyze the data at various levels of detail. Key insights:

- The dataset contains base-resolution images with size like `(31872, 30252)` pixels.
- Average aspect ratio 0.9757: the majority of slides are nearly square, but there are even longer slides with aspect ratios up to 1.86.
- The MPP variation is very small (0.2501 to 0.5038 microns) which makes the spatial scaling fairly consistent most of the time.

To provide an example of the dataset, Figure 4. showcases five random whole-slide images from the dataset.

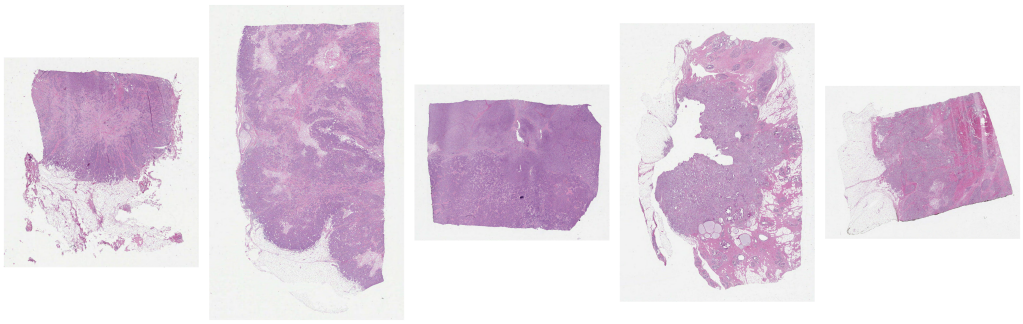


Figure 4. *Five randomly selected whole-slide images from the dataset.*

Some of the feature information has already been extracted from the images and mapped to the tabular data, such as texture parameters (contrast, entropy, energy) and biomarker expressions (ER, PR, Ki67, HER2). Such features quantitatively distill complicated patterns of image, offering clues about heterogeneity and other morphological properties.

3.2 Data Splitting

Data splitting should be performed properly to make the experimental outcomes stable and reliable. The data is split into 3 partitions: 151 being assigned to training (60%), 51 validation (20%), and 50 testing (20%). Data is stratified during split in order to represent the `DSS_10` feature equally in all subsets. This will make sure the censoring is stable across different partitions. The data splitting is done using `train_test_split` function from scikit-learn package in Python.

Only the final model will be tested against the test data for non-biased performance measurement. By contrast, train and validation datasets will be used in all the experiments – comparison of feature extractors, segmentation algorithms, how to choose the most discriminating tiles, training the neural network, and adjusting its hyperparameters. It allows a consistent, reproducible process for model creation and test.

3.3 Feature Extractor Comparison

Four pre-trained neural networks models as features extractors are evaluated in this experiment step: three TIA Toolbox ResNet architecture-based models and MahmoodLab's UNI model.

Pretrained ResNet models (varying in depth) are available from the TIA Toolbox Python library. These models are trained on histopathology datasets such as Kather100K, PCam and others. They are used for patch classification in computational pathology, to recognise tissues and pathological features on histological images [3].

The UNI model from MahmoodLab is a general purpose self-supervised vision encoder for pathology based on ViT architecture. It was pretrained on more than 100 million images from more than 100,000 WSIs. UNI showed state-of-the-art performance on 34 clinical tasks such as ROI classification, slide classification and feature extraction from histopathology images [6].

Evaluating feature extractors for histopathological images uses training and validation datasets. For each WSI, 100 random tiles are selected excluding those with a high proportion of white color to eliminate non-informative regions. Tile loading is accomplished using `WSIReader` from tiatoolbox Python package. Features are extracted and aggregated into WSI feature vectors with `max` operation across feature dimensions. All feature extractors are loaded using timm library in Python and changing their final layer into `Identity` layer from PyTorch. PCA (scikit-learn implementation) reduces feature dimensionality. These reduced features, along with survival times and censoring statuses, train a RSF (class `RandomSurvivalForest` from sksurv Python package) with default hyperparameters. The validation WSIs undergo the same processing, and the trained model is evaluated on the validation set using the c-index. The following pseudocode outlines this evaluation process:

2 algorithm Feature Extractor Comparison

Inputs:
$\mathcal{M}$: Feature extractor models
$\mathcal{W}_{\text{train}}$, $\mathcal{W}_{\text{val}}$: Training and validation WSIs
$\mathcal{S}_{\text{train}}$, $\mathcal{S}_{\text{val}}$: Survival times
$\mathcal{C}_{\text{train}}$, $\mathcal{C}_{\text{val}}$: Censoring statuses
# Procedure:
for each model m in $\mathcal{M}$ **do**
    Extract and aggregate features from $\mathcal{W}_{\text{train}}$
    Apply PCA to obtain $\mathcal{F}_{\text{train\_reduced}}$
    Train survival model on $\mathcal{D}_{\text{train}} = (\mathcal{F}_{\text{train\_reduced}}, \mathcal{S}_{\text{train}}, \mathcal{C}_{\text{train}})$
    Extract and aggregate features from $\mathcal{W}_{\text{val}}$
    Apply PCA to obtain $\mathcal{F}_{\text{val\_reduced}}$
    Form $\mathcal{D}_{\text{val}} = (\mathcal{F}_{\text{val\_reduced}}, \mathcal{S}_{\text{val}}, \mathcal{C}_{\text{val}})$
    Evaluate model on $\mathcal{D}_{\text{val}}$ using C-index
    Store performance metric for m
end for
# Output:
Performance metrics for all models

The performance of feature extractors was evaluated using the c-index on the validation dataset. Table 2. presents the results, highlighting the best-performing model.

Table 2. *Performance Comparison of Feature Extractors*

Model	c-index
TiaToolbox ResNet18	**0.721**
TiaToolbox ResNet34	0.711
TiaToolbox ResNet50	0.690
MahmoodLab UNI	0.715

From Table 2., it is evident that TiaToolbox ResNet18 achieved the highest Concordance Index of **0.721**, indicating higher generalization in predicting survival outcomes compared to the other models. Also, it produces significantly smaller features' vector $\mathbf{v} \in \mathbb{R}^{512}$ in comparison to MahmoodLab UNI where $\mathbf{v} \in \mathbb{R}^{1024}$.

3.4 Segmentation Comparison

This section compares four WSI background segmentation algorithms: Otsu thresholding (based on global intensity histograms), Adaptive thresholding (based on local intensity statistics), Region growing (coupling local pixels with similar appearance), and Watershed segmentation (discussed in detail in methodology section). All the algorithms were employed using `skimage` Python library. After segmentation, the masks were created on tile level information and the patches with mostly

background parts are removed. Using max pooling operation the remaining patches are aggregated into single vector $\mathbf{v} \in \mathbb{R}^{512}$ per WSI, PCA truncates features dimensions and a RSF is being trained to predict survival of the patient (once tiles have been segmented logic follows the same pattern as in feature extractor comparison part). c-index of a validation dataset is used as the performance parameter.

Table 3. displays the c-index scores of each segmentation algorithm. The highest c-index was for Watershed segmentation, demonstrating that if tissue boundaries are better defined, feature extraction will be more appropriate, and survival metrics will be more accurately predicted.

Table 3. *Performance Comparison of Segmentation Techniques*

Segmentation technique	c-index
Otsu thresholding	0.715
Adaptive thresholding	0.717
Region growing	0.703
Watershed segmentation	**0.729**

Because of its high performance, watershed segmentation was further optimized by setting its connectivity parameter to 5, 10, 15, 20 and 25. Each configuration was tested in the same training and validation steps, the c-index is displayed as Table 4. Moderate connectivity of 20 provided some performance increase, indicating that careful management of region merging can lead to higher quality segmentation and thus survival modeling.

Table 4. *Watershed Connectivity Tuning Results*

Connectivity value	c-index
5	0.722
10	0.729
15	0.736
20	**0.741**
25	0.727

3.5 Selecting Tiles

The choice of most informative regions from WSIs is crucial to performing the proper survival analysis modelling. Previous steps that included simple aggregations from all the tiles can generate a lot of computational overhead and non-informative or redundant information.

One approach for dealing with high dimensions is to project WSI-level embeddings on to a low-dimensional subspace via PCA of pooled tile-level features. There are two aggregation settings explored for the final feature vector of the patient: maximum pooling and average pooling. Additionally, it's worth exploring if tabular features are adding any information that isn't already coming from the image itself. Figure 5. illustrates that most differentiating principal components do not have predictive power and in order to achieve best performing results, the principal components number

should be in range $[60;80]$. Another important notice is that tabular data in both aggregations contribute towards higher results which may be the limitation of RSF in obtaining the mutual information that is already extracted in the tabular data. Overall, max pooling could be seen as more appropriate method. These graphs suggest that some tiles are probably disproportionately predictively important, so not all tiles are equally important in delineating survival. Focusing on these more informative areas could also offer additional performance improvements over aggregations.

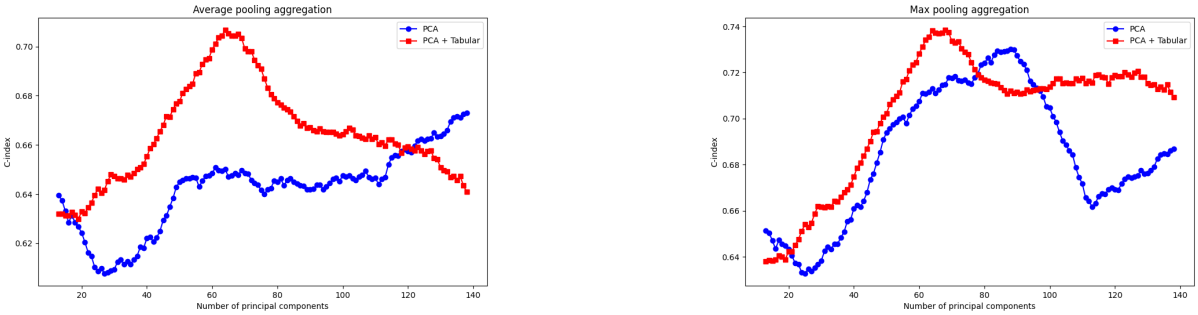
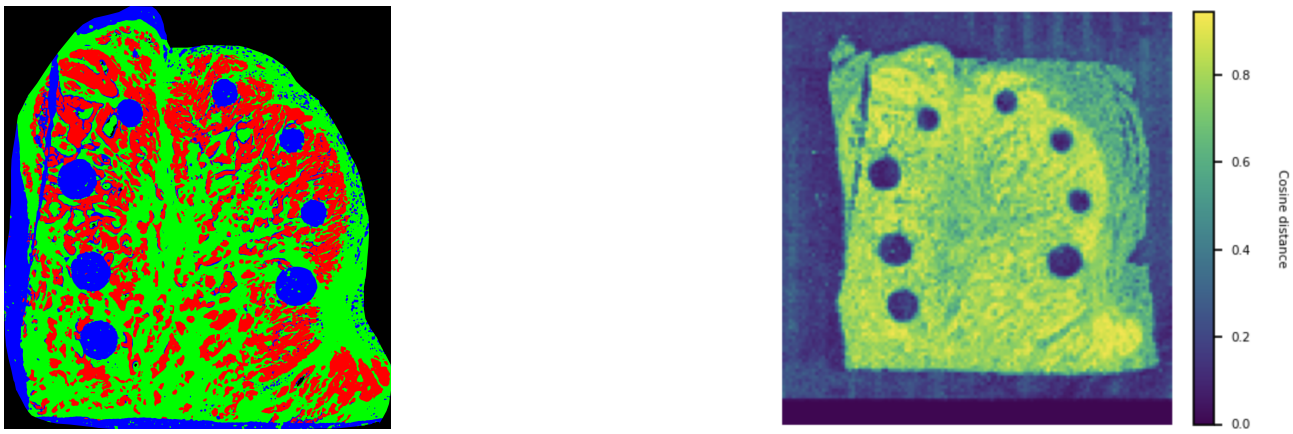


Figure 5. *WSI-level feature extraction comparison in predictive performance*

In comparison to aggregating across numerous tiles, this paper explores the technique of parametrically selecting the tiles. The first step in this process is evaluating cosine distance between each tiles' feature vector and a feature vector obtained from completely white tile (which intuitively should not hold any meaningful information). When visually evaluating the results, they are extremely similar to the deep-learning based WSI segmentation (red indicates tumour, green tissue, blue background) as presented in Figure 6. All the regions (tissue, tumour and background) are being similarly captured by the cosine distance which showcase, that there might be specific ranking percentiles that hold differentiating factors.



(a) *Deep-learning segmented* ***(b)*** *Cosine dissimilarity*

Figure 6. *Comparison of segmented WSI with cosine dissimilarity ranking*

When comparing the RSF model's performance which is being trained on specific ranked percentiles ranging from 0 to 100, results are in favor of the previous assumptions that indeed there are more important tiles and those tiles could be examined via their cosine dissimilarity ranking as pre-

sented in Figure 7. Despite the noisy origin, there are ranking percentiles that would achieve c-index higher than previous experiments with aggregations.

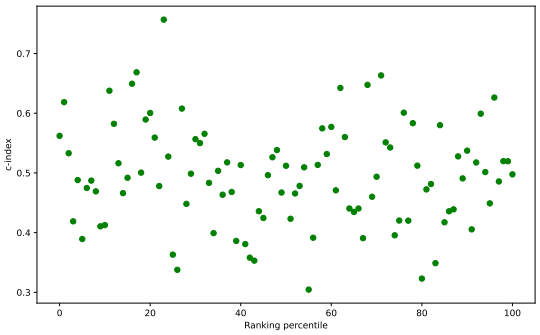


Figure 7. *Ranking percentile RSF inference results*

3.6 Multi layer perceptron setup

3.6.1 Data loading

The data integrates patient-level tabular parameters with tile-level embeddings from WSIs for survival metrics inference. The patient's tabular data that is loaded from pandas dataframe `df` gets normalized using *MinMaxScaler* so that all the tabular features are on the same number range and no bias due to different feature scales occur.

The tile-level information is obtained from pre-calculated embeddings and dissimilarity scores. Each WSI is divided into tiles and dissimilarity measure shows how similar each tile is to a pure white tile. When tiles with dissimilarity scores within dynamically configurable percentiles are chosen, the most informative bits of tissue are taken for analysis. These feature vectors of ranked tiles are then pooled using *average* or *max* pooling. At last, the aggregated tile features are combined with the normalized tabular data to get a single vector for each entity. `Dataset` and `DataLoader` classes from PyTorch are being used for efficiency.

3.6.2 Loss functions and optimization

Classification for survivability task will be penalized using the customized NLL loss which also account for class weights in order to balance the censoring ratios.

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} [-C_i \cdot \ln(1 - S_i) - \alpha \cdot (1 - C_i) \cdot \ln(S_i)] \tag{1}$$

where:

- N is the total number of samples in the dataset.
- $S_i = \texttt{survival\_probs}_i$ represents the predicted survival probability for the i^{th} sample.

- C_i is the censoring indicator for the i^{th} sample, defined as:

$$C_i = \begin{cases} 1 & \text{if the event is observed (uncensored)}, \\ 0 & \text{if the data is censored}. \end{cases}$$

- $\alpha = \texttt{pos\_weight}$ is a hyperparameter that scales the loss contribution from censored samples.

Similarly as with the classification, regression for predicting time to event will be penalized based on customized MSE loss.

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left[C_i \cdot \alpha \cdot (\hat{T}_i - T_i)^2 + (1 - C_i) \cdot \max(T_i - \hat{T}_i, 0)^2 \right] \tag{2}$$

where:

- N is the total number of samples in the dataset.
- $\hat{T}_i = \frac{\texttt{time\_prediction}_i}{D}$ represents the predicted time-to-event for the i^{th} sample, scaled by the divisor D.
- $T_i = \frac{\texttt{time\_to\_event}_i}{D}$ is the actual observed time-to-event for the i^{th} sample, scaled by the same divisor D.
- C_i is the censoring indicator for the i^{th} sample, defined as:

$$C_i = \begin{cases} 1 & \text{if the event is observed (uncensored)}, \\ 0 & \text{if the data is censored}. \end{cases}$$

- $\alpha = \texttt{pos\_weight}$ is a hyperparameter that scales the loss contribution from uncensored samples.
- $D = \texttt{tte\_divisor}$ is a scaling factor applied to both predictions and actual times to ensure similar loss ranges as for classification task.

Later, these two losses are encompasses into `SurvivalMultitaskLoss` which outputs the mean value ensuring the proper loss balance between classification and regression tasks. Such hyperparameters as `pos_weight` and `tte_divisor` will be determined in the hyperparameter optimization step.

From optimization side, PyTorch AdamW optimizer will be employed with some hyperparameters being constant: `betas` will be set to *(0.9, 0.999)*, `eps` set to 10^{-8} and `weight_decay` being 10^{-2}. Best performing learning rate will be determined in the later stages.

3.6.3 Architecture

Due to the small corpus, MLP architecture is intentionally shallow and uses extensive regularization to reduce overfitting. ReLU is the main activation function across the network. Regularization is implemented through dropout layers applied after activation functions. MLP takes a fixed input size of 563 dimensional vector (512 features from WSI feature vector and 51 features from tabular data). The classification head applies sigmoid activation function for constraining the output range to *[0,1]*. The common layer dimensions, linear transformation of the classifier and regression heads will be determined as part of hyperparameter optimization and in Figure 8. are being represented by placeholders by placeholders X, Y and Z.

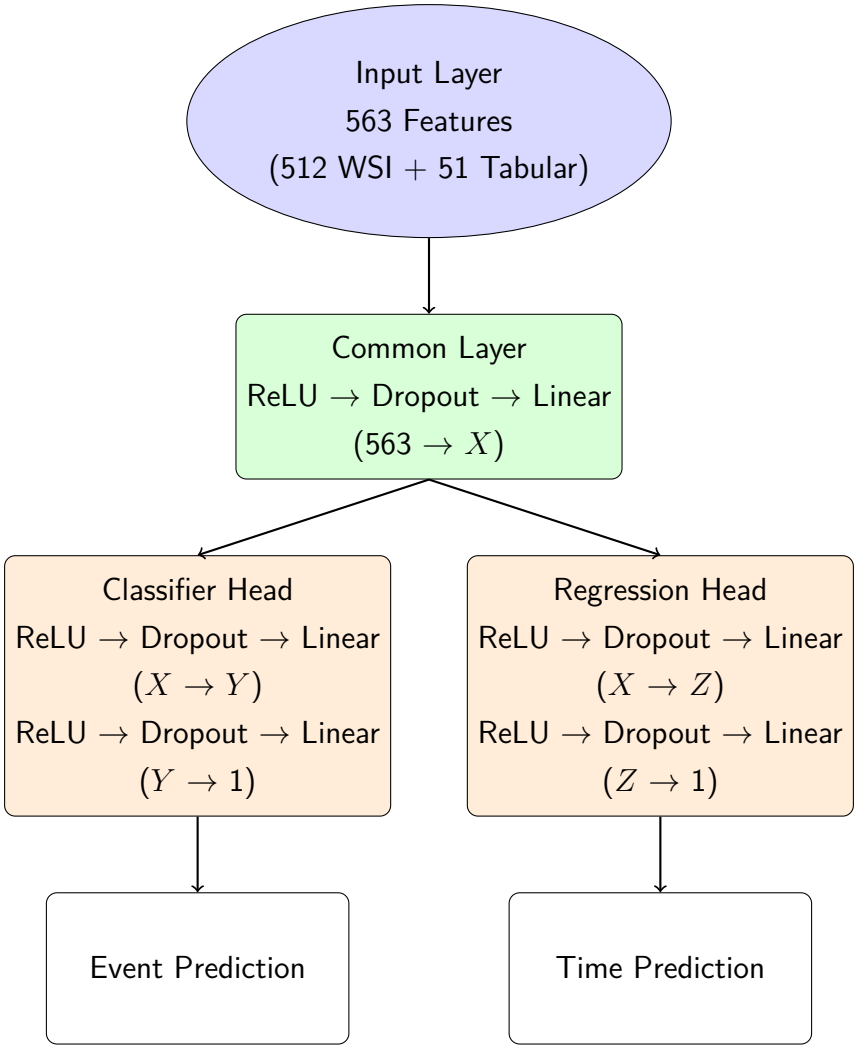


Figure 8. *Architecture of the MLP Model*

3.6.4 Callbacks

Pytorch Lightning package in Python suggests out-of-the-box callbacks that might improve the model training.

- `EarlyStopping` callback tracks validation loss and ends training if there is no significant im-

provement over given number of epochs. This prevents overfitting as the model does not start to memorize training data, and helps generalize better to unseen data.

- `ModelCheckpoint` callback will recurentially store the model state based on the performance metrics, and the best performing model will be stored for future evaluation and deployment.
- `CometLogger` callback logs and visualizes all training metrics to let monitor model performance real-time, compare training runs, and catch problems early.

3.7 Hyperparameter optimization

Bayesian optimization was being done with 500 iterations while searching for the best performing hyperparameters and the model was evaluated against validation dataset. Tunable hyperparameters are explained in the Table 5..

Table 5. *Tunable Hyperparameters*

Parameter	Explanation	Range
Pooling Methods	Determines the type of pooling operation used to aggregate features	`[max, average]`
Hidden Sizes	Specifies the number of neurons in the common hidden layers	`[64,2048]`
Classifier Head Dimensions	Defines the number of neurons in classifier head's linear layer	`[8, 128]`
Regression Head Dimensions	Defines the number of neurons in regression head's linear layer	`[8, 128]`
Batch Sizes	Indicates the number of samples processed before updating the model's parameters	`[2, 18]`
Learning Rates	Sets the step size for updating model parameters	`[0.0001, 0.01]`
Positive Class Weights	Assigns weights to the positive class in the loss function to address high censoring issues.	`[4, 10]`
Dropout Rates	Specifies the probability of dropping neurons during training	`[0.1, 0.5]`
TTE Divisors	Applies scaling factors to the time-to-event data to balance losses	`[15, 60]`
Percentile Combinations	Explores all combinations of best ranked percentiles $\{23, 71, 62, 16, 96\}$	$\{23\}$, $\{71\}$, $\{62\}$, $\{16\}$, $\{96\}$, $\{23,71\}$, ... ,$\{23, 71, 62, 16, 96\}$

3.8 Evaluating Model

Best performing model on validation dataset achieved c-index of **0.829** and MSE / MAE (calculating only for uncensored data) of **521 / 22** (in months) respectively. On the test dataset the model performed slightly worse achieving c-index of **0.823** and MSE/MAE scores of **558 / 24** (in months). Finalized pipeline is presented in the Figure 9. where intentionally aggregation is left as an extension that was not playing any role in the best performing pipeline.

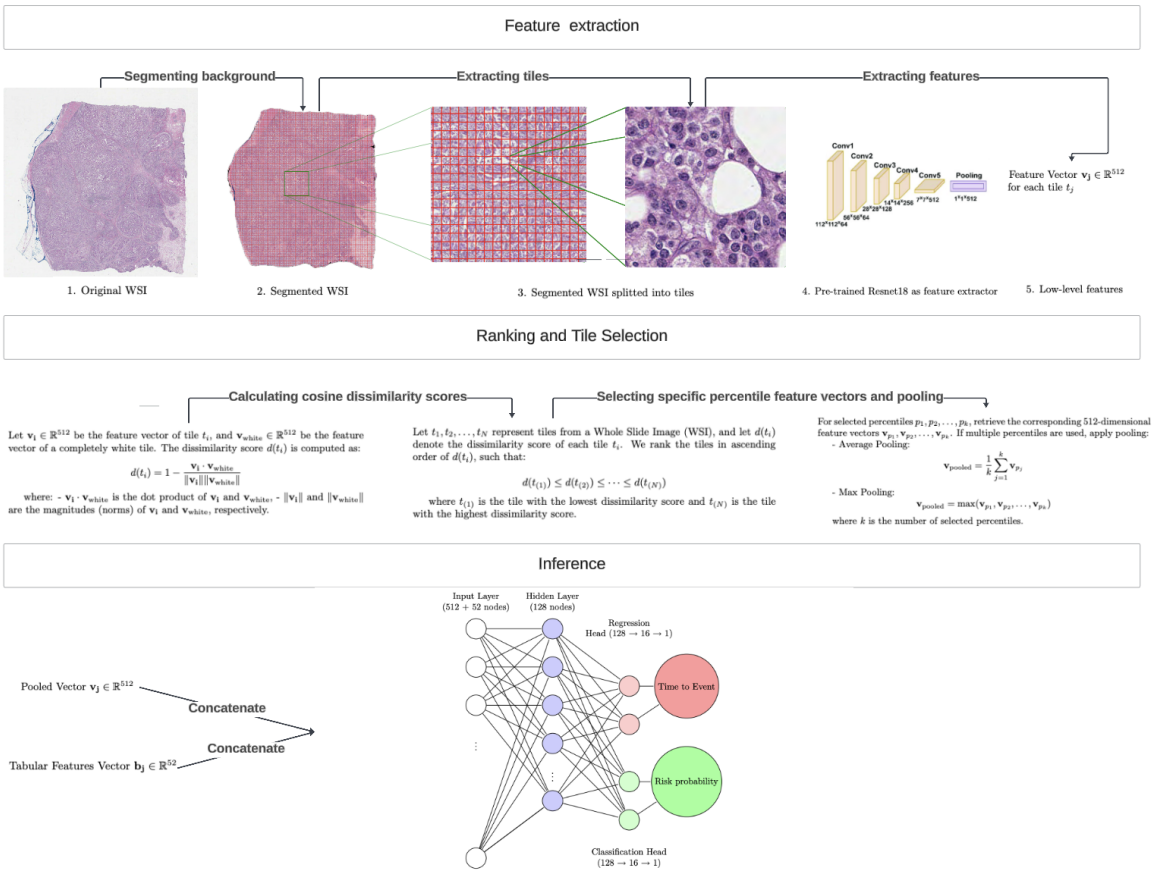


Figure 9. *Finalized pipeline*

During hyperparameter tuning, it was discovered that common layer should have hidden size of `128`, both classifier and regression heads hidden size of `16`. Pooling method made no difference as the best performing model was only using single tile representing `62th` dissimilarity percentile. Training related hyperparameters presented in the Table 6.

Table 6. *Best training related hyperparameters*

Parameter	Value
Learning rate	0.001
Batch size	4
Positive class weight	8.5
Dropout rate	0.2
TTE divisor	35

3.9 Other experiments

The scope of this thesis limits discussion of the experiments that did not contribute to the final pipeline or did not yield any significant improvements. However, they were essential in order to better understand the data and limitations that it holds.

In tile selection there were several options considered to achieve the best representation of the image information. Randomized selection is one method, and it tries to find the variance in the image as a whole by randomly picking tiles. A third approach looked at was non-deterministic tile selection in which tiles exceeding certain threshold were aggregated to identify zones of interest. It also considered clustering-based selection where tiles were selected from predefined clusters and the tiles chosen had to correspond to separate sections of the image data.

In the realm of model inference, an initial attempt was made to construct a model capable of predicting Weibull distribution parameters. This approach was motivated by the desire to model time-to-event data more accurately and the idea that survival probability is not necessarily linear, using the adaptability of the Weibull distribution for different hazard functions. Additionally, custom loss functions concordance index and hazard functions based loss functions were implemented and tested.

In contrast, there was attempt to use graphical neural networks (GNNs) to use the relational model of the WSIs. The benefit of GNNs is that it reveals granular dependencies between the data points by using a graph. But GNN training was constrained by the size of the dataset. The lack of data made training attention mechanisms in the network, the steps that determine which features are most important, not sufficient.

4 Results and Conclusions

4.1 Results

The experiments conducted demonstrated significant improvements in survival analysis using the proposed multi-task learning framework in comparison to the original paper with Cox regression [43]. The model's performance metrics are summarized in Table Table 7.

Table 7. *Model Performance Metrics*

Metric	Validation Set	Test Set	Previous research [43]
c-index	0.829	0.823	0.709
Mean Squared Error (MSE)	521 months^2	558 months^2	-
Mean Absolute Error (MAE)	22 months	24 months	-

Key highlights about the final framework:

- **Feature Extraction:** TiaToolbox ResNet18 achieved the highest c-index (0.721), demonstrating better generalization for survivability prediction compared to other feature extractors (Table 2.).
- **Segmentation:** Watershed segmentation provided the highest performance with a c-index of 0.729 in comparison to other segmentation algorithms, showcasing its ability to enhance predictive accuracy by defining the region of interest (Table 3.).
- **Tile Selection:** Ranking tiles by cosine dissimilarity identified the most differentiating regions, leading to a c-index improvement using specific ranking percentiles (Figure 7.).
- **Model Performance:** The multi-task learning model demonstrated significantly better performance with a validation c-index of 0.829 and test c-index of 0.823 in comparison to baseline methods (Table 7.).
- **Framework:** Combining tabular patient data with histopathological image features, ensuring parametric tile selection and building the final MLP, significantly enhanced survival prediction accuracy, validating the pipeline presented in Figure 9..

4.2 Limitations

Despite its success, the study faced limitations that offer opportunities for future exploration:

- **Dataset Size:** Due to the limited data of 252 entities, the model generalization and framework behaviour might not be robust and fully transferable. However, the suggested framework should be relatevily easy to reproduce given the larger datasets in the future work.
- **High Censoring Rate:** With 87% of data being censored, the results (especially related with TTE predictions) should be reviewed in order to verify the regression head robustness.
- **Tile-Level Analysis:** While selecting tiles by cosine dissimilarity was producing best results, exploring additional methods, such as attention models or clustering, could propose even more robust tile techniques.

- **Advanced Architectures:** Using advanced architectures such as transformer-based models or graph neural networks might be a way to bring features closer to the semantics.
- **Survival Loss Functions:** The study incorporated customized NLL and MSE loss functions which worked better than some existing solutions. Exploring survival-specific loss functions even further may yield additional improvements.

4.3 Conclusions

- Research identified most suitable methods among analysed for breast cancer image preprocessing, which are watershed segmentation followed by feature extraction with ResNet model which eventually gives ability to select specific tiles for assessment.
- Thesis proposed new usage of cosine dissimilarity ranking for tile selection, which could be succesfully applied in practice.
- Research proposed practical and useful framework, for survival prediction based on image and tabular data fusing.
- Thesis expanded the survival analysis scope in comparison to the original paper for this dataset, showcasing that other survivability-related metrics could be successfully infered with multi-task approach.

References and sources

[1] A. Alabdallah, M. Ohlsson, S. Pashami, T. Rögnvaldsson. “The Concordance Index Decomposition: A Measure for a Deeper Understanding of Survival Prediction Models.” In: *arXiv preprint arXiv:2203.00144* (2022).

[2] L. B. de Amorim, G. D. Cavalcanti, R. M. Cruz. “The choice of scaling technique matters for classification performance.” In: *Applied Soft Computing* (2023).

[3] M. Bilal, S. E. A. Raza, A. Azam, S. Graham, M. K. Niazi, M. Ilyas, F. Minhas, N. M. Rajpoot. “TIAToolbox as an end-to-end library for advanced tissue image analytics.” In: *Communications Medicine* (2022).

[4] F. Bray, M. Laversanne, H. Sung, J. Ferlay, R. L. Siegel, I. Soerjomataram, A. Jemal. “Global cancer statistics 2022: GLOBOCAN estimates of incidence and mortality worldwide for 36 cancers in 185 countries.” In: *CA: A Cancer Journal for Clinicians* (2024).

[5] R. J. Chen, M. Y. Lu, J. Wang, D. F. K. Williamson, S. J. Rodig, N. I. Lindeman, F. Mahmood. “Pathomic Fusion: An Integrated Framework for Fusing Histopathology and Genomic Features for Cancer Diagnosis and Prognosis.” In: *IEEE Transactions on Medical Imaging* (2022).

[6] R. Chen, M. Lu, Y. Weng, S. Wang, D. Williamson, F. Mahmood. “A General-Purpose Self-Supervised Model for Computational Pathology.” In: *arXiv preprint arXiv:2308.15474* (2023).

[7] N. Coudray, P. S. Ocampo, T. Sakellaropoulos, R. Narayan, M. Snuderl, D. Fenyo, A. L. Moreira, N. Razavian, A. Tsirigos. “Classification and mutation prediction from non–small cell lung cancer histopathology images using deep learning.” In: *Nature Medicine* (2018).

[8] D. R. Cox. “Regression Models and Life-Tables.” In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1972).

[9] D. Di, S. Li, J. Zhang, Y. Gao. “Ranking-Based Survival Prediction on Histopathological Whole-Slide Images.” In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. 2020.

[10] A. Fisher. *Cloud and Cloud-Shadow Detection in SPOT5 HRG Imagery with Automated Morphological Feature Extraction*. Accessed: 2024-12-29. 2014. URL: https://www.researchgate.net/figure/Two-examples-of-the-watershed-transform-applied-to-a-1-dimensional-signal-A-When_fig2_262985072.

[11] S. Fotso. “Deep Neural Networks for Survival Analysis Based on a Multi-Task Framework.” In: *arXiv preprint arXiv:1801.05512* (2018).

[12] R. C. Gonzalez, R. E. Woods. *Digital Image Processing*. Prentice Hall, 2008.

[13] E. Graf, C. Schmoor, W. Sauerbrei, M. Schumacher. “Assessment and comparison of prognostic classification schemes for survival data.” In: *Statistics in Medicine* (1999).

[14] F. E. Harrell Jr, K. L. Lee, D. B. Mark. “Multivariable prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors.” In: *Statistics in Medicine* (1996).

[15] T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd Edition*. Springer, 2009.

[16] H. He, E. A. Garcia. “Learning from Imbalanced Data.” In: *IEEE Transactions on Knowledge and Data Engineering* (2009).

[17] K. He, X. Zhang, S. Ren, J. Sun. “Deep residual learning for image recognition.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pages 770–778.

[18] G. Huang, Z. Liu, L. van der Maaten, K. Q. Weinberger. “Densely Connected Convolutional Networks.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pages 4700–4708. https://doi.org/10.1109/CVPR.2017.243.

[19] K. Huang, L. Liu, Z. Miao. “DeepConvSurv: A deep convolutional neural network for survival analysis with whole slide images.” In: *Medical Image Analysis* (2019).

[20] H. Ishwaran, U. B. Kogalur, E. H. Blackstone, M. S. Lauer. “Random survival forests.” In: *The Annals of Applied Statistics* (2008).

[21] I. Jolliffe. *Principal Component Analysis 2nd Edition*. Springer, 2002.

[22] K. J. Kaplan, L. Pantanowitz, editors. *Digital Pathology*. 2nd. Cham, Switzerland: Springer, 2021.

[23] J. P. Klein, M. L. Moeschberger. *Survival Analysis: Techniques for Censored and Truncated Data*. 2nd. New York, NY: Springer, 2003.

[24] N. Kumar, R. Verma, S. Sharma, S. Bhargava, G. Breen, S. Rane, N. Rajpoot. “A dataset and a technique for generalized nuclear segmentation for computational pathology.” In: *IEEE Transactions on Medical Imaging* (2017).

[25] V. Kumar, A. K. Abbas, J. C. Aster. *Robbins and Cotran Pathologic Basis of Disease*. 9th. Philadelphia, PA: Elsevier, 2015.

[26] S. Learn. *MinMaxScaler*. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html. Accessed: 2024-12-08.

[27] P. Mobadersany, S. Yousefi, M. Amgad, et al. “Predicting cancer outcomes from histology and genomics using convolutional networks.” In: *Proceedings of the National Academy of Sciences* 115.13 (2018), E2970–E2979. https://doi.org/10.1073/pnas.1717139115.

[28] M. M. F. Mohammad Abuzar Shaikh Muhammad Usama. “Deep Learning on Histopathological Images for Colorectal Cancer Diagnosis: A Comprehensive Review.” In: *Diagnostics* (2022).

[29] L. Pantanowitz, J. H. Sinard, W. H. Henricks, L. A. Fatheree, et al. “Validating whole slide imaging for diagnostic purposes in pathology: guidelines from the College of American Pathologists Pathology and Laboratory Quality Center.” In: *Archives of Pathology & Laboratory Medicine* 137.12 (2013), pages 1710–1722.

[30] O. Ronneberger, P. Fischer, T. Brox. “U-Net: Convolutional networks for biomedical image segmentation.” In: *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Springer, 2015, pages 234–241.

[31] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, D. Batra. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization.” In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017.

[32] J. Snoek, H. Larochelle, R. P. Adams. “Practical Bayesian Optimization of Machine Learning Algorithms.” In: *arXiv preprint arXiv:1206.2944* (2012).

[33] A. Spooner, E. Chen, A. Sowmya, P. Sachdev, N. A. Kochan, J. Trollor, H. Brodaty. “A comparison of machine learning methods for survival analysis of high-dimensional clinical data for dementia prediction.” In: *Scientific Reports* (2020).

[34] N. institute of standards, technology. *Cosine distance, cosine similarity, angular cosine distance, angular cosine similarity*. `https://www.itl.nist.gov/div898/software/dataplot/refman2/auxillar/cosdist.htm`. Accessed: 2024-12-08.

[35] M. Tan, Q. Le. “EfficientNet: Rethinking model scaling for convolutional neural networks.” In: *International Conference on Machine Learning* (2019).

[36] T. M. Therneau, P. M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. New York, NY: Springer, 2000.

[37] “Understanding Survival Analysis in Clinical Trials.” In: *Clinical Oncology* (2020).

[38] S. Vignesh. *The world through the eyes of CNN.* Accessed: 2024-12-29. 2020. URL: `https://medium.com/analytics-vidhya/the-world-through-the-eyes-of-cnn-5a52c034dbeb`.

[39] S. Wiegrebe, P. Kopper, R. Sonabend, B. Bischl, A. Bender. “Deep Learning for Survival Analysis: A Review.” In: *Artificial Intelligence Review* (2024).

[40] Y. Wu, M. Cheng, S. Huang, Z. Pei, et al. “Recent Advances of Deep Learning for Computational Histopathology: Principles and Applications.” In: *Cancers (Basel)* (2022).

[41] A. Zhang, Z. C. Lipton, M. Li, A. J. Smola. *Dive into Deep Learning*. d2l.ai, 2020.

[42] D. Zhu, J. Li, P. Li, Y. Fu. “Robust Deep Multi-task Learning Framework for Cancer Survival Analysis.” In: *IEEE Transactions on Medical Imaging* (2021).

[43] D. Zilenaite-Petrulaitiene, A. Rasmusson, J. Besusparis, R. B. Valkiuniene, R. Augulis, A. Laurinaviciene, B. Plancoulaine, L. Petkevicius, A. Laurinavicius. “Intratumoral heterogeneity of Ki67 proliferation index outperforms conventional immunohistochemistry prognostic factors in estrogen receptor-positive HER2-negative breast cancer.” In: *Virchows Archiv* (2024).

Appendix 1. Code samples

```python
# data_splitting.py
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

TARGET_FIELD = "DSS_10"
COLUMN_TO_DROP = "HER2_FISH" # > 200 NA

def clean_tabular_data(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy()
    for col in df.columns[1:]:
        if df[col].dtype == 'object':
            df[col] = df[col].str.replace(',','.').str.replace('-','').astype(float)

    df.drop(columns=[COLUMN_TO_DROP], inplace=True)
    return df

def preprocess_tabular_data(fname: str) -> tuple[pd.DataFrame, pd.DataFrame, pd.DataFrame]:
    df = pd.read_csv(fname)
    df = clean_tabular_data(df)
    train_df, temp_df = train_test_split(df, test_size=0.4, random_state=42, stratify=df[TARGET_FIELD])
    test_df, val_df = train_test_split(temp_df, test_size=0.5, random_state=42, stratify=df[TARGET_FIELD])

    relevant_fields = df.columns[6:].tolist()
    for field in relevant_fields:
        scaler = MinMaxScaler()
        train_df[field] = scaler.fit_transform(train_df[field].to_numpy().reshape(-1, 1))
        val_df[field] = scaler.transform(val_df[field].to_numpy().reshape(-1, 1))
        test_df[field] = scaler.transform(test_df[field].to_numpy().reshape(-1, 1))
    return train_df, val_df, test_df

# image_preprocessing.py
import itertools
import random
import os

import numpy as np
from tiatoolbox.wsicore.wsireader import WSIReader

TILE_SIZE = (224, 224)
OUTPUT_FOLDER = "processed_arrays"
MAGNIFICATION = 20
IMAGE_DIR = "images"

def preprocessing_pipe(
    path: str,
) -> np.array:
    random.seed(42)
    reader = WSIReader.open(path)
    info_dict = reader.info.as_dict()
    full_dimensions = info_dict["slide_dimensions"]
    actual_tile_size = np.dot(TILE_SIZE, MAGNIFICATION)
    x_coordinates = [x for x in range(0, full_dimensions[0], actual_tile_size[0])]
    y_coordinates = [y for y in range(0, full_dimensions[1], actual_tile_size[1])]
    location_permutations = list(itertools.product(x_coordinates, y_coordinates))

    processed_image_arrays = []
    for location in location_permutations:
        img = reader.read_rect(
```

```
            location,
            TILE_SIZE,
            resolution=info_dict["mpp"][0] * MAGNIFICATION,
            units="mpp",
        )
        processed_image_arrays.append(img)

    processed_image_arrays = np.stack(processed_image_arrays, axis=0)

    return processed_image_arrays

if __name__ == "__main__":
    files = [f"{IMAGE_DIR}/{file}" for file in os.listdir(IMAGE_DIR) if file.endswith(".svs")]
    for file in files:
        filename = file.replace("IMAGE_DIR/", "").split(".")[0]
        filename += ".npy"
        if os.path.exists(f"{OUTPUT_FOLDER}/{filename}"):
            continue
        arr = preprocessing_pipe(file)
        np.save(f"{OUTPUT_FOLDER}/{filename}", arr)


# feature_extractors.py
import os
from typing import Callable

import timm
import torch
import numpy as np
from torch import nn
from PIL import Image

PROCESSED_ARRAYS_DIR = "processed_arrays"
TILE_SIZE = (224, 224)
MODELS = [
    "hf-hub:1aurent/resnet18.tiatoolbox-kather100k",
    "hf-hub:1aurent/resnet34.tiatoolbox-kather100k",
    "hf-hub:1aurent/resnet50.tiatoolbox-kather100k",
    "hf-hub:MahmoodLab/uni"
    ]
DEVICE = "mps"
WHITE_INTENSITY_THRESHOLD = 230
WHITE_MAXIMUM_PORTION = 0.5

def extract_low_level_features(image_arr_path: str,
                               features_dir: str,
                               feature_extractor: nn.Module,
                                 transforms: Callable, thresholding: bool = True,
                                   partial_tiles: bool = True) -> None:
    np.random.seed(42)
    filename = image_arr_path.replace("{PROCESSED_ARRAYS_DIR}/", "").split(".")[0] + ".pt"
    if os.path.exists(f"{features_dir}/{filename}"):
        return
    img_arr = np.load(image_arr_path)

    features = []
    relevant_tiles = 100 if partial_tiles else img_arr.shape[0]
    if partial_tiles:
        np.random.shuffle(img_arr)

    for idx in range(img_arr.shape[0]):
        if thresholding:
```

```
            white_portion = (np.sum(np.mean(img_arr[idx], axis=-1) > WHITE_INTENSITY_THRESHOLD)) / (
                img_arr[idx].shape[0] * img_arr[idx].shape[1]
            )
            if white_portion > WHITE_MAXIMUM_PORTION:
                continue

        with torch.no_grad():
            arr = Image.fromarray(img_arr[idx])
            data = transforms(arr).unsqueeze(dim=0).to(DEVICE)
            output = feature_extractor(data).squeeze(dim=0)
            features.append(output.detach().cpu())
        if len(features) > relevant_tiles:
            break
    features = torch.stack(features, dim=0)
    torch.save(features, f"{features_dir}/{filename}")

if __name__ == "__main__":
    files = [f"{PROCESSED_ARRAYS_DIR}/{file}" for file in os.listdir(PROCESSED_ARRAYS_DIR) if
    ↪  file.endswith(".npy")]
    for model_name in MODELS:
        model = timm.create_model(
                model_name=model_name,
                pretrained=True,
                )
        feature_extractor = nn.Sequential(
            *list(model.children())[:-1]
        )
        feature_extractor.to(DEVICE)
        data_config = timm.data.resolve_model_data_config(model)
        transforms = timm.data.create_transform(**data_config, is_training=False)
        model_features_directory = f"features_{model_name.split("/")[-1].split(".")[0]}"
        for file in files:
            extract_low_level_features(file, model_features_directory, feature_extractor, transforms)
```

```
# feature_extractor_comparison.py
import os

import numpy as np
from sklearn.decomposition import PCA
from sksurv.ensemble import RandomSurvivalForest
from sksurv.metrics import concordance_index_censored
import torch

from data_splitting import preprocess_tabular_data

CSV_FILE = "full_data.csv"
MODEL_FEATURE_DIRS = [
    "features_resnet18",
    "features_resnet34",
    "features_resnet50",
    "features_uni",
]
COLUMN_TO_DROP = "HER2_01"  # does not converge
ID_COLUMN = "Hashed Accession #"

if __name__ == "__main__":
    for feature_dir in MODEL_FEATURE_DIRS:
        train_df, val_df, _ = preprocess_tabular_data(CSV_FILE)
        paths = [
            f"{feature_dir}/{file}"
            for file in os.listdir(feature_dir)
            if file.endswith(".pt")
```

```
        ]
        array_size = torch.load(paths[0]).shape[0]
        for df in (train_df, val_df):
            df.drop(columns=[COLUMN_TO_DROP], inplace=True)
            for idx in range(array_size):
                df[f"feature_{idx}"] = 0

        for path in paths:
            identifier = path.split("/")[-1].replace(".pt", "")
            feature_values = torch.load(path).numpy()
            if identifier in train_df[ID_COLUMN]:
                idx = train_df[ID_COLUMN].tolist().index(identifier)
                for feature_idx, value in enumerate(feature_values):
                    train_df.at[idx, f"feature_{idx}"] = value
            elif identifier in train_df[ID_COLUMN]:
                idx = val_df[ID_COLUMN].tolist().index(identifier)
                for feature_idx, value in enumerate(feature_values):
                    val_df.at[idx, f"feature_{idx}"] = value
            else:
                raise ValueError

        feature_columns = [
            col for col in train_df.columns.tolist() if col.startswith("feature_")
        ]
        pca = PCA(n_components=100)
        train_reduced = pca.fit_transform(train_df[feature_columns])
        val_reduced = pca.transform(val_df[feature_columns])

        train_y = np.array(
            [
                (event, time)
                for event, time in zip(train_df["DSS_10"], train_df["trukme_10"])
            ],
            dtype=[("event", "?"), ("time", "<f8")],
        )

        rsf = RandomSurvivalForest(random_state=42)
        rsf.fit(train_reduced, train_y)
        risk_scores = rsf.predict(val_df[feature_columns])
        censoring = val_df["DSS_10"]
        ci = concordance_index_censored(censoring, val_df["trukme_10"], risk_scores)[0]
        print("-" * 30)
        print(f"Analyzing extracted features from {feature_dir}")
        print(f"Concordance index {ci}")


# segmentation_comparison.py
import os

import cv2
import torch
import numpy as np
from sklearn.decomposition import PCA
from sksurv.ensemble import RandomSurvivalForest
from sksurv.metrics import concordance_index_censored
from skimage import filters, morphology, measure, segmentation
from skimage.color import rgb2gray
from tiatoolbox.wsicore.wsireader import WSIReader

from data_splitting import preprocess_tabular_data

CSV_FILE = "full_data.csv"
```

```
SEGMENTATION_METHODS = [
    "otsu",
    "adaptive",
    "region_growing",
    "watershed",
]
TILE_SIZE = (224, 224)
ID_COLUMN = "Hashed Accession #"
COLUMN_TO_DROP = "HER2_01"  # does not converge
WATERSHED_CONNECTIVITY = 20
FEATURES_DIR = "features_resnet18"
IMAGE_DIR = "images"


def segment_image(image: np.ndarray, method: str, size: tuple) -> np.ndarray:
    grayscale_image = rgb2gray(image)
    if method == "otsu":
        threshold = filters.threshold_otsu(grayscale_image)
        mask = grayscale_image < threshold
    elif method == "adaptive":
        mask = filters.threshold_local(grayscale_image, block_size=35) < grayscale_image
    elif method == "region_growing":
        mask = morphology.remove_small_objects(
            morphology.label(grayscale_image < 0.5), 50
        )
    elif method == "watershed":
        elevation_map = filters.sobel(grayscale_image)
        markers = measure.label(
            grayscale_image < filters.threshold_otsu(grayscale_image)
        )
        mask = (
            segmentation.watershed(
                elevation_map, markers, connectivity=WATERSHED_CONNECTIVITY
            )
            > 0
        )
    else:
        raise ValueError("Unsupported segmentation method.")
    return mask


def apply_mask_to_features(features: torch.Tensor, mask: np.ndarray) -> torch.Tensor:
    mask_flat = mask.flatten()
    valid_features = features[:, mask_flat]
    return valid_features


if __name__ == "__main__":
    train_df, val_df, _ = preprocess_tabular_data(CSV_FILE)
    for method in SEGMENTATION_METHODS:
        train_df, val_df, _ = preprocess_tabular_data(CSV_FILE)
        paths = [
            f"{FEATURES_DIR}/{file}"
            for file in os.listdir(FEATURES_DIR)
            if file.endswith(".pt")
        ]
        array_size = torch.load(paths[0]).shape[0]
        for df in (train_df, val_df):
            df.drop(columns=[COLUMN_TO_DROP], inplace=True)
            for idx in range(array_size):
                df[f"feature_{idx}"] = 0
```

```
        for path in paths:
            identifier = path.split("/")[-1].replace(".pt", "")
            feature_values = torch.load(path)

            image_path = f"{IMAGE_DIR}/{identifier}.svs"
            reader = WSIReader.open(image_path)
            image = reader.slide_thumbnail(resolution=1, units="power")
            mask = segment_image(image, method)
            mask = cv2.resize(
                mask, feature_values.shape[:-1], interpolation=cv2.INTER_NEAREST
            )
            valid_indices = np.where(mask > 0)
            feature_values = feature_values[valid_indices].mean(dim=0).numpy()
            if identifier in train_df[ID_COLUMN]:
                idx = train_df[ID_COLUMN].tolist().index(identifier)
                for feature_idx, value in enumerate(feature_values):
                    train_df.at[idx, f"feature_{idx}"] = value
            elif identifier in train_df[ID_COLUMN]:
                idx = val_df[ID_COLUMN].tolist().index(identifier)
                for feature_idx, value in enumerate(feature_values):
                    val_df.at[idx, f"feature_{idx}"] = value
            else:
                raise ValueError

        feature_columns = [
            col for col in train_df.columns.tolist() if col.startswith("feature_")
        ]

        pca = PCA(n_components=100)
        train_reduced = pca.fit_transform(train_df[feature_columns])
        val_reduced = pca.transform(val_df[feature_columns])

        train_y = np.array(
            [
                (event, time)
                for event, time in zip(train_df["DSS_10"], train_df["trukme_10"])
            ],
            dtype=[("event", "?"), ("time", "<f8")],
        )

        rsf = RandomSurvivalForest(random_state=42)
        rsf.fit(train_reduced, train_y)

        risk_scores = rsf.predict(val_reduced)
        censoring = val_df["DSS_10"].to_numpy()
        ci = concordance_index_censored(
            censoring, val_df["trukme_10"].to_numpy(), risk_scores
        )[0]
        print(f"Segmentation method: {method}, Concordance index: {ci}")

# dissimilarity_score_extraction.py
import os

import timm
import numpy as np
from tiatoolbox.wsicore.wsireader import WSIReader
import torch
from torch import nn
from PIL import Image
from scipy.spatial.distance import cosine

MODEL_NAME = "hf-hub:1aurent/resnet18.tiatoolbox-kather100k"
```

```
IMAGE_DIR = "images"
FEATURES_DIR = "features"
DEVICE = "mps"

files = [
    f"{IMAGE_DIR}/{file}" for file in os.listdir(IMAGE_DIR) if file.endswith(".svs")
]
tile_size = (224, 224)

model = timm.create_model(
    model_name=MODEL_NAME,
    pretrained=True,
)
device = torch.device(DEVICE)
feature_extractor = nn.Sequential(*list(model.children())[:-1])
feature_extractor.to(device)
feature_extractor.eval()

data_config = timm.data.resolve_model_data_config(model)
transforms = timm.data.create_transform(**data_config, is_training=False)


def extract_low_level_features(img_arr: np.array) -> None:
    with torch.no_grad():
        arr = Image.fromarray(img_arr)
        data = transforms(arr).unsqueeze(dim=0).to(device)
        output = feature_extractor(data).squeeze(dim=0).detach().cpu().numpy()
    return output


blank_arr = np.ones((224, 224, 3), dtype=np.uint8) * 255
blank_features = extract_low_level_features(blank_arr)

if __name__ == "__main__":
    for file in files:
        filename = file.replace(f"{IMAGE_DIR}/", "").replace(".svs", "")
        if os.path.exists(f"{FEATURES_DIR}/{filename}_dis.npy") and os.path.exists(
            f"{FEATURES_DIR}/{filename}_feat.npy"
        ):
            continue

        reader = WSIReader.open(file)
        info_dict = reader.info.as_dict()
        full_dimensions = info_dict["slide_dimensions"]
        x_coordinates = [x for x in range(0, full_dimensions[1], tile_size[0])]
        y_coordinates = [y for y in range(0, full_dimensions[0], tile_size[1])]

        dissimilarity_arr = np.zeros((len(y_coordinates), len(x_coordinates)))
        feature_arr = np.zeros((len(y_coordinates), len(x_coordinates), 512))

        for idx, y in enumerate(y_coordinates):
            for jdx, x in enumerate(x_coordinates):
                img = reader.read_rect((x, y), tile_size, resolution=0, units="level")
                features = extract_low_level_features(img)
                dissimilarity = cosine(blank_features, features)

                dissimilarity_arr[idx][jdx] = dissimilarity
                feature_arr[idx][jdx] = features

        np.save(f"{FEATURES_DIR}/{filename}_dis.npy", dissimilarity_arr)
        np.save(f"{FEATURES_DIR}/{filename}_feat.npy", feature_arr)
```

```python
# tile_selection.py
import cv2
import torch
import numpy as np
import pandas as pd
from sksurv.ensemble import RandomSurvivalForest
from sksurv.metrics import concordance_index_censored
import matplotlib.pyplot as plt

from data_splitting import preprocess_tabular_data

CSV_FILE = "full_data.csv"
COLUMN_TO_DROP = "HER2_01"  # does not converge
ID_COLUMN = "Hashed Accession #"
IMAGE_FEATURES = [f"img_{i}" for i in range(512)]

if __name__ == "__main__":
    for percentile in range(101):
        train_df, val_df, _ = preprocess_tabular_data(CSV_FILE)
        train_df.drop(columns=[COLUMN_TO_DROP], inplace=True)
        val_df.drop(columns=[COLUMN_TO_DROP], inplace=True)
        for col in IMAGE_FEATURES:
            train_df[col] = 0
            val_df[col] = 0
        percentiles = [percentile]
        for identifier, df in zip(
            (train_df[ID_COLUMN], val_df[ID_COLUMN]), (train_df, val_df)
        ):
            dissimilarity_scores = np.load(f"features/{identifier}_dis.npy")
            shape_0, shape_1 = dissimilarity_scores.shape
            background_mask = np.load(f"background_masks_20/{identifier}.npy")
            mask_resized = cv2.resize(
                background_mask, (shape_1, shape_0), interpolation=cv2.INTER_NEAREST
            )
            dissimilarity_scores_flat = dissimilarity_scores.reshape(shape_0 * shape_1)
            mask_flat = mask_resized.flatten()

            valid_indices = np.where(mask_flat > 0)
            dissimilarity_scores_filtered = dissimilarity_scores_flat[valid_indices]
            percentile_values = np.percentile(
                dissimilarity_scores_filtered, percentiles
            )
            indices = [
                np.argmin(np.abs(dissimilarity_scores_filtered - pv))
                for pv in percentile_values
            ]

            features = np.load(f"features/{identifier}_feat.npy")
            features = features.reshape(shape_0 * shape_1, 512)
            image_features = torch.tensor(features[indices, :], dtype=torch.float32)
            image_features = torch.max(image_features, dim=0).values

            idx = df[df[ID_COLUMN] == identifier].index.values[0]
            for feature_idx, img_feature in enumerate(image_features.numpy()):
                train_df.at[idx, f"img_{feature_idx}"] = img_feature

        minority_class = train_df[train_df["DSS_10"] == 1]
        majority_class = train_df[train_df["DSS_10"] == 0]
        minority_class_oversampled = minority_class.sample(
            len(majority_class), replace=True, random_state=42
        )
        train_df = pd.concat([majority_class, minority_class_oversampled])
```

```
        train_df = train_df.sample(frac=1, random_state=42)
        train_df.reset_index(inplace=True, drop=True)

        y = np.array(
            [
                (event, time)
                for event, time in zip(train_df["DSS_10"], train_df["trukme_10"])
            ],
            dtype=[("event", "?"), ("time", "<f8")],
        )
        censoring = val_df["DSS_10"] == 1
        rsf = RandomSurvivalForest(random_state=42)
        rsf.fit(train_df[IMAGE_FEATURES], y)
        risk_scores = rsf.predict(val_df[IMAGE_FEATURES])
        ci = concordance_index_censored(censoring, val_df["trukme_10"], risk_scores)[0]
        print(f"Percentile {percentile}, result {ci}")

# multi_layer_perceptron.py
from typing import Callable
from functools import partial

import torch
import pandas as pd
import cv2
import numpy as np
from torch.utils.data import Dataset
from torch import nn
import pytorch_lightning as pl

CSV_PATH = "full_data.csv"
df = pd.read_csv(CSV_PATH)
RELEVANT_FIELDS = df.columns[6:].tolist()
ID_COLUMN = "Hashed Accession #"
FEATURES_DIR = "features"
BACKGROUND_MASKS_DIR = "background_masks_20"
CENSORING_COLUMN = "DSS_10"
TIME_TO_EVENT_COLUMN = "trukme_10"


class HistopathologyDataset(Dataset):
    def __init__(self, df: pd.DataFrame, pooling: str, percentiles: list[int]) -> None:
        self.data = df
        self.pooling = pooling
        self.percentiles = percentiles

    def __len__(self) -> int:
        return len(self.data)

    def __getitem__(self, index: int) -> tuple[torch.Tensor, torch.Tensor]:
        tabular_features = torch.tensor(
            self.data.loc[index, RELEVANT_FIELDS], dtype=torch.float32
        )
        name = self.data[ID_COLUMN][index]

        dissimilarity_scores = np.load(f"{FEATURES_DIR}/{name}_dis.npy")
        shape_0, shape_1 = dissimilarity_scores.shape
        background_mask = np.load(f"{BACKGROUND_MASKS_DIR}/{name}.npy")
        mask_resized = cv2.resize(
            background_mask, (shape_1, shape_0), interpolation=cv2.INTER_NEAREST
        )

        dissimilarity_scores_flat = dissimilarity_scores.reshape(shape_0 * shape_1)
```

```
        mask_flat = mask_resized.flatten()

        valid_indices = np.where(mask_flat > 0)
        dissimilarity_scores_filtered = dissimilarity_scores_flat[valid_indices]
        percentile_values = np.percentile(
            dissimilarity_scores_filtered, self.percentiles
        )
        indices = [
            np.argmin(np.abs(dissimilarity_scores_filtered - pv))
            for pv in percentile_values
        ]

        features = np.load(f"{FEATURES_DIR}/{name}_feat.npy")
        features = features.reshape(shape_0 * shape_1, 512)
        image_features = torch.tensor(features[indices, :], dtype=torch.float32)

        if self.pooling == "average":
            image_features = torch.mean(image_features, dim=0)
        elif self.pooling == "max":
            image_features = torch.max(image_features, dim=0).values
        else:
            raise NotImplementedError

        features = torch.cat([tabular_features, image_features])
        event_indicator = torch.tensor(
            [self.data.loc[index, CENSORING_COLUMN] == 1], dtype=torch.bool
        )
        time_to_event = torch.tensor(
            [self.data.loc[index, TIME_TO_EVENT_COLUMN]], dtype=torch.float32
        )

        return features, event_indicator, time_to_event


class SurvivalMSELoss(nn.Module):
    def __init__(self, pos_weight: float, tte_divisor: float) -> None:
        super(SurvivalMSELoss, self).__init__()
        self.pos_weight = pos_weight
        self.tte_divisor = tte_divisor

    def forward(
        self,
        time_prediction: torch.Tensor,
        time_to_event: torch.Tensor,
        event_indicator: torch.Tensor,
    ) -> torch.Tensor:
        time_prediction /= self.tte_divisor
        time_to_event /= self.tte_divisor

        mse_uncensored = (time_prediction - time_to_event) ** 2 * self.pos_weight
        censored_penalty = torch.clamp(time_to_event - time_prediction, min=0) ** 2
        loss = torch.where(event_indicator == 0, mse_uncensored, censored_penalty)
        return loss.mean()


class SurvivalNLLLoss(nn.Module):
    def __init__(self, pos_weight: float = 1) -> None:
        super(SurvivalNLLLoss, self).__init__()
        self.pos_weight = pos_weight

    def forward(
        self, survival_probs: torch.Tensor, censoring: torch.Tensor
```

```
    ) -> torch.Tensor:
        survival_probs = torch.clamp(survival_probs, min=1e-8, max=1 - 1e-8)
        event_loss = -torch.log(1 - survival_probs) * censoring
        censored_loss = -self.pos_weight * torch.log(survival_probs) * (~censoring)
        loss = (event_loss + censored_loss).mean()
        return loss


class SurvivalMultitaskLoss(nn.Module):
    def __init__(self, pos_weight: float = 1, tte_divisor: float = 40) -> None:
        super(SurvivalMultitaskLoss, self).__init__()
        self.mse_loss = SurvivalMSELoss(pos_weight=pos_weight, tte_divisor=tte_divisor)
        self.nll_loss = SurvivalNLLLoss(pos_weight=pos_weight)

    def forward(
        self,
        event_prediction: torch.Tensor,
        time_prediction: torch.Tensor,
        time_to_event: torch.Tensor,
        event_indicator: torch.Tensor,
        log_func: Callable,
        cycle: str = "train",
    ) -> torch.Tensor:
        reg_loss = self.mse_loss(
            time_prediction=time_prediction,
            time_to_event=time_to_event,
            event_indicator=event_indicator,
        )
        log_func(cycle + "_reg_loss", reg_loss)
        clf_loss = self.nll_loss(
            survival_probs=event_prediction, censoring=event_indicator
        )
        log_func(cycle + "_clf_loss", clf_loss)

        return (reg_loss + clf_loss) / 2


class MLP(pl.LightningModule):
    def __init__(
        self,
        lr: float,
        input_size: int = 512,
        hidden_size: int = 128,
        classification_size: int = 16,
        regression_size: int = 16,
        pos_class_weight: float = 8.5,
        dropout: float = 0.2,
        batch_size: int = 4,
        percentiles: list = [],
        pooling: str = "max",
        tte_divisor: int = 30,
    ):
        super().__init__()
        self.batch_size = batch_size

        self.params = {
            "lr": lr,
            "hidden_size": hidden_size,
            "pooling": pooling,
            "batch_size": batch_size,
            "pos_class_weight": pos_class_weight,
            "dropout": dropout,
```

```
            "batch_size": batch_size,
            "percentiles": percentiles,
            "regression_size": regression_size,
            "classification_size": classification_size,
            "tte_divisor": tte_divisor,
        }
        self.save_hyperparameters(self.params)
        self.common_layer = nn.Sequential(
            nn.ReLU(), nn.Dropout(dropout), nn.Linear(input_size, hidden_size)
        )

        self.classifier_head = nn.Sequential(
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(hidden_size, classification_size),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(classification_size, 1),
            nn.Sigmoid(),
        )

        self.regression_head = nn.Sequential(
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(hidden_size, regression_size),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(regression_size, 1),
        )

        self.lr = lr
        self.loss = SurvivalMultitaskLoss(
            pos_weight=pos_class_weight, tte_divisor=tte_divisor
        )
        self.log_function = partial(
            self.log, on_step=False, on_epoch=True, batch_size=self.batch_size
        )

    def forward(self, x: torch.Tensor) -> tuple[torch.Tensor, torch.Tensor]:
        x = self.common_layer(x)
        event_pred = self.classifier_head(x)
        time_pred = self.regression_head(x)
        return event_pred, time_pred

    def training_step(
        self, batch: tuple[torch.Tensor, torch.Tensor], batch_idx: int
    ) -> torch.Tensor:
        x, censoring, time_to_event = batch
        x = x.view(x.size(0), -1)

        event_pred, time_pred = self(x)
        loss = self.loss(
            event_prediction=event_pred,
            time_prediction=time_pred,
            time_to_event=time_to_event,
            event_indicator=censoring,
            log_func=self.log_function,
            cycle="train",
        )

        self.log(
            "train_loss", loss, on_step=False, on_epoch=True, batch_size=self.batch_size
```

```
        )
        return loss

    def validation_step(
        self, batch: tuple[torch.Tensor, torch.Tensor], batch_idx: int
    ) -> torch.Tensor:
        x, censoring, time_to_event = batch
        x = x.view(x.size(0), -1)

        event_pred, time_pred = self(x)
        loss = self.loss(
            event_prediction=event_pred,
            time_prediction=time_pred,
            time_to_event=time_to_event,
            event_indicator=censoring,
            log_func=self.log_function,
            cycle="val",
        )

        self.log(
            "val_loss", loss, on_step=False, on_epoch=True, batch_size=self.batch_size
        )
        return loss

    def configure_optimizers(self):
        optimizer = torch.optim.AdamW(self.parameters(), lr=self.lr)
        return optimizer


# hyperparameter_optimization.py
import optuna
from sksurv.metrics import concordance_index_censored
from torch.utils.data import DataLoader
import pytorch_lightning as pl
from itertools import combinations
from pytorch_lightning.callbacks import ModelCheckpoint, EarlyStopping
from pytorch_lightning.loggers import CometLogger
import torch
import numpy as np
from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error

from data_splitting import preprocess_tabular_data
from multi_layer_perceptron import HistopathologyDataset, MLP

CSV_PATH = "full_data.csv"


def objective(trial: optuna.Trial) -> float:
    best_ranks = [23, 71, 62, 16, 96]

    lr = trial.suggest_float("lr", 1e-4, 1e-2, log=True)
    hidden_size = trial.suggest_int("hidden_size", 64, 2048, step=32)
    classification_size = trial.suggest_int("classification_size", 8, 128, step=8)
    regression_size = trial.suggest_int("regression_size", 8, 128, step=8)
    pos_class_weight = trial.suggest_float(pos_class_weight, 4, 10, step=0.5)
    dropout = trial.suggest_float("dropout", 0.1, 0.5, step=0.05)
    batch_size = trial.suggest_int("batch_size", 2, 18)
    percentiles = trial.suggest_categorical(
        "percentiles",
        [
            combo
            for r in range(1, len(best_ranks) + 1)
```

```
        for combo in combinations(best_ranks, r)
    ],
)
pooling = trial.suggest_categorical("pooling", ["max", "average"])
tte_divisor = trial.suggest_int("tte_divisor", 15, 60)
pl.seed_everything(42)

train_df, val_df, _ = preprocess_tabular_data(CSV_PATH)

train_ds = HistopathologyDataset(train_df, pooling=pooling, percentiles=percentiles)
val_ds = HistopathologyDataset(val_df, pooling=pooling, percentiles=percentiles)
train_dl = DataLoader(train_ds, batch_size, shuffle=True)
val_dl = DataLoader(val_ds, batch_size, shuffle=False)

relevant_fields = train_df.columns[6:].tolist()

model = MLP(
    lr=lr,
    input_size=512 + len(relevant_fields),
    hidden_size=hidden_size,
    classification_size=classification_size,
    regression_size=regression_size,
    pos_class_weight=pos_class_weight,
    dropout=dropout,
    batch_size=batch_size,
    percentiles=percentiles,
    pooling=pooling,
    tte_divisor=tte_divisor,
)

checkpoint_callback = ModelCheckpoint(
    dirpath="models",
    filename="model_v5_{epoch}-{val_loss:.2f}",
    save_top_k=1,
    monitor="val_loss",
    mode="min",
)
early_stop_callback = EarlyStopping(monitor="val_loss", mode="min", patience=20)
comet_logger = CometLogger(
    api_key="...",
    project_name="...",
    workspace="...",
)
trainer = pl.Trainer(
    max_epochs=200,
    accelerator="mps",
    callbacks=[checkpoint_callback, early_stop_callback],
    logger=comet_logger,
)
trainer.fit(model, train_dataloaders=train_dl, val_dataloaders=val_dl)

best_model = MLP.load_from_checkpoint(
    trainer.checkpoint_callback.best_model_path,
    lr=lr,
    input_size=512 + len(relevant_fields),
    hidden_size=hidden_size,
)

event_preds = []
event_indicators = []
tte_preds = []
tte_real = []
```

```
    best_model.to("cpu")
    best_model.eval()
    with torch.no_grad():
        try:
            for item in val_ds:
                x, c, tte = item
                event_indicators.append(c.numpy()[0])
                event_pred, time_pred = best_model(x)
                event_preds.append(event_pred.numpy()[0])
                tte_preds.append(time_pred.numpy()[0])
                tte_real.append(tte.numpy()[0])
        except (ValueError, KeyError):
            pass
    ci = concordance_index_censored(event_indicators, tte_real, event_preds)[0]
    comet_logger.log_metrics({"val c-index": ci})
    actual_events_mask = np.array(event_indicators) == 1
    filtered_tte_real = np.array(tte_real)[actual_events_mask]
    filtered_tte_preds = np.array(tte_preds)[actual_events_mask]
    mse = mean_squared_error(filtered_tte_real, filtered_tte_preds)
    r2 = r2_score(filtered_tte_real, filtered_tte_preds)
    mae = mean_absolute_error(filtered_tte_real, filtered_tte_preds)
    comet_logger.log_metrics({"val mse": mse})
    comet_logger.log_metrics({"val r2": r2})
    comet_logger.log_metrics({"val mae": mae})

    return ci


if __name__ == "__main__":
    study = optuna.create_study(direction="maximize")
    study.optimize(objective, n_trials=500)

# evaluation.py
import torch
from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error
from sksurv.metrics import concordance_index_censored
import numpy as np

from data_splitting import preprocess_tabular_data
from multi_layer_perceptron import HistopathologyDataset, MLP

BEST_PERCENTILES = [62]
BEST_POOLING = "max"
REGRESSION_HEAD_SIZE = 16
CLASSIFIER_HEAD_SIZE = 16
HIDDEN_SIZE = 128
CHECKPOINT_PATH = "models/model_v5_155-0.68"
CSV_PATH = "full_data.csv"


def evaluate_best_model() -> None:
    _, _, test_df = preprocess_tabular_data(CSV_PATH)

    test_ds = HistopathologyDataset(
        test_df, pooling=BEST_POOLING, percentiles=BEST_PERCENTILES
    )

    relevant_fields = test_df.columns[6:].tolist()

    model = MLP.load_from_checkpoint(
        CHECKPOINT_PATH,
        lr=0.01,
```

```
    input_size=512 + len(relevant_fields),
    hidden_size=HIDDEN_SIZE,
    classification_size=CLASSIFIER_HEAD_SIZE,
    regression_size=REGRESSION_HEAD_SIZE,
)

event_preds = []
event_indicators = []
tte_preds = []
tte_real = []
model.to("cpu")
model.eval()
with torch.no_grad():
    try:
        for item in test_ds:
            x, c, tte = item
            event_indicators.append(c.numpy()[0])
            event_pred, time_pred = model(x)
            event_preds.append(event_pred.numpy()[0])
            tte_preds.append(time_pred.numpy()[0])
            tte_real.append(tte.numpy()[0])
    except (ValueError, KeyError):
        pass
ci = concordance_index_censored(event_indicators, tte_real, event_preds)[0]
print({"test c-index": ci})
actual_events_mask = np.array(event_indicators) == 1
filtered_tte_real = np.array(tte_real)[actual_events_mask]
filtered_tte_preds = np.array(tte_preds)[actual_events_mask]
mse = mean_squared_error(filtered_tte_real, filtered_tte_preds)
r2 = r2_score(filtered_tte_real, filtered_tte_preds)
mae = mean_absolute_error(filtered_tte_real, filtered_tte_preds)
print({"test mse": mse})
print({"test r2": r2})
print({"test mae": mae})
```