SuMO - Benchmarking **Su**rvival Analysis Approaches for **M**ulti-M**o**dal Data

Marlon Helbing

Bachelor Thesis in Computer Science

Prof. Dr. Florian Büttner
Machine Learning in Oncology
Department of Computer Science
Goethe University Frankfurt
22.03.2023

Advised by
Arber Qoku

Erklärung zur Abschlussarbeit

gemäß § 35, Abs. 16 der Ordnung für den Bachelorstudiengang Informatik vom 17. Juni 2019:

Hiermit erkläre ich, Marlon Helbing, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe.

Ich bestätige außerdem, dass die vorliegende Arbeit nicht, auch nicht auszugsweise, für eine andere Prüfung oder Studienleistung verwendet wurde.

Zudem versichere ich, dass alle eingereichten schriftlichen gebundenen Versionen meiner vorliegenden Bachelorarbeit mit der digital eingereichten elektronischen Version meiner Bachelorarbeit übereinstimmen.

Frankfurt am Main, den 22.03.2023

.....

Marlon Helbing

Abstract

Cancer is among the leading causes of death worldwide. Thus, many approaches have been made to enhance our understanding of the relationship between patients and disease, especially by utilizing Deep Learning methods. This work provides a collection of modules that can be connected piece by piece to form a pipeline for continuous survival estimation on multi-view genetic data. We implement three neural networks that can make use of different methods to preprocess data, minimize dimensions by feature selection and integrate their multi-view aspect. More specifically, we will benchmark 36 different neural network settings on four scaling and three feature selection methods. Finally, we will evaluate performances across three distinct cancer types by analyzing key performance indicators, such as the c-index scores, hyperparameter importance, and time complexity.

Contents

Chapter 1

Brief history and introduction

The term survival analysis dates back to the 17th century. While its' primary purpose is to investigate fatality rates as a part of medical research, the practice has long extended across fields such as criminology or sociology. It is commonly described as a set of statistical methods used to investigate time-to-event data, such as time until failure or, in our work, the death of a patient. A widespread method for estimating such survival functions is the Kaplan-Meier estimator (Kaplan and Meier, 1958). However, using this approach, we can only model the effect of one covariate - sometimes also referred to as predictor - variable on the survival time of a patient. Technological advances made in the past years made it easier and easier to extract a human's high dimensional genetic data, consisting of features from different views such as mRNA or DNA. Thus, the proportional hazards model, first described by David Cox (Cox, 1972), can be seen as a timely invention, as it is a regression model which allows to evaluate the effect of multiple covariates simultaneously.

In recent years, neural networks (NN) have become an indispensable tool for research in nearly every scientific field. Humans process data by using highly interconnected networks of layers of neurons, which communicate by sending electric pulses to one another via synapses. In 1943, McCulloch and Pitts (McCulloch and Pitts, 1943) modeled interactions between neurons, particularly showing that a neuron may be activated depending on its' weighted input produced by the previous layer of neurons. About two decades later, research suggested that their idea of neuron communication resembles brain activities. Over the years, modeled NNs became more sophisticated and were even able to challenge humans in particular fields, such as in 2016, when DeepMind's "AlphaGo" triumphed over a renowned player in the famous game "Go". In the scope of human medicine, approaches using these models have been made for a wide variety of experiments, such as classification (Brause, 2001) and imaging tasks (Anwar et al., 2018). In particular, NNs can be combined with both long-established and modern statistical methods (Cheng and Titterington, 1994). Especially in the field of survival analysis, current literature, such as (Wissel et al., 2021) and (Tong et al., 2020) suggests that applying NNs and PH-models

together can bring forth promising results on multi-view data.

Nevertheless, these experiments were conducted independently and based on different datasets and model implementations. In this work, we provide a summary of different NNs to investigate time-to-event data and conduct a unified benchmark. In chapter 2, we give an introduction to survival analysis, explaining the PH-model and our evaluation metric, the concordance index (c-index). The foundation of our work, the NNs, are described in chapter 3. In particular, we describe the basic idea behind and challenges of NNs, followed by described visualizations of the architectures of fully connected neural networks, autoencoders, and graph convolutional networks. Chapter 4 deals with the different pipelines we implemented, which vary in their preprocessing, feature selection, and NN methods. Conclusively, in chapter 5, we benchmark our approaches on tuned hyperparameters, compare c-indices using different plots and validate our findings on three cancer types.

Chapter 2

Survival analysis

Survival analysis is a collection of statistical approaches used to investigate **time-to-event data**. The concept describes the occurrence of events of interest in terms of two outcomes, whether or not the event takes place, as well as the time elapsed until the event occurred. If the individual does not experience the event within the time of the study, is lost to follow-up during the study, or leaves the study (before the event occurs), we refer to this event as **censored**. Nevertheless, it is necessary to consider this censored data for survival analysis, as ignoring it will lead to an "underestimation of the probability of survival beyond the fixed time-point" (Watt et al., 1996). Within the scope of survival analysis, the survivor function $S(t)$ and hazard function $h(t)$ are essential to modeling survival curves. The **survivor** function represents the probability of surviving longer than a given time t. It is defined as

$$S(t) = P(T > t), \tag{2.1}$$

where T is a random variable. The **hazard** function gives the "[...] instantaneous potential per unit time for the event to occur, given that the individual has survived up to time t" (David and Mitchel, 2012). It is defined as

$$h(t) = \lim_{\Delta t \to 0} \frac{P(t \leq T < t + \Delta t | T \geq t)}{\Delta t}. \tag{2.2}$$

The functions can be seen as two sides of a coin. On the one hand, the survivor function gives the probability of the event not occurring, on the other hand, the hazard function gives the instantaneous potential per unit time of the opposite. Note that the hazard function is not a probability because we divide by Δt (results can be greater than 1). If we know one of the two functions, we can always find the other one by applying

$$S(t) = e^{-\int_0^t h(u)\, du} \tag{2.3}$$

or

$$h(t) = -\frac{dS(t)/dt}{S(t)}. \tag{2.4}$$

Whilst the goals of survival analysis vary, this thesis focuses on examining the influence of multiple covariates on the survival of an individual. To do so, the proportional hazards-model comes in handy.

2.1 Proportional hazards-model

The **proportional hazards-model** (**PH-model**) is characterized by the hazard model formula

$$h(t, \mathbf{X}) = h_0(t) \cdot e^{\sum_{i=1}^{p} \beta_i X_i}. \tag{2.5}$$

We refer to $h(t, \mathbf{X})$ as the **risk score** and $\sum_{i=1}^{p} \beta_i X_i$ as the **risk ratio**. Furthermore, we define $\mathbf{X} = (X_0, X_1, .., X_p)$ as the predictor variables or covariates. They are the representations of input values on which we perform survival analysis on and are time-independent. $\boldsymbol{\beta} = (\beta_0, \beta_1, .., \beta_p)$ are the coefficients of our covariates and $h_0(t)$ is called the **baseline hazard**. It is the hazard if all the predictor variables are set to 0 ($\mathbf{X} = (0, 0, .., 0)$) and adds time dependency to the hazard function. It is also an unspecified function, thus making the proportional hazards-model a **semiparametric** model, consisting of a finite- and an infinite-dimensional component. In our case, the exponential expression refers to the finite component, whereas the baseline hazard refers to the infinite component. As we deal with real-life scenarios in survival analysis, the infinite component in our semiparametric model helps us represent the real world's randomness factor. The popularity of this model stems from the infinite component: It creates the possibility of making approximations that sparsely discriminate from the actual parametric model. To apply the hazard function, we must calculate the coefficient values of our hazard function $\boldsymbol{\beta}$ = (β_0, β_1,..,β_p). These can be estimated with **maximum likelihood estimation**.

In general, a likelihood function describes "the joint probability of obtaining the data actually observed on the subjects in the study as a function of the unknown parameters [...] in the model being considered" (David and Mitchel, 2012). The observed data in our case refer to the predictor variables, while the unknown parameters are represented by the $\boldsymbol{\beta}$, thus we describe the likelihood function as L($\boldsymbol{\beta}$). The likelihood formula is represented by

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{k} L_i, \tag{2.6}$$

where k denotes the amount of uncensored individuals. A specific likelihood is described as

$$L_i = \frac{h(t)_i}{\sum_{j \in R(t_i)} h(t)_j}, \tag{2.7}$$

where $R(t_i)$ denotes the risk set for individual i, which comprises all individuals in the study whose survival time is equal to or greater than that of individual i. Note that k denotes all *uncensored* individuals, thus censored individuals can be in the risk set of an uncensored individual but never appear in the numerator.

Hence, it is also called **partial** maximum likelihood estimation. To estimate β_z, we solve the following system of equations

$$\frac{\partial lnL}{\partial \beta_z} = 0, \text{ z} = 1,2,3,...,\text{p}, \tag{2.8}$$

where p is the number of coefficients. With the capability to estimate the hazard function $\hat{h}(t, \boldsymbol{X})$, it is now possible to approximate survival curves (Fig. 2.1) by transforming the said function into an estimated survivor function using

$$\hat{S}(t, \mathbf{X}) = \hat{S}_0(t)^{e^{\sum_{i=1}^{p} \hat{\beta}_i X_i}}. \tag{2.9}$$

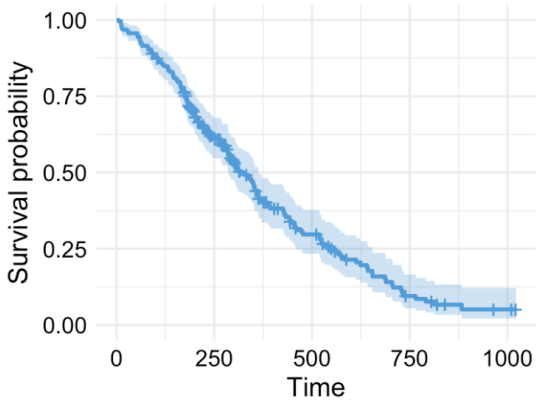


Figure 2.1: *Survival graph of a patient calculated with Cox regression*

Lastly, the PH-model is called **proportional** because it requires that "the hazard for one individual is proportional to the hazard for any other individual, where the proportionality constant is independent of time" (David and Mitchel, 2012) (**PH assumption**). Mathematically, this implies that the quotient of the estimated hazard functions for two individuals is independent of time t, as the baseline hazards cancel themselves out:

$$\frac{\hat{h}(t, \boldsymbol{X^*})}{\hat{h}(t, \boldsymbol{X})} = e^{\sum_{i=1}^{p} \hat{\beta}_i (X_i^* - X_i)} = \hat{\theta}. \tag{2.10}$$

Thus, both hazards are proportional to one another and coincide with the PH assumption. θ can be seen as a proportionality constant between the two hazard functions. This ratio is also called the **hazard ratio**.

2.2 Evaluation method : Harrell's c-index

Harrell's concordance index (c-index) is a common way to evaluate risk models in survival analysis, where data samples may be censored (Harrell et al., 1982).

It is represented by

$$c = \frac{\#concordantpairs}{\#concordantpairs + \#discordantpairs}, \tag{2.11}$$

or mathematically expressed

$$c = \frac{\sum_{i \neq j} 1\{\eta_i < \eta_j\} 1\{T_i < T_j\} e_j}{\sum_{i \neq j} 1\{T_i < T_j\} e_j}, \tag{2.12}$$

where

- T represents the individual's duration (time to event or time until censorship),
- η_i is the risk score, which corresponds to the estimated hazard function output $\hat{h}(t_i, \boldsymbol{X})$ at a fixed time t_i,
- t_i represents the duration value for patient i,
- e is the event indicator (e = 0 censored, e = 1 not censored),
- 1 is the identity matrix,
- i and j represent an arbitrary pair of individuals.

If both T_i and T_j are not censored, the pair (i,j) is a concordant pair if $\eta_i > \eta_j$ and $T_i < T_j$ and discordant if $\eta_i > \eta_j$ and $T_i > T_j$.
If both T_i and T_j are censored, we dismiss this pair, as we have no follow-up on whether or when the individuals experienced the event.
Suppose one of the variables, say T_j, is censored, while the other variable, T_i, is not censored.
If $T_j < T_i$, we do not know if the event had occurred for individual j before it did for i. Thus we dismiss this pair.
If $T_j > T_i$, we know that the event has occurred for individual i before it may happen for individual j ; (i,j) is a concordant pair if $\eta_i > \eta_j$ and discordant if $\eta_i < \eta_j$.

Chapter 3

Neural networks

Neural networks (NN) aim to mimic the way biological neurons interact with each other (Fig. 3.1). The primary goal is to train or fit a neural net to accomplish a specific task using some data and, subsequently, enable it to make accurate predictions on new, unknown data. We refer to the two data sets as *train* and *test* data, respectively. Training and testing a NN on the same data is a methodical error and would lead to learning the representation of given data perfectly, which results in the inability to generalize well to new data.

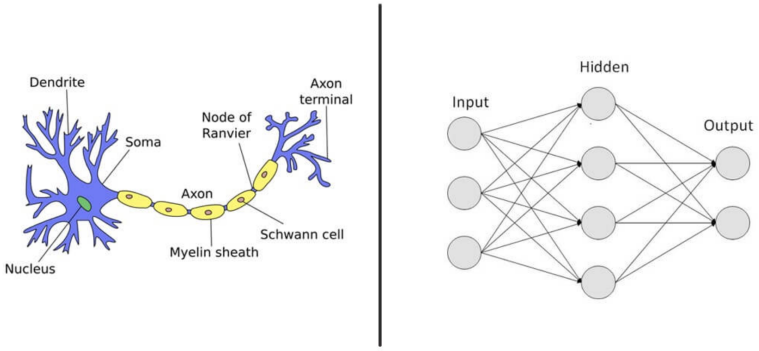


Figure 3.1: *Comparison between biological and artifical neural networks. The blue structures are simulated with layers of neurons and the axon is the connectivity between them, visualized with arrows.*

Even though NNs can get vastly complicated, the foundation of each, the **perceptron**, (Fig. 3.2), is rather simple. Given inputs $x_1, x_2, ..., x_n$,weights $\omega_1, \omega_2, ..., \omega_n$ and bias b, the predicted output $\hat{y}$ of the neuron can be described as $\hat{y} = \sum_{i=1...n}(x_i \cdot \omega_i) + b$.

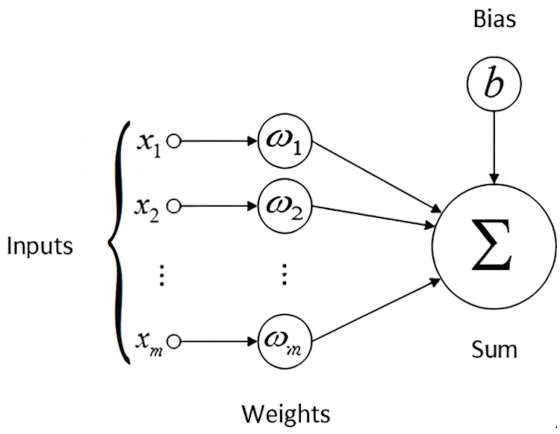


Figure 3.2: *Example of a perceptron.*

In actual use cases, we almost always have multiple neurons per layer, each with its own weights. Ordinarily, this is defined in vector notation. Assuming our layer has m neurons, we get the following:

$$\hat{\mathbf{y}} = \mathbf{W} \times \mathbf{x} + \mathbf{b} = \begin{pmatrix} \omega_{1,1}, \omega_{1,2}, ..., \omega_{1,n} \\ \omega_{2,1}, \omega_{2,2}, ..., \omega_{2,n} \\ ... \\ \omega_{m,1}, \omega_{m,2}, ..., \omega_{m,n} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ .. \\ b_m \end{pmatrix}, \tag{3.1}$$

where $\mathbf{W}$ is the weight matrix and each row i represents the weights of neuron i. Such an equation calculates the output of a layer k based on the output of a previous layer $k-1$ and is referred to as **propagation rule**.

Currently, the described model represents a basic linear regression. To introduce non-linearity, **activation functions** ϕ are applied to each neuron. They define the output of a neuron given its' input and vary depending on the task. For example, the *sigmoid* activation function, $\phi(x) = \frac{1}{1+e^{-x}}$, is frequently used in binary classification tasks, as it maps the output to [0,1]. The most commonly used activation function is the *rectified linear unit* (Nair and Hinton, 2010), or *ReLU* for short. Negative outputs are mapped to 0, and positive values stay the same. Mathematically, it is expressed as $\phi(x) = max(0, x)$, where x is the input into the activation function. Using this activation function can lead to so-called dead neurons, which occur when most of the inputs for the *ReLU* neurons are negative. As *ReLU* maps these inputs to 0, the output becomes 0, which ultimately leads to parts of the NN becoming inactive. To not completely ignore negative inputs, the *parametric rectified linear unit*, or *PReLU*, can be

used. It is described by $\phi(x,k) = max(0,x) + k \cdot min(0,x)$. The output function then becomes

$$\hat{\mathbf{y}} = \phi(\mathbf{W} \times \mathbf{x} + \mathbf{b}). \tag{3.2}$$

Note that despite the differences, the parameters of the model, i.e., weights and bias, maintain similar functionalities to those of linear regression: the weights determine the shape of the function, while the bias can shift the activation function to the left or right.

Forwarding data through an entire network, which we refer to as forward pass, will probably result in a huge dissimilarity between predicted output $\hat{y}$ and wanted output y if the parameters of the network are selected at random. This loss is described by using a **loss**, sometimes also referred to as **cost**, **function**, for example the *mean squared error (MSE)* :

$$\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2, \tag{3.3}$$

where n refers to the number of samples. To minimize the loss function, it is necessary to adjust all the initially randomly selected parameters in the NN such that they approach a minimum value. We do so by deriving each parameter, i.e., finding the gradient of all parameters. As this process begins at the output and propagates towards the input layer to calculate the derivative of each parameter, it is commonly referred to as **backpropagation**. The gradient for parameter μ_i points to the direction of its steepest increase. Thus, we take the negative gradient to move towards the minimum of the loss function, which explains why the **optimization algorithm** for NNs is called *gradient descent*. Computing the gradient for the whole training dataset at once may lead to high computational costs. Therefore, most NNs train in batches, which are disjoint subsets of data. For batches of size k, in each batch iteration, each parameter μ_i is updated by calculating

$$\mu_{i\_new} = \mu_{i\_old} - \eta * avg(\sum_{j=1}^{k} grad(\mu_{i\_old,j})), \tag{3.4}$$

where η is the learning rate, i.e., learning step and $avg(\sum_{j=1}^{k} grad(\mu_{i\_old,j}))$ refers to the mean gradient of the current batch for parameter μ_i (Fig. 3.3). Note that the learning rate's value can drastically change the NN's performance. If it is too big, the local minimum may never be reached as it bounces back and forth between the convex function for gradient descent, too small, and finding the local minimum takes up a considerable amount of time. To reduce the optimization algorithms' time complexity even more, *stochastic* gradient descent (Amari, 1993) is a suitable method, which only considers batches of size one for weight updates.

In general, the neural network is unlikely to converge to the global or local minimum of the loss function after a single pass through the entire training dataset. Consequently, we train it for a number of **epochs**. Nonetheless, a

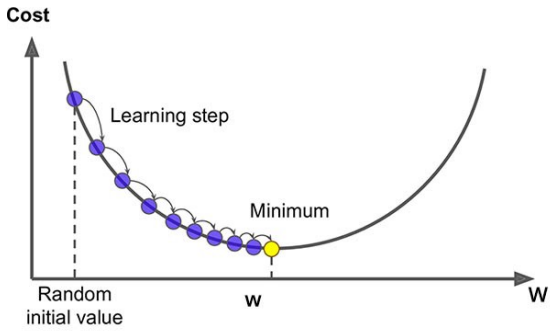
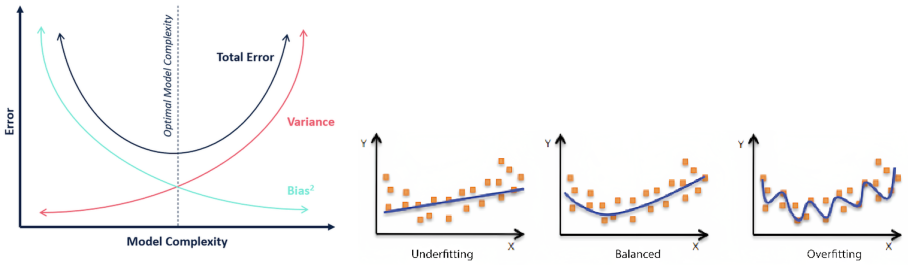


Figure 3.3: *Gradient descent of a single parameter. With each batch, we move a step closer to the minimum of our loss function.*

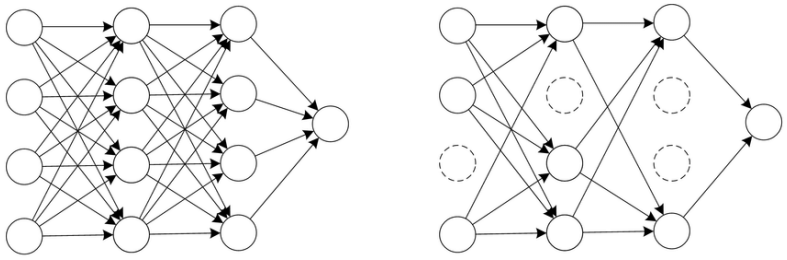
NN that minimizes the loss after a set of training epochs might still return inaccurate predictions on the test data. One reason for this could be that the model is **overfitted**. Such a model has parameters fitted in a way that each training sample, even if they are exceedingly dissimilar (high variance), is accurately predicted (low bias). In contrast, its' counterpart, an underfitted model, describes the opposite (low variance, high bias). A well-trained NN finds the sweet spot in the **variance-bias tradeoff** and brings forth good results in both train and test data (Fig. 3.4).



(a) (b)

Figure 3.4: (a): *The variance-bias tradeoff. A model that adjusts its' parameters towards the training data ideally has high complexity and overfits, whereas not paying attention to the train set during the model fit leads to low complexity and an underfit.* (b): *Examples for underfitted, balanced and overfitted models. Coordinates (x,y) refer to coordinates of two-dimensional samples.*

In order to address overfitting during NN training, nodes are randomly deactivated or dropped out in each batch based on a certain probability. Consequently, these nodes cannot shift their parameters towards perfectly fitting train data (Fig. 3.5). With this method, known as **dropout**, we can simulate different network configurations without defining them (Srivastava et al., 2014).



(a) Standard Neural Network (b) Network after Dropout

Figure 3.5: *Dropping out random nodes lead to training different NNs in terms of nodes per layer.*

Another possibility is to add a penalty term to the loss function :

$$\lambda \cdot \sum_{j=1}^{n} \beta_j^2. \tag{3.5}$$

where λ is the regularization rate, β_j parameter j, and n the number of total parameters in the NN. If a model starts to overfit, parameters tend to have highly specific values so that each training sample can accurately be predicted (Fig. 3.6). Therefore, these parameters are described by large absolute values. Since the penalty term essentially is the sum over all squared parameters, these large values will be penalized, thus preventing the model from overfitting. This method is known as **L2**, i.e **euclidean**, **regularization**.

Another significant issue in NNs with many layers is that the distribution of inputs to layers deep within may change after each batch, resulting in a chase toward a forever-moving target. Normalizing the batch for each layer separately may make the learning process more stable and reduce the needed number of training epochs for a good model fit. This is known as **batch normalization** (Ioffe and Szegedy, 2015). Given a batch $\mathbf{x} = (x_1, x_2, ..., x_n)$, the output of each layer separately is scaled with

$$y_i = \frac{x_i - \mathbb{E}(\mathbf{x})}{\sqrt{Var(\mathbf{x}) + \epsilon}}, \tag{3.6}$$

where $\mathbb{E}(\mathbf{x})$ is the expected value of the batch, $Var(\mathbf{x})$ the variance of the batch and ϵ a small value for numerical stability.

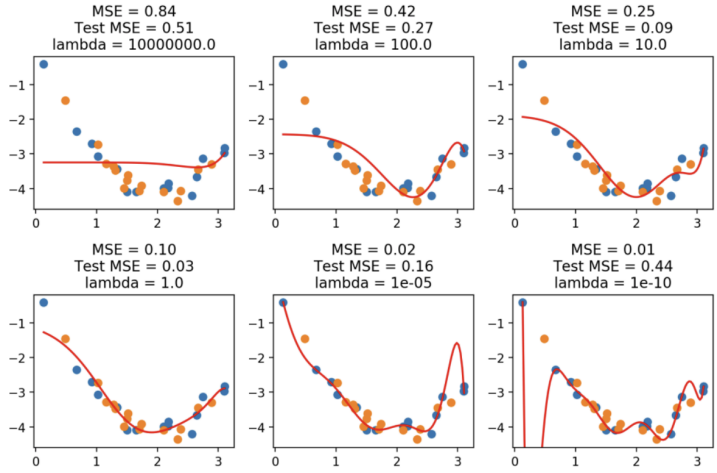


Figure 3.6: *Different regularization rates produce different model fits. If the value of λ is too great, the model underfits, too small and it overfits. Blue dots represent training samples, orange dots test samples.*

NNs nowadays have multiple layers between the first input and final output to process the immense amounts of data made accessible in the 21st century. We define the first layer as the input layer, the last layer as the output layer, and the layers in between as hidden layers. If every neuron in layer i is connected to every neuron in layer $i+1$ for the whole network, it is referred to as **fully connected neural network** (**FCNN**) (Fig. 3.7).

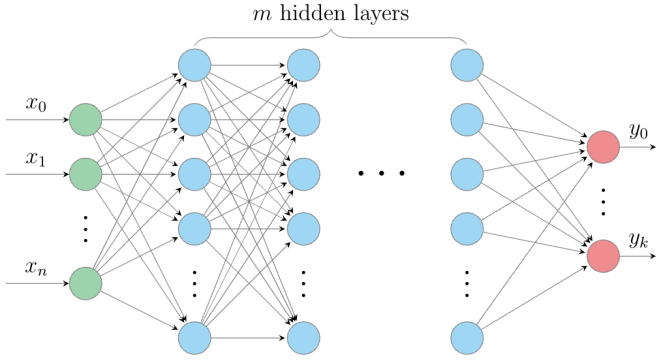


Figure 3.7: *Topology of a fully connected neural network, containing an input layer, m hidden layers and an output layer.*

By choosing a smaller dimension for each layer l_{i+1} compared to l_i, we get increasingly abstract representations of our original data, which is excellent for, e.g., classification tasks that result in a relatively small number of output values. Another possibility is first to reduce layer dimensions, resulting in a latent representation of input values and then mirroring the layers to reproduce the original input. This particular type of FCNN is called **autoencoder** (**AE**). The architecture consists of an encoder, a layer representing the compressed representation of inputs, i.e., bottleneck layer, and a decoder (Fig. 3.8). The encoder compresses data into the bottleneck layer and the decoder, a reflection of the encoder, learns to reconstruct the original data while minimizing the reconstruction error, i.e., MSE loss. Common use cases for AEs are dimensionality reduction and image manipulation, such as denoising.

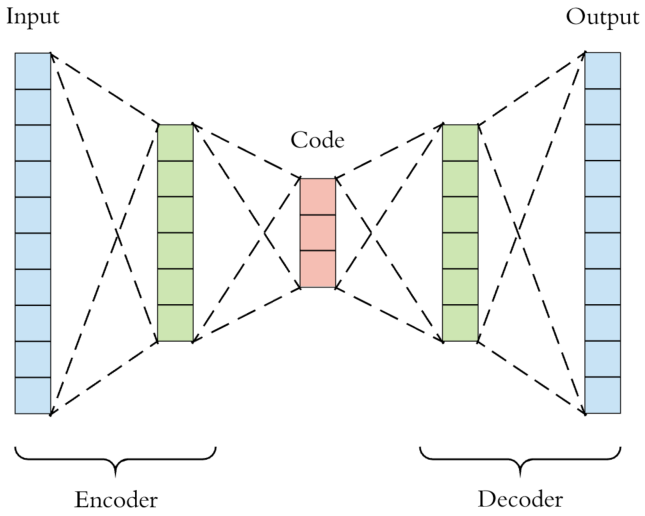


Figure 3.8: *A typical autoencoder. Input values are compressed with encoding layers and decompressed with decoding layers.*

Real-world data sets often consist of features that may be interconnected with one another. To model these relationships, we use graphs, where features of samples are represented as nodes and connections between them as edges. **Graph convolutional networks** (**GCN**) are a subcategory of graph neural networks, which can be used to process such network inputs. The initial data is divided into two parts: a feature matrix $\mathbf{H}$ of size $\mathbf{N} \times \mathbf{F}$, where $\mathbf{N}$ represents the number of nodes and $\mathbf{F}$ the number of features per node, and an adjacency matrix $\mathbf{A}$ of size $\mathbf{N} \times \mathbf{N}$, which represents graph connectivity. The forward pass of a GCN and FCNN are highly similar (Fig. 3.9). Assuming we are currently calculating the input value for layer $k+1$, using formula 3.2 without bias for simplification, we get

$$\mathbf{y}_{k+1} = \phi(\mathbf{W_k} \times \mathbf{y_k}). \tag{3.7}$$

The propagation rule for the GCN is

$$\mathbf{H}^{k+1} = \phi(\hat{\mathbf{A}}\mathbf{H}^{k}\mathbf{W}^{k}), \tag{3.8}$$

where $\mathbf{H^k}$ is the input, $\mathbf{W^k}$ the weight matrix for layer k and $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{1}$ with $\mathbf{1}$ being the identity matrix. The reason for using $\hat{\mathbf{A}}$ instead of $\mathbf{A}$ is to ensure the existence of self-loops. For each node a in the graph, $\hat{\mathbf{A}}\mathbf{H^{k-1}}$ sums up the feature vectors of all neighboring nodes, including node a itself. Since adjacency matrices are generally not normalized, simply multiplying them with $\hat{\mathbf{A}}$ would result in a loss of the original scale. Therefore, following the proposal of Kipfing and Welling, $\hat{\mathbf{A}}$ is symmetrically normalized (Kipf and Welling, 2016). This results in the propagation rule

$$\mathbf{H}^{k+1} = \phi(\tilde{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\mathbf{H}^{k}\mathbf{W}^{k}), \tag{3.9}$$

where $\tilde{\mathbf{D}}_{ii} = \sum_j \hat{A}_{ij}$. To reduce the chance of overfitting, graphs can be coarsed with pooling layers. For example, features can be summed up or averaged for each node.

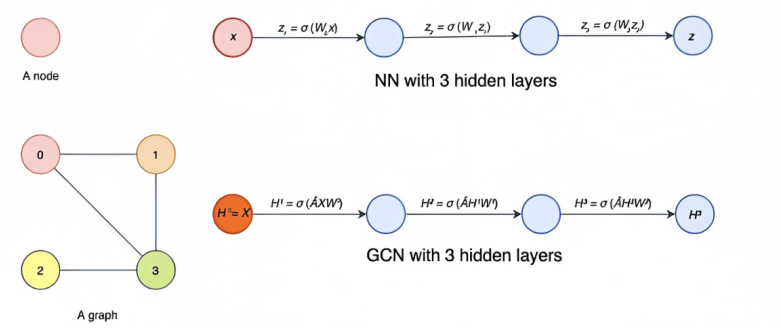


Figure 3.9: *Simplified propagation rule of a GCN compared to one of a FCNN. While the NN takes a single value (node) as an input, the GCN receives a network of values. σ is an arbitrary activation function.*

Chapter 4

Model outline

In the following, we will describe the general pipeline for the different models we seek to compare in a unified benchmark. This includes a description of what the initial data looks like and our pre-processing methods, different scaling approaches, and three feature selection methods. Finally, we illustrate our NN methods and detail how we integrate them with the PH-model to train the networks on survival analysis.

4.1 Data reading

We train, validate and test our models on data from 3 different cancer types, i.e., KIRC, LUAD, and LIHC, provided by the TCGA database. Per cancer, we have 20531 mRNA and 20122 DNA features in addition to the duration and event values for each sample, respectively. Samples with missing or negative duration values are dropped as they cannot be used for survival modeling. Additionally, for each view, we examine whether there are features that solely contain missing values and remove them. Finally, we store data for each cancer type separated in a complete data set, where each row represents a sample and columns represent data across all views, duration and event value, a cumulative sum of the number of features across all views, the name of views included in the subset and their feature names.

4.2 Data split

We split the data into a train (80%) and test (20%) set using `train_test_split`. While training the NN, parameters will be adjusted to minimize the loss function as described in chapter 3. Nevertheless, we also need to tune hyperparameter settings such as dropout probability or learning rate, which are established prior to the training process. Optimizing both these parameter types on the train data split makes the model more vulnerable to overfitting. Thus, we apply **5-fold cross-validation** with `StratifiedKFold`, which splits the train set into five sets, each of which comprises four folds for training and one fold for validating, with the latter being used to evaluate different hyperparameters and the performance of the model (Fig. 4.1). We stratify all splits by non-censored events (Table 4.1).

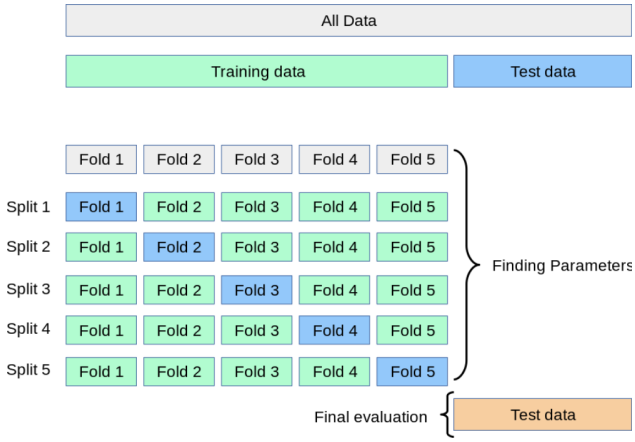


Figure 4.1: *5-fold cross-validation.*

Cancer type	# samples overall	# samples in split	# event = 1 in split
KIRC	742	475, 118, 149	174, 43, 54
LUAD	590	377, 95, 118	140, 36, 44
LIHC	437	279, 70, 88	110, 27, 35

Table 4.1: *Overview of the three cancer types and their number of samples with respect to our data split.*

4.3 Data scaling

The different features in our data may vary in degrees of magnitude or range. For example, if the value of one feature were measured in a range between 1 and 10 and another between 100 and 1000, our models would disproportionately emphasize the high values, resulting in biased results. Therefore, we propose four different methods to shift features x of each view and for each sample i so that they are scaled in the same way.

Standard scaling

We rescale each feature by removing the mean and shifting the data to unit variance:

$$x_{i,scaled} = \frac{x_i - mean(x)}{std(x)}. \tag{4.1}$$

Scaled values will have a mean and standard deviation of 0 and 1, respectively.

Minimum maximum scaling (MinMax)

We rescale the features so that they are between 0 and 1 by applying

$$x_{i,scaled} = \frac{x_i - min(x)}{max(x) - min(x)}. \tag{4.2}$$

Minimum values will be rescaled to 0, and maximum values to 1.

Maximum absolute value scaling (MaxAbs)

The maximum absolute value method scales each feature by dividing it by the maximum absolute feature value across all samples:

$$x_{i,scaled} = \frac{x_i}{maxabs(x)}. \tag{4.3}$$

Values will be scaled between -1 and 1.

Robust Scaling

Each of the previously mentioned preprocessing methods suffers from outliers. We use robust scaling to lose the high sensitivity towards the presence of outliers by removing the median and scaling features based on the interquartile range (iqr):

$$x_{i,scaled} = \frac{x_i - median(x)}{iqr(x)}. \tag{4.4}$$

Scaled values will have a median and iqr of 0 and 1, respectively.

For all preprocessing types, we calculate the measurements such as mean or maximum on the current train split and use these to transform the train and validation split as well as the test set, as it otherwise would lead to biased results. We make use of `StandardScaler`, `MinMaxScaler`, `RobustScaler` and `MaxAbsScaler`. Additionally, we remove views that have less than 10% feature values that differ from 0 after scaling.

4.4 Feature selection

Training our NNs on the entirety of over 20 thousand features for each view would entail lengthy training epochs and high memory consumption. In practical use cases, redundant or irrelevant features such as those with zero variance may even lead to an more inaccurate model. Feature selection can be defined as choosing a minimal subset of features, which used as the input to the NN will produce an optimally predictive model (Tsamardinos and Aliferis, 2003). We propose two feature selection methods and one graph structure tailored for the GCN.

4.4.1 Principal components analysis

We employ **principal component analysis (PCA)**, a linear dimensionality reduction technique that utilizes singular value decomposition (SVD) to project the data onto a lower-dimensional space. We perform SVD on our data matrix X of size $\mathbf{N} \times \mathbf{F}$, where $\mathbf{N}$ is the number of samples and $\mathbf{F}$ the number of features. This yields $X = U\Sigma V^T$, where the columns of $U\Sigma$ contain the principal components (PC) (Wall et al., 2003). Afterward, we recast the data along the PC axes. We apply principal component analysis with `PCA`. For each view, we can select how many PCs should be computed. Note that the minimum of samples and number of features for the current train split determines the maximum number of components. For each split, we compute the PCs on the train split and recast the train and validation split as well as the test set along these axes.

4.4.2 Empirical variance

The **empirical variance** of a feature can be calculated with:

$$\frac{\sum_{i=1}^{n}(x_i - \mu)}{n-1}, \tag{4.5}$$

where n is the number of samples, μ the mean of the feature calculated across all samples and x_i the feature value for sample i. Low variance implies that the feature values for a specific feature are similar, while high variance indicates the opposite, i.e., that the values are diverse. Features with low variance only contribute to model complexity without providing predictive power. We select the top k features with the highest variance from the training split for each view and split and subsequently choose these same features for the validation split and test set. Note that this feature selection method is ineffective when we standardize data beforehand as this scales the variance of each feature to 1.

4.4.3 Protein-protein-interaction

For each sample, we create a **protein-protein-interaction** (PPI) graph, in which nodes represent proteins and edges represent protein-protein interactions.

Each node can contain feature values of mRNA and DNA data, given that there is a relationship between the protein and the feature. When we use this graph as an input to the GCN, we need to make sure that each node has the same amount of values assigned to it. In the case of missing protein-feature links, we propose either augmentation by median or zero values or deleting those proteins. Protein data was downloaded from the STRING database (Mering et al., 2003). Note that protein-protein interaction networks do not serve as a feature selection method per se since we select all possible features with protein mappings.

4.5 Models

4.5.1 Multi-view FCNN

We train an FCNN for each view separately (**multi-view FCNN**, **MV_FCNN**). The outputs are then concatenated and passed to a final layer, which compresses dimensions to a one-dimensional value before passing it to the PH-model.

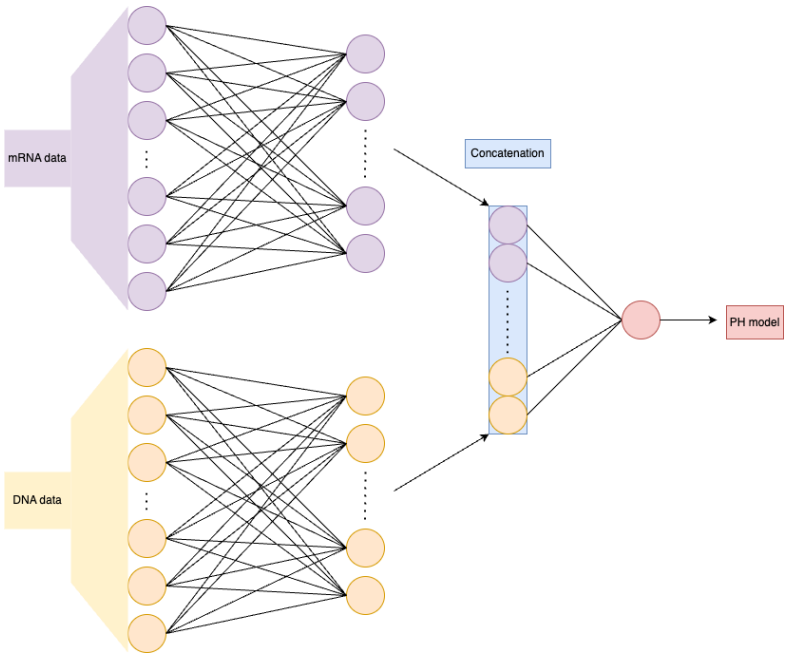


Figure 4.2: *Architecture of the multi-view FCNN.*

4.5.2 AE

We train an AE for each view separately (**single AE**, **SAE**). For the integration of the bottleneck values, we propose five different methods. The latent representations are concatenated (**C**), element-wise averaged (**EA**) or maximized (**EM**) (across different views) or overall averaged (**OA**) or maximized (**OM**). The bottleneck of view i can either solely be reconstructed with the associated decoder or with an additional decoder of a view $j \neq i$ (**cross decoding environment**) (Tong et al., 2020). The total MSE loss then comprises both losses. Furthermore, the integrated output can be passed to yet another AE (**hierarchical AE**,**HAE**) (Wissel et al., 2021) before finally passing data to a single-view FCNN. The single-view FCNN has the same architecture as the multi-view FCNN but takes only one view, in this case, the integrated latent representation, as an input. Note that the latter AE may be used without a decoder (**second decoder environment**) and can only use OA or OM as an integration method, as we already integrated all views in the former AE.

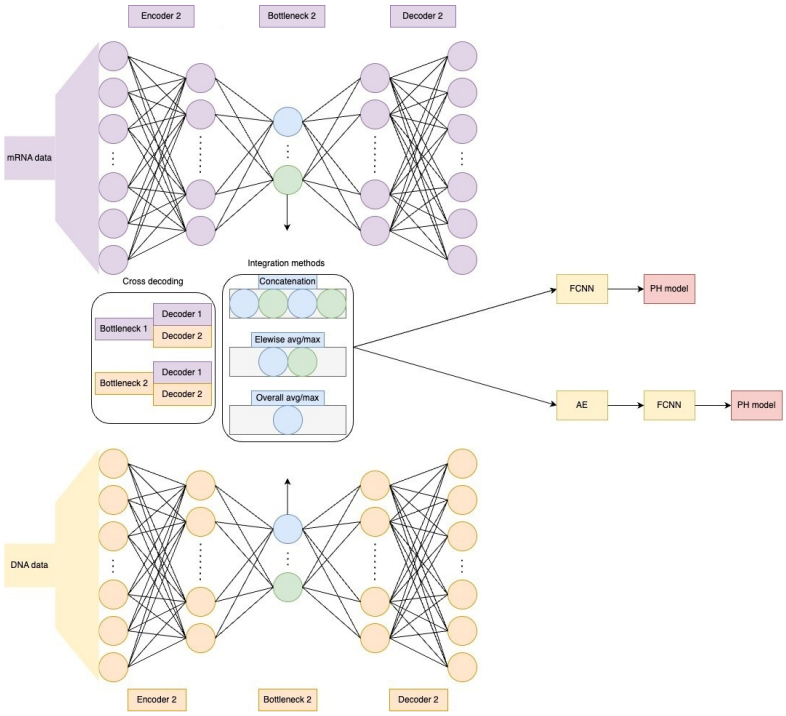


Figure 4.3: *Architecture of the AE.*

4.5.3 GCN

The GCN consists of two convolutional layers (Morris et al., 2019) with a self-attention pooling layer (Lee et al., 2019) in between. Then we take the maximum feature value for each node (protein), concatenate them, and pass it to a single-view FCNN.

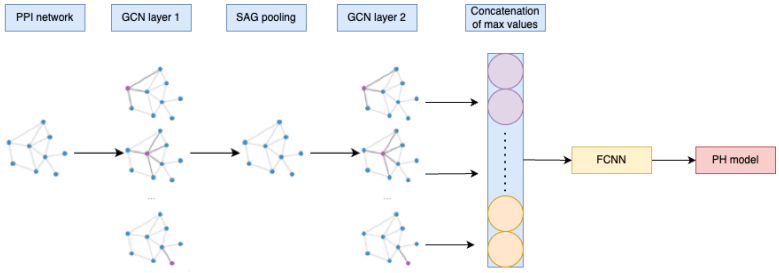


Figure 4.4: *Architecture of the GCN.*

4.5.4 Integration of PH-model

To simulate the partial maximum likelihood calculation used for the PH-model, we train our NNs on the **negative partial log-likelihood** loss (Katzman et al., 2018). Therefore, the one-dimensional output of the FCNN becomes the risk ratio, is passed to the PH-model and finally used to calculate the estimated hazard function output $\hat{h}(t, \boldsymbol{X})$, i.e., risk score. Note that for AEs, we train the pipeline with a linear combination of MSE and negative partial log-likelihood losses.

4.5.4.1 Negative partial log-likelihood

The **negative partial log-likelihood** is described by

$$l(\boldsymbol{\beta}) = -\frac{1}{N_{E=1}} \sum_{i:E_i=1} \left(\hat{z}_{\boldsymbol{\beta}}(x_i) - log \sum_{j \in R(t_i)} e^{\hat{z}_{\boldsymbol{\beta}}(x_j)} \right), \tag{4.6}$$

where $\boldsymbol{\beta} = (\beta_0, \beta_1, ..., \beta_p)$ represents the parameters of the NN, $N_{E=1}$ is the number of uncensored samples and $\hat{z}_{\boldsymbol{\beta}}(x_i)$ is the risk ratio for sample i. We refer to negative partial log-likelihood loss as survival loss (SURV). It essentially is the logarithm of the likelihood function described in section 2.1, divided by the number of non censored patients. It is important to note that *minimizing* the *negative* likelihood is equivalent to *maximizing* the *positive* likelihood. As neural networks are designed to minimize loss, we use the *negative* partial log-likelihood loss for training.

Chapter 5

Benchmark

In the following, we conduct a unified benchmark on three distinct cancer types, evaluating the performance of our various models. We optimize the hyperparameters of each model with fine-tuning and then compare these models based on their test c-index scores for each fold.

5.1 Training and tuning

Training

We train the NNs with 5-fold cross-validation on the four scaling methods described in section 4.3 and either choose 100 PC components or the top 2000 features with the highest variance for both mRNA and DNA data. For each fold, we run 100 epochs with early stopping when the validation loss stops improving. We use Adam (Kingma and Ba, 2014) as the optimization algorithm, and the validation folds to evaluate the model configuration.

Tuning

The performance of a NN depends heavily on the right choice of hyperparameters. Therefore, it is essential to fine-tune them to solve a given task. For each NN, we have a set of basic hyperparameters (Table 5.1) as well as a set specifically tailored for the different neural nets (Table 5.2). We run 100 different hyperparameter trials on each validation fold using the tree-structured Parzen estimator (Bergstra et al., 2011) for value suggestions, aiming to maximize the c-index. We only run 30 trials for the GCN as it is extremely time expensive, which we analyze in section 5.2.1.6. After tuning, we evaluate the model on the test set using the observed optimal hyperparameters, obtaining c-indices for each fold.

Hyperparameter	**Value range**
Batch size	8,16,32,64,128,256
Training epochs	100
Optimization algorithm	Adam
L2 regularization bool	True,False
L2 regularization rate	$[10^{-6}, 10^{-3}]$
Learning rate	$[10^{-5}, 10^{-1}]$
Dropout bool	True,False
Dropout probability	$[0, 1]$
Dropout per layer	True,False
Batch normalization bool	True,False
Batch normalization per layer	True,False
PReLU rate	$[0, 1]$
Activation functions per layer	None, Sigmoid, ReLU, PReLU

Table 5.1: *Basic hyperparameters to tune. The value range describes the different values the associated hyperparameter can adopt. Single values indicate fixed hyperparameters.*

Hyperparameter	**Value range**
Layer 1 dimensions FCNN	32,64,96,128
Layer 2 dimensions FCNN	8,16,32
Layer 1 dimensions AE 1	32,64,96,128
Layer 2 dimensions AE 1	8,16,32
Layer 1 dimensions FCNN	Integrated element size of AE 1
Layer 2 dimensions FCNN	8 - integrated element size of AE 1
Loss ratio (SURV/MSE)	$x, y \in [0, 1]$ so that $sum(x, y) = 1$
Layer 1 dimensions AE 1	32,64,96,128
Layer 2 dimensions AE 1	8,16,32
Layer 1 dimensions AE 2	Integrated element size of AE 1
Layer 2 dimensions AE 2	8 - integrated element size of AE 1
Layer 1 dimensions FCNN	1
Loss ratio (SURV/MSE1/MSE2)	$x, y, z \in [0, 1]$ such that $sum(x, y, z) = 1$
Layer 1 dimensions GCN	1,2,4,8
Layer 2 dimensions GCN	1,2,4,8
Layer 1 dimensions FCNN	4,8
Layer 2 dimensions FCNN	1,2
Pooling ratio	$[0.1, 1]$

Table 5.2: *Specific hyperparameters to tune for FCNN, SAE, HAE, and GCN. For HAE, we use a single layer for the FCNN, as the integration method of the second AE is overall averaging/maximizing, which returns just a one-dimensional value. Additionally, if we use no decoder in the second AE, MSE2 becomes zero.*

5.2 Results

We will start by analyzing KIRC in-depth. We choose MinMax scaling and compare various SAE and HAE models with themselves as well as with each other, followed by an examination of the GCN model's performance. As the baseline model, we use the multi-view FCNN and analyze results based on PCA and variance feature selection for mRNA and DNA data. Additionally, we evaluate different scaling methods, hyperparameter importance, and time complexity. Afterward, we compare our in-depth results to 2 additional cancers, LUAD and LIHC, validating and contrasting our findings.

5.2.1 In-depth analysis of KIRC

5.2.1.1 Comparisons of SAE models

We find that the multi-view FCNN slightly outperforms every SAE model (Table 5.3). When comparing the SAE models to one another, concat integrations yield the best results, followed by the element-wise methods. Overall maximizing performed worst in our tests, yet averaging is highly competitive with the other models (Fig. 5.1). We propose that the variance of high dimensional data compressed into a singular value is substantially better represented with averaging rather than maximizing.

Models	C-index	C-index cross
MV_FCNN	**0.728 ± 0.008**	**0.728 ± 0.008**
SAE_C	0.725 ± 0.010	0.724 ± 0.008
SAE_EA	0.724 ± 0.008	0.717 ± 0.009
SAE_EM	0.722 ± 0.005	0.719 ± 0.012
SAE_OA	0.716 ± 0.009	0.722 ± 0.005
SAE_OM	0.710 ± 0.004	0.705 ± 0.008

Table 5.3: *5-fold averaged c-indices of SAE models and multi-view FCNN based on PCA feature selection. Note that the multi-view FCNN has no cross setting, thus the same result for both columns. Best results are marked in bold.*

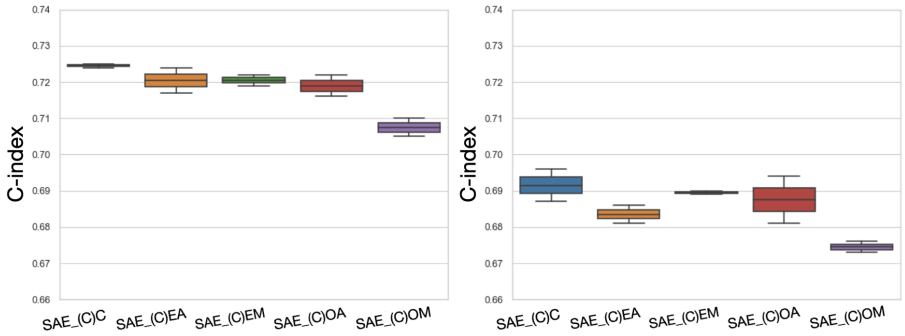


Figure 5.1: *Comparing SAEs C, EA, EM, OA, OM on aggregated cross and non-cross scores for PCA (left) and variance (right).*

To examine the relationship between cross and non-cross decoding more closely, we compare the environments to each other (Fig. 5.2). We find non-cross models to be slightly better performing. More specifically, the difference is subtle for PCA feature selection, whereas non-cross environments thrive when selecting the top 2000 features with the highest variance.

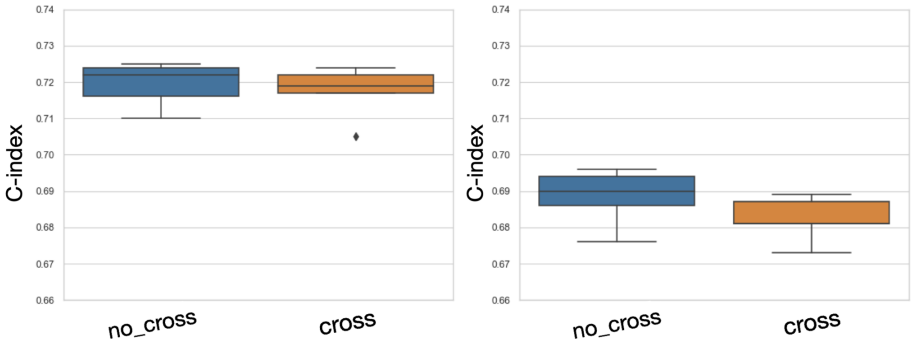


Figure 5.2: *Comparing non-cross and cross scores aggregated over each SAE model for PCA (left) and variance (right).*

5.2.1.2 Comparisons of HAE models

Similar to our SAE model benchmark, the multi-view FCNN brings forth slightly better results than HAE models except for the non-cross and non-decoding environment, where element-wise averaging coupled with overall maximizing significantly outperforms every other model (Table 5.4). Subsequently, we sought to investigate if the additional AE affected the most promising SAE model settings (Fig. 5.3). We find that element-wise averaging performs better than element-wise maximizing in every case and performs similarly to the concatenation setting. Comparing results to the SAE setting, we find that only element-wise averaging benefits from a second AE.

Models	C-index	C-index cross	C-index decode	C-index cross,decode
MV_FCNN	0.728 ± 0.008	**0.728 ± 0.008**	**0.728 ± 0.008**	**0.728 ± 0.008**
HAE_COA	0.724 ± 0.009	0.715 ± 0.007	0.727 ± 0.014	0.721 ± 0.011
HAE_COM	0.725 ± 0.007	0.725 ± 0.007	0.722 ± 0.008	0.715 ± 0.009
HAE_EAOA	0.725 ± 0.007	0.724 ± 0.007	0.722 ± 0.006	0.726 ± 0.008
HAE_EAOM	**0.733 ± 0.009**	0.726 ± 0.009	0.719 ± 0.015	0.720 ± 0.013
HAE_EMOA	0.719 ± 0.010	0.719 ± 0.011	0.718 ± 0.007	0.720 ± 0.008
HAE_EMOM	0.712 ± 0.011	0.723 ± 0.014	0.714 ± 0.006	0.712 ± 0.013

Table 5.4: *5-fold averaged c-indices of HAE models and multi-view FCNN based on PCA feature selection. Note that the multi-view FCNN has no cross/decoding setting, thus the same result for each column. Best results are marked in bold.*

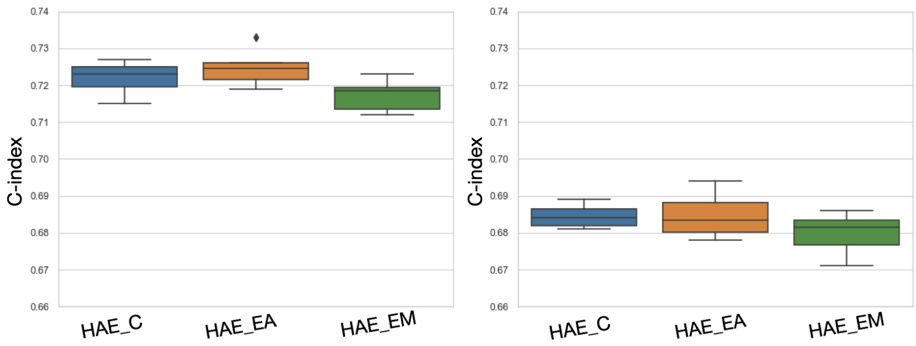


Figure 5.3: *Comparing concat, element-wise average, and element-wise maximum scores aggregated over each HAE model for PCA (left) and variance (right).*

Comparable to the results we found in the SAE analysis, overall averaging in the second AE performed better than overall maximizing for variance feature selection, strengthening our assumption (Fig. 5.4).

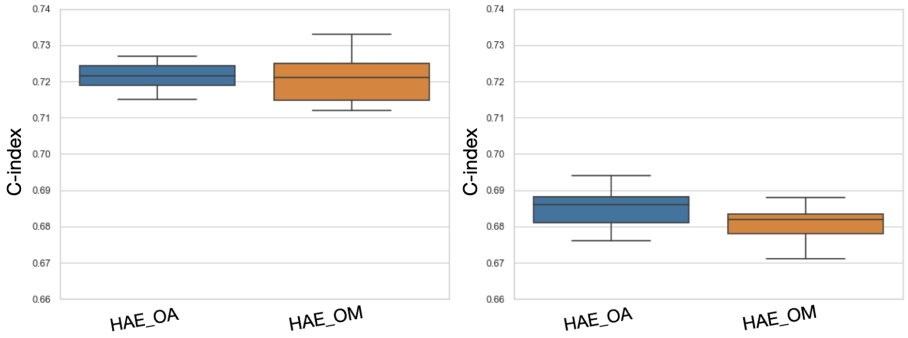


Figure 5.4: *Comparing the second AEs overall average and overall maximum setting aggregated over each HAE model for PCA (left) and variance (right).*

Finally, we were interested in analyzing how cross and decoding settings affect model performance. Upon comparing the results between non-cross and cross decoding environments, we observe highly similar results, thus proposing that this environment does not affect HAE models (Fig. 5.5). In contrast, omitting the decoder for the second AE boosts performance, especially for variance feature selection (Fig. 5.6).

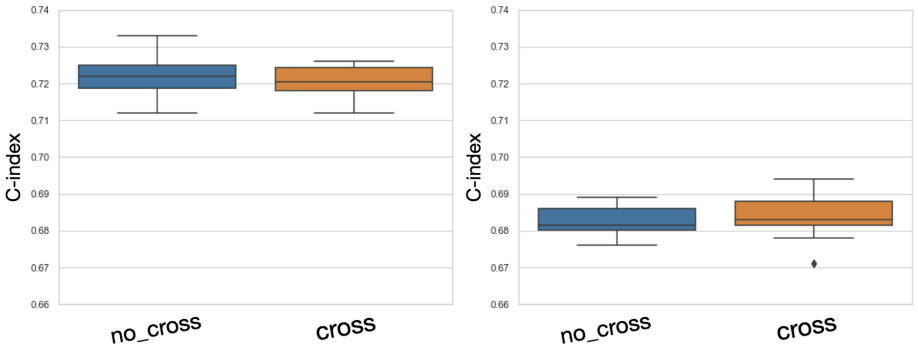


Figure 5.5: *Comparing non-cross and cross scores aggregated over each HAE model for PCA (left) and variance (right).*

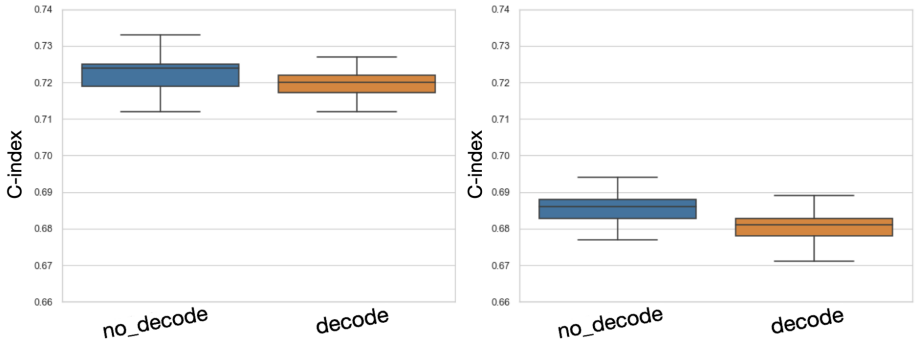


Figure 5.6: *Comparing scores when using or omitting a second decoder aggregated over each HAE model for PCA (left) and variance (right).*

5.2.1.3 GCN performance

While examining the performance of our final model, we found it to be significantly worse than every other one, independent of the proposed manipulation methods (Table 5.5). One possible explanation for this observation could be the lack of feature values per node in the input graph, as the usage of PPI-networks has been shown to have a good performance on graphs with multiple node-feature associations (Althubaiti et al., 2021). Thus, we omit this model from further analysis.

Models	**C-index**
MV_FCNN	**0.728 ± 0.008**
GCN_median	0.633 ± 0.012
GCN_zero	0.630 ± 0.006
GCN_deleted	0.624 ± 0.010

Table 5.5: *Comparison of multi-view FCNN based on PCA feature selection and GCN with different manipulation methods. Best values are marked in bold.*

5.2.1.4 Comparison of different scaling methods

To examine the effect of the different scaling methods described in section 4.3, we run a benchmark for standard, robust, and MaxAbs scaling on all SAE models as well as the multi-view FCNN and compare them to the results we obtained for MinMax scaling in section 5.2.1.1 (Fig. 5.7). We find the multi-view FCNN to outperform every SAE model for three preprocessing types and be highly competitive for one. Furthermore, concat integration yields the second-best results overall and is followed by element-wise integration methods. Overall averaging is highly competitive in 2 cases, and overall maximizing performs worst. MinMax scaling produces the best results overall. Because of the high similarity between different scaling types, we omit them from further analysis and continue benchmarking on the best scaling procedure, i.e., MinMax.

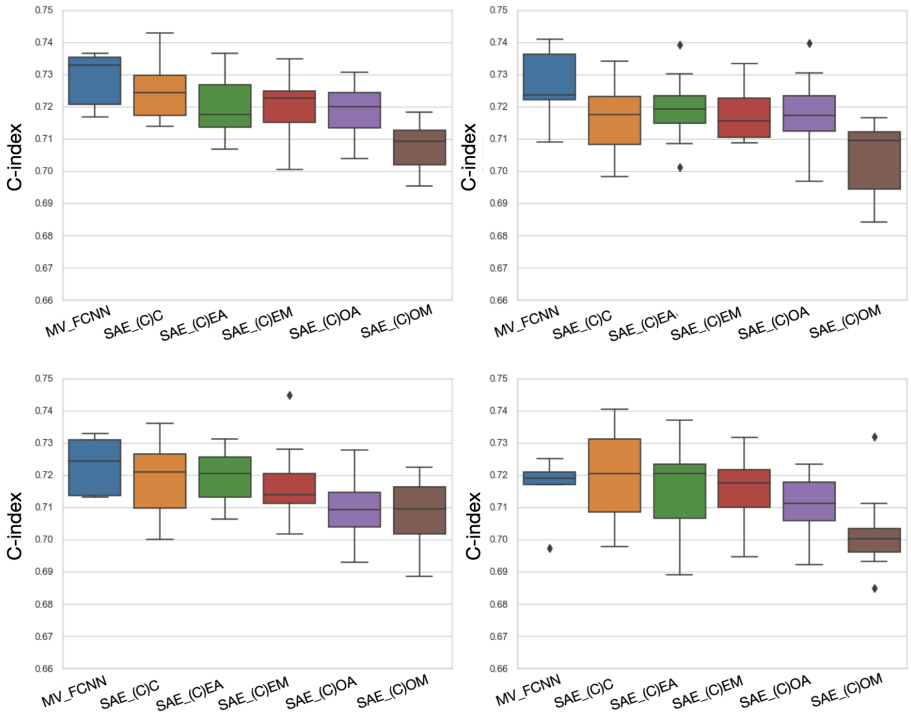


Figure 5.7: *Comparing multi-view FCNN and SAEs C, EA, EM, OA, OM on aggregated cross and non-cross scores for MinMax (upper left), MaxAbs (upper right), standard (bottom left), and robust (bottom right) scaling using PCA feature selection.*

5.2.1.5 Hyperparameter importance

We measured how changes in hyperparameter settings affected training and results for each fold of each model by applying the fANOVA importance estimator (Hutter et al., 2014). The batch size was selected as the most crucial hyperparameter in one-third of all tests, followed by loss ratio settings for the different AE models and selection of activation functions, specifically for the first layer of the FCNN after data has been processed in AEs, both appearing in about one-sixth of all tests (Fig. 5.8). We find batch sizes of 32 and 64 and PReLU activation functions to deliver the best results, whereas loss ratio settings and PReLU rate show no pattern. Thus, we assume the latter two must be specifically tailored for each fold to achieve the best results.

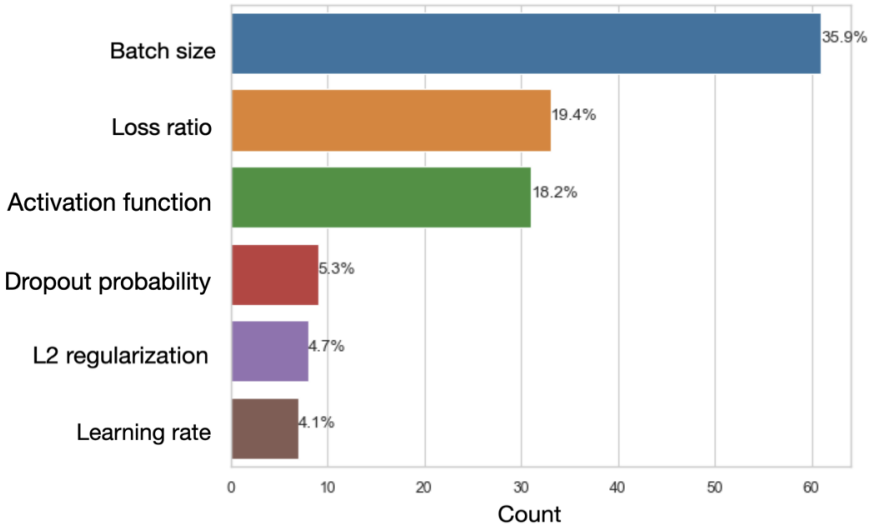


Figure 5.8: *The top six most essential hyperparameters. Count and percentage values were measured across each fold of each model.*

5.2.1.6 Time complexity

When assessing the strength of a model, it is essential to consider not only its performance score but also the time required to generate results. Therefore, we compare how long in median it takes to optimize a model on a single fold for 100 hyperparameter trials. We use KIRC cancer for the measurements, which had 742 samples overall. We find that the multi-view FCNN takes about 8 minutes, which is the fastest across all models. We measured a median of 13 minutes for SAE settings, while HAE settings took about 20. We did not examine a significant difference between the more subtle differences, i.e., cross decoding or usage of a second decoder. The GCN performed worse regarding time complexity, taking about 40 minutes. Note that the 30 GCN trials were run on the CPU, as it overloaded the GPU, resulting in memory errors.

5.2.2 Extending findings with LIHC and LUAD

As results stemming from PCA feature selection significantly outperformed those from selecting the top 2000 features with the highest variance for each view but nonetheless showed similar relationships between different models, we drop the latter method for the following comparisons. Additionally, since different scaling procedures are highly similar, as shown in section 5.2.1.4, we continue using the most potent scaling method, i.e., MinMax.

Overall average integration is a viable substitute for element-wise averaging in SAEs

In section 5.2.1.1, we pointed out how overall averaging performed well compared to other SAE models. We also find these results for LIHC and LUAD, which suggests that overall averaging may be a less memory-expensive alternative to element-wise averaging (Table 5.6).

Models	C-indices	C-index avg
OA	0.681,0.707,0.719	0.702 ± 0.016
EA	0.682,0.699,0.721	0.701 ± 0.016

Table 5.6: *5-fold averaged c-indices of all three cancer types aggregated over non-cross and cross environments of overall averaging and element-wise averaging SAE models.*

Cross and decoding environments don't affect performance

We have shown that non-cross and cross environments show similar performance for both SAE (Fig. 5.2) and HAE (Fig. 5.5) when using PCA feature selection, which we can confirm on two additional cancer types (Fig. 5.9). In contrast to our previous analysis of KIRC, the activity of the second decoder does not influence the results (Fig. 5.10).

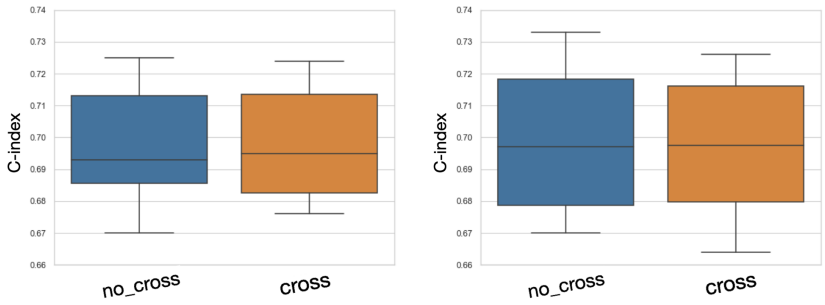


Figure 5.9: *Comparing scores for non-cross and cross environments aggregated over all three cancer types over each SAE (left) and HAE (right) model.*

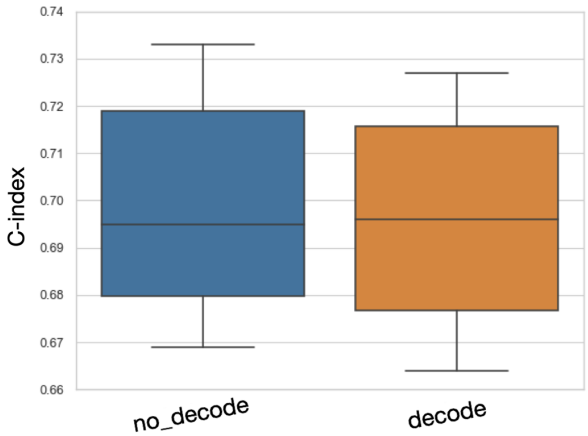


Figure 5.10: *Comparing scores for non-decoding and decoding environments aggregated over all three cancer types over each HAE model.*

The multi-view FCNN and HAE models integrating averaging and concatenation show the strongest performance

For both KIRC and LUAD, the multi-view FCNN outperforms every other model (Fig. 5.11). Furthermore, we find element-wise averaging methods in HAE settings, in particular, coupled with overall average integration in the second AE, to be the second strongest model, closely followed by concatenation integration for KIRC. This aligns with our analysis condoned in section 5.2.1.2. For LIHC, we find the second strongest models to be the HAEs using concatenation integration, followed by the multi-view FCNN. The best model is the SAE using overall averaging, significantly outperforming every other model.

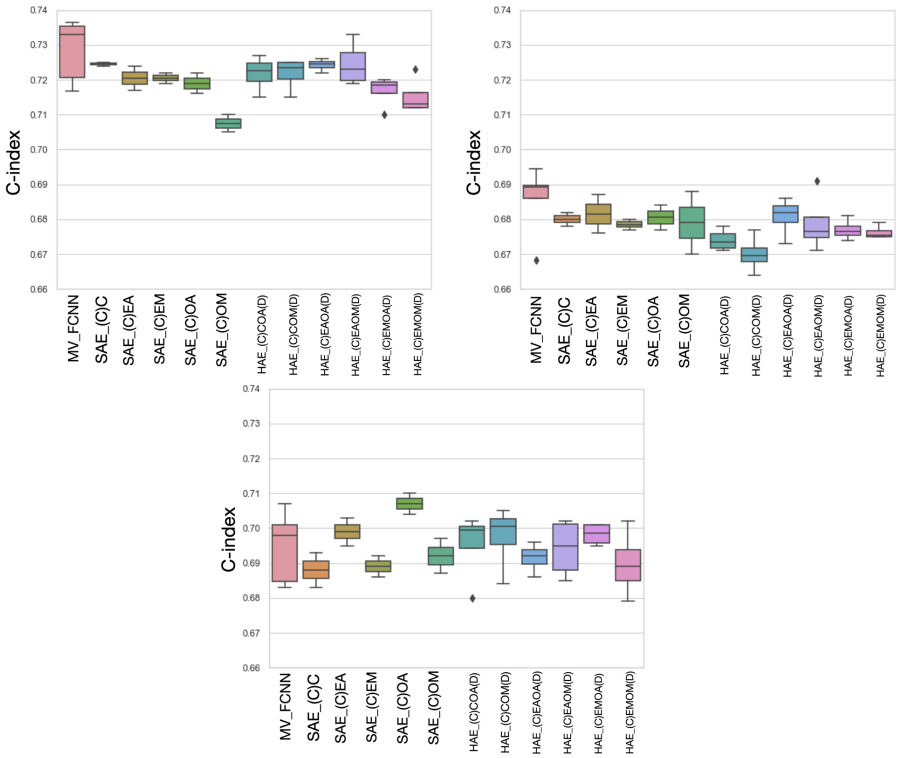


Figure 5.11: *Comparison of the MV FCNN aswell as all SAE and HAE models aggregated over cross and second decoder environments for KIRC (left), LUAD (right) and LIHC (bottom).*

Different cancer types affect the performance of hyperparameter optimization

As we observed similar results for LUAD and KIRC concerning the relative performance of different models but found deviating results for LIHC, we investigated this discrepancy by comparing the results of each trial in the hyperparameter optimization process. We find different cancer types to affect how accurately the tree-structured Parzen estimator can optimize hyperparameters over the span of 100 trials (Fig. 5.12). While KIRC and LUAD show progressively smaller variances of c-indices for more finished trials, variances in LIHC are not affected as fast. This may result in inaccurate results for LIHC across different models, as the optimization algorithm would need to skim through more than 100 trials to find the best hyperparameters with more certainty. We thus propose that different cancer types need a different amount of hyperparameter optimization trials to reduce the amount of variance and find the optimal settings.

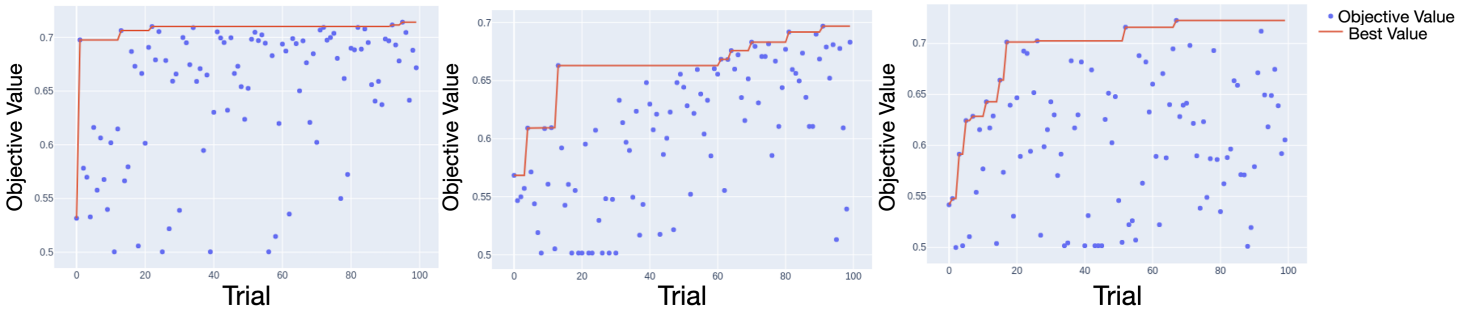


Figure 5.12: *Performance of the hyperparameter optimization of SAE_ OA over 100 trials on a single fold for KIRC (left), LUAD (middle), and LIHC (right). The objective value to be optimized is the c-index.*

5.3 Glossary for model abbreviations

Abbreviation	Model
MV_FCNN	Multi-view fully connected neural network
SAE	Single autoencoder
HAE	Hierarchical autoencoder
GCN	Graph convolutional network

Table 5.7: *Models and their abbreviations.*

Abbreviation	Integration
C	concat
EA	element-wise average
EM	element-wise maximum
OA	overall average
OM	overall maximum

Table 5.8: *Integration methods for the AEs and their abbreviations.*

Abbreviation	Environment
C_	Cross_
_D	_decoder

Table 5.9: *Environments for the AEs and their abbreviations. _ is to be replaced with a SAE model for the cross environment or SAE/HAE model for the second decoder environment.*

5.4 Libraries

We utilized various libraries such as NumPy, scikit-learn, PyTorch, and Pandas to implement the modalities for our models. The PH-model is based on pycox (Katzman et al., 2018). Optuna was used for hyperparameter optimization (Akiba et al., 2019) and seaborn for plot visualization (Bisong, 2019).

Chapter 6

Conclusion and future work

The current landscape of NN is broad and offers diverse approaches to problems across different fields. Particularly in the area of human medicine, its' usage has become more and more common. The goal of this thesis was to conduct a unified benchmark of different NN approaches trained for survival analysis on multi-view data. First, we provide a timeline of events that trace the development of survival analysis and NNs, showing the timely necessity of our thesis. We give an overview of the most important concepts associated with our work and introduce different models, which lay the path to our benchmark. We first conduct in-depth experiments on one cancer type, i.e., KIRC, comparing models one-to-one and aggregated over different environments such as cross decoding. We show that the multi-view FCNN and HAEs integrating element-wise averaging and concatenation bring forth the most promising results. Furthermore, our analysis reveals a high degree of similarity in the results for different scaling methods and displays how every other model outperforms the GCN. We establish the importance of three hyperparameters, i.e., batch size, loss ratio, and activation functions, which contributed to the performance of different models the most. Additionally, we show that the GCN was much more time expensive than every other model. To validate and contrast our findings with regard to model performance, we benchmark two additional cancer types, i.e., LUAD and LIHC. We show that overall average integration is a viable substitute for element-wise averaging in SAEs and that cross and decoding environments do not affect performance. We establish the multi-view FCNN as well as HAE models integrating averaging and concatenation as the strongest models overall. Finally, we display that different cancer types may need more hyperparameter optimization trials to find optimal results.

In section 5.2.2, we saw a high similarity between LUAD and KIRC concerning the relative performance of different models, which may result from those two cancer types having similar feature patterns. Thus, it would be interesting to analyze whether such a relationship exists and, if, how it affects model performance.

It has been shown that the integration of multiple views can increase the per-

formance for survival analysis approaches (Huang et al., 2019). As we only used two omics in our analysis, i.e., mRNA and DNA, we propose to enhance this work by including more views and analyzing which models improve, in particular models in the cross environment, since we could only use one cross decoding setting, i.e., decode mRNA using the decoder associated with DNA data and vice versa, in this work. Furthermore, it could be of interest to see how different combinations of views affect models.
Lastly, our pipelines could be enhanced with more sophisticated methods, such as learning rate schedulers for the NNs, nested cross-validation for less biased hyperparameter tuning, or pruning methods for hyperparameter optimization.

Bibliography

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

Sara Althubaiti, Maxat Kulmanov, Yang Liu, Georgios V Gkoutos, Paul Schofield, and Robert Hoehndorf. Deepmocca: A pan-cancer prognostic model identifies personalized prognostic markers through graph attention and multi-omics data integration. *bioRxiv*, pages 2021–03, 2021.

Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993.

Syed Muhammad Anwar, Muhammad Majid, Adnan Qayyum, Muhammad Awais, Majdi Alnowami, and Muhammad Khurram Khan. Medical image analysis using convolutional neural networks: a review. *Journal of medical systems*, 42:1–13, 2018.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

Ekaba Bisong. Matplotlib and seaborn. *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, 2019.

Rüdiger W Brause. Medical analysis and diagnosis by neural networks. In *Medical Data Analysis: Second International Symposium, ISMDA 2001 Madrid, Spain, October 8–9, 2001 Proceedings 2*, pages 1–13. Springer, 2001.

Bing Cheng and D Michael Titterington. Neural networks: A review from a statistical perspective. *Statistical science*, pages 2–30, 1994.

David R Cox. Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(2):187–202, 1972.

G Kleinbaum David and Klein Mitchel. *Survival analysis: a Self-Learning text*. Spinger, 2012.

Frank E Harrell, Robert M Califf, David B Pryor, Kerry L Lee, and Robert A Rosati. Evaluating the yield of medical tests. *Jama*, 247(18):2543–2546, 1982.

Zhi Huang, Xiaohui Zhan, Shunian Xiang, Travis S Johnson, Bryan Helm, Christina Y Yu, Jie Zhang, Paul Salama, Maher Rizkalla, Zhi Han, et al. Salmon: survival analysis learning with multi-omics neural networks on breast cancer. *Frontiers in genetics*, 10:166, 2019.

Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *International conference on machine learning*, pages 754–762. PMLR, 2014.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

Edward L Kaplan and Paul Meier. Nonparametric estimation from incomplete observations. *Journal of the American statistical association*, 53(282):457–481, 1958.

Jared L Katzman, Uri Shaham, Alexander Cloninger, Jonathan Bates, Tingting Jiang, and Yuval Kluger. Deepsurv: personalized treatment recommender system using a cox proportional hazards deep neural network. *BMC medical research methodology*, 18(1):1–12, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International conference on machine learning*, pages 3734–3743. PMLR, 2019.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

Christian von Mering, Martijn Huynen, Daniel Jaeggi, Steffen Schmidt, Peer Bork, and Berend Snel. String: a database of predicted functional associations between proteins. *Nucleic acids research*, 31(1):258–261, 2003.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

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

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Li Tong, Jonathan Mitchel, Kevin Chatlin, and May D Wang. Deep learning based feature-level integration of multi-omics data for breast cancer patients survival analysis. *BMC medical informatics and decision making*, 20:1–12, 2020.

Ioannis Tsamardinos and Constantin F Aliferis. Towards principled feature selection: Relevancy, filters and wrappers. In *International Workshop on Artificial Intelligence and Statistics*, pages 300–307. PMLR, 2003.

Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. Singular value decomposition and principal component analysis. *A practical approach to microarray data analysis*, pages 91–109, 2003.

DC Watt, TC Aitchison, RM Mackie, and JM Sirel. Survival analysis: the importance of censored observations. *Melanoma research*, 6(5):379–385, 1996.

David Wissel, Daniel Rowson, and Valentina Boeva. Hierarchical autoencoder-based integration improves performance in multi-omics cancer survival models through soft modality selection. *bioRxiv*, pages 2021–09, 2021.