Project Laboratory Report

Department of Telecommunications and Media Informatics

Author:	**Dávid Nagy**
Neptun:	**A936R6**
Specialization:	**Infocommunication**
E-mail address:	**nagydavid02@gmail.com**
Supervisor:	**Bálint Gyires-Tóth, PhD**
E-mail addres:	**toth.b@tmit.bme.hu**
Supervisor:	**Dániel Unyi**
E-mail addres:	**unyi.daniel@tmit.bme.hu**

Title: Classification of Gene Expression Data Using Deep Learning

Task

The goal is to build a machine learning model that can classify patients with different types of cancer, based on their gene expression data. First, I want to explore whether there is a machine learning model that can learn on the cBioPortal dataset [11], second I want to achieve an accuracy as high as possible.

2023/24 II. semester

1 Theory and previous works

1.1 Introduction

Artificial intelligence (AI) is a rapidly advancing field of computer science focused on creating systems that can perform tasks that typically require human intelligence. These tasks include understanding natural language, recognizing patterns, learning from experience, and making decisions. AI has applications in various domains, from self-driving cars to personalized recommendations on streaming platforms, and its development holds the potential to profoundly impact industries and societies worldwide. To get an overview, as well as a deeper understanding of AI, Stuart Russel, and Peter Norvig's book is recommended [1].

Machine learning (ML) is a branch of AI that enables systems to learn from data and improve their performance over time without being explicitly programmed. It encompasses a variety of techniques that allow computers to recognize patterns, make predictions, and learn from experience. Bishop's book is recommended for further reading in ML [2].

Deep learning (DL) is a subset of ML that deals with algorithms inspired by the structure and function of the human brain, known as artificial neural networks. These networks consist of multiple layers of interconnected nodes (neurons) that process information in a hierarchical manner. DL has revolutionized fields like computer vision, natural language processing, and speech recognition, achieving remarkable performance in tasks that were once considered very challenging [6]. Goodfellow, Bengio, and Courville's book is a good starting point for delving deeper into this field [3].

The connection between gene expression data and ML is significant in the field of bioinformatics and computational biology [7]. Gene expression data provides information about the activity of genes in cells, tissues, or organisms under different conditions.

ML techniques, particularly DL algorithms, have been increasingly applied to analyze gene expression data. These methods can identify patterns, correlations, and predictive models from large-scale datasets, helping to uncover hidden relationships between genes, identify disease biomarkers, predict clinical outcomes, and even suggest potential drug targets [8].

The utilization of ML in analyzing gene expression data offers a powerful tool for understanding the complex mechanisms underlying biological processes and diseases. It enables researchers to extract meaningful information from multi-dimensional datasets (like the two-dimensional datasets in [11]), paving the way for more targeted and personalized approaches in medicine and biotechnology.

In this paper, I present a way (in section 2.3) to extract information from the given dataset ([11]) by classifying cancer patients based on their gene expression data. I also point out future directions for this study to continue and expand the capabilities of the model I built.

1.2 Theoretical summary

1.2.1 Supervised, unsupervised and reinforcement learning

The three main branches of ML are supervised learning, unsupervised learning, and reinforcement learning.

In supervised learning, the model is trained on a labeled dataset, where each input is associated with a corresponding target or output. The goal is to learn a mapping (the model has to learn a function) from inputs to outputs, allowing the model to make predictions or decisions when given new, unseen data. Common tasks include classification and regression (see section 1.2.2).

In unsupervised learning, the model is given an unlabeled dataset and must find patterns or structure in the data without explicit guidance. Unlike supervised learning, there are no predefined target variables. Unsupervised learning algorithms aim to discover hidden patterns, group similar data points, or reduce the dimensionality of the data. An example could be anomaly detection.

Reinforcement learning involves training an agent ("an agent is nothing more than something that acts" - as stated in [1]) to interact with an environment to achieve a goal. The agent learns by receiving feedback in the form of rewards or penalties based on its actions. The goal is to learn a policy that maximizes cumulative rewards over time. Reinforcement learning is used in tasks where an agent must make a sequence of decisions, such as game playing, robotics, and autonomous driving.

1.2.2 Classification and regression problems

The backbone of my work is centered around classification and regression tasks, so I work with supervised learning problems. To get a sense of what kind of projects I deal with, I introduce some brief examples for easier understanding.

Let's see a classification problem first: acceptance at university. Students are applying to a university, that accepts them based solely on their test points and grades. One of the students got 9 out of 10 (9/10) on the test and 8/10 on the grades. This student did well and got accepted. Another student got 3/10 on the test and 4/10 on the grades, they got rejected. And now we have a third student who got 7/10 on the test and 6/10 on the grades and we are wondering if they get accepted or not. There's an intuitive way to decide: looking at previous data. This scenario is demonstrated on the 1. figure. In this figure, the horizontal axis shows the test results, and the vertical axis shows the grades. The first student I mentioned is marked with a green check mark (accepted), the second with a red X (rejected), and the third with a yellow question mark (we don't know yet if they are accepted or not). The red dots represent students who didn't get accepted, the blue dots mark accepted students, this is the previous data. Looking at the figure, we can draw a line (a red arrow pointing at it on the 1. figure), that separates the students based on their acceptance. We say that this line separates the students well, because most of the dots above the line are blue, and most of the dots under the line are red. Plotting the student with 7/10 test and 6/10 grade we can conclude that they are very likely to have been accepted because the corresponding point on 1. figure (yellow question mark) is above the line. This is a binary classification task because we split students into two categories: accepted and rejected.

Tasks with more than two categories are multi-class classification problems. My research also falls into this area, because the model I will introduce later (2.3) splits patients into three classes of cancer. To sum up, classification predicts discrete categorical labels.

Figure 1: Acceptance at the example university. Students marked with red didn't get accepted, while the blue ones did.
Source: https://www.youtube.com/watch?v=46PywnGa_cQ, 2024.05.13.

On the other hand, regression predicts continuous numerical values. As a very brief example let's assume we have several years of data on July temperatures in Morocco (e.g. expressed in °C). Based on this, we want to predict this year's July temperature in Morocco. This regression task aims to approach the continuous function of this weather factor.

Further information about classification and regression problems is provided in Bishop's book [2] (see the "Linear Models for Regression" and "Linear Models for Classification" sections).

1.2.3 Neural networks

A short introduction to neural networks is needed to understand the motivation behind my work.

Multilayer perceptron

My work mainly focuses on neural networks, so I will briefly present the one I deal with: the multilayer perceptron.

A multilayer perceptron (MLP) is a type of artificial neural network (ANN) that consists of multiple layers of nodes, or neurons, arranged in a feedforward manner, as the 2. figure shows. The first layer of neurons is called the input layer, the last is called the output layer and the layers between them are the hidden layers. Both the inputs and outputs of a neuron are numerical values.

In an MLP, information flows from the input layer, through one or more hidden layers, to the output layer, this is called forward propagation. Each neuron in a layer is connected to every neuron in the subsequent layer, and each connection is associated with a weight. The MLP learns from data by adjusting these weights through a process called backpropagation, which involves computing gradients and updating weights to minimize a loss function. Loss functions measure the difference between predicted and actual values (also called "labels"), guiding machine learning models to minimize prediction errors during training. To understand exactly how MLPs and backpropagation algorithms work, and to see different loss functions and their meanings, I recommend this book for the interested reader [2].

MLPs are highly flexible and can learn complex non-linear relationships in data. They are capable of automatically extracting features from raw input data and can capture intricate patterns in high-dimensional datasets. This makes them suitable for a wide range of tasks, from image and speech recognition to natural language processing and beyond.

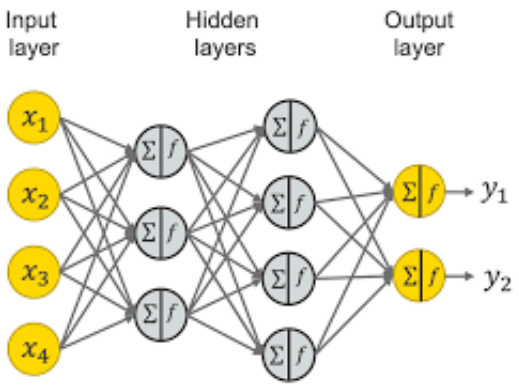

Figure 2: A simplified model of a multilayer perceptron.
Source: [9]

Underfit, overfit and prevention techniques

In machine learning, the goal is to create a model that can generalize well to unseen data. However, sometimes models can perform poorly due to underfitting or overfitting.

Underfitting occurs when a model is too simple (e.g. there are too few neurons in it) to capture the underlying structure of the data. A model that underfits performs poorly (meaning: with low accuracy) on both the training data and unseen data.

Overfitting occurs when a model is too complex (e.g. it contains too many neurons) and memorizes the noise present in the training data. Usually, a sign of overfitting can be that the model performs very well on the training data but poorly on unseen data. It's undesirable because it memorizes the training data rather than learning the general patterns.

To prevent underfitting and overfitting we often split the dataset into three disjoint parts: a training set, a validation set, and a test set. The training set is used to train the model, the validation set is used to assess model performance while training, and the test set is used to evaluate the final model, after training.

To understand the 3. figure, I first need to clarify the concepts of the number of epochs, the training error, and the validation error.

An epoch refers to one complete pass of the entire training dataset through the neural network. During each epoch, the model undergoes forward propagation (the prediction process), loss calculation (evaluating the loss function based on the prediction - given by the model - and the corresponding label), and backpropagation. Multiple epochs are typically used to allow the model to learn from the data iteratively.

The training error (or training loss) measures how well the model fits the training data, thus a lower training error indicates that the model fits the training data well. Training loss is typically calculated using a loss function that compares the predicted outputs of the model with the actual target values (labels) in the training dataset. Validation error (or validation loss) measures how well the model generalizes to unseen data. It's calculated in the same way as training error, by comparing the predicted outputs of the model with the actual target values, but in a separate dataset: the validation dataset. A lower validation loss indicates that the model is generalizing well to unseen data.

Ideally, we want both training error and validation error to be low. If a model has a low training error but a high validation error, it indicates overfitting. If both training and validation errors are high, it indicates underfitting.

In the 3. figure the horizontal axis shows the number of epochs and the vertical axis shows the value of the loss function evaluated on both the training and the validation dataset in each epoch. In this figure, this model is underfitting before the 100th epoch (because both training and validation losses are high), and overfitting afterwards. Thus, we have to stop training at the 100th epoch (exactly where the purple arrow shows in the 3. figure). Intuitively, this is what the early stopping algorithm does: tracks the validation loss, and when it starts to increase, it stops the training iteration. This avoids overfitting, as well as unnecessary computation time.

The interested reader can get acquainted with many other overfit prevention techniques including ensemble learning, regularization, and cross-validation by studying Murphy's book [4].

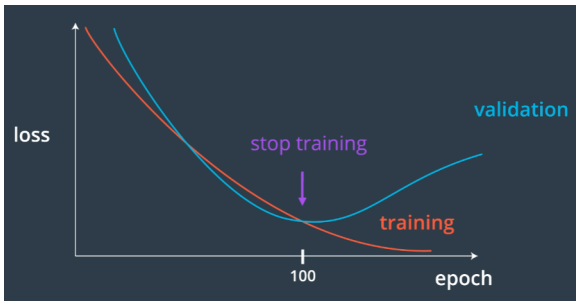

Figure 3: The concept of underfit, overfit, and early stopping.
Source: https://www.youtube.com/watch?v=b5934VsV3SA, 2024.05.14.

Data pre-processing

The final building block for summing up the theory of my project is data pre-processing.

In real-world applications, raw data often comes in various forms, may contain noise, missing values, or inconsistencies, and may not be in a format directly usable by machine learning algorithms. Data preprocessing aims to address these issues and prepare the data for further analysis and model building. By employing appropriate preprocessing techniques, we can enhance the performance and reliability of our machine learning systems. I use dimensionality reduction, normalization, and imputation (replacing missing values with a calculated estimate - e.g. with median values).

In datasets with a large number of features (features are the input variables that represent the characteristics of the data - in my main work, genes will be the features), dimensionality reduction techniques like Principal Component Analysis (PCA) can be applied to reduce the number of features while preserving important information. This helps in reducing computational complexity, mitigating the curse of dimensionality, and improving model performance.

Normalization (scaling the value of features to a similar range - e.g. scaling every feature value to the [0,1] interval) ensures that all features contribute equally to the model.

1.3 Starting point, previous works on this project

I was new to the neural network, deep learning methodology, and the gene expression topic too at the start of the semester. Thus, I had no prior related knowledge or work. I only know about one similar work in my department, from which I got help for loading the appropriate dataset. It's named "loading_gene_expression_data.ipynb", in the GitHub repository 2.4.

2 Own work on project

2.1 The basics

As a newcomer to machine learning and the Python programming language, I had to start with introductory courses. I completed several courses on Kaggle [12]: Python, Pandas, Intro to Machine Learning, and Intro to Deep Learning. This way I got acquainted with the tools I used throughout the semester. Also, I explored development environments and tools like Jupyter Notebooks (files with .ipynb extension in the referenced GitHub repository 2.4), Google Colaboratory [15], and Kaggle [12].

Every course followed the same structure: there were multiple topics within a course, and each topic had a theoretical and a practical (programming) part. So I could always test my knowledge to see if I understood the subject well enough.

After completing the mentioned courses I submitted solutions to the Titanic and Housing Prices competitions on Kaggle. In the Titanic solution, I used a model called RandomForestClassifier, which is designed for binary classification (categorizing data into two classes). I achieved a 77.5% accuracy on this classification task. In the Housing Prices competition, being a regression task, I used a model called RandomForestRegressor and achieved a 17860 score, measured with the Mean Absolute Error (MAE) loss function. Both of my results are considered entry-level results. For further information about the courses and competitions I reference the Kaggle website [12].

My corresponding programming work (courses and competitions) can be found in the directory named "basics" on my GitHub repository 2.4.

Now let's see what I've learned in detail. I summarized the completed courses in the following tables.

Table 1: Python course topics and their description

Topic	**Description**
Syntax, variable assignment, numbers	An overview of Python syntax, variable assignment, and arithmetic operators.
Functions	Calling functions and defining our own, and using Python's builtin documentation.
Booleans and conditionals	Using booleans and Python's if-then statements for branching logic.
Lists	List indexing, slicing, and mutating.
Loops and list comprehension	For and while loops, and a special Python feature: list comprehensions.
Strings and dictionaries	Two fundamental Python data types: strings (immutable), dictionaries (mutable).
Working with external libraries	Builtin libraries, importing external libraries, operator overloading.

Table 2: Pandas course topics and their description

Topic	**Description**
Creating, reading, and writing	The two core objects: the DataFrame and the Series. Reading, writing, and inspecting CSV files.
Indexing, selecting, assigning	Index-based, label-based, and conditional selection, assigning data.
Summary functions and maps	Getting an overview of the dataset, and transforming data.
Grouping and sorting	Groupwise analysis, multi-indexes, sorting.
Data types and missing values	Checking the data types, replacing missing values.
Renaming and combining	Changing index names and/or column names, combining data from multiple DataFrames and/or Series.

Table 3: Intro to Machine Learning course topics and their description

Topic	**Description**
How models work	Decision tree: the basic building block for some of the best models in data science.
Basic data exploration	Loading and interpreting the data using Pandas.
My first machine learning model	Choosing features and building a model with scikit-learn library.
Model validation	Measure the performance of the model, to test and compare alternatives. Metrics for model quality.
Underfitting and overfitting	Two of the most common model "diseases".
Random forests	A forest of decision trees.
Machine learning competitions	Getting to know the process of Kaggle competitions.

Table 4: Intro to Deep Learning course topics and their description

Topic	**Description**
A single neuron	Linear units, the building blocks of deep learning. Intro to Keras - a deep learning framework.
Deep neural networks	Hidden layers, activation functions, modularity.
Stochastic gradient descent	The meaning of the loss function, the optimizer, the learning rate, and the batch size.
Overfitting and underfitting	Learning curves, the model's capacity, and early stopping.
Dropout and batch normalization	Further techniques preventing overfit.
Binary classification	Accuracy, cross-entropy, and the sigmoid function.

2.2 Diving deeper into neural networks

After familiarizing myself with the basics, I had to delve deeper into the topic of neural networks to build more robust and powerful models. I've read chapters 1-4. from the book of Gábor Horváth et al. [5] about neural networks, and I also completed an introductory course into deep learning, using PyTorch as a framework [13]. Let's see the results of my programming work - implementing the todo-s in the notebooks - in this course. Every directory and file I reference in this subsection can be found in my GitHub repository under the "udacity_work" directory 2.4. The datasets I mention are well-known, benchmark datasets and can be found on Kaggle [12].

First, I implemented some of the steps in the training of a neural network that analyses student data. It included one-hot encoding, scaling, and some of the backpropagation steps. My solution is in the StudentAdmissionSulutions.ipynb.

Then I moved on to getting to know PyTorch. I made programming work with tensors, training neural networks, the Fashion-MNIST dataset, inference and validation, loading image data, and transfer learning. These projects can be found in the "pytorch_intro" direcory.

After I felt comfortable with PyTorch I learned about Convolutional Neural Networks (CNNs) and did projects like building an MLP for the MNIST dataset, implementing filters, visualizing layers (convolutional and max pooling), and designing a CNN for classifying the CIFAR-10 dataset. The 4. figure shows some of images classified by the CNN. Over each picture, there are written two categories. The model's prediction is the one before the brackets and the actual category is between the brackets. If the two are the same, it's written in green, else in red. It was a randomly chosen sample, so we can conclude, that this model does a pretty good job of classifying the images because there are only 2 red labels over the displayed 16 images. It reached 76%, but everyone can run the Jupiter Notebook (it is named: cifar10_cnn_solution.ipynb) and see the test result locally, it will be around 70% every time. My corresponding work is in the "cnn" directory.

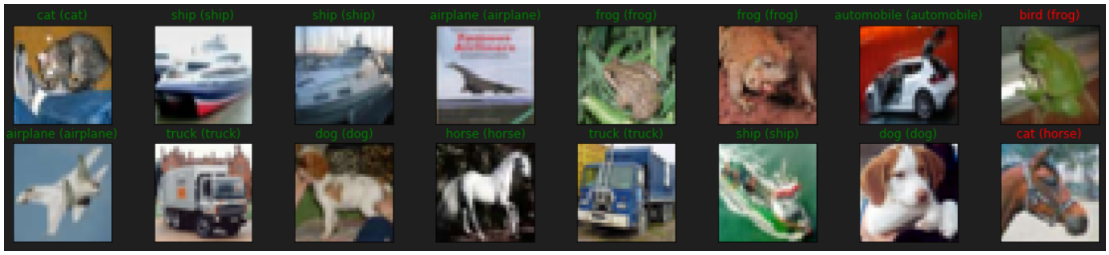

Figure 4: A part of the classified CIFAR-10 images with CNN.

An interesting application of CNNs is the style transfer task. In the Style_transfer_Solution.ipynb notebook I recreated a style transfer method that is outlined in the paper by Gatys et al. [10]. An example is shown in the 5. figure, where the content image is of a woman, and the style image is one of Robert Delaunay's paintings. The generated target image still contains the woman but is stylized with the circles and rainbow colors. The fundamental idea behind this problem is to separate the content and the style of an image. It can be done with mathematical methods (e.g. using the Gram matrix), which is explained in [10].

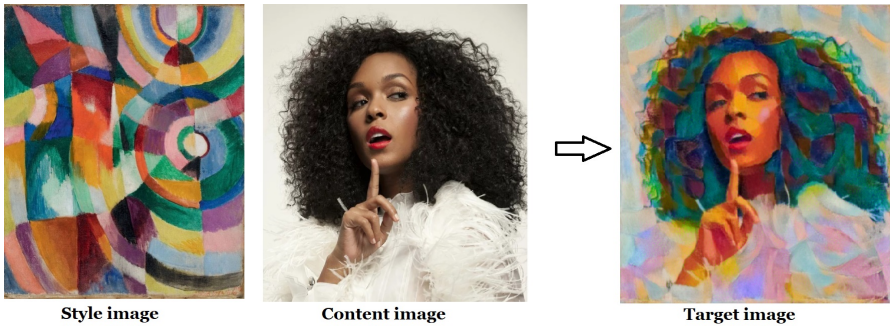

Figure 5: The result of style transfer.

There was only one big chapter left in this course, the topic of Recurrent Neural Networks (RNNs). Here I learned about Long Short Term Memory (LSTM) models, the gates that these architectures are using (namely: the learn, forget, remember, and the use gate), then I implemented an LSTM that makes character-level text prediction (Character_Level_RNN_Solution.ipynb). It takes a text as input and predicts the following N (number of characters to predict, given by the user) characters of that text. Another remarkable application idea is to make a sentiment prediction LSTM. In the Sentiment_RNN_Solution.ipynb I built a model that takes a review as input and predicts whether the review is positive or negative.

At the end of this learning journey, I got an insight into deploying ML models. The "Deploying PyTorch models" topic of the course explains how we can translate Python code into C++ with PyScript, achieving a much faster execution time.

To close off this subsection, I must mention my MLP solutions for the Titanic and Housing Prices Kaggle competitions. On the Titanic, I achieved a 1.2% better accuracy than with the RandomForest solution in the previous subsection. In the Housing Prices competition, I reached a much worse MAE score, than before. Both projects combined my knowledge learned from the courses, including data pre-processing, normalization, designing model architecture, training, validating, and testing the model. These works can be found in the "mlp_kaggle" directory.

2.3 The finish line

After learning all the necessary theory and trying out my abilities to build simple MLPs, my main task has come: classifying cancer patients based on their gene expression data. First, I got to know the Sztaki server environment: using a Linux terminal, starting a Docker container, and doing my work within this. I watched a video about gene expression [14], to understand the cBioPortal dataset [11]. The programming works and the data is in the "main_task" directory.

First, I loaded three mRNA expression tables. One contains information about breast cancer, another

one about glioblastoma (a type of brain tumor), and the last one about ovarian cancer patients. Every table contains many missing values, unscaled data, and more than 10000 genes. Thus, I applied imputation, replacing missing values with the corresponding median value. Then I applied dimensionality reduction (PCA) because I chose genes as my features and there were more than 10000 of them. This way, important information was kept in the data, but I got 200 features instead of the original 10000+, so I reduced computation time as well. I used normalization too, in the form of min-max scaling, so after PCA every value was mapped into the [0,1] interval. Consequently, input values don't vary on a too large scale, so they're far better for training than unnormalized values. The reason behind this is the fact that deep learning models use gradient calculation and large scale data makes the training process more unstable and slower. The exact math can be found in Gábor Horváth's book [5].

Data was cleaned, so I joined the different cancer types' DataFrame into one dataset, I shuffled the samples (to reduce overfitting chances) then I divided it into three parts: train, validation, and test dataset. I had to assign the tables with labels too, so I created the following three categories: Breast cancer, Glioblastoma, and Ovarian cancer. In magnitude, it was a 2800x200 array, meaning about 2800 patients' data and the compressed 200 gene features.

After I pre-processed the data I used a logistic regression model first to predict the probabilities of the categories. Then it classified the patients into the three classes I mentioned, based on the highest probability calculated in the previous step. It achieved 92% accuracy on the test set. It got all the breast cancer and ovarian cancer samples right but found none of the glioblastoma patients. My conclusion was that it had too few samples and the model couldn't learn the patterns behind it, an underfit phenomenon occured. So I loaded a twice as big dataset for the glioblastoma samples, and with it, the model reached 100% accuracy on the test set.

For interest, I built an MLP for this task too. I say for interest, because the flawless logistic regression solution is simpler, so it perfectly does the job. But when designing the MLP, I kept in mind that I had a relatively small dataset (about 2800x200). So a few layers and nodes may be enough, to avoid overfitting, thus I chose only one hidden layer with 32 neurons. The net has 3 output neurons because there are 3 classes right now (breast cancer, glioblastoma, ovarian cancer). This is one of the future project ideas, to make more classes, and this way to enhance the the width of the model. The other one is to train on more data samples from each cancer type, thus deepening the model's abilities.

2.4 Summary

My main task was to build a model that could classify cancer patients with reasonable accuracy (above 70%). As stated in the introduction 1.1, AI has powerful applications in genomics, thus my task is a prospective one to enhance machine learning models' capabilities in this area. My solution supports this statement, as I have built a flawless classifier for three cancer types.

I used all the techniques listed in the introduction, but most importantly: data pre-processing, multilayer perceptrons, and logistic regression models. In the topic of data pre-processing my most used tools were normalization, imputation and dimensionality reduction (PCA).

The most notable result is the 100% accurate classifier, trained on real cancer data, collected by US hospitals [11]. Also, remember that I spotted an underfit phenomenon and resolved the issue by loading more data. The achieved accuracies in the mentioned Kaggle competitions (Titanic and Housing Prices) are also worthwhile because they mark which directions for development are prospective and which aren't.

In conclusion, this field has much more potential than described in this paper, so I will continue my research enhancing this model's width (increasing the number of investigated cancer types) and depth (training on more samples in each cancer type).

References

[1] Russell, S., & Norvig, P. (Pearson, 2009). Artificial Intelligence: A Modern Approach. 3rd Edition.

[2] Bishop, C. M. (Springer, 2006). Pattern Recognition and Machine Learning.

[3] Goodfellow, I., Bengio, Y., & Courville, A. (MIT Press, 2016). Deep Learning. http://www.deeplearningbook.org, 2024.05.12.

[4] Murphy, K. (MIT Press, 2012) Machine Learning. A Probabilistic Perspective.

[5] Horváth, G. et al. (Panem Könyvkiadó Kft., 2006) Neurális hálózatok.

[6] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521(7553), 436–444*

[7] Alipanahi, B., Delong, A., Weirauch, M. et al. (2015). Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. *Nat Biotechnol 33, 831–838.*

[8] Angermueller C, Pärnamaa T, Parts L, Stegle O. (2016). Deep learning for computational biology. *Mol Syst Biol.*

[9] Miguel-A-Ramirez et al. (2022). Poisoning Attacks and Defenses on Artificial Intelligence: A Survey.

[10] Leon A. Gatys et al. (2016). Image Style Transfer Using Convolutional Neural Networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[11] *Dataset on cancer patients, collected by US hospitals.* https://www.cbioportal.org/datasets, 2024.05.12.

[12] *Kaggle - a data science community.* https://www.kaggle.com, 2024.05.14.

[13] *Intro to Deep Learning with Pytorch.* https://www.udacity.com/course/deep-learning-pytorch--ud188, 2024.05.14.

[14] *Gene expression video.* https://www.youtube.com/watch?v=X4oxrewkDpQ, 2024.05.14.

[15] *Google Colaboratory* https://colab.research.google.com, 2024.05.15.

Attached documents

- The GitHub repository containing my programming work: https://github.com/flash4242/project_lab