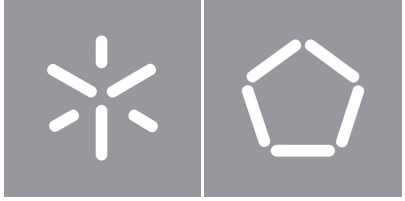


University of Minho
School of Engineering

Leonardo de Freitas Marreiros

Development of a software application based on deep learning to predict drug sensitivity of cancer cell lines

october 2023



University of Minho
School of Engineering

Leonardo de Freitas Marreiros

Development of a software application based on deep learning to predict drug sensitivity of cancer cell lines

Masters Dissertation
Master's in Informatics Engineering

Dissertation supervised by
Professor Miguel Rocha

october 2023

Copyright and Terms of Use for Third Party Work

This dissertation reports on academic work that can be used by third parties as long as the internationally accepted standards and good practices are respected concerning copyright and related rights.

This work can thereafter be used under the terms established in the license below.

Readers needing authorization conditions not provided for in the indicated licensing should contact the author through the RepositóriUM of the University of Minho.

License granted to users of this work:



CC BY-NC-SA

https://creativecommons.org/licenses/by-nc-sa/4.0/

Acknowledgements

I would first like to express my gratitude to Professor Miguel Rocha for this opportunity. His expertise, valuable support, and guidance were instrumental in shaping the direction of my research. I would also like to thank João Correia for his constant support, availability, and the well-informed insights and opinions he always provided.

I would like to extend my appreciation to my other colleagues at OmniumAI, namely João Capela Ribeiro and Fernando Cruz, for their useful suggestions and technical assistance when needed.

To my mother and my brother, I owe them their ceaseless encouragement and unwavering belief in my capabilities. Thank you for always rooting for my success.

Lastly, I want to express my deep gratitude to my closest friends, who have not only provided me with fun moments but also offered sincere advice and loyal friendship. Thank you for always being there.

To all those mentioned and countless others who have supported me, I am deeply thankful for the role you've played in these last five years of university, helping me reach this milestone in my academic journey.

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, Braga, october 2023



Leonardo de Freitas Marreiros

Abstract

With the increase in Deep Learning (DL) popularity, the need for efficient and effective model development and optimization has emerged. Neural Architecture Search (NAS) is a research area that aims to automate the time-consuming and iterative tasks involved in building and tuning DL models. This approach has already proven to be effective in various fields, such as computer vision and natural language processing, and holds potential in the field of cancer treatment.

Cancer is a group of diseases characterized by the uncontrolled growth and spread of abnormal cells in the body. The high degree of diversity among tumors, along with drug resistance acquired during (or before) treatment, makes it challenging to find effective therapies for all types of cancer. High-throughput screening (HTS) is a laboratory technique used to rapidly test large numbers of compounds against a biological target or assay to identify potential drug candidates. In cancer research, HTS is often used to identify compounds that have the potential to inhibit the growth of cancer cells or interfere with specific cancer-associated processes. Since the number of compounds with drug-like properties is much greater than the number of compounds that can be analyzed in HTS processes, DL has been used recently to predict drug response based on omics data.

This work integrated a DL-based framework capable of handling single and multi-input datasets of various types into the company's internal platform, where the work was carried out. Using Automated Machine Learning (AutoML) tools to reach this goal, namely tools with NAS capabilities, this new module automates the search for the best DL models that produce the most suitable results across a range of prediction tasks. The effectiveness of the developed work was validated in the context of drug response prediction by comparing the performance of the NAS-generated models with manually tuned models previously developed. The models found using this approach achieved comparable performance without the need for human expertise, proving the potential of NAS in cancer treatment, but also as a useful tool for automating and optimizing the model selection process in Machine Learning.

Keywords Cancer, deep learning, AutoML, NAS, drug sensitivity, drug synergy

Resumo

Com o aumento da popularidade de aprendizagem profunda, surgiu a necessidade de desenvolver e otimizar modelos de forma eficiente e eficaz. A pesquisa de arquiteturas neuronais é uma área de investigação que visa automatizar as tarefas demoradas e iterativas envolvidas na construção e afinação de modelos de aprendizagem profunda. Esta abordagem já provou ser eficaz em vários domínios, como a visão por computador e o processamento de linguagem natural, e tem potencial no domínio do tratamento do cancro.

O cancro é um grupo de doenças caracterizadas pelo crescimento descontrolado e pela disseminação de células anormais no corpo. O elevado grau de diversidade entre os tumores, juntamente com a resistência aos medicamentos adquirida durante (ou antes) do tratamento, torna difícil encontrar terapias eficazes para todos os tipos de cancro. A triagem de alto rendimento é uma técnica laboratorial utilizada para testar rapidamente um grande número de compostos contra um alvo biológico ou um ensaio para identificar potenciais candidatos a medicamentos. Na investigação do cancro, esta técnica é frequentemente utilizada para identificar compostos com potencial para inibir o crescimento de células cancerosas ou interferir com processos específicos associados ao cancro. Uma vez que o número de compostos com propriedades semelhantes a medicamentos é muito maior do que o número de compostos que podem ser analisados em processos de triagem de alto rendimento, métodos de aprendizagem profunda tem sido usados recentemente para prever a resposta a medicamentos com base em dados ómicos.

Este trabalho integrou uma framework baseada em aprendizagem profunda capaz de lidar com conjuntos de dados de vários tipos na plataforma interna da empresa, onde o trabalho foi desenvolvido. Recorrendo a ferramentas de aprendizagem de máquina automática para atingir este objetivo, nomeadamente ferramentas com capacidades de pesquisa de arquiteturas neuronais, este novo módulo automatiza a procura dos melhores modelos de aprendizagem profunda que produzam os resultados mais adequados num conjunto de tarefas de previsão. A eficácia do trabalho desenvolvido foi validada no contexto da previsão da resposta a fármacos, comparando o desempenho dos modelos gerados por pesquisa de arquiteturas neuronais com modelos afinados manualmente desenvolvidos anteriormente. Os mod-

elos encontrados através desta abordagem obtiveram um desempenho comparável sem necessidade de recorrer a especialistas humanos, comprovando o potencial da pesquisa de arquiteturas neuronais no tratamento do cancro, mas também como uma ferramenta útil para automatizar e otimizar o processo de seleção de modelos em aprendizagem de máquina.

Palavras-chave Cancro, aprendizagem profunda, aprendizagem de máquina automática, pesquisa de arquiteturas neuronais, sensibilidade a fármacos, sinergia farmacológica

Contents

1 Introduction **1**

1.1 Context and motivation . . . 1

1.2 Main aims and objectives . . . 3

1.3 Outline . . . 3

2 State of the Art **5**

2.1 Machine Learning . . . 5

2.1.1 Problem Definition . . . 6

2.1.2 Data Collection . . . 7

2.1.3 Data Preparation . . . 7

2.1.4 Data splitting . . . 8

2.1.5 Model and Input Representation . . . 8

2.1.6 Model Training . . . 9

2.1.7 Model Validation . . . 10

2.1.8 Machine Learning Models and Algorithms . . . 11

2.1.9 Artificial Neural Networks . . . 13

2.1.10 Deep Learning . . . 15

2.1.11 Bias-Variance Tradeoff . . . 18

2.2 AutoML . . . 20

2.2.1 Input Representation and Feature Engineering . . . 21

2.2.2 Algorithm Selection and Hyperparameter Otimization . . . 23

2.2.3 Performance Improvements . . . 27

2.2.4 Model Evaluation . . . 28

2.2.5 Neural Architecture Search . . . 29

2.2.6 Tool Comparison . . . 31

2.3 Cancer Treatment and Drug Response . . . 34
2.3.1 Cancer Treatment . . . 34
2.3.2 Drug Resistance and Drug Combination . . . 35
2.3.3 Drug Response Screening . . . 36
2.4 Deep Learning for Drug Response Prediction in Cancer . . . 37
2.4.1 Data in drug response prediction . . . 38
2.4.2 Predicting Single-Drug Sensitivity . . . 40
2.4.3 Predicting Drug Combination Effects . . . 43

3 Software Development 46
3.1 Software tools used in the development . . . 46
3.1.1 NNI and AutoGluon comparison study . . . 47
3.2 Core platform . . . 50
3.3 NNIPredictor . . . 53
3.3.1 Overview . . . 53
3.3.2 Implementation . . . 56

4 Results 69
4.1 Introduction . . . 69
4.2 Single input drug sensitivity prediction . . . 70
4.2.1 Datasets . . . 70
4.2.2 Experimental setup . . . 71
4.2.3 Results and Discussion . . . 72
4.3 Predicting drug synergy in cancer cell lines . . . 78
4.3.1 Datasets . . . 78
4.3.2 Experimental setup . . . 79
4.3.3 Results and Discussion . . . 82

5 Conclusions and future work 86
5.1 Conclusions . . . 86
5.2 Future work . . . 88

A NNI-AutoGluon preliminary performance comparison 112

B A549/ATCC NNI-AutoGluon comparison study 115

C CCRF-CEM NNI-AutoGluon comparison study **116**

List of Figures

1 A general workflow for machine learning tasks. 6
2 An example of an artificial neural network with two hidden layers. 14
3 Types of deep learning architectures. 17
4 Scenarios that arise from the bias-variance tradeoff. 19
5 An overview of AutoML pipeline covering input representation and feature engineering, model generation, and model evaluation. 20
6 Partial representation of the search strategy algorithm A_1. Ellipses represent the selection of a classifier, rectangles represent tunable hyperparameters. 23
7 Overview of architecture search using reinforcement learning. 26
8 An example of drug-response curves. 37
9 An example of a dose-response surface for the combination of the two previous drugs. . 37
10 Common inputs and outputs used in deep learning-based drug response prediction models. 40
11 Compound represented as molecular fingerprints. 40
12 Representation of the main Omnia packages 52
13 Illustration of the steps involved in an Omnia pipeline and the key concepts associated with each step 53
14 NAS domain model including main concepts. 54
15 NNIPredictor Class diagram. 56
16 Overview of the deep neural network (DNN) model space, including some of the primary possible configurations. The top image represents the overall model space architecture; subsequent images detail individual model components. 58
17 Overview of the convolutional neural network (CNN) model space 60
18 Overview of the 1D-CNN model space. 61

19 Overview of the recurrent neural network (RNN) model space with some of the possible parameter configurations. 62
20 Overview of the hybrid model space. 63
21 Overview of the multi-input model space. 64

22 Performance of different input representations and search strategies on the NCI 1 and NCI 109 classification tasks quantified using area under the receiver operating characteristic curve (AUC-ROC). Higher scores mean better performance. 73
23 Performance of different input representations and search strategies on the PC-3, CCRF-CEM and NCI-60 A549 regression tasks quantified using Root Mean Squared Error (RMSE). Lower scores mean better performance. 74
24 Comparison of the best model found using NNIPredictor with the best model and 5 and 11 model ensembles from the host group's previous study. 75
25 Performance scores of models with different gene expression representations using NNIPredictor vs the best model found for the same case in the host group's previous study. 82
26 Performance scores of models with different drug representations using NNIPredictor vs the best model found for the same case in the host group's previous study. 83
27 Performance scores of models trained with mutation (mut) and copy number variation (cnv) data in addition to gene expression ($expr_{DGI}$) and drug features ($drugs_{ECFP4}$) using NNIPredictor vs the best model found for the same case in the host group's previous study. 84

List of Tables

1 Overview of the different AutoML tools discussed in this section. 31

2 Published studies that have used deep learning for drug response prediction. 39

3 Details on the datasets used in this study. 71

4 Average fingerprint representations ranks. 74

5 Additional search strategy details for classification tasks. 77

6 Additional search strategy details for regression tasks. 77

7 Different types of methodological variables tested in this study. 81

8 A549/ATCC Dataset. Results represent the average between the three rounds of tests. . 113

9 CCRF-CEM Dataset. Results represent the average between the three rounds of tests. . . 114

10 A549/ATCC Dataset - Round 1. Scores represent the average squared error. 115

11 A549/ATCC Dataset - Round 2. Scores represent the average squared error. 115

12 A549/ATCC Dataset - Round 3. Scores represent the average squared error. 115

13 CCRF-CEM Dataset - Round 1. Scores represent the average squared error. 116

14 CCRF-CEM Dataset - Round 2. Scores represent the average squared error. 116

15 CCRF-CEM Dataset - Round 3. Scores represent the average squared error. 116

Acronyms

AI Artificial Intelligence.

ANN Artificial neural network.

API Application Programming Interface.

AtomPair atom pair fingerprints.

AUC area under the dose-response curve.

AUC-ROC area under the receiver operating characteristic curve.

AutoML Automated Machine Learning.

BERT Bidirectional Encoder Representations from Transformers.

BO Bayesian Optimization.

BPTT backpropagation through time.

CASH combined algorithm selection and hyperparameter optimization.

CCLE Cancer Cell Line Encyclopedia.

CI combination index.

CNN convolutional neural network.

CNV copy number variations.

COSMIC Catalogue Of Somatic Mutations In Cancer.

DAG directed acyclic graph.

DDR DNA damage response.

DGI drug-gene interaction.

DGIdb The Drug-gene Interaction Database.

DL Deep Learning.

DNN deep neural network.

DTP Developmental Therapeutics Program.

EA evolutionary algorithm.

EC_{50} half maximal effective concentration.

ECFP extended connectivity fingerprint.

FDA Food and Drug Administration.

FPKM Fragments Per Kilobase of transcript per Million mapped reads.

GAT graph attention network.

GCN graph convolutional network.

GD gradient descent.

GDSC Genomics of Drug Sensitivity in Cancer.

GI_{50} growth inhibition of 50% of cells.

GIN Graph Isomorphism Network.

GISTIC Genomic Identification of Significant Targets in Cancer.

GNN graph neural network.

GP gaussian processes.

GPT Generative Pre-trained Transformer.

GPU Graphics Processing Unit.

GRU gated recurrent unit.

GS Grid Search.

HTS High throughput screening.

IC_{50} half maximal inhibitory concentration.

IGTD Image Generator for Tabular Data.

LayeredFP RDKit layered fingerprints.

LDA linear discriminant analysis.

LIME Local Interpretable Model-agnostic Explanations.

LSTM long short-term memory.

MACCS Molecular ACCess System.

MAE Mean Absolute Error.

MAF mutation annotation format.

ML Machine Learning.

MLP multi-layer perceptron.

MoA Mechanism of Action.

MSE Mean Squared Error.

MSigDB Molecular Signatures Database.

MTE Molecular Transformer Embedding.

NAS Neural Architecture Search.

NCG Network of Cancer Genes.

OOB out-of-bag.

PCA principal component analysis.

RDKitFP RDKit fingerprints.

REFINED REpresentation of Features as Images with NEighborhood Dependencies.

ReLU rectified linear unit.

RF Random forest.

RFE Recursive Feature Elimination.

RL reinforcement learning.

RMSE Root Mean Squared Error.

RNN recurrent neural network.

RS Random Search.

SA Similarity Augmentation.

SGD stochastic gradient descent.

SHAP SHapley Additive exPlanations.

SiLU Sigmoid Linear Unit.

SMAC Sequential model-based algorithm configuration.

SMBO surrogate model-based optimization.

SMILES simplified molecular-input line-entry system.

SVM Support vector machine.

tanh hyperbolic tangent.

TCGA The Cancer Genome Atlas.

TGDRP Twin Graph neural networks for Drug Response Prediction.

TPE tree-structured Parzen estimators.

UMAP Uniform Manifold Approximation and Projection.

Chapter 1

Introduction

1.1 Context and motivation

Over the past decade, Artificial Intelligence (AI), particularly Machine Learning (ML) and DL, have revolutionized various fields, such as computer vision, natural language processing, and healthcare by enabling the development of advanced algorithms and models. DL is a class of methods from the field of ML that utilizes artificial neural networks to analyze and model complex data. These models are composed of multiple processing layers and are able to learn complex functions and discover higher-order representations of raw data [1]. DL is particularly useful for handling large volumes of high-dimensional and noisy data, such as omics datasets. In the field of cancer treatment, DL has the potential to predict drug sensitivity and synergy from large-scale omics and chemical datasets, by capturing the complex biological relationships present in the data. However, applying DL requires significant expertise and resources, which has limited its use in this field.

AutoML is a research area that aims to automate the time-consuming, iterative tasks of building and tuning ML models. AutoML enables the building of high-quality ML models without expert knowledge by creating a number of pipelines with different algorithms and parameters during training. NAS is a subfield of AutoML that focuses on automating the process of optimizing and designing the architecture of DL models. This approach has shown promising results in various domains, which might also translate to the field of cancer treatment.

Cancer is the name given to a set of related diseases characterized by uncontrolled proliferation of cells and potential spread. Finding effective treatments for all types of cancer is a complex task due to their heterogeneity. This has led to an increase in personalized cancer treatments.

Precision medicine is a medical approach focused on treating the specific characteristics of a person or group of patients with cancer or other diseases. In the context of oncology, it relies on identifying druggable tumor genomic alterations that might improve patients' survival and quality of life [2]. This

process includes generating drug sensitivity profiles for several distinct genomic/ epigenomic variants, covering many drugs whose efficiency can be evaluated with High throughput screening (HTS) [3].

HTS is a useful method used in the discovery of promising drugs, enabling the evaluation of millions of compounds against biological targets. In recent years, various datasets from extensive drug screening projects have been made accessible to the public [3] [4]. Nonetheless, this method has clear limitations, namely the fact that the number of compounds with drug-like properties is several orders of magnitude greater than the number of compounds that can be analyzed in HTS processes [5].

Computational methods, particularly AI algorithms and tools, can help identify the most promising drug candidates by predicting the sensitivity of cancer cells to specific drugs, therefore reducing the experimental effort required. These tasks have already seen widespread use of machine/deep learning methods using publicly available anti-cancer drug screening datasets [6] [7].

One aspect of precision medicine involves the use of drug combinations to treat cancer. Due to the potential for drug resistance, it is often necessary to use a combination of drugs to effectively treat cancer [8]. Drug synergy, which is the enhancement of combination effects beyond what is expected from each drug individually, is a desirable outcome in these cases as it can increase treatment efficacy without requiring an increase in dosage [9]. The development of accurate ML/ DL models for predicting drug synergy could greatly aid in designing and developing combination therapies, while preventing the screening of all conceivable drug pairs.

Recent work in the main supervisor's research group at the Centre of Biological Engineering [10] has already performed a systematic analysis of the impact of different methodological choices on the predictive performance of DL-based drug response prediction models. A variety of preprocessing and modeling strategies were tested in an attempt to understand which approaches produce the best results for both drug sensitivity and drug synergy prediction. One of the objectives of this work is to improve and generalize the work developed in this study by addressing the automation of the search for the best models for each available dataset.

OmniumAI, the company where the work will be carried out, has been developing an internal platform called Omnia, seeking to implement AutoML approaches for biological data. Another objective of this thesis will be to integrate into Omnia a versatile NAS-based single- and multi-input module capable of addressing a wide range of problems, including predicting drug sensitivity and synergy of cancer cell lines.

1.2 Main aims and objectives

The main aim of this work is to develop an application that can automate the creation of deep learning models for different prediction tasks, including the effect of chemical compounds and pairs of compounds on given cell lines characterized by multi-omics datasets (genomics, epigenomics, mutations, gene expression, proteomics, etc).

This encompasses a set of specific scientific and technological objectives:

- Review relevant literature and existing tools on deep learning methods and their applications in predicting drug sensitivity and synergy;
- Study in detail the previous work developed in the host group on this topic [10], aiming to improve on it and design/implement methods for automatic search of the best DL models;
- Study the Omnia framework and its AutoML capabilities;
- Compare AutoML tools and choose one that best aligns with the goals of this work;
- Implement general-purpose NAS-based single- and multi-input module, an integrate it with the Omnia framework;
- Design and implement a single input AutoML pipeline, that can take into account different DL models and architectures, validating it with distinct drug sensitivity datasets for specific cell lines;
- Design and implement AutoML methods supporting multiple types of input data, including multi-omics data (genomics and transcriptomics) that characterize cancer cells, and chemical characteristics of drugs, working over the Omnia platform.

1.3 Outline

This manuscript is structured as follows:

Chapter 1 introduces the research problem and outlines the main research objectives. The chapter also provides a summary of the structure of the manuscript.

Chapter 2 provides the state of the art for this work. It is divided into four sections. The first part provides an overview of a typical machine learning pipeline and introduces the main concepts of deep learning. The following section describes the various stages of the AutoML pipeline, such as data preparation, feature engineering, algorithm selection, and hyperparameter optimization, and includes the particular case

of neural architecture search. The third section of the chapter briefly describes cancer and drug-based cancer treatments. Finally, the use of deep learning for predicting drug response in cancer is explored, highlighting the successes and challenges of using these approaches in this context.

Chapter 3 describes the developed software, including the decisions made regarding technologies, a description of the Omnia platform and how the new module integrates with this platform, and a detailed description of the framework and its capabilities.

Chapter 4 presents two case studies. The first involves predicting drug sensitivity with single-input data, whereas the second involves predicting drug synergy with multi-omics data. For each case study, the datasets used are described, the experimental setup is outlined and the results of these experiments are analyzed.

Chapter 5 summarizes the main contributions and conclusions of the thesis, discusses its limitations, proposes possible solutions, and suggests ideas for future improvement.

Chapter 2

State of the Art

2.1 Machine Learning

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) that involves training algorithms to make predictions or decisions based on data. It is based on the idea that systems can learn from data, identify patterns, and make decisions with minimal human intervention [11].

There are two main machine learning types: supervised and unsupervised learning, although other paradigms, such as semi-supervised and reinforcement learning (RL) approaches also exist.

In unsupervised learning, the algorithm does not receive labeled training examples but must instead find patterns in the data on its own. Clustering and dimensionality reduction are examples of this type of learning problems.

In supervised learning, the algorithm is trained on a labeled dataset, where the correct output is provided for each sample in the training set. The goal is to make predictions on new, unseen examples. This is the most common scenario and can be used for classification, regression, and ranking tasks, among others.

In classification tasks, the output variable is categorical. For example, a cancer prediction task, consisting of image classification may assign to each image a category, such as *cancer* or *normal* tissues. In regression tasks, the output variable is continuous, such as in predicting the probability of survival for patients with a specific type of cancer. Finally, in ranking tasks, the goal is to order items according to some measure [12].

The process of building a machine learning model typically follows a sequence of steps. This pipeline usually involves several stages, including data preparation, model selection and training, and model evaluation and fine-tuning. Moreover, this process is often iterative and may involve multiple rounds of model training and evaluation until a satisfactory model is obtained. A general workflow for machine learning tasks is shown in Figure 1.

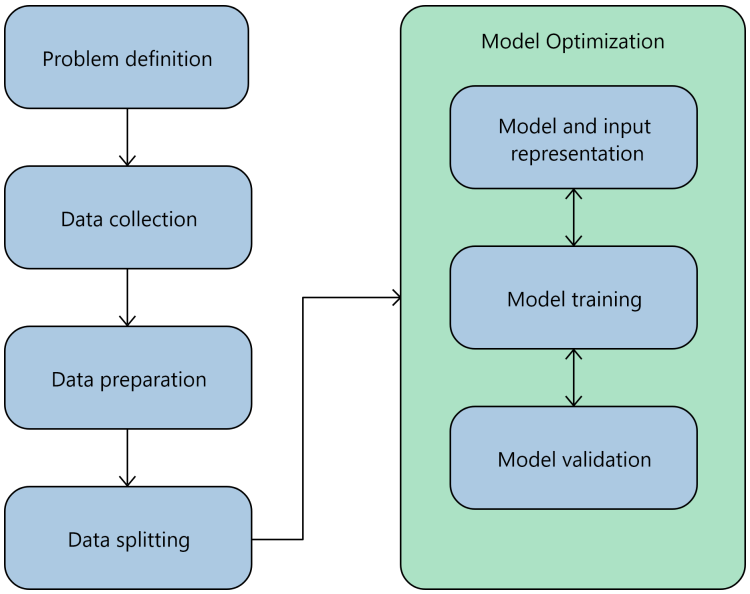


Figure 1: A general workflow for machine learning tasks.

2.1.1 Problem Definition

The first step of a machine learning pipeline implies the definition of the problem to be solved. It involves identifying the input data, the output data, and the type of machine learning task.

In this step, a data scientist should identify the data type that will be used as input to the machine learning model. This may include structured data, such as tabular data with rows and columns, or unstructured data, such as text, images, or audio. This will help determine the type of machine learning model most appropriate for the task and identify any preprocessing steps that may be needed. For example, if the input data is text data, it may need tokenization and stemming of the words in the text. If the input data is image data, a data scientist may need to resize the images or apply image augmentation. Finally, considerations about the format and quality of the data, as well as any missing or corrupted values, may need to be addressed.

The problem definition also implies identifying the type of data the machine learning model will produce as output (continuous or discrete), which will determine the type of machine learning task as a regression or classification task, for example.

The machine learning task should be defined in terms of the specific problem that is trying to be solved. This may involve identifying the specific goals and objectives of the task, as well as any constraints or limitations that need to be considered. For example, when trying to predict the presence or absence of patient cancer, the machine learning task may be defined as a classification problem with the goal of minimizing false negatives (i.e., patients predicted with the absence of cancer who actually have it) and/

or false positives.

Defining the problem clearly and accurately is an essential step in the machine learning pipeline because it determines the type of machine learning task that is being solved and guides the rest of the process, helping to select the appropriate data, evaluation metrics, and machine learning models, and to interpret the results correctly.

2.1.2 Data Collection

Collecting data is the process of acquiring data from various sources to use in a machine learning project. It is an essential step in the machine learning pipeline, which can significantly impact the model's performance. The most common ways to collect data are through public datasets, web scraping, databases, or synthetic or manually created datasets [13].

The easiest and fastest way to collect data is to use public datasets. Many public datasets are available on websites such as Kaggle, UCI Machine Learning Repository, or OpenML. Data can also be collected from online Application Programming Interfaces (APIs).

Data can be collected from databases, such as relational databases or NoSQL databases, using queries, but also using Web scraping to build datasets based on information from websites, extracting data from their HTML source code.

Gathering data is an important step in the machine learning pipeline, as it ensures that the model has access to high-quality, relevant data that it can use to learn and make accurate predictions. It is crucial to consider the quality and relevance of the data when gathering it, as well as any privacy or ethical issues that may be involved.

2.1.3 Data Preparation

Cleaning and preprocessing the data is the process of preparing the data, which will be used to train and test the model. It involves removing any inconsistencies or errors in the data and ensuring they are in a usable and consistent format.

It is common for datasets to have missing values. These missing values need to be addressed as they can impact the model's performance. There are several ways to handle missing values, such as imputing the missing values with the mean or median of the variable or dropping the rows with missing values.

Frequently, numerical values have different scales, which can affect the importance a model gives to a feature. To prevent this, numerical data can be normalized to ensure that all the features are on the same scale. This can be done by subtracting the mean of the data and dividing by the standard deviation,

or by scaling the data to a range between 0 and 1. Normalizing the data can help the model to converge more quickly and improve its performance.

There may also be the need to encode categorical features as numerical values if the data includes categorical features. This can be done using one-hot encoding, which creates a new binary column for each category, or using ordinal encoding, which assigns a numerical value to each category. Encoding the categorical data as numerical values allows the model to process the data more easily.

If the data are imbalanced, with one class significantly more represented than others, this can impact the model's performance. A data scientist might need to balance the data by undersampling the majority class or oversampling the minority class. This can improve the model's ability to learn and generalize to new, unseen data.

2.1.4 Data splitting

Splitting the data into sets is a common step in the machine learning pipeline. It involves dividing the data into distinct subsets that can be used for different purposes, such as training, tuning, and testing. The model should be trained and tested in independent sets because this ensures that it is not being evaluated on data that it has seen during training.

The training set is used to train the model. It should be a large enough sample of the data to provide the model with enough examples to capture the complexity of the problem. Typically, the training set is between 60% and 90% of the data.

The test set is used to evaluate the final performance of the machine learning model after training is done. It should be a separate, unseen sample of the data that has not been used in any other way during the machine learning process. The test set measures the model's generalization ability, and the results are used to assess the model's performance on new, unseen data. Typically, the test set is between 10% and 30% of the data.

There may also be a validation set used to evaluate the model's performance during training or to find the cutoff point for the training procedure; This set is typically used to find the best set of hyperparameters that maximizes the predictive capability of the model before the final evaluation using the test set.

2.1.5 Model and Input Representation

Model and input representation refers to the way in which the model and the input data are structured and represented. The choice of model structure and input representation can have a significant impact on the performance of the model, and they are an important and often iterative aspect of the machine learning

pipeline.

Model representation might affect how a model captures the complex patterns in the data. The model representation refers to the type of model, the architecture of the model, the parameters of the model, and the functions used to define the model. Selecting a machine learning model depends on the type of task, since different types of tasks might require different types of models, and the type of data, since some models may be more suitable for certain types of data than others. Some of the most popular machine learning models are briefly reviewed in section 2.1.8.

Likewise, different features and how they are represented may affect how a model learns. The performance of the model on the test set can be used to evaluate the effectiveness of the selected features. In addition, some models may require more feature processing, while others are able to handle raw data, and the way a feature is represented (e.g., raw data, images, graphs) can also be more suitable for specific model algorithms and can highlight different aspects of the data and therefore lead to a model learning different patterns in the data. The process of feature processing and input representation is further explored in section 2.2.1.

Once features are selected and the model structure is chosen, the model is fitted on the data by a training algorithm, and its performance is evaluated, at which point this process might be repeated as long as a termination criterion (e.g., number of iterations, time budget, target performance) is not achieved.

2.1.6 Model Training

Training a machine learning model involves iteratively adjusting the model's parameters to minimize the loss function using an optimization algorithm.

A loss function is a function used to measure the error or difference between the predicted and true values, thus giving an idea of how well the model fits the data. Some standard loss functions are Mean Squared Error (MSE), typically used for regression tasks, and Cross-entropy loss, typically used for classification tasks.

An optimization algorithm is used to minimize the loss function and find the most suitable set of parameters that incur the best predictions. Learning algorithms depend on the model structure chosen. Examples of learning algorithms include gradient-descent based methods for linear/ logistic regression or neural network-based models, or tree-induction methods for decision/ regression trees, among many others.

2.1.7 Model Validation

The next step is to validate the model on a set not used for training to assess its generalization performance or how well it can predict new data, based on a performance metric.

Error estimation methodologies

The most basic approach to model validation is holdout validation which involves the typical data splitting into train and test sets. However, because machine learning is an iterative process, a validation set can be used to evaluate the model before its final evaluation with the test set.

K-fold cross-validation is a resampling procedure used to evaluate the performance of machine learning models. It involves dividing the data into k folds that are equal in size and mutually exclusive. The model is trained on $k - 1$ folds of the data and evaluated on the remaining fold. This process is repeated k times, with a different fold used as the test set each time. The results are then averaged to get an overall estimate of the model's performance. This method provides a more reliable performance estimate since it is not influenced by the specific subset of data used on a single train/ test split.

Evaluation metrics

To evaluate the predictive performance of a model, i.e., the difference between the predicted values and the true values, several evaluation metrics are available depending on the task type.

For classification tasks, accuracy measures the proportion of correct predictions made by the model; precision measures the proportion of true positive predictions out of all the positive predictions; recall measures the proportion of actual positive cases that were correctly predicted; F1 score is a harmonic mean of precision and recall and area under the receiver operating characteristic curve (AUC-ROC). A ROC curve plots the true positive rate against the false positive rate at different classification thresholds. AUC measures the area underneath the entire ROC curve hence representing the ability of the model to distinguish between positive and negative cases.

For regression tasks, the Mean Absolute Error (MAE) measures the average absolute difference between the predicted and true values; MSE measures the average squared difference between the predicted and true values; the Root Mean Squared Error (RMSE) is the square root of the MSE, which is more interpretable than the MSE, and R^2 score measures the proportion of the variance in the target variable that is explained by the model.

Tuning

Fine-tuning and optimization are essential steps in the machine learning pipeline that involve adjusting the hyperparameters and other settings of the model, aiming to find a combination that best performs on the evaluation metric. This evaluation is performed using the estimation methods (holdout or cross-validation) and the metrics defined above.

This process is often a time-consuming, trial-by-error task. Several approaches to automating this hyperparameter search include Grid search, Random search, and Bayesian optimization, which will be explored in section 2.2.2.

2.1.8 Machine Learning Models and Algorithms

In this section, some of the most popular machine learning models are briefly reviewed. In addition to the most basic models, ensembles and models that use this technique are explored.

Linear regression

Linear regression is one of the simplest machine learning regression algorithms and is well-suited to tasks where a linear relationship between the input features and the continuous output may be assumed as a good approximation. Here, the model attempts to find the linear function that best maps from the input to the output variables. The linear regression equation's parameters (or coefficients) are learned from the data, which are adjusted to minimize the loss function during training.

Logistic regression

Logistic regression is an algorithm used for classification tasks when the output variable is categorical, and as before, a linear relationship between the input features and the output may be assumed as a good approximation. The model applies a logistic function (sigmoid) to the product of input features by model weights, mapping the input features to a probability between 0 and 1. Similar to linear regression, the parameters (or coefficients) are estimated from the training data, using an optimization algorithm, such as gradient descent, to minimize classification errors.

Support vector machines

Support vector machines (SVMs) [14] are kernel-based machine learning methods. To be able to model non-linear problems, the model maps the input data to a higher-dimensional space using a kernel function,

where the data can be linearly separated. In this higher-dimensional space, the algorithm constructs a hyperplane that maximally separates the classes, for classification tasks, or one that fits the data as well as possible, for regression tasks. The data points that are closer to the hyperplane and influence its position and orientation are called support vectors. The support vectors and hyperplane are used to calculate the predicted class or value for new data.

Decision and regression trees

Trees are a type of machine learning model that is used for both classification and regression tasks [15].

A decision tree is composed of internal nodes representing a condition regarding a particular feature, edges representing possible values for that given condition, and leaves representing a predicted class or value.

When training the model, the algorithm starts by selecting the feature and threshold that result in the most significant reduction in entropy, or another measure of disorder or uncertainty. This process is repeated recursively on each of the resulting sub-trees until a stopping criterion is reached (i.e., homogeneous examples regarding the classification, maximum depth).

To reduce the complexity of a tree, when a problem has a large set of features, for example, decision tree algorithms can include a pruning step that removes branches that use features with low importance.

Ensembles

Ensembles can be used to improve robustness and model performance. An ensemble is a set of models whose individual predictions are combined in some way to improve the overall performance of the system. Ensembles are often more accurate than the individual estimators that make them up [16]. For this to be true, the estimators that make up the ensemble need to be accurate and diverse. Two estimators are diverse if they make different errors on new data points.

Generally, an ensemble makes predictions by taking the majority vote of the predictions made by the individual models for classification tasks. For regressions tasks, the predictions of the individual models are averaged to produce a final prediction. Combining different models with distinct architectures and parameters, and different biases and variances ensures ensemble methods are able to capture a broader range of patterns in the data and are more robust to noise and outliers, leading to more accurate predictions.

Random forests

Random forests (RFs) [17] use an ensemble method known as bagging, where each model is trained on a random sample of data in a training set selected with replacement (bootstrapping). It is a combination of multiple decision trees, making predictions by averaging or voting the predictions of the individual decision trees. One-third of each training sample is set aside as test data, known as the out-of-bag (OOB) sample, which is used for cross-validation, completing the prediction.

To ensure low correlation among decision trees, random forests use feature randomness, which generates a random subset of features (each tree only considers a subset of features).

Gradient boosting

Gradient boosting is another ensemble method that combines multiple weak models to get better performance as a whole. Weak models have high bias or variance and do not perform well by themselves but can be used as building blocks to create more complex and robust models. This algorithm uses gradient descent optimization to minimize an arbitrary differentiable loss function. The model creates a sequence of weak learners, typically decision trees, and trains each one sequentially (boosting) to correct the mistakes of the previous one (following the negative gradient).

The model starts by fitting an initial model to the training data. The algorithm then fits a new model to the residual errors of the first model and repeats this step for a fixed number of iterations. The final model is a weighted sum of the individual models, where the weights are determined based on the performance of the weak learners on the training data.

XGBoost [18] and LightGBM [19] are popular variations of the gradient boosting algorithms.

2.1.9 Artificial Neural Networks

Artificial neural networks (ANNs) are machine learning models inspired by the human brain's structure and function. For supervised learning, these models consist of an input layer, one or more hidden layers, and an output layer. Each layer contains several units or neurons. Figure 2 shows an example of an ANN.

In typical feed-forward networks, the information flows from input neurons to output neurons. The input layer receives the input data and passes it through the hidden layers, which process the data and transform it using a set of weights and biases. The output layer receives the processed data from the hidden layers and generates the model's predictions.

Each neuron in the hidden layers receives input from the neurons in the previous layer and applies a

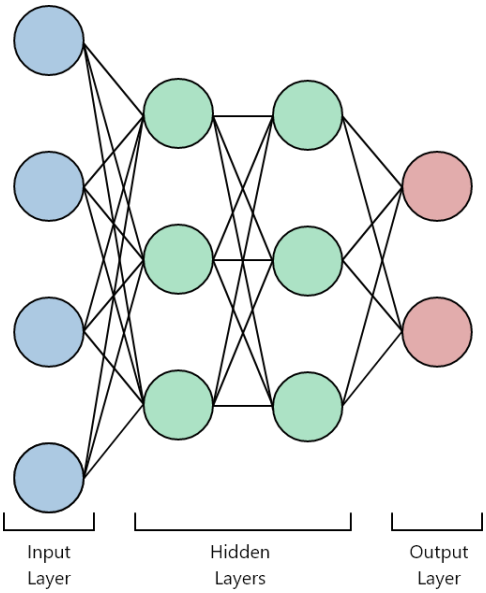


Figure 2: An example of an artificial neural network with two hidden layers.

non-linear activation function to the weighted sum of the inputs. Some of the most widely adopted non-linear activation functions for hidden units are the sigmoid function, the hyperbolic tangent (tanh) function, and, in recent years, the rectified linear unit (ReLU) function.

The output of the activation function is then passed to the neurons in the next layer. This process is repeated until the data reaches the output layer, which generates the model's predictions. The output layer may consist of a single neuron for regression and binary classification tasks or multiple neurons for the remaining classification tasks, with each neuron corresponding to a class label.

Learning is achieved by iteratively adjusting the weights and biases of the layers to minimize the loss function. This process is typically performed using an optimization algorithm based on gradient descent, such as the popular backpropagation algorithm, and more recent variants as stochastic gradient descent (SGD), and the Adam algorithm [20]. The number of times the optimization algorithm will cycle through the entire training dataset is called an epoch. The number of epochs is a hyperparameter that can be set before training the model.

Backpropagation involves propagating the error in the model's predictions back through the network and adjusting the weights and biases to reduce the error. The number of samples that are processed together in one forward/ backward pass is called the batch size. The error is defined by the gradient of the loss function, which is then used to update the model's parameters.

Backpropagation allows neural networks to model complex, non-linear relationships between inputs and outputs by efficiently computing the gradients in the network. However, backpropagation can get trapped on a local minimum which may be good or bad depending on how close the local minimum is

to the global minimum. In addition, this method can be very time-consuming and incur other problems, such as vanishing/ exploding gradients. Vanishing gradients appear when the values of a gradient are too close to zero, and the model stops learning, making the network hard to train. On the other hand, the exploding gradient problem occurs when the gradients become too large to update the model's weights.

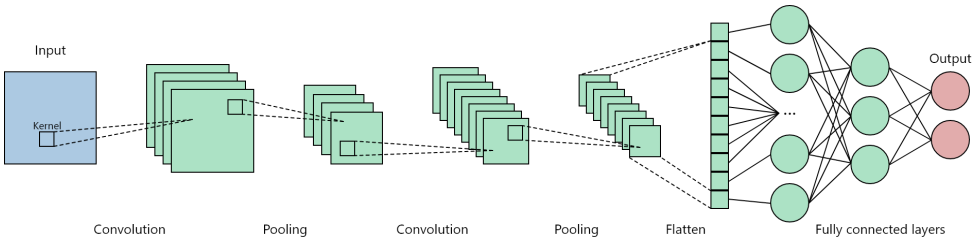
2.1.10 Deep Learning

Deep Learning (DL) is a subset of machine learning which differs in model architecture and data requirements. DL models are based on the principles described in section 2.1.9 but are more complex, generally consisting of more than three layers (also called deep neural networks (DNNs)), and are able to learn representations of data with multiple levels of abstraction [1]. Many DL models typically do not require some of the data preprocessing and feature engineering involved with machine learning since they can capture patterns and relationships directly from raw input data [1].

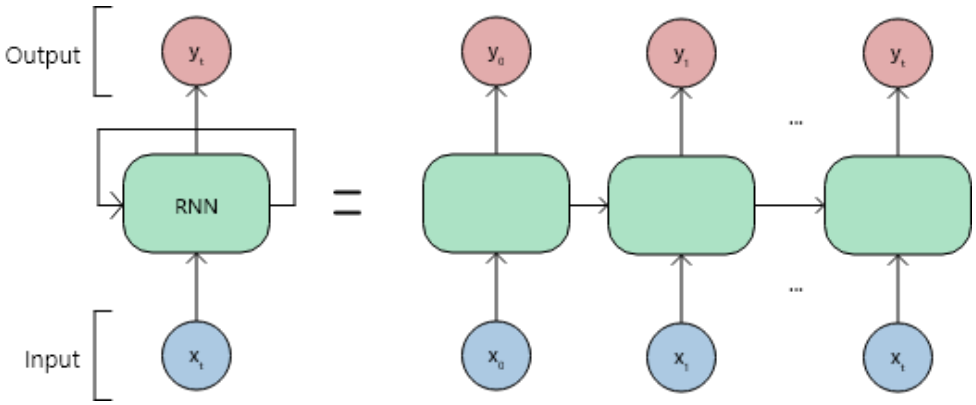
With the increase in computational power and the widespread use of Graphics Processing Units (GPUs) in this area, DL models got more complex. The growth in DL interest and the increasing availability of large public datasets also meant a significant increase in model parameters. Due to these reasons, along with the need for more specialized architectures to handle specific data types and tasks, a number of alternative DL architectures, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) were designed to overcome some of the limitations of ANNs. Some of the DL architectures are illustrated in Figure 3.

CNNs [22] (Figure 3a) are a type of neural network particularly well-suited for grid-like data such as images. A key aspect of CNNs are convolutional layers which apply a set of learned filters to the input data and generate feature maps. These filters (also called kernels) are a small matrix of weights designed to detect specific patterns or features by applying linear transformations to the input data, and feature maps represent the presence or absence of these patterns. The kernel/ filter moves a number of units (stride length) every time it is applied to cover the entirety of the input. Convolutional layers can handle different dimension data, i.e., 1D convolutional layers are used on one-dimensional data, such as audio, or text data; 2D convolutional layers are used on two-dimensional data, such as images; and so on.

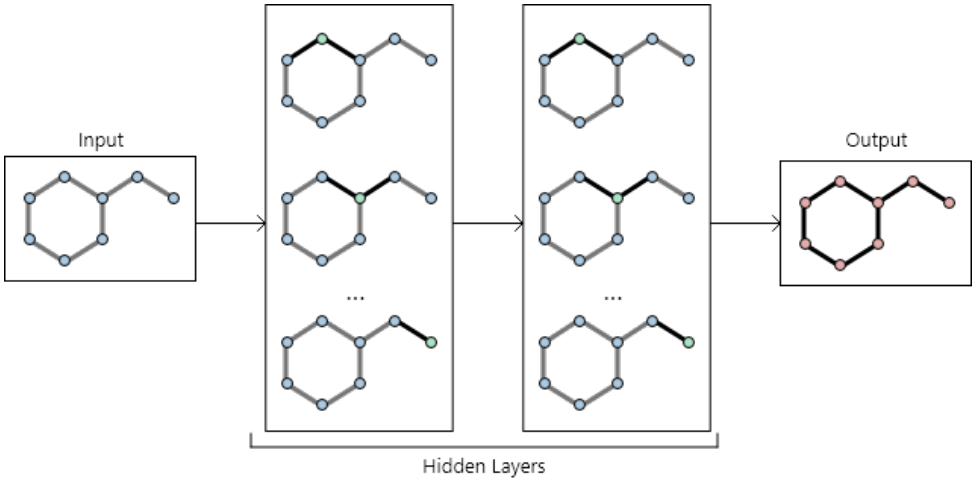
Pooling layers are used to reduce the dimension of convoluted feature maps by applying a pooling operation. The most used types of pooling are max pooling and average pooling, which return the maximum and average value of the feature map, respectively. Finally, one or more fully connected layers are usually used at the end of the network to produce the final output. These layers connect every neuron in one layer to every neuron in the next layer.



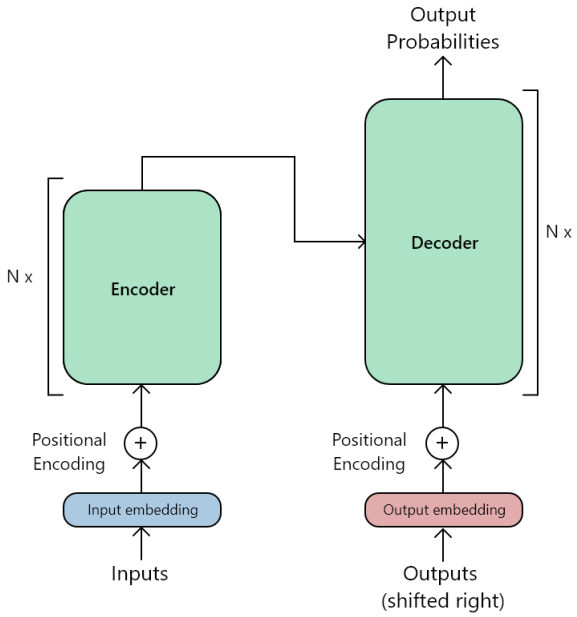
(a) A convolutional neural network (CNN) showing convolution, pooling and flattening layers leading to fully connected layers. These networks can be used in image classification, as well as natural language processing and signal processing tasks.



(b) A recurrent neural network (RNN) where the network's architecture is unrolled to show the flow of information through multiple stages, where each stage represents a time step. These networks can be used for language modeling, speech recognition, and time series prediction.



(c) A graph neural network (GNN). The green nodes in the hidden layers represent nodes that receive information from neighboring nodes through the message passing process. These networks can be used for tasks like node classification, edge prediction, and graph classification.



(d) A simplified transformer architecture, specifically highlighting the encoder and decoder components. It does not show the multi-head self-attention and feed-forward layers which are key components of the transformer architecture; adapted from [21].

Figure 3: Types of deep learning architectures.

RNNs (Figure 3b) are a type of neural network particularly well-suited for sequential data, such as time series, audio, or text. A key aspect of RNNs is their internal memory. Decisions are made based on the current input, but also on the previous input. A RNN is able to capture the dependencies between different elements in the sequence by cycling the information through a loop where the output of the network at a given time step is fed back into itself as input at the next time step. RNNs are trained using backpropagation through time (BPTT) [23] and more recent variants. This involves unrolling the network across time and training it on multiple time steps at once, using the same weight updates as in standard backpropagation.

There are many variations of RNNs, including long short-term memory (LSTM) networks [24] and gated recurrent units (GRUs) [25], which aim to address some of the obstacles of RNNs, such as the vanishing gradient problem.

Graph neural networks (GNNs) (Figure 3c) are a type of neural network well-suited for graph-structured data, such as chemical compounds. GNNs work by taking in a graph as input and processing it through a series of layers, each of which operates on the graph's nodes and edges. At each layer, the GNN computes new representations for the nodes in the graph by aggregating information from their neighbors, using a message passing mechanism. The message passing mechanism is typically implemented as a neural

network layer that takes as input the node's previous representation, the representations of its neighbors, and the edges connecting them. The updated node representations are then passed on to the next layer, where the process is repeated. There are multiple variations of GNNs, such as graph convolutional network (GCN) [26] and graph attention network (GAT) [27].

Transformers [21] (Figure 3d) are a type of neural network similar to RNNs often used in natural language processing tasks. Transformers are composed of an encoder and a decoder and use an attention mechanism that decides what parts of the input sequence are important at each step. The inputs and outputs are embedded and encoded so that the order of the sequence is preserved. The encoder receives the embedded input and passes it to an attention layer producing attention vectors which are then passed into a feed-forward layer before moving onto the encoder. The encoder can have multiple attention layer - feed-forward blocks to gradually capture more complicated relationships between the input sequence.

The decoder works directly on the embedded and encoded target output sequence during training. The transformer model uses a masked attention block on the decoder to predict the next element in the output sentence by only considering the previous element and masking the other elements. The outputs from this block are then passed through an attention layer along with the attention vectors generated by the encoder. The final attention vectors are then input into a feed-forward network which generates a single vector with a dimension equal to the number of unique elements learned by the model. The softmax activation function is applied to this vector to produce a set of probabilities for each element. These probabilities predict the likelihood of the next element of the output sequence.

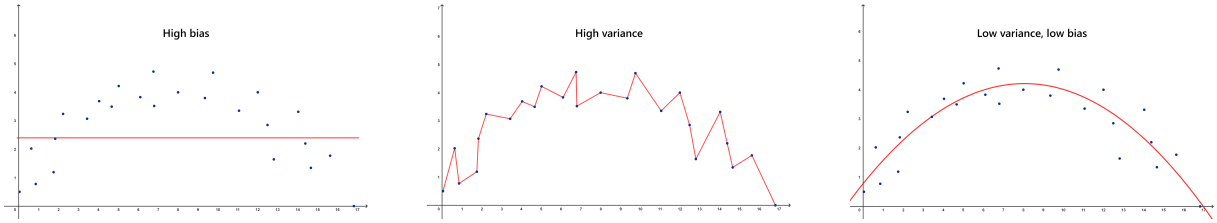
Some variations and extensions of the transformer architecture include Bidirectional Encoder Representations from Transformers (BERT) [28], Generative Pre-trained Transformer (GPT) [29] and RoBERTa [30]. A common characteristic of these methods is that they were pre-trained on a large amount of data in an unsupervised way before being fine-tuned on specific tasks.

The complexity of deep learning models poses additional challenges, namely, finding large amounts of labeled raw data for specific prediction tasks, sensitivity to hyperparameters which leads to more time fine-tuning the model, and a significant amount of computation power required when compared to machine learning methods.

2.1.11 Bias-Variance Tradeoff

When training a machine learning model, it is difficult to find a trade-off between a model that can learn the underlying patterns present in the data and a model that can generalize well on unseen data. This is known as the bias-variance tradeoff and is related to the phenomena of underfitting and overfitting.

Figure 4 shows, with a very simple example, an illustration of the different scenarios that arise from the bias-variance tradeoff.



(a) Underfit (b) Overfit (c) Good fit

Figure 4: Scenarios that arise from the bias-variance tradeoff. The blue dots represent the different input data points, and the red line represents the model function.

Bias is the difference between the average prediction value and the true prediction values. Models with high bias are said to be underfitted since they pay little attention to the training data and cannot capture the underlying patterns in it. Underfitted models lead to high error on both training and test sets.

Variance measures the amount of variability or spread in the predictions made by a model. A model with high variance pays much attention to training data and is not able to generalize well to unseen data. A model with high variance is said to be overfitting since it learned from noise and random fluctuations in the training data rather than its underlying patterns. Overfitted models lead to small error on the training data and high error on test data.

A model cannot minimize bias and variance at the same time, thus why this balance is so essential and depends on each task at hand. There are a number of techniques used to balance this tradeoff.

Some of these methods have already been mentioned. Cross-validation helps to reduce the risk of overfitting by ensuring that the model is not overly sensitive to the specific characteristics of the training data. By selecting the most important features (feature selection), the model is not prone to learn from patterns in features that do not yield high predictive power.

Regularization techniques prevent an overly complex model. These include L1 regularization (LASSO Regularization [31]), which adds a penalty term to the loss function based on the absolute values of the coefficients, and L2 regularization (Ridge regression [32]), which adds a penalty term based on the squared values of the coefficients.

Early stopping of model training based on the performance on a validation set can also avoid over-training the model. By monitoring the validation set over training iterations, early stopping halts the training and saves the current model once validation loss stops decreasing.

Increasing the size of the training set can help the model generalize better; however, this option is not

always available. In this case, data augmentation might help by generating new training data based on transformations to the existing training data.

DL techniques such as weight decay (term added to the cost function proportional to the magnitude of the weights) and dropout [33] (turning off a random subset of neurons during training) add a penalty to the model's objective function to prevent the model from learning overly complex patterns in the data. Reducing the number of layers or neurons in layers can further reduce complexity, thus avoiding overfitting.

2.2 AutoML

With the ubiquitous implementation of ML in a variety of fields, from image recognition [34], sentiment analysis [35], and speech recognition [36], to cancer prognosis and prediction [37], Automated Machine Learning (AutoML) has emerged as a new research field to optimize parts of the machine learning pipeline automatically.

Building an ML pipeline usually involves a profound knowledge of ML algorithms and methods, requiring domain experts with longstanding experience within a specific domain. This team can build a sensible ML pipeline with adequate specialized data preprocessing, domain-driven feature engineering, fine-tuned model, hyperparameter selection, and performance evaluation. However, this process is usually a complex and iterative procedure by trial and error until the desired predictive performance is achieved, which means that even experts require substantial resources and time to create well-performing models.

Therefore, AutoML allows for optimizing a tedious, time-consuming tuning process and making machine learning accessible for any domain expert without having to rely on a data scientist.

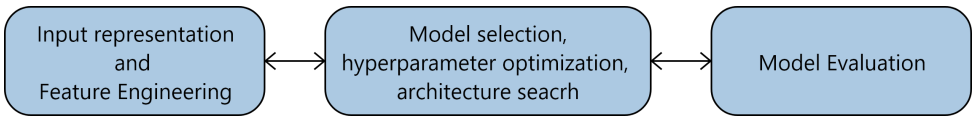


Figure 5: An overview of AutoML pipeline covering input representation and feature engineering, model generation, and model evaluation.

AutoML is an optimization problem with a vast search space, including features, hyperparameters, models, and architectures, thus involving a broad scope of candidate solutions. AutoML gives the appearance of "automation" by considering all tunable arguments into the whole optimization procedure internally. It is important to note that AutoML is still NP-complete despite the automation advantage [38, 39]. As a result, several heuristic techniques are proposed to solve the sophisticated optimization, which will be explored in the following sections.

The AutoML pipeline consists of several steps (Figure 5), starting with input representation and feature engineering, followed by model generation, and finally, model evaluation. These parts together make up the full scope of general ML applications. However, there is a particular case: Neural Architecture Search (NAS), which targets deep learning models where features, models, and algorithms are configured simultaneously.

2.2.1 Input Representation and Feature Engineering

There may be several ways to represent inputs. For example, compound data can be represented as simplified molecular-input line-entry system (SMILES) strings, molecular graphs, or compound images. Using different input representations on model architectures specific for those input representations data types may significantly impact performance. For instance, CNNs and GNNs are architectures that can take raw data, such as SMILES strings or compound graphs, respectively, as input and learn features directly from the data, not requiring explicit feature engineering. Generating the different input representations can be incorporated into the AutoML pipeline. This way, the performance of the final model can be improved by finding the most effective representation for the given task and data.

Feature engineering consists of generating and selecting features from a given data set for the subsequent modeling step. This step is considered to increase the performance of an ML pipeline significantly. Feature engineering is a difficult task since it is highly domain-specific and challenging to generalize. It requires the creation of features that provide valuable insights from the data itself and consider any limitations of the learning algorithm being used, such as how each model handles missing data, or the scale of data (e.g., SVMs and neural networks are sensitive to the scale of the input features). Therefore, it is usually a time-consuming task performed manually in a trial-and-error manner.

These shortcomings have driven the appearance of automated feature engineering frameworks, which aim to ease this process and improve the performance of subsequent models. Feature engineering can be divided into three fields: feature extraction, feature construction, and feature selection.

In most cases, feature extraction is a dimensionality reduction process used to transform raw data and extract non-redundant and informative features according to specific metrics. The most common approaches are principal component analysis (PCA) [40], linear discriminant analysis (LDA) [41], and recently developed autoencoders [42, 43].

Feature construction is a highly domain-related task that involves discovering missing relationships between the original features. Although domain knowledge is often necessary for the successful transformation of certain types of features, there are also instances where simple features can be transformed

easily without specialized knowledge to extract more meaningful features. Operations such as minimum, maximum, addition, subtraction, and mean are typically used for numerical features. Cartesian-Product is used for nominal features, and operations such as conjunctions, disjunctions, and negation are typically used for boolean features.

As it is impossible to test all possibilities, some automatic feature construction methods have been proposed to automate the evaluation of this search space. These methods showed results comparable to or even superior to those achieved by human expertise [44, 45].

Khurana et al. proposed Cognito [46], which explores various feature construction choices in a hierarchical and non-exhaustive manner, while progressively maximizing the model's accuracy through a greedy exploration strategy. Additionally, the system allows users to specify domain or data-specific choices to prioritize the exploration. The same authors later introduced a similar strategy based on RL [47], using a performance-driven exploration of a transformation graph, which systematically and compactly captures the space of given options. These authors also proposed a technique called Learning Feature Engineering (LFE) [48] based on meta-learning (Section 2.2.3). LFE recommends a set of valuable transformations to be applied on features without relying on model evaluation or explicit feature expansion and selection, therefore incurring a substantially lower computational cost.

Finally, feature selection consists of building a subset of relevant and non-redundant features based on the original feature set. It aims to improve performance, speed up training and avoid overfitting. Unsupervised algorithms like variance threshold and correlation matrixes are based on intrinsic properties of the data that use statistical measures to quantify the relationship between features or the relevance of each feature individually (e.g., filtering low-variance features and correlated features). Supervised algorithms use statistical measures to evaluate the relationship between features and the target variable (e.g., univariate selection, correlation coefficient, ANOVA), select features as a part of the learning process (e.g., feature importance), or evaluate the performance of different combinations of features using a machine learning model and select the combination that performs the best (e.g., Recursive Feature Elimination (RFE) [1]).

The above are standard methods to perform feature selection and are already supported by modern AutoML tools. The search space for these algorithms is made up of hyperparameters of these methods. For example, when using variance threshold, we need to determine the threshold value on which features with a training-set variance lower than this value will be removed.

More advanced feature engineering methods are usually implemented in dedicated feature engineering frameworks, such as ExploreKit [49], AutoLearn [50], and FeatureHub based on a variation of the expand-

reduce method [51]. Or [52, 44] based on genetic programming, an evolutionary algorithm technique.

2.2.2 Algorithm Selection and Hyperparameter Otimization

The task of model selection involves optimizing algorithm selection and their corresponding hyperparameters. This problem can be viewed as a single hierarchical hyperparameter optimization problem, where the choice of algorithm is itself a hyperparameter. This notion was introduced as the combined algorithm selection and hyperparameter optimization (CASH) problem in [53]. Instead of selecting an algorithm and then optimizing its hyperparameters separately, the hyperparameter optimization process can be integrated with the selection of the algorithm. This allows for both steps to be executed simultaneously, rather than sequentially.

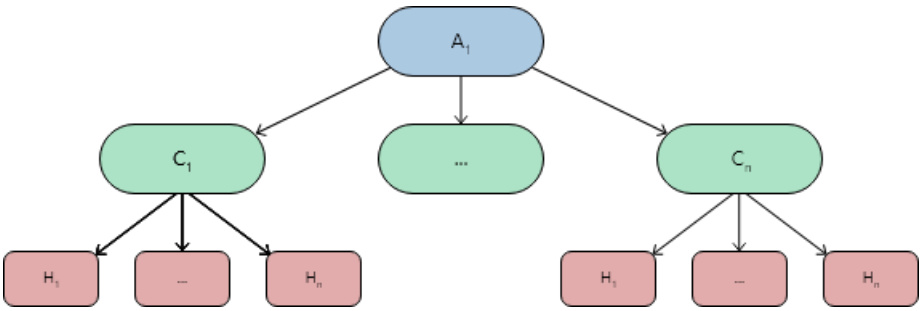


Figure 6: Partial representation of the search strategy algorithm A_1. Ellipses represent the selection of a classifier, rectangles represent tunable hyperparameters.

The search space can be interpreted as a tree as visualized in Figure 6. A_1 is the root node and represents an optimization algorithm with a child node for each classifier C. Each classifier has the according mandatory hyperparameters as child nodes H.

In the following section, various optimization strategies (also referred to as search strategies) will be introduced as potential solutions to the problem at hand.

Grid and Random Search

Grid and random search are the simplest and most naive optimization strategies. These simple search methods make no assumption about the search space.

Simply put, Grid Search (GS) explores a grid of configurations, considering a set of possible values for each parameter, testing all combinations, and selecting the best-performing. Random Search (RS) considers only a randomly selected set of configurations to test, selecting the best one.

GS is easy to implement and parallelize. However, it scales poorly when the search space is significant, since the number of evaluations grows exponentially with the number of hyperparameters.

A simple alternative to GS is RS which relies on sampling hyperparameter configurations from a user-specified set of hyperparameter values until a specific budget for the search is exhausted. RS is also easy to implement and parallelize. While it may be empirically better than GS [54], RS consumes resources running evaluations as long as the given budget is not exceeded and does not guarantee an optimal global configuration.

Since these simple search approaches do not exploit the knowledge gained from past evaluations, they are not usually efficient. However, due to their simplicity, these methods are still popularly used in AutoML.

Bayesian Optimization

Bayesian Optimization (BO) is a probabilistic surrogate model-based optimization (SMBO) method with two indispensable components, a probabilistic model and an acquisition function. BO builds a probabilistic model, such as gaussian processes (GP) [55], random forests [56], or tree-structured Parzen estimators (TPE) [57] that maps the configurations to their performance metric score. The probabilistic model is trained on the observations of the underlying function tested so far and is used to predict the function's behavior on unseen configurations.

Then, it defines an acquisition function to select the next configuration to evaluate based on the probabilistic model, e.g., expected improvement and upper confidence bounds, to balance exploration and exploitation of the areas that are more likely to contain the optimal solution, during the search process. A new sample is generated by optimizing the acquisition function and used to update the probabilistic model once it is evaluated on each iteration. The process of updating the probabilistic model with new observations and recomputing the acquisition function is repeated until a stopping criterion is met, such as a maximum number of evaluations, a time limit, or a satisfactory solution is found.

Some of the most popular methods lack efficiency when the number of data samples is significant. To alleviate this problem, the BO-based hyperband (BOHB) [58] algorithm was proposed, which combines the strengths of BO and Hyperband [59], a bandit-based approach to hyperparameter optimization. Fabolas [60] is another efficient BO procedure that finds high-quality solutions 10 to 100 times faster than other BO methods.

Evolutionary Algorithms

Evolutionary algorithms (EAs) are a collection of various population-based optimization algorithms inspired by biological evolution. EAs have high robustness and broad applicability since no assumptions about the problem's nature are needed. This method rarely finds a global optimum solution but quickly discovers a good local one.

This algorithm works by maintaining a population of individuals, each encoding a solution to a problem, that evolves towards better performance by applying reproduction operators, including crossover (exploit) and mutation (exploration) operators to create novel individuals.

In this case, every individual is represented by a set of genes encoding properties (e.g., types of layers, hyperparameters). A mutation slightly changes an individual to generate a new one by altering one gene (e.g., changing one type of layer or hyperparameter). Crossover generates a new individual by combining two individuals (ancestors) from the last generation (e.g., keeping a certain amount of the genetic information of each parent). The individuals that undergo mutation and crossover are chosen based on the fitness score, which measures how good an individual is.

The algorithms work by iterating between mutation, crossover, selection, and replacement. Selection is the process of choosing individuals from the current population to participate in reproduction operations (e.g., crossover and mutation) to generate new individuals. Various algorithms can make the selection process. The first is rank selection, in which the probability of an individual's selection is proportional to its fitness value. The second is the roulette wheel selection, where the probability of an individual being selected is based on their relative fitness. The third method is tournament selection [61], which draws k individuals randomly from the population and chooses the best individual from the group as the winner.

Replacement is the process of incorporating the new individuals generated by the reproduction operators into the population. There are several replacement strategies, including replacing the entire population with the newly generated individuals or keeping a portion of the best-performing individuals and replacing the rest with the new individuals generated by the genetic operators. These steps are repeated until the population converges or until a number of generations, time limit, or satisfactory solution is reached. A population converges when most of the individuals share the same genes.

Reinforcement Learning

Reinforcement learning (RL) is a robust and general optimization framework. Figure 7 illustrates an overview of RF-based AutoML.

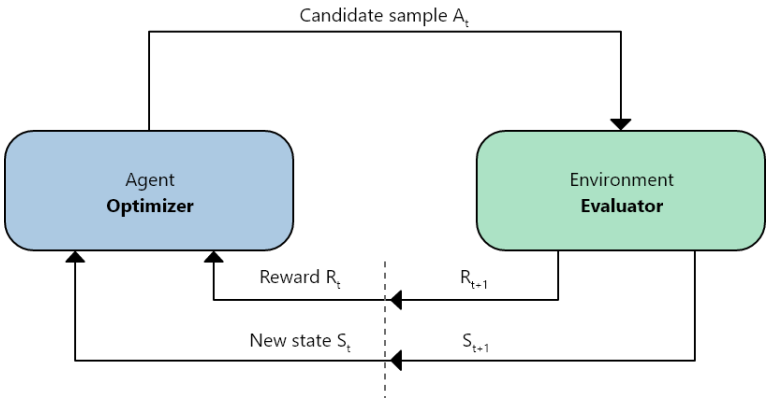


Figure 7: Overview of architecture search using reinforcement learning.

In RF, an agent interacts with an environment and learns to take actions that maximize a reward function. The agent learns by trial and error, receiving a reward or penalty for each action it takes. The agent's goal is to learn a policy that maximizes the cumulative reward over time, which the agent uses to decide what action should be taken for a given state.

When applied to AutoML, the agent acts as the optimizer, and its performance is measured by the environment, which serves as the evaluator. In this case, the agent executes an action A_t at each step t to sample a new configuration from the search space. The environment then sends a new state based on evaluations and a reward (performance of the configuration), R_t, which is used to update the agent's sampling strategy (policy). The environment refers to using a standard training procedure to train and evaluate the configuration generated by the agent, after which the corresponding results (such as accuracy) are returned. This process is repeated until a number of iterations or a time limit is reached or when the sampling strategy converges, meaning it is no longer changing significantly. Reaching the latter criteria incurs considerable time and computational resources since these algorithms typically require a large number of interactions with the environment to learn an optimal policy.

Gradient Descent

Gradient descent (GD) is a powerful optimization method. It is an iterative algorithm to find a given function's local minimum/ maximum. However, it only applies to differentiable functions. Nevertheless, for CASH problems, the objective is usually not differentiable or even not continuous; therefore, gradient descent, in general, is not applicable.

Nonetheless, for some differentiable loss functions, such as squared loss and logistic loss, continuous hyperparameters can be optimized by gradient descent, which offers the most accurate information where better configurations are located.

2.2.3 Performance Improvements

The previous sections covered various steps and techniques for building an ML pipeline. In this section, two popular performance improvement methods are introduced. These techniques speed up the optimization procedure, as well as improve the overall performance of the generated ML pipeline.

Pipeline ensembling

Ensemble methods consist of combining multiple ML models to create predictions. The combined predictions from various models often obtain better performance and generalization ability than the individual model [62].

While searching for a well-performing ML pipeline, AutoML frameworks create a large number of different pipelines, which perform almost as well as the best performer. Instead of discarding these configurations, the set of best-performing ones can be used to create an ensemble [63], thus improving robustness and limiting variance and bias errors associated with a single pipeline configuration.

Stacking [64] is a commonly used method in which multiple layers of machine learning pipelines are created, with each layer being a typical machine learning pipeline. The predicted output of each preceding layer is added as a new feature to the training data of subsequent layers, allowing later layers to potentially correct incorrect predictions made by previous layers [65].

Meta-learning

When a data scientist begins building a new model, he does not always start all over again but learns from previous tasks. AutoML methods usually start from scratch. Meta-learning aims to exploit this shortcoming by observing how different machine-learning algorithms perform on a wide range of learning tasks and then learning from this experience. A model is trained on this diverse set of tasks to solve new tasks with only a handful of training samples, leading to faster convergence with less trial and error [66].

Meta-learning can increase the efficiency of building an automatic ML pipeline by assessing the importance of single hyperparameters, removing unimportant hyperparameters from the search space, or identifying promising regions [67]. Further, this method can find the prior tasks most similar to a new one and pick the most promising configuration for each (warm-starting). Warm-starting can be used to start a CASH algorithm, providing an initial configuration [68]. Meta-learning is also suitable for pipeline structure search [69].

2.2.4 Model Evaluation

Once a new configuration has been generated, its performance must be evaluated. There is usually a trade-off between a faster evaluation and its accuracy. Model evaluation algorithms are presented in the following subsections.

Direct Evaluation

This is the simplest, most used method where the model is fitted on the entire training set, and the performance is evaluated on the validation set. Since the number of generated configurations can be significant, this method is expensive despite being accurate. Some other techniques have been proposed to fasten this strategy by trading accuracy for efficiency.

Sub-sampling

Fitting a model can take several hours and even days, depending on the dataset used. A common method to accelerate this process is to train configurations with a subset of the training data. This can be achieved by using only a subset of samples, a subset of available features, or using multi-fidelity evaluations such as SuccessiveHalving [70], Hyperband [59], and Fabolas [60].

SuccessiveHalving works by first creating a random set of configurations. Then, all currently remaining configurations are evaluated. The better half of configurations is transferred to the next iteration, and this process is repeated from the second step until only one configuration remains. This way, SuccessiveHalving prevents wasting a lot of training time in unpromising configurations.

Hyperband is also based on the idea of iteratively halving the set of configurations based on their performance. Fabolas uses a low-fidely model at the early stages of optimization to quickly evaluate a large number of different configurations and identify the most promising regions of the search space. As the optimization progresses, FABOLAS uses higher-fidelity models to gather more accurate information about the performance of the most promising configurations.

Early Stopping

Early stopping is a popular method to prevent overfitting in classic ML. However, in AutoML, it is used to terminate the evaluation of unpromising configurations early. Such configurations can usually be easily pinpointed early in model training. If the performance monitored on the validation set is poor, the evaluator can terminate the training and feedback the candidate as unpromising.

For example, by predicting the learning curve of an ML model, the authors of [71] could discard probably bad-performing configurations without complete training.

Despite being time efficient, early stopping rejects configurations with bad early-stage performance that may have eventually turned out to be good after sufficient training.

Surrogate models

For configurations that can be easily quantified, one method is to build a model that serves as a surrogate evaluator, sparing the computationally expensive model training. A surrogate model is an approximation of the true model and is used to predict the performance of given configurations from the experience of past evaluations [72], preventing the need to train and evaluate the true model for each configuration.

Parameter sharing

Another strategy that may lead to faster convergence and better performance is to use parameters of the previously trained models to warm-start the model training for the current evaluation. This is based on the fact that parameters learned with similar configurations can be close to each other, which means parameters of the latter configuration can be a good starting point for training.

Although this method may be helpful, it may also incur bias since different start points may lead convergence to different local optima. When sharing parameters, models also share the same gradients during the training process, guiding them to the same solution, even if they would have converged to different solutions if they had been trained independently.

2.2.5 Neural Architecture Search

With the general popularity of deep neural networks, data scientists are often tasked with manually engineering these increasingly complex networks. As with classic machine learning, there was an increase in research to automate this process, Neural Architecture Search (NAS), which aims to find the best agnostic neural network architecture.

This field is a subfield of AutoML with some overlap on the already discussed topics. Similar to typical AutoML, NAS methods' most relevant characteristics are search space, search strategy, and performance estimation.

Search space

The search space refers to the potential neural architectures that can be represented by the NAS algorithm and greatly determine how difficult the optimization problem becomes. The search space is composed of parameters such as hyperparameters (e.g., batch size, learning rate), the number of layers, the type of operation each layer can execute (e.g., convolution, pooling), and the corresponding hyperparameters of these operations (e.g., kernel and stride size for convolution layers).

The standard search space (entire-structured search space) is composed of all possible neural architectures, considering every specified operation for each layer.

To explore deeper, more general models, cell-based search space has been proposed [73, 74] in which the neural architecture is formed of a fixed number of repeating cell structures (blocks), similar to what is done by human-designed models [75]. This cell-based search significantly reduces the search space while also achieving better performance.

Search strategy

NAS evolutionary algorithms work similarly as discussed previously, using the same methods presented in Section 2.2.2 to explore their search space. When applied to NAS, mutations could be adding or removing layers, adding a skip connection, or altering hyperparameters. EA has been applied to NAS to entire-structured network architectures [76], as well as cell-based architectures [77, 78].

Within the context of NAS with reinforcement learning, the agent is required to learn a policy to construct network architectures. The trained networks will return a validation metric as a reward for the agent. RL has been used in NAS with various agent policies. Zoph et al. [79] were among the first to apply reinforcement learning to NAS and used a policy gradient to train the agent modeled by a RNN, which predicts a variable string to depict the architectural hyperparameters. The same authors later proposed a method [74] to build cell-based neural architectures using Proximal Policy Optimization (PPO) [80] for agent training.

When it comes to Bayesian Optimization that relies on Gaussian processes as the surrogate, there is a challenge to overcome, namely, how to measure the similarity of two neural architectures, which is not defined in Euclidean space. This is due to the fact that GP requires measurable distances between two points. The authors of NASBOT [81] overcome this problem by developing a distance metric in the space of neural network architectures, which can be computed efficiently via an optimal transport program. Network edit-distance [82] is another metric to map the distance between architectures based on how many operations are needed to morph one neural network to another.

For Gradient-based approaches, the most critical challenge is to convert non-differentiable optimization

problems into differentiable ones. DARTS [83] was a gradient descent-based pioneering algorithm to search for neural architectures, which used a soft-max function to change the search space to a continuous one. An encoder-decoder framework is used in [84], which also maps discrete configurations into a continuous search space.

Performance estimation

Training every given architecture on training data and evaluating its performance on a validation set demands a considerable amount of time and resources [79, 76]. To reduce this computational burden, some of the strategies already mentioned have also been used in NAS.

Sub-sampling has been used in [60] by training with a subset of training data, by using lower-resolution images in computer vision tasks [85], or by reducing the model size, such as training with fewer filters per layer [74]. Learning curve prediction has also been used in NAS [86] to speed up the search.

A form of parameter sharing is the "one-shot" method, where candidate architectures share the same weights [87] [88]. One-shot NAS methods embed the search space into an overparameterized supernet, where all possible architectures directly inherit weights from the supernet. This method significantly speeds up architecture search by only training the supernet, but it suffers from inevitable bias.

2.2.6 Tool Comparison

In this section, some of the most popular AutoML tools will be analyzed in terms of their features and capabilities (Table 1).

Table 1: Overview of the different AutoML tools discussed in this section.

AutoML tool	CASH	Algorithms	NAS	Pipeline Ensembling	Citation Count
Auto-Weka	Bayesian Optimization	ML, MLP	No	Stacking, Vote	1524
Auto-Sklearn	Bayesian Optimization	ML, MLP	No	Selection	1920
TPOT	Evolutionary Algorithm	ML	No	Stacking	216
Auto-Keras	Bayesian Optimization, RS	DL	Yes	No	589
AutoGluon	Fixed Defaults	ML, DL	Yes	Stacking, Bagging	223
H2O	Random Search	ML, DL	No	Stacking	203
NNI	BO, EA, GS, RS, others	ML, DL	Yes	No	-[a]

[a] NNI does not have an introducing paper; therefore, the citation count is not available.

Algorithms refer to the estimators natively supported by these tools. ML: machine learning models; MLP: multilayer perceptron; DL: deep learning; BO: Bayesian Optimization; EA: Evolutionary Algorithm; RS: Random Search; GS: Grid Search.

One of the first AutoML frameworks to appear was Auto-WEKA [53], based on the WEKA Java ML library. Auto-Weka can handle basic preprocessing tasks, such as missing value imputation and normalization. It was the first framework to address the CASH problem where the algorithm choice is a hyperparameter. Currently, the most recent version is called Auto-WEKA 2.0 [89]. Auto-WEKA uses a tree-based BO method, Sequential model-based algorithm configuration (SMAC) [90]. Furthermore, Auto-WEKA uses some ensembling methods after model selection to improve its predictions.

Auto-sklearn [69] is an AutoML tool built on top of the popular Python scikit-learn ML library. Auto-sklearn has also been getting ongoing improvements [91]. This tool also uses a BO method, namely an improved version of Sequential model-based algorithm configuration (SMAC), SMAC3 [92]. Auto-sklearn makes use of meta-learning to warm start the hyperparameter optimization procedure. This works by evaluating a set of meta-features (i.e., number of data points, features, and data skewness) for each dataset in a dataset repository and then applying BO to determine and store a machine learning pipeline with strong empirical performance for each dataset. Then, given a new dataset, its meta-features are computed, and the 25 closest stored pipelines are selected for evaluation before starting BO with their results. The original 140 datasets used have since been updated. An up-to-date list of OpenML task IDs can be found on their github[1]. Auto-sklearn uses ensemble selection by Caruana et al. [93] to build an ensemble based on the model's prediction for the validation set.

Another popular AutoML tool originally developed to automate biomedical data science is TPOT [94]. Regarding data preparation, this tool supports data imputation, standardization, and normalization. Separation into categorical and numerical features is based on a column's number of unique values. Preprocessing acts as a single block preceding model selection. However, most of its preprocessing is left to the user. TPOT also includes algorithms from the sci-kit library and some others. TPOT uses a genetic programming algorithm to automate and generate ML pipelines, which is implemented in the Python package DEAP [95].

TPOT currently supports neural network models as an experimental feature. However, the developers caveat that these models might not be necessary for multiple reasons: firstly, these models take a more significant amount of time and computing power to train than the implemented counterparts; TPOT will occasionally learn pipelines that stack several estimators, which can be nearly identical to some deep learning models, while generally optimized much faster; finally, TPOT's default estimators frequently are more manageable to introspect than deep learning models.

Auto-Keras [82] is a tool based on the popular library Keras, which focuses more on searching for

[1] https://github.com/automl/auto-sklearn/blob/master/scripts/update_metadata_util.py

deep learning architectures. The architecture search is performed with BO and network morphism as a metric to map the distance between architectures. It also uses random search, and Hyperband [59] to tune non-architectural hyperparameters.

AutoGluon [65] is another Python library for AutoML. AutoGluon implements two levels of preprocessing: model-independent and model-specific. This division avoids unnecessary calculations when they are not required. In addition to classical operations like normalization, gap-filling, and categories encoding, the framework has a module for converting datetime indexes into numerical features. AutoGluon explores the search space of models in a predefined order. Reliable models, such as random forests, are trained before less reliable models, such as k-nearest neighbors. This tool considers models such as neural networks, some sci-kit learn implementations, and others, with the possibility of adding custom models.

These models are ensembled in a stack of multiple layers and trained layer-wise, guaranteeing high-quality predictions within a given time restriction. Some methods used to increase accuracy include the use of skip connections both in multi-layer ensembling and neural network embedding. K-fold bagging is used to prevent overfitting and reduce variance in predictions by splitting the data in various ways with mindful tracking of out-of-fold examples.

H2O AutoML [96] is available in Python, Java, Scala, and R. This tool supports automatic type detection, categorical encoding using OneHotEncoding, imputation methods, and normalization. Preprocessing is optional for some models. Similarly to AutoGluon, available algorithms are tested in a fixed order (which can be customized by the user) set to start with models that consistently provide good results. The time spent on some algorithms is purposefully larger than on others according to their perceived or estimated "value" of each task. H2O AutoML uses random search on a set of hyperparameters they consider most important for each algorithm. In the end, the best-performing configurations are stacked to create an ensemble.

NNI [97] is a Python toolkit for AutoML, which includes feature engineering, hyperparameter optimization, and neural architecture search. NNI works with algorithms from the sci-kit library as well as TensorFlow [98], PyTorch [99], and MXNet [100]. This tool supports various search strategy algorithms, including Random and Grid Search, a variation of an evolutionary algorithm [101], a policy-based reinforcement learning algorithm [79], a TPE Bayesian optimization algorithm, and an array of one-shot methods. Furthermore, this toolkit expects users to define their own neural architectures and provides an Application Programming Interface (API) of mutations that introduce variability to the models.

2.3 Cancer Treatment and Drug Response

2.3.1 Cancer Treatment

Cancer is a group of diseases characterized by abnormal cell growth and the development of certain traits known as the hallmarks of cancer. These hallmarks, which are progressively acquired by many tumors, include the ability to evade growth suppression, maintain proliferative signaling, and metastasize. Other characteristics of cancer may include the induction of angiogenesis, resistance to cell death, modification of cellular metabolism, and evasion of immune destruction [102].

Despite their similarities, tumors exhibit a high degree of diversity, not only in their histological appearance but also at the molecular level, resulting in a large array of treatment strategies. The treatment of cancer typically involves a combination of modalities such as surgery, chemotherapy, radiation therapy, and targeted therapies, and may also include immunotherapy and hormone therapy [103, 104]. Cancer treatment aims to remove or destroy cancer cells and prevent their spread to other body parts while minimizing adverse side effects on the patient.

Chemotherapy, which is often used as a synonym for cytotoxic drugs, refers to a type of cancer treatment that targets rapidly dividing cells. These drugs can be harmful to both cancerous and healthy cells, leading to potential negative side effects. Molecularly targeted therapy, which specifically targets proteins altered in tumors, may also be classified as chemotherapy. However, these drugs aim to inhibit, for instance, the biochemical pathways crucial for tumor maintenance and are often less toxic than cytotoxic drugs [104]. Immunotherapies, another form of targeted cancer therapy, aim to stimulate or enhance the immune system's ability to recognize and attack cancer cells. These therapies may include monoclonal antibodies, which are designed to bind to specific proteins or cells in the immune system or on the surface of cancer cells, and stimulate or inhibit their function [104]. Other immunotherapies, such as checkpoint inhibitors, target proteins that inhibit the immune response, allowing the immune system to more effectively recognize and attack cancer cells.

Personalized cancer treatments refer to therapies tailored to the specific molecular characteristics of an individual's cancer. These treatments are based on the understanding that different types of cancer, even within the same tissue or organ and histological subtype, can have different molecular drivers and vulnerabilities [105]. Molecular profiling technologies, such as next-generation sequencing, are used to analyze cancer cells' genomic, transcriptomic, epigenomic, proteomic, and metabolomic characteristics. This can reveal the specific genetic mutations, changes in gene expression, and other molecular abnormal-

ities that drive the development and progression of cancer, thus increasing treatment success by designing treatments adapted explicitly to these characteristics.

2.3.2 Drug Resistance and Drug Combination

The efficacy of drug-based anti-cancer therapies is often reduced after an initial period of responsiveness due to the emergence of drug resistance mechanisms in cancer cells. These mechanisms may be intrinsic, meaning the presence of resistance-conferring factors in tumor cells is present prior to therapy, rendering it ineffective. Drug efficacy may also be reduced by acquired drug resistance, which develops during the treatment of initially sensitive tumors and may be caused by mutations that arise during treatment, as well as various other adaptive responses [8].

A variety of mechanisms have been implicated in drug resistance in cancer. The activation of alternative signaling pathways that are functionally similar to the targeted pathway can promote survival and proliferation, allowing cancer cells to evade treatment effects [8]. Drug resistance can also be caused by changes in how drugs are transported across cell membranes (drug efflux) [106]. Target alteration can influence cancer drug resistance in a number of ways, such as the development of mutations and or upregulation or overexpression of the target protein [106]. Drug resistance may be caused by changes in enzymatic conditions that lead to the lack of activation or inactivation of the drug [107]. The DNA damage response (DDR) is a system that repairs direct or indirect DNA damage caused by chemotherapy or other drugs. Epigenetic modifications, such as DNA methylation and histone modification, can also affect the expression of proteins involved in DNA repair, leading to an increased ability to repair the damage caused by drugs [106, 107, 8]. Other mechanisms, such as the protective role of the tumor microenvironment [8] [107], reduced drug uptake [107], cell death inhibition [106], and others, are also involved in drug resistance.

Administering a combination of drugs that work through different mechanisms may be able to overcome resistance caused by a single drug. Many current drug combinations are being used or evaluated in clinical trials for cancer treatment. An example is the pembrolizumab/ pemetrexed/ carboplatin combination that is being evaluated for the treatment of non-small cell lung cancer and includes two chemotherapeutic drugs (pemetrexed and carboplatin) and a checkpoint inhibitor (pembrolizumab) [108]. Dabrafenib/ trametinib combination is used to treat advanced melanoma and uses two targeted therapy drugs [109]. Other known drug combinations are cetuximab/ irinotecan, pertuzumab/ trastuzumab/ docetaxel, and ibrutinib/ rituximab, among many others.

Ideal drug combinations are those that have additive synergistic effects, i.e., when combining drugs,

the drugs may produce an effect greater than the sum of their individual effects, an effect known as drug synergy. This can be due to various mechanisms, such as the drugs targeting different proteins within the same pathway or targeting parallel pathways. In addition, drug synergy allows for the use of lower dosages of each individual drug, potentially reducing the risk of toxic side effects [9]. Nevertheless, developing drug combinations poses inherent scientific challenges along with economic considerations, intellectual property issues, and logistical obstacles [110].

The underlying difficulty of identifying effective anti-cancer therapies due to drug resistance, cancer complexity, lack of preclinical and clinical trial data, high cost, and several other reasons, call for new strategies and treatments to predict drug response.

2.3.3 Drug Response Screening

High throughput screening (HTS) is a laboratory technique used in the early stages of drug discovery to identify potential new treatments against a specific target or disease. In cancer research, HTS is often used to identify compounds that have the potential to inhibit the growth of cancer cells or interfere with specific cancer-associated processes, such as angiogenesis or cell proliferation. HTS assays can be either biochemistry-based or cell-based and involve administering the compounds being tested at various concentrations to cancer cell lines and measuring the resulting cellular response [111].

Screening results are typically represented as dose-response curves, which show the relationship between the concentration of a compound being tested and its effect on the cell or biochemical system being studied (Figure 8). This is often done through graphical representation, with the concentration of the compound or the function of dosage (log concentration) on the x-axis and the response on the y-axis. The shape of the curve can provide information about the compound's mode of action and its potential for further development as a therapeutic agent. As the dose-response curve is generally sigmoidal, it could be represented by a Hill equation [112]. Additionally, the potency of a compound can be determined by measuring the concentration required to produce a specific response, such as the half maximal effective concentration (EC_{50}), half maximal inhibitory concentration (IC_{50}), or the growth inhibition of 50% of cells (GI_{50}), among others.

The effect of drug combinations on cancer cells can be represented using a surface plot, where the x and y-axes represent the concentrations of the two drugs in the combination, and the z-axis represents the resulting effect (Figure 9). This allows for a more detailed visualization of the effect of different combinations of drug concentrations.

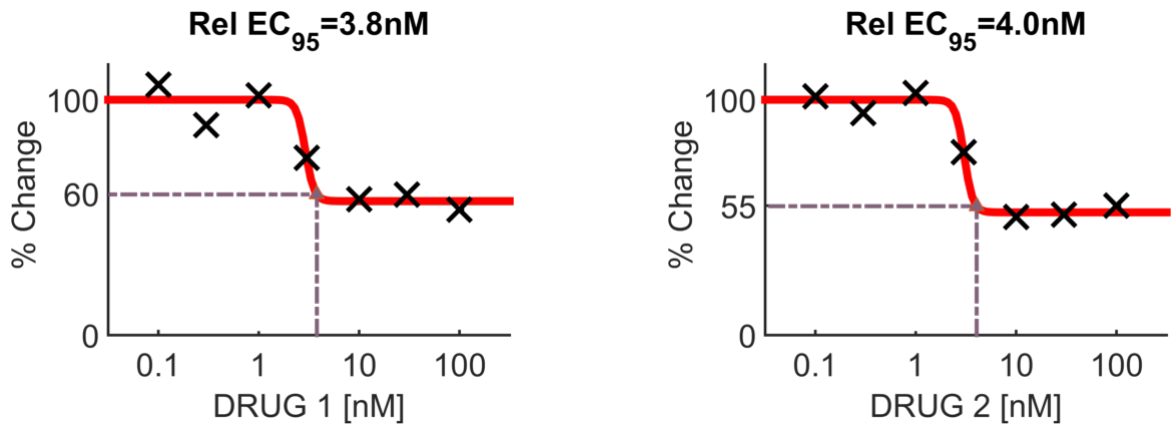


Figure 8: An example of drug-response curves. Figures generated using Combenefit [113].

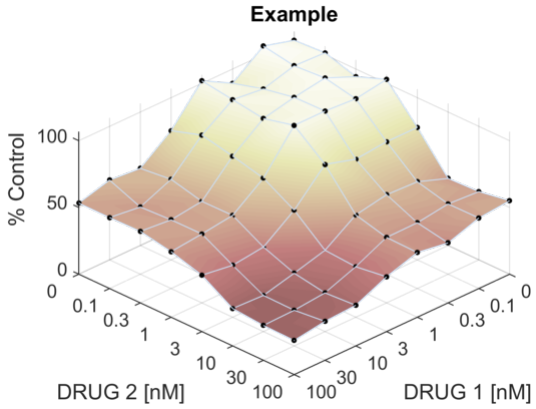


Figure 9: An example of a dose-response surface for the combination of the two previous drugs. Figure generated using Combenefit [113].

There are several ways to quantify the effects of drug combinations. One commonly used method is the combination index (CI), calculated using the Loewe additivity model [114]. The CI is a measure of the interaction between two drugs in combination. A CI of 0 indicates complete synergy, a CI of 1 indicates no interaction, and a CI greater than 1 indicates antagonism [115]. Another method is the Bliss independence model [116], which is similar to the Loewe additivity model but allows for the possibility of more than two drugs in combination. The combination index can be calculated using various software tools, such as CalcuSyn [117].

2.4 Deep Learning for Drug Response Prediction in Cancer

NAS represents a significant leap forward in the field of ML, acting as an automation tool for identifying the best-performing neural network architecture for a given dataset. Over the past decade, deep learning

has driven breakthroughs in diverse areas such as computer vision, natural language processing, speech recognition, and reinforcement learning. While there are many reasons for the rise in DL popularity, including DL's ability to automate feature extraction, as well as an increase in data availability and computational resources, its success can also be primarily attributed to specialized, high-performing neural architectures. NAS is becoming increasingly prevalent for automating the time-consuming architecture design step.

NAS has already outperformed human-designed architectures in many tasks. It has been used to design networks that meet or exceed the performance of hand-crafted architectures [74, 101, 118, 119, 120]. Despite its distinction in numerous other domains, there is a notable scarcity of studies that have specifically addressed the field of drug response prediction. Only a recent study, AutoCDRP [121], focuses on this specific task. AutoCDRP uses a surrogate model to predict the performance of GNN architectures sampled from a search space, allowing it to select the optimal architecture based on evaluation performance. The findings from this study demonstrated that the optimal GNN architecture generated consistently outperformed other cancer drug-response prediction state-of-the-art designs.

DL methods have the ability to handle large volumes of high-dimensional and noisy data, such as omics datasets, and may be able to capture and model complex biological data relationships. For these reasons, DL is well-suited for predicting the response of cancer cells to different drugs and drug combinations, which is essential for developing personalized and effective cancer treatments. Figure 10 provides an overview of common input data and output variables used in deep learning-based drug prediction models.

In this section, the previous work developed by the host group [10, 122, 123], and other DL methods for drug response prediction will be explored, including the different approaches and architectures that have been employed. Table 2 summarizes some of the studies that have used deep learning for drug response prediction.

2.4.1 Data in drug response prediction

Many input representations can be derived from drug compounds, including SMILES strings, molecular fingerprints, chemical descriptors, chemical graphs typically used on GNNs, or images typically used on CNNs (Figure 10). Molecular descriptors are numerical representations of a molecule's physical and chemical properties obtained through logical and mathematical transformations [139, 140]. A molecular fingerprint encodes the structure of a molecule into a numerical vector [139]. There are multiple ways to represent a compound as a fingerprint (Figure 11). In substructure key-based fingerprints, the vector values are set according to the presence or absence of predefined substructures (i.e., '1' if the given substructure is present and '0' if absent) [139]. Path-based fingerprints are molecular representations that encode the

Table 2: Published studies that have used deep learning for drug response prediction.

Study	Training Datasets	Input Data Types	Model	Prediction Task
DeepDSC [124]	CCLE [125], GDSC [126]	mRNA EXP, FP	Stacked AE + DNN	Drug sensitivity
CDRScan [127]	COSMIC [128], GDSC [126]	MUT, FP	5 CNN ensemble	Drug sensitivity
MOLI [129]	GDSC [126]	mRNA EXP, MUT, CNV	DNN subnetwork for each data type + final DNN prediction subnetwork	Drug sensitivity
TGSA [130]	CCLE [125], GDSC2, ZINC15 [131], ChEMBL [132]	mRNA EXP, MUT, CNV, CG	TGDRP (including a pre-trained GNN) + SA	Drug sensitivity
DRPreter [133]	CCLE [125], GDSC2	mRNA EXP, MUT, CNV, SMILES	2 GNNs + Transformer + MLP	Drug sensitivity
DeepSynergy [134]	Merck Compound Screen [135]	mRNA EXP, FP, drug chemical features	DNN	Drug synergy
MatchMaker [136]	DrugComb [137], GDSC [126]	mRNA EXP, chemical descriptors	DNN subnetwork for each drug + final DNN prediction subnetwork	Drug synergy
DeepDDS [138]	Merck Compound Screen [135]	SMILES, mRNA EXP	GNN + MLP + FCN	Drug synergy

MUT: mutation data; CNV: copy number variation data; mRNA EXP: mRNA expression data; SMILES: Simplified Molecular-Input Line-Entry System strings; FP: molecular fingerprints; CG: chemical graph; AE: autoencoder; TGDRP: Twin Graph neural networks for Drug Response Prediction; SA: similarity augmentation; MLP: Multi-Layer Perceptron; FCN: Fully Connected Network;

connectivity of atoms within a molecule, emphasizing the specific paths or bonds between atoms [139]. Circular fingerprints encode circular atom environments up to a given diameter [139]. Alternatively, end-to-end DL approaches can be used since they can learn relevant features directly from raw input data, eliminating the need for explicitly calculated descriptors and fingerprints.

Cell line features usually include gene expression data, which are typically continuous features that have undergone some form of normalization, copy number variations (CNV), somatic mutations, which are usually binarized (presence/ absence of an alteration), or other omics data (proteomics, metabolomics, etc.). Target information is typically employed to create features that indicate the pathways related to a specific target. The outputs of these models are values describing the drug dose-response relationship. For single drugs, this is usually half maximal inhibitory concentration IC_{50} and growth inhibition of 50% of cells GI_{50}, which can be binarized for classification tasks, or AUC. For drug combinations, the output value is usually a value that quantifies the effects of drug combinations based on a given reference model such as the Loewe additivity [114] or Bliss independence model [116].

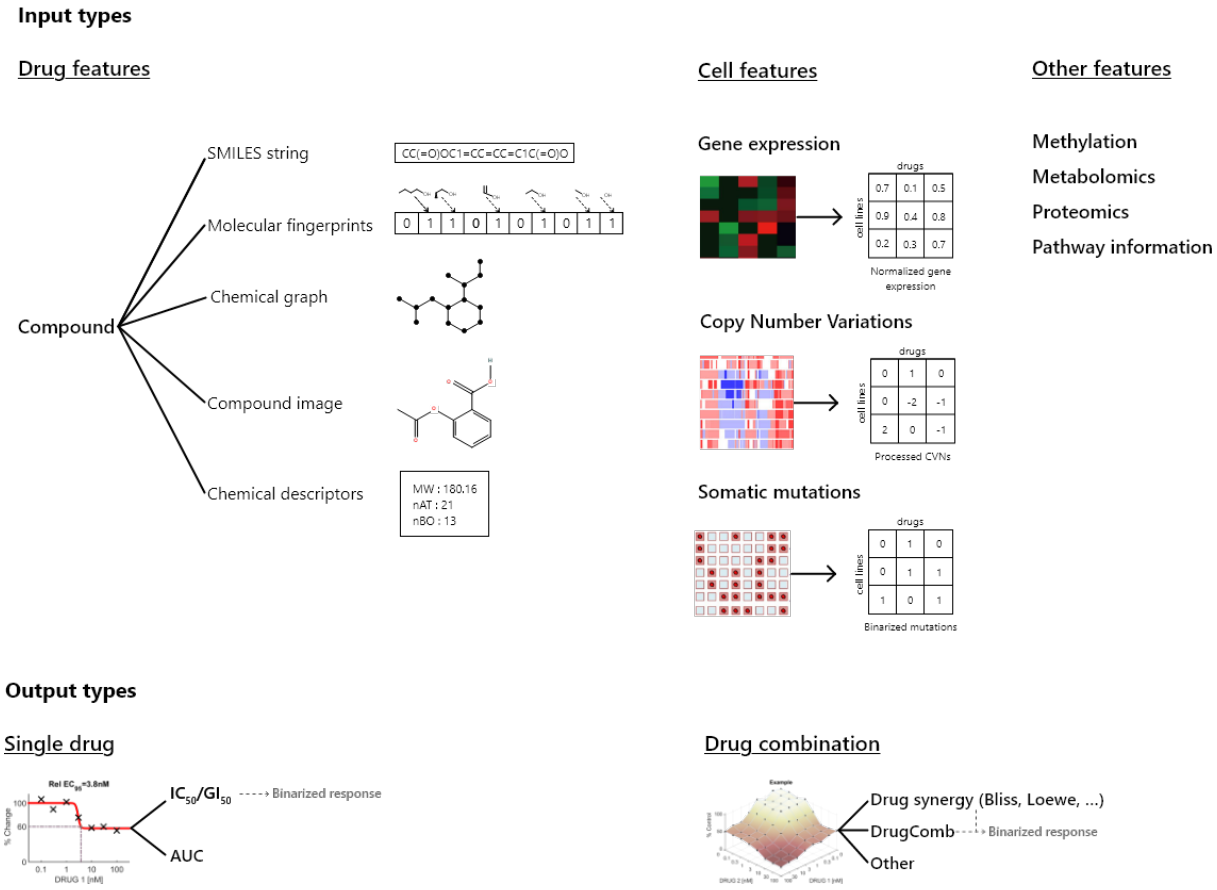
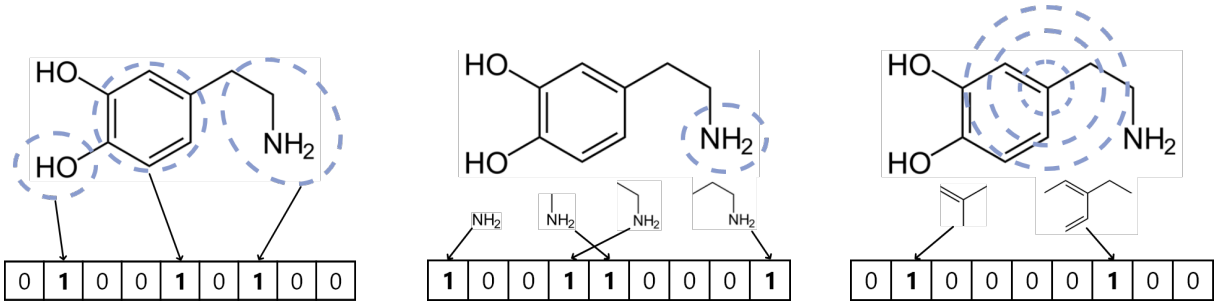


Figure 10: Common inputs and outputs used in deep learning-based drug response prediction models. Adapted from [10].



(a) Substructure key-based (b) Topological path-based (c) Topological circular

Figure 11: Compound represented as molecular fingerprints.

2.4.2 Predicting Single-Drug Sensitivity

The host group [10] used the data provided by the Cancer Target Discovery and Development (CTD^2) Beat AML challenge to assess how different input data types, preprocessing, and model architectures affect the results of drug sensitivity in acute myeloid leukemia. Besides drug sensitivity data, the available dataset contained other omics data, including high-confidence somatic mutations and RNA-Seq gene expression

data; and clinical data characterizing the patient-derived tumor specimens. These inputs were tested in four different combinations, each including all or a subset of the data types. Furthermore, from each combination derived variations with different data processing and modeling characteristics. A multimodal model was used with separate subnetworks for each data type, with each subnetwork responsible for encoding the features of that data type. The encoded representations from each subnetwork were then concatenated into a single tensor and fed into a final fully connected subnetwork, which was responsible for predicting the area under the dose-response curve (AUC) value for specimen-inhibitor pairs.

SMILES strings were retrieved from the corresponding inhibitor names using PubChem [141]. These were used to train fully connected layers when featurized using molecular fingerprints (ECFP4/ECFP6) or pre-computed Mol2vec embeddings, to train LSTMs when one-hot encoded, 1D CNNs and TextCNN models. When used, the gene expression data underwent feature selection, either selecting the N most variable genes, using the SelectKBest feature selector from scikit-learn, or protein-coding genes. The clinical data was used to train models containing the entire data or only a small subset. All of these model combinations and variations were tuned using Random Search with a maximum of 60 combinations of hyperparameter values, and the best-performing models were used in heterogeneous ensembles.

It was concluded that models trained with mutational data as input showed better performance. Feature selection based on univariate statistical tests (SelectKBest) and only using protein-coding genes instead of the N most variable genes when using gene expression data improved model performance. Including clinical features in addition to the expression, mutation, and drug features slightly improved performance. Finally, the type of architecture used for the drug subnetwork had less impact on the predictive performance of the models than the types of input features that were used.

Another approach to this problem was the DeepDSC model [124] using a stacked autoencoder and a feedforward neural network. This model uses cell line expression data and compound fingerprints collected from two public datasets, Cancer Cell Line Encyclopedia (CCLE) and Genomics of Drug Sensitivity in Cancer (GDSC), as input. Firstly, the gene expression data is fed into a stacked deep autoencoder producing encoded features with a much smaller dimension than the original data. The encoded features are then concatenated with the computed fingerprints and fed into a fully connected feedforward neural network.

The model was trained on both datasets individually, using 10-fold cross-validation, achieving relatively high predictive performance. The model was validated using leave-drug-out validation, where the model is trained on a dataset that includes all drugs except for one and tested on a test set that includes only the left-out drug. The performance of the model decreased significantly when tested on the leave-drug-out experiment, thus concluding that it was not able to effectively generalize to new compounds.

CDRScan [127] is an ensemble of five CNNs with different architectures. The input data types (mutation data and molecular fingerprints), were derived from the Catalogue Of Somatic Mutations In Cancer (COSMIC) cell line project (CCLP) and Genomics of Drug Sensitivity in Cancer (GDSC) databases. Four of the five CNNs adopt a dual convergence architecture where an independent series of convolutions were performed on each input data type. The two sets of convoluted features are then merged and subject to another series of convolutions before predicting the IC_{50} values for cell line-drug pairs from the GDSC. The last CNN adopts a conventional approach where both the input data types are trained together. The predictions of the five models are then averaged to define the final outcome of CDRScan.

Two out of the five individual models of CDRScan performed better than the final ensemble model, however, the authors argue that the overall generalizability of CDRScan is improved by the usage of an ensemble. Similar to [10], the authors claim mutation data does indeed have a significant influence on the responsiveness of anticancer drugs. Finally, the ensemble model was used to address the feasibility of drug repurposing using CDRscan, having identified 37 approved drugs (out of 1487) with potential new cancer indications.

MOLI [129] is a similar approach to [10], having different subnetworks for each input data type. It employs 3 DNNs, one for gene expression, one for somatic mutations, and one for CNVs to learn latent representations from different omics data. The learned features from the encoding subnetworks are concatenated and fed into a classification subnetwork, which predicts drug response. MOLI is a classification model that uses a combination of binary cross-entropy loss and a triplet loss. This model was trained on the GDSC dataset and validated using the PDX Encyclopedia and The Cancer Genome Atlas (TCGA) datasets.

The authors concluded that not only did multi-omics data achieve better results than single-omics data, but their later integration after representation was also superior when compared to approaches that merge different feature types before training. Furthermore, when trained on a drug-targeted input, MOLI achieved significantly better performance, suggesting that transfer learning can improve the prediction performance for the targeted drugs.

TGSA [130] applies Twin Graph neural networks for Drug Response Prediction (TGDRP) and a Similarity Augmentation (SA) module. This method uses gene expression, mutation, and copy number variation from the CCLE dataset and the IC_{50} values from the GDSC2 dataset. The Catalogue Of Somatic Mutations In Cancer (COSMIC) dataset was used as a feature selection method to select cancer-related genes. TGDRP is a two-branch network, where each branch is a GNN to represent cell lines and drugs individually. A drug is represented as a molecular graph that takes atoms as nodes and chemical bonds as edges. A

Graph Isomorphism Network (GIN) [142] is used to update atom (node) features through the message passing procedure. A cell line is represented as a gene–gene interaction graph. A GAT [143] is used as the message passing procedure in this branch. The drug and cell line representations are concatenated, and fed to a fully connected network to produce the final predicted IC_{50} values. TGSA uses a pre-training strategy on the drug branch GNN using node-level self-supervised on the ZINC15 dataset and graph-level multi-task supervised pre-training on the ChEMBL dataset.

A similar approach that revealed even better results than TGSA is DRPreter [133]. DRPreter is composed of three main parts: a cell line graph representation module, a drug graph representation module, and a drug response prediction module. Cell lines are represented as graphs where nodes are genes, and the edges represent the interactions between genes. Only genes involved in 34 selected cancer pathways were selected. Drugs are represented as graphs with atoms as nodes and chemical bonds as edges. Two GNNs were used to encode these graph representations. These resulting embeddings were then input into a transformer-based cell line and drug fusion module, where they were updated to reflect inter-pathway and pathway-drug relationships. After some more processing steps, the IC_{50} of the cell line-drug pair was predicted using a multi-layer perceptron (MLP), with the two graph-level embeddings as inputs.

2.4.3 Predicting Drug Combination Effects

The host group [10, 122] carried out a similar approach to single drug sensitivity in the context of drug synergy prediction by evaluating the impact of different preprocessing steps, types of input data, and DL architectures on the final performance of the methods. This time, the authors built models with two combinations of input data types: one using drug features for both drugs and gene expression data and the other with additional omics data, using mutation data and copy number variations. Once again, models were built using a multimodal approach, with each input data type having its own feature-encoding subnetwork. Each drug in a combination had a separate feature-encoding subnetwork, with weights being shared between the two drug subnetworks. The learned representations from each subnetwork were then concatenated and fed into a fully connected drug synergy prediction subnetwork.

The impact of the model architecture used to process the gene expression values was first tested, along with the influence of the arrangement and order of the genes (chromosome position order vs. clustering order). The results revealed that fully connected gene expression subnetworks trained using the original gene expression values (tabular format) outperformed 1D and 2D convolutional gene expression subnetworks, regardless of gene arrangement and order. Several other predefined gene sets were tested. The authors noticed that using prior biological knowledge for feature selection proved to be beneficial.

Furthermore, different drug representations were tested, with the model trained on ECFP4 fingerprints achieving the best performance. Finally, the authors found out that including mutation and CNV data slightly decreased model performance in this study.

DeepSynergy [134] was the first study in the literature to use deep learning to predict drug combination effects. DeepSynergy uses drug chemical features, fingerprints, and gene expression data to predict a drug synergy score (Loewe Additivity [114] score was used). The model is a DNN that receives concatenated vectors, including the features of two drugs and one cell line, as input. It was trained using data from the Merck Compound Screen. DeepSynergy performed significantly better than the machine learning models it was compared to; however, when presented with data from previously unseen cell lines or compounds, performance was considerably worse.

Since DeepSynergy, more recent approaches have been developed that do not involve concatenating all features into one input vector. MatchMaker [136] uses chemical descriptors for both drugs and cell line gene expression to predict drug synergy. The model was trained on DrugComb, reported to be the largest drug combination dataset, containing drug combinations from several studies, and using gene expression data gathered from the GDSC dataset. MatchMaker is composed of two subnetworks and a final prediction module. Each of the two subnetworks is a DNN sharing the same architecture that receives an input with each drug's features together with the cell line's features. The resulting embedding vectors with a representation of each drug conditioned on the corresponding cell line gene expression are concatenated and input to a final DNN that predicts the final output.

DeepDDS [138] is a deep learning model that classifies drug combinations as synergistic or antagonistic. The Merck Compound Screen dataset was used to train the model that receives gene expression data of cancer cell lines and drug SMILES, based on which the chemical structure of a drug can be converted to a graph, as input. The genomic feature representation of cancer cell lines is encoded by a multi-layer perceptron (MLP). Each of the drug's chemical structure is represented as a graph where nodes are atoms and edges are chemical bonds. The authors tested both a GAT and a GCN to extract the features of drugs. The cell line and drug embedding vectors are then concatenated and fed into a fully-connected network for the binary classification of drug combinations. A drug combination is predicted as synergistic or antagonistic if its synergy score (in this case, Loewe Additivity score) is higher than 10 or lower than 0, respectively.

The model using a GAT achieved slightly better results than the GCN one, however, both outperformed all the other methods used to compare DeepDDS. The model was also validated on leave-drug combination-out, leave-drug-out, and leave-tissue-out experiments and on an independent test set released

by AstraZeneca. The overall performance of all models decreased, which was to be expected. Finally, the authors tested the viability of DeepDDS in predicting novel synergistic combinations, by testing on combinations of targetted drugs approved by the Food and Drug Administration (FDA). The results of this experiment were consistent with the literature, having identified drug combinations observed in previous studies or under clinical trials.

Chapter 3

Software Development

3.1 Software tools used in the development

In this section, the tools selected for the development of this thesis and the reasoning behind these decisions are discussed. The main goal of this work is to develop a generic module, supporting multiple types of input data, and that uses NAS to consider DL models with different architectures and different hyperparameters. This module should be able to identify the most suitable model that can effectively handle the provided data.

To reach this goal, a set of decisions regarding the selection of tools and technologies had to be made, which involved both literature research and the evaluation and comparison of distinct software tools. These decisions include the underlying programming language used to develop the module, the deep learning library, and, most importantly, selecting an AutoML tool.

Since Omnia, the internal platform where the work will be carried out, already uses Python and this programming language is compatible with most deep learning libraries and AutoML tools, it was the most straightforward choice for this project. Python is a high-level, interpreted, and general-purpose dynamic programming language with an extensive package library and is widely used in ML projects [144].

PyTorch [99] and TensorFlow [98] are the most popular deep learning libraries, each with its own strong points. PyTorch was chosen as the DL library for this work mainly for its "pythonic" and object-oriented style, as well as its popularity in the research community. Looking at the data from Papers with Code, it is noticeable PyTorch's rise in popularity, with 60% of repositories implemented in PyTorch, and only 3% implemented in TensorFlow.

The most crucial decision to be made in this work lied in the selection of an AutoML tool. The initial step to determine if an AutoML tool was suitable for the project was to check if it included NAS capabilities. Thus, tools such as Auto-Weka [53], Auto-Sklearn [69], TPOT [94], and H2O [96] were immediately disregarded. The most promising AutoML tools with NAS capabilities included AutoGluon [65], Auto-PyTorch [145],

NNI [97] and Auto-Keras [82]. Since PyTorch was chosen as the deep learning library, Auto-Keras was removed as a possible candidate. Auto-PyTorch was also excluded due to its discontinued development, having received its latest update over a year ago. Thus, the main options for this subject were AutoGluon, which is already being used in Omnia, and NNI, which includes state-of-the-art algorithms and is actively maintained and developed.

In order to make an informed decision and choose the most suitable AutoML tool that aligns with the project's objectives and delivers the best possible results, further research was conducted to evaluate their respective capabilities. The following section aims to identify the key differences between these tools and compare their customization, and flexibility.

3.1.1 NNI and AutoGluon comparison study

NNI and AutoGluon are both open-source AutoML tools that offer NAS capabilities. Both tools can search various neural network architectures to find the best-performing one. However, there are some key differences between these two tools.

The main difference lies in how automatic the search for optimal deep learning models is. While AutoGluon requires minimal user interaction, i.e., NAS is built into the AutoML pipeline, NNI requires the user to define the model space, mutations, search strategy, etc. On the one hand, NNI offers much more flexibility and state-of-the-art algorithms for both search strategies and model mutations. On the other hand, AutoGluon uses a meta-learning approach (bagging), which allows the framework to learn from previous search iterations and apply this knowledge to future iterations. Furthermore, AutoGluon also includes ensembling.

NNI[1]

Search Strategies NNI provides an array of exploration or search strategies that can be divided into multi-trial and one-shot. The main difference is that one-shot strategies evaluate multiple architectures simultaneously by training a single supernet that encompasses all possible architectures, while multi-trial strategies evaluate architectures independently. Either approach has its strengths and limitations.

Multi-trial strategies include grid and random search as well as an evolutionary algorithm [101], a

[1] When the development of this module started, and for most of its duration, the latest NNI version was version 2.10. Therefore, the following information is regarding this version. However, it is important to note that a new version was released since then, and the project was updated accordingly, yet new functionalities were not explored. The new release addressed several critical issues, meaning updating the project was necessary. Consequently, some of the written information in this module might have changed in the newest version. It is recommended to refer to the latest documentation and release notes for the most up-to-date information and instructions.

Bayesian optimization algorithm [146], and a reinforcement learning algorithm [79]. One-shot strategies include a gradient descent-based algorithm [83], a reinforcement learning algorithm [147], and several others. Most of these algorithms are based on implementations present in state-of-the-art literature about this topic.

Finally, it is worth mentioning that instantiating an exploration strategy is very straightforward and requires only a couple of lines of code at most.

Search Spaces Search space refers to all possible neural network architectures that can be explored during the search process. NNI provides an API of Mutation Primitives that are utilities to build blocks in the search space. These mutation primitives are usually straightforward to use and understand. This library includes: `LayerChoice`, which selects from a type of layer; `ValueChoice`, which selects from some candidate values; `InputChoice`, selects from some inputs (tensors); `Repeat`, which repeats a block by a variable number of times; `Cell`, popularly used in literature [79, 74, 83], which is a sub-network that can vary in its operations and connectivity pattern, and some other cell variants [148, 149].

NNI also contains pre-defined model spaces from literature, denominated as `Model Space Hub`. One way to use the model space is to directly leverage the searched results, i.e., using state-of-the-art searched architecture along with its associated weights (transfer learning). Using these model spaces to search for a new architecture on a particular dataset is also possible. So far, the model spaces are all built for image classification tasks, with plans to add natural language processing NAS models such as NAS-BERT [150] and LightSpeech [151].

Other features NNI provides commonly used model evaluators for regression and classification tasks built upon the PyTorch-Lightning library. These can use this library's features, like callbacks, to rule out unpromising configurations early. It is also possible to use a custom PyTorch-Lightning evaluator. Finally, it is possible to visualize the running experiment using the dedicated frontend, which provides significant helpful information. TensorBoard is also integrated with NNI, allowing users to visualize and analyze the neural network architectures and performance metrics generated during the search process.

AutoGluon

Search Strategies AutoGluon's search strategies are limited to random search and Bayesian optimization.

Search Spaces Regarding search spaces, AutoGluon provides an API to introduce mutations into models. This module, named `Space`, includes: `Categorical`, which chooses from a list of categorical values; `Real`, search space for numeric hyperparameter that takes continuous values; `Int`, and `Bool`, which work similarly, used for hyperparameters of each value type.

AutoGluon has a built-in fully connected network that allows for specifying model space and hyperparameter values with a fixed value or a search space using the `Space` API. AutoGluon automatically chooses default settings for unspecified hyperparameters, which can also be fixed values or search spaces.

Other features Although not directly related to NAS, AutoGluon includes some features that distinguish it from NNI that are worth mentioning. AutoGluon's predictions are further enhanced by using both bagging and ensembling. AutoGluon automatically selects validation data from provided training set using stratified sampling. The fraction of the provided training data to hold out for validation can be specified. AutoGluon also offers interpretability, providing feature importance scores.

Customization and Flexibility

Both AutoGluon and NNI provide customization and flexibility options for neural architecture search. AutoGluon allows users to define their own custom models with the added benefit of bagging and ensembling when integrated with the typical AutoGluon workflow. Autogluon provides flexible hyperparameter tuning options, allowing users to specify constraints on the hyperparameters or use default values. The documentation does not mention the possibility of adding custom mutation primitives or search strategies.

NNI supports multiple deep learning frameworks, including TensorFlow, PyTorch, and MXNet. NNI allows users multiple degrees of customization. It is possible to define custom model mutations, custom evaluators (using plain PyTorch or PyTorch-Lightning), and even custom search strategies, providing interfaces to make these processes easier.

Limitations and other remarks

One main limitation of NNI, when compared to AutoGluon, is that trained models are not saved. Top models are exported as dictionaries of configurations, meaning that to evaluate predictions on a test set, it is required to instantiate the model and retrain it again before predicting on the test set. Furthermore, these discarded models could be saved for stacking, which NNI does not support. Although holding out a part of the training data for validation is not troublesome, NNI's evaluators do not automatically select validation data from the provided training set. NNI doesn't offer any interpretability features, which could

be a future concern, particularly for DL.

AutoGluon's main weakness in performing NAS lies in its limited API to mutate the model space. AutoGluon's `Space` module is comparable to a single NNI primitive (`ValueChoice`). In addition, AutoGluon does not support many search strategies, which is a big part of NAS. Also, AutoGluon has limited customization options. Adding custom mutators, search strategies, or evaluators is impossible. In comparison to NNI, the documentation of AutoGluon was somewhat insufficient. The documentation does not mention the `Space` module, nor does it have a dedicated section for search strategies.

Conclusion

This study compared NNI and AutoGluon NAS capabilities. Both tools provide significant benefits for automating the DL pipeline, saving time and effort for users. However, they also have their unique weaknesses. AutoGluon provides a user-friendly interface for non-experts to quickly build DL models. On the other hand, NNI offers greater flexibility and customization options, including many state-of-the-art features and algorithms.

Despite their strengths, both tools also have some limitations. AutoGluon only offers fully connected models for deep learning. Other deep learning architectures have to be programmed manually. Implementing custom model spaces is more straightforward on NNI. Arguably so, since NNI treats NAS as an independent module, whereas in AutoGluon, it is a part of the overall AutoML pipeline. Generally, NNI is more modular; it provides the pieces to build an AutoML pipeline. AutoGluon, on the other hand, offers this AutoML pipeline from the get-go.

Overall, AutoGluon offers a more automatic and easy-to-use approach to AutoML, and, despite its lacking NAS capabilities, it provides excellent results. On the other hand, NNI allows a more versatile and complete NAS development, with many implemented features present in literature.

In the context of this work, since the main objective is to use NAS to its full extent, NNI best aligns with this goal by offering a simple yet comprehensive library to build multiple DL model architectures. Further preliminary results comparing both tools in terms of performance were explored in appendix A.

3.2 Core platform

OmniumAI, the company where the work was carried out, has been developing an internal platform called Omnia with the aim of using AutoML techniques to create accurate ML models with minimal effort. Omnia includes tools to analyze biological, biochemical, and biomedical data such as compounds, proteins,

genes, metabolomics, and transcriptomics.

Omnia's main package includes generic objects that the remaining packages can use. It contains utilities such as the base objects for fitting and transforming data, validation methods for input data, the implemented models and base objects for setting up new models, the tabular predictor to fit multiple models to input data, the pipeline transformers to chain multiple transformers and read and write file utilities.

Other packages, such as the Compounds package, extensively use the methods provided by Omnia's main package to develop specialized methods for processing compound data, such as SMILES. This package contains feature extraction methods to extract meaningful information from SMILES strings, including molecular fingerprints, 2D and 3D descriptors, and Mol2Vec embeddings. It also includes compound standardization and molecular splitting methods for SMILES data. Similarly, the remaining packages offer relevant tools for their respective data types.

Omnia simplifies the process of transforming data and obtaining the most optimal model for the given dataset through its `Pipeline` object. The `Pipeline` object can chain multiple transformers and a single predictor in a pipeline. Transformers include featurizers and standardizers. Currently, only one predictor is available, the `TabularPredictor`, which uses Autogluon's class by the same name. The `TabularPredictor` class can be used on regression or classification tasks to make predictions on new data. A tabular predictor can fit several model algorithms to input data in a tabular format. The predictor can then predict the output variable or new data using the fitted models.

Furthermore, a tabular predictor can be used to perform hyperparameter tuning on a dataset. This will automatically fit all models on the dataset and use the hyperparameter search space (given or default) to find the best model configuration. It can train several configurations until the best model is found. It can also perform ensemble learning using bagging and stacking automatically.

Omnia's latest addition includes a pipeline hyperparameter optimization module that uses NNI to obtain the best-performing parameters in a configuration file. This file includes a search space, which can contain transformers and predictors with either default settings or an array of choices for their parameters. Alternatively, this search space can also be a preset with a degree of thoroughness (light, medium, heavy).

The models available in Omnia include various shallow ML models such as linear models, support vector machine, K-nearest neighbors; boosting models such as random forests, XGBoost, LightGBM, CatBoost; and some baseline DL models for tabular data. The first main contribution of this work to the Omnia platform was to extend its capabilities to support different types of data, rather than just tabular data, i.e., 2-dimensional and 3-dimensional data.

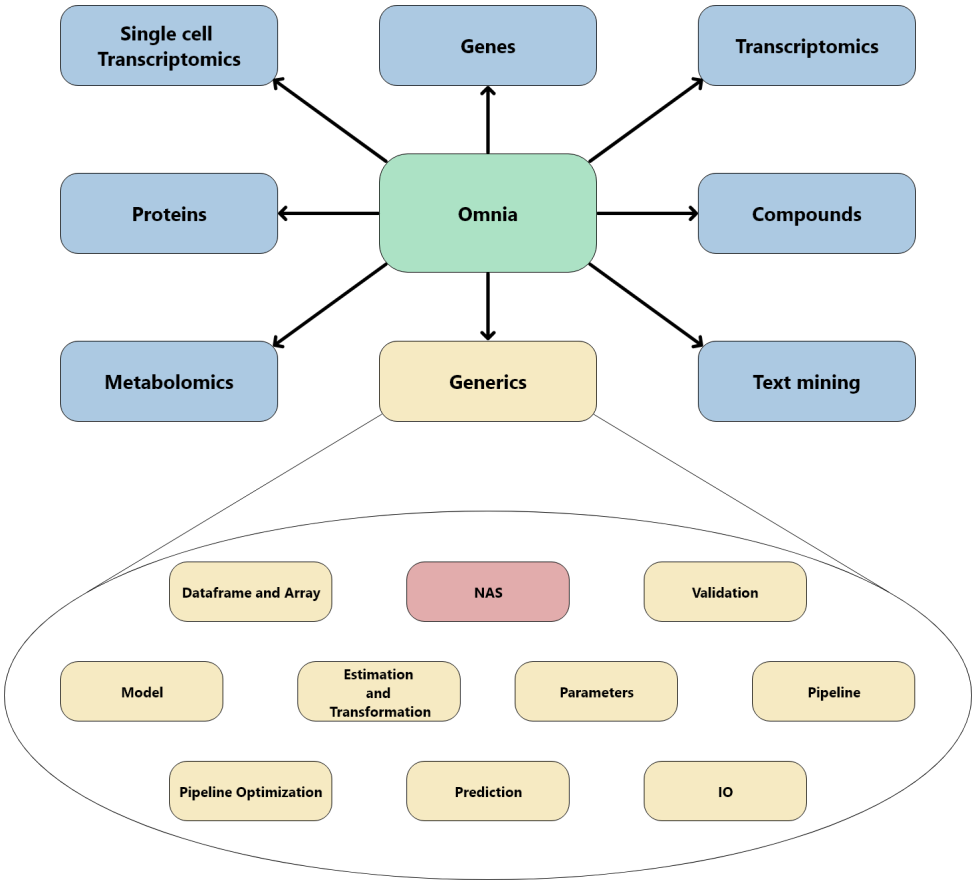


Figure 12: Representation of the main Omnia packages. Highlighted in blue are the already implemented packages, and in yellow is the working directory package where the new NAS sub-package was implemented.

The second main contribution was to improve the search for DL architectures and tuning of their hyperparameters. With Omnia's current functionalities, only a few DL model parameters are tunable, such as the number of layers, the hidden layer size, and whether to use batch normalization layers. Using NNI, it is possible to create much more complex and dynamic search spaces incorporating skip connections, different layer type possibilities for the same layer, using subnetworks with specific connectivity patterns and operations as building blocks that are repeated a dynamic number of times to construct the overall neural network architecture.

Adding a new predictor, named `NNIPreditor`, into the Omnia platform addressed these objectives and further enhanced its capabilities. Along with the new predictor, a more comprehensive range of DL algorithms and architectures became available, including DNN, 1D-CNN, CNN and RNN (as well as LSTM and GRU variants) models. This new module was developed in the main Omnia package since the predictor will be used in the `Pipeline` object.

Figures 12 and 13 illustrate an overview of the platform, highlighting the module added and how it

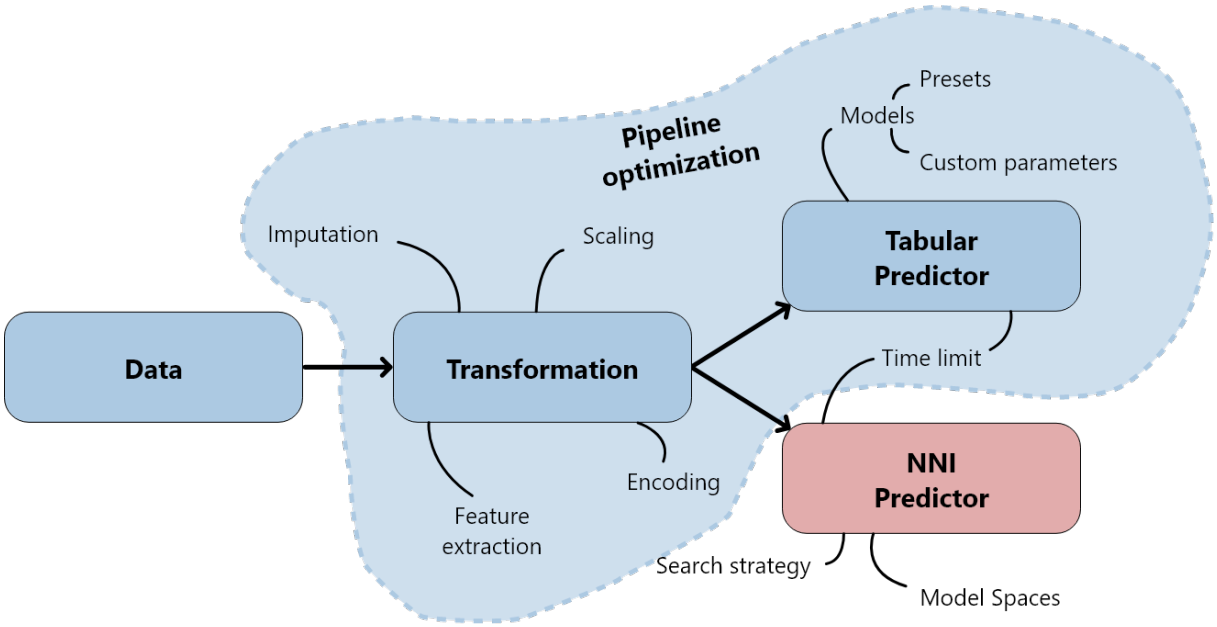


Figure 13: Illustration of the steps involved in an Omnia pipeline and the key concepts associated with each step. Newly added steps added as a result of this work are highlighted in red.

fits into the Omnia architecture; and the typical `Pipeline` process, highlighting the new steps where the `NNIPreditor` acts, respectively.

3.3 NNIPredictor

In this section, an initial overview introduces the implemented framework, highlighting the limitations of the current platform and the solutions and enhancements provided by this work. This initial subsection also describes the core concepts and components associated with the new module. The following subsection provides a more in-depth description of each of these main components.

3.3.1 Overview

The effectiveness of deep learning models is highly dependent on the architecture of the neural network they employ. The type and number of layers, the number of neurons in each layer, and the connections between the layers can significantly impact the network's performance. Thus, only using baseline DL models with little hyperparameter search is possibly hindering the potential of deep learning in this area.

Implementing this new predictor into the Omnia architecture encompasses many improvements. Available implemented DL models have simple architectures and often yield suboptimal results. This, coupled with the lacking model search spaces and the absence of models for non-tabular data, causes the

focus of Omnia on DL to fall short when compared to the current state and importance of this area. `NNIPredictor` improves on the existing simple fully-connected models for tabular data with more elaborate and diverse architectures based on state-of-the-art approaches.

This predictor also extends Omnia's current capabilities by allowing most data types, including higher dimensional ones. In the context where it is inserted within the Omnia pipeline, `NNIPredictor` comes after a data transformation step which might change the original input data shape to be appropriate for the different architectures. Based on the transformed data dimensionality, there is a 1D-CNN and an RNN model (which may have RNN, GRU, and LSTM layers), appropriate for two-dimensional inputs; and a CNN model appropriate for three-dimensional inputs. A hybrid model that fits any of these data types and combines the different layer types depending on the data dimensionality is also available. All of these models have complex and dynamic search spaces, which increase the probability of finding an optimal architecture for a given dataset.

Another key aspect of this predictor is its ability to handle multi-input data. It is possible to use `NNIPredictor` on a dataset containing multiple input data types by using the dedicated model for this feature. This model creates a subnetwork for each given input, and then uses the previously mentioned models to extract features from the corresponding input data. The resulting learned embeddings are concatenated before passing through subsequent layers for further processing and final prediction. This method allows the model to leverage information from different data types, leading to potentially improved performance and better utilization of the available information.

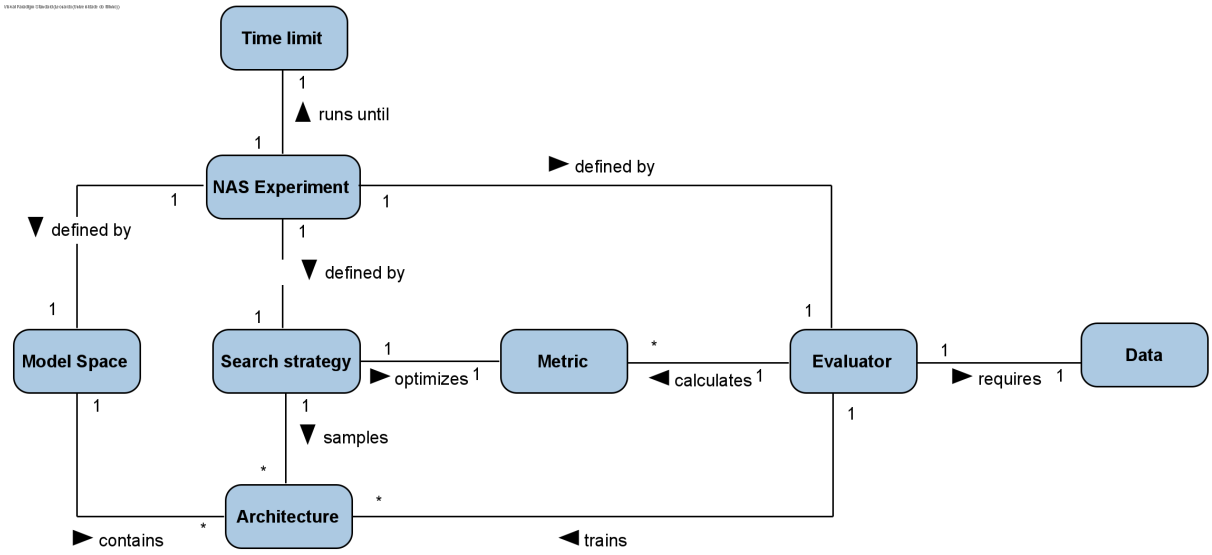


Figure 14: NAS domain model including main concepts.

Finally, one of the main advantages of this module is how straightforward it is to use. In its simplest

form, it only requires the input data, inferring the rest automatically. The problem type (regression, binary, multi-class) and the output size are deduced based on the target column values. The predictor uses all the models that fit the given dataset if not specified. Although it is possible to set a considerable number of parameters, i.e., the search strategy to use, time limit, number of epochs, batch size, all these values have a default in order to make its use as low-code as possible, thus following the same goal as Omnia.

To further understand `NNIPredictor`, it is essential to understand some main NNI components. There are three main components for a NAS task, namely, a model search space which defines all the possible configurations for a model; a search strategy which is an algorithm dedicated to exploring the model space; and a model evaluator that reports the performance of every model tested in that space.

Figure 14 shows the main system elements and their relation. A `NAS Experiment` is a process that trains and evaluates multiple models until a `Time limit` is reached. A `NAS Experiment` may also end if a maximum number of configurations is tested or the model search space is exhausted. The experiment is defined by a `Model Space`, a `Search strategy` and an `Evaluator`. A `Model Space` is made up of many `Architectures` which the `Search Strategy` explores. The `Search Strategy` samples `Architectures` from the `Model Space` in an attempt to optimize a `Metric` (i.e., achieve a better performance than the last sampled architecture). These sampled `Architectures` are trained and validated by the `Evaluator`, which, based on some `Data`, calculates `Metrics` to evaluate the performance of the trained `Architecture`.

Figure 15 shows a class diagram for `NNIPredictor`, including the most relevant classes and their interactions. Within the `NNIPredictor` package, there is a `Model Space Hub` package which includes all the created model spaces. The `Evaluator` class is used to set up and generate the correct evaluator based on the problem type and other training parameters. This class has a subclass, `LightningModule`, which inherits the PyTorch-Lightning class by the same name. Its purpose is to define the training logic. The `NasExperiment` class inherits the NNI class by the same name and adds an abstraction layer to the process of starting an experiment and exporting results. This class requires objects of two other NNI classes to start an experiment, the `Strategy` class, which includes all the search algorithms, and the `ModelSpace` class, which all the developed models inherit. The main class, `NNIPredictor`, creates as many `Nas Experiment` as model spaces inferred or specified and then selects the best model architecture out of all experiments.

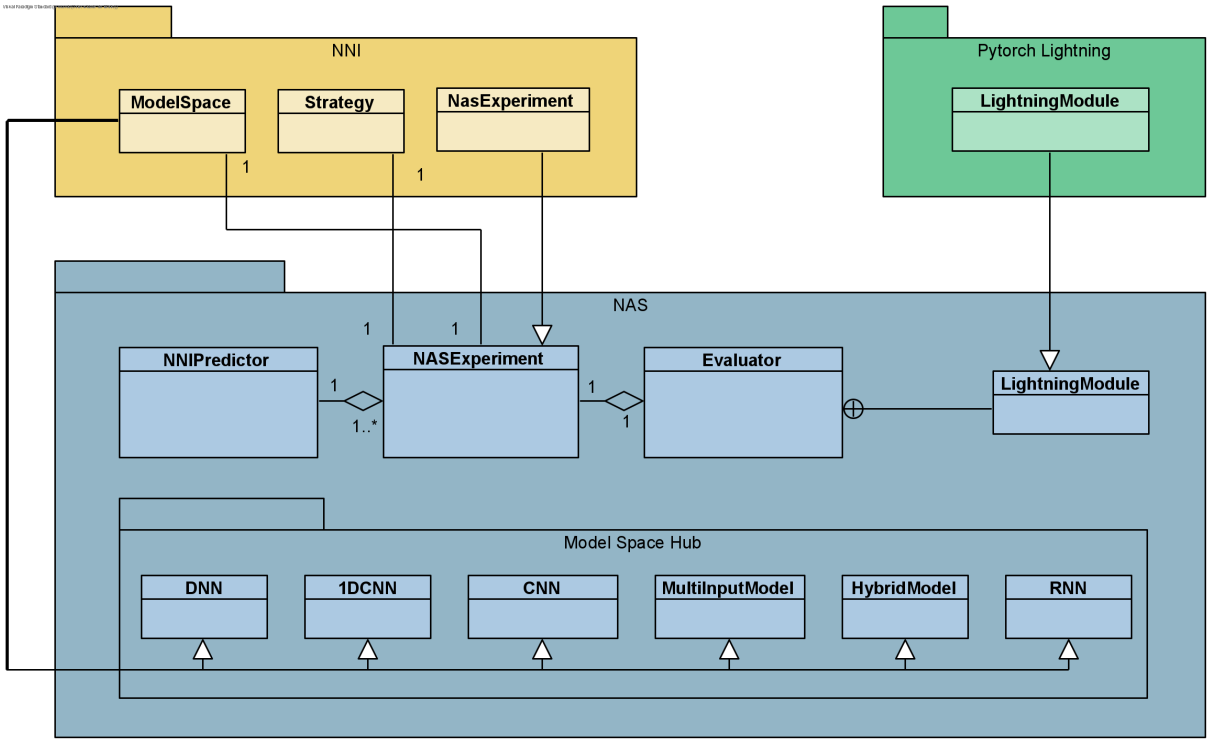


Figure 15: NNIPredictor Class diagram.

3.3.2 Implementation

Model Spaces

A model space can be seen as a base model and possible mutations on the base model. It is defined to express a set of architectures that contains potentially good-performing models. Different architectural choices can be made within this space, such as selecting between a Max Pooling or an Average Pooling layer for a specific layer in the model. Each choice introduces variation in the model's structure, resulting in different outcomes depending on the characteristics of the data being processed. By exploring these sets of options within the model space, the chances of finding models that are best suited for a particular task or dataset are significantly increased.

Some of the model spaces implemented are adapted from state-of-the-art models, while others follow a readability approach, prioritizing simplicity. An essential aspect of these models is their applicability to regression, binary, and multi-class classification tasks. Typically, models for different tasks use different activation functions on the last layer. In regression, the activation function is omitted on the last layer (i.e., uses linear activation) since these tasks aim to predict a continuous value where a constrained output like that of tanh and ReLU are not appropriate. For binary classification tasks, it is common to use a sigmoid activation on the last layer. This is because the output of the sigmoid function represents the probability of

belonging to the positive class, which makes it easy to classify the data by setting a threshold value (e.g., if the output is below 0.5, it belongs to one class, otherwise it belongs to the other class). The softmax activation function is typically applied on the last layer of multi-class classification tasks. This activation function returns a probability distribution over multiple classes, which add up to 1. These probabilities can be seen as the likelihood of a sample belonging to each respective category.

Initially, these models received the problem type as a parameter and used it to set the last activation function. However, in order to improve readability and easier interpretation, every model omits the last layer activation function. Instead, loss functions that incorporate the activation function as a part of their algorithm are used during training and the activation functions are applied externally to the model's output when making predictions.

As for the hidden layer activation, currently, the most successful and widely-used activation function is ReLU. Using NNI, this hidden layer activation function could be one of the choices within the model search space. NNI also provides an `AutoActivation` module that implements the work developed by Prajit R. et al [152]. This module suggests using automatic search techniques to discover new activation functions. Regardless of these possibilities, it was determined that the potential benefits of incorporating these strategies would not sufficiently compensate for the significant increase in the search space size. Given the magnitude of possible architectures for each model, further growth would only prolong the process of reaching an optimal configuration.

Despite numerous attempts to replace ReLU with hand-designed alternatives, none have proven to be consistently better. The implemented models in `NNIPredictor` use the Swish (implemented as Sigmoid Linear Unit (SiLU) in Pytorch) activation function. This activation function is similar to ReLU and was found to slightly improve results on deeper models across a number of challenging datasets [152].

DNN The recent interest in DL has shown great success in handling unstructured data such as text, image, and audio. However, DL still sometimes falls behind when it comes to predicting tabular data. Currently, state-of-the-art models for predicting tabular data include gradient-boosting ensemble models such as XGBoost [18], LightGBM [19] and CatBoost [153]. These models generally outperform DL models.

A recent study [154] reviewed the latest DL models for tabular data and compared their performance against these popular boosting methods using multiple datasets. Although this study did not use any form of hyperparameter search, which could have influenced the outcomes, the results revealed that only some DL models were more accurate than boosting models in classification tasks, while none outperformed boosting models in regression tasks. Furthermore, the computation time of the tested DL models was far superior to the counterpart.

In a different study [155], Gorishniy et al. overviewed and compared the state-of-the-art DL models and introduced two strong baseline architectures. This study also compared the best-performing DL models with the gradient-boosting models. The main takeaways from the DL comparison stated that one of the introduced baseline models, a ResNet-like architecture [75], turns out to be an effective model that none of the competitors can consistently outperform. According to the results, the ResNet model showed the second-best average rank across all datasets tested. Regarding the DL comparison against the gradient-boosting methods, there is still no one-size-fits-all solution, with the latter still dominating some tasks.

The model space implemented into `NNIPredictor` was based on the ResNet-like architecture introduced in the Gorishniy et al. study [155]. There were multiple reasons for selecting this particular model. It showed consistent and competitive results even when compared against the latest DL models. Despite having achieved good results, its architecture is relatively simple to understand and does not involve any complex or costly operations, thus avoiding large computation times.

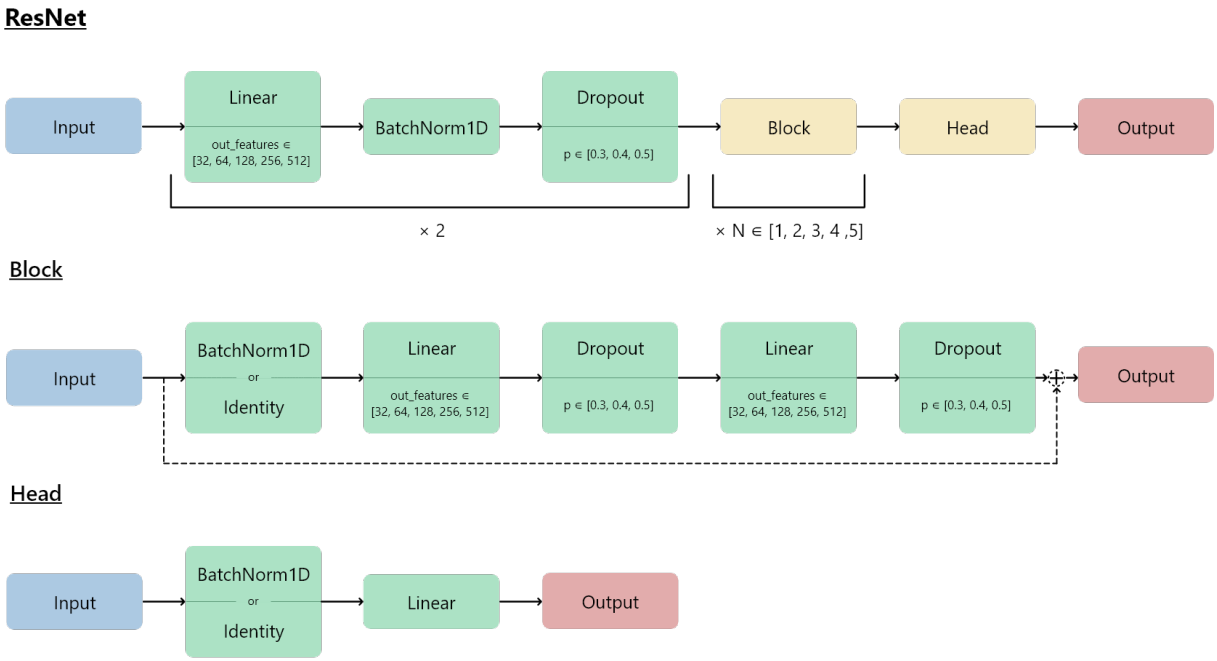


Figure 16: Overview of the DNN model space, including some of the primary possible configurations. The top image represents the overall model space architecture; subsequent images detail individual model components.

Figure 16 visually represents this model space. The model initiates with linear, batch normalization, and dropout layers where the hidden size and dropout probability are chosen from a list of possible values. Following this, the model repeats a Block a dynamic amount of times. This Block comprises a combination of normalization, linear, and dropout layers. In this case, the normalization layer can be omitted if, instead,

an identity layer is chosen. Additionally, this block can perform an element-wise addition between the transformed tensor and the original input tensor, known as a residual connection [75], to help the model learn the difference between the initial input and the transformed output more efficiently. Instead, it may simply use the transformed tensor. Note that the decisions made by each Block may not be the same. Finally, the architecture ends with another component, the Head, composed of an optional normalization layer and the final output layer.

CNN Recent development on convolutional networks using NAS has moved on from designing search spaces as a list of sequential layer-wise operations, such as the early work of Zoph and Le [79] and Baker et al. [156], to searching for the best cell structure, where each cell contains several operations. These cells are then repeated multiple times [74, 83, 149].

A cell structure represents a building block, a sub-network, repeated multiple times to create a complete network that can vary in its operations and connectivity pattern. A cell consists of multiple nodes, each representing a combination of a set of operations (e.g., convolution, pooling). In particular, a cell takes outputs from previous cells as input, and each node takes as input previously constructed nodes (or the cell inputs), applies an operator to each input, and combines the output of the two operators (e.g., summing). The output of the cell is the concatenation of some or all of the nodes in the cell.

A NASNet search space [74] defines the architecture of a convolutional network using this approach, and uses NAS to learn the structure of two types of cells. Normal cells are convolutional cells that return a feature map of the same size as the input. Reduction cells are convolutional cells that return a feature map of half the input size. These cells are then stacked to build a network. In this study, each cell comprises five nodes, and each node is defined by different discrete choices, including selecting the input for each operation from the previous states, selecting the operation to apply, and how to combine the operation's output.

In a different study, NasBench-201 [149], each architecture consists of a stack of searched cells interspersed by residual blocks that downsample the feature map's size and double its channels. Each cell in the search space is represented as a densely connected DAG. For every $i < j$, there is an edge from i-th node to j-th node. Each edge in the DAG is associated with an operation that transforms the hidden state from the source node to the target node and each node represents the sum of all hidden states of the edges pointing to that node.

The model space implemented into `NNIPredictor` is based on NASNet and NasBench-201. Figure 17 shows the macrostructure of each candidate architecture. The model space uses NASNet's CIFAR-10 architecture containing three Normal cell blocks connected by Reduction cells; however, it uses the cell

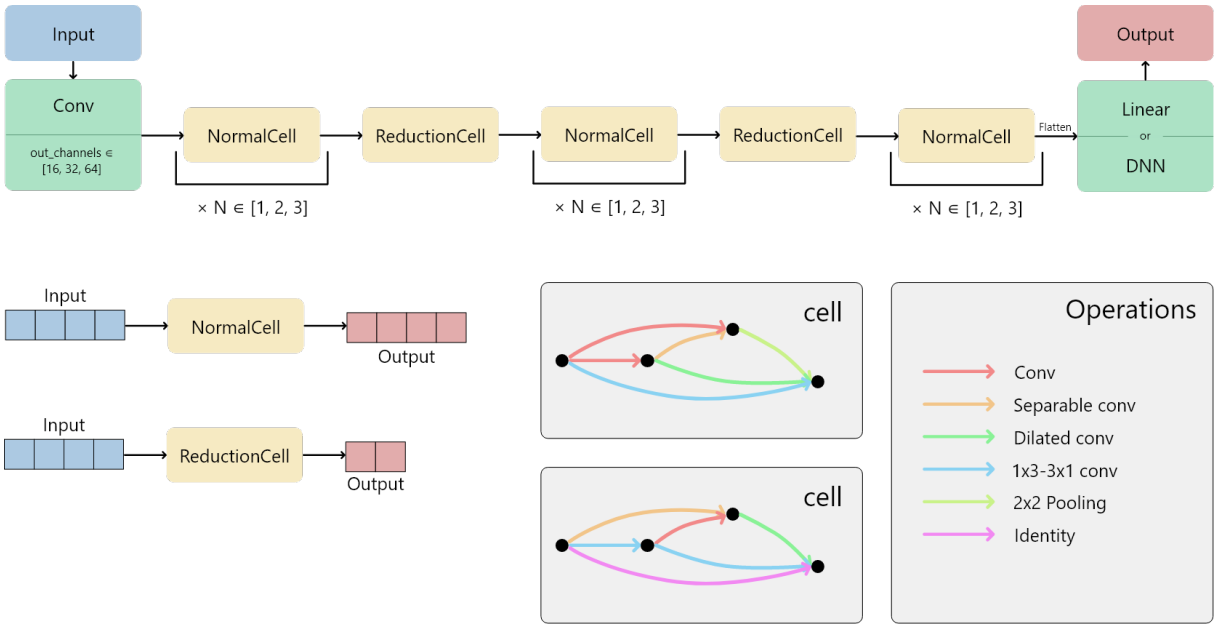


Figure 17: Top: Overview of the CNN model space. Bottom-left: Normal cells keep the input dimension while Reduction cells divide it. Bottom-right: Examples of Cells with 4 nodes. Each cell is a directed acyclic graph (DAG), with edges representing operations chosen from a preset operation set. Adapted from [149].

structure variant proposed in NasBench-201 with each cell being a DAG and each edge being associated with an operation, as shown in Figure 17.

All possible operations are selected from a predefined operation set containing a combination of operations used in NASNet and NasBench-201. The implemented operation set includes the following operations: identity, convolution (1-by-1, 3-by-3 and 5-by-5), depthwise separable convolution (3-by-3, 5-by-5 and 7-by-7), 3-by-3 dilated convolution (with different possible values of dilation), 1-by-3 followed by 3-by-1 convolution and 2-by-2 average and max pooling. Note that the convolutions in this operation set include activation, convolution, and batch normalization operations, which were omitted for simplicity. The final part of this model can be a linear layer or a fully connected block, using the implemented DNN model space.

1D-CNN While CNNs have been widely used in various image and video processing tasks, achieving remarkable results, its use in tabular data is rare. The effectiveness of CNNs on these tasks derives from their ability to exploit the spatial arrangement of data, capturing local patterns in input data.

Input images have two essential characteristics: local connectivity and spatial locality. The convolutional kernels used by CNNs are able to leverage these properties to extract features. Local connectivity means that all of the inputs that go into a given kernel are within a small, localized portion of the input

image. Spatial locality means that inputs where the convolutional kernel is applied are highly correlated. Therefore, processing these elements together allows the network to extract meaningful feature representations. By exploiting these characteristics, a convolutional kernel can learn to identify elements like edges, shapes, gradients, and textures.

It is possible to apply a one-dimensional convolutional layer to tabular data, however, this layer acts as described above and is designed to exploit a spacial locality correlation among features. This means that the convolutional kernel expects that adjacent columns are spatially correlated, i.e., their relative position has meaning. This is not true for most tabular datasets. Consequently, the benefits of using such techniques are not reaped.

In a Kaggle challenge meant to improve drug development by developing algorithms that accurately predict the Mechanism of Action (MoA) of a drug based on its cellular signature, a one-dimensional CNN model obtained the 5th position by itself, without using any ensemble methods, such as the higher placed competition [157]. This model used an interesting approach to deal with tabular data's lack of spatial locality. The original input features are first passed through a fully connected layer to increase their dimensionality and then reshaped into multiple channels, each representing a signal of a specific size. These signals correspond to groups of ordered features, each capturing a different ordering pattern. However, the values observed in these signals are not direct representations of the original features but rather a non-linear combination of them. Thus, the original features have been re-ordered in a meaningful way for the following convolutional layer.

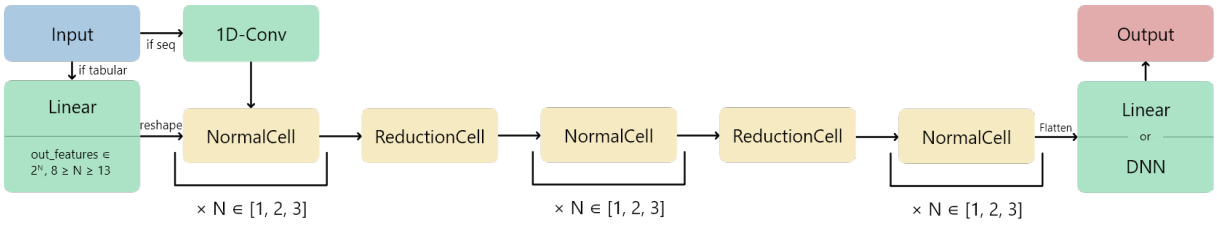


Figure 18: Overview of the 1D-CNN model space.

Figure 18 represents the 1D-CNN model implemented into the `NNIPredictor`. This model space was designed to be compatible with both tabular and sequential data. For tabular data, the model uses the same approach as [157] to re-order the input data's features. For sequential data, the model starts with a simple convolutional layer. The subsequent architecture components adapt the implemented CNN model space to fit these data types, hence using 1D layers instead of 2D.

RNN Although there have been some instances of NAS applied to RNNs, these tasks have predominantly gravitated towards the highly successful Transformer architecture instead of its RNN predecessor due to its superiority in terms of model performance [158].

The existing RNN model spaces found in research literature and other well-known RNN-based architectures, even when not directly related to NAS, tend to be complex and tailored for specific tasks, making it difficult to adapt and integrate them with NNI. To overcome these difficulties, it was decided to develop an RNN model from scratch rather than adapting a state-of-the-art model, allowing for easier integration with NNI.

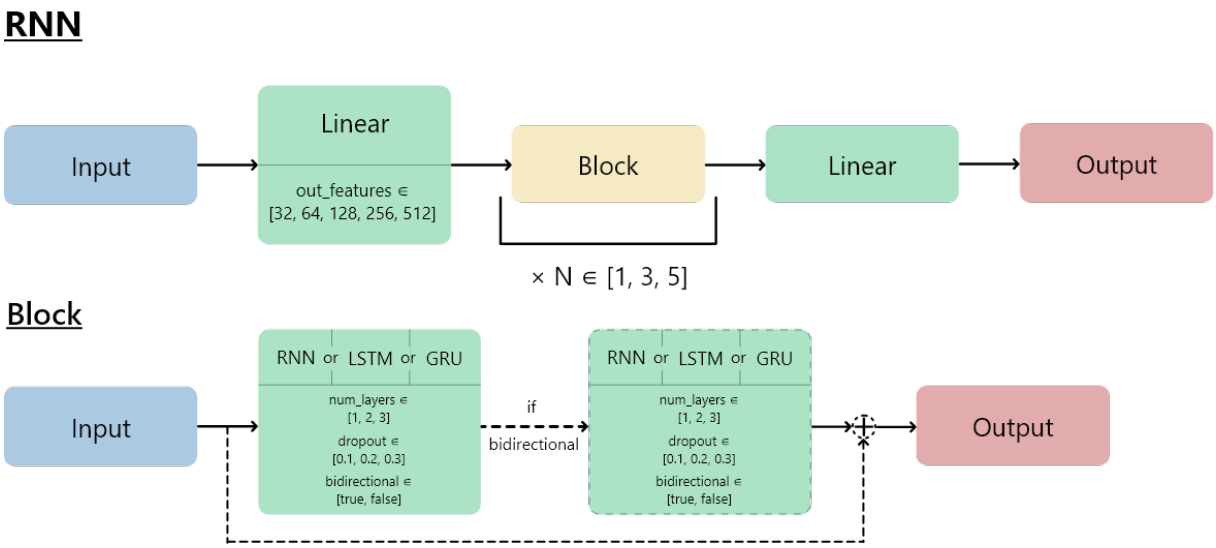


Figure 19: Overview of the RNN model space with some of the possible parameter configurations.

Figure 19 represents the RNN model space implemented into the `NNIPredictor`. The model consists of an initial linear layer, followed by a variable number of RNN blocks, and a final decoding linear layer. The RNN block contains one or two RNN layers, which can be either bidirectional or unidirectional. When the first layer is bidirectional, the resulting hidden size is doubled; thus, to keep the hidden size constant, the second layer is unidirectional and halves the (doubled) hidden size. There is no second RNN layer when the first layer is unidirectional. The RNN layers can be either RNN, GRU, or LSTM layers. Additionally, each block contains a possible residual connection.

HybridModel The final single-input model implemented is a hybrid model capable of handling any of the supported data types. This model intuitively adjusts its architecture based on the dimensions of the input data. It can integrate one- and two-dimensional convolutional, recurrent, and linear layers. This approach allows the model to capture the different complex patterns present in the input data by exploiting each layer type's inherent characteristics and strengths.

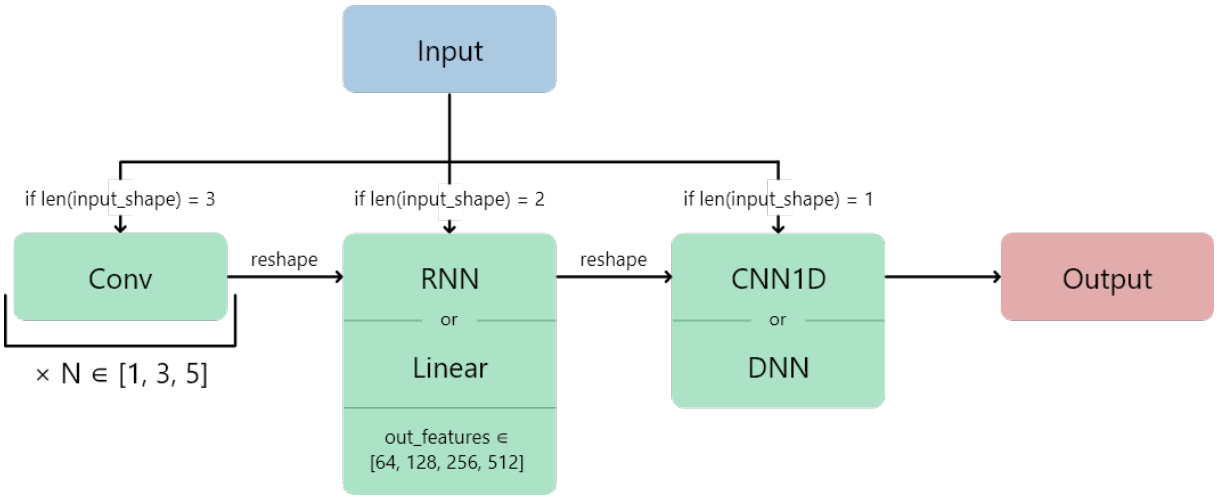


Figure 20: Overview of the hybrid model space.

This model space employs the previously described models in its architecture except for the CNN model since it outputs a one-dimensional embedding, which would not be suitable for the (possible) following recurrent layers. Figure 20 shows a representation of the hybrid model. This model adapts itself according to the shape it receives as a parameter. If the given shape is three-dimensional, the model starts with a group of convolutional layers repeated a variable amount of times depending on the height and width of the input data. The convolutional layers are followed by either a recurrent layer segment (the implemented RNN model) or a singular linear layer. The final part of this model is a choice between the implemented 1D-CNN or DNN model spaces. The resulting features are reshaped in each step according to the decisions made.

Multi-input model Previously described models were designed to work on a single data type. However, real-world datasets often contain multiple data types that can complement or enhance the understanding of each other. Multimodal models are designed to handle multiple data types, such as tabular, text, image, etc., and adequately capture the unique characteristics and patterns in each. These models are able to combine and integrate these various data types to generate more informative models and more accurate predictions.

Multimodal models are typically composed of three parts. At first, each input is fed into its own corresponding encoding subnetwork. Each input is processed by these subnetworks that extract and convert features into a common representation for subsequent layers. The next step is to combine the learned representations from each input into a single representation. A common approach is to concatenate the representations. Finally, the joint representation is fed into a final fully connected prediction subnetwork.

The implemented model, represented in Figure 21, uses all previous single-input models as subnet-

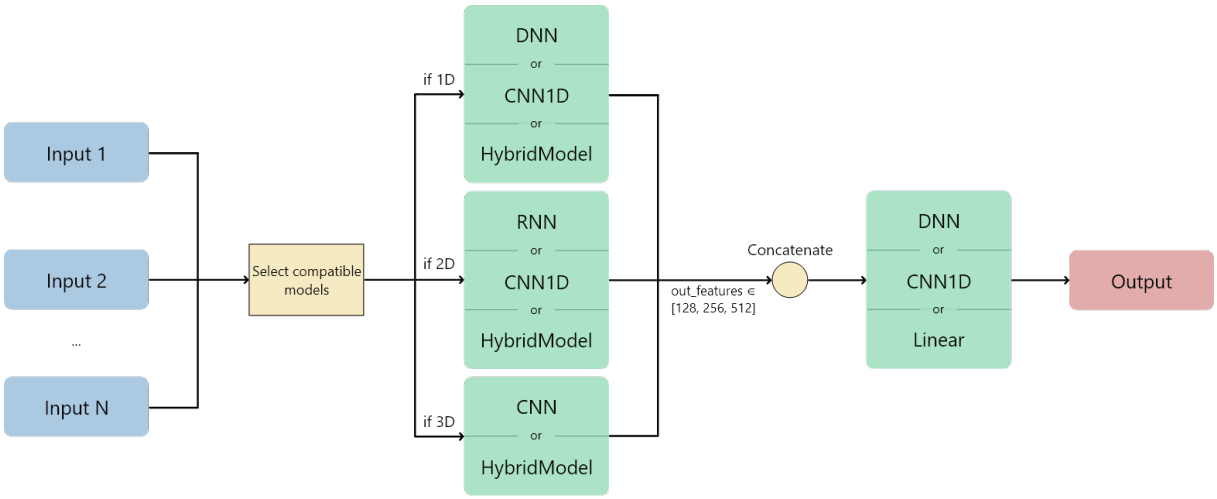


Figure 21: Overview of the multi-input model space.

works. For each input, a selection is made among the available single-input models that are compatible with that specific input. For instance, if the inputs resulting from the previous Omnia pipeline data transformation step consist of tabular data and 2D data, the model will consider the DNN, 1D-CNN and HybridModel for the tabular data subnetwork, and the RNN, 1D-CNN, and HybridModel for the 2D data subnetwork. The outputs of each subnetwork are concatenated and fed into a final prediction subnetwork, which can be either a single linear layer or the DNN or 1D-CNN model spaces.

Data readers

One of this work's main advantages is its versatility regarding data types. The data readers are a few functions dedicated to reading the train and test data. These functions read a dataset from a Pandas DataFrame or Numpy array, split it into training and validation sets (this division is exclusive to the training data reader), and convert the data to PyTorch tensor datasets. This process is not as straightforward as it may seem since these functions account for both Pandas DataFrames and Numpy arrays and must handle single or multi-input data and several problem types. For instance, considering the multi-input data scenario, the input DataFrame has to be divided accordingly, and each of these segments must then be transformed into tensors before being used to generate a PyTorch tensor dataset. When dealing with a specific problem type, the label tensor might need to be converted to a particular data type (e.g., float or long).

While the test data reader is only used to predict and score the final resulting model, the Evaluator uses the training data reader on each model configuration to retrieve the tensor datasets (train and validation). NNI requires serializing this and other Evaluator elements, such as the DataLoaders. Serialization allows

for the model evaluator to be re-instantiated on another machine or process. Both the model and its evaluator must be properly serialized.

To make NNI correctly serialize the model evaluator, `nni.trace` decorator should be placed on some of its functions and objects. Whenever `nni.trace` is placed on a function, its arguments should never be a large object since NNI will serialize these parameters and there is a size limit on serialization. Therefore, the training data reader cannot have the input DataFrame or array as a parameter. Instead, it is recommended to have all the data construction process inside the function (i.e., read a CSV inside the function).

While this should not be a problem for many NNI users, in the context of this work, `NNIPredictor` is placed within a pipeline after data transforming. This means that the data in the original dataset file given to the Omnia `Pipeline` object differs from the data that reaches this point. It is also impossible to save the training data as a global object and then use it in the training data reader because when the model evaluator uses this function, it won't "remember" this object. Given all these constraints, a viable solution was found, which involves using the joblib library to cache the data when `NNIPredictor` is initialized and then reading the cached data during the training data reading process. This approach avoids all of the above-mentioned difficulties.

Strategy

Similar to the tests conducted in appendix A, only multi-trial strategies are available in the new predictor. These strategies involve training each sampled model from the model space independently. NASNet [74] is an example of multi-trial NAS. Multi-trial NAS requires a model evaluator to assess the performance of each sampled model and an exploration strategy to sample models from a defined model space. One-shot strategies were not considered since they only support a limited set of NNI mutation primitives.

A critical characteristic of these search strategies is that they were all designed to maximize the specified metric. However, since maximizing a metric doesn't always lead to the best results (e.g., mean squared error), these metrics are inverted to work around this aspect.

Evaluator

This class returns an adequate model evaluator for the given dataset and parameters. A model evaluator is used to train and validate generated models, evaluating their performance. There are several ways to implement a model evaluator using NNI: using plain PyTorch to implement the training code and wrapping it around the dedicated NNI class for this scenario, using the built-in evaluators integrated with the PyTorch-

Lightning library, or building a custom evaluator using PyTorch-Lightning.

Given the number of different cases this evaluator has to handle and the convenience of PyTorch-Lightning, the decision was made to implement the final option. PyTorch-Lightning is a PyTorch wrapper that provides a simplified interface to train models and significantly reduces the boilerplate code involved with writing vanilla PyTorch training logic. PyTorch-Lightning has two main components: the `LightningModule` and the `Trainer`. The `LightningModule` is in charge of all the training details, including defining training and validation loops (receives batch data, does the forward pass, calculates loss, and returns it) and optimizers. Once the `LightningModule` is defined, the `Trainer` is used to fit the model. It has numerous configuration options, such as the number of epochs, GPUs to use, and callbacks.

Defining a custom NNI evaluator with PyTorch-Lightning involves defining these two main components and having the training module inherit an NNI class. In addition, evaluators communicate with search strategies via two API calls, one for intermediate metrics results called after every validation epoch ends and the other to report the final metrics result after fitting has ended. Then, all of these components (including `LightningModule`, trainer, and dataloaders) need to be wrapped into an NNI `Lightning` object, which is the object passed onto the NAS experiment.

The training and validation steps are slightly different depending on the problem type. If the batch received contains more than two elements, the inputs and target must be separated differently, indicating a multi-input problem. Following this step are the forward pass, loss calculation, and metric calculation. These steps vary slightly depending on regression, binary, or multi-class tasks. As described previously, the loss and metrics are reported back to NNI after each validation epoch and after training has ended. Although the metrics reported can be a dictionary, only the metric marked as default will be used by the search strategy to sample a new architecture.

The default metric used by the search strategy can be set as a parameter in the predictor. The available metrics are those calculated during training and validation (mean squared error, root mean squared error, mean absolute error, coefficient of determination, Spearman correlation, and Pearson correlation for regression tasks and accuracy, precision, recall, f1, and AUC-ROC for classification tasks). If a default metric is not set, the loss will be used to guide the search strategy algorithm. An important step in this process is to invert certain metrics if they are to be used by the search strategy since these algorithms are designed to maximize it (e.g., loss or mean squared error are multiplied by -1).

The choice of the loss function depends on the nature of the problem at hand. For regression tasks, mean squared error [159] is used as the loss function. For binary tasks, binary cross-entropy with logits

[160] is used, and for multi-class tasks, cross-entropy [161] is the chosen loss function. It is important to note that the binary cross-entropy with logits and cross-entropy metrics incorporate a sigmoid function and a logarithmic softmax function, respectively. These specific loss functions, which incorporate activation functions within their algorithm, were deliberately selected due to the absence of activation functions in the final layer of any of the model spaces implemented.

As for the implemented `Trainer`, it sets the maximum number of epochs, the number of and what device to use, and callbacks. Early stopping is implemented with a patience of twenty epochs. This implies that if there is no noticeable improvement (at least 0.0005) in the loss for twenty epochs, that model will stop its training, thus discarding unpromising architectures. Additionally, a learning rate finder is set to optimize initial learning for faster convergence.

The same evaluator used to train the model architectures within the model space can also be used to retrain the best-performing model discovered on the entire training data. In this scenario, the training and validation datasets are combined and placed into the train dataloader, while the validation dataloader is left empty.

NAS Experiment

A NAS Experiment is a class built to find the optimal model architecture for a singular model space. This class inherits NNI's class by the same name and acts as a further abstraction layer. Its main functionality is setting up and starting an experiment, which, as mentioned before, requires a model space, a search strategy, and a model evaluator. A NAS experiment is completed when the search space is exhausted (i.e., every possible configuration has been tested), when a time limit is reached, or when a set number of maximum trials is reached.

It is also possible to provide an optional configuration file to be used by the NAS experiment. This configuration file might contain specific settings, such as the maximum duration for each trial or the upper limit for the number of trials, which aren't intrinsic `NNIPredictor` parameters. It may also contain settings to execute an experiment on a remote server. If no configuration file is provided, a default configuration is inferred from the current setting of model space, evaluator, and strategy.

Besides running an experiment, this class is also used to retrieve the experiment summary, including details such as elapsed time, number of trials tested, individual trial results and their respective scores, as well as the top-performing trial, including its score and associated configuration. Moreover, this class is used to export an instance of the top-performing model. Lastly, since NNI does not store model weights when exporting models, this class also includes a method to retrain the top model using the complete

training dataset.

NNI Predictor

This is the primary interface developed to perform NAS. It works effectively with just the training data as a prerequisite, but additional parameters can be specified, such as model spaces and strategy to use, target metric for strategy optimization, maximum predictor duration, the maximum number of epochs, validation set size, batch size, learning rate, experiment name, and the aforementioned configuration file.

The main `fit` method starts by reading the provided data, creating the training and validation datasets, and determining the input shape and output dimension. Subsequently, if given a model space list, these models are validated (i.e., check if the model spaces specified are adequate for the provided data); otherwise, the model space list is inferred. Following this, the next step involves selecting the appropriate evaluator based on the nature of the problem and the specified predictor parameters. The time budget allocated for the experiments' duration is divided by the number of model spaces to be tested.

After this, a NAS Experiment is sequentially initiated for each of the inferred or specified model spaces. The best-performing model from each experiment is saved. Then, these models are compared, and the overall best model is identified and set as a class attribute. This method accepts a refit parameter, which, if set to true, refits the best model on the entire training set.

In addition to this method, the `NNIPredictor` also provides methods to predict and score the target values for a test dataset, utilizing the best model identified during the search.

Chapter 4

Results

4.1 Introduction

This chapter aims to present and discuss the results of applying NAS to the same tasks the host group previously tested for drug response prediction [10, 123, 122]. The host group had previously conducted extensive experiments (Section 2.4), testing numerous configurations of DL models to achieve optimal results. Their efforts were manual, time-consuming, and required a deep understanding of the parameters and hyperparameters involved. In contrast, this study uses NAS to automate this process. With its ability to search through a vast space of possible architectures, NAS has the potential to match or even surpass the performance of the models manually crafted by the host group.

In recent years, ML has been a powerful tool in computer-aided drug design, especially in predicting drug response in cancer, where selecting suitable input features is an important step. The first study carried out by the host group aimed to understand which input representations are the most suitable for predicting drug sensitivity in cancer cell lines by testing several featurization methods and DL algorithms.

Drug resistance is a significant challenge in cancer treatment, and combining multiple drugs can help overcome resistance and improve efficacy. Some of these combinations may even be more effective than expected, exceeding the effects of each individual compound used alone, a phenomenon called drug synergy. The second task carried out by the host group aimed to explore the impact of different methodological variables, such as preprocessing steps, types of input data, and DL architectures, on the performance of DL-based drug synergy prediction models.

Past research has evaluated the impact of different types of input data, including gene expression features, somatic mutations, CNVs, proteomics data, and other drug target-specific features, as well as the impact of using end-to-end DL methods to learn the relevant compound features directly from the compound structures instead of traditional fingerprints and descriptors. Furthermore, studies have evaluated the impact of different preprocessing steps, such as feature selection and reduction, and with different DL

architectures. Although various methodologies have been proposed to address the problem of predicting drug synergy, the impact of different methodological aspects on the performance of these methods is still unclear. The work carried out by the host group allowed the identification of the types of input data that are more predictive of drug response, as well as the feature selection and data representation methods that produce the best results.

This chapter will detail the results of these tasks and compare the performance of the NAS-generated models with the manually-tuned models of the host group, highlighting the efficiency and effectiveness of NAS in this context. The goal is not only to achieve comparable or superior performance but also to demonstrate the potential of NAS as a tool for automating and optimizing the model selection process in ML, thereby saving valuable time and human resources.

4.2 Single input drug sensitivity prediction

4.2.1 Datasets

The datasets used in this study are the same that were used previously in [10, 123] by the host group. These datasets are human cancer cell line drug screenings where a single cell line was isolated. These single-cell line datasets were used to allow a single input type, i.e., to allow for the study of the impact of different compound representations without the need to consider cell line features.

The NCI 1 and NCI 109 represent datasets of chemical compounds screened for activity against non-small cell lung and ovarian cancer cell lines, respectively. In each dataset, a chemical compound is labelled as positive if it is active against the corresponding cancer, or negative otherwise. A balanced version of these datasets was used, which has also been widely used in literature [162, 163].

Two single-cell line cytotoxicity datasets were used to develop regression models. These datasets, PC-3 and CCRF-CEM, initially retrieved from ChEMBL [132], were filtered and cleaned by the authors of a drug sensitivity prediction study [164]. In addition to these preprocessing steps, the original IC_{50} output values were transformed into $-log(IC_{50})$ values.

The remaining larger dataset was derived from the NCI-60 dataset. A subset of this dataset containing the A549/ATCC cell line was selected and curated by removing low-quality experiments, compounds lacking sensitivity data, and compounds for which the corresponding SMILES strings could not be obtained using the files provided by the Developmental Therapeutics Program (DTP). Alike the previously mentioned datasets, the output values in this case are also measured as $-log(IC_{50})$ sensitivity values.

Table 3 summarizes the datasets used in this study. This data can be retrieved from https://

Table 3: Details on the datasets used in this study.

Dataset	Compounds	Output Variable	Output Type
NCI 1	3466	sensitive/not sensitive	binary
NCI 109	3431	sensitive/not sensitive	binary
PC-3	4294	$-log(\text{IC}_{50})$	continuous
CCRF-CEM	3047	$-log(\text{IC}_{50})$	continuous
A549/ATCC	20730	$-log(\text{IC}_{50})$	continuous

github.com/sw33zy/Development-of-a-Software-Application-for-Drug-Sensitivity-Prediction.

4.2.2 Experimental setup

Six different types of molecular fingerprints were evaluated in this study: extended connectivity fingerprint (ECFP) (ECFP4 and ECFP6), Molecular ACCess System (MACCS), atom pair fingerprints (AtomPair), RDKit fingerprints (RDKitFP), RDKit layered fingerprints (LayeredFP). ECFP fingerprints are a popular binary molecular representation based on the Morgan algorithm [165] that encodes a molecule's structural features by considering its circular neighborhood [166]. ECFP4 fingerprints use a radius of 2 to define the circular neighborhood surrounding each atom, while ECFP6 fingerprints use a radius of 3. MACCS is a type of substructure key-based fingerprint typically composed of 166 bits, with each bit corresponding to a specific substructure pattern [167]. AtomPair is a topological fingerprinting method that encodes spatial relationships between pairs of atoms in a molecule [168]. RDKitFP is another topological fingerprint that encodes all subgraphs in the molecule within a set range, hashes each subgraph, and uses these hashes to generate a bit vector of fixed length. The LayeredFP uses the same subgraph algorithm as the RDKitFP, but sets multiple bits in the final fingerprint based on different atom and bond type definitions. Both RDKitFP and LayeredFP are original fingerprinting methods developed by the RDKit [169] project. The SMILES strings contained in each dataset were transformed into each of these representations as bit vectors of size 1024 (except for the MACCS fingerprints, which have a fixed 166-bit size) and used as input data.

Besides fingerprint representations, a one-hot encoding method was also evaluated, which takes a dataset of SMILES strings and converts them into a one-hot encoding of the characters in these strings. In this case, the vocabulary is not composed of the letters in the input strings, but rather the atoms and bonds in the molecules. Lastly, Mol2Vec [170] is an unsupervised approach to learn vector representations of

molecular substructures by applying the Word2Vec algorithm [171]. This method treats Morgan algorithm substructures as "words" and compounds as "sentences". In this study, the molecules in each dataset were transformed into 300-dimensional embeddings using a pre-trained Mol2vec model and used as input.

A caveat: in the work developed by the host group [10], the authors also benchmarked GNN algorithms and representations; however, since these approaches are not yet compatible with the work developed in this thesis, they were not tested.

Each of these representations was used as input for the NNIPredictor. The model space parameter was left empty intentionally to showcase the framework's ability to determine suitable models automatically. Nonetheless, as all of these representations result in one-dimensional vectors (except for One-Hot encoded SMILES), the framework inferred the DNN, 1DCNN, and HybridModel. In the case of One-Hot encoding, which results in two-dimensional vectors for each compound, the HybridModel and RNN model spaces were tested. To draw conclusions regarding the search strategies, each representation was evaluated against all available search strategies (similar to appendix A).

Overall, 160 different experiments were conducted, considering the mentioned five datasets, trying four different search strategies for the NAS and the eight input representations above discussed.

For a fair comparison, most of the remaining model training and evaluation settings were kept consistent with those outlined in [10]. The exact same dataset splits (70% training set and 30% test set) as those employed in the previous study were used, keeping the same instances allocated for both the training and test sets. Similarly, all models were also trained up to 100 epochs with a batch size of 256 samples and using the Adam [20] optimization algorithm. Binary cross-entropy with logits was used as the loss function for classification models, while mean squared error was used for regression models. Unlike the previous study, an early stopping mechanism is implemented, with a patience of 20 epochs (i.e., if a model does not demonstrate significant improvement over a span of 20 epochs, its training is halted).

The resulting experiments were tested with a time limit of 12 hours each and using a single NVIDIA GeForce RTX 2080 Ti GPU.

4.2.3 Results and Discussion

This section aims to analyze and discuss the performance obtained in the different configurations tested. Figure 22 shows classification results with model performance measured using AUC-ROC. Figure 23 reports model performance using RMSE for regression tasks.

Regarding the NCI1 dataset, ECFP6, MACCS, and AtomPair fingerprints achieved similar performance, with a rounded AUC-ROC score of 0.83. The remaining fingerprint representations, ECFP4, RDKitFP and

LayeredFP outperformed all other methods with a rounded AUC-ROC score of 0.85. LayeredFP was the overall top-performing fingerprint representation among them. Interestingly, the same trio of ECFP6, MACCS, and AtomPair fingerprints that performed lower on the NCI1 dataset, also performed comparatively lower results on the NCI109 dataset. The remaining fingerprinting methods showed similar performance once more, with the LayeredFP ranking first again with a AUC-ROC score of 0.84.

On the CCRF-CEM dataset, the AtomPair fingerprint achieved the lowest RMSE (0.67), closely followed by the ECFP6 with a RMSE of 0.68. In this case, the MACCS fingerprint resulted in a moderately worse score when compared to the remaining fingerprinting methods (RMSE = 0.74). The ECFP fingerprints achieved a similar RMSE (both 0.58 when rounded) on the PC-3 dataset. The MACCS fingerprint was the worst-scoring fingerprint with a RMSE of 0.62. On the larger A549 dataset, ECFP4 was the best fingerprint (RMSE = 0.75), followed by the ECFP6 and LayeredFP which both reached a rounded RMSE of 0.76. Similar to the previous datasets, MACCS was the worst-performing fingerprint.

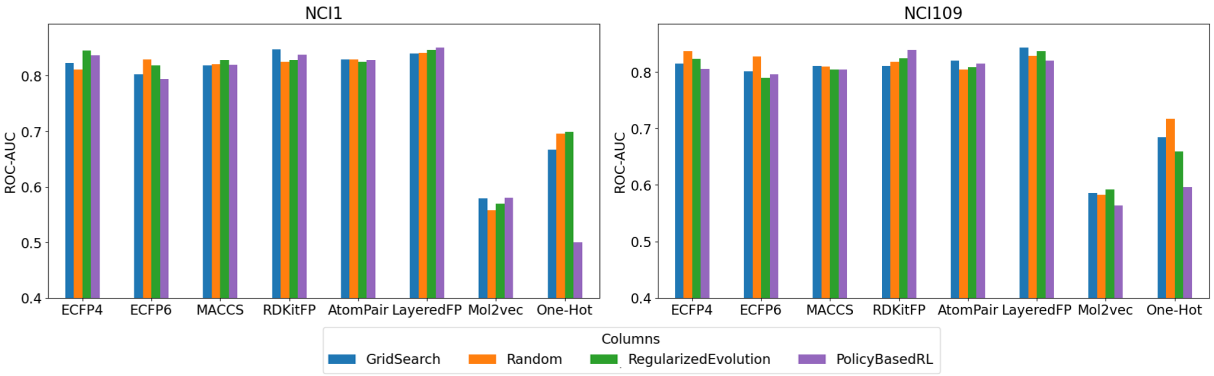


Figure 22: Performance of different input representations and search strategies on the NCI 1 and NCI 109 classification tasks quantified using AUC-ROC. Higher scores mean better performance.

Overall, results were consistently worse when using Mol2Vec embeddings, ranking last place in every task tested. These results are compatible with the ones from previous work by the host group [10, 123]. Mol2Vec is a complex model that learns embeddings in an unsupervised manner by being pre-trained on a dataset with almost 20 million compounds [170]; however, there is a chance that the datasets used differ significantly from the original training data, with many unseen substructures, thus resulting in limited transferability and consequently, suboptimal results. Fine-tuning the pre-trained model on these datasets may allow the embedding model to capture the relevant compound substructures, improving performance.

Using one-hot encoded SMILES led to the second-worst performance across all tasks. In typical character-wise tokenization, all atoms with the same atomic number within the SMILES string are represented equally. However, despite having the same atomic number, the characteristics of each atom can

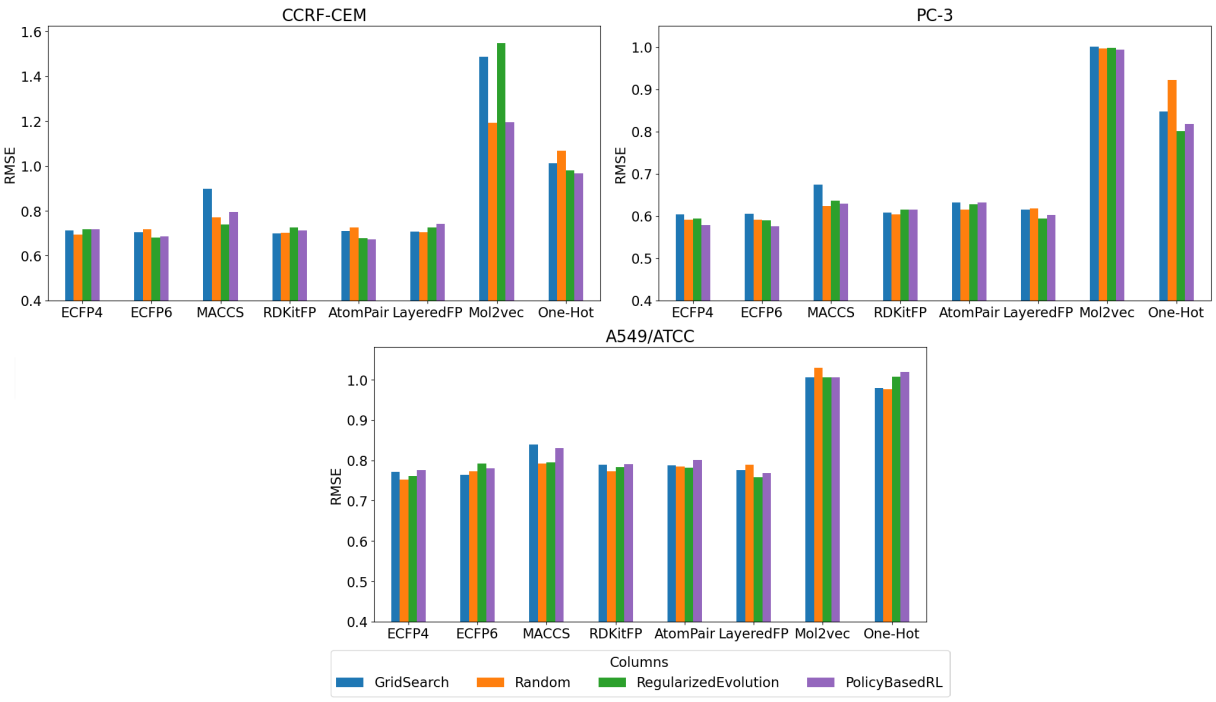


Figure 23: Performance of different input representations and search strategies on the PC-3, CCRF-CEM and NCI-60 A549 regression tasks quantified using RMSE. Lower scores mean better performance.

differ significantly based on their environment. Thus, this method may be too abstract and chemically inaccurate, resulting in less predictive performance. These results are in agreement with a recent study that compared SMILES tokenization schemes [172].

Representing SMILES as graphs where atoms are nodes and bonds are edges and using GNNs to learn these representations, thus preserving spatial relationships between atoms and bonds, may lead to better performance. In fact, using graph representations and GNN algorithms revealed promising results in some previous studies, including the previous work developed in the host group [10, 123, 173].

Table 4: Average fingerprint representations ranks.

Fingerprint	**Global avg rank**	**Classification avg rank**	**Regression avg rank**
ECFP4	2.8	3	2.7
ECFP6	2.6	4.5	1.3
MACCS	6	6	6
RDKitFP	3.2	2	4
AtomPair	4	4.5	3.7
LayeredFP	2.4	1	3.3

The results for the molecular fingerprints were relatively similar to those previously reported [10, 123]. Table 4 displays each fingerprint's global average rank and average rank in classification or regression tasks across all five datasets. Similar to the previous study, LayeredFP also performed consistently well, ranking first overall, but closely followed by the popular ECFP. The results suggest that different fingerprints may be more suitable for different tasks. LayeredFP ranked first in both classification tasks, along with RDKitFP ranking second. These same fingerprints did not perform as well in the regression tasks, with ECFP showing better results. The MACCS fingerprint seems to fall short compared to the remaining fingerprints, ranking last in every task. MACCS fingerprints encode molecules in much shorter vectors (166-bit vectors) than the other fingerprints used (1024-bit vectors). A shorter vector size means this fingerprint encodes the structural information of a molecule in a more simplified way, which may not capture the complexity of molecular structures as effectively as a longer one. The difference in vector size may be why MACCS fingerprints resulted in worse overall performance. Furthermore, these findings are consistent with a recent study [174] where the authors compared their new fingerprint representation to several others, including MACCS. In that study, MACCS fingerprint ranked among the less effective representations.

In general, performance scores were usually similar between many models, especially fingerprints. These fingerprint representations are all designed to capture different aspects of molecular structure, but there may be significant overlap in the information they encode. This is consistent with the results of previous studies [10, 123, 175].

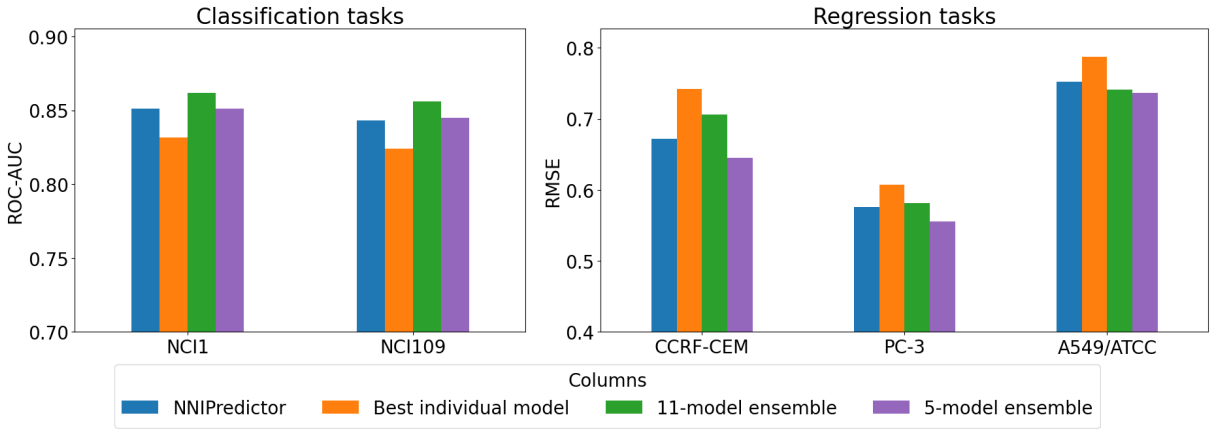


Figure 24: Comparison of the best model found using NNIPredictor with the best model and 5 and 11 model ensembles from the host group's previous study [10, 123]. On the left are classification tasks measured using AUC-ROC (higher is better) and on the right are regression tasks measured using RMSE (lower is better).

Figure 24 compares the results obtained by using NNIPredictor on the same tasks as [10, 123]. The

results are compared with the best individual model discovered in the previous study, as well as ensembles composed of 5 and 11 models, which were also proposed in this study. For the most part, NNIPredictor manages to match or even surpass the performance of the 5-model ensemble for classification tasks and the 11-model ensemble for regression tasks. The best model found by NNIPredictor is, on average, marginally better (2.3% relative to AUC-ROC score) on classification tasks, and slightly better (6.8% relative to RMSE score) on regression tasks, when compared to the best individual model found in [10, 123]. These results may be due to the more complex models used and the limited number of hyperparameter combinations tested in the previous study that were explored using random search. Using NNIPredictor made it possible to search a much wider number of hyperparameter combinations (increasing the chances of finding an optimal configuration) without the need for human expertise to set these combinations.

By analyzing Figures 22 and 23, it does not seem like a superior search strategy exists. The tested search strategies did not exhibit distinct advantages; instead, they consistently produced similar results without any search strategy consistently outperforming the others. Thus, the efficiency of a search strategy seems to depend on the problem's characteristics, with no single algorithm being able to outperform all others for every possible problem.

This finding is in agreement with many previous studies. A study by Real et al [101] compared RL and evolutionary algorithms along with a random search approach in the context of NAS. Their results suggest that RL and evolutionary algorithms have comparable prediction performance. While both methods outperformed random search, the margin of improvement was relatively small. Additionally, the results of a different study, [176], showed that random search with an early stopping mechanism performs as well as ENAS [147], which is an RL-based leading NAS method. Furthermore, Yu et al [177] demonstrated that the state-of-the-art NAS search algorithms are not significantly better than random search. Finally, a recent AutoML benchmark and survey [178] where several popular search strategy algorithms were tested revealed that, except for grid search, all algorithms perform on average very similarly, with almost 70% of all configurations not being significantly worse than the best result (less than 1.9% mean absolute accuracy difference).

As a result, it is not reasonable to only take into account performance measures when ranking search strategies; other aspects like scalability or method overhead should also be considered. Tables 5 and 6 provide additional details for each search strategy. The average score column is relevant since it depicts how well a search strategy converges toward an optimal architecture. The average number of trials is significant since more configurations tested may increase the likelihood of discovering an optimal architecture. Grid search consistently tests more configurations, which was to be expected since the overhead

Table 5: Additional search strategy details for classification tasks.

Search strategy	NCI			NCI109		
	Avg AUC-ROC	Avg nr trials	Winner	Avg AUC-ROC	Avg nr trials	Winner
GridSearch	0.814	2424	1	0.800	2476	3
Random	0.805	1653	2	0.792	1646	3
RegularizedEvolution	0.819	1680	3	0.803	1643	1
PolicyBasedRL	0.810	907	2	0.794	868	1

The AUC-ROC column represents the average score across every configuration tested for each input representation. The number of trials is averaged across the trials tested for each input representation. The winner column represents the number of times the corresponding search strategy found the best-performing model.

Table 6: Additional search strategy details for regression tasks.

Search strategy	CCRF-CEM			PC-3			A549		
	Avg RMSE	Avg nr trials	Winner	Avg RMSE	Avg nr trials	Winner	Avg RMSE	Avg nr trials	Winner
GridSearch	1.863	1115	1	2.167	1052	0	1.108	289	3
Random	2.144	744	3	1.925	617	3	1.756	182	3
RegularizedEvolution	1.575	875	2	1.446	665	2	1.699	190	2
PolicyBasedRL	1.793	619	2	1.637	442	3	1.498	162	0

The RMSE column represents the average score across every configuration tested for each input representation. The number of trials is averaged across the trials tested for each input representation. The winner column represents the number of times the corresponding search strategy found the best-performing model.

of this algorithm is very small. On the other hand, the PolicyBasedRL consistently tests a smaller amount of configurations, given this algorithm is much more complex and involves a higher computational burden. Random search and RegularizedEvolution seem to be a middle ground regarding the number of trials tested. The RegularizedEvolution algorithm consistently outperformed the remaining in terms of average scoring metric. For these reasons, the evolutionary algorithm will be used as the search strategy for the following chapter's experiments.

Some final remarks regarding model configurations were investigated (details can be found here[1]). It is worth noting that none of the search strategies converged to the same architecture, even when using the same input representation and dataset (i.e., only changing search strategy). This was expected, given the magnitude of possible configurations. Moreover, most of the 160 experiments showed the best results when containing 1D-CNN layers, suggesting that these datasets may benefit from this type of architecture.

[1] https://github.com/sw33zy/Development-of-a-Software-Application-for-Drug-Sensitivity-Prediction

This could be because 1D-CNNs can capture sequential dependencies and structural patterns, which are particularly relevant in molecular datasets. Furthermore, it was observed that most experiments converged towards deeper models (averaging 9/10 layers), compared to the previous study [10, 123], which averaged 5 to 6 layers. Despite the increased depth, the models did not appear to be overfitting, suggesting that deeper models may be better suited for drug response prediction.

4.3 Predicting drug synergy in cancer cell lines

4.3.1 Datasets

The data used in this study was preprocessed and used in [10, 122] by the host group. The authors used the ALMANAC [4] dataset downloaded from CellMinerCBD [179] to predict drug combination effects in the form of ComboScores. The ComboScore is the sum of differences between observed and expected cell line growth for all dose combinations for a cell line and drug combination. This dataset was initially filtered by averaging the ComboScore of experiments involving the same cell line and drug combination and removing the MDA-MB-468 and MDA-N cell lines for not being present across the multiple datasets used.

The compound identifiers present in the original dataset were used to map compounds to their respective SMILES strings using the files provided by the Developmental Therapeutics Program (DTP) for the ALMANAC dataset. For unmapped compounds using the previous method, PubChemPy [141] was used to retrieve canonical SMILES based on drug names. The obtained SMILES strings were then preprocessed using the ChEMBL Structure Pipeline [180]. The preprocessed SMILES were used to compute different compound representations, detailed in section 4.3.2.

RNA-Seq gene expression data: RNA-Seq gene expression data for the cell lines in the form of log-transformed Fragments Per Kilobase of transcript per Million mapped reads (FPKM) values was also downloaded from CellMinerCDB [179]. This data was curated by removing genes with constant expression values across all cell lines. The resulting dataset was filtered using specific gene sets, which will be detailed in the next section. Besides these specific gene sets, Uniform Manifold Approximation and Projection (UMAP) [181] was tested as another dimensionality reduction method. This method has four main parameters that can significantly impact the resulting embedding. The number of neighbors was set to 20 when learning the manifold structure of the data, which resulted in embeddings with 50 components. Pearson correlation was used as the distance metric, and the minimum embedding distance was

maintained at 0.1.

Mutation data: Mutation data from whole-exome sequencing for the cell lines was derived from a mutation annotation format (MAF) via CBioPortal [179]. Silent mutations and mutations in noncoding regions were excluded. The data was then binarized and summarized at the gene level, indicating gene alterations with '1' and no mutations with '0'. This data was further filtered using drug-gene interactions from The Drug-gene Interaction Database (DGIdb) to create a gene-level mutation dataset with 636 genes. A new binary event matrix was created to generate a pathway-level dataset, where '1' indicates at least one alteration in a given pathway in a cell line. Genes were mapped to pathways using gene sets from the Molecular Signatures Database (MSigDB) [182, 183] derived from the Reactome [184] database, which have been filtered to reduce the redundancy between gene sets, resulting in 1510 features.

CNV data: A dataset containing Genomic Identification of Significant Targets in Cancer (GISTIC) putative CNVs for the cell lines was downloaded from CBioPortal [179]. This dataset was filtered to keep the 952 genes in the drug-gene interactions list obtained from DGIdb.

The omics datasets were merged with the drug response dataset individually, aligning cell line data with each experiment. The dataset examples were split between training, validation, and test sets using a leave-drug-combinations-out approach, ensuring that drug combinations appearing in one set were excluded from the others. After splitting the expression dataset, features were scaled to a range between zero and one (min-max scaling). Scripts to generate the data and splits for this study case were provided by the authors and are available at https://github.com/BioSystemsUM/drug_response_pipeline/blob/master/almanac/README.md

4.3.2 Experimental setup

This study tested several combinations of input data types, feature engineering methods, and input representations to predict drug combination effects summarized as ComboScores, using the ALMANAC dataset [4]. These tests can be categorized into two distinct types:

1. *expr + drugA + drugB* - combining RNAseq gene expression data (expr) along with drug features for both drugs (drugA, drugB);

2. *expr + mut + cnv + drugA + drugB* - combining RNAseq gene expression data (expr), mutation data (mut), copy number variation features (cnv) and drug features (drugA, drugB).

Type 1 input combinations were tested first to evaluate the impact of the gene expression data. Several feature selection methods were tested using more comprehensive or more selective lists of genes. A compact representation using a dimensionality reduction method was also evaluated. The following methods were evaluated in this study:

- $expr_{protein\ coding}$: list of strictly protein-coding genes derived from filtering the original dataset, resulting in a total of 18779 genes;
- $expr_{landmark}$: the original dataset was filtered to keep the 978 "landmark" genes defined by the L1000 project [185], a high-throughput gene expression assay. These genes are considered to be representative of all RNA transcripts;
- $expr_{DGI}$: The Drug-gene Interaction Database (DGIdb) was used to obtain a set of 994 genes that interact with the drugs screened in the ALMANAC project. Among these, 976 genes were present in the gene expression dataset;
- $expr_{COSMIC}$: a filtered dataset version containing 713 of the 723 cancer-related genes obtained from the COSMIC [128] Cancer Gene Census;
- $expr_{NCG}$: an alternative cancer-specific gene list comprising 2372 genes was obtained from the Network of Cancer Genes (NCG) [186]. After filtering, the expression dataset includes 2362 genes identified in the COSMIC Cancer Gene Census and other genes implicated in cancer;
- Combined gene lists: $expr_{DGI\ +\ landmark}$ with 1815 genes/features and $expr_{DGI\ +\ NCG}$ containing 3037 genes/features;
- $expr_{UMAP}$: Uniform Manifold Approximation and Projection (UMAP) [181] was used to identify critical structures in the high-dimensional gene expression data and preserve them in a lower-dimensional embedding (50 components).

Subsequently, the gene expression data was kept constant and the drug data was modified to evaluate the influence of different drug representations:

- $drugs_{ECFP4}$: 1024-bit ECFP fingerprints [166] with a radius of 2;
- $drugs_{LayeredFP}$: 1024-bit layered fingerprints developed by the RDKit [169] project;

- $drugs_{MTE}$: 512-dimensional Molecular Transformer Embeddings (MTEs) [187] that contain abstract features describing general molecular structure as a result of training a transformer neural network to translate between two distinct text-based molecular representations.

Finally, type 2 input combinations were explored to determine whether adding more cell line data, such as mutation and CNVs, would improve predictive performance. Two combinations were tested: one containing mutation data summarized at the gene level and CNVs, with both datasets filtered to exclusively include drug-gene interaction (DGI) genes ($expr_{DGI}$ + $mut_{DGI,\ gene\text{-}level}$ + cnv_{DGI} + $drugs_{ECFP4}$), and another containing mutations summarized at the pathway level and CNVs ($expr_{DGI}$ + $mut_{pathway\text{-}level}$ + cnv_{DGI} + $drugs_{ECFP4}$). In total, 12 different combinations were tested, which are summarized in Table 7.

Table 7: Different types of methodological variables tested in this study.

Different gene expression subnetworks	**Different drug encoding schemes**	**Extra omics data**
• $expr_{protein\ coding}$ + $drugs_{ECFP4}$	• $expr_{DGI}$ + $drugs_{LayeredFP}$	• $expr_{DGI}$ + $mut_{DGI,\ gene\text{-}level}$ + cnv_{DGI} + $drugs_{ECFP4}$
• $expr_{landmark}$ + $drugs_{ECFP4}$	• $expr_{DGI}$ + $drugs_{MTE}$	• $expr_{DGI}$ + $mut_{pathway\text{-}level}$ + cnv_{DGI} + $drugs_{ECFP4}$
• $expr_{DGI}$ + $drugs_{ECFP4}$		
• $expr_{COSMIC}$ + $drugs_{ECFP4}$		
• $expr_{NCG}$ + $drugs_{ECFP4}$		
• $expr_{DGI\ +\ landmark}$ + $drugs_{ECFP4}$		
• $expr_{DGI\ +\ NCG}$ + $drugs_{ECFP4}$		
• $expr_{UMAP}$ + $drugs_{ECFP4}$		

Each of these combinations was used as input for the NNIPredictor using the multimodal model space. The evolutionary algorithm was chosen as the search strategy due to the conclusions drawn from the previous section. The experiments were tested with a time limit of 7 days each and using a single NVIDIA GeForce RTX 2080 Ti GPU.

Most model training and evaluation settings were consistent with those outlined in [10, 122] to ensure a fair comparison. The exact same leave-drug-combinations-out dataset splits containing 239,943 examples (80%) for the training set, and 30,007 and 30,001 examples, respectively, for the test and validation sets (10% each) as those employed in the previous study were used. Similarly, all models were also trained up to 500 epochs with a batch size of 64 samples, using the Adam [20] optimization algorithm and MSE as

the loss function. An early stopping callback with a patience of 20 epochs was also implemented to stop the learning process once the loss stops improving.

4.3.3 Results and Discussion

This section analyzes and discusses the performance of the twelve different methodological approaches to predict drug combination effects using the ALMANAC [4] dataset. In addition, the results obtained by using NNIPredictor will be compared to the results obtained by the previous study [10, 122].

The best-performing log-transformed min-max scaled gene expression data were the combination of genes with known or potential interactions with drugs screened in the ALMANAC project with genes considered to be representative of the transcriptome as a whole ($expr_{DGI + landmark}$) (Figure 25). Following this combination were the experiments using COSMIC cancer-related genes and UMAP embeddings. COSMIC genes seem to be representative of the genes present in the cell lines from the ALMANAC dataset since this feature selection method resulted in better performances in both this study and [10, 122]. However, the other cancer-specific gene list ($expr_{NCG}$) resulted in the worst-performing gene list. A possible reason for this may be the increased number of features.

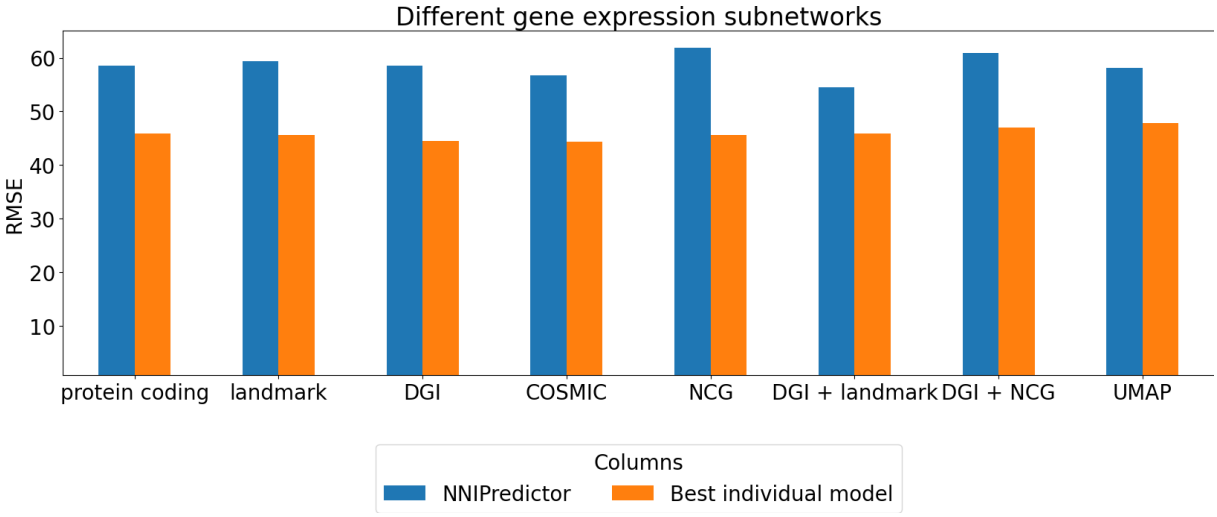


Figure 25: Performance scores of models with different gene expression representations using NNIPredictor vs the best model found for the same case in the host group's previous study [10, 122] (lower is better). The drug features were kept consistent using a ECFP4 molecular representation for each drug.

In general, it appears feature selection methods that resulted in a smaller number of features produced models with better performance scores. Notably, the smallest gene list ($expr_{COSMIC}$) and the dimensionality reduction method ($expr_{UMAP}$) ranked in the second and third positions in terms of performance, respectively. Nonetheless, since $expr_{DGI + landmark}$ ranked first, using prior biological knowledge does seem to be

beneficial and other gene lists and their combination might result in better performance. These findings are in agreement with the host group's previous study. Different gene expression representations, such as representing cell lines as gene-interaction graphs or using methods to transform tabular omics data into images (REpresentation of Features as Images with NEighborhood Dependencies (REFINED) [188] or Image Generator for Tabular Data (IGTD) [189]) may also impact the performance and should be tested in the future.

In the following experiments, only the $expr_{DGI}$ representation for gene expression data will be considered. Although this representation was not the top performing in this study, this decision was upheld in order to be able to compare the results from both studies.

Contrary to the previous study results, using LayeredFP did result in better performance than ECFP4 fingerprints (Figure 26). This is consistent with the results of the previous case study where LayeredFP achieved the lowest average rank. Furthermore, despite not having resulted in better performance, this fingerprinting method achieved the second-best drug encoding scheme score in the previous study, with the performance difference between ECFP4 and LayeredFP methods not being substantial. Thus, these results further consolidate LayeredFP as a reasonable choice for many drug response prediction datasets. Using Molecular Transformer Embedding (MTE) resulted in worse performance, which seems to be in agreement with the lower predictive ability of embedding methods for drug response prediction also noticed in the last case study.

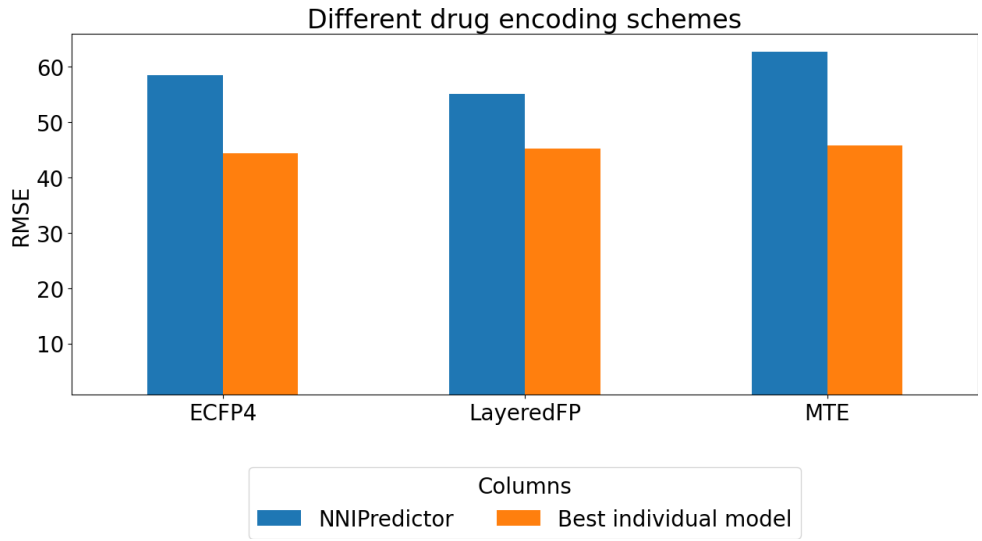


Figure 26: Performance scores of models with different drug representations using NNIPredictor vs the best model found for the same case in the host group's previous study [10, 122] (lower is better). The gene expression features were kept consistent using the $expr_{DGI}$ representation.

Including mutation and CNV data resulted in comparable or better performance (Figure 27). The per-

formance improved when using mutation features at the pathway level instead of gene-level features. This may be because more genes are considered when summarizing the mutation data at the pathway level. Using drug targets or drug-gene interactions (DGIs) as a gene selection method may limit the generalizability of a model since new compounds in the test set may have different DGIs than the relevant ones used for training. Thus, relevant genetic information may be lost when only DGI genes are considered. Moreover, the mutation dataset summarized at the pathway-level is less sparse than the gene-level dataset.

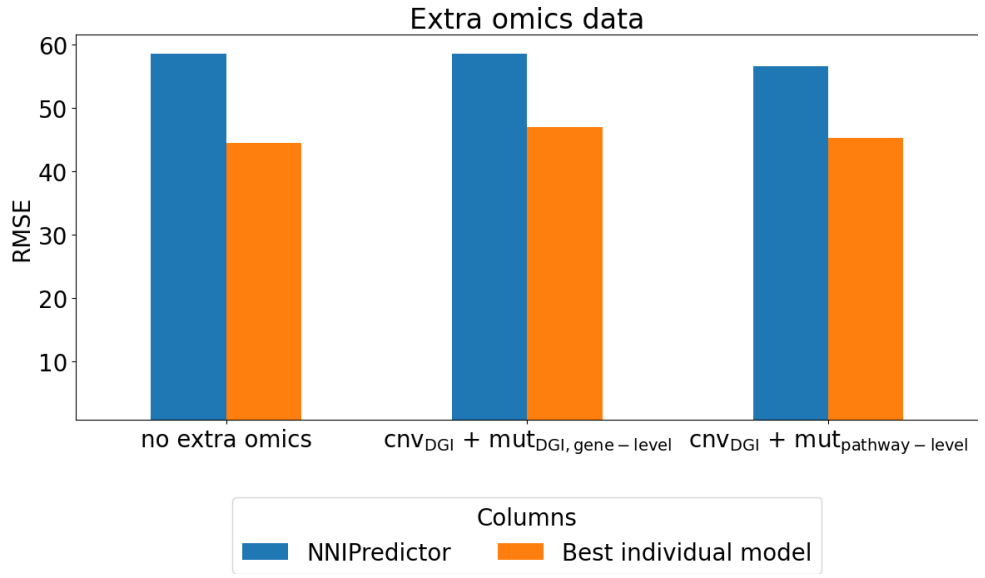


Figure 27: Performance scores of models trained with mutation (mut) and copy number variation (cnv) data in addition to gene expression ($expr_{DGI}$) and drug features ($drugs_{ECFP4}$) using NNIPredictor vs the best model found for the same case in the host group's previous study [10, 122] (lower is better).

Using NNIPredictor consistently resulted in suboptimal results when compared to the top models found in the previous study. This is due to multiple reasons. Most importantly, in the previous study, a total of 50 configurations was evaluated for each model, whereas using NNIPredictor averaged about 30 configurations for each experiment (these details can be found here[2]). Since each subnetwork in the multi-input model uses the model spaces implemented into NNIPredictor as building blocks and each subnetwork encompasses independent decisions, the resulting search space grows exponentially. Therefore, having only evaluated 30 configurations out of possible millions, the search strategy might not have had sufficient time to explore and reach a configuration that leads to better predictions. In the future, it would be beneficial to implement a simplified multimodal model without such a complex search space or conduct longer experiments.

Additionally, it was observed that a considerable number of experiments appeared to suffer from un-

[2] https://github.com/sw33zy/Development-of-a-Software-Application-for-Drug-Sensitivity-Prediction

derfitting. This was evident from certain models having achieved better performance on the test set than on the training set. It seems that this issue was caused by the EarlyStopping mechanism implemented. While beneficial in preventing overfitting and conserving computational resources, this mechanism appeared to be terminating promising configurations prematurely. This led to the generation of underfitted models, as the training process was halted before the model could fully converge and generalize. This issue is also noticeable from the detailed results, where experiments that evaluated a larger amount of configurations also showed a larger average MSE across all trials tested. Another reason that substantiates this observation was that model configurations trained for a longer time (and a larger amount of epochs) appeared to result in better performance. A patience higher than 20 epochs should be tested in the future.

Chapter 5

Conclusions and future work

5.1 Conclusions

The main goal of this thesis was to develop a generic framework for model selection and optimization that can effectively handle multiple types of input data. To achieve this, the framework uses NAS to automatically search and evaluate a wide range of DL models, considering different architectural configurations and hyperparameter settings. By exploring a search space of possible architectures, it is possible to identify which architecture(s) can effectively handle the characteristics of the given data.

NNIPredictor, the primary contribution of this work, is a highly versatile and easy-to-use module that facilitates the typical DL workflow. The developed module adapts itself to the input data by inferring the problem type (regression, binary classification, or multi-class classification), the output size, and (optionally) the compatible models, to provide an optimized DL architecture that is best suited for the input data. However, it is also flexible enough to customize parameters such as the search strategy, time limit, number of epochs, batch size, etc. This module includes models suitable for most data types, such as DNN and 1D-CNN models for tabular data; 1D-CNN and an RNN model which can incorporate RNN, GRU, and LSTM layers for sequential data; and a CNN model for the remaining grid-like data. Additionally, a hybrid model is available that adapts to the dimensionality of the input data, combining different layer types. The architecture of these models might vary in depth, layer types, connections, and other model parameters depending on the configurations chosen by a search strategy that explores the possible model configurations. For multi-input datasets, NNIPredictor supports a multimodal approach, creating dedicated subnetworks for each data type or subset. The model selects compatible single-input models for processing each input.

To evaluate the potential of NAS and the performance of the developed framework, the module was validated using various multi-omics datasets to predict drug sensitivity and synergy. From these case studies, it was possible to draw the following general conclusions:

- It was confirmed the potential of NAS as a tool for automating and optimizing the model selection process in ML, thereby saving valuable time and human resources. The results obtained were comparable to the previous studies carried out by the host group.

- Search strategy algorithms perform mostly similarly; other aspects, such as method overhead and the average score across all tested configurations, should be considered. Results suggest evolutionary algorithms are a reasonable choice.

- Most molecular fingerprint representations yield comparable results. This may be because of significant overlap in the information they encode. Other methods like one-hot encoding SMILES and Mol2Vec embeddings seem to have lower predictive efficacy for drug response prediction tasks.

- Using prior biological knowledge to extract features and reduce the size of the high-dimensional omics datasets does seem to positively impact the ability of DL models to learn relevant features.

This framework not only saves time but also makes DL more accessible to a broader range of professionals, such as in the medical field. NNIPredictor eliminates the need for the time-consuming hyperparameter tuning step, which often requires a deep understanding of the underlying algorithms. This automation, coupled with the ability to adapt to various data types, facilitates the benefits of DL without the steep learning curve.

Despite the increased accessibility that NNIPredictor and NAS offer, there is still much room for improvement. The NAS-generated models obtained from the multi-input data synergy case study were still subpar when compared to manually-tuned models. These results, however, are likely to be improved using an optimized multimodal model with a simpler search space or allowing more computational time for each experiment. In addition, it is important to note that NNIPredictor does not escape the constraints of the No Free Lunch Theorem [190]. It is inevitable that this framework will not perform optimally across all possible problems. Furthermore, in the context where it is inserted in the Omnia platform, this module is placed after a feature engineering process. As the results showed, the features derived from raw data have an influence on the model's performance, therefore, this tool does not eliminate the need for feature engineering and thoughtful input representation.

Another limitation of the developed work is the lack of interpretability. The "black-box" nature of DL makes it challenging to explain model decisions. This lack of interpretability is a critical factor, especially in fields like healthcare, where understanding the reasons behind a model's predictions is crucial for trust and acceptance. Finally, this framework does not guarantee the reproducibility of results. Random seeds

for number generation were not set because search strategy algorithms also make different decisions in each iteration, further contributing to the non-reproducibility of outcomes.

5.2 Future work

The work developed in this thesis can be used as a starting point to achieve an even more complete framework that yields better results. This section will explore some recommendations and suggestions for extending and improving the work undertaken in this thesis.

One way to improve the predictive performance and robustness of the models generated is to implement ensembles. A voting ensemble that averages the predictions of the top-performing models would be a simple solution for this limitation. More elaborate ensemble methods could be used, such as stacking ensembles that consist of stacking the output of individual models and using another model to compute the final prediction [191].

Implementing additional models would be a straightforward way to extend the functionalities of the developed framework. In particular, transformers have emerged as a powerful architecture resulting in remarkable performance in various tasks. This type of architecture has been used recently in cancer drug response prediction with great success [192, 193]. Different datasets might be better suited for specific architectures; therefore, implementing various new models could improve performance in certain tasks. In the same sense, support for graph datasets along with GNN architectures would be a logical next step for this work.

To address the limitations of this framework, methods like SHapley Additive exPlanations (SHAP) [194], Local Interpretable Model-agnostic Explanations (LIME) [195] or Anchors [196] could be used to gain insights into how the models arrive at their predictions. Result reproducibility could be ensured using PyTorch's guidelines[1] to limit the number of sources of nondeterministic behavior and NNI's latest functionalities which include a seed parameter for search strategies.

Since biological or chemical datasets tend to be small, employing complex neural networks may not always be practical, especially when conventional machine learning methods can yield competitive results. Combining the benefits of transfer learning and NAS by training a complex neural network on a larger dataset and then applying NAS to optimize and retrain the network's final prediction subnetwork using a smaller dataset could be an intriguing concept to explore in a future work.

[1] https://pytorch.org/docs/stable/notes/randomness.html

Bibliography

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[2] Dearbhaile C. Collins, Raghav Sundar, Joline S.J. Lim, and Timothy A. Yap. Towards precision medicine in the clinic: From biomarker discovery to novel therapeutics. *Trends in pharmacological sciences*, 38:25–40, 1 2017. ISSN 1873-3735. doi: 10.1016/J.TIPS.2016.10.012. URL https://pubmed.ncbi.nlm.nih.gov/27871777/.

[3] Jordi Barretina, Giordano Caponigro, Nicolas Stransky, Kavitha Venkatesan, Adam A. Margolin, Sungjoon Kim, Christopher J. Wilson, Joseph Lehár, Gregory V. Kryukov, Dmitriy Sonkin, Anupama Reddy, Manway Liu, Lauren Murray, Michael F. Berger, John E. Monahan, Paula Morais, Jodi Meltzer, Adam Korejwa, Judit Jané-Valbuena, Felipa A. Mapa, Joseph Thibault, Eva Bric-Furlong, Pichai Raman, Aaron Shipway, Ingo H. Engels, Jill Cheng, Guoying K. Yu, Jianjun Yu, Peter Aspesi, Melanie De Silva, Kalpana Jagtap, Michael D. Jones, Li Wang, Charles Hatton, Emanuele Palescandolo, Supriya Gupta, Scott Mahan, Carrie Sougnez, Robert C. Onofrio, Ted Liefeld, Laura MacConaill, Wendy Winckler, Michael Reich, Nanxin Li, Jill P. Mesirov, Stacey B. Gabriel, Gad Getz, Kristin Ardlie, Vivien Chan, Vic E. Myer, Barbara L. Weber, Jeff Porter, Markus Warmuth, Peter Finan, Jennifer L. Harris, Matthew Meyerson, Todd R. Golub, Michael P. Morrissey, William R. Sellers, Robert Schlegel, and Levi A. Garraway. The cancer cell line encyclopedia enables predictive modelling of anticancer drug sensitivity. *Nature*, 483:603–607, 3 2012. ISSN 1476-4687. doi: 10.1038/NATURE11003. URL https://pubmed.ncbi.nlm.nih.gov/22460905/.

[4] Camalier R. Crowell J. A. Govindharajulu J. P. Hollingshead M. Anderson L. W. Polley E. Rubinstein L. Srivastava A. Wilsker D. Collins J. M. Doroshow J. H. Holbeck, S. L. The national cancer institute almanac: A comprehensive screening resource for the detection of anticancer drug pairs with enhanced therapeutic activity. *Cancer research*, 77:3564–3576, 7 2017. ISSN 1538-7445. doi: 10.1158/0008-5472.CAN-17-0489. URL https://pubmed.ncbi.nlm.nih.gov/28446463/.

[5] Wang D. Mason M. J. Szalai B. Bulusu K. C. Guan Y. Yu T. Kang J. Jeon M. Wolfinger R. Nguyen

T. Zaslavskiy M. AstraZeneca-Sanger Drug Combination DREAM Consortium Jang I. S. Ghazoui Z. Ahsen M. E. Vogel R. Neto E. C. Norman T. Tang E. K. Y. ... Saez-Rodriguez J. Menden, M. P. Community assessment to advance computational prediction of cancer drug combinations in a pharmacogenomic screen. *Nature communications*, 10, 12 2019. ISSN 2041-1723. doi: 10.1038/S41467-019-09799-2. URL https://pubmed.ncbi.nlm.nih.gov/31209238/.

[6] Michael P. Menden, Francesco Iorio, Mathew Garnett, Ultan McDermott, Cyril H. Benes, Pedro J. Ballester, and Julio Saez-Rodriguez. Machine learning prediction of cancer cell sensitivity to drugs based on genomic and chemical properties. *PloS one*, 8, 4 2013. ISSN 1932-6203. doi: 10.1371/JOURNAL.PONE.0061318. URL https://pubmed.ncbi.nlm.nih.gov/23646105/.

[7] Hossein Sharifi-Noghabi, Soheil Jahangiri-Tazehkand, Petr Smirnov, Casey Hon, Anthony Mammoliti, Sisira Kadambat Nair, Arvind Singh Mer, Martin Ester, and Benjamin Haibe-Kains. Drug sensitivity prediction from cell line-based pharmacogenomics data: guidelines for developing machine learning models. *Briefings in bioinformatics*, 22, 11 2021. ISSN 1477-4054. doi: 10.1093/BIB/BBAB294. URL https://pubmed.ncbi.nlm.nih.gov/34382071/.

[8] Sandra; Longley Daniel B.; Johnston Patrick G. Holohan, Caitriona; Van Schaeybroeck. Cancer drug resistance: an evolving paradigm. *Nature Reviews Cancer 2013-sep 24 vol. 13 iss. 10*, 13, sep 2013. doi: 10.1038/nrc3599.

[9] Ronald J. Tallarida. Quantitative methods for assessing drug synergism. *Genes Cancer*, 2:1003, 11 2011. ISSN 19476019. doi: 10.1177/1947601912440575. URL /pmc/articles/PMC3379564//pmc/articles/PMC3379564/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC3379564/.

[10] Delora Soeiro Baptista. A framework for predicting drug sensitivity and synergy in cancer cells using deep learning. 2022.

[11] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997. ISBN 9780071154673. URL https://books.google.pt/books?id=EoYBngEACAAJ.

[12] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN 026201825X.

[13] Yuji Roh, Geon Heo, and Steven Euijong Whang. A survey on data collection for machine learning:

A big data - ai integration perspective. *IEEE Transactions on Knowledge and Data Engineering*, 33 (4):1328–1347, 2021. doi: 10.1109/TKDE.2019.2946162.

[14] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[15] Leo Breiman. Classification and regression trees. *Wadsworth Brooks/Cole Advanced Books Software*, 1984.

[16] Thomas G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*, pages 1–15, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-45014-6.

[17] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[18] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[19] Guolin Ke, Qi Mak, and Qi Wang. Lightgbm: A highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 3149–3157, 2017.

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

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30:6000–6010, 2017.

[22] Yann Lecun and Y. Bengio. Convolutional networks for images, speech, and time-series. 01 1995.

[23] P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. doi: 10.1109/5.58337.

[24] Sepp Hochreiter and J"urgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[25] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

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

[27] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Li'o, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2018.

[28] Jacob Devlin, Ming Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 1:4171–4186, 10 2018. doi: 10.48550/arxiv.1810.04805. URL https://arxiv.org/abs/1810.04805v2.

[29] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. URL https://github.com/codelucas/newspaper.

[30] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. 7 2019. doi: 10.48550/arxiv.1907.11692. URL https://arxiv.org/abs/1907.11692v1.

[31] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, pages 267–288, 1996.

[32] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[33] Nitish Srivastava, Geoffrey E 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.

[34] H. C. Shin et al. Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE Transactions on Medical Imaging*, 35(5): 1285–1298, 2016. doi: 10.1109/TMI.2016.2528162.

[35] K. Schouten and F. Frasincar. Survey on aspect-level sentiment analysis. *IEEE Transactions on Knowledge and Data Engineering*, 28(3):813–830, 2016. doi: 10.1109/TKDE.2015.2485209.

[36] Jayashree Padmanabhan Melvin Jose Johnson Premkumar. Machine learning in automatic speech recognition: A survey. *IETE Technical Review*, 32(4):240–251, 2015. doi: 10.1080/02564602.2015.1010611.

[37] Konstantina Kourou, Themis P. Exarchos, Konstantinos P. Exarchos, Michalis V. Karamouzis, and Dimitrios I. Fotiadis. Machine learning applications in cancer prognosis and prediction. *Computational and Structural Biotechnology Journal*, 13:8–17, 2015. doi: https://doi.org/10.1016/j.csbj.2014.11.005.

[38] Avrim L. Blum and Ronald L. Rivest. Training a 3-node neural network is np-complete. *Neural Networks*, 5(1):117–127, 1992. doi: https://doi.org/10.1016/S0893-6080(05)80010-3.

[39] B. Korte B. Korte, J. Vygen and J. Vygen. *Combinatorial optimization*. Springer, 2012. ISBN 978-3-642-24487-2.

[40] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. doi: 10.1080/14786440109462720.

[41] Stan Z. Li and Anil Jain, editors. *LDA (Linear Discriminant Analysis)*, pages 899–899. Springer US, Boston, MA, 2009. ISBN 978-0-387-73003-5. doi: 10.1007/978-0-387-73003-5_349. URL https://doi.org/10.1007/978-0-387-73003-5_349.

[42] Qinxue Meng, Daniel Catchpoole, David Skillicom, and Paul J. Kennedy. Relational autoencoder for feature extraction. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 364–371, 2017. doi: 10.1109/IJCNN.2017.7965877.

[43] Ozan İrsoy and Ethem Alpaydın. Unsupervised feature extraction with autoencoder trees. *Neurocomputing*, 258:63–73, 2017. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2017.02.075. URL https://www.sciencedirect.com/science/article/pii/S0925231217304186. Special Issue on Machine Learning.

[44] Jianbin Ma and Xiaoying Gao. A filter-based feature construction and feature selection approach for classification using genetic programming. *Knowledge-Based Systems*, 196:105806, 2020. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2020.105806. URL https://www.sciencedirect.com/science/article/pii/S095070512030191X.

[45] Parikshit Sondhi. Feature construction methods : A survey. 2009.

[46] Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Parthasrathy. Cognito: Automated feature engineering for supervised learning. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 1304–1307, 2016. doi: 10.1109/ICDMW.2016.0190.

[47] Udayan Khurana, Horst Samulowitz, and Deepak Turaga. Feature engineering for predictive modeling using reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. doi: 10.1609/aaai.v32i1.11678. URL https://ojs.aaai.org/index.php/AAAI/article/view/11678.

[48] Fatemeh Nargesian, Horst Samulowitz, Udayan Khurana, Elias Khalil, and Deepak Turaga. Learning feature engineering for classification. pages 2529–2535, 08 2017. doi: 10.24963/ijcai.2017/352.

[49] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 979–984, 2016. doi: 10.1109/ICDM.2016.0123.

[50] Ambika Kaul, Saket Maheshwary, and Vikram Pudi. Autolearn — automated feature generation and selection. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 217–226, 2017. doi: 10.1109/ICDM.2017.31.

[51] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10, 2015. doi: 10.1109/DSAA.2015.7344858.

[52] Binh Tran, Bing Xue, and Mengjie Zhang. Genetic programming for multiple-feature construction on high-dimensional classification. *Pattern Recognition*, 93:404–417, 2019. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2019.05.006. URL https://www.sciencedirect.com/science/article/pii/S0031320319301815.

[53] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 847–855, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450321747. doi: 10.1145/2487575.2487629. URL https://doi.org/10.1145/2487575.2487629.

[54] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305, feb 2012. ISSN 1532-4435.

[55] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, page 2951–2959. Curran Associates Inc., 2012.

[56] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05, page 507–523, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 9783642255656. doi: 10.1007/978-3-642-25566-3_40. URL https://doi.org/10.1007/978-3-642-25566-3_40.

[57] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 2546–2554. Curran Associates Inc., 2011. ISBN 9781618395993.

[58] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1437–1446. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/falkner18a.html.

[59] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18(1):6765–6816, jan 2017. ISSN 1532-4435.

[60] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. 05 2016.

[61] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. 11 2017.

[62] Zhi-Hua Zhou. *Ensemble Learning*, pages 411–416. Springer US, Boston, MA, 2015. ISBN 978-1-4899-7488-4. doi: 10.1007/978-1-4899-7488-4_293. URL https://doi.org/10.1007/978-1-4899-7488-4_293.

[63] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. *Automatic Frankensteining: Creating Complex Ensembles Autonomously*, pages 741–749. 06 2017. ISBN 978-1-61197-497-3. doi: 10.1137/1.9781611974973.83.

[64] David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80023-1. URL https://www.sciencedirect.com/science/article/pii/S0893608005800231.

[65] Alexander Shirkov Hang Zhang Pedro Larroy Mu Li Alexander Smola Nick Erickson, Jonas Mueller. Autogluon-tabular: Robust and accurate automl for structured data. 2020. doi: https://doi.org/10.48550/arXiv.2003.06505.

[66] Joaquin Vanschoren. *Meta-Learning*, pages 35–61. Springer International Publishing, Cham, 2019. ISBN 978-3-030-05318-5. doi: 10.1007/978-3-030-05318-5_2. URL https://doi.org/10.1007/978-3-030-05318-5_2.

[67] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Learning hyperparameter optimization initializations. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10, 2015. doi: 10.1109/DSAA.2015.7344817.

[68] Marius Lindauer and Frank Hutter. Warmstarting of model-based algorithm configuration. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32, 09 2017. doi: 10.1609/aaai.v32i1.11532.

[69] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems 28 (2015)*, pages 2962–2970, 2015.

[70] Kevin G. Jamieson and Ameet S. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. *ArXiv*, abs/1502.07943, 2016.

[71] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=S11KBYclx.

[72] Katharina Eggensperger, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 1114–1120. AAAI Press, 2015. ISBN 0262511290.

[73] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. pages 2423–2432, 06 2018. doi: 10.1109/CVPR.2018.00257.

[74] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc Le. Learning transferable architectures for scalable image recognition. pages 8697–8710, 06 2018. doi: 10.1109/CVPR.2018.00907.

[75] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

[76] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 2902–2911. JMLR.org, 2017.

[77] Martin Wistuba. Deep learning architecture search by neuro-cell-based evolution with function-preserving mutations. In Michele Berlingerio, Francesco Bonchi, Thomas Gärtner, Neil Hurley, and Georgiana Ifrim, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 243–258, Cham, 2019. Springer International Publishing. ISBN 978-3-030-10928-8.

[78] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. 11 2017.

[79] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=r1Ue8Hcxg.

[80] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 07 2017.

[81] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric P. Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 2020–2029, Red Hook, NY, USA, 2018. Curran Associates Inc.

[82] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '19, page 1946–1956, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330648. URL https://doi.org/10.1145/3292500.3330648.

[83] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1eYHoC5FX.

[84] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/933670f1ac8ba969f32989c312faba75-Paper.pdf.

[85] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. 07 2017.

[86] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[87] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 550–559. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/bender18a.html.

[88] Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, Chuan Zhou, Zongyuan Ge, and Steven Su. One-shot neural architecture search: Maximising diversity to overcome catastrophic forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):2921–2935, 2021. doi: 10.1109/TPAMI.2020.3035351.

[89] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, 18(25):1–5, 2017. URL http://jmlr.org/papers/v18/16-261.html.

[90] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05, page 507–523, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 9783642255656. doi: 10.1007/978-3-642-25566-3_40. URL https://doi.org/10.1007/978-3-642-25566-3_40.

[91] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. *arXiv:2007.04074 [cs.LG]*, 2020.

[92] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022. URL http://jmlr.org/papers/v23/21-0888.html.

[93] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 18, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138385. doi: 10.1145/1015330.1015432. URL https://doi.org/10.1145/1015330.1015432.

[94] Trang T Le, Weixuan Fu, and Jason H Moore. Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics*, 36(1):250–256, 2020.

[95] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13: 2171–2175, jul 2012.

[96] Erin LeDell. H2o automl: Scalable automatic machine learning. 2020.

[97] Microsoft. Neural Network Intelligence, 1 2021. URL https://github.com/microsoft/nni.

[98] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[99] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[100] Tianqi Chen, Mu Duan, Yutian Li, Min Zhang, Min Lin, Naiyan Wang, Yu Liu, Jian Wang, Denny Lian, and Qiong Liu. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[101] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2019.

[102] Douglas Hanahan and Robert A Weinberg. Hallmarks of cancer: the next generation. *Cell*, 144(5): 646–674, 2011.

[103] Janet Y Chen and Ingrid Judson. Combinatorial drug therapy for cancer in the post-genomic era. *Nature reviews cancer*, 4(12):931–939, 2004.

[104] John J Lee and Razelle Kurzrock. Combining immunotherapy and targeted therapies in cancer treatment. *Cancer discovery*, 8(1):6–19, 2018.

[105] R. Fisher, L. Pusztai, and C. Swanton. Cancer heterogeneity: implications for targeted therapeutics. *British journal of cancer*, 108:479–485, 2 2013. ISSN 1532-1827. doi: 10.1038/BJC.2012.581. URL https://pubmed.ncbi.nlm.nih.gov/23299535/.

[106] Genevieve Housman, Shannon Byler, Sarah Heerboth, Karolina Lapinska, Mckenna Longacre, Nicole Snyder, and Sibaji Sarkar. Drug resistance in cancer: An overview. *Cancers*, 6(3):1769–1792, 2014. ISSN 2072-6694. doi: 10.3390/cancers6031769. URL https://www.mdpi.com/2072-6694/6/3/1769.

[107] Tanweer Haider, Vikas Pandey, Nagma Banjare, Prem N. Gupta, and Vandana Soni. Drug resistance in cancer: mechanisms and tackling strategies. *Pharmacological reports : PR*, 72:1125–1151, 10 2020. ISSN 2299-5684. doi: 10.1007/S43440-020-00138-7. URL https://pubmed.ncbi.nlm.nih.gov/32700248/.

[108] Leena Gandhi, Delvys Rodríguez-Abreu, Shirish Gadgeel, Emilio Esteban, Enriqueta Felip, Flávia De Angelis, Manuel Domine, Philip Clingan, Maximilian J. Hochmair, Steven F. Powell, Susanna Y.-S. Cheng, Helge G. Bischoff, Nir Peled, Francesco Grossi, Ross R. Jennens, Martin Reck, Rina Hui, Edward B. Garon, Michael Boyer, Belén Rubio-Viqueira, Silvia Novello, Takayasu Kurata, Jhanelle E. Gray, John Vida, Ziwen Wei, Jing Yang, Harry Raftopoulos, M. Catherine Pietanza, and Marina C. Garassino. Pembrolizumab plus chemotherapy in metastatic non–small-cell lung cancer. *New England Journal of Medicine*, 378:2078–2092, 5 2018. ISSN 0028-4793. doi:

10.1056/NEJMOA1801005/SUPPL_FILE/NEJMOA1801005_DISCLOSURES.PDF. URL https://www.nejm.org/doi/full/10.1056/nejmoa1801005.

[109] Michael A. Davies, Philippe Saiag, Caroline Robert, Jean Jacques Grob, Keith T. Flaherty, Ana Arance, Vanna Chiarion-Sileni, Luc Thomas, Thierry Lesimple, Laurent Mortier, Stergios J. Moschos, David Hogg, Iván Márquez-Rodas, Michele Del Vecchio, Céleste Lebbé, Nicolas Meyer, Ying Zhang, Yingjie Huang, Bijoyesh Mookerjee, and Georgina V. Long. Dabrafenib plus trametinib in patients with braf v600–mutant melanoma brain metastases (combi-mb): a multi-cohort, open-label, phase 2 trial. *The Lancet. Oncology*, 18:863, 7 2017. ISSN 14745488. doi: 10.1016/S1470-2045(17)30429-1. URL /pmc/articles/PMC5991615//pmc/articles/PMC5991615/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC5991615/.

[110] Rachel W. Humphrey, Laura M. Brockway-Lunardi, David T. Bonk, Kathleen M. Dohoney, James H. Doroshow, Sandra J. Meech, Mark J. Ratain, Suzanne L. Topalian, and Drew M. Pardoll. Opportunities and challenges in the development of experimental drug combinations for cancer. *JNCI Journal of the National Cancer Institute*, 103:1222, 8 2011. ISSN 00278874. doi: 10.1093/JNCI/DJR246. URL /pmc/articles/PMC4415086//pmc/articles/PMC4415086/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC4415086/.

[111] Amancio Carnero. High throughput screening in drug discovery. *Clinical translational oncology : official publication of the Federation of Spanish Oncology Societies and of the National Cancer Institute of Mexico*, 8:482–490, 2006. ISSN 1699-048X. doi: 10.1007/S12094-006-0048-2. URL https://pubmed.ncbi.nlm.nih.gov/16870538/.

[112] A. V. Hill. The possible effects of the aggregation of the molecules of hæmoglobin on its dissociation curves. *The Journal of Physiology*, 40(4):iv–vii, 1910.

[113] Yohanan D, Chiara Fornari, Dennis Wang, séverine Mollard, Jo Bramhall, Frances Richards, and Duncan Jodrell. Combenefit: An interactive platform for the analysis and visualisation of drug combinations. *Bioinformatics*, 32:btw230, 04 2016. doi: 10.1093/bioinformatics/btw230.

[114] S. Loewe. The problem of synergism and antagonism of combined drugs. *Arzneimittel-Forschung*, 1953.

[115] Ting Chao Chou. Theoretical basis, experimental design, and computerized simulation of synergism and antagonism in drug combination studies. *Pharmacological Reviews*, 58:621–681, 9 2006. ISSN 0031-6997. doi: 10.1124/PR.58.3.

10. URL https://pharmrev.aspetjournals.org/content/58/3/621https://pharmrev.aspetjournals.org/content/58/3/621.abstract.

[116] C. I. BLISS. The toxicity of poisons applied jointly1. *Annals of Applied Biology*, 26:585–615, 8 1939. ISSN 1744-7348. doi: 10.1111/J.1744-7348.1939.TB06990.X. URL https://onlinelibrary.wiley.com/doi/full/10.1111/j.1744-7348.1939.tb06990.xhttps://onlinelibrary.wiley.com/doi/abs/10.1111/j.1744-7348.1939.tb06990.xhttps://onlinelibrary.wiley.com/doi/10.1111/j.1744-7348.1939.tb06990.x.

[117] Biosoft. Calcusyn. URL https://www.biosoft.com/calcusyn/.

[118] Liang-Chieh Chen, Maxwell D. Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens. Searching for efficient multi-scale architectures for dense image prediction. *CoRR*, abs/1809.04184, 2018. URL http://arxiv.org/abs/1809.04184.

[119] Xianzhi Du, Tsung-Yi Lin, Pengchong Jin, Golnaz Ghiasi, Mingxing Tan, Yin Cui, Quoc V. Le, and Xiaodan Song. Spinenet: Learning scale-permuted backbone for recognition and localization. *CoRR*, abs/1912.05027, 2019. URL http://arxiv.org/abs/1912.05027.

[120] Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. NAS-FPN: learning scalable feature pyramid architecture for object detection. *CoRR*, abs/1904.07392, 2019. URL http://arxiv.org/abs/1904.07392.

[121] Babatounde Moctard Oloulade, Jianliang Gao, Jiamin Chen, Raeed Al-Sabri, and Zhenpeng Wu. Cancer drug response prediction with surrogate modeling-based graph neural architecture search. *Bioinformatics*, 39(8):btad478, 08 2023. ISSN 1367-4811. doi: 10.1093/bioinformatics/btad478. URL https://doi.org/10.1093/bioinformatics/btad478.

[122] D. Baptista, P. G. Ferreira, and M. Rocha. A systematic evaluation of deep learning methods for the prediction of drug synergy in cancer. *PLoS Comput Biol*, 19(3):e1010200, Mar 2023.

[123] D. Baptista, J. Correia, B. Pereira, and M. Rocha. Evaluating molecular representations in machine learning models for drug response prediction and interpretability. *J Integr Bioinform*, 19(3), Sep 2022.

[124] Min Li, Yake Wang, Ruiqing Zheng, Xinghua Shi, Yaohang Li, Fang-Xiang Wu, and Jianxin Wang. Deepdsc: A deep learning method to predict drug sensitivity of cancer cell lines. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 18(2):575–582, 2021. doi: 10.1109/TCBB.2019.2919581.

[125] Nicolas Stransky et al. Jordi Barretina, Giordano Caponigro. The cancer cell line encyclopedia enables predictive modeling of anticancer drug sensitivity. *Nature*, 483:603, 3 2012. ISSN 00280836. doi: 10.1038/NATURE11003. URL /pmc/articles/PMC3320027//pmc/articles/PMC3320027/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC3320027/.

[126] Wanjuan Yang, Jorge Soares, Patricia Greninger, Elena J. Edelman, Howard Lightfoot, Simon Forbes, Nidhi Bindal, Dave Beare, James A. Smith, I. Richard Thompson, Sridhar Ramaswamy, P. Andrew Futreal, Daniel A. Haber, Michael R. Stratton, Cyril Benes, Ultan McDermott, and Mathew J. Garnett. Genomics of drug sensitivity in cancer (gdsc): a resource for therapeutic biomarker discovery in cancer cells. *Nucleic acids research*, 41, 1 2013. ISSN 1362-4962. doi: 10.1093/NAR/GKS1111. URL https://pubmed.ncbi.nlm.nih.gov/23180760/.

[127] Yoosup Chang, Hyejin Park, Hyun-Jin Yang, Seungju Lee, Kwee Yum Lee, Tae Kim, Jongsun Jung, and Jae-Min Shin. Cancer drug response profile scan (cdrscan): A deep learning model that predicts drug effectiveness from cancer genomic signature. *Scientific Reports*, 8, 06 2018. doi: 10.1038/s41598-018-27214-6.

[128] Harry C. Jubb et al. John G. Tate, Sally Bamford. Cosmic: the catalogue of somatic mutations in cancer. *Nucleic acids research*, 47:D941–D947, 1 2019. ISSN 1362-4962. doi: 10.1093/NAR/GKY1015. URL https://pubmed.ncbi.nlm.nih.gov/30371878/.

[129] Hossein Sharifi-Noghabi, Olga Zolotareva, Colin C Collins, and Martin Ester. MOLI: multi-omics late integration with deep neural networks for drug response prediction. *Bioinformatics*, 35(14):i501–i509, 07 2019. ISSN 1367-4803. doi: 10.1093/bioinformatics/btz318. URL https://doi.org/10.1093/bioinformatics/btz318.

[130] Yiheng Zhu, Zhenqiu Ouyang, Wenbo Chen, Ruiwei Feng, Danny Z. Chen, Ji Cao, and Jian Wu. Tgsa: protein–protein association-based twin graph neural networks for drug response prediction with similarity augmentation. *Bioinformatics*, 38:461–468, 1 2022. ISSN 1367-

4803. doi: 10.1093/BIOINFORMATICS/BTAB650. URL https://academic.oup.com/bioinformatics/article/38/2/461/6374919.

[131] Teague Sterling and John J. Irwin. Zinc 15 - ligand discovery for everyone. *Journal of Chemical Information and Modeling*, 55:2324–2337, 11 2015. ISSN 1549960X. doi: 10.1021/ACS.JCIM.5B00559/ASSET/IMAGES/LARGE/CI-2015-00559J_0005.JPEG. URL https://pubs.acs.org/doi/full/10.1021/acs.jcim.5b00559.

[132] Anna Gaulton, Louisa J. Bellis, A. Patricia Bento, Jon Chambers, Mark Davies, Anne Hersey, Yvonne Light, Shaun McGlinchey, David Michalovich, Bissan Al-Lazikani, and John P. Overington. Chembl: a large-scale bioactivity database for drug discovery. *Nucleic acids research*, 40, 1 2012. ISSN 1362-4962. doi: 10.1093/NAR/GKR777. URL https://pubmed.ncbi.nlm.nih.gov/21948594/.

[133] Jihye Shin, Yinhua Piao, Dongmin Bang, Sun Kim, and Kyuri Jo. Drpreter: Interpretable anticancer drug response prediction using knowledge-guided graph neural networks and transformer. *International journal of molecular sciences*, 23:13919, 11 2022. ISSN 1422-0067. doi: 10.3390/IJMS232213919. URL https://pubmed.ncbi.nlm.nih.gov/36430395/.

[134] Kristina Preuer, Richard P.I. Lewis, Sepp Hochreiter, Andreas Bender, Krishna C. Bulusu, and Günter Klambauer. Deepsynergy: predicting anti-cancer drug synergy with deep learning. *Bioinformatics (Oxford, England)*, 34:1538–1546, 5 2018. ISSN 1367-4811. doi: 10.1093/BIOINFORMATICS/BTX806. URL https://pubmed.ncbi.nlm.nih.gov/29253077/.

[135] Jennifer O'Neil, Yair Benita, Igor Feldman, Melissa Chenard, Brian Roberts, Yaping Liu, Jing Li, Astrid Kral, Serguei Lejnine, Andrey Loboda, William Arthur, Razvan Cristescu, Brian B. Haines, Christopher Winter, Theresa Zhang, Andrew Bloecher, and Stuart D. Shumway. An unbiased oncology compound screen to identify novel combination strategies. *Molecular cancer therapeutics*, 15:1155–1162, 6 2016. ISSN 1538-8514. doi: 10.1158/1535-7163.MCT-15-0843. URL https://pubmed.ncbi.nlm.nih.gov/26983881/.

[136] Halil Ibrahim Kuru, Oznur Tastan, and A. Ercument Cicek. Matchmaker: A deep learning framework for drug synergy prediction. *IEEE/ACM transactions on computational biology and bioinformatics*, 19:2334–2344, 2022. ISSN 1557-9964. doi: 10.1109/TCBB.2021.3086702. URL https://pubmed.ncbi.nlm.nih.gov/34086576/.

[137] Hui Liu, Wenhao Zhang, Bo Zou, Jinxian Wang, Yuanyuan Deng, and Lei Deng. Drugcombdb: a comprehensive database of drug combinations toward the discovery of combinatorial therapy. *Nucleic acids research*, 48:D871–D881, 1 2020. ISSN 1362-4962. doi: 10.1093/NAR/GKZ1007. URL https://pubmed.ncbi.nlm.nih.gov/31665429/.

[138] Jinxian Wang, Xuejun Liu, Siyuan Shen, Lei Deng, and Hui Liu. Deepdds: deep graph neural network with attention mechanism to predict synergistic drug combinations. *Briefings in Bioinformatics*, 23, 1 2022. ISSN 14774054. doi: 10.1093/BIB/BBAB390. URL https://academic.oup.com/bib/article/23/1/bbab390/6375262.

[139] Dávid Bajusz, Anita Rácz, and Károly Héberger. *Chemical Data Formats, Fingerprints, and Other Molecular Descriptions for Database Analysis and Searching*. 12 2017. ISBN 9780124095472. doi: 10.1016/B978-0-12-409547-2.12345-5.

[140] R. Todeschini, V. Consonni, R. Mannhold, H. Kubinyi, and H. Timmerman. *Handbook of Molecular Descriptors*. Methods & Principles in Medicinal Chemistry. Wiley, 2008. ISBN 9783527613113. URL https://books.google.pt/books?id=TCuHqbvgMbEC.

[141] Sunghwan Kim, Jie Chen, Tiejun Cheng, Asta Gindulyte, Jia He, Siqian He, Qingliang Li, Benjamin A. Shoemaker, Paul A. Thiessen, Bo Yu, Leonid Zaslavsky, Jian Zhang, and Evan E. Bolton. Pubchem in 2021: new data content and improved web interfaces. *Nucleic acids research*, 49:D1388–D1395, 1 2021. ISSN 1362-4962. doi: 10.1093/NAR/GKAA971. URL https://pubmed.ncbi.nlm.nih.gov/33151290/.

[142] Keyulu Xu, Stefanie Jegelka, Weihua Hu, and Jure Leskovec. How powerful are graph neural networks? *7th International Conference on Learning Representations, ICLR 2019*, 10 2018. doi: 10.48550/arxiv.1810.00826. URL https://arxiv.org/abs/1810.00826v3.

[143] Petar Veličković, Arantxa Casanova, Pietro Liò, Guillem Cucurull, Adriana Romero, and Yoshua Bengio. Graph attention networks. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 10 2017. doi: 10.48550/arxiv.1710.10903. URL https://arxiv.org/abs/1710.10903v3.

[144] Sebastian Raschka, Joshua Patterson, and Corey Nolet. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, 11(4), 2020. ISSN 2078-2489. doi: 10.3390/info11040193. URL https://www.mdpi.com/2078-2489/11/4/193.

[145] Lucas Zimmer, Marius Lindauer, and Frank Hutter. Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl. *arXiv preprint arXiv:2006.13799*, 2020.

[146] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

[147] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing.

[148] Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. *CoRR*, abs/1902.09635, 2019. URL http://arxiv.org/abs/1902.09635.

[149] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *CoRR*, abs/2001.00326, 2020. URL http://arxiv.org/abs/2001.00326.

[150] Jin Xu, Xu Tan, Renqian Luo, Kaitao Song, Jian Li, Tao Qin, and Tie-Yan Liu. NAS-BERT: task-agnostic and adaptive-size BERT compression with neural architecture search. *CoRR*, abs/2105.14444, 2021. URL https://arxiv.org/abs/2105.14444.

[151] Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Jinzhu Li, Sheng Zhao, Enhong Chen, and Tie-Yan Liu. Lightspeech: Lightweight and fast text to speech with neural architecture search. *CoRR*, abs/2102.04040, 2021. URL https://arxiv.org/abs/2102.04040.

[152] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017. URL http://arxiv.org/abs/1710.05941.

[153] Anna Veronika Dorogush, Andrey Gulin, Gleb Gusev, Nikita Kazeev, Liudmila Ostroumova Prokhorenkova, and Aleksandr Vorobev. Fighting biases with dynamic boosting. *CoRR*, abs/1706.09516, 2017. URL http://arxiv.org/abs/1706.09516.

[154] Yejin Hwang and Jongwoo Song. Recent deep learning methods for tabular data. *Communications for Statistical Applications and Methods*, 30:215–226, 3 2023. ISSN 2287-7843. doi: 10.29220/CSAM.2023.30.2.215. URL http://www.csam.or.kr/journal/view.html?doi=10.29220/CSAM.2023.30.2.215.

[155] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *CoRR*, abs/2106.11959, 2021. URL https://arxiv.org/abs/2106.11959.

[156] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *CoRR*, abs/1611.02167, 2016. URL http://arxiv.org/abs/1611.02167.

[157] Mechanisms of action (moa) prediction | kaggle, . URL https://www.kaggle.com/c/lish-moa/discussion/202256.

[158] Krishna Teja Chitty-Venkata, Murali Emani, Venkatram Vishwanath, and Arun K. Somani. Neural architecture search for transformers: A survey. *IEEE Access*, 10:108374–108412, 2022. doi: 10.1109/ACCESS.2022.3212767.

[159] Mseloss — pytorch 2.0 documentation, . URL https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html.

[160] Bcewithlogitsloss — pytorch 2.0 documentation, . URL https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html.

[161] Crossentropyloss — pytorch 2.0 documentation, . URL https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html.

[162] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12(null):2539–2561, nov 2011. ISSN 1532-4435.

[163] Shirui Pan, Jia Wu, Xingquan Zhu, Guodong Long, and Chengqi Zhang. Finding the best not the most: regularized loss minimization subgraph selection for graph classification. *Pattern Recognition*, 48(11):3783–3796, 2015. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2015.05.019. URL https://www.sciencedirect.com/science/article/pii/S0031320315001983.

[164] Isidro Cortés-Ciriano and Andreas Bender. Kekulescope: prediction of cancer cell line sensitivity and compound potency using convolutional neural networks trained on compound images. *Journal of cheminformatics*, 11, 2019. ISSN 1758-2946. doi: 10.1186/S13321-019-0364-5. URL https://pubmed.ncbi.nlm.nih.gov/31218493/.

[165] H. L. Morgan. The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service. *Journal of Chemical Documentation*, 5(2):107–113, 1965. doi: 10.1021/c160017a018. URL https://doi.org/10.1021/c160017a018.

[166] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, 2010. doi: 10.1021/ci100050t. URL https://doi.org/10.1021/ci100050t. PMID: 20426451.

[167] Joseph L. Durant, Burton A. Leland, Douglas R. Henry, and James G. Nourse. Reoptimization of mdl keys for use in drug discovery. *Journal of Chemical Information and Computer Sciences*, 42(6):1273–1280, 2002. doi: 10.1021/ci010132r. URL https://doi.org/10.1021/ci010132r. PMID: 12444722.

[168] Raymond E. Carhart, Dennis H. Smith, and R. Venkataraghavan. Atom pairs as molecular features in structure-activity studies: definition and applications. *Journal of Chemical Information and Computer Sciences*, 25(2):64–73, 1985. doi: 10.1021/ci00046a002. URL https://doi.org/10.1021/ci00046a002.

[169] G. Landrum. Rdkit: Open-source cheminformatics. URL https://www.rdkit.org/.

[170] Sabrina Jaeger, Simone Fulle, and Samo Turk. Mol2vec: Unsupervised machine learning approach with chemical intuition. *Journal of Chemical Information and Modeling*, 58(1):27–35, 2018. doi: 10.1021/acs.jcim.7b00616. URL https://doi.org/10.1021/acs.jcim.7b00616. PMID: 29268609.

[171] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. 2013.

[172] U. V. Ucak, I. Ashyrmamatov, and J. Lee. Improving the quality of chemical language model outcomes with atom-in-SMILES tokenization. *J Cheminform*, 15(1):55, May 2023.

[173] Patrick Hop, Brandon Allgood, and Jessen Yu. Geometric deep learning autonomously learns chemical features that outperform those engineered by domain experts. *Molecular Pharmaceutics*, 15(10):4371–4377, 2018. doi: 10.1021/acs.molpharmaceut.7b01144. URL https://doi.org/10.1021/acs.molpharmaceut.7b01144. PMID: 29863875.

[174] Alice Capecchi, Daniel Probst, and Jean-Louis Reymond. One molecular fingerprint to rule them all: Drugs, biomolecules, and the metabolome. *Journal of Cheminformatics*, 12, 06 2020. doi: 10.1186/s13321-020-00445-4.

[175] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chem. Sci.*, 9:513–530, 2018. doi: 10.1039/C7SC02664A. URL http://dx.doi.org/10.1039/C7SC02664A.

[176] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search, 2019.

[177] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search, 2019.

[178] Marc-André Zöller and Marco F. Huber. Benchmark and survey of automated machine learning frameworks, 2021.

[179] Augustin Luna, Fathi Elloumi, Sudhir Varma, Yanghsin Wang, Vinodh N Rajapakse, Mirit I Aladjem, Jacques Robert, Chris Sander, Yves Pommier, and William C Reinhold. CellMiner Cross-Database (CellMinerCDB) version 1.2: Exploration of patient-derived cancer cell line pharmacogenomics. *Nucleic Acids Research*, 49(D1):D1083–D1093, 11 2020. ISSN 0305-1048. doi: 10.1093/nar/gkaa968. URL https://doi.org/10.1093/nar/gkaa968.

[180] A. Bento, Anne Hersey, Eloy Félix, Greg Landrum, Anna Gaulton, Francis Atkinson, Louisa Bellis, Marleen de Veij, and Andrew Leach. An open source chemical structure curation pipeline using rdkit. *Journal of Cheminformatics*, 12:51, 09 2020. doi: 10.1186/s13321-020-00456-1.

[181] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2020.

[182] Aravind Subramanian, Pablo Tamayo, Vamsi K. Mootha, Sayan Mukherjee, Benjamin L. Ebert, Michael A. Gillette, Amanda Paulovich, Scott L. Pomeroy, Todd R. Golub, Eric S. Lander, and Jill P. Mesirov. Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences*, 102(43):15545–15550, 2005. doi: 10.1073/pnas.0506580102. URL https://www.pnas.org/doi/abs/10.1073/pnas.0506580102.

[183] A. Liberzon, C. Birger, H. ttir, M. Ghandi, J. P. Mesirov, and P. Tamayo. The Molecular Signatures Database (MSigDB) hallmark gene set collection. *Cell Syst*, 1(6):417–425, Dec 2015.

[184] B. Jassal, L. Matthews, G. Viteri, C. Gong, P. Lorente, A. Fabregat, K. Sidiropoulos, J. Cook, M. Gillespie, R. Haw, F. Loney, B. May, M. Milacic, K. Rothfels, C. Sevilla, V. Shamovsky, S. Shorser, T. Varusai, J. Weiser, G. Wu, L. Stein, H. Hermjakob, and P. D'Eustachio. The reactome pathway knowledgebase. *Nucleic Acids Res*, 48(D1):D498–D503, Jan 2020.

[185] A. Subramanian, R. Narayan, S. M. Corsello, D. D. Peck, T. E. Natoli, X. Lu, J. Gould, J. F. Davis, A. A. Tubelli, J. K. Asiedu, D. L. Lahr, J. E. Hirschman, Z. Liu, M. Donahue, B. Julian, M. Khan, D. Wadden, I. C. Smith, D. Lam, A. Liberzon, C. Toder, M. Bagul, M. Orzechowski, O. M. Enache, F. Piccioni, S. A. Johnson, N. J. Lyons, A. H. Berger, A. F. Shamji, A. N. Brooks, A. Vrcic, C. Flynn, J. Rosains, D. Y. Takeda, R. Hu, D. Davison, J. Lamb, K. Ardlie, L. Hogstrom, P. Greenside, N. S. Gray, P. A. Clemons, S. Silver, X. Wu, W. N. Zhao, W. Read-Button, X. Wu, S. J. Haggarty, L. V. Ronco, J. S. Boehm, S. L. Schreiber, J. G. Doench, J. A. Bittker, D. E. Root, B. Wong, and T. R. Golub. A Next Generation Connectivity Map: L1000 Platform and the First 1,000,000 Profiles. *Cell*, 171(6): 1437–1452, Nov 2017.

[186] D. Repana, J. Nulsen, L. Dressler, M. Bortolomeazzi, S. K. Venkata, A. Tourna, A. Yakovleva, T. Palmieri, and F. D. Ciccarelli. The Network of Cancer Genes (NCG): a comprehensive catalogue of known and candidate cancer genes from cancer sequencing screens. *Genome Biol*, 20 (1):1, Jan 2019.

[187] Paul Morris, Rachel St. Clair, William Edward Hahn, and Elan Barenholtz. Predicting binding from screening assays with transformer network embeddings. *Journal of Chemical Information and Modeling*, 60(9):4191–4199, 2020. doi: 10.1021/acs.jcim.9b01212. URL https://doi.org/10.1021/acs.jcim.9b01212. PMID: 32568539.

[188] Omid Bazgir, Souparno Ghosh, and Ranadip Pal. Investigation of REFINED CNN ensemble learning for anti-cancer drug sensitivity prediction. *Bioinformatics*, 37:i42–i50, 07 2021. ISSN 1367-4803. doi: 10.1093/bioinformatics/btab336. URL https://doi.org/10.1093/bioinformatics/btab336.

[189] Y. Zhu, T. Brettin, F. Xia, A. Partin, M. Shukla, H. Yoo, Y. A. Evrard, J. H. Doroshow, and R. L. Stevens. Converting tabular data into images for deep learning with convolutional neural networks. *Sci Rep*, 11(1):11325, May 2021.

[190] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. doi: 10.1109/4235.585893.

[191] David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80023-1. URL https://www.sciencedirect.com/science/article/pii/S0893608005800231.

[192] Jie Gao, Jing Hu, Wanqing Sun, Yili Shen, Xiaonan Zhang, Xiaomin Fang, Fan Wang, and Guodong Zhao. Tcr: A transformer based deep network for predicting cancer drugs response, 2022.

[193] Likun Jiang, Changzhi Jiang, Xinyu Yu, Rao Fu, Shuting Jin, and Xiangrong Liu. DeepTTA: a transformer-based model for predicting cancer drug response. *Briefings in Bioinformatics*, 23(3): bbac100, 03 2022. ISSN 1477-4054. doi: 10.1093/bib/bbac100. URL https://doi.org/10.1093/bib/bbac100.

[194] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

[195] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016.

[196] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. doi: 10.1609/aaai.v32i1.11491. URL https://ojs.aaai.org/index.php/AAAI/article/view/11491.

[197] Robert H. Shoemaker. The nci60 human tumour cell line anticancer drug screen. *Nature Reviews Cancer 2006 6:10*, 6:813–823, 10 2006. ISSN 1474-1768. doi: 10.1038/nrc1951. URL https://www.nature.com/articles/nrc1951.

Appendix A
NNI-AutoGluon preliminary performance comparison

A series of preliminary tests were conducted to evaluate the ability of NNI and AutoGluon to optimize DL models and compare their different search strategies. The tests were carried out using a single-cell line cytotoxicity dataset, CCRF-CEM, from a drug sensitivity prediction study [164], and a larger dataset derived from the NCI-60 [197] dataset isolating the A549/ATCC cell line. These datasets map SMILES strings to their $-log(\text{IC}_{50})$ and $-log(\text{GI}_{50})$ values, respectively. Each test was repeated three times to ensure the consistency and reliability of the results.

The tests were executed under standardized conditions, using a single NVIDIA GeForce RTX 2080 Ti GPU, allowing 10 hours for training and up to a max of 100 epochs or 20 epochs without improvement for each trial. This ensured a fair comparison of NNI and AutoGluon by conducting tests under consistent conditions. For each dataset, some values were recorded, including the number of trials tested, the score of the best trial on the train set, its score on the test set (using the best trial configuration), and the average score along all trials. The metric used was average squared error.

Each search strategy available was tested for both tools except for NNI's one-shot strategies since these algorithms are not yet fully supporting every mutation. AutoGluon's built-in fully-connected network was the only model tested using this tool.

A549/ATCC Dataset All test cases for both NNI and AutoGluon reached 10 hours, meaning neither tool exhausted its possible configurations. Table 8 shows the average values obtained across the three iterations for the A549/ATCC dataset (refer to Appendix B for individual results). AutoGluon was found to have more consistent values between each of the three iterations (Appendix B). AutoGluon consistently tested fewer configurations than NNI, likely due to bagging and ensembling since these features result in additional computational overhead. In addition, AutoGluon's search strategies did not seem to have much impact on performance.

Regarding NNI's search strategies, some unexpected results were observed, making it difficult to draw definitive conclusions. Results were inconsistent between the three iterations, i.e., despite the reinforce-

ment learning strategy having achieved the best test set performance in two iterations, it ranked second worst in the remaining. Grid search achieved a better average train score across all tested configurations than any other search algorithm, which is surprising since this algorithm uses a brute-force approach. Additionally, it was investigated if any search strategy converged to the same best configuration; however, this was not the case, even across iterations using the same search strategy.

Table 8: A549/ATCC Dataset. Results represent the average between the three rounds of tests.

AutoML tool	Search Strategy	Number of trials	Best train score	Best test score	Avg trial score
NNI	GridSearch	317	0.5958	0.6310	0.6574
	Random	405	0.5870	0.6196	0.8058
	RegularizedEvolution	288	0.5759	0.6346	0.6998
	TPE	373	0.5876	0.6359	0.7714
	PolicyBasedRL	472	0.6014	0.6224	0.9349
AutoGluon	Random	126	0.5420	0.5412	1.4220
	Bayes	136	0.5437	0.5455	1.2014

The best test score represents the score achieved by the model configuration with the highest training score when evaluated on the test set. The average trial score represents the average score across all configurations evaluated.

CCRF-CEM Dataset As expected, both tools reached the time limit in every test case since the number of possible configurations is much higher than the number of evaluated configurations. Table 9 shows the average values obtained across the three iterations for the CCRF-CEM dataset (refer to Appendix C for individual results). AutoGluon showed better and more consistent results between the three iterations once more.

Regarding NNI's search strategies, the evolutionary algorithm strategy showed the best average score; thus, it may be possible to conclude that this algorithm succeeded in finding a local optimum. This strategy also consistently revealed good results on the best train configuration. The reinforcement learning strategy showed a high average score in all test cases, particularly in the second round (Table 14). This time, grid search showed the least performing best train configuration, which aligns with expectations; however, this algorithm still showed the second-best average trial score across all tested configurations. Once more, none of the test cases converged to the same configuration.

In terms of performance, the comparison showed that AutoGluon had better results. This is due to bagging and stacking ensembling methods used by AutoGluon, but also to the model space used to test NNI. The model was not very elaborate; it was a simple model developed to understand every mutation primitive NNI offers; there is still room for improvement in NNI's results. Nonetheless, AutoGluon's results for its fully-connected model were remarkable, reaching the same performance as the published results.

Table 9: CCRF-CEM Dataset. Results represent the average between the three rounds of tests.

AutoML tool	Search Strategy	Number of trials	Best train score	Best test score	Avg trial score
NNI	GridSearch	1161	0.5130	0.6781	0.7822
	Random	1176	0.4855	0.5540	0.9221
	RegularizedEvolution	1061	0.4643	0.5530	0.6207
	TPE	1180	0.4850	0.5474	0.9173
	PolicyBasedRL	1227	0.4909	0.5937	1.7287
AutoGluon	Random	299	0.4680	0.4427	1.0648
	Bayes	299	0.4758	0.4394	0.9176

The best test score represents the score achieved by the model configuration with the highest training score when evaluated on the test set. The average trial score represents the average score across all configurations evaluated.

The evaluation of NNI's search strategies presented some challenges, as unexpected results were observed that made it difficult to draw definitive conclusions. Therefore, further testing is required to fully understand and compare these algorithms. Despite these results, NNI's search strategies consistently showed a better average trial score than AutoGluon, which means these algorithms spent less time on under-promising configurations than their counterpart.

Appendix B
A549/ATCC NNI-AutoGluon comparison study

Table 10: A549/ATCC Dataset - Round 1. Scores represent the average squared error.

AutoML tool	Search Strategy	Number of trials	Best train score	Best test score	Avg trial score
NNI	GridSearch	333	0.5879	0.6244	0.6173
	Random	427	0.5820	0.6209	0.8052
	RegularizedEvolution	281	0.5738	0.6237	0.6835
	TPE	372	0.5859	0.6608	0.7673
	PolicyBasedRL	512	0.6008	0.6507	0.8463
AutoGluon	Random	140	0.5405	0.5416	1.5273
	Bayes	150	0.5469	0.5518	1.4069

The best test score represents the score achieved by the model configuration with the highest training score when evaluated on the test set. The average trial score represents the average score across all configurations evaluated.

Table 11: A549/ATCC Dataset - Round 2. Scores represent the average squared error.

AutoML tool	Search Strategy	Number of trials	Best train score	Best test score	Avg trial score
NNI	GridSearch	355	0.6027	0.6308	0.7064
	Random	405	0.5908	0.6234	0.8062
	RegularizedEvolution	291	0.5765	0.6384	0.6976
	TPE	393	0.5893	0.6273	0.7792
	PolicyBasedRL	420	0.6027	0.6073	0.8263
AutoGluon	Random	119	0.5404	0.5438	1.3562
	Bayes	149	0.5461	0.5444	1.1598

The best test score represents the score achieved by the model configuration with the highest training score when evaluated on the test set. The average trial score represents the average score across all configurations evaluated.

Table 12: A549/ATCC Dataset - Round 3. Scores represent the average squared error.

AutoML tool	Search Strategy	Number of trials	Best train score	Best test score	Avg trial score
NNI	GridSearch	265	0.5970	0.6379	0.6484
	Random	383	0.5882	0.6146	0.8059
	RegularizedEvolution	291	0.5775	0.6418	0.7184
	TPE	354	0.5855	0.6195	0.7677
	PolicyBasedRL	484	0.6008	0.6093	1.1322
AutoGluon	Random	121	0.5450	0.5382	1.3830
	Bayes	110	0.5381	0.5404	1.0375

The best test score represents the score achieved by the model configuration with the highest training score when evaluated on the test set. The average trial score represents the average score across all configurations evaluated.

Appendix C
CCRF-CEM NNI-AutoGluon comparison study

Table 13: CCRF-CEM Dataset - Round 1. Scores represent the average squared error.

AutoML tool	Search Strategy	Number of trials	Best train score	Best test score	Avg trial score
NNI	GridSearch	1072	0.5036	0.8320	0.6274
	Random	1210	0.4916	0.6076	0.9143
	RegularizedEvolution	1139	0.4798	0.5369	0.6392
	TPE	1215	0.4873	0.4921	0.9256
	PolicyBasedRL	1184	0.5007	0.6333	1.1525
AutoGluon	Random	298	0.4731	0.4339	1.0569
	Bayes	306	0.4774	0.4348	1.0135

The best test score represents the score achieved by the model configuration with the highest training score when evaluated on the test set. The average trial score represents the average score across all configurations evaluated.

Table 14: CCRF-CEM Dataset - Round 2. Scores represent the average squared error.

AutoML tool	Search Strategy	Number of trials	Best train score	Best test score	Avg trial score
NNI	GridSearch	1170	0.4865	0.5450	0.9718
	Random	1159	0.4811	0.5395	0.9030
	RegularizedEvolution	1069	0.4570	0.5914	0.6293
	TPE	1166	0.4858	0.5488	0.9086
	PolicyBasedRL	1346	0.5016	0.6386	3.7254
AutoGluon	Random	278	0.4684	0.4510	1.1132
	Bayes	273	0.4773	0.4435	0.8793

The best test score represents the score achieved by the model configuration with the highest training score when evaluated on the test set. The average trial score represents the average score across all configurations evaluated.

Table 15: CCRF-CEM Dataset - Round 3. Scores represent the average squared error.

AutoML tool	Search Strategy	Number of trials	Best train score	Best test score	Avg trial score
NNI	GridSearch	1243	0.5487	0.6573	0.7473
	Random	1161	0.4837	0.5150	0.9490
	RegularizedEvolution	977	0.4560	0.5304	0.5945
	TPE	1161	0.4821	0.6013	0.9177
	PolicyBasedRL	1151	0.4729	0.6093	1.3083
AutoGluon	Random	322	0.4627	0.4434	1.0243
	Bayes	319	0.4728	0.4400	0.8600

The best test score represents the score achieved by the model configuration with the highest training score when evaluated on the test set. The average trial score represents the average score across all configurations evaluated.