Orchestrating and specializing networks of classifiers

Fatah Fattah

fatahfattah@hotmail.com

July 19, 2021, 50 pages

Academic supervisor: Dr. Giovanni Sileno, g.sileno@uva.nl
Daily supervisors: Dr. Camilo Thorne, c.thorne.1@elsevier.com
Dr. Markus Schwoerer, m.schwoerer1@elsevier.com
Host organisation/Research group: Elsevier, Data Science, Life Sciences, https://www.elsevier.com/

Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Master Software Engineering
http://www.software-engineering-amsterdam.nl

Acknowledgements

During the course of writing this thesis, I could have not wished for better supervision. I would like to thank Giovanni Sileno for his inexhaustible ideas and for always helping me see the bigger picture, Camilo Thorne for his critical view through his broad knowledge, and Markus Schwoerer for helping me understand the classification task and support in gathering the datasets. Finally, I want to thank my family and friends for their never-ending support.

Abstract

Throughout history, divide-et-impera (problem decomposition) has been a popular general approach to problem-solving and it has proven itself to have many benefits in application, yet it can be rather complex to implement in a controlled and principled fashion. The idea has been applied in various technical and non-technical fields, ranging from software engineering to politics.

However, within the field of machine learning, we lack a theory and tooling to control such principled decomposition, which has traditionally been tackled using tightly coupled monolithic solutions (*i.e.*, ensemble-learning).

When considering machine learning models as independent computational units (or agents), we can think about them within a network of such agents. The relationships over the outputs of these agents form a sort of social structure that we can then coordinate to achieve some common (classification) goal. In this work, we elaborate a theory on how one can perform such coordination (or orchestration) and propose a proof-of-concept tool that encapsulates this idea which enables one to put it into practice.

Furthermore, we touch upon the explainable AI (or XAI) problem by exploring ways to integrate high-level (symbolic) reasoning with low-level (sub-symbolic) perception using Answer Set Programming. This has allowed us to embed explicit knowledge into the inferential process, with which we come a little closer to being able to reason why a system provides a certain output.

Finally, we attempt to leverage the idea of orchestration within a network of classifiers, to create specialized versions of an individual classifier, using the knowledge of the whole. This has resulted in promising performance improvements over the specific specialization tasks. We explore this idea further by creating a of joint-decision structure over the specialized classifiers to improve the original classification task. However, further investigation is required for the unexpected low performance of this step.

Contents

Chapter 1

Introduction

Increasingly complex computational goals have created the need for new solutions that decrease complexity in a principled and decomposed manner. Monolithic solutions that try to *do it all* might not be as well-equipped to handle certain situations in comparison to breaking up the problem into smaller pieces. Various fields recur to design principles for organizing systems in ways that enable cooperation and information sharing of such decomposed systems, to reach a common goal.

The famous divide-et-impera (problem decomposition) general approach to problem-solving, well-known in software engineering as well, has found computational instantiation in the concept of multi-agent systems (MAS), which are networks of individual systems (agents) that each try to tackle a smaller part of a common goal [1]. Splitting a system up into smaller components is close to a software-engineering approach and in line with good practices such as separation of concerns and loose coupling. However, it also could add complexity, and having a good principled way to tackle such a decomposition could prove itself to be a difficult task.

When we take this idea of task decomposition and try to apply it to machine learning, we quickly find out topics in which it can prove beneficial. More specifically, classification tasks that require the input of multiple models, often use ensemble learning methods that try to incorporate all models into one big multi-modal network. A prototypical example of this approach is given by the "One model to learn them all" paper by Kaiser et al. [2]. However, this leads to a tightly coupled system in which the whole network is trained in its entirety, and for this reason, the approach becomes very expensive from data and computational requirements because of the sheer size of the network. *e.g.*, a forward- and backward pass needs to go through all sub-networks, which leads to slower network throughput, especially during training. Such a network also decreases the degree to which we can reason about the whole, because of the many layers of abstraction that are added [3].

An alternative approach to this monolithic model is to separate the sub-networks into their own encapsulated units and "orchestrate" their outputs during the inferential process. This allows the orchestrator to pick-and-choose which outputs are relevant for the classification task and base the final prediction on (a combination of) the results of the individual models in the network. In a sense, the idea is to create a "social structure" of models that can communicate with each other, following the communication channels set up by the orchestrator.

This approach allows the monolith to be broken up into smaller pieces, which increases the flexibility and scalability of the system. The goal of this thesis is to elaborate a theory on how such principled decomposition can be structured and provide the tooling to orchestrate the network of models. Finally, we make use of a real-world classification task in this whole process, which is defined by the host organization, Elsevier. This is to better elaborate the ideas and concepts, prove real-world usability, and as means of experimentation and development.

1.1 Problem statement

Systems that use multiple machine learning models/classifiers, often manifest a natural hierarchy in the data flow leading to the classification result. If we take self-driving cars as an example, the network of

models might include one model that recognizes stop signs, another model to detect cars, and a third model to determine the state of the road (e.g. wet, snowy). The results from all of these models can have an impact on decisions such as slowing down or braking. On a high-level, there is the need for these models to communicate with each other and share their perception of the task/sub-task in a coordinated fashion to produce relevant inferences. øeg when we detect a "slippery road" sign and we observe that the road is wet, we want to limit our speed.

Combining multiple ML models towards one or multiple classification tasks has been done before. One popular example is from Kaiser et al. [2], in which the researchers have combined multiple models of various types (NLP and computer vision) into one network. Issues with this approach are that one loses some ability to reason on what the model does because of the layers of abstraction which are added. It could also increase the chance of over-fitting because of the increased capacity of the whole network [3]. Training the model will become more difficult in a practical sense, since the entire model, including all sub-networks, will have to be trained at the same time, depending on how the network is tied together. The sub-networks overfit and generalize at different rates as well [3], in other words, the models do not reach a locally optimal state at the same time. Lastly, versioning of the networks becomes more difficult since it is non-trivial to interchange sub-networks for selecting the most adequate models for the task. We think that some of these issues will be addressed with our approach.

Integrating symbolic and sub-symbolic computation is an important by-product of model orchestration. Orchestration requires some way to explicitly define the communication channels between our models so that we can control our data flow. Recently, efforts have been made in this integration of symbolic and sub-symbolic computation. DeepProbLog [4] is an example in which neural networks and expressive logical modelling were incorporated. NeurASP [5] is a more recent effort that tries to incorporate neural networks into answer set programming (ASP) and shows how neural networks can learn from these explicit semantic constraints which are expressed by the rules. Even though these efforts are closely related to the goal of this thesis, they focus on semantic constraints on single neural networks rather than constraints on the coordination of a set of models.

In general, the field of machine learning lacks a formal and principled way to perform such decomposition of machine learning models, together with the ability to integrate high-level reasoning (symbolic) with low-level perception (sub-symbolic).

1.1.1 Research questions

To further pursue our goal for this project, research the problem domain, and propose a solution, we have defined the following questions.

RQ 1 - How can a network of machine learning models be controlled to tackle a classification task?

SQ 1.1 - How can we define a social structure over a set of machine learning models?

SQ 1.2 How to select constraints and protocols that intervene on the inferential process of the network of models?

RQ 2 - How can we leverage a network of models to perform (re-)training?

SQ 2.1 - What benefits can be gained in terms of training performance and validation accuracy?

Motivating hypotheses

We define the following motivating hypotheses:

- When considering machine learning models as independent computational agents and their relationships as logic programming rules, we are able to form them into social structures. Once we have a social structure, we are better equipped to apply governance or orchestration over them.
- The hierarchization of machine learning models allows for better reasoning on their relationships and why/how a certain outcome was concluded.

- The decoupling of machine learning models at the inference level, rather than at the training level, allows for more separation and modularity in a network of classifiers.
- We can improve individual classifiers by leveraging the knowledge of the network to specialize an individual.

1.1.2 Research method

The thesis will be carried out in the form of an action research. In general terms, using this research method, one tries to improve upon a situation whilst learning from it as well [6]. In our case we will be elaborating on our theory of model orchestration and developing the tools to make this possible, whilst trying to solve a specific classification task. The classification task has been defined in collaboration with the host organization and represents a real world need. The task is briefly defined as "classifying images of chemical structures in patent documents" and described in more detail in section 2.4.

1.2 Contributions

In the following, we describe the various contributions of this work.

Classifiers and relationships as social structures we elaborate on an approach for governance or orchestration of a network of machine learning models. We describe the various concepts and entities that are required to enable such an approach. As far as we know, this idea has not been applied within the general field of machine learning.

Novel approach to classifier optimization in exploring our ideas on model orchestration, we are able to come up with novel approaches on how a network of models could be used to perform targeted optimization of a single classifier.

Orchestration tool we provide a (proof-of-concept) tool[1] that encapsulates the various concepts elaborated in the thesis, and provides the associated functionalities. It can be regarded as a reference architecture but also as an actually use-able tool that allows for inferences, validation, exploration and specialization of a network of models.

Classifying images of chemical structures As far as we know, this specific classification task has not been tackled before. As a by-product of this thesis, we will provide a reference architecture and baseline performance for this task.

1.3 Outline

In Chapter 2 we describe the background and related work for this thesis. Chapter 3 defines the building blocks of our system, which are used for our ideas of network orchestration. Next, Chapter 4 we go into the idea of orchestrating a network of machine learning models. We lay out our theories on how such coordination can be specified, we propose and describe a tool that we have developed to perform such orchestration, and finally we explore ways to leverage our approach of network orchestration for classifier specialization and adaptation. Then in Chapter 5 we will describe a number of machine learning models that we have constructed to be able to elaborate and experiment on the concepts and ideas that we propose. In Chapter 6 we start to apply the proposed ideas onto our classification task (described in section 2.4) by running experiments and capturing results such as performance. In Chapter 7 we discuss the main findings, answer our research questions, describe the threats to validity and propose the remaining future work. Finally, we present our concluding remarks in Chapter 8.

[1] Github - model orchestration https://github.com/fatahfattah/model-orchestration

Chapter 2

Background

This chapter will present the necessary background information for this thesis. The following sections contain information that will help in providing a better understanding of the underlying ideas and technologies, *e.g.*, the type of machine learning models, which are described in section 2.1. The goal is to create a common understanding and vocabulary of some of the concepts. We also describe a classification task that represent a real world use case of our host organization; Elsevier. The classification task will be used in later chapters to elaborate ideas and run benchmarks (section 6). Finally we provide an overview of the related work in the various domains that our research touches upon.

2.1 Machine learning

Machine learning (ML) is a sub-field of Artificial Intelligence (AI) and differentiates itself with *learning* being at its core. The general approach is to build mathematical models based on sample data (or "training data") in order to make predictions or classifications without being explicitly programmed to perform the task at hand [7].

Deep learning (DL) is a branch within machine learning that has gained great traction in many research areas and fields of industry, especially in the last decade. Research areas such computer vision, speech recognition and natural language processing have seen some great developments with an additional huge growth in interest in the technologies for many problems in various fields (e.g. self-driving cars and economics). Regular machine learning models, or "shallow" models, consist of three layer, one input layer, one or a few hidden layers and one output layer. These models become "deep" when many hidden layers are used within a neural network [7], for which each of the layers have the ability to identify features from the data, at different levels of abstraction.

Typically machine learning approaches can be categorized as either *supervised*, *unsupervised* or *semi-supervised*. Firstly supervised approaches require the engineer to provide relevant input data to the machine learning model which is labeled with their truth values. This way the machine learning model can identify patterns that are related to the given label and embed this relationship into the network. For computer vision models for example, the data can be an image and the label will be a string that describes what the image is a depiction of (*i.e.*, a car, apple etc.). This is however a costly approach since curating these labeled datasets can be expensive in terms of time and costs.

Unsupervised approaches do not need this labeled dataset and differentiate themselves in that they try to extract meaningful features without human intervention. The features are extracted through algorithms such as data clustering, which leads to categories of the data that are formed by certain patterns and similarities that the members of the category hold.

Finally we have semi-supervised learning that combines before mentioned approaches in that it used a smaller labeled dataset to guide the unsupervised learning on a large unlabeled dataset. It provides an alternative approach when curating an entire dataset may not be possible.

Machine learning involves some type of *model* that has the ability to process all this data and maintain some form of representation of the knowledge that it has learned. One example of this are neural networks which are algorithms that consist of an input layer, one or more hidden layers and an output layer. The

layers are formed by sets of nodes that can have communications with other nodes in the network through weighted edges.

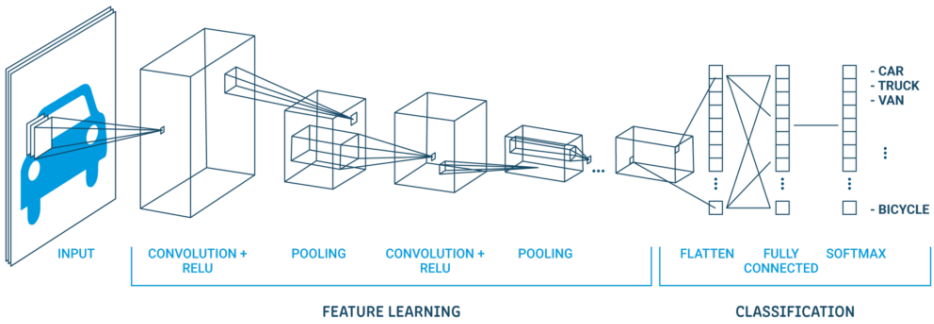

Figure 2.1: Example flow of a convolutional neural network, from `https://www.run.ai/guides/deep-learning-for-computer-vision/deep-convolutional-neural-networks/`

For this thesis we will be looking at a specific category of neural networks that are a specialized in learning features from images as input, called convolutional neural networks (CNN). These are (deep) neural networks with some additional convolutional layers at the front of the network that try to highlight the most important visual features from the input map, as depicted in figure 2.1. This is done by applying a filter over the input to create an activation map that summarize the presence of these important features. This information gets propagated through the network wherein each consecutive layer tries to learn features about the input on a different level of abstraction. After the input has gone through all layers, it will end up in the output layer for which the activated neuron(s) act as the prediction that the network has made.

Our classification task is defined in section 2.4 and describes in greater detail what this input will look like and what our goal is. Now the idea of the thesis is not exclusive to convolutional neural networks and is generic to any model or classifier. However this type of network and classification task do help in that inputs and outputs are very concrete and translatable to natural language when one tries to describe it.

2.2 Symbolic and sub-symbolic integration

Symbolic methods refer to a human-readable or explainable process, for example through formal methods or programming languages. They are usually used for deductive knowledge and are commonly associated with knowledge bases and expert systems [8]. Sub-symbolic methods on the other hand tries to find relations between input and output variables and represent a more abstract or non-human interpretable forms. Example characteristics of symbolic methods could be that they are described in symbols, are applied for reasoning and are human interpretable. Sub-symbolic characteristics are *e.g.*, that they are in the form of numbers, are applied for learning and could be noisy [8].

Throughout time, many efforts have been made by researchers to tackle the problem of integrating symbolic and sub-symbolic information or low-level perception with high-level reasoning and it is considered to become the 3rd wave of improvements in the field of AI [8]. It is also a subject that we will touch upon and explore in this thesis with the ideas of embedding explicit knowledge as relations or rules over a network of machine learning models.

Typically, in the field of machine learning, low-level perception can be carried out by models such as neural networks and high-level reasoning is in the form of logic programming. Examples of this are NeurASP [5] and Deepproblog [4]. These efforts have proved that the integration of low-level perception and high-level reasoning has the ability to improve model performance and reach convergence more quickly. The idea boils down to embedding explicit existing or learned knowledge on top of a neural network. Current literature does not consider having such a layer on top of multiple neural networks and defining relations in relation to other models.

2.3 Answer Set Programming (ASP)

For the classifiers and rules of our social structure (section 3), we leverage the expressiveness and query solving capabilities of Answer Set programming (ASP). ASP is an increasingly popular approach to declarative problem-solving and has its roots in knowledge representation and reasoning. Its success is mainly due to the rich modelling language and effective systems that enable ASP [9]. At its core, the idea of ASP is to represent a given computational problem as a logic program. The solutions to these computational problems correspond to so-called answer sets (or stable models). These are sets of facts in the world that we know or conclude to be true. Finding these answer sets (solutions) is performed by ASP solvers.

Generally the problem-solving process in ASP can be described as: defining a logic program, grounding the program using a grounder and finally generating output by solving the grounded program.

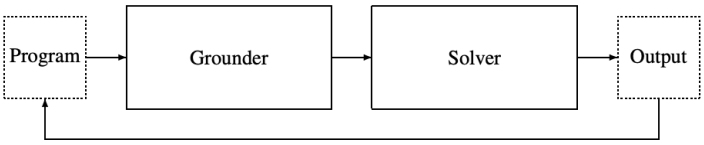

Figure 2.2: Problem solving process in ASP, from [10]

To enable ASP in our system, we make use of the "Potsdam Answer Set Solving Collection" (Potassco) [10, 11]. This is a collection of tools that enable and support in ASP.

Grounding Grounding can be seen as a pre-processing step in which a logic program with all its variables are translated into a logic program with variable-free equivalent terms [12]. The purpose of grounding is to achieve a variable substitution in its input program. Variables in ASP are essentially atoms for which we do not know the value yet, *e.g.*, `p(X).`, which when grounded can look like `p(0). p(1).`. GRINGO [12] is a popular grounder, contained in the Potassco tool set.

Solving Whenever we have a grounded logic program, which is essentially a finite propositional program, we can start searching for all answer sets, or in other words, all solutions to the logic program. These answer sets are the main means of computation in ASP and represent all possible outcomes (terms) of a logic program at a certain step. The CLASP [12] solver is contained in the Potassco tool set and is one of the popular and powerful options available.

CLINGO [10] is another one of the tools in this collection and is an aggregated ASP system that is able to ground and solve ASP programs. Internally it uses GRINGO for grounding logic programs and CLASP to solve these grounded programs. Our proposed system invokes CLINGO to enable ASP and realize various use-cases such as performing an inference.

Every logic program is constructed from terms, which form the rules, for which there are many dialects on the possible notation/syntax of a term. Since GRINGO is the underlying grounder, we are depending on its specific set of supported terms, which are described in figure 2.3.

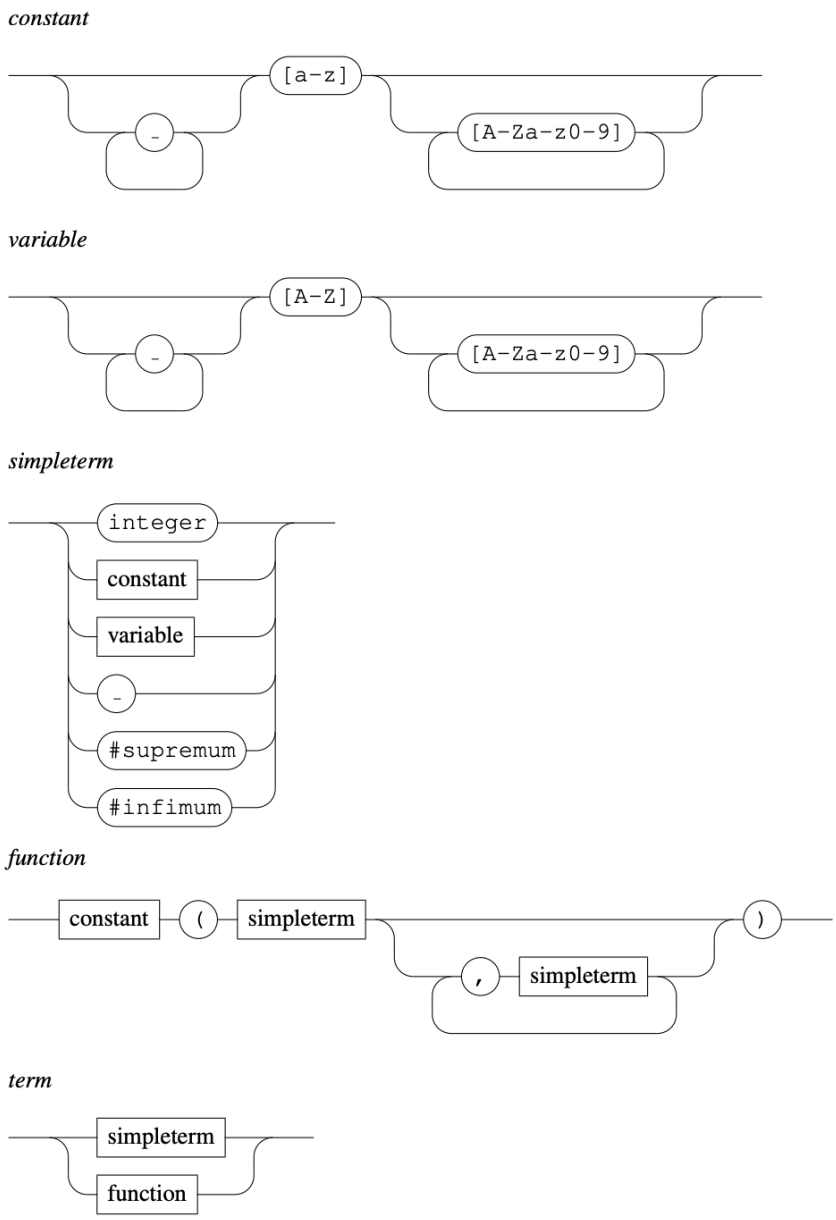

Figure 2.3: GRINGO possible terms, from [13]

Clauses are the constructs that form the logic program, which have a head and body structure:

`<head> :- <body>.`

There are three types of clauses that each behave differently.

Rule: $A_0 : -L_1, ..., L_n.$

The head A_0 of a rule is an atom in the form of a function or constant. Any L_i in the set $L_1, ..., L_n$, is a literal in the body of the form `A` or `not A` wherein the latter corresponds to the default negation of `A`. *e.g.*, a rule `A :- B.` can be read as `A` is in the answer set if `B` is also in the answer set.

Fact: $A_0.$

A fact is a rule with an empty body, for which the associated head always evaluates to `true` and thus appears in all answer sets. *e.g.*, a fact `A.` can be read as `A` is always in the answer set.

Integrity constraints: $: -L_1, ..., L_n.$

Finally we have integrity constraints (or constraints, or goals), which represent an illegal condition. They are essentially actions that remove unwanted answer sets, which are the ones that satisfy all literals in the body of the integrity constraint. *e.g.*, an integrity constraint `:- A.` can be read as `A` can never be in the answer set.

One big difference between ASP and other popular logic programming approaches such as Prolog [14] or Datalog [15], is that Prolog and Datalog implement the notion of positive programs. This means that they do not allow negations or disjunctions in the head of rules, whereas ASP does. This does make ASP more powerful, however it also comes at an increased computational cost [16].

Negation in ASP Within ASP we consider two types of negation that both have slightly different semantics; *classical* (or strong) *negation* and *negation as failure* (or default), represented as (-) and ($\sim$) respectively. In simple terms, classical negation should be used when we are sure that `p` is not the case, and default negation when we are not sure if `p` is the case.

2.4 Classifying chemical structure images in patent documents

Chemical patent documents aim to legally protect an invention in the chemical or pharmaceuticals industry and are the prime source of information regarding new compounds/drugs. The documents are however not written from a scientific perspective but rather as legal documents that try to cover as many aspects of the new information as possible so that they cover all their bases from a legal standpoint. Because of this legality characteristic of the documents, there are many chemical compounds described in the papers that are (often loosely) related to the main compound(s) that the patent is meant to protect [17]. Therefore searching through the text can be a time consuming process in which one will have to read through many related compounds to get to the actual compound(s) that are being patented [17].

The always increasing volume of patent information has become a key challenge for knowledge management systems which try to discover and utilize this information. This has lead to tasks such as patent search and analysis to become more important than ever. Analyzing these patents is important for organisations for numerous purposes such as [18]:

- Determining novelty in patents.
- Analyzing patent trends.
- Forecasting technological developments in a particular domain.
- Extracting the information from patents for identifying the infringements.
- Identifying the promising patents.
- Identifying technological competitors.

Furthermore, the patents are key information sources for commercial chemical substance databases and chemistry literature search databases such as Reaxys [19], which is a solution developed by the host organization Elsevier. Its goal is to make information from chemistry literature available to the field of research so that one does not have to delve through the vast amount of information that is out there.

Scientific literature often uses images to provide information about certain topics. In some cases because words do not suffice and often because visuals are a great way to present large amounts of data in an easily understandable way with a relatively small amount of document space. Images in chemical patents are especially interesting because, if they depict one or more chemical structures, they are often closely related to the main compound(s) or they could even be the key compound itself. Whereas the textual descriptions often contain huge amounts of information that are mainly there for legal reasons [17]. Because of this, images are a key input signal for searching through chemical patents and make the above mentioned purposes easier to realize [17]. See figure 2.4 for an example of such a chemical structure.

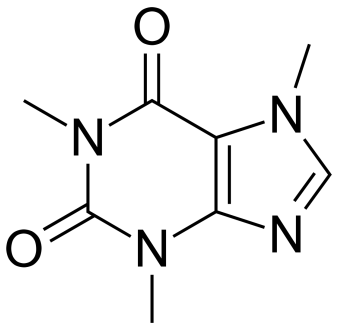

Figure 2.4: Example of a 2d chemical structure (caffeine).

The task of recognizing chemical structures within images is called Optical Chemical Structure Recognition (OCSR) and has historically mainly been tackled using rule-based approaches where the input image is vectorized after which the vectors and nodes are interpreted as bonds and atoms [20]. These solutions are sometimes open sourced (*i.e.*, OSRA [21]) and often closed sourced (*i.e.*, Chemgrapher [22]). More recently successful efforts have been made to tackle OCSR by applying deep learning which has had promising results. Examples of this are Chemgrapher [22], DECIMER [23] and MSE-DUDL [24]. However, these approaches mostly focus on extracting information, given an image for which it is certain that it contains a chemical structure depiction. The step which identifies which images contain chemical structures and which do not, is missing in these solutions, possibly increasing the chances of false positives or irrelevant outputs.

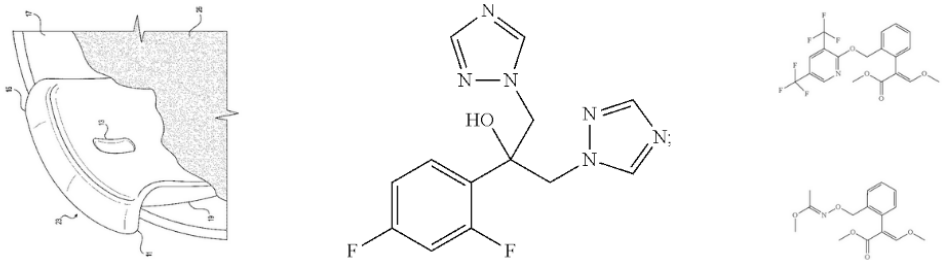

Figure 2.5: Example images containing none, one and many chemical structures

We have defined our classification task as recognizing whether an image contains none, one or many chemical structures. After one knows into which of these classes an image belongs to, they can apply additional information extraction techniques for that specific class of image, *e.g.*, translation to machine readable formats.

2.5 Related work

In this work we explore the idea of coordinating a social structure of classifiers and rules, integrating explicit knowledge over the classifiers, performing inferences and additionally to leverage the benefits of such orchestration. We are unaware of any work that does something similar to ours in those aspects.

Researcher(s)	Title	Description
Kaiser et al.[2]	One model to learn them all	An ensemble-learning approach to orchestration of multiple machine-learning models. Proposed a way to integrate multiple models with different modalities into one big network.
Read et al. [25]	Classifier chains	Review of various techniques to create classifier chains which are binary classifiers in a directed structure.
Evans et al. [26]	Learning explanatory rules from noisy data.	A framework extension of Inductive Logic Programming that tries to tackle the issue of noise or mislabelling in puts.

Table 2.1: Related work on system orchestration

In terms of system coordination, the approaches differ in that they do not consider a decomposition of machine learning models, but rather try to have a tight-coupling of the network. We are also unaware of any efforts to leverage an orchestrated system in order to create specialized versions of classifiers in the network.

Researcher(s)	Title	Description
Yang et al. [5]	NeurASP	On how neural network can learn from explicit semantic constraints, expressed by rules. The approach involves integration Answer Set Programming into neural networks.
Manhaeve et al. [4]	DeepProbLog	A probablistic logic programming language that incorporates deep learning through neural predicates.
Garcez et al. [27]	Neural-symbolic computing	A survey of recent advances in neural-symbolic computing as a method to integrate machine learning and reasoning.
Rueden et al. [28]	Informed machine learning	A review of various approaches in the field on integrating prior knowledge into the training process.

Table 2.2: Related work on integrating symbolic and sub-symbolic information

Regarding the integration of symbolic and sub-symbolic information, there are some recent and very promising works. These works differ in that they do not consider a network of models, but rather tackle the issue on the single neural network level.

Researcher(s)	Title	Description
Rajan et al. [23]	DECIMER: towards deep learning for chemical image recognition	Deep learning approach to translate images to SMILES representation.
Goh et al. [29]	Chemception	CNN that tries to predict several features from images of chemical structures.
Staket et al. [24]	MSE-DUDL	An image2sec (SMILES) approach for chemical structures in patent documents.
Fillipov et al. [21]	Optical Structure Recognition Software (OSRA)	Rules based approach to extracting chemical information from chemical images.
Oldenhof et al. [22]	ChemGrapher	Deep learning approach to extracting atom location, bonds and charges in chemical images, which outputs a graph translation.

Table 2.3: Related work on chemical image recognition

Furthermore, we also tackle a classification task related to images in chemical patent documents, which also has ongoing research efforts for. As far as we know, there has been no efforts to tackle the exact classification task that we consider, which is described in section 2.4. More importantly, our classification task seems to be an important step in enabling many of the cited papers, if one wants to apply them for a real world situation. The reason for this being that the papers make the assumption that the image they process, already contains a chemical structure, which in practice could be an image of many things, in the context of chemical patent documents.

Chapter 3

Main concepts

This chapter lays out the main concepts and definitions, providing the building blocks of our system, namely, seeing classifiers and rules as components of social structures. This will be helpful for presenting how classifiers can be orchestrated and coordination be enabled.

3.1 Social structures

Porpora [30] describes four common conceptions of what social structures are: patterns of aggregate behaviour, law-like regularities that govern behaviour, systems of human behaviour based on social positions, and collective rules and resources that structure behaviour. The last description fits the context of our research, we consider a social structure as consisting of a set of classifiers, and a set of rules specifying the mutual relationships of those classifiers.[1] This view is commonly associated with Anthony Giddens [31], describing *structure* as rules and resources and *social relationships* as abstraction from repetitive or routinized behaviour.

The characteristic of a "social" collective arises when there is some form of interaction between the individual members of that collective. In our case, a group of classifiers becomes "social" in the sense that there are means of communication or collaboration between the classifiers which allows for collective or consensus-based decision-making.

The decision-making is a collective effort in that outputs from multiple classifiers can be inquired to conclude a target outcome. *e.g.*, when we have a classifier that categorizes the weather in an image and one to determine the type of animal depicted, to conclude whether we have a picture of a cat in the rain.

It can also be consensus-based when we need the agreement/disagreement of multiple classifiers to be able to reason and conclude that a certain outcome is true. *e.g.*, when we need two classifiers to conclude that an input depicts an image of a cat.

3.2 Rules

Rules generate or reinforce the systemic patterns that can be seen within a social structure. In the context of this research, rules are a way to specify relationships between two or more machine learning models concerning their outcomes or aggregation of their outcomes. These relationships map to patterns of behaviour that we want the social system to reproduce and typically reify some type of (explicit) knowledge that we want to incorporate into the (essentially inferential) activity of the social structure. These rules typically add or exclude some information from the overall knowledge base in our system for a collective inferential process.

Suppose we have two classifiers `A` and `B`, which can tell us if facts `a` and `b` are true or false. Let us say that `b` can only be true if `a` is true as well. This means that there is some relationship between the two

[1] Note that we will not be using this definition very strictly in the sense that our proposed system can have characteristics which are more in line with other definitions of a social structure, *i.e.*, introducing a hierarchical organization based on how important the individual contributions of our classifiers are. The given definition rather acts as a tool that allows us to think about and reason on our idea of a social structure in this machine learning context.

classifiers expressed as a logical dependency: `b` is true if `B` tells us it is true AND `a` is true.

More concretely when we want to perform inferences using our machine learning models, it simply means that we apply the rules over the outcomes of our classifiers to come to a logical conclusion.

In the coming chapters, we will discuss how we can leverage these rules for coordination, and how the resulting collective can be restructured by simply modifying the rules at the social level.

3.3 Classifiers as agents

The concept of computational agents has existed for a long time. In 1977 Carl Hewitt [32] introduced a model of computation based on actors and described it as a self-contained and interactive object. Before and after this, however, there have been numerous other definitions for agents with varying views on the concept. Gokulan et al. [1] describes that there is no consensus of the definition because of the universality of the word, the concept is implemented in varying forms (e.g. robots, computer networks) and lastly because the word is not possible to generalize since there are so many application domains. A view on agents from Woolridge et al. [33] is that "agents are a hardware or (more often) software-based computer system that must employ properties such as autonomy, social ability, reactivity and pro-activeness".

When we consider machine learning classifiers as independent computational units, we can easily see how they can be regarded as agents as well; They have some specified computational goal for which they receive external input and provide some output(s). Regarding them as agents, allows us to think about how we can structure them in a network of classifiers and coordinate this network of classifiers to achieve some common goal, which is one of the main topics of this project. When we enable communication between the individual classifiers, and we define rules over their relationships, the network of classifiers becomes a social network in which there is some defined social structure thanks to the structuring property of the rules. In such a network of agents, each agent has the ability to perform their own processing (sub-)task on the given input(s), and share the output with other agents in the network. This essentially leads to a potentially available pool of knowledge (or knowledge base) which we can inquire to form a conclusion based on given, perceived or deducted facts. This architectural template allows for a collaborative inference to come to a conclusion with respect to complex classification tasks.

In the end however, the classifiers that we consider are still a rather simple type of agents. Once they are trained, they become deterministic and are not adaptive (*e.g.*, to their environment). In principle, however, more complex computational units could be considered as well with a similar approach.

Chapter 4

Orchestration

In our framework, classifiers do not have adaptive abilities on their own since they are essentially *just* computational units that can take some input and produce some output. The orchestrator functions as a sort of "manager" that organizes agents to perform a certain (inferential) task. It does this by gathering all outputs from its classifiers and insert them into the dataflow that is defined by the rules over these outputs. Another view is that the orchestrator sets the communication channels by which classifiers can spread information.

We consider classifiers as individual computational units, or agents, for which we define interfaces, protocols and wrappers so that we are able to communicate with them and enable governance (orchestration) over them. This means that we can for example take off-the-shelf classifiers and simply structure their outputs to solve some classification task, rather than having to integrate them into each other.

This chapter introduces the syntax for the social structures, classifiers and rules of our system, together with a number of additional use-cases for our idea of orchestration. Furthermore, we describe the technical system overview, elaborating on the various entities of the system and their interaction with each other. This section also describes how we have encapsulated a subset of the Answer Set Programming (ASP) (section 2.3) concepts so that our system can work with them. Then we describe how the orchestration system can be used to perform inferences, and we introduce the concept of specialization of a classifier.

4.0.1 Classifier syntax

Using a classifier in our ASP program starts with defining it as a component in the program, by means of an identifier and a set of possible outputs. This definition rule should be in the form of:

```
#external classifier(a;b;c).
```

The previous rule is prefixed with the `#external` directive, which allows us to dynamically (at run-time) ground the output of the actual classifier into the ASP rule, essentially setting its truth value with the output given by the classifier. Next, we give the list of possible output classes of the classifier, separated by a semicolon. In the example above, this means the `classifier` agent can conclude `a`, `b` or `c`. Note that `classifier(a;b;c).` is simply syntactic sugar for `classifier(a). classifier(b). classifier(c).`.

After we have defined the classifier within the program, we can use it as a literal in other rules like:

```
x :- classifier(a).
```

This means that `x` is added to our answer set, if the output of `classifier` equals `a`.

4.0.2 Expressing social structures as ASP programs

We have seen that in ASP we can conclude outcomes by applying explicit rules over facts in the world. We can apply this idea to our machine learning classifiers, by inferring the truth values of these facts from the outcomes of our classifiers. *i.e.*, we could have a classifier `p1` that predicts the truth value of `p`, which we would then insert into the ASP program. So a simple ASP program to conclude either `p` or `-p` could look like:

```
#external p1(p;-p).
p :- p1(p).
-p :- not p.
```

The clauses that refer to a classifier in the body require us to compute the inference outcomes of the classifier beforehand.

Enabling interaction between classifiers in this framework simply means to define rules over the outputs of the classifiers in our network of classifiers. Let us take an example in which we have two classifiers that together try to compute `p` or `-p`, we can write it as:

```
#external p1(p;-p).
#external p2(p;-p).
p :- p1(p), p2(p).
-p :- not p.
```

This can be read as, `p` is in our answer set if both `p1(p)` and `p2(p)` are in our answer set as well. Finally, through default negation, we conclude `-p` if `p` does not appear in our answer set.

Using rules allows us for testing several views on a classification task, adding controlled inferences. For instance, the previous example can be seen as holding a pessimistic view on `p` and requires a consensus from both classifiers over `p`. We can instead apply an optimistic view by simply changing the clauses so that we can conclude `p` from the output of just one of the classifiers:

```
#external p1(p;-p).
#external p2(p;-p).
p :- p1(p).
p :- p2(p).
-p :- not p.
```

We can expand upon this by defining rules for outcomes that are logical conclusions from aggregations of earlier observations. Let us take an example where we add additional rules with regard to `p`, so that we can conclude `q`:

```
#external p1(p;-p).
#external p2(p;-p).
p :- p1(p), p2(p).
-p :- not p.
q :- -p.
```

As we can see from these examples, this framework opens wide opportunities in terms of what type of relations we can define and how we can incorporate explicit relations to define rules over our classifiers.

4.0.3 Embedding explicit knowledge

Incorporating explicit knowledge, or integrating low-level perception with high-level reasoning, is one of the big challenges in the general field of AI and ML [4, 5, 8, 27]. The computational social structure suggested here, consisting of rules (a form of explicit knowledge) and classifiers (a form of perception-to-symbol mechanisms), could be a way to tackle this issue.

Let us take an example in which we try to classify whether an image contains an `apple` or a `carrot`. Suppose that we have a classifier `color_classifier` that has the ability to classify the colour of the most prominent object in an image. From our fruit-domain knowledge we know that apples are either red, green, yellow or a mixture. We also know that carrots are orange. We can now define the following ASP program from this domain knowledge, which we translate into rules.

```
#external color_classifier(red;yellow;green;orange).

apple :- color_classifier(red).
apple :- color_classifier(yellow).
apple :- color_classifier(green).

carrot :- color_classifier(orange).
```

We have now created a new classifier from an existing one, simply by defining explicit rules over the outputs of our classifier. These rules are defined by reasoning beforehand over the outputs of each of our classifiers in relation to the desired outcomes. This approach also allows us to answer, to some extent, why a certain fruit has been classified under a certain label.

4.0.4 Explainability and error debugging

Indeed, often we have situations in which we have a machine learning model that has the ability to perform a classification task with high accuracy, for which we do not have any idea about the internal workings of how the predictions come to be. Why is an input classified as `x` and not as `y`, and what aspects contributed to this prediction? In a sense, machine learning models are black boxes because of this and it is the core of the explainable AI (or XAI) problem [34]. With the ability of high-level reasoning through our defined rules, we come a little closer to being able to reason about why our system provides a certain output.

Suppose we take the same example from the previous section:

```
#external color_classifier(red;yellow;green;orange).

apple :- color_classifier(red).
apple :- color_classifier(yellow).
apple :- color_classifier(green).

carrot :- color_classifier(orange).
```

This knowledge base is sound from a logical point of view, apples are mostly red, yellow or green and carrots are mostly orange. Let us say that we run inferences through this social structure and see that we often incorrectly classify carrots as apples. We are then able to look at all the outputs from the ASP solver to see that carrot images are also often associated with the color `green` by our classifier, presumably this is because they have a green stem sticking out at the top. To recognize the error is trivial in this example, but we can imagine situations where we use many more classifiers, and that each of them adds a piece of information along the way. In these cases it becomes increasingly difficult to detect what piece of information led to a wrong conclusion, but still, the proposed tool could be useful as a debugging instrument.

4.0.5 Re-purposing social structures

One of the major strength of the proposed approach is that now a network of classifiers can be re-purposed towards a new classification task or add a new outcome to a current task, by simply changing the rules over their outputs.

Suppose we have the following social structure to classify whether an input image depicts a cucumber or a carrot by simply assuming that a green object is a cucumber and an orange object is a carrot:

```
#external color_classifier(green;orange;red;blue;purple).
cucumber :- color_classifier(green).
carrot :- color_classifier(orange).
```

We can now re-purpose this social structure to also detect tomatoes, eggplants and blueberries by simply adding new rules:

```
#external color_classifier(green;orange;red;blue;purple).
cucumber :- color_classifier(green).
carrot :- color_classifier(orange).
tomato :- color_classifier(red).
blueberry :- color_classifier(blue).
eggplant :- color_classifier(purple).
```

Adaptation at the social level therefore simply means changing the rules over our classifiers. As opposed to other methods such as ensemble-learning [2], here we do not need to retrain any of the classifiers in our network.

4.0.6 Grounding and solving in our system

CLINGO is our interface to the GRINGO grounder and Clasp solver (section 2.3). We interface with the CLINGO's API to perform interfaces and solve our logic programs, which are in the form of social structures. A pre-processing step before we can interface with CLINGO is that we perform inferences for all classifiers in the social structure, and are then able to determine which facts in our ASP program are true.

The following code snippet roughly describes how CLINGO is used in the orchestrator:

```
# We perform inferences for all classifiers in our social structure
inferences = self.social_structure.infer(inputs_dict)

# Instantiate a Clingo solver and solve it given our parsed program
clingo_control = clingo.Control([])
clingo_control.add("base", [], parsed_program)
clingo_control.ground([("base", [])])

# We induct the inference values into the #external atoms defined in the program, by
    setting their truth value
for classifier_name, inference in inferences.items():
    clingo_control.assign_external(clingo.Function(f"{classifier_name}", [clingo.
        Function(inference)]), True)

# Solve our grounded ASP program
clingo_control.solve(on_model=self.on_model)
```

The process involves retrieving the inferences of our classifiers, grounding the ASP program, assigning the outputs of the classifiers, and finally solving the program. Eventually, the remaining answer sets will be returned to the given *on_model* callback.

4.1 System overview

This section gives a technical overview of the proposed tool. The various concepts such as social structures, classifiers, rules and orchestration are represented in the system for which we will describe their relationships and characteristics. Figure 4.1 gives a global overview.

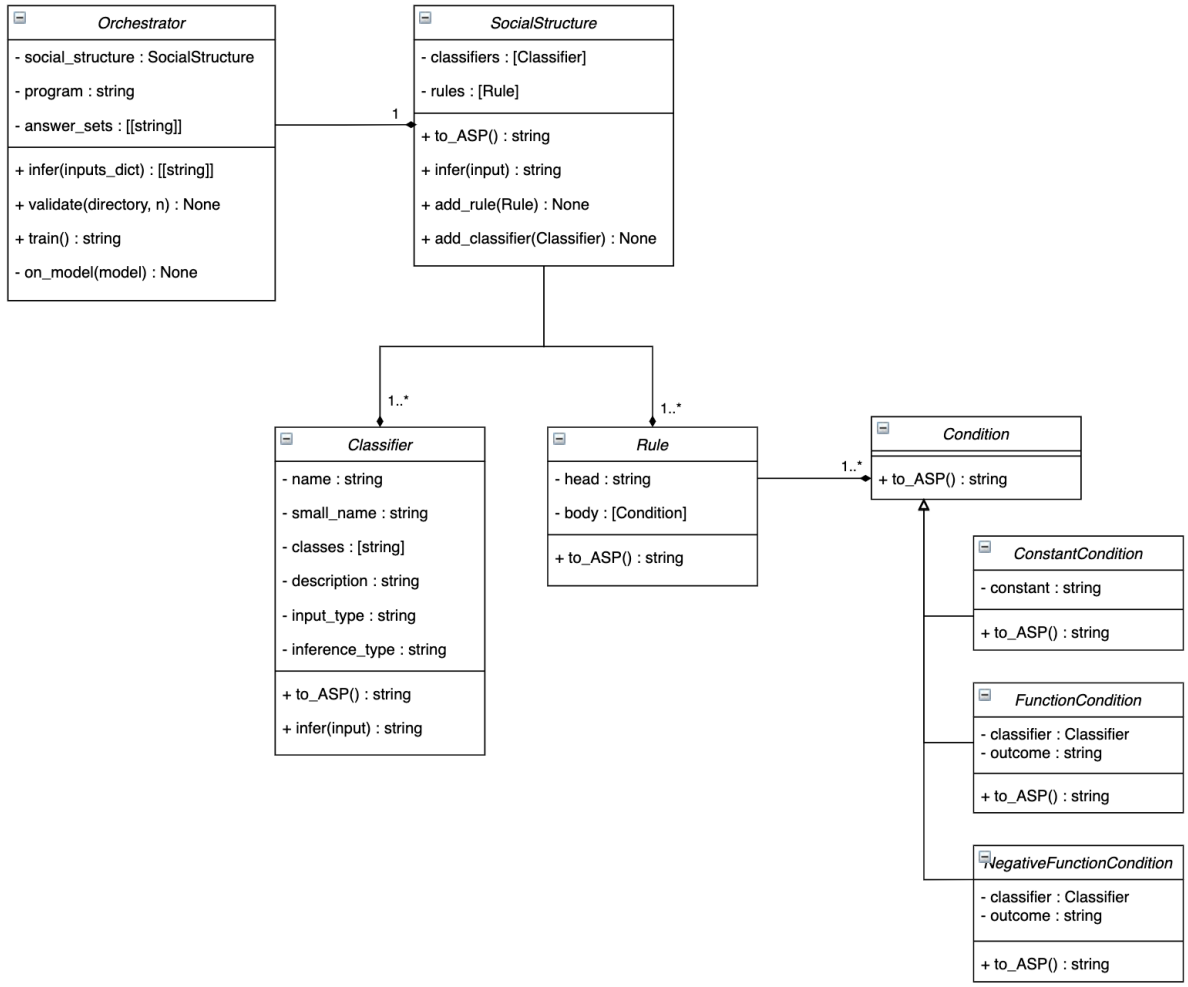

Figure 4.1: Orchestrator system overview

The orchestrator contains a single social structure, which it translates to an equivalent ASP program upon initialization. It also keeps track of all previously inferred answer_sets. Its main actions are performing inferences, validating performance over a given dataset and retraining a specialized version of a classifier.

The proposed system is able to perform its various use-cases without the need to use *all* capabilities of the underlying ASP system (CLINGO) (section 2.3). For this reason we have decided to only translate a subset of the total capabilities in CLINGO, into abstractions and interfaces. The goal of this translation is that, instead of writing ASP-programs, the end-user needs to work with object-oriented abstractions of ASP concepts, with the goal that is is more straight-forward and intuitive to work with.

To give an idea of the mentioned translation we consider the following example. Suppose we have an apple classifier `a1`, that tries to classify `apple` or `-apple`. We write the following program code:

```
Rule("apple", [FunctionCondition(a1, "apple")])
Rule("-apple", [FunctionCondition(a1, "-apple")])
```

Which corresponds to the following ASP program:

```
#external a1(apple;-apple).

apple :- a1(apple).
-apple :- a1(-apple).
```

4.1.1 Social structure class

The social structure class is the computational embodiment of the social structure concept that we have described in chapter 3. Its general task is to represent a social structure and computationally retrieve the outputs from the classifiers so that it becomes available to the orchestrator.

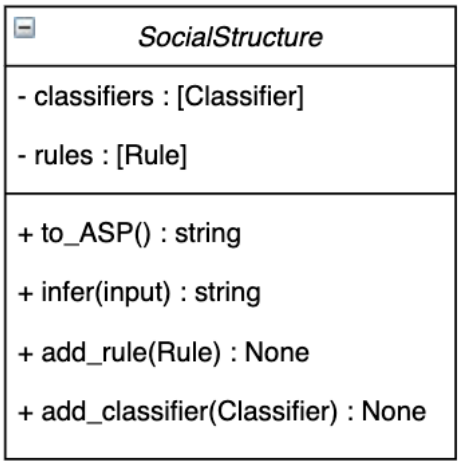

Figure 4.2: Social structure class diagram

Figure 4.2 describes the class of a social structure and how we can interface with it.

- **classifiers**: are the list of classifiers in the social structure.
- **rules**: are the rules that represent the structure of the social structure.
- **to_ASP()**: function to translate the social structure into an equivalent ASP program.
- **infer()**: function to perform an inference through the social structure.
- **add_classifier()**: add a classifier to the social structure.
- **add_rule()**: add a rule to the social structure.

4.1.2 Classifier class

In order to successfully interact with our classifiers, we have defined a class that acts as a sort of blueprint (or interface in object oriented programming), which each classifier in the system needs to inherit. The goal here is that we ensure certain functionality and presence of meta-data so that we can be certain that an object that we interact with has the expected capabilities of a classifier.

One example that we can consider is that if we want to perform inferences with classifiers, we need to ensure that they have a function, *infer()*, which we can pass an input to and retrieve an output from.

Furthermore, a classifier object can be extended with variables and functionalities that are only applicable to that classifier. Think for instance of initializing a model architecture, loading pre-trained weights if present or initializing a pre-processing transformation which can then be used when triggering an inference process.

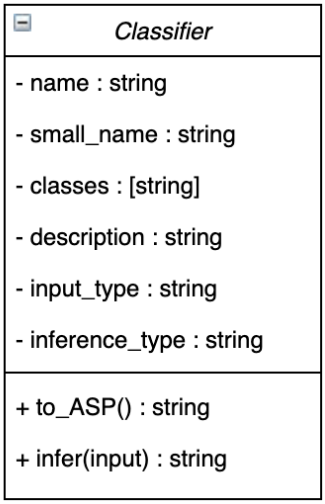

Figure 4.3: Classifier class diagram

The class core template is illustrated in figure 4.3. Down below is a brief description of each of the fields in the class:

- **name**: a descriptive name of the classifier, *e.g.*, "Color classifier"
- **small_name**: an abbreviated name of the classifier, *e.g.*, "cc"
- **classes**: a list of classes which the classifier can output. This is important in case an invoking system needs to know what possible classes it can expect.
- **description**: a natural language description of the classifier.
- **input_type**: the type of input the classifier classifies, *e.g.*, image, text, audio.
- **inference_type**: the type of classification the classifier performs, *e.g.*, classification, regression.

4.1.3 Rules and conditions classes

Figure 4.4 depicts the class diagram of rules and conditions in the proposed system. The depicted classes represent the subset of ASP functionality that we mentioned in the beginning of the chapter.

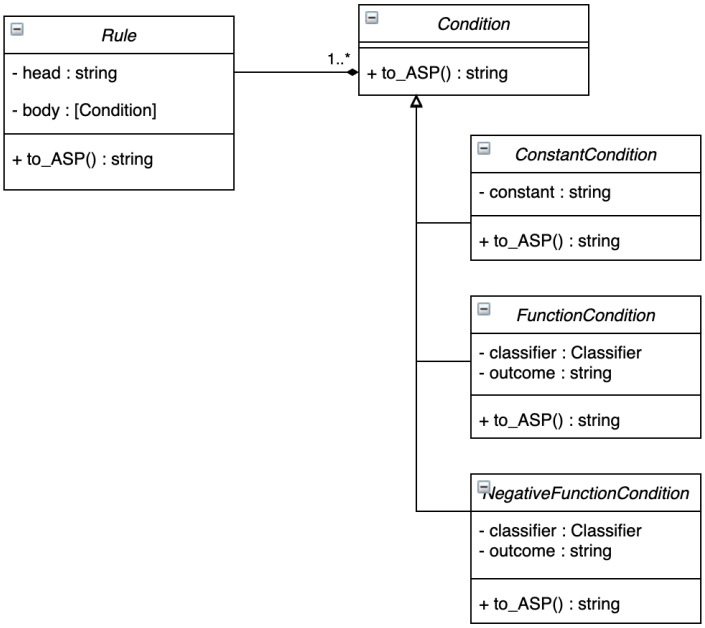

Figure 4.4: Rule and conditions class diagram

Each class has an `to_asp()` function which translates the instance of the class into its logically equivalent ASP atom. The various fields in the classes represent the literals of their corresponding ASP objects.

4.2 Preparation steps

To enable our system to properly orchestrate a network of machine learning classifiers, we assume that a couple of steps are executed beforehand. Figure 4.5 gives a visual overview of this process.

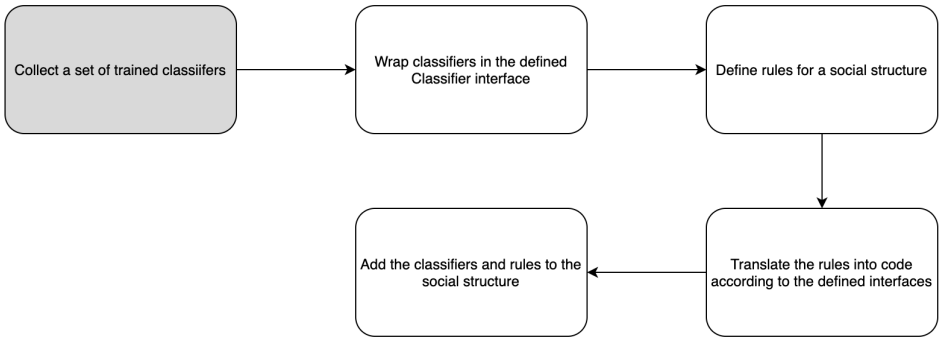

Figure 4.5: Preparation steps: overview

Initially we assume a set of already trained classifiers are collected. This step does not need to have any interaction with the proposed system and can be done with any tool of choice.

Next the classifiers need to be wrapped in the interfaces as defined in section 4.1.2. Implementations for the given functionality should be given as well so that the wrapped classifier actually produces an output. If this is not implemented, the classifier will currently simply return an empty result.

Then we need to think about the rules over our classifiers, and what kind of explicit knowledge can we think of that we want to incorporate in our social structure. Types of rules are discussed more in depth in section 3.2.

The defined rules then need to be translated into code through the defined interfaces as described in section 3.2. This step is a translation of the rules from *i.e.*, a natural language into a format that our system can understand.

Finally we need to add our wrapped classifiers and rules to our social structure through the defined API's for this task. This does not need to be done in any particular order.

After our orchestrator receives a social structure consisting of classifiers and rules, it performs an internal step wherein it translate the social structure into a logically equivalent ASP program. Suppose we have the following social structure initialization code:

```
social_structure = SocialStructure()
animals = animal_NN()

social_structure.add_classifier(animals)

social_structure.add_rule(Rule("catimage", [PositiveCondition(animals, "cat")]))
social_structure.add_rule(Rule("dogimage", [PositiveCondition(animals, "dog")]))
```

Which will get translated into the following ASP program:

```
#external animals(cat;dog).
catimage :- animals(cat).
dogimage :- animals(dog).
```

4.3 Inferences

Making an inference means that we run a data input through the social structure of classifiers and rules to form some conclusion as to the contents or class of the input. From the orchestrator's point-of-view, this involves retrieving outputs of the classifiers in the network, inserting these outputs into the ASP program and finally computing the ASP program to retrieve the answer sets. Figure 4.6 depicts the runtime flow of how an inference is performed.

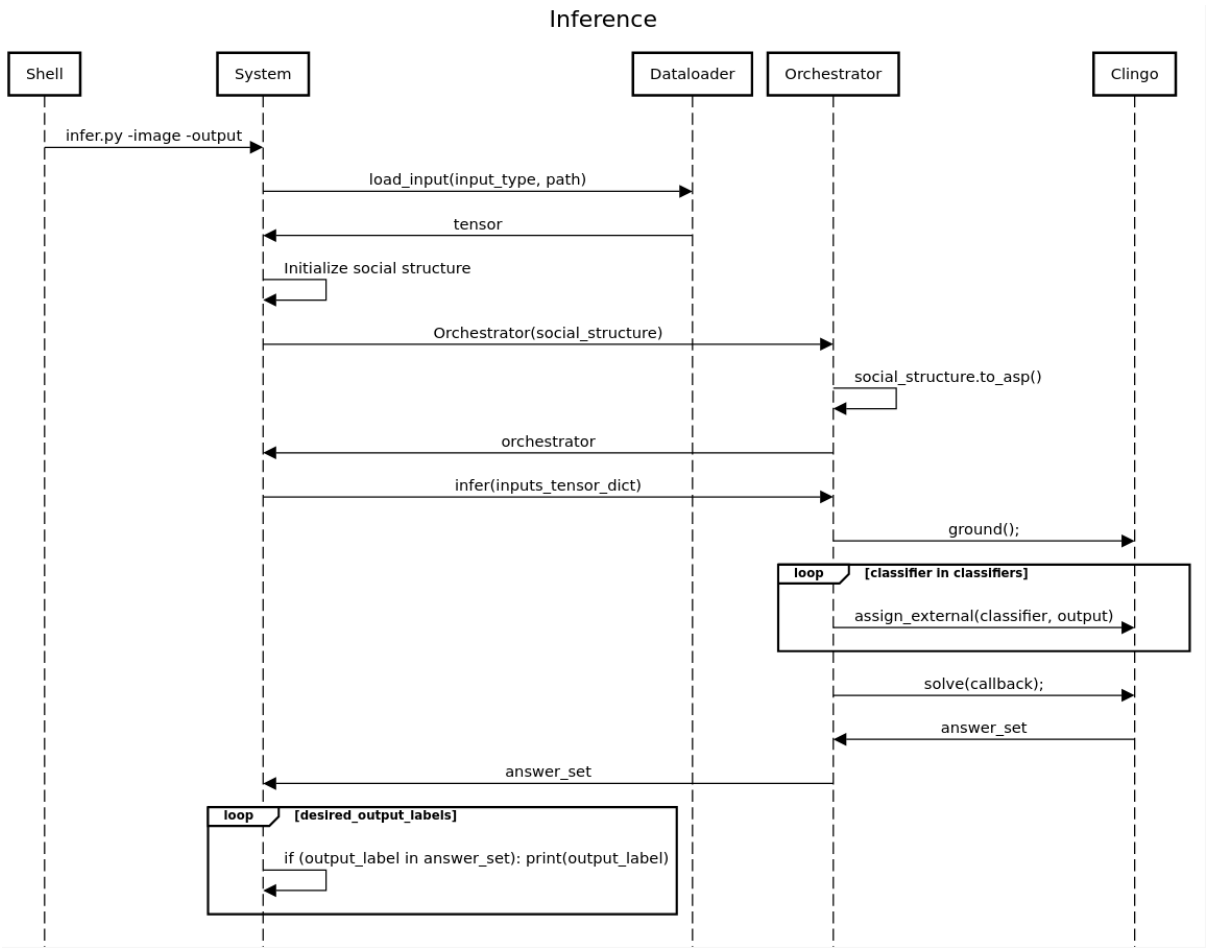

Figure 4.6: Orchestrator inference sequence diagram

The process starts with some external trigger, this could be a shell command, that tells the system that we want to perform inference on a given input. The system initializes the social structure, orchestrator, loads the input (generating a dictionary of loaded inputs, or tensors) and finally triggers the orchestrator to perform an inference.

Next, the orchestrator starts with running inferences from all classifiers in the network on the given input. It then grounds these inputs in the ASP program attached to the orchestrator with the use of the *assign_external()* function call. Finally, it computes the ASP program with the *solve()* call and then returns the answer set back to the system. Depending on whether the system is looking for specific output values, it will print the entire answer set or only the desired ones that are also included in the answer set.

4.4 Specialization

In human groupings (as well to some extent in other forms of life), collaboration typically allows actors with varying skills, experiences and knowledge to perform activities more effectively. In industry, it has many purposes such as developing new products, improving existing ones or coordinating projects. One

important characteristic of collaboration in the context of our research is that it allows an actor to inquire knowledge that they did not possess prior to collaborating. *e.g.*, a blind person might not see an object obstructing his path, however, by collaborating with his trusty guide dog, he can now safely navigate around it. When we think of the two living actors as an AI system, the object detection accuracy of the blind person has greatly increased thanks to their collaboration with another actor.

In this section, we will explore how we can leverage the idea of orchestrating social structures of classifiers and rules, so that we can do targeted improvements of a single classifier. The idea is that we can perform a meta-analysis on the target classifier to find contextual situations in which the target classifier produces true positives (*i.e.*, when performing its task correctly) and false positives (*i.e.*, when it is not performing its task correctly). By using these contextual situations as data filters, we can then train two new independent classifiers that specialize on datapoints that we are already able to classify correctly (true positives) and for those that we have a harder time with (false positives). The intuition is that, by specializing training in these two seemingly different contexts, the classifiers performance would improve (assuming the number of datapoints is sufficient). Finally, we can form a joint decision structure for inferences, in which we use the ancillary classifiers to capture the contextual situation, so that we can use the output of the specialized classifier that is trained for that specific contextual situation.

4.4.1 Meta-analysis of contextual situations

The first step in the specialization is a meta-analysis in which we capture all possible contextual situations of our ancillary classifier with regard to the true positive and false positive outcomes of our target classifier. Contextual situation here simply means any combination of an ancillary output w.r.t. the outputs of the target classifier. For each output class of our target classifier, we want to know what outputs are triggered from our ancillary classifiers and in what rate. Next, we try to extract the most significant contextual situations that are a good predictor for either true positive or false positive inferences.

4.4.2 Capturing contextual situations

Capturing these contextual situations essentially can be seen as capturing the number of co-occurrences of ancillary classifier outputs with regard to the target classifier outputs. We simply do this by running inferences of the target classifier M^{target} and capturing the outcomes of an ancillary classifier $M^{ancillary}$. The output of this step then is a contingency matrix in $\mathbb{R}^{n \times m}$ where n is the number of positive (true or false) outputs from M^{target} and m is the number of output classes from $M^{ancillary}$. An example of how this looks can be seen in table 4.1 for a binary classifier as ancillary module.

	$M^{ancillary}(B)$	$M^{ancillary}(\sim B)$
(true positive) $M^{target}(A)$	p00	p01
(false positive) $M^{target}(A)$	p10	p11

Table 4.1: Example contingency table

Here $p00$ represents the ratio, in the range of $\{0, 1\}$, in which $M^{ancillary}(B)$ is true whenever M^{target} makes a correct A inference. In this, a high ratio would mean that the two outputs are strongly associated.

4.4.3 Ranking significance of contextual situations

After we have captured the contextual situations, as described in section 4.4.2, we need to determine which of the contextual situations we should use as our filters. For this, we propose several heuristics in which we can rank the significance of the contextual situations $x \in R$.

To define the heuristics we have access to several values from the captured relations of the classes in $M^{ancillary}$ over the output classes of M^{target}. We can infer these values from the contingency table (section 4.4.2).

First of all we know, for each class of $M^{ancillary}$, the number of co-occurrences with either true positive or true negative classification, which we will name $correct\_n$ and $wrong\_n$. Consider the following table

as a running example, which represents the *correct_n* and *wrong_n* for an ancillary classifier M with classes `p, q, r`:

Ancillary output	correct_n	wrong_n
$M(p)$	110	5
$M(q)$	100	10
$M(r)$	90	20

Table 4.2: Running example values for ancillary classifier M, for an output class in M^{target}

Next we have *correct_ratio* and *wrong_ratio* which are the ratios of the *correct_n* and *wrong_n* relative to the sum of all *correct_n* and *wrong_n* for a specific output class from M^{target}.

Finally we can rank the co-occurrences according to the value of their *correct_ratio* and *wrong_ratio* from 1...n (rank 1 having the highest ratio and rank n the lowest), generating *correct_rank* and *wrong_rank* respectively. In case of equal ratios we simply consider the order in the ranking so that we can have a conclusive ranking. When adding these values to our running example, we get the following table.

Ancillary output	correct_n	wrong_n	correct_ratio	wrong_ratio	correct_rank	wrong_rank
M(p)	110	5	0.37	0.14	1	3
M(q)	100	10	0.33	0.29	2	2
M(r)	90	20	0.30	0.57	3	1

Table 4.3: Example values for ancillary classifier M, after inferring the ratios and rankings

Now that we have described the various variables that are available to us, we define the following heuristics[1]. The first option is to look at the number of related false positives, and rank according to that, so that the highest number of false positives would be ranked as first:

$$r_1 = \text{argmin}(\{x_{wrong\_rank} \mid x \in R\}) \tag{4.1}$$

The second option is to look at the rate in which the ancillary output co-occurs with true positives and false positives, compared to the rates of all other ancillary outputs.

$$r_2 = \text{argmax}(\{x_{wrong\_ratio} - x_{correct\_ratio} \mid x \in R\}) \tag{4.2}$$

The third option is to look amongst all captured relations and optimize for the one with the highest *wrong_rank* and lowest *correct_rank*. This however can end up with situations in which there is no conclusive optimal relation. Suppose there are more than one relation which have an equal `wrong_rank` and `correct_rank`, *i.e.*, `1 - 1`, we would end up with multiple relations as rank `0`.

$$r_3 = \text{argmin}(\{x_{wrong\_rank} - x_{correct\_rank} \mid x \in R\}) \tag{4.3}$$

Another options are the normalized versions of the previous two formulas, in which we also take the number of instances into account:

$$r_4 = \text{argmax}(\{((x_{wrong\_ratio} \times x_{wrong\_n}) - (x_{correct\_ratio} \times x_{correct\_n})) \,/\, (x_{wrong\_n} + x_{correct\_n}) \mid x \in R\}) \tag{4.4}$$

$$r_5 = \text{argmin}(\{((x_{wrong\_rank} \,/\, x_{correct\_rank}) \times (x_{wrong\_n} + x_{correct\_n})) \,/\, x_{wrong\_n} \mid x \in R\}) \tag{4.5}$$

[1] The heuristics proposed here are a first effort in trying to capture the significance of the explored contextual situations. Therefore a side-note should be made that they require further investigation for further refinement.

4.4.4 Tool: generating contingency matrix and rankings

Independently of the specific heuristics, the generation of a contingency matrix is an automated process for the tool we propose. It requires picking a target classifier and one or more ancillary classifiers that will be used to capture the relations over. Next we will need a dataset over which we will run inferences to capture the co-occurrences and build the contingency matrix. In principle, this dataset should consist of unseen datapoints to simulate more accurate real world performance.

Suppose we have M^{p1} as our target classifier with `p, -p` classification classes. We then have an ancillary classifier M^{q1} with `q, -q` classification classes. In code we initialize the target classifier and ancillary classifiers, after which we run the following shell command:

```
python -m explore --root_dir={path/to/exploration/dataset}
```

This will run inferences from all classifiers over the dataset and output a contingency matrix that could look like:

	$M^{q1}(q)$	$M^{q1}(-q)$
(true positive) p	p00	p01
(false positive) p	p10	p11
(true positive) -p	p20	p21
(false positive) -p	p30	p31

Table 4.4: Example contingency matrix output

In this example, `p00` represents the ratio in which M^{q1} predicted `q` whenever our target classifier M^{p1} correctly predicted `p`.

Besides the contingency matrix, we also receive the corresponding rankings which are defined in section 4.4.3, for each of the target classifier classes. In our example, we would get a rankings table such as the one below, for both `p` and `-p`.

Classifier	correct_n	wrong_n	correct_rank	r_1	r_2	r_3	r_4	r_5
$M^{q1}(q)$	p00	p01	p02	p03	p04	p05	p06	p07
$M^{q1}(-q)$	p10	p11	p12	p13	p14	p15	p16	p17

Table 4.5: Example ranking table output

Finally, we are able to extract the optimal filters, for each classification class, following the given ranking rules.

4.5 Training specialized classifiers

As a result of the previous meta-analysis, we obtain a filter that has the ability to determine whether an input falls into a discovered contextual situation, or in other words, whether an input falls into the category of previously correctly- (TP) or incorrectly (FP) classified datapoints.

To make this more clear, consider a situation in which we have a target classifier `P` and ancillary classifier `Q`. If we observe that `P` is bad at correctly classifying `P(a)` when `Q(a)` is true and better at classifying `P(a)` when `Q(b)` is true, we know that we can rely more on `P(a)` whenever `Q(b)` is true. We can then use `Q` as a sort of test condition (filter) to determine we are in contextual situation `Q(a)` or `Q(b)`, and take measures accordingly, *e.g.*, taking the output from a classifier that is specialized in either one of the contextual situations.

The degree by which the filter can make this prediction relies greatly upon the strength of the relations that were found during the meta-analysis. However, assuming that we have a filter that has some non-zero accuracy performance in making this true positive and false positive distinction, we can use it to filter our original dataset into these two categories. Once we have these two data subsets, we can use them to train two new classifiers, one for the true positive datapoints S^{TP} and one for the false positive datapoints S^{FP}.

Figure 4.7 visualizes the process of training specialized classifiers from a target dataset, ancillary classifiers and a defined filter.

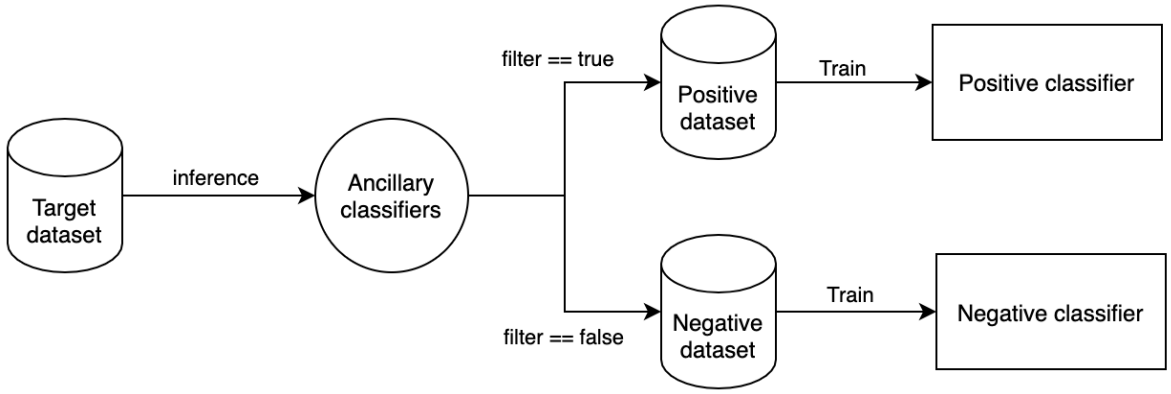

Figure 4.7: Visual overview of specialized training

The process starts with a target dataset $\mathcal{Z}^{target}$, one or more ancillary classifiers and the defined filter which is a mapping from the classes of the ancillary classifiers $\mathcal{Y}^{ancillary}$ to the classes in the target dataset $\mathcal{Y}^{target}$ which we define as $F : \mathcal{Y}^{target} \rightarrow \mathcal{Y}^{ancillary}$.

As a pre-processing step, we build our true positive dataset $\mathcal{Z}^{positive} = \{x, \mathbf{y} \in \mathcal{Z}^{target} \mid \mathbf{y} = F(x)\}$ and our false positive dataset $\mathcal{Z}^{negative} = \{x, \mathbf{y} \in \mathcal{Z}^{target} \mid \mathbf{y} \neq F(x)\}$.

We are then able to train our positive classifier $M^{positive}$ over $\mathcal{Z}^{positive}$ and our negative classifier $M^{negative}$ over $\mathcal{Z}^{negative}$.

4.5.1 Tool: training specialized classifiers

In order to operationalize the previous process, we require a few steps of initialization so that the tool knows where its inputs come from, what filters to use and where it will store its outputs. The tool also allows for some experimentation with the training configuration that will be used, *e.g.*, loss function, model architecture, batch size etc.

The process is rather straightforward and starts with giving a data directory which contains the labelled dataset. We assume here that the data for each class is separated into its own directory, and the directory name will be used as the class name for the network.

Next, we need to instantiate our baseline- and ancillary classifiers, together with our filter in the form of a dictionary. Our baseline classifier will be used to copy certain configurations (train-/test pre-processing functions) that allow for consistency in the newly generated specialized classifiers. Our ancillary classifiers are used to determine whether the defined filter is triggered or not for a datapoint, which determines whether it will be moved into the positive or negative dataset.

Finally, we provide some meta-data as to where the positive and negative classifiers will be stored and what identifier to use when referring to them.

4.6 Joint-decision structure

The final step of the specialization process is to create a joint-decision structure over our baseline- and specialized positive and negative classifiers. Each of them is specialized in their own contextual situation. and our goal is to leverage that specialization so that we can improve overall performance on the original

classification task. To create this joint-decision structure, we return back to the idea of orchestration, so that we can first assess the contextual situation over an input and then proceed to inquire the correct specialized classifier for that particular context.

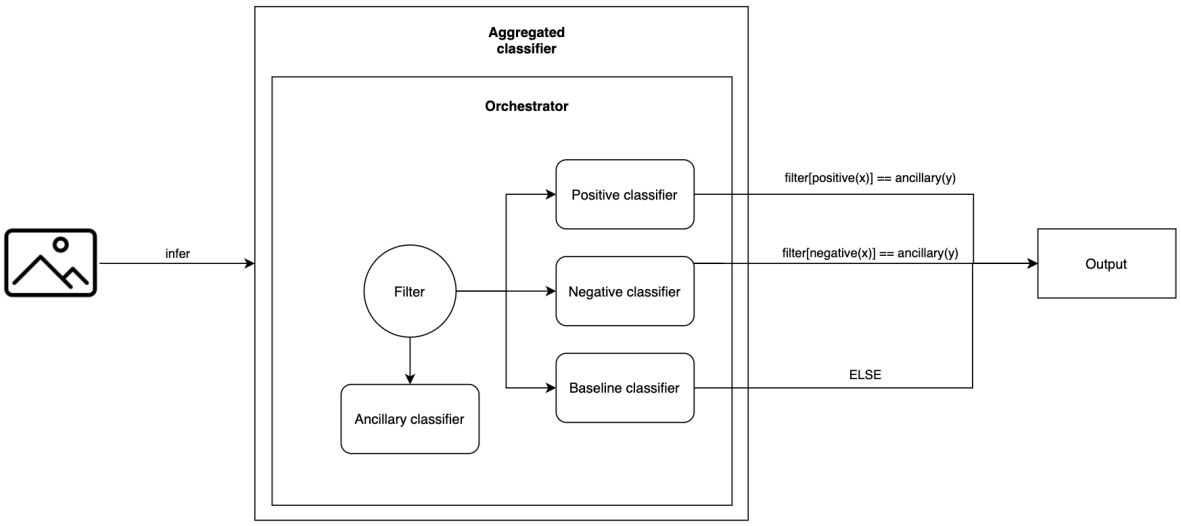

Figure 4.8: Visual overview of the joint-decision structure pipeline

Figure 4.8 gives a visual overview of the pipeline of how the decision structure could work. Essentially, we are creating a new classifier using a coordination of the context filter, ancillary classifiers and the positive-, negative- and baseline classifiers.

We have selected the following joint-decision structure after experimenting on a classification task in section 6.4. We form a social structure in which our network of models consists of the baseline, ancillary, positive and negative classifiers. Next we define our filter as a mapping from the target labels, to the ancillary classifiers. If the inference from $M^{positive}$ or $M^{negative}$ corresponds to a filter, wherein the output is the mapped value, then we take the output of one of the specialized classifiers for which this scenario is true. If the scenario does not trigger for either specialized classifier, we will fall back to the $M^{baseline}$ classifier.

This logic can be translated to the following ASP program:

```
#external ancillary(a;b).
#external positive(x;y).
#external negative(x;y).
#external baseline(x;y).

filter_x :- ancillary(a).
filter_y :- ancillary(b).

x :- positive(x), filter_x.
x :- negative(x), not filter_x.

y :- positive(y), filter_y.
y :- negative(y), not filter_y

x :- baseline(x), not y.
y :- baseline(y), not x.
```

Chapter 5

Target application resources

We will now focus on the classification task described in section 2.4. We design and train a set of classifiers, which are the resources that are available in the construction of the social structure. This section goes more into the details of the target classification task, together with how the neural networks or algorithms are designed, trained, and what data they are trained on.

This section also presents the baseline performance for each of the classifiers, some of which will be experimented further by retraining them with the use of other classifiers in the network (see sections 4.4, 6.3).

Training configuration The architecture of all models is based on the inception v3 network from Szegedy et al.[35], in which the last fully connected layer is connected to the output layer with n nodes, where n = number of output classes, and finally a softmax loss function. All models are initialized with random weights. Although the architecture was introduced five years ago, at the moment of writing, it still outperforms many of the top performing approaches in the field. Additionally, the decision to use the inception architecture is related to the wide support in ML-frameworks such as PyTorch [36] and also the fast speed in experimentation due to the relatively low number of parameters in the network [35].

All training and experimentation is performed on a single Nvidia GTX 1070 with a batch size of 8 for 10 epochs. For training, we used Stochastic Gradient Descent (SGD) [37] and momentum with a decay of 0.9, a learning rate of 0.001 and Nesterov acceleration [38]. This configuration is decided through pointers in literature [37, 38] and through experimenting.

5.1 Higher-level (HL) image classification

This classifier tries to classify a given image, extracted from a chemical patent document, into nine high-level categories in which the images can be categorized as; Drawing, chemical structure, program listing, gene sequence, flow chart, graph, math, table or character (symbol).

The dataset is curated by Piroi et al. [39] and consists of 38087 datapoints, distributed over the nine before mentioned classes. We have reduced this down to approximately 12k datapoints, for better distribution and improved speed in experimenting. The final goal is not to reach a perfectly performing classifier, but rather to use it in elaborating our ideas.

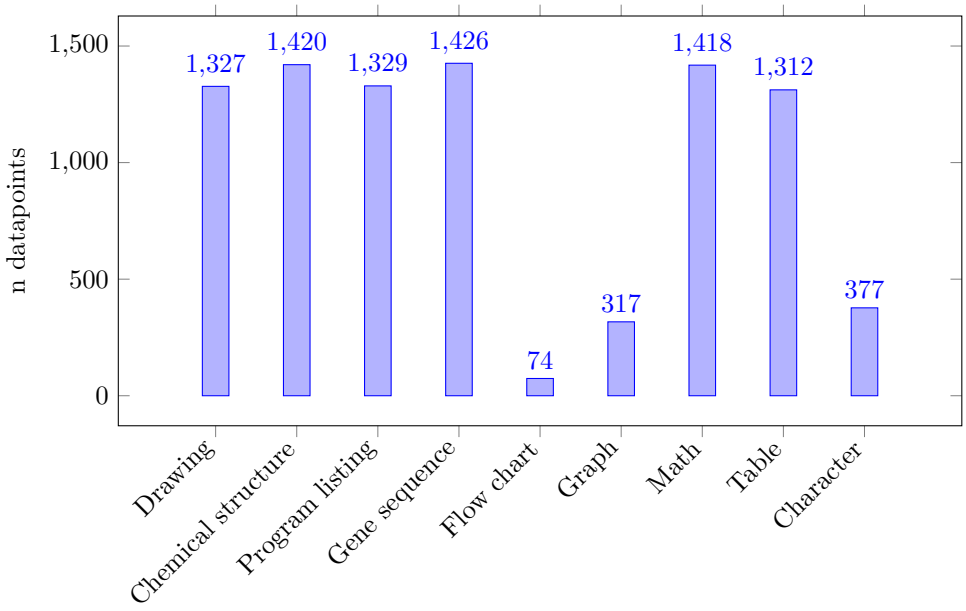

Figure 5.1: Dataset distribution over nine image classes; Drawing, chemical structure, program listing, gene sequence, flow chart, graph, math, table or character (symbol).

Baseline performance Baseline performance for this network, after training for 10 epochs, looks as follows:

Class	Precision	Recall	F1-score	Support
drawing	0.89	0.95	0.92	265
chemicalstructure	0.90	0.91	0.91	284
programlisting	0.79	0.90	0.84	266
genesequence	0.82	0.93	0.87	285
flowchart	1.00	0.07	0.12	15
graph	0.53	0.67	0.59	64
math	0.95	0.71	0.82	284
table	0.96	0.84	0.90	262
character	0.85	0.88	0.86	75
Accuracy			0.86	1800
Macro avg	0.85	0.76	0.76	1800
Weighted avg	0.87	0.86	0.86	1800

5.2 Drawing/not-drawing and not-drawing/not-not-drawing

The drawing classifier is derived from the previous classifier. We have taken the *drawing* class from the original dataset and all other classes, which are then curated into the *not_drawing* class.

At the end, the dataset contains a total of 9k images, as we selected an even distribution for the non-drawing classes in the original dataset, that sum up to the same amount as the `drawing` datapoints. The datapoints for the two classes are distributed as follows:

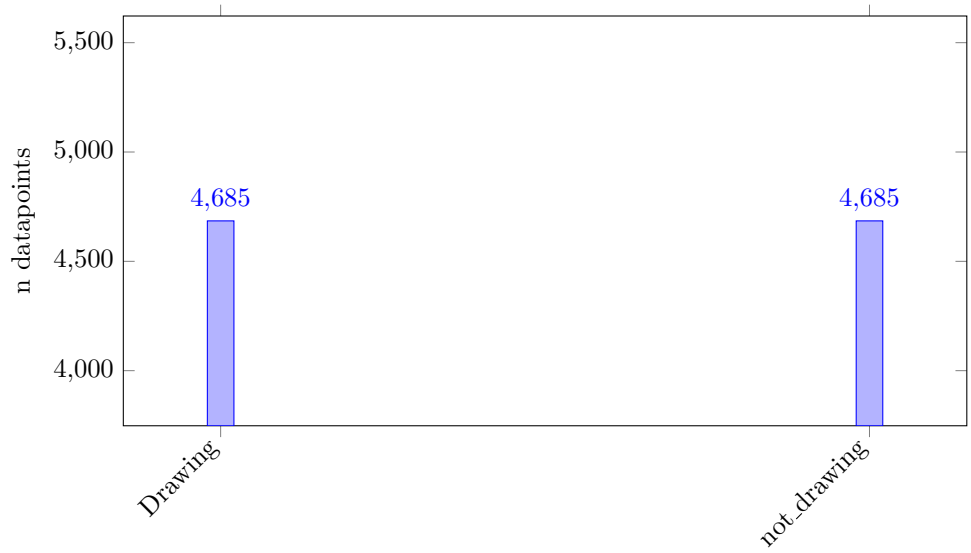

Figure 5.2: Dataset distribution over two image classes; Drawing and not drawing

Baseline performance Baseline performance for this network, after training for 10 epochs, looks as follows:

Class	Precision	Recall	F1-score	Support
drawing	0.93	0.93	0.93	685
not_drawing	0.93	0.93	0.93	685
Accuracy			0.93	1370
Macro avg	0.93	0.93	0.93	1370
Weighted avg	0.93	0.93	0.93	1370

5.2.1 Not drawing/not-not-drawing

For the drawing classifier, we have also constructed a dual classifier which classifies whether an image is *not_drawing* or *not_not_drawing*. The purpose of this dual classifier is that we are able to capture all possible outcomes (is a drawing, is not a drawing, could be a drawing) over the drawing classification and construct a conclusive inference.

To construct this classifier, we simply consider the original drawing datapoints as the *not_not_drawing* class and the *not_drawing* datapoints remains the same.

Baseline performance Baseline performance for this network, after training for 10 epochs, looks as follows:

Class	Precision	Recall	F1-score	Support
not_drawing	0.93	0.93	0.93	685
not_not_drawing	0.93	0.93	0.93	685
Accuracy			0.93	1370
Macro avg	0.93	0.93	0.93	1370
Weighted avg	0.93	0.93	0.93	1370

5.3 None-/one-/many chemical (CNCMANY) classification

Predicting whether images contain a depiction of a chemical structure is crucial for our classification task, described in section 2.4. This classifier performs a classification targeted on this specific task. We have logically three classes, namely: nonchemical, onechemical and manychemical.

The dataset is curated from the United States Patent and Trademark Office (USPTO[1]), using "Patent Grant Full Text Data with Embedded TIFF Images" from the first four weeks in the year 2021. Hereby all images with a "C" suffix in the image name are labeled as a chemical image and all remaining are labeled as non-chemical. This leaves us with a dataset of approx 140k images that either contain no chemicals or at least one chemical structure. Next, we run a tool called PraLine [40] over the chemical datapoints, to estimate the number of chemical structures in each of the images. We then split the chemical images into the onechemical and manychemical classes whenever they contain one or more than one chemical structures respectively, according to Praline. Finally, we take a subset of 7k images for each class so that we have a balanced dataset which also allows for fast experimenting.

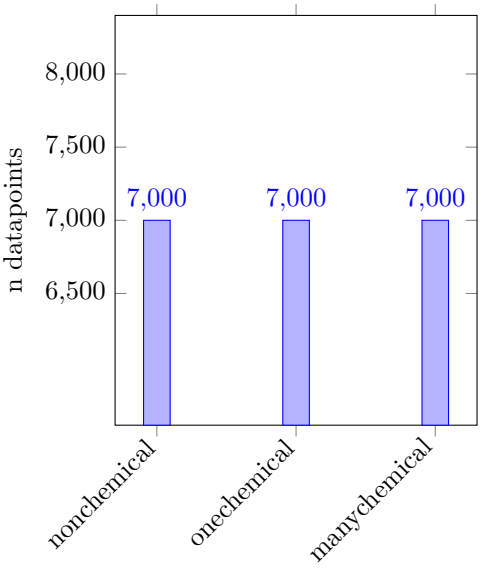

Figure 5.3: Dataset distribution of nonchemical, onechemical and manychemical images

Baseline performance for this network, after training for 10 epochs, looks as follows:

Class	Precision	Recall	F1-score	Support
manychemical	0.90	0.98	0.94	1400
nonchemical	0.99	0.97	0.98	1400
onechemical	0.96	0.90	0.93	1400
Accuracy			0.95	4200
Macro avg	0.95	0.95	0.95	4200
Weighted avg	0.95	0.95	0.95	4200

[1]United States Patent and Trademark Office bulkdata `https://bulkdata.uspto.gov/`

Chapter 6

Experiments and results

This chapter is the point in which we apply the ideas and concepts that we have introduced and elaborated up until now, on our own classification task. The goal of this application is that we can prove/display certain theories that we have laid out and also to run experiments to decide approaches for some topics such best performing filter or optimal joint-decision structure design.

Throughout the chapter we describe various results for which we use the following performance benchmarking metrics to compare the performance of the classifiers.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Precision tells us how often we are actually correct when positively classifying something.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Recall tells us how well we can find all the datapoints that belong to a certain class.

$$\text{F1-score} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

F1-score is a mean of both precision and recall in a way that it emphasizes the lowest value.

6.1 Classifying images of chemical structures

Along the way we have constructed our classifier (section 5.3) that aims to solve the classification task as described in section 2.4. We are able to make the following observation in regards to the results of this classifier:

- We are able to solve the classification task with an average F1-score of 0.95.
- We can classify images that contain more than one chemical structure with 0.94 F1-score.
- We can classify images that contain one chemical structure with 0.93 F1-score.
- We can classify images that do not contain any chemical structures with 0.98 F1-score.

6.2 Experiment: benchmarking rankings predictive capabilities

We run an experiment to determine which of the rankings generates a filter that has the best ability to predict true positive and false positive inferences for a real world classification task, as described in section 2.4. The experiment uses the CNCMANY classifier, which is described in section 5.3 as the target which we try to specialize. Next we use the Drawing classifier and its dual version, which are described in section 5.2, as our ancillary classifiers.

For each ranking, defined in section 4.4.3, the process of measuring performance has a number of steps, from end-to-end:

1. We filter the training/validation dataset according to the ranking outcomes. Each ranking experiment run receives the same input datasets because of the seed (42) that we have set for our random data generators.
2. We train an untrained classifier over this filtered dataset for 10 epochs.
3. We validate the newly trained classifier over the filtered validation set and calculate precision, recall and F1-scores for each class.

Table 6.1 gives the F1-score performance over the three output classes of the target classifier, together with the weighted F1-score average over all classes.

Ranking	F1-score (manychemical)	F1-score (onechemical)	F1-score (nonchemical)	F1-score (weighted avg)
r_1	0.93	0.92	0.96	0.93
r_2	0.92	0.81	0.98	0.96
r_4	0.82	0.83	0.99	0.95
r_5	0.92	0.81	0.98	0.96

Table 6.1: F1-score validation scores for each ranking

Note that we have omitted ranking r_3 from our benchmarks, which is because the ranking did not lead to a filter since there were no conclusive optimal relations.

We are able to observe the following findings through this experiment:

- Performance varies slightly (<5%) amongst rankings and F1-score performance of individual classes.
- Ranking 4.2 and 4.5 showed slightly better performance (<5%) amongst the rankings, in terms of F1-scores.

6.3 Experiment: performance improvements in specialized validation sets

We run an experiment to validate whether we are able to gain performance improvements over the specialization subsets of our real world use case. For this experiment we use as baseline-classifier, the CNCMANY classifier $M^{cncmany}$ 5.3 and we use the Drawing classifier $M^{drawing}$ and its dual version $M^{not\_drawing}$ as our ancillary classifiers. All metadata such as architecture, training (hyper-) parameters, etc. for our new specialized classifiers stay the same as the original CNCMANY classifier.

Using the r_5 ranking rule we come up with the following filter mapping:

```
filters = {
        'manychemical': [drawing, 'drawing'],
        'onechemical': [not_drawing, 'not_not_drawing'],
        'nonchemical': [not_drawing, 'not_drawing']
}
```

Now we describe the validation performance results comparing the specialized performance against the baseline classifier.

True positive (positive) specialization After applying our filter over the target dataset, we end up with the following subsets $\mathcal{Z}^{positive}$:

Dataset	manychemical	onechemical	nonchemical	Total
Training	2102	830	1983	4195
Validation	434	411	413	1258

Table 6.2: Training- and validation distribution of $\mathcal{Z}^{positive}$

Class	Precision	Recall	F1-score	Support
manychemical	0.91	0.95	0.93	352
nonchemical	1.00	0.98	0.99	776
onechemical	0.80	0.80	0.80	130
weighted avg	0.95	0.95	0.95	1258

Table 6.3: Validation performance of $M^{cncmany}$ over $\mathcal{Z}^{positive}$

After training our specialized classifier $M^{positive}$ over this subset, we get the following performance results:

Class	Precision	Recall	F1-score	Support
manychemical	0.87	0.99	0.93	352
nonchemical	0.99	0.98	0.99	776
onechemical	0.98	0.72	0.83	130
weighted avg	0.96	0.95	0.95	1258

Table 6.4: Validation performance of $M^{positive}$ over $\mathcal{Z}^{positive}$

In the previous two tables we can see that our specialized $M^{positive}$ becomes *slightly* (0.01) more precise in its predictions. We can also see that the dataset becomes imbalanced after the filtering step.

False positive (negative) specialization After applying our filter over the target dataset, we end up with the following subset $\mathcal{Z}^{negative}$:

Dataset	manychemical	onechemical	nonchemical	Total
Training	4898	1970	5017	11885
Validation	966	989	987	2942

Table 6.5: Training- and validation distribution of $\mathcal{Z}^{negative}$

Class	Precision	Recall	F1-score	Support
manychemical	0.91	0.98	0.94	1048
nonchemical	0.98	1.00	0.99	624
onechemical	0.99	0.91	0.95	1270
weighted avg	0.96	0.95	0.95	2942

Table 6.6: Validation performance of $M^{cncmany}$ over $\mathcal{Z}^{negative}$

After training our specialized classifier $M^{negative}$ over this subset, we get the following performance results:

Class	Precision	Recall	F1-score	Support
manychemical	0.95	0.97	0.96	1048
nonchemical	0.98	0.99	0.98	624
onechemical	0.97	0.95	0.96	1270
weighted avg	0.96	0.96	0.96	2942

Table 6.7: Validation performance of $M^{negative}$ over $\mathcal{Z}^{negative}$

In the previous two tables we can see that our specialized $M^{negative}$ becomes *slightly* (0.01) better in terms of recall and F1-score. We can also see that the dataset becomes imbalanced after the filtering step.

After running the experiments and benchmarking the specialized classifiers against the baseline classifier, we are able to make the following observations:

1. Both specialized classifiers were able to achieve (1) at least similar performance scores, with (2) less training datapoints. With $\mathcal{Z}^{positive}$ being a reduction of 74% data size, and $\mathcal{Z}^{negative}$ a reduction of 26% data size.
2. The filtering pre-processing step has a chance to construct imbalanced specialization datasets, with both $\mathcal{Z}^{positive}$ and $\mathcal{Z}^{negative}$ becoming imbalanced.
3. Both specialized classifiers improved in overall performance metrics over the baseline, with the $M^{negative}$ specialization improving in overall F1-score as well.

6.4 Experiment: optimal joint-decision structures

We have several design options in terms of how our joint-decision structure looks like, therefore we decide the optimal performing structuring through running benchmarks on our classification task.

We run the experiments with the same baseline classifier ($M^{cncmany}$), filter, ancillary classifiers($M^{drawing}$, $M^{not\_drawing}$) and specialized classifiers ($M^{positive}$, $M^{negative}$) that were used in section 6.3, for the sake of continuity and consistency.

We have come up with a number of designs, which we will describe down below and give generic example ASP programs for.

$design_1$ — if the corresponding filter for $M_{positive}$ output triggers we take the results of the $M_{positive}$ specialized classifier, else we take the output of the $M_{negative}$ specialized classifier:

```
filter_a = ancillary(a).
filter_b = ancillary(b).
x :- positive(x), filter_a.
y :- positive(y), filter_b.
x :- negative(x), not y.
y :- negative(y), not x.
```

$design_2$ — if all filters trigger we take the results of the $M_{positive}$ specialized classifier, else we take the output of the $M_{negative}$ specialized classifier:

```
filter :- ancillary(a), ancillary(b).
x :- positive(x), filter.
x :- negative(x), not filter.
y :- positive(y), filter.
y :- negative(y), not filter.
```

$design_3$ — if the corresponding filter for $M_{positive}$ output triggers we take the results of the $M_{positive}$ specialized classifier, else if the corresponding filter for $M_{negative}$ output triggers we take the result of the $M_{negative}$ specialized classifier, else we take the output of the $M_{baseline}$ classifier:

```
filter_a = ancillary(a).
filter_b = ancillary(b).
x :- positive(x), filter_a.
x :- negative(x), not filter_a.
y :- positive(y), filter_b.
y :- negative(y), not filter_b.
x :- baseline(x), not y.
y :- baseline(y), not x.
```

Now we present the performance difference over the three classes of $\mathcal{Z}_{cncmany}$ in table 6.8.

Design	F1-score (manychemical)	F1-score (onechemical)	F1-score (nonchemical)	F1-score (weighted avg)
$design_1$	0.90	0.87	0.91	0.89
$design_2$	0.76	0.75	0.78	0.76
$design_3$	0.88	0.89	0.96	0.91

Table 6.8: F1-scores for validation of each ranking

We are able to conclude that $design_3$ provides the best performing joint-decision structure.

6.5 Experiment: benchmarking joint-decision performance

We run an experiment to validate whether we are able to gain performance improvements over a baseline classifier, using a joint-decision structure which coordinates classifiers that are specialized over the classification task. In our experiment from section 6.3 we looked at performance improvements over specialization subsets, however our goal this time is to leverage the specialized classifiers to improve the original classification task. This idea is described in more detail in section 4.6.

We run the experiments with the same baseline classifier, filter, ancillary classifiers and specialized classifiers that were used in section 6.3, for the sake of continuity and consistency.

We are able to construct the following ASP program that coordinates both `positive` and `negative` specialized classifiers, according to the principles as described in section 4.6, which are in turn, a result of the experiments in section 6.4. Essentially this ASP program represents a new classifier that is the aggregation of the specialized classifiers and we shall call this $M^{aggregated}$.

```
/*Define our target classifiers*/
#external cncmany(manychemical;nonchemical;onechemical).
#external cncmany_positive(manychemical;nonchemical;onechemical).
#external cncmany_negative(manychemical;nonchemical;onechemical).

/*Define our ancillary classifiers*/
#external drawing(drawing;not_drawing).
#external not_drawing(not_drawing;not_not_drawing).

/*Translation of our filtering rules*/
filter_many :- drawing(drawing).
filter_none :- not_drawing(not_not_drawing).
filter_one :- not_drawing(not_drawing).

/*Take the positive classifier output if the corresponding filter is true */
manychemical :- cncmany_positive(manychemical), filter_many.
nonchemical :- cncmany_positive(nonchemical), filter_none.
onechemical :- cncmany_positive(onechemical), filter_one.

/*Take the negative classifier output if the corresponding filter is true */
manychemical :- cncmany_negative(manychemical), not filter_many.
nonchemical :- cncmany_negative(nonchemical), not filter_none.
onechemical :- cncmany_negative(onechemical), not filter_one.

/*Take the baseline classifier output if no conclusion was made or verify the conclusion
    */
manychemical :- cncmany(manychemical), not nonchemical, not onechemical.
nonchemical :- cncmany(nonchemical), not manychemical, not onechemical.
```

```
onechemical :- cncmany(onechemical), not manychemical, not nonchemical.
```

Now we are able to use this joint-decision structure to validate over the original validation data set of our baseline classifier $M^{cncmany}$.

The baseline performance is already provided in section 5.3, however, for completeness we will repeat it here:

Class	Precision	Recall	F1-score	Support
manychemical	0.90	0.98	0.94	1400
nonchemical	0.99	0.97	0.98	1400
onechemical	0.96	0.90	0.93	1400
Accuracy			0.95	4200
Macro avg	0.95	0.95	0.95	4200
Weighted avg	0.95	0.95	0.95	4200

Table 6.9: Baseline validation performance of $M^{cncmany}$

Now we validate the performance of our aggregated classifier $M^{aggregated}$:

Class	Precision	Recall	F1-score	Support
manychemical	0.83	0.93	0.88	1400
nonchemical	0.98	0.93	0.96	1400
onechemical	0.93	0.85	0.89	1400
Accuracy			0.91	4200
Macro avg	0.91	0.91	0.91	4200
Weighted avg	0.91	0.91	0.91	4200

Table 6.10: Aggregated validation performance of $M^{aggregated}$

We are able to observe that the performance for the overall network and all classes, decreased in our aggregated network.

We started investigation trying to debug the inferential process, which leads us to rule out one hypothesis that we had:

- We hypothesized that the poor performance was due to a mislabeling in our original dataset, and that our joint-decision structure predicted those mislabeled data points as the class that they *actually* were. However this hypothesis did not hold after a manual check of approximately 100 datapoints which either were labeled correctly but still classified incorrectly, or labeled incorrectly and also classified incorrectly in regards to what class they *actually* were.

We give the following possible reason for the disappointing performance results:

- Predicting the contextual situations (filter) are depending on the performance of our ancillary classifiers. Possible low performance could lead to inconsistent prediction of when we should take the output from either $M^{positive}$, $M^{negative}$ or $M^{baseline}$.

6.6 Tooling to control a network of machine learning models

The concept of model orchestration (section 4) has been a recurring theme throughout this research project and represents the main theory that is proposed for controlling machine learning classifiers. We elaborated on the idea of considering machine learning classifiers as individual computational units (or

agents) which we can coordinate by defining rules over their outputs. This is where the concept of a social structure arises which we are able to encapsulate and leverage, through Answer Set Programming (ASP).

Exploration and elaboration of these ideas form the basis for the tool that we have proposed in this project. Various ideas such as social structures, classifiers as independent computational units and representing relations as rules, are implemented in the tool as program code and form the building blocks for the described functionality such as performing inferences, exploring relationships, retraining specialized classifiers and enabling joint-decision structures.

More importantly we have shown a way to control networks of models and showed its feasibility by creating a proof of concept, being the tool. The result we consider here is the technical feasibility of the ideas, the various blueprints/designs of the ideas and an actual working tool that can be used.

Chapter 7

Findings and perspectives

We start this chapter with the findings that we were able to discover through this work and give answers to the questions and sub-questions of our research project. Next we discuss the threats to the validity of our research. Finally we describe the future work that remain open for our research topics.

7.1 Findings

7.1.1 RQ 1 - How can a network of machine learning models be controlled to tackle a classification task?

> **Finding 1:** We can orchestrate a network of machine learning classifiers by considering them as independent computational agents and forming them into a social structure by defining their relationships as rules.

In section 4 we have explored how we can apply the idea of orchestration to create a form of control structure which handles the inputs/outputs for the classifiers in the network of models. This orchestration allows us to create a layer of governance on top of our models, which naturally forms a flow of data in which each model's output adds to the knowledge pool of this dataflow.

> **Finding 2:** Answer Set Programming allows one to represent social structures, consisting of classifiers and rules, as logic programs, which could then be leveraged for orchestration.

Furthermore we have seen how to model the relationships of the classifier outputs, in regards to the overall dataflow leading to the network outcome, as logic program rules. Next we can compute the answer to the logic program, which represents the inference to our classification task. Finally we have seen that Answer Set Programming (ASP) is capable enough as the underlying system for modelling the network of classifiers, incorporating the relationships of their outputs and computing an overall network inferences.

> **Finding 3:** Integrating the idea of orchestrating social structures on top of Answer Set Programming, enables the development of a tool that can control a network of machine learning models for socially-conducted inferences.

We have shown the technical feasibility to integrate the idea of orchestrating social structures together with Answer Set Programming, into a tool which allows end-users to perform inferences using a network of machine learning models. The proposed tool is however still a proof-of-concept aiming to this goal and represents *one of the possible approaches*.

SQ 1.1 - How can we define a social structure over a set of machine learning models?

> **Finding 4:** Considering machine learning classifiers as independent computational agents, allows us to define a social structure, consisting of agents and relationships as rules.

The idea of agency can be applied in this context so that we are able to interface with individual classifiers as independent computational units. It creates a situation in which we can connect them to each other and govern their interaction (orchestration), but it also enables network re-structuring and generation of new information (modality). This is something that we elaborate on in section 4.0.2, 4.0.3 and 4.0.5. One could say this adds to the modularity of the system.

> **Finding 5:** Relationships over the outputs of individual classifiers can be translated into Answer Set Programming rules, forming a logic program.

In section 3.2 we describe the concept of relationships over the outcomes of classifiers, and in section 4.0.2 we explored how we can translate these relationships into rules in a social structure. The resulting principle is that we are able to represent relationships of individual classifiers as logic programming rules.

SQ 1.2 - How to select constraints and protocols that intervene on the inferential process of the network of models?

> **Finding 6:** In order to perform inference, using a network of models, we need to enforce expected behaviour and communication, which could be in the form of classes and wrappers.

From our research of section 4 we have seen that, in order to orchestrate a network of agents, we need them to behave and communicate in a reliable and expected manner. This way we can create a common understanding of how we structure inputs to the agents and respectively how their outputs will be structured.

Furthermore, we propose an approach, which is in line with an object-oriented way of thinking, on how to ensure this understanding. We do this by defining classes and wrappers that can be implemented by the individual agents, as described in section 4.1.

7.1.2 RQ 2 - How can we leverage a network of models to perform (re-)training?

> **Finding 7:** Capturing true positive and false positive datapoints, are a way of identifying datapoints which a classifier is good or bad at, in classifying. Assuming they have distinct features, we can divide a dataset into these two categories which allows us to train specialized classifiers on them.

In a computer vision task, if we assume that true positive and false positive datapoints are not *visually identical*, we logically conclude that one of them has distinct features that the other does not possess. We can then see how these distinct features, introduce noise and confusion when trying to classify them both as the same *thing*. We can reduce this confusion by training specialized classifiers on the two types of datapoints separately.

> **Finding 8:** We can use ancillary classifiers to capture contextual situations in which a datapoint is known to be either a true- or false positive. These contextual situations can then be used to predict *future* true- or false positives in the future. We can leverage this property by using the contextual situations as a filter to decide which specialized classifier's output to use as the outcome.

e.g., if we can observe that the classifier `P` is bad at correctly classifying input `a` when some fact `b` is true, but better when some other fact `c` is true, we know that we can rely more on `P`'s output whenever

`c` is true and even more if b is false. Considering facts `b` and `c` as observable contextual situations, we can leverage this idea by exploring whether we can use our ancillary classifiers to determine we are in such a contextual situation `b` or `c`. We have explored this idea in section 4.4 and have seen that we are indeed able to capture such contextual situations by analyzing the relation of the outputs of ancillary classifiers with regard to true positive and false positive datapoints.

> **Finding 9:** We can rank the significance of contextual situations, leading to more optimal specialization performance.

In the experiments of section 6.2 we have seen that we can rank contextual situations in terms of significance, proposing various heuristics to rank the relationship of a contextual situation with regard to true positive or false positive datapoints.

If our ancillary classifiers have some non-zero performing ability in identifying these contextual situations, we can use them to identify new datapoints that will be more likely a true positive or false positive. When we know this, we turn back to our idea of orchestration in which we coordinate the network of models to form a sort of joint-decision structure. Whenever the identified contextual situation is true, we use the outcome of the specialized classifier that is trained on this situation, otherwise we use the other specialized classifier.

SQ 2.1 - What benefits can be gained in terms of training performance and validation accuracy?

> **Finding 10:** Training classifiers over specialization datasets (true positives, false positives), can lead to performance improvements over the baseline.

In our experiment of section 6.3 we have seen that we can indeed create specialized classifiers that perform **better** than a baseline classifier, when we only train them over datapoints that were previously classified either correct or incorrectly.

> **Finding 11:** Forming a joint-decision structure, consisting of specialized- and ancillary classifiers, leads to worse overall task performance.

In our experiment of section 6.5 we have surprisingly seen that forming a joint-decision structure, consisting of our contextual situation filter and specialized classifiers, at the moment, does not lead to improved performance over a classification task compared to a baseline classifier. Further investigation is needed to understand why it failed our expectations.

7.2 Threats to validity

In this section, we discuss aspects of our project and results that might be subject to either internal- or external validity.

7.2.1 Generalization of the approach

Some of the experiments (sections 6.3, 6.5 and 6.4) that we have performed were done on a single classification task and reused the same set of classifiers throughout the exploration. One could therefore argue that the results are not a proper depiction of how the approaches generalize to other contexts. Ideally, we would run the experiments in a number of different contexts and measure how it performs overall to be drawing more concrete conclusions.

7.2.2 Generalization of the tool

Even though the tool is mostly a proof-of-concept, with some real world applicability, its development has mostly been done with the use of our classification task 2.4. Even though the tool is designed to support any use-case, within the bounds that we propose, it does require some knowledge of the internals

of the tool. *e.g.*, how one would perform the exploration and specialization steps, but also things like defining a social structure in code. The tool is therefore not fully automated or generic enough to be used by anybody. However, this thesis describes possibles directions on future developments and extensions, together with indications on how to reproduce the represented results, through describing the concepts and designs in a more general frame.

7.3 Future work

This thesis project has explored several ideas and directions in regard to the ideas of model orchestration and the proposed tool that we have developed along the way. We believe however that there is much more knowledge to be gained and more avenues to be explored in regard to these topics.

7.3.1 Generalization of the proposed tool

The proposed tool, including designs and architecture decisions, is essentially a proof-of-concept in its current form. It encapsulates the ideas and concepts that we elaborate on in this thesis and has the basic functionality to perform the described functionality. It would be beneficial to further develop such a tool and create a more generic solution that allows application for anyone that wants to leverage the concepts for their context.

7.3.2 Error handling of the proposed tool

Certain functionality is developed ad-hoc and only for the classification task that we have considered. Therefore there is certain error handling currently not in place, such as when a classifier does not properly extend the inference functionality and ends up returning empty outputs.

7.3.3 Classifier confidence

An aspect that is missing in our approach is the notion of confidence when performing inference. We could ask ourselves if we conclude an outcome `x` with only 10% confidence, should the conclusion be `x`? This notion of confidence could be applied to multiple aspects of the thesis such as the mentioned inference confidence but also regarding the strength of a classifier output itself. One could reason that certain classifiers in a network of models should not be able to change a conclusion on their own, but rather give a *nudge* towards being more/less confident about a conclusion.

7.3.4 Exploring additional specializations

Our current research only considers true positives and false-positive datapoints when trying to capture contextual situations with which specialized classifiers can be trained. However, this idea can be explored further by *e.g.*, considering true negatives and false negatives as well. Besides this, there are other options as well such as creating a sort of specialization loop, in which a new set of specialized classifiers are used to attempt to specialize further.

7.3.5 Joint-decision structure

In our experiments, we have failed to improve performance on a classification task, using a joint-decision structure of specialized classifiers, over a baseline classifier. However, we believe that there is still room for improvement because we were able to generate specialized classifiers that showed performance improvements over true positives and false positives. Logically, it would make sense that leveraging this *expert* knowledge in these specialized sub-tasks, would improve overall classification performance. Due to time restrictions, we were however unable to further debug the disappointing results. The general field of machine learning could however benefit if such an improvement is indeed possible.

7.3.6 Multi-modality

Can we orchestrate image/text/audio classifiers together? In this thesis, we have mainly focused on single-modality (images) and only single-output classifications. It would be interesting to explore how the proposed ideas generalize to different modalities, and what type of benefits could be discovered

with the proposed ideas and tool. One example could be to incorporate text-based models into a social structure, with which you expand the possible knowledge space of an inference.

7.3.7 Smaller dataset

In our classification task, we mostly considered models that we had an abundance of training data for. It would be interesting to explore whether there are additional/any benefits to the ideas that we propose when one has smaller datasets that are harder to reach convergence with.

Chapter 8

Conclusion

Controlling networks of machine learning models to tackle a classification task remains an open challenge within the field of machine learning. In this work, we have explored a novel approach to controlling networks of machine learning models and integrating high-level (symbolic) reasoning with low-level (sub-symbolic) perception, which is inspired by a software engineering way of thinking through separation of concerns and modularity. We consider machine learning models as independent computational agents and organize them by translating their relationships to logic programming rules, forming a social structure over them. This has allowed us to apply the idea of governance or orchestration over this social structure, opening opportunities towards coordination for inference, exploration, and specialization.

Furthermore, we have shown how we can further leverage this idea of orchestration to explore opportunities to retrain an optimized classifier. We saw that we were able to capture contextual situations in which a classifier performs well/badly, using ancillary classifiers in the network. Through experimenting, we have found ways in which we can capture the most significant contextual situation to improve performance. Splitting up a dataset according to these contextual situations allowed us to train new specialized classifiers that outperformed the baseline in their specific classification tasks. Then we looked at how we can form a sort of joint-decision structure using these specialized classifiers, to improve over the original classification task. Even though the results for this joint-decision structure approach were lower than the baseline, we believe that there is more to be investigated, that could be used to solve this unexpected outcome. The underlying motivation here is the result of generating improved specialized classifiers.

All of these findings were made whilst tackling a classification task that has not been solved before, which involved classifying whether an image from a chemical patent document, contains many, one or not any chemical structures. Concerning this goal, we proved that we can successfully construct a classifier that has an overall F1-score of 0.95.

Finally, we developed an orchestration tool during the thesis project, which encapsulates the ideas and concepts that we have touched upon. The tool can be seen as a proof-of-concept for the ideas that we propose and as a point of reference on which it can be expanded upon.

Bibliography

[1] B. P. Gokulan and D. Srinivasan, "An introduction to multi-agent systems," in. Jul. 2010, vol. 310, pp. 1–27, ISBN: 978-3-642-14434-9. DOI: 10.1007/978-3-642-14435-6_1.

[2] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit, *One model to learn them all*, 2017. arXiv: 1706.05137 [cs.LG].

[3] W. Wang, D. Tran, and M. Feiszli, "What makes training multi-modal classification networks hard?" In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 695–12 705.

[4] R. Manhaeve, S. Dumančić, A. Kimmig, T. Demeester, and L. D. Raedt, *Deepproblog: Neural probabilistic logic programming*, 2018. arXiv: 1805.10872 [cs.AI].

[5] Z. Yang, A. Ishay, and J. Lee, "Neurasp: Embracing neural networks into answer set programming," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI*, 2020, pp. 1755–1762.

[6] R. Wieringa and A. Moralı, "Technical action research as a validation method in information systems design science," in *International Conference on Design Science Research in Information Systems*, Springer, 2012, pp. 220–238.

[7] X.-D. Zhang, "Machine learning," in *A Matrix Algebra Approach to Artificial Intelligence*, Springer, 2020, pp. 223–440.

[8] E. Ilkou and M. Koutraki, "Symbolic vs sub-symbolic ai methods: Friends or enemies?" In *CIKM (Workshops)*, 2020.

[9] M. Gebser and T. Schaub, "Modeling and language extensions," *AI magazine*, vol. 37, no. 3, pp. 33–44, 2016.

[10] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. Schneider, "Potassco: The potsdam answer set solving collection," *Ai Communications*, vol. 24, no. 2, pp. 107–124, 2011.

[11] U. of Potsdam, *Potsdam answer set solving collection (potassco)*. [Online]. Available: https://potassco.org/.

[12] B. Kaufmann, N. Leone, S. Perri, and T. Schaub, "Grounding and solving in answer set programming," *AI Magazine*, vol. 37, no. 3, pp. 25–32, 2016.

[13] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele, "A user's guide to gringo, clasp, clingo, and iclingo," Oct. 2010.

[14] D. H. Warren, L. M. Pereira, and F. Pereira, "Prolog-the language and its implementation compared with lisp," *ACM SIGPLAN Notices*, vol. 12, no. 8, pp. 109–115, 1977.

[15] S. Ceri, G. Gottlob, L. Tanca, *et al.*, "What you always wanted to know about datalog(and never dared to ask)," *IEEE transactions on knowledge and data engineering*, vol. 1, no. 1, pp. 146–166, 1989.

[16] L. Naish, "Implementing negation," *Negation and Control in Prolog*, pp. 1–24, 1986.

[17] S. Akhondi, H. Rey, M. Schwörer, M. Maier, J. Toomey, H. Nau, G. Ilchmann, M. Sheehan, M. Irmer, C. Bobach, M. A. Doornenbal, M. Gregory, and J. Kors, "Automatic identification of relevant chemical compounds from patents," *Database: The Journal of Biological Databases and Curation*, vol. 2019, 2019.

[18] A. Abbas, L. Zhang, and S. Khan, "A literature review on the state-of-the-art in patent analysis," *World Patent Information*, vol. 37, Jun. 2014. DOI: 10.1016/j.wpi.2013.12.006.

[19] RELXgroup, *Reaxys*. [Online]. Available: https://www.reaxys.com/.

[20] K. Rajan, H. Brinkhaus, C. Steinbeck, and A. Zielesny, "A review of optical chemical structure recognition tools," *Journal of Cheminformatics*, vol. 12, Oct. 2020. DOI: 10.1186/s13321-020-00465-0.

[21] I. Filippov and M. Nicklaus, "Optical structure recognition software to recover chemical information: Osra, an open source solution," *Journal of chemical information and modeling*, vol. 49, pp. 740–3, Apr. 2009. DOI: 10.1021/ci800067r.

[22] M. Oldenhof, A. Arany, Y. Moreau, and J. Simm, "Chemgrapher: Optical graph recognition of chemical compounds by deep learning," *Journal of chemical information and modeling*, vol. 60, Sep. 2020. DOI: 10.1021/acs.jcim.0c00459.

[23] K. Rajan, A. Zielesny, and C. Steinbeck, "Decimer: Towards deep learning for chemical image recognition," *Journal of Cheminformatics*, vol. 12, no. 1, pp. 1–9, 2020.

[24] J. Staker, K. Marshall, R. Abel, and C. McQuaw, *Molecular structure extraction from documents using deep learning*, 2018. arXiv: 1802.04903 [cs.LG].

[25] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains: A review and perspectives," *Journal of Artificial Intelligence Research*, vol. 70, pp. 683–718, 2021.

[26] R. Evans and E. Grefenstette, "Learning explanatory rules from noisy data," *Journal of Artificial Intelligence Research*, vol. 61, pp. 1–64, 2018.

[27] A. d. Garcez, M. Gori, L. C. Lamb, L. Serafini, M. Spranger, and S. N. Tran, "Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning," *arXiv preprint arXiv:1905.06088*, 2019.

[28] L. Von Rueden, S. Mayer, J. Garcke, C. Bauckhage, and J. Schuecker, "Informed machine learning–towards a taxonomy of explicit integration of knowledge into machine learning," *learning*, vol. 18, pp. 19–20, 2019.

[29] G. B. Goh, C. Siegel, A. Vishnu, N. O. Hodas, and N. Baker, "Chemception: A deep neural network with minimal chemistry knowledge matches the performance of expert-developed qsar/qspr models," *arXiv preprint arXiv:1706.06689*, 2017.

[30] D. V. Porpora, "Four concepts of social structure," *Journal for the Theory of Social Behaviour*, vol. 19, no. 2, pp. 195–211, 1989.

[31] A. Giddens, *A contemporary critique of historical materialism*. Univ of California Press, 1981, vol. 1.

[32] C. Hewitt, *Actor model of computation: Scalable robust information systems*, 2015. arXiv: 1008.1459 [cs.PL].

[33] M. J. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," 1995, [Online]. Available: https://eprints.soton.ac.uk/252102/.

[34] R. Goebel, A. Chander, K. Holzinger, F. Lecue, Z. Akata, S. Stumpf, P. Kieseberg, and A. Holzinger, "Explainable ai: The new 42?" In *International cross-domain conference for machine learning and knowledge extraction*, Springer, 2018, pp. 295–303.

[35] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[36] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[37] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, Springer, 2010, pp. 177–186.

[38] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[39] F. Piroi, M. Lupu, A. Hanbury, and V. Zenz, "Clef-ip 2011: Retrieval in the intellectual property domain," Jan. 2011.

[40] V. A. Simossis and J. Heringa, "Praline: A multiple sequence alignment toolbox that integrates homology-extended and secondary structure information," *Nucleic acids research*, vol. 33, no. suppl_2, W289–W294, 2005.