



universität
wien

MASTERARBEIT / MASTER'S THESIS

Titel der Arbeit / Title of the Master's Thesis

Container Based Execution Stack for Neural Networks

verfasst von / submitted by

Benjamin Nussbaum, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2018 / Vienna, 2018

Studienkennzahl lt. Studienblatt /
degree programme code as it appears on
the student record sheet

926

Studienrichtung lt. Studienblatt /
degree programme as it appears on
the student record sheet

Masterstudium Wirtschaftsinformatik

Betreut von / Supervisor

Univ.-Prof. Dipl.-Ing. Dr. Erich Schikuta

Erklärung / Declaration

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Ort, Datum
Date

Unterschrift
Signature

Abstract / Zusammenfassung

Abstract

This thesis presents an execution stack for neural networks using the Kubernetes container orchestration and a Java based microservice architecture, which is exposed to users and other systems via RESTful webservices. The whole workflow including importing, training and evaluating a neural network model, becomes possible by using this service oriented approach. This work is influenced by N2Sky, a framework for the exchange of neural network specific knowledge and aims to support ViNNSL, the Vienna Neural Network Specification Language. The execution stack runs on many common cloud platforms. Furthermore it is scalable and each component is extensible and interchangeable.

Zusammenfassung

Diese Masterarbeit beschreibt einen Ausführungs-Stack für neuronale Netze, der unter Verwendung der Kubernes Container-Orchestrierung und einer Java basierten Microservice-Architektur, für Benutzer und Systeme via RESTful Webservices zugänglich gemacht wird. Der gesamte Arbeitsfluss, der Import, Training und Auswertung eines neuronalen Netzwerk-Modells beinhaltet, wird durch diese service-basierte Architektur (SOA) unterstützt. Der Ausführungs-Stack läuft auf vielen namhaften Cloud-Umgebungen, ist skalierbar und jede einzelne Komponente ist einfach erweiterbar und austauschbar.

Contents

1	Introduction	2
1.1	Motivation	3
1.2	Structure	4
1.3	Problem Statement	4
2	State of the Art	5
2.1	Machine Learning	5
2.1.1	Classification	6
2.1.2	Neural Network Frameworks	6
2.2	Container Orchestration	6
2.2.1	Docker Containers	6
2.2.2	Microservices	7
2.2.3	Comparison of Container Orchestration Technologies	7
2.2.4	Kubernetes	7
2.2.5	Docker Swarm	8
2.2.6	Kubernetes	8
3	Requirements	9
3.1	Functional Requirements	9
3.2	Non-Functional Requirements	9
4	Specification	10
4.1	Neural Network Objects	10
4.1.1	State of Neural Network Objects	10

Contents

5	Designing the API	11
5.1	REST API Documentation	11
5.1.1	Base URL	11
5.1.2	Import a new ViNNNSL XML Defintion	11
5.1.3	List all Neural Networks	14
5.1.4	Delete all Neural Networks	16
5.1.5	Get Neural Network Object	16
5.1.6	Remove Neural Network Object	21
5.1.7	Add/Replace File of Neural Network	22
5.1.8	Add/Replace ViNNNSL Definition of Neural Network	23
5.1.9	Add/Replace ViNNNSL Instanceschema of Neural Network	26
5.1.10	Add/Replace ViNNNSL Resultschema of Neural Network	28
5.1.11	Add/Replace ViNNNSL Trainingresult of Neural Network	29
5.1.12	Get Status of all Neural Networks	30
5.1.13	Get Status of Neural Network	31
5.1.14	Set Status of a Neural Network	32
5.1.15	Get Deeplearning4J Transformation Object of Neural Network	33
5.1.16	Put Deeplearning4J Transformation Object of Neural Network	34
6	Implementation of a Prototype	35
6.1	User Interface	35
7	Use Cases	37
8	Future Work	38
9	Conclusions	39
10	Acknowledgments	40
11	Dedication	41
12	Appendices	42

1 Introduction

This thesis presents an execution stack for neural networks using the *Kubernetes*¹ container orchestration and a Java based microservice architecture, which is exposed to users and other systems via RESTful web services and a web frontend. The whole workflow including importing, training and evaluating a neural network model, becomes possible by using this service oriented approach (SOA). The presented stack runs on popular cloud platforms, like *Google Cloud Platform*², *Amazon AWS*³ and *Microsoft Azure*⁴. Furthermore it is scalable and each component is extensible and interchangeable. This work is influenced by N2Sky [?], a framework to exchange neural network specific knowledge and aims to support *ViNNsL*, the Vienna Neural Network Specification Language [?].

Objectives: The first objective is to specify functional and non-functional requirements for the neural network system. This is followed by the characterisation of the API and the implementation of microservices that later define the neural network composition as a collection of loosely coupled services.

The next step is to setup a *Kubernetes* cluster to create the foundation of container orchestration.

Finally the microservices are deployed to containers and combined in a cluster.

Non-Objectives: The prototype does not fully implement the *ViNNsL 2.0*, as described in [?] and provides limited data in-/output. This is described in section TODO.

1 <https://kubernetes.io>

2 <https://cloud.google.com/kubernetes-engine>

3 <https://aws.amazon.com/eks>

4 <https://azure.microsoft.com/services/container-service>

1.1 Motivation

Machine learning has become a highly discussed topic in information technology in the past years and the trend is further increasing. It has become an essential part of everyday life when using search engines or speech recognition systems, like personal assistants. Self-learning algorithms in applications learn from the input of their users and decide which news an individual should read next, which song to listen to or which social media post should appear first. Messages are being analyzed and possible answers automatically predicted.

A recent Californian study shows that 6.5 million developers worldwide are currently involved in projects that use artificial intelligence techniques and another 5.8 million developers expect to implement these in near future [?].

Machine learning is not just a business area in the United States, survey results of 264 companies in the DACH region show, that 56 of them already use that kind of technology in production. In the near future 112 companies plan to do so or already have initial experiences (see figure 1.1). It is seen by a fifth of the decision-makers as a core area to improve the competitiveness and profitability of companies in future. [?]

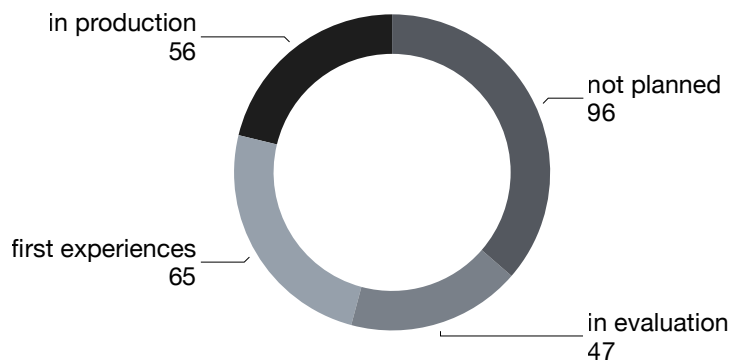


Figure 1.1: Distribution of machine learning of 264 companies in the DACH region [?]

1 Introduction

At the same time more and more companies shift their business logic from a monolithic design to microservices. Each service is dedicated to a single task that can be developed, deployed, replaced and scaled independently. Test results have shown that not only this architecture can help reduce infrastructure costs [?][?], but also reduces complexity of the code base and enables applications to dynamically adjust computing resources on demand [?].

The presented project combines these techniques and demonstrates a prototype that is open-source and is supported by common cloud providers. Developers can integrate their own solutions into the platform or exchange components ad libitum.

1.2 Structure

TODO

1.3 Problem Statement

TODO

- many different platforms
- complex field, hard to learn
- complex setup for many environments (design, training, auswertung, storage service, ...)

2 State of the Art

2.1 Machine Learning

Machine learning—the process by which computers can get better at performing tasks through exposure to data, rather than through explicit programming—requires massive computational power, the kind usually found in clusters of energy-guzzling, cloud-based computer servers outfitted with specialized processors. But an emerging trend promises to bring the power of machine learning to mobile devices that may lack or have only intermittent online connectivity. This will give rise to machines that sense, perceive, learn from, and respond to their environment and their users, enabling the emergence of new product categories, reshaping how businesses engage with customers, and transforming how work gets done across industries.(<https://www2.deloitte.com/insights/us/en/focus/signals-for-strategists/machine-learning-mobile-applications.html>) TODO CITATION

2 State of the Art

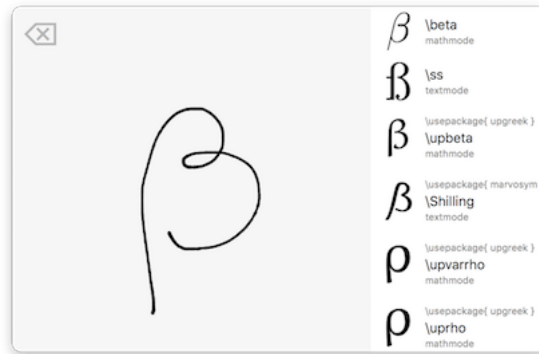


Figure 2.1: \LaTeX symbol classification app

2.1.1 Classification

LATEX Symbol classification application

2.1.2 Neural Network Frameworks

Tensorflow

Deeplearning4J

2.2 Container Orchestration

2.2.1 Docker Containers

Containers enable software developers to deploy applications that are portable and consistent across different environments and providers [?] by running isolated on top of the operating system's kernel [?]. As an organisation, Docker¹ has seen an increase of popularity very quickly, mainly because of its advantages, which are speed, portability, scalability, rapid delivery, and density [?].

Building a Docker container is fast, because they are small and and overhead of a virtual machine. The container format itself is standardized, which means that developers only

¹ <https://docker.com>

2 State of the Art

have to ensure that their application runs inside the container, which is then bundled into a single unit. The unit can be deployed on any Linux system as well as on various cloud environments and therefore easily be scaled. Not using a full operating system makes containers use less resources than virtual machines, which ensures higher workloads with greater density. [?]

2.2.2 Microservices

TODO As we break down an application into very specific domains, we need a uniform way to communicate between all the various pieces and domains. Web services have served this purpose for years, but the added isolation and granular focus that containers bring have paved a way for what is being named microservices. [?]

TODO In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies. (deloitte)

2.2.3 Comparison of Container Orchestration Technologies

2.2.4 Kubernetes

TODO Kubernetes is a system, developed by Google, for managing containerized applications across a cluster of nodes. The controlling services in a Kubernetes cluster are called the master components and have a number of unique services which are used to manage a cluster's workload and communications across the system[?].

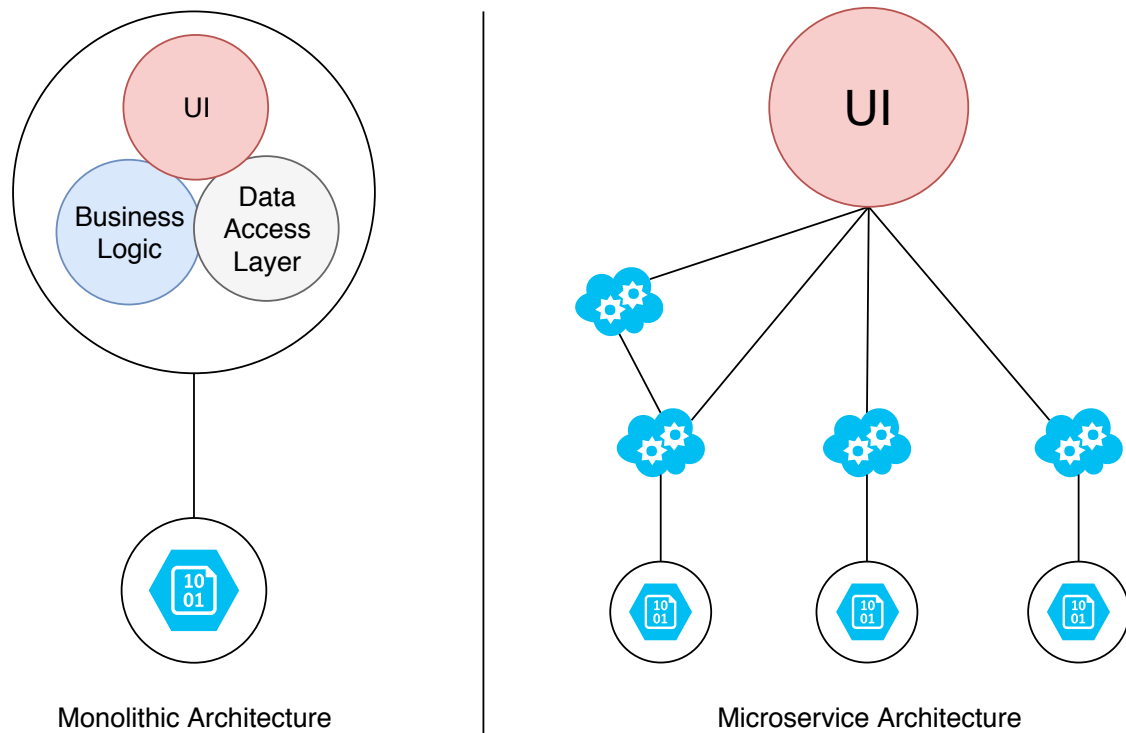


Figure 2.2: Monolithic Architecture vs. Microservice Architecture

2.2.5 Docker Swarm

2.2.6 Kubernetes

3 Requirements

3.1 Functional Requirements

3.2 Non-Functional Requirements

4 Specification

4.1 Neural Network Objects

4.1.1 State of Neural Network Objects

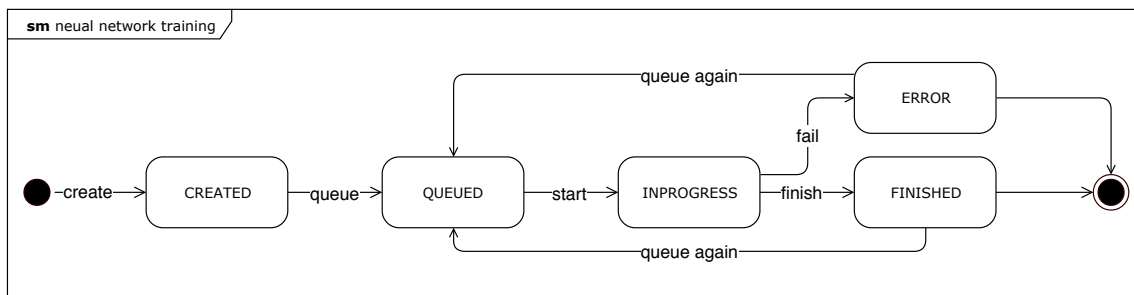


Figure 4.1: State Machine of a Neural Network

5 Designing the API

5.1 REST API Documentation

5.1.1 Base URL

`http[s]://<clusterip>`

5.1.2 Import a new ViNNsL XML Defintion

POST /vinnsl

Parameters

Type	Name	Description	Schema
Body	vinnsl <i>required</i>	vinnsl	Vinnsl

Responses

HTTP Code	Description	Schema
201	Created	No Content
500	Server Error	Error

Consumes

- application/xml

Produces

- */*

Tags

- vinns1-service-controller

Example HTTP request

Header

Content-Type: application/xml

Body

```
<vinns1>
  <description>
    <identifier><!-- will be generated --></identifier>
    <metadata>
      <paradigm>classification</paradigm>
      <name>Backpropagation Classification</name>
      <description>Iris Classification Example</description>
      <version>
        <major>1</major>
        <minor>0</minor>
      </version>
    </metadata>
```

5 Designing the API

```
<creator>
  <name>Ronald Fisher</name>
  <contact>ronald.fisher@institution.com</contact>
</creator>
<problemDomain>
  <propagationType type="feedforward">
    <learningType>supervised</learningType>
  </propagationType>
  <applicationField>Classification</applicationField>
  <networkType>Backpropagation</networkType>
  <problemType>Classifiers</problemType>
</problemDomain>
<endpoints>
  <train>true</train>
  <retrain>true</retrain>
  <evaluate>true</evaluate>
</endpoints>
<structure>
  <input>
    <ID>Input1</ID>
    <size>
      <min>4</min>
      <max>4</max>
    </size>
  </input>
  <hidden>
    <ID>Hidden1</ID>
    <size>
      <min>3</min>
      <max>3</max>
    </size>
  </hidden>
  <hidden>
    <ID>Hidden2</ID>
    <size>
```

5 Designing the API

```
<min>3</min>
<max>3</max>
</size>
</hidden>
<output>
  <ID>Output1</ID>
  <size>
    <min>3</min>
    <max>3</max>
  </size>
</output>
</structure>
<parameters/>
<data>
  <description>iris txt file with 3 classifications, 4 input vars</description>
  <tabledescription>no input as table possible</tabledescription>
  <filedescription>CSV file</filedescription>
</data>
</description>
</vinns1>
```

Example HTTP response

Statuscode: 201 CREATED

Header

Location: <https://<baseURL>/vinns1/5ade36bbd601800001206798>

5.1.3 List all Neural Networks

GET /vinns1

Responses

HTTP Code	Description	Schema
200	OK	< VinnsI > array
404	Not Found	No Content
500	Server Error	Error

Produces

- application/json

Tags

- vinnsI-service-controller

Example HTTP Response

```
[
  {
    "identifier": "5ab91658e8cc45946600ea11",
    "description": {},
    "definition": {},
    "data": {},
    "instance": {},
    "trainingresult": {},
    "result": {},
    "nncloud": {
      "status": "CREATED",
      "dl4jNetwork": "{}"
    }
  },
]
```

```
...  
]
```

5.1.4 Delete all Neural Networks

DELETE /vinns1/deleteall

Responses

HTTP Code	Description	Schema
200	OK	object
204	No Content	No Content
500	Server Error	Error

Produces

- application/json

Tags

- vinns1-service-controller

5.1.5 Get Neural Network Object

GET /vinns1/{id}

Parameters

5 Designing the API

Type	Name	Description	Schema
Path	id <i>required</i>	id	string

Responses

HTTP Code	Description	Schema
200	OK	Vinnsl
404	Not Found	No Content

Produces

- application/xml
- application/json

Tags

- vinnsl-service-controller

Example HTTP response

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<vinnsl>
  <identifier>5ab91658e8cc45946600ea11</identifier>
  <description>
    <identifier></identifier>
    <metadata>
      <paradigm>classification</paradigm>
      <name>Backpropagation Classification</name>
      <description>Face Recognition Example</description>
```

5 Designing the API

```
<version>
  <major>1</major>
  <minor>5</minor>
</version>
</metadata>
<creator>
  <name>Autor 1</name>
  <contact>author1@institution.com</contact>
</creator>
<problemDomain>
  <propagationType type="feedforward">
    <learningType>supervised</learningType>
  </propagationType>
  <applicationField>EMS</applicationField>
  <applicationField>Operations</applicationField>
  <applicationField>FaceRecognition</applicationField>
  <networkType>Backpropagation</networkType>
  <problemType>Classifiers</problemType>
</problemDomain>
<endpoints>
  <train>true</train>
  <retrain>true</retrain>
  <evaluate>true</evaluate>
</endpoints>
<structure>
  <input>
    <ID>Input1</ID>
    <dimension>
      <min>1</min>
      <max>1</max>
    </dimension>
    <size>
      <min>960</min>
      <max>960</max>
    </size>
```

5 Designing the API

```
</input>
<hidden>
  <ID>Hidden1</ID>
  <dimension>
    <min>1</min>
    <max>1024</max>
  </dimension>
</hidden>
<output>
  <ID>Output1</ID>
  <dimension>
    <min>1</min>
    <max>1</max>
  </dimension>
  <size>
    <min>1</min>
    <max>1</max>
  </size>
</output>
</structure>
<parameters/>
<data>
  <description>Input are face images with 32x30 px</description>
  <tabledescription>no input as table possible</tabledescription>
  <filedescription>prepare the input as file by reading the image files</filedescription>
</data>
</description>
<definition>
  <identifier></identifier>
  <problemDomain>
    <propagationType type="feedforward">
      <learningType>supervised</learningType>
    </propagationType>
    <applicationField>EMS</applicationField>
    <applicationField>Operations</applicationField>
  </problemDomain>
</definition>
```


5 Designing the API

```
<applicationField>FaceRecognition</applicationField>
<networkType>Backpropagation</networkType>
<problemType>Classifiers</problemType>
</problemDomain>
<endpoints></endpoints>
<executionEnvironment>
  <serial>true</serial>
</executionEnvironment>
<structure>
  <input>
    <ID>Input1</ID>
    <dimension>1</dimension>
    <size>960</size>
  </input>
  <hidden>
    <ID>Hidden1</ID>
    <dimension>1</dimension>
    <size>1024</size>
  </hidden>
  <output>
    <ID>Output1</ID>
    <dimension>1</dimension>
    <size>1</size>
  </output>
  <connections/>
</structure>
<resultSchema>
  <instance>true</instance>
  <training>true</training>
</resultSchema>
<parameters>
  <valueparameter name="learningrate">0.4</valueparameter>
  <valueparameter name="biasInput">1</valueparameter>
  <valueparameter name="biasHidden">1</valueparameter>
  <valueparameter name="momentum">0.1</valueparameter>
```

5 Designing the API

```
<comboparameter name="activationfunction">sigmoid</comboparameter>
<valueparameter name="threshold">0.00001</valueparameter>
<comboparameter name="activationfunction">sigmoid</comboparameter>
</parameters>
<data>
  <description>Input are face images with 32x30 px</description>
  <dataSchemaID>iris.txt</dataSchemaID>
</data>
</definition>
<data>
  <identifier>5ab4e69c8f136a16bf81f093</identifier>
  <data>
    <file>5ab4e69c8f136a16bf81f093</file>
  </data>
</data>
</vinns1>
```

5.1.6 Remove Neural Network Object

DELETE /vinns1/{id}

Parameters

Type	Name	Description	Schema
Path	id <i>required</i>	id	string

Responses

HTTP Code	Description	Schema
200	OK	ResponseEntity
204	No Content	No Content

5 Designing the API

HTTP Code	Description	Schema
500	Server Error	No Content

Produces

- */*

Tags

- vinns1-service-controller

5.1.7 Add/Replace File of Neural Network

PUT /vinns1/{id}/addfile

Parameters

Type	Name	Description	Schema
Path	id <i>required</i>	id	string
Query	fileId <i>required</i>	fileId	string

Responses

HTTP Code	Description	Schema
200	OK	Vinns1
404	Not Found	No Content
500	Server Error	Error

Consumes

- application/json

Produces

- application/xml
- application/json

Tags

- vinnsl-service-controller

5.1.8 Add/Replace ViNNsL Definition of Neural Network

PUT /vinnsl/{id}/definition

Parameters

Type	Name	Description	Schema
Path	id <i>required</i>	id	string
Body	def <i>required</i>	def	Definition

Responses

HTTP Code	Description	Schema
200	OK	Vinnsl
404	Not Found	No Content

5 Designing the API

HTTP Code	Description	Schema
500	Server Error	Error

Consumes

- application/xml
- application/json

Produces

- */*

Tags

- vinns1-service-controller

Example HTTP request

Request body

```
<definition>
<identifier><!-- will be generated --></identifier>
<metadata>
  <paradigm>classification</paradigm>
  <name>Backpropagation Classification</name>
  <description>Iris Classification Example</description>
  <version>
    <major>1</major>
    <minor>0</minor>
  </version>
</metadata>
```

5 Designing the API

```
<creator>
  <name>Ronald Fisher</name>
  <contact>ronald.fisher@institution.com</contact>
</creator>
<problemDomain>
  <propagationType type="feedforward">
    <learningType>supervised</learningType>
  </propagationType>
  <applicationField>Classification</applicationField>
  <networkType>Backpropagation</networkType>
  <problemType>Classifiers</problemType>
</problemDomain>
<endpoints>
  <train>true</train>
</endpoints>
<executionEnvironment>
  <serial>true</serial>
</executionEnvironment>
<structure>
  <input>
    <ID>Input1</ID>
    <size>4</size>
  </input>
  <hidden>
    <ID>Hidden1</ID>
    <size>3</size>
  </hidden>
  <hidden>
    <ID>Hidden2</ID>
    <size>3</size>
  </hidden>
  <output>
    <ID>Output1</ID>
    <size>3</size>
  </output>
```

5 Designing the API

```
<connections>
  <!--<fullconnected>
    <fromblock>Input1</fromblock>
    <toblock>Hidden1</toblock>
    <fromblock>Hidden1</fromblock>
    <toblock>Output1</toblock>
  </fullconnected>-->
</connections>
</structure>
<resultSchema>
  <instance>true</instance>
  <training>true</training>
</resultSchema>
<parameters>
  <valueparameter name="learningrate">0.1</valueparameter>
  <comboparameter name="activationfunction">tanh</comboparameter>
  <valueparameter name="iterations">500</valueparameter>
  <valueparameter name="seed">6</valueparameter>
</parameters>
<data>
  <description>iris txt file with 3 classifications, 4 input vars</description>
  <dataSchemaID>name/iris.txt</dataSchemaID>
</data>
</definition>
```

5.1.9 Add/Replace ViNNsL Instanceschema of Neural Network

PUT /vinns1/{id}/instanceschema

Parameters

Type	Name	Description	Schema
Path	id <i>required</i>	id	string

5 Designing the API

Type	Name	Description	Schema
Body	instance <i>required</i>	instance	Instanceschema

Responses

HTTP Code	Description	Schema
200	OK	object
404	Not Found	No Content
500	Server Error	Error

Consumes

- application/xml
- application/json

Produces

- */*

Tags

- vinnsli-service-controller

Example HTTP request

Request body

```
<instanceschema>  
</instanceschema>
```


5.1.10 Add/Replace ViNNSL Resultschema of Neural Network

PUT /vinns1/{id}/resultschema

Parameters

Type	Name	Description	Schema
Path	id <i>required</i>	id	string
Body	resultSchema <i>required</i>	resultSchema	Resultschema

Responses

HTTP Code	Description	Schema
200	OK	object
404	Not Found	No Content
500	Server Error	Error

Consumes

- application/xml
- application/json

Produces

- */*

Tags

- vinns1-service-controller

Example HTTP request

Request body

```
<resultschema>
</resultschema>
```

5.1.11 Add/Replace ViNNsL Trainingresult of Neural Network

PUT /vinns1/{id}/trainingresult

Parameters

Type	Name	Description	Schema
Path	id <i>required</i>	id	string
Body	trainingresult <i>required</i>	trainingresult	Trainingresultschema

Responses

HTTP Code	Description	Schema
200	OK	object
404	Not Found	No Content
500	Server Error	Error

Consumes

- application/xml
- application/json

Produces

- */*

Tags

- vinnsl-service-controller

Example HTTP request

Request body

```
<trainingresult>
</trainingresult>
```

5.1.12 Get Status of all Neural Networks

GET /status

Responses

HTTP Code	Description	Schema
200	OK	object
404	Not Found	No Content

Produces

- application/json

Tags

- nn-status-controller

HTTP response example

```
{  
  "5ab91658e8cc45946600ea11": "INPROGRESS"  
}
```

5.1.13 Get Status of Neural Network

GET /status/{id}

Parameters

Type	Name	Description	Schema
Path	id <i>required</i>	id	string

Responses

HTTP Code	Description	Schema
200	OK	object
404	Not Found	No Content

Produces

- application/json

Tags

- nn-status-controller

5.1.14 Set Status of a Neural Network

PUT /status/{id}/{status}

Parameters

Type	Name	Description	Schema
Path	id <i>required</i>	id	string
Path	status <i>required</i>	status	enum (CREATED, QUEUED, INPROGRESS, FINISHED, ERROR)

Responses

HTTP Code	Description	Schema
200	OK	object
404	Not Found	No Content
500	Server Error	Error

Consumes

- application/json

Produces

- application/json

Tags

- nn-status-controller

5.1.15 Get Deeplearning4J Transformation Object of Neural Network

GET /dl4j/{id}

Parameters

Type	Name	Description	Schema
Path	id <i>required</i>	id	string

Responses

HTTP Code	Description	Schema
200	OK	string
404	Not Found	No Content

Produces

- application/json

Tags

- dl4j-service-controller

5.1.16 Put Deeplearning4J Transformation Object of Neural Network

PUT /dl4j/{id}

Parameters

Type	Name	Description	Schema
Path	id <i>required</i>	id	string
Body	dl4J <i>required</i>	dl4J	string

Responses

HTTP Code	Description	Schema
200	OK	ResponseEntity
404	Not Found	No Content
500	Server Error	Error

Consumes

- application/json

Produces

- application/json

Tags

- dl-4j-service-controller

6 Implementation of a Prototype

6.1 User Interface

6 Implementation of a Prototype

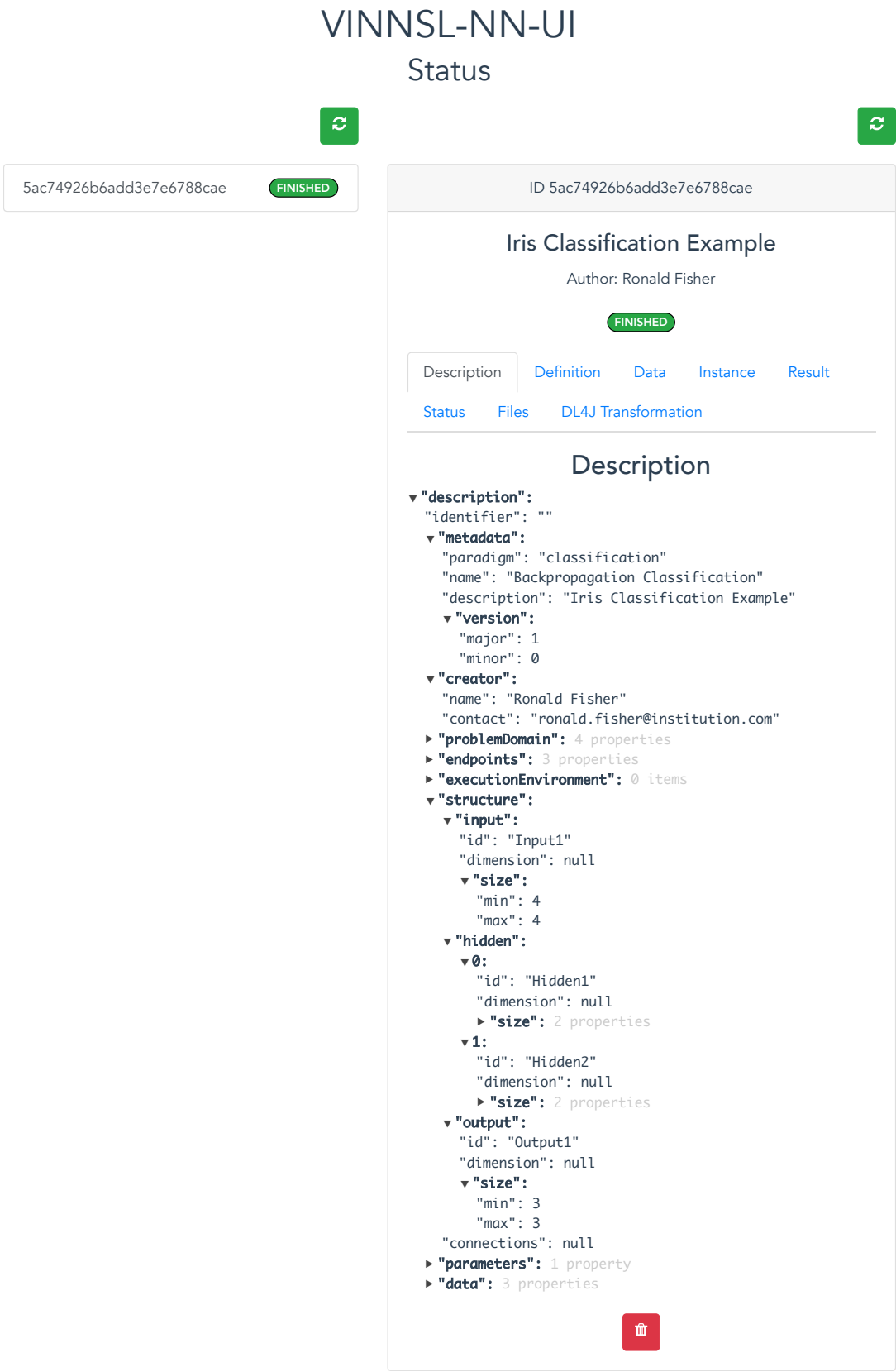


Figure 6.1: User Interface of Prototype

7 Use Cases

- 1) iris classification
- 2) MNIST?
- 3) hosted trained network

8 Future Work

TODO

- more function
- backend für tensorflow
- grafischer NN designer
- trainierte netzwerke als webservice veröffentlichen

9 Conclusions

10 Acknowledgments

11 Dedication

12 Appendices