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	Unterschrift
Date	Signature

Abstract / Zusammenfassung

Abstract

This thesis presents a **con**tainer **b**ased **ex**ecution stack for **n**eural **n**etworks (ConbexNN) 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 (genannt ConbexNN), 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 beinhält, wird durch diese service-basierte Architektur (SOA) unterstützt. Diese Arbeit ist von N2Sky, einem Framework zum Austausch spezifischen Wissens über neuronale Netze, beeinflusst und unterstützt ViNNSL, die Vienna Neural Network Specification Language. Der Ausführungs-Stack läuft auf vielen namhaften Cloud-Umgebungen, ist skalierbar und jede einzelne Komponente ist einfach erweiterbar und austauschbar.

Contents

1 Introduction

This thesis presents a **con**tainer **b**ased **ex**ecution stack for **n**eural **n**etworks (ConbexNN) 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 [SM13], a framework to exchange neural network specific knowledge and aims to support *ViNNSL*, the Vienna Neural Network Specification Language [Kop15] [BVSW08].

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 implemention 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 of ConbexNN does not fully implement the *ViNNSL* in version 2.0, as described in [Kop15] and provides limited data in-/output. Limitations are described in section 5.8.

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 Problem Statement

Getting started with machine learning and in particular with neural networks is not a trivial task. It is a complex field with a high entry barrier and most often requires programming skills and expertise in neural network frameworks. In most cases a complex setup is needed to train and evaluate networks, which is both, a processor- and memory-intense job. With cloud computing getting more and more affordable and powerful, it makes sense to shift these tasks into the cloud. There are already existing cloud platforms for machine learning, but to my present research all of them do not fulfil at least one of the following criteria:

- platform is open-source
- no programming skills required to define and train a neural network model
- can be deployed on-site and and in the cloud (of your choice)
- components extensible and replaceable by developers
- provides a RESTful interface

This thesis showcases an architecture, that tries to achieve all of that.

1.2 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 [Eva17].

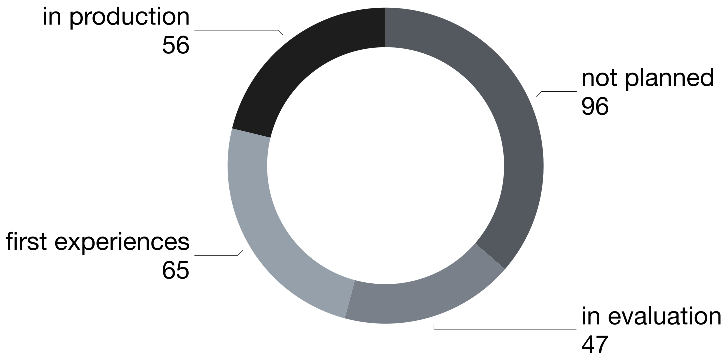

Figure 1.1: Distribution of machine learning of 264 companies in the DACH region [BB16]

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. [BB16]

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 [VGO+16][VGC+15], but also reduces complexity of the code base and enables applications to dynamically adjust computing resources on demand [VGC+15].

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

It also integrates with ViNNSL, a descriptive language that does not require programming skills to define, train and evaluate neural networks.

1.3 Structure

This thesis gives an introduction and comparison to state of the art technologies that support the microservice architecture pattern using container and container orchestration tools. This is followed by the acquaintance of Machine Learning (ML) and commonly used ML Frameworks. Featuring all these introduced technologies, requirements are defined for the implementation of ConbexNN. Main sections of this thesis are the specification, implementation and documentation following common practices. To demonstrate the operational purposes of ConbexNN, two use cases are presented.

Future work mentions ideas on how ConbexNN can be extended and integrated into other systems and the conclusion summarizes the motivation and archivements of the implemented neural network execution stack.

1.4 Related Work

1.4.1 ViNNSL

The Vienna Neural Network Specification Language (*ViNNSL*) is a domain specific language developed by the University of Vienna to describe neural network objects and is designed as a communication framework in service-oriented architectures. It is based on XML and provides the schemas that allow the creation, training and evaluation of artificial neural networks. [BVSW08]

1.4.2 N2Sky

N2Sky is a cloud-based platform developed by the University of Vienna that follows the Neural Networks as a Service paradigm and provides an implementation example of *ViNNSL*. It is designed as virtual collaboration platform allowing to exchange neural network knowledge with a neural network community. The service delivers an interface to create and train neural network objects and subsequently share them with the community. [SM13][FAS18]

2 State of the Art

2.1 Containers

2.1.1 Docker Containers

Containers enable software developers to deploy applications that are portable and consistent across different environments and providers [Bai15] by running isolated on top of the operating system's kernel [BRBA17]. As an organisation, Docker[1] has seen an increase of popularity very quickly, mainly because of its advantages compared to other solutions, which are speed, portability, scalability, rapid delivery and density [BRBA17].

Building a Docker container is fast, because images do not include a guest operating system. The container format itself is standardized, which means that developers just 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. [Joy15]

2.2 Microservices

The micoservice architecture pattern is a variant of a service-oriented architecture (SOA). An often cited definition originates from Martin Fowler and James Lewis:

1 https://docker.com

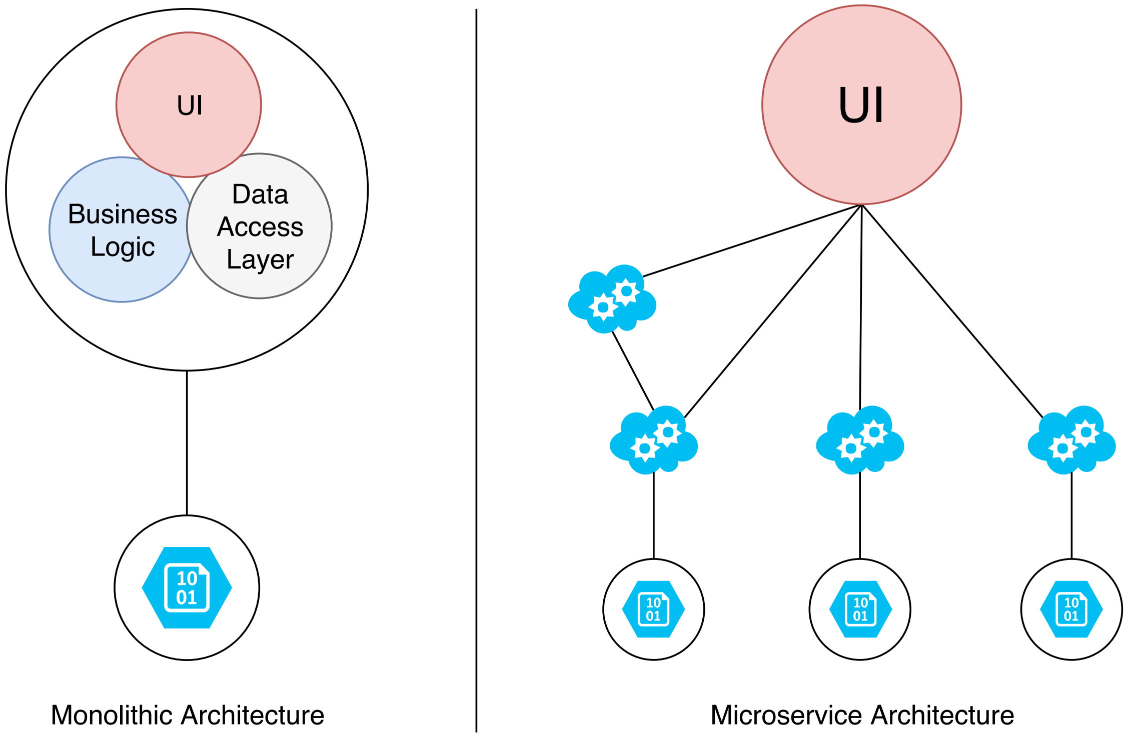

Figure 2.1: Monolithic Architecture vs. Microservice Architecture

> 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. [LF14]

Figure 2.1 shows the architectural difference between the monolithic and microservice architecture. Monolithic applications bundle user interface, data access layer and business logic together as a single unit. In the microservice architecture each task has its own service. The user interface puts information together from multiple services.

2.3 Container Orchestration Technologies

As every single microservice runs as a container, we need a tool to manage, organise and replace these containers. Services should also be able to speak to each other and to be restarted if they fail. Services under heavy load should be scaled for better performance. To deal with these challenges container orchestration technologies come into place. According to a study from 2017 published by Portworx, Kubernetes is the most frequently used container orchestration tool in organizations, followed by Docker Swarm and Amazon ECS. [Por]

This section describes the architecture of the mentioned container orchestration technologies and compares them.

2.3.1 Kubernetes

Kubernetes is the third container-management system (after Borg and Omega) developed by Google [BGO$^+$16] for administering applications, that are provided in containers, in a cluster of nodes. Services that are responsible for controlling the cluster, are called master components [Ell16]. Figure 2.2 shows the Kubernetes core architecture, which includes the Master server, the nodes and the interaction between the components.

Master Components

The master consists of the core API server, that provides information about the cluster and workload state and allows to define the desired state [Bai15]. The master server also takes care of scheduling and scaling workloads, cluster-wide networking and performs health checks [Ell16]. Workloads are managed in form of so-called pods, which are various containers that conclude the application stacks [Bai15].

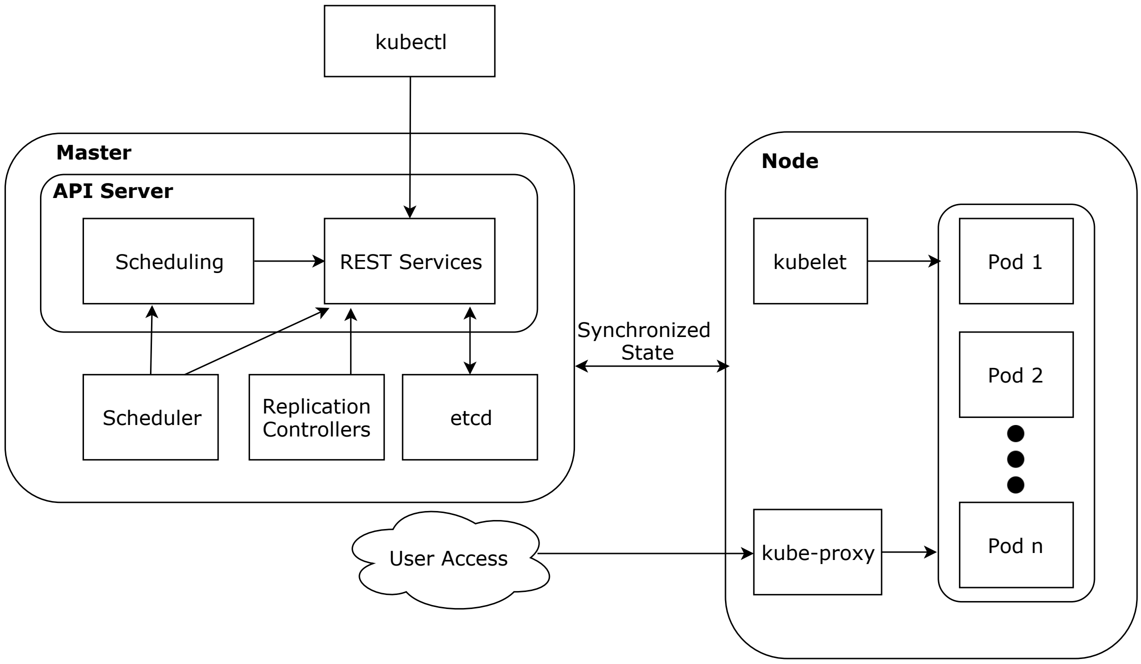

Figure 2.2: Kubernetes core architecture [Bai15]

etcd Etcd is a key-value store, accessible by a HTTP/JSON API, which can be distributed across multiple nodes and is used by Kubernetes to store configuration data, which needs to be accessible across nodes deployed in the cluster. It is essential for service discovery and to describe the state of the cluster, among other things. [Ell16]

Etcd can also watch values for changes [Bai15].

kube-apiserver The API server acts as the main management point for the cluster and provides a RESTful interface for users and other services to configure workloads in the cluster. It is a bridge between other master components and responsible of maintaining health and spreading commands in the cluster. [Ell16]

kube-scheduler The scheduler keeps track of available and allocated resources on each specific node in the cluster. It has an overview of the infrastructure environment and needs to distribute workload to an acceptable node without exceeding the available resources. Therefore each workload has to declare its operating requirements. [Ell16]

kube-controller-manager The controller manager mainly operates different controllers that constantly check the shared state of the cluster in `etcd` via the apiserver [Kubb] and if the current state differs towards the desired state it takes compensating measures [Ell16].

For example the node controller's task is to react when nodes go offline or down. The replication controller makes sure that the defined number of desired pods is identical to the number of currently deployed pods in the cluster and scales applications up or down accordingly. The endpoints controller populates the endpoints to services [Kubb]

cloud-controller-manager Kubernetes supports different cloud infrastructure providers. As each cloud provider has different features, apis and capabilities, cloud controller managers act as an abstraction to the generic internal Kubernetes constructs. This has the advantage that the core Kubernetes code is not dependent on cloud-provider-specific code. [Kubb]

Node Components

Servers that accomplish workloads are called nodes. Each workload is described as one or more container/s that has/have to be deployed. Node components run on every node in the cluster, providing the Kubernetes runtime environment [Kubb], that establishes networking and communicates with the master components. They also take care of deploying the necessary containers on a node and keep them running [Ell16]. Kubernetes requires a dedicated subnet for each node server and a supported container runtime [Kubb].

kubelet The kubelet is the primary agent running on each node in the cluster, responsible for running pods [Kubb]. It communicates with the API server to receive commands invoked by the scheduler. Interaction takes place with the etcd store to read and update configuration and state of the pod.

Pods are specified by the *PodSpec*, which defines the workload and parameters on how to run the containers [Ell16]. The kubelet process is responsible that the containers described in the specification are running and healthy [Kubb].

kube-proxy The proxy service is in charge of forwarding requests of defined services to the correct containers. On a basic level, load balancing is also done by the proxy. [Bai15]

Container Runtime The container runtime is an implementation running containers. Currently Docker, rkt, runc and OpenContainer runtimes are supported. [Kubb]

Pods A pod is the smallest deployable unit in a cluster consisting of a group of one or more containers, which share network and storage. [Kubc]

Addons

Cluster DNS Cluster DNS server keeps track of running services in the cluster and updates DNS records accordingly. This allows an easy way of service discovery. Containers include this DNS server in their DNS lookups automatically – that way a service can resolve another service by its name. [Bai15]

Ingress Ingress handles the traffic from outside the cluster and forwards it to the correct service using the dns service acting as a proxy server. Currently there are two official implementations: `ingress-gce` and `ingress-nginx`. *Ingress* also provides basic load balancing. [Kuba]

Dashboard The dashboard is a web-based user interface that allows to manage *Kubernetes* clusters and applications running in the cluster [Kubb]. It also provides access to log messages in each pod.

Minikube

Minikube is a tool to run a single-node Kubernetes cluster locally on computers supporting various virtual machine drivers.

2.3.2 Docker Swarm Mode

Docker Swarm Mode is the successor of *Docker Swarm* and implements a cluster management and orchestration tooling directly built into *Docker*[2] .

Components

Docker hosts can run in swarm mode, a swarm consists of one or more hosts that act as managers and workers. Hosts can be managers, which means delegators of work, or workers, that run services, or both. [Doc]

Service Services are definitions of tasks that will be executed on manager or worker nodes, specified by which container image to use and which commands to execute [Doc].

A service has attributes attached to it, that define its optimal state.

Services can be replicated, attached to storage and network resources and expose ports to the outside, defined by attributes. You can change the attributes while runtime, without restarting a service. [Doc]

Task A task is a running container itself which is assigned to the service. It is managed by the *swarm manager*. Manager nodes assign tasks to worker nodes, respecting the service scale.[Doc]

2 https://docker.com

Nodes A node is a Docker instance that is a participant in the the *swarm*. Nodes are typically distributed across multiple physical machines (in the cloud), but can also run on a single computer. [Doc]

Manager Nodes Manager nodes are responsible for deploying applications and dispatching tasks to worker nodes. Secondly one elected leader manager node supervises functions to maintain the desired state of the swarm (defined by the service). [Doc]

Worker nodes Worker nodes execute tasks from the manager nodes and notify them about the current state of its tasks.[Doc]

Load balancing & DNS Like Kubernetes, the swarm manager uses ingress to expose and load balance services.

An internal DNS component assigns each service a DNS entry automatically. [Doc]

Announcements

On October 17, 2017 at the conference *DockerCon*[3], Docker announced that it would integrate Kubernetes into the Docker platform.

2.3.3 Comparison

Community

The following table shows a comparison of publicly available metrics on *GitHub*, trying to represent community interest in the previously mentioned orchestrator softwares. Both projects are open-sourced and released under the *Apache-2.0*[4] license. These metrics were collected on June 21, 2018 and are rounded to the nearest ten. Comparing the numbers it

3 https://europe-2017.dockercon.com/
4 http://www.apache.org/licenses/LICENSE-2.0

can be assumed that the open-source community has currently a stronger interest in the Kubernetes project. In July 2018, Kubernetes won the *OSCON Most Impact Award* at the O'Reillys Open Source Conference [Boh].

	Kubernetes [5]	Docker Swarm Mode [6]
Contributors	1.700	165
Commits	66.820	3.530
Stars	37.830	5.150
Forks	13.220	1.060

Feature Differentiation

The handling of *Docker Swarm Mode* and Kubernetes is similar in many aspects, like load balancing with Ingress, service discovery via DNS, and the definition language YAML. Auto-scaling, which means increasing or decreasing running instances of a service as the load changes over time, is not directly available in *Docker Swarm Mode*, in contrast to Kubernetes.

Docker Swam Mode provides the possibility to mount local volumes or folders into a container. Kubernetes has two APIs available: Volumes and Persistent Volumes. Volumes are an abstraction with several different implementations for cloud storages (like AWS, Azure) and are bound to the lifecycle of a pod. Once a pod is removed, also the volume data is gone. Persistent Volumens allow data to be persisted independently from a pod.

Both technologies provide an easy to install development environment. Kubernetes is available via the minikube package as well as in the newest version of Docker Community Edition. Docker Swarm Mode is also available via the Docker application.

2.3.4 Decision

Taking into account the community size, the feature-richness and the out-of-the-box support by the major players Amazon AWS, Microsoft Azure and Google Cloud Engine, Kubernetes is the selected technology for the presented execution stack.

2.4 Machine Learning

The term *machine learning* originates from a 1959 article by Arthur Samuel [Sam59] presenting a method how computers can learn to play a better game of checkers than human.

Today, as a research area within artificial intelligence, machine learning is generally known as the process that trains computers to improve performance specific tasks through exposure to data, rather than through explicit programming by using statistical techniques. It is used to conceive complex models that lead themselves to prediction.

There are different approaches to machine learning, like decision trees or predicition rules [MCM13]. This thesis focuses on neural networks.

2.5 Neural Networks

Neural networks, or more correctly artificial neural networks, are derived from the neural system of the human brain. Neurons are the basic element of the nervous system and can be divided into three essential features: the dendrites, the soma and the axon. The human brain consists of about 10 billon neurons, which communicate through a network of axons and synapses. Artificial neural networks try to imitate this connection. [Hau98]

2.5.1 Classification of Neural Networks

This thesis focuses on Backpropagation networks, but to provide a general overview, Figure 2.3 shows a classification of neural networks by Haun [Hau98] divided into three levels based on connection type, neuronal behavior and learning methods. The first level classifies into feedback and feedforward networks. The second level into linear and non-linear networks, the third one into supervised and non-supervised networks.

Feedforward networks

Feedforward networks consist of connections in one direction only. Neurons are connected between different layers and normally spread from input to output through hidden layers. The output can be calculated from the input directly. Loops are not allowed [Hau98].

Linear and non-linear networks Linear networks use linear activation functions. The output of a neuron is bound directly to the value of activation function. Non-linear networks use non-linear activation functions, which means that the activation value is one, if the sum of all input values exceed a threshold value, otherwise it is zero. [Hau98]

A Perceptron network is for example a linear network.

Supervised and unsupervised networks Supervised networks compare its output values with the correct answer during training and continuously adapt input values to approximate both values. Unsupervised networks do not have such information and must learn through an inherent mechanism. It is presumed that impulses and reactions relate in a way that produces correct behaviour after the learning period is finished. [Hau98]

Feedback networks

Feedback networks are networks where neurons are also connected between different layers, but the output of a neuron can be connected to an input of another neuron. Output values are therefore dependant from previous input values of the neural network. [Hau98] Feedback networks are further grouped into defined constructed networks and trained networks. Defined constructed networks already have a defined structure when the data is presented, for example Hopfield networks. Trained networks can be trained

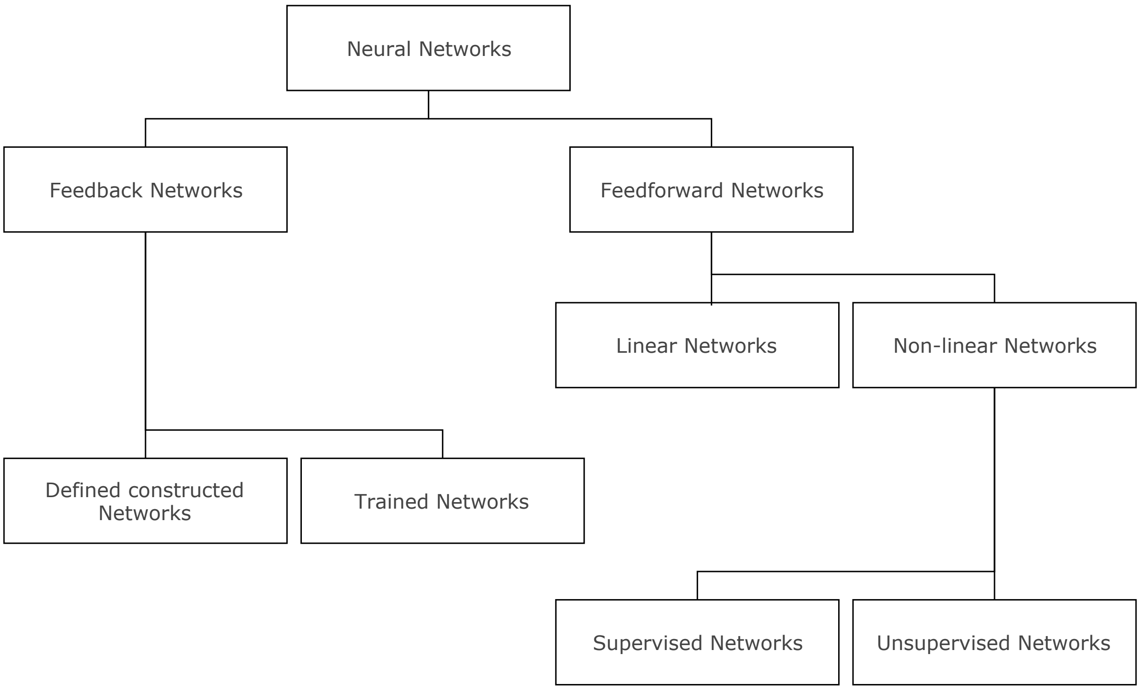

Figure 2.3: Classification of neural networks by Haun [Hau98]

with supervised and unsupervised training, for example the ART-1 (Adaptive Resonance Theory). [Hau98]

2.5.2 Backpropagation Networks

Backpropagation is not a network design per se, but a supervised learning algorithm. It is used for example in Multi-Layer-Perceptrons. The purpose is to change weights on hidden layers in the network, based on a calculated (net) output error, to improve network accuracy. [Frö]

An input vector is forward-propagated through all layers until the output layer, which is then compared to the desired output. This step results in the error values using a loss function. In process of backpropagation the weights are then updated to minimize the loss function. The process is repeated until the net error is approximately zero. [Frö]

2.5.3 Neural Network Frameworks

Neural network frameworks provide an abstraction and simplification to complex programming challenges [Mak18] regarding neural network models and the simulation of the training and evaluation processes. Developers are given helper functions to build a network according to their liking. Most frameworks also provide implementations of the backpropagation algorithm, activation functions and data structures to load training data into memory. Frameworks can also help to transform raw data, like images, into data that is more suitable to neural network training.

There are currently many popular neural network frameworks on the market. Google's TensorFlow is having the biggest impact in terms of contributions and community (see Comparison).

TensorFlow

TensorFlow[7] is an open-sourced framework, released under the Apache 2.0[8] license, that was developed by the *Google Brain Team* as successor to the proprietary software DistBelief, which was used for research on various use cases including unsupervised learning, image classification, object detection and many more. TensorFlow is an interface and implementation for machine learning algorithms, featuring support for a wide range of devices (from mobile phones to server hardware) and GPU cards. [Dea15]

According to Google's whitepaper [Dea15], in their implementation, a *tensor* is a typed, multidimensional array, that supports a variety of tensor element types . It can be seen as abstraction to scalars, vectors and matrices [RM17].

TensorFlow computations are expressed as directed dataflow graph, in which each node has zero or more inputs and outputs. Values, called tensors, flow along the edges of the graph. [Dea15]. Figure 2.4 shows a matrix multiplication of input X with a matrix W. Vector b is then added to this matrix terminating in output O.

These computations can be executed on either on local or the distributed implementation, on a single or multiple devices [Dea15].

7 https://github.com/tensorflow/tensorflow
8 http://www.apache.org/licenses/LICENSE-2.0

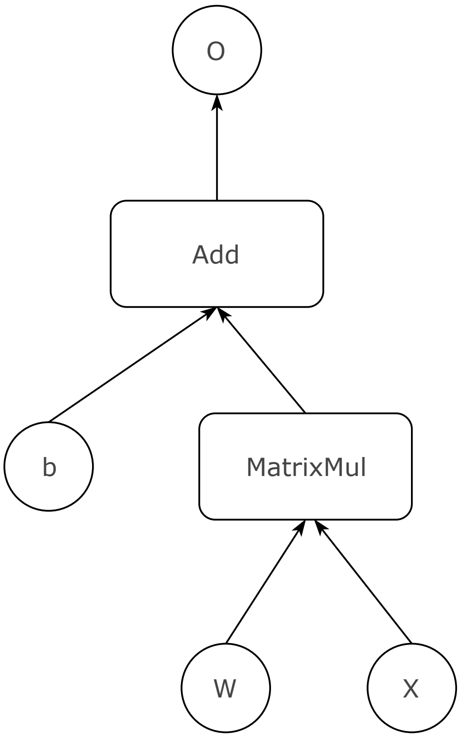

Figure 2.4: Tensor flow computation graph, adapted from [Dea15]

TensorFlow Programming Stack The TensorFlow programming stack consists of multiple API layers, as is illustrated in Figure 2.5. On the lowest level, the TensorFlow Kernel, is the distributed execution engine. The low-level APIs are implemented in different programming languages, including Python, C++, Java and Go. Currently only Python provides higher-level TensorFlow APIs.

The mid-level API provides access to layers, datasets and metrics, the high-level API adds estimators, which encapsulate training, evaluation and prediction of models [Tena].

Implementation Example For framework demonstration purposes, the source code classifying the Iris dataset from the first use case (see section 8.1) is implemented using TensorFlow. For better understanding of the TensorFlow syntax and functionality, this commented code example[9], written by the TensorFlow authors, is pointed out.

In the first step the program features a parser for the Iris dataset and defines feature columns. The TensorFlow `NetworkClassifier` class is then instantiated building a neural

9 https://github.com/tensorflow/models/blob/v1.9.0/samples/core/get_started/premade_estimator.py

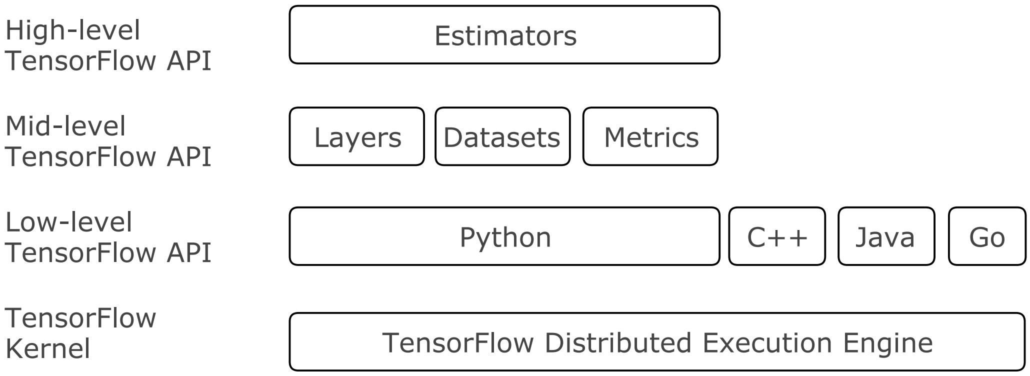

Figure 2.5: TensorFlow Programming Stack, adapted from [Tenb]

network with two hidden layers of ten nodes each. The classifier offers a function for network training called `train()`, which is followed by evaluating the accuracy of the trained network in the final step.

```
#  Copyright 2016 The TensorFlow Authors. All Rights Reserved.
#
#  Licensed under the Apache License, Version 2.0 (the "License");
#  you may not use this file except in compliance with the License.
#  You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
#  Unless required by applicable law or agreed to in writing, software
#  distributed under the License is distributed on an "AS IS" BASIS,
#  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#  See the License for the specific language governing permissions and
#  limitations under the License.
"""An Example of a DNNClassifier for the Iris dataset."""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import tensorflow as tf
```

```
import iris_data


parser = argparse.ArgumentParser()
parser.add_argument('--batch_size', default=100, type=int, help='batch size')
parser.add_argument('--train_steps', default=1000, type=int,
                    help='number of training steps')

def main(argv):
    args = parser.parse_args(argv[1:])

    # Fetch the data
    (train_x, train_y), (test_x, test_y) = iris_data.load_data()

    # Feature columns describe how to use the input.
    my_feature_columns = []
    for key in train_x.keys():
        my_feature_columns.append(tf.feature_column.numeric_column(key=key))

    # Build 2 hidden layer DNN with 10, 10 units respectively.
    classifier = tf.estimator.DNNClassifier(
        feature_columns=my_feature_columns,
        # Two hidden layers of 10 nodes each.
        hidden_units=[10, 10],
        # The model must choose between 3 classes.
        n_classes=3)

    # Train the Model.
    classifier.train(
        input_fn=lambda:iris_data.train_input_fn(train_x, train_y,
                                                 args.batch_size),
        steps=args.train_steps)

    # Evaluate the model.
    eval_result = classifier.evaluate(
```

```
        input_fn=lambda:iris_data.eval_input_fn(test_x, test_y,
                                                args.batch_size))

    print('\nTest set accuracy: {accuracy:0.3f}\n'.format(**eval_result))

    [...]


if __name__ == '__main__':
    tf.logging.set_verbosity(tf.logging.INFO)
    tf.app.run(main)
```

Deeplearning4J

Deeplearning4J is an open-source machine learning library by the Eclipse Foundation released under the Apache 2.0 license. It provides a set of components, to read from various data sources and build neural networks. Deeplearning4J provides support for CPU and GPU processing. [Met14]

For large processing, the framework supports Hadoop, a way of scaled data storing and processing across numerous computer servers [Met14].

ND4J The framework is built on top of ND4J, a numerical computing engine, which implements n-dimensional array objects (tensors) for Java [ND4a]. It is a library, that is optimized for GPU processing with a CUDA backend and supports all common operations to manipulate matrices [ND4a]. Parts of core are written in C++ to increase performance of numerical operations [ND4b].

Main Features The framework supports convolutional and recurrent nets and deep nets of various types. Furthermore it provides implementation of backpropagation and optimization algorithms, various activation- and loss functions as well as hyperparameters. [DL4]

The user interface features a computation graph and visualization tools, further explained in section 8.1.6.

Implementation Example For framework demonstration purposes, the source code classifying the Iris dataset from the first use case (see section 8.1) is implemented using Deeplearning4J.

For better understanding of the Deeplearning4J syntax and functionality, this commented code example[10], written by the Adam Gibson and released under Apache License Version 2.0, is pointed out.

```
/**
 * @author Adam Gibson
 */
public class CSVExample {

    private static Logger log = LoggerFactory.getLogger(CSVExample.class);

    public static void main(String[] args) throws  Exception {

        //First: get the dataset using the record reader. CSVRecordReader handles
        loading/parsing
        int numLinesToSkip = 0;
        char delimiter = ',';
        RecordReader recordReader = new CSVRecordReader(numLinesToSkip,delimiter);
        recordReader.initialize(new FileSplit(new
        ClassPathResource("iris.txt").getFile()));

        //Second: the RecordReaderDataSetIterator handles conversion to DataSet
        objects, ready for use in neural network
        int labelIndex = 4;     //5 values in each row of the iris.txt CSV:
        4 input features followed by an integer label (class) index.
        Labels are the 5th value (index 4) in each row
```

10 https://github.com/deeplearning4j/dl4j-examples/blob/master/dl4j-examples/src/main/java/org/deeplearning4j/examples/dataexamples/CSVExample.java

```
int numClasses = 3;     //3 classes (types of iris flowers) in the
iris data set.
Classes have integer values 0, 1 or 2
int batchSize = 150;    //Iris data set: 150 examples total. We are
loading all of them into one DataSet (not recommended for large data sets)

DataSetIterator iterator = new
RecordReaderDataSetIterator(recordReader,batchSize,labelIndex,numClasses);
DataSet allData = iterator.next();
allData.shuffle();
SplitTestAndTrain testAndTrain = allData.splitTestAndTrain(0.65);
//Use 65% of data for training

DataSet trainingData = testAndTrain.getTrain();
DataSet testData = testAndTrain.getTest();

//We need to normalize our data. We'll use NormalizeStandardize (which
gives us mean 0, unit variance):
DataNormalization normalizer = new NormalizerStandardize();
normalizer.fit(trainingData);           //Collect the statistics (mean/
                                                                 stdev)
from the training data. This does not modify the input data
normalizer.transform(trainingData); //Apply normalization to
                                      the training data
normalizer.transform(testData);  //Apply normalization to
                                      the test data.
This is using statistics calculated from the *training* set


final int numInputs = 4;
int outputNum = 3;
long seed = 6;


log.info("Build model....");
```

```
        MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
            .seed(seed)
            .activation(Activation.TANH)
            .weightInit(WeightInit.XAVIER)
            .updater(new Sgd(0.1))
            .l2(1e-4)
            .list()
            .layer(0, new DenseLayer.Builder().nIn(numInputs).nOut(3)
                .build())
            .layer(1, new DenseLayer.Builder().nIn(3).nOut(3)
                .build())
            .layer(2, new
            OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
                .activation(Activation.SOFTMAX)
                .nIn(3).nOut(outputNum).build())
            .backprop(true).pretrain(false)
            .build();

        //run the model
        MultiLayerNetwork model = new MultiLayerNetwork(conf);
        model.init();
        model.setListeners(new ScoreIterationListener(100));

        for(int i=0; i<1000; i++ ) {
            model.fit(trainingData);
        }

        //evaluate the model on the test set
        Evaluation eval = new Evaluation(3);
        INDArray output = model.output(testData.getFeatureMatrix());
        eval.eval(testData.getLabels(), output);
        log.info(eval.stats());
    }

}
```

Further Neural Network Frameworks

There are several other popular neural network frameworks [Mak18], that should be mentioned here. The following table lists this projects including their website.

Framework Name	Projekt Website
Caffe	http://caffe.berkeleyvision.org
Microsoft Cognitive Toolkit/CNTK	https://www.microsoft.com/en-us/cognitive-toolkit
PyTorch	https://pytorch.org
Keras	https://keras.io/

Comparison

Community Statistics (like *stars*, *contributors* and *forks*) of open-source projects hosted on the version control platform GitHub, increasingly influence the community and other developers. Every user on GitHub can show interest in a project by giving it a star, or copy the complete source code (a fork). Programmers that contributed code to a project are called contributors. In a blog article[11], the founder of a machine learning framework called *Leaf* announced the suspension of the development. The announcement featured a screenshot that compared Leaf to TensorFlow by the amount of stars on GitHub.

TensorFlow is currently the leading framework in terms of stars and contributors. Backed by Google, TensorFlow is fully integrated into the Google Cloud and Android platform. It has also been adopted by several large companies, like IBM, Twitter and Airbus [Mak18]. Furthermore tech blogs rather report on TensorFlow than other framworks. A Google search for TensorFlow on the popular tech blog *DZone.com* returns over 5.300 results, while Deeplearning4J got less than 500 (as of July 17, 2018). The following table compares the statistics of the two mentioned projects on GitHub.

11 https://medium.com/@mjhirn/tensorflow-wins-89b78b29aafb

	TensorFlow [12]	Deeplearning4J [13]
Contributors	1.500	225
Commits	36.450	23.260
Stars	105.120	9.320
Forks	65.410	4.350

2.5.4 Decision

ConbexNN will be implemented in the Java programming language. Deeplearning4J is based on C/C++ with a Java based library on top, while TensorFlow is based on Python. Deeplearning4J has a clear and understandable framework architecture, broad support for machine learning algorithms and data transformation. It further offers a model import tool for TensorFlow models. From the author's point of view, the source code of Deeplearning4J programs is cleaner and easier to interpret, as well as the transformation from the ViNNSL to Deeplearning4J is preferable to alternatives.

As part of future work it is desirable to implement a native TensorFlow worker service in addition to Deeplearning4J.

3 Requirements

This section defines functional and non-functional requirements for ConbexNN. The neural network execution stack focuses on two main target groups: data scientists and developers.

Data scientists use the provided services in a deployed environment (cloud or own computer) to develop and train their neural networks. The system should be easy to setup and no programming knowledge should be needed to get started.

Developers can extend the neural network stack with features or use the provided web services to implement their own custom solution.

3.1 Functional Requirements

Due to the fact that neural network training requires a lot of computing power, the main requirement is to design an architecture that can be executed in the cloud or on-site cluster hardware.

To enable developers to extend the application, it is designed as a platform that is open-sourced and documented. The easy setup on local computers and the micro-services, that feature a clear structure and manageable code base, make it easier to get acquainted with the architecture.

The neural network platform should also offer a way to be extended or used by external applications and services, therefore a documented RESTful webservice is provided, that can be consumed by various clients.

3.1.1 User Interface

The user interface shall be a web application that gives a quick overview of all neural networks and their training status. The frontend uses the RESTful API as backend source and does not cover the whole function range of the API.

Mockup

Figure 3.1 shows a sketch of the user interface. On the left side the user can see a list of all created or imported neural networks. Next to the names of the networks, there is an icon representing the training status. In the detailed view on the right side, the title and id of the network is shown followed by an indicator when training is in progress. The visualisation of a neural network is divided into tabs.

The tabs “Description”, “Definition”, “Instance” and “Result” represent the eponymous ViNNSL Description XML file into a graphical tree view. When enough information is provided by ViNNSL XML files, the worker service performs a transformation into the internally used model representation of the *Deeplearning4J* Framework. The *Deeplearning4J* Tab shows the transformed object. In the “Files” tab, imported files of the storage services are listed and can be selected as training- or testset.

3.2 Non-Functional Requirements

3.2.1 Quality

The execution stack shall comply with the following quality features:

- Standard RESTful API
- the user interface works on all common browsers and devices (responsive design)
- loading time of the user interface should be less than three seconds

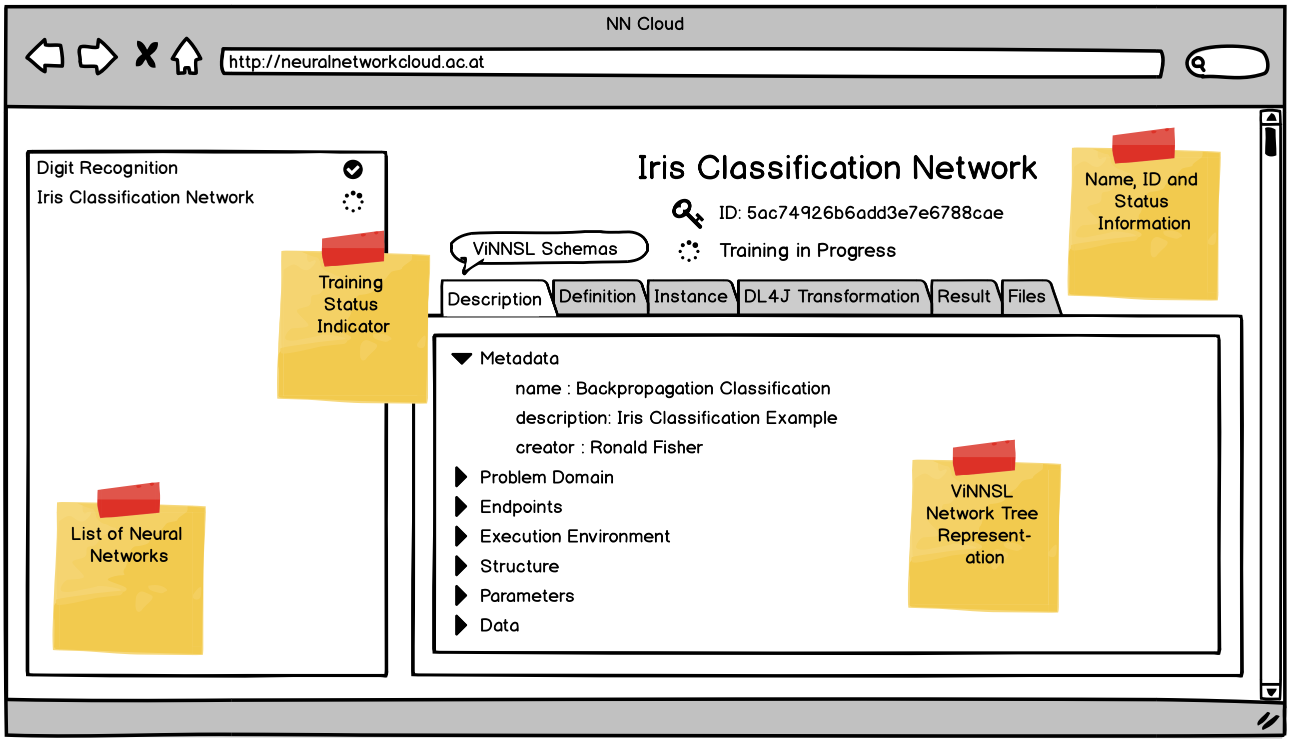

Figure 3.1: Mockup: User Interface of Frontend Service

3.2.2 Technical

3.2.3 Software

- Kubernetes
- Docker
- Java Standard Platform
- Maven Plugin for Java

3.2.4 Hardware

- Kubernetes compatible hardware or Cloud account (Amazon Web Services, Google Cloud Engine)

3.2.5 Documentation

The documentation is provided in Section 6 or online on SwaggerHub[1].

3.2.6 Source Code

The source code is released on GitHub [2].

3.2.7 Developer Environment

Developers can use any Java Based development environment.

1 https://app.swaggerhub.com/apis/a00908270/
2 https://github.com/a00908270/

4 Specification

4.1 Use Case

Figure 4.1 shows the UML use case diagram.

4.1.1 Use Case Descriptions

Use Case	**Import Neural Network**
Description	An existing ViNNSL XML file with a neural network description is imported via the vinnsl web service into the database.
Priority	primary
Actors	Data Scientist
Preconditions	ViNNSL neural network XML description file
Postconditions	—
Normal Course of Events	* The actor sends a POST request to the ViNNSL web service including a XML body* The web service validates and imports the XML file and returns the HTTP status code 201 CREATED
Alternative Courses	* The post request is sent by an application or other service
Exceptions	If the validation fails or an error occurs, the web service returns the HTTP status code 500
Assumptions	Access to the `vinnsl-service`

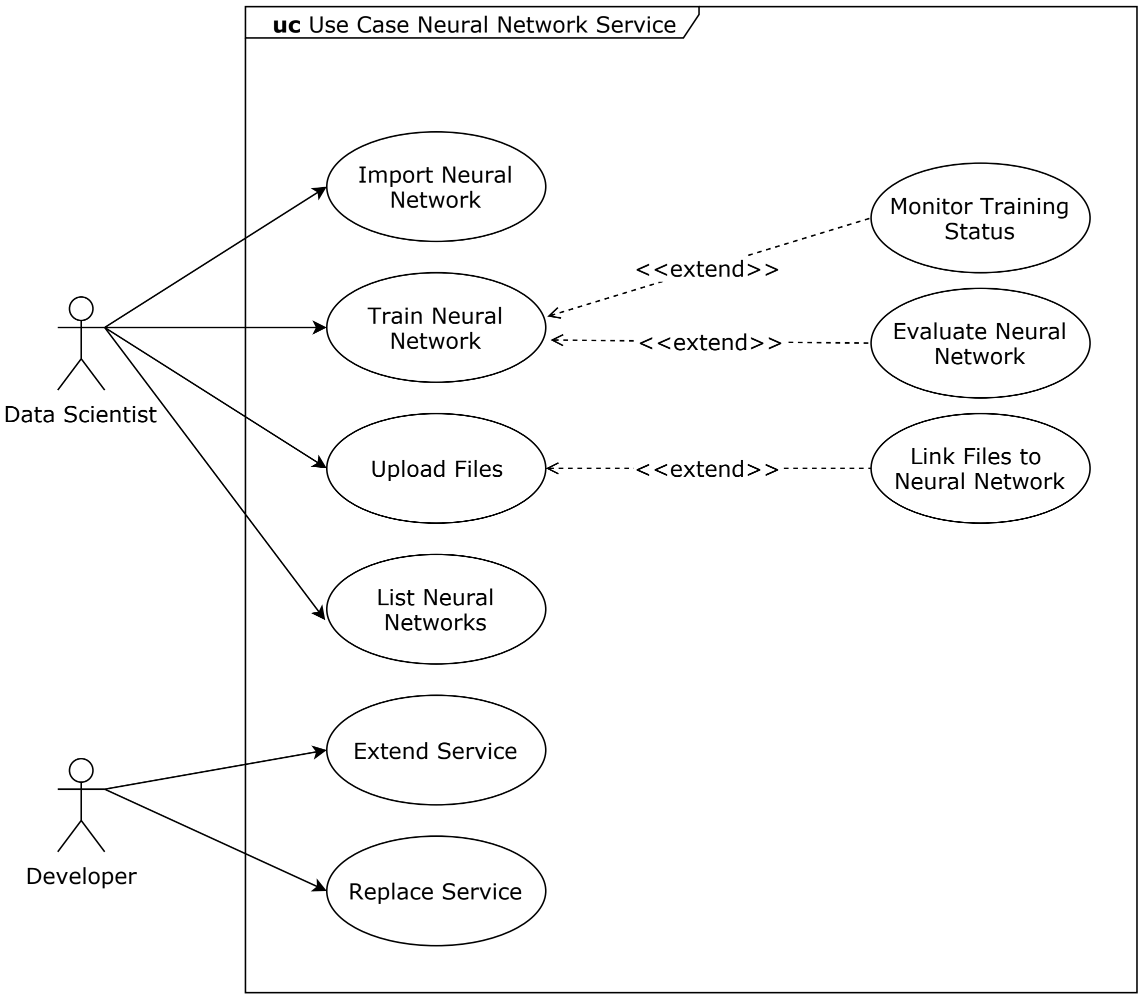

Figure 4.1: UML Use Case Diagram

Use Case	Train Neural Network
Description	An imported neural network is trained by passing the configuration over to the worker service.
Priority	primary
Actors	Data Scientist
Preconditions	Imported ViNNSL neural network XML description, definition and instance file
Postconditions	—
Normal Course of Events	* The actor sends a POST request to the working service including the identifier of the neural network that should be trained* The webservice validates the request, adds the network into the training queue and returns the HTTP status code 200.
Alternative Courses	* The post request is sent by an application or other service
Exceptions	If the validation fails or an error occurs, the webservice returns the HTTP statuscode 500
Assumptions	Access to the `vinnsl-nn-worker`
Extensions	* Monitor Training Status * Evaluate Neural Network

Use Case	Monitor Training Status
Description	The Data Scientist monitors the training status to evaluate the trained network afterwards.
Priority	secondary
Actors	Data Scientist
Preconditions	Training of neural network started
Postconditions	—

Use Case	Monitor Training Status
Normal Course of Events	* The actor sends a GET request to the status endpoint of the vinnsl service including the identifier of the neural network that is in progress.* The web service validates the request, and returns the training status along the HTTP status code 200.
Alternative Courses	* The post request is sent by an application or other service
Exceptions	If the validation fails or an error occurs, the web service returns the HTTP statuscode 500
Assumptions	Access to the `vinnsl-service`
Extensions	—

Use Case	Evaluate Neural Network
Description	The Data Scientist evaluates the accuracy of the network after its training
Priority	primary
Actors	Data Scientist
Preconditions	Training of neural network successfully finished
Postconditions	—
Normal Course of Events	* The actor sends a GET request to the status endpoint of the vinnsl service including the identifier of the neural network that is finished.* The web service validates the request, and returns the ViNNSL XML file including the result scheme.
Alternative Courses	* The post request is sent by an application or other service
Exceptions	If the validation fails or an error occurs, the webservice returns the HTTP statuscode 500
Assumptions	Access to the `vinnsl-service`
Extensions	—

Use Case	Upload Files
Description	The Data Scientist uploads files, that are usable as datasets (f.ex. CSV files or pictures) to the storage service
Priority	primary
Actors	Data Scientist
Preconditions	—
Postconditions	—
Normal Course of Events	* The actor sends a POST request to the storage service endpoint containing a multipart file.* The web service validates the request, and returns the unique identifier of the file along the HTTP status code 200.
Alternative Courses	* The post request is sent by an application or other service* The file is uploaded with the provided HTML upload form provided by the storage service
Exceptions	If the upload fails or an error occurs, the web service returns the HTTP statuscode 500
Assumptions	Access to the `vinnsl-storage-service`
Extensions	—

Use Case	**List Neural Networks**
Description	Imported neural networks are listed
Priority	primary
Actors	Data Scientist
Preconditions	Imported ViNNSL neural network XML description file
Postconditions	—
Normal Course of Events	* The actor sends a GET request to the ViNNSL web service optionally including a neural network identifier* The web service validates and returns the XML file(s).
Alternative Courses	The request is sent by an application or other service

Use Case	List Neural Networks
Exceptions	If the validation fails or an error occurs, the web service returns the HTTP statuscode 500
Assumptions	Access to the `vinnsl-service`

Use Case	Extend Service
Description	An existing micro service can be extended by developers
Priority	secondary
Actors	Developer
Preconditions	source code and developer environment present
Postconditions	—
Normal Course of Events	* The developer downloads the source code and extends functionality of a micro service* The modified service is deployed into kubernetes
Alternative Courses	—
Exceptions	—
Assumptions	—

Use Case	Replace Service
Description	An existing micro service can be replaced by developers
Priority	secondary
Actors	Developer
Preconditions	source code and developer environment present
Postconditions	—
Normal Course of Events	* The developer writes a new implementation of an existing service respecting the API definition (see API Docuentation)* The service is deployed into kubernetes
Alternative Courses	—

Use Case	Replace Service
Exceptions	—
Assumptions	—

4.2 Sequence Diagram

Figure 4.2 shows the sequence diagram of a neural network training process and which microservices are involved in the communication. The *vinnsl service* is the main communication hub that enables access to the neural network object and all of its data and also provides interfaces to update it. The *vinnsl storage service* most importantly stores necessary binary data used by the neural network objects. On one hand that are tables and pictures on the other hand the binary (trained) *Deeplearning4J* model. The *vinnsl worker service* has the role of training the neural network models.

4.2.1 Sequence of Training

New neural networks are created by sending a `POST` request including a XML ViNNSL network description in the request body. The *vinnsl service* creates a new neural network based on the definition and answers with the HTTP status code 201 (CREATED). The location header points to the URL where the created network can be retrieved. The URL contains the unique identifier. Using this identifier the next step is to add the ViNNSL definition XML file to the network. This is done via a `POST` request appending the id and the `/definition` endpoint. The XML file is placed in the request body. Resources that are required for the training (like the training set) need to be uploaded to the storage service, which returns a unique file id. Before the training can start, the training set needs to be linked to the neural network. This is possible with the `/addfile` endpoint.

Next, the network is marked for training by calling the worker service with its identifier. The worker service confirms that the training is queued. As soon as the training is finished, the worker service updates the neural network object with the result schema and uploads the trained binary model to the storage service for retraining.

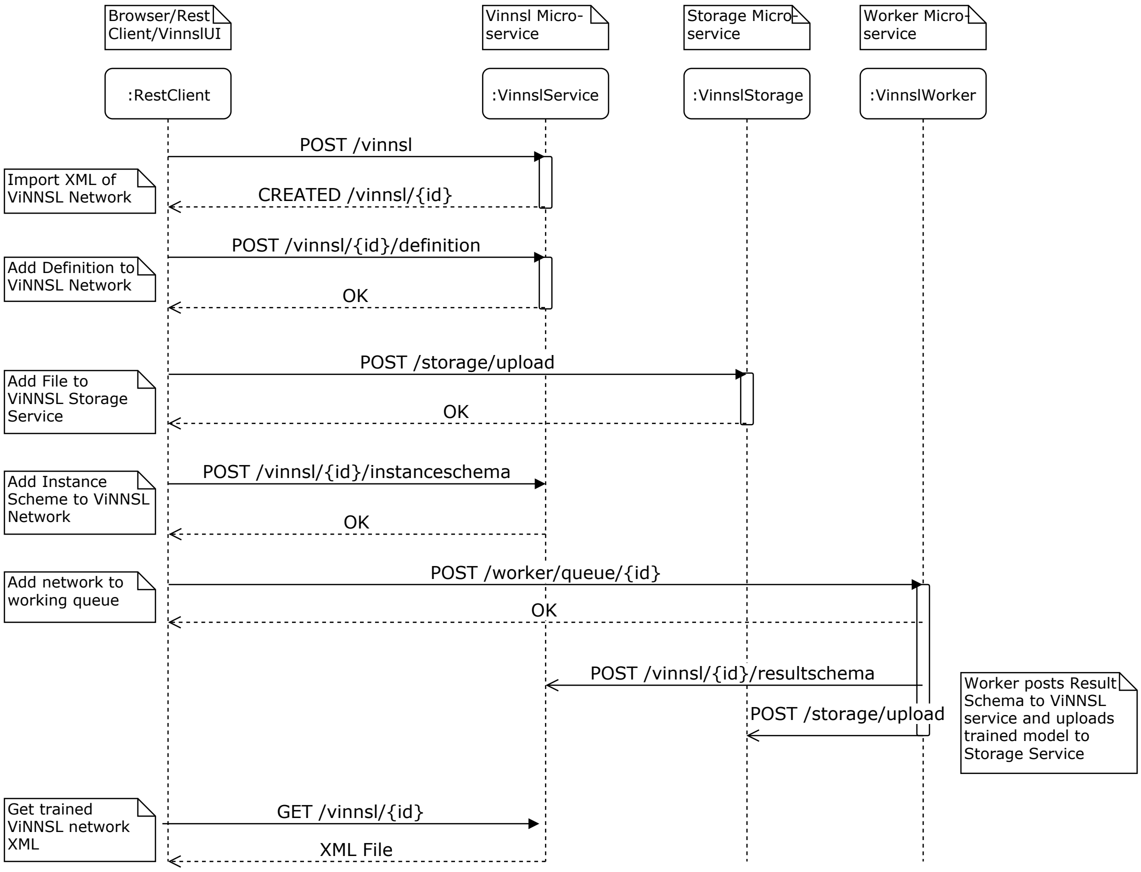

Figure 4.2: Training Sequence Diagram

A simple `GET` request to the vinnsl service along with the identifier returns the current trained neural network model.

4.3 Data Model Design

4.3.1 vinnsl-service

All neural network data managed by the `vinnsl-service` is stored in a documented-oriented database. The saved documents will internally be mapped to Java Classes. The main object is `vinnsl`.

`vinnsl` is the primary object owning the `_id` field that is unique. The `nncloud` property stores the status of the network and the representation of the transformed *Deeplearn-*

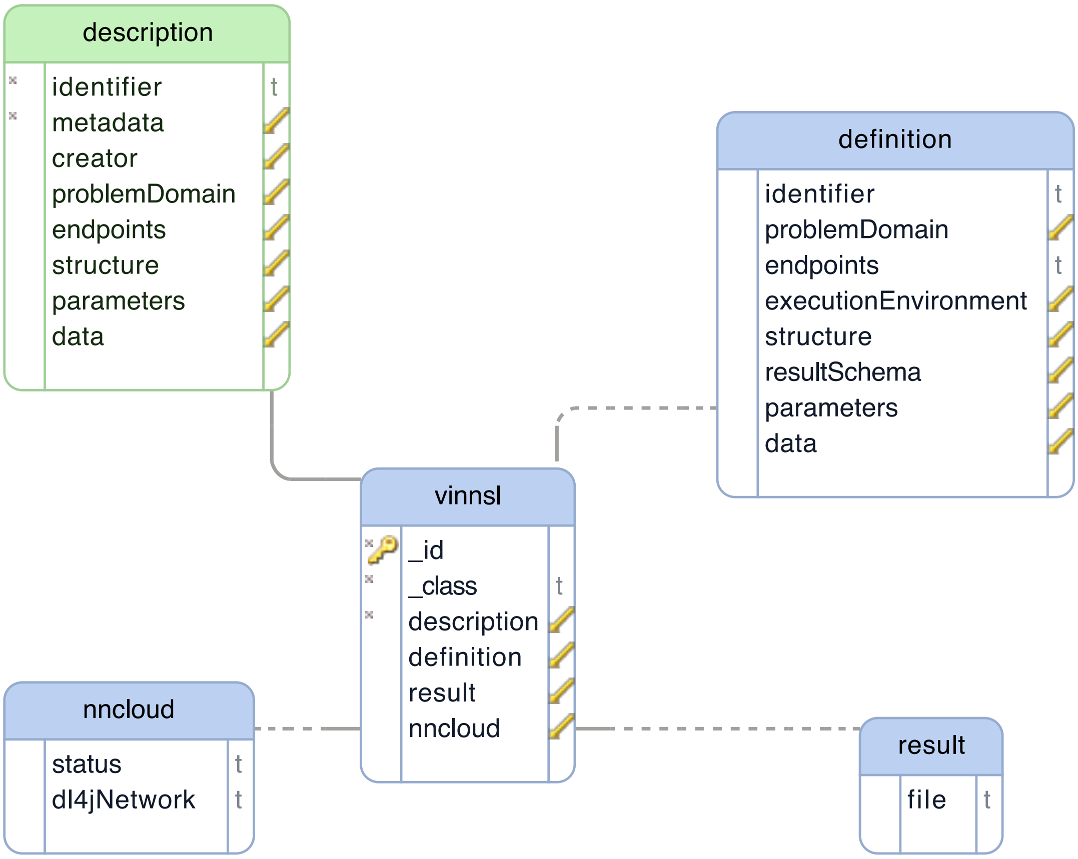

Figure 4.3: NoSQL Data Model

ing4J network. `description`, `definition`, `instance`, `training` and `result` represent the ViNNSL 2.0 Schema, generated from the provided XML Schema Definition files. See [Kop15] to get a listing and description on all provided properties of ViNNSL 2.0.

Figure 4.3 shows the data schema.

4.3.2 storage-service

The `storage-service` stores binary files and their metadata, either directly in the file system or inside a database. Each file needs to have a unique id, a filename, a content type and an upload date.

Attribute field	Description
id	a unique file id that can be referred to (f.ex in `vinnsl-service`)
filename	the original filename when uploaded
content type	the MIME type standardized in RFC 6838 (f.ex text/plain)
upload date	date and time of original upload
metadata	a field for arbitrary additional information

Example of stored file:


```
{
    "_id" : ObjectId("5ab4e69c8f136a16bf81f093"),
    "filename" : "iris.txt",
    "aliases" : null,
    "chunkSize" : NumberLong(261120),
    "uploadDate" : ISODate("2018-03-23T11:35:56.700Z"),
    "length" : NumberLong(2700),
    "contentType" : "text/plain",
    "md5" : "f0e89bd71f7bb9e584e685aeb178a5aa"
}
```


4.4 Overview Microservices

The neural network cloud execution stack consists of four main services that expose a RESTful API to users and two supporting services in charge of persisting data. Figure 4.4 displays an overview of the service architecture, including the exposed endpoints and storage backends.

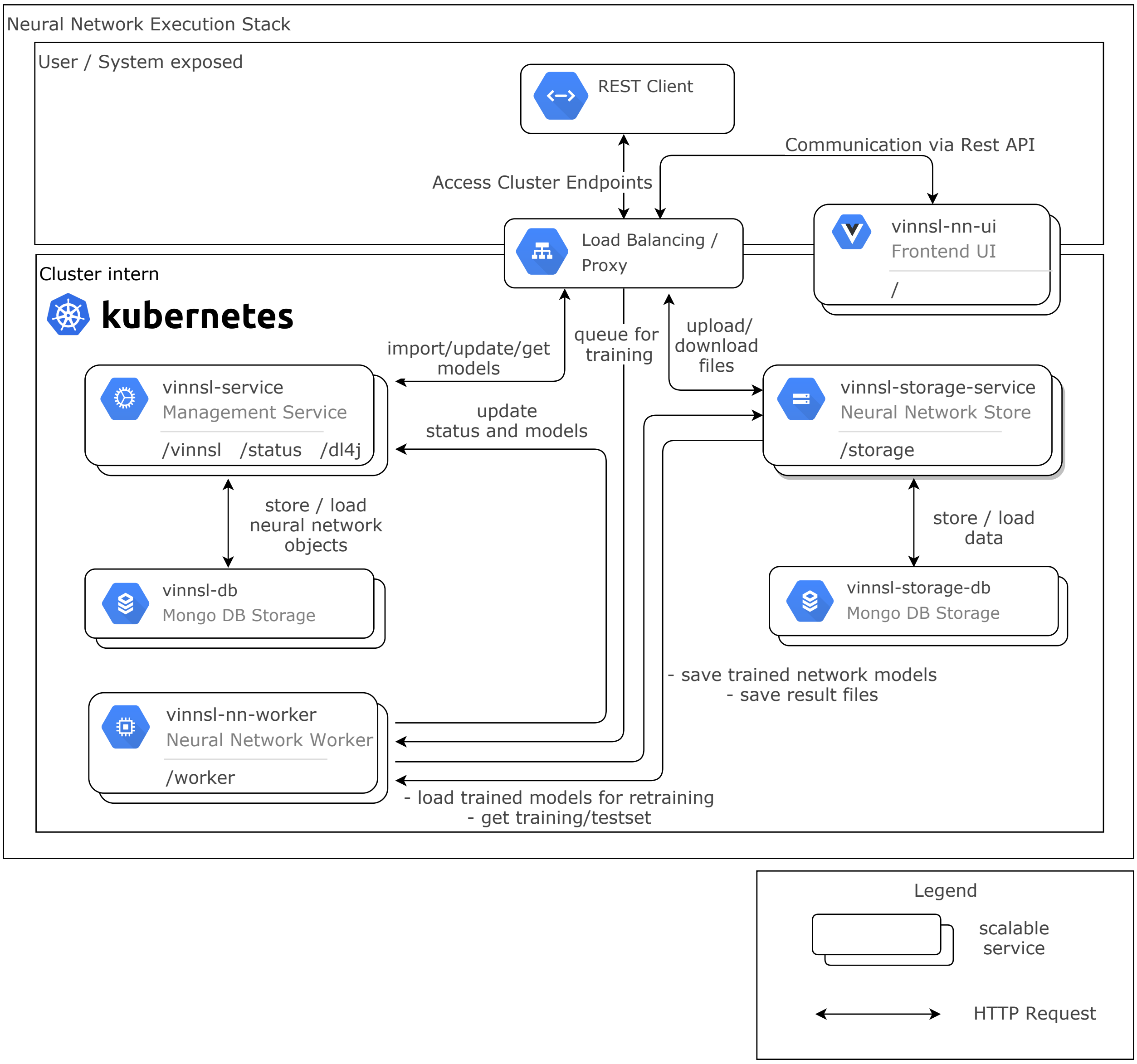

Figure 4.4: Architectural Overview of the Neural Network Stack

4.4.1 Vinnsl Service (vinnsl-service)

The `vinnsl-service` is responsible for handling the import, management and manipulation of neural network objects and their status. It maps the CRUD[1] operations to HTTP methods. A new neural network is created by sending a `POST` request to the `/vinnsl` endpoint containing a ViNNSL Definition XML as body. Sending a `GET` request to the `/vinnsl` route returns a JSON containing all ViNNSL neural network objects.

The `vinnsl-service` depends on the `vinnsl-db` service, which runs a MongoDB database to store the objects.

4.4.2 Worker Service (vinnsl-nn-worker)

The `vinnsl-nn-worker` implements a queue management for neural network training and transforms ViNNSL neural network models into *Deeplearning4J* models. It provides a wrapper of the *Deeplearning4J* platform, that handles the training or evaluation of the network.

4.4.3 Storage Service (vinnsl-storage-service)

Binary files, like trained network models, images or csv files are essential in the pocess of creating and training neural networks. File management is handled by the `vinnsl-storage-service`.

4.4.4 Frontend UI (vinnsl-nn-ui)

The Frontend UI is a web application that gives a brief overview of all neural network models, their training status and linked files.

1 Create, Read, Update, Delete

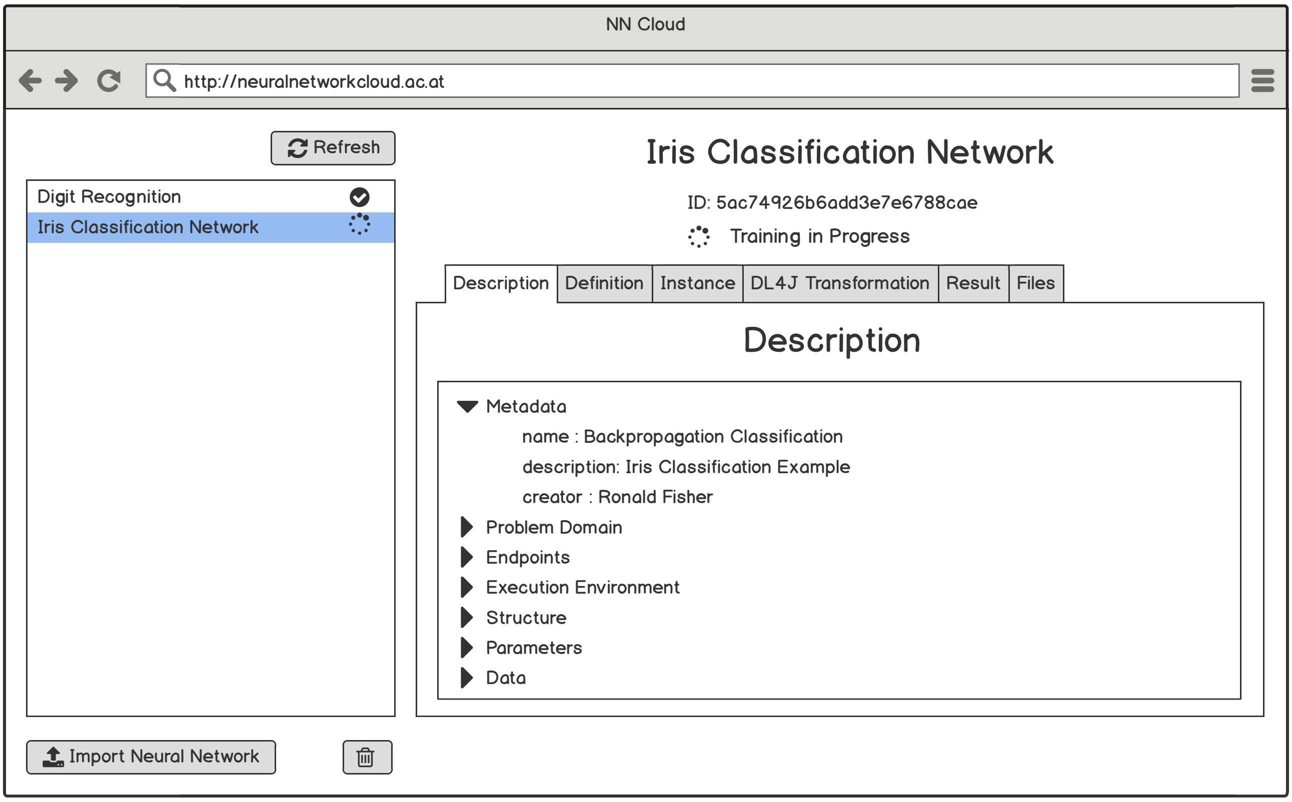

Figure 4.5: User Interface Design for vinnsl-nn-ui

4.5 User Interface Design

Based on the mockup in section 3.1.1, a user interface design has been created, that will later be implemented as a web application. Buttons to import or delete a neural network and to refresh the user interface have been added to the design.

Figure 4.5 shows the user interface design for the frontend web service.

4.6 Service Discovery and Load Balancing

Service Discovery is the process of finding a way to connect to a specific service. This applies within the cluster, which is typically firewalled from the internet. As Kubernetes allows services to be scaled, there is also a logic that knows and decides how network traffic is routed. This is called *Load Balancing*. Figure 4.6 shows an overview of the microservices, their endpoint URL and the domain name service. External access to specific services is managed by *Ingress*.

4.6.1 Kubernetes DNS-based Service Discovery

`kube-dns` is the Kubernetes add-on that starts a pod with a DNS service and configures the kubelets to resolve DNS names over this service. It listens on port 53, the standard DNS port. Services in a cluster are assigned a *DNS A record* derived from their service metadata name specified in the *ServiceSpec*. [Kubd]

The following code snippet is an extract of the *ServiceSpec* for the `vinnsl-service` defining the metadata name:

```
{
  "kind": "Service",
  "apiVersion": "v1",
  "metadata": {
    "name": "vinnsl-service",
    ...
  }
}
```

Structure of the Hostname

The full hostname record is composed of the zone, kind, namespace of the cluster and the metadata name of the service.

Name	Description
zone	the cluster domain (default using minikube: *cluster.local*)
kind	kind of pod (default for services: *svc*)
ns	namespace (default using minikube: *default*)
hostname	hostname from service metadata name

Example The vinnsl-service running on a local minikube cluster gets the following DNS record name: `vinnsl-service.default.svc.cluster.local`.

Service Discovery

Using the Kubernetes DNS, a microservice instance (kubelet) can now lookup other services by using DNS Queries.

Example For example the tool `nslookup` can query the DNS service for the IP address of the `vinnsl-service` within the cluster.

```
/ # nslookup vinnsl-service
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      vinnsl-service
Address 1: 10.102.84.122 vinnsl-service.default.svc.cluster.local
```

In this example the service is reachable at the IP address *10.102.84.122*.

External Access and Load Balancing

External access from outside the cluster to specific services is managed and provided through the *Ingress* API object. The associated implementation is called *Ingress controller* and is obligatory. Currently there are two official implementations: `ingress-gce` and `ingress-nginx`. [Kuba]

Minikube runs the `ingress-nginx` implementation as default and also provides basic load balancing by configuring a `nginx` [2] web server. Kubernetes configures `nginx` to use the *least-connected* load balancing mechanism, which means that the *next request is assigned to the server with the least number of active connections* [ngi].

2 https://archive.ics.uci.edu/ml/datasets/iris

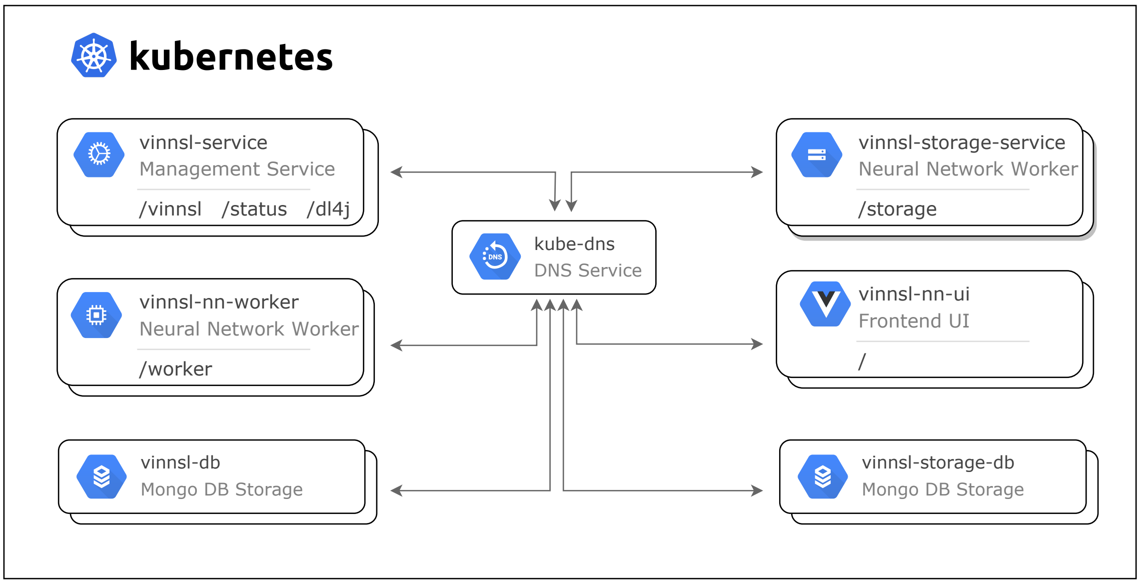

Figure 4.6: Service Discovery with kube-dns

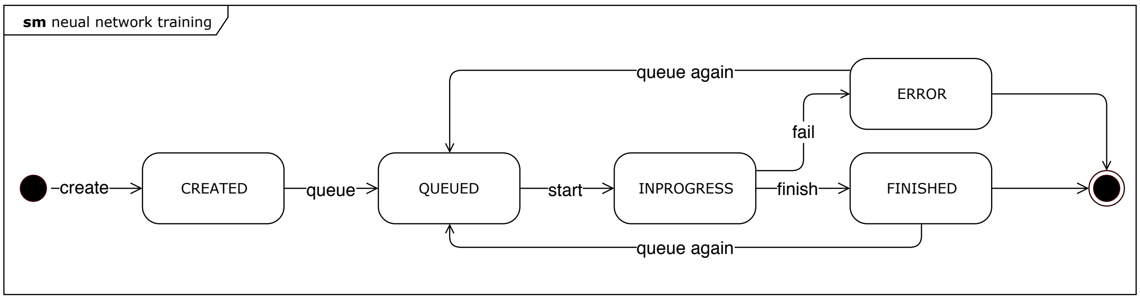

Figure 4.7: State Machine of a Neural Network

4.7 Neural Network Objects State

The state of neural network objects is saved in the `NnCloud` object. When the object is instantiated the default value is `CREATED`. When the network is queued, the worker service gathers all the necessary data from the vinnsl and vinnsl storage service and changes the state to `QUEUED`. During the network training, the worker changes the state to `INPROGRESS`. As soon as the training is finished, the worker service uploads the results and updated network state to the storage service and subsequently changes the state to `FINISHED`. Trained networks can be queued for retraining: in that case the state returns to `QUEUED`. If errors occur during the training process the state will be set to `ERROR`.

Figure 4.7 visualizes the state changes in a state machine.

5 Implementation

Following the specification, this section showcases an implementation of ConbexNN, using microservices glued together by *Kubernetes*. This represents the execution stack for neural networks. Backend components are realized with *Java* and the *Spring Boot* framework and expose a RESTful API. The processing and training of neural networks is done by the *Deeplearning4J* framework. Database and file storage are powered by *MongoDB*. The frontend service is implemented using *Vue.js* and the *Twitter Bootstrap* UI framework, visualizing and consuming backend services.

5.1 Source Code

The source code of the implemented microservices is released on *GitHub*. The following table gives an overview of available services and their corresponding repository.

Name	Repository Link
vinnsl-service	https://github.com/a00908270/vinnsl-service
vinnsl-nn-ui	https://github.com/a00908270/vinnsl-nn-ui
vinnsl-storage-service	https://github.com/a00908270/vinnsl-storage-service
vinnsl-nn-worker	https://github.com/a00908270/vinnsl-nn-worker

The *ViNNSL* XSD schema, specified in [Kop15], including (generated) examples, is released on GitHub with permission from Dipl.-Ing. Thomas Kopica. JAXB class generation of the XML files is already included in the release with the intention of making it easier to include *ViNNSL* into new services.

Name	Repository Link
vinnsl-schema	https://github.com/a00908270/vinnsl-schema

5.2 Releases

Docker Contrainers ready for deployment in a *Kubernetes* cluster are released on *DockerHub*. The following table references the released repositories.

Name	Repository Link
vinnsl-service	https://hub.docker.com/r/a00908270/vinnsl-service/
vinnsl-nn-ui	https://hub.docker.com/r/a00908270/vinnsl-nn-ui/
vinnsl-storage-service	https://hub.docker.com/r/a00908270/vinnsl-storage-service/
vinnsl-nn-worker	https://hub.docker.com/r/a00908270/vinnsl-nn-worker/

5.3 Framework Dependencies

All services are written in *Java* and built using the *Apache Maven* build automation and dependency management tool.

5.3.1 Spring

Spring is a *Java* framework consisting of many modules. Most importantly this project uses its feature so set up `RestController` instances that listen on specified endpoints.

Used in following services: `vinnsl-service`, `vinnsl-nn-ui`, `vinnsl-storage-service`, `vinnsl-nn-worker`

Spring Boot

Spring Boot is an extension to the framework that allows *Java* applications to run stand-alone by embedding a web server directly into the application. [Spra]

Used in following services: `vinnsl-service`, `vinnsl-nn-ui`, `vinnsl-storage-service`, `vinnsl-nn-worker`

Spring Data MongoDB

Spring Data provides an abstracted database access layer to MongoDB in form of a POJO (Plain Old Java Object). [Sprb]

Used in following services: `vinnsl-service`, `vinnsl-storage-service`

5.3.2 Swagger

Swagger is used to generate a live documentation of all web service endpoints in this project and allows to try out requests directly in the user interface.

Used in following services: `vinnsl-service`, `vinnsl-storage-service`, `vinnsl-nn-worker`

5.3.3 Fabric8

Fabric8 packs the generated executables from the build process into a *Docker* container that can run in a *Kubernetes* cluster.

Used in following services: `vinnsl-service`, `vinnsl-nn-ui`, `vinnsl-storage-service`, `vinnsl-nn-worker`

5.3.4 Deeplearning4J

Deeplearning4J is used by the worker service to train and evaluate neural networks.

A detailed introduction to Deeplearning4J can be found in Section 2.5.3.

Used in following services: `vinnsl-nn-worker`

5.4 Security

Ingress supports HTTPS encrypted connections. Authentication or restrictions are not implemented in the prototype.

5.5 User Interface

5.5.1 vinnsl-nn-ui (Frontend UI)

The `vinnsl-nn-ui` is a single page application (SPA) that displays all neural networks and their details in a web based frontend. Figure 5.1 shows a screenshot of the user interface.

Architecture

The web application is a Javascript based frontend, using the *Vue.js* and *Twitter Bootstrap* framework. The single main controller, called `VinnslUI`, provides methods to fetch a list of neural networks and their status. Additionally it queries for available files from the storage service and enables to connect them to a neural network.

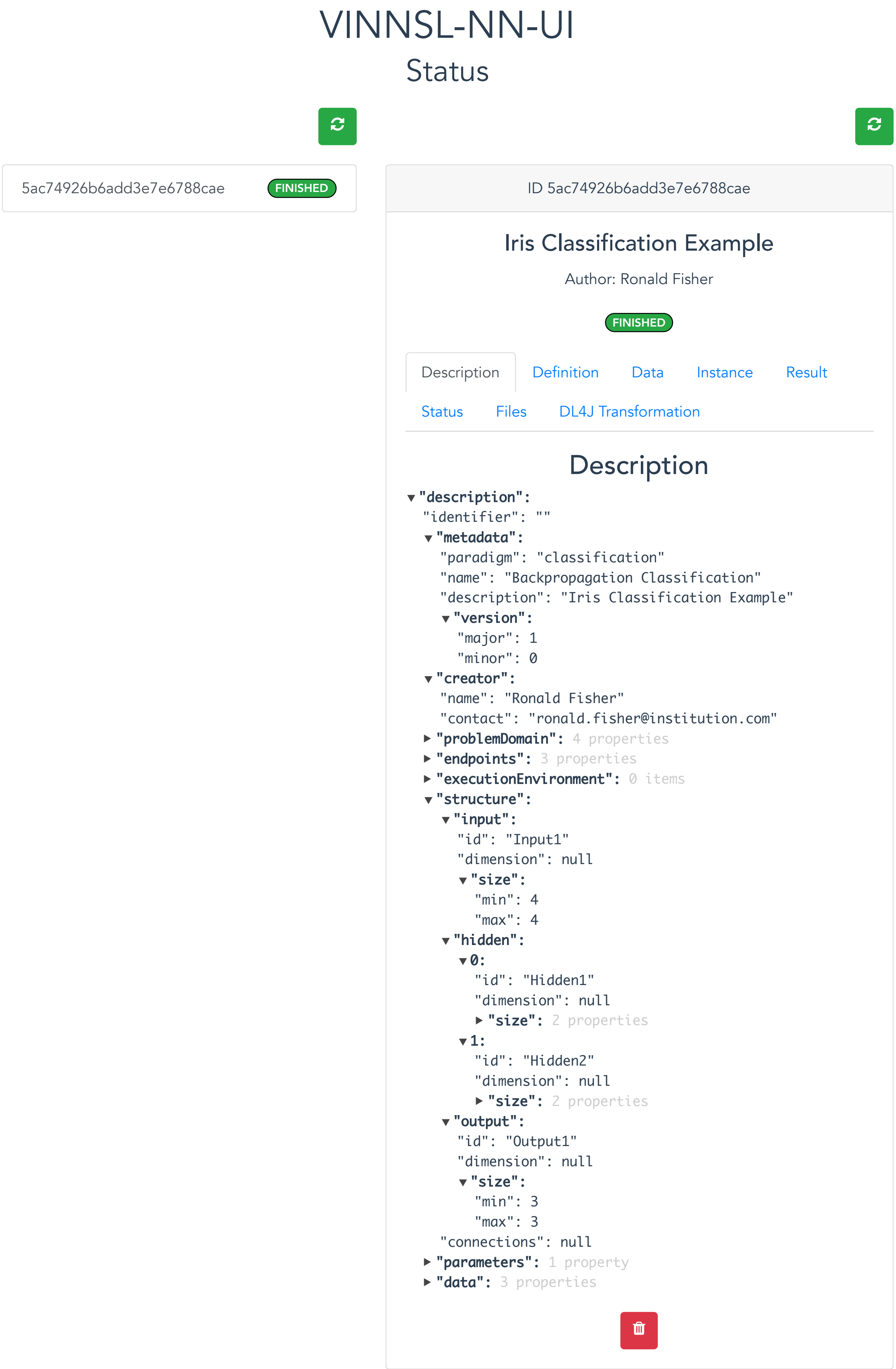

Figure 5.1: User Interface of ConbexNN

5.6 Endpoints

The following table gives an overview of the provided RESTful endpoints provided by different services. They are made available via *Ingress* outside the *Kubernetes* cluster.

Service Name	Exposed Endpoints
vinnsl-service	`/vinnsl`, `/status`, `/dl4j`
vinnsl-nn-ui	`/`
vinnsl-storage-service	`/storage`
vinnsl-nn-worker	`/worker`

5.6.1 Additional Endpoints

Additional endpoints are used internally and are not directly exposed outside the *Kubernetes* cluster. They can be reached by using port forwarding to directly access the service in the cluster.

/health The health endpoints returns the status of the application. `UP` if the application is running as expected, `DOWN` if parts of the application fail (like lost connection to the database). *Kubernetes* and *Ingress* use this endpoint to detect disturbances in the application.

/swagger The API, provided by the services, is documented and *Swagger* provides a web interface to the documentation.

5.7 Class Diagrams

This section features class diagrams of the provided RESTful services. All of them, as mentioned, are based on Java *Spring Boot* and use the *Spring Boot Data* layer if connecting to a database.

5.7.1 vinnsl-service

The *vinnsl service* is the main communication hub that enables access to the neural network objects and all of its data and provides interfaces to update it. The service connects to a *MongoDB* database where all its persisted data is stored via the *Spring Data* template. Fig. 5.2 shows the class diagram of the *vinnsl service*.

VinnslServiceApplication

`VinnslServiceApplication` is the main class that initializes the *Spring Boot* configuration and *MongoDB* repository.

VinnslServiceController

`VinnslServiceController` is a *Spring* `RestController` implementing all Mappings for the endpoint `/vinnsl`. All required dependencies on *MongoDB* are injected by Spring Boot.

NnStatusController

`NnStatusController` provides methods to get the current training status of one or all individual neural network(s). Methods are exposed at the `/status` endpoint. Other services, like the *vinnsl worker service* can also update the status.

Dl4JServiceController

`Dl4JServiceController` is a controller that allows manipulation of the *Deeplearning4J* property of a neural network using the `/dl4j` endpoint.

Vinnsl

The *vinnsl* class is a *POJO*[1] representation of the *ViNNSL* XML structure and used across different services.

NnCloud

The `NnCloud` class is an extension to `Vinnsl`, used to store the status and the *Deeplearning4J* representation of a neural network.

5.7.2 vinnsl-storage-service

The *vinnsl storage service* is a web service for storing and retrieving files in a *MongoDB* database. *GridFS*, which enables to store large data is activated. Figure 5.3 shows the class diagram.

VinnslStorageApplication

`VinnslStorageApplication` is the main class that initializes the *Spring Boot* configuration and *MongoDB* repository.

1 Plain Old Java Object

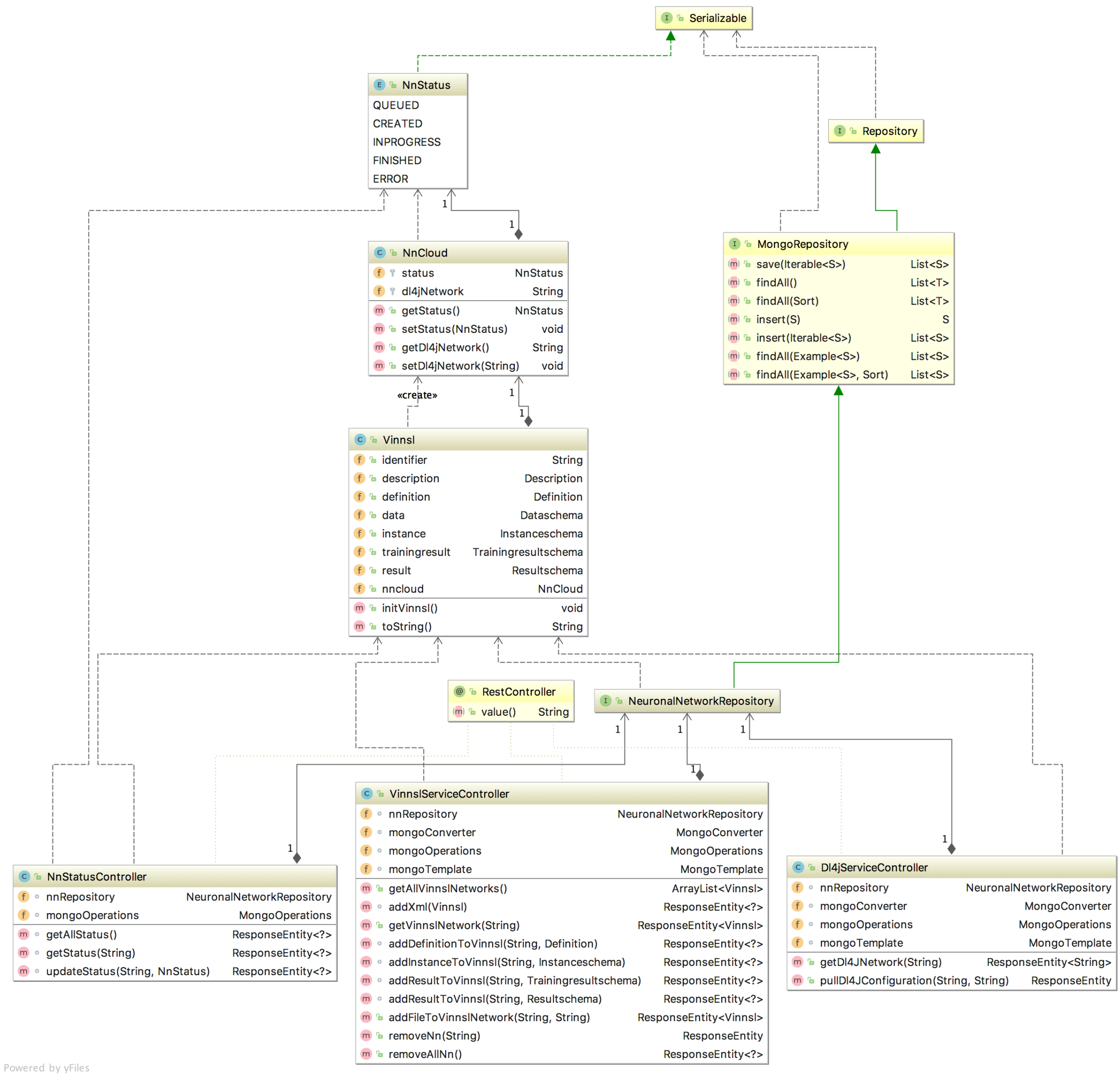

Figure 5.2: Class Diagram of vinnsl-service

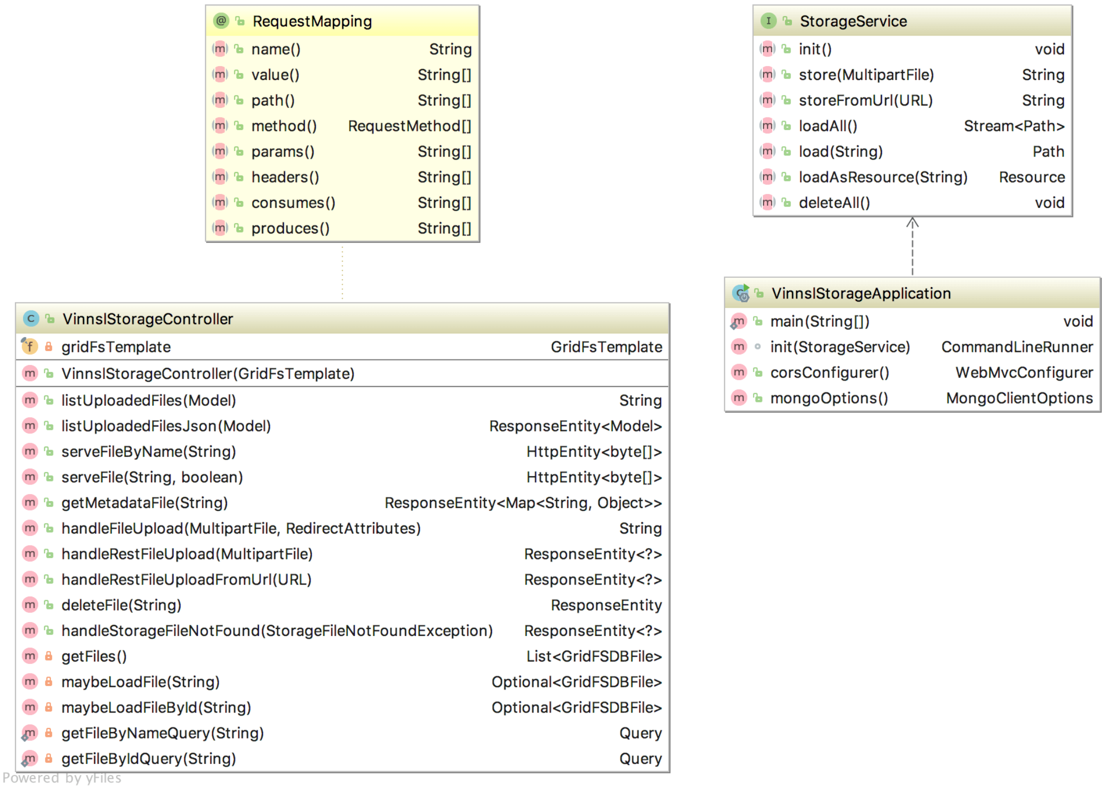

Figure 5.3: Class Diagram of vinnsl-storage-service

VinnslStorageController

`VinnslStorageController` makes retrieving and uploading files available via the `/storage` endpoint.

For one, there is an HTML form that enables a `Multipart` file upload from a browser, which is handled by the `handleFileUpload()` method. Secondly instead of directly uploading a file, a *URL* can be given as parameter via the `handleRestFileUploadFromUrl`. The storage service takes care of downloading and storing the file. The controller uses the *GridFS* template as an abstraction to the *MongoDB* database.

5.7.3 vinnsl-worker-service

The *vinnsl worker service* is the component, which is used for training and evaluating neural networks, executing the *Deeplearning4J* framework. Figure 5.4 shows the class diagram.

MappingUtil / VinnslDL4JMapperImpl

The classes `MappingUtil` and `VinnslDL4JMapperImpl` are responsible for mapping a *Vinnsl* to a *DeepLearning4J* network that can be trained.

The mappings are done in the inner classes of the `MappingUtil`. `VinnslDL4JMapperImpl` initializes the necessary objects and calls the right methods to perform the mapping.

Worker Controller

`WorkerController` is a `RestController` that exposes the `/worker/queue` endpoint and can be used to schedule neural networks for training.

WorkerQueue

`WorkerQueue` is the data structure that stores the identifiers of the queued networks in memory.

Worker

The worker class checks the `WorkerQueue` periodically and if not empty polls the first element. It fetches the associated *Vinnsl* network from the *vinnsl-service* and hands it over to the *Dl4JNetworkTrainer*. The service further sets the training status to `INPROGRESS`.

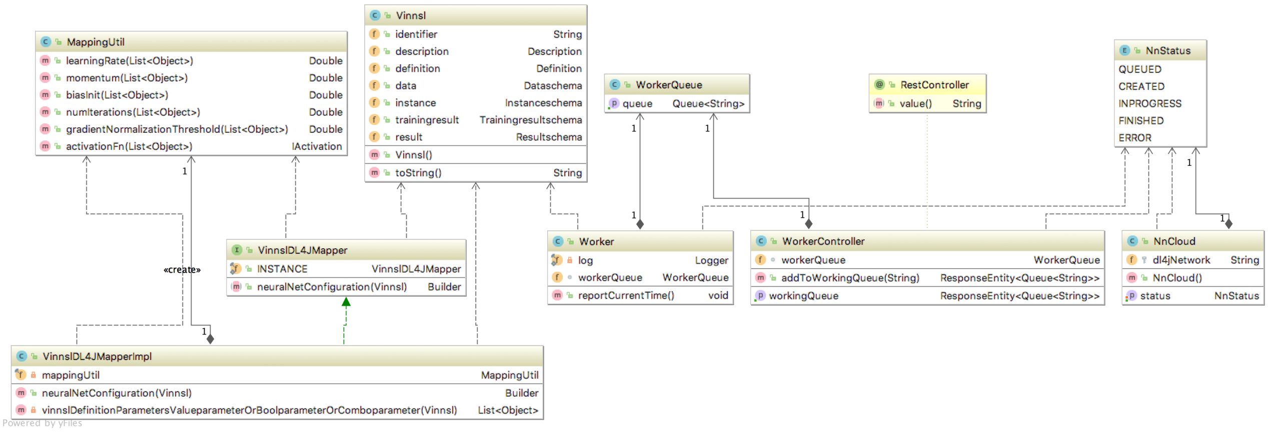

Figure 5.4: Class Diagram of vinnsl-worker-service

Dl4JNetworkTrainer

The training is initiated by the `Worker` class. The *network trainer* fetches and parses the training data if necessary (for example *comma separed value* files) and initializes the `MappingUtil`. The transformed *Deeplearning4J* model contains the neural network structure and parameters, required for training and is attached to the *ViNNSL* model.

Next the *Deeplearning4J* `UI Server` is initialized, which visualizes the training process. Test and training data is split and the training is started. After the training process is finished, the result is uploaded to the storage service.

5.7.4 vinnsl-nn-ui

The frontend service consists of one single controller named `VinnslUI`. The `getStatus()` method retrieves all neural network ids and their status. This is stored in `vinnslList`. When selecting a neural network from the list, the neural network object is loaded by executing `getDetailsById()`. The response is stored in `currentVinnslItem`.

Figure 5.5 gives an overview of the used methods and stored variables.

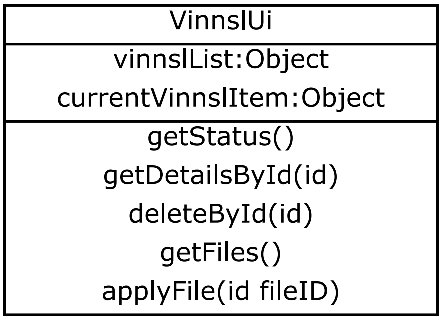

Figure 5.5: VinnslUI Vue Class

5.8 Limitations

5.8.1 Neural Network Design

The prototyped ViNNSL to Deeplearning4J mapper currently supports only multi-layers and fully connected backpropagation networks.

5.8.2 Parameters

The ViNNSL to Deeplearning4J mapper currently supports the following parameters:

1. learningrate
2. momentum
3. biasInput
4. epochs
5. threshold
6. activationfunction

6 API Documentation

Base URL

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

6.1 vinnsl-service

6.1.1 Import a new ViNNSL XML Defintion

`POST /vinnsl`

Parameters

Type	Name	Description	Schema
Body	**vinnsl** *required*	vinnsl	Vinnsl

Responses

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

Consumes

- `application/xml`

Produces

- `*/*`

Tags

- vinnsl-service-controller

Example HTTP request

Header

```
Content-Type: application/xml
```

Body

```
<vinnsl>
  <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>
```

```
<creator>
  <name>Benjamin Nussbaum</name>
  <contact>nussbaum@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>
```

```
            <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>
</vinnsl>
```


Example HTTP response

Statuscode: 201 CREATED

Header

```
Location: https://<baseURL>/vinnsl/5ade36bbd601800001206798
```

6.1.2 List all Neural Networks

```
GET /vinnsl
```

Responses

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

Produces

- `application/json`

Tags

- vinnsl-service-controller

Example HTTP Response


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

```
    ...
]
```

6.1.3 Delete all Neural Networks

`DELETE /vinnsl/deleteall`

Responses

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

Produces

- `application/json`

Tags

- vinnsl-service-controller

6.1.4 Get Neural Network Object

`GET /vinnsl/{id}`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	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>
```

```
    <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>FaceRecoginition</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>
```

```
        </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</file
    </data>
</description>
<definition>
    <identifier></identifier>
    <problemDomain>
        <propagationType type="feedforward">
            <learningType>supervised</learningType>
        </propagationType>
        <applicationField>EMS</applicationField>
        <applicationField>Operations</applicationField>
```

```
    <applicationField>FaceRecoginition</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>
```

```
            <comboparameter name="ativationfunction">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>
</vinnsl>
```


6.1.5 Remove Neural Network Object

`DELETE /vinnsl/{id}`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string

Responses

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

HTTP Code	Description	Schema
500	Server Error	No Content

Produces

- `*/*`

Tags

- vinnsl-service-controller

6.1.6 Add/Replace File of Neural Network

`PUT /vinnsl/{id}/addfile`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string
Query	**fileId** *required*	fileId	string

Responses

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

Consumes

- `application/json`

Produces

- `application/xml`
- `application/json`

Tags

- vinnsl-service-controller

6.1.7 Add/Replace ViNNSL Definition of Neural Network

`PUT /vinnsl/{id}/definition`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string
Body	**def** *required*	def	Definition

Responses

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

HTTP Code	Description	Schema
500	Server Error	Error

Consumes

- `application/xml`
- `application/json`

Produces

- `*/*`

Tags

- vinnsl-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>
```

```
<creator>
  <name>Nussbaum</name>
</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>
   <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>
```


6.1.8 Add/Replace ViNNSL Instanceschema of Neural Network

```
PUT /vinnsl/{id}/instanceschema
```

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string

Type	Name	Description	Schema
Body	**instance** *required*	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

- vinnsl-service-controller

Example HTTP request

Request body

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

6.1.9 Add/Replace ViNNSL Resultschema of Neural Network

`PUT /vinnsl/{id}/resultschema`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string
Body	**resultSchema** *required*	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

- vinnsl-service-controller

Example HTTP request

Request body

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

6.1.10 Add/Replace ViNNSL Trainingresult of Neural Network

```
PUT /vinnsl/{id}/trainingresult
```

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string
Body	**trainingresult** *required*	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>
```

6.1.11 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"
}
```


6.1.12 Get Status of Neural Network

`GET /status/{id}`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string

Responses

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

Produces

- `application/json`

Tags

- nn-status-controller

6.1.13 Set Status of a Neural Network

`PUT /status/{id}/{status}`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string
Path	**status** *required*	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

6.1.14 Get Deeplearning4J Transformation Object of Neural Network

`GET /dl4j/{id}`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string

Responses

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

Produces

- `application/json`

Tags

- dl4j-service-controller

6.1.15 Put Deeplearning4J Transformation Object of Neural Network

`PUT /dl4j/{id}`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string
Body	**dl4J** *required*	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

6.2 vinnsl-storage-service

6.2.1 Handle File Upload from HTML Form

`POST /storage`

Parameters

Type	Name	Description	Schema
FormData	**file** *required*	file	file

Responses

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

Consumes

- `multipart/form-data`

Produces

- `\*/*`

Tags

- vinnsl-storage-controller

6.2.2 List all Files

`GET /storage`

Responses

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

Produces

- `application/json`

Tags

- vinnsl-storage-controller

6.2.3 Download File by Original Filename

`GET /storage/files/name/{filename}`

Parameters

Type	Name	Description	Schema
Path	**filename** *required*	filename	string

Responses

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

Produces

- `\*/*`

Tags

- vinnsl-storage-controller

6.2.4 Download or Show File by FileID

`GET /storage/files/{fileId}`

Parameters

Type	Name	Description	Schema
Path	**fileId** *required*	fileId	string
Query	**download** *optional*	download	boolean

Responses

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

Produces

- `\*/*`

Tags

- vinnsl-storage-controller

6.2.5 Delete File by FileID

`DELETE /storage/files/{fileId}`

Parameters

Type	Name	Description	Schema
Path	**fileId** *required*	fileId	string

Responses

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

HTTP Code	Description	Schema
403	Forbidden	No Content

Produces

- `\*/*`

Tags

- vinnsl-storage-controller

6.2.6 Get File Metadata by FileID

`GET /storage/metadata/{fileId}`

Parameters

Type	Name	Description	Schema
Path	**fileId** *required*	fileId	string

Responses

HTTP Code	Description	Schema
200	OK	< string, object > map
404	Not Found	No Content

Produces

- `\*/*`

Tags

- vinnsl-storage-controller

6.2.7 Upload MultipartFile

`POST /storage/upload`

Parameters

Type	Name	Description	Schema
FormData	**file** *required*	file	file

Responses

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

Consumes

- `multipart/form-data`

Produces

- `application/json`

Tags

- vinnsl-storage-controller

6.2.8 Upload File by URL

`GET /storage/upload`

Parameters

Type	Name	Description	Schema
Query	**url** *required*	url	string

Responses

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

Produces

- `application/json`

Tags

- vinnsl-storage-controller

6.3 vinnsl-worker-service

6.3.1 getWorkingQueue

`GET /worker/queue`

Responses

HTTP Code	Description	Schema
200	OK	< string > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Produces

- `\*/*`

Tags

- worker-controller

6.3.2 addToWorkingQueue

`PUT /worker/queue/{id}`

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string

Responses

HTTP Code	Description	Schema
200	OK	< string > array
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- worker-controller

7 Deployment

7.1 Local Machine

Prerequisites

- Install `kubectl` tool from: https://kubernetes.io/docs/tasks/tools/install-kubectl
- Install `minikube` tool from: https://github.com/kubernetes/minikube/releases
- git tool installed

Run minikube `minikube start`

Starts the minikube cluster

Setting up Clone the repository

```
git clone https://github.com/a00908270/vinnsl-nn-cloud.git
cd kubernetes_config/
```

Run Services in Cluster

```
# MongoDB for vinnsl-service
kubectl --context $CONTEXT create -f mongo.yaml
# Vinnsl Service
kubectl --context $CONTEXT create -f vinnsl-service.yaml
# MongoDB for vinnsl-storage-service
kubectl --context $CONTEXT create -f mongo-storage-service.yaml
```

```
# Vinnsl Storage Service
kubectl --context $CONTEXT create -f vinnsl-storage-service.yaml
# Vinnsl NN Worker Service
kubectl --context $CONTEXT create -f vinnsl-nn-worker.yaml
# Vinnsl Frontend UI Webapp
kubectl --context $CONTEXT create -f vinnsl-nn-ui.yaml
```

Enable and Set Up Ingress Sets up a proxy to make services available at the endpoint specified in the API Specification.

```
kubectl --context $CONTEXT apply -f ingress.yaml
```

7.2 Google Cloud Instance

This section describes how to deploy the execution stack into a Kubernetes cluster in the Google Kubernetes Engine.

Prerequisites

- Google Account with activated billing or credits
- `kubectl` tool on local machine installed: (`https://kubernetes.io/docs/tasks/tools/install-kubectl/#install-kubectl`)
- gcloud SDK locally installed (`https://cloud.google.com/sdk/downloads`)

Create Cluster

```
gcloud beta container --project "nn-cloud-201314" clusters create "cluster-1"
--zone "us-central1-a" --username "admin" --cluster-version "1.8.8-gke.0"
--machine-type "n1-standard-1" --image-type "COS" --disk-size "15" --scopes
"https://www.googleapis.com/auth/compute",
"https://www.googleapis.com/auth/devstorage.read_only",
```

```
"https://www.googleapis.com/auth/logging.write",
"https://www.googleapis.com/auth/monitoring",
"https://www.googleapis.com/auth/servicecontrol",
"https://www.googleapis.com/auth/service.management.readonly",
"https://www.googleapis.com/auth/trace.append"
--num-nodes "4" --network "default" --enable-cloud-logging
--enable-cloud-monitoring --subnetwork "default" --addons
HorizontalPodAutoscaling,HttpLoadBalancing,KubernetesDashboard
```

Clone the repository Clone the `vinnsl-nn-cloud` project and swtich into the google-cloud folder.

```
git clone https://github.com/a00908270/vinnsl-nn-cloud.git
cd kubernetes_config/google-cloud/
```

Run Services in Cluster

```
# MongoDB for vinnsl-service
kubectl --context $CONTEXT create -f mongo_small.yaml
# Vinnsl Service
kubectl --context $CONTEXT create -f vinnsl-service.yaml
# MongoDB for vinnsl-storage-service
kubectl --context $CONTEXT create -f mongo-storage-service_small.yaml
# Vinnsl Storage Service
kubectl --context $CONTEXT create -f vinnsl-storage-service.yaml
# Vinnsl NN Worker Service
kubectl --context $CONTEXT create -f vinnsl-nn-worker.yaml
# Vinnsl Frontend UI Webapp
kubectl --context $CONTEXT create -f vinnsl-nn-ui.yaml
```

Enable and Set Up Ingress Sets up a proxy to make services available at the endpoint specified in the API Specification.

```
kubectl --context $CONTEXT apply -f ingress.yaml
```

8 Use Cases

As a demonstration of the implemented ConbexNN, this thesis features two use cases with practical relevance.

8.1 Iris Classification Example

Ronald A. Fisher published 1936 in his paper *The use of multiple measurements in taxonomic problems* [Fis] a dataset that is known as the *Iris flower data set*.

The data set [Fis] features 50 examples of three Iris species: Iris setosa, Iris virginica and Iris versicolor. A table lists four measured features from each sample: the length and the width of the sepals and petals.

This use case shall showcase the use of the implemented prototype to create a neural network, train and evaluate it, using this dataset.

8.1.1 Dataset

The dataset exists in the UCI Machine Learning Repository [DKT17] as a *CSV* (comma separated value) file[1] which will be used for training. The first example has a sepal length/width of 5.1cm/3.5cm, a petal length/width of 1.4cm/0.2cm and is an Iris setosa.

The first lines of the dataset explain the structure of the dataset. The columns are formatted for better readability. The species column is an enumerated value.

1 https://archive.ics.uci.edu/ml/datasets/iris

Index	Iris species
0	Iris setosa
1	Iris virginica
2	Iris versicolor

```
Sepal length, Sepal width, Petal length, Peta width, Iris species
5.1         , 3.5        , 1.4         , 0.2       , 0
4.9         , 3.0        , 1.4         , 0.2       , 0
<more lines>
```

8.1.2 Prerequisites

- Kubernetes Cluster running
- Services from the Neural Network Execution Stack deployed in cluster
- Hostname `cluster.local` resolves to Minikube instance

8.1.3 Create the neural network

Request

```
POST https://cluster.local/vinnsl
```

BODY

```
<vinnsl>
  <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>
<creator>
  <name>Nussbaum</name>
</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>
           <min>3</min>
           <max>3</max>
       </size>
      </hidden>
      <output>
       <ID>Output1</ID>
       <size>
           <min>3</min>
           <max>3</max>
       </size>
      </output>
    </structure>
    <parameters>
       <valueparameter>learningrate</valueparameter>
       <valueparameter>biasInput</valueparameter>
       <valueparameter>biasHidden</valueparameter>
       <valueparameter>momentum</valueparameter>
       <comboparameter>ativationfunction</valueparameter>
       <valueparameter>threshold</valueparameter>
    </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>
</vinnsl>
```

Response

`201 CREATED`

Aside from the HTTP Status Code, we also get HTTP headers in the response. The one needed for further requests is named `location`. The value of this field is the URL of the network, that was created and can be used to get and update fields on the dataset.

In this example the following value is returned:

Header Name	Header Value
location	https://cluster.local/vinnsl/5b1811a046e0fb0001fa28cc

The id of the new dataset is 5b1811a046e0fb0001fa28cc. In the following requests the id is shortened as `{id}`.

8.1.4 Add ViNNSL Definition to the Neural Network

The ViNNSL definition XML contains metadata like name and description of the network as well as the stucture of the neural network model. There is one input and one output layer defined. In between there are two hidden layers. It is also possible to specify additional parameters.

The activation function is set to tangens hyperbolicus, the learning rate is 0.1 and the training is limited to 500 iterations. A seed, set to 6, allows a reproducible training score.

Request

`POST https://cluster.local/vinnsl/{id}/definition`

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>
<creator>
  <name>Nussbaum</name>
</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>
    <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>
```

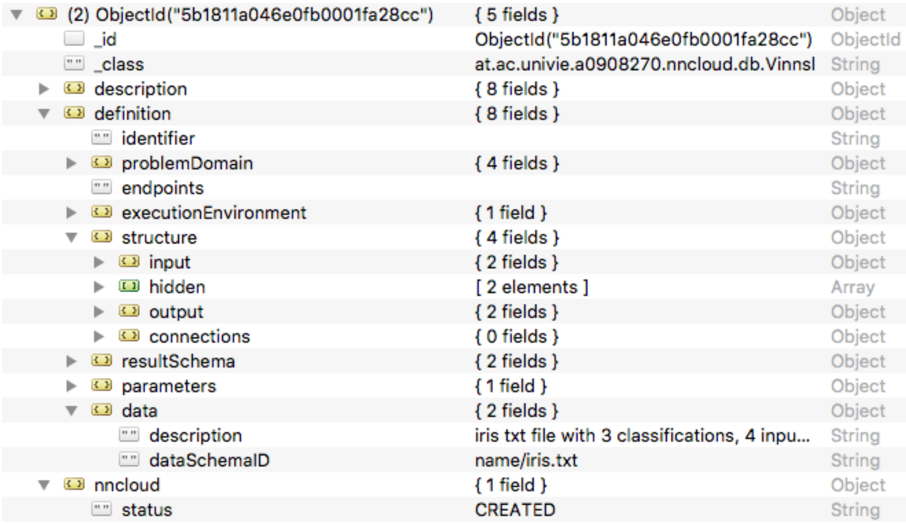

Figure 8.1: Neural Network Datastructure visualized in the Robo3T[2] application

Response

`200 OK`

Figure 8.1 shows a graphical visualisation of the neural network data structure, after adding the description and definition in *ViNNSL* XML. It is noticeable that *description* and *definition* have been transformed into objects. The status is initialized with the value `CREATED`.

8.1.5 Queue Network for Training

Request

`POST https://cluster.local/worker/queue/{id}`

Response

`200 OK`

2 https://www.robomongo.org

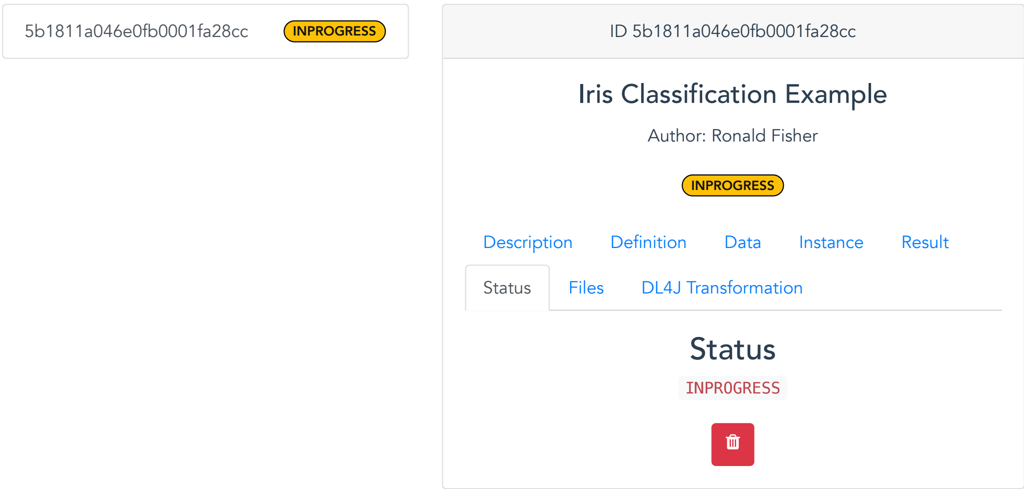

Figure 8.2: ViNNSL NN UI shows training in progress

8.1.6 Training

During the training it is possible to open the graphical user interface, called *DL4J Training UI* in a browser, that is provided with the *Deeplearning4J* package, to see the learning progress of the neural network.

```
https://cluster.local/train/overview
```

DL4J Training UI

Figure 8.3 shows the network training of the Iris Classification. The overview tab provides general information about network and training.

- Top left: score vs iteration chart - value of the loss function
- Top right: model and training information
- Bottom left: Ratio of parameters to updates (by layer) for all network weights vs. iteration
- Bottom right: Standard deviations (vs. time) of: updates, gradients and activations

[Dee]

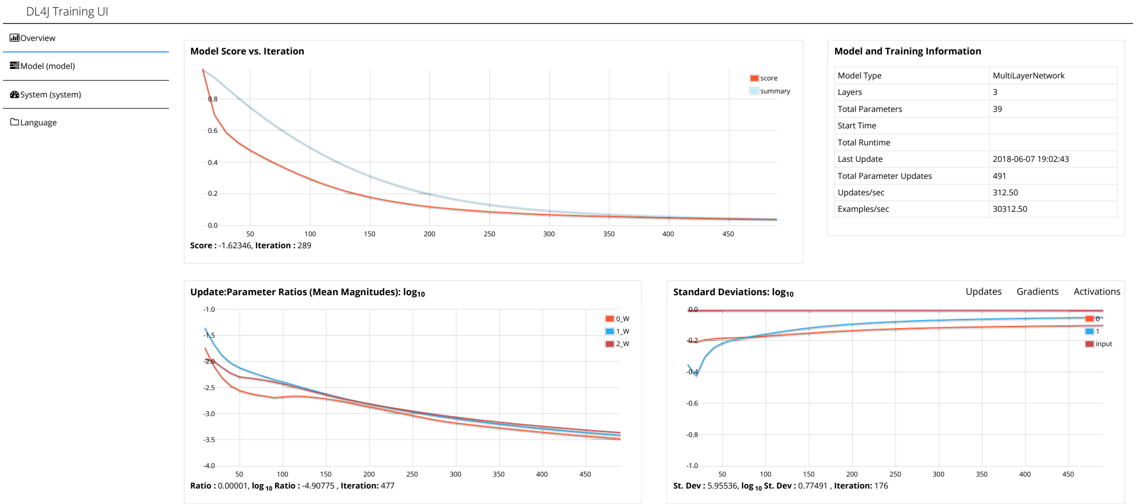

Figure 8.3: *DL4J Training UI* shows training progress of Iris Classification network

The second tab provides information about the neural network layers of the model. Information includes:

- Table of layer information
- Layer activations over time
- Histograms of parameters and updates
- Learning rate vs. time

[Dee]

8.1.7 Testing

Testing takes place automatically after training and evaluates the accuracy of the trained neural network. In this case 65 percent of the dataset is used for training and 35 percent for testing.

8.1.8 Evaluation Result

As soon as the training and testing process is finished, a file with the testing report is ready on the storage server. Figure 8.4 clarifies the updated data structure of the neural network object. A result file with id `5b19972052faff0001cb6bbf` was uploaded to the

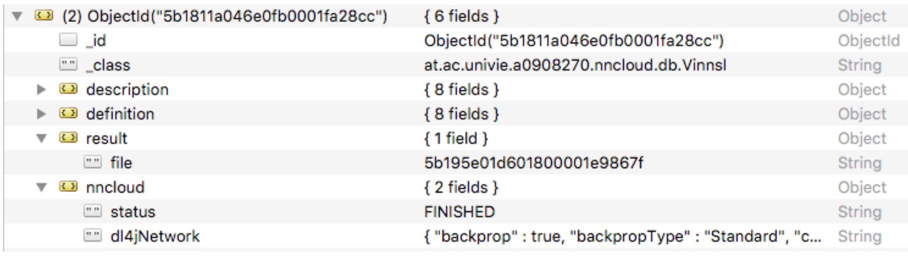

Figure 8.4: Neural Network Datastructure of the fninished network visualized in the Robo3T[3] application

storage service. The status is changed to `FINISHED` and the transformed *Deeplearning4J* model representation is updated in the field *dl4jNetwork*.

In the *ViNNSL NN UI*, the result file can be viewed by switching to the *Data* tab and selecting *See File* under the headline *Result Data*.

```
[...]

Examples labeled as 0 classified by model as 0: 19 times
Examples labeled as 1 classified by model as 1: 17 times
Examples labeled as 1 classified by model as 2: 2 times
Examples labeled as 2 classified by model as 2: 15 times


==========================Scores========================================
 # of classes:    3
 Accuracy:        0.9623
 Precision:       0.9608
 Recall:          0.9649
 F1 Score:        0.9606
Precision, recall & F1: macro-averaged (equally weighted avg. of 3 classes)
========================================================================
```

By examining the result file, it can be noticed that the accuracy of the network was 96 percent. All *Iris setosa* and *Iris versicolor* from the testset were recognized correctly, two *Iris virginica* were incorrectly recognized as *Iris versicolor*.

3 https://www.robomongo.org

8.2 Wine Score Classification

The second use case shows that a very similar ViNNSL Network can be used on a different data set containing a large collection of wine reviews from a platform called *WineEnthusiast*[4]. The dataset's feature columns were reduced and the lines limited to twenty-thousand.

8.2.1 Dataset

The dataset which will be used for training, contains 20.000 elements with two feature columns and two possible classes. The first feature column is the category of wine (red or white wine), the second is the price (numerical). There are two possible classes: the first class is applicable if the rating score, that has possible values between 0 and 100, is below 90. Otherwise the second class is applicable.

Possible Classification

Index	Review Rating Score
0	Review rating score <= 90
1	Review rating score > 90

Wine Category

Index	Wine Category
0	Red wine
1	White wine

4 https://www.winemag.com/?s=&drink_type=wine

The first wine has a rating score higher than 90, is a red wine and costs 235 US-Dollars. The second wine is also red, has also a rating over 90, but costs only 65 US-Dollars.

The first lines of the dataset explain the structure of the dataset. The columns are formatted for better readability.

```
Rating Score Class, Category, Price (US$)
1                 , 0       , 235
1                 , 0       , 65
<more lines>
```

8.2.2 Prerequisites

- Kubernetes Cluster running
- Services from the Neural Network Execution Stack deployed in cluster
- Hostname `cluster.local` resolves to Minikube instance

8.2.3 Create the neural network

Request

```
POST https://cluster.local/vinnsl
```

BODY

```
<vinnsl>
  <description>
    <identifier><!-- will be generated --></identifier>
    <metadata>
      <paradigm>classification</paradigm>
      <name>Backpropagation Classification</name>
      <description>Wine Classification Example</description>
      <version>
```

```
    <major>1</major>
    <minor>0</minor>
  </version>
</metadata>
<creator>
  <name>Nussbaum</name>
  <contact>nussbaum@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>2</min>
        <max>2</max>
    </size>
   </input>
   <hidden>
    <ID>Hidden1</ID>
    <size>
        <min>3</min>
        <max>3</max>
    </size>
```

```
      </hidden>
      <hidden>
       <ID>Hidden2</ID>
       <size>
           <min>3</min>
           <max>3</max>
       </size>
      </hidden>
      <output>
       <ID>Output1</ID>
       <size>
           <min>2</min>
           <max>2</max>
       </size>
      </output>
    </structure>
    <parameters>
       <valueparameter>learningrate</valueparameter>
       <valueparameter>biasInput</valueparameter>
       <valueparameter>biasHidden</valueparameter>
       <valueparameter>momentum</valueparameter>
       <comboparameter>ativationfunction</valueparameter>
       <valueparameter>threshold</valueparameter>
    </parameters>
    <data>
       <description>wine csv file with 2 classifications, 2 input vars</description>
       <tabledescription>no input as table possible</tabledescription>
       <filedescription>CSV file</filedescription>
    </data>
 </description>
</vinnsl>
```

Response

```
201 CREATED
```

Aside from the HTTP Status Code, we also get HTTP headers in the response. The one needed for further requests is named `location`. The value of this field is the URL of the network, that was created and can be used to get and update fields on the dataset.

In this example the following value is returned:

Header Name	Header Value
location	https://cluster.local/vinnsl/5ac6a8796522050001dfff3b

The id of the new dataset is 5ac6a8796522050001dfff3b. In the following requests the id is shortened as `{id}`.

8.2.4 Add ViNNSL Definition to the Neural Network

The ViNNSL definition XML contains metadata like name and description of the network as well as the stucture of the neural network model. There is one input and one output layer defined. In between there are two hidden layers. It is also possible to specify additional parameters.

The activation function is set to tangens hyperbolicus, the learning rate is 0.1 and the training is limited to 500 iterations. A seed, set to 6, allows a reproducible training score.

Request

```
POST https://cluster.local/vinnsl/{id}/definition
```

BODY

```
<definition>
<identifier><!-- will be generated --></identifier>
<metadata>
  <paradigm>classification</paradigm>
   <name>Backpropagation Classification</name>
   <description>Wine Classification Example</description>
  <version>
    <major>1</major>
    <minor>0</minor>
  </version>
</metadata>
<creator>
  <name>Nussbaum</name>
  <contact>nussbaum@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>2</size>
   </input>
   <hidden>
```

```
     <ID>Hidden1</ID>
     <size>3</size>
    </hidden>
    <hidden>
     <ID>Hidden2</ID>
     <size>3</size>
    </hidden>
    <output>
     <ID>Output1</ID>
     <size>2</size>
    </output>
    <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>wine csv file with 2 classifications, 2 input vars</description>
     <dataSchemaID>name/wines.csv</dataSchemaID>
  </data>
</definition>
```

Response

200 OK

8.2.5 Queue Network for Training

Request

POST https://cluster.local/worker/queue/{id}

Response

200 OK

8.2.6 Evaluation Result

As soon as the training and testing process is finished, a file with the testing report is ready on the storage server. A result file with id `5ac6a8796522050001dfff3b` was uploaded to the storage service. The status is changed to `FINISHED` and the transformed *Deeplearning4J* model representation is updated in the field *dl4jNetwork*.

In the *ViNNSL NN UI*, the result file can be viewed by switching to the *Data* tab and selecting *See File* under the headline *Result Data*.

```
[...]

Examples labeled as 0 classified by model as 0: 4648 times
Examples labeled as 0 classified by model as 1: 548 times
Examples labeled as 1 classified by model as 0: 959 times
Examples labeled as 1 classified by model as 1: 845 times


==========================Scores===========================================
```

```
 # of classes:    2
 Accuracy:        0.7847
 Precision:       0.7178
 Recall:          0.6815
 F1 Score:        0.6946
Precision, recall & F1: macro-averaged (equally weighted avg. of 3 classes)
===========================================================================
```

By examining the result file, it can be noticed that the accuracy of the network was 78.5 percent. The network was pretty good at classifying the ratings soly based on wine category and price.

9 Future Work

The flexibility of the presented neural network stack opens up many opportunities for further work and integration into already existing frameworks and applications. This section points out a few ideas.

9.1 ViNNSL Compatibility

ViNNSL compatibility is limited in the current prototype of ConbexNN and could be fully implemented to be fully compatible with other systems. See section 5.8 for current limitations.

9.2 Integration in N2Sky

N2Sky features a graphical editor to design the neural network structure and training of the model, as seen in Figure 9.1. *N2Sky* also uses the *ViNNSL* language to model neural networks. It enables to run the training process in the neural network stack by using the provided API.

9.3 Neural Network Backends

Presently the *Deeplearning4J* platform undertakes the task of network training. ViNNSL XML files are transformed into a *Deeplearning4J* model before training. With manageable development effort the API could be extended to support direct import of *Deeplearning4J*

Figure 9.1: N2Sky Neural network evaluation process [FAS18]

models. Furthermore the Framework provides support for Keras[1], a python framework that is fitted to run on top of TensorFlow, Microsoft Cognitive Toolkit and Theano [dl418] [ker18]. By extending the API to enable an import of these frameworks, demand from other target audiences could be covered.

9.4 Graphical Neural Network Designer

The *ViNNSL* XML scheme could be used to design and validate *ViNNSL* networks in a graphical editor, presenting a drag&drop interface. Another possible function could be an integration of the neural network stack directly into the visual designer to import and train networks into the cluster without leaving the application.

1 https://keras.io

9.5 Deploy trained Models as Web Service

After the training of a neural network model is finished, a useful functionality would be to expose the result for further predictions as a web service. Client applications could run their requests against the trained models to receive model predictions.

9.6 Integrate into other Platforms

There are neural network platforms on the market that could be integrated. According to a Gartner report from February 2018, *KNIME* is currently leading in the category "Data Science and Machine Learning Platforms" [Gar].

9.6.1 KNIME

KNIME Analytics Platform[2] is open-source at its core[3] and already features a *Deeplearning4J* integration[4]. Figure 9.2 shows a screenshot of the application designing a Multi Layer Perceptron network and exporting it to a *Deeplearning4J* model. As the application is open-source and extensible, an option to export and train models using the presented execution stack could be added.

9.7 Full featured Web Application

The graphical interface of ConbexNN provides a quick overview over neural networks and their status, but does not cover all features specified in the RESTful API. It could be extended to behave like a fully featured web application that can be used as an alternative to the API. It could also provide a functionality to integrate plugins into the user interface.

2 https://www.knime.com
3 https://github.com/knime/knime-core
4 https://github.com/knime/knime-dl4j

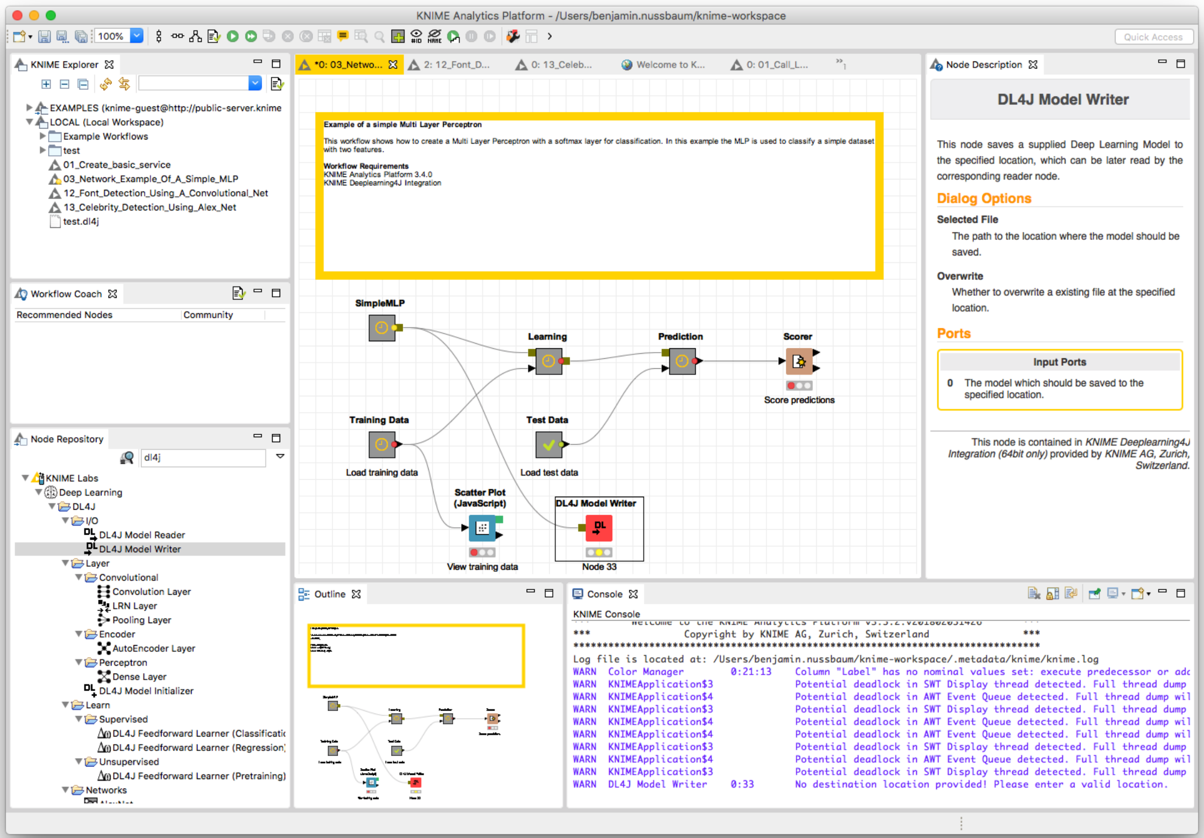

Figure 9.2: Screenshot of *KNIME Analytics Platform* using *Deeplearning4J* Integration

10 Conclusion

This thesis presented an open-source execution stack for neural network simulation in an effective and efficient way using simple RESTful webservices fostering Kubernetes Cloud container orchestration and microservices. Using this technique it becomes possible to scale individual services easily and automatically according to current load. Each component is fully interchangeable, as long as the documented RESTful API is implemented. ConbexNN was demonstrated and evaluated on the Iris flower and a wine rating data set. Furthermore various ideas, to integrate this solution into other neural network platforms, were given.

Using ViNNSL as domain specific modelling language, enables users to define neural networks without explicit programming skills.

The appendix lists how to set up and use this stack in various ways on popular Cloud platforms.

11 Acknowledgments

I would like to thank my girlfriend and my whole family for the support and patience during the time I was occupied doing research, the long nights I was programming code, who were always pushing me forwards to achieve my goals.

Furthermore I would like to express my deepest appreciation to my supervisor and lecturer Mr. Univ.-Prof. DI Dr. Erich Schikuta for his ideas, support and input, which made this thesis possible.

Bibliography

[Bai15] Baier, Jonathan: *Getting Started with Kubernetes*. Packt Publishing, 2015

[BB16] Björn Böttcher, Dr. Carlo V. Daniel Klemm K. Daniel Klemm: Machine Learning im Unternehmenseinsatz / Crisp Research AG. Version: 2016. https://www.unbelievable-machine.com/downloads/studie-machine-learning.pdf. 2016. – Forschungsbericht

[BGO+16] Burns, Brendan; Grant, Brian; Oppenheimer, David; Brewer, Eric; Wilkes, John: Borg, Omega, and Kubernetes. In: *Communications of the ACM* 59 (2016), apr, Nr. 5, 50–57. http://dx.doi.org/10.1145/2890784. – DOI 10.1145/2890784

[Boh] Bohn, Björn: *OSCON: Kubernetes gewinnt den Preis in der Kategorie "Größte Auswirkung"*. https://www.heise.de/developer/meldung/OSCON-Kubernetes-gewinnt-den-Preis-in-der-Kategorie-Groesste-Auswirkung-4116 html, Retrieved: 2018-07-20

[BRBA17] Bashari Rad, Babak; Bhatti, Harrison; Ahmadi, Mohammad: An Introduction to Docker and Analysis of its Performance. In: *IJCSNS International Journal of Computer Science and Network Security* 17 (2017), 03, Nr. 3, S. 228–235

[BVSW08] Beran, P. P.; Vinek, E.; Schikuta, E.; Weishaupl, T.: ViNNSL - the Vienna Neural Network Specification Language. In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 2008. – ISSN 2161–4393, S. 1872–1879

[Dea15] Dean, Jeff: *TensorFlow: Large-scale machine learning on heterogeneous systems*. Nov 2015

[Dee] Deeplearning4J: *Visualize, Monitor and Debug Network Learning*. https://deeplearning4j.org/visualization

[DKT17] Dheeru, Dua; Karra Taniskidou, Efi: *UCI Machine Learning Repository*. `http://archive.ics.uci.edu/ml`. Version: 2017

[DL4] DL4J Authors, The: *Features*. `https://deeplearning4j.org/features`, Retrieved: 2018-07-20

[dl418] *https://deeplearning4j.org/model-import-keras*. `https://deeplearning4j.org/model-import-keras`. Version: 05 2018, Retrieved: Sunday, 10. June 2018

[Doc] Docker: *Swarm mode key concepts*. `https://docs.docker.com/engine/swarm/key-concepts/`

[Ell16] Ellingwood, Justin: *An Introduction to Kubernetes - DigitalOcean*. Version: 2016. `https://www.digitalocean.com/community/tutorials/an-introduction-to-kubernetes`, Retrieved: 2018-05-08

[Eva17] Evans Data Corporation: *AI, ML, and Big Data Survey 2017, Vol. 2*. Version: 2017. `https://evansdata.com/reports/viewRelease.php?reportID=37`

[FAS18] Fedorenko, Andrii; Adamenko, Aliaksandr; Schikuta, Erich: N2Sky - A Neural Network Problem Solving Environment Fostering Virtual Resources. In: *IJCNN 2018 : International Joint Conference on Neural Networks*, 2018

[Fis] Fisher, Ronald A.: The Use Of Multiple Measurements In Taxonomic Problems. In: *Annals of Eugenics* 7, Nr. 2, 179-188. `http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x`. – DOI 10.1111/j.1469–1809.1936.tb02137.x

[Frö] Fröhlich, Jochen: *Neural Networks with Java - Backpropagation*. `https://www.nnwj.de/backpropagation.html`, Retrieved: 2018-07-20

[Gar] Gartner: *Magic Quadrant for Data Science and Machine-Learning Platforms*

[Hau98] Haun, M.: *Simulation Neuronaler Netze: eine praxisorientierte Einführung ; mit 23 Tabellen*. expert-Verlag, 1998 (Reihe Technik). – ISBN 9783816915447

[Joy15] Joy, A. M.: Performance comparison between Linux containers and virtual machines. In: *2015 International Conference on Advances in Computer Engineering and Applications*, 2015, S. 342–346

[ker18] *Keras: The Python Deep Learning library*. `https://keras.io`. Version: 5 2018, Retrieved: Sunday, 10. June 2018

[Kop15] Kopica, Thomas: *Vienna Neural Network Specification Language 2.0*, Masterthesis, 2015

[Kuba] Kubernetes Authors, The: *Ingress*

[Kubb] Kubernetes Authors, The: *Kubernetes Components*. `https://kubernetes.io/docs/concepts/overview/components/`, Retrieved: 2018-05-10

[Kubc] Kubernetes Authors, The: *Kubernetes Concepts - Pods*. `https://kubernetes.io/docs/concepts/workloads/pods/pod/`, Retrieved: 2018-05-12

[Kubd] Kubernetes Authors, The: *Kubernetes DNS-Based Service Discovery*. `https://github.com/kubernetes/dns/blob/master/docs/specification.md`, Retrieved: 2018-05-31

[LF14] Lewis, James; Fowler, Martin: *Microservices: a definition of this new architectural term*. 2014

[Mak18] Makadia, Mitul: Top 8 Deep Learning Frameworks. In: *DZone* (2018), 03

[MCM13] Michalski, R.S.; Carbonell, J.G.; Mitchell, T.M.: *Machine Learning: An Artificial Intelligence Approach*. Springer Berlin Heidelberg, 2013 (Symbolic Computation). `https://books.google.at/books?id=-eqpCAAAQBAJ`. – ISBN 9783662124055

[Met14] Metz, Cade: *The Mission to Bring Google's AI to the Rest of the World*. `https://www.wired.com/2014/06/skymind-deep-learning/`. Version: 2014, Retrieved: 2018-07-20

[ND4a] ND4J Authors, The: *N-Dimensional Arrays for Java*. `https://nd4j.org/index.html`, Retrieved: 2018-07-20

[ND4b] ND4J Authors, The: *Speed*. `https://nd4j.org/benchmarking`, Retrieved: 2018-07-20

[ngi] nginx: *Using nginx as HTTP load balancer*. `http://nginx.org/en/docs/http/load_balancing.html`, Retrieved: 2018-05-31

[Por] Portworx: *Portworx Annual Container Adoption Survey 2017*

[RM17] Raschka, S.; Mirjalili, V.: *Machine Learning mit Python und Scikit-Learn und TensorFlow: Das umfassende Praxis-Handbuch für Data Science, Predictive Analytics und Deep Learning*. mitp-Verlag, 2017 (mitp Professional). https://books.google.at/books?id=JM5CDwAAQBAJ. – ISBN 9783958457355

[Sam59] Samuel, Arthur L.: Some studies in machine learning using the game of Checkers. In: *IBM JOURNAL OF RESEARCH AND DEVELOPMENT* (1959), S. 71–105

[SM13] Schikuta, Erich; Mann, Erwin: N2Sky - Neural networks as services in the clouds. In: *The 2013 International Joint Conference on Neural Networks (IJCNN)*, IEEE, aug 2013. – ISBN 978–1–4673–6129–3, 1-8

[Spra] Spring: *Spring Boot*. https://spring.io/projects/spring-boot, Retrieved: 2018-06-03

[Sprb] Spring: *Spring Data MongoDB*. https://projects.spring.io/spring-data-mongodb/, Retrieved: 2018-06-03

[Tena] TensorFlow Authors, The: *Estimators*. https://www.tensorflow.org/versions/master/guide/estimators, Retrieved: 2018-07-15

[Tenb] TensorFlow Authors, The: *Getting Started for ML Beginners*. https://www.tensorflow.org/versions/r1.5/get_started/get_started_for_beginners, Retrieved: 2018-07-15

[VGC+15] Villamizar, M.; Garcés, O.; Castro, H.; Verano, M.; Salamanca, L.; Casallas, R.; Gil, S.: Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In: *2015 10th Computing Colombian Conference (10CCC)*, 2015, S. 583–590

[VGO+16] Villamizar, M.; Garcés, O.; Ochoa, L.; Castro, H.; Salamanca, L.; Verano, M.; Casallas, R.; Gil, S.; Valencia, C.; Zambrano, A.; Lang, M.: Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures. In: *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2016, S. 179–182