|  |
| --- |
| Abbildung 1Neural network. If then else / Wikimedia, CC BY-SA |
| Anforderungsspezifikation: Optimierung von neuronalen Netzwerken  **Projektteam 17**  **Lukas Zoller**  **Remo Hofmann**  **Projektbetreuer**  **Jürgen Vogel**  **Version 1.0, 29.11.2019** |
| **Berner Fachhochschule**  Departement für Technik und Informatik  Titelabbildung: Neural network. If then else / Wikimedia, CC BY-SA |

Inhaltsverzeichnis

[1 Zweck des Dokuments 3](#_Toc25940433)

[2 Vision 3](#_Toc25940434)

[3 Projektzielsetzung 3](#_Toc25940435)

[3.1 Ausgangslage 3](#_Toc25940436)

[3.2 Stakeholder 3](#_Toc25940437)

[3.3 Projektziele 4](#_Toc25940438)

[4 Systemabgrenzung 4](#_Toc25940439)

[4.1 Prozessumfeld 4](#_Toc25940440)

[Systemumfeld 5](#_Toc25940441)

[4.2 Nicht unterstützte Projektziele 5](#_Toc25940442)

[5 Anforderungen 6](#_Toc25940443)

[5.1 Quellen und Vorgehen 6](#_Toc25940444)

[5.2 Funktionale Anforderungen 6](#_Toc25940445)

[5.2.1 Use Cases 6](#_Toc25940446)

[5.2.2 Use Case Diagramm 7](#_Toc25940447)

[5.2.3 Detaillierte Anforderungen 7](#_Toc25940448)

[5.3 Qualitätsanforderungen 9](#_Toc25940449)

[6 Glossar 10](#_Toc25940450)

[7 Literaturverzeichnis 11](#_Toc25940451)

[8 Anhang 11](#_Toc25940452)

[8.1 GUI-Mockup-Zeichnung 11](#_Toc25940453)

[8.2 UML Klassendiagramm 11](#_Toc25940454)

[9 Versionskontrolle 12](#_Toc25940455)

# Zweck des Dokuments

Dieses Dokument beschreibt die Ziele und Anforderungen für das Projekt "Optimierung von neuronalen Netzwerken". Es wird im Rahmen des Moduls BTI7082pq erarbeitet und bewertet.

# Vision

Üblicherweise wird für den Bau von neuronalen Netzwerken grosse Erfahrung in diesem Bereich benötigt. Insbesondere die Wahl der Hyperparameter eines neuronalen Netzwerks, also wie viele Ebenen ein solches Netzwerk besitzen soll, wie viele Knoten pro Layer, usw. sind Fragen, die man nicht ohne weiteres beantworten kann. Das Projekt liefert eine Applikation, welche diese Fragen beantworten, beziehungsweise die optimale Auswahl der Hyperparameter liefern soll. Der Hauptnutzen der zu entwickelnden Applikation liegt darin, dass verhältnismässig unerfahrene Personen ohne weitere Hilfe neuronale Netzwerke mit geringer Fehlerquote parametrisieren können. Bereits existierende Tools mit ähnlicher Funktion wie die zu entwickelnde Applikation sind nach Kenntnis der Autoren ausschliesslich als Library und für InCode-Anwendung verfügbar. Im Gegensatz dazu soll die hier beschriebene Applikation ein Standalone-Tool sein, das auch von Nutzern ohne grosse Programmiererfahrung verwendet werden kann.

# Projektzielsetzung

## Ausgangslage

Obwohl neuronale Netzwerke heute in verschiedensten Bereichen verwendet werden, wissen die wenigsten Verwender neuronaler Netzwerke eine Antwort auf die Frage welche Werte die Hyperparameter (z.B. Anzahl Layers, Anzahl Knoten Layer, etc.) dieses Netzwerks annehmen sollen, damit eine Möglichst hohe Performance resultiert – d.h. eine Kategorisierung durch das Netzwerk möglichst wenige falsche Zuordnungen macht. Heute wird dieses Problem hauptsächlich durch Trial and Error gelöst. Dafür wird eine grosse Erfahrung im Umgang mit neuronalen Netzwerken benötigt. Für Personen ohne diese Erfahrung wird es schwierig, gute Hyperparameter zu wählen.

Die Applikation soll das Problem der Optimierung der Hyperparameter lösen. Die Ausgangslage eines jeden Nutzers von neuronalen Netzwerken ist ein Trainingdatensatz, zu dem er ein möglichst performantes Netzwerk bilden möchte. Unsere Applikation nimmt als Input diesen Datensatz, testet verschiedene Konfigurationen von Hyperparametern und gibt die optimale Konfiguration zurück. Weitere Daten können anschliessend mit der Applikation und dem trainierten neuronalen Netzwerk kategorisiert werden.

Zwar existieren bereits Tools für die Optimierung von Hyperparametern von neuronalen Netzwerken. Diese Tools sind jedoch nur in Libraryform zugänglich und können somit nur von Programmierern verwendet werden. Unsere Applikation soll als Standalone-Tool auch von Nichtprogrammierern verwendet werden können.

## Stakeholder

Durch das Ziel, die Applikation zum Standalone-Tool zu machen, ist schlussendlich jeder Anwender ein Stakeholder, der ein neuronales Netzwerk erstellen möchte und sich nicht bereits sicher ist, wie dieses zu konfigurieren ist. Indirekt sind auch die Ersteller von Datensätzen (Trainingdaten für die neuronalen Netze, sowie Daten, welche klassifiziert werden sollen) Stakeholder der Applikation. Die Applikation wird in Python programmiert. Dafür wird die Keras-Library eingebunden, welche bereits gängige Methoden für die Klassifizierung bereitstellt und ihrerseits auf die Theano und Tensor-Flow Library für Python zurückgreift. Auch die Entwickler dieser Bibliotheken sind indirekte Stakeholder der Applikation.

## Projektziele

1. Standalone-Tool
2. Optimierung der Hyperparameter eines neuronalen Netzwerks bezüglich eines Trainingdatensatzes. Optimierung meint hier das Testen verschiedener Hyperparameterkonfigurationen eines neuronalen Netzwerks und anschliessend die Auswahl derjenigen Konfiguration, die auf einem Testdatensatz die geringste Fehlerquote generiert.
3. Kategorisierung weiterer Daten mit dem optimierten Netzwerk

Nicht-Ziele:

1. Vorbereitung der Daten: Daten müssen der Applikation in verarbeitbarer Form übergeben werden.
2. Daten müssen inhaltlich geeignet sein für neuronale Netzwerke: bestimmte Fragestellungen können nicht mit neuronalen Netzwerken bearbeitet werden obwohl die Daten der Form halber verarbeitet werden könnten. In solchen Fällen muss der Nutzer die Entscheidung treffen, ob die Applikation überhaupt geeignet ist.

# Systemabgrenzung

## Prozessumfeld

Das Prozessumfeld der Applikation besteht in technischer Hinsicht lediglich aus Input- und Outputdaten. Die Applikation benötigt Inputdaten, sogenannte Trainingdaten, um neuronale Netzwerke zu trainieren. Trainingdaten, auch «labelled data» genannt, bestehen aus einer bestimmten Anzahl von Fällen, sowie den zugehörigen Labels (Kategorien). Beispielsweise könnte ein Trainingdatensatz für die Erkennung von Katzenfotos wie folgt aussehen: Eine Tabelle mit 257 Spalten und 1000 Zeilen. Dabei stellen wir uns vor, dass es sich bei den ersten 256 Spalten um die Pixelwerte eines bestimmten 16\*16-Pixel-Bildes in Schwarzweiss handelt, der letzten Spalte um das besagte Label 1 oder 0, ob es sich um ein Katzenbild oder nicht handelt, und dass die 1000 Zeilen dieser Tabelle 1000 Bildern entsprechen.

Die Applikation wird mit derartigen Trainingdaten neuronale Netzwerke trainieren und testen. Mit einem trainierten Netzwerk lassen sich anschliessend «unlabelled datasets» kategorisieren. Beispielsweise könnte man mit dem Netzwerk (mit einer gewissen Genauigkeit) herausfinden, ob ein bestimmtes Bild eine Katze abbildet oder nicht. In diesem Fall würde der Applikation ein «unlabelled dataset» übergeben – also ein Datensatz mit 256 Spalten, der wieder pro Zeile die Pixelwerte eines Bildes enthält. Die Applikation kann diese Fälle mit dem trainierten Netzwerk kategorisieren und diesem unlabelled dataset eine weitere Spalte mit den entsprechenden Labels hinzufügen.

Zusammengefasst nimmt die Applikation also zwei Typen von Daten entgegen: erstens labelled datasets für das Training der neuronalen Netzwerke und unlabelled datasets für die Kategorisierung mit bereits trainierten Netzwerken. Zweitens gibt die Applikation Outputdaten in Form von kategorisierten Datensätzen zurück. Da das System eine Standalone-Applikation ist, welche auf einem Computer installiert wird, braucht es neben diesen Datensätzen keine weiteren Schnittstellen. Die Inputdaten müssen dem System in einem bestimmten Mass an Standardisierung übergeben werden (z.B. csv, nur zweidimensionale Arrays, etc). Anderfalls könnte der Aufwand für die Vorverarbeitung der Daten ins Unermessliche steigen.

## Systemumfeld

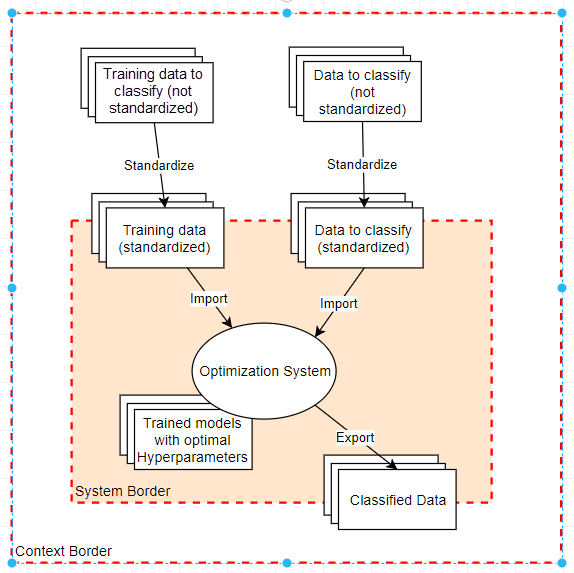


Abbildung 2: Systemgrenze und Kontextgrenze des Optimierungssystems

Wie in der Abbildung ersichtlich, werden vom System standardisierte Datensätze benötigt und standardisierte klassifizierte Datensätze ausgegeben. Als Input werden vom System zwei Arten von Datensätzen verarbeitet: Erstens klassifizierte Datensätze für das Training und das Testen der neuronalen Netzwerke, zweitens nicht klassifizierte Datensätze, die anschliessend mit einem neuronalen Netzwerk klassifiziert werden.

Die Hyperparameter der optimierten Modelle können ebenfalls ausgelesen werden und z.B. in anderen Applikationen weiterverwendet werden.

## Nicht unterstützte Projektziele

Die trainierten Modelle, können nicht in anderen Applikationen verwendet werden. D.h. es existiert beispielsweise keine Python-Schnittstelle, mit welcher man optimierte Modelle nach Python transferieren könnte. Es existiert lediglich die Möglichkeit, die Hyperparameter abzulesen und weiterzuverwenden.

Datensätze werden nur in formaler Hinsicht geprüft (z.B. können keine 3D-Arrays verarbeitet werden). Die inhaltliche Eignung wird jedoch in keiner Weise durchgeführt. Wird ein Trainingdatensatz eingelesen, wird ein neuronales Netzwerk trainiert und weitere, dem Trainingdatensatz ähnliche Datensätze ohne Labels können damit kategorisiert werden, ob die Kategorisierung nun Sinn macht oder nicht. Der Benutzer der Applikation muss somit ein gewisses Wissen über den Verwendungszweck von neuronalen Netzwerken mitbringen, um sinnvolle Resultate zu erzielen.

# Anforderungen

## Quellen und Vorgehen

Die Ideen der Anforderungen an die Applikation wurde durch Studium eines online-Kurses (Machine Learning von Andrew Ng, https://www.coursera.org/learn/machine-learning#syllabus) und eines Buches (Neuronale Netze selbst programmieren von Tariq Rashid) ermittelt. Dort wird darauf verwiesen, dass es sehr viel Erfahrung braucht, die Hyperparameter für ein Modell zu setzen.

Zusammen mit dem Projektbetreuer wurden dann die Anforderungen an die Applikation spezifischer ermittelt.

## Funktionale Anforderungen

### Use Cases

Für die Applikation wurden folgende fünf Use Cases erarbeitet.

1. Datensatz einlesen: Hier werden entweder klassifizierte oder nicht klassifizierte Datensätze in einer bereits standardisierten Form eingelesen (CSV) und die Felder erkannt.
2. Modell automatisch parametrisieren und trainieren: Dies ist der Schwerpunkt der Applikation. Zu einem bestimmten Datensatz werden verschiedene Modelle getestet und das Model mit der geringsten Fehlerquote auf dem Testdatensatz (bzw. dessen Hyperparameter) ausgewählt. Hierbei ist ein Unterziel, dass der User vor einer zu langen Berechnung der Hyperparameter gewarnt wird.
3. Vergleich der Resultate der Hyperparametersuche in visueller Form.
4. Modell manuell parametrisieren und trainieren: Dieser Vorgang ist ähnlich wie Use Case 2, jedoch kann der Benutzer auch die Hyperparameter für seine Daten im Vornherein und ohne vorherige Optimierung angeben. Das Modell kann dann ebenfalls trainiert werden.
5. Auswertung nicht klassifizierter Daten durch ein bestehendes Modell: Sobald ein Modell trainiert wurde, soll es auch in der Lage sein, neue Datensätze auszuwerten und Ergebnisse zu liefern. Diese Ergebnisse werden dann in standardisierter Form ausgegeben.

Als Verdeutlichung dieser Anforderungen dient das Use Case Diagramm in Abbildung 3. Zusätzlich kann das GUI-Mockup Diagramm hinzugezogen werden, welches sich unter Punkt 8.1 befindet (Abbildung 4) Dieses soll verdeutlichen, wie sich der User in der Applikation bewegen kann.

### Use Case Diagramm

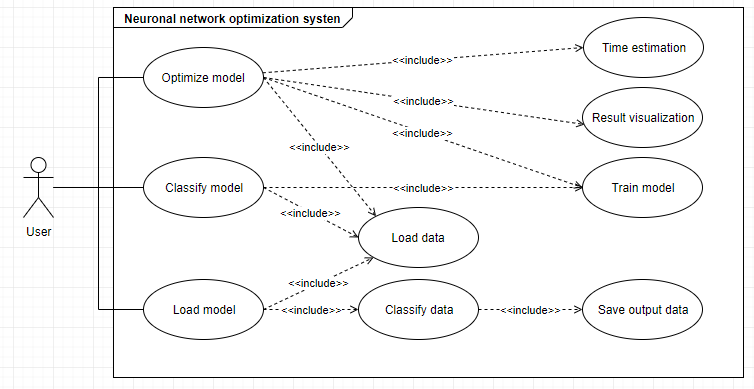


Abbildung 3: Use Case Diagramm der Optimierungsapplikation

### Detaillierte Anforderungen

Alle Muss-Prioritäten müssen vorhanden sein, um die Applikation als "ready" zu definieren (Minimum Viable Product, MVP).

Weitere Prioritäten werden nach Implementierung der Muss-Prioritäten angegangen (von P1 bis P3 absteigend).

#### Use Case 1 (Datensätze einlesen)

Im Use Case Diagramm (Abbildung 3) beinhaltet dies den Punkt "Load data", welcher basisbestandteil aller weiteren Use Cases ist.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Status** | **Prio** | **Beschreibung** |
| **F1.1** | Reviewed | Muss | Ein User hat die Möglichkeit, standardisierte Daten einzulesen. Diese Datei kann per Suche über die gängige Ordnerstruktur ausgewählt werden. |
| **F1.2** | Reviewed | Optional (P1) | Falls ein Datensatz nicht eingelesen werden kann, wird dies dem User gemeldet. (Mit einem Hinweis, weshalb es nicht funktioniert hat). Der Satz wird übersprungen, mit dem Einlesen wird fortgesetzt |
| **F1.3** | Reviewed | Optional (P1) | Vorschaudaten des eingelesenen Datensatzes werden zur Kontrolle angezeigt. |
| **F1.4** | Reviewed | Muss | Der User bestätigt, dass mit den eingelesenen Daten gearbeitet werden soll. |

#### Use Case 2 (Modell automatisch parametrisieren und trainieren)

Im Use Case Diagramm (Abbildung 3) beinhaltet dies die Punkte "Optimize model", "Time estimation" und "Train model"

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Status** | **Prio** | **Beschreibung** |
| **F2.1** | Reviewed | Muss | Der User kann die Funktion wählen, ein optimales Modell zu seinen Trainingsdaten erstellen zu lassen. Hierbei gelangt er in ein neues Menü. |
| **F2.2** | Reviewed | Muss | Der User kann bei den verschiedenen Hyperparametern entscheiden, wie gross die Range zum Testen sein soll. |
| **F2.3** | Reviewed | Muss | Der User kann abfragen, wie lange die Berechnung der optimalen Parameter mit den geladenen Training- und Testdaten sowie den Ranges für die Hyperparameter voraussichtlich dauert. Eine Schätzung der Laufzeit wird angezeigt. |
| **F2.4** | Reviewed | Muss | Der User wird vor Start der Berechnung darauf hingewiesen, falls die vorberechnete Dauer höher als ein definierter Grenzwert ist und kann dann entweder die Ranges anpassen oder muss erneut bestätigen, dass alles so durchgeführt werden soll. |
| **F2.5** | Reviewed | Muss | Das System liefert dem User ein mit optimalen Parametern versetztes und bereits trainiertes Modell, sowie die genauen Angaben zu den gefundenen optimalen Hyperparametern zurück. |

#### Use Case 3 (Vergleich von Ergebnissen der Parametrisierung)

Im Use Case Diagramm (Abbildung 3) beinhaltet dies den Punkt "Result visualization". Welche nach einer erfolgreichen Parametrisierung angeschaut werden kann.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Status** | **Prio** | **Beschreibung** |
| **F3.1** | Reviewed | Optional (P1) | Nach der Hyperparameteroptimierung kann der User Ergebnisse der Auswertung anzeigen lassen. Diese Anzeige visualisiert nach Möglichkeit die verschiedenen Hyperparameterkombinationen und die damit erzeugte Trefferrate. |
| **F3.2** | Reviewed | Optional (P3) | Der User kann bei dieser Visualisierung nach Hyperparameter filtern. |

#### Use Case 4 (Modell manuell parametrisieren und trainieren)

Im Use Case Diagramm (Abbildung 3) beinhaltet dies die Punkte "Classify model" sowie "Train Model.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Status** | **Prio** | **Beschreibung** |
| **F4.1** | Reviewed | Optional (P2) | Ein User soll auch in der Lage sein, ein Modell nach eigenen Kriterien zu trainieren. Anstatt dass die optimalen Hyperparameter ausgewertet werden, kann der User die Parameter selber eingeben. |
| **F4.2** | Reviewed | Optional (P2) | Nach Bestätigung wird auch hier das Modell mithilfe der eingelesenen Datensätze trainiert |

#### Use Case 5 (Auswertung nicht klassifizierter Daten durch ein bestehendes Modell)

Im Use Case Diagramm (Abbildung 3) beinhaltet dies die Punkte "Load Model", "Classify data" und "Save output data".

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Status** | **Prio** | **Beschreibung** |
| **F5.1** | Reviewed | Optional (P2) | Zu einem bereits trainierten Modell können unklassifizierte Daten hinzugeladen werden. |
| **F5.5** | Reviewed | Optional (P2) | Die Daten werden anhand dieses Modells klassifiziert. Als Output wird wieder ein Standardisiertes File (csv) generiert. |

## Qualitätsanforderungen

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Status** | **Prio** | **Beschreibung** |
| **Q1.1** | Reviewed | Muss | Usability: Alle notwendigen Schritte zum Erstellen, Optimieren und testen eines Neuronalen Netzes sind in einem einheitlichen GUI implementiert. |
| **Q1.2** | Reviewed | Muss | Zuverlässigkeit: Die gelieferten Daten sollen deterministisch und auch reproduzierbar sein. |
| **Q1.3** | Reviewed | Muss | Es muss eine Laufzeitabschätzung mit einer angestrebten Genauigkeit von +/- 25% der geschätzten Zeit als Warnungsgrundlage vor jeder Berechnung gemacht werden. |

# Glossar

|  |  |
| --- | --- |
| Neuronales Netzwerk | Das künstliche neuronale Netz (KNN) ist bis zu einem gewissen Grad dem Aufbau des biologischen Gehirns nachempfunden. Es besteht aus einem abstrahierten Modell miteinander verbundener Neuronen, durch deren spezielle Anordnung und Verknüpfung sich Anwendungsprobleme aus verschiedenen Bereichen wie der Statistik, der Technik oder der Wirtschaftswissenschaften computerbasiert lösen lassen. Das neuronale Netz ist ein Forschungsgegenstand der Neuroinformatik und Teilgebiet der künstlichen Intelligenz. Neuronale Netze müssen, bevor sie Problemstellungen lösen können, trainiert werden.[[1]](#footnote-1) |
| Layer | Die *Neuronen* (auch *Knotenpunkte*) eines künstlichen neuronalen Netzes sind schichtweise in sogenannten *Layers* angeordnet und in der Regel in einer festen Hierarchie miteinander verbunden. Die Neuronen sind dabei zumeist zwischen zwei Layers verbunden (*Inter-Neuronlayer-Connection*), in selteneren Fällen aber auch innerhalb eines Layers (*Intra-Neuronlayer-Connection*).  Zwischen den Layers oder Schichten ist jedes Neuron der einen Schicht immer mit allen Neuronen der nächsten Schicht verbunden.  Beginnend mit der *Eingabeschicht* (*Input Layer*) fließen Informationen über eine oder mehrere *Zwischenschichten* (*Hidden Layer*) bis hin zur *Ausgabeschicht* (*Output Layer*). Dabei ist der Output des einen Neurons der Input des nächsten.[[2]](#footnote-2) |
| Hyperparameter | Ein Hyperparameter ist ein Parameter, der zur Steuerung des Trainingsalgorithmus verwendet wird und dessen Wert im Gegensatz zu anderen Parametern vor dem eigentlichen Training des Modells festgelegt werden muss.[[3]](#footnote-3) Beispiele für Hyperparameter sind: Anzahl hidden Layers, Anzahl Knoten pro Layer, Learning Rate (für den Optimierungsalgorithmus), etc. |
| Klassifizierte Daten | (Englisch auch labelled data genannt) Klassifizierte Datensätze enthalten eine Anzahl von Fällen, die mit bestimmten Labels versehen sind. Ein Beispiel wäre ein Datensatz in welchem pro Zeile die Pixelwerte eines Fotos enthalten sind (Spalten 1-256) und in einer weiteren Spalte der Wert 0 oder 1 ob es sich um ein Katzenfoto oder nicht handelt.  Klassifizierte Datensätze werden für das Training und das Testen von Modellen verwendet. |
| Testdaten | Testdaten sind klassifizierte Datensätze und werden dazu verwendet, ein bereits trainiertes Modell zu prüfen. Dabei wird mit dem neuronalen Netzwerk eine Klassifikation der Daten durchgeführt. Diese Klassifizierung wird dann mit der wahren Klassifikation der Fälle aus dem Testdatensatz verglichen. Es kann somit ein Fehlerquote angegeben werden, wie viele Fälle das neuronale Netzwerk eine korrekt klassifiziert hat. |
| Trainingdaten | Klassifizierte Daten, mit welchen das neuronale Netzwerk trainiert wird. |

# Literaturverzeichnis

**Neuronale Netze selbst Programmieren – ein verständlicher Einstieg mit Python**

*Rashid, Tariq, Neuronale Netze selbst Programmieren – ein verständlicher Einstieg mit Python, O’Reilly Media Inc., übersetzt aus dem Englischen von dpunkt.verlag GmbH, Heidelberg (D), 1. Ausgabe, 2017*

# Anhang

## GUI-Mockup-Zeichnung

Die GUI-Mockup-Zeichnung soll verdeutlichen, wie sich ein User in dem Programm bewegen kann.

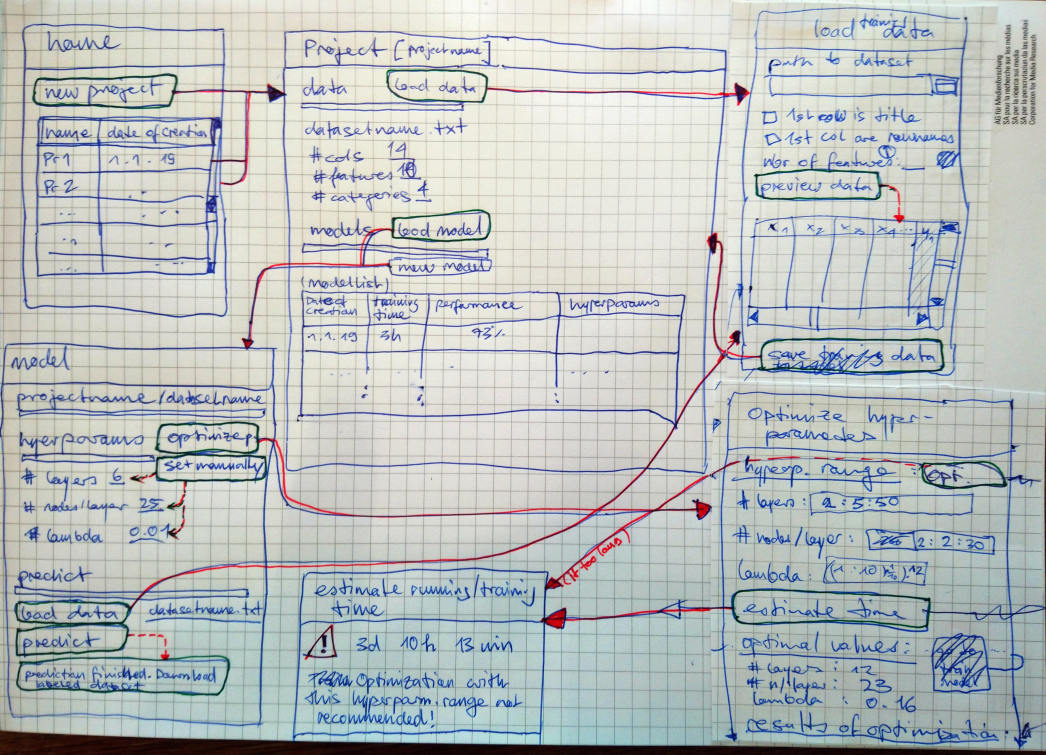


Abbildung 4: GUI-Mockup Zeichnung der Optimierungsapplikation

## UML Klassendiagramm

Das Klassendiagramm (stand vom 03.10.2019) dient zur Verdeutlichung, wie die verschiedenen Programmklassen zusammenarbeiten. Das Diagramm ist unter <https://github.com/lukaszoller/hyperOptimize/blob/master/1%20Requirements/UML_Diagram.png> ersichtlich.

# Versionskontrolle

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Datum** | **Beschreibung** | **Autor** |
| X0.1 | 26.09.2017 | Dokument erstellt (Punkte 1 bis 3) | L.Z., R.H. |
| X0.2 | 30.09.2019 | Punkte 1 bis 3 überarbeitet nach Review der Peer-Group. | R.H. |
| X0.3 | 09.10.2019 | Kapitel 4 erarbeitet | R.H. |
| X0.4 | 09.10.2019 | Review L.Z., an Projektbetreuer zum Review weitergeleitet. | L.Z. |
| X0.5 | 16.10.2019 | Feedback zu Kapitel 4 von Projektbetreuer eingebaut. | R.H. |
| X0.6 | 20.10.2019 | Kapitel 5 erarbeitet an L.Z. weitergeleitet. | R.H. |
| X0.7 | 25.10.2019 | Nach Feedback von Gerhard Schwab die angesprochenen Punkte angepasst. An Projektbetreuer zum Review weitergeleitet. | L.Z., R.H. |
| X0.8 | 28.11.2019 | Review von L.Z. eingebaut. | L.Z. |
| X0.9 | 29.11.2019 | Nach Review von Projektbetreuer, die ihm noch fehlenden Punkte eingebaut. | R.H. |
| V1.0 | 29.11.2019 | Fertiges Dokument für Abgabe |  |

1. <https://www.bigdata-insider.de/was-ist-ein-neuronales-netz-a-686185/> (Zuletzt abgerufen am 20.10.2019) [↑](#footnote-ref-1)
2. <https://jaai.de/kuenstliche-neuronale-netze-aufbau-funktion-291/> (Zuletzt abgerufen am 20.10.2019) [↑](#footnote-ref-2)
3. <https://de.wikipedia.org/wiki/Hyperparameteroptimierung> (Zuletzt abgerufen am 20.10.2019) [↑](#footnote-ref-3)