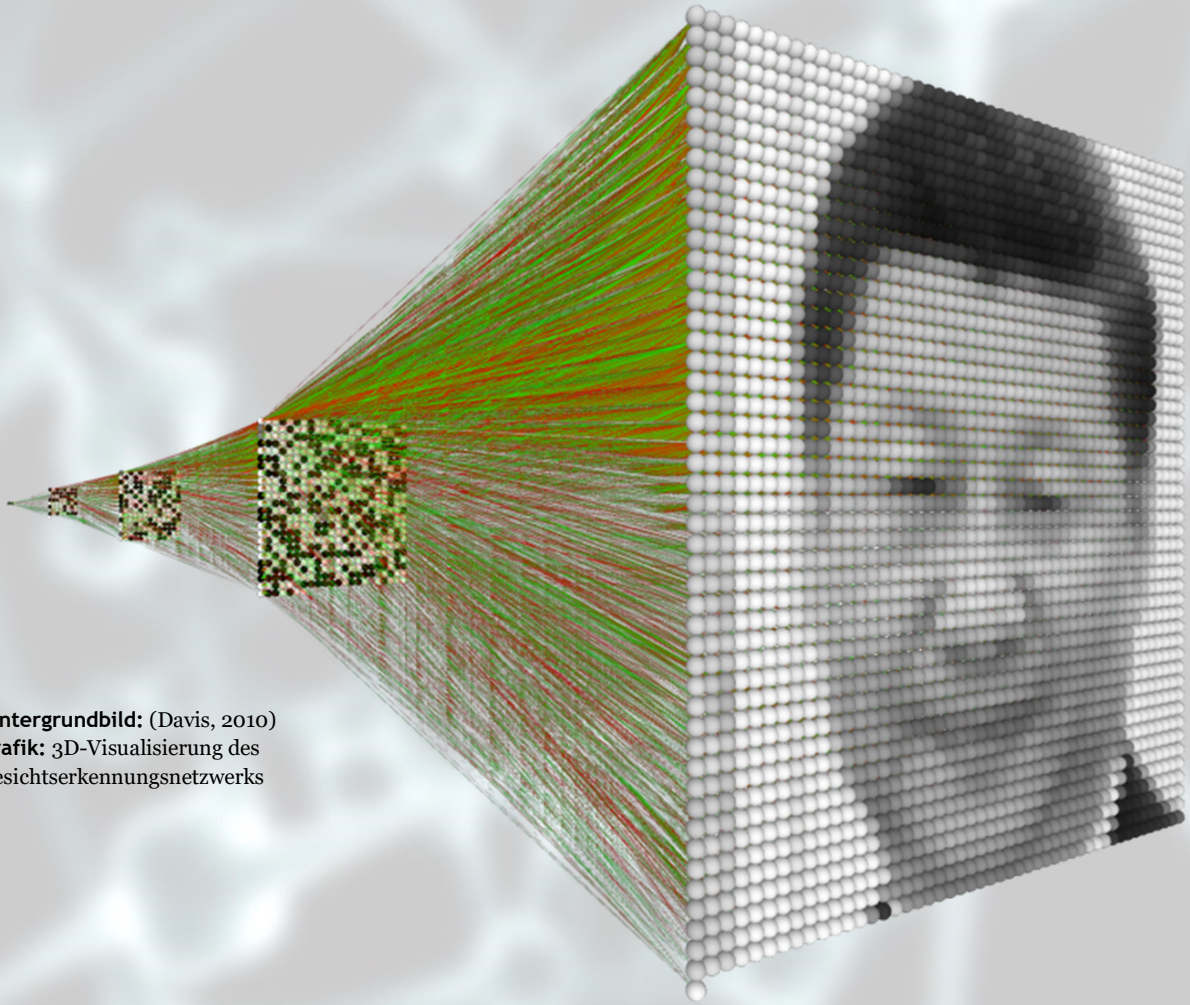
Implementierung eines künstlichen neuronalen Netzwerks zur Gesichtserkennung

Maturaarbeit 2017
Informatik an der Kantonsschule Sursee



Hintergrundbild: (Davis, 2010)
Grafik: 3D-Visualisierung des Gesichtserkennungsnetzwerks

Autor

Fabian Bösiger
Weingartweg 13
6205 Eich

Betreuer

Abdelhakim Ghezal
Oberdorfstrasse 22
6207 Nottwil

Abstract

Künstliche neuronale Netzwerke sind Algorithmen, welche gut geeignet sind, Daten zu klassifizieren. Sie sind wie ihr biologisches Vorbild, das Gehirn, aufgebaut: Neuronen sind miteinander über verschieden gewichtete Verbindungen verknüpft. Die Gewichte der Verbindungen werden durch ein Trainingsverfahren so verändert, das die Fehlerrate bei der Klassifizierung eines Trainingsdatensatzes möglichst klein wird. Wenn eine kleine Fehlerrate erreicht wurde, kann das künstliche neuronale Netz dazu verwendet werden, Daten zu klassifizieren, die nicht im ursprünglichen Trainingsdatensatz enthalten sind. In dieser Arbeit wird ein künstliches neuronales Netzwerk implementiert und anschliessend trainiert, um Gesichter zu erkennen.

Vorwort

Ein grosser Teil der Informatik beschäftigt sich damit, unsere Umwelt nachzuahmen. Physikalische Simulationen, welche das Wetter voraussagen, Computerspiele realer wirken lassen oder Flugzeuge aerodynamischer designen haben schon längst einen Platz in unserem Alltag. Doch wer die Nachrichten verfolgt, hat sicherlich schon etwas von selbstlernenden Programmen gehört. Mit dem Gehirn als Vorbild sind sie in der Lage, selbstständig Zusammenhänge zu erkennen und Schlussfolgerungen zu ziehen. Um diese selbstlernenden Programme geht es in dieser Arbeit; sogenannte künstliche neuronale Netze. Ich fand Simulationen schon immer faszinierend, denn mit ihnen kann man nicht nur in die Vergangenheit blicken, sondern auch die Zukunft voraussagen. Angenommen, man könnte jedes Naturgesetz in eine Simulation übernehmen, gäbe es dann überhaupt noch einen Unterschied zwischen unserer „realen“ Welt und der Simulierten? Hätten simulierte Menschen genauso ein Bewusstsein wie „reale“ Menschen? Oder sind wir es, die in einer Simulation leben? Natürlich sind solche Fragen weit hergeholt und gehören mehr der Philosophie an, als den Naturwissenschaften, doch genau diese Fragen machen das Thema künstliche Intelligenz und Simulationen so spannend für mich.

Ich möchte hiermit allen Personen danken, welche diese Arbeit ermöglichten. Ein spezieller Dank gilt Herrn Ghezal für die Betreuung dieser Arbeit sowie meiner Familie für die Bereitstellung der Portraits.

Inhalt

- *Quellenverweise in der Fusszeile: Idee/Inhalt von entsprechender Quelle*
- *Es werden verschiedene Begriffe verwendet, um künstliche neuronale Netze zu beschreiben. Sämtliche Begriffe verweisen, falls nicht ausdrücklich als „natürliches neuronales Netz“ beschrieben, auf ein künstliches Netz.*

1 Einleitung

1.1 Das künstliche neuronale Netzwerk

1.1.1 Biologisches Vorbild

Ein künstliches neuronales Netzwerk ist seinem Vorbild, dem natürlichem neuronalen Netz, also dem Gehirn nachempfunden. Wie sein biologisches Vorbild besteht ein künstliches neuronales Netz aus Neuronen und Verbindungen zwischen Neuronen. Die Signale der Neuronen werden über diese Verbindungen weitergeleitet. Aus tausenden Verbindungen entstehen Netze, die in der Lage sind, verschiedenste Aufgaben auszuführen.[1]

Das künstliche neuronale Netz hat jedoch viel weniger Neuronen und Verbindungen als sein natürliches Vorbild und ist deshalb auch weniger leistungsfähig. Ausserdem funktionieren die Verbindungen zwischen den Neuronen von künstlichen Neuronalen Netzwerken oft mit Zahlen zwischen Eins und Null, nicht mit Wahr/Falsch-Werten so wie beim natürlichen neuronalen Netzwerk.

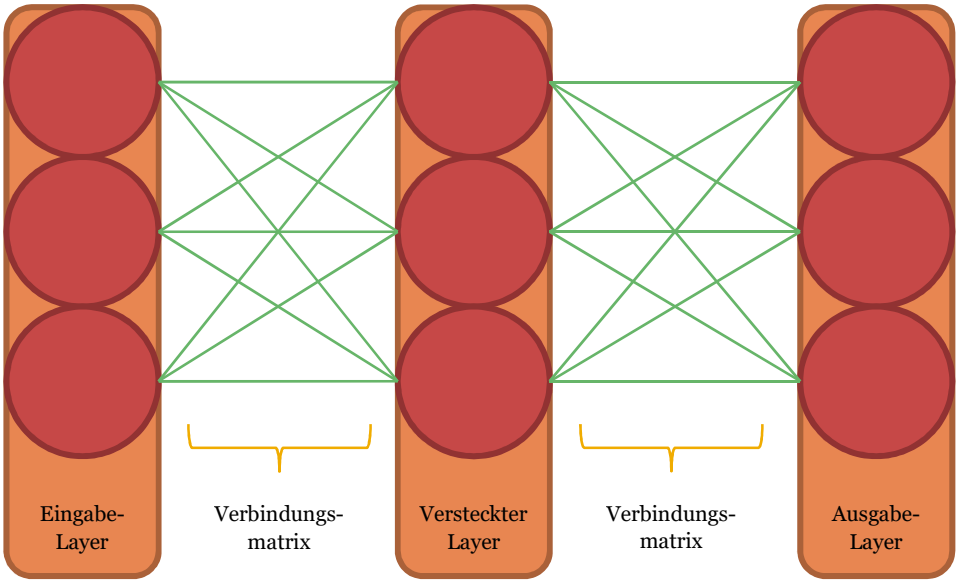


Abbildung 1: Darstellung eines künstlichen neuronalen Netzwerks. Die roten Kreise stellen Neuronen dar und die grünen Linien die Verbindungen.

1.1.2 Typen von künstlichen neuronalen Netzen

Je nach Anwendung können verschiedene Typen von künstlichen neuronalen Netzen implementiert werden. Die zwei grundlegenden Strukturen sind die Feedforward-Netze und die Rekurrenten Netze.[1]

[1] (Wikipedia, 2017)

Feedforward-Netze besitzen beliebig viele Schichten (auch Layer genannt). Die Daten werden von einer Schicht über verschieden gewichtete Verbindungen zur nächsten Schicht weitergeleitet. Es existieren keine Verbindungen, welche zurückführen.[2]

Rekurrente Netze können, im Gegensatz zu den Feedforward-Netzen, Informationen wieder in vorherige Schichten leiten. Ein Rekurrentes Netz besitzt somit eine Art Kurzzeitgedächtnis.[3]

Um Gesichter zu erkennen reicht ein Feedforward-Netz, da das Netzwerk keine Informationen der vorherigen Eingabe benötigt. Es muss nicht wissen, welche Person vor zwei Minuten vor der Kamera stand, um die aktuelle Person zu identifizieren.

1.1.3 Aufbau

Das künstliche neuronale Netzwerk besteht aus mehreren Layern, in denen sich mehrere Neuronen befinden. Jedes Neuron eines Layers ist mit allen Neuronen des nächsten Layers verbunden. Diese Verbindungen sind gewichtet. Wenn ein Signal eines Neurons über eine Verbindung zum nächsten Neuron geschickt wird, wird das Signal mit der Gewichtung der Verbindung multipliziert.[4]

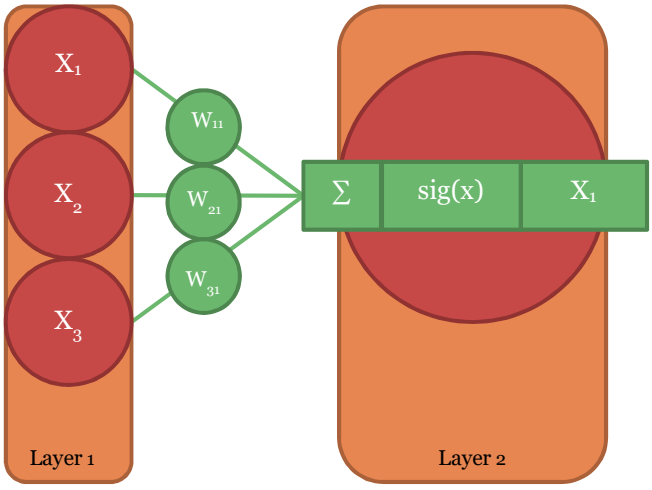


Abbildung 2: Diese Abbildung veranschaulicht, wie der Wert eines Layers durch die Werte des vorherigen Layers berechnet wird.

Der Wert eines Neurons des zweiten Layers hängt von den Werten der Neuronen des ersten Layers ab. Zuerst werden die Werte der Neuronen des ersten Layers mit den Gewichten ihrer jeweiligen Verbindungen multipliziert, also X_1 mit W_{11}, X_2 mit W_{21} und X_3 mit W_{31}. Anschliessend werden die daraus resultierenden Werte addiert. Damit der neue Wert zwischen 0 und 1 bleibt, wird dieser durch eine Sigmoidfunktion gegeben. Die Ausgabe der Sigmoidfunktion ist dann der Wert des Neurons des zweiten Layers.

Der Layer, welcher die Informationen in das Netz gibt (im Fall der Gesichtserkennung die Schwarzweiss-Werte des Eingabebilds), wird Eingabelayer genannt. Die darauf folgenden

[2] (Wikipedia, 2017)
[3] (Wikipedia, 2017)
[4] (Wikipedia, 2017)

Layer werden Versteckte Layer genannt, da nicht genau bestimmt werden kann, welche Eigenschaften die Werte der Neuronen repräsentieren. Aus dem Letzten Layer, dem Ausgabelayer, kann das vom Netzwerk bestimmte Resultat abgelesen werden (In diesem Fall die Namen der Testpersonen, da jedes Ausgabeneuron für die Wahrscheinlichkeit einer Person steht).

1.1.4 Anwendungsbereiche von künstlichen neuronalen Netzen

In praktisch allen Anwendungsbereichen, in denen eine grossen Menge von Eingabewerte in eine vergleichsweise kleine Anzahl Ergebnisse überführt werden muss, können künstliche neuronale Netze angewendet werden. Deshalb sind künstliche neuronale Netze sehr gut für Text-, Bild- und Gesichtserkennung geeignet, da die Bildpunkte direkt als Eingabewerte dienen. Auch bei Vorhersagen, bei denen viele Daten miteinbezogen werden, können künstliche neuronale Netze eingesetzt werden, beispielsweise bei der Abschätzung wirtschaftlicher Prozesse (vor allem bei der Vorhersage von Aktienkursen) oder bei Wettervorhersagen.[5]

1.1.5 Lernmethode

Die Anpassung der Gewichte der Verbindungen zwischen den Neuronen wird lernen oder trainieren genannt. Dieser Teil des künstlichen neuronalen Netzwerks ist besonders wichtig, da das Netzwerk zu Beginn zufällig generiert wird und erst danach versucht wird, die Gewichte so anzupassen, dass die Eingabewerte korrekt interpretiert werden. Je länger das Netz trainiert wird, desto tiefer sinkt der Netzwerkfehler.

Man kann sich den Fehler als Funktion von sämtlichen Gewichten des Netzwerks vorstellen. Der Wert dieser Fehlerfunktion sollte so tief wie möglich sein, es muss also der Tiefpunkt der Funktion gefunden werden. Doch bei einem künstlichen neuronalen Netzwerk können mehrere zehntausende Gewichte vorkommen, das heisst die Fehlerfunktion ist von mehreren zehntausend Variablen abhängig. Bei so vielen Unbekannten ist eine „traditionelle“ Suche nach dem Tiefpunkt ausgeschlossen. Man kann sich das wie ein Navigationssystem vorstellen, dass die kürzeste Route durch ein Gebiet mit zehntausenden Baustellen sucht, aber jedes Mal wenn man abzweigt verlegen sich sämtliche Baustellen. Man kann unmöglich vorausberechnen, wie sich die Baustellen verändern, wenn man abzweigt.

Um dieses Problem zu lösen werden alle, bis auf das zu korrigierende Gewicht als Konstanten behandelt. So kann man erkennen, ob man das Gewicht erhöhen oder verringern muss, um den Netzwerkfehler zu reduzieren. Das Gewicht wird bei diesem Lösungsansatz aber nur minimal verändert, da grössere Veränderungen doch wieder negative Auswirkungen auf den Netzwerkfehler haben können. Bei dieser Methode kann jedoch nicht garantiert werden, ob wirklich das globale Minimum gefunden wurde oder ob es sich nur um ein lokales Minimum handelt. Das Beispiel mit dem Navigationssystem verändert sich folgendermassen. Um den kürzesten Weg durch das Gebiet mit den zehntausenden Baustellen zu suchten, wirft man das Navigationssystem am besten aus dem Fenster, schaut bei jeder Kurve vorsichtig voraus, ob auf diesem Weg eine Baustelle ist, und wählt die Route, bei der man keine Baustelle sieht. Manchmal kann es aber vorkommen, dass bei allen Strassen einer Abzweigung eine Baustelle steht und man muss von vorne beginnen. Dieses „Feststecken“ ist aber kein grosses Problem.

[5] (Wikipedia, 2017)

Wenn erkennbar ist, dass das Netz einen gewissen Fehler nicht mehr reduzieren kann, ist anzunehmen, dass das Netzwerk ein lokales Minimum und kein globales Minimum gefunden hat und feststeckt. Dann kann die Trainingsphase einfach von Anfang an wiederholt werden.

Um den Netzwerkfehler zu reduzieren wird also folgendermassen vorgegangen. Zuerst wird eine zufällige Verbindung ausgewählt, deren Gewichtung verbessert werden soll. Danach wird der aktuelle Fehler des Netzwerks berechnet. Anschliessend wird das Gewicht der Verbindung vergrössert und der Fehler wird noch einmal gemessen, danach wird das Gewicht der Verbindung gesenkt und der Fehler wird wiederum gemessen. Wenn einer der gemessenen Fehler, entweder der bei einer Vergrösserung oder der bei einer Verkleinerung des Gewichts, kleiner ist als er aktuelle Fehler, wird das Gewicht in die entsprechende Richtung korrigiert.

1.2 Andere Gesichtserkennungsmethoden

1.2.1 Viola-Jones-Methode

Die Viola-Jones-Methode wurde populär als die ersten Digitalkameras aufkamen, da die Methode sehr effizient ist und deshalb genutzt wurde, um die Schärfe bei Kameras automatisch einzustellen, die über nicht so viel Rechenleistung verfügen. Die Methode beruht auf verschiedenen Basismustern, welche im zu analysierenden Bild gesucht werden. Diese Basismuster erkennen Eigenschaften, die bei allen Menschen ähnlich sind. Beispielsweise ist der Bereich der Augen dunkler als der Wangenbereich. Wenn die Basismuster über das zu analysierende Bild geschoben werden, kann anschliessend bestimmt werden, wo sich mit grosser Wahrscheinlichkeit ein Gesicht befindet. Der Nachteil dieser Methode besteht darin, dass die Gesichter einer kleinen Drehung des schon nicht mehr erkennbar sind. Ausserdem ist diese Methode sehr empfindlich auf verschiedene Lichtbedingungen. Diese Methode wird hauptsächlich verwendet, um den Ort eines Gesichts im Bild zu finden, und nicht die Person zu identifizieren.[6]

1.2.2 Geometrische Vermessung

Bei dieser Methode werden Distanzen von verschiedenen Punkten auf dem Gesicht gemessen und danach mit einer Datenbank verglichen. Diese Methode benötigt jedoch wiederum einen Algorithmus zur Erkennung der Punkte auf dem Gesicht. Die Geometrische Vermessung kann auch in drei Dimensionen angewendet werden, falls 3D-Modelle der Gesichter vorliegen. Diese Methode ist nur effektiv wenn die Gesichtspunkte präzise gemessen werden können.[7]

1.2.3 Weitere Methoden

Gesichtserkennungsalgorithmen können sich auch auf weitere Kriterien eines Gesichts stützen, beispielsweise auf ovale Formen, sich bewegende Objekte oder hautfarbenähnliche Farben innerhalb des zu analysierenden Bilds.[8] Diese Methoden sind jedoch nicht empfehlenswert, da sie auch auf andere Objekte zutreffen können.

---

[6] (Wikipedia, 2015)
[7] (Wikipedia, 2017)
[8] (Wikipedia, 2014)

2 Vorgehensweise

2.1 Beschaffung von Portraits

2.1.1 Anzahl Bilder

Es existiert keine konkrete Regel, wie viele Bilder ideal sind, um neuronale Netzwerke zu trainieren. Es sind jedoch folgende Einflüsse zu beachten.

- Bei einer Zunahme der Trainingsbilder steigt auch die Trainingszeit.
- Bei einer Zunahme der Trainingsbilder sinkt die Wahrscheinlichkeit, dass das Netzwerk die Bilder nur auswendig lernt (Overfitting).
- Bei einer Zunahme der Trainingsbilder steigt die Anpassungsfähigkeit des Netzwerks, das heisst Beleuchtung, Entfernung und Ausrichtung des Gesichts spielen eine kleinere Rolle.

2.1.2 Aufnahme der Bilder

Die Bilder sollten alle dasselbe Format haben, um dem Netzwerk einen optimalen Lernprozess zu ermöglichen. Die Bilder sollten deshalb mit einem weissen Hintergrund und mit gleichem Abstand zwischen der Kamera und dem zu fotografierenden Objekt aufgenommen werden. Die Kamera wird so ausgerichtet, dass sich die Augen der Testperson in der Mitte des Bildes befinden. Dazu wurde ein eigens dafür entwickeltes Programm genutzt, siehe Kapitel 6.1.4.

2.1.3 Bilder aus dem Internet

Um das Netzwerk mit verschiedenen Datensätzen zu testen, stehen auch Trainingsdatensätze von Berühmten Persönlichkeiten zur Verfügung. Ein Datensatz beinhaltet die Bundesräte sowie die Bundeskanzlerin von 2012 bis 2016, ein Anderer die Präsidenten von Amerika, Russland und China. Sämtliche Bilder aus dem Internet wurden, wie bei den selbst aufgenommenen Bildern so zugeschnitten, so dass sich die Augen der Testpersonen immer an derselben Stelle befinden.

2.2 Programmierung des neuronalen Netzwerks

Zur Implementierung wird die Programmiersprache Processing verwendet. Sämtliche Befehle können auf der Processing-Referenz-Seite nachgeschaut werden. Es wurden zwei Libraries verwendet. Die Erste, die Video-Library von Processing, wird benötigt, um auf die Webcam zuzugreifen und somit Live-Bilder zu analysieren. Die zweite Library nennt sich „Drop“. Mit dieser Library lassen sich Medien in das Programmfenster ziehen, um sie zu laden.

Die Processing Source Code Datei, in der sich die Funktionen „setup“ und „draw“ befinden, muss sich in einem gleichnamigen Ordner befinden. In diesem Ordner befindet sich ausserdem der „data“-Ordner, in dem Dateien abgelegt sind, auf welche das Programm zugreifen kann.

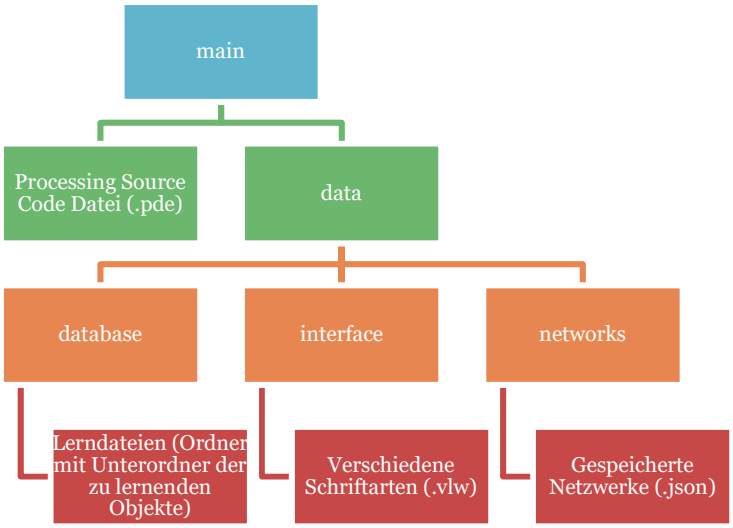


Abbildung 3: Struktur der Unterordner, auf welche das Programm zugreift.

2.2.1 Struktur

Jedes Programm besitzt eine Struktur, welche am besten möglich übersichtlich und logisch ist. Die Struktur hilft nicht nur dem Programmierer, sondern ermöglicht auch Aussenstehenden, den Code besser nachzuvollziehen. Eine grobe Struktur sollte schon vor dem Beginn des Programmierens vorhanden sein.

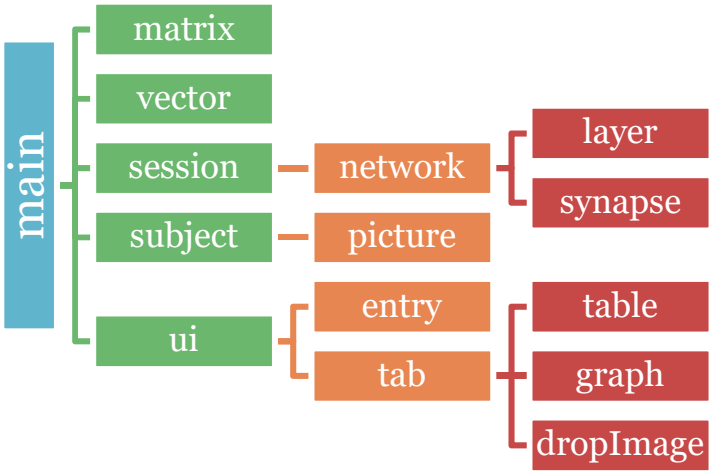


Abbildung 4: Systematische Auflistung aller Klassen, welche im Programm vorhanden sind.

2.2.2 Download der Libraries

Die Libraries können direkt im Processing Code Editor heruntergeladen werden. Dazu klicken Sie in der Menüleiste oben auf „Tools“ und anschliessend auf „Tool hinzufügen“. Im Fenster, welches sich anschliessend öffnet, klicken Sie auf den „Libraries“-Tab. Von dort aus können sie über den Filter einfach die Libraries suchen und anschliessend mit einem Klick auf den „Install“-Button installieren. Die benötigten Libraries finden Sie unter den Namen „Drop“ für die Drop-Library und „Video“ für die Kamera-Library.

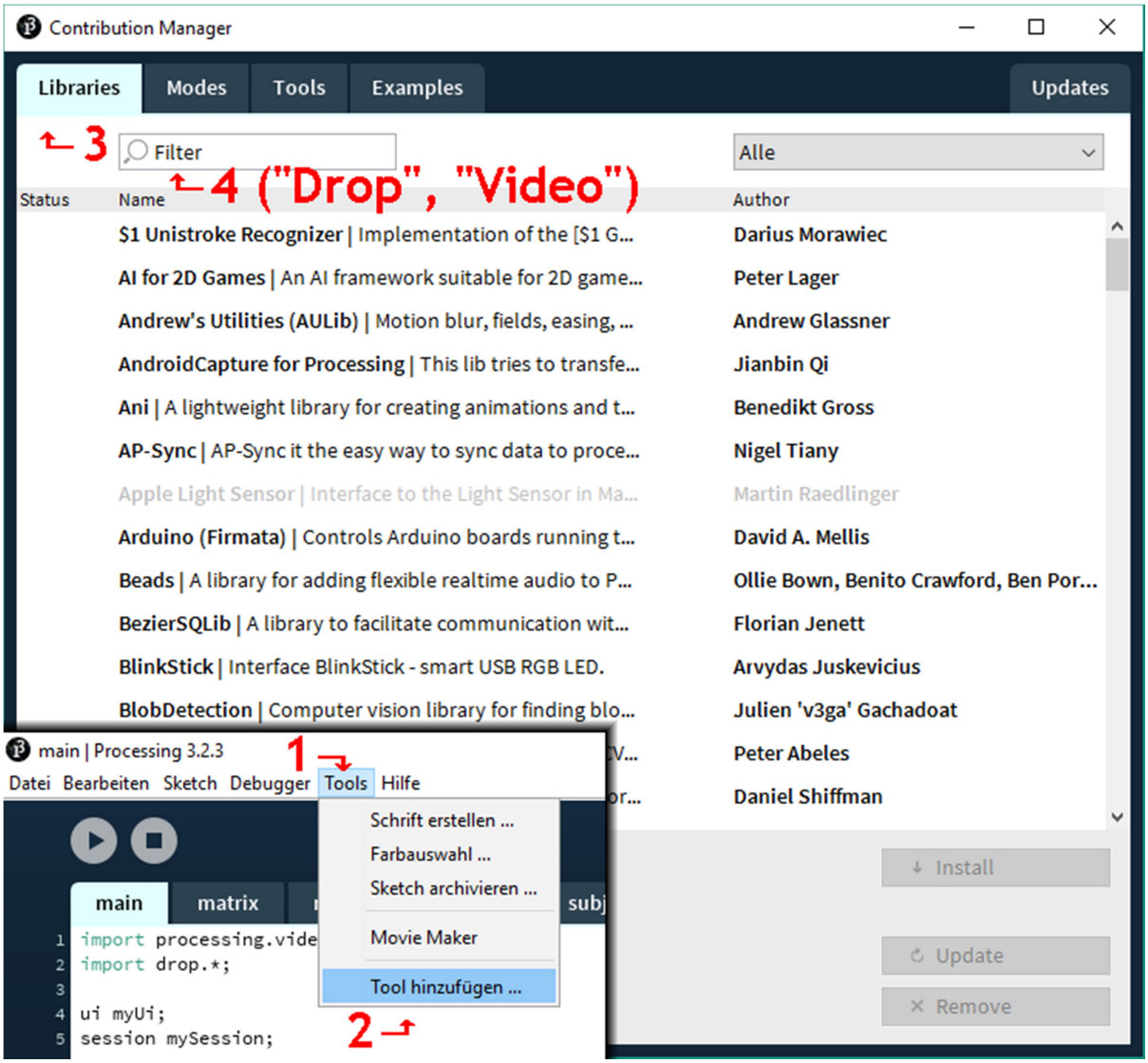


Abbildung 5: Anleitung zur Installation der benötigten Libraries.

2.2.3 Vektor- und Matrixklassen

Um eine einfache Verarbeitung von Informationen zu gewährleisten, werden sämtliche Werte des Netzwerks, beispielsweise die Gewichte der Synapsen oder die Werte der Neuronen, in Matrizen oder Vektoren gespeichert. Dazu werden Vektor- und Matrixklassen implementiert.

In der folgenden Tabelle wird der Teil des Programms erläutert, der für die Vektorklasse zuständig ist und somit alle wichtigen Funktionen, welche Vektoren verarbeiten, enthält. Zu finden ist dieser Teil des Programms unter dem „vector“-Tab.

```
class vector{
  float[] values;

  vector(float[] values){
    this.values = values;
  }
  matrix toMatrix(){
```

Vektorklasse erstellen
Variablen für die Werte erstellen.

Funktion, um einen Vektor in eine Matrix

```
                                                                    umzuwandeln
  matrix output = null;                                             Ausgabematrix
    float sizeFloat = sqrt(values.length);                          erstellen.
    if(sizeFloat == int(sizeFloat)){                                Falls die Grösse des
                                                                    Vektors eine
                                                                    Quadratzahl ist, ist der
                                                                    Vektor umwandelbar.
      int size = int(sizeFloat);
      float[][] values = new float[size][size];                     Die Dimensionen der
                                                                    Matrix betragen die
                                                                    Wurzel der Grösse des
                                                                    Vektors.
      for(int i = 0; i < values.length; i++){
        for(int j = 0; j < values[i].length; j++){
          values[i][j] =
this.values[i+j*values.length];
        }
      }
      output = new matrix(values);
    }
    return output;
  }
  float addValues(){                                                Funktion, welche die
                                                                    einzelnen Vektorwerte
                                                                    zusammenrechnet und
                                                                    ausgibt.
    float output = 0;
    for(int i = 0; i < values.length; i++){
      output = output + values[i];
    }
    return output;
  }
}
vector zerosVector(int size){                                       Funktion um einen
                                                                    Nullvektor zu erstellen.
  float[] values = new float[size];
  for(int i = 0; i < values.length; i++){
    values[i] = 0;
  }
  return new vector(values);
}
vector imageToVector(PImage inputImage, int size){                  Funktion, um die Werte
                                                                    einer Grafik in einen
                                                                    Vektor umzuwandeln.
  PImage image =
createImage(inputImage.width,inputImage.height,RGB);
  image.pixels = inputImage.pixels;
  image.resize(size,size);                                          Die Grösse der Grafik
                                                                    wird geändert.
```

```
  float[] values = new float[image.pixels.length];
  for(int i = 0; i < values.length; i++){
    values[i] = brightness(image.pixels[i])/256.0;
  }
  return new vector(values);
}
vector vectorize(int index, int size){
  float[] values = new float[size];
  for(int i = 0; i < values.length; i++){
    if(i == index){
      values[i] = 1;
    }else{
      values[i] = 0;
    }
  }
  return new vector(values);
}
int indexize(vector input){
  float max = 0;
  int index = -1;
  for(int i = 0; i < input.values.length; i++){
    if(input.values[i] >= max){
      max = input.values[i];
      index = i;
    }
  }
  return index;
}
```

Die Farblichen Informationen des Bildes werden absichtlich weggelassen, da die Farbe stark von der Uhrzeit, der Ausrichtung der Kamera, der Witterung und weiteren Faktoren abhängig sein kann.

Funktion, um einen Vektor zu erstellen, bei dem alle Werte null sind ausser an einer bestimmten Stelle.

Funktion, welche den Index des Maximalwerts eines Vektors weitergibt.

Da zwischen jedem Neuron zweier Layer eine Verbindung existiert, können die Werte der Verbindungen als Matrix dargestellt werden, wobei die Anzahl Spalten von der Anzahl Neuronen im nächsten Layer und die Anzahl Zeilen von der Anzahl im gegebenen Layer abhängt. Um die Werte der Neuronen des nächsten Layers zu berechnen, kann einfach das Matrix-Vektor-Produkt berechnet werden.

Gegebener Layer		Verbindungen, die als Matrix dargestellt werden		Nächster Layer (Vor der Bearbeitung durch die Sigmoid-Funktion)
$\begin{pmatrix} 0 \\ 0.5 \\ 1 \end{pmatrix}$	➔	$\begin{pmatrix} -2 & 5 \\ -4 & -6 \\ 7 & 4 \end{pmatrix}$	➔	$\begin{pmatrix} 0 * -2 + 0.5 * -4 + 1 * 7 = 5 \\ 0 * 5 + 0.5 * -6 + 1 * 4 = 1 \end{pmatrix}$

In der nächsten Tabelle wird der Programmcode erklärt, der für die Matrizen zuständig ist. Die meisten wichtigen Matrizenrechnungen finden in diesem Teil des Programms statt. Zu finden ist dieser Teil unter dem „matrix“-Tab.

```
class matrix{
  float[][] values;


  matrix(float[][] values){
    this.values = values;
  }
}
matrix zerosMatrix(int sizeX, int sizeY){

  float[][] values = new float[sizeX][sizeY];
  for(int i = 0; i < values.length; i++){
    for(int j = 0; j < values[i].length; j++){
      values[i][j] = 0;
    }
  }
  return new matrix(values);
}
matrix randomMatrix(int sizeX, int sizeY){




 float[][] values = new float[sizeX][sizeY];
  for(int i = 0; i < values.length; i++){
    for(int j = 0; j < values[i].length; j++){
      values[i][j] = random(-1,1);
    }
  }
  return new matrix(values);
}
JSONObject matrixToJSON(matrix input){




  JSONObject output = new JSONObject();
  output.setInt("x",input.values.length);
```

Matrixklasse

Werte der Matrix werden in diesen Variablen gespeichert.

Funktion, um eine Nullmatrix zu erstellen.

Funktion, um eine zufällige Matrix zu erstellen (Später hilfreich beim Erstellen eines zufälligen Netzwerks).

Funktion, um eine Matrix in ein JSONObject umzuwandeln um sie später zu speichern.

```
  output.setInt("y",input.values[0].length);
  JSONArray valuesMatrix = new JSONArray();
  for(int i = 0; i < input.values.length; i++){
    JSONArray valuesRow = new JSONArray();
    for(int j = 0; j < input.values[i].length; j++){
      valuesRow.setFloat(j,input.values[i][j]);
    }
    valuesMatrix.setJSONArray(i,valuesRow);
  }
  output.setJSONArray("values",valuesMatrix);
  return output;
}
matrix JSONToMatrix(JSONObject input){


  int sizeX = input.getInt("x");
  int sizeY = input.getInt("y");
  float[][] values = new float[sizeX][sizeY];
  JSONArray valuesMatrix =
input.getJSONArray("values");
  for(int i = 0; i < values.length; i++){
    JSONArray valuesRow =
valuesMatrix.getJSONArray(i);
    for(int j = 0; j < values[i].length; j++){
      values[i][j] = valuesRow.getFloat(j);
    }
  }
  return new matrix(values);
}
```

Funktion, um ein JSONObject in eine Matrix umzuwandeln.

2.2.4 Bilder Importieren

In diesem Abschnitt wird der Teil des Codes beschrieben, mit dem die Trainingsbilder geladen werden. Die Bilder werden in Vektorform gespeichert, um sie direkt in das Netzwerk eingeben zu können.

```
class subject{



  ArrayList <picture> pictures;

  String name;
  String folderName;


  int index;
  int resolution;


  vector ideal;
```

Die „subject“-Klasse beinhaltet Informationen zu der jeweiligen Person.
Diese Liste beinhaltet die Bilder der Person.
Name der Person.
Überordner, in dem der Ordner mit den Bildern der Person ist.
Index der Person.
Auflösung, in der die Bilder verarbeitet werden.
Idealausgabe des

```
                                                                    neuronalen Netzwerks
                                                                    bei der Eingabe eines
                                                                    Bildes von dieser
                                                                    Person.
  subject(String name, String folderName, int index,
int size, int resolution){
    pictures = new ArrayList <picture> ();
    this.name = name;
    this.resolution = resolution;
    ideal = vectorize(index,size);                                  Idealvektor erstellen,
                                                                    bei dem alle Werte null
                                                                    sind ausser dem
                                                                    Indexwert.
    File dir = new                                                  Ordner, in dem die
File(dataPath("database/"+folderName+"/"+name));                    Bilder der Person
                                                                    gespeichert sind.
    File [] files = dir.listFiles();                                Bilder auflisten.
    for(int i = 0; i < files.length; i++){
      addPicture(loadImage(files[i].toString()));                   Jeweiliges Bild
                                                                    hinzufügen.
    }
  }
  void addPicture(PImage image){                                    Funktion, um ein Bild
                                                                    hinzuzufügen.
    pictures.add(new picture(image));
  }
  class picture{                                                    In dieser Klasse
                                                                    werden die
                                                                    Trainingsbilder der
                                                                    Person in Form eines
                                                                    Vektors abgespeichert.
    vector pattern;
    picture(PImage image){
      pattern = imageToVector(image,resolution);                    Das Eingabebild mit
                                                                    der entsprechenden
                                                                    Auflösung in einen
                                                                    Vektor umwandeln.
    }
  }
}
```

2.2.5 Das Netz

In diesem Teil wird das neuronale Netzwerk implementiert. Die „network“-Klasse wird über die „session“-Klasse ausgeführt, welche wichtige Informationen für das neuronale Netzwerk beinhaltet, wie beispielsweise die Namen der Testpersonen. Die „network“-Klasse befindet sich im „network“-Tab.

```
class network{
  ArrayList <subject> subjects;                                     Diese Liste beinhaltet
                                                                    die Informationen der
                                                                    Testpersonen.
```

```
  layer[] layers;
  synapse[] synapses;
```

Die „synapse“-Klasse steht nicht für eine einzelne Verbindung, sondern für alle Verbindungen zwischen zwei Layer.

```
  network(int[] layersSize, ArrayList <subject>
subjects){
```

Die „layersSize“-Liste beinhaltet die Anzahl Neuronen eines Layer.

```
    this.subjects = subjects;
    layers = new layer[layersSize.length];
```

Die Grösse der „LayersSize“-Liste bestimmt die Anzahl Layer.

```
    synapses = new synapse[layers.length-1];
```

Es hat eine Synapsenklasse weniger als Layerklsssen, da die Synapsenklasse die Verbindungen zwischen den Layerklassen beinhaltet.

```
    for(int i = 0; i < layers.length; i++){
      layers[i] = new layer(i,layersSize[i]);
      if(i > 0){
        if(i < layers.length-1){
```

Bei jedem ausser dem letzten Layer ist ein Neuron immer aktiv ist. Zu diesen immer aktiven Neuronen müssen keine Verbindungen führen.

```
          synapses[i-1] = new synapse(i-1,layers[i-
1].pattern.values.length,layers[i].pattern.values.le
ngth-1);
        }else{
          synapses[i-1] = new synapse(i-1,layers[i-
1].pattern.values.length,layers[i].pattern.values.le
ngth);
        }
      }
    }
  }
```

```
  network(ArrayList <subject> subjects, JSONObject
JSONNetwork){
```

Die „network“-Klasse kann auch durch die Eingabe eines JSONObject erstellt werden.

```
    this.subjects = subjects;
    JSONArray JSONLayers =
JSONNetwork.getJSONArray("layers");
    JSONArray JSONSynapses =
JSONNetwork.getJSONArray("synapses");
    layers = new layer[JSONLayers.size()];
```

```
    synapses = new synapse[JSONLayers.size()-1];
    for(int i = 0; i < layers.length; i++){
      layers[i] = new layer(i,JSONLayers.getInt(i));
      if(i > 0){
        synapses[i-1] = new synapse(i-
1,JSONSynapses.getJSONObject(i-1));
      }
    }
  }
  JSONObject saveNetwork(){
```

Diese Funktion gibt alle wichtigen Informationen des Netzwerks im JSON-Format zurück.

```
    JSONObject JSONSave = new JSONObject();
    JSONArray JSONLayers = new JSONArray();
    for(int i = 0; i < layers.length; i++){
      JSONLayers.setInt(i,layers[i].size);
    }
    JSONSave.setJSONArray("layers",JSONLayers);
    JSONArray JSONSynapses = new JSONArray();
    for(int i = 0; i < synapses.length; i++){

JSONSynapses.setJSONObject(i,matrixToJSON(synapses[i
].pattern));
    }
    JSONSave.setJSONArray("synapses",JSONSynapses);
    return JSONSave;
  }
  void update(vector input){
```

Diese Funktion ersetzt den Eingabevektor des Netzwerks durch den Eingabevektor.

```
    for(int i = 0; i < input.values.length; i++){
      layers[0].pattern.values[i] = input.values[i];
    }
  }
  vector run(){
```

Diese Funktion gibt die Werte des Letzen Layer als Resultat zurück nachdem sie die einzelnen Layer des Netzwerks ausgeführt hat.

```
    for(int i = 0; i < layers.length-1; i++){
      layers[i].run();
```

Die „run“-Funktion von jedem Layer wird der Reihe nach ausgeführt.

```
    }
    return layers[layers.length-1].getResults();
  }
```

```
  void train(float increment){
```

Führt die „train“-Funktion, also die Trainingsfunktion einer zufälligen „synapse“-Klasse aus.

```
    int total = 0;
    for(int i = 0; i < synapses.length; i++){
```

Iteriert durch alle „synapse“-Klassen.

```
      synapse o = synapses[i];
      total = total +
o.pattern.values.length*o.pattern.values[0].length;
    }
```

Zählt die totale Anzahl an Verbindungen.

```
    int random = int(random(0,total));
```

Wählt eine zufällige Zahl zwischen Null und der Anzahl Neuronen.

```
    int count = 0;
    for(int i = 0; i < synapses.length; i++){
```

Iteriert durch alle „synapse“-Klassen.

```
      int previous = count;
```

Merkt sich die vorherige „count“-Variable.

```
      synapse o = synapses[i];
      count = count +
o.pattern.values.length*o.pattern.values[0].length;
```

Addiert die Anzahl Neuronen dieser „synapse“-Klasse zur „count“-Variable.

```
      if(random >= previous && random < count){
```

Falls die zufällige Nummer innerhalb der Anzahl Neuronen dieser „synapse“-Klasse ist, wird die Trainingsfunktion dieser „synapse“-Klasse aktiviert.

```
        o.train(increment);
        break;
      }
    }
  }
  void display(float x, float y, float w, float h){
```

Diese Funktion zeichnet das neuronale Netz.

```
    for(int i = 0; i < layers.length; i++){
      layers[i].display(x,y,w,h);
```

Führt die „display“-Funktion jedes Layers aus.

```
    }
  }
  float calculateError(){
```

Diese Funktion berechnet den Fehler des Netzwerks.

```
    float globalError = 0;
    float maxError = 0;
    for(int i = 0; i < subjects.size(); i++){
```

```
      subject thisSubject = subjects.get(i);
      for(int j = 0; j <
thisSubject.pictures.size(); j++){
        subject.picture thisPicture =
thisSubject.pictures.get(j);

        update(thisPicture.pattern);


        vector result = run();

        globalError = globalError +
error(result,thisSubject.ideal);



        maxError = maxError + layers[layers.length-
1].pattern.values.length;



      }
    }
    return globalError/maxError;




  }
  class layer{
    int index;
    vector pattern;



    int size;

    layer(int index, int size){
      this.index = index;
      this.size = size;
      if(index < layers.length-1){



        pattern = zerosVector(size+1);
        pattern.values[pattern.values.length-1] = 1;
      }else{
        pattern = zerosVector(size);
      }
    }
    void run(){
```

Jedes Testbild jeder Person wird nacheinander geladen.
Das Testbild wird dem Netzwerk als Eingabe gegeben.
Das Resultat wird abgerufen.
Dem Fehler wird die Differenz zwischen dem Resultat und dem gewünschten Resultat hinzugefügt.
Der maximal mögliche Fehler wird mit der Anzahl der Neuronen des letzten Layers addiert.

Der Fehler wird durch den maximal möglichen Fehler dividiert um eine Zahl zwischen Eins und Null auszugeben.

Jede Layerklasse enthält einen Vektor, dessen Werte die Werte der Neuronen darstellen.
Anzahl Neuronen in diesem Layer.

Falls der Layer nicht der Letzte ist, erstelle ein zusätzliches Neuron (Dauernd feuerndes Neuron).

Die „run“-Funktion des

```
      if(index < layers.length-1){
        synapses[index].run();
      }
    }
    vector getResults(){


      return pattern;
    }
    void display(float x, float y, float w, float
h){
      pushMatrix();
      float s = 2;
      for(int i = 0; i < pattern.values.length;
i++){
        noStroke();
        fill(0+(1-
pattern.values[i])*256,pattern.values[i]*256+0,patte
rn.values[i]*256+(1-pattern.values[i])*256);
        ellipse(int(i/((h-32)/s))*s+s/2+
index/float(layers.length)*(w)+x,(i%((h-
32)/s))*s+s/2+ y+16,s,s);
      }
      if(index > 0){
        for(int i = 0; i < synapses[index-
1].pattern.values.length; i++){
          if(insideRect(int(i/((h-32)/s))*s+s/2+
index/float(layers.length)*(w)+x,(i%((h-
32)/s))*s+s/2+ y+16,s,s)){println(i);


            for(int j = 0; j < synapses[index-
1].pattern.values[i].length; j++){
              float weight = synapses[index-
1].pattern.values[i][j];
              if(weight == 0){
                //stroke(0,0,0,0);
              }else
              if(weight > 0){
                stroke(0,255,0,255-255/abs(weight));
```

Layers führt die „run“-Funktion der entsprechenden „synapse“-Klasse aus, welche die neuen Werte dieses Layers aus den Werten des voranliegenden Layers berechnet.

Diese Funktion gibt das Resultat, also die aktuellen Werte der Neuronen als Vektor aus.

Diese Funktion zeichnet den Layer an der Entsprechenden Stelle.

Falls der Cursor über dem Neuron liegt, zeige die Verbindungen, welche zu diesem Neuron führen.

```
              }else
              if(weight < 0){
                stroke(255,0,0,255-255/abs(weight));
              }
              strokeWeight(1);
              line(int(i/((h-32)/s))*s+s/2+
index/float(layers.length)*(w)+x,(i%((h-
32)/s))*s+s/2+ y+16,int(j/((h-32)/s))*s+s/2+ (index-
1)/float(layers.length)*(w)+x,(j%((h-32)/s))*s+s/2+
y+16);
            }
          }
        }
      }
      popMatrix();
    }
  }
  class synapse{
```

Die „synapse“-Klasse beinhaltet alle Verbinungen von einem Layer zum Nächsten als Matrix.

```
    int index;
```

Index der jeweiligen „synapse“-Klasse.

```
    matrix pattern;
```

In dieser Matrix werden die Werte der Verbindungen zwischen den Neuronen abgespeichert.

```
    synapse(int index, int thisSize, int nextSize){
      this.index = index;
      pattern = randomMatrix(nextSize,thisSize);
    }
```

Die Werte der Verbindungen sind zu Beginn zufällig.

```
    synapse(int index, JSONObject JSONSynapse){
      this.index = index;
      pattern = JSONToMatrix(JSONSynapse);
    }
```

Eine „synapse“-Klasse kann auch mit einem JSONObject als Eingabe erstellt werden.

```
    void run(){
      float[] nextValues =
layers[index+1].pattern.values;
      float[] thisValues =
layers[index].pattern.values;
```

Die „run“-Funktion der „synapse“-Klasse berechnet die Werte des nächsten Layers.

Dieser Array beinhaltet die Werte des nächsten Layers.

Dieser Array beinhaltet die Werte des Layers, mit denen die Werte des

```
                                                    nächsten Layers
                                                    berechnet werden.
      for(int i = 0; i < pattern.values.length;     Berechnung des Matrix-
i++){                                               Vektor-Produkts.
        float added = 0;
        for(int j = 0; j < pattern.values[i].length;
j++){
          added = added +
pattern.values[i][j]*thisValues[j];
        }
        nextValues[i] = sigmoid(added);             Die Werte des Nächsten
                                                    Layers bestehen nicht
                                                    direkt aus dem Matrix-
                                                    Vektor-Produkt, sondern
                                                    werden vorher durch
                                                    eine Sigmoid-Funktion
                                                    geschickt, welche die
                                                    Werte zwischen Null und
                                                    Eins hält.
      }
    }
    float sigmoid(float input){                     Sigmoid-Funktion,
                                                    welche Ergebnisse
                                                    zwischen Null und Eins
                                                    zurückgibt.
      return 1.0/(1.0+exp(-input));
    }
    void train(float increment){                    Funktion, um das Netz
                                                    zu trainieren.
      float actualError = calculateError();         Ursprünglicher Fehler
                                                    wird ausgelesen.
      int i = int(random(0,pattern.values.length)); Eine zufällige Spalte der
                                                    Verbindungsmatrix wird
                                                    ausgewählt.
      int j =                                       Eine zufällige Zeile wird
int(random(0,pattern.values[i].length));            aus der Spalte
                                                    ausgewählt.
      float actual = pattern.values[i][j];          Das aktuelle Gewicht der
                                                    zufällig ausgewählten
                                                    Verbindung wird
                                                    ausgelesen.
      pattern.values[i][j] = actual -               Das Gewicht wird ein
actualError*abs(actual)*increment;                  bisschen gesenkt.
      float downError = calculateError();           Der Fehler des
                                                    Netzwerks mit dem
                                                    gesenkten Gewicht wird
                                                    gespeichert.
      pattern.values[i][j] = actual +               Das Gewicht wird ein
actualError*abs(actual)*increment;                  bisschen erhöht.
      float upError = calculateError();             Der Fehler des
                                                    Netzwerks mit dem
                                                    erhöhten Gewicht wird
                                                    gespeichert.
      if(upError < actualError && upError <         Falls der Fehler mit dem
```

downError){	erhöhten Gewicht kleiner ist als der aktuelle Fehler und kleiner ist als der Fehler mit dem gesenkten Gewicht, wird das Gewicht erhöht.
pattern.values[i][j] = actual + actualError*abs(actual)*increment;	
}else	
if(downError < actualError && downError < upError){	Falls der Fehler mit dem gesenkten Gewicht kleiner ist als der aktuelle Fehler und kleiner ist als der Fehler mit dem erhöhten Gewicht, wird das Gewicht gesenkt.
pattern.values[i][j] = actual - actualError*abs(actual)*increment;	
}else{	
pattern.values[i][j] = actual;	
}	
}	
}	
float error(vector actual, vector ideal){	Funktion, welche den Fehler zwischen dem aktuellen Vektor und dem idealen Vektor berechnet.
float output = 0;	
for(int i = 0; i < actual.values.length; i++){	
output = output + abs(actual.values[i]-ideal.values[i]);	Die Abstände zwischen allen Werten des Idealvektors und des aktuellen Vektors werden addiert.
}	
return output;	
}	
}	

In der nächsten Tabelle wird die „session“-Klasse erklärt. Sie beinhaltet einige wichtige Informationen für das Netzwerk, wie die Auflösung der Eingabebilder oder die Namen der Ausgabepersonen.

class session{	Die „session“-Klasse.
String[] names;	Namen der Testpersonen.
int resolution;	Auflösung des Eingabebilds.
String name;	Name des Netzwerks.
String folderName;	Ordner, in dem sich die

```
  network myNetwork;

  ArrayList <subject> subjects;
  session(int resolution, int[] hiddenLayersSize,
String folderName, String name){
    File dir = new
File(dataPath("database/"+folderName));
    File[] files = dir.listFiles();



    names = new String[files.length];



    for(int i = 0; i < files.length; i++){
      String file = files[i].getAbsolutePath();
      String[] splitted = file.split("\\\\");
      names[i] = splitted[splitted.length-1];


    }
    this.folderName = folderName;
    this.resolution = resolution;
    this.name = name;
    subjects = new ArrayList <subject> ();
    for(int i = 0; i < names.length; i++){
      subjects.add(new
subject(names[i],folderName,i,names.length,resolutio
n));
    }
    int[] layersSizes = new
int[hiddenLayersSize.length+2];
    layersSizes[0] = int(pow(resolution,2));









    for(int i = 0; i < hiddenLayersSize.length;
i++){
      layersSizes[i+1] = hiddenLayersSize[i];

    }
    layersSizes[layersSizes.length-1] =
names.length;

    myNetwork = new network(layersSizes,subjects);
```

Trainingsbilder befinden.

Das dazugehörige Netzwerk.

Liste der Testpersonen.

Ort der Trainingsbilder.

Liste mit allen Unterordnern (Je ein Unterordner für eine Testperson) erstellen.

Länge der Liste mit Namen anhand der Länge der Liste mit den Unterordnern definieren.

Namen der Testpersonen anhand der Ordnernamen definieren.

Für jeden Namen eine neue „subject“-Klasse erstellen.

Liste mit den Grössen der Layer erstellen.

Der erste Layer besteht aus den schwarzweiss-Werten des Eingabebildes und hat deshalb Breite mal Höhe des Eingabebilds an Neuronen.

Die Grössen der Hidden Layer weitergeben.

Der letzte Layer hat die Anzahl Testpersonen an Neuronen.

Netzwerk mit den entsprechenden Daten erstellen.

```
  }
  session(String name){
    this.name = name;
    JSONObject JSONFile =
loadJSONObject("data/networks/"+name+".json");
    String folderName =
JSONFile.getString("folderName");
    this.folderName = folderName;
    File dir = new
File(dataPath("database/"+folderName));
    File[] files = dir.listFiles();
    names = new String[files.length];
    for(int i = 0; i < files.length; i++){
      String file = files[i].getAbsolutePath();
      String[] splitted = file.split("\\\\");
      names[i] = splitted[splitted.length-1];
    }
    resolution = JSONFile.getInt("resolution");
    subjects = new ArrayList <subject> ();
    for(int i = 0; i < names.length; i++){
      subjects.add(new
subject(names[i],folderName,i,names.length,resolutio
n));
    }
    myNetwork = new
network(subjects,JSONFile.getJSONObject("network"));
  }
  void saveSession(){


    JSONObject JSONSave = new JSONObject();
    JSONSave.setString("folderName",folderName);

    JSONSave.setInt("resolution",resolution);

    JSONObject JSONNetwork =
myNetwork.saveNetwork();
    JSONSave.setJSONObject("network",JSONNetwork);


saveJSONObject(JSONSave,"data/networks/"+name+".json
");
  }
  void display(float x, float y, float w, float h){

    myNetwork.display(x,y,w,h);
  }
  void train(float increment){
```

Öffnet ein gespeichertes Netzwerk.

JSON-Datei laden.

Namen der Testpersonen übernehmen.

Auflösung übernehmen.

Für jede Testperson eine „subject"-Klasse erstellen.

Netzwerk erstellen.

Diese Funktion speichert die Session als JSON-Datei.

JSON-Objekt erstellen.

Ordnername zum JSON-Objekt hinzufügen.

Auflösung zum JSON-Objekt hinzufügen.

Netzwerk als JSON-Objekt speichern.

Netzwerk zum JSON-Objekt hinzufügen.

JSON-Objekt als JSON-Datei speichern.

Ruft die „display"-Funktion des Netzwerks.

Ruft die „train"-Funktion des Netzwerks.

```
    myNetwork.train(increment);
  }
  vector run(PImage input){


    if(input != null){

myNetwork.update(imageToVector(input,resolution));

      return myNetwork.run();


    }else{
      return null;


    }
  }
}
```

Gibt das Resultat des Netzwerks als Vektor zurück.

Aktualisiert den Eingabevektor des Netzwerks.
Berechnet das Resultat und gibt es als Vektor zurück.

Falls das Eingabebild ungültig ist, ist auch der Ausgabevektor ungültig.

2.2.6 User Interface

Da der Code im „ui“-Tab, der zuständig für das User Interface ist, über tausend Zeilen beinhaltet, wird er hier nicht beschrieben und erklärt. Sämtlicher Code im „ui“-Tab ist ausserdem theoretisch nicht notwendig, um ein Netzwerk zu erstellen und auszuführen, das User Interface macht jedoch die Verwendung jedoch einfacher. Für das User Interface selbst wurden keine Libraries verwendet.

2.2.6.1 Netz erstellen

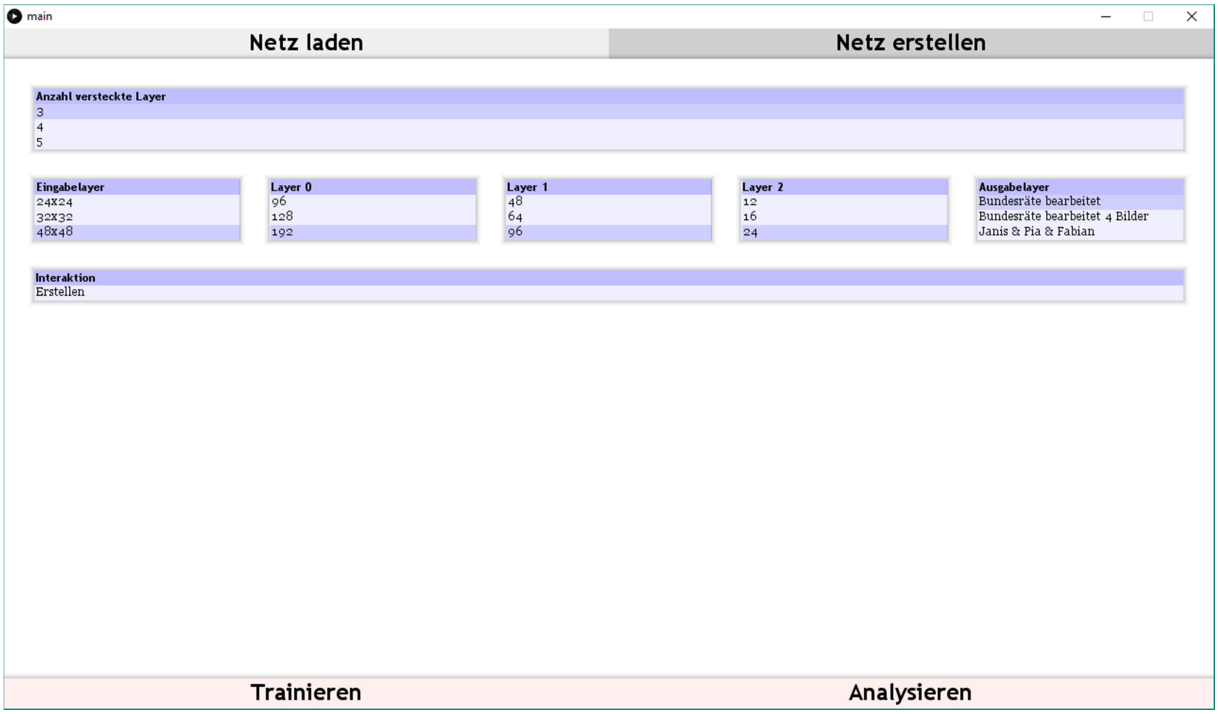


Abbildung 6: Hier können Netze erstellt werden.

Auf dieser Seite können eigene Netze erstellt werden. Zuerst kann die Anzahl versteckter Layer festgelegt werden. Zu beachten ist, dass die Tabellen jeweils mit dem Mausrad heruntergescrollt werden können. Sobald eine Auswahl getroffen wurde, erscheinen weitere Tabellen. In der ersten Tabelle wird die Grösse des Eingabebildes und damit automatisch auch die Anzahl der Eingabeneuronen festgelegt. In der letzten Tabelle können verschiedene Datensätze ausgewählt werden, welche gleichzeitig die Anzahl Ausgabeneuronen festlegt. Wenn zuvor ein oder mehr versteckte Layer ausgewählt wurde, können ausserdem die Grössen der versteckten Layer ausgewählt werden. Auch bei diesen Tabellen kann heruntergescrollt werden für weitere Optionen. Sobald auch diese Tabellen ausgefüllt wurden, erscheint eine letzte Tabelle, um die Erstellung des Netzes abzuschliessen. “. Die erstellten Netze werden im „data“-Ordner im Unterordner „networks“ im JSON-Format (JavaScript Object Notation) abgespeichert. Ganz unten im Fenster sind zwei weitere Buttons zu sehen, die leicht rötlich eingefärbt sind. Sobald ein Netz erstellt wurde, ändern sie die Farbe, werden leicht grünlich und sind somit auswählbar.

2.2.6.2 Netz laden

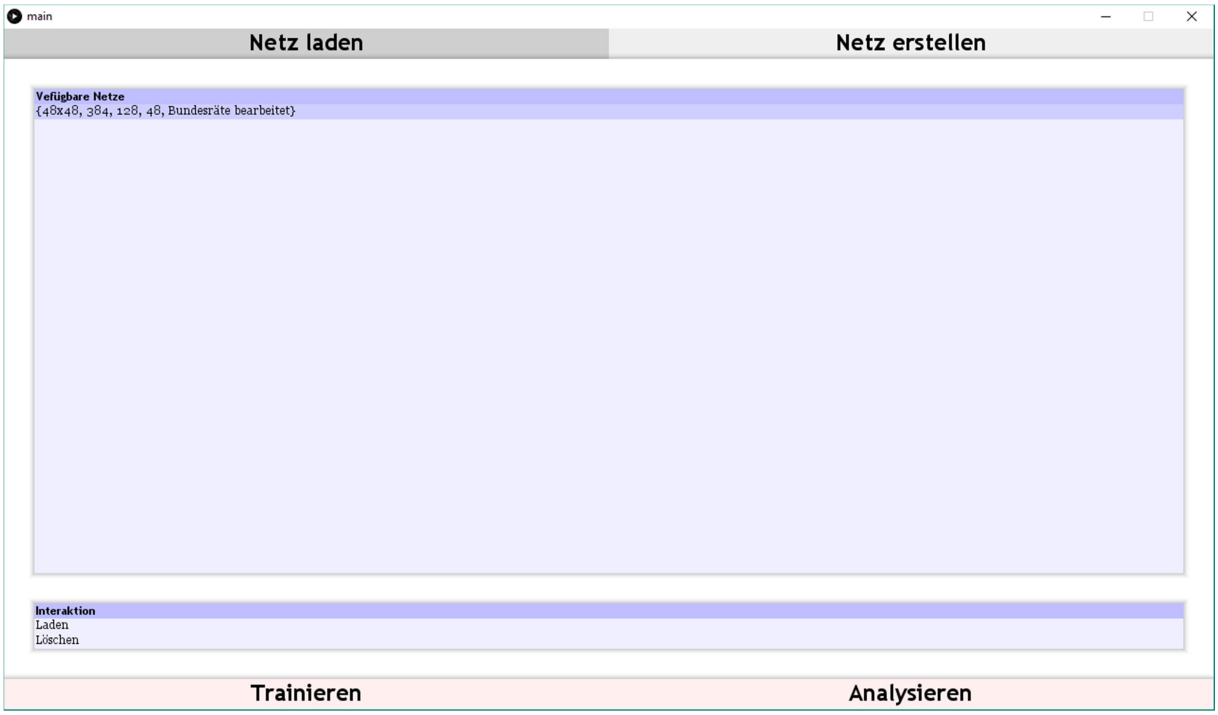


Abbildung 7: Aus der Tabelle in der "Netz laden"-Seite können bereits gespeicherte Netze geladen werden.

Nachdem bei der Auswahl beim Start des Programms auf „Netz laden“ geklickt wurde, erscheint eine Liste mit allen erstellten Netzwerken. Die Netzwerk-Dateien werden beim Erstellen automatisch benannt. Sobald eine Datei ausgewählt wurde, erscheinen unten zwei Möglichkeiten: „Laden“ oder „Löschen“. Wie schon im „Netz erstellen“-Menü kann auch hier, sobald ein Netz geladen wurde, entweder der „Trainieren“ oder der „Analysieren“-Button ausgewählt werden.

2.2.6.3 Trainieren

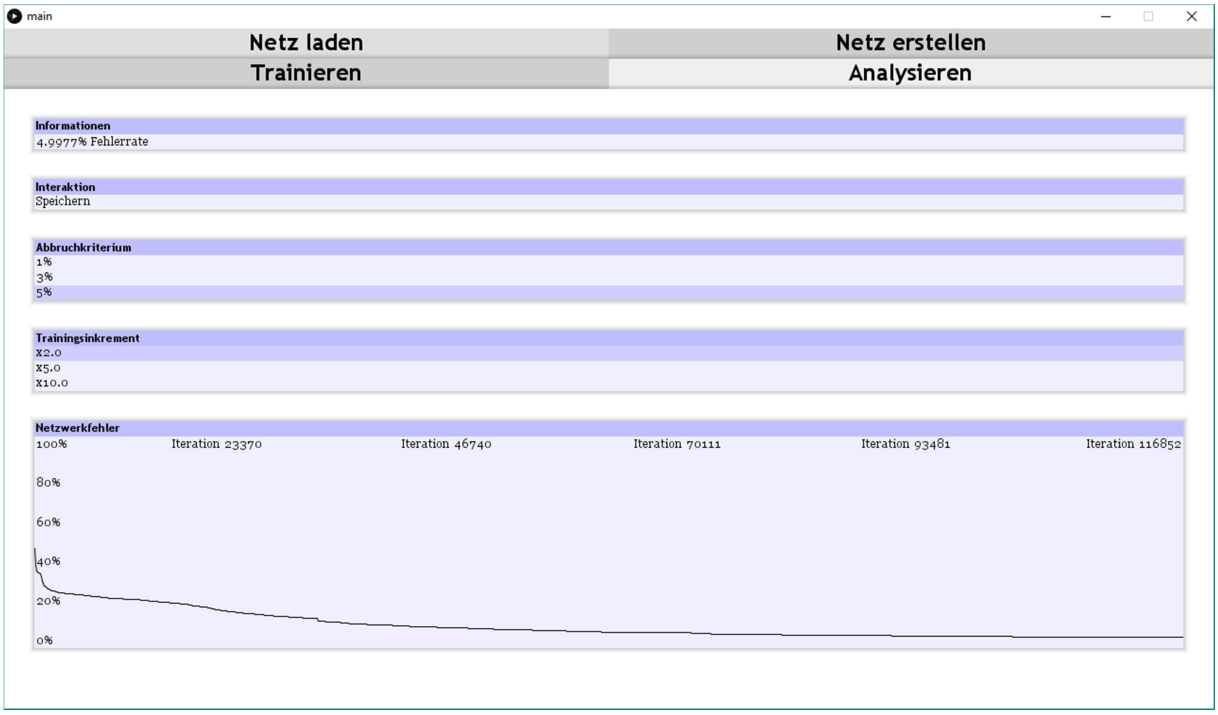


Abbildung 8: Diese Seite ist zuständig für das Training der Netze.

Nachdem ein Netz geladen oder erstellt wurde, kann es trainiert werden. Um das Training zu starten, muss ein Trainingsinkrement gewählt werden. Das Trainingsinkrement bestimmt die Zahl, um die die Gewichtung der zu trainierenden Verbindung erhöht beziehungsweise gesenkt wird. Der Einfluss des Trainingsinkrements ist im folgenden Codeabschnitt sichtbar.

```
      pattern.values[i][j] = actual -
actualError*abs(actual)*increment;
      float downError = calculateError();
```

Das Gewicht wird ein bisschen gesenkt. Der Fehler des Netzwerks mit dem gesenkten Gewicht wird gespeichert.

```
      pattern.values[i][j] = actual +
actualError*abs(actual)*increment;
      float upError = calculateError();
```

Das Gewicht wird ein bisschen erhöht. Der Fehler des Netzwerks mit dem erhöhten Gewicht wird gespeichert.

Die Fehlerrate beim Erkennen der Trainingsbilder wird ganz oben angezeigt. Ausserdem kann diese Fehlerrate auch bei dem Graphen ganz unten abgelesen werden. Das Training wird abgebrochen, sobald der Netzwerkfehler unter das Abbruchkriterium fällt. Wenn kein Abbruchkriterium ausgewählt wurde, wird das Training nicht abgebrochen und wird unendlich weiterlaufen. Vor dem Schliessen der Applikation sollte das Netzwerk immer gespeichert werden mit dem „Speichern“-Button im „Interaktion“-Kasten.

2.2.6.4 Analysieren

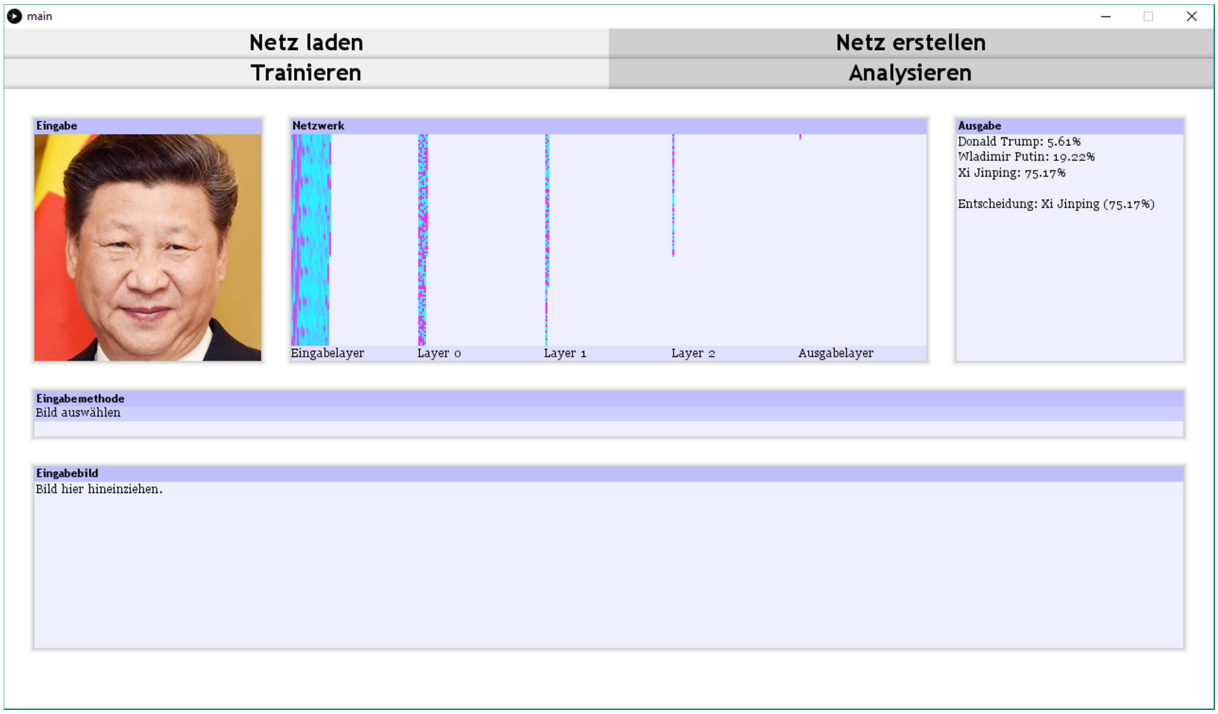


Abbildung 9: Auf der "Analysieren"-Seite können verschiedene Bilder eingefügt und getestet werden.

Im „Analysieren“-Fenster können Bilder eingefügt oder die Webcam benutzt werden, um sie als Eingabe für das Netzwerk zu verwenden. Im „Netzwerk“-Kasten sind die Neuronen der jeweiligen Layer und deren Werte zu sehen, wobei Magenta einen niedrigen und Cyan einen hohen Wert darstellt. Ganz rechts im „Ausgabe“-Kasten kann die Ausgabe des Netzwerks ausgelesen werden. Falls eine Webcam gefunden wurde erscheint im „Eingabemethode“-Kasten zusätzlich zum „Bild auswählen“-Button auch der „Live-Eingabe von Webcam“-Button. Wenn der „Bild Auswählen“-Modus ausgewählt wurde, kann per Drag and Drop ein Bild in den unten erschienenen Kasten hineingezogen werden. Mit dem „Live-Eingabe von Webcam“-Modus wird die Webcam aktiviert und als Eingabe verwendet.

2.2.7 Zusammenfügen der verschiedenen Teile

Dieser Teil des Codes befindet sich im „main“-Tab, welcher den gleichen Namen wie der Ordner haben muss, in dem sich die .pde-Dateien befinden.

```
import processing.video.*;                  Video-Library importieren.

import drop.*;                              Drop-Library importieren.

ui myUi;                                    User Interface erstellen.
session mySession;                          Session erstellen.
SDrop drop;
Capture camera;
void setup(){
  size(1280,720);                           Fenstergrösse definieren.
  drop = new SDrop(this);                   „drop“-Klasse definieren.
  mySession = null;
```

```
  String[] cameras = Capture.list();

  if(cameras.length != 0){
    camera = new Capture(this,cameras[0]);




  }else{
  }

  myUi = new ui();

void draw(){
  if(camera != null){
    if(camera.available() == true){
      camera.read();


    }
  }
  myUi.run();

}
```

Alle verfügbaren Kameras auflisten.

Falls eine Kamera vorhanden ist, diese übernehmen, um sie später als Eingabe zu benutzen.

User Interface definieren.
„draw“-Funktion.

Falls eine Kamera verfügbar ist, werden die Daten ausgelesen.

Das User Interface ausführen.

2.3 Probleme und Schwierigkeiten bei der Implementierung

2.3.1 Der richtige Netzwerktyp

Nachdem die Versuche, Gesichter mit einem Convolutional Neural Network oder mit einem Hopfield-Netz zu erkennen, nicht erfolgreich waren, beschloss ich, ein möglichst simples Netzwerk zu implementieren. Nicht nur war die Implementierung einfacher als bei den vorherigen Versuchen, auch waren die Ergebnisse vielversprechender.

2.3.2 Selbst zu definierende Variablen

Bei der Implementierung von neuronalen Netzwerken entstehen viele Variablen, deren Werte durch keine Regel bestimmt werden können. Wenn diese Werte falsch gesetzt werden, kann das teilweise enorme Auswirkungen auf das Resultat haben. Es ist schwierig, und vor allem zeitaufwendig, zu bestimmen, welche Werte Optimal sind..

Beispiele für Werte, die nicht genau berechenbar sind, sind die Anzahl der Layer sowie die Anzahl der Neuronen in den Layern. Dabei muss beachtet werden, dass eine zu grosse Anzahl an Layer beziehungsweise an Neuronen die Trainingsdauer verlängert, und eine zu kleine Anzahl eine schlechtere Erkennungsrate zur Folge hat. Nach einigen Tests hat sich gezeigt, dass zwei bis drei versteckte Layer eine gute Anzahl ist, wobei der erste Layer ungefähr fünfhundert bis tausend Neuronen hat und die darauf folgenden Layer jeweils etwa vier Mal weniger Neuronen besitzen.

2.3.3 Libraries

Codeteile, die nicht selbst geschrieben wurden, funktionieren oft nicht so, wie man sich das vorstellt. Generell gilt, dass es am besten ist, den ganzen Code selbst zu schreiben. Manchmal

ist das wegen fehlender Kenntnisse oder fehlender Zeit aber nicht möglich, dann kommen Libraries zum Einsatz. Zahlreiche kleinere Komplikationen gab es mit den beiden Libraries; ich musste beispielsweise einen neuen Windows-Account erstellen, da die Video-Library Probleme mit dem „ö"-Charakter im Dateipfad (C:/Users/Fabian Bösiger/) hatte und deshalb nicht funktionierte. Wenn also ein Problem beim Starten des Gesichtserkennungsprogramms auftreirr, stellen Sie sicher, dass kein Charakter im Dateipfad ausserhalb der ersten 128 Zeichen der ASCII-Kodierung ist (Also keine Umlaute oder Ähnliches).

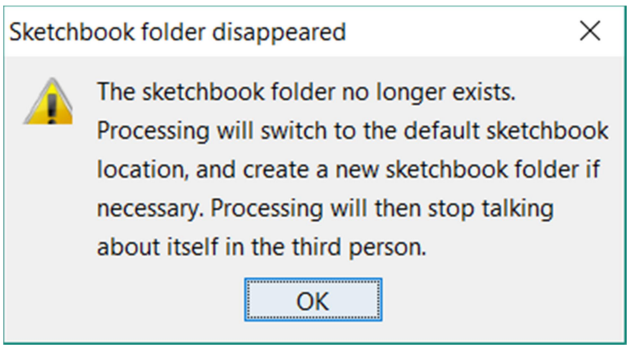


Abbildung 10: Hier hat sich ein Entwickler von Processing einen kleinen Scherz in der Fehlermeldung erlaubt.

2.3.4 Fachliteratur

Da künstliche neuronale Netze relativ neu sind (Die Idee war schon länger vorhanden, jedoch waren die Computer zu leistungsschwach), ist es entsprechend schwierig an entsprechende Literatur zu kommen. Die mit Abstand nützlichste und aktuellste Quelle bietet Wikipedia. Bibliotheken verfügen teilweise nicht einmal allgemeine Fachliteratur im Gebiet Informatik, da diese oft erneuert werden muss und deshalb nicht rentabel ist.

2.3.5 Fehler in der Trainingsfunktion

Beim Erstellen von Netzen mit verschiedenen Anzahlen Layern fiel mir auf, dass Netzwerke mit vielen Layern eine grosse Chance haben, bei einem Fehler von genau 33.33% steckenzubleiben. Das Steckenbleiben an sich ist nicht ungewöhnlich, doch immer genau beim selben Wert war doch ziemlich seltsam. Ich sah mir die Trainingsfunktion noch einmal genauer an und bemerkte, dass die „synapse"-Klasse (Die Klasse, welche die Verbindungsmatrix beinhaltet), in der eine Verbindung ausgewählt wird, rein zufällig ist. Natürlich möchte ich, dass eine zufällige „synapse"-Klasse ausgewählt wird, doch die, welche mehr Verbindungen besitzen, sollten dementsprechend auch mehr ausgewählt werden, damit jede Verbindung eine gleich grosse Chance hat, ausgewählt zu werden.

Die alte, fehlerhafte Funktion für die Auswahl der „synapse"-Klasse kann in der untenstehenden Tabelle eingesehen werden.

`void train(){`	Führt die „train"-Funktion, also die Trainingsfunktion einer zufälligen „synapse"-Klasse aus. Eine zufällige „synapse"-Klasse wird ausgewählt.
`synapses[int(random(0,synapses.length))].train();`	
`}`	

Die neue Funktion ist zwar ein bisschen grösser, doch sie Wählt die „synapse“-Klassen so aus, dass die mit mehr Verbindungen eine entsprechend grössere Chance haben, ausgewählt zu werden. Beim Ausprobieren der neuen Trainingsfunktion konnte ich ausserdem feststellen, dass sie eine viel gleichmässigere Fehlerreduzierung liefert.

```
  void train(float increment){




    int total = 0;
    for(int i = 0; i < synapses.length; i++){

      synapse o = synapses[i];
      total = total +
o.pattern.values.length*o.pattern.values[0].length;
    }
    int random = int(random(0,total));


    int count = 0;
    for(int i = 0; i < synapses.length; i++){

      int previous = count;

      synapse o = synapses[i];
      count = count +
o.pattern.values.length*o.pattern.values[0].length;


      if(random >= previous && random < count){









        o.train(increment);
        break;
      }
    }
  }
```

Führt die „train“-Funktion, also die Trainingsfunktion einer zufälligen „synapse“-Klasse aus.

Iteriert durch alle „synapse“-Klassen.

Zählt die totale Anzahl an Verbindungen.

Wählt eine zufällige Zahl zwischen Null und der Anzahl Neuronen.

Iteriert durch alle „synapse“-Klassen.
Merkt sich die vorherige „count“-Variable.

Addiert die Anzahl Neuronen dieser „synapse“-Klasse zur „count“-Variable.
Falls die zufällige Nummer innerhalb der Anzahl Neuronen dieser „synapse“-Klasse ist, wird die Trainingsfunktion dieser „synapse“-Klasse aktiviert.

3 Resultate

3.1 Bundesrats-Erkennung

3.1.1 Daten und Fakten

Trainingsbilder

Trainingsbilder pro Person	*3*
Hintergrund der Trainingsbilder	*Einfarbig*
Name Trainingsbilderdatensatz	*Bundesräte bearbeitet*

Testbilder

Anzahl Testbilder	*8*
Testbilder pro Person	*1*
Hintergrund der Testbilder	*Unterschiedlich*
Im Trainingsdatensatz vorhanden	*Nein*

Netzwerk

Auflösung Eingabebild	*48x48*
Anzahl versteckte Layer	*3 (384, 128, 48)*
Anzahl Ausgabeneuronen	*8*
Name der Netzwerk-Datei	*{48x48, 384, 128, 48, Bundesräte bearbeitet}*

Resultat

Richtig erkannte Testbilder	*5 von 8*
Falsch erkannte Testbilder	*3 von 8*
Netzwerkfehler zur Zeit des Tests	*5%*

3.1.2 Beschreibung

Ein neuronales Netzwerk wird darauf trainiert, die Bundesräte sowie die Bundeskanzlerin zu erkennen. Pro Testperson wurden nur drei Trainingsbilder verwendet, das heisst das Netzwerk hat nur wenige Trainingsdaten zur Verfügung. Das begründet die vergleichsweise schlechte Erkennungsrate (nur 26% Wahrscheinlichkeit bei der Erkennung von Ueli Maurer). Die Testbilder sind nicht gleichzeitig auch die Trainingsbilder, das heisst das Netzwerk hat die Testbilder noch nie „gesehen“. Eine weitere Schwierigkeit für das Netzwerk sind die unterschiedlichen Hintergründe bei den Testbildern. Die Trainingsbilder stammen von der Website der Schweizerischen Eidgenossenschaft .

3.1.3 Werte

Testperson	Ausgabe des Netzwerks	Korrektheit	Sicherheit	Sicherheit wahre Person
Berset	Berset	WAHR	31%	31%
Burkhalter	Leuthard	FALSCH	64%	21%
Casanova	Casanova	WAHR	60%	60%
Leuthard	Leuthard	WAHR	68%	68%
Maurer	Maurer	WAHR	26%	26%
Schneider-Ammann	Maurer	FALSCH	48%	41%
Sommaruga	Leuthard	FALSCH	33%	32%
Widmer-Schlumpf	Widmer-Schlumpf	WAHR	37%	37%

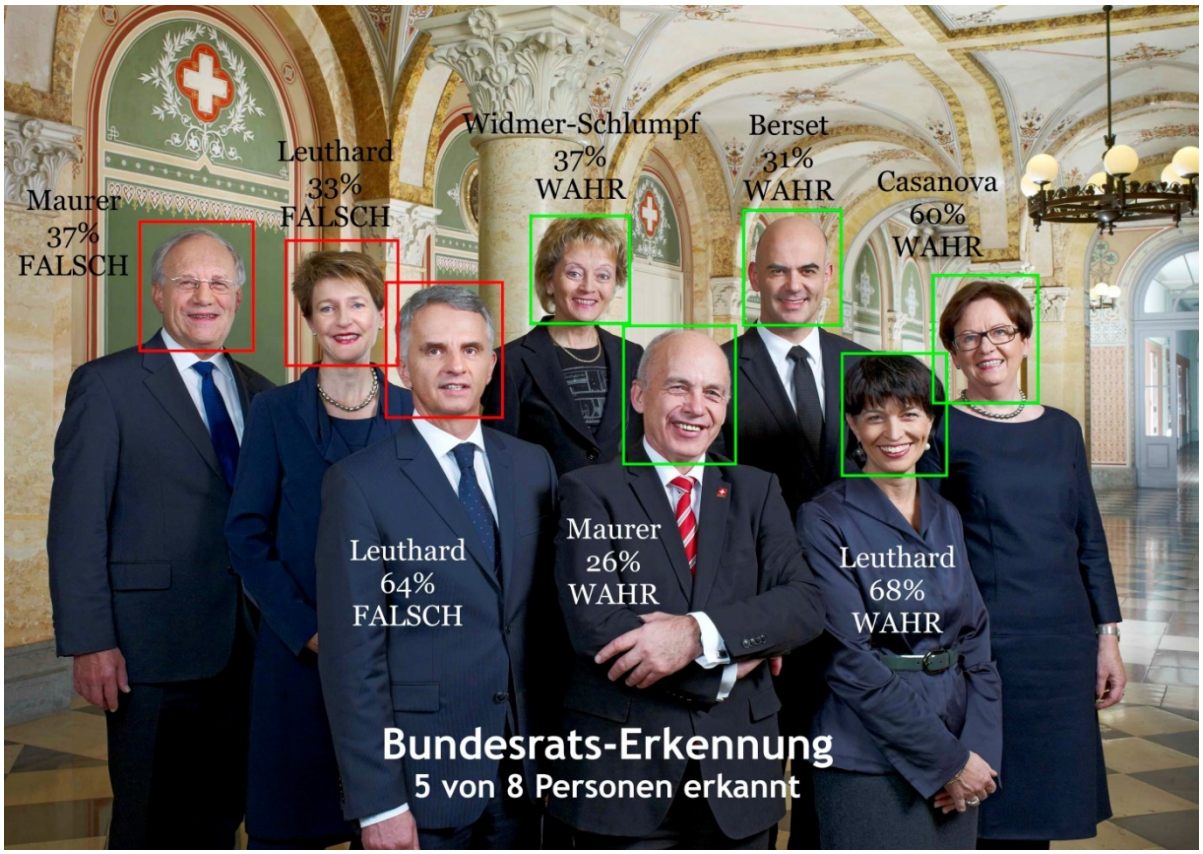


Abbildung 11: Die grün umrahmten Bundesräte wurden vom Netz erkannt, die rot Umrahmten nicht. Grafik: (Schweizerische Eidgenossenschaft, 2017)

3.2 Weltführer-Erkennung

3.2.1 Daten und Fakten

Trainingsbilder	
Trainingsbilder pro Person	*7*
Hintergrund der Trainingsbilder	*Unterschiedlich*
Name Trainingsbilderdatensatz	*Weltführer*
Testbilder	
Anzahl Testbilder	*3*
Testbilder pro Person	*1*
Hintergrund der Testbilder	*Unterschiedlich*
Im Trainingsdatensatz vorhanden	*Nein*
Netzwerk	
Auflösung Eingabebild	*48x48*
Anzahl versteckte Layer	*3 (512, 192, 64)*
Anzahl Ausgabeneuronen	*3*
Name der Netzwerk-Datei	*{48x48, 512, 192, 64, Weltführer}*
Resultat	
Richtig erkannte Testbilder	*3 von 3*
Falsch erkannte Testbilder	*0 von 3*
Netzwerkfehler zur Zeit des Tests	*3%*

3.2.2 Beschreibung

Ein neuronales Netz wird trainiert, zwischen Donald Trump, Wladimir Putin und Xi Jinping zu unterscheiden. Pro Testperson erhält das Netzwerk sieben Trainingsbilder. Die Testbilder sind wie schon im letzten Beispiel nicht im Trainingsdatensatz enthalten. Bei diesem Test haben nicht nur die Testbilder unterschiedliche Hintergründe, sondern auch die Trainingsbilder. Die Resultate dieses Tests sind viel besser ausgefallen als die beim Test 3.1, wahrscheinlich wegen der erhöhten Anzahl Trainingsbilder und der reduzierten Anzahl an Testpersonen.

Abbildung 12: Hier erkennt das Netzwerk erfolgreich den russischen Präsidenten.

3.2.3 Werte

Testperson	Ausgabe des Netzwerks	Korrekt?	Sicherheit	Sicherheit wahre Person
Trump	Trump	WAHR	87%	87%
Jinping	Jinping	WAHR	75%	75%
Putin	Putin	WAHR	77%	77%

3.3 Andere Anwendungsmöglichkeiten (Blatterkennung)

3.3.1 Daten und Fakten

Trainingsbilder

Trainingsbilder pro Gattung	*7*
Hintergrund der Trainingsbilder	*Weiss*
Name Trainingsbilderdatensatz	*Blatterkennung*

Testbilder

Anzahl Testbilder	*2*
Testbilder pro Gattung	*1*
Hintergrund der Testbilder	Weiss

Im Trainingsdatensatz vorhanden	*Nein*
Netzwerk	
Auflösung Eingabebild	*32x32*
Anzahl versteckte Layer	*3 (384, 192, 96)*
Anzahl Ausgabeneuronen	*2*
Name der Netzwerk-Datei	*{32x32, 384, 192, 96, Blatterkennung}*
Resultat	
Richtig erkannte Testbilder	*2 von 2*
Falsch erkannte Testbilder	*0 von 2*
Netzwerkfehler zur Zeit des Tests	*1%*

3.3.2 Beschreibung

Selbstverständlich können künstliche neuronale Netze nicht nur zur Gesichtserkennung verwendet werden. In diesem Test wird ein künstliches neuronales Netz dazu verwendet, zwischen Ahorn- und Buchenblättern zu unterscheiden. Mit dieser Methode könnte beispielsweise eine Handy-App entwickelt werden, mit dem der Benutzer ein Blatt fotografieren kann und anschliessend von der App Informationen zur Gattung eingeblendet bekommt.

Abbildung 13: Erfolgreiche Erkennung eines Ahornblatts.

3.3.3 Werte

Testblatt	Ausgabe des Netzwerks	Korrekt?	Sicherheit	Sicherheit wahres Blatt
Ahorn	Ahorn	WAHR	90%	90%
Buche	Buche	WAHR	74%	74%

3.4 Familien-Erkennung

3.4.1 Daten und Fakten

Trainingsbilder	
Trainingsbilder pro Person	*12*
Hintergrund der Trainingsbilder	*Weiss*
Name Trainingsbilderdatensatz	*Familie*
Testbilder	
Anzahl Testbilder	*12*
Testbilder pro Person	*3*
Hintergrund der Testbilder	Weiss
Im Trainingsdatensatz vorhanden	*Nein*
Netzwerk	
Auflösung Eingabebild	*32x32*
Anzahl versteckte Layer	*3 (768, 256, 64)*
Anzahl Ausgabeneuronen	*4*
Name der Netzwerk-Datei	*Nicht Angegeben*
Resultat	
Richtig erkannte Testbilder	*12 von 12*
Falsch erkannte Testbilder	*0 von 12*
Netzwerkfehler zur Zeit des Tests	*3%*

3.4.2 Beschreibung

In diesem Test wurden pro Testperson je zwölf Trainingsbilder verwendet, bei denen darauf geachtet wurde, dass der Hintergrund weiss ist. Die Testbilder sind nicht im Trainingsdatensatz enthalten, haben jedoch denselben weissen Hintergrund. Bei diesem Test wurden pro Testperson drei Testbilder verwendet. Das Ergebnis ist sehr zufriedenstellend, fast alle Testbilder wurden mit einer Wahrscheinlichkeit von über 80% als richtig erkannt.

3.4.3 Werte

Testperson	Ausgabe des Netzwerks	Korrekt?	Sicherheit	Sicherheit wahre Person
Janis	Janis	WAHR	87%	87%
Janis	Janis	WAHR	74%	74%
Janis	Janis	WAHR	63%	63%
Pia	Pia	WAHR	80%	80%
Pia	Pia	WAHR	87%	87%
Pia	Pia	WAHR	92%	92%
Mitrasch	Mitrasch	WAHR	88%	88%
Mitrasch	Mitrasch	WAHR	95%	95%
Mitrasch	Mitrasch	WAHR	93%	93%
Fabian	Fabian	WAHR	89%	89%
Fabian	Fabian	WAHR	90%	90%
Fabian	Fabian	WAHR	85%	85%

4 Diskussion

4.1 Overfitting

Beim Trainieren von neuronalen Netzen besteht stets die Gefahr von Overfitting, also eine Überanpassung an die Trainingsbilder. Dabei sinkt zwar der Fehler bei der Erkennung von Trainingsbildern, jedoch steigt der Fehler bei der Erkennung von Testbildern.[9]

Das Netzwerk erkennt in allen Tests nicht nur Bilder, welche trainiert wurden, sondern auch Bilder die es noch nie gesehen hat. Diese Tatsache bestätigt, dass das neuronale Netz nicht nur auswendig lernt, sondern Zusammenhänge in den Eingabewerten erkennt.

4.2 Nachvollziehbarkeit der Verbindungsgewichte

Da neuronale Netze sich selbst trainieren, ist es fast unmöglich, zu verfolgen, auf was das Netzwerk bei der Eingabe Achtet, um die Resultate zu erhalten. Um genau zu sehen, wie das künstliche neuronale Netz „denkt“, müsste man die Gewichtung jeder Verbindung nachvollziehen und verstehen, warum eine spezifische Verbindung ihren Wert hat. Doch bei weit über zehntausend Verbindungen ist das beinahe unmöglich und würde einen enormen Zeitaufwand erfordern.

4.3 Minimale Anzahl Trainingsbilder

Aus den verschiedenen Versuchen kann geschlossen werden, dass nur drei Trainingsbilder pro Testperson, wie beim Versuch 3.1 zu wenig ist, um zuverlässig Gesichter zu erkennen. Viel bessere Resultate ergeben sich mit einer grösseren Anzahl Trainingsbildern pro Testperson, wie beim Versuch 3.4, bei dem ein dutzend Trainingsbilder verwendet wurden und dementsprechend bessere Resultate erzielt wurden.

4.4 Hintergrund

Beim Versuch 3.4 liegen die Sicherheiten der Ausgaben des Netzwerks fast immer zwischen 80-90%. Ein Grund für diese hohen Sicherheiten ist derselbe Hintergrund bei Trainings- und Testbild. Gleiche Hintergründe sind sicher von Vorteil, aber weniger wichtig als grosse Trainingsdatensätze, denn bei grossen Trainingsdatensätzen mit unterschiedlichen Hintergründen ist die Chance einiges grösser, dass das neuronale Netz merkt, dass der Hintergrund keine Rolle spielt. Wenn jedoch nur kleine Trainingsdatensätze vorhanden sind, sind gleiche Hintergründe vorteilhaft.

4.5 Selber zu definierende Variablen durch das neuronale Netz definieren

Wie bereits im Kapitel 2.3.2 beschrieben, können selbst zu definierende Variablen einen reibungsfreien Trainingsablauf verhindern. Um diese Variablen zu vermeiden, könnte ein Netzwerk implementiert werden, welches selbst die Werte besagter Variablen definieren kann. Somit hätte das künstliche neuronale Netzwerk die Fähigkeit, sich selbst zu verändern. Damit ein Netzwerk aber die Fähigkeit hat, zu bestimmen, welche Anzahl Layer am besten für die gegebenen Eingabebilder sind, müsste es mit einem Gedächtnis ausgestattet sein, was

[9] (Wikipedia, 2016)

für das Problem der Gesichtserkennung aber eher eine übertriebene Massnahme wäre. Um selbst zu definierende Variablen zu vermeiden sind evolutionäre Algorithmen gut geeignet. Ein evolutionärer Algorithmus würde aus zufällig generierten Netzen mithilfe einer Fitnessfunktion die besten auslesen, diese kombinieren, mutieren und aus der neuen Generation von Netzen wiederum die besten auslesen. Doch neuronale Netze durch einen evolutionären Algorithmus zu verbessern wäre für den Zweck der Gesichtserkennung zu ineffizient.

4.6 Personen mit höherem Wiedererkennungswert

In den verschiedenen Tests fiel auf, dass die Netzwerke teilweise Tendenzen zeigten, eine Person zu bevorzugen. Dieses Verhalten könnte durch unzureichendes Training Entstanden sein. Eine andere Theorie ist, dass das Netz eher Personen mit schwarzen Haaren bevorzugt, denn hohe Schwarz- oder Weisswerte im Eingabefeld haben eine hohe Aktivität der Eingabeneuronen zur Folge. Diese hohen Eingabewerte könnten wiederum zur Folge haben, dass das Netzwerk diese Personen bevorzugt, also dass die dazugehörigen Neuronen eher eine grössere Aktivität aufweisen. Diese Theorie konnte ich jedoch nicht bestätigen und es ist möglich, diese Bevorzugung von Personen durch eine längere Trainingsphase zu reduzieren.

4.7 Zukunft von künstlichen neuronalen Netzen

Die ständige Weiterentwicklung von Quantencomputer hat auch einen grossen Einfluss im Bereich Machine Learning. Quanten-Prozessoren (QPUs) nutzen das Verhalten von Subatomaren Teilchen, anstelle von üblichen Transistoren. Die Möglichkeit, nicht auf entweder Null oder Eins beschränkt zu sein bringt enorme Vorteile bei Optimierungsproblemen mit sich. Die Dauer der Trainingsphasen bei neuronalen Netzen könnten mit dieser Technologie um ein Vielfaches reduziert werden und die Anzahl Verbindungen somit erhöht werden.[10]

[10] (D-Wave Systems, 2017)

5 Quellen

5.1 Quellen der Portraits für den Test 3.2

Sämtliche Bilder wurden im August 2017 von Wikimedia Commons (GNU-Lizenz für freie Dokumentation) heruntergeladen. Änderungen wurden vorgenommen.

Donald Trump

[1] https://upload.wikimedia.org/wikipedia/commons/f/f5/Donald_Trump_August_2015.jpg
[2] https://upload.wikimedia.org/wikipedia/commons/0/0b/Donald_Trump_by_Gage_Skidmore_10.jpg
[3] https://upload.wikimedia.org/wikipedia/commons/a/a6/Donald_Trump_%285440393641%29_%28cropped%29.jpg
[4] https://upload.wikimedia.org/wikipedia/commons/0/0b/President_Trump_2.jpg
[5] https://upload.wikimedia.org/wikipedia/commons/d/dd/Donald_Trump_2013_cropped_more.jpg
[6] https://upload.wikimedia.org/wikipedia/commons/a/a7/Donald_Trump_March_2015.jpg
[7] https://upload.wikimedia.org/wikipedia/commons/thumb/0/0e/Donald_Trump_Pentagon_2017.jpg/435px-Donald_Trump_Pentagon_2017.jpg
[8] https://upload.wikimedia.org/wikipedia/commons/9/9c/Trump_first_weekly_address.jpg

Xi Jinping

[1] https://upload.wikimedia.org/wikipedia/commons/c/cc/Xi_Jinping_March_2017.jpg
[2] https://upload.wikimedia.org/wikipedia/commons/thumb/2/28/Xi_Jinping_Sept._19%2C_2012.jpg/383px-Xi_Jinping_Sept._19%2C_2012.jpg
[3] https://upload.wikimedia.org/wikipedia/commons/e/ed/Xi_Jinping_2016.jpg
[4] https://upload.wikimedia.org/wikipedia/commons/3/3f/Xi_Jinping_October_2015.jpg
[5] https://upload.wikimedia.org/wikipedia/commons/2/27/Xi_Jinping_Sanya2013.jpg
[6] https://upload.wikimedia.org/wikipedia/commons/4/40/Xi_Jinping_Mexico2013.jpg
[7] https://upload.wikimedia.org/wikipedia/commons/thumb/7/72/Xi_Jinping_October_2013_%28cropped%29.jpg/418px-Xi_Jinping_October_2013_%28cropped%29.jpg
[8] https://upload.wikimedia.org/wikipedia/commons/5/5f/Xi_Jinping_in_British_Parliament.jpg
[9] https://upload.wikimedia.org/wikipedia/commons/f/f5/Donald_Trump_August_2015.jpg

Wladimir Putin

[1] https://upload.wikimedia.org/wikipedia/commons/d/d1/Vladimir_Putin_12020.jpg
[2] https://upload.wikimedia.org/wikipedia/commons/1/1d/Vladimir_Putin_12023_%28cropped%29.jpg
[3] https://upload.wikimedia.org/wikipedia/commons/c/cf/Vladimir_Putin-6.jpg
[4] https://upload.wikimedia.org/wikipedia/commons/thumb/a/a4/Putin_%28cropped%29.jpg/220px-Putin_%28cropped%29.jpg
[5] https://upload.wikimedia.org/wikipedia/commons/a/a9/Vladimir_Putin_official_portrait.jpg
[6] https://upload.wikimedia.org/wikipedia/commons/4/45/Vladimir_Putin_-_2006.jpg
[7] https://upload.wikimedia.org/wikipedia/commons/e/ee/Vladimir_Putin_12022.jpg
[8] https://upload.wikimedia.org/wikipedia/commons/d/d7/2017-01-11_Vladimir_Putin_at_a_Meeting_on_the_295th_anniversary_of_the_Russian_Prosecution_Service%2C_04.jpg

5.2 Literaturverzeichnis

Davis, C. (30. April 2010). *Flickr*. Von https://www.flickr.com/photos/53416677@N08/4972916707 abgerufen

D-Wave Systems. (10. 09 2017). *D-Wave*. Von https://www.dwavesys.com/quantum-computing/applications abgerufen

Processing Foundation. (2017). *Download - Processing*. Von https://processing.org/download/ abgerufen

Processing Foundation. (2017). *Reference - Processing*. Von https://processing.org/reference/ abgerufen

Processing Foundation. (2017). *Video - Processing*. Von https://processing.org/tutorials/video/ abgerufen

Schlegel, A. (2017). *Drop*. Von http://www.sojamo.de/libraries/drop/ abgerufen

Schweizerische Eidgenossenschaft. (3. Januar 2017). *Der Bundesrat*. Von https://www.admin.ch/gov/de/start/bundesrat/bilder-und-reden-des-

bundesrats/offizielle-bundesratsbilder/printversionen-und-einzelportraits.html abgerufen

Wikipedia. (11. September 2014). *Gesichtserkennung (Fotografie) - Wikipedia*. Von https://de.wikipedia.org/wiki/Gesichtserkennung_(Fotografie) abgerufen

Wikipedia. (5. März 2015). *Viola-Jones-Methode - Wikipedia*. Von https://de.wikipedia.org/wiki/Viola-Jones-Methode abgerufen

Wikipedia. (21. Dezember 2016). *Überanpassung - Wikipedia*. Von https://de.wikipedia.org/wiki/%C3%9Cberanpassung abgerufen

Wikipedia. (23. April 2017). *evolutionärer Algorithmus - Wikipedia*. Von https://de.wikipedia.org/wiki/Evolution%C3%A4rer_Algorithmus abgerufen

Wikipedia. (9. Juni 2017). *Gesichtserkennung - Wikipedia*. Von https://de.wikipedia.org/wiki/Gesichtserkennung abgerufen

Wikipedia. (28. Juni 2017). *JavaScript Object Notation - Wikipedia*. Von https://de.wikipedia.org/wiki/JavaScript_Object_Notation abgerufen

Wikipedia. (18. Juni 2017). *Künstliches neuronales Netz - Wikipedia*. Von https://de.wikipedia.org/wiki/K%C3%BCnstliches_neuronales_Netz abgerufen

6 Anhang

6.1 Beigelegte Programme

Sämtliche Programme sind auf der beigelegten CD zu finden.

6.1.1 Neural Network Toolkit

Mit dem „Neural Network Toolkit“ können künstliche neuronale Netze erstellt, trainiert und getestet werden. Auch frühere Versionen sind vorhanden, die jedoch nicht alle funktionsfähig sind. Die Anleitung für das Programm finden sie im Kapitel 2.2.6 „User Interface“.

6.1.2 Neural Network Visualizer

Dieses Programm stellt neuronale Netzwerke dreidimensional dar. Von diesem Programm kommt auch die Grafik auf dem Titelblatt.

6.1.3 Image Croper

Dieses Programm wurde verwendet, um die Bilder zuzuschneiden.

6.1.4 Image Croper Cam

Eine kleine Abwandlung vom Programm 6.1.3 „Image Croper“. Bei dieser Abwandlung muss kein Bild ausgewählt werden, sondern die Bildeingabe kommt direkt von der Webcam.

6.1.5 Format My Code

Ein kleines Tool, welches Programmcode schöner gestaltet. Es entfernt leere Codelinien und verwandelt Tabulator-Abstände in Doppelleerschlag-Abstände. Dieses Programm wurde benutzt, um den Code in diese Arbeit sauber einzufügen.

6.2 Bestätigung

Ich erkläre hiermit, dass ich die vorliegende Maturaarbeit selbständig und ohne unerlaubte fremde Hilfe erstellt habe und dass alle Quellen, Hilfsmittel und Internetseiten wahrheitsgetreu verwendet wurden und belegt sind.

Unterschrift, Datum

.....