Universität Bremen

Behaviour-based detection of Visual Interaction Obstacles with 1D and 2D Convolutional Neural Networks

Bachelor Thesis at the Cognitive Systems Lab
Prof. Dr.-Ing. Tanja Schultz
Faculty 3: Mathematics and Computer Science
University of Bremen

by

Anthony Mendil

Supervisor:

Mazen Salouz

Examiner:

Felix Putze
Unknown

Day of registration:	1. Mustermonat 1851
Day of submission:	3. Mustermonat 1963

I hereby declare that I am writing this work independently and have not used any sources or resources other than those specified.

Bremen, the 3. Mustermonat 1963

Zusammenfassung

... deutsch ...

Der deutsche Abstract wird in jedem Fall benötigt.

Abstract

... english ...

Der englische Abstract wird nur benötigt, wenn die Arbeit in englischer Sprache verfasst wird.

Contents

List of Figures

List of Tables

1. Introduction

- papers von denen lesen. da sind richtig gute ineleitunegn an denen man sich orierntieren kann.
-

Neural networks greatly benefit from lots of data. As there are only 40 **TODO: richtige zahl herausfinden** overall games from participantzs available, more data can potentially improve the classification results. The cognitive systems lab already implemented a system that takes the original game logs and simulates user behaviour in order to create new logs. As a result it is possible to create any number of games out of a single original one, from which some may be better than others. However, in order to simulate glare effect games multiple addtional steps and changes are neccesary.

- vielleict begrüdung: referenz zu denen. weil es zietgt dass man unter verwnedung simulierter spiele eine deutliche verberseerung der ergebnisse auf den echten spelen sehen kann...

- fragen an mazen:
- demographic
- original mapping of cards to colours
- nachfragen ob static mapping so ist wie ich denke
- sind whickser nur standardabweichung oder was ist das?
- ist es ok dass ich meine eigene tabelle gemacht habe für delta e interpretation? im interent stand es gibt keine allgemeine..hab auch keine von cielab gefunden
- ich brauche mll knecht

- Einleitung: Was es für verschiedene Obstacles gibt, was schon gemacht wurde, was ich mache, wieso keine eeg daten obwohl sie da sind (sind interaktion obstacle zu erkennen und nutzung zu verbessern ist nicht da wenn man eeg maske tragen muss)

- memory game kurz erklären

- libraries und so die ich benutz habe? pandas keras..

- **simulation**:
- simulatin damit man mehr daten hat
- generell simulator erklären und sinn von similarity matrix
- similaity matrix erstelung und gedanken (plus anpassungen an memory game, anpassungen am simulator damit similarity matrix benutzt werden kann)
- es gab invalid logs und hab code geschrieben der das überprüft damit man die rausnehmen kann. vor simulation (validLogsCollecot)
- similarity matrix mapper, waren vorher nicht gmapped und mapper macht das und ersetz alte matrix durch die neue für glare effect
- erklären wie ich korrektheit der matrix überprüft habe
- erklären wie ich korrektheit der farbextraktion und unterschied berechnung überprüft habe (online rechner und anfangs matrix für no obst erstellt und mit alten verglichen aber gibg nicht weil struktur nicht zu erkennen war bei alter matrx von vorherigen arbeieten)
- initially simulationergebnisse mit plots für qualität (plus erklärung)
- sagen was noch nicht optimal und, was am simulator dafür geändert wurde (2 sachen: random decay ab 10 zügen plus eine andere sache) mit begründung und wie die ergebnisse am ende aussahen (zwischenschritte eher niht glaube ich. nur kurz erwähnen)
- darüber reden wie es im realfall ist: wir benutzen nicht alle simualtionen sondern nur die besten, plots zeigen mi nur den besten (vor und nach änderung vielleicht, oder nur nach änderung (aber dann sieht man nicht das änderung sinnvoll war))
- paried t-test kram um signifikante unterschiede /keine unterschiede zu zeigen für verschiedene bedingenen (siehe mazens nachricht)

- **modelle**:
- feature engenierring: also was die komponenten sind und so, wie sie berechnet wurden
- für 1d cnn wuren die so übernommen
- erklärung komponenten von 1d cnn
- (villeicht auch mal nicht mit allen featires probieren, aber da hatte ich problme mit dem cnn. man müsste glaube ich die struktur ändern)
- struktur 1 d cnn mit begründung
- 2d cnn komponenten erklärunen (snythetic image erklären und zeigen wie berechnet wurde und wie es aussieht)
- idee von 2d cnn für diesen fall
- 2d cnn struktur erklären und begründen

dies	ist	eine	Tabelle
mit		zwei	Zeilen

Table 1.1: Tabelle mit einer langen Unterschrift

- **Training und analyse**:
- vielleiht gucken welche falsch erkannt werden und woran es liegt, also wenn die zum beispiel echt schlect oder gut sind obwohl es nicht so sein sollte (rausnhemen und gucken wie ergebnisse sind, vielleicht nur bei bestem modell)
- in gleichen tabellen ergebnise vor änderung am simulator und nach änderung betrachten und vergleichen
- training auf welchem rechner/n,was von: cpu oder gpu oder beides, hardware kurz erwähnen, vpn (fernzugriff)
- train test splt, leave one out k fold (+ begründung mit deep learing ..reliable results etc, randomness)
- kurz erklären wieso weniger züge besser sind bei dieser hci erkennung
- (vielleicht kram zu adaptive learning rate ändern und gucken wie es so ist, ansonsten begründen wieso ich das nicht brauche)
- 20 steps und 40 steps für beide trainieren mit sd0x bis sd10x plus sd20x
- (entweder so dass 1d cnn mit 20 steps auch konvergence erreicht oder danach nochmal anpassung von 20 steps 1d cnn damit es kovergiert)
- darüber schreiben dass letzen n züge nicht mehr so gut simuliert sind (zeigen) und deshalb 40 steps nicht so viel besser ist
- mit maximalen guten steps (glaube 16 züge oder so) trainieren und vergleichen
- statistical tests um signifikante unterschiede /keine unterschiede zu zeigen für verschiedene modelle etc (siehe mazens nachricht)

- in anhang alle skripte aufzählen wie markus

1.1 Anmerkungen

Zitationen [Rab89] sind keine Wörter sondern Referenzen und stellen somit keinen Teil des Satzes dar. In anderen Worten: Der Satz muss auch noch funktionieren, wenn die Zitation einfach entfernt wird.

Index-Einträge

Wir haben Tabellen 1.1 und Bilder 1.1.

Figure 1.1: Bildunterschrift

2. Memory Game

The game used for the data collection is a matching pairs game written in Kotlin for Android devices. The original version was developed by Rafael Miranda **TODO: ref** in the context of a bachelor thesis. Originally, the game consisted of cards picturing animals and was only used to model general poor eyesight. Later is was expanded by adding a coloured card set to also model colour blindness. In the following the current functionalities of the game will be explained, only covering those related to the coloured cards. One game consists of 14 cards and each round consists of two cards being turned face up. If the cards match, they will be removed from the field. Otherwise they are turned face down again. The player wins once all 7 pairs have been discovered.

The game differentiates between two groups of obstacles: memory obstacles and visual obstacles. Additionally there is a classic mode providing the possibility to play without any obstacles which uses the red-green card set, shown in fig **TODO: ref**. The memory obstacle mode consists of the same card set and an additional side task: Every time a card is flipped a random number between 0 ans 10 is called out and the player must add all these numbers. After the final pair was found the sum of all numbers is requested. The game contains modes for two types of visual obstacles: Colour blindness and the glare effect. Colour blindness, more precisely a red-green weakness, is simulated by replacing the red-green card set with a brown shifted card set. The glare effect describes a sczenatio in which a lightsource shines onto the display, reflects from it and makes it more difficult to differnentiate the colours of the cards. This effect can be seen in figure **TODO: ref**.

To help in case of interaction obstacles the game provides multiple ways of assistance. Although they are not relevent for this thesis as only data is used where probants had no assistance, they will be explained briefly. There are two types of assistance: memory and visual assistance. Memory assistance is provided by flipping the cards from the previous turn once two non matching cards are flipped. After a short period of time they are turned face down again. If the game is played with visual assistance, each pair gets assigned a letter that is called out once a card is flipped. This introduces a new way of orientation that does not rely on the sense of sight, but instead on the sense of hearing. There is a game mode for each combination of obstacle (no obstacle, memory obstacle, colour blindness, glare effect) and assistance (no assistance, memory assistance, visual assistance). There were also other assistances implemented such as increasing the size of cards if they are revelaed, but they were only used for tests. The glare effect no assistance and the no obstacle no assistance modes are the only ones relevant in this thesis. Figure **TODO: ref** and **TODO: ref** show screenshots in the two modes.

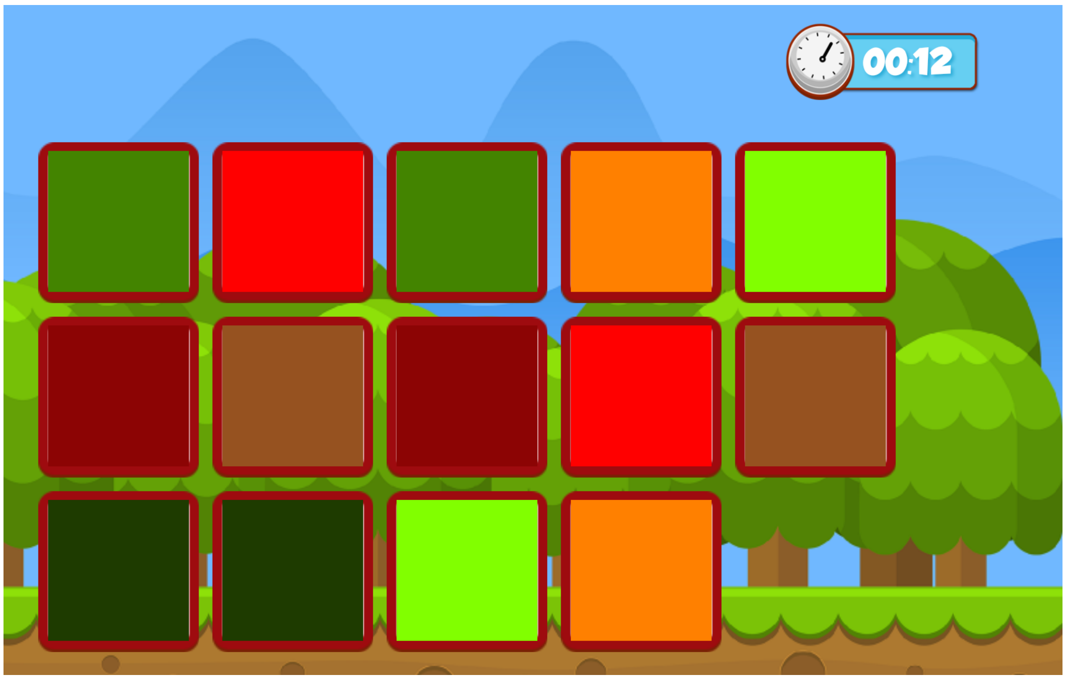


Figure 2.1: Screenshot of the game without obstacles. All cards are turned face up.

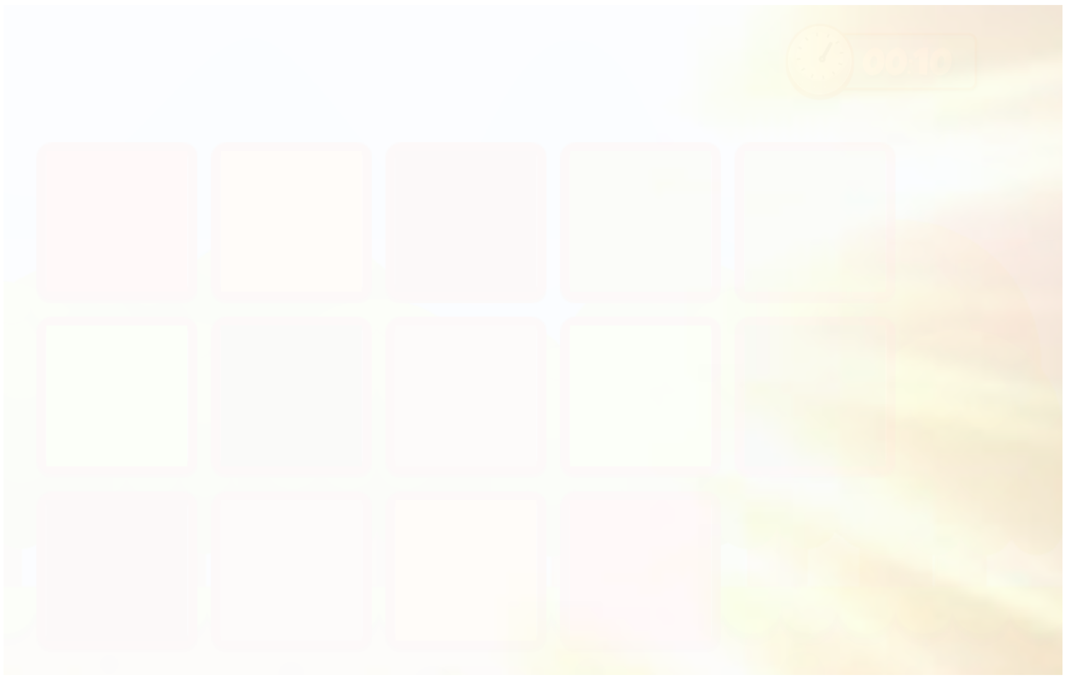


Figure 2.2: Screenshot of the game field with simulated sunlight. Sunbeams are especially visible on the right hand side. All cards are turned face up.

3. Collected Data

While the probands played the memory game, behavioral data and brain activity data was collected. Relevant for this work is only the behavioural data recorded in glare effect and no obstacle games. The data was collected from 22 probands. **TODO: grobe demographic herausfinden, mazen fragen**. Each participant except one recorded 2 no obstacle and 1 glare effect game. The one exception did not record any glare effect games. As a result there are 44 no obstacle and 21 glare effect games.

3.1 Behavioural data

In order to record bavioural data, each card was initally given a fixed card code and is was recorded which pairs of cards were turned face up in each round. The card code consists of two numbers separated by a dot. The first number specifies the colour of the card (between 1 and 7 as there are 7 colours) and second number specifies if it is the first or second card with that color in the order of the cards on the field **TODO: nachfragen ob das so ist, aber müsste weil am anfang zum beispiel 5.2**. The bevioral data was saved in logs, consisting of the card codes of each round followed by the unix timestamps of when the cards were flipped. Data of 20 rounds and therefore at maximum 40 flipped cards was recorded. If the game was completed in less than 20 turns, the remaining card code entries were filled with 0.0 and the remaining timestamp entries were filled with 0. An example of the sequence of card codes recorded during a game can be seen below.

```
5.2,2.2,4.2,4.1,6.1,7.2,6.2,6.1,1.2,1.1,7.1,2.2,2.1,7.1,5.2,5.1,3.1,
2.1,7.1,3.1,7.2,7.1,3.2,2.2,2.1,2.2,3.1,3.2,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0
```

TODO: inital mapping herausfinden With the assignments of for instance of 5 for red and 2 for green, the first two entries in the log file mean that first the second red card and then the first green card was turned face up. This mapping is static as colours have the same numbers across all recorded games. However, the simulator requires dynamic mapping of the card codes. Additionally dynamically mapped similarity values for the card codes are needed which are missing in the original logs.

By dynamic mapping of the card codes is meant, that the colours are not assigned to fixed numbers but that the numbers are assigned in the reveal order specific to each game. This becomes clear, by looking the card code sequence from above, but dynamically mapped.

```
1.1,2.1,3.1,3.2,4.1,5.1,4.2,4.1,6.1,6.2,5.2,2.1,2.2,5.2,1.1,1.2,7.1,
2.2,5.2,7.1,5.1,5.2,7.2,2.1,2.2,2.1,7.1,7.2,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0
```

TODO: sobald ich originales mapping weiß, weiß ich auch welche farben die karten hatten und kann die folgenden farben im text ersetzen In this dynamic mapping of the card codes, each entry likewise consists of 2 numbers that are seperated by a dot. However, the numbers are differently interpreted. The first number specifies what number of colour it is to be revealed and the second number specifies if it was the first or the second card in that colour. In the example above the colour green is the first to be revealed and it was the first green crad. Therefore the first entry is 1.1. Then a red card is turned face up, meaning the second entry is 2.1 since is was the first red card and so on. Once the other red card is dicovered, the entry will be 2.2. This pattern continues throughout the whole sequence. This means that the mapping is specific to each game and depends on the reveal order of the cards. The differnece bewteen static and dynamic similarity assignments is explained in section **TODO: ref zu similarity matrix bei simulator und da erklären was die unterschiede sind und was der simulator benutzt? genauso köönte man dann aber auch die card codes mapping da erklären?**

Logs with remapped card codes and added similarity values were already provided, but there were two issues regarding the glare effect logs. The first one being that the similarity matrix for the glare effect game was just a placehoulder and did not correclty portray the color differneces under the influence of sunlight. To successfully simulate glare effect games the simulator requires such an similarity matrix. Therefore a new one is created in section ref. The second issue was that although the card codes were dynamically mapped, the similarity values were not. This meant that the assignment of simialrity values to numbers for the cards compared were identical in all games, but the numbers of the cards stood for different colours in different games. To solve this issue the newly created similarity matrix for the glare effect was used to determine the dynamically mapped similarity values for each game. The old values in the glare effect logs are then replaced with the new values and correct mapping in section ref. The reason why these issues only apply to glare effect logs and not to no obstacle logs is that the simulator does not use any similarity values when simulating no obstacle games, meaning that these values are ignored anyway. Therefore there is no need to replace them. It should also be noted that both the glare effect and the no obstacle logs additionally include four statistical features for the last turn. These consist of the number of remaining cards, the number of never revealed cards, the highest number of times the same card was revealed and the number of rounds since all pairs were found. These four values are also ignored, as they are newly calculated for every turn in section ref. Last but not least all logs end with a label, describing in which game mode the log was created.

3.2 Brain activity

Eeg data collected from all proabnds during the game. However, there is one problem with using the eeg data: The overlyiong context of this work is to find interaction obstacles and ultimatley imporve the user interaction. This means that the way of discovering such an interaction obstacle should not worsen the interaction experience. If all people had to waer eeg masks when interacting with software just for the purpose of noticing interaction obstacles, the interaction experience itself would suffer. Additionally it is not cost efficient for every user to use and eeg sensor. As a result the decision was made not to use eeg data and instead only use the behvioural data that is collected direclty through the interaction and stays unnoticed by the user. As eeg data is not used in this work it will not be further explained.

4. CMM-based Cognitive User Simulation

The cognitive system lab developed the computer program cognitive memory model (CMM). It utilizes concepts of the act-r-theory to model the cognitive human memory. ACT-R is a cognitive architecture that aims to describe and integrate central basic mechanisms of human cognition. **TODO: ref:** https://www.google.com/search?q=intimates+deutsch&oq=iminates+&aqs=chrome.1.69i57j0l7.2543j0j15&sourceid=chrome&ie=UTF-8 In the context of the matching pairs game the CMM was used to implement a generative model. By modelling memorized and forgotten revealed cards and deciding in each turn whether to explore unknown cards or to exploit the gathered knowledge to reveal matching pairs, the course of a matching pairs game can be simulated taking the capability of human memory into account. Furthermore it is possible to define a similarity matrix to simulate similarities between cards in matching pairs games. This matrix includes similarity values for each colour combination of cards. Such a similarity matrix can be used in the context of visual interaction obstacles to emulate human confusion. In related research it was used to simulate the confusion caused by colour blindness and the usage of a red-green card set. For this thesis the only two modes that are relevant are the glare effect (no assistance) and the no obstacle (no assistance) modes. In order to simualte no obstacle games no simialrity matrix is needed, but for glare effect games a new similarity matrix is created in section **TODO: ref**.

To accurately simulate the performance of human memory multiple parameters are randomly initialized and then repetedly optimized using a genetic optimization algorithm. These parameters can be further extended. The genetic algorithm works by repeadily simulating game sessions and updating the parameters by applying mutation and selection operations. To select the best parameter values two performance measurements are defined and used to compare the simulated game sessions and a real game as reference. These perfomarmce measurements are the number of matching pairs and the number of penalties per round. Penalties are given if a card is revealed whose partner has already been seen before but the pair was not picked up. The optimization of the parameter population is repeated over many generations so that the performance in the simalted games best fits that in the real game. As example one of the paramameters optimimzed by the genetic optimization algorithm is the similarity decay. It reduces the similarity effects during the course of the game, since the user may learn to adapt to the visual interaction obstacle and therefore be less effected by it. Furthermore the similarity effect is continuelasly reduced as there are less cards in the game after pairs were discovered.

As input the simulator takes a number of game logs and simulated a specified amount of new sessions using the genetic optimization algorithm. For instance 20 game logs can be passed to simulate 1000 new sessions out of each game log, resulting in 20000 simulated games. The data contained in the game logs is shown in section **TODO: ref**. In the context of simulating no obstacle games improvements to the simulator were made that are explained in section **TODO: ref**.

5. Data Preparation

To prepare the data for the training many steps were neccesary. One big step is to increse the available dta by simulaing user behaviour. Therefore the simulator tor in chapter **TODO: ref** is used. For simulating games with no opbstacle there is no need for a similarity matrix. However, in order to sucessfully simulate glare effect games, a special similarity matrix is required. While without any obstacles the color differnes are independent from the position of the crads, this is not the case in glare effect games since the intensity of the light is not the same across the whole field. As a result the first step is to create a similarity matrix that descirbes the differneces of colours under the influence of the glare effect. Once completed, it replaces the old similarity matrix in the glare effect logs. Additionally all logs need to be remapped since they were mapped statically instead of dynamically. As not all logs are valid and can be used in the simulator, the invalid log must be removed. These valid logs are then used to simulate user behaviour and thereby create new logs. These simulated logs are sorted by their quality and the simualtion is evaluateed. If the performance in the simulated games is similar to that in the original games it is preceeded with the gerneration of the features for the training. Otherwise changes to the simulator ar made and the simulation, the sorting and the evaluation is repeated. In case of very bad simulations of glare effcet logs it would have also been possible to change the approch of creating the similarity matrix or find a new approch. However, the simualtinos of glare effect were very good, which can be seen in **TODO: ref**, which is why the spproach was kept.

5.1 Similarity matrix creation

5.1.1 Conceptualization

The aim is to create a similarity matrix that descibes the differneces between colours under the influence of the simulated sun light. The difficulty hereby is that the intensity of the light is not spread evenly across the whole field. Instead, as it would be if a real sun would shine onto the display, a certain are has the highest intensity and the intensity is lower the further away from that area. Adittionally there are wide lines of higher intensity comming from the brightest area, that simulate sunbeams. This influence of the simulated sunlight can be seen in figute 5.1.

Figure 5.1: Screenshot of the game field with simulated sunlight. Sunbeams are especially visible on the right hand side. All cards are turned face up.

TODO: vielleicht bild raus nehmen weil es oben schon einmal ist und referenzien nach oben

As a result simply extracting the rgb values for one pixel of each card in one game and calculating the differneces has two major problems: Firstly, the differences are highly influenced by the position of the specific cards. If the matrix is calculated using a single glare effect game the differneces are only representative for exactly that game and may be completely differnet if the cards are positioned differently. Secondly, extracting single pixels for each card in an image could lead to color values that do not represent the overall observed colour due to varying brightnesses on differnet points of a card. This behaviour is undesired, as the final similarity matrix is supposed to represent the differneces of the observed colours of the cards. **TODO: vielleicht quelle suchen die sagt das menschen farbenm ion bereichen mitteln oder so** In order to solve these problems a more complex approach then simply extracting single pixels out of a game was needed.

The problem of the positional influence can be solved by creating similarity matrices for more than one game and calculating the mean color differences of those matrices. The idea is that by creating using so many games that all or most combinations of positions and colours are included and calculating the mean of all comparissons for , the result will not be influenced by the position of the cards. However, first needs to be determined which number of games satisfies this condition. Each game consists of 14 cards, with each two cards having the same colours. When determining the difference between two colours there are two differnet cases:

- 1. The colours are not the same: In each game there are exactly 4 combinations of positions for those two colours, because there are two cards for each colour.

- 2. The colours are the same: In each game there is only one combination of positions for those colours.

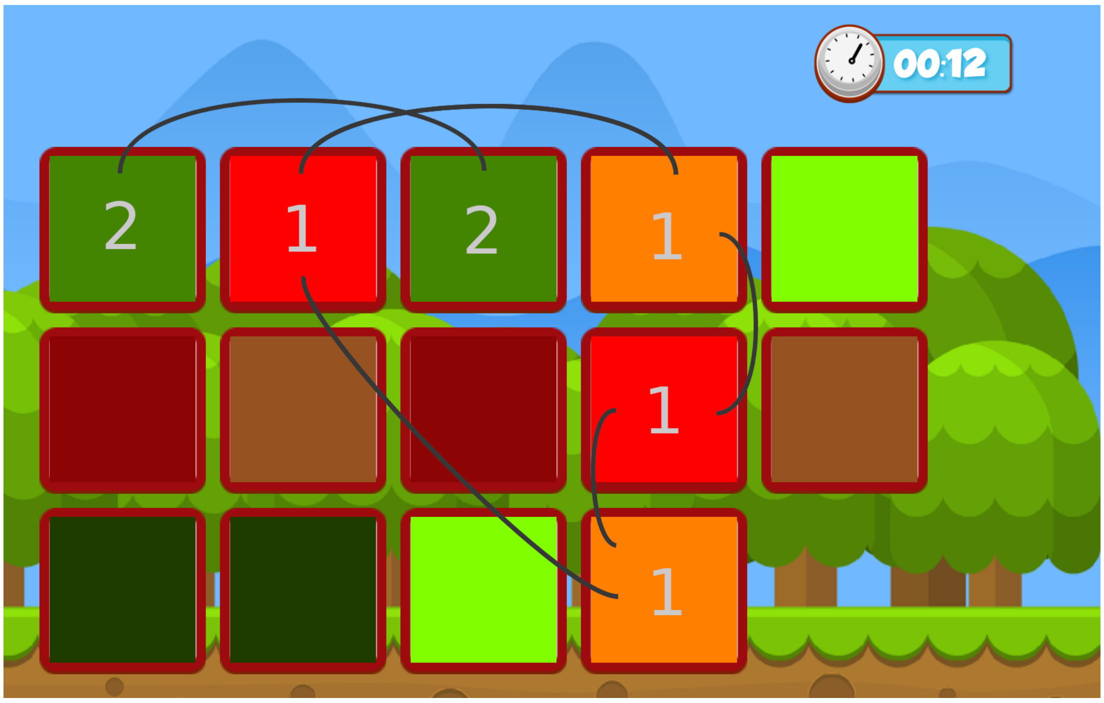


Figure 5.2: Examplary showing the number of different comparissons for case 1 and 2. The numbers on the cards represent the case and the edges indicate unique comparissons. All cards are turned face up. A game without any obstacles is used only for the purpose of clarifying the differnet comparissons. All screenshots used for the actual calculation are taken from glare effect games.

First it is calculated how many screenshots are necessary for the first case. For two different coloured arbitrary but fixed cards there can be

$$C_1 = 14 \cdot 13 = 182$$

combinations of positions are possible. The reason that it is not not $14 \cdot 14$ is that 14 comparissons of cards with themself would be included. We formulate the condition that enough screenshots need to be taken so that a arbitrary but fixed combination of positions for two different colours is included with a probability of 95%.

As there are 4 such combinations of positions in each game the probanility P(A) to get a specific combination in a game is

$$P(A) = \frac{4}{182}$$

The counter probability P(¬A) is

$$P(\neg A) = 1 - \frac{4}{182} = \frac{178}{182}$$

The number of necceary screenshots to include a specific combination with 95% probaility is calculated by solving

$$\begin{aligned}
1 - P(\neg A)^n &\geq 0.95 \\
1 - \left(\frac{178}{182}\right)^n &\geq 0.95 \\
1 &\geq 0.95 + \left(\frac{178}{182}\right)^n \\
\left(\frac{178}{182}\right)^n &\leq 0.05 \\
n &\geq log_{(\frac{178}{182})}(0.05) \\
n &\geq 134.8024
\end{aligned}$$

This means that about 135 Screenshots of games are needed in the first case. Now we perform the same calculation for the second case where there is only one combination of positions for the colours. However, there are less combinations of positions that are relevant than in the first case. For example for two green cards at index 0 and 1 the 182 comparissons would include a comparison of the first green card and the second green card as well as a comparisson of the second green card and the first green card. This goes for every two cards with the same colour, meaning there are only half as many different comparissons in the second case than in the first case. As a result the number of possible combinations in the second case is

$$C_2 = \frac{C_1}{2} = \frac{14 \cdot 13}{2} = 91.$$

Now we calculate how many screenshots must be taken to include a specific combination of positions for two equal colours with a probability of 95%.

As there is one such combinations of positions in each game the probanility P(A) to get a specific combination in a game is

$$P(A) = \frac{1}{91}$$

The counter probability P(¬A) is

$$P(\neg A) = 1 - \frac{1}{91} = \frac{90}{91}.$$

The number of necceary screenshots to include a specific combination with 95% probaility is calculated by solving

$$\begin{aligned}
1 - P(\neg A)^n &\geq 0.95 \\
1 - \left(\frac{90}{91}\right)^n &\geq 0.95 \\
1 &\geq 0.95 + \left(\frac{90}{91}\right)^n \\
\left(\frac{90}{91}\right)^n &\leq 0.05 \\
n &\geq log_{(\frac{90}{91})}(0.05) \\
n &\geq 271.111
\end{aligned}$$

This means that about 272 screenshots of games are needed in the second case. As there are more screenshots needed for the second case, the first case is also included. This inclusion is caused by the fact that each screenshot includes comparissons for the fist as well as the second case. To have a buffer the decision was made to take screenshots of 300 games. In theory these screenshot include any arbitrary but fixed combinations of positions and colours with a probability of over 95%, which should eliminate the problem of the positional influence.

Nonetheless, this does not solve the second problem of extracting single pixels that can potentially be directly in the area of a sunbeam. This can be fixed, by extracting all pixels in a certain are of the card and averaging their rgb values.

Last but not least it needs to be clarified how the colour differneces are calculated. To capture how colour differences are observed by humans, the rgb colour scale is not suitable. Using the rgb colour scale simirlarly stromng perseaved color differnces do not neccessarily have the same eucliidean distance. Contrary to that the CIELAB colour space aims to do exaclty that. Although not perfect, it more accurately descirbes human colour perception than the rgb colour space. Therefore the colour differences described in the similarity matrix are calculated after converting the colours into the CIELAB colour space. The colour distance is called Delta E score. The lower this score is the more similar appear the colours to human eyes. In related work, when creating the similarity matrix for the effect of colour blindness, the CIE1976 colour model was used for the calculations. However, the formula for calculating the colour difference has since then been improved multiple times, resulting in the CIE2000 colour model. Through multiple modifications of the formula it got closer to a visual equidistance, meaning simirlarly stromng perseaved color differnces have more similar Delta E scores than before. Therefore when creating the similarity matrix for the glare effect the CIE2000 colour model is used instead of the old one.

To sum it up, the chosen approach is to take 300 screenshots of glare effect games with all cards turned face up, extracting areas of rgb values for each card, converting the average rgb values of ares into CIELAB colours, calculating a similarity matrix for each game that contains the delta E scores and finally averaging the delta E scores across all similarity matrices. To achieve values between 0 and 1, the colour differneces additionally need to be scaled down in the end.

5.1.2 Screenshot extraction

To play the memory game and take screenshots an emulator for the pixel 2 in android studio was used. In order to take the screenshots in the needed way and collect additionally necessary information some changes to the memory game were made. In the way the game is intended there are always only maximum two cards turned face up. However, for the screenshots all cards need to be turned around so that the colour differneces can be calculated. Therefore the memory game was adjusted so that all cards are turned face up once the player turns a card. Additionally it was changed so that cards stay face up once they get turned around for the first time. This makes it possible to take screenshots with the colours of all cards visible. Furthermore, for the creation of the similarity matrix it needs to be known what the original colours without the influnece of the simulated sun are. To collect the screenshots and the according colours of the cards in each game a semi-automatic approach was chosen. Once a game is started and a card is flipped, resulting in all cards being turned face up, the colours of all cards are saved in a list. Then a screenshot is manually taken and afterwards the game is exited to enter the main menu. From there on the process is repeated 300 times. As a result there are 300 hundred screenshots of games and a two dimensional list that includes the colours of the cards in the 300 games. The first dimension specifies the game and the second dimension the index of the card. The values of this list are stored in a text file before the game is completely closed. To assure that no mistakes were made when taking the screenshots, the number of collected screenshots is observed during the whole process and afterwards all screenshots are manually checked so that all cards are turned face up.

5.1.3 Implementation

This section will show and explain the implementation of the similarity matrix generation. Code is only included if it helps to further understand what is being explained. First of all the screenshots and the information about the original colours of the cards are loaded. Before the actual calculation of a similarity matrix for each image, the average rgb values of the cards on the field need to be determined.

```python
def determine_glare_rgb_values(image):
'''
Calculates the rgb average rbg values for each card in an image.
:param image: The screenshot of the memory game with all cards
turned face up.
:return: The average rbg values in the specified 150 x 150 pixel
areas for each card.
'''
glare_rgb_values = []
for corner in card_corners:
x_border = corner[0] + 150
y_border = corner[1] + 150
card_values = []
for x in range(corner[0], x_border, 1):
for y in range(corner[1], y_border, 1):
coordinates = x, y
pixel_values = image.getpixel(coordinates)
card_values.append(pixel_values[:-1])
card_r = int(round(np.mean([color[0] for color in card_values])))
card_g = int(round(np.mean([color[1] for color in card_values])))
card_b = int(round(np.mean([color[2] for color in card_values])))
glare_rgb_values.append((card_r, card_g, card_b))
return glare_rgb_values
```

Listing 5.1: Add caption

The cards are each $250 \cdot 250$ pixels big and the rgb values are extracted from squared $150 \cdot 150$ pixel areas. The coordinates the pixels are extracted from are manually selected in gimp. As a result the areas are only correct for the used resolution of 1080p. Once the mean rgb values in those areas are calculated and converted into LAB colours the similarity matrix can be created. LAB colours are neccessary to calculate the colour differnece using the CIE2000 colour model. Each of the 28 cells in the similarity matrix falls into one of the two cases explained in **TODO: ref**, meaning there are either 4 or only 1 unique combination for the calculation of each cell value. For every combination the lab colour distance is determined and the values for each cell are averaged. This completes the steps for a single image, resulting in a similarity matrix with unscaled values. Once a similarity matrix with unscaled values for every image is created, the average of all matrices is calculated. During the whole calculation of the single matrices it is being kept track of the highest occurring colour differnece. The last step neccessary to complete the final similarity matrix is to divide all values in the matrix by use the highest colour differnce to scale them down to be between 0 and 1.

5.1.4 Testing

To verify the correcteness of the color extraction and the calculation of the similarity matrix, the main funtionalities are manually tested. The colour extraction was tested by extracting colours of a screenshot with the skript and compoaring the rgb values to the actual values in the images using gimp. Furthermore the calculation of the delta e score was tested, by comaring results of the script with those of an online calculator.

To assure that the 300 games actually include most of the combinations and therefore eliminate the positional influence of cards, the coverage of combinations can be additionally calculated. Therefore the number of unique combinations included in the 300 screenshots must be divided by the overall number of possible combinations. The number of unique combinations can be counted during the process of creating the similarity matrix, but the number of possible combinations must seperatly be determined. The similarity matrix for glare effect has 28 entries, from which 7 are comparissons between same and 21 between different colours. With *C_1* and *C_2* being the possible combinations for two arbitrayry but fixed coloured cards in the two cases mentioned above, the number of overall possible combinations C is

$$\begin{aligned} C &= 21 \cdot C_1 + 7 \cdot C_2 \\ &= 21 \cdot 182 + 7 \cdot 91 \\ &= 4459. \end{aligned}$$

The 300 games used for creating this matrix include 99.57% of all possible combinations. Furthermore is was verified that certain combinations are not included significantly more often than others and in that sense have stronger influence on the result. **TODO: code ergänzen und nachprüfen wie oft die combinationen vorkamen..histogramm?**.

5.1.5 Final Matrix

In figure 5.3 the final similarity matrix can be seen, dsiplaying how colour differneces are observed under the influence of the simulated sunlight, averaged over 300 games. As mentioned before, the lower the delta e score the more similar the cards appear to the human eye. The maximum delta E score that occured during the calculation was 8.861. All values are divided by that number to achieve scores between 0 and 1.

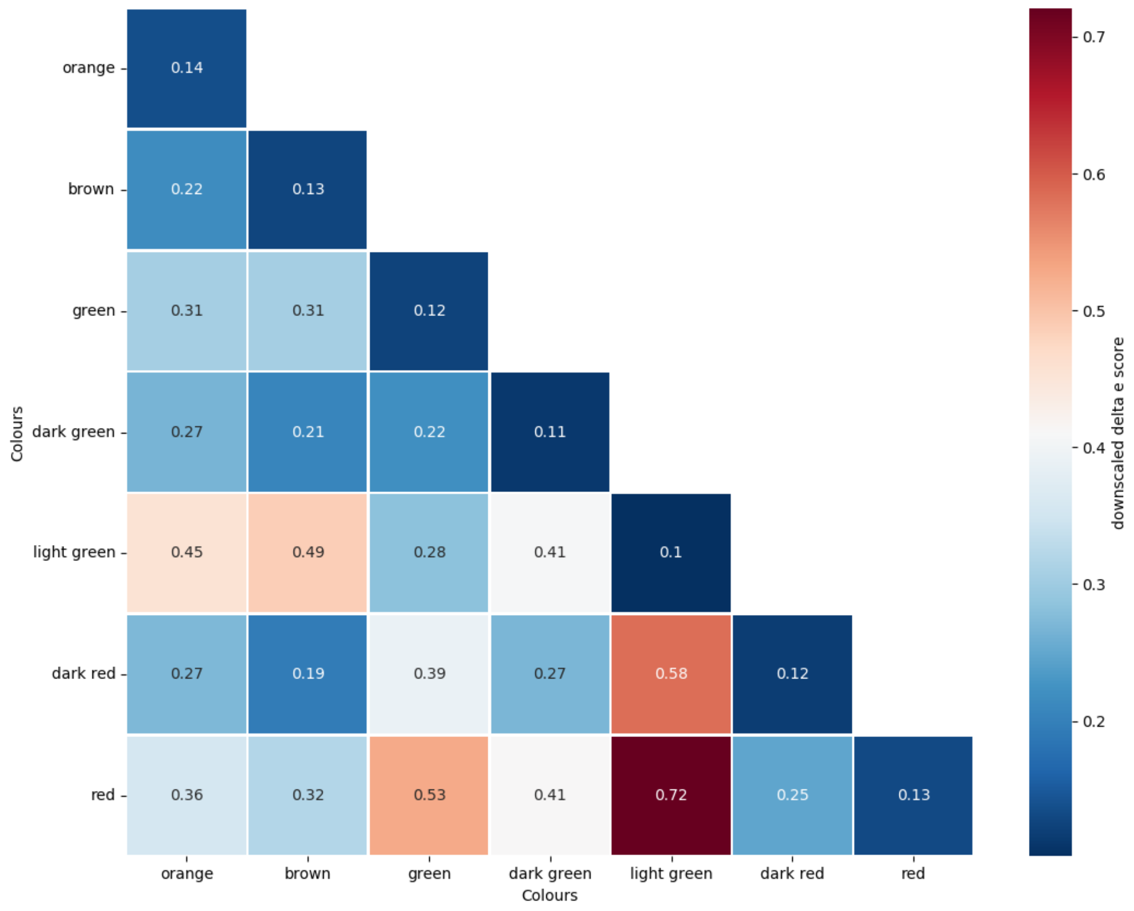


Figure 5.3: Add caption

The delta E score is less a definitive answer, and instead a helpflu metric to apply to a specific use case. Although there are tendencies regarding the interpretation of colour differences, there is no definitive table that descriebes what the different scores mean. One reason for this is that the perceived colour differnece may vary in different situations and circumstances. For instance, is the colour difference perceived differently depending on how long the colours are exposed to the human eye, as humans can over time adapt to the colours. This means that in a setup where all colours are always visible the colour difference will likely be perceived weaker than in the case of the memory game, as the cards are only flipped momentarily, leaving little time for adaptation. Table ref was created through own observation and in that sense is highly influenced by the strength of the sense of sight from the creator of this work. Therefore it should not be taken definitive but only serve the purpose of claryfying how the values in the matrix can roughly be interpreted.

Table 5.1: add caption.

Delta E	Downscaled values	Perception of difference
≤ 1.33	≤ 0.15	Not perceptible by human eyes
$1.33 - 2.66$	$0.15 - 0.3$	Only perceptible through close observation
$2.66 - 4.43$	$0.3 - 0.5$	Perceptible colour difference
≥ 4.43	≥ 0.5	Major colour difference

Before actually using the constructed simialry matrix in the simulator it is unknown if the chosen approach results in high quality simulations. If this is not the case it is possible to change the concept or try other approches.

5.2 Adaptation and correction of logs

One problem with the logs is, that the mapping was done statically when they were created. However, the simulator requires dynamic mapping. The difference is emplained in chapter **TODO: ref**. To do so there was already a script available, that was used after replacing some deprecated functions from libraries with supported ones. Furthermore the code was extended so that the old placehoulder similarity matrices in the logs are replaced by the new one. As the code was initially written for similarityg matrices with 21 entries and the one for glare effcet has 28 entries, the code for remmaping the logs had to be adapted. Finally all remapped logs are saved in new files. To check whether the remapping is still done correctly, the resulting logs were compared to logs that were remapped by the original script. The only thing that was not original were the replaced deprecated functions as this was neccesary to execute the script. Besides differnet similarity values due to differnet matrices as well as the fact that the logs now also contained entries for same coloured cards, the mapping was identical.

5.3 Removal of invalid logs

For the simualation to work, the game log that is used for the simulation must have had all cards turned around at least once. If this is not the case the log is classified as invalid. Additionally to removing invalid logs, the aim is to create a balanced data set for training. Therefore the number of games from each participant in each game mode have to be equal. This means that neither is it allowed to have more no obstacle than glare effect games, nor is it allowed to include more no obstacle games from a particiapnt than glare effect games, and vice versa.**TODO: quelle finden für balanced kram wieso das wichtig ist und begründen: vermutlich sowas wie: gerenelization weil sonst stärkere tendez su einer klasse und so (für gleiche anzahl), und gleiche participants muss man dann suchen** As stated in chapter **TODO: ref** from the 22 participants there are 44 no obstacle and 21 glare effect game logs.

To collect the data used for the simulation, a script was written that collects one no obstacle and one glare effect log from each participant. Half of the available no obstacle logs are not collected, because there are not as many glare effect logs. Once the logs are collected, they are validated and the valid ones are saved in new files. If at least one the two logs from a participant from different game modes is invalid, both are removed. Otherwise the training data would be inconsistent in that sense that the logs from different game modes could be from different participants. From the 22 no obstacle and 21 glare effect logs collected by the script, one glare effect log was invalid. This resulted in 20 no obstacle and 20 glare effect logs being used for simulation. These 40 real logs combined with those simulated form the data used for training.

```
def validate_log(log):
    '''
    Validates logs.
```

```
  :param log: The log to validate.
  :return: If the log is valid.
  '''
  needed_entries = ['1.1,', '2.1,', '3.1,', '4.1,', '5.1,', '6.1,', '
     7.1,', '1.2,', '2.2,', '3.2,', '4.2,', '5.2,', '6.2,', '7.2,']
  for needed_entry in needed_entries:
    if needed_entry not in log:
      return False
  return True
```

Listing 5.2: Add caption

5.4 Simulation of user behaviour

The whole process of simulating user behaviour is divided into two parts: First configuartion files are created, using a generic optimization algorithm, that contain the optimized parameter values and secondly these configuration files are used to simulate games.

Multiple additions were made to the simulator. In total four classes were added. Two are for generating configuration files for the simulation of game with and without the glare effect obstacle (NoObst_ConfigGenerator_dataCollection2020.java and GlareEffect_ConfigGenerator_dataCollection2020.java) and the other two for using those files to simulate new games based on the user behaviour in the original games (NoObstWinStrategyTrainingDataGenerator_dataCollection2020.java and GlareEffectWinStrategyTrainingDataGenerator_dataCollection2020.java). These classes utilize the functionalities already implemented in the simulator. **TODO: grob erklären wie code funktioniert** However, as the simulator only worked with similarity matrices that have 21 entries, instead of the 28 of the matrix for glare effect games, the simulator was expanded so that it can also handle matrices with 28 entries. After all changes and additions were completed, each log was used to simulate 1000 games, resulting in 40000 logs. From these logs 20000 are no obstacle games and the other 20000 are glare effect games. The 40000 logs contain 1000 no obstcle and 1000 glare effect logs for each of the 20 probants.

5.5 Sorting logs by quality

The siulated logs ware sorted by their quality, using the root mean squaed error between the perfoamces....**TODO: klären genau was da gemacht wird mit mazen, ich galube rmse für alle perfomace werte in einem game und dem originalen game** as measurement. This is important because it enables to only use the best n simulated logs for training instead of using all of them or a random subset.

5.6 Evaluation of simulation

To evaluate whether the performance in the simlated logs is similar to the one in the original logs two perofmance measurements are used: The matching pairs in each round and the penalties in each round. **TODO: erklären was penalty ist** The Initial simulation results can be seen in figure 5.4 and 5.5. **TODO: erklären was whiskers sind**

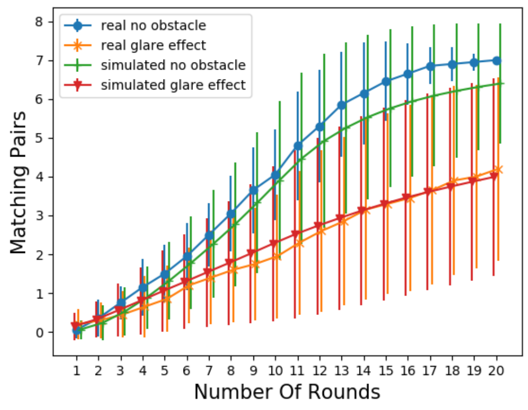


Figure 5.4: Add caption

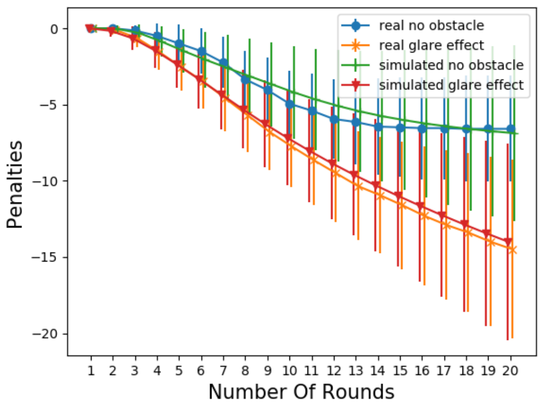


Figure 5.5: Add caption

It can be seen that the simualtions of glare effect games are very good, meaning that the constructed similarity matrix creates good simulation results. As a result there is no need to fin a differnet approach for creating the similarity matrix. However, escpecially when looking at the matching pairs per round after the tenth round, it is noticeable that the perofmance in the simualted no obstacle games is not as good as in the original ones. This observation inspired two changes to the simulator. The first one was to introduce a new parameter for the optimization algothithm to optimize. It was called randomizing decy and is a value between 0 and 1, and adresses the problem of worser performance in the löast turn compared to real games. The boundary limit, describing..., is multiplied with the randomizing decay each round beginning with the 20th round. This reduces the ... and therefore results in better perfomance in the last 10 steps. The second change was to reduce the probability of revealing a random card (instead of puruing a win strategy) by 0.2 (randomrevealprobabilty is a random value betwen 0 and 1). This was done to increase the overall performance in the simulations by reducing the randomness. The results of these changes can be seen in figure 5.6 and 5.7.

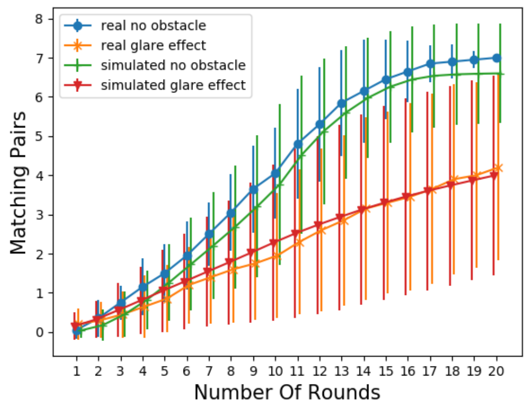


Figure 5.6: Add caption

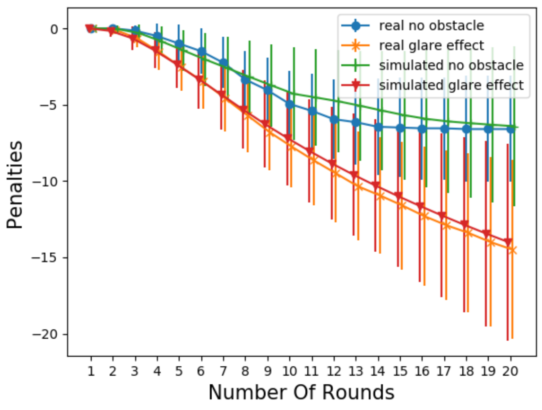


Figure 5.7: Add caption

TODO: fertig schreiebn

It is visble that a improvement to the simulator was made regarding the perfomrance measurements. (ich glaube lkeine statistischen tests fdafür nur wenn ich nopch zeit

habe..). Although the graphs show that the simuletad data is very close to the original one, statistcal test will give futher knowledge and certainty.
- t-stochastic test kram (siehe mazens nachricht) -> paired t test: gucken ob no obstacle und glare effet signifikant unterscheidlich sind -> ja sind sie
- vielleicth auch gucken ob verbesserung das signifikant besser gemacht hat, aber kann eventuell auch vernachlässigt werden weil man es visuell sieht und der unterscheid nicht starks sein wird. Dennoch könnte es gut sein dass zu machen.
- auch schriewben was paired ttest überhapt ist und wozu es gut ist.

Table 5.2: p values in paired t-test for different comparissons of matching pairs per round. All 1000 simulated games per real game were used. The following abreviatrions are used: real glare effect - r_g, real no obstacle - r_n, simulated glare effect - s_g, simulated no obstacle - s_n

	s_n	s_g	r_n
r_g	$5.7e-06$	0.0688	$9.3e-07$
r_n	$1.8e-10$	$2.7e-06$	
s_g	$1.7e-05$		

Table 5.3: p values in paired t-test for different comparissons of penalties per round. All 1000 simulated games per real game were used. The following abreviatrions are used: real glare effect - r_g, real no obstacle - r_n, simulated glare effect - s_g, simulated no obstacle - s_n

	s_n	s_g	r_n
r_g	$4.8e-06$	$1.1e-06$	$3.2e-06$
r_n	0.0195	$5.6e-06$	
s_g	$7.2e-06$		

Problem: Glare simualted und real haben signifikanten unterschied. gleiches gilt für no obstacle simulated und real.

Tried to find the amount of simulated games per real games and the ranghe of steps to be included so that all values are as desired. The search was manually done by varying the number of steps and the ratio of simulated and real data. However, a configuration resuts in all values as desired could not be found. If less simulations are used, although the red emphasized undesired values change to the better the fomerly given and desired non significant difference in matching pairs between simualed and real glare effect games vansiheds and turns significant. This results at leats in their beeing oinly one undesired value instead of 3. An example of such an configuration can bee seen in ref (10 steps und sd10x) ..:

Table 5.4: p values in paired t-test for different comparissons of matching pairs per round. For the simualted data only the best 10 simulated logs from each real log are used. Only the first 10 rounds are used. The following abreviatrions are used: real glare effect - r_g, real no obstacle - r_n, simulated glare effect - s_g, simulated no obstacle - s_n

	s_n	s_g	r_n
r_g	0.0064	$1.5e-05$	0.0058
r_n	0.0882	0.0039	
s_g	0.0044		

Table 5.5: p values in paired t-test for different comparissons of penalties per round. For the simualted data only the best 10 simulated logs from each real log are used. Only the first 10 rounds are used. The following abreviatrions are used: real glare effect - r_g, real no obstacle - r_n, simulated glare effect - s_g, simulated no obstacle - s_n

	s_n	s_g	r_n
r_g	0.0020	0.0564	0.0014
r_n	0.9005	0.0012	
s_g	0.0017		

A configuration in the middle could not be found as increasinf the number of simulations or the number of steps included results in opther undesired values.

It should be empghazised that this does not mean that the simulation of the glaree ffect games is bad. This only means that the paired t test finds a significant difference between the simulations and the real glatre effect games. If the two lists of mean values that supposedly are significantly different are manually compared it can be seen that in reality the difference is not that significanrt.

Table 5.6: mean values of matching pairs for real and simulated games. sd10x. for the first 10 rounds

	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10
real	0.2	0.3	0.45	0.65	0.85	1.2	1.4	1.6	1.75	1.95
simulated	0.135	0.27	0.405	0.6	0.79	1.105	1.34	1.53	1.72	1.915

The highest differnece is in round 6 with a differnece of 0.095 matching pairs in that round. This shows that it does not mean that all glare effect simulations used are bad regarding the matching pairs, only because the ttest finds a significant difference. The fact that the paired t test concludes a significant differnece could be related to the fact that the range of the value sis verry small ranging only from .. to .. meaning a samall differnece is much according tpo the ttest..**TODO: checken ob das logisch ist, also gucken was der ttest überhaupt macht und das begründen** (auschnitt beschrieben wenn ich weniger nehme.). This would also fit with the fact that using more steps and therefore having a bigger range of values for matching pairs in glare effect games, the difference is not significant (see example where all data is used, ref).

altoough no clear boundary was found in which all simulations have no significant ifference to the according real games was not found it can be said that the simulator is capable to at least simulate the first 10 rounds realistic enough to not be significantly different accord to the paired t test, given only the 10 best simulations are used. This analysis also indicated, that it is not ideal to use all simulated data for training as their is a significant difference between the simulated and the real games. As it can be seen in chapter results, using a certain amount of simualtions increaeses the overall performance copmared to only using real games. This observation paired with the knowledge that using too many simulations will result in a significant difference between the simulated and the real games, leads to the theory theory that there should be a sweet spot regaridng how many simulations to use. By using differnet amount of steps and simualetd data this this sweet spot is tried to be discovered in chapter ref to rsults.

meaning the smulator is not perfect..but..

TODO: das hier nach besser einordnen. Vielleicht stattdessen sd10x nehmen und sagen, wenn man das plottet was oben beschrieben wird dann sieht man auch dass die simulirten spiele den echten im schnitt noch ähnlciher sind. insbesondere die ersten 10 züge. Der simulator hat vor allem in späteren zügen problme mit der genauiogkeit.

However, during the training not all of the simulations are used. Only a certain number of the best simulations are used. The reason for this is that using to many simulations during training results in a strong adaption towards the simulations which in turn results in worse accuracy on real games. Therefore a sweet spot of the ratio between real and simulated games must be found. This can be seen in chapter **TODO: ref zu training chapter**. Figures 5.8 and 5.9 show the performance differnece between real and simulated games if only the best 10 simualted games from each probant are used, resulting in a ration of 1 real game to 10 simulated games.

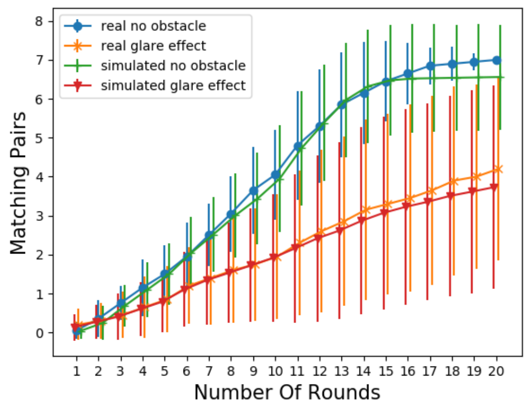


Figure 5.8: Add caption

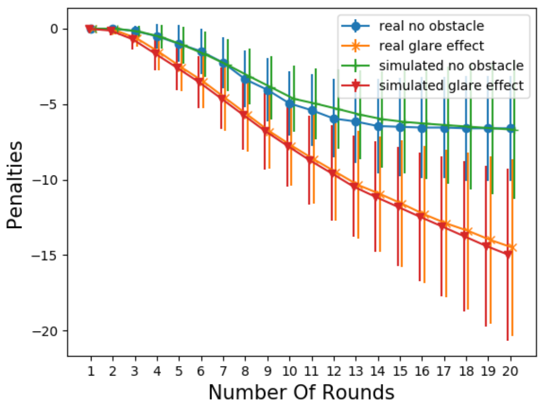


Figure 5.9: Add caption

Overall the quality of the simualtions is very high when comparing the performance in the simalted and teh origianl games. If only the best simulations are used the perfomace becomes even more similar.

5.7 Feature generation

5.7.1 1D CNN features

For the 1D CNN five statistcal features for each step are used: The card codes, the number of cards left, the number of never revealed cards, the highest number of times the same card was revealed and the number of steps since all pairs were found. They are calculated using the game logs from **TODO: ref zu dem data chapter**. These features can be direclty fed into the 1d CNN explained in chapter ..**TODO: ref**.

The code for calculating the statistal features out of game logs was already given. A script was written that incorparates this funtionality in order to calculate the features for all logs. Additionally, after creating the neccessary directory structure the script saves the files for the splits of the raw data and the features. The reason for saving the features is that they do not have to calculated before evrey training and instead can be loaded from files. The reason for also saving the raw data even though this work does not need them anymore is, that the working group that was collaborated with also trained other models and does not directly load the features but instead caalculates them before each training. **TODO: das ist komisch erklärt. vielleicht besser erlätern**

5.7.2 2D CNN features

For the second approach of using a 2D CNN further steps are taken. Therefore synthetic images in gray scale colours are created using the features mentioned above. As the images are in gray scale colours, only one color channel is needed. One can visually think of this approach as creating graphs for each feature that display their values in each timestep, taking a photograph of each graph and stacking them on top of eachother. This can be clarified by looking at the code that produces the synthetic images.

```python
def create_image(game, components=[True, True, True, True, True]):
  '''
  Creates synthetic images out of the five staticial features.
  :param game: The statical features in each step.
  :param components: Five values describing which of the features
  should be used to create the image.
  :return: The synthetic image.
  '''
  card_codes = np.zeros((7, steps))
  cards_left = np.zeros((8, steps))
  never_revealed_cards = np.zeros((14, steps))
  max_same_card_reveals = np.zeros((20, steps))
  rounds_since_done = np.zeros((27, steps))

  x_position = 0
  for step in game:
    card_code = math.floor(step[0])
    first_or_second = int(round((step[0] % 1) * 10))
    if card_code != 0:
      card_codes[card_code - 1][x_position] = first_or_second
    pairs_left[int(step[1] / 2)][x_position] = 1
    never_revealed_cards[int(step[2])][x_position] = 1
    max_same_card_reveals[int(step[3])][x_position] = 1
    rounds_since_done[int(step[4])][x_position] = 1
    x_position += 1

  image = np.zeros((0, steps))
  if components[0]:
    image = np.vstack((image, card_codes))
  if components[1]:
    image = np.vstack((image, max_same_card_reveals))
  if components[2]:
    image = np.vstack((image, rounds_since_done))
  if components[3]:
    image = np.vstack((image, cards_left))
  if components[4]:
    image = np.vstack((image, never_revealed_cards))

  return image
```

Listing 5.3: Add caption

By using pseudo colors, the images can be displayed with colours, like in figure 5.10. These 75 · 40 · 1 (height · width · colour channels) images are direclty fed to the 2D CNN explained in chapter .. **TODO: rerf**. By setting the origin of the image to the lower instead of the upper left corner a more naural looking image is created and can be seen in figure 5.11.

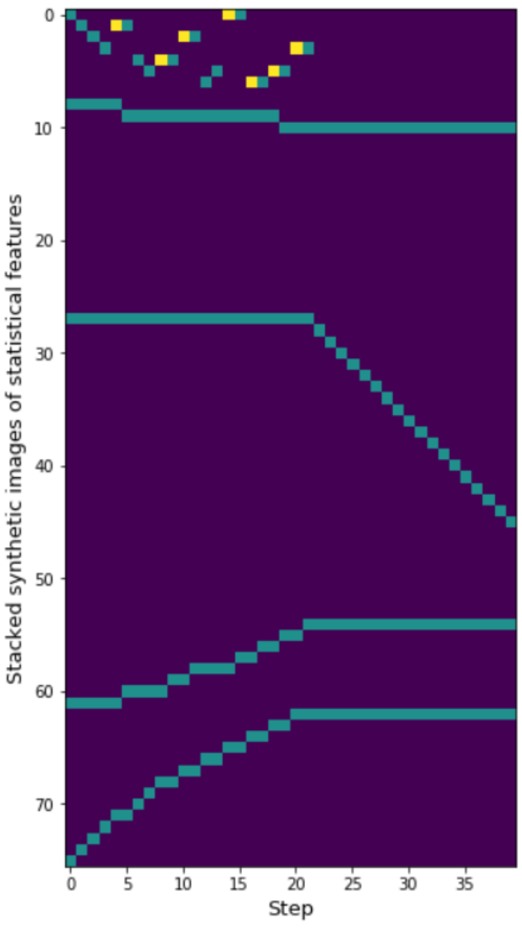


Figure 5.10: Add caption

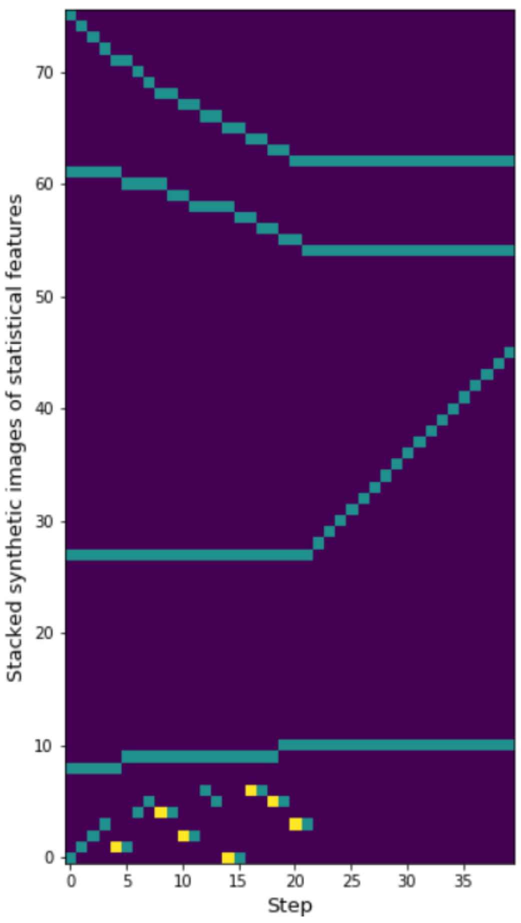


Figure 5.11: Add caption

The width of the image is 40 because that the number of steps recorded. Table **TODO: ref** shows which of the areas in the image describe which statical feature. The different vertical spaces given to statistical features were purposefully chosen. The aim was to use as much space as neccessary but as little as possible. The card codes have 7 pixels of vertical space since there are 7 different colours in the game. As each round consists of two cards being turned face up and the impossibility to chose the same card twice in one turn, a card can be at maximum revealed 20 times during 20 rounds. Considering that this value is at minimum 1 because it is not possible to flip no cards, the range of 20 values is sufficient for this value. Furthermore 14 steps are at minimum needed to complete the game, since there are 14 cards. As a result this value can range from 0 to 26, meaning a vertical space of 27 is needed. In order to save space, the statistcal feature of the number of cards left was converted to the number of pairs left. By dividing by 2 the vertical space neccessary to visualize this feature is halved, without information being lost. Last but not least the number of never revealed cards can range from 0 to 13. The value 14 is not possible because two cards have to be chosen each turn. The stacking order was chosen so that the coloured pixels of different statistical features rarely touch each other. **TODO: cnn anpassen und neu trainieren. Dimesnion zahlen anpassen in bachelor arbeit. Alle bilder nochmal machen. Code korrigiere in tex**

Table 5.7: add caption. Upper and lower boundaries are inclusive.

Statistical feature	Range	Vertical space in the image
Card codes	1-7	0-6
Maximum of same card reveals	1-20	7-26
Steps since game ended	0-26	27-53
Pairs left	0-7	54-61
Never revealed cards	0-13	62-75

The decision was made in the card code section to use different colours depending on if its the first or second card of a colour. The idea is that this emphasizes important behavioural characteristics that help deciding whether the visual obstacle being the sunlight is involved or not. By comparing the the card code sections of two images from which one is created using a glare effect and the other one with a no obstacle game, these visual differneces become clear.

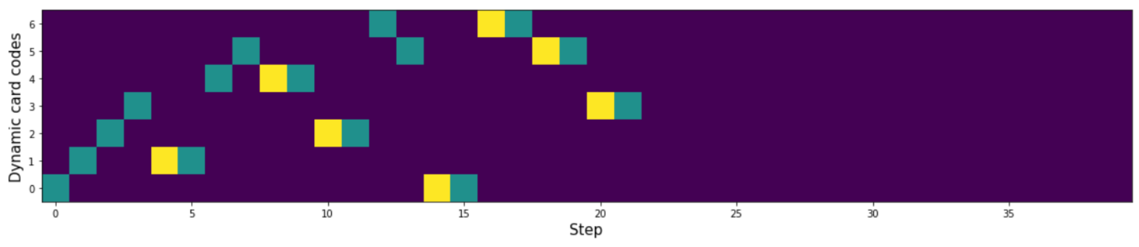


Figure 5.12: Add caption

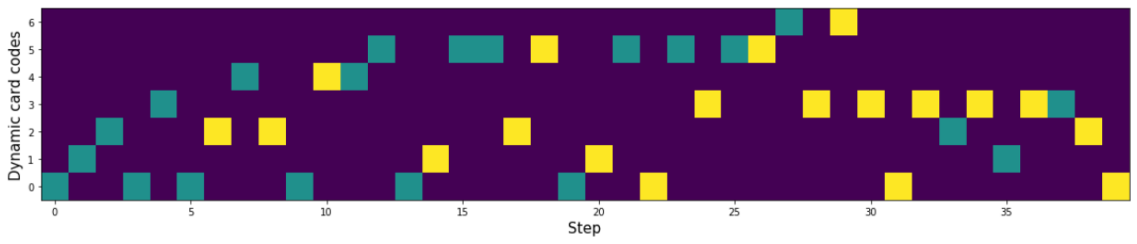


Figure 5.13: Add caption

In 5.12, showing the image for an no obstacle game, there are yellow pixels directly left of green ones, meaning that the probant flipped a card, knew that he had already seen the matching card and direclty flipped it. However, this happens less often in figure 5.13 which shows the image for an glare effect game. This is not the case in every game, but the theory is that separated colours in the image are more likely in glare effect games than in no obstacel games. Therefore it could be beneficial for the model to also learn these characteristics. If the same colour was used for all card codes, there would be no visual differnece between flipping the same card in consequtive turns and flipping two different cards with the same colour. This of course would also mean that the model could not differentiate bewteen those two cases.

6. Convolutional neural networks

As convolutional layers are complex and have many variations for differnet use cases, only concepts and functionaslities that are relevant for this work will be covered. This also menas that 3 dimensional cnns will not be explained, but in generel cnns function the same way whether they have 1, 2 or 3 dimensions. The main differnce lies in how the filter, also known as kernel or festure detector, moves acrross the data. This is later explained in more detail. (**TODO: ich glaube input data ist keine regel sondern nur meist so..zum beispiel kann man auch 1d convoltion bei 1d, 2d, oder 3d daten machen**).
First of all the core concepts of cnns will be explained and afterwards the models used in this work will be described.

On of the core concepts of convoluntional neural networks are the concolutions.

Unituitievly, a 2d cnn must not neccerailty have 2 dimensional data as input.

notes bei models:
- stride = wie viele schritte auf einmal, default 1 deshalb vermute ich bei mir 1
- maxpooling: pool size = 2*2, und bei keras wird stride dann defauklt auch zu 2*2 (=pool size)
- input size = 85*40 glaube ich
- filter weights are randomly initialized, so that they dont all initially learn the same feauters. To briefly explain the behaviour of filters a seperation bettween two cases can be made: The first case is that not all features of high quality have been learned yet, high quality meaning that learning them would lower the cost function. In that case it is highly unlikely that each filter would begin to resemble other filters, as that would most certainly result in an increase of the cost function and therefore no gradient descent algorithm would head in that direction. The other case is that all features of high quality have already been leanred by a subset of the avaibable filters. In this case the cost function would not be increased if the remaining filters learn similar features than the other filters.
- in einem der schritte wird eine zeile usgelassen, aber ich trainiere nicht nochmal

alles neu (oder?)

- 1d and 2d convolution - reduction of dimension - relu - number od filter, random initializion, etc - kernel and kernel size - loss function and optimizer, stochastic gradeint decent - Dropout layers - Pooling, max pooling - flatten - dense relu, softmax - droput layer, pooling ändert nichts an der anzahl der fetsure maps - **TODO: bessere quelle finden, buch oder so** - stride (ganz kurz, beschriebt verhalte wenn etwas über bleibt, zero padding oder weglassen) - back propagation

7. Classification

7.1 Hardware resources

- server for storing the training data as the results - 3 computers accesseed via ssh. One is more powerfull ... specs table than the other two .. specs table. These 3 computers have direct acces to the data server and load the training data from there. While the data server is optimized for storing data the other 3 computers used focus on computing power for fast training of modells.

7.2 Training setup

- one script for training the 1D cnn and one for the 2d cnn. Only difference is in the structure of the cnns and that before training the 2d cnn the loaded statistical feaures must first be used to create the synthetic images. - Both scripts load the same statistical features, already divided into 20 splits. As there are statistical features for 20 real and 20000 simlauted no obstacles games as well as the same amount for glare effects games, this results in each split containing data from 19 real and 19000 simualted games for each of the two classes, resulting in 38 real and 38000 simulated games available for the training. From the left out 2 real and 2000 simulated games, only the data from the real games is used for testing. This is done becauuse it is not desirable that the models adapt to the simualted data as in reality they will be only used in real games.

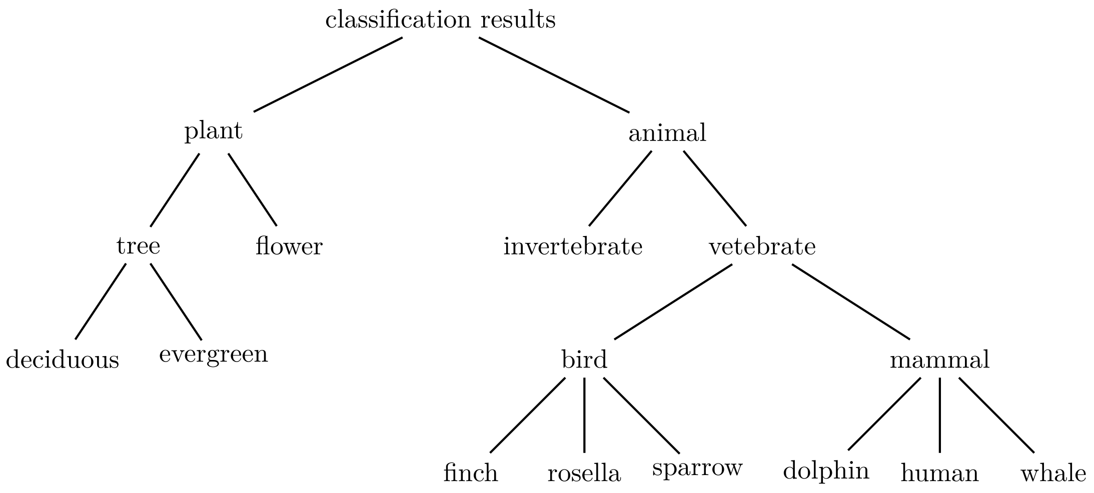
- it can be chosen in a list which ratios between simualted and real games schould be traiuned. It was chosen to use 0, .. 10, 20

- it can be chosen how many steps in the game should be used. The less the faster is the decision is made and assistance can be applied, but the longer wated the better the results (likely). So modells for different amounts of steps are trained. Steps: 10, 20, 30, 40 (or in turns divide by 2).

- The structure and hyper parameters of the modells for all 1d cnns are identical. Same goes for all 2d modells. The only differnece is the input shape of the modells since it depends on teh number of steps used in the training. Identical structure for modells of the same type are used so that different training results for different amount of steps can be reliably compared. It should be noted the structure and hyper paramters are not the same between the 1d and 2d cnns. This is only important for models of the same type.

- The scripts create and use a specific directory structure conceptualized in figure ref.

... figure



structure erklären

- in everey iteration: multithreading 20 cores. one for each split and the resuls for each split in every iteratrion . This also means all modells are traing using the cpu. Since all used computers especially the one from table ref have many fast cores, multithreading gives fast training results and enables parallel training of multiple modells **TODO: ich glaube das geht nicht mit gpu aber nochmal checken**.

7.3 Models

For determining whether probands were blinded, one dimensional and two dimensional convolutional neural networks were utilized. One dimensional convolutional neural networks have become popular for time series classifications **TODO: ref**. Two dimensional coovolutinal neural networks are mostly used for image processing. However, a possibility is to create synthetic images from the data and use these images in a two dimensional convolutional network. The widths of the data and the feature maps in the visualizations are calculafed for the case that all 20 rounds are used. If a diferenet nuber of rounds is used the widths are differentbut can be determined likewise **TODO: klarstellen**. The cnn models are intentionally created not very complex for two reasons: Firstly, a lot of training needs to be done with differnet configurations regarding the steps includes and the ratios of simulkated to real data. As the time this work is created in is limited, training all the modells for a sufficient analysis would take too long. Especially the training of the 2d cnns requires a lot of time. Secondly, to assure that the structures are complex enough, more complex modells were exemplary trained and they showed either similar or worse performance.

7.3.1 1D CNN

The structure of the 1D CNN used is shown in figure 7.1.

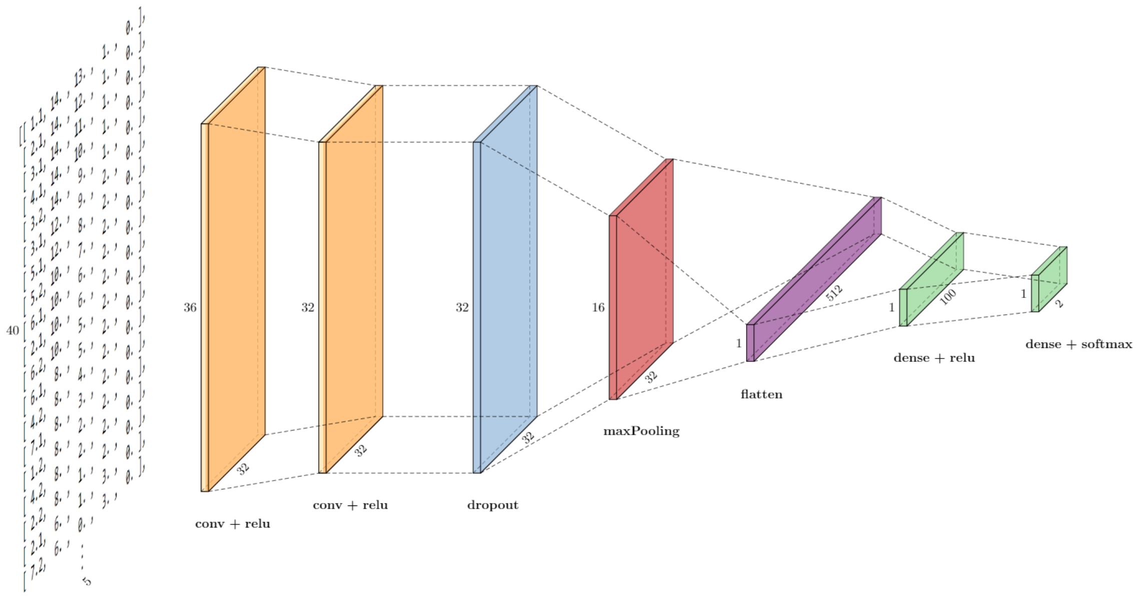


Figure 7.1: Add caption

TODO: anpassen an 1d cnn diagramm, weil es vorher falsch war The input data consist of the 5 statical features explained in .. **TODO: ref** for each of the 40 steps (20 rounds), resulting in the input dimensions 40 · 4. Initially the data is passed to the first convolutional layer of the network. The first layer has 32 filters and the kernel has a size of 5, meaning that every step from the kernel includes all data from 5 steps in the game. This convolution results in 32 festure maps each with dimensions of 36 · 5. These feature maps are passed to the second convolutional layer with the same number of filters and the same kernel size as the first layer. This again results in 32 feature maps and a decresed dimensionality of 32 · 5. Afterwards a dropout layer with a rate of 0.5 randomly sets input units to 0 and by that helps to prevent overfitting. Then a one dimensional max pooling layer with a pool size of 2 reduces the dimensinalty to 16 · 5. And finally the network is completed with a flatten layer and two dense layers **TODO: vielleicht genauer erklären was dense macht, aber eigentlich ja oben auch schon, nur einen groben satz hier**. Both convolutional layers, and the first dense layer use a rectified linear unit as activation function. The last dense layer uses softmax normalization to bring the data down two two values for the two classes. The elemets of the output vector are between 0 and 1 and sum up to 1 and specify the result of the classification.

total trainable parameters: 57.486

7.3.2 2D CNN

The structure of the 1D CNN used is shown in figure 7.2.

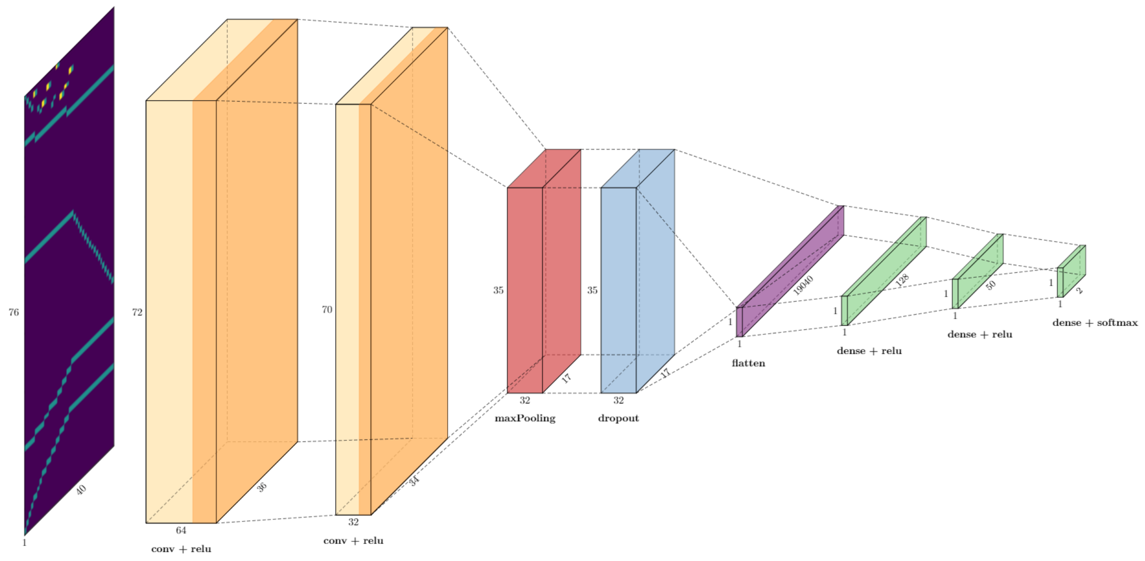


Figure 7.2: Add caption

The input data consist of the synthetic images generated in .. **TODO: ref** using the 5 statistical features, resulting in the input dimensions 75 · 40. Initially the data is passed to the first convolutional layer of the network. The first layer has 64 filters and the kernel has a size of 5 · 5, meaning that each step from the kernel includes a 5 · 5 are of the synthetic image. This convolution results in 64 festure maps each with dimensions of 71 · 36. These feature maps are passed to the second convolutional layer with 32 filters and a kernel size of 3 · 3. This results in 32 feature maps and a decresed dimensionality of 69 · 34. Then a two dimensional max pooling layer with a pool size of 2 · 2 reduces the dimensinalty to 35 · 17. It is important to note that this does not happen by default. Due to the fact that 69 is odd, by default the last row would be ignored during a max pooling with dimensions 2 · 2 dimensions, meaning the dimensions would be 34 · 17 instead. As this is undesirable due to the potentially lost information in the last row, zero padding was used to create an additional row with zeros. This additional row results in the pooling not having to ignore the last row **TODO: weil sich dimension auf 76 ändert stattdessen schreieb: da die height und width gerade sind und das pooling 2 mal 2 ist, wird beim pooling keine reihe oder spalte ausgelassen, es wird also kein zero padding benötigt**. Afterwards a dropout layer with a rate of 0.2 randomly sets input units to 0 and by that helps to prevent overfitting. And finally the network is completed with a flatten layer and three dense layers **TODO: vielleicht genauer erklären was dense macht, aber eigentlich ja oben auch schon, nur einen groben satz hier**. Both convolutional layers, and the first two dense layers use a rectified linear unit as activation function. The last dense layer uses softmax normalization to bring the data down two two values for the two classes. The elements of the output vector are between 0 and 1 and sum up to 1 and specify the result of the classification.

total trainable parameters: 2.463.928

Visualizing intermediate activations

TODO: vielleichtz diese section rausnehemen. ich denke es ist unnötig, weil es nicht dazu beiträgt was meine arbeit zeigen soll

An interesting thing about 2d cnns is that it is possible to vizualize the reprenestations learned by the network. This helps to understand the meaning of individual filters. Intermediate activations canm be vizualized by displaying the feauture maps that are the outpu of various convolution and pooling layers in the network, given an image is passed to the input layer. **TODO: ref** To extract these feature maps, the model is first trained for one split and then run in prediction mode. Vizualizing each feature map produced by filters gives a view into how an input is decomposed into the different filters learned by the cnn. The decision was made not to focus too much on vizualizing all feature maps but instead to only vizualize one feature map created by each layer. As each feature map can learn different features, the following visualizations are only small fractions of what the cnn learns. Thgerefore they should only serve as exemplary illustration of what the different layers of the cnn output.

TODO: bisschen beschreiebn was was ist von den bildern und was man sieht

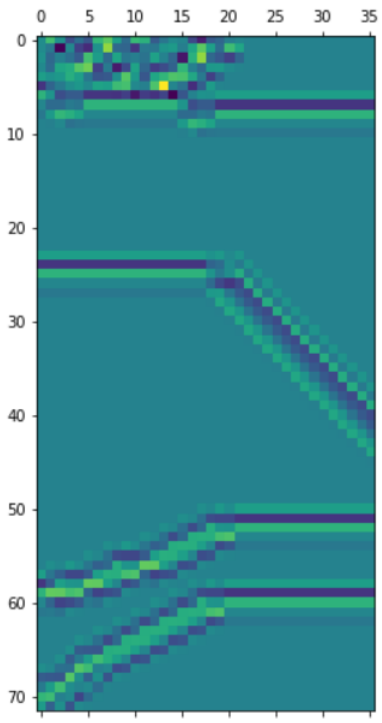


Figure 7.3: Add caption

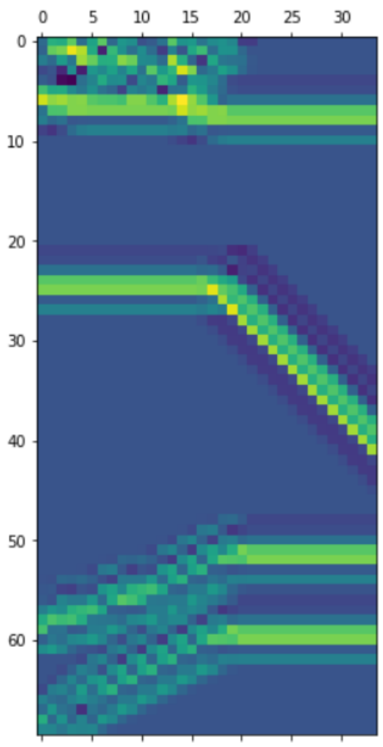


Figure 7.4: Add caption

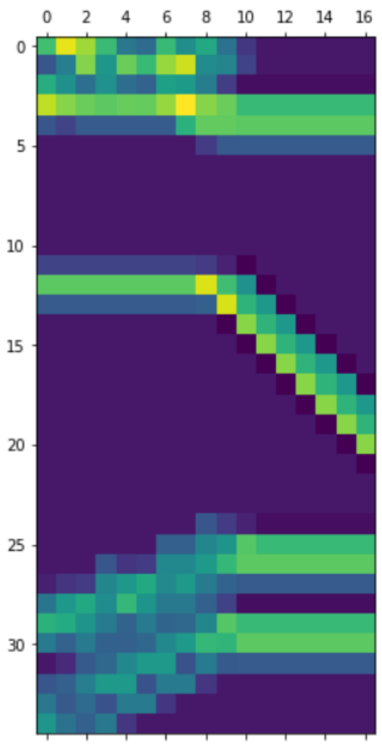


Figure 7.5: Add caption

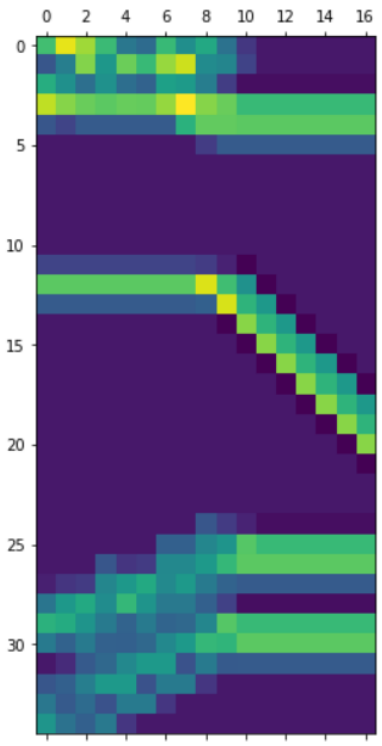


Figure 7.6: Add caption

It can be seen that the original structure of the image is still recognizable after all convolutional, pooling and dropout layers. When dealing with cnns that have more layers, usually the original image is at some point not recognizable anymore. The remaining layers of the 2d cnn effectively opperate on one dimensionaml data and create the output. Therfore they will be left out in this vizualization.

TODO: da wird ein buch erwähnt, wo der typ erzählt wieso es gut ist das zu machen. Die zitate umschreiben in meinen text: https://towardsdatascience.com/visualizing-intermediate-activation-in-convolutional-neural-networks-with-ke

7.4 Training and evaluation

Table 7.1: 2D CNN. best accuracys for 5, 10, 15 and 20 steps. in brackets behind the accuiracy is the epoch.

Simulated Games	5 steps	10 steps	15 steps	20 steps
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
20				

Table 7.2: 1D CNN. best accuracys for 5, 10, 15 and 20 steps. in brackets behind the accuiracy is the epoch.

Simulated Games	5 steps	10 steps	15 steps	20 steps
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
20				

Table 7.3: 2D CNN. 20 turns. config2

Simulated Games	Best Accuracy (Epoch)	Best Loss (Epoch)
0	0.7888 (14)	0.4822 (51)
1		
2		
3		
4		
5	0.8450 (17)	0.4768 (20)
6		
7		
8		
9		
10	0.8475 (20)	0.4796 (10)
20	0.8437 (6)	0.4763 (6)

- 1d cnn können besser mit kurzen sequencen umgehen als 2d. aber 2d sind bei mehr schritten deutlsich besser. (erste einschätzung)

Table 7.4: 2D CNN. 20 turns. config1

Simulated Games	Best Accuracy (Epoch)	Best Loss (Epoch)
0	0.7963 (18)	0.4931 (98)
1		
2		
3		
4		
5	0.8437 (53)	0.4821 (31)
6		
7		
8		
9		
10		
20		

- auch zeigen dass es sinnvoll ist simulierte daten mit zu vermwendet, also sd0x mit bestem vergleichen und so

- vielleiht gucken welche falsch erkannt werden und woran es liegt, also wenn die zum beispiel echt schlect oder gut sind obwohl es nicht so sein sollte (rausnhemen und gucken wie ergebnisse sind, vielleicht nur bei bestem modell)
- in gleichen tabellen ergebnise vor änderung am simulator und nach änderung betrachten und vergleichen
- training auf welchem rechner/n,was von: cpu oder gpu oder beides, hardware kurz erwähnen, vpn (fernzugriff)
- train test splt, leave one out k fold (+ begründung mit deep learing ..reliable results etc, randomness)
- kurz erklären wieso weniger züge besser sind bei dieser hci erkennung

- (vielleicht kram zu adaptive learning rate ändern und gucken wie es so ist, ansonsten begründen wieso ich das nicht brauche)
- 20 steps und 40 steps für beide trainieren mit sd0x bis sd10x plus sd20x
- (entweder so dass 1d cnn mit 20 steps auch konvergence erreicht oder danach nochmal anpassung von 20 steps 1d cnn damit es kovergiert)
- darüber schreiben dass letzen n züge nicht mehr so gut simuliert sind (zeigen) und deshalb 40 steps nicht so viel besser ist
- mit maximalen guten steps (glaube 16 züge oder so) trainieren und vergleichen
- statistical tests um signifikante unterschiede /keine unterschiede zu zeigen für verschiedene modelle etc (siehe mazens nachricht)
- letztlich war es wert 2D cnn auszupribieren. Werden selten für sowas verwendet haben in diesem fall aber signifank besser ergebnisse erzielt.
- dfalls tatsächlich btraining mit initialen daten besser ist als mit optimierten: therie könnte sein, dass durch verringerung des randomness, das netz nicht mehr so robust ist wie vorher und bei test auf den echten daten schlechter abschneidet

8. Prospect

- es wurde zwar impolementiert nicht alle features zu verwenden, aber es wurde nie ausprobiert. Hätte man testweise machn können.
-Vielleicht hätte man ein weiteres statistical feature für penalties machen können, so wie es beim qualitätscheck der simulationnen berechnet wird.

Bibliography

[Rab89] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.