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.
-

- relevance of topic - glare detectin has been done before with for example a light detector but not with behavioural data (volatile?) -

Neural networks greatly benefit from lots of data. As the number of participants that provided data is very limited, 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.

- gereal approach, simulation, matrix creation, often used 1d cnn, novel approach of creating synthetic images and using 2d cnns shows in many cases improved classiiction resuts. Finally four voting based system one each for 5, 10, 15 and 20 rounds are implemented, combining the prediction of 400 models each to decide the final predictions.

- 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...

- Einleitung: Was es für verschiedene Obstacles gibt, was schon gemacht wurde, was ich mache,

- one dimensional convolurional neural netweotks refered to as 1d convnets

# 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 [Mir18] 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 [Ihr19]. 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 figure 2.1. 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 scenario in which a light source shines onto the display, reflects from it and makes it more difficult to differentiate the colours of the cards. This effect can be seen in figure 2.2.

To help in case of interaction obstacles the game provides multiple ways of assistance. Although they are not relevant for this thesis as only data is used where subjects 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 where also other assistances implemented such as increasing the size of cards if they are revealed, 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 2.1 and 2.2 show screenshots in the two modes.

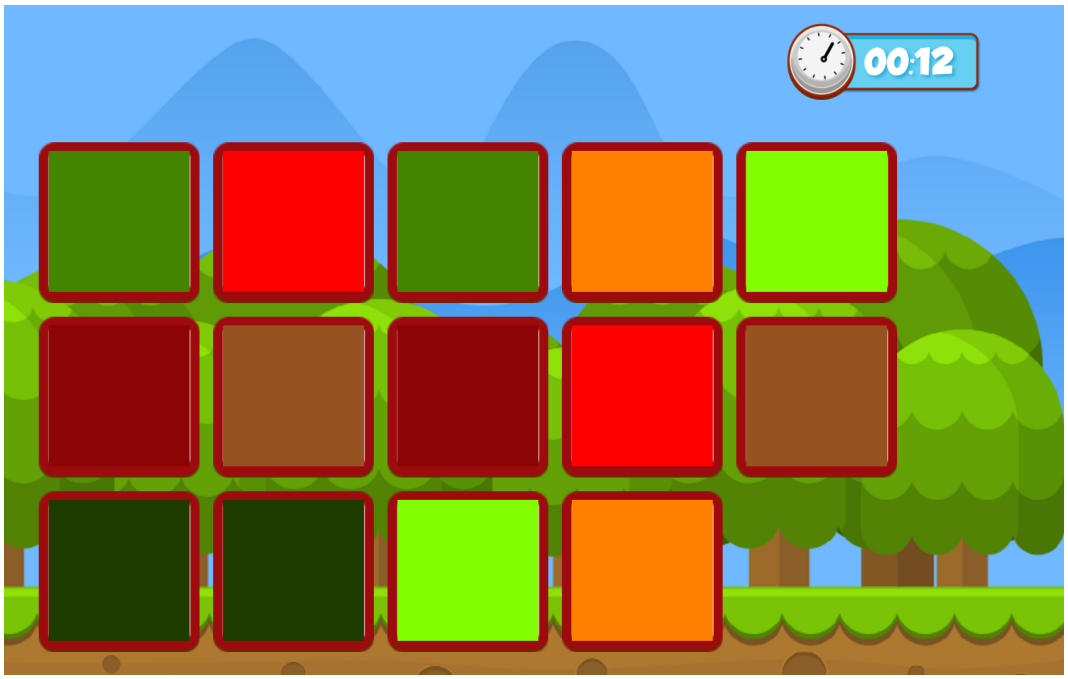


Figure 2.1: Screenshot of the game without obstacles. All cards are turned face up.



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. CMM-based Cognitive User Simulation

The cognitive system lab developed the computer program cognitive memory model (CMM). It is a general computational cognitive model of human memory inspired by the ACT-R theory [SP18a]. ACT-R stands for adaptive control of thought-rational and is a cognitive architecture that aims to explain how the components of the mind work to produce coherent cognition [ABB+04]. 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 [SP18b]. Furthermore it is possible to define a similarity matrix to simulate similarities between cards in matching pairs games. This matrix includes similarity values between 0 and 1 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 [SPIS19]. For this thesis the only two modes that are relevant are the glare effect and the no obstacle modes, both without any assistance. In order to simulate no obstacle games no similarity matrix is needed, but for glare effect games a new similarity matrix is created in section 5.1 Similarity matrix creation.

To accurately simulate the performance of human memory multiple parameters are randomly initialized and then repeatedly optimized using a genetic optimization algorithm. These parameters can be further extended. The genetic algorithm works by repeatedly 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 performance 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 simulated games best fits that in the

real game. As example one of the parameters optimized 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 continually reduced as there are less cards in the game after pairs were discovered [SPIS19].

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 chapter 4 Collected Data. In the context of simulating no obstacle games changes to the simulator were made that are explained in section 5.6.1 Results before and after changing the simulator.

# 4. Collected Data

While the subjects played the memory game, behavioural 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 eeg data is not used, because the aim is to exploit behavioural patterns of natural HCI situations without using any additional sensors. Therefore eeg data will not be further explained. The data was collected from 22 subjects. **TODO: grobe demographic herausfinden, mazen fragen**. Each subject 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. The data was already provided and was not collected during this bachelor thesis.

In order to record behavioural data, each of the 14 cards was initially 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 colour. Once a game is started, all cards are shuffled. The initial assignment of colours to numbers between 1 and 7 is shown in table 4.1.

| Colour | Number |
|---|---|
| Dark green | 1 |
| Brown | 2 |
| Red | 3 |
| Light green | 4 |
| Green | 5 |
| Orange | 6 |
| Dark red | 7 |

Table 4.1: Static assignment of colours to numbers

The behavioural data was saved in logs, consisting of the card codes of each round followed by the 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. The timestamps are irrelevant for this work and therefore not shown.

With the assignments shown in table 4.1, the first card code in the first sequence on the right means that the first card to be flipped was the second green card. In the same turn, the second card that was turned around was the second brown card. This mapping is static as colours have the same numbers across all recorded games. However, the simulator requires a dynamic mapping of the card codes. 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.

Static mapping:
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

Dynamic mapping:
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

In this dynamic mapping of the card codes, each entry likewise consists of 2 numbers that are separated 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 of that colour. In the example above the colour green is the first to be revealed and therefore assigned to the value one. As it is the first green card revealed the second component of the card code is 1 as well. As a result the first entry is 1.1. Then a brown card is turned face up, meaning the second entry is 2.1 since is was the first brown card. Once the other red card is discovered, the entry for that step 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. These card codes are the foundation for the features calculated in section 5.7 Feature generation.

Logs with remapped card codes were already provided, but an additional component in the glare effect logs is needed in order to successfully simulate new games. The logs are used in the simulator to generate new games. When simulation the glare effect, the simulator requires a similarity matrix, that that correctly describes the colour differences of the cards under the influence of the glare effect, which does not exist, yet. The creation of this matrix is done in section 5.1 Similarity matrix creation and its values are added to the glare effect logs in section 5.2 Incorporation of the new similarity matrix. Contrary to the glare effect logs, in the no obstacle logs, no changes need to be made, as the simulator does not use any similarity values when simulating no obstacle games. It should also be noted that the provided logs with remapped card codes 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 ignored, as they are newly calculated for every turn as explained in section 5.7.1 1D CNN features. Last but not least all logs end with a label, describing in which game mode the log was created.

# 5. Data Preparation

To prepare the data for the training many steps were necessary. One big step is to increase the available data by simulating user behaviour. Therefore the simulator explained in chapter 3 CMM-based Cognitive User Simulation is used. For simulating games with no obstacle there is no need for a similarity matrix. However, in order to successfully simulate glare effect games, a special similarity matrix is required. Therefore the first step is to create a similarity matrix that describes the differences of colours under the influence of the glare effect. Once completed, it is added to the glare effect logs. As not all logs are valid and can be used in the simulator, the invalid ones must be removed. The valid logs are then used to simulate user behaviour and thereby create new logs. These simulated logs are sorted by their quality and the simulation is evaluated. The evaluation inspired some changes to the simulator, after which the games are simulated again. The quality of the simulations before and after the changes to the simulator are analysed. Lastly, to prepare the data for training, the features for the 1d convnet and the 2d convnet are created.

## 5.1 Similarity matrix creation

### 5.1.1 Conceptualization

The aim is to create a similarity matrix that describes the differences 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. Additionally there are wide lines of higher intensity coming from the brightest area, that simulate sunbeams. This influence of the simulated sunlight can be seen in figure 2.2.



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.

As a result simply extracting the RGB values for one pixel of each card in one game and calculating the differences 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 differences are only representative for exactly that game and may be completely different if the cards are positioned differently. Secondly, extracting single pixels for each card in an image could lead to colour values that do not represent the overall observed colour due to varying brightnesses on different points of a card. This behaviour is undesired, as the final similarity matrix is supposed to represent the differences 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 one 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 colour differences of those matrices. The idea is that by using enough games so that all or most combinations of positions and colours are included and calculating the mean of all comparisons for each combination of colours, 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 different 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.

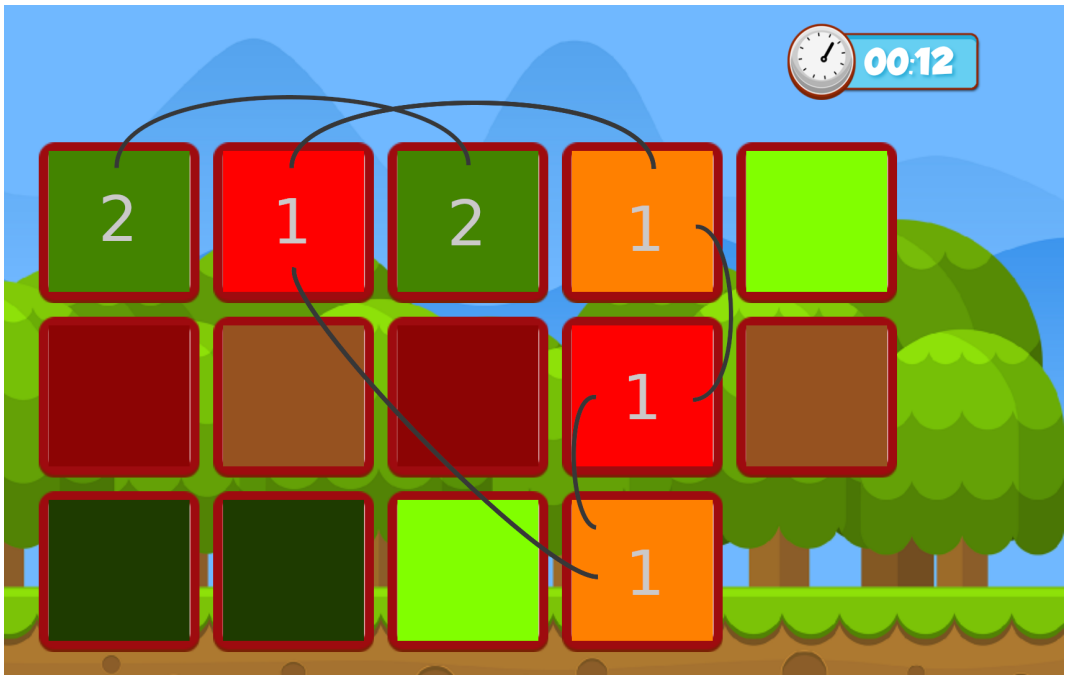


Figure 5.2: Exemplary showing the number of different comparisons for case 1 and 2. The numbers on the cards represent the case and the edges indicate unique comparisons. All cards are turned face up. A game without any obstacles is used only for the purpose of clarifying the different comparisons. All 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 comparisons 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 probability 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 necessary screenshots to include a specific combination with 95% probability 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 comparisons would include a comparison of the first green card and the second green card as well as a comparison 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 comparisons 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 probability 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 necessary screenshots to include a specific combination with 95% probability 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 comparisons 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 differences are calculated. To capture how colour differences are perceived by humans, the RGB colour space is not suitable, as it is perceptually nonuniform. A colour space is perceptually uniform if the measured colour differences are proportional to the human perception of such differences. Contrary to the RGB colour space, the that the L*a*b* colour space is perceptually uniform [Pas01]. Therefore the colour differences described in the similarity matrix are calculated after converting the colours into the L*a*b* colour space. In related work, when creating the similarity matrix for the effect of colour blindness, the CIE1976 colour model was used for the calculations [SPIS19] of the colour differences. However, the formula has since then been improved multiple times, resulting in the CIE2000 colour model. Through multiple modifications of the formula the calculated colour differences became closer to how they are perceived by humans [Fis02]. Therefore when creating the similarity matrix for the glare effect the CIE2000 colour model is used instead of the CIE1976 colour model. The resulting colour distances are referred to as Delta E values [Fis02]. The lower the Delta E value, the more similarly the colours are perceived by humans.

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 areas into L*a*b* colours, calculating the Delta E values for each combination of colour to create a similarity matrix for each screenshot and finally averaging the values in the 300 similarity matrices. As there are cards with seven different colours in the game, the similarity matrix has 28 entries. To achieve values between 0 and 1, the colour differences are additionally 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 differences 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 influence 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.

```
def determine_glare_rgb_values(image):
'''
Calculates the 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: Code for calculating the average rgb values of each card in an image.

The cards are each $250 \cdot 250$ pixels big and the RGB values are extracted from squared $150 \cdot 150$ pixel areas (given an emulator with 1920 x 1080 resolution is used). 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 necessary to calculate the colour difference using the CIE2000 colour model. Each of the 28 cells in the similarity matrix falls into one of the two cases explained in section 5.1.1 Conceptualization, 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 difference. The last step necessary to complete the final similarity matrix is to divide all values in the matrix by the highest occurred colour difference in order to scale them down to values between 0 and 1.

### 5.1.4 Testing

To verify the correctness of the colour extraction and the calculation of the similarity matrix, the main functionalities are manually tested. The colour extraction was tested by extracting colours of a screenshot with the script and comparing the RGB values to the actual values in the images using gimp. Furthermore the calculation of the Delta E score was tested, by comparing 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 is 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 separately be determined. The similarity matrix for glare effect has 28 entries, from which 7 are comparisons between same and 21 between different colours. With $C_1$ and $C_2$ being the possible combinations for two arbitrary 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. The final test of the created similarity matrix, however, will be to actually use it to simulation glare effect games and see whether they are simulated accurately. This is done in section 5.6 Evaluation of simulation.

### 5.1.5 Final Matrix

In figure 5.3 the final similarity matrix can be seen, displaying how colour differences 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 occurred during the calculation was 8.861. All values are divided by that number to achieve scores between 0 and 1.

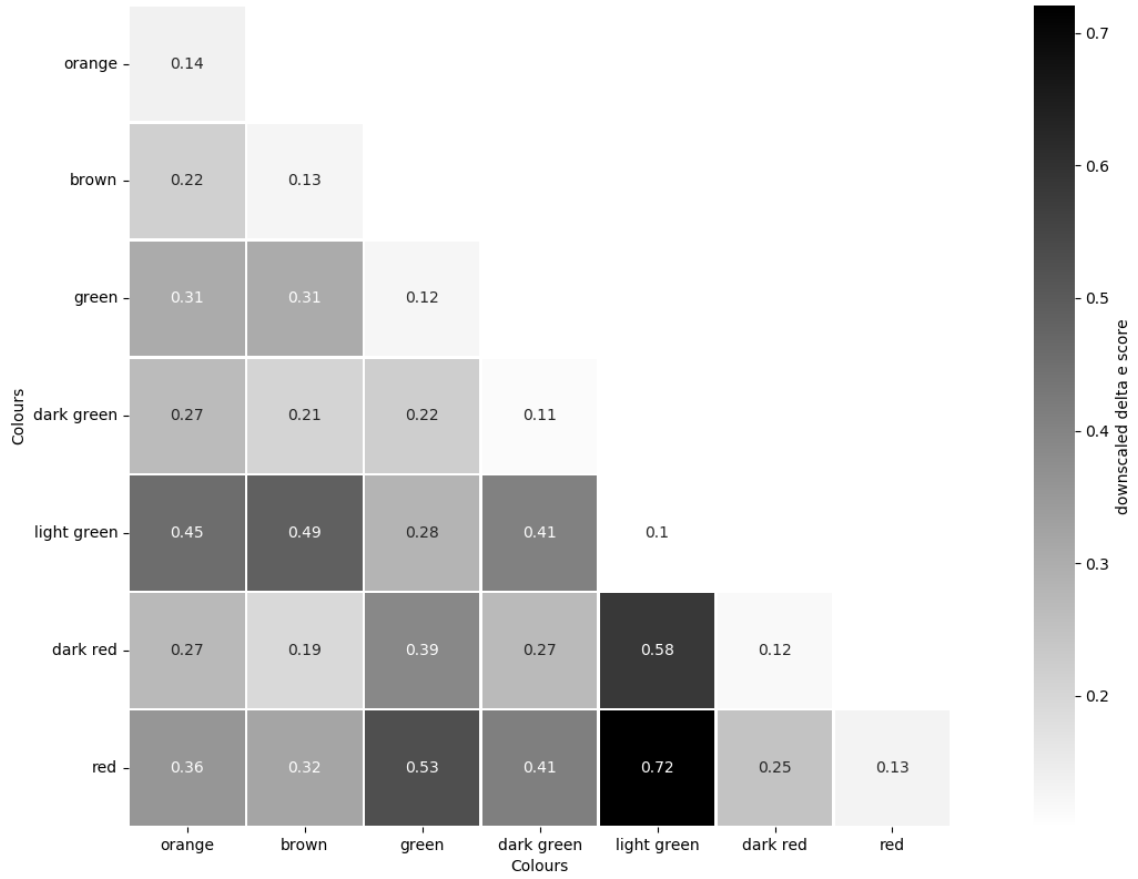


Figure 5.3: The final similarity matrix containing similarity values for each combination of coloured card on the field. The lighter the colour used for the representation the less a difference is noticeable

The delta E score is less a definitive answer, and instead a helpful metric to apply to a specific use case. Although there are tendencies regarding the interpretation of colour differences, there is no definitive table that describes what the different scores mean **TODO: source**. One reason for this is that the perceived colour difference 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 **TODO: source**. 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 5.1 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 clarifying how the values in the matrix can roughly be interpreted.

| Delta E | Downscaled values | Perception of difference |
|---|---|---|
| $\leq 1.33$ | $\leq 0.15$ | 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 |
| $\geq 4.43$ | $\geq 0.5$ | Major colour difference |

Table 5.1: How the similarity can roughly be interpreted. Created through own observation.

**TODO: vielleicht tabelle und erklärung darüber raus nehmen weil ich keine quelle dafür habe**

Before actually using the constructed similarity 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 approaches.

It should be also noted that the similarity matrix has 28 entries. The one that was used for describing the confusion that happens during colour blindness only has 21 entries, because all 7 values describing cards of the same colour can be left out as the colours are equal. Since in glare effect games, originally equally coloured cards can look different due to the influence of the simulated sunlight, these 7 values must be included in the glare effect similarity matrix.

## 5.2 Incorporation of the new similarity matrix

The newly created similarity matrix can now be added to the glare effect logs. Therefore each of the 28 similarity values in the matrix is assigned to a dynamically mapped card code. To do so there was already a script available, that was used after some small adjustments. The main changes were to replace the deprecated functions from libraries with supported ones and incorporate the new similarity matrix. As the code was initially written for similarity matrices with 21 entries and the one for glare effect has 28 entries, the code needed further adjustments. After all changes were completed, the new glare effect logs with remapped logs were saved in new files. To check whether the similarity assignments are still done correctly, the resulting logs were compared to logs that were created by the original script. Besides different similarity values due to different 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 and balancing of the data set

For the simulation 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 balance the data set for training. Data imbalance can force the classification algorithm to be biased towards the majority class. This can lead to concepts of the minority class not being learned adequately [WLW+16]. Therefore equal numbers of games from each of the two game modes are included. Furthermore the decision was made that the number of games from each participant in each game mode should be equal, too. This means

that the data should not contain more no obstacle games from a participant than glare effect games, and vice versa.

As stated in chapter 4 Collected Data 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.

## 5.4 Simulation of user behaviour

The whole process of simulating user behaviour is divided into two parts: First configuration files are created, using a generic optimization algorithm, that contain the optimized parameter values and secondly these configuration files are used to simulate games. How the simulator functions is explained in chapter 3 CMM-based Cognitive User Simulation.

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 and the other two for using those files to simulate new games based on the user behaviour in the original games. These classes utilize the functionalities already implemented in the simulator. 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 adjusted 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 obstacle and 1000 glare effect logs for each of the 20 subjects.

## 5.5 Sorting logs by quality

The simulated logs ware sorted by how close their performance is to that in the real game used for their simulation. The value by which is sorted, is the average of two values. The first one is the root mean squared error (RMSE) between the matching pairs in the real and the simulated game for each round. The second one is the RMSE between the penalties in the real and the simulated game for each round. These two performance measurements are explained in section 5.6 Evaluation of simulation. The formula for calculating the RMSE is shown below.

$$RMSE = \sqrt{(\frac{1}{n})\sum_{i=1}^{n}(y_i - x_i)^2}$$

$n$ : number of rounds
$y_\mathrm{i}$ : performance value in real game
$x_\mathrm{i}$ : performance value in simulated game

Sorting the logs accord to their quality makes it possible to only use s certain number of the best simulated logs during the training instead of all of them or a random subset.

## 5.6 Evaluation of simulation

To evaluate whether the performance in the simulated logs is similar to the one in the original logs two performance measurements are used: The matching pairs in each round and the penalties in each round. Penalties are given if a card is revealed whose partner has already been seen before but the pair was not picked up. Some initial observations inspired changes to the simulator. The quality of the simulations regarding the performance measurements is analysed before and after the changes to the simulator, by comparing the performance between simulations and real games. Furthermore a subset of the simulated data was discovered that mostly shows no significant difference between the performance of the simulated and the real games.

### 5.6.1 Results before and after changing the simulator

The Initial simulation results can be seen in figure 5.4 and 5.5. The horizontal line extending from the curves, also called whiskers, describe the standard deviation of the corresponding value.

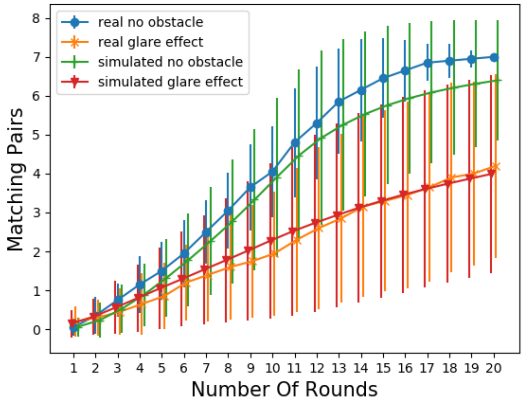


Figure 5.4: Comparison of matching pairs per round of simulated and real glare effect an no obstacle games. Before the changes to the simulator.

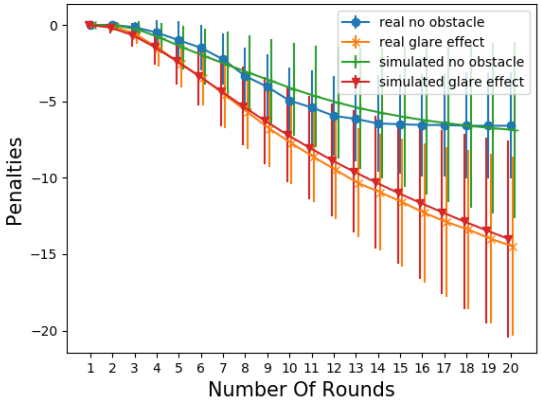


Figure 5.5: Comparison of penalties per round of simulated and real glare effect an no obstacle games. Before the changes to the simulator.

At first glance it can be seen that the simulations of glare effect games are very similar to the real games, meaning that the constructed similarity matrix creates good simulation results. As a result there is no need to find a different approach for creating the similarity matrix, and no changes to the simulator regarding the simulations of glare effect games are made. However, the simulation of no obstacle games is not quite as accurate. Especially the number of matching pairs after the tenth round in the simulated games is lower than in the real games. Additionally the standard deviation indicated by the whiskers regarding the no obstacle games is noticeably higher in simulated than in the real games. This is not very string in the first turns but gets worse in later turns. Furthermore, however less noticeable, is are the penalties in the simulated games between round 8 and 18 lower than in the real games.

The fact that matching pairs per round for the no obstacle simulations are noticeably lower than in the real games inspired some changes to the simulator. As there are less cards on the field in later rounds and the experience of the user may have improved over the course of the game, in reality the degree of randomness is likely decreased in later turns. To simulate this a new parameter was introduced that is optimized by the generic optimization algorithm. It is called RANDOMIZING DECAY and is a value between 0 and 1. After the 10th round this value is used to reduce the amount of randomness in the choice of the next card and by that increases the performance in those turns. Another value that also influences the random behaviour in the simulations was optimized so that there is overall less randomness. The results of these changes can be seen in figure 5.6 and 5.7.

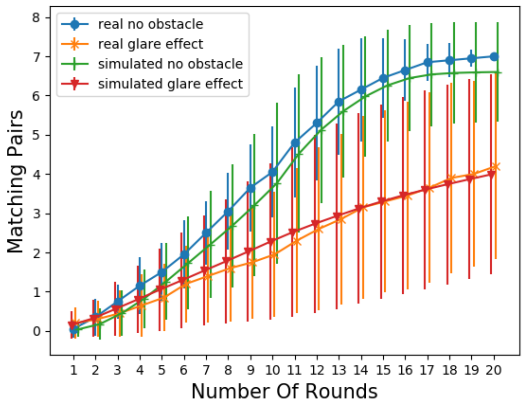


Figure 5.6: Comparison of matching pairs per round of simulated and real glare effect an no obstacle games. After the changes to the simulator.

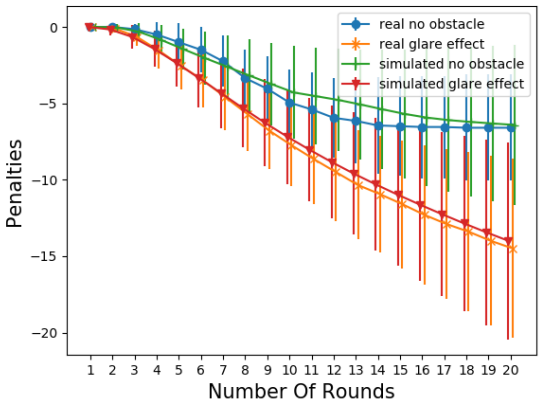


Figure 5.7: Comparison of penalties per round of simulated and real glare effect an no obstacle games. After the changes to the simulator.

As the changes only affected the simulation of no obstacle games, the following statements only refer to those games. It is noticeable that the number of matching pairs per round in the simulated games is closer to the real games. Additionally the standard deviation of the simulated games regarding the matching pairs per round gets noticeably smaller after the 10th round and by that is closer to that in the real games. However, the reduction of randomness also resulted in slightly less penalties after round 10 and by that the difference in penalties between the simulations and the real games becomes bigger. On the contrary, the standard deviation of the penalties in the last 6 rounds of the simulated no obstacle games got slightly closer to that in the real games. Reasons for the negative impact on the penalties are unknown. It seems that the improvements regarding the matching pairs overweight the deteriorations of the penalties, but if these changes actually improve the overall quality of the simulations can not be said without further analysis.

However, another interesting observation can be made. The standard deviations of the two performance measurements in the last rounds of the simulated no obstacle games is notably higher than in the real games. This might be caused by the fact that there are in general less or even no cards on the field if all pairs were discovered. This theory also fits to the fact that this is not an issue in the glare effect games, as there are more cards on the field in later turns. This concludes, that there is a

possibility that the simulator struggles to accurately simulate rounds once there only a few or no cards remain on the field.

Although major tendencies of similarities and differences are visually noticeable in figure 5.4, 5.5, 5.6 and 5.7, a more accurate analysis is needed to to make reliable statements. Therefore multiple paired samples t-test are performed, comparing real no obstacle, simulated no obstacle, real glare effect and simulated glare effect games with each other. For each combination two tests are performed. One comparing the mean penalties per round and the other one comparing the mean numbers of matching pairs per round. The p value expresses whether the compared lists of mean values are significantly different or not. Values higher than 0.05 indicate no significant difference, while values below 0.05 indicate a significant difference. It would be optimal if all comparisons between different game mode show significant difference, while comparisons between real and simulated games of the same type show no significant difference. All paired sample t-tests were performed before and after the changes to the simulator. This was done to see whether the changes to the simulator improved or deteriorated the results of the test or if they had any impact on them at all. First the results of the paired sample t-tests, when including all 20 rounds and all simulated games, are analysed. Tables 5.2 and 5.3 show the results before changing the simulator, while table 5.4 and 5.5 show the results afterwards.

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 | |
|---|---|---|---|---|---|---|
| | p | t | p | t | p | t |
| r_g | $1.7e-06$ | $-6.8179$ | $0.0688$ | $-1.9292$ | $9.3e-07$ | $7.1041$ |
| r_n | $5.7e-07$ | $7.3571$ | $2.7e-06$ | $6.5765$ | | |
| s_g | $5.3e-06$ | $6.2480$ | | | | |

Table 5.2: p and t values in paired sample t-tests for different comparisons of matching pairs per round before the changes to the simulator. All 1000 simulated games per real game were used.

| | s_n | | s_g | | r_n | |
|---|---|---|---|---|---|---|
| | p | t | p | t | p | t |
| r_g | $3.6e-06$ | $-6.4297$ | $1.1e-06$ | $-7.0423$ | $3.2e-06$ | $6.4897$ |
| r_n | $0.2011$ | $-1.3242$ | $5.6e-06$ | $6.2238$ | | |
| s_g | $5.4e-06$ | $6.2394$ | | | | |

Table 5.3: p ant t values in paired sample t-tests for different comparisons of penalties per round before the changes to the simulator. All 1000 simulated games per real game were used.

| | s_n | | s_g | | r_n | |
|---|---|---|---|---|---|---|
| | p | t | p | t | p | t |
| r_g | $5.7e-06$ | $-6.2181$ | $0.0688$ | $-1.9292$ | $9.3e-07$ | $7.1041$ |
| r_n | $1.8e-10$ | $12.2469$ | $2.7e-06$ | $6.5765$ | | |
| s_g | $1.7e-05$ | $5.7107$ | | | | |

Table 5.4: p and t values in paired sample t-tests for different comparisons of matching pairs per round after the changed to the simulator. All 1000 simulated games per real game were used.

| | s_n | | s_g | | r_n | |
|---|---|---|---|---|---|---|
| | p | t | p | t | p | t |
| r_g | $4.8e-06$ | $-6.2967$ | $1.1e-06$ | $-7.0423$ | $3.2e-06$ | $6.4897$ |
| r_n | $0.0195$ | $-2.5519$ | $5.6e-06$ | $6.2238$ | | |
| s_g | $7.2e-06$ | $6.1045$ | | | | |

Table 5.5: p ant t values in paired sample t-tests for different comparisons of penalties per round after the changes to the simulator. All 1000 simulated games per real game were used.

It can be seen that all tests comparing different game mods show significant difference in penalties per round as well as matching pairs per round, which is desired. However significant difference is not desired in the four test comparing real and simulated games of the same mode, which are highlighted by colours. Before the changes to the simulator, the matching pairs per round in simulated and real no obstacle games show significant differences. The same goes for the penalties per round in simulated and real glare effect games. As the changes to the simulator only affected the simulation of no obstacle games the significant difference between the matching pairs in simulated and real glare effect games prevails. Despite the fact that the matching pairs per round in simulated no obstacle games seem closer to the real games in figure 5.6, the statistical tests still show a significant difference between them. Additionally, the small deterioration of the penalties that were caused by the changes to the simulator and can be observed in figure 5.7, led to a significant difference in the penalties per round between simulated and real no obstacle games. Before the changes to the simulator this difference was not significant.

In total, before the changes to the simulator 2 out of 4 tests comparing games of the same mode, result in undesired significant differences. After the changes to the simulator this increased to 3 out of 4 tests. However, this does not lead to the conclusion that the changes to the simulator decreased the quality of the simulations, as the matching pairs per round shown in figure 5.6 in simulated no obstacle game got noticeably more accurate.

### 5.6.2 Using a subset of the data

In the statictical tests in section 5.6.1 Results before and after changing the simulator, all all 1000 simulations per participant are used. As some simulations are better than others, using a subset only containing a number of best simulations could lead to further findings. Since the simulated games are sorted by their quality in section 5.5 Sorting logs by quality, it is possible to only use a certain number of best simulations for the comparisons, and see whether this improves the undesired properties mentioned above. Furthermore the number of rounds in the comparison can be changed, so that not all 20 rounds are included. The consideration of including less rounds is of importance, because it can reveal which parts of the game are simulated accurately and which are not. The search for an configuration of those two settings was manually done by varying the number of steps and the ratio between simulated and real data. A configuration of those two settings that produces interesting results consisted of using only the best 15 simulated games per game mode from each participant and only including the first 10 rounds of each game. The results before the changes to the simulator can be seen in table 5.6 and 5.7, while the results afterwards are shown in table 5.8 and 5.9. The two performance measurements in the first 10 rounds after the changes to the simulator when only using the best 15 simulated games per game mode from each participant are additionally portrayed in figure 5.8 and 5.9

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 | |
|---|---|---|---|---|---|---|
| | p | t | p | t | p | t |
| r_g | 0.0087 | −3.3342 | $6.4e-05$ | 6.9956 | 0.0058 | 3.5985 |
| r_n | 0.0032 | 3.9837 | 0.0037 | 3.8893 | | |
| s_g | 0.0056 | 3.6218 | | | | |

Table 5.6: p and t values in paired sample t-tests for different comparisons of matching pairs per round before the changes to the simulator. Only the best 15 simulated logs from each real log an the first 10 rounds are used. are used.

| | s_n | | s_g | | r_n | |
|---|---|---|---|---|---|---|
| | p | t | p | t | p | t |
| r_g | 0.0025 | −4.1496 | 0.1741 | 1.4756 | 0.0014 | 4.5577 |
| r_n | 0.6650 | 0.4476 | 0.0012 | 4.6627 | | |
| s_g | 0.0021 | 4.2596 | | | | |

Table 5.7: p values in paired t-test for different comparisons of penalties per round before the changes to the simulator. Only the best 15 simulated logs from each real log an the first 10 rounds are used. are used.

| | s_n | | s_g | | r_n | |
|---|---|---|---|---|---|---|
| | p | t | p | t | p | t |
| r_g | 0.0064 | −3.5291 | $6.4e-05$ | 6.9956 | 0.0058 | 3.5985 |
| r_n | 0.0763 | 3.8893 | 0.0037 | 3.8893 | | |
| s_g | 0.0041 | 3.8201 | | | | |

Table 5.8: p and t values in paired t-test for different comparisons of matching pairs per round after the changes to the simulator. Only the best 15 simulated logs from each real log an the first 10 rounds are used. are used.

| | s_n | | s_g | | r_n | |
|---|---|---|---|---|---|---|
| | p | t | p | t | p | t |
| r_g | 0.0019 | −4.3207 | 0.1741 | 1.4756 | 0.0014 | 4.5577 |
| r_n | 0.8388 | −0.2095 | 0.0012 | 4.6627 | | |
| s_g | 0.0016 | 4.4302 | | | | |

Table 5.9: p values in paired t-test for different comparisons of penalties per round after the changes to the simulator. Only the best 15 simulated logs from each real log an the first 10 rounds are used. are used.

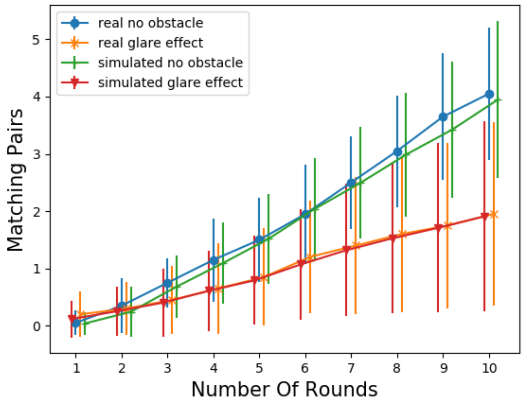


Figure 5.8: Comparison of matching pairs per round of simulated and real glare effect and no obstacle games in the first 10 rounds. Only the best 15 simulations per real game were used. After the changes to the simulator.

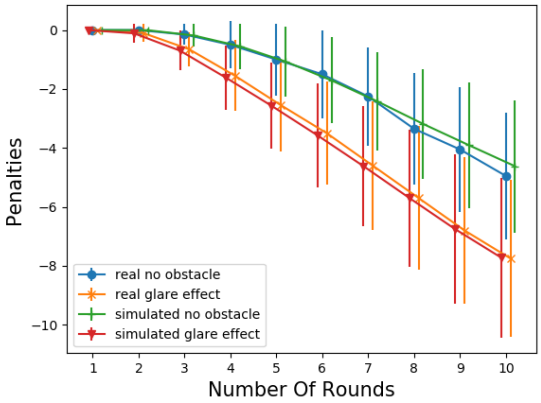


Figure 5.9: Comparison of penalties per round of simulated and real glare effect and no obstacle games in the first 10 rounds. Only the best 15 simulations per real game were used. After the changes to the simulator.

Interestingly, including all the simulated data in the tests, results in less undesired result before the changes to the simulator compared to afterwards, but using only a small subset of the data results in the opposite. Before the changes to the simulator, when including only the best 15 games and the first 10 rounds in the tests, only the two 2 tests in table 5.7 for penalties comparing real and simulated games of the same mode show no significant difference. The two tests in table 5.6 comparing matching pairs of real and simulated games of the same mode show a significant difference. Contrary to that, after the changes to the simulator, the test in table 5.8 for matching pairs in real and simulated no obstacle games shows no significant difference, meaning more tests have the desired outcome.

As all but one test produce desired results after the changes to the simulator, it was additionally searched for a configuration that only result in desired outcomes of the paired sample t-tests.The only undesired result after the changes and using the specified subset of the data is highlighted in red in table 5.8. It expresses a significant difference in matching pairs between real and simulated glare effect games. However, a configuration that showed no significant difference in all tests were it is desired was not found. To achieve non significant difference between matching pairs in real and simulated glare effect games, a noticeably larger subset of the data had to be included, which resulted in a signifiant difference in matching pairs between real and simulated no obstacle games. Nonetheless it should be pointed out that this does not mean that the simulation of the glare effect games is bad regarding the number of matching pairs per round. If the two lists of mean values for matching that are significantly different according to the paired sample t-test are manually compared, it can be seen that the difference is very small, with the highest difference being 0.093 in round 6. These values can be seen in table 5.10.

| | r. 1 | r. 2 | r. 3 | r. 4 | r. 5 | r. 6 | r. 7 | r. 8 | r. 9 | r. 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| real | 0.2 | 0.3 | 0.45 | 0.65 | 0.85 | 1.2 | 1.4 | 1.6 | 1.75 | 1.95 |
| simulated | 0.117 | 0.253 | 0.403 | 0.61 | 0.793 | 1.107 | 1.32 | 1.527 | 1.71 | 1.91 |

Table 5.10: Mean values of matching pairs for real and simulated glare effect games for the first 10 rounds. Only the 15 best simulations per real game are used.

### 5.6.3 Conclusion of the evaluation

Firstly it can be said using the created similarity matrix for the glare effect simulations produces games with performance close to the real games. Therefore the aim of creating a similarity matrix that describes the colour differences of cards on the filed under the influence of the glare effect was achieved.

Regarding whether the attempt to improve the simulation of no obstacle games by doing changes to the simulator was successful, no clear statement can be made. When all simulated data is used, the paired sample t-tests indicate, that the quality of the simulations may have gotten worse, while when using only the best 15 simulations and only the first 10 rounds, the tests indicate the opposite.

Although no subset of the data was found in which all simulations have no significant difference to the according real games regarding the paired sample t-test, it can be said that the simulator is capable to simulate the first 10 rounds very accurately, given only the 15 best simulations are used. The statistical tests and the previous considerations regarding the too large standard deviation in later turns indicate, that the simulator performs less good in later turns. Having said that, these are only indications that would need further research to be confirmed. Doing so might lead to improving the simulator.

This analysis also indicates, that it might not ideal to use all simulated data for training as that results in a significant difference between the simulated and the real games according to the paired sample t-test. Using to many simulations compared to real games could decrease the performance of the classifiers on real data if they adapt too much to simulated games. Furthermore is should be analysed how using different amounts of turns in the classification impacts the classification results. In section 8.3 Training results and evaluation different configurations regarding ratios of simulated to real games and the number of rounds includes, are tested.

Nonetheless it should be emphasized that the aim of no significant difference between real and simulated games is set very high. As shown in table 5.10, the difference can be very little and the paired sample t-test still concludes a significant difference. Overall the simulations and real games are very similar when comparing them with the two performance measurements. If only the best simulations are used the performance becomes even more similar.

## 5.7 Feature generation

### 5.7.1 1D CNN features

For the 1D CNN five statistical 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 original game logs as well as the ones simulated in section 5.4 Simulation of user behaviour. Their structure is explained in chapter 4 Collected Data. The calculated features can be direclty fed into the 1d CNN explained in section 8.1.1 1D CNN.

The code for calculating the statistical features out of game logs was already given. A script was written that incorporates this functionality in order to calculate the features for all logs. Additionally, after creating the necessary 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 every training and instead can be loaded from files. The reason for also saving the raw data even though this work does not need them any more is, that the working group that was collaborated with also trained other models and does not directly load the features but instead calculates them before each training.

### 5.7.2 2D CNN features

For the second approach of using a 2D CNN further steps are taken. Therefore synthetic images in grey scale colours are created using the features mentioned above. As the images are in grey scale colours, only one colour 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 each other. This can be clarified by looking at the code that produces the synthetic images.

```
1  def create_image(game, components=[True, True, True, True, True]):
2    '''
3    Creates synthetic images out of the card codes and the staticial
         features.
4    :param game: The statical features in each step.
5    :param components: Five values describing which of the features
6    should be used to create the image.
7    :return: The synthetic image.
8    '''
9    card_codes = np.zeros((7, steps))
10   cards_left = np.zeros((8, steps))
11   never_revealed_cards = np.zeros((14, steps))
12   max_same_card_reveals = np.zeros((20, steps))
13   rounds_since_done = np.zeros((27, steps))

15   x_position = 0
16   for step in game:
17     card_code = math.floor(step[0])
18     first_or_second = int(round((step[0] % 1) * 10))
19     if card_code != 0:
20       card_codes[card_code - 1][x_position] = first_or_second
21     pairs_left[int(step[1] / 2)][x_position] = 1
22     never_revealed_cards[int(step[2])][x_position] = 1
23     max_same_card_reveals[int(step[3])][x_position] = 1
24     rounds_since_done[int(step[4])][x_position] = 1
25     x_position += 1

27   image = np.zeros((0, steps))
28   if components[0]:
29     image = np.vstack((image, card_codes))
30   if components[1]:
31     image = np.vstack((image, max_same_card_reveals))
32   if components[2]:
33     image = np.vstack((image, rounds_since_done))
34   if components[3]:
35     image = np.vstack((image, cards_left))
36   if components[4]:
37     image = np.vstack((image, never_revealed_cards))

39   return image
```

Listing 5.2: Code for creating a synthetic image from the card codes and the statistical features.

By using pseudo colors, the images can be displayed with colours, like in figure 5.10. These $76 \cdot 40 \cdot 1$ (height $\cdot$ width $\cdot$ colour channels) images are directly fed to the 2D CNN explained in section 8.1.2 2D CNN. By setting the origin of the image to the lower instead of the upper left corner a more natural looking image is created and can be seen in figure 5.11.

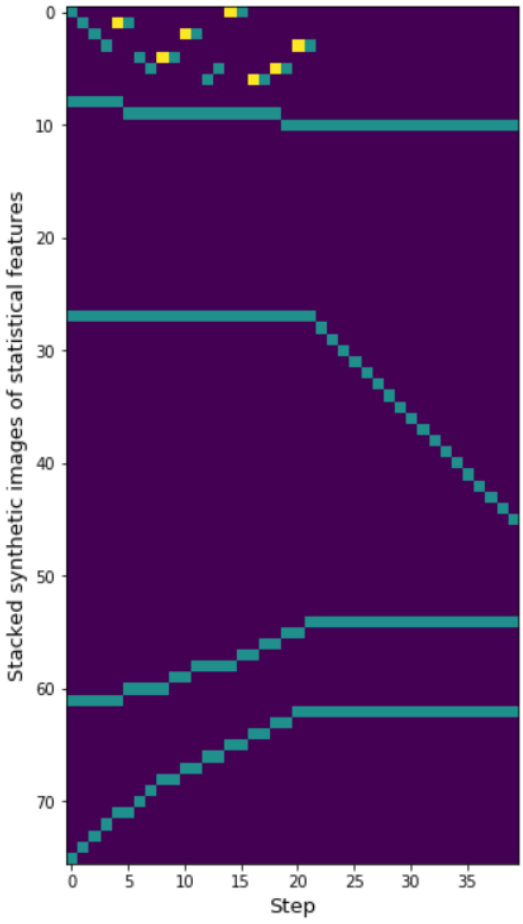


Figure 5.10: Synthetic image created using the card codes and the statistical features. Origin of the image in the top left corner.

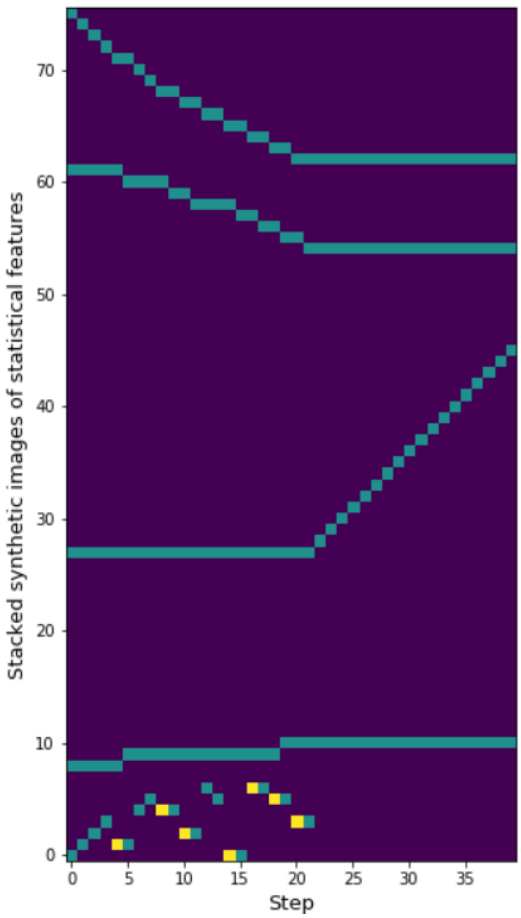


Figure 5.11: Synthetic image created using the card codes and the statistical features. Origin of the image in the bottom left corner.

The width of the image is 40 because that the number of steps recorded. Table 5.11 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 necessary but as little as possible. 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 statistical feature of the number of cards left was converted to the number of pairs left. By dividing by 2 the vertical space necessary to visualize this feature is halved, without information being lost. Instead of representing the numbers left on the field,

the value now represents the number of pairs left. Furthermore 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, meaning that it is impossible to not reveal any card. The stacking order was chosen so that the coloured pixels of different statistical features rarely touch each other.

| 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 |

Table 5.11: Range and vertical space of the different component in the synthetic image.

Regarding the visual representation of he card codes two two decisions were made: Using different colours for the first and the second card of a colour and combining the information about the 14 cards in an vertical area of size 7. For each card code the vertical position of the representing pixel is decided by the first number and the colour is chosen according to the second number. As a result each row contains the information about cards of a specific colour. This representation of the card codes is not chosen arbitrarily. On the contrary, it was inspired by reasons that need explanation.

Instead of combining the information about the 14 cards in an vertical area of size 7, it would have been also possible to use double the vertical space so that the first and the second card of a colour each have separate areas. The first cards and the second cards of a colour would each have individual vertical spaces with sizes of 7. However, it was decided against it. This decision was based on a consideration regarding the structure of the convolutional neural networks used in this work and the concept of the receptive field, explained in section 6.3 Receptive field. How CNNs function is explained in chapter 6 Convolutional neural networks and which models are used in this work is described in section 8.1 Models. In short, the structure of the model influences the sizes of the receptive fields of the features in layers of the network. The receptive field of a feature describes how much of the original input image has influence of the feature. The size of the receptive field is generally bigger the more convolution and pooling layers the network consists of. This also means that if the structure of the cnn is chosen small, the receptive field size of individual features might not be large enough to be influenced by a local area big enough to include far distant pixels of the input image. In this work, the 2d cnn used is constructed with a small number of layers. Therefore it is desirable that pixels in between an important dependency can be assumed are close to each other so that the small cnn is able to learn that dependency. Usually in classical image recognition it is only possible to choose the structure of the models depending on what is to be recognized in the image. However, since in this case the images used are created synthetically, it is also possible to create them while considering the structure of the model. Meaning the images can be created so that they are better suited for the models. The chosen representation of the card codes exploits exactly this freedom. As the dependencies

between cards of the same colours can be estimated to be very important for the classification, the representation is chosen so that if cards of the same colour are flipped in quick succession the pixels describing them are also locally close to each other in the synthetic image. The dependency between such close pixels does not require a large receptive field in order to be learned by the network. Therefore this representation is better suited for the small structure of the cnn that is being utilized.

The main reason that motivated the use of different colours derived from the first decision to combine information about 14 cards in a vertical space of 7. If the same colour was used for all card codes, there would be no visual difference between flipping the same card in consecutive turns and flipping two different cards with the same colour. This of course would also mean that the model could not differentiate between those two cases, although they stem from completely different behaviour. Using different colours for the first and the second card of a colour fixes this issue.

These two decisions resulted in an interesting visual difference between no obstacle and glare effect games that could potentially be beneficial if learned by the model. 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, a noticeably visual difference can be observed.

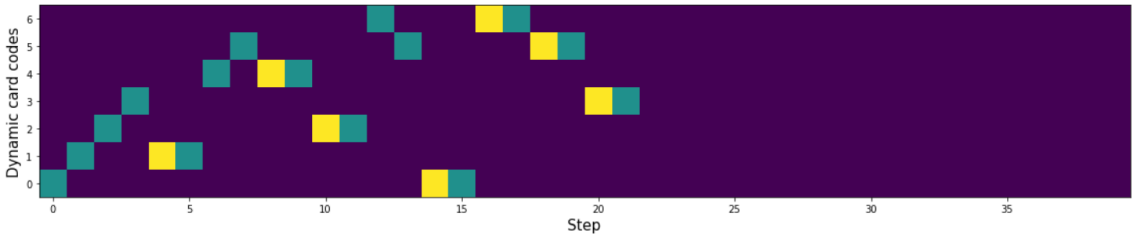


Figure 5.12: Card code section of a synthetic image for a no obstacle game.

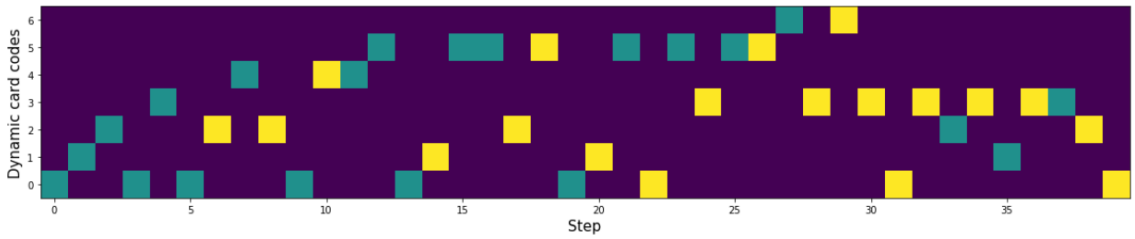


Figure 5.13: Card code section of a synthetic image for a glare effect game.

In 5.12, showing the image for an no obstacle game, there are yellow pixels directly left of green ones, meaning that the subject flipped a card, knew that he had already seen the matching card and directly 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 separations of different coloured cells in the image are more likely in glare effect games than in no obstacle games. Therefore it could be beneficial for the model to learn these characteristics. It should be noted that what the cnn exactly learns or what exact influence the chosen representation has can only be speculated.

Especially the uncertainty in the influence of the different colours shows, that the approach of creating synthetic images is very experimental. Although the way cnns function can be considered when deciding how to create the synthetic image, this does not guarantee good classification results.

# 6. Convolutional neural networks

This chapter introduces convolutional neural networks (CNNs), also known as convnets. CNNs are a type of deep-learning models that can be used for various tasks. While one dimensional convolutional neural networks (1D CNNs) can be used for sequence classifications **TODO: source das buch oder anderes damit diverser**, two dimensional convolutional neural networks (2D CNNs) are almost universally used in the area of computer vision **TODO: source das buch oder anderes damit diverser**. There are other types of CNNs, but this work utilizes only those two. Therefore only those types will be explained. Furthermore only the concepts relevant for this work will be explained. Due to the fact that the concepts of CNNs are more natural in the context of images, 2D CNNs will be used for their explanation. Once the concepts are understood for 2D CNNs they can easily be applied to 1D CNN, as there is little difference between the two types of models. The differences that exist, will be explained.

CNNs consist of multiple components called layers. Which combination of layers to use is highly dependent on the data and its complexity. Same goes for the choice of the so called hyper parameters. These parameters play a major role in how the model is trained and how it will perform. Two important hyper parameters are the number of epochs and the learning rate. While the number of epochs describes how often the whole training data should be passed to the network, the learning rate controls how much the network adapts to the training data in each epoch. How neural networks are trained in explained in chapter 7 Training neural networks. More specifically to CNNs are the parameters for the size of the patches extracted from the inputs and the depth of the output feature map, described in section 6.1.1 Convolution layers. Finding the best hyper parameters to optimize the network can be very cumbersome.

## 6.1 Layers

The convolutional neural networks used in this work consists of the following types of layers: convolutional layers, dense layers, also known as densely connected layers, dropout layers, pooling layers and flatten layers. They will be explained without going to much into detail about the exact calculations performed.

### 6.1.1 Convolution layers

Convolutions operate over three dimensional tensors and are defined by two key parameters: The size of the windows that extract patches from the inputs and the depth of the output feature map. Typically the size of these windows is 3 x 3 or 5 x 5. The depth of the output feature map can also be interpreted as the number of filters computed by the convolution. A convolution is performed by moving these two dimensional windows step by step across the three dimensional input feature map and extracting a three dimensional patch of surrounding features in each step. Each resulting patch has the height and width of the window and the depth of the input feature map. Figure 6.1 exemplary shows the valid locations of 3 x 3 windows in an three three dimensional input feature map with a height and width of 5 and an arbitrary depth.

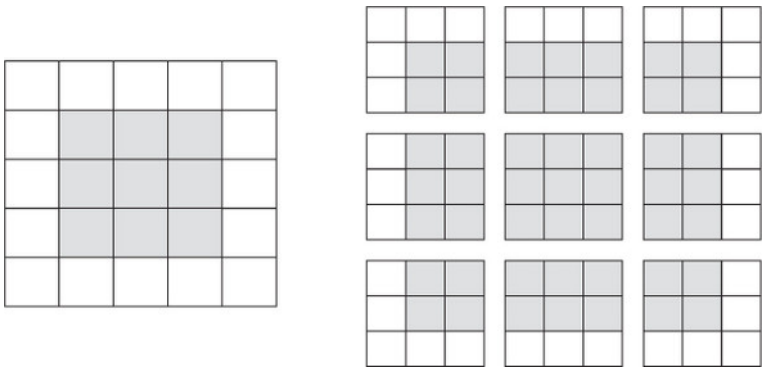

Figure 6.1: Valid locations of 3 x 3 windows in a feature map with arbitrary depth and a height of 5 x 5. Top view fn the feature map. The depth is not shown.

Afterwards, each of these three dimensional patches is transformed into a one dimensional vector that has the specified depth of the output feature map. This transformation is done by calculating a tensor product with a weight matrix, called the convolutional kernel. The kernel has the same dimensions as the window that extracts the patches. The weights of the convolutional kernel are randomly initialized and learned during the training of the network. Once a vector for each patch is calculated they are combined into a three dimensional output map. This is done in a way so that every spatial location in the output feature map correspond to the same location in the input feature map. The whole process of a convolution is shown in figure 6.2.

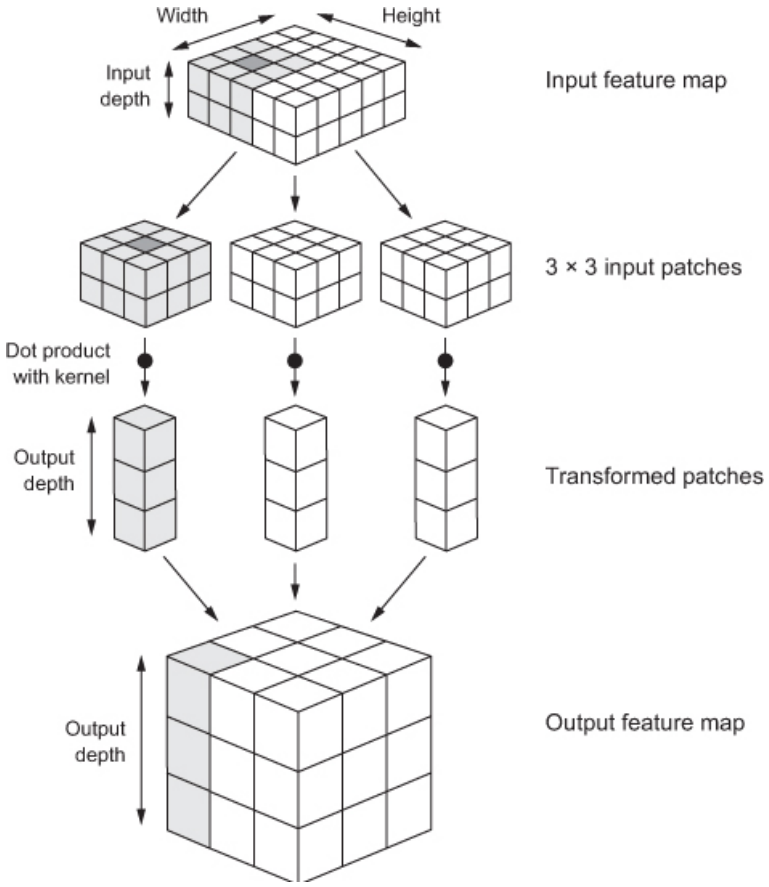


Figure 6.2: The whole process of a two dimensional convolution.

The way the convolution is performed is also what makes convolution layers fundamentally different from densely connected layers that are explained in section 6.1.5 Dense Layers. Dense layers learn global pattern in their input feature space, while convolution layers learn local patterns. These local patterns are found in the patches of specified size extracted from the input. This characteristic of learning local patterns, gives convolutional neural networks two important properties. First of all the patterns they learn are translation invariant, meaning that for instance after learning a pattern in the lower left corner of an image it can be recognized anywhere in the image. Contrary to that densely connected networks would have to learn the pattern new if it appeared in a different location on the image. The second characteristic is that convolutional neural networks can learn spatial hierarchies of patterns. The first convolutional layer can learn small local patters like edges and the next convolution layer will receive the features of the first layer and learn larger patterns, and so on. This results in the ability to learn increasingly complex patterns.

Last but not least it should be mentioned that the 3D input tensor of such a convolution layer can be created from an image, where the height an the width of the image are that of the tensor and the different colour channels of the image are represented by the depth of the tensor. However, after an image is passed through a convolution layer a new output feature map is produced and its depth is specified by one of the parameters mentioned above. As a result the depth channels no longer represent the different colour channels of the input image. Instead they are referred to as filters, that encode specific aspects of the input data.

### 6.1.2 Max-Pooling layers

There are multiple types of pooling layers, but max-pooling is the only pooling operation used in this work as is tends to work better than the other ones. Pooling operations are generally used to downsample the feature maps. Max pooling is performed by extracting windows, usually the size of 2 x 2, from the input feature maps and outputting the highest value of each channel. Such a window size halves the sizes of the feature maps. This aggressive downsampling has two effects. Firstly, the computational complexity is reduced massively as their are less values to process and secondly, it has as result that the data in the patches extracted from the inputs in successive convolution layers contain information about increasingly larger areas of the original input image. Therefore the size of the region in the input data that has influence on the individual features created by the convolution is increased. **TODO: ref zu** https://theaisummer.com/receptive-field/ This is a fundamental concept of convolutional neural networks and is further explained in section 6.3 Receptive field.

### 6.1.3 Dropout layers

The aim is to create models that generalize, meaning they perform well on data that is has never seen before. The central obstacle of generalization is overfitting. It describes the scenario in which the model adapts to much to the training data which reduced its ability to generalize. An effective way to reduce overfitting is using a dropout layer. Dropout is applied to a layer by randomly setting a number of output features of the layer during training to zero. The rate in which this happens can be specified and is usually between 0.2 and 0.5. In short, the idea is, that randomly setting output to zero produced noise that can break up random patterns which are not significant and otherwise would be memorized by the model if no noise was present. Hinston, who originally developed this technique, refers to these non significant patterns as *conspiracies*.

### 6.1.4 Flatten layers

A flatten layer simply receives a multi dimensional input tensor, in case of two dimensonal cnns this tensor is three dimensional, and flattens it into a 1d tensor that can be fed to a densely connected network.

### 6.1.5 Dense Layers

After flattening, the resulting 1d tensor is passed to a densely connected network. This network consists of multiple dense layers that classify the input features. In dense layers each feature depends on the whole of the input data. Each layer applies tensor operations, that involve weights randomly initialized and learned during the training of the network. The final layer is a dense layer with softmax activation that returns an array in the size of the number of classes, specifying probabilities for each class. The class with the highest probability is chosen as the final prediction. Activation functions are briefly explained in section 6.2 Activation functions.

## 6.2 Activation functions

In this work, activation functions are used in combination with convolutional and dense layers. One activation function, the softmax function, has been already mentioned in section 6.1.5 Dense Layers. It produces an array that implies the prediction of the model. In short, activation functions simply receive an input and calculate an output depending on that function. The other activation function that is utilized in this work is the rectified linear unit (relu). It enables the the layer to learn non linear transformations of the input data. In comparisson to otherwise being limited to linear transformations this gives the model access to a much richer hypothesis space. All the relu does is take the input and return 0 if the input in less than 0 and otherwise return the input value. For instance if combined with a convolution layer, each value in the output feature map is passed to the relu. This results in a new feature map that is passed to the next layer.

## 6.3 Receptive field

One of the basic concepts of convolutional neural networks is the receptive field, or field of view, of a feature in a certain layer of the network **TODO: source website**. As mentioned before in dense layers the features depend on the whole input, while a feature in a convolution only depends on a local region of the input. This region is the receptive field of that feature.

The receptive field size of features in the network can be increased in multiple ways. One way is by stacking more convolution layers, which increases the receptive field size linearly by theory. This effect is visualized in figure 6.3. Another, more effective way, is to use a pooling layer for subsampling, which increases the receptive filed size multiplicatively. As a result these methods can be used to increase the size region of the original input image that has influence on features individual features in deeper layers.

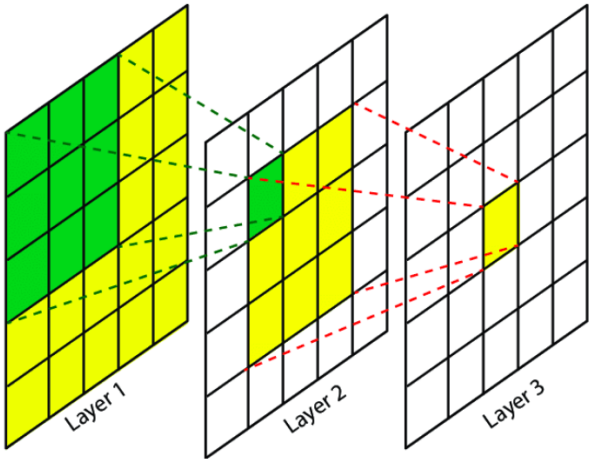


Figure 6.3: Visualization of a linearly growing receptive field is stacked convolutions.

These conscideration are important when building a model. As stated by blaa..: „Since anywhere in an input image outside the receptive field of a unit does not affect the value of that unit, it is necessary to carefully control the receptive field, to ensure that it covers the entire relevant image region.“ The features are here referred to as units. This means that if there are regions of a certain size in the input image that should influence individual features as a whole, is must be made sure when constructing the model that the receptive field in deeper layers is large enough. **TODO:** https://arxiv.org/pdf/1701.04128.pdf

## 6.4 Differences of one dimensional convolutional neural networks

One dimensional convolutional networks can be used to classify sequence data. Instead of a 3d tensor, they receive 2d tensors as inputs. Such sequence data could for instance consist of muscle activity over time and the task could be to classify the type of movement. The convolutions explained in section 6.1.1 Convolution layers are two dimensional as the extracted patches from the image are two dimensional. The convolutions in 1D CNNs are one dimensional, meaning they extract local one dimensional patches from the sequence. Apart from this the convolution is performed similarly. The process of a one dimensional convoluion is shown in figure 6.4.

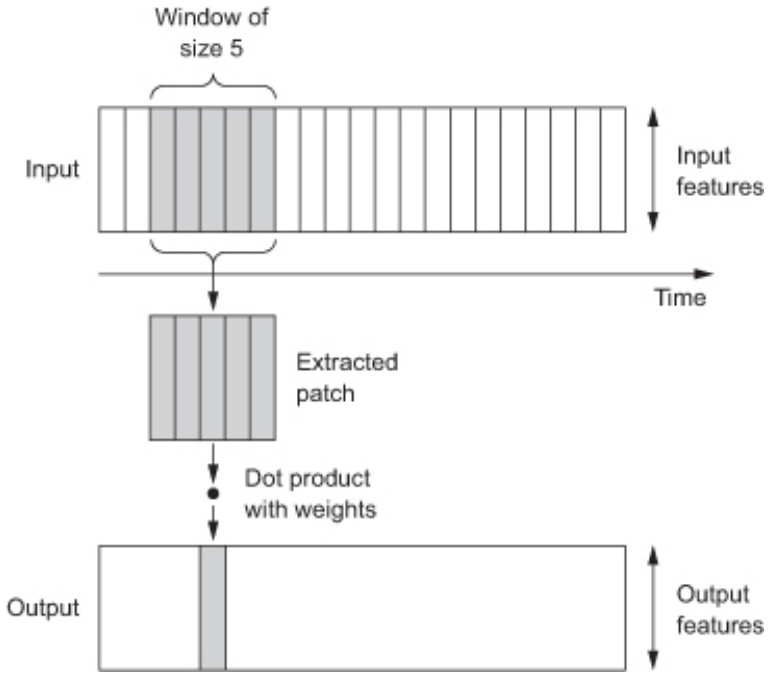


Figure 6.4: The whole process of a one dimensional convolution.

The other difference is how the max-pooling operation is performed. Instead of extracting 2D patches and and outputting the maximum value, this is done for 1d patches. The effect of this operation are the same as is 2d cnns.

The special characteristics of 2D CNNs also apply to 1D CNNs. They are transaltion inviarant and capable to learn hierarchical paterns. Furthermore the dropout, flatten and dense layers as well as activations function in the same way. The behaviour explained in section 6.3 Receptive field can also be applied to 1D Cnns: The patches extracted from the inputs in successive convolution layers contain information about increasingly larger areas of the original input sequence.

# 7. Training neural networks

Although the different components of CNNs have been covered in chapter 6 Convolutional neural networks, it has not been addressed yet, how these networks are trained to make accurate predictions. Therefore the general concepts involved in training neural networks will be briefly explained.

The relationship between the core concepts revolving around training neural networks is shown in figure 7.1. The fundamental data structure in neural networks is the so called layer. There are many different types of layers that are chained together to form the network. In a classification task, what the network does is map the input data to a prediction. This is also called forward passing, as each layer performs operations on the received data and passes the reults to the next layer. Once this chain reaches the output layer and by that outputs a prediction, the loss function takes that prediction as well as the actual target label and calculates a loss value which is a measure of how well the network's predictions match the expected outcome.

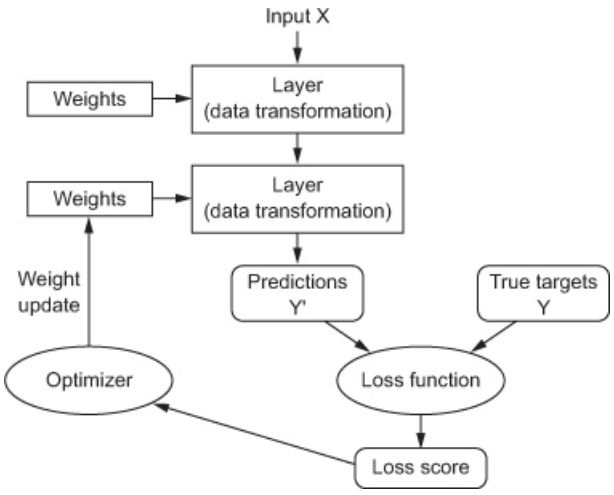


Figure 7.1: Core concepts in training neural networks.

This value is used by the optimizer to update the networks weights. This is done through an algorithm called backpropagation which works backward from the output layer to the input layer and computes the contribution that each weight had in the loss value. Using a method called stochastoc gradient descent, the wights are adjusted in small steps to reduce the loss. The procedure of making predictions, calculating the loss and updating the weights is repeated many times, so that the loss is step by step reduced, resulting in more accurate predictions.

# 8. Classification

## 8.1 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: source**, while wo dimensional coovolutinal neural networks are mostly used for image processing. However, a possibility is to create synthetic images from the data ,as done in section 5.7.2 2D CNN features, and use these images in a two dimensional convolutional network. The dimensions of the data and the feature maps in the visualizations are calculafed for the case that all 20 rounds are used. If a different number of rounds is used the values differ.

The models do not neccessarily produce the best results possible. We use best practice CNN 1d **TODO: source** and 2d **TODO: source** topology. As the amount of collected real data with 20 recordings for each label is very limited, it was decided not to split the data into training, test and validation data, and instead only into training and test data using leave-one-out cross validation method, where we train each model with data from n-1 subjects and thus we test on the data from the remaining subject. As there is no separate validation set, meaning an optimization would have to be done on the test data, it was avoided to optimize the hyper parameters individually for each model to ensure generalizing on new testing data. Instead, common best practice hyper parameters were optimized over all the models for better generalizing ability.

### 8.1.1 1D CNN

The structure of the 1D CNN used is shown in figure 8.1.

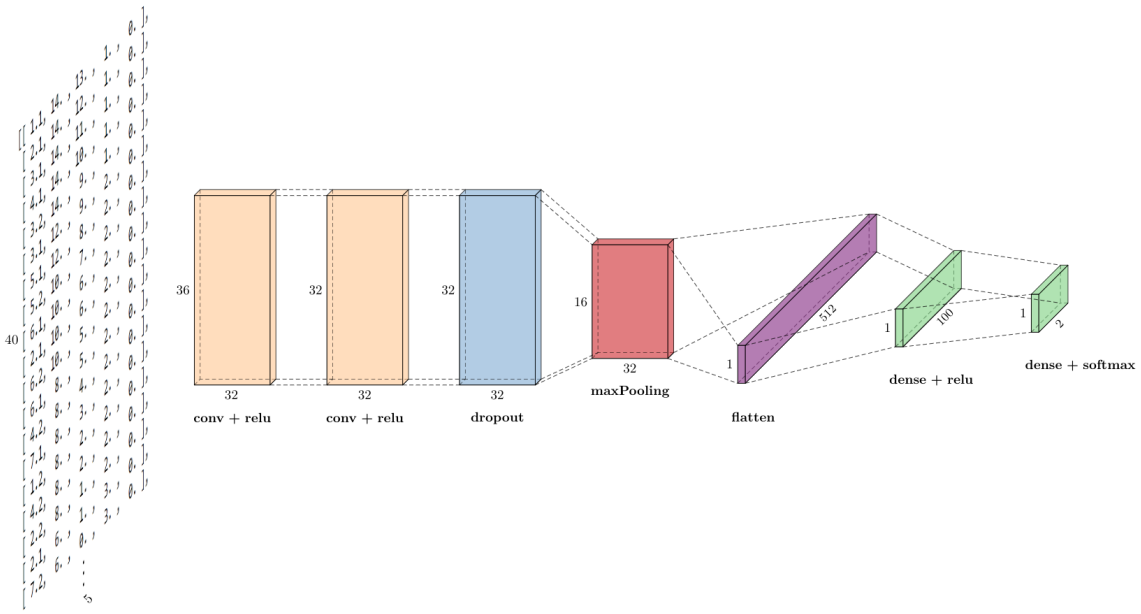


Figure 8.1: Visualization of the utilized 1d cnn topology. The dimensions represent those of the feature map in that layer. As example, the first convolutional layer produces a feature map with the size 32 x 36 with 36 being the number of filters computed.

The input data consist of the 5 statical features, explained in section 5.7.1 1D CNN features, for each of the 40 steps (20 rounds), resulting in the input dimensions $40 \cdot 5$. Initially the data is passed to the first convolutional layer of the network. 32 filters are computed by the first layer and the convolutional kernel has a size of 5, meaning that every step from the kernel includes all data from 5 steps in the game. The resulting feature map has a size of 32 x 36. This feature maps is passed to the second convolutional layer that computes the same number of filters and with the same kernel size as the first layer. This results in a feature map with a size of 32 x 32. Afterwards a dropout layer with a rate of 0.5 randomly sets input features to 0 to reduce overfitting. Then a one dimensional max pooling layer with a pool size of 2 reduces the size of the feature map to 32 x 16. Afterwards, a flatten layer forms the features into one dimension. These features are finally passed to a densely connected network, consisting of two dense layers. Both convolutional layers, and the first dense layer use a rectified linear unit as activation function. The last dense layer uses softmax as activation function to bring the data down two two values for the two classes. The elements of the output vector are between 0 and 1, sum up to 1 and specify the result of the classification. In total 57486 parameters are trained in this model.

As optimizer the Adam optimizer from keras is used that implements the Adam algorithm. Adam optimization is a stochastic gradient descent method **TODO: source keras**. The learning rate is set to 0.0001 and not changed during the training. The batch size is set to 32 and the model is trained for 2500 epochs. As mentioned before, this is a best practice configuration, optimized over all the 1d cnn models and not indivually for each model.

### 8.1.2 2D CNN

The structure of the 2D CNN used is shown in figure 8.2.

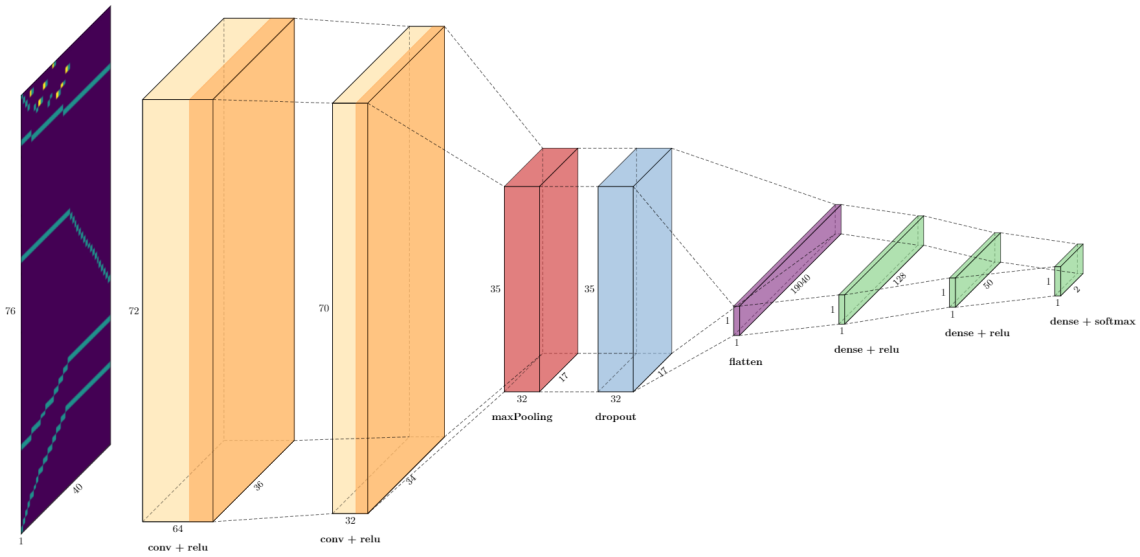


Figure 8.2: Visualization of the utilized 2d cnn topology. The dimensions represent those of the feature map in that layer. As example, the first convolutional layer produces a feature map with the size 72 x 36 x 64 with 64 describing the number of filters computed.

The input data consist of the synthetic images generated in section 5.7.2 2D CNN features, using the 5 statistical features. This results in the input dimensions $76 \cdot 40 \cdot 1$. Initially the data is passed to the first convolutional layer of the network. 64 filters are computed and the kernel has a size of 5 x 5, meaning that each step from the kernel includes a $5 \cdot 5$ sized area of the synthetic image. This convolution results in a feature map with a size of 64 x 72 x 36. This feature map is passed to the second convolutional layer that computes 32 filters with a kernel of size of $3 \cdot 3$. This results in a feature map with the size of 32 x 70 x 34. Then a two dimensional max pooling layer with a pool size of $2 \cdot 2$ reduces the size of the feature map to 32 x 35 x 17. Afterwards a dropout layer with a rate of 0.2 randomly sets input units to 0 to reduce overfitting. Afterwards, a flatten layer forms the features into one dimension. These features are finally passed to a densely connected network, consisting of three dense layers. Both convolutional layers, and the first two dense layers use a rectified linear unit as activation function. The last dense layer uses a softmax activation function to bring the data down two two values for the two classes. The elements of the output vector are between 0 and 1, sum up to 1 and specify the result of the classification. In total 2463928 parameters are trained in this model.

As optimizer the Adam optimizer from keras is used that implements the Adam algorithm. Adam optimization is a stochastic gradient descent method **TODO: soruce keras**. The learning rate is set to 0.00001 and not changed during the training. The batch size is set to 32 and the model is trained for 1000 epochs. As mentioned before, this is a best practice configuration, optimized over all the 2d cnn models and not indivually for each model.

## 8.2 Training setup

There is one script each for training the 1D cnn and the 2d cnn. The only differences are in the structure and the hyper parameters 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 2 real and 2000 simulated games not used for training, only the data from the real games is used for testing. **TODO: leave one out cross validation erwähnen, generell umformulieren weil es maximal so viele simulierte sind und nicht generell** This is done becauuse for the real world isuge it is not relevant how the models perform on simulated data. Furthermore it can be chosen for which ratios between simualted and real games models should be trained. It was chosen to use ratios of one real game to 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, and 20 simulated games. Moreover it can be chosen how many turns in the game should be used (a turn consists of flipping 2 cards). In real world scenarios, the less the faster the decision is made and assistance can be applied, but if waited the longer the decision will likely be correct more often. To analyse this, models for different amounts of turns are trained. The turns chosen are the first 5, 10, 15 and 20. More than 20 turns were not recorded.

Once one of the scripts is executed, required directory structures are created and training processes for the different ratios of real to simulated games are performed successively. The training for one ratio consists of parallelized training of 20 splits using multiprocessing. The training for each split is repeated 20 times. As a result for each ratio 20 models are created for every split. This means that 400 models are trained for every ratio. As there are 12 different ratios trained, this results in 4800 models trained for each execution of one of the scripts. As mentioned above, the training is done for 4 different amounts of steps for the 1d cnn and the 2d cnn. This adds up to 38400 models being trained. Furthermore this done twice: Once before the changes to the simulator and once afterwards, to see if the changes improved the performance of the models. All in all this results in 76800 models. For each model the training history is saved in a file. Saving and loading the historeis is neccessary due to the parallel training of multiple models through multiprocessing. Once the training of the 4800 models of one script is completed, the training histories are loaded from the files. The results from the 400 training histories for each ratio of real to simulated games are averaged, plotted and the final figures are saved for the analysis. The reason for averaging the results over many models is, that deep neural networks by default initialize their weights randomly, resulting in potentially different outcomes of each training **TODO: stattdessen begründung mit stochastic models mit referenz**. By averaging the results, the values and therefore the interpretation becomes more reliable and meaningful. All models are trained on the CPU. The fact that 76800 models are trained led to the decision, not to save all models but instead first evaluate the results and then retrain and save the models that create the best results.

## 8.3 Training results and evaluation

**TODO: sd und alle anderen abkürzungen erklären**

The training results for the various configurations can be seen in table 8.1, 8.2, 8.3 and 8.4. The first two tables contain the results before the changes to the simulator and the last to the results afterwards. The analyiss will only focus on the accuracy, but the loss is also listed for the sake of completeness. The best accuracies for each number of turns in each table are highlighted in green. If multiple ratios of real to simulated game for a certain number of turns produce the same accuracy only the one with the smallest amount of simulations is highlighted. The accuracies in the tables are those from the best epochs. In most of the cases the models start overfitting after a certain epoch resulting in a deterioration of the accuracy after reaching a local maximum. All models except the 1d cnns for 5 rounds and ratios of sd0x and sd1x reach such a local maximum in accuracy before the training is stopped.

It should be claryfied that the chosen configurations of models to be trained make no claim to completeness.

| | 5 r. | | 10 r. | | 15 r. | | 20 r. | |
|---|---|---|---|---|---|---|---|---|
| | Acc. | Loss | Acc. | Loss | Acc. | Loss | Acc. | Loss |
| sd0x | 65.12% | 0.6584 | 81.12% | 0.5116 | 78.37% | 0.5377 | 79.62% | 0.492 |
| sd1x | 71.25% | 0.6385 | 80.88% | 0.4777 | 77.25% | 0.5355 | 80.12% | 0.4999 |
| sd2x | 68.5% | 0.6476 | 77.38% | 0.5283 | 78.25% | 0.5275 | 81.25% | 0.4887 |
| sd3x | 69.38% | 0.6494 | 75.5% | 0.5252 | 76.13% | 0.5329 | 79.75% | 0.4999 |
| sd4x | 67.5% | 0.6436 | 75.88% | 0.538 | 77.25% | 0.5322 | 81.25% | 0.4975 |
| sd5x | 72.25% | 0.6212 | 76.5% | 0.5252 | 77.75% | 0.5338 | 81.25% | 0.4985 |
| sd6x | 71.87% | 0.6346 | 74.75% | 0.547 | 76.25% | 0.5376 | 79.38% | 0.5075 |
| sd7x | 70.25% | 0.6252 | 73.62% | 0.5395 | 76.0% | 0.5326 | 80.0% | 0.5078 |
| sd8x | 74.25% | 0.6121 | 72.5% | 0.5509 | 77.0% | 0.535 | 79.75% | 0.4993 |
| sd9x | 74.88% | 0.5826 | 74.63% | 0.5483 | 76.0% | 0.542 | 81.75% | 0.4987 |
| sd10x | 74.87% | 0.5971 | 74.5% | 0.5558 | 77.12% | 0.5394 | 80.87% | 0.5036 |
| sd20x | 73.25% | 0.5831 | 73.5% | 0.5493 | 75.63% | 0.5355 | 80.75% | 0.5013 |

Table 8.1: Best accuracies and losses for 5, 10, 15 and 20 steps produced by the 1d cnn. Before the changes to the simulator.

| | 5 r. | | 10 r. | | 15 r. | | 20 r. | |
|---|---|---|---|---|---|---|---|---|
| | Acc. | Loss | Acc. | Loss | Acc. | Loss | Acc. | Loss |
| sd0x | 63.5% | 0.6471 | 74.0% | 0.4991 | 79.62% | 0.4726 | 79.0% | 0.4778 |
| sd1x | 68.12% | 0.619 | 76.62% | 0.5171 | 79.5% | 0.4984 | 81.75% | 0.4735 |
| sd2x | 68.25% | 0.6382 | 80.0% | 0.5243 | 80.38% | 0.4851 | 84.12% | 0.4573 |
| sd3x | 69.5% | 0.6392 | 78.88% | 0.5518 | 80.25% | 0.5038 | 84.62% | 0.4718 |
| sd4x | 67.63% | 0.6415 | 76.38% | 0.5685 | 79.62% | 0.5106 | 85.0% | 0.4726 |
| sd5x | 70.88% | 0.6339 | 74.63% | 0.5668 | 80.0% | 0.51 | 84.88% | 0.474 |
| sd6x | 70.62% | 0.638 | 77.0% | 0.5623 | 80.12% | 0.5077 | 84.88% | 0.4776 |
| sd7x | 69.75% | 0.6543 | 75.38% | 0.5762 | 79.5% | 0.5176 | 85.0% | 0.4839 |
| sd8x | 70.0% | 0.6467 | 74.25% | 0.5775 | 80.38% | 0.5164 | 85.0% | 0.484 |
| sd9x | 71.25% | 0.6395 | 78.75% | 0.5696 | 81.5% | 0.5064 | 85.0% | 0.4814 |
| sd10x | 68.75% | 0.6296 | 77.75% | 0.5674 | 80.12% | 0.5098 | 85.0% | 0.4802 |
| sd20x | **73.0%** | 0.5833 | 76.63% | 0.5486 | 79.75% | 0.5064 | 84.88% | 04704 |

Table 8.2: Best accuracies and losses for 5, 10, 15 and 20 steps produced by the 2D CNN. Before the changes to the simulator.

| | 5 r. | | 10 r. | | 15 r. | | 20 r. | |
|---|---|---|---|---|---|---|---|---|
| | Acc. | Loss | Acc. | Loss | Acc. | Loss | Acc. | Loss |
| sd0x | 63.87% | 0.6698 | 81.88% | 0.5014 | 78.5% | 0.5401 | 78.62% | 0.5107 |
| sd1x | 67.75% | 0.6648 | 79.0% | 0.5095 | 75.75% | 0.5305 | 79.38% | 0.5101 |
| sd2x | 66.0% | 0.6506 | 79.25% | 0.4845 | 76.0% | 0.5325 | 78.37% | 0.514 |
| sd3x | 64.38% | 0.6585 | 77.38% | 0.5037 | 77.62% | 0.5261 | 79.25% | 0.5113 |
| sd4x | 67.0% | 0.6377 | 75.38% | 0.5289 | 77.62% | 0.5331 | 80.25% | 0.5027 |
| sd5x | 67.0% | 0.6452 | 76.75% | 0.5146 | 78.75% | 0.5226 | 79.0 | 0.5042 |
| sd6x | 70.25% | 0.6427 | 75.25% | 0.5335 | 78.75% | 0.5208 | 79.38% | 0.5049 |
| sd7x | 70.25% | 0.6449 | 76.88% | 0.5114 | 78.0% | 0.5129 | 78.75% | 0.5115 |
| sd8x | 71% | 0.628 | 76.25% | 0.523 | 77.25% | 0.5261 | 79.12% | 0.5156 |
| sd9x | 70% | 0.6276 | 75.37% | 0.5226 | 77.62% | 0.5281 | 78.63% | 0.522 |
| sd10x | 71.5% | 0.6242 | 76.25% | 0.5261 | 76.38% | 0.5331 | 78.62% | 0.4986 |
| sd20x | 71.88% | 0.6246 | 72.38% | 0.5617 | 76.25% | 0.5416 | 79.12% | 0.5 |

Table 8.3: Best accuracies and losses for 5, 10, 15 and 20 steps produced by the 1D CNN. After the changes to the simulator.

| | 5 r. | | 10 r. | | 15 r. | | 20 r. | |
|---|---|---|---|---|---|---|---|---|
| | Acc. | Loss | Acc. | Loss | Acc. | Loss | Acc. | Loss |
| sd0x | 63.62% | 0.6461 | 73.87% | 0.4929 | 80.0% | 0.4557 | 78.5% | 0.476 |
| sd1x | 67.88% | 0.6446 | 76.12% | 0.531 | 79.5% | 0.4911 | 79.75% | 0.4874 |
| sd2x | 69.75% | 0.6463 | 79.62% | 0.5243 | 80.87% | 0.4905 | 80.0% | 0.4859 |
| sd3x | 69.5% | 0.6449 | 77.5% | 0.5507 | 82.25% | 0.4994 | 80.0% | 0.4865 |
| sd4x | 69.0% | 0.6347 | 72.75% | 0.5684 | 80.12% | 0.5156 | 80.0% | 0.495 |
| sd5x | 69.38% | 0.6342 | 73.62% | 0.5634 | 80.5% | 0.5122 | 80.0% | 0.4917 |
| sd6x | 71.0% | 0.6326 | 74.5% | 0.5586 | 80.62% | 0.5038 | 79.5% | 0.4874 |
| sd7x | 70.0% | 0.6328 | 71.62% | 0.555 | 80.0% | 0.5061 | 79.5% | 0.4874 |
| sd8x | 69.88% | 0.6287 | 72.75% | 0.5567 | 80.25% | 0.5057 | 79.62% | 0.4874 |
| sd9x | 69.12% | 0.6238 | 72.38% | 0.5532 | 79.5% | 0.5067 | 78.88% | 0.4887 |
| sd10x | 69.5% | 0.6203 | 73.25% | 0.5536 | 79.5% | 0.5062 | 79.0% | 0.4904 |
| sd20x | 69.88% | 0.5935 | 73.0% | 0.5494 | 79.62% | 0.5005 | 78.75% | 0.4874 |

Table 8.4: Best accuracies and losses for 5, 10, 15 and 20 steps produced by the 2D CNN. After the changes to the simulator.

The results show that using a certain number of simulated games for the training can often significantly improve the classificatin resuls over using only the real data. A good example is the accuracy for sd0x compared to sd4x for 20 rounds for the 2d cnn in table 8.2. Using no simulations the accuracy is 79.0% while for a ratio of 9 simulations for each real game the accuracy is 85.0%. Furthermore is can be seen that using to many simulations compared to real games can decrease the accuracy. In table 8.2, for 10 rounds, the accuracy for sd2x is 80.0% and less when using more simulations. This behaviour highly varies depending on the configuration of the model. Sometimes the models produce the best results without any simulations, while other times using simulations strongly outperforms using only real data. It is possible that the optimal ratios of simulated to real games are not included the tested configurations. Especially the 2d cnn for 5 rounds before the changes to the simulator achieves the best accuracy with the highest ratio that was tested. Therefore the optimal ratio could be higher than 20 simulations for each real game.

When looking at the best accuracies of 1d and 2d cnns for each number of rounds, it can be observed, that the accuracy of the 1d cnns do not get much better and sometimes even worse after more than 10 rounds, while this is not the case for 2d cnns. This can be seen in figure 8.3. The fact that the 1d cnns do not profit from more than 10 rounds might be related to the findings from section 5.6 Evaluation of simulation, that indicate that the simulaor perfroms worse in later rounds.

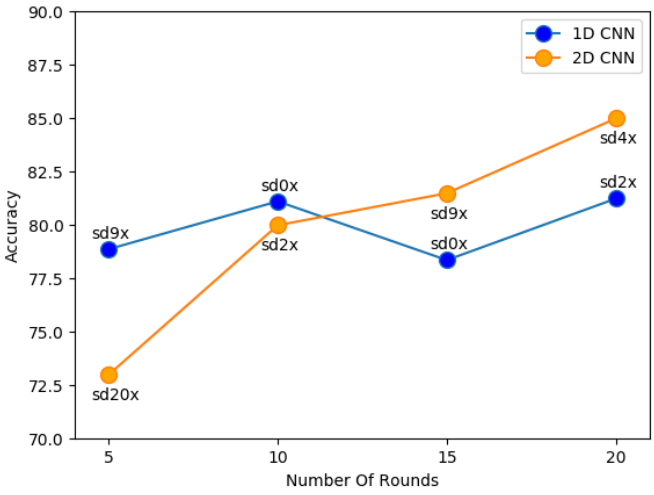


Figure 8.3: Best results for the different number of rounds before the changes to the simulator. Ratios of real to simulated games is additionally described.

However, if the case, this begs the questions why this does not apply to 2d cnns. There is a possibility, that the worse perfromance in later turns has a strong impact on the statistacal features fed to the 1d cnn, but once these feautures are used to create the synthetic images for the 2d cnn this negative impact decreases. Figure 8.3 additionally shows the ratios of simulated to real data for each of the best results. The fact that the 2d cnn after 10 rounds still have the best results when using a substantial amount of simulations while the the best results for the 1d cnns after the 10th round use very little simulations, supports the theory from above. Nonetheless the findings are not sufficient to verify the theory. For that more research would be required. However, if found to be true, it could mean that 2d cnns using synthetic images have the potential to handle inacurate simulations better than 1d cnns. From a different perspective, this could also indicate that, assuming the simulator does actually perform worse in later rounds and is improved in that regard, the accuracy of the 1d cnns after 20 rounds could be improved.

It was conscidered to perform statical tests comparing different models and ratios to reveal significant or insignificant differeneces. However, the decision was made that this does not realy gives reliable results, because the results are not neccesarily the best achievable as the models were not individually optimized. Therefoe the results of ststistical tests performed on the achieved accuracies, would not justify any staements regarding whether 1d cnns or 2d cnns perform significntly better in different configurations. Such statistical test would only show significant differences or indiffierences in the results, but could not be interpreted for any valuable findings, as not all models are used to their full potential. For choosing the best configurations from those tested no statistical tests are used. Instead simply those with the highest accuracies are chosen.

| | 5 r. | 10 r. | 15 r. | 20 r. | Average |
|---|---|---|---|---|---|
| 1D CNN - before | 74.88% (sd9x) | 81.12% (sd0x) | 78.37% (sd0x) | 81.25% (sd2x) | 78.91% |
| 1D CNN - afterwards | 71.88% (sd20x) | 81.88% (sd0x) | 78.75% (sd5x) | 80.25% (sd4x) | 78.19% |
| 2D CNN - before | 73.0% (sd20x) | 80.0% (sd2x) | 81.5% (sd9x) | 85.0% (sd4x) | 79.88% |
| 2D CNN - afterwards | 71.0% (sd6x) | 79.62% (sd2x) | 82.25% (sd3x) | 80.0% (sd2x) | 78.22% |

Table 8.5: Highest accuracies before and after the changes to the simulator. Furthermore the rations of simulated to real games and the average accuracies of the best results from each configuration are shown.

Table 8.5 shoes the best result for all categories before and after changing the simulator. It can be seen that although some accuracies are slightly better after the changes to the simulator the results are on average noticeably worse than before. For instance is the accuracy of 85.0% from the 2d cnn for 20 rounds decreased to 80.0%. The reason for this is unknown and can be highly complex, due to the many factors involved. A theory is, that reucing the randomness of the simulator and therefore making the simulated games more similar to the real games, may have resulted in decreased generalisation in the models and therefore less robustness on unknown data. However, this theory can hardly be varified. The descreased performce after the changes leads to the decision to only use the models trained on the simulated data created before the changes to the simulator. The chosen configurations can be seen in table 8.6.

| | 5 r. | 10 r. | 15 r. | 20 r. |
|---|---|---|---|---|
| 1D CNN | 74.88% (sd9x) | 81.12% (sd0x) | | |
| 2D CNN | | | 81.5% (sd9x) | 85.0% (sd4x) |

Table 8.6: The chosen configurations and their accuracies (averaged over 400 models each).

The 1dnns trained, perform better for 5 and 10 rounds than 2d cnns while 2d cnns perform better for 15 and 12 rounds. Depending on the number of rounds it can therefore be benefitial to use either 1d cnns or 2d cnns. The 4 best configurations for the differnet numbers of rounds are used in section 8.4 Voting based systems. Figures 8.4, 8.5, 8.6 and 8.7 show the average training and test accurcy during the course of training the models of these 4 configurations.

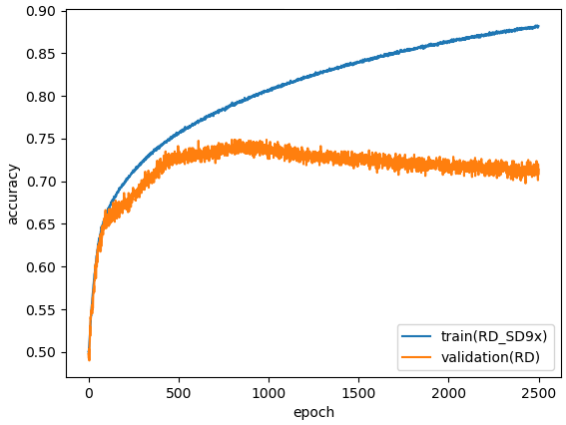


Figure 8.4: Average training and validation accuracy of 1d cnns on 5 rounds and a ratio of 9 simulated games for each real game. Best validation accuracy is 74.88% at epoch 812.

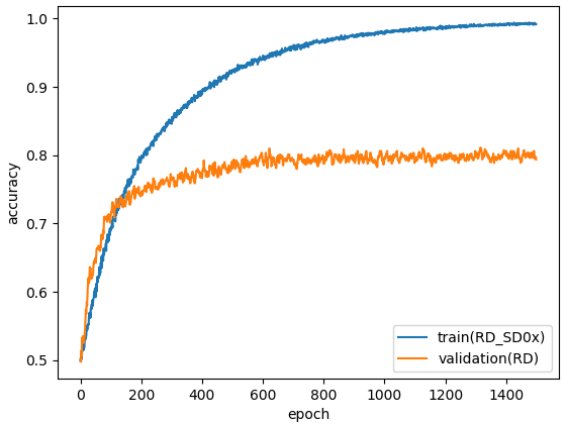


Figure 8.5: Average training and validation accuracy of 1d cnns on 10 rounds using no simulated games. Best validation accuracy is 81.12% at epoch 1317.

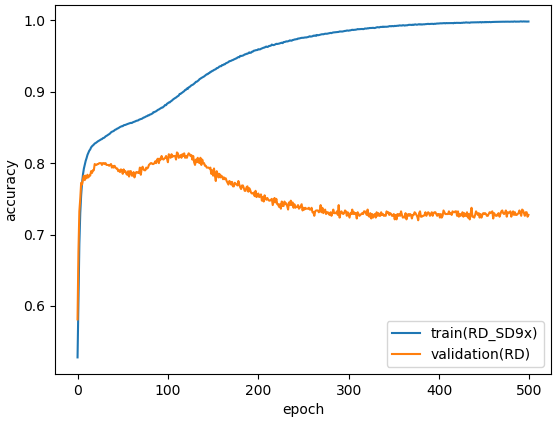


Figure 8.6: Average training and validation accuracy of 2d cnns on 15 rounds and a ratio of 9 simulated games for each real game. Best validation accuracy is 81.5% at epoch 111.

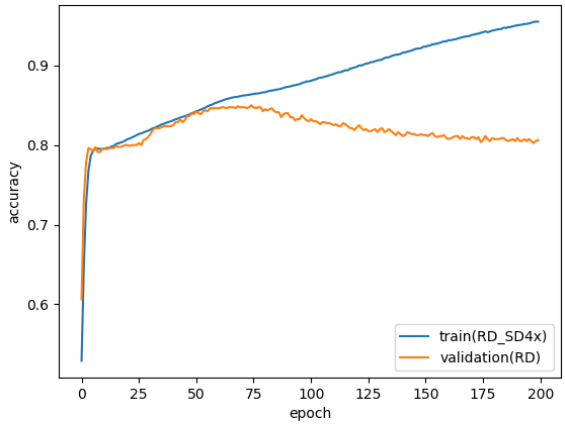


Figure 8.7: Average training and validation accuracy of 1d cnns on 20 rounds and a ratio of 4 simulated games for each real game. Best validation accuracy is 85.0% at epoch 75.

## 8.4 Voting based systems

The accuracies shown in section 8.3 Training results and evaluation are not representitive of how the models would perform in real life scenarios. Since the accuracy is the mean accuracy of 20 repititions for each split there is no single modeel that produces that accuracy. And due to the fact that stochastic models can produce different outcomes each time they are trained, using only one of the models for the prediction does not produce relaible results **TODO: quelle**. For analysing how the modles would perform in real life scenarios a voting based system is deployed **TODO: quelle die sagt dass man sowas benutzt**. It consists of multple models classifying the input data and then voting for the final classification. A voting based sytsem for each of the best cnns and ratios of simulated to real data for each number of step is created. As shown in table 8.5 the 1d cnn creates the best accuracy for 5 and 10 rounds while the 2d cnn creates the best accuracies for 15 and 20 rounds. Hence, in total 4 voting based systems are created, one for each of the 4 number of rounds. Each of the 20 repititions of a split is tested as before, only with the two real games (one no obstacle and one glare) that were not included in the training. However instead of direclty calculating the accuracy of each repitition of that split, the output labels are saved for all repitions and used to perform a voting for the final label. Meaning that if for instance 15 out of 20 repitions classyfied the game as a glare effect game, while the other 5 classified it as a no obstacle game, the final classification will be that of a glare effect game. This is done for the two test games specific to each split. Using the final labels created by the voting, the accuracy of that split is calculated. In the example above, if assumed that both final labels would be correct, the accuracy of that split would be 100%. This procedure is repeated for each of the 20 splits and the accuracies of the splits are averaged.

As none of the models have been saved after the training, the 400 models for each of the 4 voting based systems, are retrained and saved. In total 1600 models are trained and saved for the voting based systems. However, they are trained for less epochs than in section 8.3 Training results and evaluation, as the models would otherwise overfit. For each of the 4 configurations the number of epochs in chosen so that the accuracy of the models in the last epoch, and therefore the accuracy of the saved model, is in the area of the local maxima. The number of epochs was intentionally chosen only in a rough area of the local maximum of the test accuracy and not exactly at the maximum. As mentioned in section 8.1 Models there is no validation set and therefore using the exact number of epochs as optimal for the test data would mean that the models would be optimized according to the test data and the results would be less representative of the real world performance of the systems. Table 8.7 shows amongst other things at which epoch the training of the different models was stopped and what the average accuracies were after the according epoch, with and without voting. The slight difference of the average accuracies without voting shown in the table and the best accuracies shown in table 8.6 are a result of two factors: Firstly, the number of training epochs was only chosen roughly in the area of the maximum accuray and secondly, due to the nature of stochastic models the accuracy in a specific epoch can vary in every training even though averaged over many models.**TODO: source**

| | training epochs | avg. acc. | acc. of VBS | Overruled correct classifications | Overruled false classifications |
|---|---|---|---|---|---|
| 5 r. | 650 | 73.13% | 72.5% | 38 | 33 |
| 10 r. | 1300 | 80.13% | 80.0% | 32 | 25 |
| 15 r. | 90 | 80.63% | 80.0% | 9 | 4 |
| 20 r. | 60 | 85.0% | 85.0% | 0 | 0 |

Table 8.7: Number of epochs the models were trained for, their average accuracy with and without voting as well as the number of overruled correct and false classifications.

When comparing the average accuracies without voting with the accuracies of the voting based systems, it can bee seen that the accuracy of the voting based systems is eithher sligly lower or identical. In some cases such a system may have a better accuracy that without voting, but this is not always the case. Two key factors that this is influenced by are the numbers of correct and false classifications that are overruled by the majority in the voting. If more correct classification are overruled by wrong ones than wrong by correct ones, the accuracy of the voting based system will be worse than the average accuracy without voting. The reason is that once a correct classification is overruled by the majority, it has no influence on the calculated accuracy any more, because the accuracy of the voting based system is determined by the final labels. An good example of such a case occured in the voting based system for 5 rounds, where in one of the splits, 11 out of the 20 models voted incorrect, resulting in 9 correct predicitoins being ignored. Table 8.7 also shows how many correct and false predictions were overruled by their opposition. In every system except the one for 20 rounds, more correct predictions are overruled than false ones, resulting in a lower accuracy of those voting based systems compared to the accuracy calculated without voting.

As shown by the accuracies that are all higher than 50%, the majority of the 400 models in each system make the correct prediction. However, their votes are not evenly distributed across all splits. A voting based systems highly profits from low fluctuation in the voting distribution between the models. If the voting in the majority of splits has similar results, such that for instance around 15 models vote correct and 5 vote wrong, the accuracy of those splits will all be 100% and the average accuracy of the voting will be higher than the average over all models without voting. But if in for instance in the first split 20 models vote correct and 0 incorrect, while in the second split 9 vote correct and 11 incorrect, the overall performance of the voting based system suffers. Instead it would be better if in the example above, in the first split 15 vote correct and 5 incorrect and in the second split 14 vote correct and 6 incorrect, resulting in the accuracy of both split being 100%. In total in both explained distributions, 29 models vote correct and 11 incorrect, but the performance of the voting based system that has less fluctuation in the distribution performs massively better. A way to achieve less fluctuation in the voting distribution can be to optimize the hyper parameters and the topology of the models.

It is important to mention that these voting based systems are not meant for production use, even if the fact that they are not optimized is overlooked. The training of models for production use, unlike before, should not be divided into 20 splits from which each split excludes real data for testing, but instead all of the real data should be included in the training. The validation of these systems would be done with be done by collecting data from new participants.

In the context of detecting interaction obstacles, models that use less rounds are of higher relevance, as they make their prediction earlier. If only one of the 4 voting based systems was used, the one for 10 rounds with an accuracy of 80% should be the preferred choice. Making the prediction after 5 rounds with an accuracy of 72.5% seem less ideal. Same goes for the two voting based systems for 15 and 20 rounds. The system trained on 15 rounds is not more accurate than the system for 10 rounds and waiting 20 rounds before making a prediction is too long. By then the user might have already left the game, due to a bad experience caused by the interaction obstacle. However, it is also possible to use all 4 systems. Each 5 rounds could be predicted whether an interaction obstacle is involved. This could be especially usefull as the glare effect, unlike for instance colour blindness, might only be an issue for a couple turns. If predicted that the sun is no issue anymore, the eventual adaptiation could be removed again. Continuing the adaptation eventhough there is no need for it anymore can be a bas user experience.

# 9. Conclusion

A similarity matrix was created that successfully describes the colour differences of the cards under the influnece of a simulated sun shining on the screen. It was used in a cognitive user simulator to accurately simulate glare effect games. This simulator was also used to simulate no obstacle games. It was attempted to improve the simulator, but it could not be clearly said if that was successfull. Although there are some indications that the realisticness of the simulations ws improved there are also others findings that suggest the opposite. One of them being that the classification results using the simulations were worse after the changes to the simulator. Additional findings suggest that the simulator performs less good in later rounds of the game. Furthermore it was shown that incorporating certain amounts of simulated data improves the classificatiuon results significantly.

For the classification of the visual interaction obstacle being the glare effect two approaches were implemented. Firstly, a 1d cnn was utilized that classifies a sequence of statistsical features. The second approach was less common: Creating synthetic images out of the naturally sequential data and use 2d cnns to classify the cretaed images. Classifiers for four different numbers of rounds were tested. While the best result for 5 and 10 steps were achieved by the 1d cnns, the 2d cnns putfermored the 1d cnns when using 15 and 20 rounds. Findings indicate the possibility that 2d cnns are less effected by inaccurate simulations than 1d cnns.

Four voting based systems, one each for 5, 10, 15 and 20 rounds were implemented, that decide the final classification by a voting of multiple models. The systems for 5 and 10 rounds utilze 1d cnns while those for 15 and 20 rounds 2d cnns. The accuracies of the four systems in the order of the number of rounds were 73.13%, 80.13%, 80.63% and 85.0%. As the topologies and the hyper parameters of the models are not individually optimized and instead best practice values were chosen it is advisable to optimize them before using them. Two different usages are suggested for such voting based systems. Either use a single system trained to classify games after 10 rounds, as the accuracy is high and the decision is made relatively early in the game, or use all four voting based systems over the course of the game, depending on the number of round. This would mean making a prediction every 5 rounds until round 20 and adjusting the eventual adaptation accordingly. When conscidering

that the glare effect is a volatile visual obstacle meaning that adaptation may not be neccessary througout the whole game only because it was in for instance round 5, making a prediction each 5 steps would bring the advantage of being able two switch the adaption on and off more often. In the case of persistant visual obstacles such as colour blindness this is not useful, as the adaption is needed throughout the whole game.

## 9.1 Prospect

Regarding the simulation of user behaviour instead of only simulating once and using a number of the best sumlations, it could be simulated multiple times to take only the single best simualation from each run. This could improve the overall quality of the simulations used to train the models. Regarding the simulator further research could be done to find out whether the simulator actually performs worse in later turns, as indicated by findings in this work. It this is the case and the reasons are discovered, further attempts could be made to improve the simulator.

Creating synthetic images out of sequnetial data and using the image for an image classification is very experimental and only one way of creating such images was tested in this work. Therefore it could be analysed how different visual representations of the sequential data influence the performance of the classifier, as it is possible that a better representation exists.

As mentioned in section 8.1 Models the data was only split into training and test data without separate validation data as the amount of game logs collected is very small. This is also the main reason why the models were not optimized, as a separate validation set would be needed. The models could be optimized on the validation data and then tested on the test data, assuring that they perform well on data that they have never seen before. A new user study could be performed in order to collect the necessary data to optimize the models. Such an optimization could significantly improve the classification results.

Last but not least it could be interesting to further inverstigae the creation of synthetic image and the rather unconventional usage of image recognition using 2d cnns on what was originally a sequence. In this work this approach outperfomred the more conventional use of 1d cnns in multiple cases. There is a possibility that the 2d cnns were more capable in handling inacurate simulations. Further research could potentially reveal that this is actually the case or that there are other aspects to using synthetic images and 2d cnns that make them outperform 1d cnns in certain sequence classifications. Both would be interesting findings.

# A. Appendix

## A.1 Scripts

In this work code was added two three existing projects. Furthermore multiple scripts independent from those projects were written.

All scripts independent from already existing projects are located in the digital folder *scripts* that was added to the digital submission. The folder contains a subfolder called *classification* as well as the following scripts:

- similarity_matrix_generator.py
- similarity_matrix_mapper.py
- similarity_matrix_plotting.py
- valid_logs_collector.py
- feature_generation.py
- best_results_plotting.py

The first three script are were used to create the similarity matrix, add it to the existing game logs and to create figure 5.3. If the required images are provided in full hd it can be used to create a similarity matrix for any type of visual obstacle. Otherwise the areas pixels are extracted from need to be adjusted. The script *valid_logs_collector.py* loads all real game logs and saves the valid ones in new files. Once the games are simulated, the script *feature_generation.py* loads all raw game logs and creates the statistical features for them, mentioned in section 5.7.1 1D CNN features. Lastly *best_results_plotting.py* was used to create figure 8.3. Some parts in *similarity_matrix_mapper.py* and *feature_generation.py* are taken from the Memory-Statistic project but a lot had to be added or rewritten in order to be utilized for this work.

The aforementioned subfolder *classification* contains the following scripts:

- glare_effect_classification_cnn_1D.py
- glare_effect_classification_cnn_2D.py
- glare_effect_cnn_voting.py

The first two are the script for training and testing the 1d and 2d cnnc, while *glare_effect_cnn_voting.py* contains the code for the voting based systems.

The following additions were made to the CMM project:

- NoObst_ConfigGenerator_dataCollection2020.java
- GlareEffect_ConfigGenerator_dataCollection2020.java
- NoObstWinStrategyTrainingDataGenerator_dataCollection2020.java
- GlareEffectWinStrategyTrainingDataGenerator_dataCollection2020.java

The first two are used to create configuration files that contain the optimized parameters for the simulation and the last two load the files and simulate the according no obstacle and glare effect games. They use functionalities from the simulator and already existing script served as example. These additions are uploaded on a new branch of the project called **TODO: namen suchen**. The changes made in an attempt to improve the simulator can also be found in this branch.

Furthermore these additions were made to the Memory-Statistic project:

- memory_data_analysis_glare_effect.py
- sorting_simulated_data_quality_glare.py
- glare_effect_plotting.py
- leave_one_subject_out_glare_effect.py

The first one is simply a copy of an already existing script that contained functionalities to which minor adjustment were made so that is could be used in the context of this work. The other three use those functionalities.
*sorting_simulated_data_quality_glare.py* was used to sort the simulations according to how close their performance is to the real games and *glare_effect_plotting.py* loads those sorted simulations as well as the real game in order to plot the two performance measurements, mentioned in section 5.6 Evaluation of simulation. Furthermore it was used to perform the various paired sample t-tests. Finally the script *leave_one_subject_out_glare_effect.py* creates the splits with the raw games logs, from which the statistical features are calculated. Most of the code in these four scripts created with already existing scripts as example.

Lastly the changes to the Memory-Game project made to collect the data for the creation of the similarity matrix are also uploaded on a new branch called **TODO: namen suchen**.

## A.2 Data

The following data is stored on the *share* server:

- screenshots used for the creation of the similarity matrix
- raw game logs as well as statical features for those games before and after the changes to the simulator
- training results before and after the changes to the simulator
- scripts used for training and testing
- models used in four voting based systems
- a presentation that was held regarding the two approaches for 1d and 2d cnns

# Bibliography


[ABB+04] John Anderson, Daniel Bothell, Michael Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. An integrated theory of the mind. *Psychological review*, 111:1036–60, 11 2004.

[Fis02] J. Fischer. Untersuchung der farbabstandsformeln des cielab farbraums auf ihre eignung, farbrauschen quantitativ und physiologisch richtig zu beschreiben, 2002.

[Ihr19] M. Ihrig. Analyse und vergleich von benutzerverhalten und gehirnaktivität bei verschiedenen versionen eines memory-spiels zum test adaptiver hilfestellungen, 2019.

[Mir18] R. Miranda. Recognising & dynamically adapting to visual and memory-based hci obstacles based on behavioural data, 2018.

[Pas01] George Paschos. Perceptually uniform color spaces for color texture analysis: An empirical evaluation. *Image Processing, IEEE Transactions on*, 10:932 – 937, 07 2001.

[SP18a] Mazen Salous and Felix Putze. Behaviour-based working memory capacity classification using recurrent neural networks. In *ESANN 2018 – 26th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 159–164, Brugge, Belgium, 2018.

[SP18b] Mazen Salous and Felix Putze. Behaviour-based working memory capacity classification using recurrent neural networks. 04 2018.

[SPIS19] Mazen Salous, Felix Putze, Markus Ihrig, and Tanja Schultz. Visual and memory-based hci obstacles: Behaviour-based detection and user interface adaptations analysis. 10 2019.

[WLW+16] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy. Training deep neural networks on imbalanced data sets. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 4368–4374, 2016.