BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

AUTOMATIC GRAPHICS TOOL SELECTION USING FREE-HAND SKETCHES

AUTOMATICKÁ VOLBA GRAFICKÉHO NÁSTROJE POMOCÍ RUČNĚ KRESLENÝCH SKIC

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **RICHARD HARMAN**
AUTOR PRÁCE

SUPERVISOR **doc. Ing. MARTIN ČADÍK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

147457

Institut:	Department of Computer Graphics and Multimedia (UPGM)
Student:	**Harman Richard**
Programme:	Information Technology
Specialization:	Information Technology
Title:	**Automatic Graphics Tool Selection Using Freehand Sketches**
Category:	Computer Graphics
Academic year:	2022/23

Assignment:

1. Create an overview of the related work and select the best possible approach for recognition and placement of graphics tools (such as cursors and arrows) based on freehand sketches. List existing literature and software.
2. Design the sketches which will correspond to the graphics tools. The tools will be provided by the supervisor and will include at least a cursor, cursors connected with line segments into a polygon, and an arrow.
3. Collect the data necessary to train a sketch classifier.
4. Experiment with the classifier training procedure to get the best possible results.
5. Create a demo application which will recognize the sketches and will place the graphics at the position of the sketch.
6. Discuss the pros and cons of the selected approach and propose what could be improved in the future work.

Literature:

1. Zhang, S. Liu, C. Zhang, W. Ren, R. Wang, and X. Cao, "SketchNet: Sketch Classification with Web Images," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 1105-1113, doi: 10.1109/CVPR.2016.125.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Čadík Martin, doc. Ing., Ph.D.**
Head of Department:	Černocký Jan, prof. Dr. Ing.
Beginning of work:	1.11.2022
Submission deadline:	10.5.2023
Approval date:	31.10.2022

Abstract

This thesis works on classifications of sketch representations of telestration tool. This classification is being developed for telestration application as part of the free-hand drawing section. After choosing the sketch representations, collecting data with specifically created application I created two datasets. First dataset was used to train a convolutional neural network to classify the sketches. Second dataset was used to train segmentation neural network to classify lines and ellipses in sketches. Both networks was then implemented into application to showcase their usage in real-time and experiment on their accuracy. This application also contains post-processing process to recreate the telestration tool representations from the sketches.

Abstrakt

Táto práca je zameraná na klasifikáciu skíc reprezentujúcich telestračné nástroje. Táto klasifikácia bola vytvorená pre telestračnú aplikáciu ako súčasť sekcie na kreslenie voľnou rukou. Po vybratí skíc reprezentujúcich nástroje, zozberaní dát za pomoci aplikácie, ktorú som vytvoril na tento účel, som vytvoril dva datasety. Prvý dataset bol použitý na trénovanie konvolučnej neurónovej siete na klasifikáciu skíc. Druhý dataset bol použitý na trénovanie segmentačnej neurónovej siete pre rozlíšenie línií a elíps v skici. Obe siete boli implementované do aplikácie na ukážku ich funkcionality v reálnom čase a zároveň pre experimentovanie a zisťovanie ich presnosti. Táto aplikácie tiež obsahuje proces dodatočného spracovania, vďaka ktorému vie reprodukovať reprezentácie telestračných nástrojov zo skíc.

Keywords

sketch classification, automatic tool selection, image segmentation, convolutional neural network, computer vision, image processing, human – computer interaction

Kľúčové slová

klasifikácia skíc, automatický výber nástrojov, segmentácia obrázkov, konvolučné neurónové siete, počítačové videnie, spracovanie obrazu, interakcia človek – počítač

Reference

HARMAN, Richard. *Automatic Graphics Tool Selection Using Freehand Sketches.* Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Martin Čadík, Ph.D.

Rozšírený abstrakt

Táto práca sa zaoberá vytvorením automatického nástroja na spracovanie skíc telestračných nástrojov. Tento nástroj je implementovaný pre telestračnú aplikáciu, ktorá slúži na športové analýzy pre športových analytikov a trénerov. Táto aplikácia obsahuje priestor na kreslenie do športových záznamov voľnou rukou, v ktorom neurónová sieť bude rozoznávať určité skice užívateľov a následne ich bude aplikácia meniť na funkcie naimplementované v aplikácii. ktoré tieto skice reprezentujú.

Konvolučná neurónová sieť bude rozoznávať skice reprezentujúce telestračné nástroje. Tieto skice boli vybrané po zozbieraní návrhov od malej skupiny užívateľov, podľa zhody viacerých užívateľov. Následne pomocou mnou vytvorenej aplikácie na zbieranie dát boli zozbierané dáta od 97 účastníkov zberu dát. Dáta obsahovali nakreslené skice, aj informácie o ich kreslení. Tieto dáta sa následne použili na vytvorenie dvoch datasetou.

Prvý dataset bol vytvorený na trénovanie konvolučnej neurónovej siete pre rozoznávanie ktorú telestračnú funkciu skica reprezentuje. Druhý dataset slúži na trénovanie segmentačnej neurónovej siete, ktorá je potrebná pre správne vytvorenie funkcie zo skice.

Segmentačná neurónová sieť sa používa na získanie pozícií a veľkostí objektov tvoriacich skicu, pre replikovanie funkcie na rovnakých pozíciách. V prvej verzii segmentačnej siete, bol pokus o získanie týchto informácií pomocou segmentačnej masky obsahujúcej smery ťahu na kreslených pixeloch v obrázku. Táto sieť bola trénovaná na datasete vytvorenom zo zozbieraných dát.

Pre experimentovanie a ukážku funkčnosti natrénovaných modelov oboch neurónových sietí som vytvoril druhú aplikáciu. Po nakreslení skice reprezentujúcej jednu zo 7 funkcií vybraných pre túto prácu v danej aplikácii, model konvolučnej neurónovej siete klasifikuje danú skicu a určí ktorú funkciu má táto skica reprezentovať. Následne model segmentačnej siete vytvorí masku, ktorá prejde procesom dodatočného spracovania a následne aplikácia podľa týchto spracovaných dát pretvorí reprezentáciu klasifikovanej funkcie na rovnakej pozícii ako sa nachádzala skica.

Po prvej sérii experimentov som zistil, že segmentačná sieť nevytvára dostatočne presné masky, na to aby sa dali spracovať. Na porovnanie som zapisoval body v aplikácii a vytváral tieto smerové masky v aplikácii. V prípade masiek vytvorených aplikáciou, proces spracovania bol funkčný a aplikácia dokázala zreprodukovať funkcie zo skíc.

Následne som vytvoril druhú verziu segmentačného datasetu, ktorý rozoznával elipsy a línie v skice. Po natrénovaní tejto siete na novom datasete, som model tejto siete vložil do aplikácie namiesto starého nefunkčného modelu a pomocou malých úprav procesu dodatočného spracovania bola aplikácia schopná zreprodukovať funkcie zo skíc pomocou oboch neurónových sietí.

V tejto aplikácii boli následne zreprodukované experimenty, ktoré ukázali, že druhá verzia segmentačnej neurónovej siete vytvorila dostatočne presné masky na zreprodukovanie funkcií vo všetkých prípadoch a konvolučná neurónová sieť klasifikovala skice s úspešnosťou 99.64 %.

Aktuálne má aplikácia možnosť spracovania skíc dvoma spôsobmi. Prvý je pomocou smerových masiek vytvorených aplikáciou a druhý je pomocou modelu segmentačnej neurónovej siete.

Automatic Graphics Tool Selection Using Free-hand Sketches

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of doc. Ing. Martin Čadík Ph.D. and consultants Ing. Jan Brejcha Ph.D. and Mgr. Dominika Trebatická from ChyronHego Corporation. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Richard Harman
May 8, 2023

Acknowledgements

I would like to thank my supervisor doc. Ing. Martin Čadík Ph.D. and both of the consultants, Ing. Jan Brejcha Ph.D. and Mgr. Dominika Trebatická from ChyronHego Corporation, for guiding me through the process of creating this thesis, giving me advice along the way and passing on their knowledge.

Contents

Chapter 1

Introduction

Hand-drawn sketches have long been used as a method of conveying ideas and concepts in various fields, including architecture, engineering, and design. However, the process of converting these sketches into digital forms, such as CAD drawings or 3D models, can be time-consuming and error-prone. In recent years, advances in machine learning, particularly neural networks, have led to the development of tools that can automatically interpret and convert hand-drawn sketches into digital forms.

My work will focus on sketches of sports analysts and coaches and convert them to basic telestration tools needed for sport analysis. This tool is planned to be used as an input method for the telestration software created by *ChyronHego Corporation*, which will ease the work for users, because they will not need to activate the functions manually. Usual users of these tools are coaches and sports analysts, so I will try to incorporate as many users from sporting background as possible for both data collection and testing.

This thesis will focus on the use of machine learning techniques for the automatic selection of the most appropriate graphical tool based on the interpreted hand-drawn sketches. The thesis will investigate current state of the art in sketch recognition and graphical tool selection, and will propose the best method for automatic selecting the most appropriate tool for a given sketch.

The goal of this thesis is to develop a prototype of a system which will be able to convert hand-drawn sketch representation of telestration tools chosen for this thesis, into telestration tools. The thesis will start by picking neural network architecture that will provide sketch recognition as accurate as possible.

Creation of the dataset will also be an important part of completing this task, since the sketched representations of the telestration tools will be picked, and there are not any datasets with these specifically picked sketches in existence. Different types of data will be collected from users who then will go through the pipeline to create a dataset for training a chosen neural network.

Chapter 2

Related work

2.1 Sketch Classification

Works on sketch recognition and classifications date back to the development of Sketch-Pad [16], a device that made it possible to transfer hand-made drawings from a computing device, thus recording the first sketches. Since then, there have been a lot of different approaches to recognize different types of sketches.

Some sketch classification projects focused on hand-written words, others on sketches or doodles of real-life objects. Khatri et al. [6] focused on creating a neural network able to recognizing hand-written numbers via learning vector quantization abbreviated. They deconstructed their input images into vectors and trained the neural network to recognize the numbers based on these vectors. They found out the problems with recognition of this type is usually due to different hand-writing styles of users.

Yuan and Jin [18] created an Intelligent Whiteboard used as a tool for Computer Supported Cooperative Learning and measured an efficiency of sketching in the learning process and total usability of this type of communication with a device. They mainly used detection of strokes and post-processing features to recreate sketches into UML class diagrams, digital circuits, chemical diagrams and many other things. For post-processing process, they detected certain feature points in strokes and from these points they recognized primary symbols they chose, which inspired me in training my Stroke recognition neural network in the Section 4.2.2 and Section 4.3 and my post-processing in Section 4.6.

There was one work that tried to develop a sketch feature representation named *Sketch-Net* [19], which tried to recognise sketches based on their real web images as seen in the Figure 2.1.

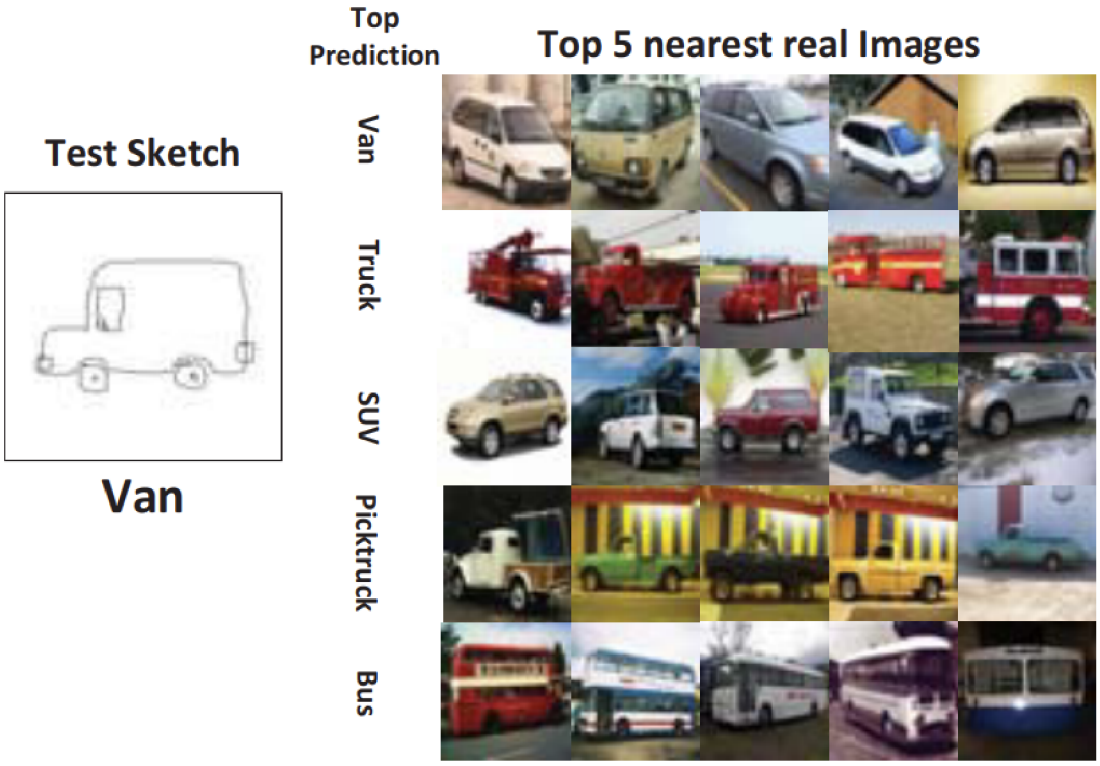


Figure 2.1: Examples of correctly classified sketches on a test case with real images as references. The first column shows the test sketch, and the top 5 category predictions are displayed from the top row to the bottom, in sequence. Specifically, each row of real images represents the nearest visually similar images in its category. This Figure is from *SketchNet* thesis [19].

This was not applicable to my thesis because the tools that I will classify will have only a symbolic representation, and it is not a sketch mirroring the exact look of real-life figure.

One of the biggest open-source datasets created in sketch recognition history was done by Google Inc. named *Quick, Draw!* consisting of more than 50 million sketches divided into 345 categories with size of 28 by 28 pixels, contributed by players playing a game with the same name. Many works have been done with this dataset. One of them was focused on training the Recurrent neural network to learn sketch abstractions for reconstruction, expanding and finishing sketches based on input unfinished sketch images [4]. There were other works interested in sketch classification and since sketch classification is my main goal, I was interested in those.

One of image classification works realised on *Quick, Draw!* dataset was done by Tran and Lu [17] who architected Deep Convolutional neural network for sketch classification based on residual neural network architecture, also known as *ResNet*. They found an issue with hard recognition with similarly looking classes, for example, classes `Pigeon`, `Seagull` and `Standing bird`. Difference in hand-written drawing styles of users made it difficult to recognize these similar classes. Based on this knowledge, Guo et al. [3] tried to use K-Nearest Neighbour Algorithm with K-Means++ initialization and weighted voting where they created centroid of sketched points and then weighted the points according to the distance from the centroid. They compared this technique with Convolutional neural

network, only to find out that Convolutional neural network outperformed their algorithm and stayed the best technique for image classification at the time.

There are other studies done with Deep Convolutional neural networks. An extensive dataset created by Eitz et al. [2], usually mentioned in the studies with the name *TU-Berlin*, is used a lot as a benchmark across many studies focused on sketch classification. The *TU-Berlin* dataset consists not only of full classified sketches, but also of partially drawn sketches.

Seddati et al. [14] created a system for the sketch classification and similarity search using the *TU-Berlin* dataset. This study proposed a Deep Convolutional neural network named *ConvNet* that classifies the images, and after the classification it also performs features extraction from the sketch images. On those features they use k-Nearest neighbours algorithm for similarity search in the sketch image classes. Their architecture has fewer and smaller kernels than previous attempts and got 75.42 % accuracy on the *TU-Berlin* dataset, overcoming all previously tested architectures and also humans which had 73 % accuracy of classification on this dataset. Seddati et al. [13] also created a second version of their architecture where they proposed the multi-task learning of their neural network. New version of *ConvNet* is trained for predicting the object category, but it also predicts the completeness of the sketch. With this approach they achieved 77.69 % accuracy on the *TU-Berlin* dataset, outperforming their previous version of *ConvNet*. These two networks developed by Seddati et al. are also being referred to as *DeepSketch* and *DeepSketch2*.

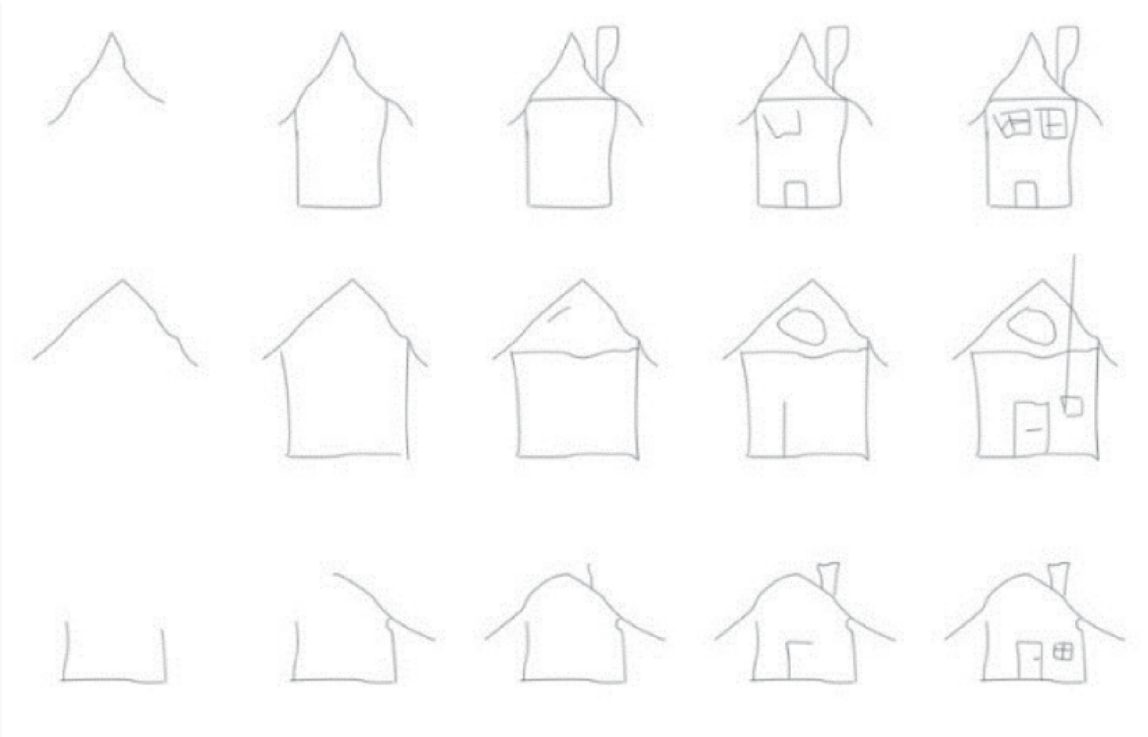


Figure 2.2: Examples of partially drawn versions of the sketches classified as a class named *House* from training of *DeepSketch2* [13].

By discussing problems with classification, which mainly consisted of different hand-drawing styles and similar looks of complex sketches, I concentrated on the problems that might occur in my work. The data collected by me will have a wide variety of users from

different backgrounds to capture as many hand-drawing styles as possible. Sketches of the classes used in my work will all be simple and have easily recognizable differences which, it is hoped, will eliminate the issue of high similarity of complex sketches.

2.2 Classification Neural Network

Neural network architecture used for this thesis is residual network with 18 hidden layers named *ResNet-18* created by He et al. [5]. Residual networks were created to ease the training of deep networks. This network was first to have layers of learning residual functions with a reference to the layer inputs, instead of learning unreferenced functions. The *ResNet-18* had smaller error rate than plain state-of-the-art method at the time. With networks that had more layers, this gap between errors of *ResNet* architectures and other architectures got even broader which can be seen in the Figure 2.3.

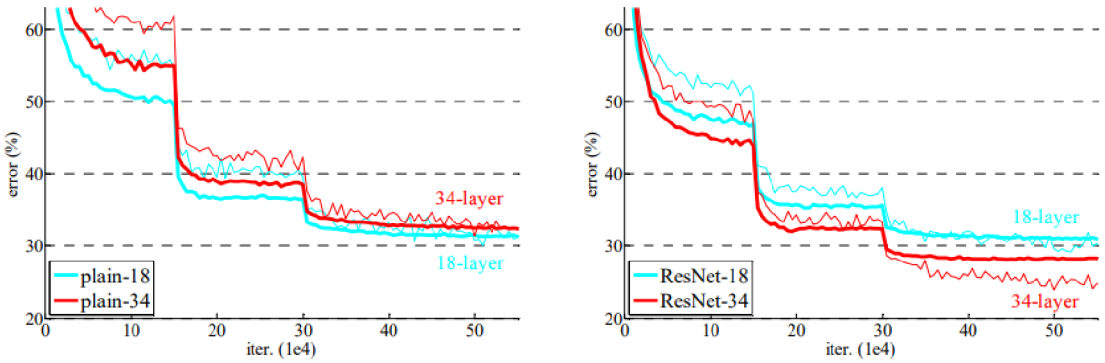


Figure 2.3: Experiments tried with *ResNet-18* and *ResNet-34* by He et al. [5], while training on *ImageNet* dataset [1]. Thin curves denote training error, and bold curves denote validation error of the centre crops. Plain networks of 18 and 34 layers (left). *ResNets* of 18 and 34 layers (right). In this plot, the residual networks have no extra parameter compared to their plain counterparts. Figure is also taken from the work of He et al. [5].

For training this neural network I used a *Cross-Entropy* loss function which originates from Rubinstein [12]. Loss function of neural network penalizes wrong outputs of neural network and helps it learn. The *Cross Entropy* builds upon the idea of entropy from information theory and calculates the number of bits required to represent or transmit an average event from one distribution compared to another distribution. The *Cross entropy* loss function measures the difference between the discovered probability distribution of a machine learning classification model and the predicted distribution. It is usually used for models where there are three or more classification possibilities, but there is also binary version of this loss function. For this thesis, I used a multi-class version of this loss. Calculating the *Cross Entropy* across multiple classes is done according to this equation:

$$H(P,Q) = -\sum_{x=1}^{N} P_x * \log(Q_x)$$

Where:

- $H(P,Q)$ is the *Cross Entropy* function between the two probability distributions P and Q

- P_x is the probability of the event x in the distribution P
- Q_x is the probability of the event x in the distribution Q
- log is the base-2 logarithm, meaning that the results are in bits

This calculation is for the discrete probability distributions although a similar calculation can be used for continuous probability distributions using the integral across the events instead of the sum. The result will be a positive number measured in bits and will be equal to the entropy of the distribution if the two probability distributions are identical.

To optimize learning of the neural network I used *Adam* optimizer developed by Kingma and Ba [7]. The *Adam* optimizer is an efficient method of stochastic gradient-based optimization that only requires first-order gradients with little memory requirement. The name *Adam* is derived from adaptive moment estimation. The optimizer is used for changing the attributes of the neural network while learning.

2.3 Segmentation Neural Network

The Segmentation neural network used in this work is firstly used to try to recognize the directions that the strokes in the sketches have been drawn in, and later this network is used to classify the lines and the ellipses in the sketch for recreating of sketched functions.

In the history of working with sketch strokes there is one work that inspired me in choosing my network. Qin [9] presented an intelligent method for classifying pen strokes in an on-line sketching system. This method used linearity and convexity of curves to identify curves, lines, circles, ellipses, and many other types of curves. I wanted to achieve similar goal using the neural network.

I chose the segmentation neural network for image segmentation to extract certain features from my sketches. I used *U-net* architecture developed by Ronneberger et al. [10], originally used for Biomedical Image Segmentation. This network was an improvement of the original version done by Shelhamer et al. [15]. The *U-net* is a convolutional neural network modified and extended to work with fewer training images in order to reach higher segmentation accuracy. Its architecture is symmetric and consists of two main parts and it can be seen in the Figure 2.4. Left part is called a contracting path, constituted by the general convolutional process. Right part is constituted by transposed 2D convolutional layers. Image is an input and output is a segmentation map.

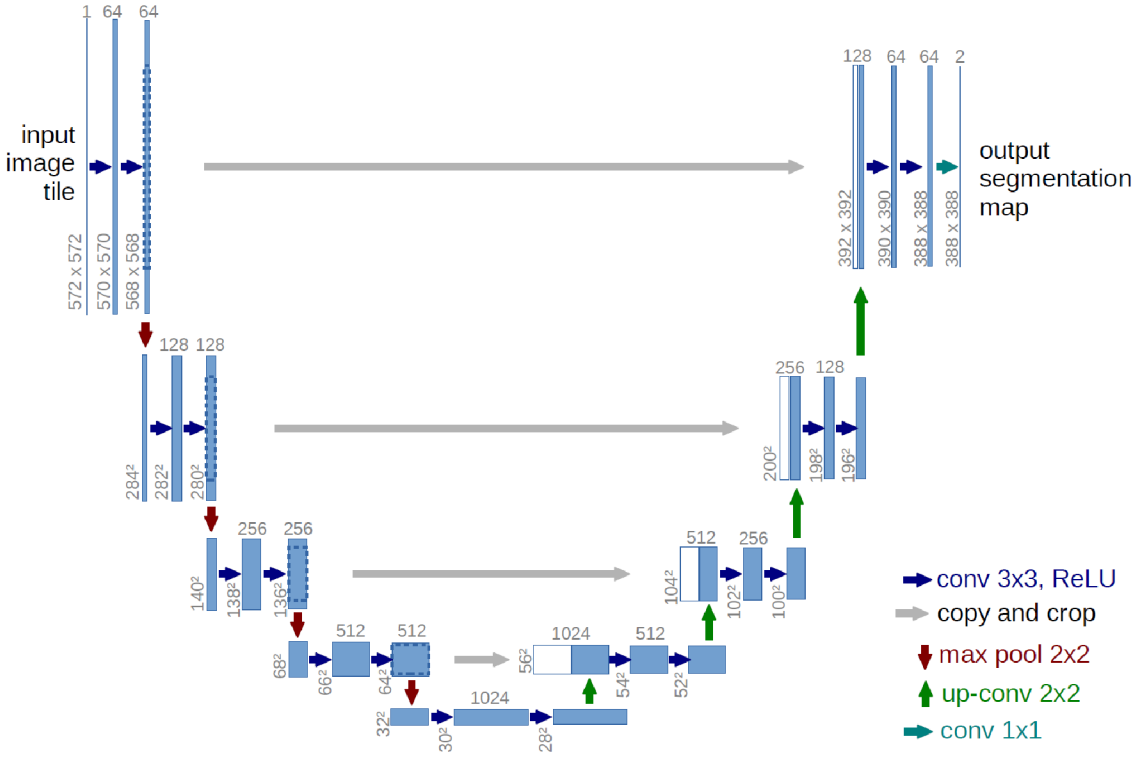


Figure 2.4: U-net architecture. Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on the top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. Figure from Ronneberger et al. [10].

For learning of this neural network I used the multi-class *Cross Entropy* loss function with weighted classes and *Adam* optimizer function. Both are mentioned and explained in the Section 2.2.

2.4 Post-processing Functions

The post-processing realised in this thesis is inspired by work of Yuan and Jin [18] in which they recognized features of different objects selected for their work. For my work, I needed to recognize lines and ellipses in the sketches and then in their positions recreate the functions these sketches are meant to represent.

2.4.1 Digital Difference Analyzer line algorithm

For connecting points that were not directly next to each other, when creating the first version of the segmentation dataset mentioned in the Section 3.3.2, I used the *Digital Difference Analyzer* (DDA) line algorithm as mentioned in Žára [20]. This algorithm is used for an interpolation of variables over an interval between the start and the end point. It has only one cycle and each new coordinate is calculated based on the last calculated coordinate. We calculate new values until we reach the end point coordinates.

Input of the DDA line algorithm:

$$Starting\ coordinates = (X_0,\ Y_0)$$

$$Ending\ coordinates = (X_n,\ Y_n)$$

Equations to calculate for each X coordinate:

$$X_i = X_{i-1} + 1$$

$$Y_i = Y_{i-1} + m$$

Where m is calculated with this equation:

$$m = \frac{Y_{end} - Y_{start}}{X_{end} - X_{start}}$$

2.4.2 K-means clustering and Silhouette analysis

When post-processing the points from the segmentation neural network, in some classes I needed to create clusters from the points to recognize ellipses from all the points retrieved from the sketch and to recreate them in those positions. For this purpose, I used *K-means clustering* developed by MacQueen [8]. This process is used for similarity grouping or clustering. It classifies the points into K number of clusters based on the nearest cluster centroid, serving as a prototype for the final cluster. For this process to work properly, I needed to know how many ellipses are in each sketch, since *K-means clustering* requires an input of number of clusters to divide the points into which in my case means, how many ellipses I want to divide the points into. To find out the number of clusters I used the Silhoutte analysis developed by Rousseeuw [11]. The silhouette analysis measures of how close each point in one cluster is to the points in the neighbouring clusters, and thus provides a way to assess parameters, as, for example, number of clusters, visually. This measure has a range of [-1, 1]. The name for this measure is silhouette coefficient. The silhouette coefficient near value of +1 indicates that the sample is far away from the neighbouring clusters. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighbouring clusters, and negative values indicate that those samples might have been assigned to the wrong cluster. To find the best number of clusters from the points, I calculated the Silhouette coefficient for different number of clusters on the same points, and the number of clusters with the highest coefficient is then chosen as the final number of clusters for *K-means clustering* algorithm.

Chapter 3

Dataset

3.1 Choosing data

Simple way to represent sketches is by their images, but there are many other features that might help me to find a new way of sketch classification. I picked these features to gather:

1. Images of sketched area in `.png` format
2. Timestamps of when each pixel of the sketch was drawn
3. Colour of each pixel of the sketch
4. Position coordinates of each pixel of the sketch
5. Position of when stroke changed its direction

Then I store the data about users who participated in the data collection process:

1. Age
2. Biological gender
3. Occupation
4. Relation to sport

All of the data will be stored in *JSON* format for its easy translating to python dictionaries.

3.1.1 Selecting representing symbols

Tools that I will try to convert into functions will be symbolised by classes, each function will represent one class with the same name. These classes needed their own symbol representation that had to be chosen before the beginning of data collection. These symbols were chosen by 10 participants who all created their own ideas for symbol representation of each function. For the purpose of my work I chose 7 functions, that I will classify and translate. The names of these functions are `Arrow`, `Cursor Light`, `Cursor Linked`, `Cursor Linked Closed`, `Cursor Player`, `Light Shaft` and `Zone Polygon`. Their visual design representations can be found in Appendix A. After collecting the ideas on symbol representations from the participants, I selected one symbol for each tool. I chose the most

occurring representations from sketches I gathered from participants in this collection. Example sketches of these symbols can be found in Appendix B.

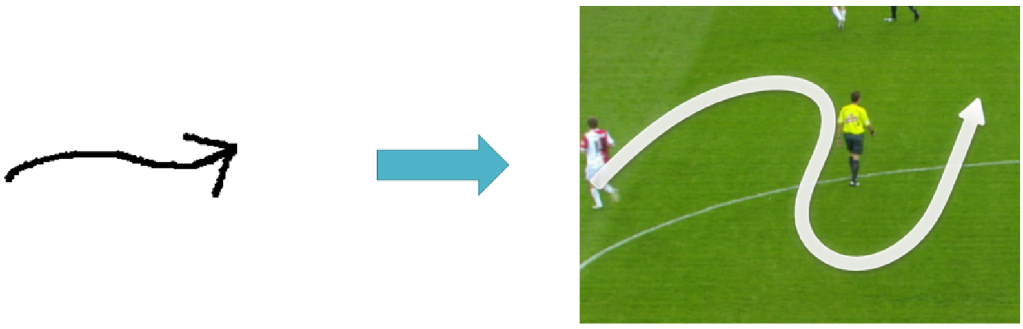


Figure 3.1: Sketched symbol (left) representing function (right) named `Arrow`.

3.2 Dataset collection

3.2.1 Subjects

The data was collected from 97 participants from different backgrounds and ages from 6 years to 73 years, to comprehend a big diversity of drawing styles. The participants mainly consisted of students who contributed to 76 % of all the data collected as seen in Table 3.3, and they were between the ages 18 and 22 years. Also 72 % of the participants were sport oriented as seen in Table 3.2. Participants divided into their biological genders can be seen in Table 3.1.

Biological gender	Count
Male	71
Female	26

Table 3.1: Number of participants in dataset collection by their biological gender.

Relation to sport	Count
True	70
False	27

Table 3.2: Number of participants in dataset collection by their relationship with sports.

Occupation	Count
Student	74
Teacher	4
Pension	4
Driver	3
Entrepreneur	2
Football player	2
Soldier	1
Officer	1
Football team manager	1
Entrepreneur	1
Headmaster	1
Mechanic	1
Medic	1
Recruiter	1

Table 3.3: Number of participants in dataset collection by their occupation.

3.3 Sorting datasets

The data collected from participants consists of 1437 images including 7 classes mentioned in Section 3.1.1. Each class contains around 200 images with the data about every particular image: different strokes, coordinates of points coloured, colour of points and timestamp when each of them was coloured. At the end of the data collection, I created two different datasets. One dataset for a classification model which will estimate the class of an image and one dataset for a segmentation model which will estimate the direction of strokes from an image. The direction of the stroke will be needed for post-processing and recreation of the tool from the sketch.

3.3.1 Classification dataset

I created the ground truth of the classification dataset by classifying all images collected from the participants and augmenting each image by flipping each image on x axis or y axis or both axes randomly. The images, originally without a background, were assigned white background, since strokes were coloured with darker colours, and on white background they will be easily recognisable. On original images, different strokes had different colours, but for the case of the classification I changed all strokes to black colour. Then I resized each image to half of the original size, which is 960 pixels width and 540 pixels height, so the images will take less space and training will be done faster.

Figure 3.2: `Cursor Linked Closed` sketched symbol before (left) and after (right) colour correction. Strokes are different colours before the correction, to illustrate the difference in strokes in an image.

The images were divided into three separate folders for training, validating and testing the accuracy of the model. I randomly picked 7 participants whose images were later used for the validation after each training epoch and 7 for testing of the model. The rest of the images were used for training. The random selection was done by using Python's library named *random* with function *sample* that has a uniform distribution. It selected 10 unique numbers from the list of all user identification numbers, I split them into the validation set and the test set, and then saved them in the a configuration file. The final version of classification dataset consists of 1437 images before augmentation.

	Arrow	Cursor Light	Cursor Linked	Cursor Linked Closed	Cursor Player	Light Shaft	Zone Polygon
Train	187	187	189	184	184	183	183
Validation	10	10	10	10	10	10	10
Test	10	10	10	10	10	10	10

Table 3.4: Number of images before augmentation in Classification dataset divided into corresponding classes.

The final version of classification dataset after augmentation consists of 2874 images.

	Arrow	Cursor Light	Cursor Linked	Cursor Linked Closed	Cursor Player	Light Shaft	Zone Polygon
Train	374	374	378	368	368	366	366
Validation	20	20	20	20	20	20	20
Test	20	20	20	20	20	20	20

Table 3.5: Number of images after augmentation in Classification dataset divided into corresponding classes.

3.3.2 Segmentation dataset – first version

To recreate the functions from original sketches, I needed to recognize some basic objects as lines and ellipses in the sketch. For this purpose I wanted to train the segmentation neural network to recognize the directions of the stroke from which I would be able to recognize directional changes and objects in the sketch. To train this segmentation neural network I had to create a dataset from the data I collected.

The first version of the Segmentation dataset is created from the collected data about the points sketched into the image, and timestamps of when they were sketched. The output of my segmentation model will be a mask in *numpy* array format with the same size as the input image with the points on indexes where the points were sketched. Each of these points will be filled with 9 classes which represent the direction of the stroke on that exact point. Directions are represented in the Table 3.6.

Array index	Direction on X axis	Direction on Y axis	Visualization
0	0	0	·
1	1	0	→
2	1	-1	↘
3	0	-1	↓
4	-1	-1	↙
5	-1	0	←
6	-1	1	↖
7	0	1	↑
8	1	1	↗

Table 3.6: Directions representation by index in vector. This vector is a part of each pixel in mask of Segmentation dataset.

Since the data was collected using graphical tablet connected to the computer, and this tablet had lower resolution, some points were missing because the tablet didn't capture each pixel. The application draws the lines between two points that it captures on the tablet, and the tablet captured less points when the sketch was drawn in a short amount of time. I found out about this issue when I tried to recreate the images by creating a new image coloured on all points captured and comparing this image to the original image. If two points following one after another in the collected data were not neighbours, but were in the same stroke, I filled the line between them to connect them by adding neighbouring points from the first point until I reached the second point. Positions of the points used to fill the distance are calculated using the *Digital Difference Analyzer* line algorithm [20].

After filling the points and creating the mask for each image saved as *numpy* array, I paired images with their masks and augmented them in the same way as I did with the classification dataset mentioned in Section 3.3.1, by flipping them. Then all the images and masks are resized down from the original size of 1920 times 1080 pixels. Final images and their masks have 480 pixels width and 288 pixels height. The colour of strokes on all images was also changed to only black colour and images colour mode is set to *RGB* mode.

Recreating of images also showed me that early versions of data collecting application corrupted some logs about the points, so after blacklisting those images I got 887 images in the final dataset before augmentation and 1774 images after the augmentation. The

number of images divided by class before the augmentation can be seen in Table 3.7 and after the augmentation can be seen in Table 3.8.

	Arrow	Cursor Light	Cursor Linked	Cursor Linked Closed	Cursor Player	Light Shaft	Zone Polygon
Train	99	108	109	104	108	107	108
Validation	10	10	11	12	12	11	12
Test	8	9	10	9	10	10	10

Table 3.7: Number of images before augmentation in Segmentation dataset divided into corresponding classes.

	Arrow	Cursor Light	Cursor Linked	Cursor Linked Closed	Cursor Player	Light Shaft	Zone Polygon
Train	198	216	218	208	216	214	216
Validation	20	20	22	24	24	22	24
Test	16	18	20	18	20	20	20

Table 3.8: Number of images after augmentation in Segmentation dataset divided into corresponding classes.

I also tried to cut out only drawn parts of the image, so the drawn part would fill the whole image. I wanted to see if this option would be better for segmentation training. After some experiments, I found out that it creates more errors in segmentation training, so this option was turned off during the final training of segmentation model and the full image was used for training.

3.3.3 Segmentation dataset – second version

As mentioned in the Section 3.3.2, to recreate the sketches, I needed to recognize some objects in the sketches. First approach I tried was teaching the network to recognize the directions of the strokes. This approach did not work, so I scraped the idea and started again. This time I tried to teach a neural network to recognize the lines and ellipses. For this segmentation neural network I had to create a new version of the segmentation dataset.

The second version of Segmentation dataset is created from the images in the classification dataset. Information about these images are in Section 3.3.1. These images were resized to 480 pixels width and 288 pixels height. The output of my segmentation model will be a *numpy* array with the same size as the input image with the points on coordinates where points were coloured. Each of these points will be filled with 3 classes which represent 3 things I want to differentiate in an image, which are lines, ellipses and a background. Each point will fall in one of these 3 categories, which can be seen in Table 3.9.

Index	Name
0	Background
1	Line
2	Ellipse

Table 3.9: Names of classes and their indexes in vector. Vector with these indexes is a part of each pixel in mask of Segmentation dataset.

To create the mask, I need to differentiate the lines and the ellipses manually. I took the images from classification dataset and I used the fact that these images are named according to their class name, which helped me simplify the creation of their masks. Classes `Arrow` and `Zone Polygon` are only made out of lines. The masks for these classes were created by selecting class index 1 on each pixel corresponding to drawn pixels in the image. Class `Cursor Player` is only made out of ellipse. With these classes I did not have to differentiate lines and ellipses. Mask for this class was created by selecting class index 2 on each pixel corresponding to drawn pixels in the image. Other classes are made out of both lines and ellipses, so, to create their masks, I needed to differentiate lines and ellipses in the image.

To differentiate the lines and the ellipses I saved all positions of the pixels that are a part of the sketch in the image. Finding ellipses in the image is done using *OpenCV* library tools. I found circles in those images by finding inner contours in them. To find contours, I loaded the picture, changed the color scheme to grey-scale and applied thresholds from *OpenCV* library named `THRESH_BINARY` and `THRESH_OTSU`. Then I used the *findContours* function inputting the thresholded data with the Retrieval mode of the function set to `RETR_TREE` and the Method of the function set to `CHAIN_APPROX_SIMPLE`. I chose only inner contours from the hierarchy.

I took the sketched pixels positions and compared them to the contours positions and then set the mask index to 2 on all pixels that were close to the inner contours. All the sketched pixels remaining after that were set to index 1.

Inner contours are outlines of a curving or irregular figure. In class `Cursor Linked Closed` there was an issue because the inner outline of the whole sketched function also had its inner contour. To filter these contours, I compared the width and height of the contour to the width and height of the whole sketch. If the contour was above half of the height or half of the width, I filtered it out, since the ellipses in class `Cursor Linked Closed` should not be half the size of the whole sketch.

Since the images in this dataset are all from the classification dataset, they are already augmented. Because of necessity of recognizing of the ellipses in some classes, I had to manually delete some images from the dataset. The issue with these images was, that they had incomplete ellipses, and therefore did not have inner contours, so they were not recognized as ellipses. After manually deleting these images, the dataset consists of 2356 images with augmented images, and without augmented images it is 1178 images. Number of images divided by the class without augmentation can be seen in Table 3.10 and after the augmentation can be seen in Table 3.11.

	Arrow	Cursor Light	Cursor Linked	Cursor Linked Closed	Cursor Player	Light Shaft	Zone Polygon
Train	187	149	104	105	184	148	183
Validation	10	8	8	8	10	10	10
Test	10	6	4	5	10	9	10

Table 3.10: Number of images without augmentation in the second version of Segmentation dataset divided into corresponding classes.

	Arrow	Cursor Light	Cursor Linked	Cursor Linked Closed	Cursor Player	Light Shaft	Zone Polygon
Train	374	298	208	210	368	296	366
Validation	20	16	16	16	20	20	20
Test	20	12	8	10	20	18	20

Table 3.11: Number of images after augmentation in the second version of Segmentation dataset divided into corresponding classes.

Chapter 4

Method

4.1 Annotation tool

4.1.1 Framework

For the collection of the data I created an annotation application which helped me record all the chosen types of the data from Section 3.1. I wanted to replicate real-life usage scenarios of analysing a sport match, and based on my personal experience, analysis of sport events is usually done on devices that have some kind of touch input. I used a graphical tablet, which made it possible for users to draw by hand, thus replicating the most common usage. To use this graphical tablet, it had to be connected to a computer device.

I chose to create an application for a desktop using *Python* and *PyQt* framework. I chose them because they are great for prototyping of any kind. *PyQt* is a *Python* binding for *Qt*, which is a set of *C++* libraries and development tools providing platform-independent abstractions for graphical user interfaces. *PyQt* offers several widgets, such as buttons or menus, all designed with a basic appearance across all supported platforms and all of them are well documented. These widgets simplified the whole process of creating a desktop software which was able to record all the data needed, since *PyQt* already has a drawing widgets that I was able to use for my application. *Python* also has *JSON* library which helped me store recorded data in *JSON* format.

4.1.2 Application functionalities

The annotation application is capable of saving data I picked in Section 3.1 in their corresponding formats. The application has a list of all 7 sketch representations of the telestration tools that I want users to draw. It has their visual representation and example sketches assigned to each telestration tools name.

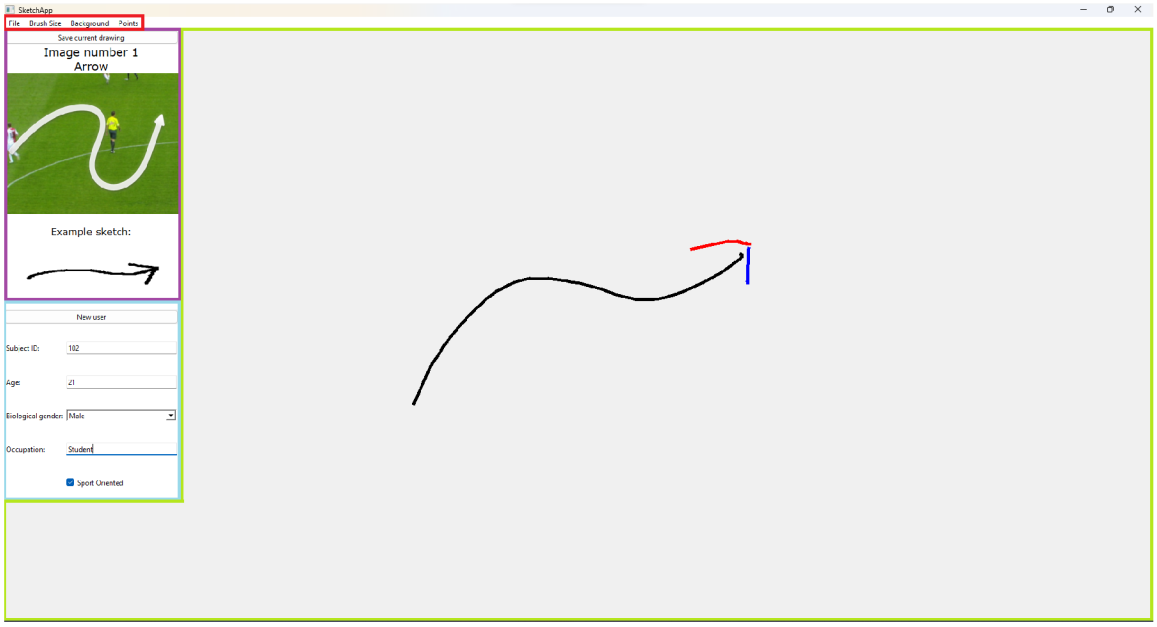


Figure 4.1: Sketching Application used for collection of the data, divided to the sections of the application with their section labels. Red border outlines the *Menu* section. Purple border outlines the *Showcase* section and light blue border outlines the *Form* section. Green border outlines the *Drawing* section.

Before the users start to draw each sketch, there is a form used to get information about the user. This form is located in the *Form* section of the Figure 4.1. Above the form, there is a button named „New user“ which calls a program function that prepares the environment of the application for a new user. The program function then saves all the data about the user that was drawing, if there was anyone drawing before. Then the program function empties the form for a new user and switches the active sketch that is supposed to be drawn to the beginning of the list of the sketches. In the *Showcase* section of the Figure 4.1 is the visualisation of the telestration tool that active sketch is meant to represent. Name of the telestration tool is written above the visualization. Below the visualization there is an example sketch of the telestration tool. There is also a button named „Save current drawing“ that saves the image of the drawing board under the name corresponding to the telestration tool sketch that was supposed to be drawn. After pressing the button to save the image, program function saves the sketch and switches the active sketch to the next sketch on the list of all 7 sketches. If the current function is at the end of the list when this button is pressed, next active sketch will be the one at the beginning of the list.

The *Drawing* section of the Figure 4.1 is the drawing area where users draw using the graphical tablet connected to the computer while the application is on. It is also possible to draw on it with a computer mouse by holding the left button of the mouse and moving around in the wanted shape. The *Drawing* section has by default a white background, but this can be changed to an image background, which is a screenshot from the football match. Background can be changed in drop-down menu named „Background“ located in the *Menu* section of Figure 4.1. Colour of strokes while drawing is automatically changed for each new stroke. There are 5 colours on a list which are always in the same order. This is used to recognize different strokes in an image even if the strokes are drawn over each other,

because when two strokes overlay, you end up with only the last one drawn on top. These colours are easily changeable, since the final image does not keep background when saving the image. Images are saved with transparent background in `.png` format and therefore stroke colours could be easily changed to any colour desired.

The *Menu* section in the Figure 4.1 is made out of drop-down menus which contains features for changing the width of the pen for drawing, changing background, manually saving an image or clearing the drawing area. Then there is also drop-down menu named „Points“ which gives an option to a user to draw out any important point it finds. Important points are the points where there is a directional change of stroke, or where the stroke starts or ends. They are important for recreating functions from their positions and recognizing lines and circles in the drawings.

4.2 Neural networks

4.2.1 Tool classification

The main part of this thesis was a classification of hand drawn sketches. After creating the Classification dataset mentioned in Subsection 3.3.1, I trained a convolutional neural network with it. I chose Residual Network architecture (ResNet) which is a deep learning model used for computer vision applications. This network architecture is described in the Section 2.2.

The loss function used for training is *CrossEntropyLoss* function with base settings and as an optimizer I used *Adam* optimizer. Both the loss function and optimizer were from the *PyTorch* library.

The dataset is split into 3 sets of data: validations set, testing set and training set as mentioned in the Subsection 3.3.1. After each training epoch, the model is validated on the validation set and has its accuracy checked on class predictions. The highest accuracy model is then saved and checked on testing data at the end of the training. Training ends after 50 epochs because, during the testing I found out that around that number, validation loss is only growing and the model starts to be over-fitted.

Every time I check the accuracy of the model on any set of data, I also print out the confusion matrix of this model. Some of these confusion matrices are visualized. The visualization is done at the beginning of the training if I found any saved model and at the end of the training, when checking accuracy of the most accurate model, chosen from the whole training. Example of visualized confusion matrix is seen in Figure 4.2.

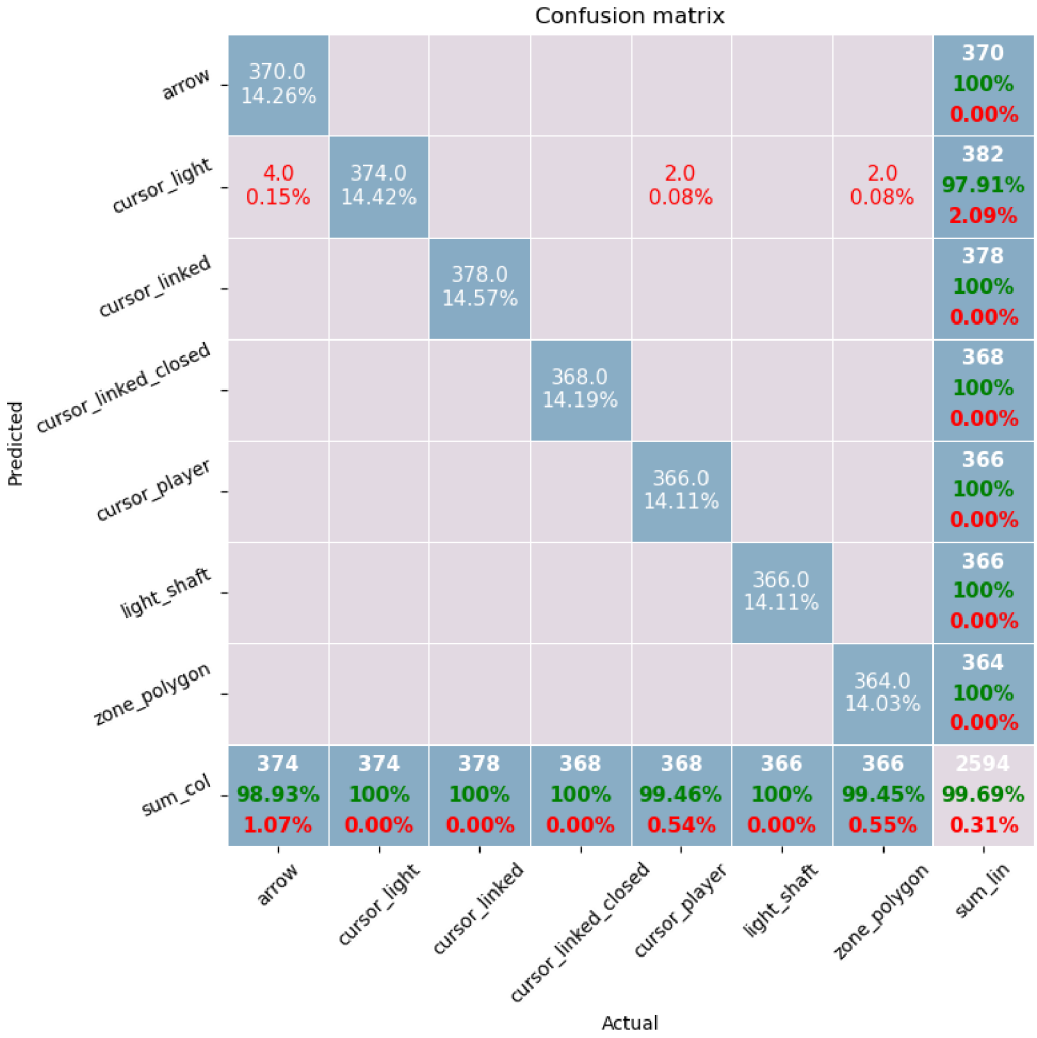


Figure 4.2: Confusion matrix on training set of images at the end of training, computed with the most accurate model of the convolutional neural network for classification. Percentages under the number of times the class has been chosen are the percentages from the total number of all predictions, based on the number of images in the set. Blue column and blue row are sums of that column or row they are part of. In this sums columns and rows, right guesses percentages are written in green color and wrong guesses percentages are written in red color. Blank cells means 0 guesses in that particular combination of predicted and actual class.

4.2.2 Strokes segmentation

Image classification was the first step of translating hand-drawn sketches into the application functions. To be able to properly transform a sketch into the function, I need to locate all parts of the sketch and filter through them, to be able to pinpoint the function in the same positions it was drawn at. I can record each point drawn with timestamps and then use post-processing based on the class from my classification network. The problem that might come up is that in the telestration application, that this work is being developed for, might not store this data or it might be unable to store so much data in such a short

amount of time or it might be unable to capture this data, which is something I can not control. While developing the new approach, I tried to find a way of how to use the images used for the image classification mentioned in the Subsection 4.2.1 and to be able to gather more information about the drawn sketch from them, for post-processing purposes. This approach will also use less memory, since both neural network will have the same image used as the input.

Locating and recreating tools just from sketches is possible only if I know the direction of the stroke and where the stroke started. I can estimate the stroke direction only if I have the order of drawn points based on the time it was drawn. Then from the data gathered I created the system of masks of each image, containing the stroke direction of each drawn pixel. The pixel in mask is filled with nine possible directions which can be seen in Table 3.6 in the Section 3.3.2.

In this case, I used an image segmentation to generate image masks containing the directions of the strokes in the image. For the image segmentation I used *U-Net* algorithm which encodes the image data to find features in the image, and then decodes the image data to generate the segmentation from the features found.

Training of segmentation model starts with checking if there is any model saved from the training before. If it exists, we save its accuracy on validation data for comparison. Each epoch is finished with checking accuracy on validation data and then comparing the accuracy to the last most accurate model to find the new most accurate model. After comparing, I visualize first two masks from the validation set and compare them to the visualizations of masks that came out of the neural network side by side as seen in the Figure 4.3.

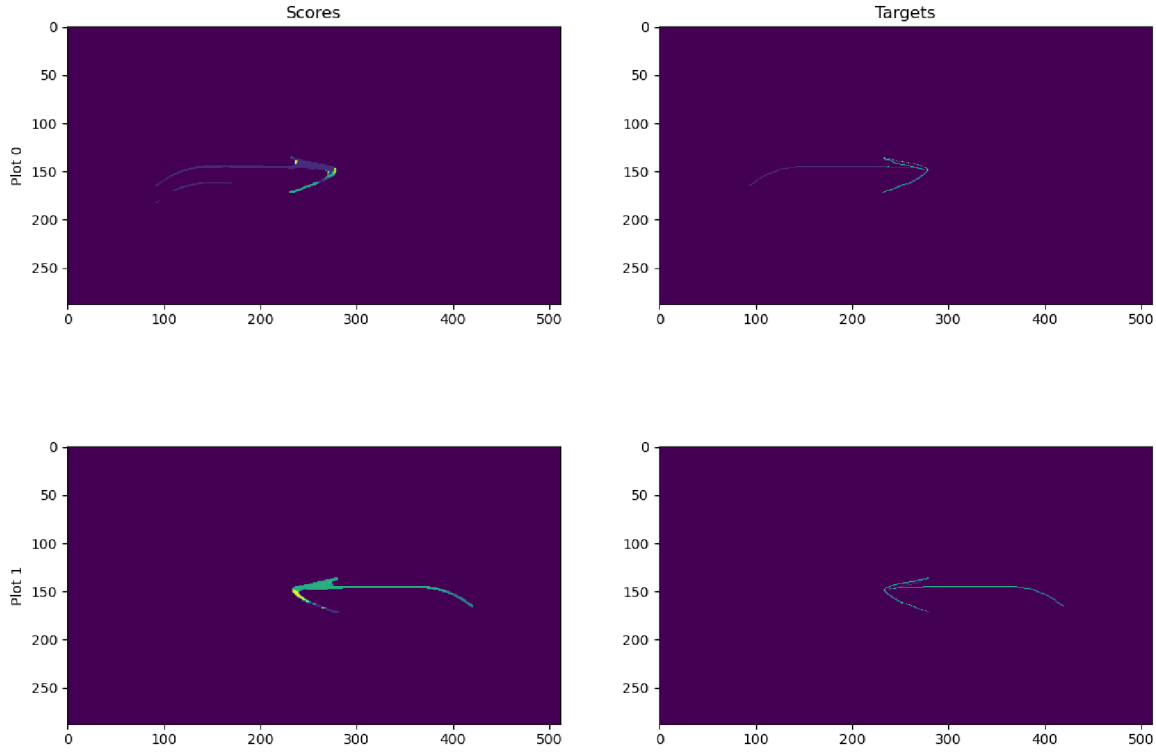


Figure 4.3: Visualization of masks, masks from the dataset (right) and segmentation neural networks estimations (left). These visualizations are created after each training epoch.

For training I used *CrossEntropyLoss* function with specified weights for each class to prevent the model from focusing on the background, since the majority of every picture does not contain any stroke or drawn pixel, and therefore the majority of the picture has no direction, which is the first class from Table 3.6. The *Tensor* array containing weights is represented in the Table 4.1.

	·	→	↘	↓	↙	←	↖	↑	↗
Weight	1	50	50	50	50	50	50	50	50

Table 4.1: Numbers representing weights for *CrossEntropyLoss* function.

At the end of the training, I check accuracy on the training set, validation set and testing set. After that I visualized the training loss and validation loss in one graph as seen in Figure 4.4. This visualization helped me set the maximum number of epochs to 25 because in the graph I can clearly see from which point the validation loss is only growing and therefore the model is over-fitted.

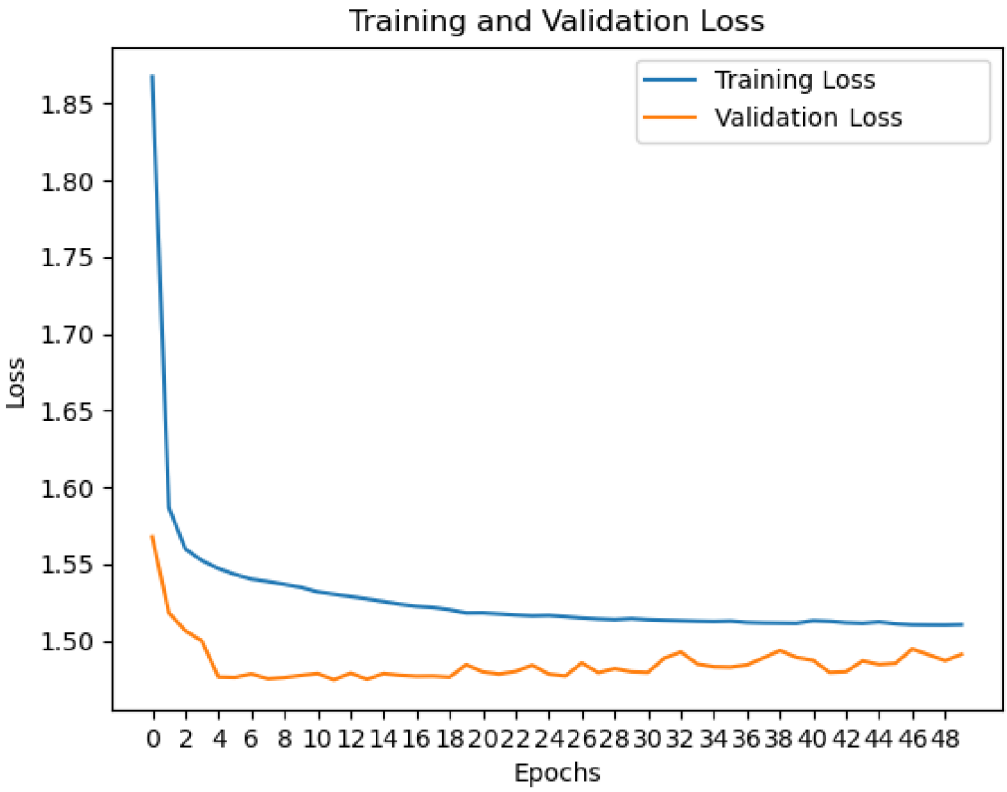


Figure 4.4: Graph of losses throughout the training of segmentation neural network.

4.3 Segmentation of lines and ellipses

After the first phase of experiments as seen in the Section 5.1, I found out that the approach of the first segmentation network described in 4.2.2 was not working. I developed a new approach with a new dataset described in the Section 3.3.3 and needed to train the segmentation neural network with this dataset. This segmentation neural network is completely based on segmentation neural network described in the Section 4.2.2, so I will only mention the differences between these networks.

This segmentation neural network divides the sketch image into three classes: Background, Lines, Ellipses. Output of this network is a mask with the same dimensions as input image, and each pixel is made out of vector representing 3 different classes. Which class the pixel belongs to is based on index in the vector. Indexes for each class are in the Table 3.9. Only one of these classes is set to number 1 on each pixel, representing which class this pixel belongs to. Other indexes are set to number 0. Visualizations done when checking the accuracy of the network are shown in Figure 4.5.

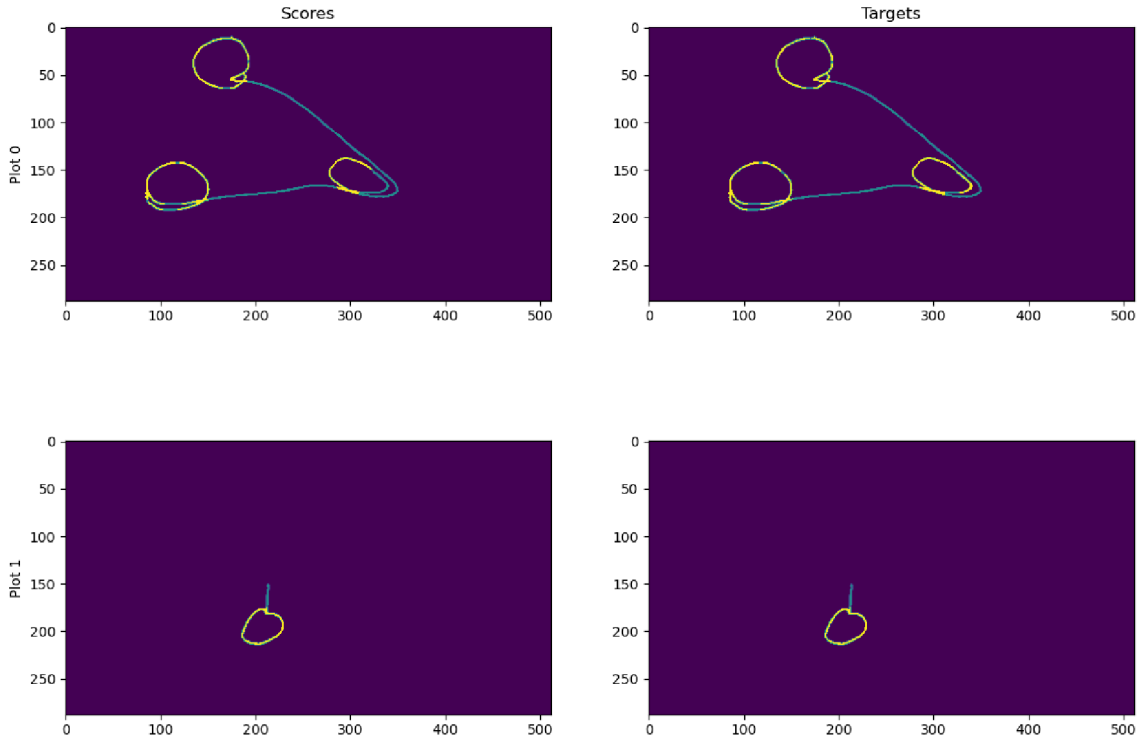


Figure 4.5: The visualization of masks, the target mask (right) and the segmentation neural networks estimations (left), pixels coloured with yellow are representing the pixels belonging to the class `Ellipse`, pixels coloured with green belong to the class `Line`, purple colour represents the class `Background`.

The *CrossEntropy* loss function *Tensor* is changed to one in Table 4.2.

	Background	Line	Ellipse
Weight	1	50	50

Table 4.2: Numbers representing weights for *CrossEntropyLoss* function based on classes in the second version of segmentation network.

The network is trained on a new dataset described in the Section 3.3.3. The number of epochs in configuration of this network was changed from 25 epochs to 100 epochs. Training and Validation losses by epoch are visualized in Figure 4.6.

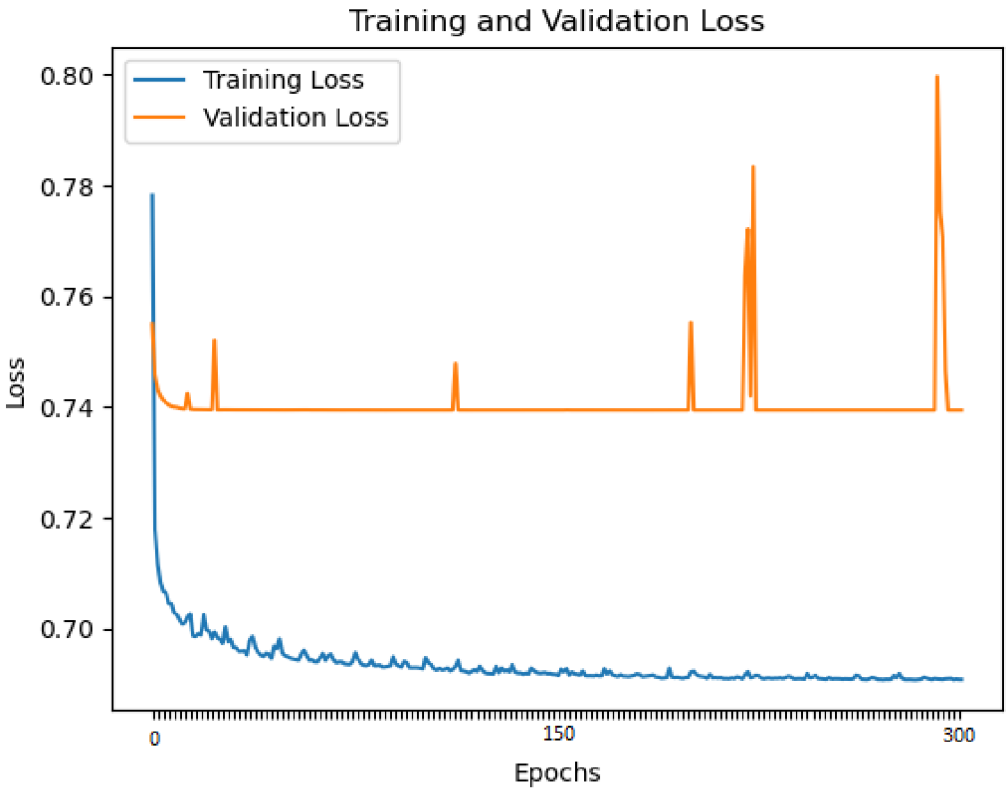


Figure 4.6: Graph of losses throughout the training of segmentation neural network. Tested while training for 300 epochs to find the best estimation of epoch to stop training the model. Validation loss is slowly declining from 0.76 to around 0.74 and training loss is declining from around 0.78 to around 0.69.

4.4 Technical details

The neural networks trained in this work were trained on the computer with `NVIDIA RTX 3060Ti` graphics card with 8 GB of memory and `Intel Core i5-12400F 2.5 GHz` Processor. I chose *Python* as my main programming language for this thesis, since it is rich with many libraries for working with artificial intelligence, sorting and visualizing data and mathematical computing. For these, I mainly used libraries *Numpy*, *PyTorch*, *MatPlotLib* and *pandas*. The convolutional neural network and segmentation neural network were both trained in *CUDA* device format.

4.4.1 Hyper-parameters of classification network

Hyper-parameters for training of the tool classification network consists of number of classes that network will recognize, which is one for each telestration tool sketch to recognize, so 7 classes together. Then a learning rate of 1^{-3} and a batch size of 8 images. Number of epochs that model will train for is 50.

4.4.2 Hyper-parameters of first version of segmentation network

Hyper-parameters for training of the segmentation neural network consists of number of classes, which in this case is 9, for each direction a pixel can have. These directions can be seen in the Table 3.6. The learning rate was set to 1^{-4} and batch size to 4 images in a batch.

4.4.3 Hyper-parameters of second version of segmentation network

Hyper-parameters for the training of this segmentation neural network are the same as hyper-parameters from the first version of the segmentation neural network, mentioned in the Subsection 4.4.2. Only change is in the number of classes, which in this case is 3, and number of epochs increased to 200 epochs. The 3 classes can be seen in the Table 3.9.

4.5 Showcasing tool

To show how the neural network models trained in this work will be used, I developed an application similar to the application I created for the data collecting. Both applications are created using the same technologies which are described in the Section 4.1.

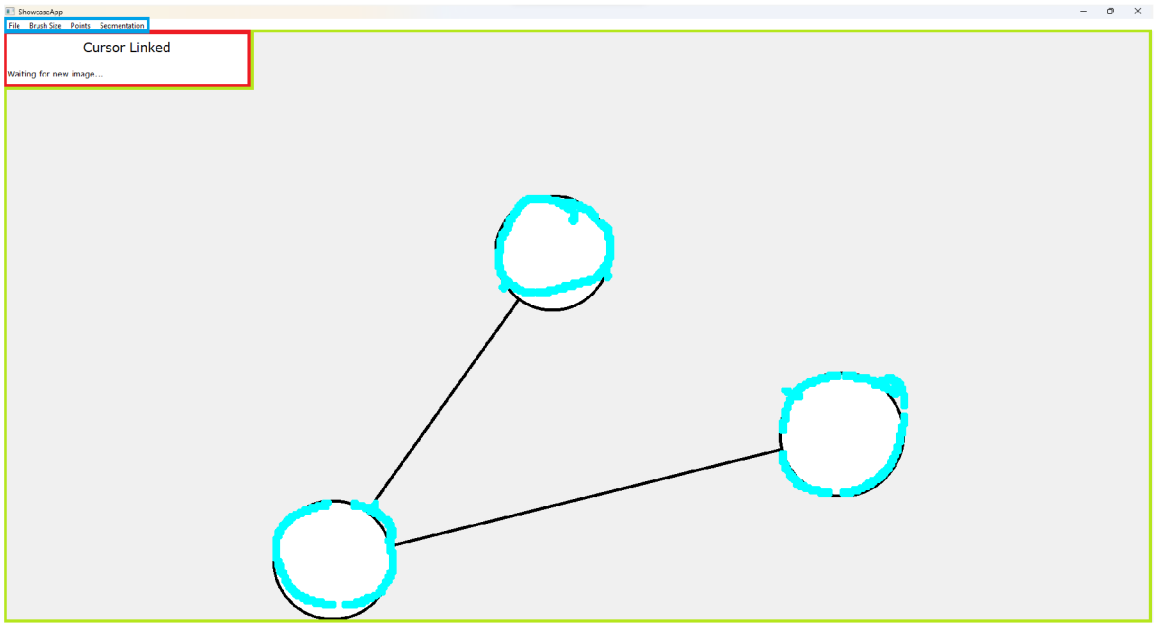


Figure 4.7: *Showcase* Application used to show the real-life usage of neural network models trained in this work. This application is divided to sections with their section labels. The Red border outlines the *Info* section. The blue border outlines the *Menu* section and the green border outlines the *Drawing* section. In the *Drawing* section, we can see a recreated telestration tool named `Cursor Linked` in black colour, and important points used for the recreation of this tool, selected by the *Showcase* application, in cyan.

4.5.1 Application functionalities

The *Showcase* application records the sketch of the user drawn in the *Drawing* section of the application from the Figure 4.7, and after the point is drawn in this section, the application

starts a time counter that is 1.5 second long. After this timer runs out of time, the drawn sketch is temporally saved as an image and this image is then used as an input for the classification neural network and also the segmentation neural network. The time of 1.5 second was chosen after going through the collected data and finding out the average times it took users to draw a new stroke in the sketch after finishing a previous stroke.

The image that is temporarily saved goes through the process of resizing and colour changing mentioned in the Section 3.3.1, the same process each image from the classification dataset went through. The modified image is then passed to the classification network. This network predicts which telestration tool the sketch is supposed to represent. The name of the predicted tool is then printed out in the *Info* section from the Figure 4.7.

The same image is then passed through the segmentation neural network which outputs the mask of the input image as mentioned in the Section 3.3.3. This mask then goes through the post-processing explained in the Section 4.6.

Post-processing is finished with printing of the representation of the telestration tool that sketch is meant to represent. When this process is finished, the application waits for the user to start drawing a new sketch. When user starts to draw a new sketch, the application clears the *Drawing* section from the Figure 4.7, which becomes empty.

The *Showcase* application also includes a *Menu* section which enables user to change the brush size, clear the drawing manually, turn off the printing of recognized points which are being printed in cyan as seen in the Figure 4.7. The application also enables users to change if the segmentation model is used for tool recreating process, or, instead the application records the points and recreates the tool with the mask the application creates itself from recorded points based on the first version of segmentation dataset explained in the Section 3.3.2. Post-processing then changes, based on, if the model was used or not. If the model was used, the application uses a new version of post-processing. If a points recorder from the application was used and the application created the direction mask from the points recorded while drawing, we use the old version of post-processing.

The information about what the application is doing at the moment is being printed out in the *Info* section from the Figure 4.7.

4.6 Post-processing

Post-processing of points has two versions based on two versions of the segmentation neural network. The second version has small adjustments to fit the changes in the masks. These adjustments are explained in the Subsection 4.6.3.

4.6.1 First version of post-processing

A direction mask outputted from the segmentation neural network contains points from the whole sketch. To recreate the telestration tool that the sketch is meant to represent, I need to filter out only the features from the image. These features will help me recognize the positions and sizes of the parts of the sketches which I will then reconstruct and print in the *Showcase* application I created. This application records sketches drawn by the user and uses both neural network models trained in this work, to reconstruct the telestration tool that sketch is meant to represent.

The direction mask, which we can see in the Figure 4.3, needs to be processed for each class of the tool differently, to retrieve needed information about the positions from the mask. Nevertheless, beginning of the post-processing process is the same for each class.

Pseudo-code for sorting algorithm created by me can be seen in the Algorithm 1. This process starts by taking all the points from the mask which is in *numpy* array format, excluding points where the direction is represented by class zero, also symbolised as „·“ in the Table 3.6. I need to find a point, where any of the strokes in the image starts. To find this starting point, I take the sides of the sketch, because I assume that points on the side will either be the end or the beginning of a stroke. I check the leftmost point and see if the direction of this point is to the right side on X axis. If it is, it means that it is a starting point for the stroke and in that case, it is my starting point and the first active point for recognizing the strokes. If the direction is different, it means the stroke did not start at that point and I check the direction of the rightmost point in the sketch and that will be the starting point. Then I search for neighbouring points to the starting point and also the active point. Those are the points that are one pixel away from the active point. The neighbouring point becomes the new active point. The point, that became inactive, is removed from the list of all points and labeled as sorted. If there are more neighbouring points for one point, then I pick the one that is in the direction of the active point, assuming they were a part of the same stroke. If there is no neighbouring point left, I assume it is the end of the stroke and after removing this end point, I again search for the rightmost and leftmost points, repeating the whole process. This creates an ordered list of points, divided to strokes. The end of the stroke is added to the list by adding the point on X and Y positions [0, 0] with direction class 0.

Program filters important points from the ordered list. Important are the points where the direction is changed, or the start points and ending points of the strokes. These points will help us split apart lines and circles and also reconstruct the telestration functions representation. A list of important points in assumed order they were drawn is then returned to finish post-processing based on the class of telestration tool they are meant to represent.

When the function which purpose is to print out the visual representations of the telestration tool into the application receives the list of important points, it creates another copy of this list without the points which represent the end of strokes. This list is used in particular cases for some reasons which will be explained in the Subsection 4.6.2.

Algorithm 1 An algorithm to sort points by strokes

```
points_list ← numpy.nonzero(segment_mask)      ▷ List of drawn pixels from the image
active_point ← starting_point                  ▷ Starting point is chosen by a function
neighbours ← empty list
empty_point ← [0, 0, 0]
while length of points_list ≠ 0 do
    for point B in points_list do              ▷ Create a list of neighbouring points
        if point B is neighbour of active_point and point B ≠ active_point then
            neighbours.append(point B)
        end if
    end for
    sorted_points.append(active_point)         ▷ Label active point as sorted
    points_list.delete(active_point)           ▷ Delete the active point from the list
    if length of neighbours ≠ 0 then           ▷ If any neighbour has been found
        Chose ← False
        for point C in neighbours do           ▷ Compare the directions
            if point C direction == last sorted_points direction then
                active_point ← point C
                Chose ← True
            end if
        end for
        if Chose ≠ True then                   ▷ If we did not found point with the same direction
            active_point ← closest point by [X, Y] position from neighbours
        end if
    else                                       ▷ Found end of the stroke
        sorted_points.append(empty_point)      ▷ Append end of stroke representation
        active_point ← new starting point      ▷ Choose a new starting point with function
    end if
end while
```

4.6.2 Post-processing for different classes

Class Arrow

If the class of tool that the sketch represents is `Arrow`, I connect the important points from the list with lines. If I find the end of the stroke in the list of points, which is represented with point on positions [0, 0] and direction of class 0 from the Table 3.6, I skip this point and create new line from the next point, which means it is a beginning of a new stroke. Recreated tool of class `Arrow` can be seen in the Figure 4.8.

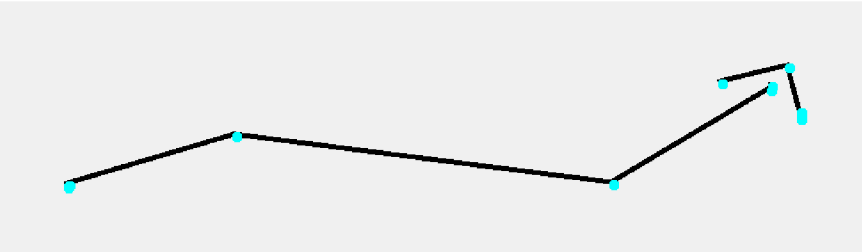


Figure 4.8: Recreated function of class `Arrow` with highlighted important points using cyan circles in the *Showcase* application.

Class Cursor Light

For the representation of telestration tool of class `Cursor Light` the only part needed for recreation are the points of the ellipses in the image, since the direction of the light in the final tool representation will always be vertically up. For this particular reason, I use the list of important points without null points which represents end of the strokes. This list is sent to the function which deletes outlier points from the list, which in this case are the points of the lines coming out of the circle. The function deletes them out with an intention to only have the points of sketched ellipse, without the end of the lines coming from the ellipse. The points at the end of the lines which are coming up from ellipse can be seen as two blue circles at the top of the image in the Figure 4.9.

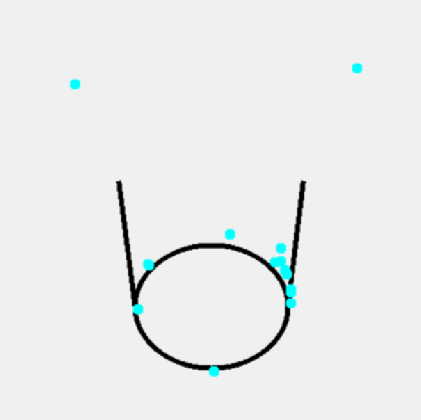


Figure 4.9: Recreated telestration tool of class `Cursor Light` with highlighted important points using cyan circles in the *Showcase* application.

After receiving the list without outlier points, I find the sides of the drawing by locating the most left, right, top and bottom coordinates in the list. With those coordinates, I calculate the location and size of the ellipse and draw it out. Then I draw two lines from the sides of the ellipse upward, representing the light in the original tool.

Classes Cursor Linked and Cursor Linked Closed

Recreation of these two classes of telestration tools can be seen in the Figure 4.10 and Figure 4.11. These are the most complicated to recreate, but their recreation is based on the same principle of recognizing ellipses and lines in the image. For this task of recognizing

and dividing points into the different lines and circles I used K-means clustering method. Theory behind this algorithm can be found in Subsection 2.4.2. Since points have much less direction changes than circles, I used K-means to divide all the important points to clusters and deleted outlier points, which were directional changes in lines that connected circles, therefore leaving me with clusters of ellipse points.

K-means needs a number of clusters inputted, to cluster the points properly. For this, I used a silhouette score which is explained in the Subection 2.4.2. The index number of cluster which the point belongs to is then appended on the end of the data of each point, right after its direction index. These clusters are passed through the similar process as mentioned in the Subsection 4.6.2, where firstly I cut out the outlier points, and after that I assume that each cluster is just the representation of a circle from the original sketch.

The ellipses are drawn after getting their location and size from those clustered points. Size is calculated by getting the furthest points from each direction of X and Y coordinates. I save the information about the centre of the cluster to be able to connect all ellipses with the lines. There is the only difference between the classes `Cursor Linked` and `Cursor Linked Closed`, where in the class `Cursor Linked Closed` I connect all the centers and in the class `Cursor Linked` I do not connect the last ellipse with the first ellipse.

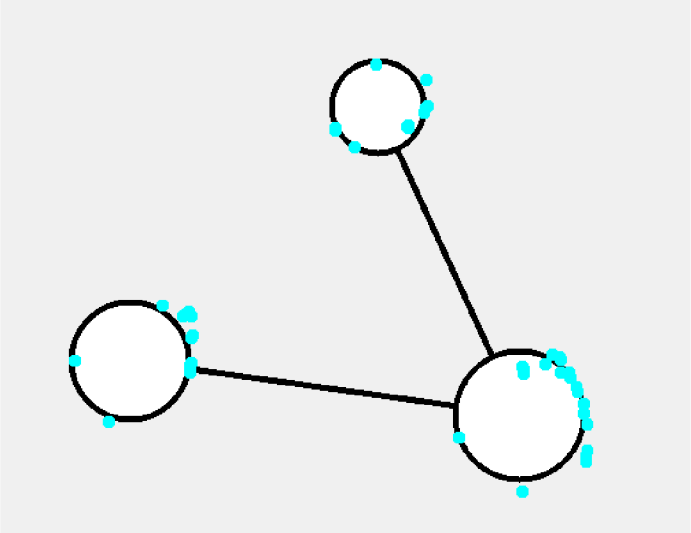


Figure 4.10: Recreated telestration tool of class `Cursor Linked` with highlighted important points using cyan circles in the *Showcase* application.

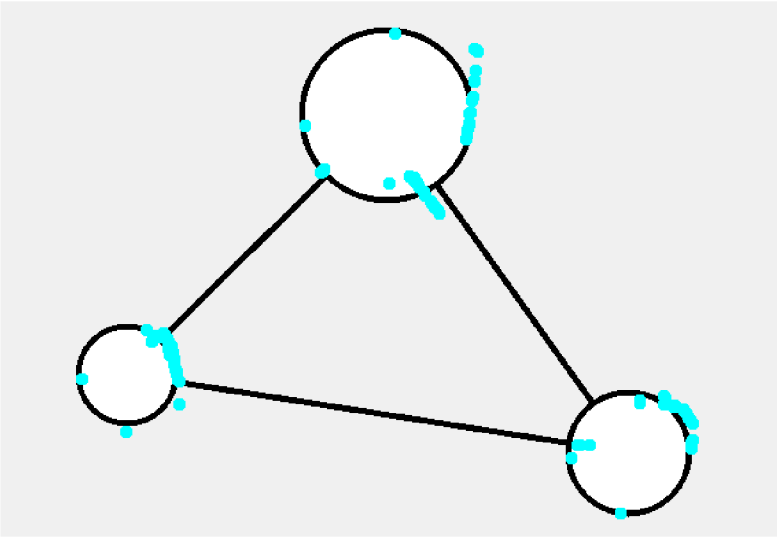


Figure 4.11: Recreated telestration tool of class `Cursor Linked Closed` with highlighted important points using cyan circles in the *Showcase* application.

Class Cursor Player

The recreation of telestration tool of class `Cursor Player` is drawn after calculating the size of the ellipse and locating the centre based on the important points list without the representation of ends of strokes in the list.

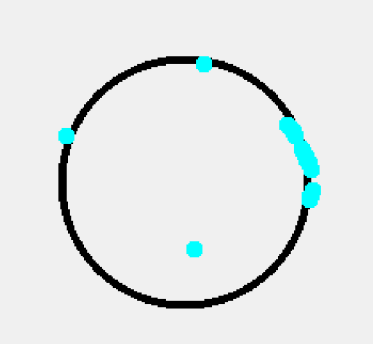


Figure 4.12: Recreated telestration tool of class `Cursor Player` with highlighted important points using cyan circles in the *Showcase* application.

Class Light Shaft

With this class of telestration tool recreation as seen in the Figure 4.13, I use *K-means* algorithm with 2 clusters to identify the ellipse and the line. Ellipse is the cluster with more points, since it has more direction changes than straight line. After identifying the ellipse and calculating its size, I draw the ellipse and make the light in upward direction, without considering its drawn direction, since the light in the function will always go upward.

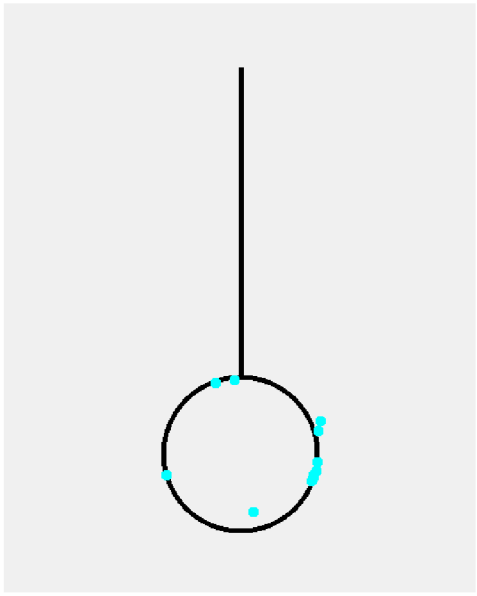


Figure 4.13: Recreated telestration tool of class `Light Shaft` with highlighted important points using cyan circles in the *Showcase* application.

Class Zone Polygon

In the `Zone Polygon` telestration tool class recreation, as seen in the Figure 4.14, I identify the rightmost, leftmost, highest and lowest coordinate. Then I identify the point that is the closest to each corner. These corners are created by combining the furthest coordinates for both X and Y axes and getting 4 corner coordinates: *[Rightmost X coordinate, Lowest Y coordinate], [Leftmost X coordinate, Highest Y coordinate], [Rightmost X coordinate, Lowest Y coordinate], [Leftmost X coordinate, Lowest Y coordinate].* After finding the 4 closest points to those corners, I connect them with the lines to create a polygon.

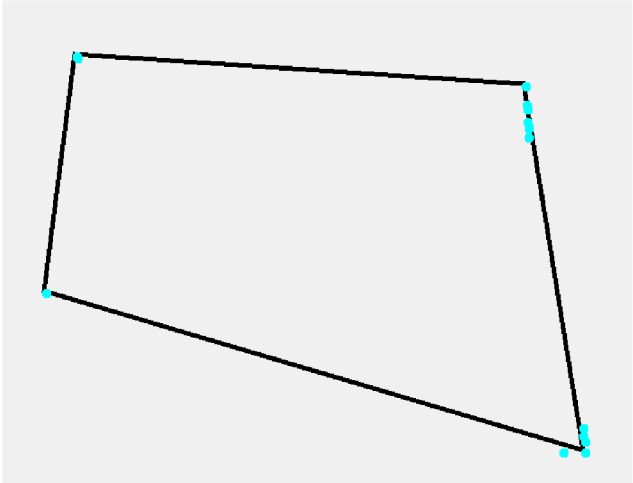


Figure 4.14: Recreated telestration tool of class `Zone Polygon` with highlighted important points using cyan circles in the *Showcase* application.

4.6.3 Post-processing changes with the second segmentation network

Mask array from the second version of the segmentation network consists of three classes described in the Subsection 3.3.3. To recreate the function we only need the lines and ellipses, so I create two lists of points divided into the points identified as the class `Line` and points identified as the class `Ellipse`.

The list of points identified as class `Ellipse` is passed to the function that prints out the points as they are, without filtering any points. This new version makes clustering for `K-Means` algorithm much easier and more efficient. With this list of points identified as the class `Ellipse`, there is also no necessity to cut out outlier points, because the accuracy of this neural network is adequate to exclude any outlying points. This list is used to recreate the classes `Cursor Light`, `Cursor Linked`, `Cursor Linked Closed`, `Cursor Player` and `Light Shaft` with the same processes as mentioned in the Subsection 4.6.2.

The list of points identified as the class `Line` is sorted by finding neighbours of the points using the Algorithm 2. This algorithm sorts the points in order by their neighbours in the list. Neighbouring points are the points closest to each other from the whole list. Then with the sorted points, I try to find any direction changes and the ends and the beginnings of the lines. This sorted and filtered list of points identified as the class `Line` is then sent to the printing function. I use this approach only for recreating of the classes `Arrow` and `Zone Polygon`.

Algorithm 2 An algorithm to put line points in order

```
line_points_list ← array of line points
sorted_list ← empty list
active_point_index ← 0                          ▷ Starting point is the beginning of the list
while length of line_points_list ≠ 0 do
    sorted_list.append(line_points_list[active_point_index])
    line_points_list.delete(line_points_list[active_point_index])
    if length of line_points_list ≠ 0 then  ▷ To find new active point, function is called
which finds the closest point to the last active point which is now in sorted_list
        active_point_index ← Closest point to the last point in sorted_list
    end if
end while
```

Chapter 5

Experiments

Experiments were done to identify the accuracy of the final version of both networks were done by drawing each class 20 times by 5 different participants of the study and collecting how correct the outcome was. I selected 5 different mistakes that could impact the whole process of classification and recreation of the telestration tool from its sketched representation and create an unsatisfactory outcome. These mistakes were:

1. Incorrect classification of image
2. Incorrect directions of segmented points from segmentation network
3. Incorrectly deleted outlying points
4. Incorrect K-means cluster classification

5.1 First phase of testing

First testing was split into two versions of creation of directional masks. Creation of the direction mask is explained in the Section 4.2.2. One was computed from the recorded points when the image was drawn. The other one was outputted from the segmentation neural network mentioned in the Section 4.2.2. Final scores of the experiments are in the Table 5.1.

Table 5.1: Experiment outputs

Class	Mask creation	Output of experiment	Number
Arrow	Segmentation	Incorrect Class	0
		Incorrect Directions	100
		Incorrect Outliers	0
		Incorrect K-means	0
		Correct	0
	Point Recording	Incorrect Class	0
		Incorrect Directions	3
		Incorrect Outliers	0
		Incorrect K-means	0
		Correct	97

Continued on next page

Table 5.1 – continued from previous page

Class	Mask creation	Output of experiment	Number
Cursor Light	Segmentation	Incorrect Class	0
		Incorrect Directions	100
		Incorrect Outliers	0
		Incorrect K-means	0
		Correct	0
	Point Recording	Incorrect Class	0
		Incorrect Directions	0
		Incorrect Outliers	17
		Incorrect K-means	0
		Correct	83
Cursor Linked	Segmentation	Incorrect Class	0
		Incorrect Directions	100
		Incorrect Outliers	0
		Incorrect K-means	0
		Correct	0
	Point Recording	Incorrect Class	0
		Incorrect Directions	0
		Incorrect Outliers	12
		Incorrect K-means	15
		Correct	73
Cursor Linked Closed	Segmentation	Incorrect Class	2
		Incorrect Directions	98
		Incorrect Outliers	0
		Incorrect K-means	0
		Correct	0
	Point Recording	Incorrect Class	0
		Incorrect Directions	0
		Incorrect Outliers	10
		Incorrect K-means	16
		Correct	71
Cursor Player	Segmentation	Incorrect Class	1
		Incorrect Directions	99
		Incorrect Outliers	0
		Incorrect K-means	0
		Correct	0
	Point Recording	Incorrect Class	0
		Incorrect Directions	0
		Incorrect Outliers	0
		Incorrect K-means	0
		Correct	100

Continued on next page

Table 5.1 – continued from previous page

Class	Mask creation	Output of experiment	Number
Light Shaft	Segmentation	Incorrect Class	0
		Incorrect Directions	100
		Incorrect Outliers	0
		Incorrect K-means	0
		Correct	0
	Point Recording	Incorrect Class	0
		Incorrect Directions	0
		Incorrect Outliers	6
		Incorrect K-means	0
		Correct	94
Zone Polygon	Segmentation	Incorrect Class	0
		Incorrect Directions	100
		Incorrect Outliers	0
		Incorrect K-means	0
		Correct	0
	Point Recording	Incorrect Class	0
		Incorrect Directions	0
		Incorrect Outliers	0
		Incorrect K-means	0
		Correct	100

After this phase of testing I found out that the output of segmentation neural network was incorrect and I was unable to identify important points in the arrays it outputted. This problem is shown in the Table 5.1 in each class where segmentation was used, nearly 100 % of the images had incorrect mask.

5.2 Second phase of testing

After this I created a new version of segmentation network which recognises 3 different parts of a sketch, which are lines, ellipses and a background. Creation of a new dataset is described in the Section 3.3.3 and the new segmentation network is described in the Section 4.3. Since the whole segmentation network was reworked, I repeated the experiments using newly trained network implemented with slightly changed post-processing described in the Subsection 4.6.3. The results can be found in the Table 5.2. In this phase I was looking for the same mistakes as in the first phase of testing. These mistakes are described in 5.1. Only change was that instead of incorrect directions of segmented points from segmentation neural network, this time I was looking for incorrect segmentation of lines and ellipses by segmentation neural network.

Table 5.2: Experiment outputs

Class	Mask creation	Output of experiment	Number
Arrow	Segmentation	Incorrect Class	0
		Incorrect Segmentation	0
		Incorrect Outliers	0
		Incorrect K-means	0
		Correct	100
Cursor Light	Segmentation	Incorrect Class	0
		Incorrect Segmentation	0
		Incorrect Outliers	0
		Incorrect K-means	0
		Correct	100
Cursor Linked	Segmentation	Incorrect Class	0
		Incorrect Segmentation	0
		Incorrect Outliers	0
		Incorrect K-means	2
		Correct	98
Cursor Linked Closed	Segmentation	Incorrect Class	1
		Incorrect Segmentation	0
		Incorrect Outliers	0
		Incorrect K-means	4
		Correct	95
Cursor Player	Segmentation	Incorrect Class	0
		Incorrect Segmentation	0
		Incorrect Outliers	0
		Incorrect K-means	0
		Correct	100
Light Shaft	Segmentation	Incorrect Class	1
		Incorrect Segmentation	0
		Incorrect Outliers	0
		Incorrect K-means	0
		Correct	99
Zone Polygon	Segmentation	Incorrect Class	0
		Incorrect Segmentation	0
		Incorrect Outliers	0
		Incorrect K-means	0
		Correct	100

From data I gathered in the second phase of experiments, as seen in the Table 5.2, the second version of segmentation neural network brought an excellent outcome since all of the masks outputted from the segmentation neural network were usable enough to recreate the telestration tool visualizations from them.

While doing the experiments, I recorded the time it took for each class to segment and post-process the whole image. Average time for each class in milliseconds are in the Table 5.3.

	Arrow	Cursor Light	Cursor Linked	Cursor Linked Closed	Cursor Player	Light Shaft	Zone Polygon
Average time (ms)	275.9	284.75	516.6	536.4	286.5	278.4	312.1

Table 5.3: Average time in milliseconds that it took to classify a sketch, create a mask using segmentation model and post-process the mask, divided by the class of the image. The time was measured while doing the second phase of the experiments.

In the two phases of experiments done in the *Showcase* application I tested the classification neural network and two version of segmentation neural network. The experiments were done in real-time with different participants. Average times it took the application to go through the whole process, divided by different classes, are recorded in the Table 5.3. These experiments show that the two neural networks used in the final version of the *Showcase* application are functional and usable in real-time.

Chapter 6

Conclusion

This thesis was mainly focused on training a neural network to classify a hand-drawn sketch into one of seven tools portrayed by its specific symbols, which were specified after the testing with a small group of users. Since this classification is planned to be implemented into an application for coaches and sport analysts, who will be drawing these sketches of symbols in free-hand drawing environment, I created an application to collect the data in a real-life scenario.

After collecting the data from 97 different users, I was able to use this data to create a dataset for an image classification. I used supervised training for the convolutional neural network using the dataset of collected images divided into seven classes. The 2100 sketches drawn in the experiments done with this neural network in the *Showcase* application described in the Section 4.5 showed 99.64 % accuracy of the classification network, which we can see in the Table 5.1 and the Table 5.2. This therefore means that I successfully trained a convolutional neural network to classify the 7 telestration tool sketch representations seen in the Appendix B.

In the second part of this thesis, I recreated the classified sketch into a telestration tool representation. For this purpose I proposed an algorithm to find some features and important parts of each sketch, based on its class. I trained a segmentation neural network for creating the masks for each sketch. These masks consisted of the directions in which the strokes in an image were painted. I trained it on the dataset that I created from the collected data mentioned in the Section 3.3.2. Problem occurred when, after 1400 experiments seen in the Table 5.1 and done in the *Showcase* application, I found out that this approach was not working and 0 % of the created masks were usable for recreation of the functions based on the sketches. The segmentation neural network was unable to identify the directions of the strokes properly.

For evaluation purposes, in the *Showcase* application I recorded each point drawn and created these masks manually. From these manually created masks, I was able to use post-processing described in the Section 4.6 on manually created masks to recreate the final functions from the original sketches.

Then I created the second version of the segmentation neural network with its own dataset described in the Section 3.3.3. This segmentation neural network is able to recognize lines, ellipses and a background in a sketched image. The outputted masks of this neural network were used for recreating the functions from the sketches. The non-functional segmentation neural network was replaced in the *Showcase* application with the second version of the segmentation neural network and after subtle changes in post-processing

process I did some more experiments with the new network. The experiments showed that the masks created by this neural network were all usable to recreate the original functions.

Bibliography

[1] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. et al. ImageNet: A large-scale hierarchical image database. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, p. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[2] Eitz, M., Hays, J. and Alexa, M. How Do Humans Sketch Objects? *ACM Trans. Graph. (Proc. SIGGRAPH)*. 2012, vol. 31, no. 4, p. 44:1–44:10.

[3] Guo, K., Woma, J. and Xu, E. *Quick, Draw! Doodle Recognition*. Stanford University, 2018. Available at: https://cs229.stanford.edu/proj2018/report/98.pdf.

[4] Ha, D. and Eck, D. A Neural Representation of Sketch Drawings. In: *International Conference on Learning Representations*. 2018. Available at: https://openreview.net/forum?id=Hy6GHpkCW.

[5] He, K., Zhang, X., Ren, S. and Sun, J. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, p. 770–778.

[6] Khatri, S. K., Dutta, S. and Johri, P. Recognizing images of handwritten digits using learning vector quantization artificial neural network. In: *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*. 2015, p. 1–4. DOI: 10.1109/ICRITO.2015.7359307.

[7] Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. *CoRR*. 2014, abs/1412.6980.

[8] MacQueen, J. Some methods for classification and analysis of multivariate observations. In:. 1967.

[9] Qin, S. Intelligent Classification of Sketch Strokes. In: *EUROCON 2005 - The International Conference on „Computer as a Tool“*. 2005, vol. 2, p. 1374–1377. DOI: 10.1109/EURCON.2005.1630216.

[10] Ronneberger, O., Fischer, P. and Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *CoRR*. 2015, abs/1505.04597. Available at: http://arxiv.org/abs/1505.04597.

[11] Rousseeuw, P. J. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*. 1987, vol. 20, p. 53–65. DOI: https://doi.org/10.1016/0377-0427(87)90125-7. ISSN 0377-0427. Available at: https://www.sciencedirect.com/science/article/pii/0377042787901257.

[12] Rubinstein, R. The Cross-Entropy Method for Combinatorial and Continuous Optimization. *Method. Comput. Appl. Prob.* USA: Kluwer Academic Publishers. sep 1999, vol. 1, no. 2, p. 127–190. DOI: 10.1023/A:1010091220143. ISSN 1387-5841. Available at: https://doi.org/10.1023/A:1010091220143.

[13] Seddati, O., Dupont, S. and Mahmoudi, S. DeepSketch 2: Deep convolutional neural networks for partial sketch recognition. In: *2016 14th International Workshop on Content-Based Multimedia Indexing (CBMI)*. 2016, p. 1–6. DOI: 10.1109/CBMI.2016.7500261.

[14] Seddati, O., Dupont, S. and Mahmoudi, S. DeepSketch: Deep convolutional neural networks for sketch recognition and similarity search. In: *2015 13th International Workshop on Content-Based Multimedia Indexing (CBMI)*. 2015, p. 1–6. DOI: 10.1109/CBMI.2015.7153606.

[15] Shelhamer, E., Long, J. and Darrell, T. Fully Convolutional Networks for Semantic Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2017, vol. 39, no. 4, p. 640–651. DOI: 10.1109/TPAMI.2016.2572683.

[16] Sutherland, I. E. Sketch Pad a Man-Machine Graphical Communication System. In: *Proceedings of the SHARE Design Automation Workshop*. New York, NY, USA: Association for Computing Machinery, 1964, p. 6.329–6.346. DAC '64. DOI: 10.1145/800265.810742. ISBN 9781450379328. Available at: https://doi.org/10.1145/800265.810742.

[17] Tran, E. and Lu, W. *Free-hand Sketch Recognition Classification*. Stanford University, 2017. Available at: http://cs231n.stanford.edu/reports/2017/pdfs/420.pdf.

[18] Yuan, Z. and Jin, G. Sketch Recognition Based Intelligent Whiteboard Teaching System. *2008 International Conference on Computer Science and Software Engineering*. 2008, vol. 5, p. 867–870.

[19] Zhang, H., Liu, S., Zhang, C., Ren, W., Wang, R. et al. SketchNet: Sketch Classification with Web Images. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, p. 1105–1113.

[20] Žára, J., Beneš, B., Sochor, J. and Felkel, P. *Moderní počítačová grafika*. 2.th ed. Praha: Computer Press, 2005. 80-82 p. ISBN 80-251-0454-0.

Appendix A

Tools visualizations

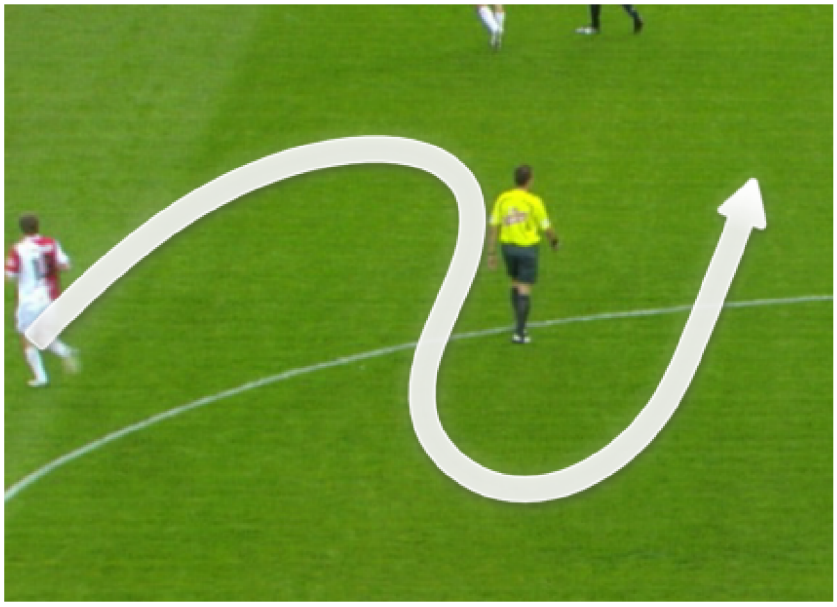


Figure A.1: Arrow tool visualization.[1]

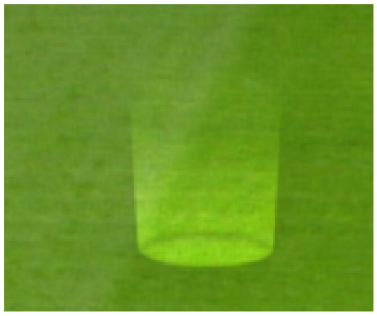


Figure A.2: Cursor Light tool visualization.[1]

[1]Courtesy of ChyronHego.

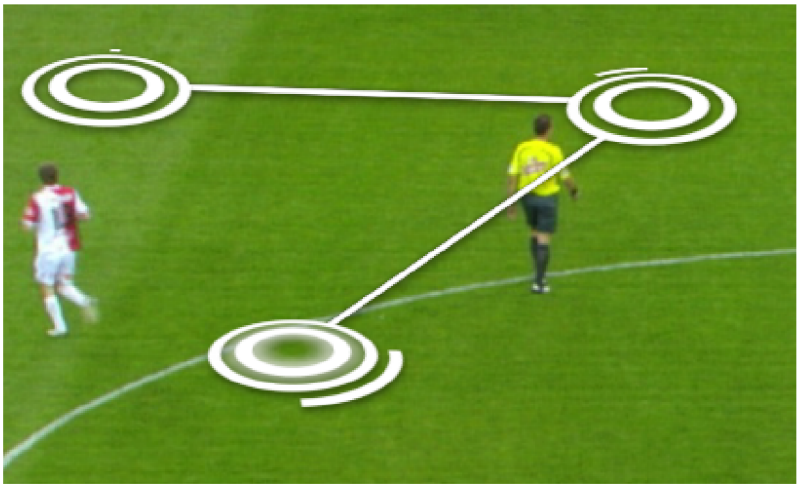


Figure A.3: Cursor Linked tool visualization.[1]

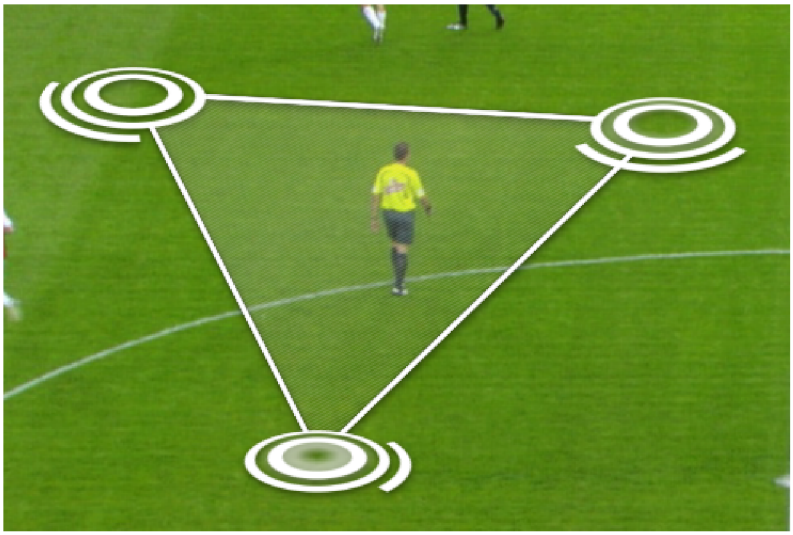


Figure A.4: Cursor Linked Closed tool visualization.[1]

Figure A.5: Cursor Player tool visualization.[1]

[1]Courtesy of ChyronHego.

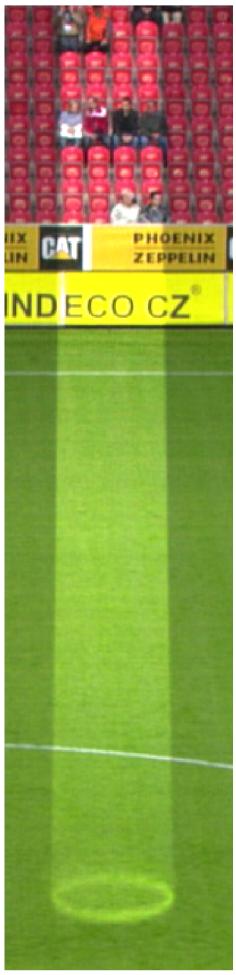


Figure A.6: Light Shaft tool visualization.[1]

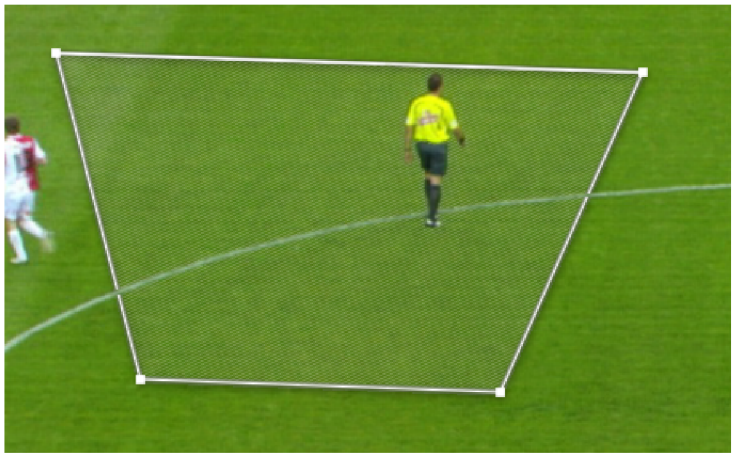


Figure A.7: Zone Polygon tool visualization.[1]

[1]Courtesy of ChyronHego.

Appendix B

Tools sketch representations

Figure B.1: Sketch representation of the function Arrow.

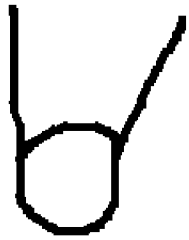


Figure B.2: Sketch representation of the function Cursor Light.

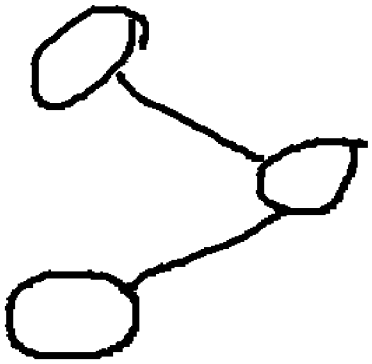


Figure B.3: Sketch representation of the function Cursor Linked.

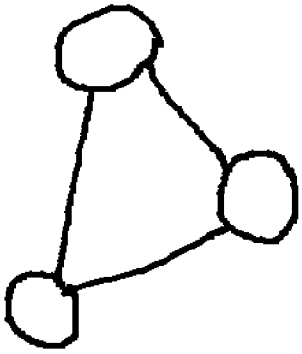


Figure B.4: Sketch representation of the function Cursor Linked Closed.

Figure B.5: Sketch representation of the function Cursor Player.

Figure B.6: Sketch representation of the function Light Shaft.

Figure B.7: Sketch representation of the function Zone Polygon.

Appendix C

Scientific Poster

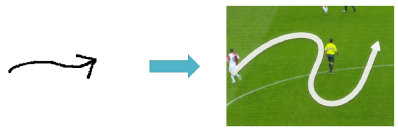
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

2023
Brno University of Technology

Chyron

AUTOMATIC GRAPHICS TOOL SELECTION USING FREE-HAND SKETCHES

Made by Richard Harman
Supervisor: doc. Ing. Martin Čadík Ph.D.
Consultants: Ing. Jan Brejcha Ph.D. and Mgr. Dominika Trebatická from ChyronHego Corporation



[IMAGE CAPTION] Arrow function (right) and sketch representing this function (left) which is a courtesy of ChyronHego

INTRODUCTION

THIS WORK FOCUSES ON CONVERTING SKETCHES OF SPORTS ANALYSTS AND COACHES TO TELESTRATION TOOL REPRESENTATIONS NEEDED FOR SPORT ANALYSIS.
IT'S PLANNED TO BE IMPLEMENTED TO RECOGNIZE SKETCHES IN FREE HAND DRAWING SECTION OF SPORT ANALYSIS APPLICATION.

OBJECTIVE

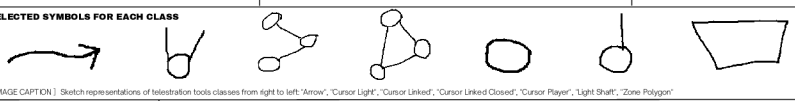
Sketch classification for translation of hand-drawn sketches into one of 7 telestration tools from ChyronHego Corp. Application.

METHODOLOGY

- Selected a symbol as a representation of each function
- Created an application in *Python* using *PyQT* framework to collect data needed to create a dataset. Data were split in these sections:
  - Images of sketches in *.png* format
  - Timestamps of sketched pixels
  - Colour of sketched pixel
  - Position of sketched pixel
  - Change of stroke drawing direction
- Collected data from 97 participants - each class sketched on the white background and on the image from football match to recreate real-life usage
- Created a dataset for training a CNN for classification and another dataset for image segmentation that classifies lines and ellipses in the image for recreating of the telestration tools
- Trained both neural networks
- Implemented another app to showcase usage of trained networks

[IMAGE CAPTION] Help block from data gathering application, used to fill users data and show user what to draw

SELECTED SYMBOLS FOR EACH CLASS



[IMAGE CAPTION] Sketch representations of telestration tools classes from right to left: "Arrow", "Cursor Light", "Cursor Linked", "Cursor Linked Closed", "Cursor Player", "Light Shaft", "Zone Polygon"

USED TECHNOLOGIES

- PYTHON
- PYQT
- PYTORCH
- NUMPY
- PIL
- MATPLOTLIB

RELATED PAST WORK

[1] S. K. KHATRI, S. DUTTA AND P. JOHRI, "RECOGNIZING IMAGES OF HANDWRITTEN DIGITS USING LEARNING VECTOR QUANTIZATION ARTIFICIAL NEURAL NETWORK," 2015 4TH INTERNATIONAL CONFERENCE ON RELIABILITY, INFOCOM TECHNOLOGIES AND OPTIMIZATION (ICRITO) (TRENDS AND FUTURE DIRECTIONS), NOIDA, INDIA, 2015, PP. 1-4, DOI: 10.1109/ICRITO.2015.7359307.

[2] O. SEDDATI, S. DUPONT AND S. MAHMOUDI, "DEEPSKETCH: DEEP CONVOLUTIONAL NEURAL NETWORKS FOR SKETCH RECOGNITION AND SIMILARITY SEARCH," 2015 13TH INTERNATIONAL WORKSHOP ON CONTENT-BASED MULTIMEDIA INDEXING (CBMI), PRAGUE, CZECH REPUBLIC, 2015, PP. 1-6, DOI: 10.1109/CBMI.2015.7153606.

[3] SHENGFENG QIN, "INTELLIGENT CLASSIFICATION OF SKETCH STROKES," EUROCON 2005 - THE INTERNATIONAL CONFERENCE ON "COMPUTER AS A TOOL", BELGRADE, SERBIA, 2005, PP. 1374-1377, DOI: 10.1109/EUROCON.2005.1630218.

[CAPTION] *CONFUSION MATRIX VISUALIZATIONS FROM TRAINING OF CLASSIFICATION NETWORK AND IT'S ACCURACY ON TRAINING SET (RIGHT) AND VALIDATING SET (LEFT)*

CONCLUSION

- Created data collecting application
- Created two datasets from collected data
- Succesfully trained convolutional neural network on own dataset to recognize 7 classes with 96% accuracy
- Succesfully trained segmentation neural network to recognize lines and ellipses
- Created showcasing application with usage of both neural networks and post-processing to recreate functions from sketches