Smartboard

(Inteligentna tablica przeglądarkowa)

Jakub Skalski Cezary Świtała

Praca inżynierska

Promotor: dr Marek Adamczyk

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

22 czerwca 2023

Abstract

The aim of this work is to design and develop a web application featuring interactive whiteboard functionality while also utilizing machine learning algorithms for shape reconstruction and handwriting assistance. The project entails developing a suitable user interface and backend that would enable collaboration among multiple users on a shared whiteboard simultaneously.

Celem pracy jest zaprojektowanie i zbudowanie aplikacji internetowej z funkcjonalnością tablicy interaktywnej wykorzystującej algorytmy uczenia maszynowego do rekonstrukcji kształtów i wspomagania pisma ręcznego. Projekt obejmuje opracowanie odpowiedniego interfejsu użytkownika i zaplecza, które umożliwiłyby równoczesną współpracę wielu użytkowników na tej samej tablicy.

Contents

Chapter 1

Introduction

1.1 Motivation

About a half a year into our studying at the university the COVID-19 pandemic happened. It forced us to adapt to a new reality where we could only communicate and collaborate with our friends over the Internet. One of the challenges at that time was finding an alternative to whiteboards and chalkboards of our classrooms that we grew so accustomed to.

We have soon come to a realization that there are lots of available online whiteboards that allow for simultaneous work of multiple people and we have eventually settled for one that provided a reasonable variety of free features, but we were never entirely satisfied with it. Why? Because:

- drawing on them didn't feel quite right. Shapes often came out very clumsy and written words had to be drawn very carefully to be legible.
- user interface was not very comfortable. It required a lot of effort to perform simple actions like drawing regular shapes.
- some basic and useful options required paid subscriptions seemingly without any reason other than trying to get users to pay.

That's when the idea of creating our own free, feature-rich and easy to use online whiteboard came up.

The goal of this project was not to provide some fully-fledged implementation that could compete with already available solutions, but to lay some solid foundations for future development of such board.

1.2 The actual board

The board application is deployed under this link:

`https://wizzwriters.github.io/spelldraw-app/`

The detailed user manual can be found in chapter 6, but to get acquainted with the application, you can try drawing some shapes. If you hold your cursor still when drawing a line, the shape correction will be activated. Try to activate this functionality while drawing a triangle.

You can also write some English word like 'whiteboard' and then use the select tool to change it into proper text. All you need to do is hold the selection box over the word for a moment. Make sure that the entire word fits inside the box.

1.3 Components

The project consists of several major components that are dependant on each other, but should be considered separate entities.

- **Front-end application** – provides the user interface with basic tools for drawing on a virtual board.
- **Machine learning models** – enhance the user handwriting experience.
- **Back-end server** – sets up and assists in connections between users.

1.4 Responsibilities

We divided the implementation responsibilities according to our own personal areas of expertise.

Cezary was responsible for:

- Designing the user interface.
- Implementation of the front-end application and the back-end server.
- Workflow, continuous integration and deployment.

While Jakub was responsible for:

- Data acquisition and processing.
- Research and development of machine learning models suitable for the task.
- Model to application integration.

Chapter 2

Front-end Application

2.1 Overview

Our web application is the main user interface for drawing shapes on the board. Due to its highly interactive nature, it is implemented as a *single-page application* (SPA), which is a type of website that downloads all of its assets once and then generates content dynamically with JavaScript. Such approach makes the website feel like a native application and allows for an easy and quick modification of page content in response to user interaction. SPAs are usually implemented using some kind of dedicated JavaScript framework. Our application is not an exception to that rule.

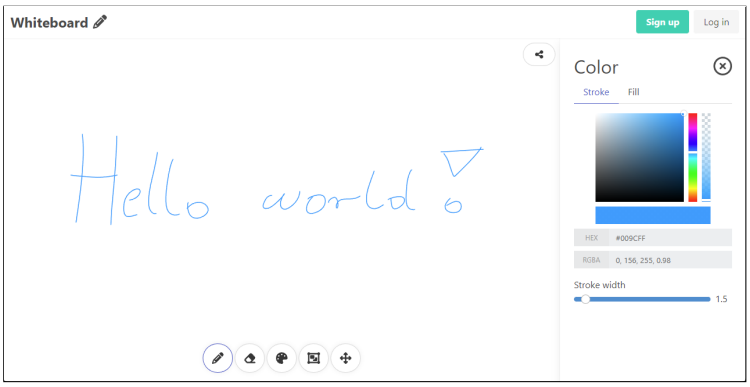

Figure 2.1: Screenshot of the application interface

2.2 Technology stack

The application is written entirely using TypeScript programming language with *Vue.js* framework. It also uses HTML to define the structure of components and Sass language for styling together with Bulma framework. You will find a short description of both frameworks in the subsequent sections.

2.2.1 Vue.js

Vue.js is a JavaScript framework for building web user interfaces. Structure of Vue application is based on components, which are reusable units of user interface that encapsulate the HTML structure with all associated programming logic, state and styles in a single file.

One of the core features of this framework is the declarative rendering of HTML templates based on JavaScript state. The other one is reactivity. Vue can automatically track state changes and re-render components when there is a need to. It is also designed to be lightweight, which in turn means that any additional functionalities like routing or state management are optional and can be installed in a form of plugins provided by the Vue team or by the Vue community. Examples of such plugins used in our project include *Pinia* used for global state management and *Vue Router* used for routing.

For this project we have chosen a new version of this framework – *Vue 3*, which provides a new API called *Composition API* that is very easy to read, has native support for TypeScript language and allows for creating reusable parts of component logic called *composables*.

2.2.2 Bulma

Bulma is a lightweight CCS styling framework. It provides utility classes that help with layout and typography, styles for elements as well as whole components like navigation bars. Every class is designed to be responsive and usable on devices of any size.

This framework shares a lot of similarities with other CSS frameworks like Bootstrap. Qualities that make it stand out are modularity, customizability and an absence of JavaScript code, which implies effortless integration with any JavaScript environment.

2.2.3 Tensorflow.js

For the technical part of this thesis, we export models in Keras format and import them into our application through the Tensorflow.js framework. It is important to keep the models small for the quick inference time and also simple, since Tensorflow.js is not fully compatible with Keras and regular TensorFlow API but supports most of the basic computational graph functionalities. Models are initialized in a lazy fashion to ensure that the application can load smoothly. This comes at a cost of having to load the weights on the fly during the first inference. Furthermore, due to the simplistic nature and limited capabilities of Tensorflow.js an extensive model interface has to be prepared on the side of the application that would take care of

formatting the outputs and managing the models. The models are not quantized although they do come in a frozen format meaning that most of the data revolving around the training process is stripped from the model.

2.3 Canvas

Canvas is the core component of our application. It is the surface upon which tools can be used to draw shapes. The most obvious and common choice for implementing such functionality in web applications seems to be the HTML Canvas, but ultimately we chose to base our implementation on *Scalable Vector Graphics* (SVG) instead.

SVG is a very popular and commonly used vector image format based on *Extensible Markup Language* (XML) and supported by all modern browsers. It is a markup language that can be modified by JavaScript just like HTML.

The reason behind choosing SVG over the HTML Canvas is that it can be generated and supervised by the Vue.js framework. This in turn makes for a seamless integration with the rest of the app and allows us to make use of its built-in reactivity features.

2.4 Tools

In the scope of this thesis we were aiming on providing some very basic tools that would make the whiteboard usable. We can hope that in the future this list will grow significantly to make our app more robust, but for now the implemented tools are as follows:

- **Draw tool** – for drawing simple lines. It also gives access to the shape correction functionalities.
- **Erase tool** – used to erase any shapes on its path just like we do with an eraser on a physical whiteboard.
- **Select tool** – can be used to select a collection of shapes and then perform actions on them. It also allows users to perform text recognition on their handwriting.
- **Move tool** – allows for movement around the canvas.
- **Color picker** – provides interface for changing color and other properties of drawn shapes.

There is no point in dwelling on the technical details of every single tool, but in the following sections we will give an overview of two representative examples.

2.4.1 Draw tool technicalities

The drawing tool is probably the most crucial to get right. The whole user experience largely depends on how enjoyable drawing is and on how aesthetically pleasing the drawn shapes are.

We achieved satisfactory effects by sparsely collecting the cursor position from the user and approximating the intended shape with Bézier curves, which gives them a very smooth and organic look. [1]

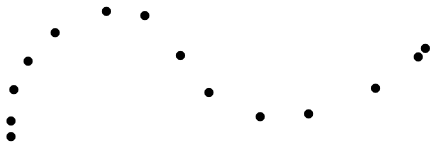

Figure 2.2: Exemplary set of points collected from user

Shapes drawn with this tool are just a collection of points recorded when the user was making the drawing movement. Figure 2.2 shows an example of such a set. The starting point and the ending point will always be included. Points that lie in between will be collected using some arbitrarily chosen constant time interval.

When the shape is drawn on canvas, its points will be used both as starting and control points of the Bézier curves. We will use as many cubic curves as we can and if that leaves any remainder it will be used for quadratic or linear curve. The next step will be to smooth out transitions between the curves.

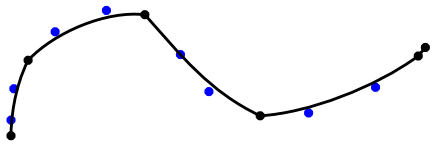

Figure 2.3: Approximation without smooth transitions

Figure 2.3 shows what the line would look like if we treated some of the collected points as control points of the curves (those are painted in blue). To make the transitions seamless we adjust positions of starting points to fall exactly between neighbouring control points.

(a) Transition before adjustment (b) Smooth transition after adjustment

Figure 2.4: Results of adjusting the starting point

After applying this method to all starting points in the shape, we get the final curve. The difference between the resulting curve and the initial set of points is usually not noticeable to user.

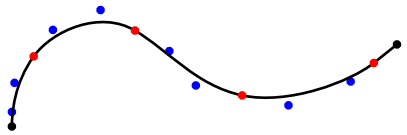

Figure 2.5: Final curve with all used points shown

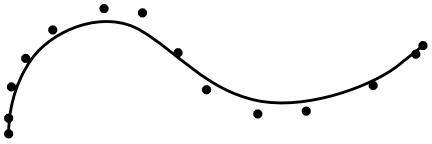

Figure 2.6: Final curve compared to the initial set of points

2.4.2 Erase tool technicalities

The eraser tool requires a way to detect intersection between the current position of the cursor and the shapes drawn on canvas. Also, the user should be able to press down and move around the canvas, erasing all of the shapes on his path.

Testing if some point intersects with the stroke of a shape can be done via API provided by browser for SVG elements. This means that the test has to be performed by the Vue component that holds the reference to the actual rendered SVG object. This is where the concept of *Event Bus* [2] came in handy. Event bus is a global object that provides a way to broadcast and subscribe to events. Every shape component would then subscribe to the event emitted by the erase tool and check for intersection with specified points. This mechanism is used by other tools as well.

A big inconvenience of other online boards is that eraser tools sometimes skip over shapes. Since we only receive mouse events every so often (even if we did, we wouldn't want that because it will greatly affect the performance), we need to check some points in-between registered positions to make sure that we don't miss any shapes. To do that, we sample space between two measured positions in intervals shorter than the minimum stroke width.

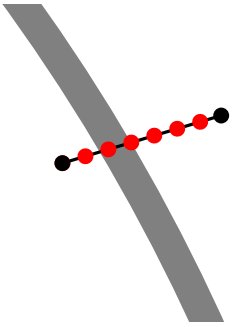

Figure 2.7: Samples (shown in red) taken from the segment between two positions

As seen in Figure 2.7 even if our measured positions skipped over some shape, at least one of the samples taken from segment between them must land on the stroke because distance between them is smaller than the thinnest stroke.

2.5 Shape correction and handwriting recognition

In addition to the various drawing tools, a major part of this application is its ability to recognize and correct shapes drawn by the user. It is able to correct some simple geometrical shapes, as shown in Figure 2.8.

(a) Shape drawn by user (b) Shape after correction

Figure 2.8: Example of shape correction

It can also recognize handwritten words and change them into text, as shown on Figure 2.9.

(a) Word written by user (b) Recognized word displayed as text

Figure 2.9: Examples of handwriting recognition

2.5.1 Technicalities

Both of those functionalities are powered by machine learning models that will be described in great detail in subsequent chapters. All the application has to do is provide drawn shapes in the required format. For this purpose, we implemented hidden HTML canvases that the shapes selected for correction get squeezed into. The canvas is then fed into *TensorFlow.js* framework to be processed by the models.

Figure 2.10: Image generated for the handwritten word above

Chapter 3

Shape Correction

3.1 Motivation

Drawing regular shapes in a graphical editing tool or an online whiteboard can be highly beneficial for users, as it provides a convenient and efficient way to create visually appealing and geometrically precise graphics. Regular shapes, such as circles, squares, triangles, and rectangles, are simple and visually pleasing, making them ideal for a wide range of design and illustration purposes. Such functionality would greatly enhance the overall user experience by reducing the time and effort required to manually draw precise shapes, allowing users to focus on their creative expression.

3.2 Problem description

Given an image of a drawing provided by the user, our task is to either produce a suitable reconstruction of the shape contained in the image or reject the correction entirely. Although ellipse, parallelogram and triangle are the primary shapes we would also like to have the ability to tell them apart from other drawings that would then be rejected. We also require the solution to be swift and light enough for an online web application.

3.3 Methodology

For this task we would employ a two-step process utilizing deep learning techniques. First, the preprocessed image is passed to the deep classifier that recognizes which shape the image represents. Should the image be recognized as one of the primary shapes, it is then passed to its corresponding regressor which in turn identifies its characteristic vertices on which the reconstruction can be performed.

3.4 Data

The training data consist of 80000 fixed 70x70 pixel size images with corresponding labels and characteristic vertices. Each image is binary and white on black. Vertices are supplied only for the primary shapes. The split between ellipses, parallelograms, triangles and others is even, in order to prevent models from becoming biased towards some particular shape. The data comes from the Hand-drawn Shapes (HDS) dataset © 2022 by Francois Robert and is licensed under CC BY 4.0.

Figure 3.1: Exemplary data

3.5 Models

3.5.1 Classifier

Loss function

Since this is a simple mnist-like classification problem cross entropy is very much the default choice here.

$$H(p,q) = -\sum_{x \in \chi}^{M} p(x) \log(q(x)) \tag{3.1}$$

Architecture

Classifier architecture is rather simple and somewhat resembles that of AlexNet. We use ReLU activation functions and some dropout layers. We start with feature extraction using convolutions, then flatten their outputs which are then passed through dense layers with softmax as the final activation. The overall size of the architecture is 264516 parameters, which amounts to a megabyte worth of data.

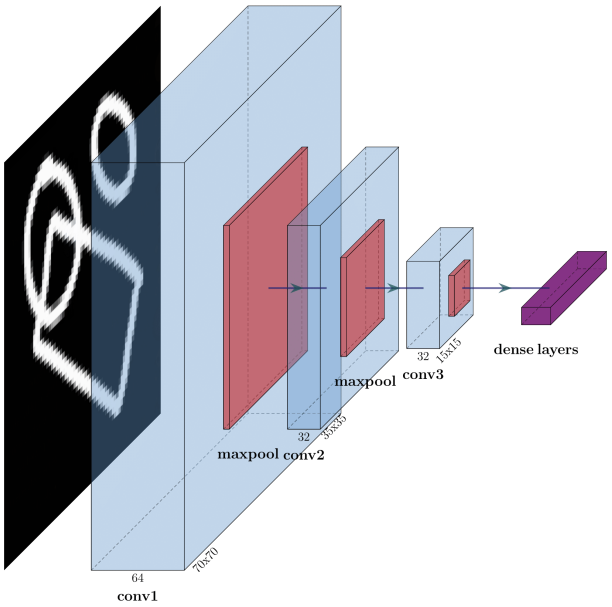

Figure 3.2: Classifier architecture

Training

We measure training success using the categorical accuracy metric. Should the model reach a plateau the learning rate would be lowered and the training would continue. Finally, we choose the epoch where the model performed best on the validation set.

Results

The model scores 98% accuracy on the test set which is very much sufficient for our purposes.

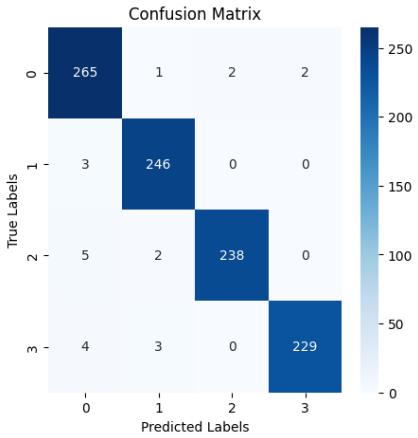

Figure 3.3: Confusion matrix

From the confusion matrix in Figure 3.3 we conclude that the model does not

exhibit bias towards any class in particular. The model is also simple enough to apply SHAP's [3] DeepExplainer. An approach based on the concept of integrating Shapley [4] values into deep neural networks. It provides a more global interpretation of the model by sampling from the background data set and assigning importance values to each feature. Shapley values represent the average marginal contribution of each feature across all possible feature combinations.

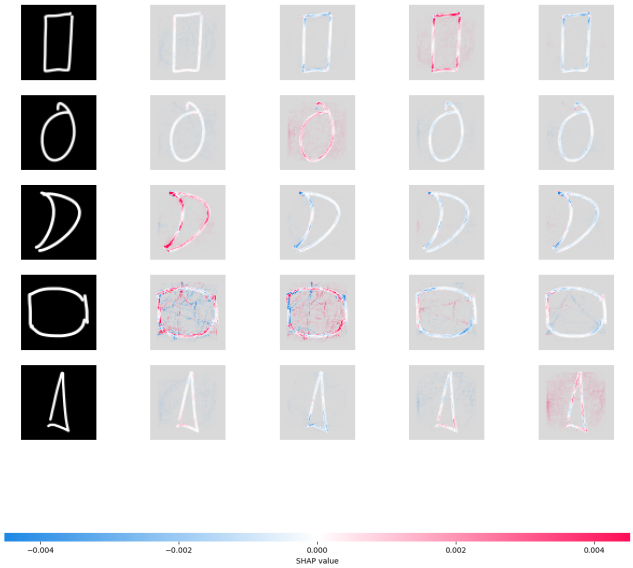

Figure 3.4: Shapley values obtained from DeepExplainer

Figure 3.4 shows Shapley values superimposed onto the corresponding image features, where each column represents a different class. This explanation reveals that the network is in fact looking at the shape and that the true class produces highest values for the relevant features. Another way to test what the network is paying the most attention to in order to make its predictions is to perform model-agnostic analysis with Lime as was done in figure 3.5.

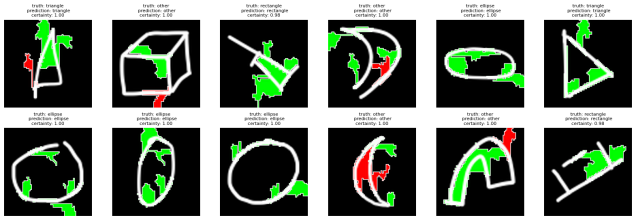

Figure 3.5: Lime explanation

3.5.2 Regressors

Loss function

Mean square error is usually a good first try and was performing reasonably well, but the neural network errors were quite peculiar, mostly due to a strange problem definition that such a loss function would imply. Since ordering in which the network returns the desired vertices should not matter, we opted to optimize the Chamfer distance [5] instead, leading to superior results. The Chamfer distance between two sets of points A and B is defined as:

$$C(A,B) = \frac{1}{|A|}\sum_{a\in A}\min_{b\in B}|a-b| + \frac{1}{|B|}\sum_{b\in B}\min_{a\in A}|a-b| \tag{3.2}$$

where $|A|$ and $|B|$ represent the cardinality of sets A and B respectively, $\|\cdot\|$ denotes the Euclidean distance between two points and min denotes the minimum value over a set.

Architecture

In the early development it became apparent that stacking convolutional layers was the fastest way to improve performance, but only up to a certain point. We addressed this issue by introducing residual connections. Such a residual network architecture, also commonly referred to as ResNet, is a popular deep learning architecture that has been widely adopted in various computer vision tasks. ResNet introduces the concept of residual connections, which allow the model to skip some layers and create shortcuts in the network. This design choice enables the network to effectively handle the challenges of training very deep neural networks. The final model uses 286288 parameters and also weighs one megabyte.

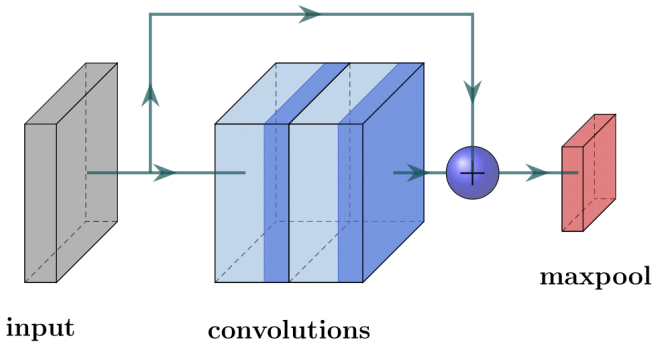

Figure 3.6: Residual block

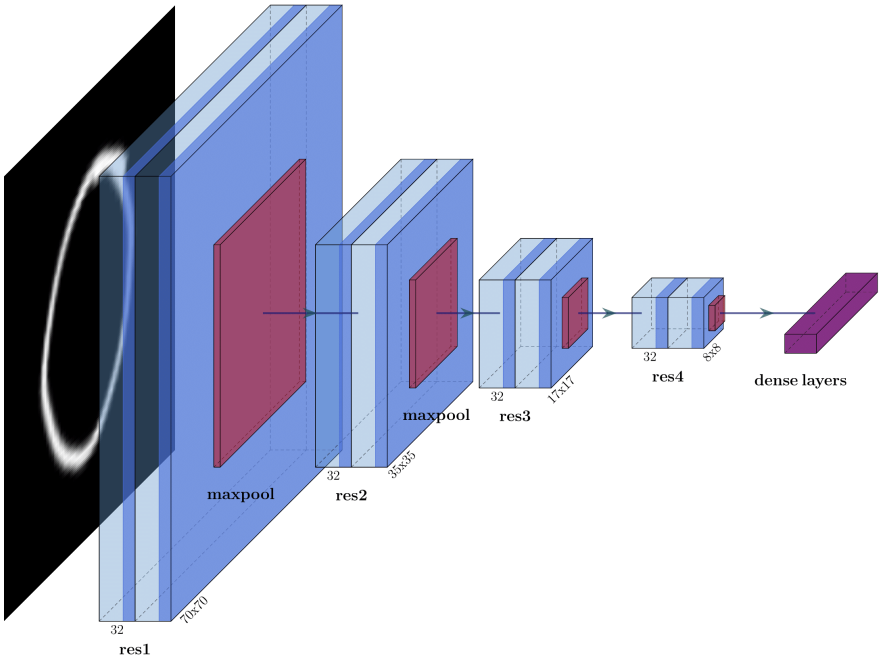

Figure 3.7: Regressor architecture

Metrics

When comparing point clouds, two common metrics used are the Dice coefficient and the Intersection over Union (IoU). These metrics are mostly used in tasks like object detection or segmentation, but we can imagine the clouds to have some convex shape and calculate the metrics on those. Their values range from 0 to 1, where 0 indicates no overlap between the point clouds, and 1 indicates perfect alignment. The Dice coefficient and IoU are defined as follows:

$$\text{Dice} = \frac{2 \times |A \cap B|}{|A| + |B|} \qquad \text{IoU} = \frac{|A \cap B|}{|A \cup B|} \tag{3.3}$$

where $|A \cap B|$ and $|A \cup B|$ represent the overlap and union areas of point clouds A and B. Both the Dice coefficient and IoU provide a measure of similarity or overlap between two point clouds, but they have slightly different interpretations. The Dice coefficient emphasizes the relative size of the intersection compared to the sizes of the individual point clouds, while the IoU focuses on the proportion of the intersection with respect to the union.

Results

In the end we achieved 0.94 and 0.84 accuracy for dice and IoU respectively. In general this is mostly sufficient, but since the errors concern 70x70 images even the smallest of errors can turn out to be very noticeable should the image be up scaled.

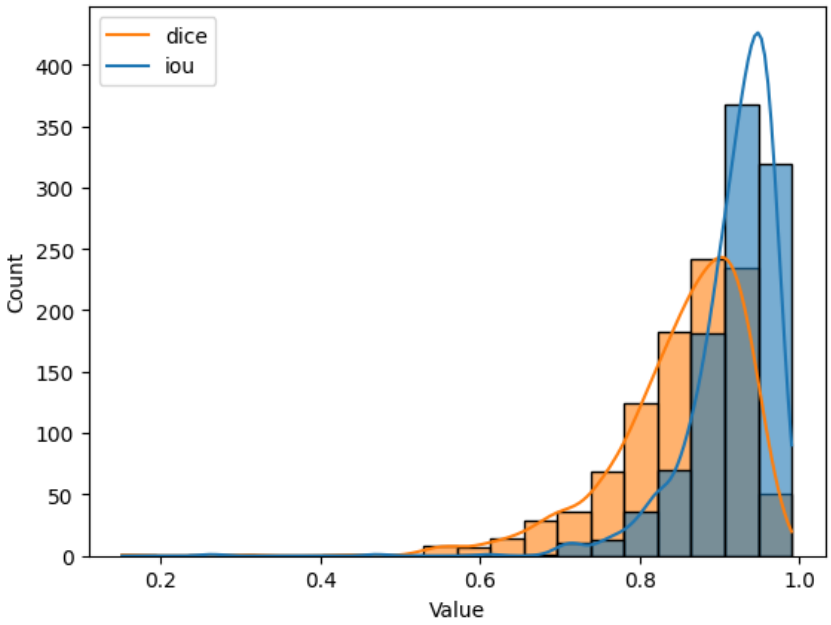

Figure 3.8: Samples distribution according to chosen metrics

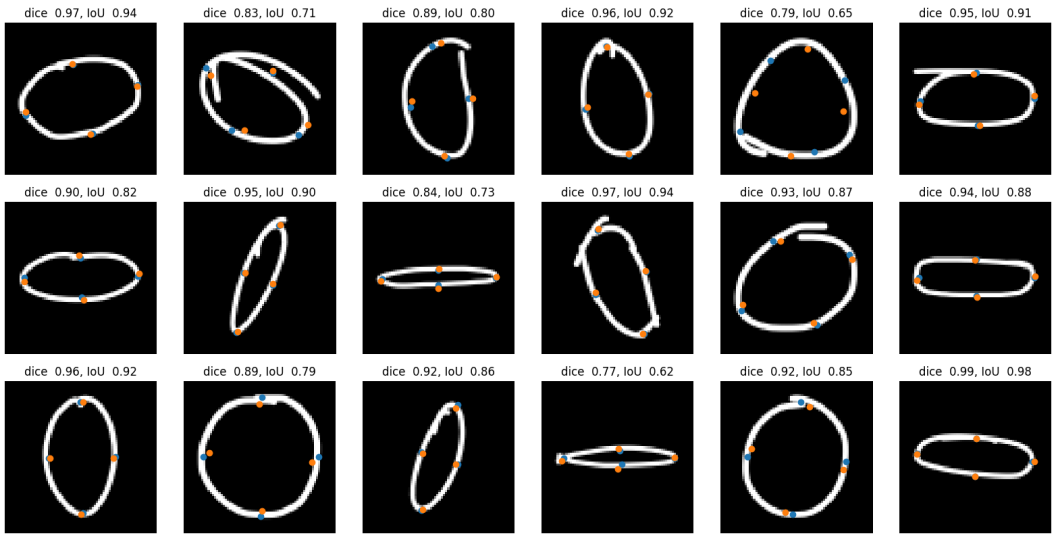

Figure 3.9: Model predictions (orange dots) for randomly sampled ellipses

Chapter 4

Handwriting Recognition

4.1 Motivation

Handwriting is a natural and intuitive way for humans to express their ideas and thoughts. By incorporating handwriting recognition, users can directly write or draw on a whiteboard and have their writings turned into text, making the writing process easy and convenient. It eliminates the need for using a keyboard for input, allowing for a more fluid and creative experience.

4.2 Problem Description

The objective is to develop a model that can accurately transcribe handwritten text into machine-readable form and guess what word does the text most likely represent. The handwriting would then be replaced with the print image of the obtained word on the side of the application.

4.3 Methodology

During the inference phase, we pass the handwritten image through the trained model. The model outputs a sequence of probability distributions over classes (tokens) at each time step. We then decode this CTC output (see 4.3.1) using best path strategy and map it back into letters. Finally, we compare the decoded word against the entire English corpus using the Levenshtein distance and choose the most matching word as the final output of the handwriting recognition system.

4.3.1 Connectionist Temporal Classification

In traditional sequence labeling problems, the lengths of the input and output sequences are usually matched, making the task straightforward. However, in certain cases, such as handwriting recognition, the length of the input sequence can be different from the length of the target transcription sequence. This misalignment between input and output sequences makes the direct application of standard supervised learning approaches challenging. Connectionist Temporal Classification (CTC) addresses this problem by allowing the model to learn the alignment between input and output sequences in an end-to-end manner, without the need for explicit alignment annotations. In order to achieve this, the CTC introduces a special epsilon label and allows for token repetitions.

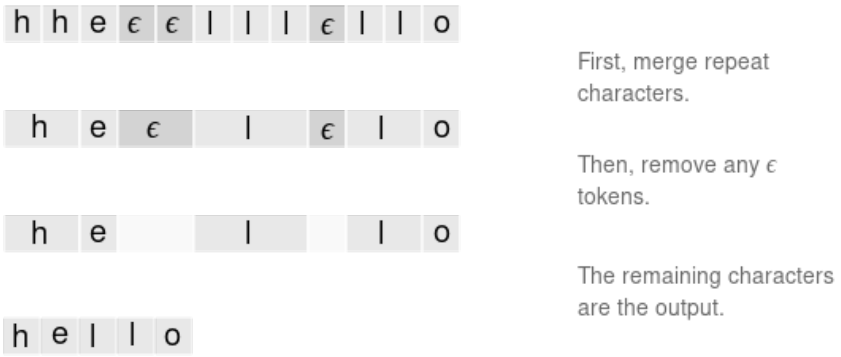

Figure 4.1: CTC alignment

Models trained in the manner of Connectionist Temporal Classification output time steps, each corresponding to a softmax token distribution. We use LSTMs to help the network infer the chronological ordering of the time steps in relation to the ordering of the actual letters present in the input image.

4.3.2 Pathfinding

To obtain the desired text output, we need to walk the graph of CTC token probabilities. A path is produced by choosing a token at each of the time steps and assembling them into a sequence. The probability of any particular path is the product of the probabilities of its tokens:

$$P(s_1, s_2, ..., s_T) = \prod_{t=1}^{T} y(s_i, t) \tag{4.1}$$

where y is the matrix of probabilities returned by the model, $y(s, t)$ is the probability for the $s - th$ token in time step t and T is the total number of time steps.

One approach to assembling a path would be to use the so-called greedy search strategy, which simply chooses the path that maximizes this probability. However, one could also leverage the other paths to maximize the probability of the produced text instead:

$$y* = \underset{x}{\operatorname{argmax}}\, p(y|x) \tag{4.2}$$

However, it is not feasible to check all possible paths considering the exponential growth of the search space with the length of the input sequence. Over time many strategies were proposed to combat this problem. Beam search, seemingly the most common one amongst them, is an extension of the greedy search strategy that maintains a list of the top-k most promising paths, known as the beam width. At each time step, all possible extensions of the paths in the beam are considered, and the top-k paths are retained on the basis of their probabilities. Beam search can improve the solution quality compared to greedy search, but it comes at the cost of increased computation due to maintaining multiple paths.

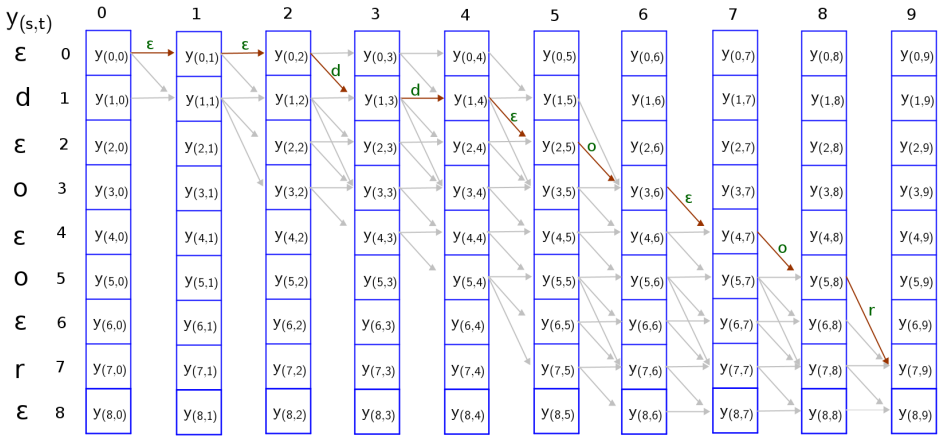

Figure 4.2: Paths leading to the valid output

4.4 Data

The training data consist of around 100000 fixed 128x32 pixel size images with corresponding labels. Each image is binary and white on black. The data comes from IAM dataset [6].

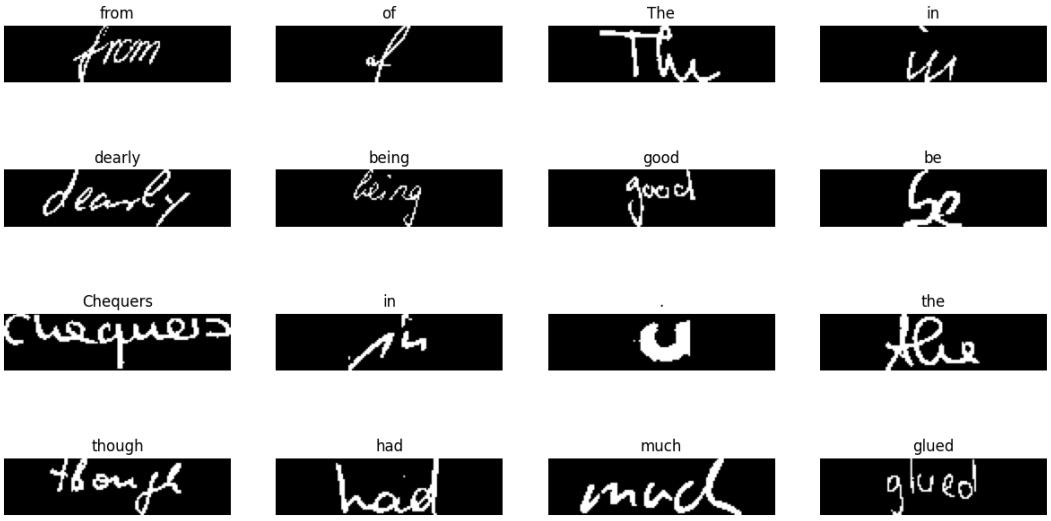

Figure 4.3: Exemplary data

4.4.1 Data processing

The unified binary format is essential since we need to narrow the gap between the IAM and application data as much as possible. We would also like to clean the data from any background noise. This cannot be achieved with just setting a static threshold that decides the binarization due to the data coming in different styles, illumination and artifacts. For this purpose we used the adaptive thresholding method [7] which computes the average of a window centered on a given pixel that is then used to decide the new value of said pixel.

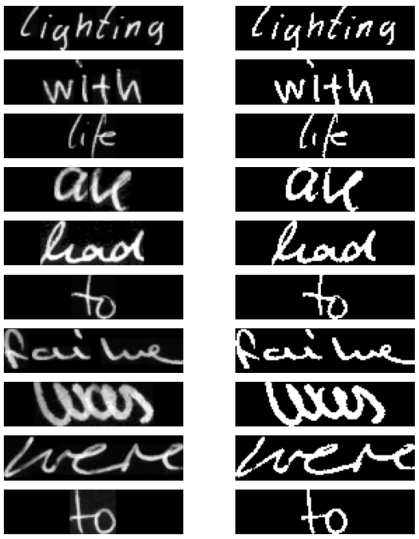

Figure 4.4: Image processing comparison

4.5 Spatial transformer

Spatial Transformer Network (STN) [8] is a neural network module that can learn to designate affine transformation parameters and then apply that transformation to the input image or feature map. The usefulness of the Spatial Transformer Network for handwriting recognition tasks lies in its ability to handle spatial variations and distortions in the input data. When dealing with handwritten characters, there can be variations in scale, rotation, and translation, making it challenging to accurately recognize and classify them. By incorporating the STN into the architecture, the network can learn to align and normalize the input images, making them more consistent and easier to process.

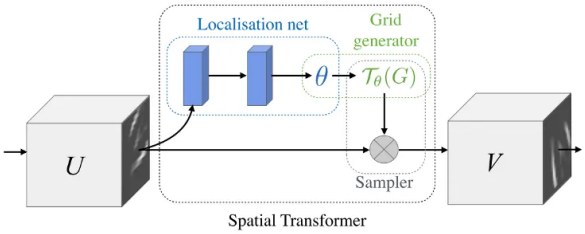

Figure 4.5: Spatial transformer module

The spatial transformer maintains an internal neural network that attempts to compute the optimal transformation parameters θ. Input is then passed to the sampler, which is a special layer that applies the transformation by sampling from a regular grid and interpolating the input.

4.5.1 Affine transformations

An affine transformation refers to a type of geometric transformation that preserves parallel lines and ratios of distances. It encompasses a variety of transformations, including translations, rotations, scalings, shears, and reflections. Mathematically, an affine transformation can be represented as $T_\theta(x) = \theta * x + b$.

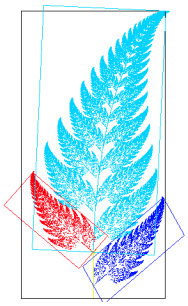

Figure 4.6: Variants of affine transformations where every leaf present can be transformed into every other leaf

4.5.2 Transforming images

To produce a transformed image we start with a regular output grid V of the desired size and map its coordinates to the input U. Notice that the mapping is continuous while our grids are discrete, and thus each pixel sampled from U is a result of interpolating its neighbors. Since we require this operation to be differentiable a bilinear interpolation is used.

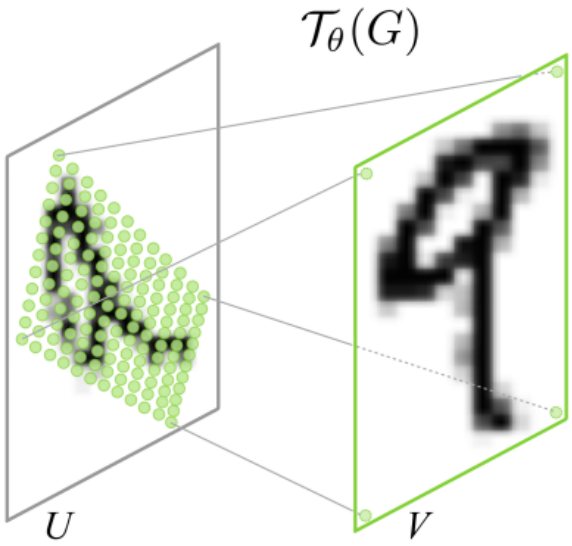

Figure 4.7: Sampling from a regular grid

4.5.3 Training

Spatial transformers are rather difficult to train, mostly because there are way more detrimental transformations than there are beneficial ones. This is troublesome since many of the undesired transformations lead to loss or even complete deletion of the information contained in the input. To address this we initialize the weights so that the transformer defaults to identity transformation at first. This guarantees a reasonable start, but the training process is still unstable; that is to say, loss would occasionally spike, and it would take several epochs to recover. We solve this issue by introducing an L2 weight regularization term to the loss function for the last regression layer in the localization network.

4.6 Model

4.6.1 Architecture

The architectural design of our handwriting recognition system revolves around the use of bidirectional LSTM layers for better feature processing and a spatial transformer for an improved feature extraction. The best results were achieved with Nadam optimizer, HeNormal initializer and hyperbolic tangent activations for the LSTM layers. Model is comprised of 579344 parameters and weights two megabytes.

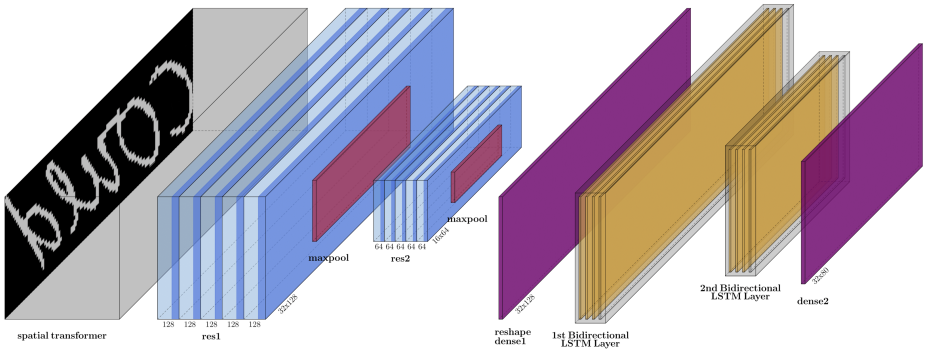

Figure 4.8: Handwriting recognition system architecture

4.6.2 Metrics

We evaluate our model performance using Character Error Rate (CER) computed with Levenshtein distance that measures the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another. The Levenshtein distance between two strings s and t can be calculated using dynamic programming. Let $|s|$ and $|t|$ represent the lengths of strings s and t respectively. We define a matrix D of size $(|s|+1) \times (|t|+1)$. The initialization steps are as follows:

$$D[i,0] = i \quad \text{for } i = 0 \text{ to } |s|$$

$$D[0,j] = j \quad \text{for } j = 0 \text{ to } |t|$$

The Levenshtein distance can be computed using the following recursive formula for $1 \leq i \leq |s|$ and $1 \leq j \leq |t|$:

$$D[i,j] = \begin{cases} D[i-1,j-1] & \text{if } s[i] = t[j] \\ 1 + \min(D[i-1,j], D[i,j-1], D[i-1,j-1]) & \text{otherwise} \end{cases}$$

The final Levenshtein distance is given by $D[|s|,|t|]$.

4.6.3 Results

The final model scores 22% CER on the test set. We find this to be acceptable since we are only looking for word matches anyway.

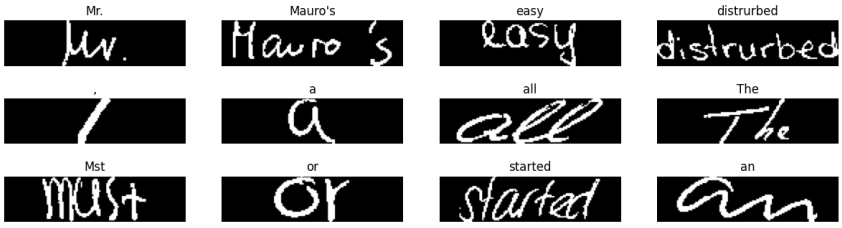

Figure 4.9: Model predictions

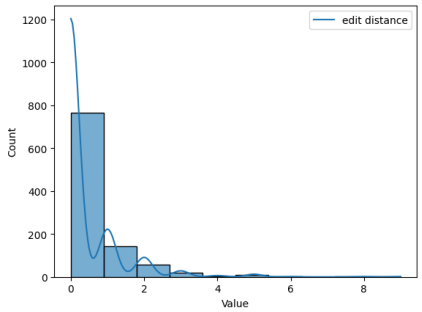

Figure 4.10: Edit distance distribution for the test set

In figure 4.11 we average classes across all time steps. This new probability distribution format represents the prevalence of each of the tokens in the image and allows us to run SHAP explainer on it. We see that the model adequately spots each of the letters present.

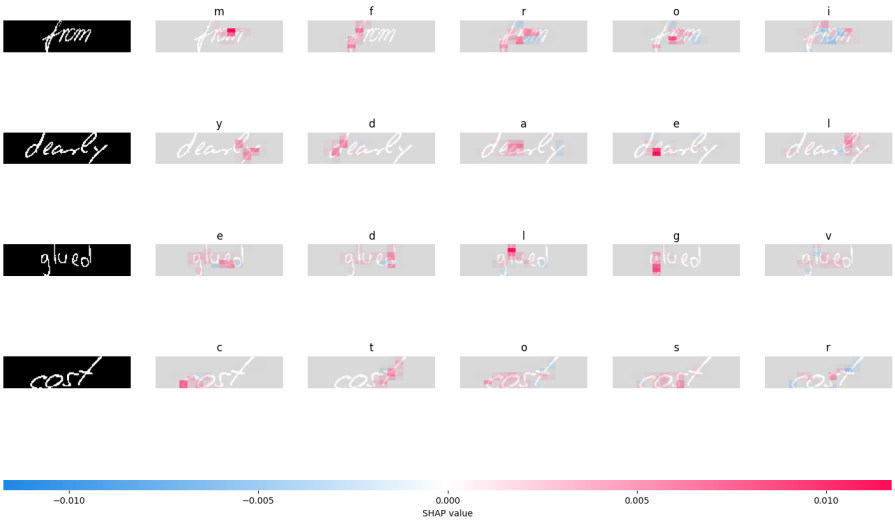

Figure 4.11: Shapley values for the most prominent characters

Chapter 5

Back-end Server

5.1 Overview

In the scope of this work, we were aiming to provide a way for multiple users to join single virtual board and work together on it. This functionality required a server that would set up connections between users. Since the front-end application turned out to be a big undertaking on its own, we decided to, for now, leave out some of the more complex features like board persistence. Our server simply keeps track of created boards and users that joined them, so that it can assist in exchanging information about the state of the board between them.

5.2 Technology stack

5.2.1 Express.js

We needed a web application framework that would reflect the minimalistic nature of its task while also being scalable and robust enough to handle anything that we might think of in the future. We decided to choose *Express.js* framework which is the most popular choice for the *Node.js* server environment. It provides basic set of functionalities for handling HTTP requests and introduces a concept of *middleweres* which are functions that can be chained together to handle requests. It can also be easily extended by external packages that provide some extra functionalities. In our case we used *Socket.io* package for bidirectional data flow between server and clients and *Redis* packege for communicating with database.

5.2.2 Socket.io

To assist in data exchange, the server needed a way to asynchronously inform users about changes made to boards. That is where the *socket.io* came in handy,

This package provides a framework for event-driven bidirectional communication between server and connected clients. Both the client and the server can emit and react to certain named events. For example, when user draws a new shape on his board, an event is emitted to the server with payload that contains all the information about the shape and the server reacts by broadcasting this event to all users connected to this board.

Broadcasting is achieved with Socket.io rooms. A room is a channel used to broadcast events to a subset of users. In our case a single room maps directly to some virtual board so when the user opens a board, on the back-end he also joins the room associated with it and receives all events directed there.

Chapter 6

User Manual

6.1 Creating a new board

After you enter the main page of the application, a new board will be created automatically and you will be redirected to the board view. This board will exist as long as at least one user is connected to it. This means that if you are the only user and you close the tab with the application (or refresh the site), the board will no longer be available after that.

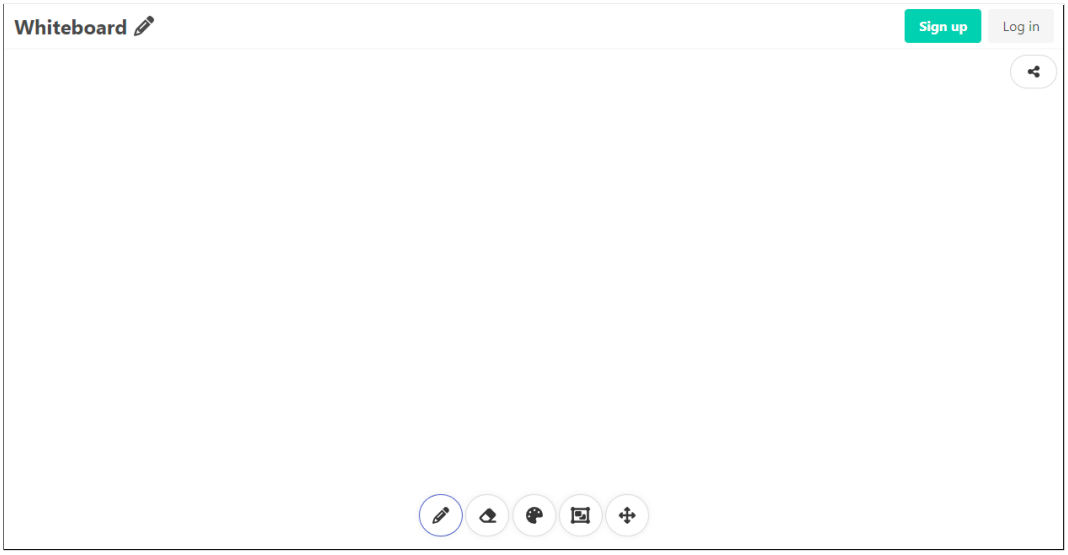

Figure 6.1: Board view on a widescreen device

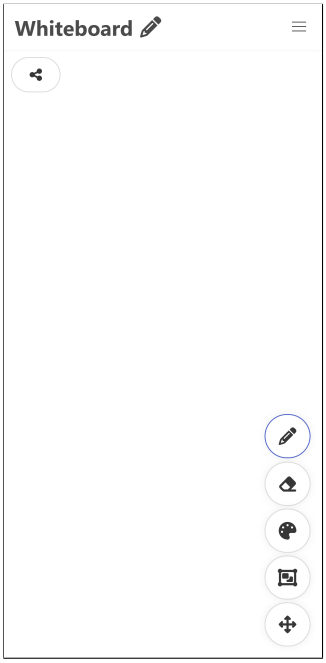

Figure 6.2: Board view on a mobile device

The toolbar is located at the bottom of the board canvas. On widescreen devices the toolbar is displayed horizontally in the center of the viewport while on mobile devices it is displayed vertically in the bottom right-hand corner.

At the top of the canvas we can see the share button. It is located on the right-hand side when on widescreen device or at the left-hand side when displayed on a mobile.

6.2 Drawing a shape

To draw a shape you need to select the draw tool from the toolbar. Button for the draw tool is shown on Figure 6.3. To start drawing, you need to press the cursor somewhere on the canvas and drag it. The path of the cursor will be traced to make a smooth line.

Figure 6.3: Draw tool button

6.3 Using shape correction

When using the draw tool, you can request shape correction. It will try to recognize the shape you are currently drawing and change it into one of the regular shapes

such as a rectangle, triangle, or ellipse. To do that, you need to hold your cursor still for a moment while drawing. After a while, a loading bar will appear over the cursor; at this point if you let go, correction will not take effect. This mechanism exists in case you were holding it unintentionally. If you hold it long enough and the shape is successfully recognized, it will replace the shape you were drawing.

(a) Unsuccessful shape correction (b) Successful shape correction

6.4 Changing stroke, width and fill color

You can change the default color, fill and width of the stroke of the drawn shapes. To do that, click the color tool icon in the toolbar which will expand the sidebar with color picker. You can then use it to set the desired color and width of the shape. To edit the fill properties instead, you need to select the the tab that says *Fill*.

Figure 6.5: Color tool button

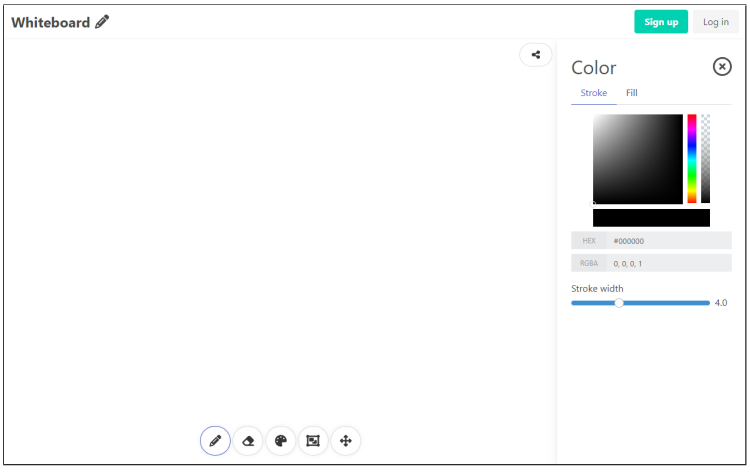

Figure 6.6: Application with the color sidebar expanded

6.5 Selecting drawn shapes

When you want to select one or more shapes to perform some action on them, you need to choose the select tool from the toolbar. Next you can draw a select box on the canvas and any shape fully contained in it will be selected. Selected shapes will be highlighted.

Figure 6.7: Select tool button

You can also use that tool to change properties of already drawn shapes. Simply select them and then open the color tool. Changing properties in color picker will affect all selected shapes.

6.6 Using handwriting recognition

Using handwriting recognition is quite similar to using shape correction. You need to select the written word using the select tool and hold the select box in place for a while. Just like before, a loading bar will appear and after a moment the drawn shapes will be replaced by the recognized word.

Figure 6.8: Successful recognition of a handwritten word

6.7 Moving around the board

In order to move across the canvas when viewing a board, first select the move tool shown in the figure 6.9 from the toolbar. Then simply press down on the canvas and drag it in the desired direction.

Figure 6.9: Move tool button

6.8 Sharing the board

To collaborate with other people over the internet you need to share your board URL with them. To do that you can click the share button (shown on figure 6.10) this will expand the sidebar. Link to your board will be visible in the top section. You can use the copy button to copy it to your clipboard.

Figure 6.10: Share button

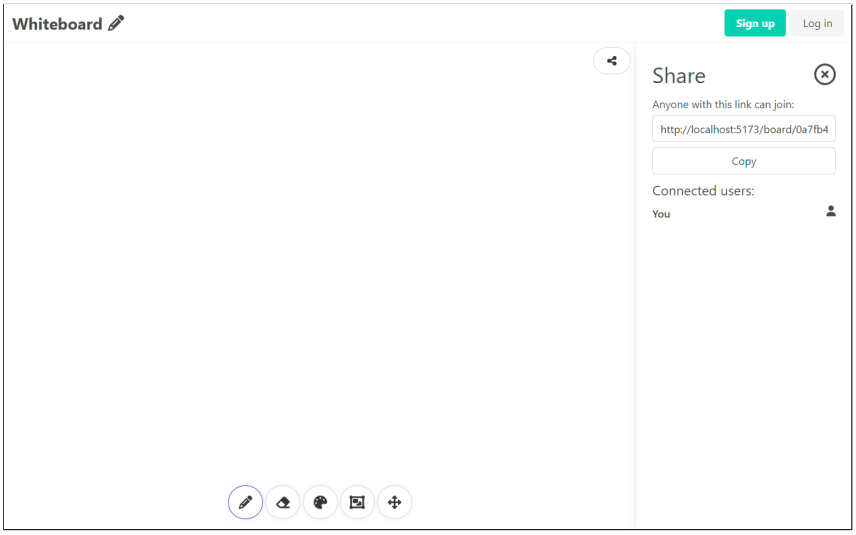

Figure 6.11: Application with the share sidebar expanded

After some other users join your board, you will see them in the list of connected users in this sidebar. Each user will have a random color assigned to them so you can tell them apart.

6.9 Erasing shapes

Using the erase tool you can remove previously drawn shapes from the board. Simply choose the erase tool in the toolbar and drag pressed cursor across the canvas. Any shape that intersects the path taken by the cursor will be removed.

Figure 6.12: Erase tool button

Chapter 7

Summary

In this work, we set out to design and develop a web application that would offer an alternative to the classical whiteboards. We were successful in implementing a working system that integrates front-end application with machine learning models.

Regrettably, we did not manage to provide a robust enough back-end to allow for storage of board states and user data. Also, we were aiming to provide a flexible user interface with lots of customization options, but eventually other aspects of the implementation turned out to be more urgent. Regarding our machine learning components there are still many potential improvements like enhancing handwriting recognition with attention paradigm and learning to perform shape correction on lines and images of arbitrary size that would vastly improve the reconstruction accuracy.

That being said, we are excited about the future of this project. There is still a long road ahead but we are confident that with enough perseverance we can make the project come to its full fruition.

Bibliography

[1] Kim Arvin S. Silvoza, Ryan A. Blonna, Rowel O. Atienza *A Natural Handwriting Algorithm for Tablets*, https://eudl.eu/pdf/10.4108/icst.mobilware.2013.254278

[2] https://dzone.com/articles/design-patterns-event-bus

[3] https://shap.readthedocs.io/en/latest/index.html

[4] https://christophm.github.io/interpretable-ml-book/shapley.html

[5] Nguyen et al. (2021). *Point-set Distances for Learning Representations of 3D Point Clouds*

[6] iam, https://fki.tic.heia-fr.ch/databases/iam-handwriting-database/

[7] Derek Bradley, Gerhard Roth (2007). *Adaptive Thresholding Using the Integral Image*

[8] Max Jaderberg, Karen Simonyan, Andrew Zisserman, Koray Kavukcuoglu. *Spatial Transformer Networks*

[9] Keras documentation, https://keras.io/about/

[10] Lime repository, https://github.com/marcotcr/lime

[11] https://eudl.eu/pdf/10.4108/icst.mobilware.2013.254278

[12] https://repositum.tuwien.at/retrieve/1835

[13] https://www.tensorflow.org/js/tutorials/conversion/import_keras

[14] http://cs231n.stanford.edu/reports/2017/pdfs/810.pdf

[15] Vue.js site: https://vuejs.org

[16] Bulma site: https://bulma.io/

[17] Express.js site https://expressjs.com/

[18] Socket.io site https://socket.io/