Hochschule Esslingen
University of Applied Sciences

Department of Computer Engineering

Classification of Large Biological Image Sets Within the Browser

Bachelor Thesis

Christoph Friedrich

Summer Term 2018

First Advisor: Prof. Dr. Jürgen Koch, University of Applied Sciences Esslingen
Second Advisor: David Dao, ETH Zürich

Declaration

I assure the single handed composition of this bachelor thesis only supported by declared resources.

Esslingen, May 29, 2018	
Place, Date	Christoph Friedrich

Contents

1. Motivation

These days a lot of data is collected to improve the way cars are used or phones are unlocked. Autonomous driving and face recognition are two examples where large data sets are required to improve algorithms and thereby make cars more capable and phones easier to use.

One of the most interesting areas large collections of data can be used with significant benefits is in health care. Modern technologies provide possibilities beyond anything that could have been conceived just a few years back. For example, artificial intelligence enables identifying malignant melanomas by discriminating them from harmless moles using a standard cellphone camera.

However, a lot of high potential data is still being unused. One reason is that computer laymen often don't know how to use the data they already have. It seems to be a privilege of computer scientists and computer enthusiasts to take advantage of such data and the potential it holds.

For example, if biologists and doctors had a tool that would help them classify and structure their specimen in an easy way, or even automate such work, it could vastly increase their productivity and diagnostic capacity.

The objective of this bachelor thesis is to evaluate how regular users without any computer science background could access the potential of machine learning in their daily work to assist them in performing intellectually demanding tasks.

Stated a little pathetically: This project wants to help democratizing access to artificial intelligence.

2. Architecture and Technology

This chapter is about introducing different software architectures and technologies that are used to build web applications. This chapter is structured as followed. First it shall be explained why web applications offer possibilities desktop application can not, a glimpse into JavaScript will be taken. After that a more detailed look will be taken on React and its underlaying software architecture.

2.1. Software Application Architectures

Most software applications fall into one of these categories:

- Desktop application
- Web application
- App for mobile devices

There is a clear trend towards web applications as they come with two significant benefits:

- No installation is required by the end user
- The runtime environment (web browser) is usually available on all platforms and standardized

This trend is reflected by the usage of JavaScript (JS) as the programming language of choice for web applications (see Fig. 2.1).

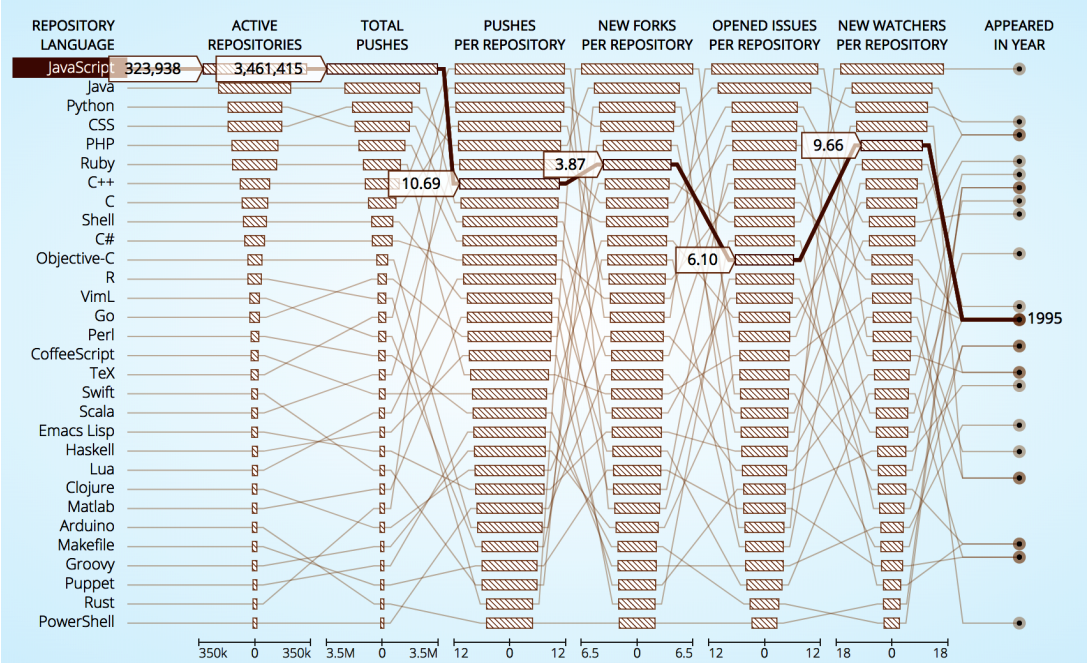

Figure 2.1.: Github Repositories Using JavaScript 2014
source:[3]

2.2. JavaScript

Today JavaScript is one of the most used programming languages world wide.

JavaScript first appeared in 1995. Originally developed at Netscape by Brendan Eich under the name of "LiveScript" it had a simple purpose, namely to dynamically manipulate the HTML Document Object Model (DOM) tree in the browser.

At about the same time a company called Sun worked on a programming language for embedded and mobile devices called Java. As Java became more and more popular, Netscape considered it a good marketing move to rename its LiveScript to Javacript, even though there was little technological similarity between these two languages.

Java is a regular, static, and highly typed programming language. It runs on a virtual machine and needs to be compiled, whereas the single threaded JS only runs in a browser and is script language.

Nevertheless they share a C related syntax and some similarities regarding naming

conventions. While Java is a true object oriented language, JavaScript has incorporated elements supporting object oriented programming only over time.

The following analysis shall show why Java Script is in many ways superior for web development. Jeff Attwood the co-founder of the computer programming question-and-answer website Stack Overflow and Stack Exchange once said "Any application that can be written in JS, will eventually be written in JavaScript". [1] [5] [7]

2.2.1. JavaScript Distributed Architectures

Most modern web designs rely on a three-tier architecture (Fig. **??**). On the server side computation is done and data is stored in databases. The client fetches data from the server and makes it accessible at the user interface. Often the user is able to change data on the client side.

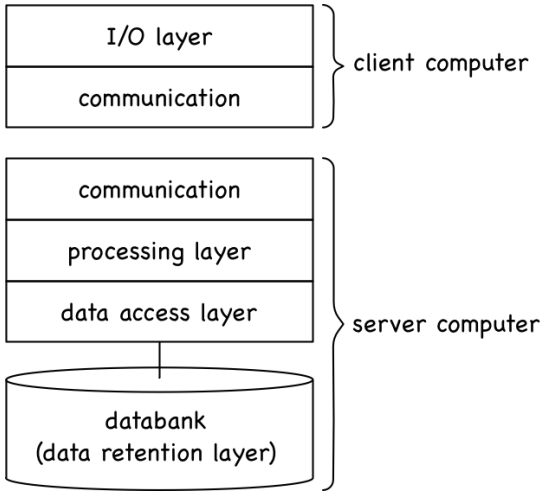

Figure 2.2.: Three-Tier Architecture
source:[8]

For example, to delete or add a user a request is sent to the server and a database change is done. For client side it facto mandatory to use JS, but on server side a large spectrum of languages is available, Java, CSharp, PHP, Ruby these are just the most famous one.

Once a user is added on client side, code is needed there in JavaScript. Same action

needs to be implemented in a different language on server, therefore a second implementation is needed for the same action. But the JavaScript ecosystem is undergoing rapid growth, with Node.js JS is coming to the server and developers are now able to create complete applications in JS.

That's big step, distributed systems that can use shared modules. It even affects the job market, front-end developers and back-end developers have been two separated jobs for over decades but now are one. People do not need to learn two languages anymore if they want to do full-stack web development, everything is covered with JavaScript.

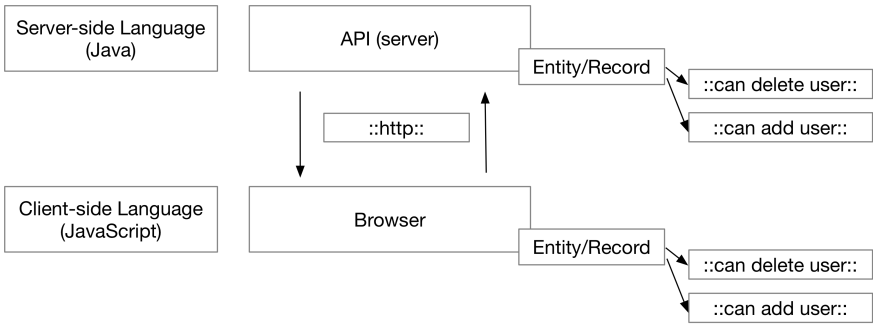

Figure 2.3.: No Shared Modules
source:[17]

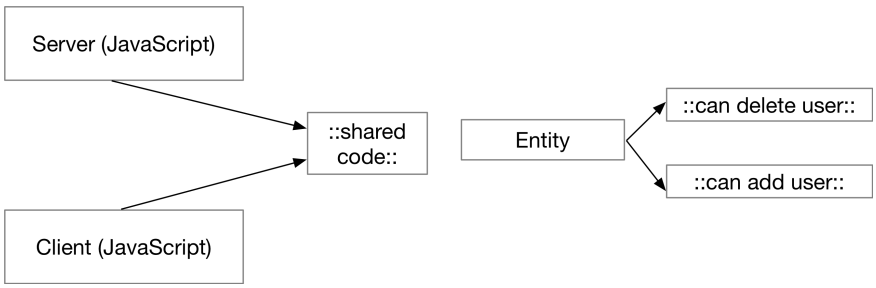

Figure 2.4.: With Shared Modules
source:[18]

2.2.2. Multi-Platform Support

Deploying an application for the web browser is only one aspect of modern web development, often a separate implementation is needed for mobile devices or even

desktop implementations. Two major operating systems are currently dominating the smart-phone market Android and IOS.

Both support native App developing but that brings back the problem of having to implement the same functionality twice. Sure smart-phone Apps are different from browser based applications, starting with the touch event and ending up with totally different layouts.

With JavaScript developers can take advantage of modern frameworks like React Native or Electron, which give the possibility to code in JS, the JavaScript is communicating natively with implemented components written in Java on Android, Objective C on iOS, CSharp on Windows via an abstraction called bridge. Everytime JS is called it forwards the call to the native code implementation, the response is passed back to the implemented abstraction JS.

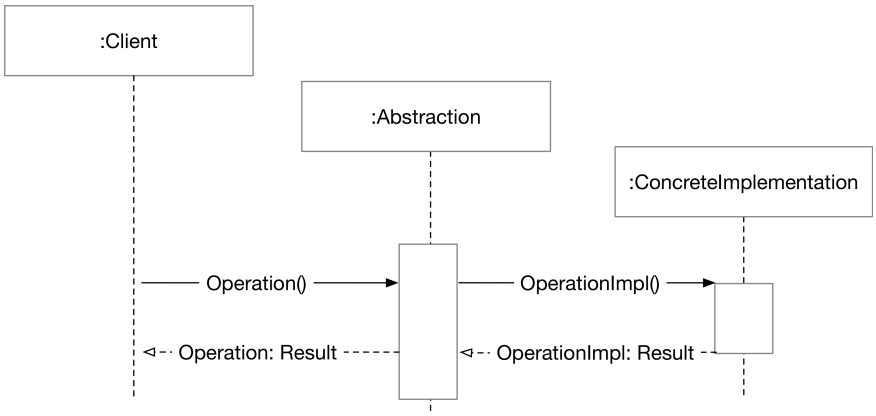

Figure 2.5.: Bridge Pattern
source:[19]

2.2.3. Single Threading and Concurrency

The single treaded paradigm and non blocking architecture JS is bond to, can be considered a weakness. Developing with JavaScript will generate errors and behavior that is perplexing . This section is about analyzing why JS is relying on single threading and how it is still possible to add concurrency.

JS was developed to manipulate the DOM, the DOM is a limited resource. Executing

two pieces of code, asynchronously changing the same part of DOM would not be good. Given this single purpose it was decided to make JavaScript single threaded.

Actions that do not change the DOM should be able to run asynchronously. That makes sense, since fetching data from the server or reading a large list of files can take a lot of time and would block the browser, buttons would not be clickable anymore and any rendering would stop.

How is it possible that JavaScript offers the opportunity to run code concurrently, even though it is single threaded? The following example how multi-threading is implemented in JavaScript.

Every time a function in JS is called, it is put on the call stack. The result of that function is popped out from stack. The functions on stack are executed synchronously one after another.

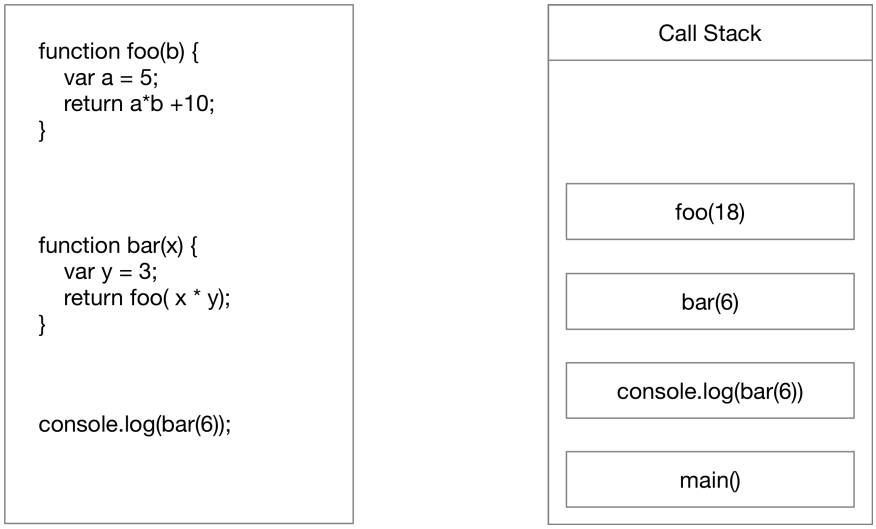

Figure 2.6.: Call Stack
source:[9]

To execute code concurrently JavaScript needs to call the Web API, provided by the browser. The web browser is written in C++, so it fully supports multi-threading.

The Web API is called with a function that needs to be executed and a callback. The callback tells what to do after the asynchronous function finished executing. It is

mandatory to provide a callback with each asynchronous function.

Once the thread finished executing it passes the callback to as so called Callback Que. The Callback Que is a simple list with all callbacks that need to be executed.

Further an event loop is responsible for checking if current stack is empty, if that's the case, the callback is put on stack and executed synchronously. The decoupling of the caller from the response allows for JavaScript to do other things while waiting for asynchronous operations to complete and their callbacks to fire.

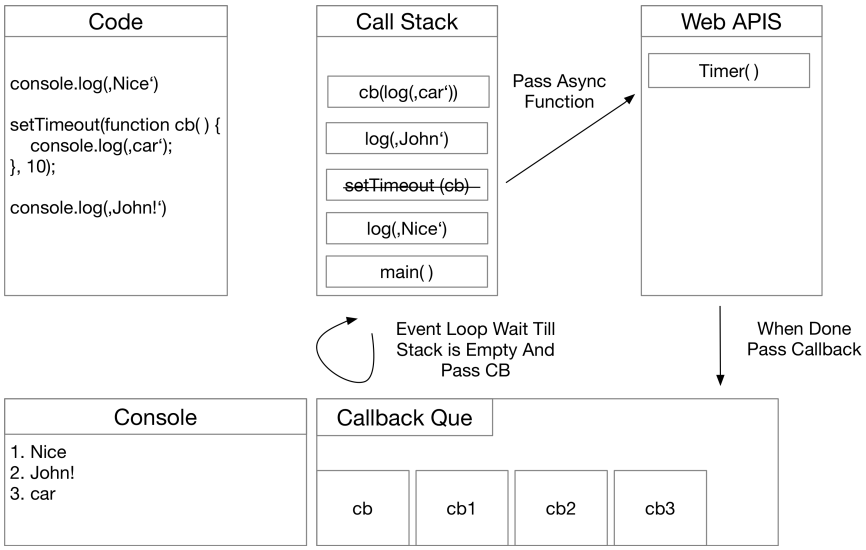

Figure 2.7.: Concurrency in JavaScript
source:[11]

2.2.4. Functional Programming Paradigm

There is no need to use functional programming in JavaScript, since this is more a philosophy question and definitely nothing somebody needs to do when programming in JavaScript.Other ways of coding, like object oriented programming, are supported in JavaScript as well. This section shall explain what functional programming is and how it is implemented in JavaScript. No global variables or states are used in functional programming, the output of a function only depends on the input arguments.

Not functional

```
var name = „Christoph“;
var greeting = „Hi, I’m“;
console.log(greeting + name)
```

Functional

```
function greet(name) {
    return „Hi, I’m“ + name;
}
console.log(greet(„Chris“))
```

Figure 2.8.: Imperative and Functional Programming
source:[16]

Another concept is the use of high-order functions. Giving the possibility to use functions as inputs and outputs.

High-Order Functions

```
function makeAdjectifier(adjective) {

    return function (string) {

        return adjective + „ „ + string;
        };
}

var coolifier = makeAdjectifier(„cool“);

coolifier („Car“)
```

Figure 2.9.: High-Order Functions
source:[15]

Data mutation shall be avoided.

Bad (Data is Mutated)

```
var rooms = [„H1“, „H2“, H3“];

room[1] = „H3“;
```

Good (No Mutation)

```
var rooms = [„H1“, „H2“, H3“];

var newRooms =
rooms.map(function(rm) {

    if( rm == „H3) { return „H4“;}

    else { return rm;}
});
```

Figure 2.10.: Data Mutation
source:[12]

2.3. React

Another key technology used in this project, is React. React is a JavaScript library for developing user interfaces.

Back in 2011, Facebook noticed that it was getting hard to maintain their Application and to run it flawlessly, because of the growing number of features. Many people were hired and the team size expanded dramatically. With the growing team size it took longer to publish urgent updates, too many people were involved and concerns could not be separated in a satisfying manner.

A Facebook engineer called Jordan Walke decided to change that, in the same year he created FlaxJs, first prototype of React. Jordan was allowed to keep on working and created React (2012).

A short time later Instagram was bought by Facebook and both companies agreed on using React as the new core technology for User Experience. Further they agreed on making React publicly available.

In early 2013 at JS ConfUS, React became an open source project. Facebook CEO Mark Zuckerberg, speaking on the this conference said "Our biggest mistake was betting too much on HTML5" and promised to provide better experiences with React.

Currently React is getting more and more popular. A trend analysis, by Google shows that React is the leading modern User Experience JavaScript framework.

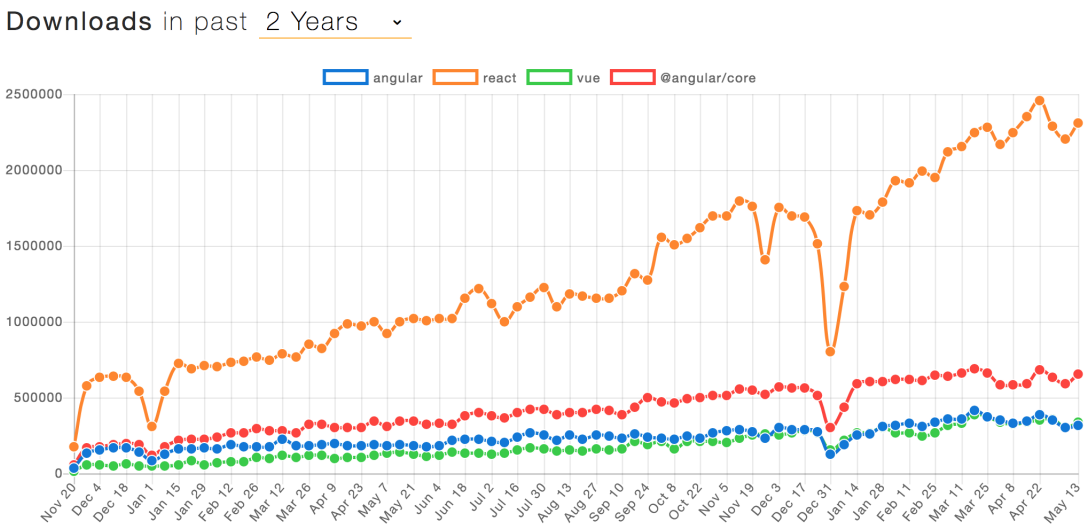

Figure 2.11.: Downloaded NPM Packages
source:[2]

2.3.1. Classic MVC Model

For decades it was best practice to use the Model View Controller software architecture. The model stores the data presented in a view, in simple systems the model may contain business logic.

The model shall be independent of views and controllers. In case the model changes the controller may inform the views (passive Model). With an alternative active model implementation, the model informs the views. One model can have multiple views.

The view serves to present model data to user, there can be many views on the same model data. All views are updated in case of a model data change. A view may furthermore present interactive elements like buttons.

Interaction with these elements creates events that usually are handled by the controller. The controller controls the model and view state, based on user input. It transforms events caused by user action into method calls of the model, the controller furthermore controls the state of view, for example activate or deactivate buttons.

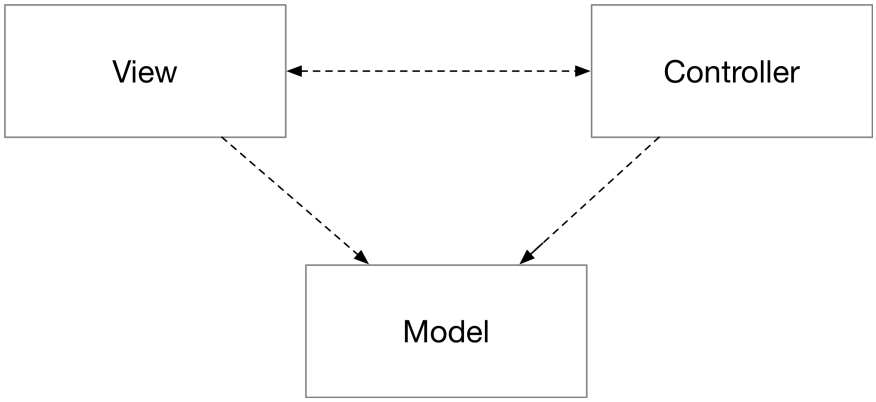

Figure 2.12.: MVC Pattern
source:[19]

The so called separation of concerns makes it easier to split work and, due to that it makes it easier to maintain code. But it also comes with an increased complex setup process, changes for example in the model or the controller affect the whole entity. The bidirectional communication in the MVC structure makes it hard to debug, changing one entity causes a cascading effect across the codebase.

React tries to get rid of that by introducing a different architecture called Flux. The following section shall explain how Flux is different from the MVC pattern.

2.3.2. Flux

Structure and Data Flow

In Flux data flows in a single direction, the unidirectional data flow is central to Flux. Dispatcher, stores and views are independent nodes with different inputs and outputs. The actions simply consist of objects with the new data.

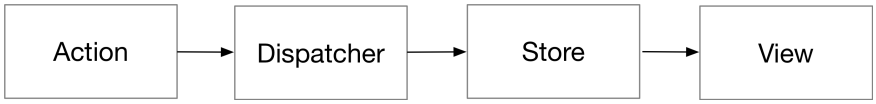

Figure 2.13.: Flux Software Architecture
source:[13]

The dispatcher takes care of handling all actions and forwarding it to the right store. The store holds the data and actions to change this data. Once data was changed the store alerts all views that are affected by this data change, causing a re-rendering.

It is also possible that a view generates an action, this happens mainly through user interaction. This action is also passed to the dispatcher.

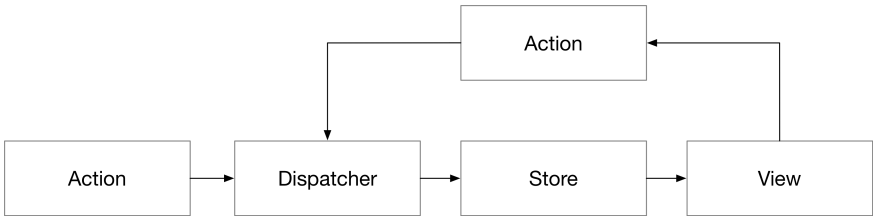

Figure 2.14.: Flux With View Action
source:[14]

Dispatchers

Dispatchers can be considered as a central hub that is managing all data flows in a Flux application.

The only way to change data, is through actions the dispatcher is providing. A dispatcher has no real intelligence of its own, it only provides a simple mechanism to distribute action over several stores.

This becomes more important when big applications shall be built. The dispatcher then handles dependencies between different stores, he takes care of updating the different stores in a specific order.

Stores

Stores contain the application's logic and sates. This role can be considered somehow similar to a model in the traditional MVC,the difference is that stores manage the data of many objects and not a single record of data like ORM models do. They manage the application state for a whole domain.

For example Facebook's Lookback Video Editor utilized a TimeStore that kept track of the playback time position and the playback state. On the other hand, the same application's ImageStore kept track of a collection of images.

Views and Conroller-Views

In React all views are composable and freely re-renderable. at the top of each view, the highest in hierarchy, there is a view that listens to events that are sent by the store. It is called a controller-view.

The controller-view fetches the data from store and passes it down to all descendants, causing a re-rendering of that views. Often the entire state of a store is passed down a chain of views allowing each descendant to take what he needs.

Actions

As already mentioned action change the store . Each action has payload with new data, every time an action occurs a dispatch is triggered to the stores. User-interaction is not the only of actions, it can also come from other places like the server.

Conclusion

The Flux architecture is improving data consistency, it offers better possibilities to debug. The unidirectional data flow makes that much easier. Since one can always follow the flow of actions.Furthermore with having the state and all the logic that is updatitng the state at one place it is also possible to do more meaningful unit tests.

2.4. React and Flux

This section is about analyzing how React is implementing the Flux Software Architecture. React depends on encapsulated components that manage their own state.

Components are written in JavaScript, so data can easily passed through the Application. Everything that needs to be rendered is still written in HTML and parsed by React. Each component has its own controllers, there is no more discrimination between what is shown in the Browser and business logic.

There are no separated HTML and JavaScript files, only JavaScript. Every component is fully standalone and testable by its own. This makes React scalable and easy to test. There are no cascading dependencies. Every time the state of a component changes, the render function is called and the HTML is re-rendered with the changed data. Components can be nested, e.g. a board game that consists of squares.

```
1 class Board extends React.Component {
2   renderSquare(i) {
3     return <Square value={i} />
4     }
5 }
```

Each of that squares is a component and part of the whole board game application. There shall bee a value assigned to the squares, values are passed through the prop value.

```
1 class Square extends React.Component {
2   render() {
3     return (
4       <button className="square">
5         {this.props.value}
6       </button>
7     );
8   }
9 }
```

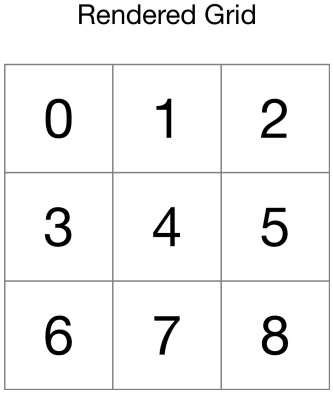

Figure 2.15.: Grid with numbers
source:[6]

Now when clicked on a square, this square shall show the 'X'. The value can now be considered a local state, it is definitely a private part of the square. First an initial state needs to be added.

```
1  class Square extends React.Component {
2    constructor(props) {
3      super(props);
4      this.state = {
5        value: null,
6      };
7    }
8
9    render() {
10     return (
11       <button onClick={() => this.setState({value: 'X'})}>
12           {this.state.value}
13       </button>
14     );
15   }
16 }
```

If this.setState is called, an update is scheduled by React and the value is merged in the correct component state. Furthermore the component and all of its descendants are re-rendered. If a Square was clicked it would now show a 'X' in the grid.

Lifting State Up

Often data needs to be aggregated from multiple child components. Then it makes sense to lift all states up to the parent component. The parent component then passes down the individual states for the child components.

E.g. in a Tic Tac Toe game, to determine who has won, the value of all squares components would need to be checked. That is technically doable, but a better approach is to save all states in the parent component and the parent component can simply check one array, in order to determine who has won.

The Square component is no longer keeping its own state, it receives the value from its parent Board. It informs the parent when it was clicked. These components are called controlled components.

```
1  class Board extends React.Component {
2    constructor(props) {
3      super(props);
4      this.state = {
5        squares: Array(9).fill(null),
6        winner: null
7      };
8    }
9
10   renderSquare(i) {
11     return <Square value={i} />;
12   }
13
14   calculateWinner(this.state.squares){
15       ...
16       this.setsState({winner: whoeverWon})
17  }
18
19   render() {
20     const status = 'Next player: X';
21
22     return (
23       <div>
24         <div className="status">{status}</div>
25         <div className="board-row">
```

```
26            {this.renderSquare(0)}
27            {this.renderSquare(1)}
28            {this.renderSquare(2)}
29          </div>
30          <div className="board-row">
31            ...
32          </div>
33          <div className="board-row">
34            ...
35          </div>
36        </div>
37      );
38    }
39  }
```

Virtual DOM

Manipulating the DOM is the main thing modern, interactive web applications do. Unfortunately it is a lot slower than running JavaScript. The HTML DOM is always tree structured, which makes it easy to traverse through it, but it also offers poor performance.

Additionally some modern frameworks update the DOM more often than it needs to be updated. For example changing one item in a list of ten items would lead to re-rendering all ten items. React is addressing this problem by introducing a virtual DOM.

The Virtual DOM is an exact representation of the HTML DOM with JavaScript. Once states have changed the virtual DOM is updated first, then the previous Virtual DOM is compared to the current Virtual DOM. Only what is different will be updated in the HTML DOM.

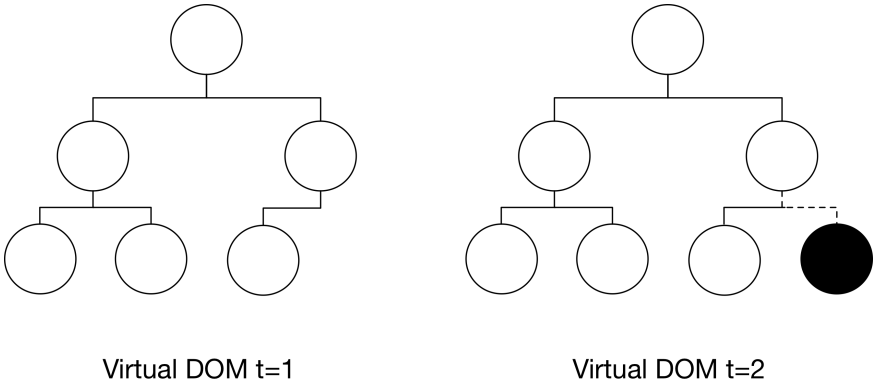

Figure 2.16.: Virtual DOM Camparison
source:[?]

Redux

Redux is a predictable state container for JavaScript apps. It makes applications more consistent, by organizing all in the app. Redux evolves the ideas of the Flux software architecture.

The gist of Redux is that all states are kept as an object tree ,in a single store. It is only possible to change this object tree by emitting an action, this action is an object describing what happened. To specify how these actions change the state tree, pure reducers are written.

The only difference to Flux is that Redux does not have a Dispatcher or support many stores. There is just a single store with a single root reducer. In bigger applications the root reducer can be split into smaller reducers, each operating independently on the different parts of the state tree.

Redux adds a lot of overhead to an application, a lot of more files and code are created. Considering that Redux should only be used if the following points are true.

- Reasonable amounts if data is changing over time
- A single source of truth is needed

- Keeping all of the states in a top-level component is no longer sufficient.

```
class Board extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      squares: Array(9).fill(null),
      winner: null
    };
  }

  renderSquare(i) {
    return <Square value={i} />;
  }

  calculateWinner(this.state.squares){
      ...
      this.setsState({winner: whoeverWon})
 }

  render() {
    const status = 'Next player: X';

    return (
      <div>
        <div className="status">{status}</div>
        <div className="board-row">
          {this.renderSquare(0)}
          {this.renderSquare(1)}
          {this.renderSquare(2)}
        </div>
        <div className="board-row">
          ...
        </div>
        <div className="board-row">
          ...
        </div>
      </div>
    );
  }
}
```

3. Deep Learning

Not only in Computer Vision, such as object detection but also in natural language processing or speech recognition, and even human-level video game play deep neural networks are used to solve domain specific tasks. In this chapter it shall be explained how those networks function by going into details with a feed-forward neural network. Furthermore a glimpse is taken about tricks of the trade such as batch normalization and initialization.

3.1. Feed-Forward Neural Networks

A feed-forward neural network consist of one input layer, H hidden layers and one output layer. Each layer consist of units (or neurons) where each unit of a given layer is fully- connected to every unit of the previous layer. The graphical representation can be described mathematically as a combination of matrix multiplication and activation functions. The activation function models when an artificial neuron fires and is usually a non-linearity. In theory, it is this non-linear activation function, which allows the network to learn any function approximation. Each unit in a hidden layer can be mathematically described by:

$$y_j = f(z_j) \tag{3.1}$$

$$z_j = \sum w_{ij} * x_i + b_j \tag{3.2}$$

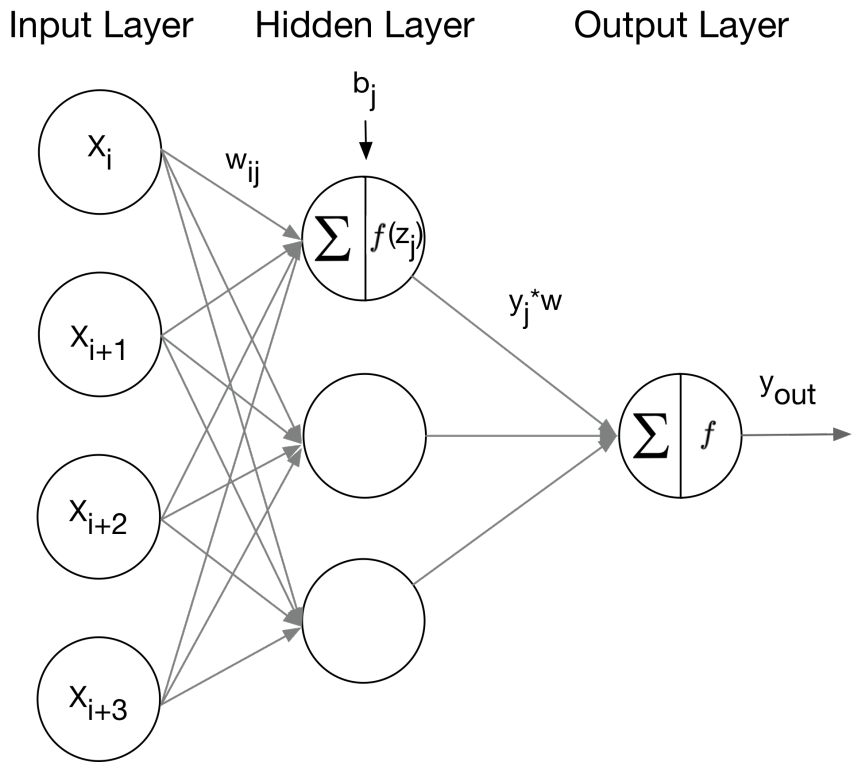

Figure 3.1.: Schematic view of a FCN. Schematic of a simple two layer fully connected neural network (FCN)

source:[?]

Where y_i is the output of the unit from the current layer. The output is transformed with a non-linear activation function. z is called the preactivation and w denotes the edge weights. x_i is the output of a unit from the previous layer. At every layer first the total input z is computed, z is the weighted sum of the outputs from the layer before. After that a non linear function $f(\cdot)$is applied to z in order to get the output y_j of the unit.

All differential functions can be used as a activation function of a neural network. Most common activation functions are sigmoid, $f(z) = \frac{1}{exp(-z)}$, the hyperbolic tangent, $f(z) = \frac{exp(z)-exp(-z)}{exp(z)+exp(-z)}$ or the rectifier linear unit (ReLU) $f(z) = max(0; z)$. ReLu will be used for all future examples.

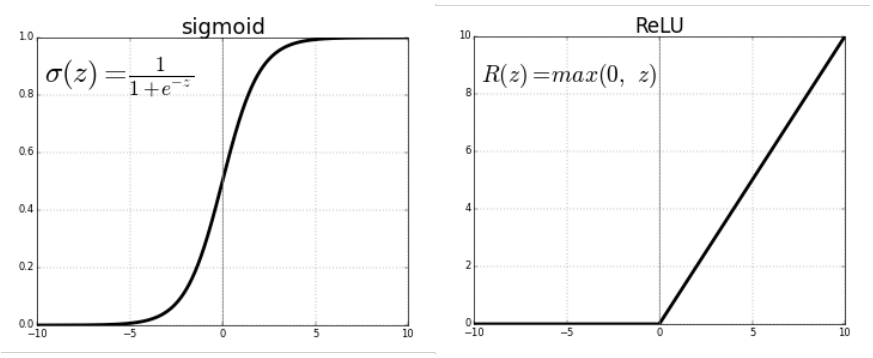

Figure 3.2.: Sigmoid And ReLu Activation Function
source:[?]

3.1.1. Backpropagation

To train a neural network two things need to be done. First a forward pass, second an error backpropagation. Given a certain input, a forward pass calculates the output of each unit. A predefined cost function E then compares the resulting outputs y_{out} with the correct answer and determines the error. A common cost function for example is squared error loss.

$$E = \sum \frac{1}{2}(target - y_{out}^2) \tag{3.3}$$

Given this error it is now possible to check how weights need to be adjusted in order to bring the E to a minimum. By applying chain rule of derivatives.

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x} \tag{3.4}$$

Taking the Error in consideration the algebra looks like this:

$$\Delta_W E = \frac{\partial E}{\partial W_l} = \frac{\partial E}{\partial y_{out}}\frac{\partial y_{out}}{\partial z_{out}}\frac{\partial z_{out}}{\partial y_{n-1}}\frac{\partial y_{n-1}}{\partial z_{n-1}} \tag{3.5}$$

Using the calculated gradient $\Delta_W E(y_{out})$ it is now possible to determine next update

for the weights matrix by a gradient descent update:

$$W_l^{t+1} = W_l^t - \eta \Delta_{W_l^t} E(y_{yout)} \tag{3.6}$$

Where W^t, W^{t+1} are the current weights and the weight matrix for the updated weights. η is the learning rate.

3.1.2. Weight Update

It requires a lot of computation time to calculate the gradient from whole dataset. Mini-batches can be used to calculate the gradient only based on a few sampled data points. Using this analytic gradient a parameter update is performed. There are several ways to do this update. But the most common way is Stochastic gradient descent (SGD). Parameters are simply changed along the negative gradient direction in order to minimize the error.

3.1.3. Initialization

To train a neural network initial weights need to be set. The most common way is a random initialization were every value is randomly set for example from a Gaussian.

3.1.4. Batch normalization

To avoid too big values or too small values in data a normalization is performed.

$$x \Rightarrow \hat{x} = \frac{x - \mu}{\sigma} \Rightarrow x_{norm} = \gamma \hat{x} + \beta \tag{3.7}$$

μ and σ are mean standard deviation of the dataset. *gamma* and σ scale and shift the parameters. Batch normalization accelerates training time and makes a deep neural

network less sensitive to initialization issues.

3.2. Classification

A really common problem for neural networks is the classification of images, or more general the classification of data into a specific category. The best performing neural network architecture for classifying images are convolutional neural networks. All convolutional neural networks consist of convolutional layers followed by pooling layers and some fully connected layers.

3.2.1. Convolutional layer

In the convolutional layer several kernels are applied to extract spacial related features. Each output from the the convolutional layer is called a feature-map. Each feature-map was created by a different convolutional kernel of the layer.

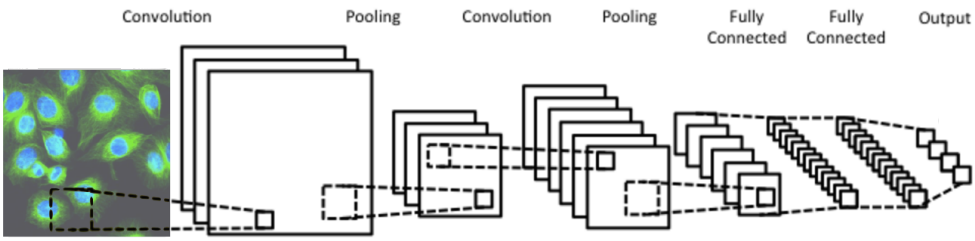

Figure 3.3.: Schematic view of a FCN. Schematic of a simple two layer fully connected neural network (FCN)

source:[?]

3.2.2. Pooling layer

After features were extracted into feature maps the pooling layer reduces the size of the feature maps and makes the computation tractable. A very simple implementa-

tion of a pooling layer is the max-pooling layer, which uses a sliding windows over the input and selects the maximum of each window.

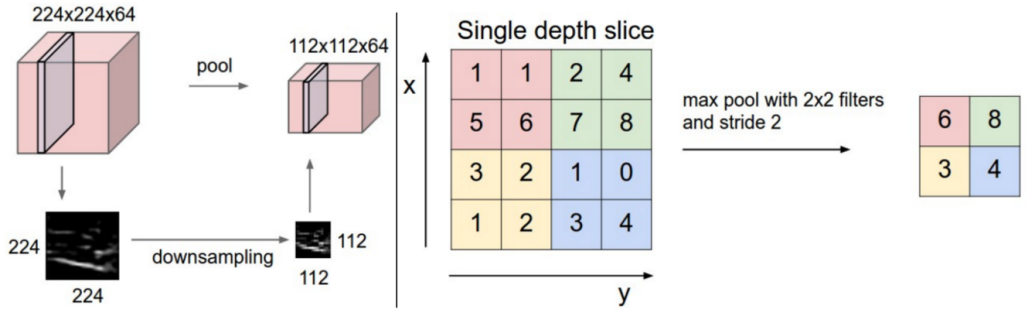

Figure 3.4.: Schematic view of a FCN. Schematic of a simple two layer fully connected neural network (FCN)

source:[?]

3.2.3. Data augmentation

Convolutional neural networks need a lot of data to generate suffice output. If not enough data can be provided the data can be artificially augmented by using different approaches. Pictures can be turned or flipped, cropped, blurred or even resized. This makes it possible to generate a lot of data which can be used to further improve training.

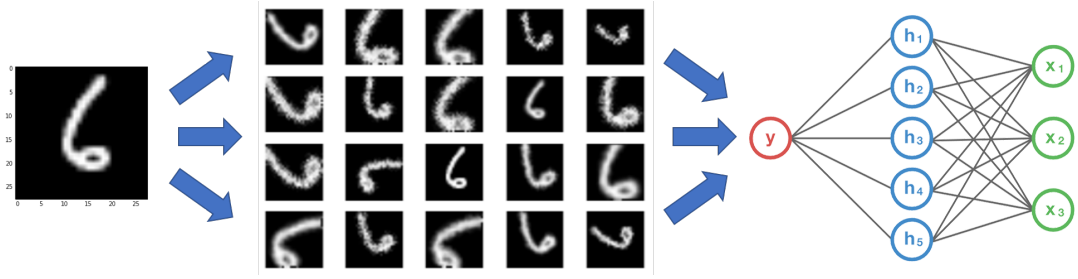

Figure 3.5.: Data Augmentation

source:[?]

4. Related work

This chapter is about introducing CellProfiler Analyst (CPA). CPA is a first generation tool for interactive data exploration and classification of large biological image sets. It was first introduced in 2016 at Broad Institute of MIT and Harvard []. The following section is separated into an overview of CPA's main functionality and an exploration of its key features, Classifier and Image Gallery. This project is highly influenced by CPA and aims to improve it. The software is free and open source, available at http://www.cellprofiler.org and from GitHub under the BSD-3 license. It is available as a packaged application for Mac OS X and Microsoft Windows and can be compiled for Linux.

4.0.1. CellProfiler Analyst

CPA is an GUI based tool. It provides easy possibilities for data exploration and classification via an interactive user interface. There are only a few comparable tools for this purpose. Most common tools are Ilastik [], CellCognition[] and WND-CHARM [].

All of these tools lack the possibility of suitable companion visualization tools, made for big datasets. Also they miss selectable classifier algorithms. Further CPA adds a big advantage to all existing tools by providing interactive object classification and image viewing.

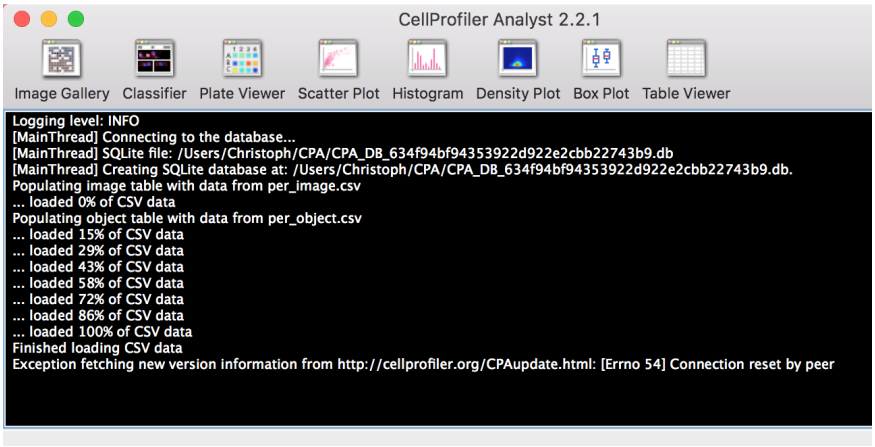

Figure 4.1.: CPA Main View Different Views Are Selectable
source:[6]

4.0.2. Classifier

Cell Profiler Analyst Classifier makes it possible to create categories and to classify cells. After fetching images from a predefined dataset it is possible to classify images by simply drag and dropping it on a category. Multi select features are also implement and enable to drag and drop multiple images at the same time. Further it is possible to add an unlimited amount of categories by clicking the add category button. After annotation is done the algorithms can be trained. Once training is done more images can fetched and the be automatically annotated by clicking the evaluate button. If results are not suffice further images can be annotated to train the algorithm in order to improve results. Different algorithms can be used to classify the uncategorized images, such as RandomForest Classifier, AdaBoost Classifier and many more.

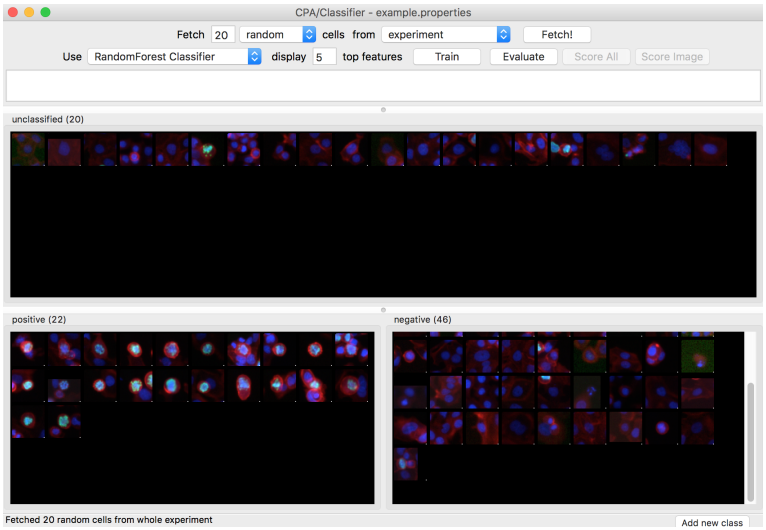

Figure 4.2.: CPA Main View Different Views Are Selectable source:[6]

4.0.3. Visualization

In order to explore the dataset and to prediction and validation result a good visualization is needed. CPA offers different ways of displaying data and results. One simple visualization possibility is the Image Gallery in were images from the dataset can selected and be watched in their original size. Filters can be applied to only select images with a experiment specific meta data. Their are many more ways to display and explore data in CPA.

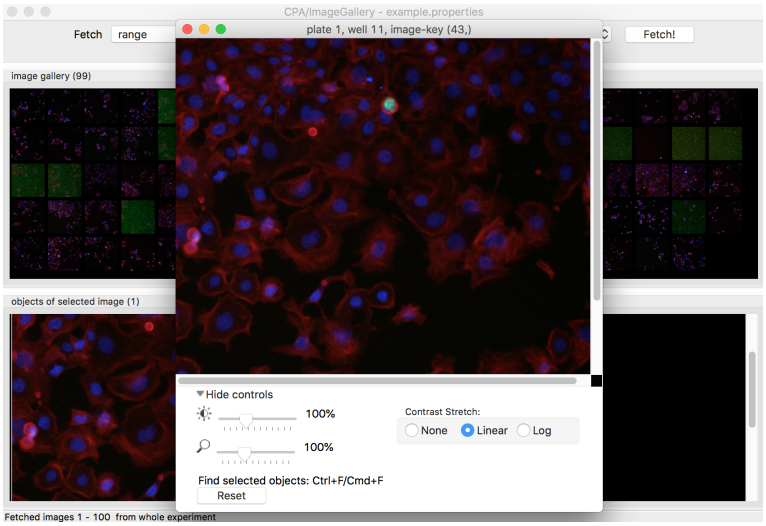

Figure 4.3.: CPA Main View Different Views Are Selectable
source:[6]

4.0.4. Downsides of CPA

To make CPA running a few steps need to be done first. A potential user needs to install and set up a MySQL database. Further configuration files need to be written. The powerful user interface is also complex. Considering these things a potential user can be scared off before even using CPA.

5. CYTO AI

CYTO AI is the first fully web-based image labeling and classifying tool there is. Due to the web-based design no installation of databases or any configuration is needed to use all advantages of modern machine learning and convenient labeling. Further no data will be uploaded to any server, all data never leaves the browser which makes CYTO AI very performant and sets a high standard of privacy at the same time. The following sections shall explain how CYTO AI works and which techniques and architectures were used to build it.

5.0.1. System overview

First pictures need to be uploaded by clicking on the upload button, whole folders can be selected. Mind that also all images in subfolders will be uploaded. In order to categorize images, categories need to be created. That is possible by clicking on the plus icon in the categories list. After all necessary categories have been created images can be annotated. There are several was to annoate an image. It is possible to drag an image and drop it on the wished category. A different way is to click on one picture in order to select it an then use the keyboard to annotate the image. Following keys are supported:

- [1] [2] [3] [4] [5] [6] [7] [8] [9] [0] where [1] is the first category in list.
- [⇐] backslash to delete a given category

Further it is possible to navigate through all images with the arrow keys [⇑] to go up in row and [⇓] to go down in row. Further [⇒] to go right in column and [⇐] to go left in column.

Also it is possible to blend out certain categories by clicking on the category, clicking another time will blend in the category. That gives the possibility to only show certain categories or to only show unlabeled images. If all categories are blended out it is still possible to annotate using this category the new labeled image will stay visible.

If pictures were labeled it makes sense to sort them, this is possible by clicking the sort button. This improves the overview and makes reviewing labels easier. Uncategorized images will always appear at the top.

Another feature that improves the overview is the slider. The slider makes it possible to adjust the number of displayed images per row.

If pictures have at least been labeled with two different categories it is possible to automatically categorize all unlabeled images by clicking the "fit" button. By that all unlabeled images will be given a category and a number will be shown under the image. This number is the probability an image belongs to the given category.

Figure 5.1.: CYOTO AI
source:[?]

To save all labels and categories and settings the button can be pressed. To import again the Open button can be used.

5.0.2. System architecture

Component structure

Machine Learning API

5.0.3. Performance

5.0.4. Future perspective

6. Zusammenfassung

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Bibliography

[1] A brief history of JavaScript ? Ben Aston ? Medium.

[2] Downloaded npm packages.

[3] GitHut - Programming Languages and GitHub.

[4] GitHut - Programming Languages and GitHub. Last accesed on 2018-05-15.

[5] JavaScript is the Future of Enterprise Application Development.

[6] The logo for the react open source javascript library.

[7] Wie unterscheidet sich JavaScript von Java?

[8] Christoph Friedrich. Three-tier architecture.

[9] Christoph Friedrich. Call stack, 2018.

[10] Christoph Friedrich. Complex mvc example, 2018.

[11] Christoph Friedrich. Concurrencv, 2018.

[12] Christoph Friedrich. Data mutation, 2018.

[13] Christoph Friedrich. Flux software architecture, 2018.

[14] Christoph Friedrich. Flux with view action, 2018.

[15] Christoph Friedrich. High-order function, 2018.

[16] Christoph Friedrich. Imperative and functional programming, 2018.

[17] Christoph Friedrich. No shared modules, 2018.

[18] Christoph Friedrich. With shared modules, 2018.

[19] Joachim Goll.

[20] Lothar Papula. *Mathematische Formelsammlung für Wissenschaftler und Ingenieure*. Vieweg Verlag, 2003.

[21] Gartner. M. arktanteile der betriebssysteme am endkundenabsatz von smartphones weltweit von 2009 bis 2017.

List of Figures

List of Tables

Listings

A. Anhang zum Systementwurf

Allgemeine Beschreibung des Anhangs

A.1. Diagramme

Hier werden Diagramme platziert, die in den Textkapitel zuviel Platz beanspruchen.

A.2. Tabellen

Hier werden Tabellen platziert, die in den Textkapitel zuviel Platz beanspruchen.

A.3. Quellcodelistings

Hier werden Tabllen platziert, die in den Textkapitel zuviel Platz beanspruchen.