Hochschule Esslingen
University of Applied Sciences

Department of Computer Engineering

Web-Based Deep Learning Algorithms for Classification of Human Cells

Bachelor Thesis

Christoph Friedrich

Summer Term 2018

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

Declaration

Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

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

Esslingen, June 9, 2018
Place, Date

Christoph Friedrich

Contents

List of Figures

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 provide a solution to regular users without any computer science background so they can access the potential of machine learning in their daily work to assist them in performing intellectually demanding tasks. The thesis was developed in the context of biological cell qualification, but can be applied to any other domain as well.

2 Architecture and Technology

This chapter is about introducing different software architectures and technologies that are used to build web applications. The first section discusses the advantages and disadvantages of web applications versus desktop applications. The second part deals with JavaScript as the primary programming language for web application user interfaces. Thereafter we will take a more detailed look into a modern framework for web-based user interfaces called React and its underlying software architecture Flux. Complex user interfaces built on React can be further enhanced using a framework called Redux, which will be discussed in the fourth section. And finally we will introduce TensorFlow.js as the key library for machine learning within browsers.

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 popularity of JavaScript as the programming language of choice for web applications (see Fig. 2.1).

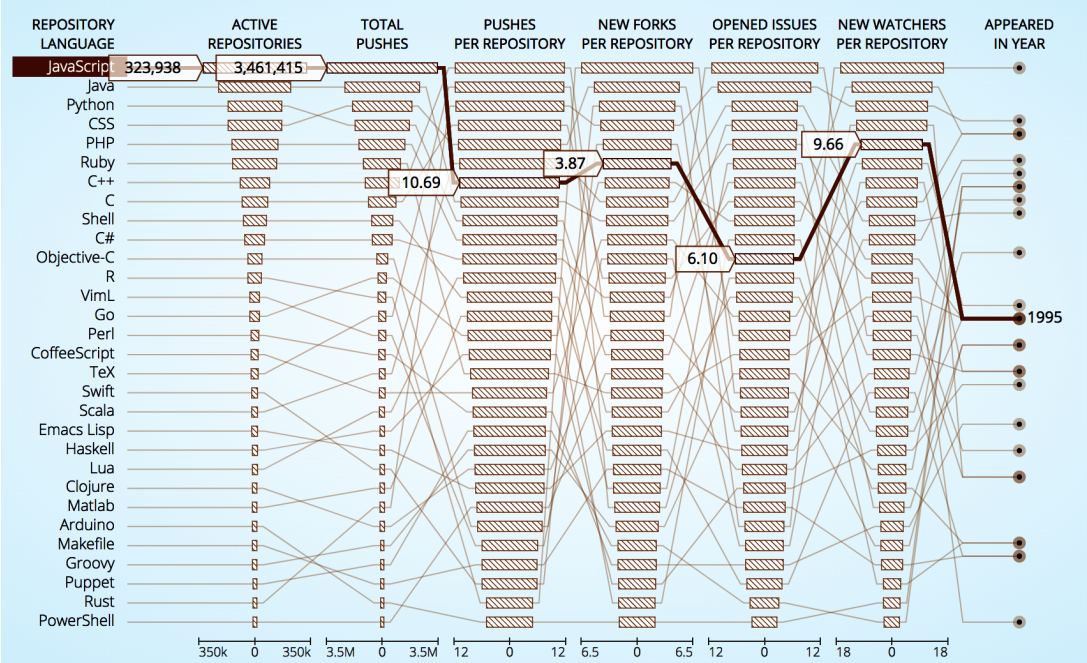

Figure 2.1: Numbers of Github repositories using JavaScript 2014 [1]

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 increasingly popular,

Netscape considered it a good marketing move to rename its LiveScript to JavaScript, 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 JavaScript only runs in a browser and is a script language.

Both languages have in common a C related syntax and exhibit some similarities with regard to naming conventions. While Java is a true object oriented language, JavaScript has incorporated elements supporting object oriented programming only over time [16].

The JavaScript community considers this language superior for web development. Jeff Attwood, the co-founder of the computer programming question-and-answer website Stack Overflow and Stack Exchange, even said: "Any application that can be written in JavaScript, will eventually be written in JavaScript" [17].

2.2.1 JavaScript distributed architectures

Most modern web designs rely on a three-tier architecture (Fig. 2.2). The bottom tier is usually a database system, responsible for storing and retrieving data. Typical technologies employed here are relational database management systems (RDBMS) and, more recently noSQL database systems like key value stores [2].

The second tier is usually responsible for executing the application logic. The technological base for this layer is Java, .NET, and more recently Node.js.

The top most tier is typically responsible for the user interface, but can also contain application logic. The most widely used technology nowadays is a web browser with JavaScript programs.

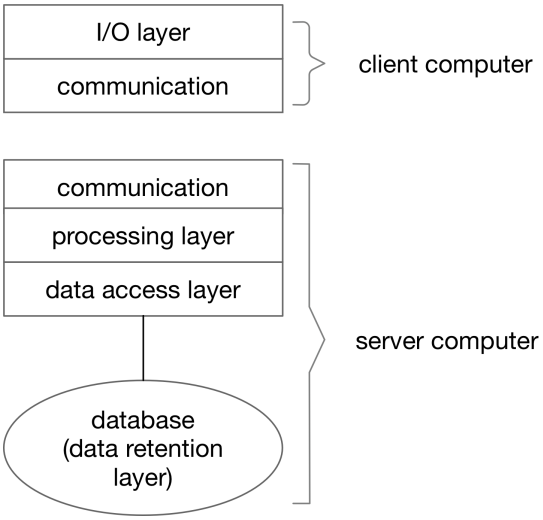

Figure 2.2: Three-Tier architecture [2]

The three tier architecture requires a wide spectrum of technological know-how: Knowledge about database management systems which is a discipline on its own, then deep knowledge of the Java or .NET ecosystem, and knowledge about HTML, cascaded style sheets, JavaScript and possibly JavaScript frameworks used for the design of the user interface.

To reduce the amount of programming languages to be used some people have advocated JavaScript also as a server side language. This has become popular with the Node.js framework which permits JavaScript developers to employ their programming language skills on the client side as well as on the server side.

On the account of type safety, this approach fosters rapid application development. Using a single language on both sides adds the chance of sharing modules, thus further reducing development time (see Figs. 2.3 and 2.4). This novel approach even affects the job market. Iin the past front-end and back-end developers had required different skills. This has changed now and people can do full-stack web development with just good JavaScript knowledge.

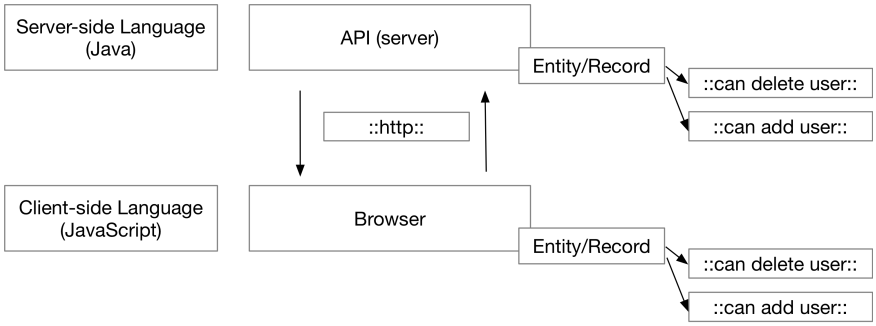

Figure 2.3: Duplication of functionality due to heterogeneous programming languages

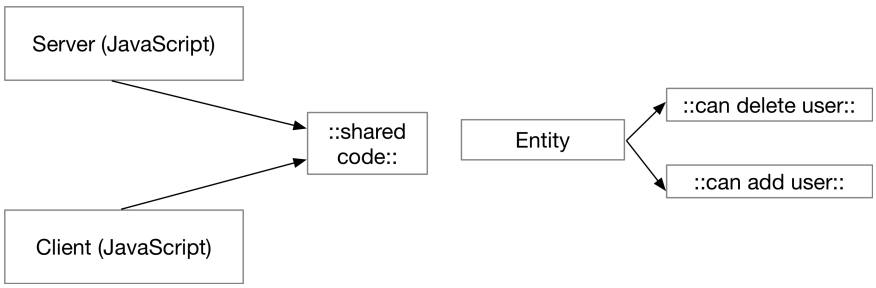

Figure 2.4: Reuse of shared modules due to a single language approach

2.2.2 Multi-Platform support

It is expensive to develop an application for a number of platforms like Android or iOS for tablets and Windows, Linux or MacOs for desktop computers. Creating applications for any of these platforms requires large amounts of platform specific code while little can be shared. Java has provided a way out of this dilemma with the exception of the popular iOS. Apple does not allow Java to run on their mobile devices.

There exist other cross-platform frameworks like QT. However, they require people to program in C++ which is not the most efficient way to create user interfaces anymore.

With JavaScript developers can take advantage of modern frameworks like React Na-

tive or Electron, which make it possible to almost completely code in JavaScript.

The JavaScript communicates with native components that might be written in Java on Android, in Objective C on iOS, in CSharp on Windows and so on. A so called "bridge" (see Fig. 2.5) forwards calls from JavaScript to native components and returns responses back to the JavaScript part. This facilitates user interfaces to have a native look and feel even though they are written in JavaScript [18].

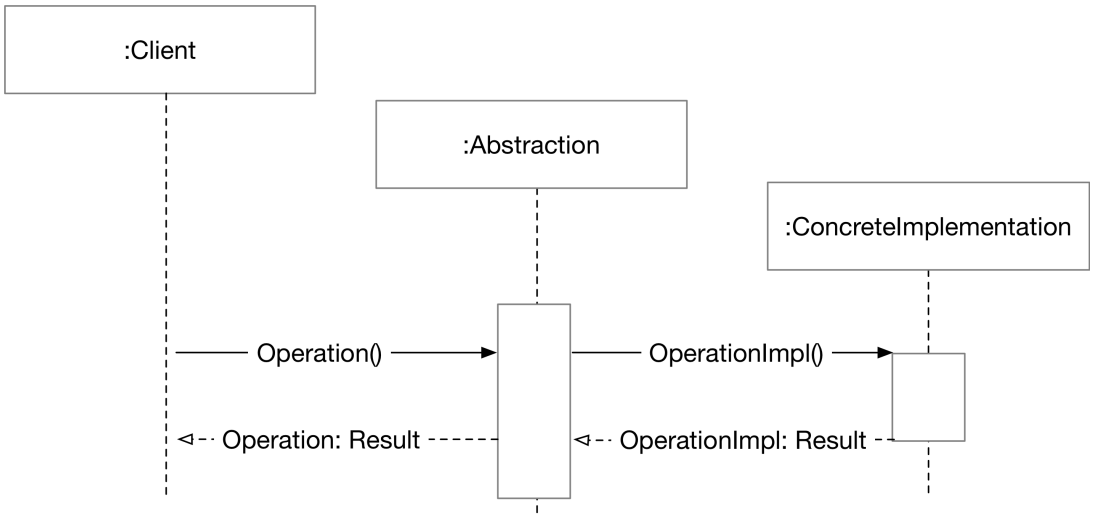

Figure 2.5: Bridge pattern [2]

2.2.3 Single threading and concurrency

JavaScript is a single threaded programming language with some extension for asynchronous processing. A JavaScript program may never have an infinite loop, regardless at which level. This is due to the fact how JavaScript handles concurrency. JavaScript was developed to manipulate the DOM. The DOM is a singular structure and it is easily conceivable that changing it from different threads would result in a big mess. Thus it made no sense to have JavaScript support concurrency.

On the other hand the browser had to interact with remote servers which could lead to large delays until a response would have been received. Blocking the single JavaScript thread with such calls would have lead to a rather poor user experience,

basically blocking any user interaction while the browser was waiting for the servers response.

The designers of JavaScript therefore provided a way to execute outside asynchronous calls with so called "callbacks". Every time a function is called in JavaScript, it is put on a call stack. Once the function returns the result is popped from the stack. Each function on the stack is executed purely synchronously one after the other (see Fig. 2.6).

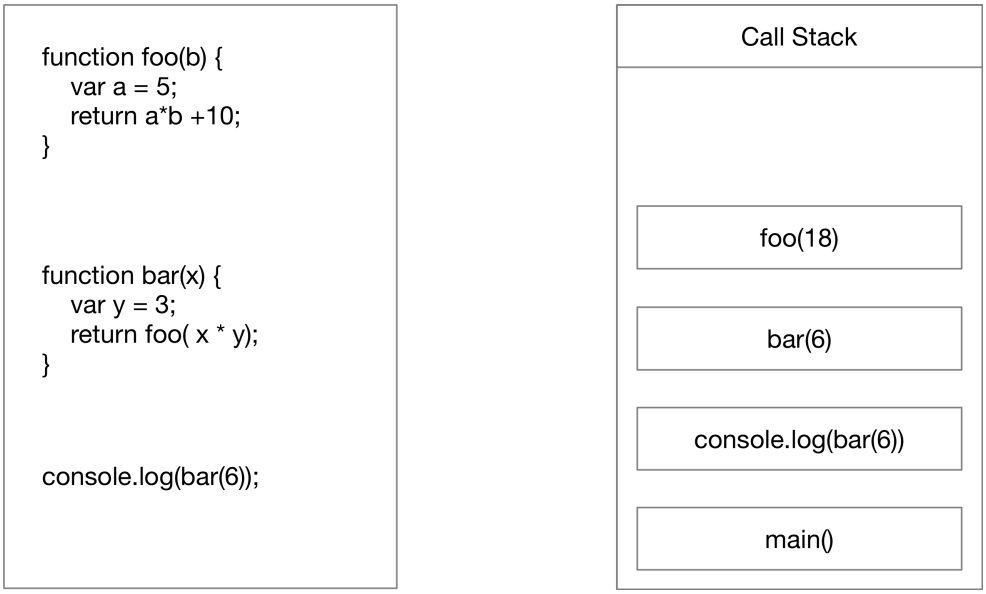

Figure 2.6: Call Stack

In order to execute code concurrently, JavaScript needs to call the Web API provided by the browser. The web browser runs on a multi threaded operating system so it fully supports concurrency.

The Web API is called with the function that needs to be executed, and a callback function. The callback function is called by the asynchronous function when it finishes execution. It is mandatory to provide a callback with each asynchronous function call.

Once the thread executing the asynchronous function is finished it passes the callback

to the so called "callback queue". The callback queue is a simple list containing all callbacks waiting for execution.

An event loop continuously checks if the call stack is empty. As soon as that is the case the first callback function from the callback queue is placed on the callstack and executed synchronously. This decoupling of the caller from the response allows JavaScript to do other things while waiting for asynchronous operations to complete and their callbacks to fire (see Fig. 2.7).

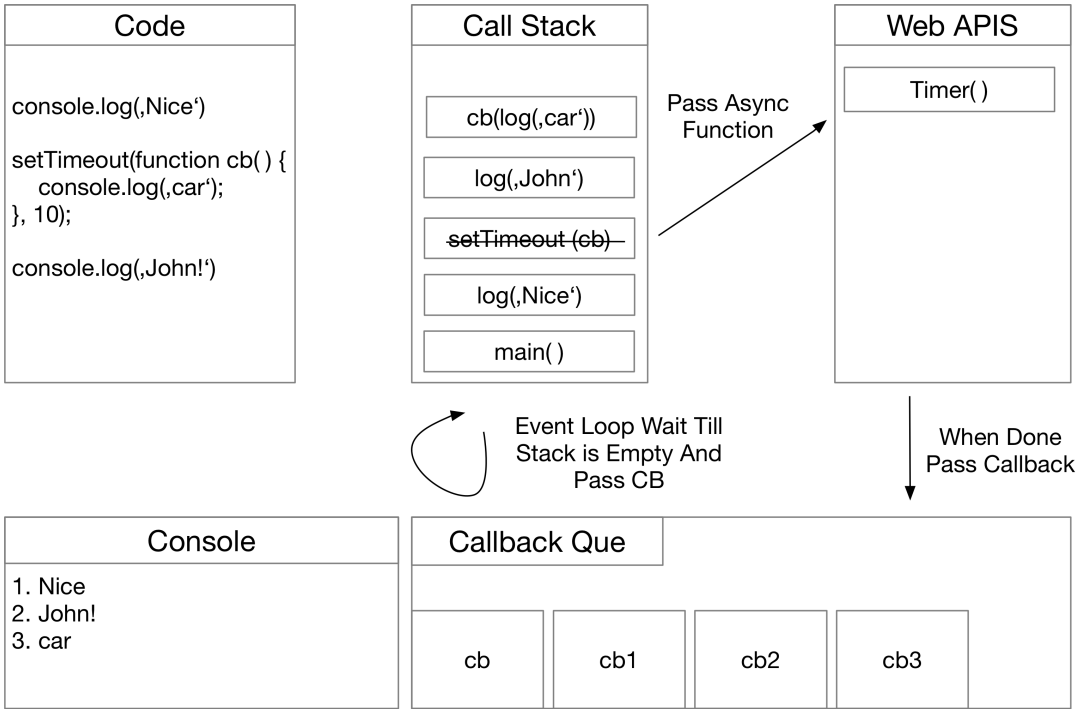

Figure 2.7: Concurrency in JavaScript

2.2.4 Functional programming paradigm

JavaScript is a functional programming language, that is it is possible to pass functions as parameters and to return functions as results. In the functional programming paradigm the return value of a function shall not depend on global variables or state variables, it shall purely depend on its input arguments (see Fig. 2.8)[19].

In the latest releases of the JavaScript standard provisions have been made to better support the object oriented paradigm.

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

In JavaScript it is possible to define functions as part of higher-order functions 2.9.

High-Order Functions

```
function makeAdjectifier(adjective) {

    return function (string) {

        return adjective + „ „ + string;
        };
}

var coolifier = makeAdjectifier(„cool“);

coolifier („Car“)
```

Figure 2.9: High-order functions

2.3 Software patterns for web-development

2.3.1 The classic model view controller model

For a long time it was best practice to use the model view controller (MVC) architectural pattern for implementing user interfaces. In this pattern, the model stores the

data presented in one or more views. In simple systems the model may contain some business logic (see Fig 2.10).

The controller controls the model and view state, based on user input. For example it activates or deactivates buttons. It also transforms events caused by user actions into method calls of the model [2].

The view serves to present model data to the user. There can be many views on the same model data. In case of a model data change all views are updated. Beside presenting model data the view also provides interactive elements.

The model shall be independent of views and controllers. In case the model changes, the controller may inform the views (passive model). With the alternative active model implementation, the model informs the views of any change.

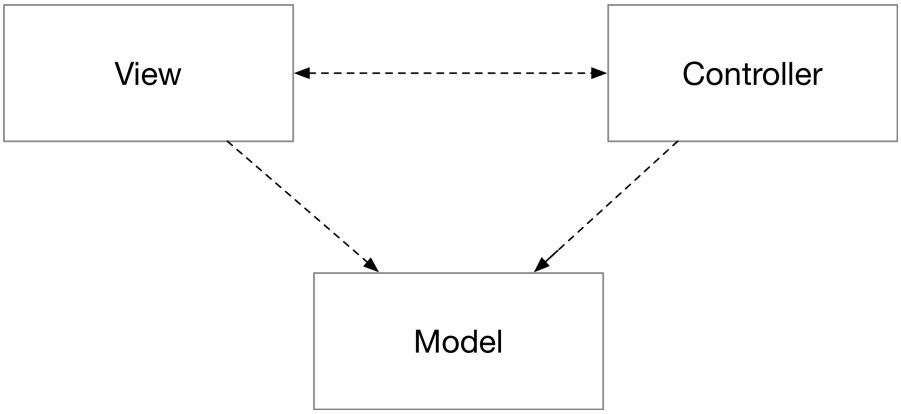

Figure 2.10: MVC software architecture [2]

The so called principle "separation of concerns" makes it easier to split work and thereby makes it easier to maintain code. A drawback is the increased number of classes and larger complexity. For example, changes 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 has a cascading effect across the codebase.

The React framework tries to keep the benefits of the MVC pattern while at the same time avoiding some of its disadvantages. To that end it employs a different architec-

ture called Flux which shall be explained below.

2.3.2 Flux

Structure and data flow

The Flux architecture consists of actions, dispatchers, stores and views. In Flux data flows in a single direction. The unidirectional data flow is central to the Flux concept (see Fig. 2.11). Dispatcher, stores and views are independent nodes with different inputs and outputs. The actions simply consist of objects with the new data. The following sections are inspired by the official Flux guide [3].

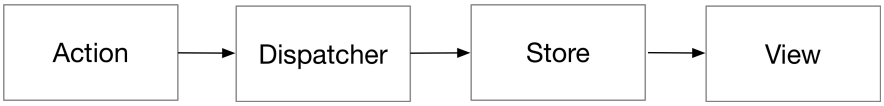

Figure 2.11: Flux software architecture [3]

The dispatcher takes care of handling all actions and forwards them to the proper 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 (see Fig. 2.12).

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

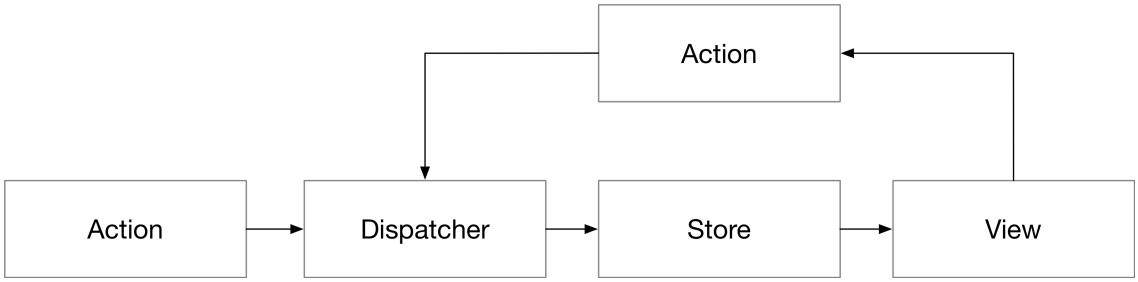

Figure 2.12: Flux with view action [3]

Stores

Stores contain the application state and the logic. Their role is similar to that of the model in the MVC, but they can contain the state of several objects.

Stores typically comprise the states of a domain within the application. Stores register with the dispatcher with a callback function. The callback has an action as a parameter. A switch case based on the action type determines which method within the store should be executed.

In this way a store is updated by an action. As soon as the store is updated, it reports that a state has changed so that all views can get the new state and update themselves.

Dispatchers

In the core of Flux lies the dispatcher. It is responsible for the control of the data flow within the application. In principle, it consists of a list of callbacks into the various stores. It serves only to distribute the actions to the various stores. Each store registers itself with a dispatcher providing a callback . The dispatcher can also consider dependencies between different stores and control the order of the callbacks.

Views and controller-views

In React all views are composable and freely re-renderable. At the top of the view hierarchy there is a hidden view that listens to events that are sent by the store. It is called a controller-view. The controller-view fetches the data from the store and passes it down to all descendants, causing a re-rendering. Often the entire state of a store is passed down a chain of views allowing each descendant to take what it needs.

Actions

The dispatcher provides a dispatch method that has an action as a parameter. Actions can either be sent directly to the dispatcher or created via a creator function and sent to the dispatcher. The creation of an action usually takes place in the event handler of a view, for example due to a user interaction or a browser event.

Conclusion

The Flux architecture improves data consistency. The unidirectional data flow makes applications based on Flux much easier to debug, since one can always follow easily the flow of actions. Furthermore, having the state and all logic updating the state in one place it is also possible to do more meaningful unit tests.

2.4 React and Flux

The key framework used in this project is React, a JavaScript library for developing user interfaces [5].

Back in 2011 Facebook noticed that it was getting hard to maintain their application and to run it flawlessly, due to the growing number of features. Many people were hired and the team size expanded significantly. 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, a first prototype of React. Jordan was allowed to keep on working and created React in 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 this conference said: "Our biggest mistake was betting too much on HTML5". He promised to provide better experiences with React.

Currently React is getting increasingly popular. A trend analysis by Google shows that React is the leading modern user interface JavaScript framework (see Fig. 2.13).

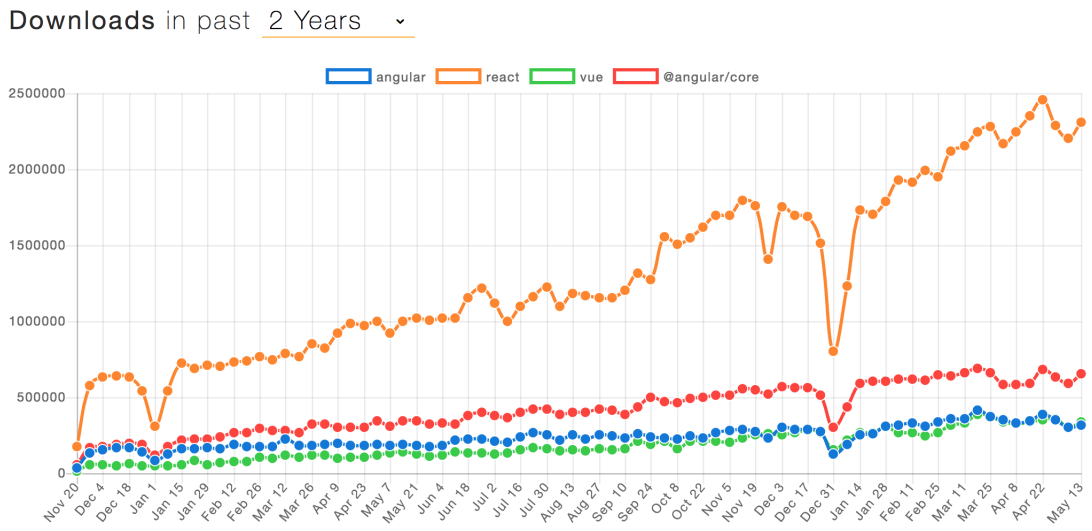

Figure 2.13: Downloaded npm packages [4]

This section is about analyzing how React is implementing the Flux software architecture. React is based on encapsulated components that manage their own state. An application consists of nested components, somehow similar to the tree structure of an HTML DOM.

Components are written in JavaScript, so data can easily be passed through the application. Any element that needs to be rendered is still written in HTML and parsed by React. Each component has its own controllers.

In React, there are no HTML files decorated with special tags but rather the HTML is generated from within JavaScript code. Every component is fully standalone and testable on 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, for example a board game that consists of squares (see Fig. 2.14).

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

Listing 2.1: Nested square component

Each square is a component and part of the whole board game application. There shall be a value assigned to the squares. Values are passed down to lower components via the "props" variable.

```
class Square extends React.Component {
  render() {
    return (
      <button className="square">
        {this.props.value}
      </button>
    );
  }
}
```

Listing 2.2: Passing down variables with props

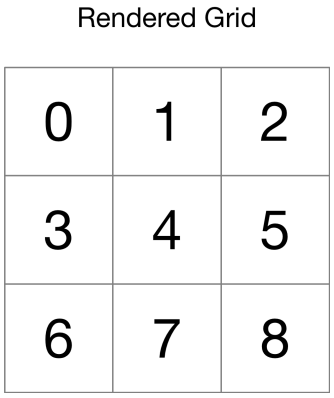

Figure 2.14: Grid with numbers

Squares that have been clicked on shall show an 'X'. This value can 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 }
```

Listing 2.3: Components with states

When `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 an 'X' in the grid.

2.4.1 Lifting state up

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

For example in a Tic Tac Toe game, to determine who has won the value of all square components would need to be checked. While that is technically feasible, a better approach is to save all states in the parent component. The parent then checks the 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. Such components are called "controlled components" (see Fig. 2.15).

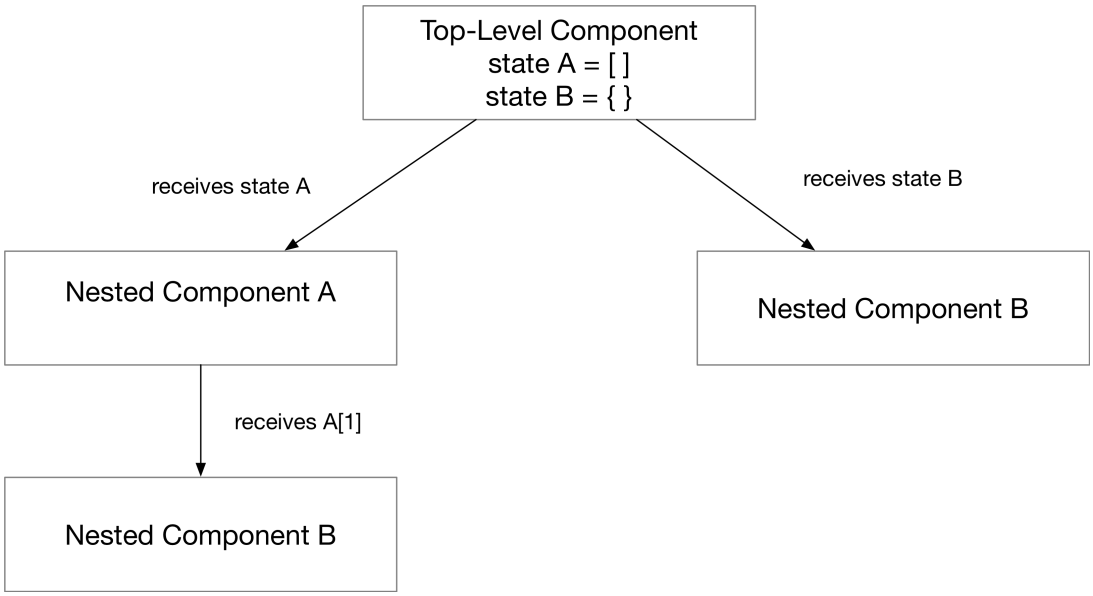

Figure 2.15: Top-level component

2.4.2 Action- and dataflow

Taking the last examples it is clear that data is only passed in one way, from the top-level component to a child component. Another key principle is that actions are only passed up.

For example if a nested component B is clicked or is triggered on a different event, some state B shall be changed in the top-level component. That is possible because all functions that change the data in the top-level component can be passed down to nested components via the `props` variable (see Fig. 2.16).

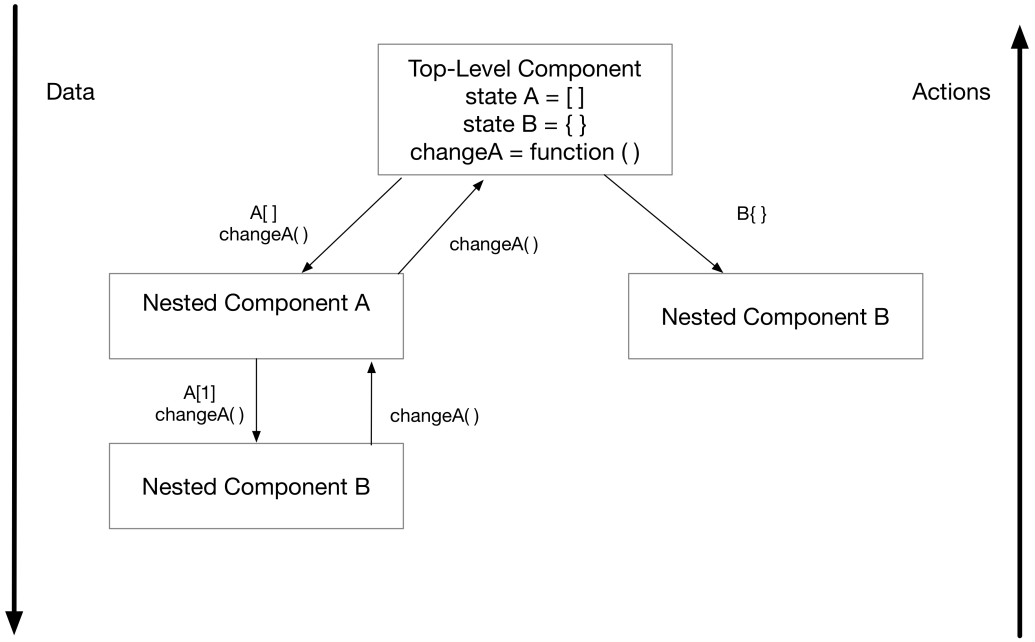

Figure 2.16: Action- and dataflow

2.4.3 High-order components

As already known from the JavaScript introduction chapter a high-level function can take another function as an argument. Similar to that a high-order component is simply a high-order function that takes a stateless component and returns the changed component. This makes it possible to add functionality to components on the fly.

```
1  import React from 'react';
2
3  const withSecretToLife = (WrappedComponent) => {
4    class HOC extends React.Component {
5      render() {
6        return (
7          <WrappedComponent
8            secretToLife={42}
9            {...this.props}
10         />
11       );
```

```
12      }
13    }
14
15    return HOC;
16  };
17
18  export default withSecretToLife;
```

Listing 2.4: High-order component

```
1  import React from 'react';
2  import withSecretToLife from 'components/withSecretToLife';
3
4  const DisplayTheSecret = props => (
5    <div>
6      The secret to life is {props.secretToLife}.
7    </div>
8  );
9
10 const WrappedComponent = withSecretToLife(DisplayTheSecret);
11
12 export default WrappedComponent;
```

Listing 2.5: Wrapped component

2.4.4 Virtual DOM

Interactive web applications mainly consist of code that manipulate the DOM. Manipulating the HTML DOM is an expensive operation since it often times results in unnecessary re-rendering of DOM elements.

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 a replication of the HTML DOM but within JavaScript. State changes lead to the creation of a new virtual DOM. React compares the new Virtual DOM with the previous one and just applies the differences between the two to the HTML DOM (see Fig. 2.17).

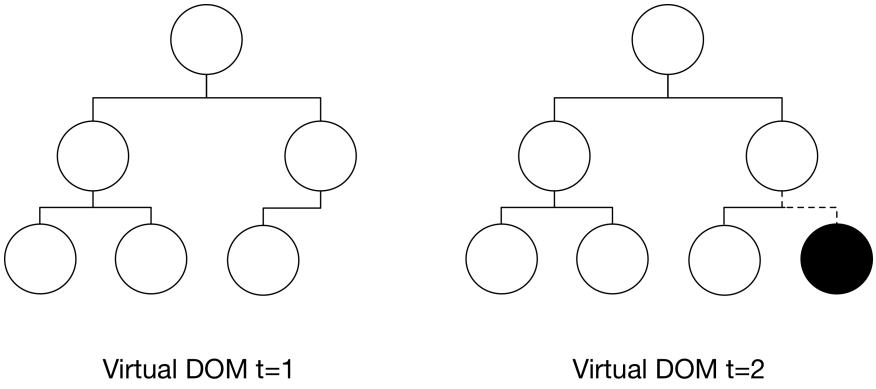

Figure 2.17: Virtual DOM comparison

React is a relatively new framework with a new underlaying software architecture for user interfaces. This architecture leads to less files and more encapsulation. Therefore it scales better than classic MVC when having more views on the same data. Due to the Virtual DOM React is really performant when updating the DOM.

2.4.5 Redux

Redux evolves the ideas of the Flux software architecture. This section is based on the official Redux guide [22].

The primary idea of Redux is to keep all states in a single store. The only way the state may be changed is by emitting actions. Actions are objects describing what happened, either at the user interface or at the client server interface. So called "pure reducer" functions define how actions change the state tree.

The major difference between Flux and Redux is that Redux does not have a dispatcher and does not support more than one store. With a single root reducer there is just a single store. In larger applications the root reducer can be split into several reducers, each operating independently on different parts of the state tree.

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

- A considerable amount of data is changing over time
- A single source of truth is needed (all states in one place)
- Keeping all of the states in a top-level component is no longer sufficient

It should further be mentioned that Redux destroys a key principle of React by providing access to the state from all components in an application directly. It is no longer obvious what data is used in which component (see Fig. 2.18).

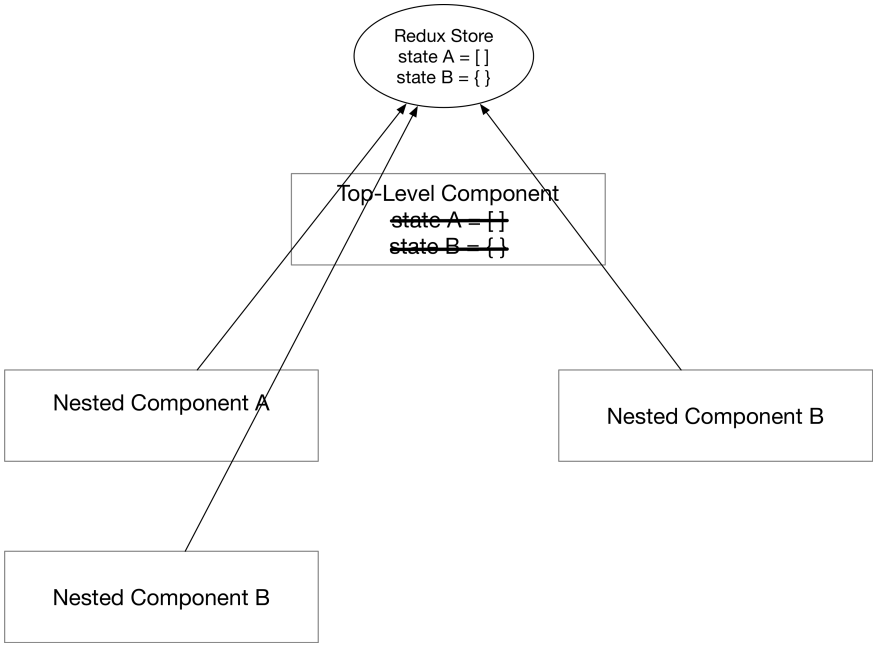

Figure 2.18: Redux store

2.5 TensorFlow.js

TensforFlow.js is an open source JavaScript library for training and deploying neural networks in a browser and on Node.js. First introduced in Spring 2018 it already provides intuitive APIs to build and train models from scratch in a browser. Existing models can be loaded and re-trained. Normal TensorFlow models can be imported

and used.

TensorFlow is computation intensive. It can be significantly accelerated by accessing graphic processors via a WebGl interface. This facilitates to train a network and execute predictions in a reasonable time. In the following we will discuss some core concepts of TensorFlow.js [21].

2.5.1 Core concepts

Tensors are sets of numbers with an additional shape attribute. The shape attribute defines the tensors dimensions, in other words how to interpret the set as a numerical array. Tensors are immutable, the values of a tensor can not be changed.

```
1  // 2x3 Tensor
2  const shape = [2, 3]; // 2 rows, 3 columns
3  const a = tf.tensor([1.0, 2.0, 3.0, 10.0, 20.0, 30.0], shape);
4  a.print(); // print Tensor values
5  // Output: [[1 , 2 , 3 ],
6  //          [10, 20, 30]]
7
8  // The shape can also be inferred:
9  const b = tf.tensor([[1.0, 2.0, 3.0], [10.0, 20.0, 30.0]]);
10 b.print();
11 // Output: [[1 , 2 , 3 ],
12 //          [10, 20, 30]]
```

Listing 2.6: Tensors

It is possible to create variables from tensors. Such variables have the same structure as the tensor but they are mutable.

```
1  const initialValues = tf.zeros([5]);
2  const biases = tf.variable(initialValues); // initialize biases
3  biases.print(); // output: [0, 0, 0, 0, 0]
4
5  const updatedValues = tf.tensor1d([0, 1, 0, 1, 0]);
6  biases.assign(updatedValues); // update values of biases
```

```
biases.print(); // output: [0, 1, 0, 1, 0]
```

Listing 2.7: Variables

Tensors provide operations that can be performed on them. In TensorFlow.js there is a wide variety of algorithms that can be applied to tensors. These operations do not change a tensor, instead they create new tensors.

```
const d = tf.tensor2d([[1.0, 2.0], [3.0, 4.0]]);
const d_squared = d.square();
d_squared.print();
// Output: [[1, 4 ],
//          [9, 16]]
```

Listing 2.8: Operations

TensorFlow.js models are comparable to functions as they take inputs and produce outputs. TensorFLow.js provides a high-level API called `f.model()` to construct a model with layers.

```
const model = tf.sequential();
model.add(
  tf.layers.simpleRNN({
    units: 20,
    recurrentInitializer: 'GlorotNormal',
    inputShape: [80, 4]
  })
);

const optimizer = tf.train.sgd(LEARNING_RATE);
model.compile({optimizer, loss: 'categoricalCrossentropy'});
model.fit({x: data, y: labels});
```

Listing 2.9: Model

2.5.2 Performance

Even though TensorFlow.js can access the local graphics processing unit (GPU), the Tensorflow.js development team reports that TensorFlow implementations written in

Pyhton or C++ perform 1.5 to 2 times faster. Small models seem to train faster in the browser, but larger models will perform 10-15 faster in Python than on JavaScript [21].

2.5.3 Summary

TensorFLow.js makes constructing neural networks, training and predicting quite easy. With GPU support such networks run relatively fast even in JavaScript, although they are not as performant as TensorFlow distributions written in C++ or Python.

TensorFlow.js distributes the computational load across many computers and thereby scales much better than if an algorithm would be running on a single server, even if that was very performant.

3 Deep Learning

Deep learning is a class of neural network optimization methods for networks that have a significant number of hidden layers between their input and output layer. Compared to learning algorithms of more simple network structures, the methods of deep learning offer a stable learning success even in the presence of numerous hidden layers.

Deep learning has been successfully applied to object detection in computer vision, in natural language processing, and speech recognition. In this chapter we will explain how such artificial neural networks function by looking at feed-forward network as an example [8].

3.1 Feed-forward neural networks

A feed-forward network comprises one input layer, H hidden layers and one output layer. Each layer consists of a number of elements called "neurons". Each neuron of any given layer is fully connected to each neuron of the previous layer. Mathematically, a neural network is represented by a combination of matrix multiplication and activation functions. An activation function determines when a neuron fires. It usually is nonlinear. The nonlinear characteristic of the activation function is what enables the network to learn. Each neuron in a hidden layer can be described by equations 3.1 and 3.2

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

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

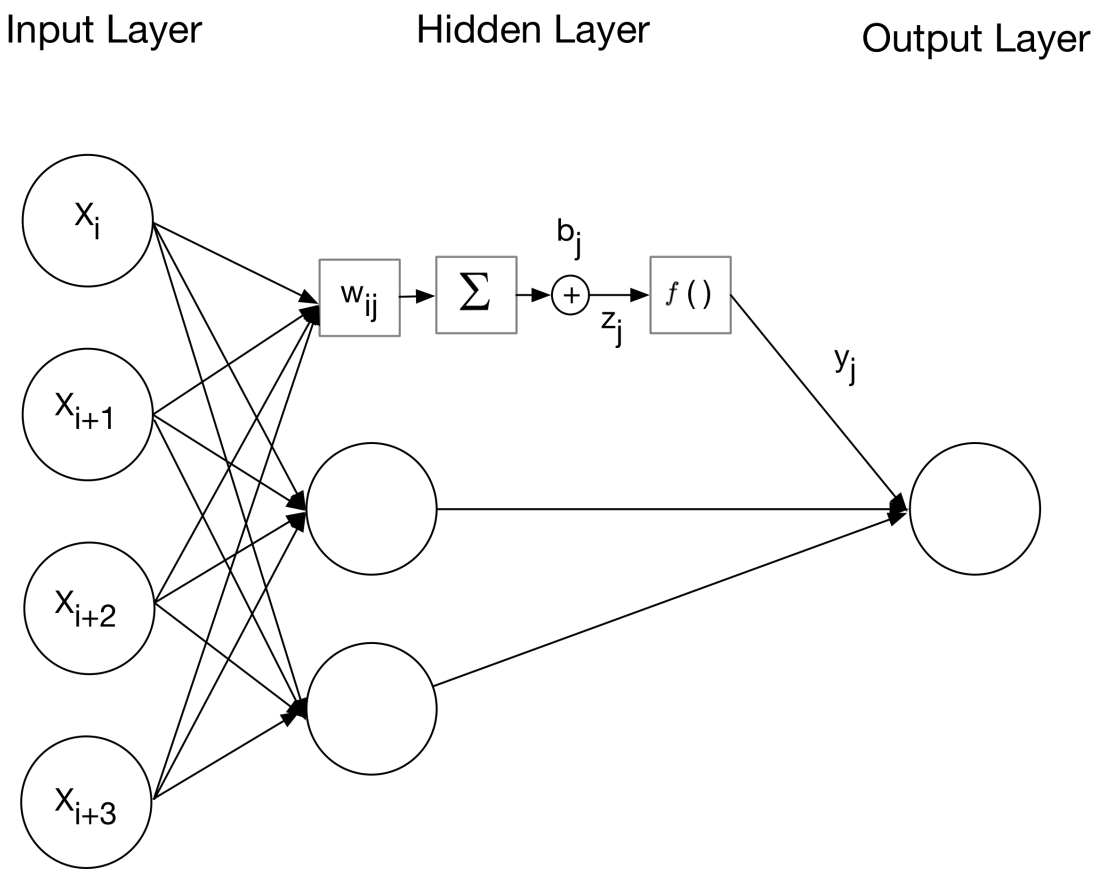

Figure 3.1: Schematic view of a simple two layer fully connected neural network (FCN)

The outputs of the preceding layer are each weighted with a weight w_{ij} before they are accumulated. The resulting sum is shifted by adding a bias b_j to generate the input z_j for the activation function $f(\cdot)$. The output of the activation function y_j is then taken as the input for the next hidden layer or output layer (see Fig. 3.1).

Any continuous functions can be used as activation functions of a neural network. Most common activation functions are sigmoid, $f(z) = \frac{1}{e^{-z}}$, 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.

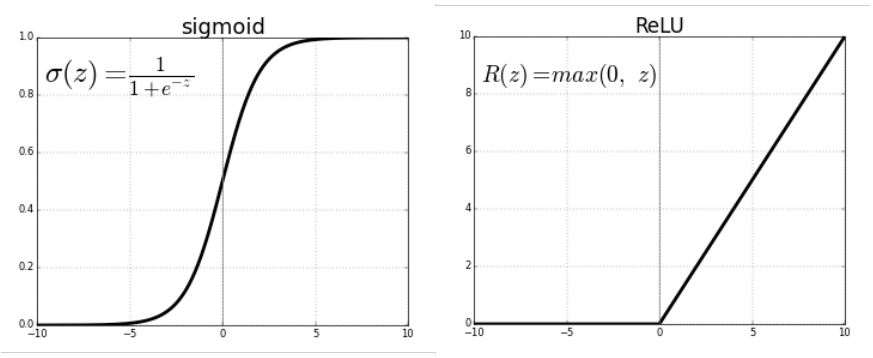

Figure 3.2: Sigmoid and ReLu activation function (from[6])

3.1.1 Backpropagation

Training a neural network means iteratively applying a forward pass followed by an error backpropagation. During the forward pass the outputs for each neuron are calculated based on its inputs. The resulting outputs y_{out} are then compared with the correct answer via a cost function E. A common cost function for example is squared error loss (see equation 3.3):

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

Based on the output of E it is now possible to check how weights need to be adjusted in order to minimize the error. Applying the chain rule we get equation 3.4.

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

Considering the error E we get equation 3.5

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

Using the calculated gradient $\Delta_W E(y_{out})$ it is now possible to determine the next weight matrix update using a gradient descent (equation **??**):

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

where W^t, W^{t+1} are the current and the updated weight matrices, respectively, and η is the learning rate.

3.1.2 Weight update

Calculating the gradient for a complete dataset can take a lot of computation time. To reduce the amount of computation required 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 perform this update, the most common one being the stochastic gradient descent (SGD). In this algorithm, parameters are simply changed along the negative gradient direction in order to minimize the error.

3.1.3 Initialization

A neural network needs its weights initialized before it can be trained. Random initialization is the most common way to create the initial weights.

3.1.4 Batch normalization

To avoid numerical problems due to extreme values the input data is normalized (see equation 3.7).

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

μ and σ are the mean and the standard deviation of the dataset, respectively. γ 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 common task for neural networks is the classification of images, or more general the classification of data into a specific category. The best performing neural network architectures for classifying images are convolutional neural networks (CNN). All convolutional neural networks consist of convolutional layers followed by pooling layers and some fully connected layers.

3.2.1 Convolutional layer

In a 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.

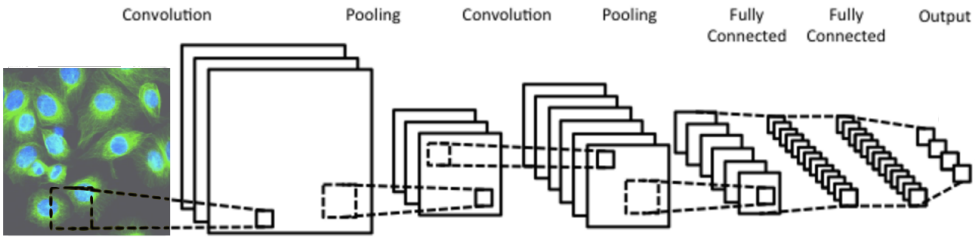

Figure 3.3: Schematic view of a Convolutional Neural Network (CNN) (from [8])

3.2.2 Pooling layer

After features have been extracted into feature maps, the pooling layer reduces the size of the feature maps and makes them computationally tractable. A simple implementation of a pooling layer is the max-pooling layer, which uses a sliding windows over the input and selects the maximum of each window.

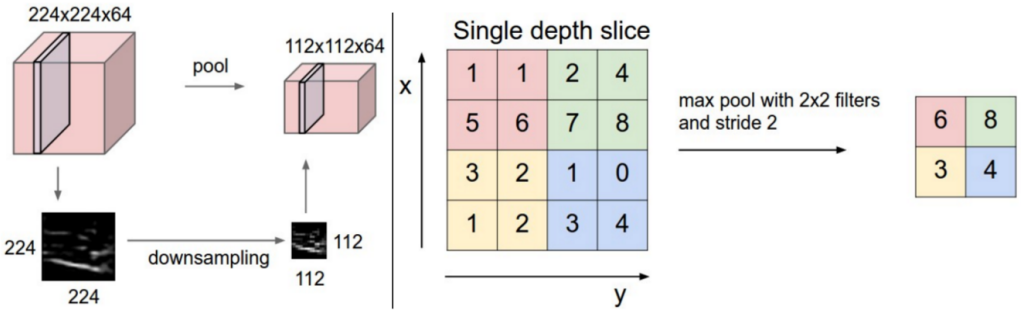

Figure 3.4: Pooling layer (from[8])

3.2.3 Data augmentation

Convolutional neural networks need a lot of data to generate satisfactory output. If not enough data can be provided the data can be artificially augmented by using different approaches. images 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.

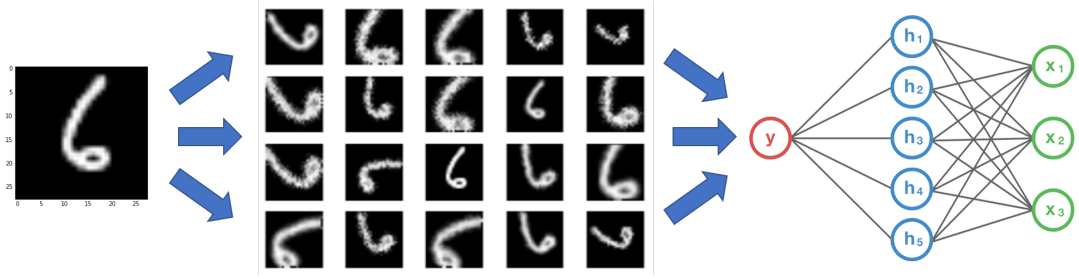

Figure 3.5: Data augmentation(from [7])

4 Image Based Profiling

Modern fluorescence microscopy combined with high throughput biotechnology, automatically quantifying biological properties in images, is now widespread. They provides new insights into human cells and are powerful technologies for studying cell biology. Using this technology more than more than one hundred thousand images can be produced per day. This makes automated image analysis a necessity. Image-based profiling aims at getting as much data as possible from a biological sample and to encoding it in a proper way. Image-based profiling experiments capture a wide range of data from biological samples without prior knowledge of the existence of any markers. Data mining and machine learning techniques can then be applied to identify patterns [12] [9].

4.0.1 Drug discovery

One important application of image-based profiling is identifying biological mechanisms of actions (MOA), for example checking damage of DNA replications due to chemical perturbations. When developing new drugs and making predictions about unknown chemical compounds it is important to know what chemical compounds cause which biological mechanisms of action. Morphological profiles can predict these MOAs for chemical compounds.

4.0.2 Typical workflow

There are two approaches in image-based phenotyping of perturbations. It should be considered that they are different. The first approach is called phenotypic screen-

ing. Phenotypic screening uses pre-defined, specific phenotypes which are compared with the specimen in order to identify drugs or drug targets that might have affected them.

The second method is called profiling of perturbations. Here a computer is trained with a large set of samples, both affected by perturbations and unaffected ones. Once trained the computer can analyze new specimen and for perturbed ones identify the drugs or drug targets that have affected them. This approach doesn't require the specification of any features such as cell size, intensity, shape, or texture. It furthermore permits how any specimen would look if it was affected by a specific perturbation.

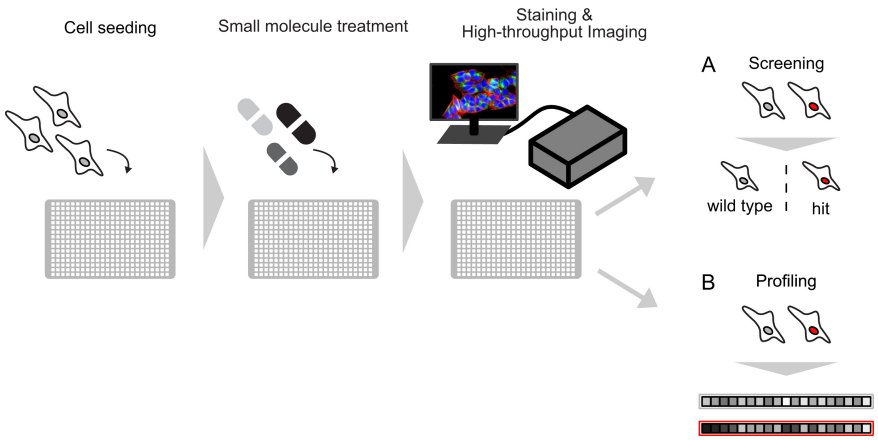

Figure 4.1: Typical workflow of image-based small molecule experiments (from[9])

Fig. 6.2 illustrates a typical workflow of image-based small molecule experiments. First cells need to be attached to plates. In a second step cells are perturbed. Then after staining the cells, images are taken using automated microscopes. At the end either screening approaches (A) or (B) profiling approaches can be used to classify perturbed cells.

Segmentation

Small molecule profiling is based on staining subcellular structures (Fig. **??**). An accurate segmentation of cells can be achieved by in intensity-based thresholding

and other approaches. A number of computational applications for segmentation-free analysis in image-based profiling have been developed, the most famous one being the Cell Profiler [13].

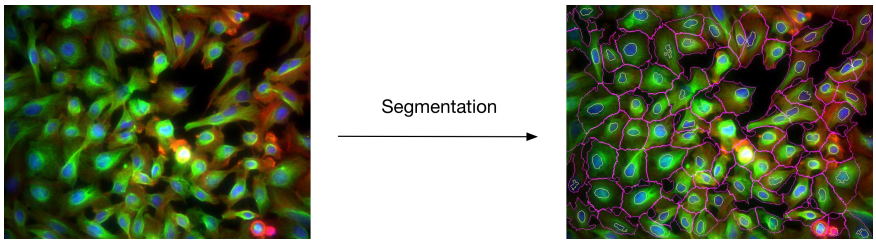

Figure 4.2: Segmentation (from [10])

Classification

A typical objective for profiling experiments is the classification of compounds that cells have been perturbed with. Common classifiers are random forests and deep neural networks that are employed to achieve higher accuracy in predicting biological MOAs.

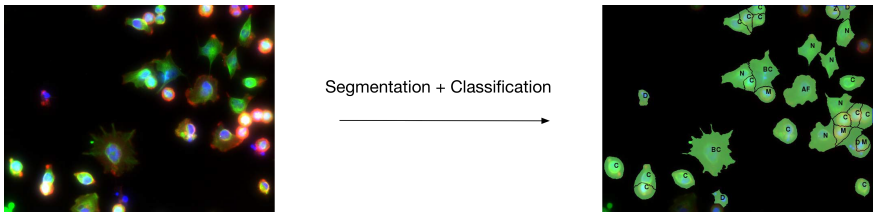

Figure 4.3: Classification (from [10])

5 Related work

This chapter is about presenting CellProfiler Analyst (CPA), a first generation tool for interactive data exploration and classification of large biological image sets. It was first introduced in 2016 at the Broad Institute of MIT and Harvard. The following section gives an overview of CPA's main functionality and an exploration of its key features, the classifier and image gallery.

This project is highly influenced by CPA and basically aims to improve it and make it accessible to anybody without having to install any software. CPA is free and open source, available at http://www.cellprofiler.org and from GitHub under a BSD-3 license. It is available as a packaged application for Mac OS X and Microsoft Windows and can be compiled for Linux.

5.0.1 CellProfiler Analyst

CPA is a GUI based tool. It provides features for data exploration and classification via its user interface. There are only a few comparable tools for this purpose, the most prominent being Ilastik, CellCognition and WND-CHARM.

All these tools lack a suitable companion visualization tool capable of handling large datasets. Furthermore they do not provide selectable classifier algorithms. Another big benefit CPA has compared to all other existing tools that it incorporates an interactive object classification and image viewing feature.

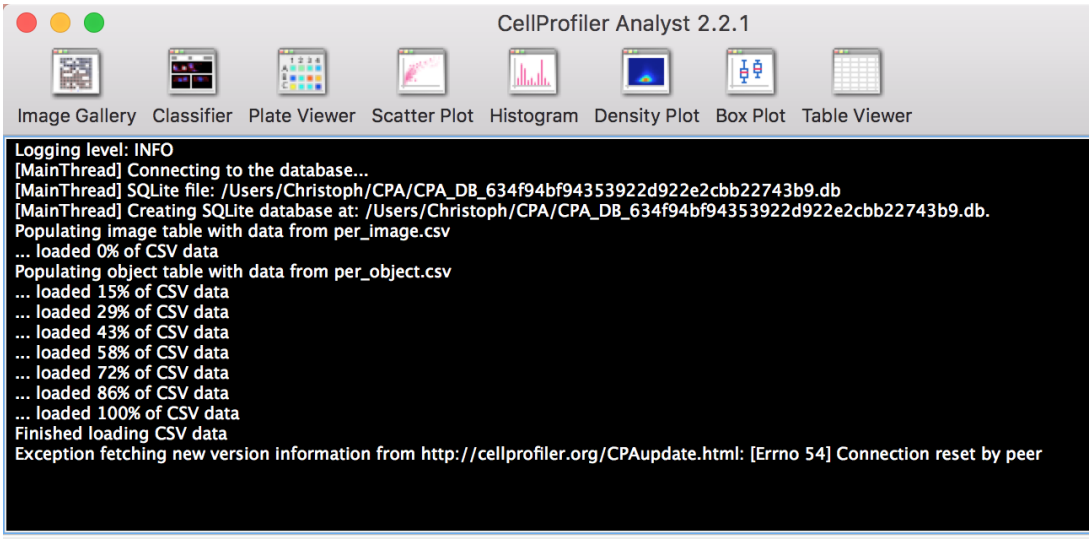

Figure 5.1: CPA main view, different views are selectable (from [11])

5.0.2 Classifier

The 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 them on a category. Multi-select features are also implemented 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 be fetched and then be automatically annotated by clicking the "evaluate" button.

If results do not suffice further images can be annotated to train the algorithm. Different algorithms can be used to classify the uncategorized images, such as RandomForest Classifier and AdaBoost Classifier.

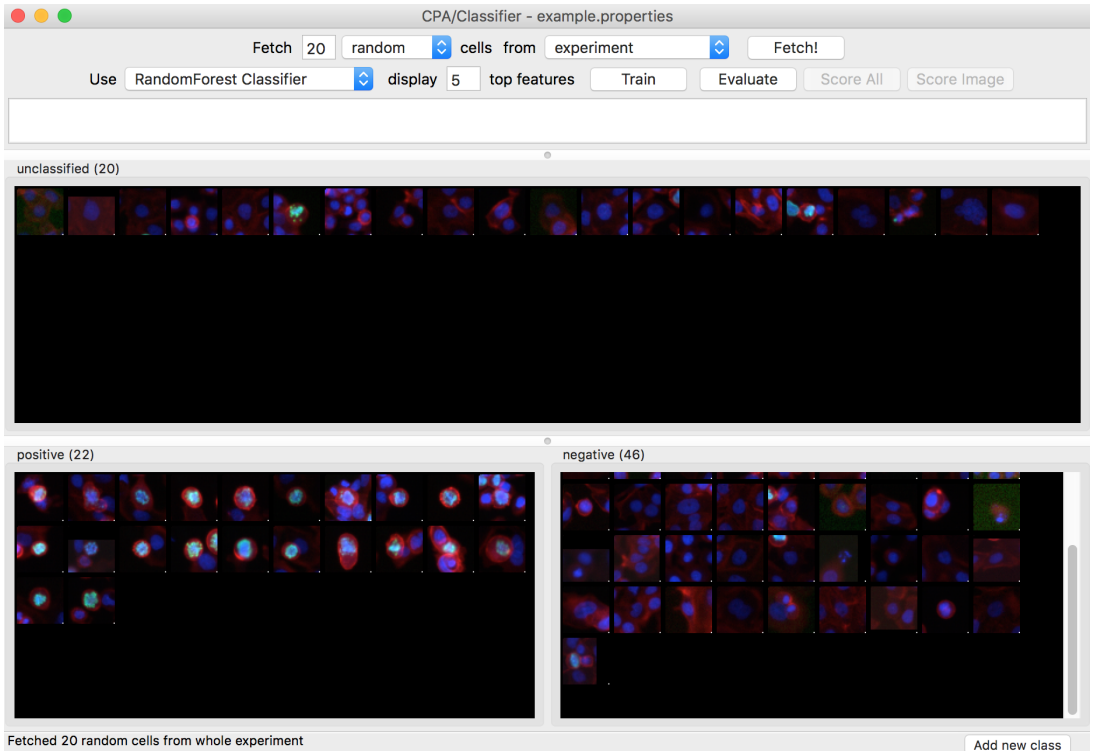

Figure 5.2: CPA classifier view (from [11])

5.0.3 Visualization

In order to explore a dataset and to predict and validate a result, a good visualization is needed. CPA offers different ways of displaying data and results.

One simple visualization possibility is the Image Gallery where images from the dataset can be selected and displayed in their original size. Filters can be applied to only select images with experiment specific meta data. There are many more ways to display and explore data in CPA.

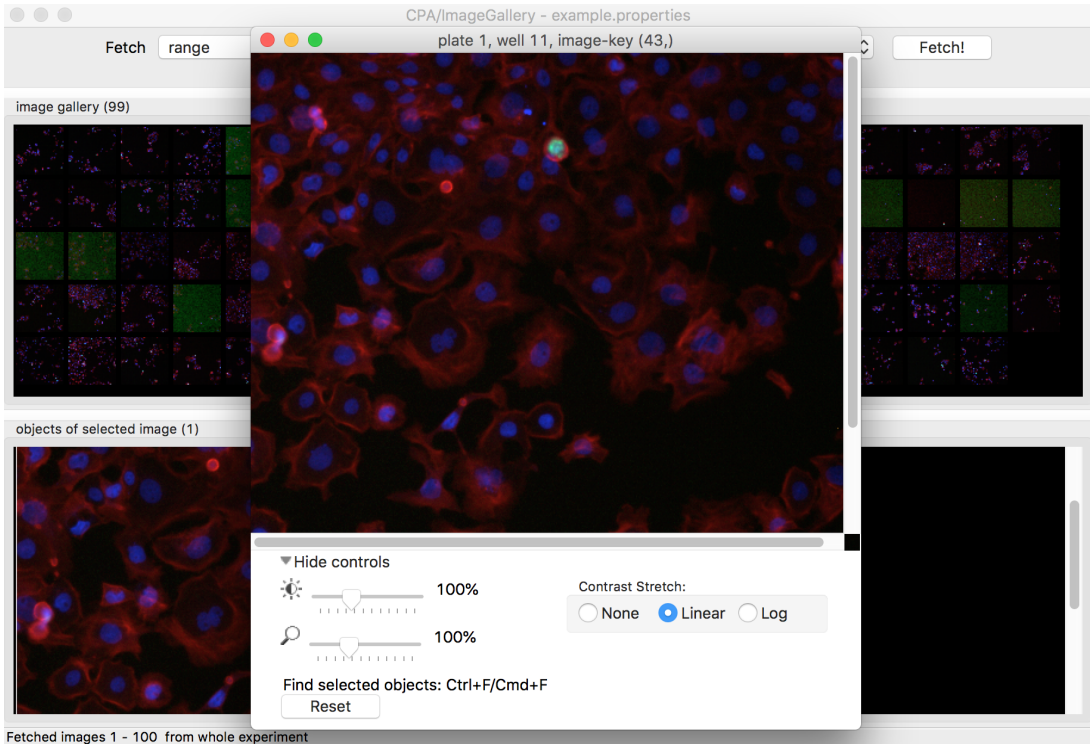

Figure 5.3: CPA full image view (from [11])

5.0.4 Downsides of CPA

Getting CPA to run on a computer requires installation first. This installation is not trivial as the system requires a MySQL database system to work.

Furthermore, the powerful user interface makes the system complex and difficult to use. This can scare off potential users.

6 CYTO AI

CYTO AI is the first fully web-based image labeling and classifying tool publicly available. Based on the JavaScript frameworks React and TensorFlow.js, all advantages of modern machine learning and labeling technology could be employed in a zero installation effort, completely platform independent, and fully browser based solution.

This rich client application does not require the upload of any data to a server. This helps both, privacy concerns as well as performance. The following sections shall explain how CYTO AI works and which techniques and architectures were used to build it.

Figure 6.1: CYTO AI

6.0.1 Workflow overview

The workflow starts with the creation of categories. Each image to be classified will have to be assigned to a category.

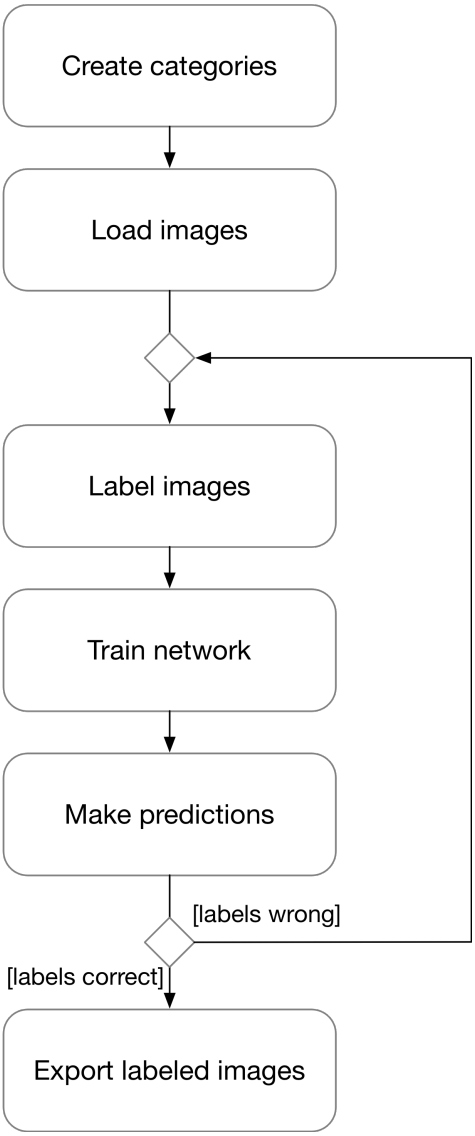

Figure 6.2: Workflow

Now the images need to be made available to the application. They are being up-

loaded to the local browser cache. It is possible to select entire folders which will result in being all images in that folder and their sub-folders being uploaded.

Once all images have been uploaded a knowledgeable user has to classify them by dragging them to the respective category field.

After sufficient samples haves been categorized the network can be trained. Depending the amount of images this can take more or less time. The training results in a trained network. Which now can be used to classify the previously unclassified images. This machine labeling needs to be controlled by a knowledgeable user and erroneous annotations have to be manually corrected.

Thereafter the network is trained again. This procedure is repeated until the labeling error rate by the trained network is satisfactory.

Now the labeled images can be exported and used by others to train their networks. In future it will be possible to export the network parameters so that a complete trained can be shared with others.

6.1 Working with CYIO AI

6.1.1 Creating categories

To add a category click on button A in figure 6.3. To delete a category click on the corresponding trash can symbol. It is not possible right now to directly change a category. For changing a category you need to delete the old one and create a new one.

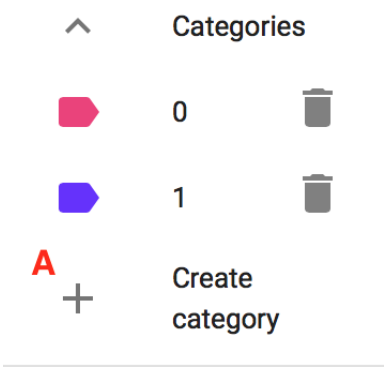

Figure 6.3: Create a ctagory

6.1.2 Uploading images

In order to upload images click on button A in figure 6.4. In Chrome and Firefox it is possible to select entire folders. All images have to be in the portable network graphics format (PNG).

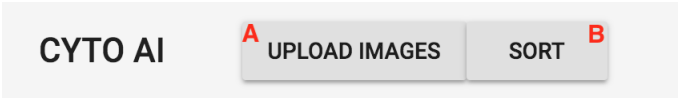

Figure 6.4: Image upload

6.1.3 Labeling images

There are two ways to annotate an image.

- Drag an image and drop it on the desired category.
- Select an image and use the keyboard for annotation

The following keys are supported for image annotation:

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

To navigate through all images you can use 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.

6.1.4 Automatically classifying images

Once sufficient images have been labeled manually the network can be trained and you can use it to automatically classify the remaining images. To start the training and classification process click the click button A in figure 6.5. The neural network will be trained and all unlabeled images will be categorized.

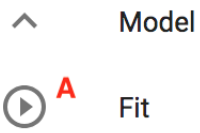

Figure 6.5: Classify images

A number will appear under each image, indicating the probability of the image belonging to the given category.

6.1.5 Exporting

To save all labels, categories, and settings click on the "save" button A in figure 6.6. To import click on "open" button B in figure 6.6.

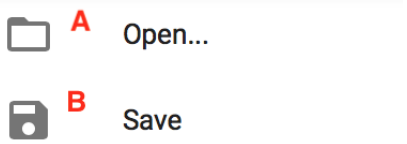

Figure 6.6: Export and import labels

6.1.6 Filtering and sorting

You can blend out certain categories by clicking on the category. Clicking a second time will again blend in the category. This enables you to show just specific categories or all unlabeled images. If all categories are blended out you can still annotate. Newly labeled images will stay visible.

Figure 6.7: Blending out images with category

You can sort labeled images by clicking on the "sort" button B in figure 6.4. This improves the overview and makes reviewing labels easier. Uncategorized images will always appear at the top.

Using the slider in figure 6.8 you can adjust the number of displayed images per row.

Figure 6.8: Slider adjusts number of pictures displayed per row

6.2 IDE setup

In order to develop CYTO AI a suitable development environment has to be installed. It needs to provide integrated Git support as well as easy debugging, syntax highlighting, and an intelligent code completion.

We use the integrated terminal and install a JavaScript syntax checker called ESLint. Further we use a code formatter called "Prettier".

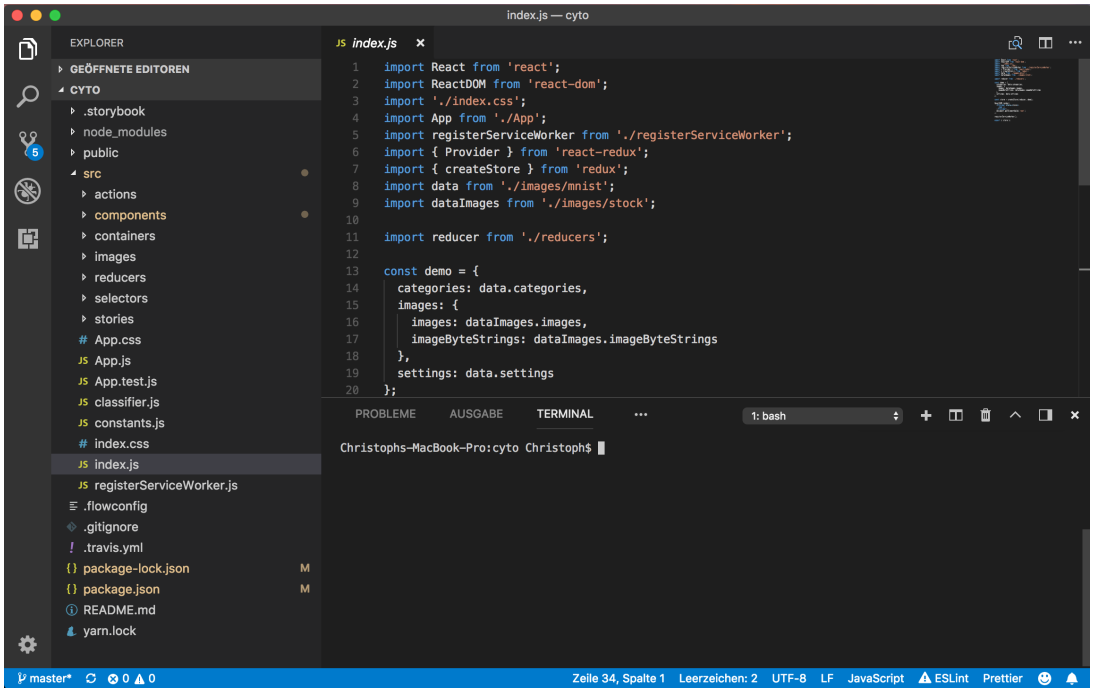

Figure 6.9: IDE setup

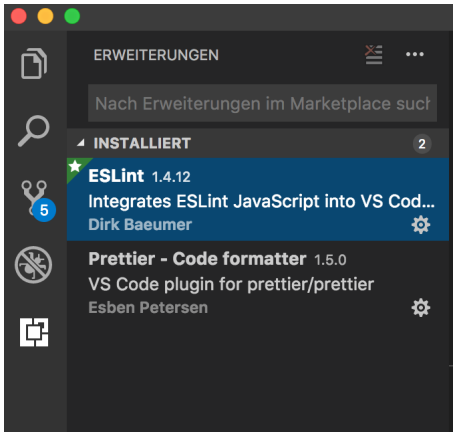

Figure 6.10: Used extensions

6.3 Package management

For managing dependencies and necessary frameworks the Node package manager `npm` is used.

Packages can be installed via:

```
npm i --save string-hash
```

and be used by importing them:

```
import hash from 'string-hash'
let identifier = hash(e.target.result)
```

All packages used are being saved in the `node_modules` folder. Further all used packages will be recorded in the package.json file.

6.4 The edit-debug cycle

Make sure to download and install Node.Js. It can be found under

```
https://nodejs.org/en/download/
```

The application needs to be downloaded:

```
git clone https://github.com/cytoai/cyto.git
```

Within the application folder the following commands are used to run the app.

After downloading all dependencies need to be installed:

```
npm install
```

Then the local Node.js server can be created by:

```
npm start
```

To debug within in the browser a plugin "React Developer Tools" for Google Chrome needs to be installed. With React Developer Tools it is possible to inspect rendered components and highlight DOM updates.

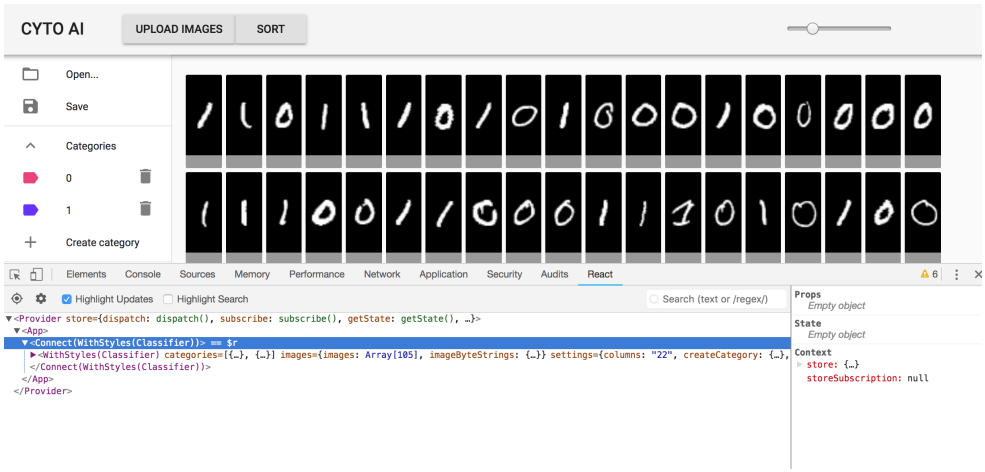

Figure 6.11: React developer tools

The project was set up to reload the browser every time the code changes. There is no need to clear the cache or reload the browser manually.

6.5 System architecture overview

To explain the system structure we will walk through the process of adding a new component to the existing application. This component shall display a list of categories which are stored an a Redux store.

6.5.1 Adding a component

To add a new component "Categories" we create a file named `src/components/categories.js`. It should look like listing 6.1.

```
1 const Categories = ({
2   categories
3 }) => {
4   return (
5       {categories.map(category => (
```

```
6             <ConnectedCategory
7               key={category.identifier}
8               identifier={category.identifier}
9               categories={categories}
10            />
11          ))}
12    );
13  };
14
15  export Categories
```

Listing 6.1: Categories component

The data shall be fetched from a variable `categories` that is placed in the Redux store. In order to fetch this data the component shall be connected to the Redux store. To that end a "High-Order Component" `ConnectedCategories` is created. The connector provides access to the `categories` array in the Redux store.

```
1  const mapStateToProps = (state, props) => {
2    return state;
3  };
4
5  const mapDispatchToProps = (dispatch, props) => {
6    return {
7      createCategory: (action, element) => {
8        const categoryName = element.value;
9        const category = {
10         color: randomcolor(),
11         description: categoryName,
12         identifier: uuidv4(),
13         index: index++,
14         visible: true
15       };
16       dispatch(createCategoryAction(category));
17     };
18   };
19 };
20
21 const ConnectedCategories = connect(mapStateToProps,
     mapDispatchToProps)(
22   Categories
```

```
23 );
24
25 export default ConnectedCategories;
```

Listing 6.2: Connector component

Further it shall be possible to add a new category to the category list. Therefore an action is needed that takes the new category name and the category color as a payload.

```
1  const Categories = ({
2    categories
3  }) => {
4    return (
5        {categories.map(category => (
6            <ConnectedCategory
7              key={category.identifier}
8              identifier={category.identifier}
9              categories={categories}
10           />
11         ))}
12   );
13 };
14
15 export Categories
```

Listing 6.3: Actions for changing categories

A reducer will now be called and the action will be passed to it as a parameter.

```
1  const categories = (state = [], action) => {
2    switch (action.type) {
3
4    case ADD_CATEGORIES:
5        return action.categories;
6
7      case CREATE_CATEGORY:
8        return [...state, action.category];
9
10     default:
11       return state;
12   }
```

```
13 };
14
15 export default categories;
```

Listing 6.4: Reducer for changing categories

In short, for each added component that needs access to the states a connector needs to be created. Actions and reducers also need to be created to change the state, if not already existing.

The directory structure would now look like this:

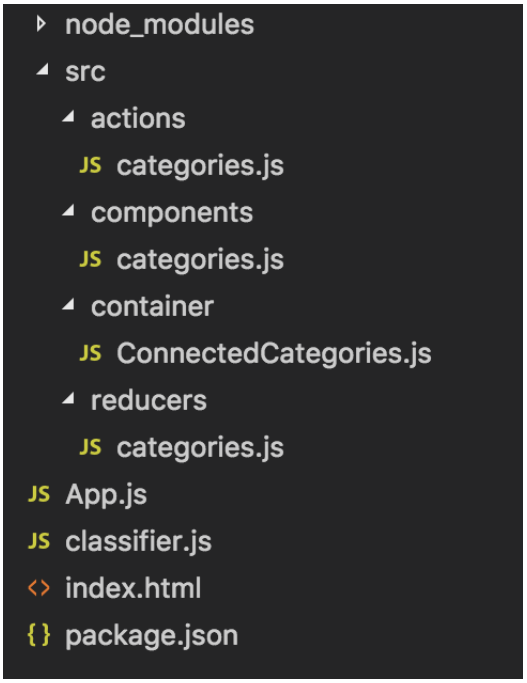

Figure 6.12: System architecture overview

6.5.2 The machine learning API

The machine learning API was built with TensorFlow.js. Every time the "fit" button is clicked an asynchronous API call is made. The calls look like this:

```
1 async function trainOnRun(imageTags) {
```

```
2    const dataset = new Dataset();
3    dataset.loadFromArray(imageTags);
4    await run(dataset);
5    return null;
6  }
```

Listing 6.5: Machine learning API

where `dataset` is an instance for handling the data, and `imageTags` are the HTML image tags. The API is using a CNN provided over a Content Delivery Network (CDN).

7 Summary and Outlook

As Samuel Buttler used to say: "Brevity is very good, where we are and are not understood." So this summary shall be rather brief.

At the start of the project we had a very powerful but somewhat difficult to install and use classification application written in Python with more than 27.000 lines of code. It looked like an almost impossible task to port such significant piece of work to an entire new architecture and completely new technologies in just a few months.

Now, at the end of this work and in cooperation with some highly talented people, it looks like the decision to move to the new web centric development paradigm has paid for itself.

All major technological issues could be resolved and there seems to be good reason to believe that continuing this way would result in less development effort and better user experience compared to the traditional way of implementation.

What is still to be done? Thus far, the tool can classify entire images. A big step forward would be the capability to segment images, that is to locate image areas with objects of interest prior to the classification process.

However, this would place considerable demands on the tool and would require another substantial development effort. The user interface would have to be redesigned so that images could be annotated.

Furthermore, it will soon be possible to export models (trained nets) with Tensorflow.js. The declared objective of Goggle is to make this possible in the upcoming versions (current version 0.10). It then is conceivable to make the exported models ac-

cessible to other users via a web platform, sharing trained classification networks.

Bibliography

[1] "GitHut - Programming Languages and GitHub," last accesed on 2018-05-15. [Online]. Available: http://githut.info/

[2] Goll, Joachim, *Architektur- und Entwurfsmuster der Softwaretechnik*. Springer Vieweg, 2013.

[3] F. Inc., "Flux Software Architecture," 2013, last accesed on 2018-05-15. [Online]. Available: https://facebook.github.io/flux/docs/overview.html

[4] "Downloaded NPM Packages," last accesed on 2018-05-15. [Online]. Available: http://www.npmtrends.com/angular-vs-react-vs-vue-vs-@angular/core

[5] "Official React Tutorial," last accesed on 2018-05-15. [Online]. Available: https://reactjs.org/tutorial/tutorial.html

[6] "Activation functions," last accesed on 2018-05-15. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[7] A. J. Ratner, H. R. Ehrenberg, Z. Hussain, J. Dunnmon, and C. Ré, "Learning to Compose Domain-Specific Transformations for Data Augmentation," sep 2017. [Online]. Available: http://arxiv.org/abs/1709.01643

[8] D. Dao, "Image-based chemical-genetic profiling using Deep Neural Networks," Master's thesis, TECHNISCHE Universität München, 2016.

[9] C. Scheeder, F. Heigwer, and M. Boutros, "Machine learning and image-based profiling in drug discovery," *Current Opinion in Systems Biology*,

may 2018. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S2452310018300027

[10] G. Pau, "High-throughput image analysis with EBImage EBImage," *Image Processing*, 2018.

[11] T. R. Jones, I. H. Kang, D. B. Wheeler, R. A. Lindquist, A. Papallo, D. M. Sabatini, P. Golland, and A. E. Carpenter, "Cellprofiler analyst: data exploration and analysis software for complex image-based screens," *BMC Bioinformatics*, vol. 9, no. 1, p. 482, Nov 2008. [Online]. Available: https://doi.org/10.1186/1471-2105-9-482

[12] T. R. Jones, A. E. Carpenter, M. R. Lamprecht, J. Moffat, S. J. Silver, J. K. Grenier, A. B. Castoreno, U. S. Eggert, D. E. Root, P. Golland, D. M. Sabatini, and E. M. Scolnick, "Scoring diverse cellular morphologies in image-based screens with iterative feedback and machine learning." [Online]. Available: http://www.pnas.org/content/pnas/106/6/1826.full.pdf

[13] T. R. Jones, I. H. Kang, D. B. Wheeler, R. A. Lindquist, A. Papallo, D. M. Sabatini, P. Golland, and A. E. Carpenter, "Cellprofiler analyst: data exploration and analysis software for complex image-based screens," *BMC Bioinformatics*, vol. 9, no. 1, p. 482, Nov 2008. [Online]. Available: https://doi.org/10.1186/1471-2105-9-482

[14] R. C. Martin, *Agile software development: principles, patterns, and practices*, internat. ed. ed., ser. Alan Apt series. Upper Sadle River, NJ: Pearson Prentice Hall, 2012, eS-Flandernstraße.

[15] E. Gamma, Ed., *Design patterns: elements of reusable object-oriented software*, 26th ed., ser. Addison-Wesley professional computing series. Boston ; Munich [u.a.]: Addison-Wesley, 2003, eS-Flandernstraße. [Online]. Available: 04

[16] J. Bewersdorff, *Objektorientierte Programmierung mit JavaScript: Direktstart für Einsteiger*, 2nd ed., ser. SpringerLink : Bücher. Wiesbaden: Springer Vieweg, 2018. [Online]. Available: http://dx.doi.org/10.1007/978-3-658-21077-9

[17] D. Louis and P. Müller, *Java: eine Einführung in die Programmierung*, 2nd ed., ser. Hanser eLibrary. München: Hanser, 2018. [Online]. Available: http://dx.doi.org/10.3139/9783446453623

[18] S. Purewal, *Learning Web Application development: [build quickly with proven Javascript techniques]*, 1st ed. Beijing ; Köln [u.a.]: O'Reilly, 2014, standort Göppingen. [Online]. Available: http://deposit.d-nb.de/cgi-bin/dokserv?id=4556796&prov=M&dok_var=1&dok_ext=htm

[19] R. Steyer, *JavaScript: die universelle Sprache zur Web-Programmierung*. München: Hanser, 2014. [Online]. Available: http://www.hanser-elibrary.com/action/showBook?doi=10.3139/9783446439474

[20] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[21] "TensorFlow.js." [Online]. Available: https://js.tensorflow.org/faq/

[22] "Introduction - Redux." [Online]. Available: https://redux.js.org/introduction

[23] "A brief history of JavaScript ? Ben Aston ? Medium," last accesed on 2018-05-15. [Online]. Available: https://medium.com/@benastontweet/lesson-1a-the-history-of-javascript-8c1ce3bffb17

[24] "JavaScript is the Future of Enterprise Application Development," last accesed on 2018-05-15. [Online]. Available: https://www.logicroom.co/javascript-is-the-future-of-application-development/

[25] "Wie unterscheidet sich JavaScript von Java?" last accesed on 2018-05-15. [On-

line]. Available: https://www.java.com/de/download/faq/java{_}javascript.xml

Index