Technical Guide

CA400

GUI for learning about Neural Networks

Student Name: Liam McAweeney

Student Number: 1415152

Project Supervisor: Charlie Daily

Abstract

This project involves developing a web application to create neural network models without writing any code. This involves research on topics and technologies I have not worked with before.
The application breaks down the process of creating a neural network and allows the user to piece it together using a graphical interface, the user hasn't to write any code in the implementation of a model. Their is real time analysis of the models training, this will help the user better understand and learn about what the network is doing and how it is performing.

This application promotes learning through exploration and discovery.

Introduction

Overview

The aim of this project is to develop a web application for users to learn about and create their own neural networks.

Motivation

The motivation for this project stems from my strong interest in the field neural networks. This field is very topical in today's world,we are living in a data driven world where everything around us is generating data. More and more applications of neural networks are being implemented each week which improve the way we live.

From my own experience, delving into neural networks is a daunting task because of the perceived difficulty involved. Saying that, when you start research neural networks, the fundamental theory behind them is not difficult. The implementation of them however can become very tricky, especially for those who have little or no experience in programming.

I created this project to help people learn about and experiment with neural networks without having to get into the nitty gritty of code, but be using a simple user friendly application. The hope is that this project would spark an interest in programming neural networks for those novice in the field.

Research

There has been a number of significant research elements involved in this project. I had to gain a thorough understanding about neural networks and techniques I could use.

Although I had a good understanding of neural networks, I thought it would be a good idea to try understanding it from my end user's perspective. I used multiple resources when researching, mainly videos tutorials for youtube.com and blogs from medium.com. This insight helped my develop a simplified design to allow the implementation of a NN become easy, understandable to the user.

When research techniques for implementing the NNs I found two libraries which would be very helpful, Keras and TensorFlowJS (TFJS). I initially opted to use Keras.

Real time analysis of the training of a NN was a vital user requirement and so I had to thorough research this, which led me recognized TJFS potential and I decided to change my implementation to use it instead of Keras.

While developing this project, I used youtube to help understand more specific aspects of Python and React.

I researched design elements to ensure the UI would appeal to the users and promote user retention. Such articles can be found in the appendix.

Glossary

Keras: An open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML.

Tensorflow: A free and open-source software library for dataflow and differentiable programming across a range of tasks. Highly regarded for machine learning applications. In this application TensorFlowJS is used and is abbreviated to TFJS

React: A JavaScript library for building user interfaces. It is the view layer for web applications.

ORM: Object-relational mapping in computer science is a programming technique for converting data between incompatible type systems using object-oriented programming languages.

SPA: Single-Page Applications are Web apps that load a single HTML page and dynamically update that page as the user interacts with the app. SPAs use AJAX and HTML5 to create fluid and responsive Web apps, without constant page reloads. This means much of the work happens on the client side, in JavaScript.

NN: Artificial neural networks (ANN) or connectionist systems are computing systems vaguely inspired by the biological neural networks and astrocytes that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.

General Description

System Functions

This application is designed to be a point of learning and experimentation for people interest in neural networks. The graphical user interface strives to user-friendly, intuitive and most importantly as simplistic as possible.

When implementing a NN model, there are multiple phases. In general these phases tend to merge and cause a lack of understanding in turn impairs the accuracy of models. To try tackle this issue I segmented each step ensure the user could differentiate between them. This segmentation approach affected the design phase of the application which will be discussed later in the design section.

Designing a model

The first phase of implementing a NN model is the design phase, thus it is the first functionality of the application. Here the users decides what they are going to classify, the number of hidden layers in the network and what each layer is made of. After a user designs their model they give it a name and it saved for using in the next phase.

Training a model

The next phase in implementing a model is the training phases. This phases is the most time consuming as training a model can take a long time depending on the complexity of it’s design. The user selects a model which they have previously designed. The user decides the number of epochs, batch size, number of inputs and the validation split. They then start the training. Real time analysis of the models training is displayed for the user, this will help to user understand the effect of complex networks in terms or accuracy, loss and time taken. The user can see if the accuracy isn’t improving and can implement early stopping manually. After the training is complete the model is saved.

Analysing a model

After training a model the user can examine it in detail in the analytical phase. Here they can compare different models in terms of accuracy, loss, validation, time taken to training etc. This will be achieve through dynamically displaying various metrics gained from the training phase through interactive charts.

Classifying with a model

After training models and analysing them,the user can classify an input using the model. Correctly there is two datasets for the system, numbers(0-9) and color. The user can create an input and classify it using their model which they fully created themselves. This is the final functionality in terms on implementing a model.

User summary

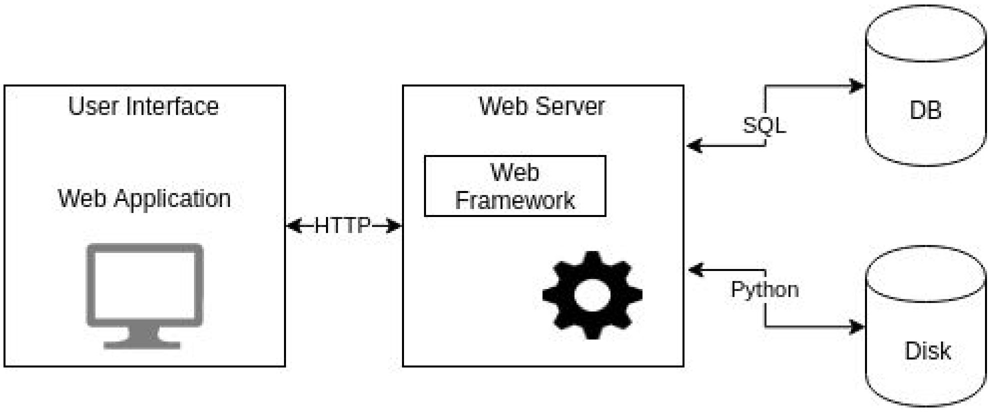
The summary functionality is the dynamic loading of the user's homepage. It will contain information about the user's models and best model in terms of accuracy for example. As more information is gathered there is a potential to expand on this functionality.

Design

The application contains the majority of the requirements outlined in the functional specification. The requirements have evolved and adapted which is natural in any software development lifecycle process.

Reviewing my requirements ensured my application's functionality was accurate, consistent and necessary.

System Architecture



The diagram above is a high level view of the system's architecture.

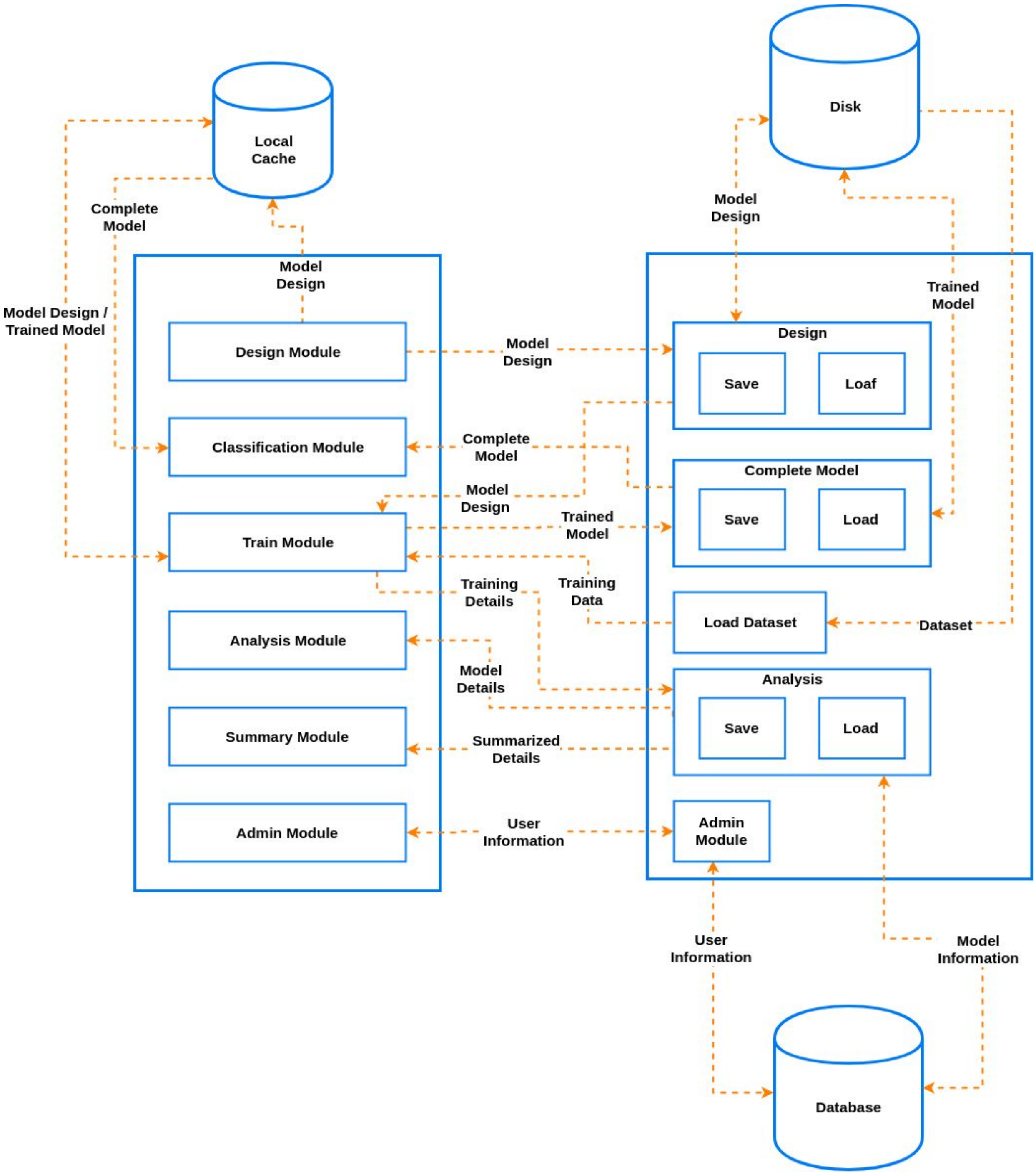
The client tier, where the user interacts with the application is built using react and bootstrap. React is a open-source Javascript and HTML library created by facebook. I decided to use React as I wanted to design an SPA for efficiency of rendering the application. This is very important as the model's implementation is fully handled in the browser, ie. if the user is training a model, rendering the real time analysis will not impair it. React is known for its performance capabilities and it will interact well with TFJS.
Bootstrap is open-source CSS framework which is popular for web application development. This tier connects to the web server through a HTTP connection.

TFJS is handling the implementation of the NN models and means that the complex logic is no longer in the backend but in the frontend. This allows the use of a lightweight backend.

The web framework which is connected to the client tier is Flask. Flask is lightweight web framework which is python in python. It receives requests from the frontend and gets the information wanted from the database or disk, parses it and sends it back in a response. The database I opted to use for this project is a MySQL database. I choose this because I have more experience with them and I already have an extensive learning curve.

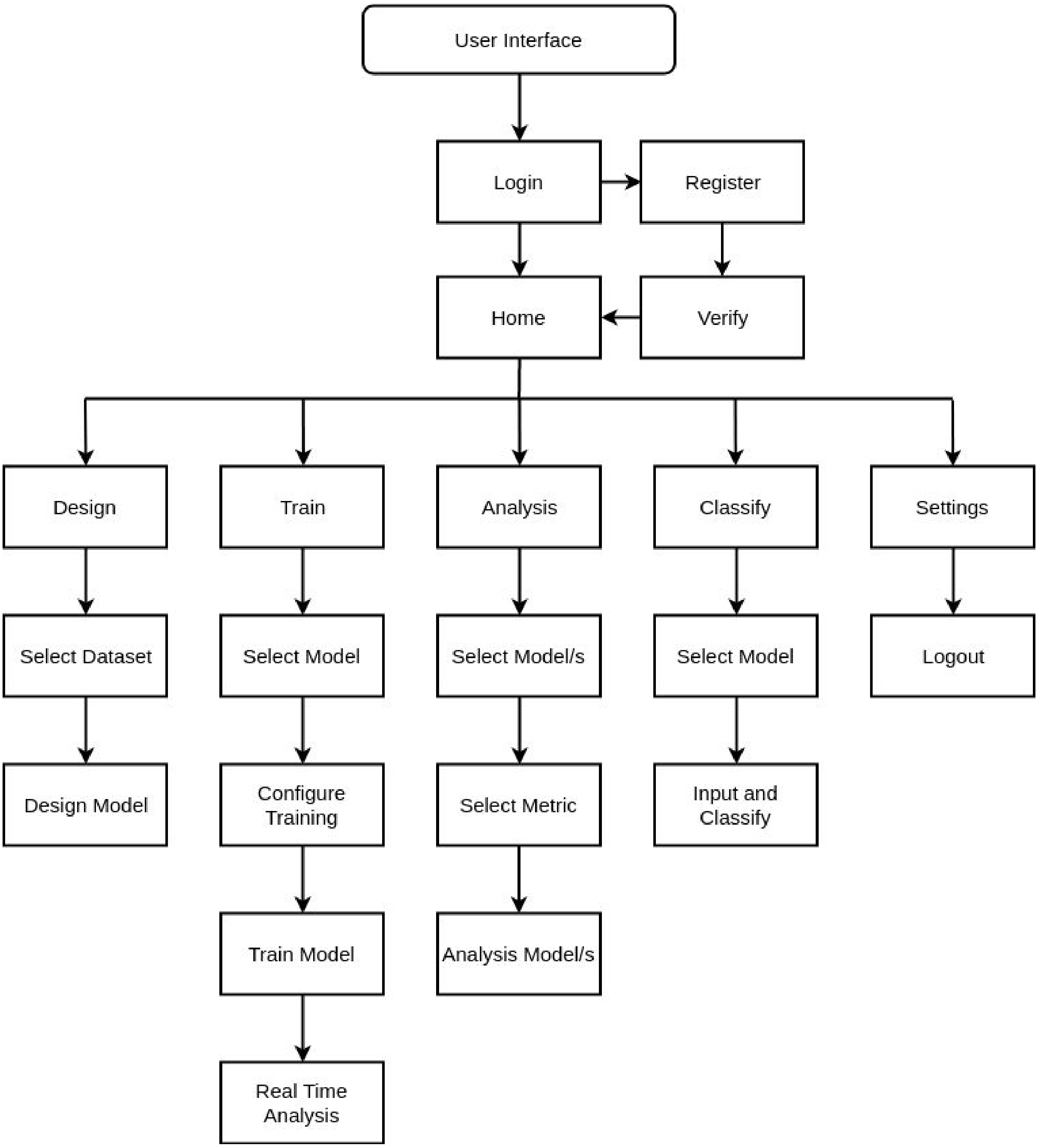
High Level Designs

High Level System Diagram



Above is a high level design for the system. It shows the interaction between modules on the on the client's side and the server's side. It also shows the interaction between modules on the server and the database and disk storage.
The client also interacts with the browser's local storage. This is done when possible to ensure optimal side when retrieving data. The datasets used for training the models are stored on disk storage on the web server.

High Level System Functionality Diagram



The diagram above shows the core functionality of the system and how each function lead to the next. Initially the systems check if the user is logged in or registered. If the user is logged in, they will go straight to the home page and if there not logged in they must first login to navigate to the homepage. If the user is not register they must register and then follow the link which the system automatically by emailed to them to verify their email address.

Once the user is on the homepage they will see summarised details regarding their models. The purpose of this is to allow the user see the progress they are making in terms of accuracy of their models. From here they can navigate to the main functions of the system.

Design

When in the design component the user must select which dataset they intend to design their model for. Based on the dataset chosen the input and output layers of the model will automatically be defined. The configuration options for the design are made available dynamically depending on the dataset chosen for the model, this is performed to ensure the user model will not create a design not applicable to their chosen dataset. From here the user can add layers to the model. The configuration details are save for the analysis and the model is compiled and saved to storage. The address of the where the model is save and other details are save on the database.

Train

When in the train component the user must select the model they want to train. When the model is selected a a summary of the model’s design is displayed on the screen to remind the user of the design. The user can then configure the training methodology and start the training. Once the training has commenced, real time accuracy and loss metrics are displayed, this is at batch level and at epoch level. The information is displayed in the form of a trend graph, best demonstrating the upward accuracy trend and downward loss trend. The user will be able to inspect the training to spot overfitting or underfitting. The user will be able to manually activate early stopping if they deem appropriate.
When the training is complete the user will be prompted with the over accuracy and loss of the model and the model will be saved to storage.
Each model design can have many trained versions, this functionality will allow users to better learn how to recognize over complexity in design, too much training of the model or not enough.

Analysis

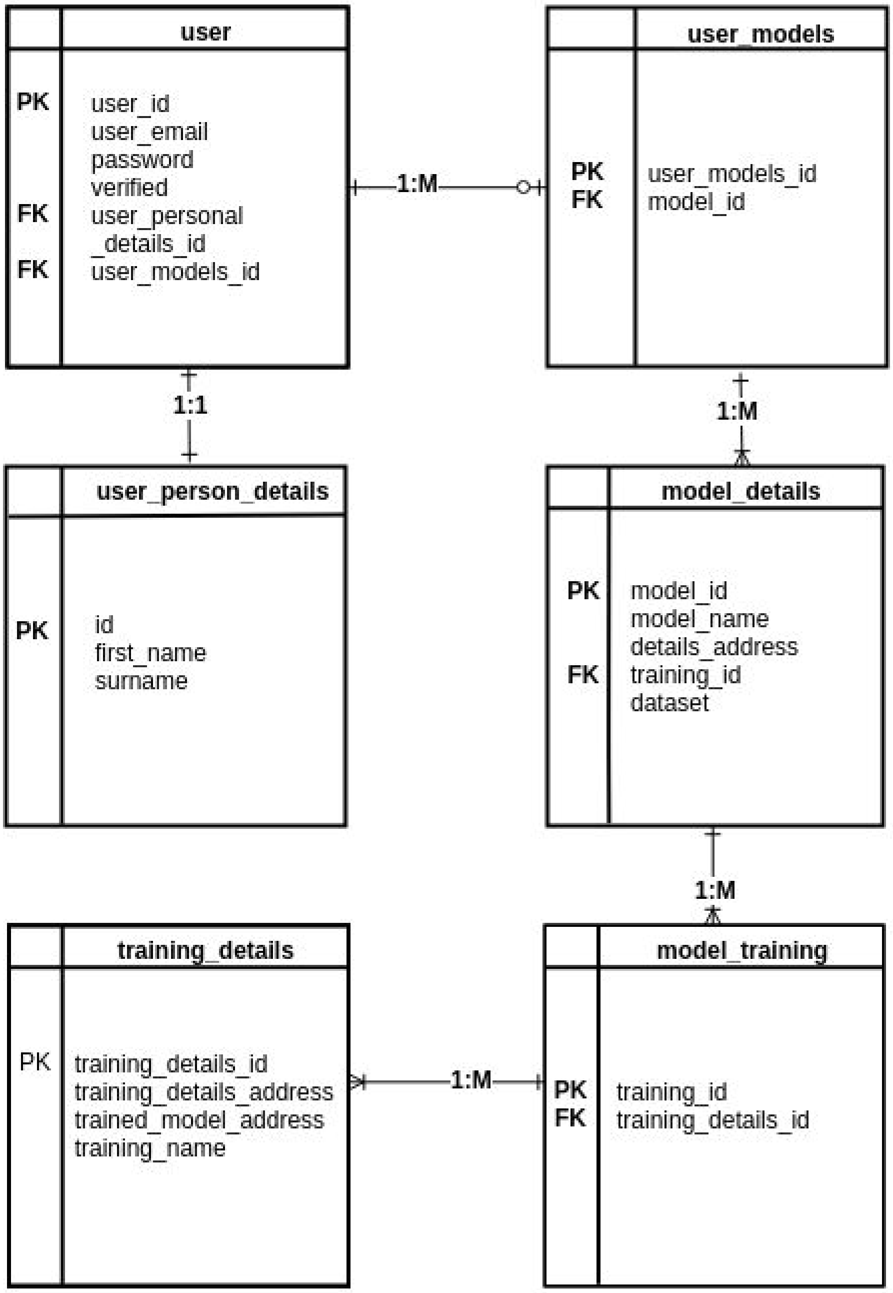
In the analytical section the user must select the model or models they want to analyze. They can analyze individual or multiple models at a single time. There are several metrics in which the user can analyze the models under. The charts are interactive and easy to use.

Classification

Here the user once again selects a model, the model with the highest accuracy for the chosen design is recommended to the user. Depending on what the model was designed to classify, the user can input and then perform the classification.

The user can log out through navigating to the settings and the logout button.

Entity Diagram



This diagram illustrates how different entities relate.

<u>Users: </u>The user table will store information about the user, this includes their email and password. The user's password is appended by a salt and is then hashed for maximum security. The user's personal details are stored in a separate table which it has a one-to-one

relationship with. Since a user can have multiple models, they have a one to many relationship with user_models.

User Person Details: This table contains person details about the user. This information is stored in a separate table as few modules need access to it. This is a security procedure to protect the user's person details.

Users models: This table is connects the user to their models. This table ensure 3rd normal form and no redundant data.

Model Details: This table stores information on a models design, this includes the address of where the model is stored and the name given to the model. Because each design can have multiple trained models it has a one-to-many relationship with the model training table.

Model Training: This table is connects the design to their trained models. This table ensure 3rd normal form and no redundant data.

Training Details: This table stores information on each trained model including the address where the model is stored and where the analytical data is stored

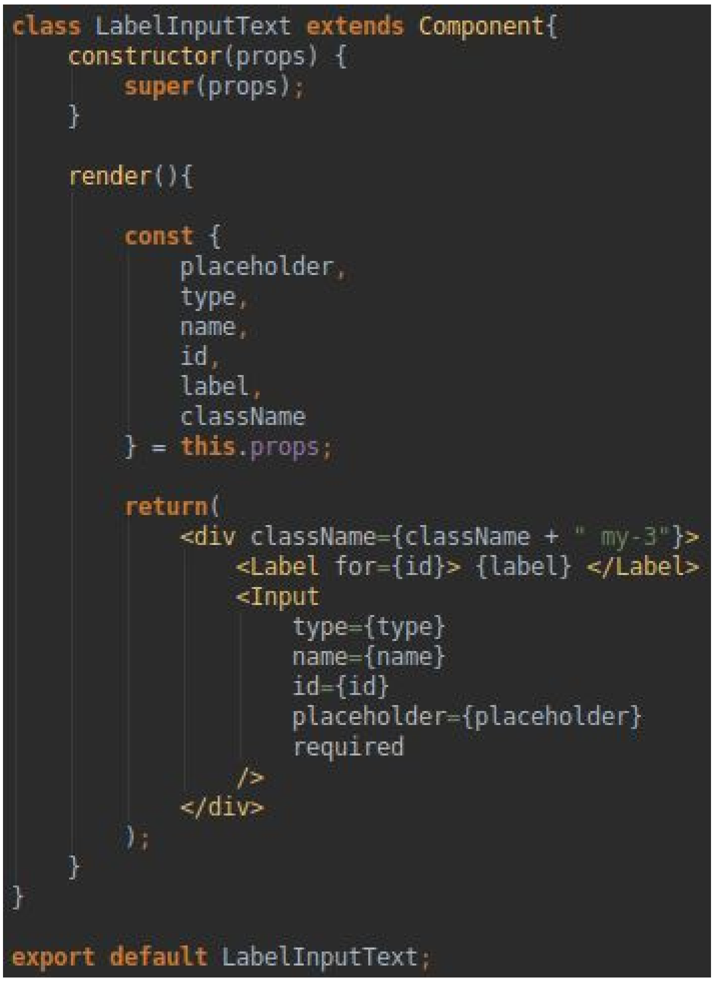
Implementation

The implementation of this application can be segmented into two main components, the frontend and the backend. In this section I will describe how the main functionality in each component is implemented. I will also discuss the main libraries used.

Frontend

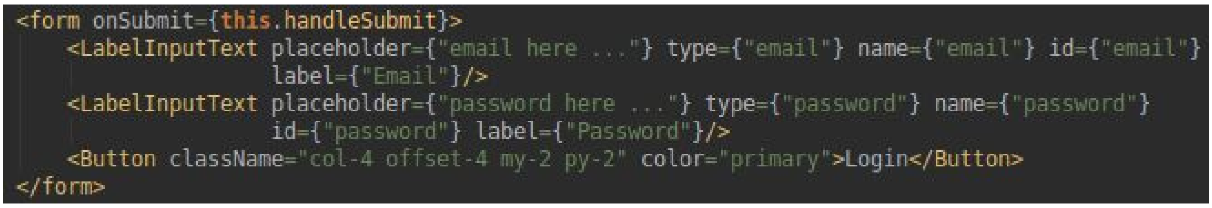
React

The frontend is written using react, as stated before react was chosen for performance reasons and simplicity of integration of TFJS. React encourages the creation of reusable components. An example, which is my most reused component is the input component. I created a fully customizable component which other components implement then. This feeds into the hierarchical nature of react and its components.

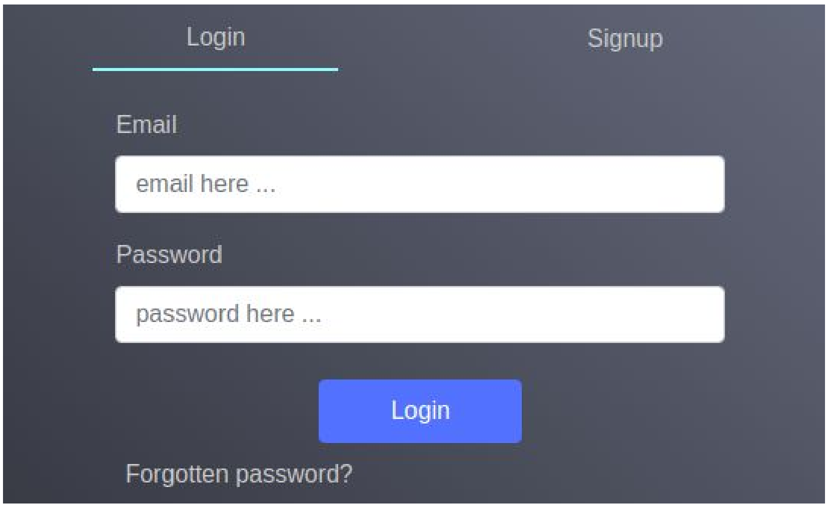


```
class LabelInputText extends Component{
    constructor(props) {
        super(props);
    }

    render(){

        const {
            placeholder,
            type,
            name,
            id,
            label,
            className
        } = this.props;

        return(
            <div className={className + " my-3"}>
                <Label for={id}> {label} </Label>
                <Input
                    type={type}
                    name={name}
                    id={id}
                    placeholder={placeholder}
                    required
                />
            </div>
        );
    }
}

export default LabelInputText;
```

The components name is LabelInputText. When using this component, the parent component must pass information in the form of variables of constants into it, these are known as props. As you can see this component uses two other components. Label and Input, both of which are ReactStrap components, I will discuss this library in a later section. The label variable which is passed into this component is then passed to the Label component. ClassName is the equivalent of class in HTML. “my-3” is a bootstrap class which is margin on the top and bottom of the component of 3 rem. This particular component is a stateless component which means it doesn’t have is own lifecycle management, but inherits its parent’s state.



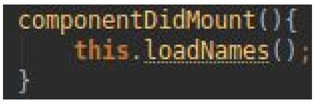
```
<form onSubmit={this.handleSubmit}>
    <LabelInputText placeholder={"email here ..."} type={"email"} name={"email"} id={"email"}
                    label={"Email"}/>
    <LabelInputText placeholder={"password here ..."} type={"password"} name={"password"}
                    id={"password"} label={"Password"}/>
    <Button className="col-4 offset-4 my-2 py-2" color="primary">Login</Button>
</form>
```



Here is an example of the component being implemented. The reuse of components further uses the rendering time of the application.

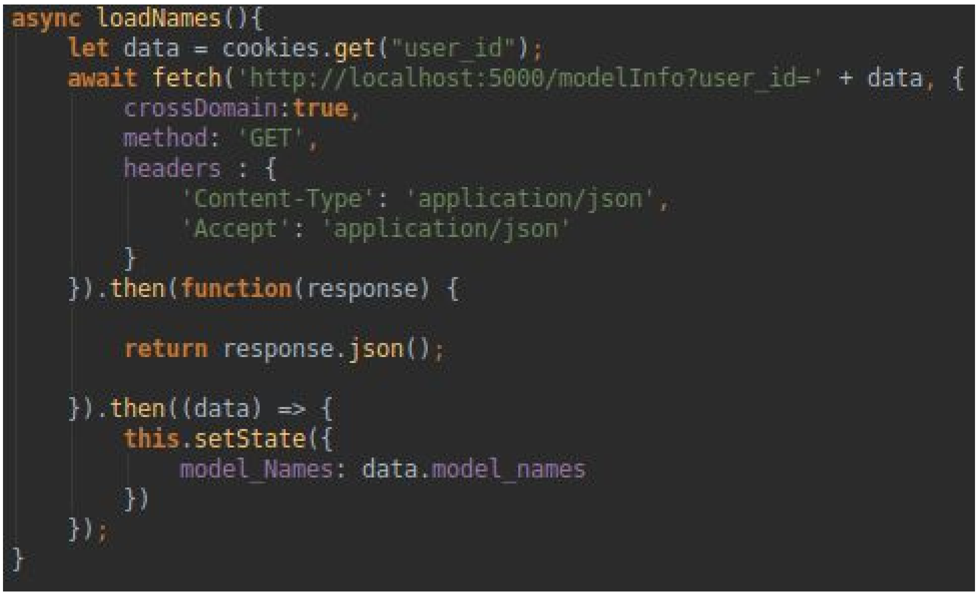
As briefly discussed above, a stateless component as one which has not lifecycle management system. This means the component doesn’t decide when it changes or rerenders, but it’s closest predecessor which is stateful does. React has a notion of state, this is state private variables inside the component. These variables influence what that component renders. When the state of a component is updated the component itself rerenders and so does all its successors. An example would be when the user selects a model for analysis. After they select the model from the dropdown, each of the charts and information outputted has to be updated to be representative of that model.

When a component is rendered if it contains a componentDidMount function it will be executed it the rendering was successful.



```
componentDidMount(){
    this.loadNames();
}
```

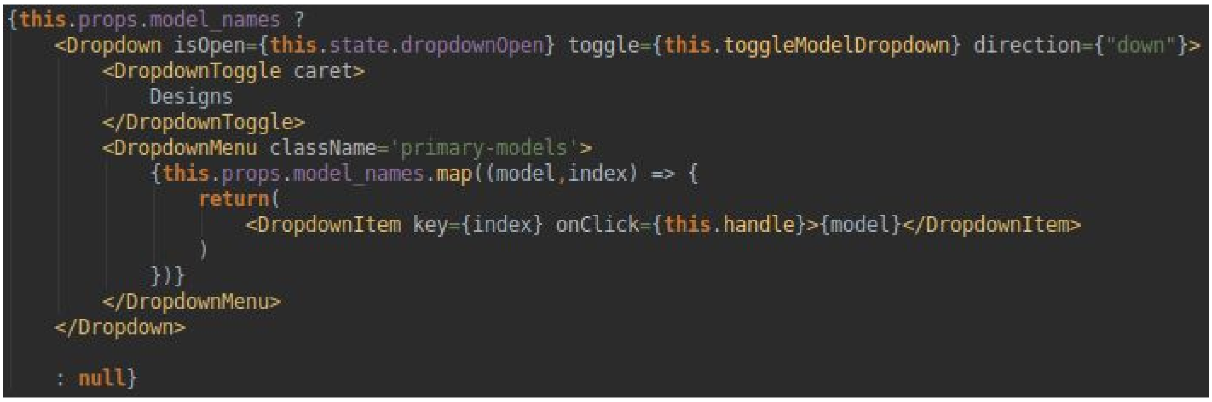
This example calls the loadNames function. This function is an async function which makes a request to the backend for all the model names associated with the user and uses them to populate a dropdown.



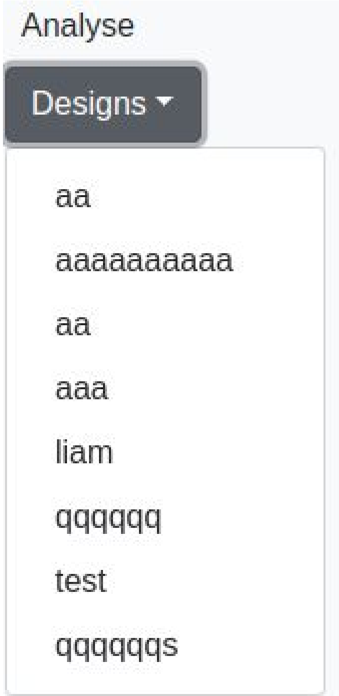
```
async loadNames(){
    let data = cookies.get("user_id");
    await fetch('http://localhost:5000/modelInfo?user_id=' + data, {
        crossDomain:true,
        method: 'GET',
        headers : {
            'Content-Type': 'application/json',
            'Accept': 'application/json'
        }
    }).then(function(response) {

        return response.json();

    }).then((data) => {
        this.setState({
            model_Names: data.model_names
        })
    });
}
```

As you can see a fetch call which uses AJAX to connect the backend over HTTP to get the names. When the function receives a response it passes it into a then function, this handles the promise return due to the function being async. The response is then converted into a json file and parsed for the model names. They are then saved to state, which then causes a rerender.

These names are then passed as props to the sidebar component which contains the dropdown component.



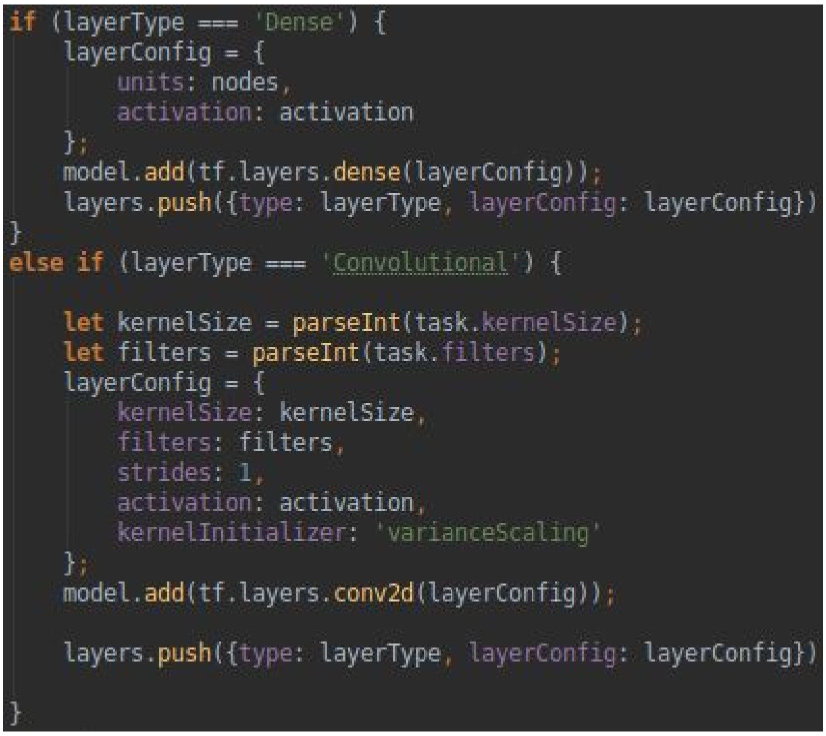
```
{this.props.model_names ?
    <Dropdown isOpen={this.state.dropdownOpen} toggle={this.toggleModelDropdown} direction={"down"}>
        <DropdownToggle caret>
            Designs
        </DropdownToggle>
        <DropdownMenu className='primary-models'>
            {this.props.model_names.map((model,index) => {
                return(
                    <DropdownItem key={index} onClick={this.handle}>{model}</DropdownItem>
                )
            })}
        </DropdownMenu>
    </Dropdown>

    : null}
```



In the above snippet the uses of reacts “Inline If-Else with Conditional Operator” checks if there was any names passed, if not it doesn’t render the component. The dropdown then iterates of the array of names passed to it and renders a dropdown option for each and passes a function to be executed if clicked.

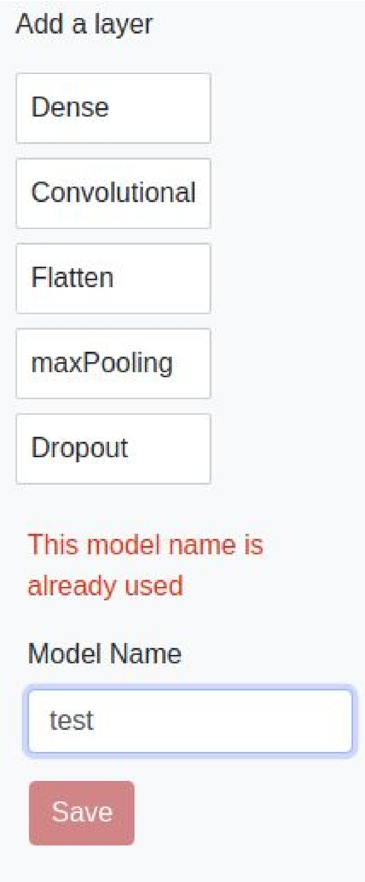
Model Implementation

The design component handles the generation of the models architecture. When created a model, TFJS is used. First a model is initialized by the sequential function which layer object can be added to to create the models topology. Simply a layer is defined by the user through the drag and drop mechanism implemented and then added to the model. There are five types of layers which can be added. Below is dense layer and convolutional layer.



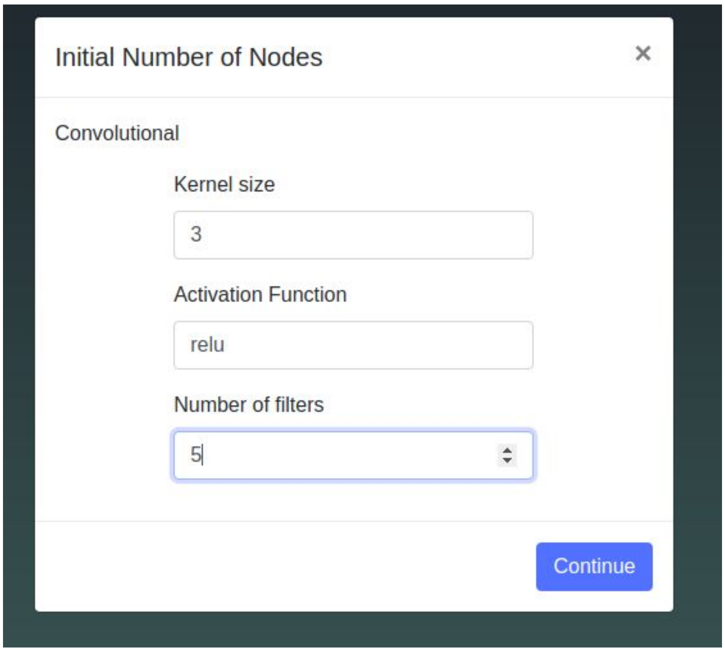
```
if (layerType === 'Dense') {
    layerConfig = {
        units: nodes,
        activation: activation
    };
    model.add(tf.layers.dense(layerConfig));
    layers.push({type: layerType, layerConfig: layerConfig})
}
else if (layerType === 'Convolutional') {

    let kernelSize = parseInt(task.kernelSize);
    let filters = parseInt(task.filters);
    layerConfig = {
        kernelSize: kernelSize,
        filters: filters,
        strides: 1,
        activation: activation,
        kernelInitializer: 'varianceScaling'
    };
    model.add(tf.layers.conv2d(layerConfig));

    layers.push({type: layerType, layerConfig: layerConfig})

}
```

The layerConfig is parsed for the user’s inputs and then added to the model. It is also added to a layers array which is used in the analytics and summary components. The model is then saved backend.



This model name is already used

Model Name: test



Initial Number of Nodes

Convolutional

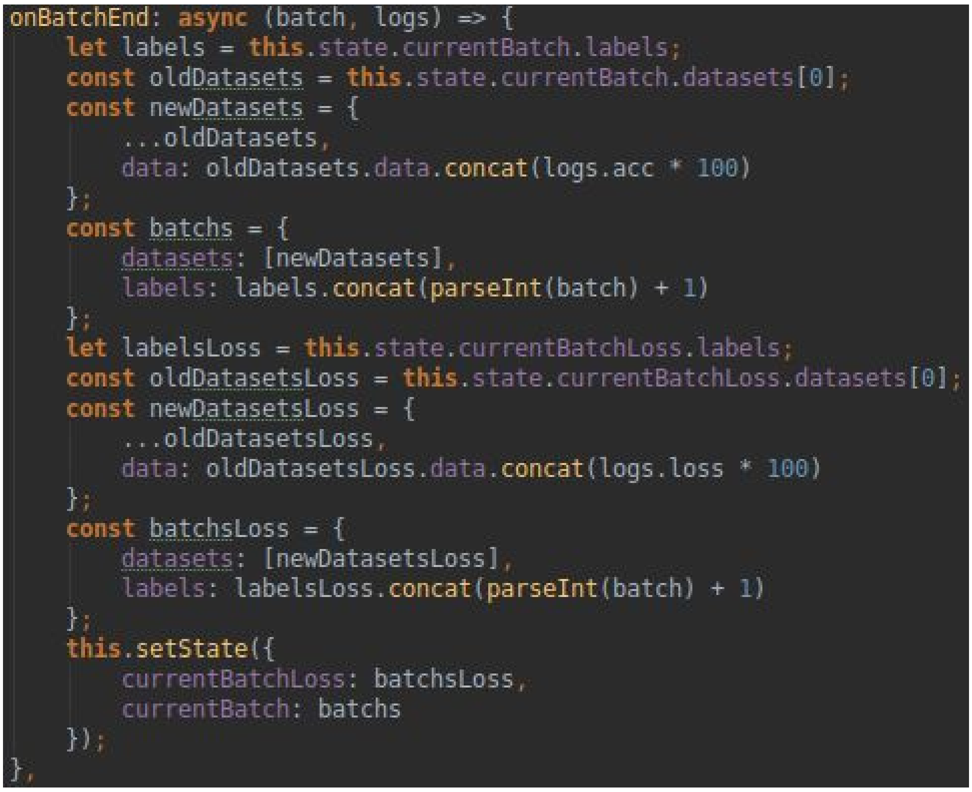
Kernel size: 3

Activation Function: relu

Number of filters: 5

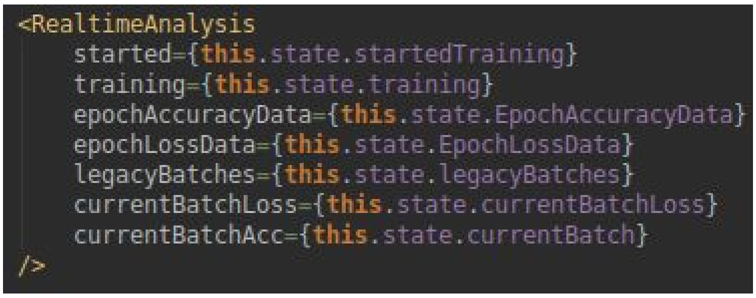
In the images above you can see the layers a user can add to their model, They simply drag the layer into the model representation which is the main display. When this is completed they are prompted with a modal regarding specifics on the layer.

In the Train component the model is loaded from the backend and user then configures the training they want to implement. Once the user has finalised the training configuration, the model is ready to be fitted, to the dataset. The fit function has been customized to obtain several metrics for analytics. This was completed through the use of callbacks at the start and end training, start and end of each epoch and the start and end of each batch. This allows for real time analysis of the training and for further comparison of the models performance compared to other models.



```
onBatchEnd: async (batch, logs) => {
    let labels = this.state.currentBatch.labels;
    const oldDatasets = this.state.currentBatch.datasets[0];
    const newDatasets = {
        ...oldDatasets,
        data: oldDatasets.data.concat(logs.acc * 100)
    };
    const batchs = {
        datasets: [newDatasets],
        labels: labels.concat(parseInt(batch) + 1)
    };
    let labelsLoss = this.state.currentBatchLoss.labels;
    const oldDatasetsLoss = this.state.currentBatchLoss.datasets[0];
    const newDatasetsLoss = {
        ...oldDatasetsLoss,
        data: oldDatasetsLoss.data.concat(logs.loss * 100)
    };
    const batchsLoss = {
        datasets: [newDatasetsLoss],
        labels: labelsLoss.concat(parseInt(batch) + 1)
    };
    this.setState({
        currentBatchLoss: batchsLoss,
        currentBatch: batchs
    });
},
```

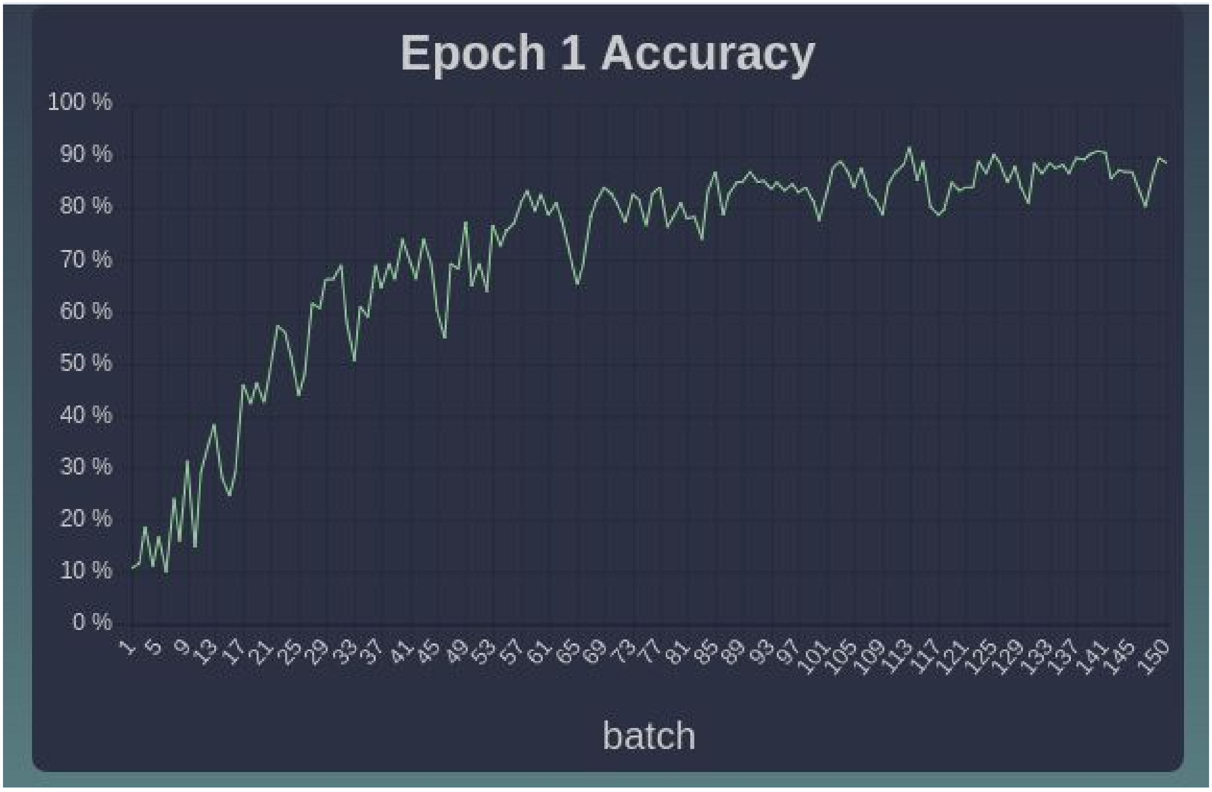
The snippet above is the callback function executed at the end of each batch. All the information is parsed and stored to ensure maximum performance of real time charting for analysis. All the information gathered is passed to the RealtimeAnalysis component which displays the trends during training.



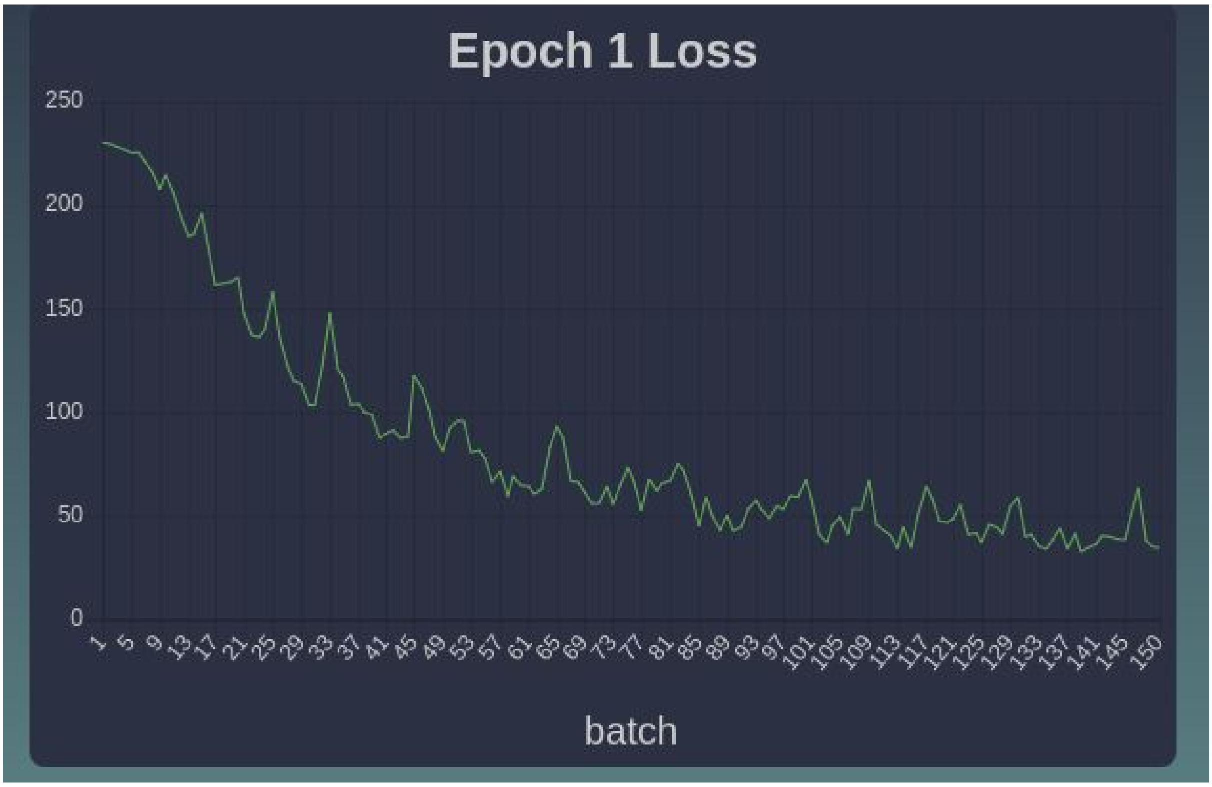
```
<RealtimeAnalysis
    started={this.state.startedTraining}
    training={this.state.training}
    epochAccuracyData={this.state.EpochAccuracyData}
    epochLossData={this.state.EpochLossData}
    legacyBatches={this.state.legacyBatches}
    currentBatchLoss={this.state.currentBatchLoss}
    currentBatchAcc={this.state.currentBatch}
/>
```

This results in the following snippets.

The accuracy dramatically improving over the first epoch

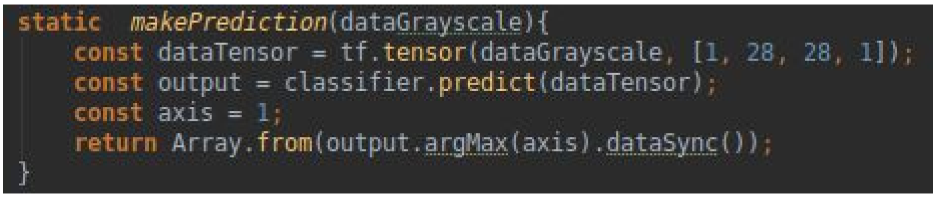


The loss Dramatically decreasing over the first epoch also.



As you can see this provides the user with very useful information about their model.

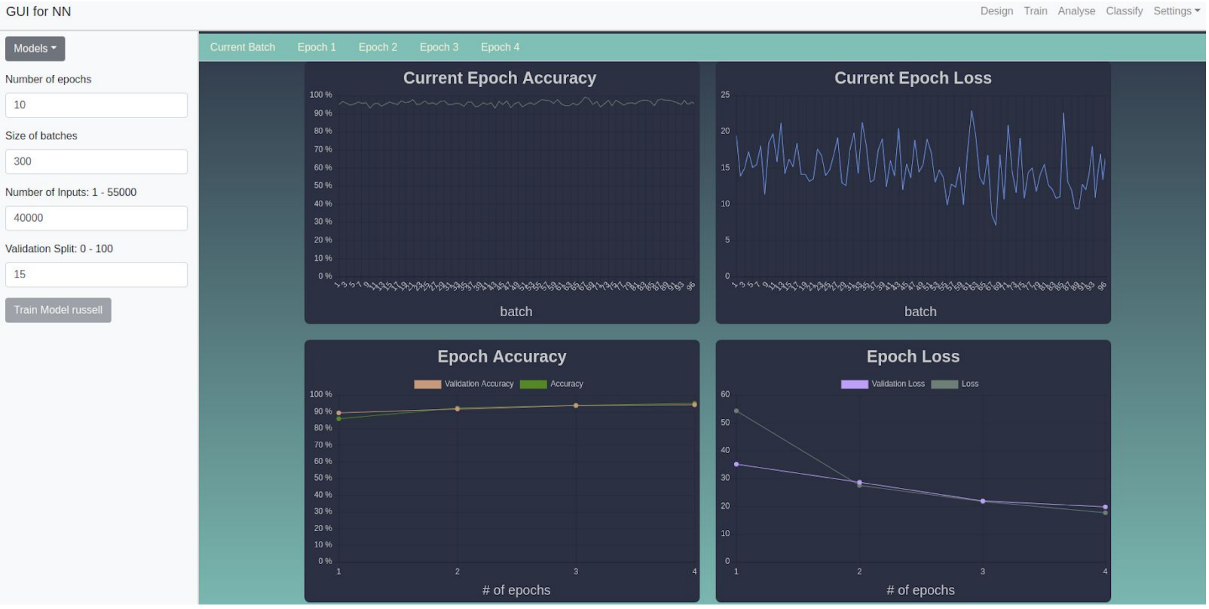
When using a model to make a classification on a handwritten digit, the user draws a number into a canvas and model make a prediction on the number and outputs it.



```
static  makePrediction(dataGrayscale){
    const dataTensor = tf.tensor(dataGrayscale, [1, 28, 28, 1]);
    const output = classifier.predict(dataTensor);
    const axis = 1;
    return Array.from(output.argMax(axis).dataSync());
}
```

User Interface

As previously mentioned I decided to use react its performance so I could harness is hierarchical design pattern to create a SPA. The UI must be simplistic but adaptive to further maintenance. Components must be built for dynamism and reusability. The design of this application it built of reusability and consistency. The application must be uniform in design and layout. It should allow the user to easily master it and be intuitive to use. Since the creation of a NN model follows a linear path, I decided to make the user's journey through this application mapo to that journey. This was implemented through a clear hierarchy of functionality. This in turn ensures the user's experience of creating a model is simply, intuitive and beneficial in terms of learning outcomes.



Number of epochs: 10

Size of batches: 300

Number of Inputs: 1 - 55000: 40000

Validation Split: 0 - 100: 15

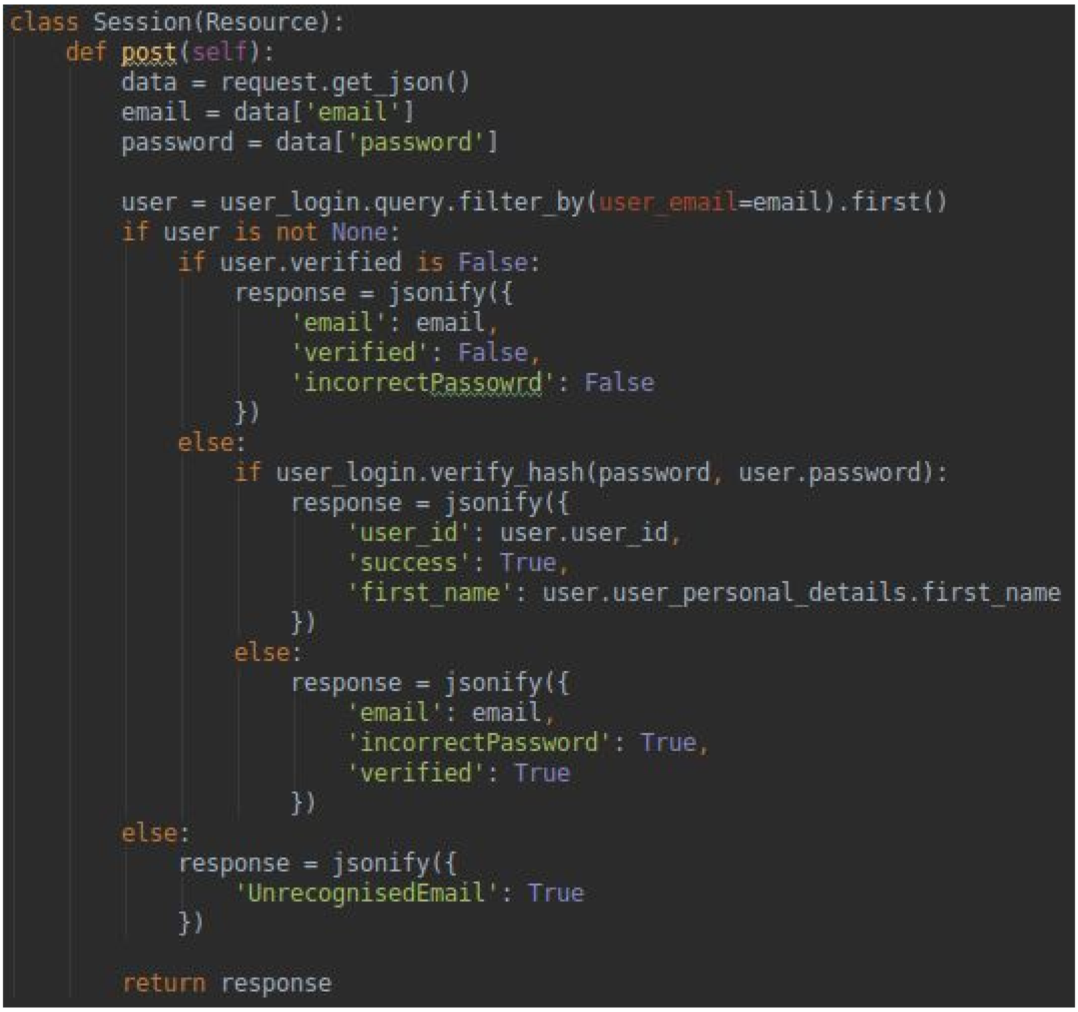
Above is a screenshot of a model training.The sidebar on the left is consist through the each of the model development stages, the navbar is consistent through the whole application and as well as the color scheme.

Backend

Flask

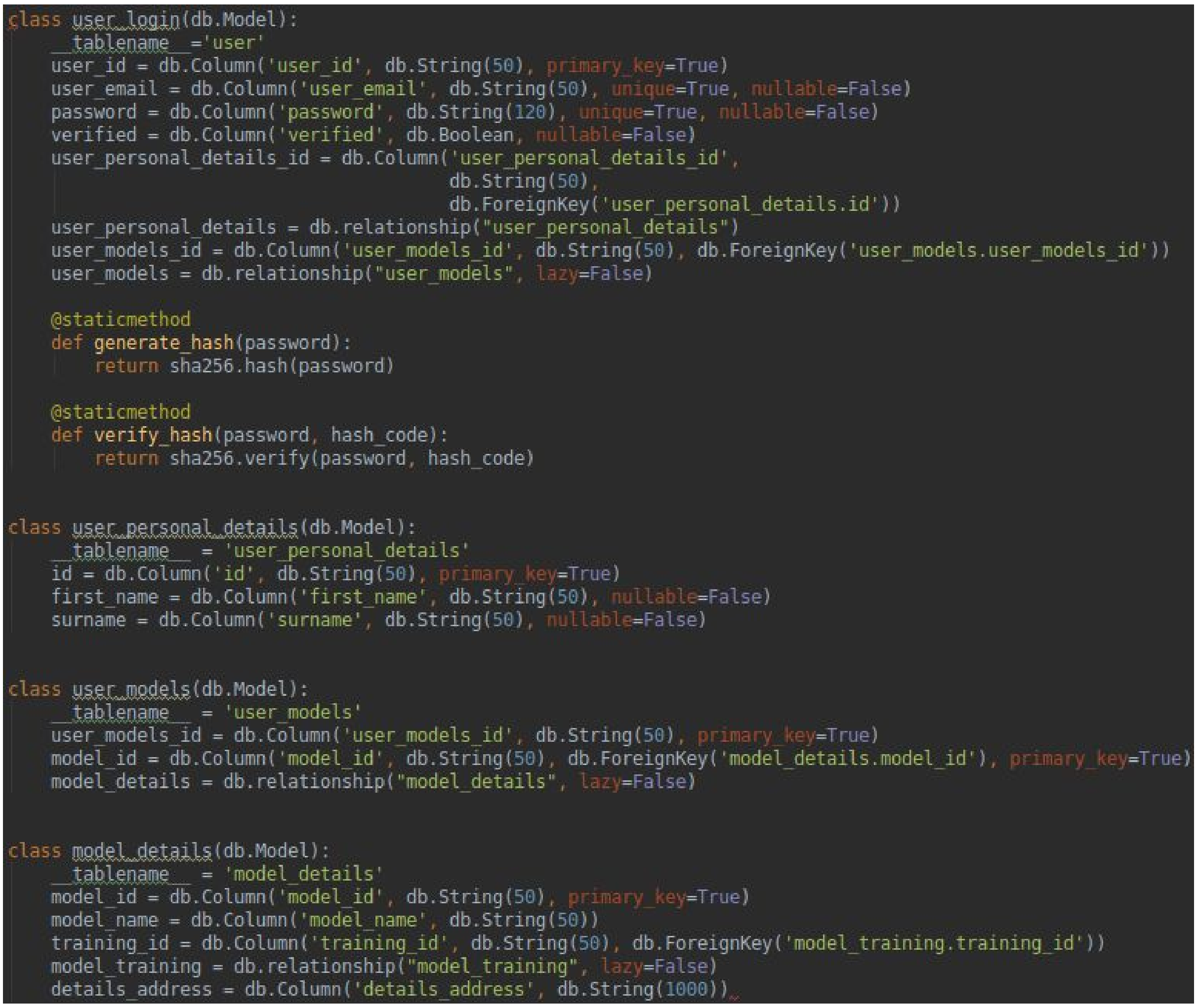
The backend is written in Python and uses the Flask. I decided to use it has a small learning curve relative to other Python web frameworks such as Django. During the implementation of this application I had developed models with Keras on the backend but migrated to TFJS as it best enabled real time analysis of training a model. The backend is lightweight, it handle user administration, saving and loading models to and from disk, parsing model analytical data and interacting with the database. I am using an ORM to interact with the MySQL database.

Below is flask handling a user logging into the system.



```
class Session(Resource):
    def post(self):
        data = request.get_json()
        email = data['email']
        password = data['password']

        user = user_login.query.filter_by(user_email=email).first()
        if user is not None:
            if user.verified is False:
                response = jsonify({
                    'email': email,
                    'verified': False,
                    'incorrectPassowrd': False
                })
            else:
                if user_login.verify_hash(password, user.password):
                    response = jsonify({
                        'user_id': user.user_id,
                        'success': True,
                        'first_name': user.user_personal_details.first_name
                    })
                else:
                    response = jsonify({
                        'email': email,
                        'incorrectPassword': True,
                        'verified': True
                    })
        else:
            response = jsonify({
                'UnrecognisedEmail': True
            })

        return response
```

Below is an few of the classes created to map to instances in the Db.



```
class user_login(db.Model):
    __tablename__ ='user'
    user_id = db.Column('user_id', db.String(50), primary_key=True)
    user_email = db.Column('user_email', db.String(50), unique=True, nullable=False)
    password = db.Column('password', db.String(120), unique=True, nullable=False)
    verified = db.Column('verified', db.Boolean, nullable=False)
    user_personal_details_id = db.Column('user_personal_details_id',
                                         db.String(50),
                                         db.ForeignKey('user_personal_details.id'))
    user_personal_details = db.relationship("user_personal_details")
    user_models_id = db.Column('user_models_id', db.String(50), db.ForeignKey('user_models.user_models_id'))
    user_models = db.relationship("user_models", lazy=False)

    @staticmethod
    def generate_hash(password):
        return sha256.hash(password)

    @staticmethod
    def verify_hash(password, hash_code):
        return sha256.verify(password, hash_code)


class user_personal_details(db.Model):
    __tablename__ = 'user_personal_details'
    id = db.Column('id', db.String(50), primary_key=True)
    first_name = db.Column('first_name', db.String(50), nullable=False)
    surname = db.Column('surname', db.String(50), nullable=False)


class user_models(db.Model):
    __tablename__ = 'user_models'
    user_models_id = db.Column('user_models_id', db.String(50), primary_key=True)
    model_id = db.Column('model_id', db.String(50), db.ForeignKey('model_details.model_id'), primary_key=True)
    model_details = db.relationship("model_details", lazy=False)


class model_details(db.Model):
    __tablename__ = 'model_details'
    model_id = db.Column('model_id', db.String(50), primary_key=True)
    model_name = db.Column('model_name', db.String(50))
    training_id = db.Column('training_id', db.String(50), db.ForeignKey('model_training.training_id'))
    model_training = db.relationship("model_training", lazy=False)
    details_address = db.Column('details_address', db.String(1000))
```

Libraries

While implementing this application I used different libraries, below is a short summary on each of the libraries I found most useful.

Frontend

TensorFlowJS: Used for creating NN models and using them for classification.
ReactStrap: Provides predefined components and strategic layout scheme to enforce layout consistency.
WebPack: A module bundler. Its main purpose is to bundle JavaScript files for usage in a browser.

Backend

Flask SQLAlchemy: ORM used for mapping instances of python classes to instance in a table in the database.
Flask Restful: Enable the creation and implementation of restful APIs.
Passlib: Used for hashing and adding salt to passwords before adding to the database for extra security.

Validation

Environment Testing

Environment testing represents the testing of this application in various real-world environments that it may be exposed to. This means different operating systems, different browsers, and various screen resolutions.

Initially I tested that the behaviour of the UI was normally in each of the browsers. Since most of the business logic is hosted in the browser this wouldn't be enough. I performed a smoke test (which os a non-exhaustive set of test that aims at ensuring the most important functions of the application operate correctly) on each browser

The smoke test consisted of:

- Designing a model
- Training a model
- Using a model for classification

OS	BROWSER	1920x1200	3840x2160
Windows 10	Chrome	Passed	Passed
Windows 10	Firefox	Passed	Passed
Windows 10	Edge	Passed	Passed
Ubuntu 17	Chrome	Passed	Passed
Ubuntu 17	Firefox	Passed	Passed

The web application operated normally across each of the browsers.

While testing on browsers I tested the application on a tablet and it rendered perfectly, but the functionality of training a model was impair. Through further research I realised that I would have to implement a library which would be more suited to a mobile device. Due to this I decided to remove the system requirement for responsiveness in mobile devices. Although this is definitely an aspect I will work on in the future.

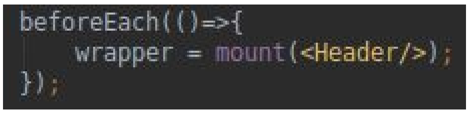
Unit Testing

Unit testing is testing which focuses on individual components of software projects ensuring that each component functions as required. The complex components in this application as situated on the frontend.

React

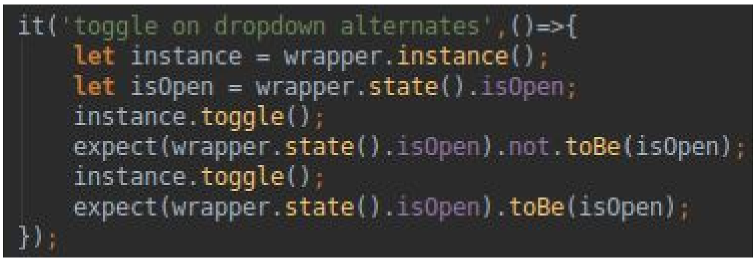
While unit testing my frontend I used Jest and Enzyme. Jest is a JavaScript testing framework with a focus on simplicity. It has lots of tutorials on testing with react, this is why I chose it.Enzyme is a Javascript testing utility for React that makes it easier to test your

React Components' output. You can also manipulate, traverse, and in some ways simulate runtime given the output.
When running unit tests on React components, you must mock them, eg. create a copy of the component.



```
beforeEach(()=>{
    wrapper = mount(<Header/>);
});
```

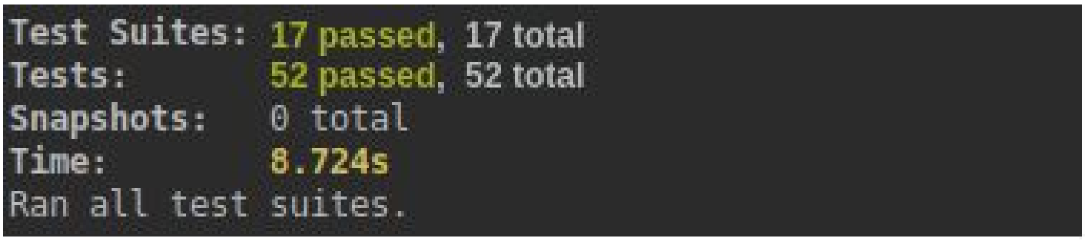
Above I mocked the Header component and assigned it to wrapper. The beforeEach function is ran beforeeach test in its test suite.



```
it('toggle on dropdown alternates',()=>{
    let instance = wrapper.instance();
    let isOpen = wrapper.state().isOpen;
    instance.toggle();
    expect(wrapper.state().isOpen).not.toBe(isOpen);
    instance.toggle();
    expect(wrapper.state().isOpen).toBe(isOpen);
});
```

Above is a snippet to of a unit test. I create an instance of the component which is not “alive”,i.e render for the first time with default state. I then store the value of the isOpen state in a local variable. I then activate the toggle function,which the user will do with a click. This test checks that when the user hits the dropdown button, the components state “isOpen” alternates from its previous state, and can continuously.

On the frontend, 52 unit tests are complete and passing.

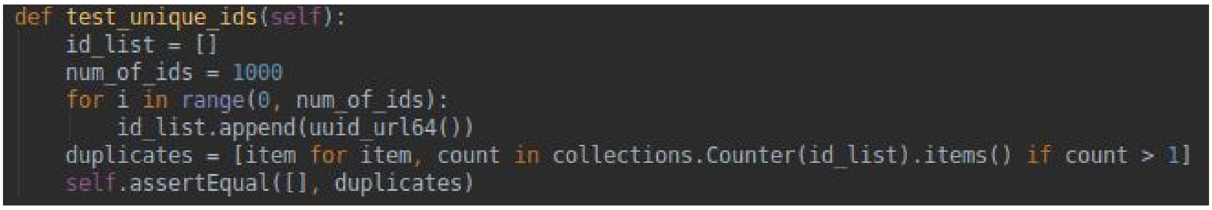


```
Test Suites: 17 passed, 17 total
Tests:       52 passed, 52 total
Snapshots:   0 total
Time:        8.724s
Ran all test suites.
```

Flask

When unit testing the backend I used the Unittest unit testing framework as it was very easy to implement alongside Flask. Unittest was inspired by JUnit and has a similar flavor as major unit testing frameworks in other languages.
The basic layout of a test is a test suit which consists of test cases, test cases which are the unit tests and a test runner which orchestrates the execution of tests and provides the outcome.

The unit test below simple test to check that the function for creating unique strings does so.



```python
def test_unique_ids(self):
    id_list = []
    num_of_ids = 1000
    for i in range(0, num_of_ids):
        id_list.append(uuid_url64())
    duplicates = [item for item, count in collections.Counter(id_list).items() if count > 1]
    self.assertEqual([], duplicates)
```

A list with a thousand strings is created by a for loop and the function I created. A list comprehension is then used to check if there is an element in the list repeat, if so the test will false.

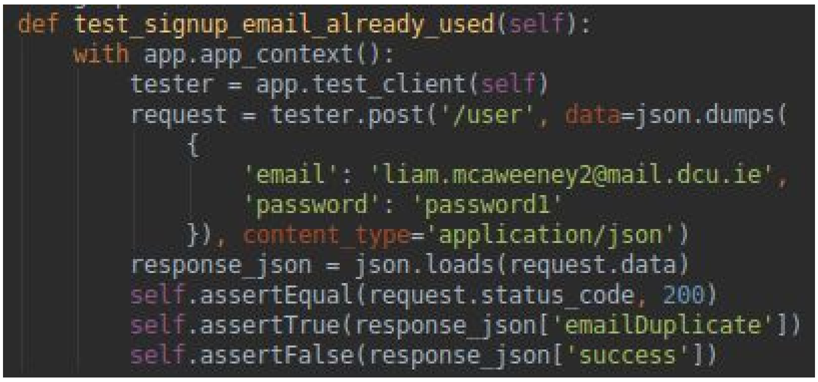
Refactoring

To perform refactoring on code it is essential to have the all test passing for the component you want to refactor. Once this was the case, I was able to evaluate the code for smells. This is code that violates fundamental design principle. The main code smells I found were duplicated code, long list of parameters. I moved duplication of code by extracting the code into a function and calling it in its place. The long list of parameters was corrected by making an object which held them and just passed it as a single parameter. This is known as a DTO (Data Transfer Object). This process wouldn't of been possible without unit testings. Unit testing provides more that prove of functionality but is the basis of all maintenance of a project. Completing this project has given me a better appreciation for testing.

Integration Testing

Integration testing is testing if components of a system interact as expect. This may be two functions or connection between different systems in the application, ie. backend to database and frontend to backend.

An example integration test is testing when the backend receives a post request containing a user's information and how that interacts with the database to form a response. The snippet below is testing just that. Using unittest I mock a client sending a post request to the backend containing an email address and a password. The email address is already registered in the database. The test should return stating that it is already in use and that therefore it is a duplicated email error.



```python
def test_signup_email_already_used(self):
    with app.app_context():
        tester = app.test_client(self)
        request = tester.post('/user', data=json.dumps(
            {
                'email': 'liam.mcaweeney2@mail.dcu.ie',
                'password': 'password1'
            }), content_type='application/json')
        response_json = json.loads(request.data)
        self.assertEqual(request.status_code, 200)
        self.assertTrue(response_json['emailDuplicate'])
        self.assertFalse(response_json['success'])
```

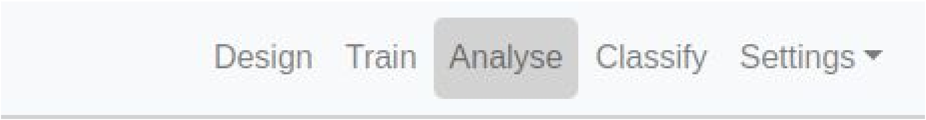
When originally developing the backend for the application, when I created endpoints I tested them using Postman which is a ready easy to use client side application to test them. This led to the use of postman for initial integration testing before finally integrating them with the frontend.

Usability Testing

When initially designing the application,I always had the end user in mind. From intra I learnt that this approach helps to reduce risk. This is why I used Shneidnerman's golden rules to steer my design and ensure user satisfaction. Below I will name some rule and how I have tried to achieve them. Consistency is an important aspect of the application as it helps the users achieve their goals efficiently

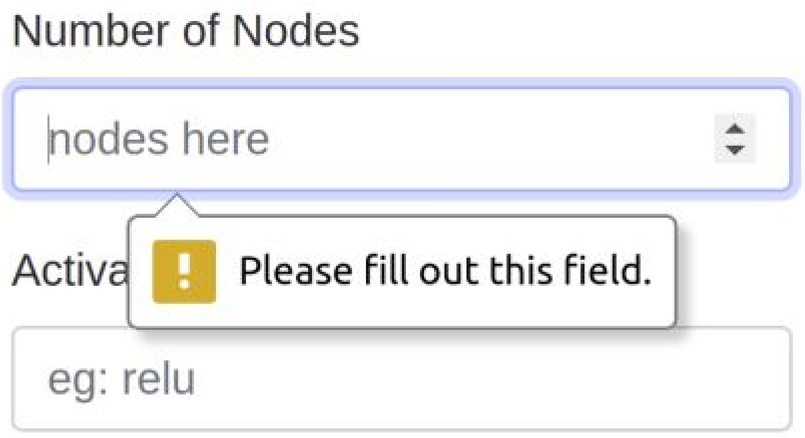
Strive for consistency:
The design of the system is consistent in layout and in application. The fonts, colors, buttons, navbar, sidebar etc. are all consistent. This ensures the application looks professional and improves user flow.

Offer informative feedback:
It is important to inform the user of what state the application is in. An example would be that the section the user is in will be highlights in the navbar. Also when the system is preparing to fit a model a loading animation is output to let the user know that the system is functional and not just not-responsive.



Design dialog to yield closure:
The application is design to fit the journey taken when creating a model. This creates a flow through the application and also provides the user with a sense of accomplishment when they complete a classification and reach the last of the sense functionalities in terms of their journey.The image above describes the journey.

Offer simple error handling:
When a user inputs into the system, if there is an error a very simple and clear notification is prompted to the user. It is light in color and fades in to prevent a sense of hostility.



Number of Nodes

nodes here

Activa

Please fill out this field.

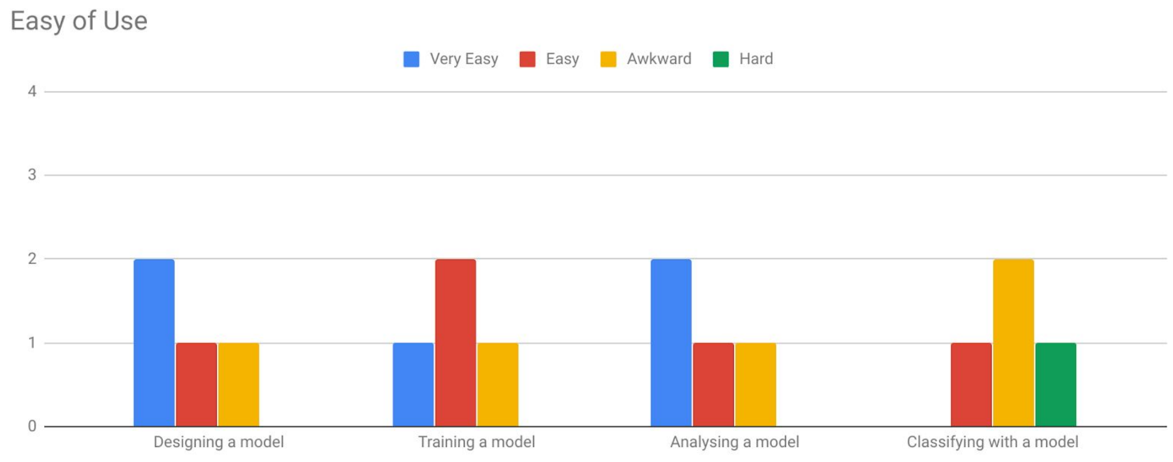
eg: relu

Support internal locus of control:
User who are familiar to a system may not want to follow the systems path but to initiate their own. This is typical of web users and must be accommodated to retain them. When designing this system I segmented each step in the creation of a model. This allows for the user to deverge on their own path and pick up at different phases of the model's creation. This also facilitates the repeating the same step to improve learning outcomes.

User Evaluation

As this application is a tool for learning, it is vital that is user-friendly and intuitive to use. To ensure this I perform two rounds of user testing on small groups. The first round was completed on the 19th of March and was extremely beneficial in refining the final product. The second round was complete on the 16th of April, this round was to evaluate the changes made based on the first round. Each round of evaluation was completed by three different students. The evaluation was a simple survey where the user rated the system in terms easy of use and added any other comments they wanted to. The surveys was completed anonymously.

Survey 1



From analysing this feedback above I was able to concentrate on what the end user wanted to improve not me. This help me prioritize the last few weeks of development to ensure the users feedback used fully.

Comments:

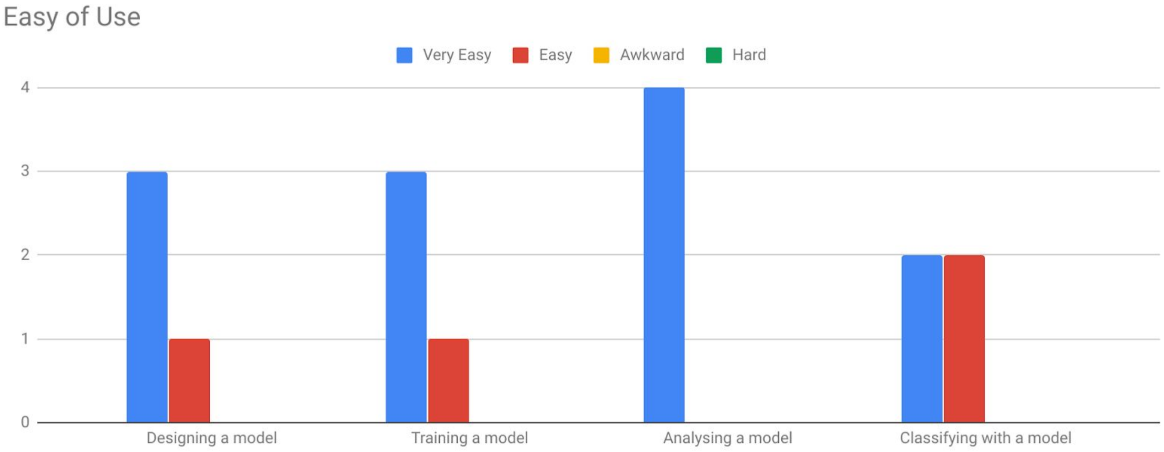
- ➔ It would be nice to able to distinguish the different types of layers in the design.
- ➔ Classifying a number could be clearer.
- ➔ It is hard to remember the exact model I designed when analysing the training

The first comment was an issue I was award of but was pushing of fix, this comment help me prioritise the issue. I changed the representation of each type of layer to clarify the differences.

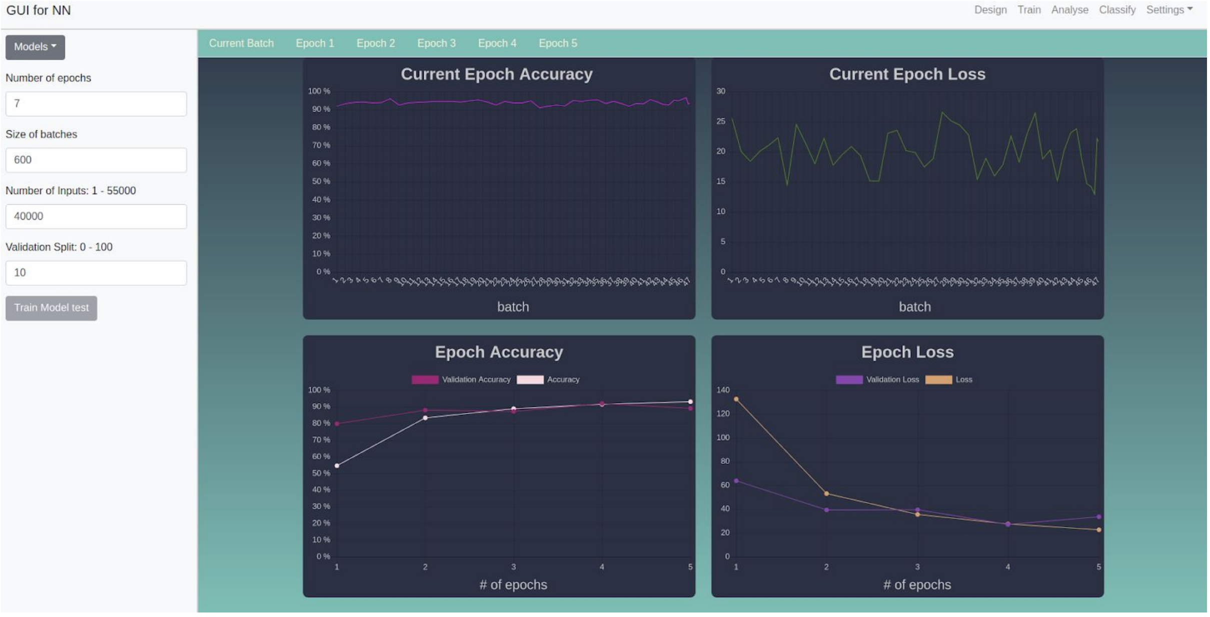
The second comment was very difficult to process as I was not sure what I could do. I took a step back and realised what the comment really meant. The issue was that when someone wanted to classify a number they had to select that is was a number they wanted to classify not a colour. This shouldn't have been an option as a classifier could only classify one or the other.

The last comment was an issue I never thought of but was clearly valid. I overcome this by displaying a summary of the design when a user selects a model, they could then select another one if they pleased.

Survey 2



After I completed the corrections uncovered from the initial feedback, I asked fellow members of my class to evaluate the application once again. As you can see the feedback is a substantial improvement from the initial evaluation. I was most pleased with the appreciation of the analysis section. I understand that I need to improve the implementation of the classification functionality. I believe this will become easier as I implement other datasets for classification. When I asked user's what their favourite functionality was, the majority said it was the real time analysis of the training. They found it informative and beneficial.



GUI for NN

Models

Number of epochs: 7

Size of batches: 600

Number of Inputs: 1 - 55000: 40000

Validation Split: 0 - 100: 10

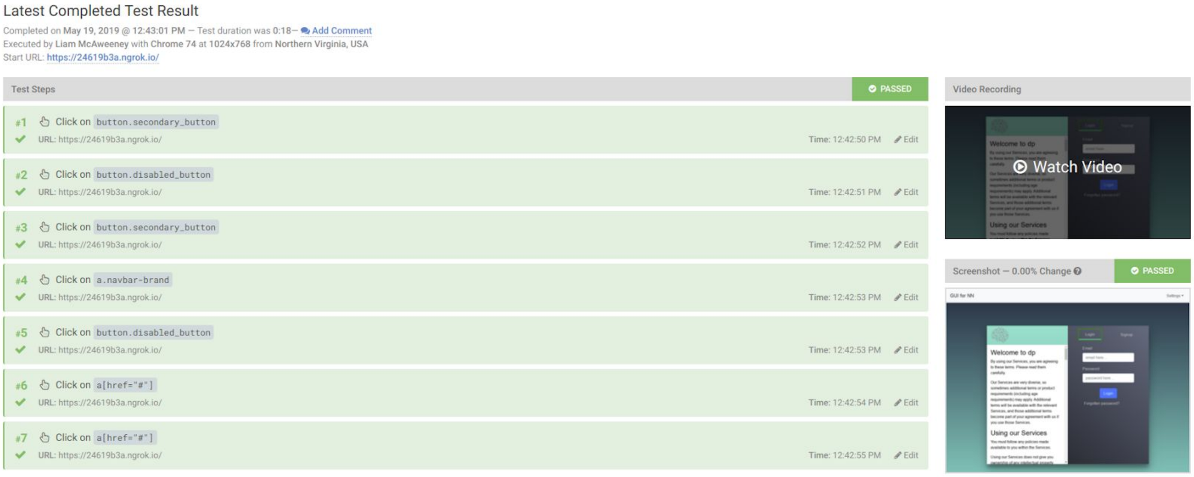
Train Model test

Final comments included:

- Other datasets would be cool.
- Maybe an explanation of each sections would be handy to have.

UI Testing

For UI testing I used a Chrome extension called Ghost Inspector which is a web browser monitoring tool that allows you to record UI actions your the browser. These recordings are then used to UI tests. It works by tracking css selectors and screenshotting the UI for visual differences. I used this as regression testing of my UI when I added a new feature. You can schedule tests to perform on a regular basis. This tool proved extremely useful when altering the backend, to check if it affected the frontend of the application.



Latest Completed Test Result

Completed on May 19, 2019 @ 12:43:01 PM — Test duration was 0:18— Add Comment
Executed by Liam McAweeney with Chrome 74 at 1024x768 from Northern Virginia, USA
Start URL: https://24619b3a.ngrok.io/

Test Steps — PASSED

#1 Click on `button.secondary_button`
✓ URL: https://24619b3a.ngrok.io/ Time: 12:42:50 PM Edit

#2 Click on `button.disabled_button`
✓ URL: https://24619b3a.ngrok.io/ Time: 12:42:51 PM Edit

#3 Click on `button.secondary_button`
✓ URL: https://24619b3a.ngrok.io/ Time: 12:42:52 PM Edit

#4 Click on `a.navbar-brand`
✓ URL: https://24619b3a.ngrok.io/ Time: 12:42:53 PM Edit

#5 Click on `button.disabled_button`
✓ URL: https://24619b3a.ngrok.io/ Time: 12:42:53 PM Edit

#6 Click on `a[href="#"]`
✓ URL: https://24619b3a.ngrok.io/ Time: 12:42:54 PM Edit

#7 Click on `a[href="#"]`
✓ URL: https://24619b3a.ngrok.io/ Time: 12:42:55 PM Edit

Above is an example of an execution of test checking the login functionality.

Problems & Resolutions

Learning a new language:
I decided to use React for developing the frontend which I had never previously used. React was the perfect framework to use for developing this application so the learning curve involved would be worth, not just in terms of this project but also to provide me with a more well-rounded skillset.
Resolution:
This was resolved through the undertaking of a Udemy course, as well as various Youtube tutorials and following blogs on medium.com.

Working with TensorFlow:
I decided on this application not because I have experience or an in depth knowledge in neural networks but quite the opposite. I have never worked with neural networks before but I wanted to use this opportunity to further my skillset and provide the same opportunities to others.
Resolution:
This involved hours of online tutorials on tensorflow and again blogs from hackermoon.com and medium.com.

Rendering charts in real time:
To ensure the user gained as much understanding as possible about what is actually involved in training a network, real time information was vital. This proved very difficult as i was using keras and was trying to open a websocket to relay the training from the backend.
Resolution:
When trying to solve this issue I had to take a set back and look at the bigger picture, I was confining myself to using keras. I researched keras and related libraries for options. I discover TensorFlow new JS library which has only been released 2018 for the first time. I explored this option soon to realise that it was the perfect solution, users could now train models on their own machines and classify in real time as the classification was now being completed in the browser.

Results

Overall I am extremely satisfied with the finished product of my application. I fulfilled all of the requirements that I had planned in the early stages of development, making small adaptations to them and implementing additional features along the way. My goal was to allow the user explore neural networks without having the fear of generating the code to do it. In turn this would provide the user with the confidence that they are capable of doing so and hopefully trying to implement them through code.

I have overcome a steep learning curve which was very much it. I have now created a web application fit for deployment using technology I hadn't heard of 2 years ago. The knowledge gained in terms of machine learn has definity inspired me to pursue a career in this field.

Future Work

Publication

In the near future, the plan is to deploy and maintain the project. Since the application very much as potential the hope it so gain a steady stream of users which will in turn motivate the maintenance and further growth of the system.

Open Source

This is a goal I have always had since I first started using open source projects. I hope to one day launch this as an open source project where like minded people could you help further the application and enlighten more people with the beauty that is neural networks.

Tutorials

While I currently enable the exploration of neural networks I would like to further provide users with the opportunity to gain a more in depth knowledge of the implementation through the creation of tutorials.

User's own Dataset

Currently the application is integrated with two datasets but was built in a manner so that other datasets would be easily integrated. The plan is to allow the user upload their own data which will already be preprocessed to train a model which they decided.

Appendices

https://uxplanet.org/simple-user-interface-tips-to-enhance-customer-retention-4067c93bc854
https://uxdesign.cc/simple-or-minimal-design-it-doesnt-matter-while-its-elegant-a5fd174b24ad
https://uxplanet.org/lean-and-mean-power-of-minimalism-in-ui-design-5ca37eb32ac8