

Sindre Eiklid  
Rickard Loland  
Maren Skårestuen Grindal

# Neural Network for Recognizing Features in Music

Bachelor's thesis in Programming  
Supervisor: Ali Shariq Imran  
May 2022

Sindre Eiklid  
Rickard Loland  
Maren Skårestuen Grindal

# **Neural Network for Recognizing Features in Music**

Bachelor's thesis in Programming  
Supervisor: Ali Shariq Imran  
May 2022

Norwegian University of Science and Technology  
Faculty of Information Technology and Electrical Engineering  
Department of Computer Science

# Sammendrag

Tittel:	Neural Network for Recognizing Features in Music
Dato:	19.05.2022
Deltakere:	Sindre Eiklid Rickard Loland Maren Skårestuen Grindal
Veileder:	Ali Shariq Imran
Oppdragsgiver:	EC-Play
Kontaktperson:	Joakim Skare
Nøkkelord:	Maskinlæring, Kunstig Nevralt Nettverk, API, Musikk, Webservice, Fullstack, Programmering, Scrum
Antall sider:	73
Antall vedlegg:	12
Tilgjengelighet:	Åpen

---

**Sammendrag:** Gjennom de siste to tiårene har det blitt gjort stor fremgang rundt identifisering og klassifisering av trekk og egenskaper i musikk, gjennom bruk av fremgangsmåter som neural networks og maskinlæring. Til tross for dette er feltet fremdeles under utvikling, og det er en mangel på samlede implementeringer som dekker både beat-identifisering og akkord-klassifisering. Som en del av denne avhandlingen gjøres forsøk på å bruke og kombinere eksisterende verktøy, bibliotek og modeller innen feltet for å trekke ut data rundt tempo og akkorder i musikk.

Selv om perfekt identifisering og klassifisering ikke er oppnåelig, gjøres en undersøkelse av fordeler og ulemper av klassiske algoritmer, Hidden Markov modeller og neural networks. Denne undersøkelsen har formålet å finne den beste løsningen gitt begrensningene satt ut i oppgaven. Ideer for hvordan løsningen kan forbedres utenfor slike begrensninger er også utforsket. En algoritmisk tilnærming for analyse av onset styrke ble valgt for identifisering av beats. For akkorder ble en randomisert implementasjon av grid search gjort på et neural network som fødtes med forprosessert lyddata uten vokaler. Gjennom denne tilnærmingen ble top-1 nøyaktighet på 82% oppnådd for akkord-klassifisering, klart overlegen det algoritme-baserte løsninger klarte.

# Abstract

Title:	Neural Network for Recognizing Features in Music
Date:	19.05.2022
Authors:	Sindre Eiklid Rickard Loland Maren Skårestuen Grindal
Supervisor:	Ali Shariq Imran
Employer:	EC-Play
Contact Person:	Joakim Skare
Keywords:	Machine Learning, Artificial Neural Network, API, Music, Website, Full stack, Programming, Scrum
Pages:	73
Attachments:	12
Availability:	Open

---

**Abstract:** There has been significant progress made over the past two decades regarding the identification and classification of traits and features in music using neural network and machine learning approaches. However, this field is still developing and there is a lack of unified implementations covering detection of beats and classification of chords. As part of this study, attempts are made at using and combining existing tools, libraries, and models within the field to extract data about tempo and chords from music.

Though perfect identification and classification is unreachable, an examination of the comparable benefits and drawbacks of classical algorithms, hidden Markov models and neural networks is performed. This with the end goal of achieving the best solution given initial constraints. Ideas on how to further improve on this solution sans constraints are also explored. An algorithmic approach analyzing onset strength was chosen for identifying beats. For chord classification, a randomized grid search was performed on a neural network being fed preprocessed audio data sans vocals. With this approach, a top-1 accuracy of 82% was achieved for chord identification, far superior to algorithmic solutions attempted.

# Preface

This thesis was written at the Norwegian University of Science and Technology in Gjøvik, at the Department of Computer Science. It was written by Sindre Eiklid, Maren Skårestuen Grindal and Rickard Loland.

For this work, we want to thank EC-Play for giving us the chance to work on this thesis, and especially Joakim Skare as product owner and Kristoffer Skare as technical consultant. We want to thank all the faculty members that helped answer questions and support this work. In particular we wish to thank Ali Shariq Imran for his consistent, quality guidance throughout the process.

From each of us working on this thesis, to each other, a thank-you goes out for good cooperation and work throughout the semester.

# Table of contents

---

<b>Preface</b>	<b>iii</b>
<b>Table of contents</b>	<b>iv</b>
<b>Table of appendices</b>	<b>vii</b>
<b>Figures</b>	<b>viii</b>
<b>Tables</b>	<b>xi</b>
<b>Abbreviation</b>	<b>xiii</b>
<b>Glossary</b>	<b>xiv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background	1
1.2 Subject	1
1.3 Delimitation	2
1.4 Target audience	2
1.4.1 Product	2
1.4.2 Report	2
1.5 Group background	3
1.6 Group organization	3
1.7 Thesis structure	3
<b>2 Development process</b>	<b>5</b>
2.1 Scrum	5
2.2 Version control	6
<b>3 Requirements specification</b>	<b>8</b>
3.1 Functional requirements	8
3.1.1 High-level use case descriptions	9
3.1.2 Low-level use case description	10
3.2 Product backlog	10
3.3 Domain model	11
3.4 Operational requirements	12
3.5 Security requirements	12

---

<b>4 Architecture</b>	<b>12</b>
4.1 Project structure	12
4.2 Network structure	13
<b>5 Technologies</b>	<b>15</b>
5.1 React	15
5.2 Tensorflow	15
5.3 Librosa	15
5.4 Spleeter	16
<b>6 Implementation</b>	<b>17</b>
6.1 Algorithms	17
6.1.1 Beat algorithm	17
6.1.2 Chord algorithm	23
6.2 Machine learning	32
6.2.1 Gathering training dataset	33
6.2.2 Determining model layout	37
6.2.3 Final design	41
6.3 API	44
6.3.1 Endpoints	44
6.4 Graphical User Interface	45
6.4.1 Design	45
6.4.2 Features	46
6.4.3 Components	46
6.4.4 Routing	48
<b>7 Development environment</b>	<b>50</b>
<b>8 Testing and Code Quality</b>	<b>51</b>
8.1 Workflows	51
8.2 Code review	52
8.3 User testing	52
8.3.1 Subjects	53
8.3.2 Process	53
8.4 React testing	55
<b>9 Deployment</b>	<b>58</b>
<b>10 Results and analysis</b>	<b>60</b>
10.1 Results	60
10.2 Recommendation and further work	68
10.2.1 Dataset preprocessing	68

10.2.2 Improved beat detection	68
10.2.3 Model selection	69
10.2.4 Implement pattern recognition	69
10.2.5 Web application and API requests	70
10.3 Conclusion	70
10.3.1 Product	70
10.3.2 Self-evaluation	71
10.3.3 Gantt chart	72
10.3.4 Risk assessment	72
10.3.5 Final thoughts	73
<b>11 References</b>	<b>74</b>
<b>A Project Plan</b>	<b>78</b>
<b>B Kanban Board</b>	<b>94</b>
<b>C Group Rules</b>	<b>95</b>
<b>D Standard Agreement</b>	<b>96</b>
<b>E Status Report</b>	<b>102</b>
<b>F Meeting Notices</b>	<b>104</b>
<b>G Gantt Chart</b>	<b>106</b>
<b>H Random Search Results - Raw Data</b>	<b>107</b>
<b>I User Testing</b>	<b>116</b>
<b>J Prototypes of Web Application</b>	<b>119</b>
<b>K Meeting Minutes</b>	<b>123</b>
<b>L Time Log</b>	<b>136</b>



## Table of appendices

Index	Name	Description
A	Project Plan	Initial plan document for the project
B	Kanban Board	Visualized method used to track issues and tasks done during the process
C	Group Rules	Rules for group behavior and work
D	Standard Agreement	Agreement for academic collaboration, signed by group, client and NTNU representative
E	Status Reports	Reports made concerning the status of the project
F	Meeting Notices	Notices of meetings with advisor or client
G	Gantt Chart	Final diagram of when work was performed
H	Random Search Results - Raw Data	All the random search results with each combination of parameters
I	User Testing	Data from user testing
J	Prototypes of Web Application	Diagrams and models not included in main report
K	Meeting Minutes	Short notes of what was discussed at meetings throughout the semester
L	Time Log	Logs of time spent working on different tasks throughout the thesis

## Figures

Index	Description	Page
1	Development process diagram	6
2	Use case diagram	8
3	Domain model	11
4	Project Structure	12
5	Network Structure	13
6	Waveform of “Ain’t no sunshine” (48000 sample rate) with beats plotted	17
7	Onset strength of “Ain’t no sunshine” with beats plotted	18
8	Multiple plots of onset strength of “Ain’t no sunshine” with beats plotted to compare different sample rate	19
9	Onset strength of “Ain’t no sunshine” with beats plotted, start and end of song	20
10	Onset strength of “Let Her Go” with beats plotted	20
11	Onset strength of “Ain’t no sunshine” with bass (left) and drums (right) splitted	21
12	Aggregate accuracy results for the algorithm on dataset	22
13	Pitch Class Profile for the song “Team”	24
14	Chord identification using 80% thresholding	25
15	50% thresholding	25
16	30% thresholding	25
17	Chord identification on Lorde’s “Team” using HMM	26
18	Chromagram from “I’m still here”	27
19	Original sample rate	29
20	Downsampled to 22050 Hz	29
21	Chord algorithm results for sample dataset with weighting	31

22	Aggregate accuracy results for the algorithm on dataset	32
23	JSON format for processed dataset	33
24	Chromagram with empty frames as padding	34
25	Chromagram with extended initial frame as padding	34
26	Chromagram padded through even distribution	34
27	Code snippet of <code>extendMatrix()</code> - Extends the rows evenly in a matrix, found in <code>preprocessing.py</code> under NN/API folder <sup>1</sup>	35
28	Model accuracy (left) and loss value (right) plots	41
29	Graph visualization of under and overfitting	41
30	Model summary of the convolutional layers	42
31	Model summary of the dense layers	43
32	Example of media queries	46
33	Input page components	47
34	Output page components	47
35	Status page components	48
36	Excerpt from the <code>App.js</code> file	49
37	NN/API workflow	52
38	Testing of input validation	55
39	Testing of buttons	56
40	Testing of inputs' initial state	56
41	Testing of result displaying	57
42	<code>docker-compose.yml</code>	58
43	Dockerfile for NN internal API	59

1

<https://github.com/sindre0830/Neural-Network-for-Recognizing-Features-in-Music/blob/ec5e23588e1dd3c4ef70995292552e2b0ffb05ce/NN/API/preprocessing.py#L228>

44	Basic confusion matrix (36)	60
45	Recall and accuracy similarity - small datasets marked in red	62
46	Circle of fifths, visualization (38)	62
47	Chord circuit visualization	64
48	Aggregate results from accuracy analysis. P = Precision, R = Recall, F1	65
49	Confusion matrix, x-axis = Predicted, y-axis = True	65
50	Detailed accuracy results by chord	66
51	Five worst-performing chords in medium dataset, x-axis = Predicted, y-axis = True	67
52	Time log for hours worked per week	71

Figures in project plan:

1	Project Organization Diagram	5
2	Product sketch	8
3	Gantt Chart	14

## Tables

Index	Description	Page
1	Field description of domain model	11
2	Amount of data points per label	36
3	Comparing accuracy (higher is better) and loss (lower is better) results from 10 models with dropout and 10 models without.	39
4	Comparing accuracy (higher is better) and loss (lower is better) results from 10 models with batch normalization and dropout, and 10 models with batch normalization without dropout.	40
5	Final model prediction results from a chromagram of D Major	44
6	Endpoints description	44
7	Tools	50
8	Linting checkers for each language	51
9	Feedback from client	53
10	Feedback from user testing	54
11	Evaluation metrics	61
12	Validation dataset sizes divided into three broad size categories	63
13	Formulas for finding chords for Top-2 and Top-4 accuracy in chord	64

### Tables in project plan:

1	Goals	3
2	Components	3
3	Domains	4
4	Roles	5
5	Columns in product backlog	7
6	Labels in product backlog	7
7	Risk Assessment	11

8	Risk evaluation matrix	11
9	Planned controls for handling risks	12
10	Tools	12

## Abbreviations

Abbreviation	Meaning
AI	Artificial Intelligence
API	Application Programming Interface
BPM	Beats Per Minute
CQT	Constant-Q Transform
CSS	Cascading Style Sheets
HMM	Hidden Markov Model
ML	Machine Learning
NN	(Artificial) Neural Network
STFT	Short Time Fourier Transform
UI	User Interface

## Glossary

Term	Definition
Artificial Neural Network	Computing systems inspired by the biological neural networks that constitute animal brains. An ANN is based on a collection of connected units or nodes. Simply called Neural Networks in this thesis.
Chord	An aggregate of musical pitches or notes sounded simultaneously.
Chromagram	Visualization closely related to the twelve different pitch classes.
Chromatic scale	A set of twelve pitch classes used in tonal music, with notes separated by the interval of a semitone.
Circle of fifths	A way of organizing the chromatic scale as a sequence of perfect fifths; pitches with intervals that match a frequency ratio of 3:2.
Constant-Q Transform	A mathematical transform related to Fourier transform, it transforms a data series to the frequency domain.
Fourier Transform	A mathematical transform that decomposes functions depending on space or time, into functions depending on spatial frequency or temporal frequency. Frequently used for decomposing the waveform of a musical chord into terms of the intensity of its constituent pitches.
Hidden Markov model	A stochastic model used to model pseudo-randomly changing systems, where the process is assumed to be a Markov chain with unobservable states.
Hyperparameter	A Neural Network parameter whose value is used to control the learning process. Examples include the size and shape of a neural network.
Markov chain	A stochastic, or random, model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.
Modulo operation	In computing, the modulo operation returns the remainder or signed remainder of a division, after one number is divided by another (called the modulus of the operation).
Note (Music)	A musical term with multiple meanings including pitch and pitch class; in this report, it can be assumed that it refers to pitch classes.
Octave	The interval between one musical pitch and another with double its frequency.



Onset strength	A one-dimensional onset strength envelope can be calculated as a function of time that responds to proportional increase in energy summed across approximately auditory frequency bands (1).
Overfitting	The production of an analysis which corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably.
Pitch class	A set of all pitches that are a whole number of octaves apart.
Tone (Music)	A steady periodic sound characterized by its duration, pitch, intensity (or loudness), and timbre (or quality).
Waveform	The waveform of a signal is the shape of its graph as a function of time, independent of scale or displacement factors.

# 1 Introduction

This is an introduction to our thesis. The product repository, provided by GitHub, can be found here:

<https://github.com/sindre0830/Neural-Network-for-Recognizing-Features-in-Music>.

## 1.1 Background

The company EC-Play was founded in 2014, and specializes in musical education, in particular for younger audiences. This is accomplished through a system of color-coding chords and displaying these in step with the beat. This system is also intended to be ideal for learning in groups, or as inexperienced musicians.

These color-coded chords are manually created and matched with the beat, using EC-Play's proprietary tools. However, despite the quality of these tools, manually adding each and every beat and chord to the song as well as organizing them into patterns is a tedious and lengthy process. The purpose of this project therefore is to alleviate this process somewhat through the use of machine learning and other tools. The idea is to produce a product that can automatically collect the beats in the song, find their matching chords, and if possible organize them in patterns. Then, these can be output to the user, who is able to edit them using the existing tools in a hopefully much faster process.

## 1.2 Subject

The process of identifying features in music such as notes, chords, tempo and patterns remains a challenging one to automate, as hard data such as sound frequencies and signal strength must be paired with the subjectivity of the human ear and subtle differences detected therein. In this paper, attempts to accomplish this process and implement it for practical use will be presented, using various approaches rooted in existing algorithms and libraries as well as neural network designs to find the optimal approach.

A large focus of this thesis relates to the processing and preprocessing of the available data, as many different processing techniques are made in order to attempt to isolate relevant data as much as possible. This includes dividing audio-files into beat-length chunks, performing resampling so as to test for ideal sample rates, designing uniform-padding algorithms to normalize the length of beat samples from songs at different tempo, and using libraries to extract vocals and individual instruments from songs.

When performing chord classification, a comparison is made between algorithmic approaches and neural network models which examines the potential of algorithmic solutions, and whether neural networks can outcompete them. It also uses basic neural model design principles to determine the ideal model design, as well as tuning said design using randomized grid search.

A core element of this work was the practical application of the final solution. Thus, design and coding of a web-app front-end with which to interface with the solution was also implemented and deployed using Google Firebase and Docker.

## 1.3 Delimitation

For this project, the work to be done was divided into four separate sections. First the beat recognition solution. Second, a solution for chord recognition. Third, a method of performing pattern recognition to group chords together. Fourth, a front-end user interface through a web application. In addition to these solutions, an API connecting these separate sections together with a database connection for periodic saving of data. The client made it clear that they did not necessarily expect all three recognition solutions to be completed as the workload might be too excessive, and supposed they might be worked on sequentially in the order outlined.

At the outset, the group's initial plan was to complete all four of these as can be seen in attachment A. However, as work on the beat and chord recognition progressed it became clear that attempting to also perform pattern recognition would be too ambitious. The scope of the project was adjusted to focus on the other three sections instead. This was partly due to the work involved in the pattern recognition itself, and partly due to uncertainty regarding the achievable accuracy of the chord recognition solution at that point of the project.

As such the group did not implement any pattern recognition solution. The final product includes solutions for recognizing beats and chords in music, a graphical user interface in web-app form with which to use these solutions, a database to store results, an API to control the interaction between these modules, and the deployment of these elements to a docker container service.

## 1.4 Target audience

### 1.4.1 Product

The product is intended for internal use by the client, EC-Play. As such, it has been developed in close concert with the client, in regards to input and output formatting and user interface design.

The client adds new songs to their database through a proprietary interface which allows for definition of beat timestamps, chords and patterns to be created manually. The product has as a purpose to automate the first two of these three tasks to a certain extent.

As this is used internally, ease of use and reliability are important qualities of the product to be delivered. The users will have musical expertise and can be expected to parse and understand outputs easily.

While the client's previous solution utilized the uploading of files, this product instead parses Youtube links and automatically downloads links passed in for parsing, making the process simpler and removing a step in the user pipeline.

### 1.4.2 Report

Here, the target audience is anyone who wishes to read and understand how this project was approached and solved by the group. This includes the client, thesis advisor, examiners, and any readers who stumble upon this report out of interest in music theory and machine learning.

## 1.5 Group background

The group has solid experience with the programming side of this project, with each team member working with the tasks they are most familiar with. This includes machine learning libraries and processes, frontend web development, API design and database management.

However, the group has no expertise in the field of music theory, and therefore relied heavily on help and feedback from the client when such expertise was needed, in addition to the various works cited throughout this thesis. As such, a lot of research and competence acquisition was done especially in the early phases of this project, in order to better understand and efficiently evaluate the beat and chord recognition tasks.

## 1.6 Group organization

The group broadly split work into three categories: front-end, back-end and report writing. All members of the group were naturally expected to work on the latter.

- Maren Skårestuen Grindal worked on the front-end solution, including web-development and user testing, as well as setting up the API, database, and deployment.
- Rickard Loland worked exclusively on the back-end, with solutions for beat recognition and chord recognition, and being responsible for evaluating the results and batch results from training and testing.
- Sindre Eiklid also worked exclusively on the back-end, with solutions for beat recognition and chord recognition, as well as setting up the internal NN API, and deployment.

## 1.7 Thesis structure

The thesis is divided into the following chapters:

1. **Introduction** - This chapter introduces the client and background for the thesis, as well as the task to be done and the group's approach to it.
2. **Development process** - Here, the process used to organize and work on the project is described.
3. **Requirements specification** - The requirements for the final products are defined here.
4. **Architecture** - Describes the project and network structures and designs.
5. **Technologies** - Explains key libraries and technologies used during the project.
6. **Implementation** - This chapter covers how the key challenges and tasks of the project were approached and solved.
7. **Development Environment** - Covers what tools were used to create the final product.
8. **Testing and Code Quality** - Discusses what testing procedures and suites were utilized for the project, and how quality of code was ensured and supported.
9. **Deployment** - In what form the final product is deployed and used.
10. **Results and analysis** - In this final chapter, the results of the project are discussed in detail, and ideas for further work that could be implemented for future iterations are outlined.

Additionally, the thesis has a glossary and a list of abbreviations near the top. These are intended to help understand the topics and concepts discussed. Internal references within the document are formatted like so: “chapter 6.1.3”. References to external resources are added to the reference list in chapter11 and added with brackets in the text like so: “Lorem Ipsum (1)”.

## 2 Development process

### 2.1 Scrum

For the project, a decision was made internally in the group to use an agile, scrum-based development process based on scrum development principles (2). There were multiple reasons for this. First of all, the nature of the project task as outlined by the client made the scope of the project something which would likely be re-evaluated multiple times during the development process. As mentioned in chapter 1.3, this re-evaluation did occur and resulted in a reduced scope, thus somewhat vindicating this choice. Second, the client made it clear that they were interested in close communication and input on the progress and nature of the product during development, which suited the group well. Third, the flexible nature of scrum development allowing for shorter sprints of work that could be quickly assessed and changed between team-members as needed seemed to suit a small project like this quite well. Finally, each member of the group has a lot of experience working with agile development processes from our bachelor program, making it a natural fit.

The process was organized with weekly sprint meetings on Mondays, where work done during the previous sprint was discussed and approved and plans for the following sprint were made. Code done was reviewed through pull requests between sprints, and any issues noted were brought up during the sprint as well as on the pull request itself, more on this below in chapter 2.2.

As mentioned in chapter 1.5, the group started out with very little knowledge of musical theory. Because of this, weekly meetings with the client in order to clarify issues or questions related to musical topics were held, which also allowed the client frequent input on the work being done.

## 2.2 Version control

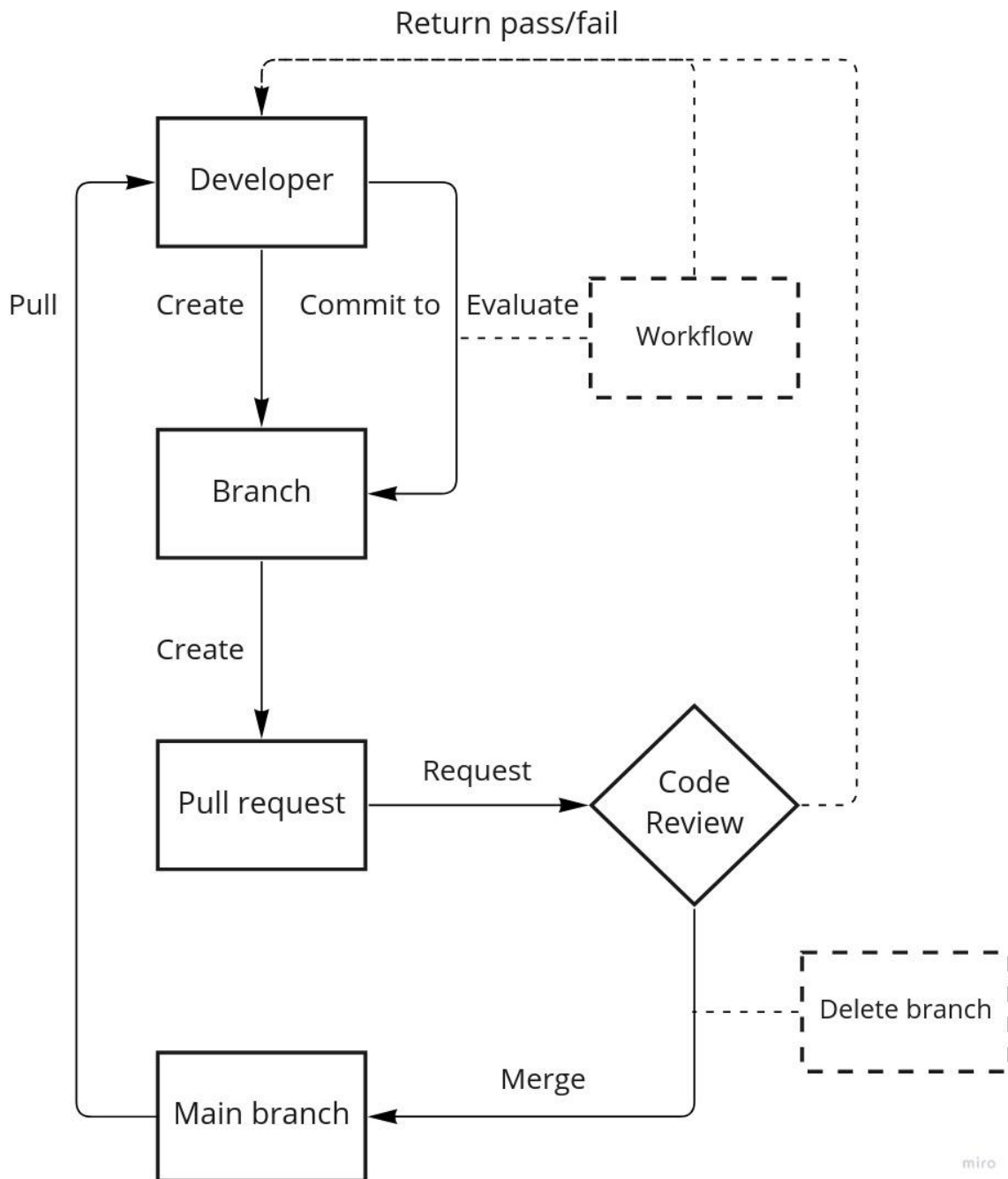


Figure 1 - Development process diagram

Figure 1 visualizes the trunk based development strategy and our QA assurance during the development process. Trunk based development strategy is based on frequent merges meaning each feature to be implemented are developed in a separate branch, then merged with the main branch once no errors or issues are found (3). This allows the main branch to be up-to-date with new features and fixes implemented quickly. When multiple developers work on the same product, an up-to-date main branch reduces the likelihood of merge conflicts which are time consuming and require multiple developers to perform.

Once we have distributed tasks during the scrum sprints, the developer creates a new branch named “*module/feature*” (i.e. *api/diagnosis-endpoint*). This makes it easy to organize all the branches that are created during the development process.

The developer then pushes atomic commits to the branch, this means that each change is pushed as separate commits. When unrelated changes are merged into a single commit, it becomes harder to recover if a bug or an issue appears. For the commit message format, we would add the related issue being worked on “[*#issue\_id*] *Brief description*”. The issues would then appear on GitHub with a hyperlink to the issue. This allowed us to give each commit a broader context without writing large commit messages. For each commit, the relevant workflow(s) would perform simple QA assurance automatically and publish results to the developer, more on this under chapter [8.1](#).

Once the developer has implemented the feature into the branch, a pull request is created. Here we write “*closes #issue\_id*” to automatically close the related issue and move it to *Done* in the kanban board once the pull request is merged. Upon creation of the pull request, all workflows are notified to run regardless of relevance. This assures that everything in the main branch will work once merged. Once all of the workflows are passed, the developer assigns another team member to perform a manual code review. A positive side effect of the trunk based development strategy is that the code reviews are short and fast to perform.

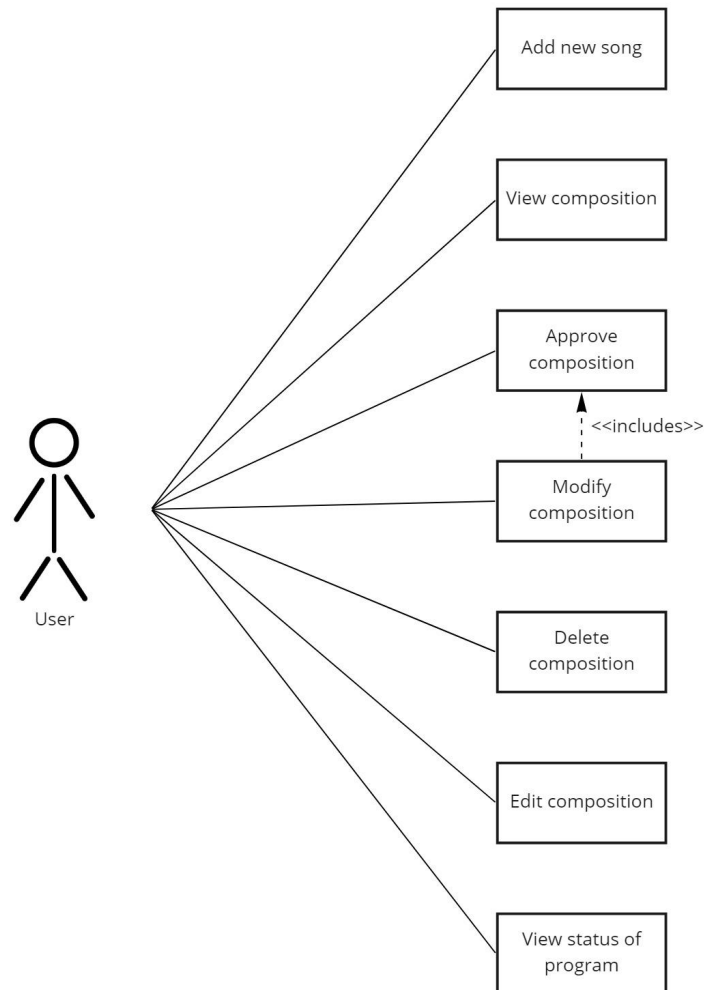
If the reviewer denies the pull request, an explanation is given on what is desired to change. If a large amount of changes are requested, we would do it over Discord or wait until the next sprint meeting. If the reviewer approves the pull request, the developer merges the branch with main and deletes the old branch, this keeps the repository organized and less cluttered. The code reviews are explained in more detail under chapter [8.2](#).



## 3 Requirements specification

### 3.1 Functional requirements

The purpose of our product is to simplify the addition of new compositions to EC-Play's database, and make the process faster and more efficient than the purely manual one they have been using so far.



miro

Figure 2 - Use case diagram

### 3.1.1 High-level use case descriptions

In this subchapter, all use cases from *figure 2* are described. Each action contains a goal and a brief description of what it does. A precondition common for all actions is the need for an internet connection.

<b>Name</b>	Input song link
<b>Actors</b>	User
<b>Goal</b>	Analyze a new song.
<b>Description</b>	The user inputs a YouTube-link they want analyzed and clicks on the submit button.

<b>Name</b>	View composition
<b>Actors</b>	User
<b>Goal</b>	View the result of an analyzed song.
<b>Description</b>	The user can click the arrow button to display a specific song's result. The arrow button can be clicked again to hide the result.

<b>Name</b>	Delete composition
<b>Actors</b>	User
<b>Goal</b>	Delete the result of an analyzed song.
<b>Description</b>	The user can click the trashcan icon to delete a song.

<b>Name</b>	Approve composition
<b>Actors</b>	User
<b>Goal</b>	Approve the results of an analyzed song.
<b>Description</b>	The user can click the approved button on a pending song to mark it as approved. When a song is approved, it is no longer possible to modify it.

<b>Name</b>	Edit composition
-------------	------------------

<b>Actors</b>	User
<b>Goal</b>	Mark the results of an analyzed song as not approved.
<b>Description</b>	The user can click the edit button on an approved song to reverse the approved mark. The result can now be modified and approved again.

### 3.1.2 Low-level use case description

In addition to the high-level descriptions, we have created one low-level description for the “modify composition” action. The reason it is only for this action, is because this is the most complicated action the user can take. It involves more steps and input validation than all of the other tasks.

<b>Name</b>	Modify composition
<b>Actors</b>	User
<b>Goal</b>	Update the results of an analyzed song.
<b>Description</b>	The user can modify a pending song. They can update some of the values, and leave others as is, or update all of them. The input from the user is validated, and they will get a message if something they have inputted is invalid. The format to input the values in, are described as text above the input field.
<b>Preconditions</b>	The song has to be marked as pending, and be opened (viewed) for the user to be able to modify it.
<b>Postconditions</b>	The user has updated and approved a song.
<b>Main flow</b>	<ol style="list-style-type: none"> <li>1 User views the song</li> <li>2 User fills in the fields they want to change</li> <li>3 User clicks the approve button</li> </ol>
<b>Exceptions to main flow</b>	<ol style="list-style-type: none"> <li>2.1 User inputs incorrect value(s) or format <ol style="list-style-type: none"> <li>2.1.1 The website tells the user what field is wrong</li> <li>2.1.2 Retry step 2</li> </ol> </li> <li>3.1 Something is wrong the the API connection <ol style="list-style-type: none"> <li>3.1.1 The website tells the user that something is wrong</li> <li>3.1.2 Updating of song is canceled</li> <li>3.1.3 Retry step 3 or alert sysadmin</li> </ol> </li> </ol>

## 3.2 Product backlog

We utilized GitHub Projects for the product backlog which allowed for smart commits, labels, and easy organization. Layout description and list of labels can be found in attachment [A](#)

and a screenshot of the kanban board during the end of the development process with a hyperlink can be found in attachment [B](#).

### 3.3 Domain model

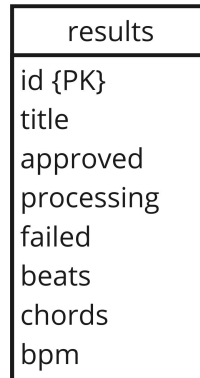


Figure 3 - Domain model

As shown in *figure 3*, the domain model is very simple, the only collection is *results* which stores information about each song inputted to the API. Firebase is used to store this database which EC-Play already has a lot of experience with. A fresh firebase project is used during the development process to avoid any security risks or other issues which could compromise the clients existing data. Table 1 describes each field's type and use case.

Field	Type	Description
id	string	YouTube video ID. This is unique and is used as the primary key in the table.
title	string	Title of the YouTube video.
approved	boolean	Whether the results have been approved by the user or not. If it is approved, the user can't edit the results anymore.
processing	boolean	Whether the youtube video is still processing. Used to show all the songs that are being analyzed on the status page.
failed	boolean	Whether the analysis failed. Will be listed on the status page.
beats	array[number]	Array of timestamps where the program found a beat.
chords	array[string]	Array of chords that the machine learning model outputted.
bpm	number	Calculated BPM.

Table 1 - Field description of domain model

## 3.4 Operational requirements

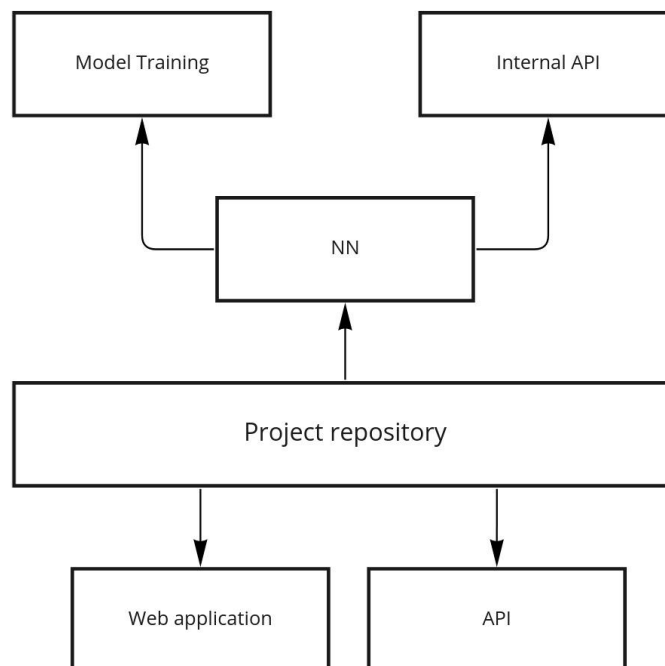
Since this product will be used In-house, the client didn't have any operational requirements. We have implemented a simple solution which will start the program again upon server restart or any crashes. This will keep the program up even with a few hiccups which should automatically fix themselves. Since the client will host this on their own servers, we are not responsible for the initial set-up, but have provided a guide on deployment provided in the README. More on deployment under chapter 9.

## 3.5 Security requirements

As defined in the project plan we are not required to safeguard against attackers, but we have taken user errors into consideration. This includes validating user input and proper error handling. Input validation is done both client- and server-side. Users can modify a song's title, BPM, beats, and chords. All of these are checked against allowed formats, including a check to see if it is a valid chord or not. YouTube links that are inputted by users are checked to be actual YouTube links before being sent to the API. Additionally, the youtube-dl library will not permit invalid links, making it an additional protection. No database changes will be performed if a field does not follow its given format.

# 4 Architecture

## 4.1 Project structure



miro

Figure 4 - Project Structure

As shown in *figure 4*, our project is divided into three separate modules. The core module is our central API, which controls the flow of data between our other two modules as well as the database and the Youtube API. The web application consists of our input, results, and status pages, making up the front-end. Finally NN encompasses all the neural network aspects of our project, including model training and an internal API.

## 4.2 Network structure

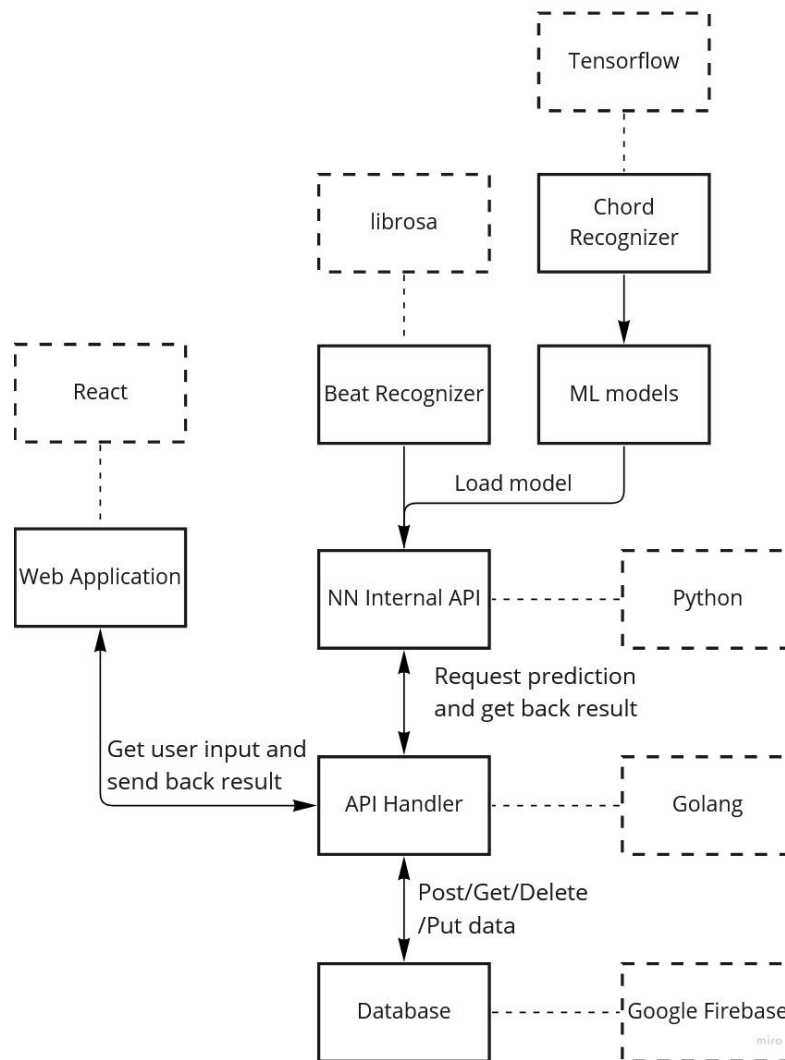


Figure 5 - Network Structure

*Figure 5* expresses how the modules communicate with each other. The entry point of the system is the web application. The YouTube link inputted here is sent to the API, which retrieves the title of the video and adds it to the database. The ID of the video is then further passed to the internal API of the NN.

Preprocessing is performed here, first filtering out vocal data and then resampling and fourier transformation are done. The result is passed to the neural network as chromagrams, each chromagram covering the duration between successive beats as the beat algorithm detects them.

At this point we perform post-processing on the resulting outputs to get the results for the song overall and represent it in the output format we want - separate lists for beat timestamps and chords with the same length that can be matched on index.

The data obtained from the NN internal API is then updated in the relevant document in the database. If any part of the process fails, the web page is notified and displays an error message to the user. This will also mark the song as *failed* in the database, and the user will be able to see it under the "Failed songs" section on the status page.

## 5 Technologies

### 5.1 React

React is a JavaScript library for building user interfaces out of different components. A component represents a segment of the website, and is reusable. Some examples could be a button or a list of names.

There are two ways to define a component in React: function components and class components. Function components are JavaScript functions, while class components are classes (4). Earlier, only class components could use state, which is a core feature of React. However, in React 16.8, hooks were introduced (5). Hooks allow the use of state in function components as well. Function components also have a simpler syntax. For these reasons, we decided to use function components in our application.

Some of React's important features:

- Reusability: breaking down the UI into components makes development more efficient as you do not have to write as much code.
- Efficiency: React only updates what is necessary for a change to happen. If only one component in an application is changed, only this will be rerendered (6).

### 5.2 Tensorflow

Tensorflow is an end-to-end open source platform for machine learning (7). It touts itself as an ecosystem for tools and other resources that facilitate the creation and design of machine learning solutions, and is a mainstay library for many mainstream machine learning and neural networking solutions. It is available for use in many different environments including JavaScript and mobile devices, although the most complete and well-known implementation is its Python API library.

Its key features include

- A large and robust library of machine learning functions.
- Simple and intuitive syntax.
- A plethora of documentation and examples.

### 5.3 Librosa

Librosa is a python package for music and audio analysis (8). This library includes functions for performing evaluation and visualization of audio-based media including beat detection, pitch shifting, feature extraction such as chromagrams and spectrograms, filtering and onset detection. It can be installed as a library through pip or similar dependency management tools, and also has significant integration with the scikit-learn python package.

Key features include:

- Built-in parsing of various audio-file extensions such as .wav, .mp3, etc...
- Integrates with matplotlib to make visualization simple.



## 5.4 Spleeter

To quote its github repository, Spleeter is a “Deezer source separation library with pretrained models written in Python using Tensorflow (...)” (9). This library, developed by the company Deezer, can perform splitting of audio sources into separate stems, and can be run in three different modes - 2-stem, 4-stem and 5-stem. All stems separate vocal performances from instrumentals and other sources of audio, with the 4- and 5-stem modes also splitting percussion, bass and optionally piano out.

Install is done as a normal Python package through pip or similar dependency management tools. The library is simple to use both through the command line and included as a library in Python code. Split audio files are saved as separate .wav files for each stem.

## 6 Implementation

This chapter describes the implementation process of the product. It starts by describing the development of beat and chord recognition from algorithmic solutions to the neural network models. Then we describe the process of developing the API and, lastly, the graphical user interface.

### 6.1 Algorithms

#### 6.1.1 Beat algorithm

While researching alternative beat detection algorithms, it was discovered that most libraries and repositories performing beat detection on GitHub and other public resources were deprecated, often relying on outdated libraries that are no longer available, or require older versions of Python. Because of this, it was difficult to perform comparative analysis between multiple algorithms, in order to find the best performing algorithm.

We ended up finding two up-to-date libraries and implementing their algorithms into our API: Aubio<sup>2</sup> and Librosa<sup>3</sup>. Aubio uses a variation of a two-state causal beat tracker algorithm while librosa follows a three-step pipeline: measure onset strength, estimate tempo from onset correlation and pick peaks in onset strength approximately consistent with estimated tempo.

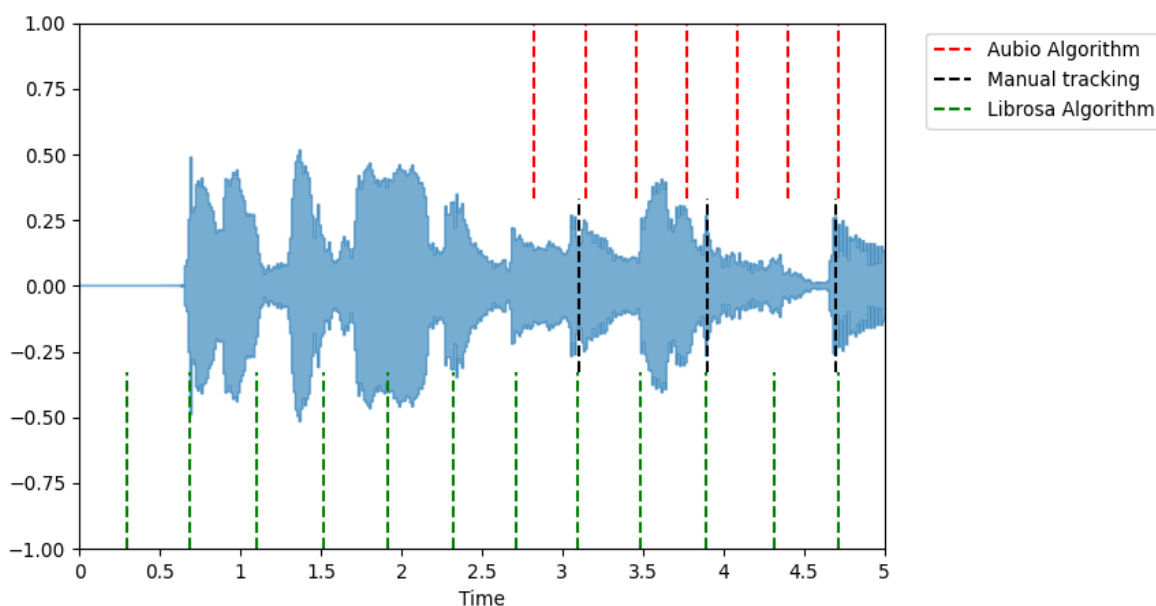


Figure 6 - Waveform of “Ain’t no sunshine” (48000 sample rate) with beats plotted

As shown in *figure 6*, the Aubio beat tracker gave inaccurate results and this was consistent throughout the development process. Librosa would end up being more accurate compared to our testing dataset provided by the client.

<sup>2</sup> <https://aubio.org/>

<sup>3</sup> <https://librosa.org/>

Once we started looking at Spleeter, an algorithm for splitting audio based on vocals and other instruments, we had problems with Aubio as they both required Numpy (a popular library in Python) at different versions. Since the Aubio beat tracking algorithm gave worse results compared to the Librosa beat tracker, we decided to remove Aubio from the API in favor of Spleeter.

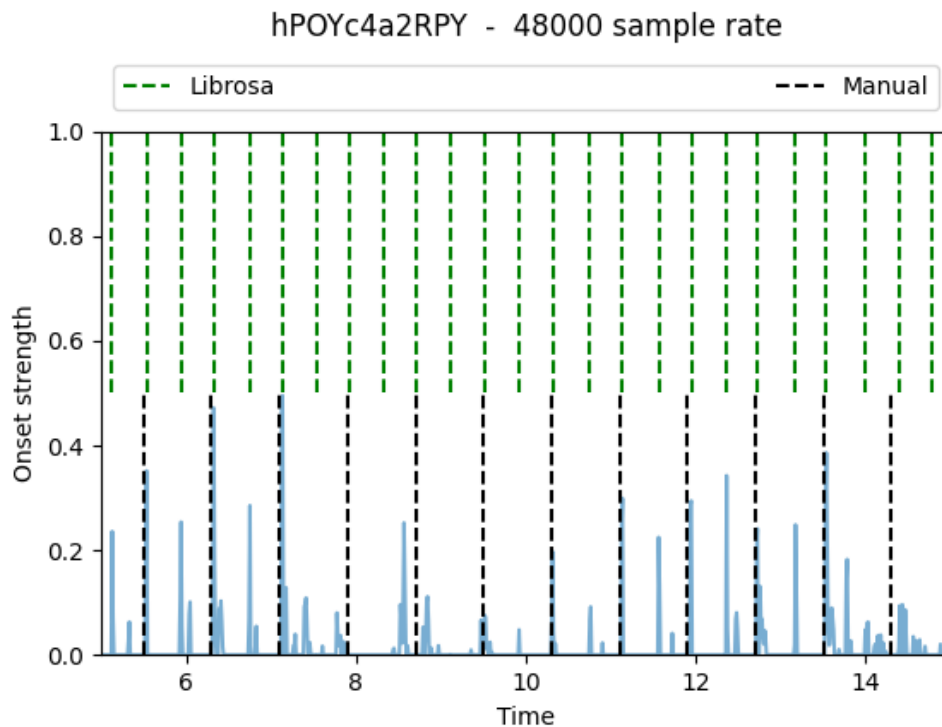
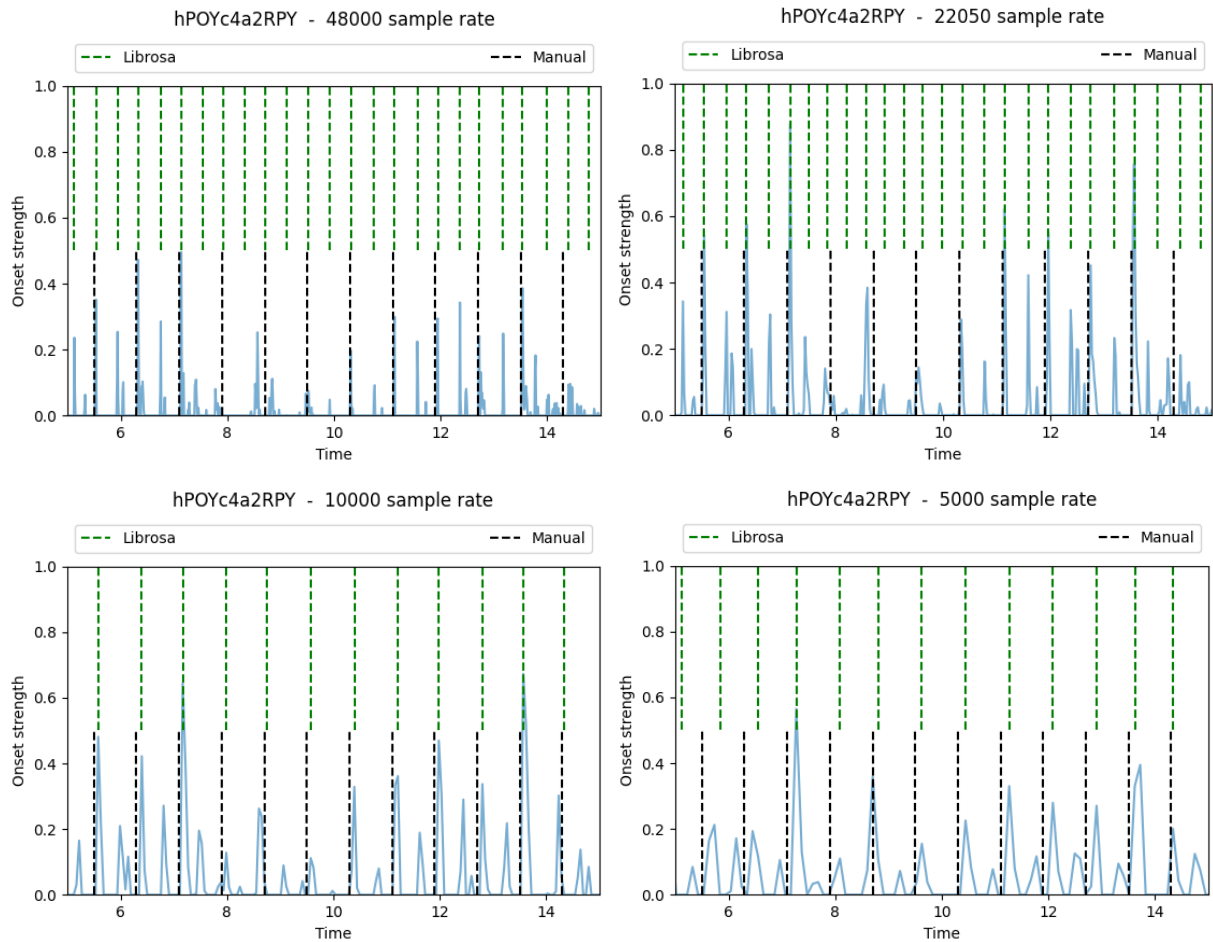


Figure 7 - Onset strength of “Ain’t no sunshine” with beats plotted

Figure 7 showcases the same results as figure 6, but with onset strength plotted in the background and with the timeframe between 5 and 15 seconds. Onset strength shows the change in pitch strength between frames and is used to find a pattern in the beat algorithm (1). While it might look like the Librosa beat tracking algorithm is wrong, it’s actually picking up the offbeat as well. An offbeat refers to something between normal beats that could be seen as a beat but usually aren’t counted when calculating the BPM. While it wasn’t a mistake by the algorithm to count it, we would prefer it without it. One solution to this problem was to downsample the audio file. This would reduce the number of details and keep only the most important pitch changes like beats.



**Figure 8 - Multiple plots of onset strength of “Ain’t no sunshine” with beats plotted to compare different sample rate**

When reducing the sample rate it was important to not lose too much information. As showcased in *figure 8*, 10,000 in sample rate would keep enough information to pick the beats while being able to filter out noise like offbeats. When reducing it to 5,000, there were audible differences and the algorithm wasn't able to properly detect where the beats occurred. This can be seen on the graph bottom right in *figure 8* where the important onset strength spikes were missing compared to the other graphs.

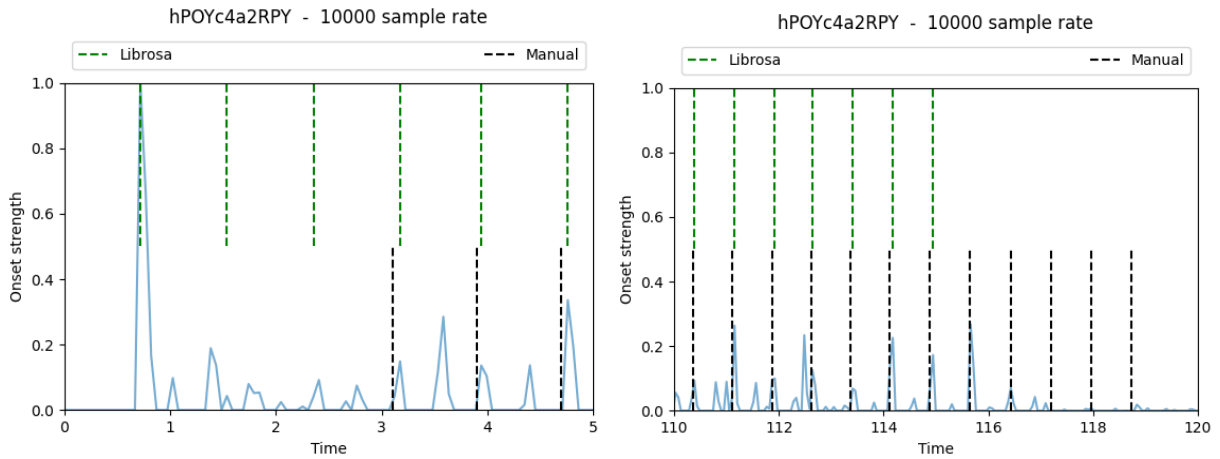


Figure 9 - Onset strength of “Ain’t no sunshine” with beats plotted, start and end of song

Figure 9 showcases the start and the end of the song and one of the current issues with the algorithm. When looking at onset strength it will often be wrong at the start of the song as seen with the initial spike. Even though the spikes between the initial spike and the first beat spike are noisy, the algorithm isn’t able to filter it out. The same issue is happening at the end of the song where it slowly fades out. Since the client wants the ability to edit the results, it will have to be fixed there. This could be fixed with a method that can pick the first and the last beat through machine learning, more on this under chapter 10.2.2.

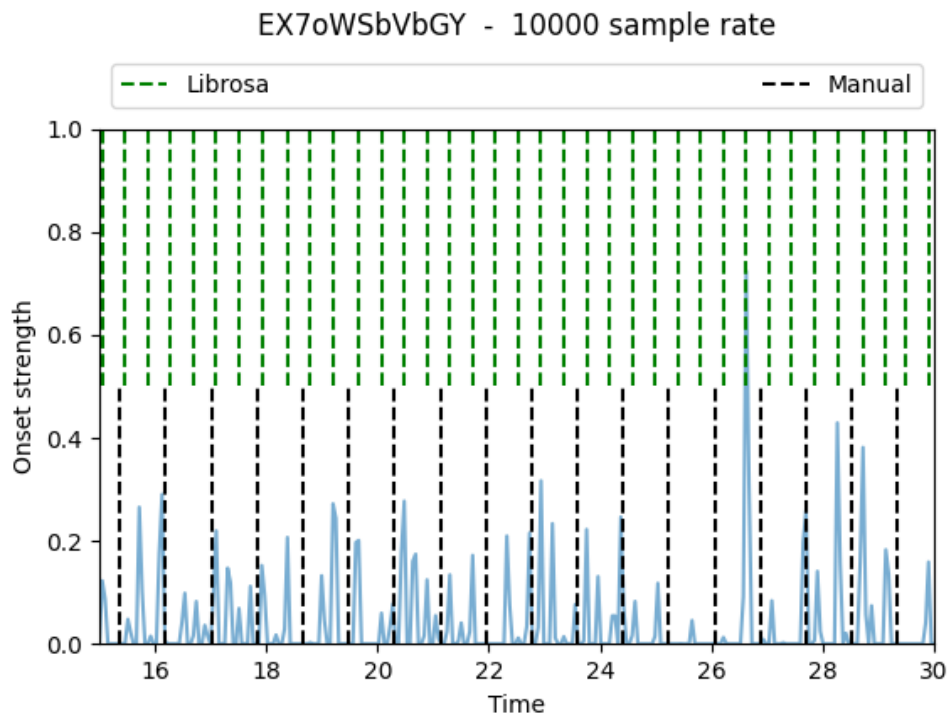
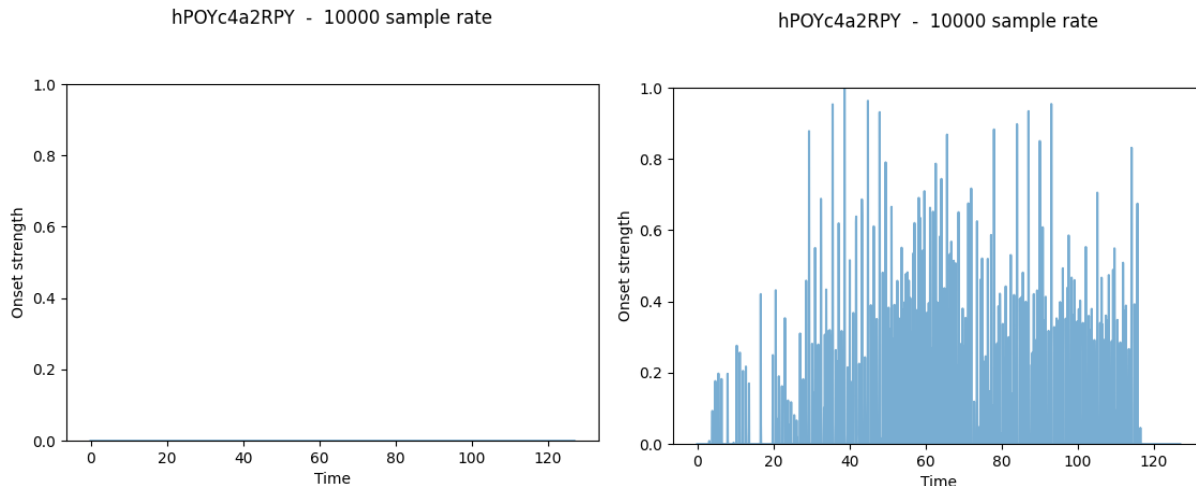


Figure 10 - Onset strength of “Let Her Go” with beats plotted

Since the Librosa beat algorithm uses BPM to find the beat spikes, songs with a varied BPM will be another issue for this algorithm. This can be seen in *figure 10* where the onset spikes are more random while the manual beat tracking has different spacing between them. The algorithm tries to find one uniform BPM which won't work for these types of songs. Songs with a varied BPM are rare and the few that might occur needs to be manually fixed.



**Figure 11 - Onset strength of “Ain’t no sunshine” with bass (left) and drums (right) splitted**

When determining the beats in a song there are a few instruments that are more relevant. For instance, both drums and bass often keep in tact with beats while piano is less important. Through the Spleeter library we are able to get just the drums and the bass, but the results were less than satisfactory and would often result in too little information for the algorithm to function. As seen in the plot to the left in *figure 11*, there wasn't enough information to create any onset strength spikes from bass alone. Drums, as seen on the plot to the right, did have enough information to generate spikes, but were inconsistent. When listening to the splitted audio files, it is very easy to pinpoint where the Spleeter program lost information when removing other instruments. It was usually around vocals, but other instruments left their dent as well. Spleeter ended up not being used in the preprocessing process for the beat algorithm, but is an important factor for the chord recognizer as explained further below.

There were several challenges when assessing the beat algorithm's performance. First of all, the manually labeled dataset that must be used as ground truth is very inconsistent in its accuracy. Songs with variable or unique tempo tend to be meticulously labeled, and match well with the onset strength of the song. Songs that have more consistent tempo are sometimes more sloppily labeled however. And all songs have the issue of being labeled by ear, rather than by evaluation of the onset strength of the song.

As such, there is consistently a small offset between the manual plot of a beat and onset strength peaks that poses a challenge in truly evaluating the beat algorithm performance. For example, there were several songs where plotting the two against the onset showed that the algorithm found beats that more closely matched with plots of onset strength. The reality is then, that even a result of 100% accuracy when compared to the

manually labeled beats cannot be relied on to actually be a perfect match with the audio. However, this fact cannot be easily accounted for when evaluating the overall performance - other songs have their manual plots closer to the onset strength, and either way the dataset is necessary to use as the ground truth for evaluating the algorithm.

In theory a new dataset made from manually evaluating the onset and matching the peaks with the dataset beats might be closer to reality and make a more reliable ground truth, but the amount of work this would take was not deemed feasible.

The next challenge was in determining what accuracy should look like for the algorithm dataset. Say for example that manually plotted beats are found at 1, 2 and 3 seconds into a song. The algorithm finds a beat at 1.95 seconds. What %accuracy is given to this observation?

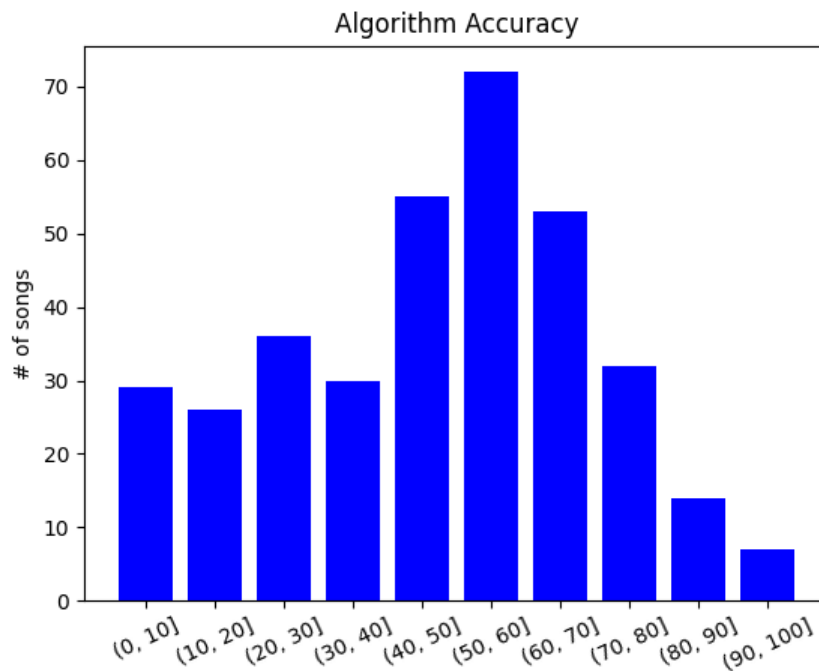


Figure 12 - **Aggregate accuracy results for the algorithm on dataset**

One solution is to just use the distance between beats as the guide - since there is conveniently 1 second down to the previous beat in this example, it is 95% accurate. But this has an obvious issue - if instead of 1.95 seconds, a beat was found at 1.15 seconds, it would be evaluated as only 15% accurate. Since it is closer to the previous beat however, it is reasonable to evaluate it compared to that beat at 1 second rather than the one at 2 seconds. If this is adjusted for, 0% accuracy could then be said to be at 0.5, 1.5, 2.5 seconds in this scenario, making the 95% accurate example instead 90% accuracy. Using this approach, it can be seen in *figure 12* above that the overall accuracy of the algorithm is fairly disappointing, most songs falling in the 40% - 70% accuracy range, with far more below that than above.

This evaluation is in some ways better, but still brings with it a host of problems. First off, it works well to evaluate songs with relatively consistent tempo, but makes the accuracy look even worse than it is for songs where the algorithm struggles to identify beats because of tempo changes. Second, because smaller differences between manual and algorithm beats are now amplified, the previously mentioned issue where the manual dataset tends to

not be extremely accurate compared to the onset strength is also made worse. For a real example, one song analyzed with the algorithm was found to be 81% accurate with the first approach, but only 40.5% after making the adjustment above.

The reason for this is because the algorithm results consistently fell ~0.15 seconds off the manually labeled beats, a fairly big divergence for in particular faster songs where the time between beats might be only about 0.6 seconds for example, and this is halved again to define the space for 0 - 100% accuracy. But looking at the onset plot of the song, peaks in frequency and offset-strength often tended to occur approximately right between the algorithm and manual labels. In other words, if we used the onset itself as the ground truth, each approach would be equally (in)accurate, at about 75 - 80% with the manual dataset over- or undershooting the mark and the algorithm doing the opposite. But because the manual labels have to be used as the ground truth, instead it is evaluated to 40%.

There is nothing we can do about this when extracting the accuracy data, but it can be kept in mind when evaluating the overall performance. Unfortunately this means that it is tough to separate what songs are being analyzed badly by librosa, and which songs are simply victims of unfortunate evaluations because of issues like this - the noise is very large. Furthermore, since offbeats detected could be considered inaccurate when evaluating the algorithm accuracy, an argument could be made that the accuracy range should be even smaller. In our example above, finding a beat at 1.5 seconds could be considered finding the offbeat and simply discarded, in which case the 0% accuracy thresholds should be at 1.25 and 1.75 seconds. However, with such a narrow window for evaluation, the inaccuracies of the manual labeling are simply amplified too much, and the analysis becomes practically useless. For the purposes of this project, the client agrees that offbeat detection is no problem.

Because of this issue where the beat dataset tends to be fairly imprecise, properly evaluating the performance of the beat algorithm solution proved to be very difficult. Similarly, due to lacking a solid dataset to use as baseline comparison, the librosa algorithm solution was used as the final beat-algorithm solution. The primary reason for this was most libraries with relevant music samples we were able to find were either no longer available, or did not have any accompanying metadata outlining beat timestamps. Because of this, we faced only one option for solving this using machine learning: using the fairly inaccurate dataset from EC-Play as our ground truth.

After spending some time considering its performance versus the algorithm when compared to onset-strength plots, we did not believe that the dataset available would result in a well-trained neural network. Therefore we elected not to attempt to make one, and focus our efforts on the chord classification instead. More on this under chapter [10.2.2](#).

### 6.1.2 Chord algorithm

Compared to beats, there are a lot more easily available options for performing chord analysis using existing algorithms. In general, the process of finding the chords in a piece of music can be divided into three steps. First is detecting the pitch classes in the audio file, which can be identified as the frequency rate of any given frame ([10](#)).

Pitch classes are created using octave equivalence – a concept describing perceived similarity of notes that are separated by whole octaves ([11](#)) – as compared to pitches, which strictly separate these types of notes. While this is not ideal for all possible musical files in terms of octave emphasis and position, Purwins ([12](#), p.48) argues that octave equivalence is



important in especially Western harmonies. For the purpose of analyzing popular music, simplifying analysis by looking at pitch classes rather than pitches makes a lot of sense.

Once pitch classes are identified, chroma features can be extracted. This is done by extracting the feature vector from the audio file using short-term fourier transform (STFT), Constant-Q Transform (CQT) or similar.

With this, we can identify the pitch classes in each audio frame for a given file. Now, all we need to do is identify what chords are most likely being played based on the strength of each pitch class. Multiple chords can be present in any given frame, but accurately identifying more than one is a daunting challenge.

The basic process of performing chord detection is to define what pitch classes make up each chord, and from there matching the pitch class data for a frame of audio to the chords. Through this process we find which chords potentially match the data. This is a process known as Pitch Class Profiling (PCP), a process introduced in 1999 which according to Helmholtz et al (13) “has long been the prime feature on which chord detection algorithms are based.” This basic approach to pitch class profiling has been iterated on since then, accounting for different frequency representations such as STFT or CQT, as well as being used as basis for machine learning and deep learning models (14).

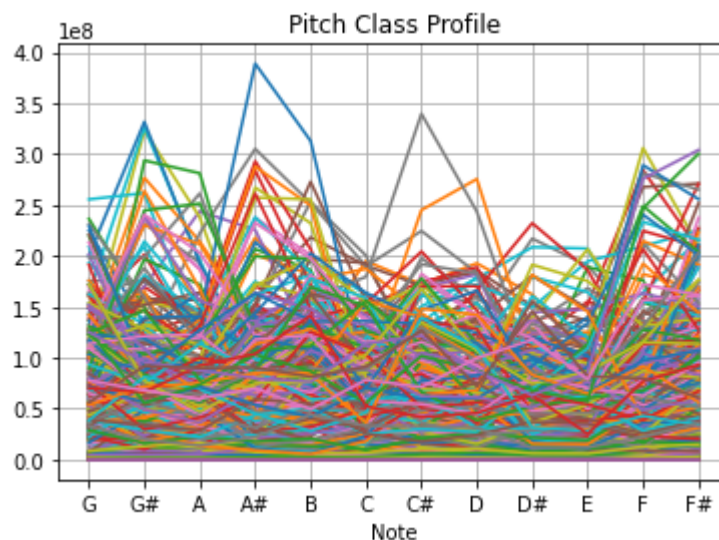


Figure 13 - Pitch Class Profile for the song “Team”

A basic application of the pitch class profile algorithm includes a lot of noise. The graph in *figure 13* was generated from the song “Team” by the artist Lorde which visualizes this. Each line represents one profile performed on a specific frame of the song. The y-axis represents the strength of each pitch’s signal, with 4.0 being the strongest. What we can see from this data is that the vast majority of the frames analyzed do not find any outstanding dominant notes above 2.0, which is only 50% of maximum signal strength. And barely any notes reach 3.0 or higher, at 75% strength.

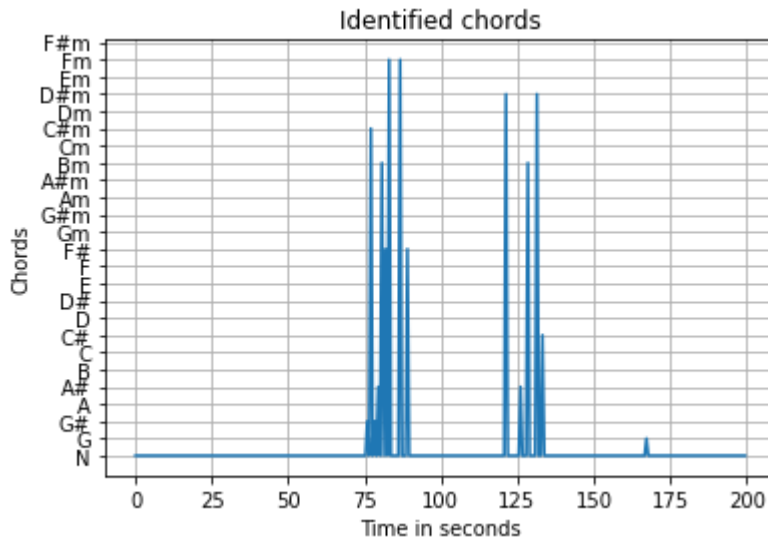


Figure 14 - Chord identification using 80% thresholding

Figure 14 is the result of a basic pitch class profile algorithm attempting to extract the chords of this song from the data using thresholding, where only keys at or above 80% of max signal strength are used. As can be seen, for a majority of the song’s duration, the algorithm is unable to detect any dominant chord or chords and returns “no chord” from its analysis. Only in a few short sections around seconds 75 – 100 and 120 – 135 as well as once at second 176 was the algorithm able to identify any chords.

When this result is compared with the Pitch Class Profile above, the reason for no chords being found is fairly obvious. As mentioned, most of the timestamps profiled have no keys managing to reach 80% signal strength, which would be around 3.2 on the y-axis of the graph. The threshold in this case is too strict to be able to identify any chords in a vast majority of frames.

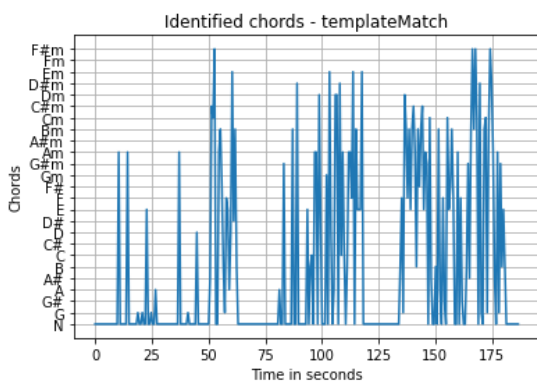


Figure 15 - 50% thresholding

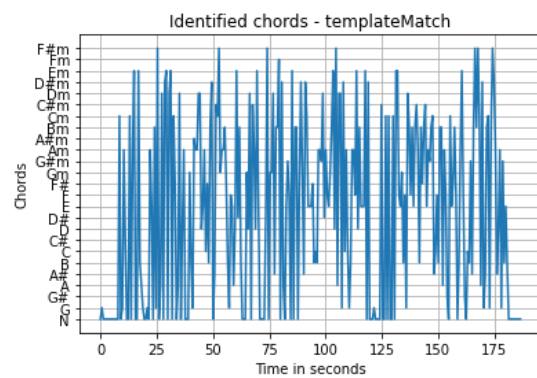


Figure 16 - 30% thresholding

By testing various different thresholds on this data, more reasonable results can be obtained. Looking at the PCP, even 50% thresholding will exclude most timestamps from identifying any chords, as no keys reach 50% signal strength. At 30%, however, a much more detailed graph with comparatively fewer instances of “no chord” is achieved.

This presents an obvious question when using PCP to find chords; what threshold is the most appropriate? What process should be used to decide on this? Are there alternative approaches besides grouping keys based on simple signal-strength thresholding that can be used when determining the likely chord of a song? This is where post-processing algorithms come in. These are used on the pitch class data before attempting to profile the chords using said data. This step can be vital for extracting actually useful data during the pitch class profiling process. Some examples of improved approaches to this algorithm includes Lee's Enhanced Pitch Class Profile (EPCP) (15), or finding chord patterns using binary intensities through use of for example Hidden Markov Model (HMM) or nearest-neighbor approaches (13).

By using a HMM approach, the Lorde song can be run through the process again, and the results can be compared to the chord identification table for the basic PCP algorithm. However, an issue here is that HMM relies on using only the previous result to inform the following ones, rather than looking at past events as a whole. HMMs also require large datasets in order to achieve good results (16). As such, with insufficient or poor data this solution will perform quite poorly. This was observed during testing, as HMM approaches performed initially quite poorly compared to other algorithms, as can be seen in figure 17. In this case, the HMM solution ended up "stuck" on classifying each chord as a D# major. Similar trends were observed with other songs attempted during algorithm testing. This is most likely due to the small datasets attempted during this part of the process, as we were testing non-machine learning solutions. In addition to this, it was noted that the HMM approach was very slow and computing intensive compared to other tested algorithms.

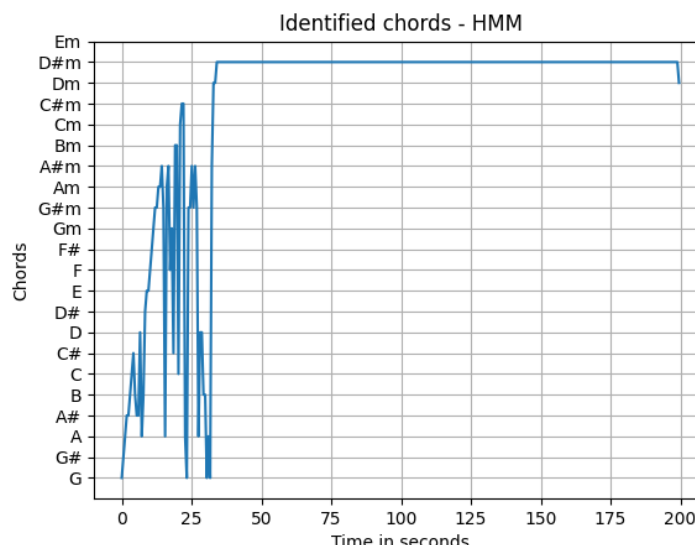


Figure 17 - Chord identification on Lorde's "Team" using HMM

As part of this process, a few different existing solutions were tried in order to find out if any performed decently well on the sample dataset, and if so, which got the best results. These included ones from the pyACA library, as well as several standalone solutions found on GitHub from users such as Das<sup>4</sup> and Navgire<sup>5</sup>.

<sup>4</sup> <https://github.com/orchidas/Chord-Recognition>

<sup>5</sup> <https://github.com/saliinavgire/Chord-recognition>

These solutions approach the issue of classifying chords in different ways, but all use the Viterbi algorithm as part of this process. This algorithm, first proposed by Andrew Viterbi in 1967, is a popular algorithm within the subjects of speech- and audio-recognition, and especially in the context of HMM (17). The Viterbi algorithm is an example of dynamic programming, where calculations are based on the results of previous calculations, thus saving time through reuse. In this way it is similar to a Markov process or chain, where the probability of an event only depends on the state attained in the previous event.

By comparing the output of multiple algorithms, we have a sturdy base with which the manually curated dataset can be compared.

During testing of the various chord-recognition algorithms, a trend where similarly bad results were reached by multiple algorithms on the same parts of a song. Post-processing of the data for better visualization confirmed this issue, as wrong or no chords were being identified for some songs, typically the more challenging ones. Initially, it was believed that this error rate might stem from difficulty with the actual profiling algorithms trying to extract the chords using the notes identified. This would make sense since most areas showing serious issues were songs considered “more difficult” to learn by the client, and thus usually had strange time signatures, many chord changes, or other such features.

However, a thorough review of the data painted a different picture: consistently when these results occurred, a look at the generated chromagrams for the song would display different notes than expected for the frames in question.

An example shown here in *figure 18*; the Sia song “I’m still here”. For the timeframe of 29 to 39 seconds into the song, multiple separate sources gave the chord progression D major, F# minor, E major, B minor. However, the pitches detected in the chromagram for this part of the song poorly matches up with these chords. Here we should probably explain the makeup of a chord: a chord consists of three notes, the 1st, 3rd and 5th notes of its scale. If the chord is minor, the 3rd is flattened, otherwise it is not (18).

The chord D major, which should start off this section of the song, consists in order of the notes D, F# and A. If we look between seconds 29.5 and 31, we can see that D and A are both present but A only for about half the expected duration when compared with D. Meanwhile, the note F# is not really present at all. Nevertheless, a D major chord could be detected here.

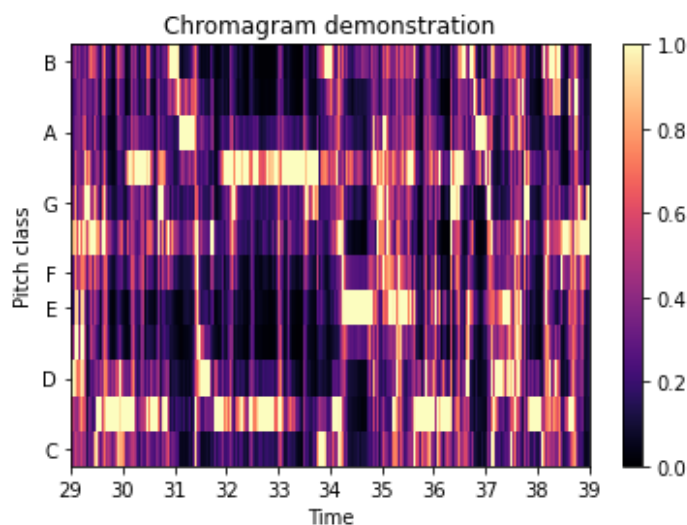


Figure 18 - Chromagram from “I’m still here”

Worse off is the next chord of F# minor, which is comprised of the notes F#, A# C#. However, for this entire section of the song, neither F nor F# can claim to be the dominant note, and indeed they rarely appear at all. It is clear that no algorithm looking at this chromagram would be able to identify the chord F# minor, regardless of algorithm used.

Next up is the chord of E major, which consists of the notes E, G# and B. Here it can be seen that E appears fairly strong from 34 to 35.5 seconds, but both G# and B are again mostly missing.

The final chord is a B minor, consisting of notes B, D, F#. In this case, B, F# and D are all present to some extent towards the end of the chromagram – but the noise is fairly big, and it is somewhat difficult to tell exactly which keys to pick out when deciding the chord.

This level of performance is not acceptable for the purposes of reliably identifying chords, and so further research had to be done. At a surface glance, it makes very little sense why the chromagram would return very different pitches from what one would expect, however in this case it is not the case that this chromagram is wrong. Rather, it is a result of noise.

The source used for chords in this song from the client references specifically the chords being played by the piano during this portion of the song – the dominant instrumental chords. However, during research it was discovered that the vocals being sung by the artist over this portion of the song uses different chords entirely. Furthermore, the vocal performance dominates the signal data and therefore is what the chromagram primarily picks up on.

This presents an obvious challenge for future development of this application. How does one identify the relevant chords through this level of noise interference? The most obvious answer here is using filters. By processing the signal data, unwanted frequencies can be discarded and analysis can be performed on the remaining data. The hope then is that this will be able to reduce the noise enough for the instrumental piano chords to come through.

However, even if this works, it runs into another problem. Simply put, a filter that discards unnecessary or unwanted signals and amplifies the signals wanted from this song could perhaps be made. But how can this solution be scaled up so that it can work with all kinds of songs from many different genres, with varying amounts of noise on different frequencies? Even if it is assumed that instrumentals are always relevant and vocals should always be discarded, the high-pitch vocals for a singer like Sia is very different from the deep vocals of e.g. Johnny Cash.

There are a variety of purported vocal filters available to test. Initial work was done using librosa's vocal separation library. However, the results were not significantly improved from the non-filtered audio – some vocal data was filtered out, but not a significant amount, and the chromagram/algorithm results were not noticeably improved. By writing the audio back to .wav files after processing is performed, manual verification of this could be performed – vocals still clearly audible after masking is performed.

Similar results were had with a few other resources such as one solution from GitHub user tsurumeso<sup>6</sup>. However, a breakthrough was had with the library Spleeter<sup>7</sup>. This library, developed by the company Deezer, is provided freely under the MIT license to use by anyone. Testing it on some troublesome songs revealed very good performance separating vocals from instrumentals. Additionally, this library also provides the option to separate

---

<sup>6</sup> <https://github.com/tsurumeso/vocal-remover>

<sup>7</sup> <https://github.com/deezer/spleeter>

instrumentals using its stem parameter. With 2 stems, only vocals are isolated and all instruments are grouped; with 5, drums, bass and piano are all separately identified and isolated as well. However, this process does mean significantly reduced quality – the isolated piano is noticeably more distorted and noisy than the full grouped instrumentals.

Using this tool, interference caused by noise from instruments or vocals playing separate chords can be minimized. Thus tests to find the best combination of instruments and vocals for identifying chords in the song can be performed. Unfortunately, there is no easy way to set up one “best” configuration for all songs, and in the end we settled on removing vocals only. However further work could be done to also separate instrumentals. For example, if the song was split into many different versions with different numbers of instruments and vocals and then evaluated, the results could be plotted to show whether some instruments are heavily featured in the chromagram or not – the typical example here might be the percussion. In this case the percussion signals can be safely discarded when finding the chords of the song. This principle could be used to find other instruments that potentially are not contributing to the chord progression and discard them. However, this does risk dropping important but rarely contributing instruments in some cases, and the issue of significantly reduced fidelity and decreased efficiency of the library still remains. As such, implementing this was not a priority for this work.

A plot of pyACA's detected chords for the same section of Sia's song can be seen in *figures 19 and 20*, after using Spleeter to isolate the vocals. Note that the order of chords on the y-axis is not the same, making it slightly difficult to parse. The first plot is performed with the original sample rate of 48000Hz, and the second is downsampled to 22050 Hz. While neither is ideal, a much better match with the expected chords can be seen – the start is noisy, but D Major is fairly dominant. Next it detects F# major instead of f# minor - the only difference between the two is whether the supporting A note is sharp or not, so this is not too surprising. Third, it very clearly identifies the E major chord. Finally, it struggles with the B minor chord, instead again detecting a lot of D Major – the only difference between the two are the B and D notes themselves, so it is again fairly close. In this case, the non-normalized audio results are slightly better than the results for normalized, 22kHz audio.

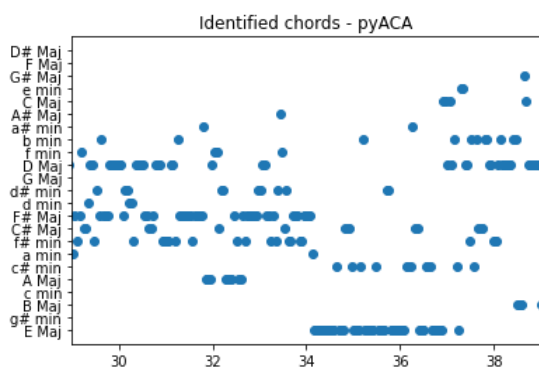


Figure 19 - Original sample rate

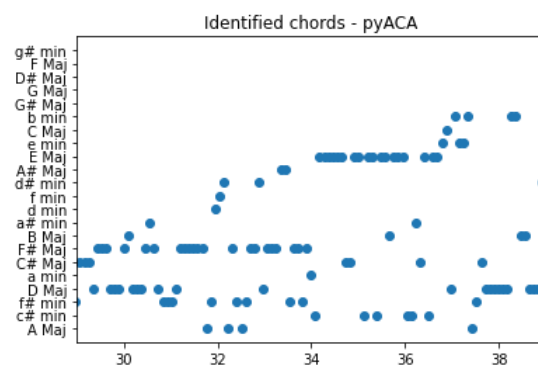


Figure 20 - Downsampled to 22050 Hz

These results are significantly better than those of the native audio files, and warranted further research. By running the dataset through Spleeter and comparing algorithm results for both the original and split dataset, it was noticed that the performance of normalized sampling rates varied wildly. Normalizing to 22kHz generally had very little effect

compared to the original sampling rate. However, further halving of the samples could do anything from amplifying the desired chords to halfway erasing them. As such, normalization to 22kHz was decided on for this project.

At this point, work turned towards processing the results. One issue with the algorithms as used is that they allowed for poor tuning of returned data. For example, it hardly makes sense to check individual frames for chords. Looking at the plotted chords of “I’m Still Here” above, dozens and dozens of chords are being detected over just a short 10-second window. When discussing this with the client, they noted that songs do not change chords nearly this often, and proposed the idea of looking at potential chord changes around beats. This formed the basis for further processing work.

Since chords often change on a beat, only looking at chords around beat timestamps makes some sense. However, this presented two notable problems. The first is one of accuracy – in order to accomplish this, good accuracy is needed on the beat timestamps. If these are significantly off, such an approach risks looking at offbeat timestamps and inaccurately determining chords.

The second is identifying what data to use. For example, one song returned pitch classes for 20 frames per second. To accomplish this goal of classifying a chord on every beat, assuming there are two beats per second, we have 10 frames – 5 on each side – “belonging” to that beat. However, does it make sense to use the data from all these frames to determine the chord of this beat? Or should only for example the 2 closest frames be used? How do we determine the answer to this?

For the first question, an evaluation of beat algorithm results was required. The first evaluation performed was comparing the difference in milliseconds between the beat timestamps returned by the chosen beat algorithm and the manually plotted dataset, with the time-difference between beats at that point of the song. For example, if the difference was 0.01 seconds and the beats were separated by 0.6 seconds, an error of  $0.01/0.6 = 0.0167$  or 1.67% would be detected. However, this is not in itself sufficient to decide the accuracy of the algorithm – after all, if the number of beats differs from the manually plotted beats, then excess beats are simply discarded and not used in the comparison. Furthermore, the discarded beats are going to be the least accurate beats detected as well.

An example here is a song which is a steady 60 BPM, but where the algorithm also detects an offbeat or otherwise mistakes the tempo as 120 BPM – not super uncommon for beat detection algorithms, though sample rate normalization can help alleviate this. In such a scenario, the onbeats would be compared and probably achieve very low margins of error – the result would look great. If the offbeats were evaluated as equivalent to the onbeats however, the result would look awful. Which comparison is most fair? In this scenario where the issue is purely detection of offbeats, it is reasonable to say the former is a fair evaluation. After discussing this with the client they agreed – but there are other scenarios where beats can be misidentified or off, in particular for variable-tempo songs.

We have previously discussed the issues with the beat dataset, which presents us a challenge in defining accuracy or error margin in this case. We did observe the error margin being noticeably lower for songs with steady tempo, and larger for songs with large tempo changes. Additionally, comparing the manual and algorithm timestamps with the onset strength plot for the song, it was observed that the algorithm often matched up better for songs which had steady tempo - reference *figure 8* in chapter 6.1.1. As such, a distinction in performance could be made where the algorithm was found more than sufficiently accurate

for these songs with predictable beat patterns, but unreliable for less predictable music – which was as expected.

For deciding what data to use, evaluating based on beat timestamps was the approach used. Looping through an array of the timestamps for the song – both manual or algorithm was tested – the current and next element was passed into a function, which isolated the part of the song between those two timestamps and performed evaluation solely on that. The result would be an array of chords, after which another algorithm picked out the most dominant chord – first by simply looking for a plurality, and if none could be found, using weighting to favor the first chords detected. This was done because after a note is played on a piano or a guitar, its strength wanes slowly over time, and loses information. Also tested was weighting later chords heavier, in case the beat-defining instruments overpower the chord-defining instruments. This was tested on a small part of the dataset, and neither approach revealed noticeable changes in results for the chord algorithm. Weighting towards early notes chords performed slightly better – though not outside the margin of error - as can be seen below.

Result rest:	Weighted towards first notes	Weighted towards last notes	Difference
Song	Manual	Manual	FirstVsLastMan
N8BXtM6onEY	59.94%	59.64%	0.3%
RkVXynWwSaI	26.39%	27.43%	-1.04%
97S0ckV9VfM	34.79%	32.5%	2.29%
YQRHrco73g4	78.62%	78.986%	-0.366%
l9RnuTQfXyg	39.57%	38.85%	0.72%
GSCC_7xMLA8	15.417%	15%	0.417%
TvbpQx-dQfQ	54.494%	53.65%	0.856%
Xy9Fc-st29U	74.24%	74.24%	0%
F6LIoTe67h4	51.47%	52.35%	-0.88%
2REgCv8tSF0	22.816%	22.573%	0.243%
Aggregate	45.775%	45.52%	0.255%

*Figure 21 - Chord algorithm results for sample dataset with weighting*

Once this was done, a new and more readable output for chord classification was in place – and could be more easily compared to the dataset directly. After evaluating the chord algorithm results vs the manually plotted chords, results were sorted by accuracy% into 10 bins, to visualize the results in more easily readable format.

As can be seen in *figure 22*, the vast majority of songs passed through the song algorithm achieved an accuracy between 40 and 80%, with smaller but still significant numbers scattered through the other bins.

It is important to note here that due to the nature of what the algorithm needs to do – correctly identify the same chord – the output from the algorithm is binary. Either it gets it correct, or it is wrong. This approach was chosen for one simple reason, namely that the algorithm is being evaluated for use in the final product, whose purpose is to reduce the manual work done to identify and update chords.

In discussions with the client, the idea of using additional accuracy metrics for neighboring chords was also considered. The reason for this is that neighboring chords are very similar (sharing 2/3 notes) and can be tough to tell apart by humans. Thus finding a neighboring chord instead of a completely unrelated chord can be considered more accurate in a way, and would be useful for evaluating the algorithm on its own terms. This could be



done by implementing an algorithm to find the neighboring chords to the chord algorithm's output chord, and seeing if any of those match the ground truth label as well.

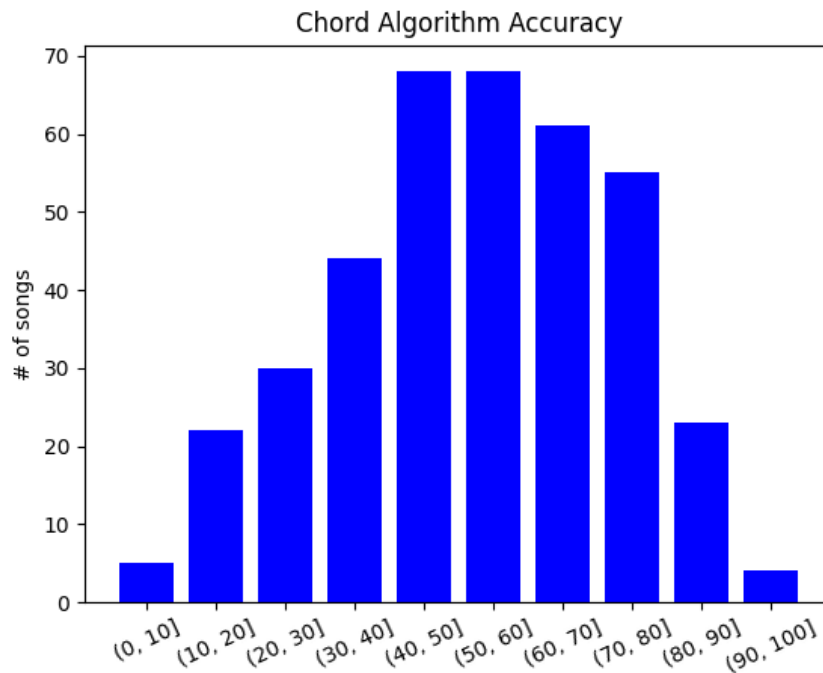


Figure 22 - Aggregate accuracy results for the algorithm on dataset

In the end we decided against implementing this, as while it would be valuable and interesting data for this thesis, it would also take time and effort away from the core elements of the project, and in particular the final product for which it would have no use. After all, whether the misidentified chord is very close to correct or a complete miss, the client will have to manually correct it using the same amount of work either way.

Going back to the accuracy then, the performance of the chord algorithm definitely leaves something to be desired. With the average accuracy between 50 and 60%, a majority of chords are technically correctly identified, but with such a large spread and many chords wrong, the manual reviewing process will still generally need to look at each chord and evaluate them individually. The exception would be patterns where chord progressions are clearly identified by the algorithm which does crop up, giving the manual review a simple template to follow for correcting the missed chords.

It is tempting then to say that such accuracy would not reduce the workload for manual review by 50 – 60% despite the accuracy, but even dummy data can be helpful as a guideline for manual editing. Thus while the chord-identification itself is not as helpful as would be preferred when using the algorithmic approach, it still has some use.

Nevertheless, the accuracy is not especially exciting, and it must be hoped that improved performance can be seen with a machine learning approach.

## 6.2 Machine learning

After creating our pipeline and evaluating the initial algorithm results, attention was turned to the machine learning portion of the work. At this point, it was already decided that machine

learning would be used to create a NN for solving chord recognition, as we did not believe the dataset was sufficiently accurate and precise to make a good NN solution for beat detection. Additionally, the beat detection algorithm performed reasonably well when compared to graphs of song onset strength - you can read more on this in chapter 6.1.1.

Thankfully where the beat-timestamp dataset was lacking, the chord dataset was comparatively very reliable. Since the chords can be evaluated on a binary “correct/not correct” scale, training a neural network to perform evaluation on this dataset would be comparatively straightforward. The main challenge here related to subjectivity in human classification as the human ear is not perfect, as well as occasionally the issue of separating out only a single chord where multiple chords were present. This second issue manifested most commonly where the instrumentals and the vocals of the song use different chords, and as mentioned in chapter 6.1.2 the Spleeter library was introduced to help solve this issue.

With this in mind, a process pipeline for creating this neural network was designed. First, more preprocessing was needed, in order to pass the data into the Neural Network correctly. Once that was complete, the design phase of the pipeline would begin, during which the actual design of the model and its parameters would be determined.

### 6.2.1 Gathering training dataset

```
"GudvNP9AwNM": {  
  "chords": [  
    "A",  
    "A",  
    ...  
    "A",  
    "A"  
  ],  
  "beats": [  
    2.85,  
    3.45,  
    ...  
    88.04999999999987,  
    88.64999999999986  
  ]  
},
```

Figure 23 - JSON format for processed dataset

The training dataset is based on the dataset of the client. This is a JSON file called songs.json and is required when running the functions to generate the training dataset. For confidential reasons, the file won't be available in the repository. As seen in *figure 23* the client dataset is processed into a simple JSON object with youtube id, chords, and beats so it can be reproduced with any dataset that provides these raw values.

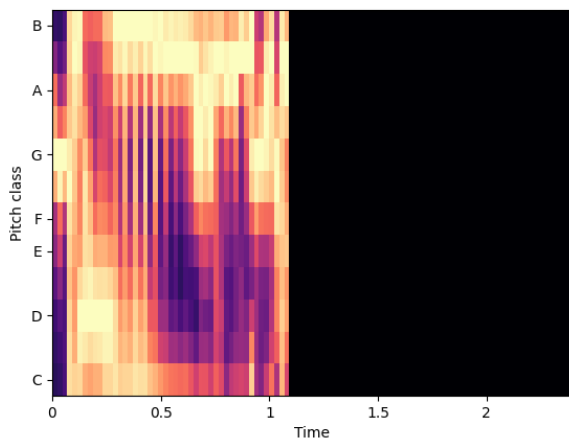


Figure 24 - Chromagram with empty frames as padding

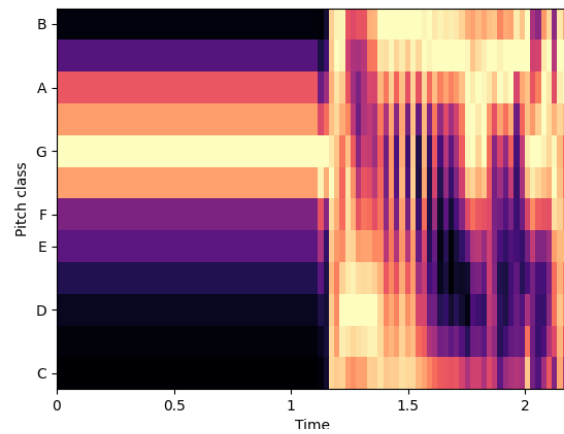


Figure 25 - Chromagram with extended initial frame as padding

Since neural networks require a uniform shape in the input data, it is crucial for the training set to be properly padded to the same size. The training dataset will consist of chromagram matrices between two beats, the x-axis (time) could be different for each matrix. The first and simplest solution is to append empty frames until we reach the size of the longest time frame. This will result in data with massive black bars as seen in *figure 24*. Since neural networks try to find comparable features in each data point, these black bars will skew the predictions towards initial length instead of the actual data gathered in the chromagram.

The next solution would be to extend the first or last frame as shown in *figure 25*. While this will be better than *figure 24*, the same issue applies where the results will be heavily dependent on the initial length of the chromagram.

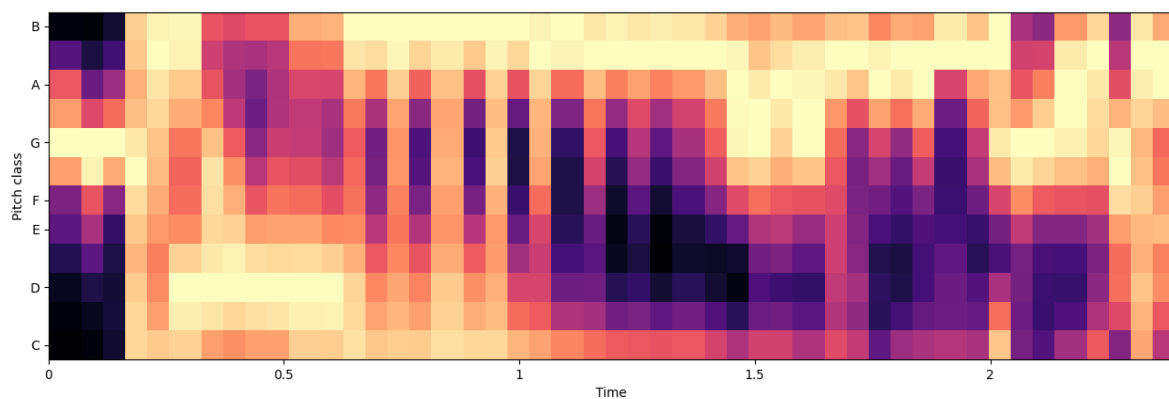


Figure 26 - Chromagram padded through even distribution

The best and final solution that ended up being used was padding through even distribution as shown in *figure 26*. This solution keeps all the information without adding anything new that might influence the predictions.

```

228 # Extend matrix on the X axis by n length evenly.
229 def extendMatrix(mat: np.ndarray, max_length: int):
230     INIT_LENGTH = mat.shape[1]
231     if INIT_LENGTH == max_length:
232         return mat
233     length = INIT_LENGTH
234     max_length -= INIT_LENGTH
235     double = max_length // INIT_LENGTH
236     rest = max_length - (double * INIT_LENGTH)
237     # double the size of the matrix
238     for i in range(double):
239         length += INIT_LENGTH
240         for j in range(length):
241             if j % (i + 2) != 0:
242                 continue
243             mat = np.insert(mat, j + i + 1, mat[:, j + i], axis=1)
244     if rest == 0:
245         return mat
246     # apply the rest evenly
247     skip = length / rest
248     total = 0
249     i = 0
250     j = 0
251     while j < length:
252         if j > total:
253             mat = np.insert(mat, i + 1, mat[:, i], axis=1)
254             total += skip
255             i += 1
256         i += 1
257         j += 1
258     return mat

```

Figure 27 - Code snippet of `extendMatrix()` - Extends the rows evenly in a matrix, found in `preprocessing.py` under `NN/API` folder<sup>8</sup>

There were no solutions available that would evenly distribute the rows in a matrix so we had to create our own function as shown in *figure 27*. First, we would calculate the amount of times the rows would need to double (i.e. amount of times each frame would be extended) as well as the remainder. To double the matrix we would simply copy a frame and insert it after the copied frame  $n$  times. It was also important to check if the index in the iteration was one of the inserted frames then skip it, this was done by a simple modulo operation checking if the index of the double loop with an addition of 2 would return a

<sup>8</sup>

<https://github.com/sindre0830/Neural-Network-for-Recognizing-Features-in-Music/blob/ec5e23588e1d3c4ef70995292552e2b0ffb05ce/NN/API/preprocessing.py#L228>

remainder or not. Once the matrix has been extended, we have to apply the remainder. Here, a skip variable has to be calculated in order to pad in the proper location; if for example the remainder is large, we will occasionally pad two frames in a row, whereas if the remainder is tiny we might skip 8 - 10 frames.

When the normalization function was finished, we created a simple function that iterates through each song in the JSON file. It downloads the song, generates and normalizes chromagrams between each beat, and appends both the matrix and the label to the dataset. In the end, it is saved as a file under the Model-training component in the project folder for model training.

Major	Amount	Minor	Amount
A	33774	A	1287
B	1525	B	10653
C	3717	C	146
D	29316	D	525
E	26777	E	2809
F	715	F	120
G	4810	G	116
A#	111	A#	69
C#	1352	C#	5701
D#	167	D#	174
F#	675	F#	23369
G#	154	G#	344

Table 2 - Amount of data points per label

We ended up with a total of 148,406 data points in the training set. As shown in *table 2*, there is a lot of inequality between the labels. This will be an issue further discussed in chapters [6.2.1](#), [6.2.3](#), [10.1](#) and [10.2.1](#).

Once the dataset is gathered, we perform `train-test-validation-split`. This splits up the database into 70% for training, 10% for testing, and 20% for validation. During the model fitting, it uses the training data to get the initial weights. The model then uses testing data to test the weights, then apply changes to improve the weights. This causes the testing data to influence the model fitting and would be a poor choice while analyzing the model after model fitting. This is where the validation data is used to get an unbiased analysis of the model. While looking for the model design, tiny changes in the model are tested to find the best layout. To reduce the difference between the models, the `train-test-validation-split` is saved to file and loaded from there so the dataset is the same for each run.

## 6.2.2 Determining model layout

Once the proper input format and size had been determined, work on designing the model could begin. First, the basic layout of the model was chosen. Since the input data would be chromagrams between two beats a 2D convolutional input layer was chosen. The data would also need to at some point be flattened, in order to be passed through the final dense output layer properly. This meant the model would always need these three layers – a Conv2D input layer, a flatten layer, and a dense output layer.

The input layer's shape must be (12, 47, 1) where the tuple corresponds to the y-, x-, and z-axis of a matrix (z-axis is used in color images for instance where the matrix is multi-dimensional). The Dense layer has a size of 25 to match the 24 potential chords with an extra output value where the model calculates the possibility of the input being none of the labels it trained on. Sigmoid activation was chosen for the output layer, as this function normalizes input values to fit between 0 and 1. This is ideal as the classification in this case should return exactly one chord as the identified chord, and nothing for the other chords. Since we only care about whether any given chord has been identified as the best fit for the chromagram input or not, the actual values of the output have themselves no intrinsic value. As such, a sigmoid function helps make the output more readable, as the normalization removes noise from "runner-up" chords and clearly defines the identified chord.

At this point, a barebones skeleton of the model has been determined, which must be iterated on and added to. This should be done in a scientific and – as much as possible – unbiased way, and some way with which to evaluate the final model's performance should be decided. For this process, a search approach evaluating many different model configurations was decided on. To evaluate the results, first a basic look at their resulting accuracy on the validation data was done. Then for the best performing models, further scrutiny was performed – more on this later.

A model search process consists of creating models with varying parameters and running the train-test-validation-split dataset through these models. Then, the accuracy and loss numbers for these models can be compared, in order to see which model-designs achieve better results.

In order to perform such a search, Grid search and Random search were considered. At first, attempts were made to use inbuilt Keras library functions<sup>9</sup> to perform these, but here problems with data types were encountered. Due to quirks in the most recent version of Tensorflow, input data had to be formatted as lists rather than Numpy arrays so it would work with the model. However, the built-in Keras library for searching automatically transforms the label data to Numpy array as a part of the process, which leads to type mismatch errors.

Instead, designing search from scratch was the chosen method. At first, a grid search approach was chosen, however this presented a large problem in terms of runtime. Even a fairly modest set of parameters being adjusted for the search quickly resulted in thousands of models, which would take upwards of a full week to run at an optimistic estimate. Because of this, the decision was made to move to a randomized grid search instead. Where a grid search methodically tests all possible combinations of search parameters, our randomized grid search will instead run a set number of models – easily defined and controlled as needed – on a random set of those possible combinations.

While not as exhaustive as regular grid search, nor having the potential for finding unexpected and unaccounted for configurations the way a true random search can, this

---

<sup>9</sup> <https://scikit-learn.org/stable/modules/classes.html#hyper-parameter-optimizers>

approach suited the needs of the project well. It balanced having a good variety of configurations with a reasonable runtime, while maintaining a sense of unpredictability and spanning many different parameter combinations.

When deciding what parameters of the model to search on, the focus was primarily on layers and nodes. For the former, having many different options for Conv2D layers and Dense layers was the primary focus, and so multiple for loops running `model.add()` a random number of times – from 0 to 2 – was the chosen approach. This way, anywhere from 0 to 6 layers of each type could be added to the model. Additionally, each loop would use a random number of filters between 16, 32, 64 or 128 and the same for units with the exemption of 16, this is because it isn't recommended to have the output nodes on the second to the last layer be less than the output layer.

Regularizer is the last parameter included in the random search. Regularization is one of many techniques to help combat overfitting and is a necessity when dealing with an uneven dataset. The possible regularizer values are 0.0001, 0.0005, 0.001, and 0.005. Anything lower or higher than these values are extreme and not useful, while we could add more between each number, the difference would be miniscule and increase the possible combinations by a lot. In the end, we ended up with a total of 116,640,000 combinations where we would pick  $n$  combinations at random to train and compare results.

Initially, variation in kernel size for Conv2D layers was also attempted. However, it was discovered that this risked significant loss of data to the point where, once dense layers were reached, the model could crash out with errors. Kernel size decides the 2D convolution window size and helps reduce the training time as it looks at all the values within the window and combines them. While this is very helpful in convolutional neural networks with input of large matrices like images, it reduces the amount of data on both the x- and y-axis where the y-axis can potentially reach  $\leq 0$  causing the program to crash. For this reason, a consistent kernel size of 1 was chosen for all randomized convolutional layers in order to avoid this issue – as well as a size of 3 for the static initial layer in order to significantly reduce the amount of nodes for the model as a whole.

For activation of each layer, rectified linear unit (or ReLU) activation was used. This simple activation method is well established for use in machine learning, because of its useful property of outputting zero for any negative input. This is, essentially, all the function does – and that is to its advantage, as this makes it fast, easy to understand and preserves as much data as possible for the model. At the same time, it has been proven to work well. While it is a piecewise linear function that cannot extrapolate in the way nonlinear functions might, it is well suited for generalization – which is the purpose of a neural network.

As Andre Ye notes, “The strength of the ReLU function lies not in itself, but in an entire army of ReLUs... when there are enough of them, they can approximate any function just as well as other activation functions...” (19). All this with superior speed and efficiency. Because of this, searching through different activation functions was decided against.

# nr	With dropout		Without dropout	
	Accuracy	Loss	Accuracy	Loss
1	81.68%	0.74	79.74%	0.94
2	81.32%	0.67	79.53%	0.88
3	81.23%	0.79	78.69%	0.73
4	80.52%	0.71	78.62%	0.94
5	80.07%	0.76	77.79%	0.83
6	79.43%	0.78	77.68%	0.78
7	78.75%	0.74	77.61%	0.76
8	78.39%	0.75	77.41%	1.19
9	75.23%	0.90	77.18%	0.87
10	74.88%	0.89	76.24%	0.96
Avg	79.15%	0.77	78.05%	0.89

**Table 3 - Comparing accuracy (higher is better) and loss (lower is better) results from 10 models with dropout and 10 models without.**

Our dataset is rather unequal with some chords being over represented, as shown in *table 2* under chapter [6.2.1](#). This can cause the model to overfit, making it biased towards some chords. Applying a dropout layer is a good way to reduce bias as it randomly ignores neurons in the layer reducing the strictness of the model (20). When deciding upon dropout, we ran the random search with and without dropout, then compared the results. As shown in *table 3*, the accuracy difference is miniscule, but the loss value is significantly lower. Using these results, it was decided to implement dropout in all of the potential layers.



# nr	Batch normalization with dropout		Batch normalization without dropout	
	Accuracy	Loss	Accuracy	Loss
1	82.60%	0.69	79.65%	1.02
2	82.30%	0.71	79.45%	1.07
3	82.28%	0.70	78.91%	1.12
4	81.36%	0.70	78.88%	1.00
5	79.57%	0.74	78.07%	1.19
6	78.62%	0.79	77.56%	0.96
7	78.56%	0.75	77.25%	1.13
8	77.79%	0.80	75.65%	1.01
9	77.42%	0.83	75.25%	1.09
10	76.75%	0.86	75.00%	1.26
Avg	79.73%	0.76	77.57%	1.08

**Table 4 - Comparing accuracy (higher is better) and loss (lower is better) results from 10 models with batch normalization and dropout, and 10 models with batch normalization without dropout.**

Batch normalization is a technique used in deep learning where the contribution in each mini-batch gets normalized. This helps reduce the amount of epochs required to reach the optimal accuracy which reduces the amount of time model training takes (21). This is especially helpful when running the random search where multiple models are trained in one session being rather time consuming. *Table 4* compared to *table 3* shows similar results except for the average loss value in *Batch normalization without dropout* in *table 4* compared to *Without dropout* in *table 3*. This could be the result of the normalization having a negative impact on the model or just an anomaly. Either way, the result with batch normalization and dropout gave similar results while being less time consuming to train so we ended up implementing it in each potential layer alongside the dropout layer.

Once the base layout was structured and the random search functionality built, the proper run could start. 70 combinations were trained and compared where the best combination of parameters were decided on.

### 6.2.3 Final design

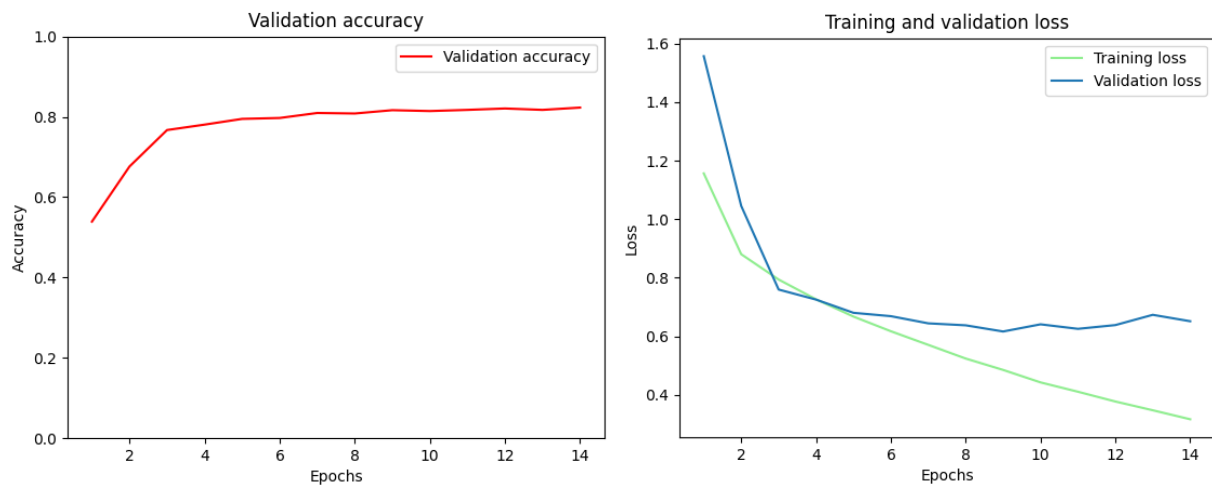


Figure 28 - Model accuracy (left) and loss value (right) plots

From the 70 combinations that were trained, the best model ended up with an accuracy of 82.64% with a loss value of 0.63 (See attachment H for the full report of the random search). As seen in *figure 28*, the accuracy grew fast the first 4 epochs, then slowed down with a small but steady increase throughout the training. While comparing *figure 28* and *figure 29*, we can see that the model is overfitting somewhat, though it is not a severe case and thus was deemed acceptable given its otherwise top results.

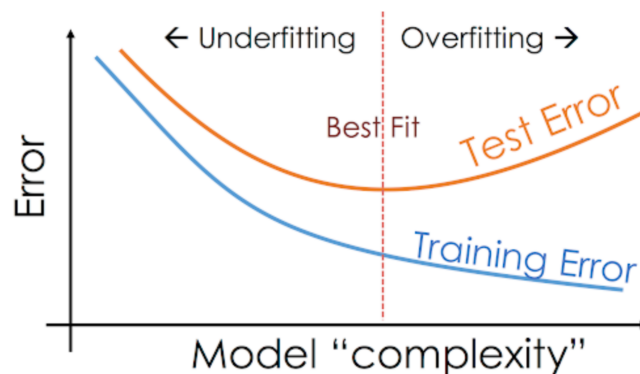


Figure 29 - Graph visualization of under and overfitting (22)

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 10, 45, 64)	640
batch_normalization (BatchNormalization)	(None, 10, 45, 64)	256
activation (Activation)	(None, 10, 45, 64)	0
dropout (Dropout)	(None, 10, 45, 64)	0
conv2d_1 (Conv2D)	(None, 10, 45, 32)	2080
batch_normalization_1 (BatchNormalization)	(None, 10, 45, 32)	128
activation_1 (Activation)	(None, 10, 45, 32)	0
dropout_1 (Dropout)	(None, 10, 45, 32)	0
conv2d_2 (Conv2D)	(None, 10, 45, 64)	2112
batch_normalization_2 (BatchNormalization)	(None, 10, 45, 64)	256
activation_2 (Activation)	(None, 10, 45, 64)	0
dropout_2 (Dropout)	(None, 10, 45, 64)	0
conv2d_3 (Conv2D)	(None, 10, 45, 64)	4160
batch_normalization_3 (BatchNormalization)	(None, 10, 45, 64)	256
activation_3 (Activation)	(None, 10, 45, 64)	0
dropout_3 (Dropout)	(None, 10, 45, 64)	0
conv2d_4 (Conv2D)	(None, 10, 45, 16)	1040
batch_normalization_4 (BatchNormalization)	(None, 10, 45, 16)	64
activation_4 (Activation)	(None, 10, 45, 16)	0
dropout_4 (Dropout)	(None, 10, 45, 16)	0
conv2d_5 (Conv2D)	(None, 10, 45, 16)	272
batch_normalization_5 (BatchNormalization)	(None, 10, 45, 16)	64
activation_5 (Activation)	(None, 10, 45, 16)	0
dropout_5 (Dropout)	(None, 10, 45, 16)	0

*Figure 30 - Model summary of the convolutional layers*

As seen in figure 30, the model consists of an initial filter of 64 with a regularizer of 0.0005. The rest of the convolutional layer filters consist of 32, 64, 64, 16, and 16 respectively.

flatten (Flatten)	(None, 7200)	0
dense (Dense)	(None, 256)	1843456
batch_normalization_6 (BatchNormalization)	(None, 256)	1024
activation_6 (Activation)	(None, 256)	0
dropout_6 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 256)	65792
batch_normalization_7 (BatchNormalization)	(None, 256)	1024
activation_7 (Activation)	(None, 256)	0
dropout_7 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 25)	6425

=====

Total params: 1,929,049  
 Trainable params: 1,927,513  
 Non-trainable params: 1,536

*Figure 31 - Model summary of the dense layers*

As shown in *figure 31*, the first layer is a flatten layer which converts the data from a matrix into a 1 dimensional array. There are only two dense layers with a unit size of 256 before the output layer. The model ended up with a total of 1,929,049 weights where 1,536 of them aren't trainable (batch normalization).

Major	Prediction	Minor	Prediction
A	98.79%	A	0.30%
B	1.65%	B	87.18%
C	0.00%	C	0.01%
D	99.99%	D	0.03%
E	25.38%	E	0.08%
F	0.17%	F	0.00%
G	97.29%	G	0.03%
A#	0.00%	A#	0.00%
C#	0.13%	C#	38.59%
D#	0.02%	D#	0.02%
F#	0.24%	F#	97.64%

G#	0.00%	G#	0.33%
None	0.01%		

*Table 5 - Final model prediction results from a chromagram of D Major*

*Table 5* shows the prediction results from a chromagram of D Major picked at random in the validation set. While the model correctly predicted D Major as the highest likelihood at 99.99%, other chords were very close. In the circle of fifths, G major, B minor, F#minor, and A major are all somewhat similar to D major in the chromagram and could easily be mistaken; more on the circle of fifths in chapter 10.1. *None* represents the output value where the model predicts the input is none of the labels it has trained on.

## 6.3 API

To connect the web application and the neural network together, an API was created. We decided to use Golang for the development, because we were already familiar with the creation of APIs with it.

### 6.3.1 Endpoints

Name	Method	Description
Analysis	POST	Analyze a song and add the result to the database.
Results	GET	Get the results of all songs that have been analyzed.
Results	DELETE	Delete the result of a song from the database.
Results	PUT	Update the result of a song in the database.
Diag	GET	Get the status of the application.

*Table 6 - Endpoints description*

As shown in *Table 6*, the API is composed of three endpoints: analysis, results, and diag. Analysis deals with sending of YouTube links and returning the result of their analysis. The results endpoint is meant for retrieving, deleting and updating the songs that are already analyzed. Diag is used for getting diagnostics of the whole application.

When the user inputs a YouTube link in the web application, a POST request is sent to the analysis endpoint in the API handler. The title of the video is then retrieved, and stored in the database along with a boolean that indicates that the song is currently being analyzed. Then, the ID of the YouTube video is passed to the NN's internal API analysis endpoint. If everything went well, the database entry is updated with the result. If something went wrong, a "Failed" boolean is added to the entry.

The web application also displays all results that are stored in the database. To retrieve them, it has to send a GET request to the results endpoint. This endpoint fetches all

songs that are not marked as “Processing” or “Failed” from the database. If the user wants to update the result of a song, a PUT request to the same endpoint is sent.

Another feature of the web application is displaying the status of the whole application. This information is retrieved from the diag endpoint. The API sends a request to the NN internal API to see if it is up and running. It also gets all songs marked as “Processing” and “Failed” from the database. All this information is then combined and sent back for the web application to display.

## 6.4 Graphical User Interface

This subchapter will describe the design and implementation of the application’s front-end module. It is developed using React and JavaScript.

### 6.4.1 Design

Before the UI was created, the design was discussed with EC-Play. They said it was up to us to decide the design, but a simple one was desirable. To make the website as accessible and easy to use as possible, we opted for a simple design that is easy to understand. We were inspired by modern and minimalist websites as these are common nowadays. The standard layout for each part of the web application consists of a navigation bar at the top, and a space underneath allocated for the content related to the page.

Responsive web design is websites’ ability to adjust to and accommodate different screen sizes (23). To improve the user experience of the application, we implemented this concept. The website works for mobile and smaller screens, as well as larger ones. Another concept that is closely related to responsive design, is the mobile first approach. As the name implies, this is based on designing for mobile devices first. Instead of adapting the desktop design to the mobile devices, the mobile design acts as a foundation that is built upon for larger devices. Some benefits to this approach:

- The focus is put on the essential content that is needed. Designs for larger screens tend to have more features and quirks to it that might transfer badly to smaller screens (24). Keeping the mobile layout focused on the core functionality is therefore desired.
- The website’s loading time is decreased because there is no need to load elements and styling that is only used for larger screens (24).

With the use of media queries in the CSS, styling elements when a certain condition is met is possible. We utilized this by adding the styling needed for the mobile design first, and then adding additional styling when the screen width is increased. An example of this is found in *Figure 32*, an excerpt from the AddSong component’s CSS file. The input field covers a set percentage of the screen’s width, and would cover an increasingly bigger portion of the screen if not for these queries limiting it.

```
42  /* small devices (tablets etc.) */
43  @media (min-width: 768px) {
44      .add-song__input input {
45          width: 50%;
46      }
47  }
48
49  /* medium devices */
50  @media (min-width: 992px) {
51      .add-song__input input {
52          width: 40%;
53      }
54  }
55
56  /* large devices */
57  @media (min-width: 1200px) {
58      .add-song__input input {
59          width: 25%;
60      }
61  }
```

Figure 32 - Example of media queries

## 6.4.2 Features

The web application is built up of three different pages: input, output, and status. The input page is where users input the YouTube links for the songs they want analyzed. Only one link can be inputted at a time. After the link has been parsed, the users get a message telling them if everything went OK or not.

The output page displays all the songs that have been analyzed. The user can then choose to manually update the results, in case there are any mistakes. Then they need to approve the song. When a song is approved, it is no longer possible to edit the result values. If the user had modified the result, this is updated in the database when the song is approved. The user can also search for and filter (approved and pending) the songs for easier navigation.

The last one is the status page. Here, the user is presented with the diagnostics of the application. All songs that are currently being processed and the ones that failed the analysis are displayed here. There is also a chapter that shows the status of the other parts of the application, for example the API.

## 6.4.3 Components

A UI in React is built up of multiple components that each serves their own purpose. This section breaks down the entire application, and goes more in depth of the components it is composed of. All components are represented with their own border color and a number. Each page has one main component that is built up from smaller ones. The Navbar component, which serves as the navigation bar, is visible throughout the whole application. This one is marked with a blue border and has the number 1. Some components are used in several places, and have the same color and number throughout all illustrations. An example

of a component that is repeated is the SongTitle component, which is represented with a dark blue color and the number 5. This component displays a title and a colored line corresponding to a specific status.

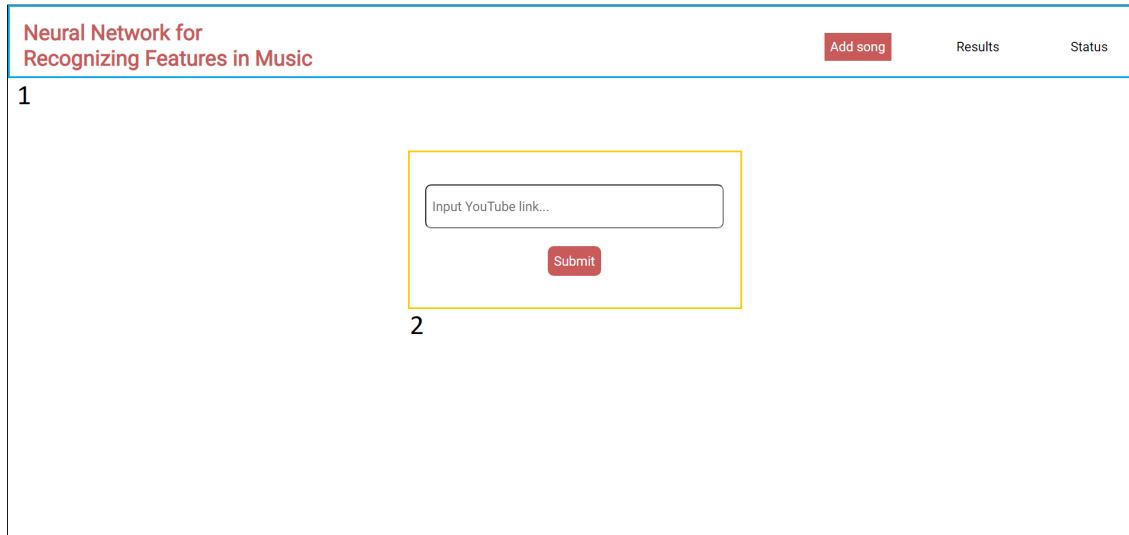


Figure 33 - Input page components

Figure 33 shows the layout of the input page which, aside from the Navbar component, only consists of the AddSong component. This component is built up of three elements: a paragraph for displaying messages to the user (not in the illustration), an input field, and a submit button. For this reason it seemed sufficient to merge them into the same component.

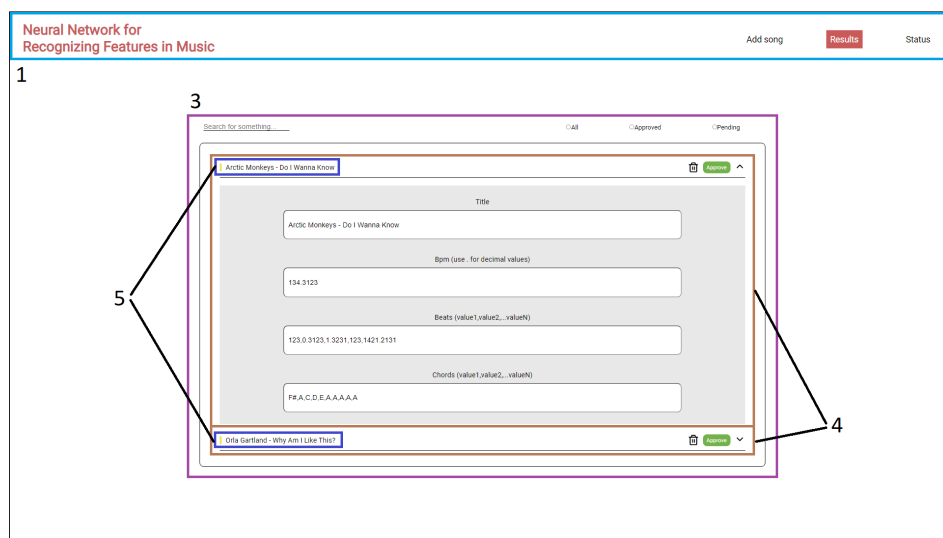
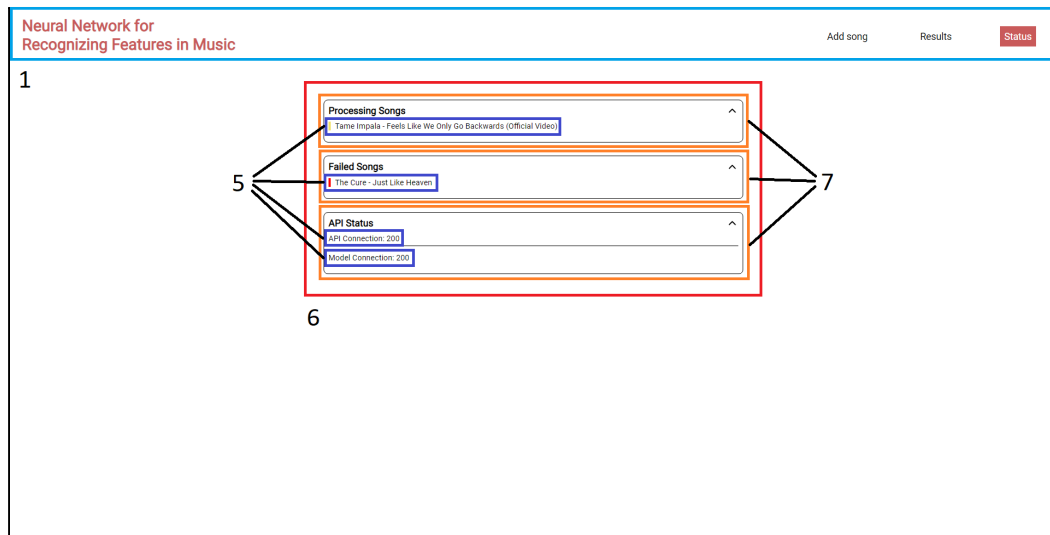


Figure 34 - Output page components



The output page, shown in *Figure 34*, is more complex and includes these components: Songs (purple and number 3), Song (brown and number 4), and SongTitle (dark blue and number 5). Songs is the main component of the output page, and fetches and manages all songs that have been successfully analyzed. It contains one Song component for each song, where the actual results are displayed. Each Song component also consists of a SongTitle component, which only shows the title and whether it is pending or approved.



*Figure 35 - Status page components*

The status page is described in *Figure 35*. It is composed of these components: Status (red and number 6), StatusList (orange and number 7), and SongTitle (dark blue and number 5). Status is the main component, and is built up of variations of the other components. StatusList displays a list of values with a suitable heading. There are three of these present on the status page. One that lists all songs that are currently being analyzed, one for all songs that failed to be analyzed, and one for the API status. Each StatusList has SongTitle components where a title and a status are shown.

#### 6.4.4 Routing

The program's navigation feature is developed with the React Router library, a routing library for React. The web application is a single-page application. When the user navigates the site the pages are not reloaded, but rewritten dynamically. This makes the application fast, as most data is only loaded once. As the users do not have to wait for a whole new page to load when navigating, the user experience is enhanced (25).

```
9  function App() {
10   return (
11     <div className="App">
12       <Navbar />
13       <Routes>
14         <Route path="/" element={<AddSong />} />
15         <Route path='results' element={<Songs />} />
16         <Route path='status' element={<Status />} />
17       </Routes>
18     </div>
19   );
20 }
```

*Figure 36 - Excerpt from the App.js file*

*Figure 36* shows the App.js file, which is a container for the application's components. All available paths are listed in the "Routes" tag along with the corresponding component. There is always a main component rendered along with the navigation bar. When a user navigates from one path to another, the currently rendered component is changed to the desired one. The default component to be rendered is AddSong, as this gives easy access to inputting a new YouTube link.

## 7 Development environment

Tool	Purpose	Usage
<a href="#">GitHub</a>	Handle hosting and distribution of code	Project organization
<a href="#">Visual Studio Code</a>	IDE for development in Golang, React, and Python	Front-end, back-end
<a href="#">Google Drive</a>	Documentation storage	Project organization
<a href="#">Google Sheets</a>	Tool for designing the Gantt chart	Project organization
<a href="#">Google Docs</a>	Cooperative writing and editing	Project report
<a href="#">Toggl</a>	Time logging tool	Project organization
GitHub issue tracker	Git tool for scrum-organization and general issue tracking	Project organization
<a href="#">GitHub workflows</a>	Quality assurance through linting and running tests	Quality Assurance
<a href="#">Miro</a>	Online whiteboard for designing models and diagrams	Design
<a href="#">Figma</a>	Tool for creating wireframes and user tests.	Design
<a href="#">Discord</a>	Online communication app	Project organization
<a href="#">Google Firebase</a>	Cloud storage	Database
<a href="#">Docker</a>	Deployment tool for instancing and managing processes in individual virtual machines.	Deployment

*Table 7 - Tools*

The tools used during the project are shown in table 7. For our IDE of choice we selected Visual Studio Code, as we were all very familiar with it. Its flexibility also lets us use it for both development in Python, Golang and React. We utilized GitHub tools for large parts of our pipeline, including testing and linting through workflows as well as tracking issues and planning our scrums. We found that using GitHub as a hub for our activities in this way worked very well, as it allowed us to centralize these separate processes and more easily track progress and how each person was doing.

Besides this, Miro and Figma were used to model and diagram our network, design UI sketches and perform user testing. Google's Drive services were used to store documentation as they allow for simultaneous editing and input. We chose to deploy our final product on Docker, as it is the VM deployment platform we have most experience with. Since EC-Play was already using Firebase for their database needs, we also used it to store our database.

## 8 Testing and Code Quality

### 8.1 Workflows

The first step in our testing environment is manually checking that the code both builds and runs on our own computer. The next step is automated workflows provided by GitHub. Workflows allows us to automatically test the code on each commit through the use of virtual machines and a set of commands. We would also wait until all workflows were finished before merging the pull request with main as it would make sure the code would run on a neutral machine (not just our own environment) and that it followed the linting style that we picked for each programming language.

Package	Language
flake8	Python
Golang checker	Golang
NPM checker	React

*Table 8 - Linting checkers for each language*

As shown in *table 8*, both Golang and React have an industry standard linting checker already provided making it easy to implement. Python on the other hand required a 3rd party package. We decided on flake8 as it has a lot of interactions on their GitHub repositories, and was rather easy to use.

All of the workflow files are saved in the github folder in the root directory of the repository. The workflow for the internal API in the Neural Network folder is shown in *figure 37*. The pipeline is as simple as: choose the operating system to work on, setup the programming language used, install dependencies, build code, and check linting style using a lint-checker.

```
name: NN Internal API

on:
  push:
    paths:
      - 'NN/API**'
  pull_request:

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.9.10'

      - name: Install dependencies
        working-directory: ./NN/API
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
          pip freeze > requirements.txt
          pip install flake8

      - name: Build
        working-directory: ./NN/API
        run: python -m py_compile ./main.py

      - name: Syntax checker
        working-directory: ./NN/API
        run: flake8 . --max-line-length 150
```

Figure 37 - NN/API workflow

## 8.2 Code review

As part of the development process, we planned to perform code review on each others pull request. Combining this with the Trunk-Based Development strategy and the workflows, the pull requests were small and already checked for the basic linting mistakes which made it faster and easier for us to conduct code reviews. These reviews, while making sure the code quality was good, would also allow each team member to get a better understanding of the entire code base.

## 8.3 User testing

After the first version of the web application's design was created, we performed user testing. We mainly wanted feedback on navigation and functionality, but asked about layout and design as well. Considering how the Covid-19 situation was at that time, it seemed safest to do the entire process online. New restrictions were suddenly introduced, which made it difficult to plan in-person testing. EC-Play is not located near us either, which would

have made physical testing more time consuming. On a positive note, this allowed for a more flexible and effective process.

### 8.3.1 Subjects

The test subjects can be divided into two groups: average users and the client. We wanted our UI to be accessible and easy to use for everyone. By testing on people outside of EC-Play, we got to see if the layout and navigation made sense regardless of the test subject's connection to the task. EC-Play were the ones who created the task, and would probably give different feedback than people that are not familiar with it. However, getting EC-Play's opinion was also important so we could see if we were on the same page regarding the features.

We performed user testing on the average user group in the beginning of February. The decision was made to test on friends and family because of Covid-19. As mentioned earlier, there were strict restrictions at the beginning of the year. For this reason, testing on random people would not have worked well. Since this group is close to us, their opinions and feedback might have been unreliable in the sense that they are too positive of our product. However, we asked follow-up questions if they appeared vague in their answers.

### 8.3.2 Process

The testing on EC-Play was more like a casual chat of what they thought of the design. It was done over Discord during one of our weekly meetings. As it was a digital meeting and they just watched a screenshare of the wireframes, it might have limited their impression. They could not get a feeling on how it is to actually navigate the site. However, the testing helped us with making sure we were on the same page as them when it came to the features of the web application. They could see what we thought were important to include, and could discuss it with us if there had been a misunderstanding during the planning phase. It is important to uncover problems in the design phase, as it makes them easier to fix than in the coding phase. *Table 9* illustrates the problems discovered, and the action we took for each one.

Feedback	Action
Wrong JSON format displayed in the output page.	We got an example of their JSON format, and changed the wireframes to reflect this.
They liked that we had added a status page. However, there should be a section that shows all songs that are currently analyzed.	We added a "Processing songs" section to the status page.

*Table 9 - Feedback from client*

The testing on the other group was also done over Discord, but this time the subjects got the link to the prototype so they could explore it themselves. They got multiple tasks regarding navigation, and would show us how they would solve them by describing their steps and thought process. After finishing the tasks, we asked them how they experienced

the navigation and layout. We also got some general feedback on the design. All answers and tasks can be found in attachment [I](#). The problems uncovered while testing this group are described in *table 10*.

Feedback	Action
It is hard to know that failed songs show up on the status page.	We added a sentence on the output page that tells the user where they can find failed songs.
Having API status on the top of the status page might be a problem, because the songs are the most important part. Not everyone will know what it means.	We moved the processing- and failed songs to the top of the status page.
There is a lot happening on the status page, which makes it confusing.	When viewing results in the output page, you can open and collapse a song. This feature was also added to the status page.
Sorting the songs would be useful. Either alphabetically or by upload date.	No action was taken here, but this could be an improvement for the future. We have a search functionality, which allows the user to find specific songs without trouble.

*Table 10 - Feedback from user testing*

The before and after design of the prototype can be found in attachment [J](#).

## 8.4 React testing

React Testing Library was used for testing the functionality of the React components. An important aspect of this type of testing is the focus on how the end user experiences the application. The implementation details are not important (26). If a user wants to click a button, they do not see the function that is called when the button is clicked. What they see is the visual representation of the button, for example the button name. *Figure 38* shows an integration test of form submitting in a Song component. It checks if the input validation of the “chords” field works correctly. To find the submit button of the form, text is used to query for the button’s name. When the button is found, a click is also triggered. If we had called the function name instead of finding the elements like this, the tests would be harder to maintain. Implementation details may change, and renaming the button click function from “handleClick” to “submitForm” would break the entire test.

```
118     it('invalid chords value is submitted', () => {
119         render(<Song value={resultPending} />, container);
120
121         // click toggle button
122         const toggleBtnEl = screen.getByTestId('arrow-down');
123         fireEvent.click(toggleBtnEl);
124
125         // change value
126         const inputEl = screen.getByLabelText(/chords/i);
127         fireEvent.change(inputEl, { target: { value: 'A,E#' } });
128
129         // click approve button
130         const approveBtnEl = screen.queryByText(/approve/i);
131         fireEvent.click(approveBtnEl);
132
133         // look for error message
134         expect(screen.getByText('Not a valid Chords format')).toBeTruthy();
135     })
```

*Figure 38 - Testing of input validation*

Most of the tests are unit tests, but there are also some integration tests to check if different parts of the application works together. The unit tests revolve around the UI, and if the elements are rendered correctly. An example can be found in *figure 39*. In a Song component, there is either a button called “Approve” or a button called “Edit” displayed. Which one is rendered is based on if the song is marked as approved or not. The tests checks songs with different approved values, to verify that only the correct button is present.



```

49 describe('button displaying', () => {
50   it('approve button is present if the song is pending', () => {
51     render(<Song value={resultPending} />, container);
52
53     const buttonEl = screen.queryByText(/approve/i);
54     expect(buttonEl).toBeTruthy();
55   })
56
57   it('approve button is not present if the song is approved', () => {
58     render(<Song value={resultApproved} />, container);
59
60     const buttonEl = screen.queryByText(/approve/i);
61     expect(buttonEl).toBeNull();
62   })
63
64   it('edit button is present if the song is pending', () => {
65     render(<Song value={resultApproved} />, container);
66
67     const buttonEl = screen.queryByText(/edit/i);
68     expect(buttonEl).toBeTruthy();
69   })
70
71   it('edit button is not present if the song is approved', () => {
72     render(<Song value={resultPending} />, container);
73
74     const buttonEl = screen.queryByText(/edit/i);
75     expect(buttonEl).toBeNull();
76   })
77 })

```

*Figure 39 - Testing of buttons*

The application contains several input fields. In the Song component, these fields are supposed to be set with an initial value when the component is rendered. However, in the AddSong component, no value is supposed to be present. A few of the tests, like the one in *figure 40*, therefore confirms the initial value of an input field.

```

17 it('input is initially empty', () => {
18   render(<AddSong />, container);
19
20   const inputEl = screen.getByRole('textbox');
21   expect(inputEl.value).toBe('');
22 })

```

*Figure 40 - Testing of inputs' initial state*

To limit calls to the database, the testing framework Jest was used to mock the results in some of the integration tests. *Figure 41* shows a test of the Songs component, where fetching from the API would be necessary to display the results. Instead, the response from fetch is mocked with the desired test results.

```
it('all songs are displayed', async () => {
  jest.spyOn(global, 'fetch').mockImplementation(() =>
    Promise.resolve({
      json: () => Promise.resolve(testResults)
    })
  );

  await act(async () => {
    render(<Songs />, container);
  });

  expect(screen.getByText('National Anthem - Lana Del Rey')).toBeTruthy();
  expect(screen.getByText('Kaleidoscope - blink-182')).toBeTruthy();

  global.fetch.mockRestore();
})
```

*Figure 41 - Testing of result displaying*

## 9 Deployment

Docker was used for the deployment of the application. Docker is a software used for containerizing applications, and uses images and containers to accomplish this. An image contains all essentials for running an application, such as dependencies and code. A container is a runnable instance of an image, and combines the application's code and dependencies into a process that is isolated from its environment (27). Docker images can run in every environment, and will be consistent throughout. This was a huge advantage for our group. Since we are using different operating systems, having an easy way to run the application was especially convenient.

A container should only focus on one thing, as this will ease both the updating and scalability of each part of the application. Updating one container will not affect any of the other containers, as they are all isolated. The different parts of the application might also need to be scaled differently, which is why having them in separate containers is the best solution (28). Because our application consists of three separate modules, it had to be a multi-container application. This means each part has its own container.

The tool Compose is used for defining such applications so it can run as a whole in an isolated environment. All that is needed is a file that contains the configuration of the application's services (29). *Figure 42* shows the compose file, where each service is configured with a name and which ports to use. However, to get the program to run properly, the user has to create their own Firebase project and create a file called "serviceAccountKey.json" in the API folder. This is used to authenticate the connection to the database, and should not be shared.

```
1  version: "3.7"
2  services:
3    nn-internal-api:
4      restart: always
5      build: NN/API/
6      container_name: nn-internal-api
7      ports:
8        - 5000:5000
9    main-api:
10     restart: always
11     build: API/
12     container_name: main_api
13     ports:
14       - 8080:8080
15    song-analysis:
16     restart: always
17     build: Web/song-analysis-app/
18     container_name: song-analysis
19     ports:
20       - 3000:3000
```

*Figure 42* - docker-compose.yml

Each service has their own Dockerfile for container creation. All of these files follow the same structure: a new folder for the application is created, dependencies are installed, the source code is copied to the new folder, and lastly the application is started. *Figure 43* shows the Dockerfile for the NN internal API, which illustrates the layout.

```
1 # syntax=docker/dockerfile:1
2
3 FROM python:3.9
4
5 WORKDIR /app
6
7 COPY requirements.txt requirements.txt
8 RUN pip install --no-deps -r requirements.txt
9
10 RUN apt-get update && apt-get -y install ffmpeg libsndfile1-dev
11
12 COPY . .
13
14 CMD export FLASK_APP=main && flask run --host=0.0.0.0
```

*Figure 43 - Dockerfile for NN internal API*

See [README.md<sup>10</sup>](#) in the product repository for detailed instructions on how to deploy the product utilizing the docker command line.

---

<sup>10</sup> <https://github.com/sindre0830/Neural-Network-for-Recognizing-Features-in-Music#readme>

## 10 Results and analysis

### 10.1 Results

As mentioned in chapter 6.1.1, we ended up implementing a solution using the librosa library to perform beat detection. We found this library worked rather well for most songs, though it was unreliable on variable-tempo songs. As we did not reinvent the wheel for this solution, and the results in terms of accuracy are discussed during the implementation, we will not spend more time on these results here.

Before we can discuss the overall results of the NN in recognizing and classifying chords however, a definition of “good” and “bad” results need to be determined. Specifically, how do we evaluate performance? In order to answer this, let’s look at a basic confusion matrix in *figure 44*.

### Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Figure 44 - Basic confusion matrix (30)

As can be seen, results can be divided into four categories. *True positives* are cases where the network correctly identifies a data point as belonging to the correct class. Similarly, *true negatives* correctly identify a data point as not belonging to an incorrect class. *False positives* identify the data point as belonging to a class it does not actually belong to. And finally, *false negatives* incorrectly identify a data point as not belonging to its true class. With these definitions, we can check *table 11* to visualize the different ways in which we might evaluate our results.

Metric	Description	Formula
Precision	Precision compares the correct positive cases with the predicted positive cases.	$\frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$

Recall	Recall compares the correct positive cases with the overall positive cases.	$\frac{\text{True Positive}}{(\text{True Positive} + \text{False Negative})}$
F1	The mean of precision and recall, which gives better measurement of incorrect classification - harmonic mean is usually used to normalize the results.	$2 * \frac{(\text{Recall} * \text{Precision})}{(\text{Recall} + \text{Precision})}$
Accuracy	Accuracy measures all correctly identified cases vs total cases	$\frac{\text{True Positive} + \text{True Negative}}{(TP + TN + FP + FN)}$

Table 11 - Evaluation metrics (31)

Due to the data we are working with and how we evaluate it however, things are somewhat more complicated. This is because in our case we are looking at a set of 24 mutually exclusive labels in which a data point either belongs or does not belong to each class, and the network attempts to classify it into one of these 24. If it fails to match any of the 24 labels, it will classify the data point as a 25th *none* label - however, we consistently observed not a single chord being identified as *none* while testing our various models. Since the validation data only consisted of data that we had trained on, it makes sense for the model to not determine any of it as *none*. If we were to input a blank matrix or something not related to a chord, the *none* label would be output.

One thing to note is that the recall score is identical to the accuracy score in our specific scenario. We can validate this by comparing with the neural network's results. For the model above using the same train-test-validation split, the results evaluation found an accuracy of 82.64% and the same number for recall. If we look at the total accurate classifications compared to total classifications in *figure 49* further down, we get 24650/29829 which is True Positives over N or the full dataset which indeed equals 0.8264 or 82.64%!

The reason for this is a difference between what is called micro and macro averaging. The difference between these approaches is especially relevant for multiclass datasets with imbalanced dataset sizes for each class - exactly the case we are working with. To explain simply, micro averages weigh each sample in the dataset equally - thus classes having more samples will have a greater impact on the average. Macro averages on the other hand weigh each class equally, and will lead to each sample for smaller classes having a larger impact than each sample for larger classes (32). Since we are weighting our results based on dataset size, we are looking at micro averages for our evaluation as this is ideal when looking at overall accuracy (32).

If we go back up to *table 11*, we see that recall considers true positives and false negatives in the dataset. Because our dataset is so skewed towards a handful of classes, what is happening is that for these classes which contribute by far the most to our results, the sample numbers for these two variables is many, many times higher than the numbers of false positives and true negatives. And since these handful of classes massively outnumber the remaining classes, these classes with larger sets for false positive and true negative classification have little impact on the accuracy score. Thus, in the formula for accuracy, TN

and FP become negligible, and the formula approximates the formula for recall as visualized in *figure 45*.

$$\frac{\text{True Positive} + \text{True Negative}}{(TP + TN + FP + FN)}$$

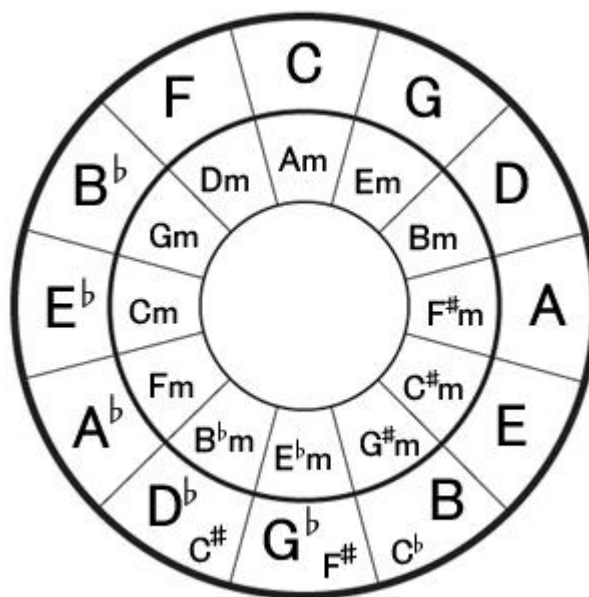
*Figure 45 - Recall and accuracy similarity - small datasets marked in red*

While this helps explain the two results being extremely similar, the fact that recall matches accuracy exactly is most likely a coincidence.

For this task, we decided that the model's accuracy was the best parameter to evaluate overall performance. However, precision, recall and F1 are still very useful parameters for identifying trends or outliers, and will be discussed throughout this section.

When analyzing the success of the neural network in detecting chords, consultation with the client, EC-Play, was done for their musical expertise. During these discussions, the idea of looking at neighboring chords was suggested, an opportunity quickly grasped as it seemed quite interesting as a method of evaluation. Three tiers of accuracy are defined - top1 is perfectly accurate, top-2 includes the nearest neighbor, and top-4 would be the most generous and include the three closest neighbors. Top-1 accuracy is thus identical to the accuracy score we find by parsing the confusion matrix using the formulas above.

This is different from how top-N accuracy is usually defined - in conventional data science, top-4 accuracy would mean that the correct chord is within the 4 top chords guessed by the neural network for a specific data point (33). However, this is a much less useful metric for the practical application of this product for EC-Play, as they are more interested in specifically finding neighboring chords as explained below - having a good top-4 accuracy for this makes it easier for them to check and edit the result manually, as they know to listen for similar chords. Thus, we elected not to look for top-N accuracy in the classical sense.



*Figure 46 - Circle of fifths, visualization (34)*

The concept of neighboring chords was previously explained during the implementation of the chapter 6.1.2 of the thesis, but to sum it up - each chord consists of 3 notes. It shares two of these notes with three other chords of the 24. The easiest way to visualize this is using the circle of fifths, which can be seen in *figure 46*.

Take the D major chord for example - its neighbors are G major, A major, and B minor. Of these, the B minor chord in the inner ring is its very closest neighbor, as it shares the most important notes for the chord. Similarly, D major is B minor's most important neighbor as well. G major and A major also share two notes, but they are not as impactful to the makeup of the chord as those shared by B minor. Additionally several other chords share one note - however, these are not so similar as to meaningfully qualify as neighbors for the purpose of this analysis.

Through this method then, these neighboring chord pairs of the inner and outer circles were defined as closest neighbors, and matches between these would count as "top-2 accurate". In other words, if a D major chord was instead guessed as B minor, it would not qualify as top 1 accuracy, but it would be included in top-2. Anything guessed as G major or A major would count for top-4.

Due to the significantly different sample size for each chord, it was considered interesting to divide these into three general groups - large sample size, medium sample size, and small sample size. Since there are 24 total chords, and a little under 30000 total samples in the validation dataset, a rough division on the number of digits was chosen. Any chord with more than 1000 samples would have a roughly proportional or greater number of datapoints in the set compared to the average, and considered large. 100 - 1000 samples is less than average but still potentially significant, and considered medium - though this is certainly the most volatile of the three groups. Any chord with less than 100 samples was considered to be a part of the small sample size group. See *table 12* for a list of which chords fit into which group.

Small dataset		Medium dataset		Large dataset	
A# minor	14	D minor	105	C# minor	1146
A# major	22	F# major	135	B minor	2141
G minor	23	F major	144	F# minor	4697
F minor	24	A minor	259	E major	5382
C minor	29	C# major	272	D major	5893
G# major	31	B major	307	A major	6788
D# major	34	E minor	565		
D# minor	35	C major	747		
G# minor	69	G major	967		

*Table 12 - Validation dataset sizes divided into three broad size categories*



In order to analyze the results when accounting for nearest neighbors in top-2 and top-4 format, a comparison to some baseline improvement is needed. In order to do this we first need to define a chord organization of the chromatic scale, similar to the circle of fifths above. As can be seen in *figure 47* below, ours is also a wheel or a circuit which starts at the chord C major, and goes through all major chords, then goes back to C minor this time in the minor scale and repeats the same chord progression but in minor instead.

Once we reach B minor at the end, we loop back and the next chord becomes C major again, thus calling it a circuit. The exact order of chords is not important as long as the relation between the chords remains consistent; thus, we know that we can always reach chord one from chord two by applying the same consistent formula.

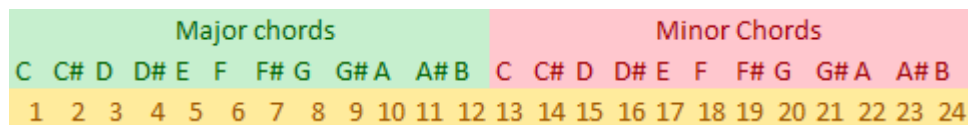


Figure 47 - Chord circuit visualization

Once we have established the chord progression, defining top-1, top-2 and top-4 can be done fairly simply by applying mathematical formulas that identify neighboring chords in the circuit above. The appropriate formulas for our circuit are laid out in *table 13*. Since these chords loop, we can also move backwards - for example instead of  $x+15$  for the top-2 accuracy of minor chords, we could do  $x-9$  instead and land on the same chord.

Major chords		Minor chords	
Top-1 accuracy	$x$	Top-1 accuracy	$x$
Top-2 accuracy	$(x + 9) \% 24$	Top-2 accuracy	$(x + 15) \% 24$
Top-4 accuracy	$(x + 5) \% 12$ $(x + 7) \% 12$	Top-4 accuracy	$(x + 5) \% 12 + 12,$ $(x + 7) \% 12 + 12$

Table 13 - Formulas for finding chords for Top-2 and Top-4 accuracy in chord

Since this method adds additional data from different labels, we need a baseline improvement to compare to as well. For each chord, we subtract the successful identifications from the total dataset, to get the number of mislabeled data points for the chord. Then, a division of this number by 23 is done to get the average number of datapoints for each of the other 23 chords, assuming they were simply randomly distributed. For top 4, multiply the result by 3, to account for the fact that the chord's 3 closest neighbors are added to the accuracy assessment. Another approach could involve weighting the false positive data points based on the size of the wrongly guessed label's dataset size.

Before calculating the results, some predictions were performed. First, a general trend observed and one that made sense with how training of neural networks is performed, was that chords in the "small" dataset would likely be poorly identified, as a lack of data

points makes it hard for the network to learn the features of these chords. For this reason these would likely be fairly inaccurate in general, and the top-2 and top-4 accuracy should perform poorly compared to the baseline improvement metric.

On the other hand, it was expected that the large dataset would perform quite well, for all three tiers of accuracy. For the medium dataset, a trend is more difficult to guess at, especially because as noted there is significantly more variation in viability of dataset size in this group. Generally, it would be expected to lay somewhere between the two other groups.

Improvement Size	Top 2	Top 4	P avg	R avg	F1 avg	Acc avg	Top2	Acc avg	Top4	Acc avg
Small:	5/9	6/9	82.78%	72.61%	75.67%	72.61%	Small:	75.33%	Small:	78.29%
Medium:	6/9	7/9	79.56%	69.96%	74.44%	69.96%	Medium:	71.85%	Medium:	76.34%
Large:	6/6	6/6	81.17%	82.36%	81.33%	82.36%	Large:	86.52%	Large:	91.58%
Total:	17/24	19/24	81.17%	74.98%	77.15%	74.98%	Total	77.90%	Total	82.07%
Weighted total:				82.64%		82.64%	Weighted	85.57%	Weighted	91.18%

Figure 48 - Aggregate results from accuracy analysis. P = Precision, R = Recall, F1

As predicted before numbers were run, the datasets for “small” chords performed fairly poorly. As can be seen in figure 48, they had somewhat decent precision compared to the other groups. This can most likely be explained by the fact that the neural network rarely considered guessing these chords at all, due to lack of training on them. As such, whenever one of these chords were identified, it was extremely likely to be accurate.

	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	Cm	C#m	Dm	D#m	Em	Fm	F#m	Gm	G#m	Am	A#m	Bm
C	610	1	12	0	28	6	0	15	0	18	1	1	0	1	2	0	24	0	4	0	0	7	0	17
C#	0	168	21	0	15	0	1	3	2	10	0	1	0	21	0	2	0	0	22	0	2	0	0	4
D	17	14	4941	2	183	1	1	55	0	265	0	3	0	47	7	0	14	0	192	0	3	6	1	141
D#	0	0	0	26	1	0	1	0	0	0	2	1	1	0	0	2	0	0	0	0	0	0	0	0
E	6	4	154	1	4565	1	1	12	0	303	0	14	0	94	0	1	33	0	109	0	0	5	0	79
F	3	1	2	0	13	107	0	7	0	1	0	0	0	0	2	0	0	1	0	0	0	6	1	0
F#	0	1	2	0	7	0	81	0	0	6	0	2	0	0	0	1	1	0	27	0	2	0	0	5
G	17	1	46	0	23	0	0	801	0	21	0	2	0	2	4	0	18	0	10	0	0	3	0	19
G#	0	2	1	0	1	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
A	11	11	225	1	340	1	0	31	0	5826	0	11	0	71	1	1	21	0	156	0	4	10	0	67
A#	2	1	0	0	1	0	0	0	0	1	14	0	0	0	2	0	0	0	0	0	0	1	0	0
B	1	1	1	0	48	0	0	1	0	8	0	193	0	3	0	1	0	0	12	0	3	0	0	35
Cm	6	0	2	0	2	0	0	0	0	1	1	0	17	0	0	0	0	0	0	0	0	0	0	0
C#m	1	10	47	0	104	0	0	2	0	73	0	3	0	839	1	0	0	0	45	0	1	1	0	19
Dm	3	0	11	0	1	4	0	0	0	5	0	0	0	1	75	0	0	0	2	0	2	0	0	1
D#m	0	1	2	0	0	0	3	0	0	1	0	1	0	0	0	21	0	0	0	0	3	0	1	2
Em	13	0	26	0	86	0	1	15	0	21	0	0	0	0	0	0	384	0	2	0	0	2	0	15
Fm	0	1	0	0	0	3	0	0	0	0	0	0	0	0	0	0	19	1	0	0	0	0	0	
F#m	3	15	214	0	180	0	5	13	0	199	1	4	0	38	2	0	2	0	3892	0	0	2	0	127
Gm	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	1	16	0	2	0	0
G#m	0	4	1	0	4	0	1	0	0	1	0	1	0	0	0	2	0	0	0	0	55	0	0	0
Am	12	0	4	0	9	5	0	2	0	42	1	2	0	4	1	0	1	0	0	0	1	173	0	2
A#m	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0
Bm	3	3	111	0	82	0	2	16	0	42	0	4	0	17	1	0	8	0	62	0	0	0	0	1790

Figure 49 - Confusion matrix, x-axis = Predicted, y-axis = True

A closer look at the confusion matrix in figure 49 confirms this - when a datapoint was misidentified as belonging to these chords, it was generally a point belonging to neighbor chords, and not more than a couple. However there are some interesting outliers here, such as the chord D# minor, which is guessed almost across the board.

A solid performance for precision was predictably let down by a much worse recall score however, as many of the data points belonging to small samples were instead identified as belonging to one of the chords with large sample size instead. Thus recall was a full 10 percentage points worse than precision, giving an F1-score of 75.67%.

Somewhat surprisingly however, despite this more than half of them managed to outperform the baseline for top-2 accuracy improvement. For top-4 accuracy, this increased to 2/3rds of the chords. As such, a weak trend of identifying chord features resulting in neighboring chord identification can be seen.

For “medium” chords, the results were most surprising of all. Across the board for precision, recall and F1, the medium sample-size group performed worse than not only the “large” group, but also the “small” one. If sorting by top1 recall is done, it can be seen that the worst recall scores are crowded by chords in the “medium”-size group, with “small”-size chords more towards the middle, as in *figure 50*. A similar trend is clear for precision.

Chord	Total	Top 1	Top 2	Top 4	Recall	Top2 recall	Diff	Top4 recall	Diff	Avg diff	Improvement Top2	Improvement Top4	Set size
Cm	29	17	17	17	58.62%	58.62%	0.00%	58.62%	0.00%	1.80%	-1.80%	-5.40%	S
F#	135	81	82	85	60.00%	60.74%	0.74%	62.96%	2.96%	1.74%	-1.00%	-2.25%	M
D#m	35	21	24	28	60.00%	68.57%	8.57%	80.00%	20.00%	1.74%	6.83%	14.78%	S
C#	272	168	168	171	61.76%	61.76%	0.00%	62.87%	1.10%	1.66%	-1.66%	-3.88%	M
B	307	193	196	244	62.87%	63.84%	0.98%	79.48%	16.61%	1.61%	-0.64%	11.77%	M
A#	22	14	14	14	63.64%	63.64%	0.00%	63.64%	0.00%	1.58%	-1.58%	-4.74%	S
Am	259	173	185	187	66.80%	71.43%	4.63%	72.20%	5.41%	1.44%	3.19%	1.07%	M
Em	565	384	399	416	67.96%	70.62%	2.65%	73.63%	5.66%	1.39%	1.26%	1.49%	M
Gm	23	16	17	17	69.57%	73.91%	4.35%	73.91%	4.35%	1.32%	3.02%	0.38%	S
Dm	105	75	79	83	71.43%	75.24%	3.81%	79.05%	7.62%	1.24%	2.57%	3.89%	M
C#m	1146	839	943	989	73.21%	82.29%	9.08%	86.30%	13.09%	1.16%	7.91%	9.59%	L
F	144	107	109	112	74.31%	75.69%	1.39%	77.78%	3.47%	1.12%	0.27%	0.12%	M
D#	34	26	27	29	76.47%	79.41%	2.94%	85.29%	8.82%	1.02%	1.92%	5.75%	S
Fm	24	19	19	19	79.17%	79.17%	0.00%	79.17%	0.00%	0.91%	-0.91%	-2.72%	S
G#m	69	55	56	58	79.71%	81.16%	1.45%	84.06%	4.35%	0.88%	0.57%	1.70%	S
G#	31	25	25	27	80.65%	80.65%	0.00%	87.10%	6.45%	0.84%	-0.84%	3.93%	S
C	747	610	617	649	81.66%	82.60%	0.94%	86.88%	5.22%	0.80%	0.14%	2.83%	M
G	967	801	819	892	82.83%	84.69%	1.86%	92.24%	9.41%	0.75%	1.12%	7.17%	M
F#m	4697	3892	4091	4256	82.86%	87.10%	4.24%	90.61%	7.75%	0.75%	3.49%	5.51%	L
Bm	2141	1790	1901	1971	83.61%	88.79%	5.18%	92.06%	8.45%	0.71%	4.47%	6.32%	L
D	5893	4941	5082	5397	83.85%	86.24%	2.39%	91.58%	7.74%	0.70%	1.69%	5.63%	L
E	5382	4565	4659	4976	84.82%	86.57%	1.75%	92.46%	7.64%	0.66%	1.09%	5.66%	L
A#m	14	12	13	13	85.71%	92.86%	7.14%	92.86%	7.14%	0.62%	6.52%	5.28%	S
A	6788	5826	5982	6547	85.83%	88.13%	2.30%	96.45%	10.62%	0.62%	1.68%	8.77%	L
Total:	29829	24650	25524	27197									

Figure 50 - Detailed accuracy results by chord

Despite this, the trend for top-2 and top-4 accuracy improvement over baseline were slightly better than for the “small” group. The “large”-size group meanwhile, performed well in all three parameters. It did lose out to the “small”-size group in the metric of precision. However, this itself is slightly misleading - the by far smallest member of this group, the chord C# minor, is significantly dragging the average down in this case.

Using this methodology, it was determined that all of the large datasets significantly overperformed the average expected improvement with top-4 accuracy, between 3.29% and a full 9.59%. All these large datasets managed to perform better than the average in terms of top 2 accuracy as well, though the improvement was not as significant. That said, the distribution of erroneously identified data points is not evenly distributed, and a trend of mistaken guesses between the four largest chords can also be seen. These are often larger than the nearest neighbor as well, showing that the network is acquiring a certain amount of bias based on the skewed size of the datasets for each individual chord.

Besides these trends, there is also a trend of a major chord being guessed as its minor counterpart, and vice versa. These chords are only separated by one note - and not the dominant note - so there is similarity between these two (19). Additionally, sometimes the tone which separates the two chords is not played, which makes it difficult even for a trained ear to tell the difference. This also goes a long way towards explaining why the chords with medium-sized datasets performed worse than those with small datasets - every single chord in the large dataset had its counterpart chord in the medium, rather than the small, dataset. Thus the observed trend of chords from smaller datasets being mistakenly classified as belonging to a chord from a much larger dataset is exacerbated by the chord similarities

themselves. This can be seen if we look at the confusion matrix for the validation dataset classification results in *figure 51*.

	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	Cm	C#m	Dm	D#m	Em	Fm	F#m	Gm	G#m	Am	A#m	Bm
C	610	1	12	0	28	6	0	15	0	18	1	1	0	1	2	0	24	0	4	0	0	7	0	17
C#	0	168	21	0	15	0	1	3	2	10	0	1	0	21	0	2	0	0	22	0	2	0	0	4
D	17	14	4941	2	183	1	1	55	0	265	0	3	0	47	7	0	14	0	192	0	3	6	1	141
D#	0	0	0	26	1	0	1	0	0	0	2	1	1	0	0	2	0	0	0	0	0	0	0	0
E	6	4	154	1	4565	1	1	12	0	303	0	14	0	94	0	1	33	0	109	0	0	5	0	79
F	3	1	2	0	13	107	0	7	0	1	0	0	0	0	2	0	0	1	0	0	0	6	1	0
F#	0	1	2	0	7	0	81	0	0	6	0	2	0	0	0	1	1	0	27	0	2	0	0	5
G	17	1	46	0	23	0	0	801	0	21	0	2	0	2	4	0	18	0	10	0	0	3	0	19
G#	0	2	1	0	1	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
A	11	11	225	1	340	1	0	31	0	5826	0	11	0	71	1	1	21	0	156	0	4	10	0	67
A#	2	1	0	0	1	0	0	0	0	1	14	0	0	0	2	0	0	0	0	0	0	1	0	0
B	1	1	1	0	48	0	0	1	0	8	0	193	0	3	0	1	0	0	12	0	3	0	0	35
Cm	6	0	2	0	0	0	0	0	0	1	1	0	17	0	0	0	0	0	0	0	0	0	0	0
C#m	1	10	47	0	104	0	0	2	0	73	0	3	0	839	1	0	0	0	45	0	1	1	0	19
Dm	3	0	11	0	1	4	0	0	0	5	0	0	0	1	75	0	0	0	0	2	0	2	0	1
D#m	0	1	2	0	0	0	3	0	0	1	0	1	0	0	0	21	0	0	0	0	3	0	1	2
Em	13	0	26	0	86	0	1	15	0	21	0	0	0	0	0	0	0	0	384	0	2	0	2	15
Fm	0	1	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	19	1	0	0	0	0	0
F#m	3	15	214	0	180	0	5	13	0	199	1	4	0	38	2	0	2	0	3892	0	0	2	0	127
Gm	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	1	16	0	2	0	0
G#m	0	4	1	0	4	0	1	0	0	1	0	1	0	0	0	2	0	0	0	0	55	0	0	0
Am	12	0	4	0	9	5	0	2	0	42	1	2	0	4	1	0	1	0	0	0	1	173	0	2
A#m	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0
Bm	3	3	111	0	82	0	2	16	0	42	0	4	0	17	1	0	8	0	62	0	0	0	0	1790

Figure 51 - Five worst-performing chords in medium dataset, x-axis = Predicted, y-axis = True

Here the correctly identified chords are marked in green, and their major/minor counterparts are marked in red. Each row represents the total dataset for that chord, with each column being guesses. As we can see, all five have a significant number of wrong guesses for their counterpart major/minor chord relative to the total wrong guesses. And all except for B major had more wrong guesses for their counterpart chords than any of their neighbors on the chord progression wheel, which is very much opposite the trend for the rest of the chords in this analysis.

Small and large-dataset chords trended to have mistaken guesses instead hit their neighbors more often than their counterparts, which is shown with the chord C# minor in *figure 51* with blue and gray instead. C# minor was by far the worst performing chord from the large dataset, and as we can see following its row it has many faulty guesses, including other large datasets and its neighbors. In comparison, only 10 data points of C# minor were mistakenly identified as C# major instead, a tiny portion of the total misclassifications.

Weighting the results of all chords evenly, the neural network manages 75% top1 accuracy score, which increases to 78% for top-2 and 82% for top-4. These numbers are not ideal, but these numbers are being significantly hampered by the results for the “medium” and “small”-size groups in such a scenario. The more interesting data can be seen when weighting instead considers the number of datapoints for each chord, giving more weight to chords with a plurality of data points. In this scenario, even top-1 accuracy reaches 82.65%, top-2 85.57%, and top-4 hits a solid 91.18%.

There are drawbacks to looking solely at this weighted result and discarding the results for chords with fewer data points as less important, some of which have been previously mentioned. Simultaneously, while a neural network that excels at all music and all chords would be ideal, the significant imbalances in the sample dataset is representative of the imbalances in the music that EC-Play itself is adding to its website. If the hundreds of songs they have added thus far include more than 25000 E major chords, and only a paltry

150 or so D# minor chords, then odds are they will continue to process songs that are overwhelmingly filled more with the former than the latter.

As such, a neural network which excels at finding the most common chords in exactly this kind of music is not only perhaps a natural result of training on the dataset, but also possibly a very good fit for the task required. A neural network solution which can fill in the vast majority of “normal” chords and allow for manual editing and addition of more unusual or uncommon chords is exactly what EC-Play were looking for when discussion of this thesis first began. Ideas for how to improve the dataset and mitigate issues caused by imbalance follows in chapter [10.2.1](#).

## 10.2 Recommendation and further work

### 10.2.1 Dataset preprocessing

Through our thorough analysis of the results from our neural network solution, it is clear that the largest issue that can be observed throughout is a tendency for data points from chords with smaller datasets to be mistakenly classified as belonging to one of the handful of largest datasets. This is not too surprising, as these 4-5 chords have orders of magnitude more data points than most of the others, and so the neural network is not trained in a balanced manner, and there is a risk of overfitting even despite using stratification during train-test-splitting.

The obvious solution then would be to better balance out the number of datapoints for each chord in the dataset. This can be achieved in a few ways. The most intuitive perhaps is to simply find more samples of the chords with fewer data points, but this will be a very difficult and time-consuming process - not only will songs with unusual or uncommon chords need to be found, but their chords will also need to be transcribed manually in order to create the dataset. A more reasonable solution is to perform resampling on the dataset ([35](#)). With some chords having a mere one or two hundred data points while others have tens of thousands, a solution utilizing both oversampling and undersampling would likely be a good solution ([36](#)).

Resampling involves simply changing the number of samples for each class in the dataset. When undersampling, some samples are removed from classes with many data points, while oversampling involves duplicating existing examples from classes with few data points. Resampling should be done with care however - oversampling can result in overfitting if performed excessively, which is definitely a significant risk in this particular case. And undersampling can lose important information ([36](#)).

### 10.2.2 Improved beat detection

For the algorithm solution implemented, a basic improvement to be made could be implementing a solution for better detecting the start and end of a song's beat. Currently, noise and denouement can cause issues for the beat algorithm solution, as it is unable to properly parse and recognize such sections, leading to it finding beats before the beat of the song properly starts, or after it is finished, as mentioned in chapter [6.1.1](#). One solution here could be to apply the algorithm separately to only the start and end of the song, so that these features are given more weight.

More obviously, a full transition to a Neural Network solution for this part of the product would be a great next step. However, the challenges with this have already been

outlined in chapter 6.1.1. If a reliable dataset of chromagrams or audio-files with accompanying beat timestamps or similar can be found, this would be a fairly manageable iteration to make. Faced with the prospect of manually creating such a dataset however, this becomes a lot more difficult.

And if such a dataset was found, it would also beg the question: how reliable and accurate is such a dataset? If it was created by hand it runs the risk of the same accuracy issues that EC-Play's dataset struggles with. If created by machine, it would either be equally questionable, or that generator itself would be the beat detection algorithm we are looking for, similar to librosa's library currently being used. Further work here seems likely to require the involvement of someone well educated and learned on the topic of audio detection.

If a workable dataset was procured, a solution similar to that of for example Mullaney (37) could be implemented for beat detection, using a convolutional neural network to perform onset detection.

### 10.2.3 Model selection

For the chord recognition, the primary improvement we suggest would be iterating on the model selection process. Here there are a few different things that could be tweaked or improved. First, the search process itself could be improved by using Keras libraries in order to implement true random searching, rather than our randomized grid search variant. This would not only allow for true random searching if desired, but would also allow the usage of keras tools such as scoring or refitting in order to better evaluate the parameters used and improve accuracy on both types of search.

While there is discussion on what kind of search to use, Maladkar (38) argues that randomized hyperparameter tuning is superior to a grid search approach due to being faster and more efficient as it does not need to look at all combinations - an advantage shared by our implementation - and it can find outliers and is more likely to reach the entire action space, which our randomized grid search does not accomplish.

Alternatively, an approach that combines the two search methods is advocated by Ismiguzel (39), where she suggests first performing randomized search in order to identify the most interesting hyperparameters for tuning, and then performing grid search using these parameters afterwards for optimization. This approach essentially uses the randomized search to weed away uninteresting or low-impact parameters for the grid search, making said grid search a lot more computationally affordable. Such approaches would be very interesting to implement for this product in the future.

### 10.2.4 Implement pattern recognition

As part of the task as outlined during the thesis selection process, EC-Play suggested the implementation of pattern recognition in addition to beat detection and chord classification. We previously discussed in chapter 1.3 and others, we initially planned to attempt solving this challenge, but this was abandoned somewhat early in the production process, due to a combination of several factors.

It is, however, a natural next step for iterating on the product we created during the course of our work. In order to accomplish this, some challenges related to inaccuracies in the beat and chord recognition process must first be overcome however.

We can imagine a scenario where our chord recognizer has identified a chord pattern of 5-3-5-3 for example, where the actual chords involved should be 4-4-4-4. For a user to manually correct these chords involves changing exactly two chords in the first and third grouping above. However, if these were combined into patterns using a pattern recognizer, a lot more work would need to be performed. In this case, rather than simply changing the chords themselves, the user would also need to remove the extra chord from the first and third group, as well as add them to the second and fourth. Even in this simple example where the pattern recognition itself worked perfectly, we can see that a lot of extra work is added. In worse-case scenarios, we risk a resulting pattern of something like 5-1-2-3-2-4-1-2-1. This would clearly do a very poor job of alleviating workload.

One alternative approach that could be taken, then, is to separate the beat and chord recognition processes from the pattern recognition process. If the user was required to first edit and approve the chords and beats before pattern recognition is performed, it would be possible to guarantee correct input and eliminate issues relating to inaccuracies snowballing as in the example above. This could be an interesting and worthwhile next step for the product, and give the pattern recognizer a much better shot at improving the work process.

### 10.2.5 Web application and API requests

The requests between the web application and API could be improved, as they have not been optimized yet. In our implementation, data is fetched from the API to the website each time the relevant component is rendered. There is no system for caching the data. This means that every time a request is sent to an endpoint, the handler retrieves data straight from the database. We use firestore for storing data, which only allows 50 000 reads for free every month. Because of this, constant calls to the database might become an issue. At the end of the development period, implementing caching was originally planned, but finishing the core features and bugs was deemed as more crucial.

The status page makes use of polling for getting a live update of the songs currently being processed. An interval of 2 seconds is set between each call to the API. As stated in the paragraph above, this can cause problems as it massively affects the database usage. One fix to this issue could be the use of webhooks. Instead of the website actively polling the API for new updates, the API could send the website a notification when an update occurred.

## 10.3 Conclusion

### 10.3.1 Product

At the end of this project, what has been produced is a quality application that we believe fulfills all the requirements of the client at the outset of this thesis work. A web-app frontend with three pages for passing in a song for processing, receiving the output in plaintext form and uploading it to a database, as well as a page to see the status of the system and any errors. An API to control communication between web, music processing and database. Solutions for extracting beat timestamps and identifying chords in songs. And a system to upload to and update the database through the results webpage.

As far as we are aware, EC-Play are quite satisfied with the work done, and are understanding of the issues faced regarding the beat dataset. As communication with them has been constant during the project, their input as to how to solve such issues and what work to prioritize has been invaluable throughout the process.

### 10.3.2 Self-evaluation

Overall, we are happy with our work done and the processes we used to accomplish it. We are proud of the final product we created, and have learned a lot about programming, music, development processes, deployment, testing and much more during our time working on our thesis. However, this does not mean everything went perfectly. Indeed there are always going to be things that could or even should be improved about any process.

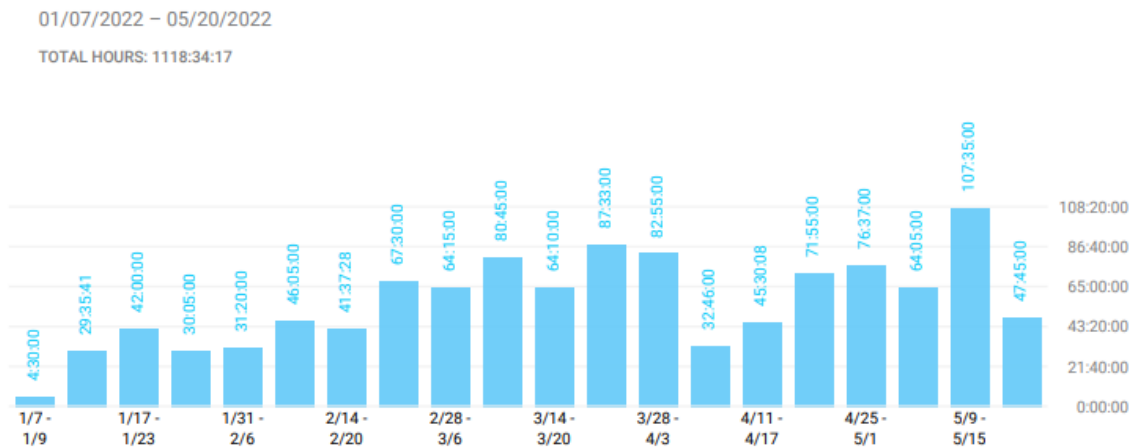


Figure 52 - Time log for hours worked per week

Our first challenge was time management, as can be seen in *figure 52*. This materialized in a few different ways. First the challenge of pouring all our work into a single project for most of the semester was one we found surprisingly tough. This despite the very prospect of doing so being something we had anticipated eagerly before starting. In particular, the February period after finishing our project plan and starting work on the programming solution for the product was a struggle from our side at times on this point. Because of this, we were not quite as productive as we hoped to be, nor did we reach the goal number of hours worked per week we had set ourselves during the planning stages. For more details on hours worked, see attachment [L](#).

While we are somewhat disappointed by this, we do not feel like the project overall was harmed by this, though no doubt there would be some smaller changes or additions that could be made had we managed to better stick with our ambitions on this point. It was a learning experience that we will be sure to bring with us to future projects. Our hours spent also grew throughout the project, partly due to learning better time management and partly due to simply needing to complete the remaining work.

In addition, we had some struggles during late March and April due to illness, as several members of the team came down with some sort of bug, including in one case Coronavirus. We had acknowledged the chance of this happening during risk assessment, and thankfully we did not have anyone out for long enough to harm overall progress.

Finally, we also had some time management issues due to being indecisive regarding questions for the client and advisor at times. Multiple times we paused working on some part of the product to await clarification or answers from these two parties, and waited a day or two until the next scheduled meeting to bring it up rather than immediately contact them. Certainly we usually had other work we could perform in the meantime, but in



hindsight this definitely contributed to poor time-management and was part of the reason why our hours did not climb as high as planned. For the future, we have learned to be more proactive in these scenarios. Indeed, we already significantly improved at this throughout the semester, and by the final month we were generally on the ball about writing questions and asking for support through channels like Teams and Discord immediately as issues cropped up.

For our approach to the problems relevant to the task, we believe we overall solved them well. The things we wish could have been improved or added to the product were either outside the scope of what we worked on, or had to be cut or truncated due to the state of the dataset available. The end result is a flexible product that should be easily iterated on in the future if the client so wishes.

### 10.3.3 Gantt chart

If we evaluate our two Gantt charts for the project plan as seen in attachment [A](#), and our final Gantt chart in attachment [G](#), we can see a general trend of work taking longer than anticipated. This is in many ways to be expected, as it would be surprising if we were able to accurately assess the amount of work each task in our chart would require. Indeed, research has shown that humans consistently underestimate how long a task will take to complete, a phenomenon known as the planning fallacy ([40](#)).

While some of this could potentially be attributed to the lower than planned work-hours per week, we believe the majority can be blamed on poor estimation and unforeseen issues and additional work cropping up during the work process. For example, we had not planned for model search implementation during the project planning phase. Not only did we add this, which took some time, but we also had to spend a long time debugging the Keras library implementation and coming up with our own solution afterwards.

### 10.3.4 Risk assessment

Looking at the risk assessment performed during project planning (see attachment [A](#)), we mostly hit the nail on the head. While we were not aware of the scope of difficulties with the dataset, or how tough it would be to find good alternatives, we did correctly identify it as a large risk for our thesis.

For our medium risks, it was more of a mixed bag. Illness did strike for multiple team members as well as our advisor at various times, and did have some impact on progress. Thankfully, the magnitude of this impact was mostly minimized. The issue where the API we use to download songs from Youtube could stop working or change is also still a potential future problem, though not one we can really affect. We implemented a simple editing solution and provide the output data in an editable format from which it can be easily transferred between EC-Play's proprietary editing tools - this is a large part of why we did not spend time implementing a more user friendly tool for editing, as we kept our product solution separate from EC-Play's website and agreed to let them handle the work of connecting the two.

On the other hand, our lack of music-theory knowledge did not present insurmountable barriers for creating the chord-recognition model. Nor did our worry regarding frequency and resampling losing information end up panning out. Similarly, our worries regarding the networking bottlenecks also seem unlikely to materialize after testing our API design.

At the end of the day, the main lesson we take away from our risk assessment in hindsight is that we should have put more emphasis on process-related risks. We have certainly learned to respect the challenges that a full product design and implementation like this involves.

#### 10.3.5 Final thoughts

The experience working on this thesis has been invaluable for all three of us, and we have had a great time throughout. We have also outlined several areas where we believe further work could be done, and give EC-Play the opportunity to pursue these should they so choose. We would like to thank EC-Play for being amazingly cooperative and helpful during the work, and for giving us the chance to take on this thesis work in the first place.

# 11 References

The references are written in the Vancouver style and we have used [NTNU's documentation](#) as a template.

1. Ellis, D. Beat Tracking by Dynamic Programming. New York; Columbia University, LabROSA; 2007 Jan 16. Available from: <https://www.ee.columbia.edu/~dpwe/pubs/Ellis07-beattrack.pdf>
2. What is Scrum? [Internet] Lexington, Massachusetts; c2020 [cited 2022 May 13]. Available from: <https://www.scrum.org/resources/what-is-scrum>
3. Hammant, P. Trunk Based Development: Introduction [Internet]. [place unknown] Paul Hammant; 2020 [cited 2022 Apr 18]. Available from: <https://trunkbaseddevelopment.com/>
4. Meta Platforms, Inc. Components and Props [Internet]. Menlo Park: Meta Platforms, Inc.; 2022 [cited 2022 Apr 15] Available from: <https://reactjs.org/docs/components-and-props.html>
5. Meta Platforms, Inc. Introducing Hooks [Internet]. Menlo Park: Meta Platforms, Inc.; 2022 [cited 2022 May 11] Available from: <https://reactjs.org/docs/hooks-intro.html>
6. Meta Platforms, Inc. Rendering Elements [Internet]. Menlo Park: Meta Platforms, Inc.; 2022 [cited 2022 Apr 15] Available from: <https://reactjs.org/docs/rendering-elements.html>
7. Abadi, M. Barham, P. Chen, J. Chen, Z. Davis, A. Dean, J. Devin, M. Ghemawat, S. Irving, G. Isard, M et al. Tensorflow: A system for large-scale machine learning [Internet]. San Francisco; 2016 [accessed 2022 Apr 29]. doi: <https://doi.org/10.5281/zenodo.5043456>.
8. McFee, B. Lostanlen, V. McVicar, M. Metsai, A. Balke, S. Thomé, C. Raffel, C. Malek, A. Lee, D. Zalkow, F. et al. San Francisco: LibROSA/LibROSA: 0.7.2; 2020 [accessed 2022 Apr 29]. doi: <https://doi.org/10.5281/zenodo.6097378>
9. Hennequin, R. Khlif, A. Moussallam, M. Voituret, F. Spleeter: a fast and efficient music source separation tool with pre-trained models [Internet]. Journal of Open Software. 2020 [cited 2022 May 13]. doi: <https://doi.org/10.21105/joss.02154>
10. Dowse, J. Tempo vs. pitch class [Internet]. 2011 Feb 22 [cited 2022 Mar 29]. Available from: <https://jbdowse.com/poib/music/>
11. Wagner, B et al. Octave equivalence perception is not linked to vocal mimicry: budgerigars fail standardized operant tests for octave equivalence. Behavior [Internet]; 2017;156(5-8):479-82. Available from: [https://brill.com/view/journals/beh/156/5-8/article-p479\\_4.xml?language=en&ebody=pdf-49903](https://brill.com/view/journals/beh/156/5-8/article-p479_4.xml?language=en&ebody=pdf-49903)

12. Purwins, H. Profiles of Pitch Classes - Circularity of Relative Pitch and Key: Experiments, Models, Music Analysis, and Perspectives [PhD thesis]. Berlin: Berlin Institute of Technology; 2005. Available from: <https://depositonce.tu-berlin.de/handle/11303/1499>
13. Hemholz, H. Medebach, I. Kokabi, O. Chord Recognition based on Template Matching. Berlin: Berlin Institute of Technology; 2015. Available from: [https://www.researchgate.net/publication/325070280\\_Chord\\_Recognition\\_based\\_on\\_Template\\_Matching](https://www.researchgate.net/publication/325070280_Chord_Recognition_based_on_Template_Matching)
14. Weiß, C. Peeters, G. TRAINING DEEP PITCH-CLASS REPRESENTATIONS WITH A MULTI-LABEL CTC LOSS. Paris: Institut Polytechnique de Paris; 2021. Available from: <https://archives.ismir.net/ismir2021/paper/000094.pdf>
15. Lee, K. Automatic Chord Recognition from Audio Using Enhanced Pitch Class Profile. Seoul: Seoul National University; 2006. Available from: [https://www.researchgate.net/publication/228347381\\_Automatic\\_Chord\\_Recognition\\_from\\_Audio\\_Using\\_Enhanced\\_Pitch\\_Class\\_Profile](https://www.researchgate.net/publication/228347381_Automatic_Chord_Recognition_from_Audio_Using_Enhanced_Pitch_Class_Profile)
16. Bjerkeseth, M. Using Hidden Markov Models for fault diagnostics and prognostics in Condition Based Maintenance systems. University of Agder, Norway. 2010. Available from: <https://uia.brage.unit.no/uia-xmlui/bitstream/handle/11250/137502/Bjerkeseth.pdf?sequence=1&isAllowed=y>
17. Kwok, R. Viterbi algorithm for prediction with HMM — Part 3 of the HMM series [Internet]. Hong Kong. 2019 Jul 21 [cited 2022 Mar 30]. Available from: <https://medium.com/analytics-vidhya/viterbi-algorithm-for-prediction-with-hmm-part-3-of-the-hmm-series-6466ce2f5dc6>
18. Nemeroff, B. What is the Difference Between Major and Minor Chords? [Internet]. Fender.com. San Francisco, California. 2021 Feb 25 [cited 2022 May 10] Available from: <https://www.fender.com/articles/play/minor-vs-major-chords>
19. Ye, A. Finally, an intuitive explanation of why ReLU works [Internet]. United States: Towards Data Science; 2020 Jul 19 [cited 2022 Apr 16]. Available from: <https://towardsdatascience.com/if-rectified-linear-units-are-linear-how-do-they-add-nonlinearity-40247d3e4792>
20. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting [Internet]. United States: Journal of Machine Learning Research (JMLR); 2014 [cited 2022 May 15]. Available from: <https://jmlr.org/papers/v15/srivastava14a.html>
21. Singla S. Why is Batch Normalization useful in Deep Neural Networks? [Internet]. United States: Towards Data Science; 2020 Jul 29 [cited 2022 May 15]. Available from: <https://towardsdatascience.com/batch-normalisation-in-deep-neural-network-ce65dd9e8dbf>

22. Penyel D. Where Are They? [Internet]. United States: Medium; 2020 Nov 27 [cited 2022 May 16]. Available from: <https://medium.com/swlh/where-are-they-b65e5669e6c4>
23. Babich, N. Responsive Web Design Tutorial and Best Practices [Internet]. San Jose: Adobe Systems; 2019 Oct 16 [cited 2022 May 11]. Available from: <https://xd.adobe.com/ideas/principles/web-design/responsive-web-design-2/>
24. Morales, J. Mobile First Design Strategy: The When, Why and How [Internet]. San Jose: Adobe Systems; 2021 Feb 16 [cited 2022 May 12]. Available from: <https://xd.adobe.com/ideas/process/ui-design/what-is-mobile-first-design/>
25. Mozilla Corporation. SPA (Single-page application) [Internet]. Mozilla Corporation: Mountain View; 2021 Oct 8 [cited 2022 Apr 16]. Available from: <https://developer.mozilla.org/en-US/docs/Glossary/SPA>
26. McCurdy, N. React Testing Library [Internet]. Kent C. Dodds and contributors; 2021 Jul 21 [cited 2022 Apr 24]. Available from: <https://testing-library.com/docs/react-testing-library/intro/>
27. Docker Inc. Orientation and setup [Internet]. Palo Alto: Docker Inc.; 2021 [cited 2022 May 13]. Available from: <https://docs.docker.com/get-started/>
28. Docker Inc. Multi container apps [Internet]. Palo Alto: Docker Inc.; 2021 [cited 2022 May 13]. Available from: [https://docs.docker.com/get-started/07\\_multi\\_container/](https://docs.docker.com/get-started/07_multi_container/)
29. Docker Inc. Overview of Docker Compose [Internet]. Palo Alto: Docker Inc.; 2021 [cited 2022 May 13]. Available from: <https://docs.docker.com/compose/>
30. Draelos, R. Measuring Performance: The Confusion Matrix [Internet]. United States: Medium; 2019 Feb 17 [cited 2022 May 15]. Available from: <https://towardsdatascience.com/measuring-performance-the-confusion-matrix-25c17b78e516>
31. Huilgol, P. Accuracy vs. F1-Score [Internet]. United States: Medium; 2019 Aug 24 [cited 2022 May 15]. Available from: <https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2>
32. Androidkt. Micro and Macro Averages for imbalance multiclass classification [Internet]. [place unknown] 2021 Aug 29 [cited 2022 May 15]. Available from: <https://androidkt.com/micro-macro-averages-for-imbalance-multiclass-classification/>
33. Riva, M. Top-N Accuracy Metrics [Internet]. Barcelona; 2021 Mar 15 [cited 2022 May 15]. Available from: <https://www.baeldung.com/cs/top-n-accuracy-metrics>
34. Theta Music Technologies, Inc. Circle of Fifths [Internet]. Tokyo; 2022 [cited 2022 May 15]. Available from: <https://trainer.thetamusic.com/en/content/html5-circle-fifths>

35. Sasada, T. Zhaoyu, L. Baba, T. Hatano, K. Kimura, Y. A Resampling Method for Imbalanced Datasets Considering Noise and Overlap. *Procedia Computer Science*. 2020:176:420.429. doi: <https://doi.org/10.1016/j.procs.2020.08.043>
36. Brownlee, J. Random Oversampling and Undersampling for Imbalanced Classification [Internet]. Vermont, Victoria. 2022 Jan 5 [cited 2022 May 10]. Available from: <https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/>
37. Mullaney T. Musical Beat Detection with Convolutional Neural Networks [Internet]. New York, United States; 2016 [cited 2022 May 18]. Available from: <http://tommyullaney.com/projects/rhythm-games-neural-networks>
38. Maladkar, K. Hyperparameter Tuning with Grid Search and Random Search [Internet]. Bengaluru, India; 2021 Sep 29 [cited 2022 May 11]. Available from: <https://analyticsindiamag.com/why-is-random-search-better-than-grid-search-for-machine-learning/>
39. Ismiguzel, I. Why Is Random Search Better Than Grid Search For Machine Learning [Internet]. United States; 2018 Jun 14 [cited 2022 May 11]. Available from: <https://towardsdatascience.com/hyperparameter-tuning-with-grid-search-and-random-search-6e1b5e175144>
40. Buehler, R. Griffin, D. Ross, M. Exploring the "planning fallacy": Why people underestimate their task completion times. *Journal of Personality and Social Psychology*; 1994. Available from: [http://web.mit.edu/curhan/www/docs/Articles/biases/67\\_J\\_Personality\\_and\\_Social\\_Psychology\\_366,\\_1994.pdf](http://web.mit.edu/curhan/www/docs/Articles/biases/67_J_Personality_and_Social_Psychology_366,_1994.pdf)

## A Project Plan - Spring 2022

# Neural Network for Recognizing Features in Music

Rickard Loland	707081
Sindre Eiklid	526361
Maren Skårestuen Grindal	526359

# Table of Contents

---

<b>Goals</b>	<b>2</b>
Business case	2
Objectives	2
<b>Scope</b>	<b>3</b>
Description	3
Delimitation	4
Domain Knowledge	4
<b>Project organization</b>	<b>5</b>
Responsibilities and roles	5
<b>Planning</b>	<b>6</b>
Development process	6
Meetings	6
Product backlog	6
Milestones	8
<b>Quality Assurance</b>	<b>9</b>
Routines	9
Documentation, source code, and storage	9
GitHub Workflows	9
Testing	9
Documentation and storage	9
Risk assessment	10
Risk evaluation matrix	11
Planned controls for handling risks	11
Tools	12
<b>Progress plan</b>	<b>13</b>
<b>References</b>	<b>15</b>



## Goals

### Business case

EC-Play works with educational technology revolving around music. According to their website, their goal is to change early-stage music education [1]. To do this, they create dynamic, easy-to-follow compositions for music targeted especially at allowing young children to follow along and learn.

Today the client has to analyze audio tracks by manually plotting beats, chords, and chord patterns. This takes a lot of time to do and has a large risk of human error. They now want to make this process mostly automatic through the use of AI.

### Objectives

Goal	Description
Effect oriented	The user can get info about the composition of an audio file
	Composition is easy to parse and follow
	The client can reduce workload
	The user can correct the output and help train the model to become better
Results-oriented	The model needs to be able to recognize beats, chords, and chord patterns in an audio file.
	The program needs to be able to receive audio files and deliver the analysis results through an API.
	UI to simplify the usage of the API and showcase the results in a conceivable manner.
	Database to store results from the analysis.
	Function in the web application to correct results from the analysis.
	Automatic training of model on an updated dataset.
	Create a model with =>75% beat accuracy with a margin of error of $\pm 5\%$ , where the time spent fixing the errors will be less than transcribing the audio track manually
Learning outcomes	Gain experience working with SCRUM methodology.
	Gain experience working with a large business-oriented project.
	Acquire more knowledge and experience working with neural networks and machine learning in the context of audio.

	Learn how to automatically compare two models where the best one is used in the program.
	Learn how to use modern solutions for web development.
	Gain knowledge through advanced cloud development.

*Table 1 - Goals*

Concerning the accuracy specified in *Table 1*, the best accuracy target is based on results from other papers researching the extraction of BPM and other features using machine learning. Here, an accuracy of 60 - 70% was achieved with a margin of error  $\pm 4\%$ , and into the 90s with a larger margin of error [4]. Designing a CNN with multifilter modules, Schreiber & Müller achieved slightly better results [5]. Both papers also suggest potential improvements to their models that we will research, to hopefully achieve better accuracy.

## Scope

### Description

As shown in *Table 2*, our task is to develop a full-stack system composed of a web application, a cloud structure through the use of API, and a machine learning model. A description of what each component should offer is listed in the table below.

Component	Description
Web Application	The web application needs to have these functionalities: <ul style="list-style-type: none"> <li>- Send a song to analysis.</li> <li>- Get results from the song analysis.</li> <li>- Correct the output of the song analysis to improve the model.</li> </ul>
API	The API interface needs to handle input and output from the web application and the machine learning models. <ul style="list-style-type: none"> <li>- The web application input should cover the audio track while the output should consist of the results from the models.</li> <li>- The machine learning input should consist of an ID and audio track while the output should consist of the ID and the results from the models.</li> </ul> <p>The cloud system also needs to track the new database entries and signal the models to train on the new data to improve accuracy automatically.</p>
Machine Learning	The artificial intelligence models need to cover beat recognition, chord recognition, and chord pattern recognition. The models also need preprocessing to handle the input.

*Table 2 - Components*

## Delimitation

Based on information from the client regarding their project specifications, we do not have the responsibility of developing extensive security mechanisms or routines for our application. We also are not responsible for facilitating the editing or QA of the AI-generated compositions beyond providing simple JSON-based outputs.

## Domain Knowledge

When creating a model to analyze features such as beats or chords, it is inevitable for music theory to somewhat seep into the project. As such, this also becomes part of its domain to a certain extent. It is important to note that the model should only analyze beats, chords, and if possible chord patterns in music. Further elements of musical theory, such as scales, are outside this domain and not relevant for our machine learning model.

For the front-end, we are creating a simple web page where users can pass in a youtube URL, and receive the parsed output from our model in a simple JSON-compatible format. Our primary focus is on the model itself, as well as the storage and transport of the resulting data to the web app and cloud database, and not on how we display this data to the end-user. As the app is meant to be used internally by EC-Play employees, an expansive editing interface is not within the scope of this thesis as they already have a solution for this.

*Table 3* is an overview of the domains and describes how we are going to use each of them.

Domain	Description
AI	Music analysis.
Web	Application to send songs for analysis and present the results.
Cloud	Communication between web application and AI.
Databases	Store results of the analysis.

*Table 3 - Domains*

## Project organization

### Responsibilities and roles

Figure 1 is a diagram of how the roles are divided. The connection between the Scrum Master and the Product Owner represents communication between the client and the Scrum Team.

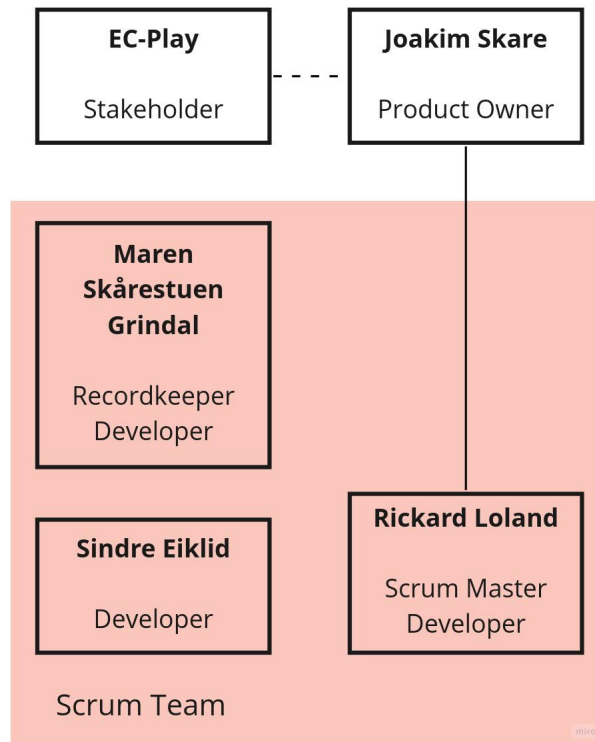


Figure 1 - Project Organization Diagram

Table 4 describes what is expected of each role.

Role	Tasks
Product Owner	Represents EC-Play's interests in the project, concretizes their vision, and is our main point of contact with the stakeholder.
Scrum Master	Ensures other team members understand their role and is responsible for mediation, tracks progress, and calls meetings when needed. Has the responsibility to implement process adjustments if the project lags behind schedule and is responsible for the communication between the Product Owner and the Scrum team
Developers	Perform their work as agreed, according to the rules set out in this document and industry standards.
Recordkeeper	Writes meeting minutes.

Table 4 - Roles

# Planning

## Development process

For our development process, we had to consider the requirements of the project. The specifications we received from EC-Play are somewhat fluid, for example in that EC-Play does not demand success in identifying chord patterns to high precision, compared to identifying beats. How good our network needs to be, and how many things it must detect, is subject to change during the course of the project. Similarly, different ways for the user to upload music, such as directly through URL, has also been discussed.

During the initial phase of the project we were discussing which development method to use, we narrowed down our options to the more linear waterfall methodology and the agile SCRUM framework. Due to the nature of our project, we valued continuous feedback from the stakeholders on our application and wanted to ensure that we could adjust our plans based on their feedback. In this way, the waterfall model did not suit our needs [2]. This is an area where SCRUM excels [3], and we also have previous experience working with the SCRUM development process. Therefore it became clear to us that this would be the best framework for our project.

Due to the scope and duration of the project, our experience made us confident that a two-week sprint duration is the best fit for our SCRUM process. We have previously worked with one-week sprints and found that organizational work related to the sprint process involved too much overhead with such short intervals. At the end of each sprint, we will have meetings with the client and discuss the progress during the sprint, and our plan for the following one.

## Meetings

We plan to arrange weekly meetings internally in the group and hold sprint review meetings every other week. We will also have weekly meetings with the client, as well as with our supervisor. We might change the frequencies of the meetings to biweekly during the development process.

## Product backlog

We will use GitHub Projects for product backlog since it can be used with smart commits and will be easily organized within our repository. We decided to have a project called Product Backlog with a basic kanban structure with review columns added. What each column refers to is shown in *Table 5*. The columns are automated for ease of use.

Board	Description	Automation
To do	Issues that haven't been started on	New issues will be automatically added here
In progress	Issues that are being worked on	Reopened issues will be automatically added here
Review in progress	Issues that are under review by a team member	Issues pending for approval will be added here
Reviewer approved	Issues that have been approved by the reviewer	Issues approved will be added here
Done	Issues that are completed	Closed issues will be automatically added here

*Table 5 - Columns in product backlog*

To organize the kanban board, we have decided to use labels. This allows us to easily filter the kanban board and will make it easier for us to keep track of everything. The labels are described in *Table 6*.

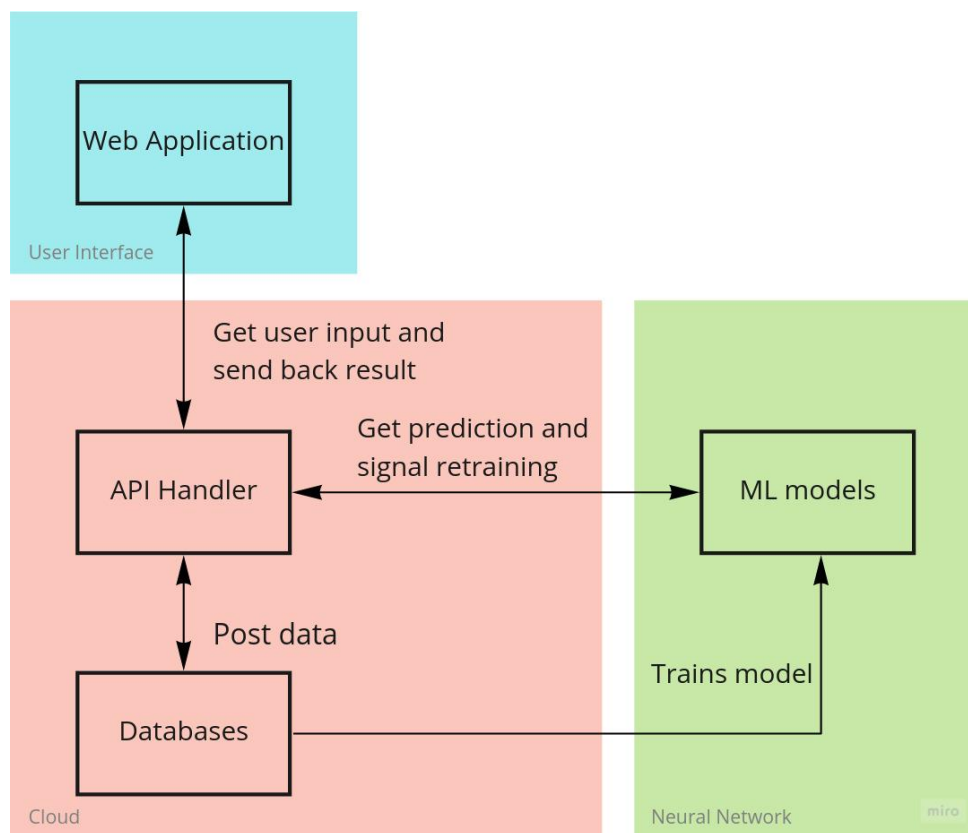
Label	Description
ui	Relates to front-end user interface
nn	Relates to the neural network
cloud	Relates to our cloud storage solution
report	Reflects all written requirements within this project
competence acquisition	Topics we need to research
bug	A problem that needs to be worked on
wontfix	Issues that will not be fixed within the deadline
high priority	Must be solved for the project MVP
medium priority	Should ideally be implemented
low priority	Not very important for the project

*Table 6 - Labels in product backlog*

## Milestones

We divide our project into three modules as shown in *Figure 2*. The first part is the user interface, the second part consists of our cloud architecture and the database, and the third and final part is our neural networks. This division comes naturally, as our application will be full-stack with front-end, back-end, and cloud computing elements.

By dividing our project into modules in this way, we also simplify the development process somewhat by allowing us to more easily work simultaneously on different modules without clashing. This is because each module can be kept strictly separate from the others during development, and combined only during later stages of the project with thorough integration testing.



*Figure 2 - Product sketch*

# Quality Assurance

## Routines

We will use trunk-based development to ensure that new code can be smoothly and painlessly integrated with our master branch.

New tasks should be added to the project backlog as an issue with relevant tags.

Commits must contain a relevant issue ID number or numbers, to more easily parse and follow each issue. The format of the commit message should look like this: [#number indicating issue if applicable] Short summary of changes

Commits should be performed regularly and not be too large.

Merging to master should only be done after the code review is finished. If the reviewer finds mistakes in the code, it will have to be communicated back to the developer and be addressed before a new code review.

Code should be properly documented through the use of comments before merging.

## Documentation, source code, and storage

Our focus for this project will be on properly following coding conventions in regards to security and quality assurance for the relevant tools and languages used during our project. To aid with this, we are making extensive use of linting tools for our code during the process.

## GitHub Workflows

Workflows is a free GitHub tool that allows us to run jobs in a virtual machine on every commit, to ensure that our codebase is running properly and new code is compatible with the existing codebase. This can help us catch unforeseen issues with our work as it potentially impacts other modules or environments. This tool can for example perform linting checks, which allows us to continuously ensure that our code is up to standards.

Additionally, we plan to utilize GitHub Workflows to run our tests on each commit, to ensure that testing is consistent and frequent, rather than simply write-and-forget.

## Testing

Where possible, we plan to perform thorough testing of our project code. We plan for unit testing, integration testing and regression testing to all be a part of our process, and implemented appropriately using the tools available for each language.

Code reviews will be run regularly, both to ensure that code is up to standards and to ensure that each member of the group is knowledgeable and comfortable with the full codebase, and not just their contribution.

## Documentation and storage

Our code is stored in a Git repo hosted by the client. Reports and documentation are hosted on Google Drive.



We plan to store the composition data for each song in a fresh database, hosted by Google Firebase. This creates a separation between existing solutions used by the client and the results from our newly developed model. This has several advantages, including security-related (no need to worry about tampering with client's existing data) as well as organizational as we can be sure what data has been generated through our model, and what has been meticulously handcrafted.

### Risk assessment

Table 7 defines the risks we think are relevant to this project. We divided the risks into categories to make the table more readable.

Nr	Risk	Category	Likelihood	Ramification
1	The training dataset contains mistakes (chords), leading to a poorly performing neural network.	Product	Likely	Severe
2	The training dataset is not big enough at ~400 songs, leading the model to perform poorly on new music.	Product	Very likely	Moderate
3	Training model on new data from users could create networking bottlenecks, leading potentially to app timeouts or similar.	Product	Likely	Moderate
4	API used for downloading songs from youtube ceases to work.	Product	Likely	Severe
5	Audio files have different frequencies and would need to be changed which could lead to poor quality.	Product	Likely	Slight
6	Some of the editing tools for the user to correct AI-generated results will not be implementable due to scope.	Product	Very likely	Slight
7	Lacking knowledge of musical theory leads to difficulties designing and evaluating the results of our model.	Product	Likely	Moderate
8	Server downtime.	Product	Unlikely	Severe
9	Drastic delay in the development process.	Product	Unlikely	Moderate
10	A team member is unable to work on the project for a long time due to illness, quitting, or similar.	Team dynamics	Unlikely	Severe
11	The scrum master is sick or unavailable and	Team	Unlikely	Slight

	is not able to do his tasks.	dynamics		
12	Conflict in the group.	Team dynamics	Unlikely	Slight

*Table 7 - Risk Assessment*

Risk evaluation matrix

The columns of *Table 8* describe the magnitude, while the rows describe the likelihood.

	Slight	Moderate	Severe
Unlikely	11, 12	9	8, 10
Likely	5	3, 4, 7	1
Very likely	6	2	

*Table 8 - Risk evaluation matrix*

Planned controls for handling risks

While we do not have controls for all risks, the selected risks shown in *Table 9* have various controls that can be applied to reduce the magnitude of impact and/or likelihood of occurrence. The selected risks are those in the red zones of our matrix or without a clear solution.

Nr	Control
1	Find another public dataset or use an algorithm to get the chords instead of a Neural Network.
2	Similar to nr. 1, we can find and use public datasets to train our model, supplementing the existing dataset.
3	We can perform training of new models at set times during the day where traffic is likely to be low, based on network traffic logs.
4	This will most likely happen in the future because APIs change and have downtime. There is no real way to avoid these interruptions in service as the client wants to use youtube-URLs to pass songs in, and they must be managed during maintenance of the codebase. If it occurs during the timeframe of our project, we will have to find a new API or potentially allow users to upload mp3 files to skip the external APIs altogether.
7	We do not have any experience with music theory, but EC-Play has stated several times that they can help us with this aspect. If needed, we do not think it would be too hard to research this on our own, as the knowledge we need seems basic.
9	We can fall back to the MVP and focus on the important aspects of getting the

	product in a usable state.
10	To minimize the impact of such an event, we will utilize code reviews and sprint meetings to make sure each team member familiarizes themselves with the work done by the rest of the team so that in a crisis they can take over the work of other members. Other than sticking to a 30 hour/person schedule and doing our best to finish work ASAP, there is little that can be done to make up for the workload lost.
11	The member with only one role will take over the responsibilities until the Scrum Master is able to do their tasks again.

*Table 9 - Planned controls for handling risks*

## Tools

The tools we are planning to use during the project are shown in *Table 10*.

Tool	Purpose	Usage
<a href="#">GitHub</a>	Handle hosting and distribution of code	Project organization
<a href="#">Visual Studio Code</a>	IDE for development in Golang, PHP, and Python	Front-end, back-end
<a href="#">Google Drive</a>	Documentation storage	Project organization
<a href="#">Google Sheets</a>	Tool for designing the Gantt chart	Project organization
<a href="#">Google Docs</a>	Cooperative writing and editing	Project report
<a href="#">Toggl</a>	Time logging tool	Project organization
GitHub issue tracker	Git tool for scrum-organization and general issue tracking	Project organization
<a href="#">GitHub workflows</a>	Quality assurance through linting and running tests	Quality Assurance
<a href="#">Miro</a>	Online whiteboard for designing models and wireframes	Design
<a href="#">Discord</a>	Online communication app	Project organization
<a href="#">Google Firebase</a>	Cloud storage	Database

*Table 10 - Tools*

## Progress plan

As shown in *Figure 3*, the Gantt chart is divided into three stages with a red line representing the deadlines. Each sprint is divided into two weeks and each week contains only workdays. The first stage is an accurate description of how we have worked with the project until now, while the other stages are only estimates. This will be updated as we continue working and will be accurate in the final report. We decided to split up the more important tasks to make it easier for us to keep track of the development process.

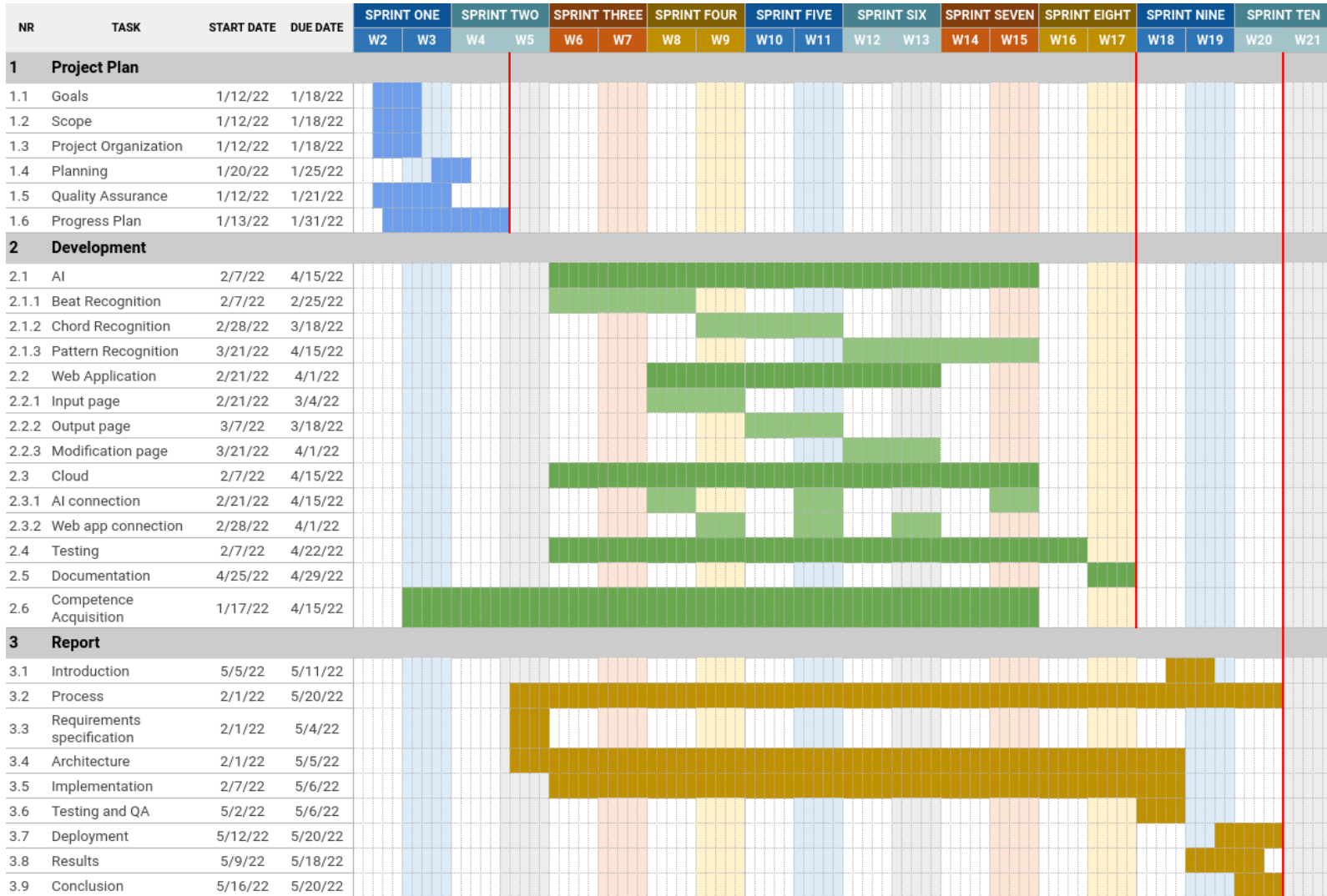


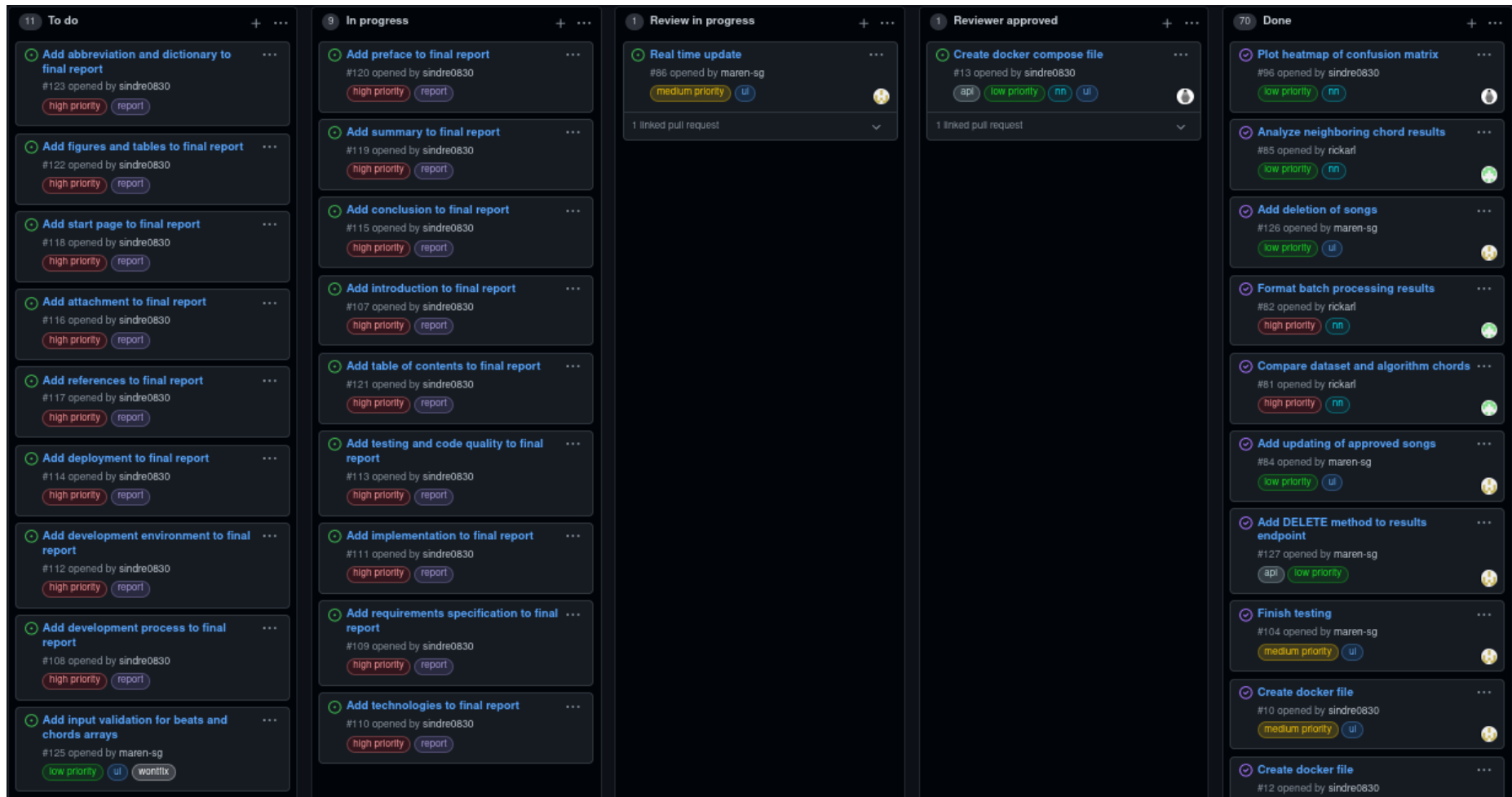
Figure 3 - Gantt Chart

## References

The references are written in the Vancouver style and we have used [NTNU's documentation](#) as a template.

1. EC-Play. Everyone Can Play [Internet]. Asker: EC-Play; 2020 [cited 2022 January 13]. Available from: <https://www.ec-play.org/>
2. Kienitz P. The pros and cons of Waterfall Software Development [Internet]. England and Wales; 2017 Feb 27 [cited 2022 Jan 18]. Available from: <https://www.dcsf.com/pros-cons-waterfall-software-development/>
3. Sommerville I. Agile software development. In: Hirsch M, editor. Software Engineering. 10th ed. England: Pearson; 2019. p. 77-78.
4. de Oliveira de Souza MS, de Souza Moura PN, Briot J-P. Music Tempo Estimation via Neural Networks -- A Comparative Analysis. ArXiv: 2107.09208v1 [Preprint]. 2021 [cited 2022 Jan 25]. Available from: <https://arxiv.org/abs/2107.09208>
5. Schreiber H, Müller M. A Single-Step Approach to Musical Tempo Estimation Using a Convolutional Neural Network. In: 19th International Society for Music Information Retrieval Conference; 2018 Sep 23-27; Paris, France [cited 2022 Jan 25]. Available from: [https://www.researchgate.net/publication/328028453\\_A\\_Single-Step\\_Approach\\_to\\_Musical\\_Tempo\\_Estimation\\_Using\\_a\\_Convolutional\\_Neural\\_Network](https://www.researchgate.net/publication/328028453_A_Single-Step_Approach_to_Musical_Tempo_Estimation_Using_a_Convolutional_Neural_Network)

## B Kanban Board



Link to dynamic kanban board (changes during development process):

<https://github.com/sindre0830/Neural-Network-for-Recognizing-Features-in-Music/projects/1>

## C Group Rules

Our group has decided on these rules:

- Scrum Master will keep in contact with the product owner and schedule/cancel meetings.
- Each team member will work approximately 30 hours per week. <sup>1</sup>
- Have a physical meeting at least once each sprint, but preferably once each week. <sup>2</sup>
- Hours should be properly logged for auditing and reporting.
- Each member has an obligation to show for all planned meetings, or else give advance notice that they will miss the meeting.
- The recordkeeper should keep a record of all scheduled meetings.
- Any costs incurred during the project not covered by the client should be split evenly within the group.

If these rules aren't followed the following measures will be taken (in the order stated):

1. Group discussion with all members discussing the rule(s) broken and attempting to solve the issue. These discussions must be logged for auditing.
2. Discussion involving group advisor with attempts at mediation.
3. Through unanimous voting from the other members, the person will be excluded from the project.

*1 This doesn't apply to the planning phase of the project as there is a limited amount of work that can be done between each meeting with the advisor/client, while the delimitations of the process are still being defined.*

*2 This will be delayed for the first few weeks of the project where some members aren't in Gjøvik.*



## D Standard Agreement



Norges teknisk-naturvitenskapelige universitet

*Fastsatt av prorektor for utdanning 10.12.2020*

### **STANDARDAVTALE**

#### **om utføring av studentoppgave i samarbeid med ekstern virksomhet**

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

#### **Forklaring av begrep**

##### **Opphavsrett**

Er den rett som den som skaper et åndsverk har til å fremstille eksemplarer av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

##### **Eiendomsrett til resultater**

Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

##### **Bruksrett til resultater**

Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

##### **Prosjektbakgrunn**

Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

##### **Utsatt offentliggjøring**

Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

### 1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt:
Veileder ved NTNU: <b>Ali Shariq Imran</b> e-post og tlf:
Ekstern virksomhet: <b>EC-Play AS</b> Ekstern virksomhet sin kontaktperson, e-post og tlf.: <small>Kristoffer Skare, kristoffer.skare@live.com, 99563000</small>
Student: <b>Rickard Loland</b> Fødselsdato: <b>11-08-1989</b>
Student: <b>Sindre Eiklid</b> Fødselsdato: <b>30-08-2000</b>
Student: <b>Maren Skårestuen Grindal</b> Fødselsdato: <b>27-06-2000</b>

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

### 2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	X
Prosjektoppgave	
Annen oppgave	

Startdato: <b>12-01-22</b>
Sluttdato: <b>20-05-22</b>

Opggavens arbeidstittel er:  
**Neural Network for Recognizing Features in Music**

<sup>1</sup> Dersom flere studenter skriver oppgave i fellesskap, kan alle føres opp her. Rettigheter ligger da i fellesskap mellom studentene. Dersom ekstern virksomhet i stedet ønsker at det skal inngås egen avtale med hver enkelt student, gjøres dette.

2 NTNU 10.12.2020

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

### **3. Ekstern virksomhet sine plikter**

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven:

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

### **4. Studentens rettigheter**

Studenten har opphavsrett til oppgaven<sup>2</sup>. Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs

institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

### 5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten

<sup>2</sup>Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

3 NTNU 10.12.2020

skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

#### Alternativ a) (sett kryss) Hovedregel

<input type="checkbox"/>	Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven
--------------------------	--

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

#### Alternativ b) (sett kryss) Unntak

<input checked="" type="checkbox"/>	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt
-------------------------------------	---

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene:

EC-Play AS vil videreutvikle programvaren, og ha eiendomsrett til denne. Studentene har rettighet til oppgaven, og kan bruke eksempler fra programvaren som er utviklet i denne.

### 6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

### 7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

### 8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

<input checked="" type="checkbox"/>	Oppgaven skal være offentlig
-------------------------------------	------------------------------

4 NTNU 10.12.2020

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss Sett dato

<input type="checkbox"/>	ett år	
<input type="checkbox"/>	to år	
<input type="checkbox"/>	tre år	

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

### 9. Generelt

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

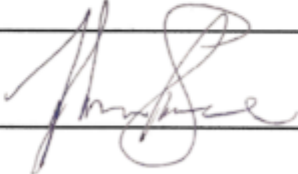
Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

5 NTNU 10.12.2020

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

#### Signaturer:

Instituttleder: Dato:		
Veileder ved NTNU: Dato:		
Ekstern virksomhet: Dato: 25.01.2022		Joakim Skare Daglig leder
Student: Richard Laland Dato: 20.07.22		
Student: Sindre Eiklid Dato: 24.01.22		
Student: Maren Skarstein Grinddal Dato: 24.01.22		

6 NTNU 10.12.2020

# E Status Report

## Status Report 21.02.2022

### API

We have planned all API endpoints, and are currently developing them. Out of three endpoints, two are almost done. The rest is estimated to be finished this week.

### Web Application

We have the wireframes ready and have done user testing to get feedback. We have also gotten feedback from the client on these designs and improved them based on the response. We will start creating the web application by next week (Week 9).

### NN

We decided to change our original plan written in the project plan document and went with algorithms first to have something to compare our models with, as well as allow us to set up and test our full pipeline as early as possible. We are implementing algorithm solutions for beat and chord recognition, and then designing and implementing our pattern recognition NN. Once this is completed, we plan to go back and design NN solutions, using the client's dataset as ground truth and comparing the results with our algorithms.

So far, a lot of our work on this area has been done in planning and research/competence acquisition. Additionally, the general framework for our system has been designed and we have implemented preprocessing steps such as converting youtube-files to WAV extension. We also have ensured that the library we are using for downloading videos works, though this has not been added to our API yet. We have completed and tested the implementation of beat analysis algorithm and started work on the chord recognition algorithm.

### Report

We have made an outline of the final report, and have started writing about user testing. We have kept notes about our process for the other parts of the project, but nothing more has been added to the report as of 21.02.

# Status Report 25.04.2022

## API

All endpoints are completed, only minor adjustments and finishing the main file is needed.

## Web Application

The main features of the web application have been developed. The application has been tied together with routing. Some improvements regarding fetching data are needed: caching and real-time update of the data.

## NN

In accordance with our changed plans as of status report 1, we implemented algorithm solutions for chord and beat accuracy, in order to have a basis of comparison for our Neural Network performance, and to implement our general pipeline. At this point, the algorithms are finished, and work has also finished performing analysis and evaluation of the results.

We have also worked on the neural network solution. We decided to focus our efforts in this field on the chord identification portion of the task, as this seemed the area where neural network was the most natural implementation, and where we judged other solutions would not have much success.

In order to find good model parameters, a randomized grid search approach was implemented in order to evaluate the impact of various parameters and find a solid combination. After running this for a couple of days, we ended up with an accuracy of 82% with little overfitting.

## Deployment

Now that our solution is nearing completion, we are starting to look closer at our deployment environment, and bringing it into focus.

## Report

A lot more work has gone into the report. We have primarily focused on two tasks here - first, planning the general contents of the report and outlining the contents of each section, whether we have something written for it or not. Second, writing about our actual implementation process and the work we have done directly. A lot of supporting information such as the process and the environment is still missing, as are the results.

Our progress on the report has been going well, and we have written about 40 pages, albeit a lot of revision is needed.



## F Meeting Notices

Messages sent as email to client and adviser with the status reports attached, see attachment [E](#).

### Meeting Notice - Client

Attendees	Maren Skårestuen Grindal Rickard Loland Sindre Eiklid Kristoffer Skare Joakim Skare
Date and time	24.02.2022, 10:00
Place	Discord (Digital)
Subject of discussion	Status Report 1 (See attachment)

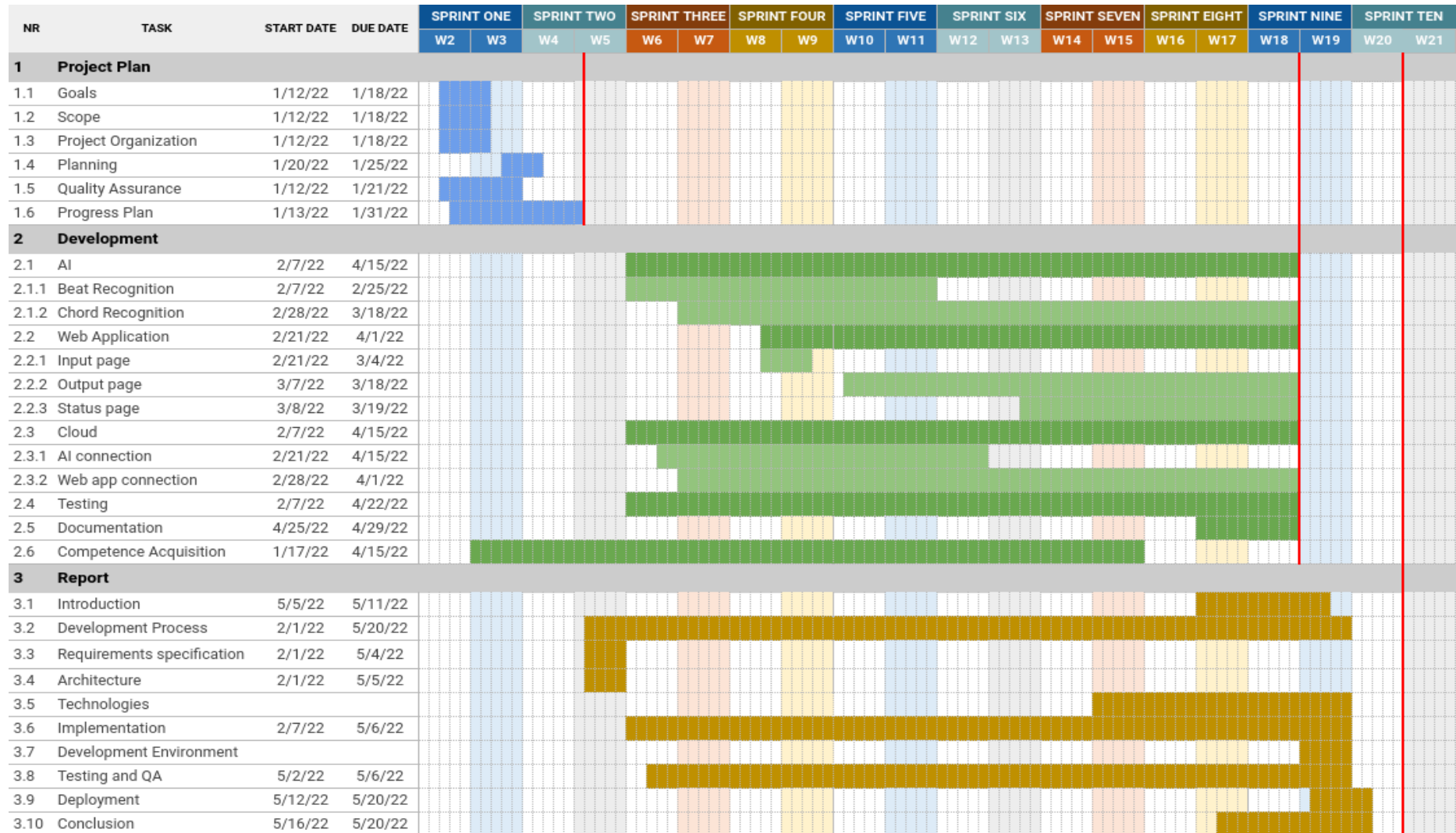
Attendees	Maren Skårestuen Grindal Rickard Loland Sindre Eiklid Kristoffer Skare Joakim Skare
Date and time	26.04.2022, 10:00
Place	Discord (Digital)
Subject of discussion	Status Report 2 (See attachment)

## Meeting Notice - Adviser

Attendees	Maren Skårestuen Grindal Rickard Loland Sindre Eiklid Ali Shariq Imran
Date and time	25.02.2022, 10:00
Place	Teams (Digital)
Subject of discussion	Status Report 1 (See attachment)

Attendees	Maren Skårestuen Grindal Rickard Loland Sindre Eiklid Ali Shariq Imran
Date and time	29.04.2022, 10:00
Place	Teams (Digital)
Subject of discussion	Status Report 2 (See attachment)

# G Gantt Chart



## H Random Search Results - Raw Data

In total, 70 models were run over four iterations where each iteration was sorted by accuracy. The best model is from the first iteration.

### First iteration

Model #	Parameters
1	val_accuracy: 0.8292601108551025   val_loss: 0.6497218012809753   initial_filter: 64   conv_layer_1: 1   conv_filter_1: 32   conv_layer_2: 2   conv_filter_2: 64   conv_layer_3: 2   conv_filter_3: 16   dense_layer_1: 2   dense_units_1: 256   dense_layer_2: 0   dense_units_2: 256   dense_layer_3: 0   dense_units_3: 128   regularizer: 0.0005
2	val_accuracy: 0.8218176960945129   val_loss: 0.7915952205657959   initial_filter: 128   conv_layer_1: 2   conv_filter_1: 128   conv_layer_2: 1   conv_filter_2: 256   conv_layer_3: 0   conv_filter_3: 256   dense_layer_1: 0   dense_units_1: 256   dense_layer_2: 2   dense_units_2: 256   dense_layer_3: 0   dense_units_3: 32   regularizer: 0.0005
3	val_accuracy: 0.8216835856437683   val_loss: 0.7047683596611023   initial_filter: 32   conv_layer_1: 2   conv_filter_1: 256   conv_layer_2: 1   conv_filter_2: 64   conv_layer_3: 2   conv_filter_3: 128   dense_layer_1: 0   dense_units_1: 64   dense_layer_2: 1   dense_units_2: 128   dense_layer_3: 1   dense_units_3: 256   regularizer: 0.0001
4	val_accuracy: 0.8202085494995117   val_loss: 0.6928550004959106   initial_filter: 128   conv_layer_1: 2   conv_filter_1: 32   conv_layer_2: 0   conv_filter_2: 256   conv_layer_3: 1   conv_filter_3: 256   dense_layer_1: 0   dense_units_1: 128   dense_layer_2: 1   dense_units_2: 256   dense_layer_3: 1   dense_units_3: 32   regularizer: 0.0005
5	val_accuracy: 0.819169282913208   val_loss: 0.6805905699729919   initial_filter: 16   conv_layer_1: 2   conv_filter_1: 16   conv_layer_2: 0   conv_filter_2: 32   conv_layer_3: 1   conv_filter_3: 32   dense_layer_1: 2   dense_units_1: 256   dense_layer_2: 1   dense_units_2: 32   dense_layer_3: 1   dense_units_3: 256   regularizer: 0.005
6	val_accuracy: 0.8126655220985413   val_loss: 0.7009409070014954   initial_filter: 256   conv_layer_1: 0   conv_filter_1: 256   conv_layer_2: 2   conv_filter_2: 32   conv_layer_3: 0   conv_filter_3: 64   dense_layer_1: 2   dense_units_1: 128   dense_layer_2: 0   dense_units_2: 32   dense_layer_3: 0   dense_units_3: 32   regularizer: 0.0001
7	val_accuracy: 0.8088772892951965   val_loss: 0.7361435294151306   initial_filter: 128   conv_layer_1: 2   conv_filter_1: 32   conv_layer_2: 0   conv_filter_2: 64   conv_layer_3: 2   conv_filter_3: 128   dense_layer_1: 0   dense_units_1: 32   dense_layer_2: 1   dense_units_2: 128   dense_layer_3: 2   dense_units_3: 128   regularizer: 0.005

8	val_accuracy: 0.8075363039970398   val_loss: 0.8547020554542542   initial_filter: 64   conv_layer_1: 0   conv_filter_1: 256   conv_layer_2: 0   conv_filter_2: 128   conv_layer_3: 1   conv_filter_3: 256   dense_layer_1: 2   dense_units_1: 256   dense_layer_2: 0   dense_units_2: 256   dense_layer_3: 1   dense_units_3: 64   regularizer: 0.001
9	val_accuracy: 0.8075027465820312   val_loss: 0.74879390001297   initial_filter: 128   conv_layer_1: 1   conv_filter_1: 64   conv_layer_2: 2   conv_filter_2: 64   conv_layer_3: 2   conv_filter_3: 64   dense_layer_1: 2   dense_units_1: 128   dense_layer_2: 1   dense_units_2: 256   dense_layer_3: 2   dense_units_3: 32   regularizer: 0.001
10	val_accuracy: 0.8043514490127563   val_loss: 0.7247094511985779   initial_filter: 16   conv_layer_1: 2   conv_filter_1: 16   conv_layer_2: 2   conv_filter_2: 128   conv_layer_3: 2   conv_filter_3: 16   dense_layer_1: 2   dense_units_1: 128   dense_layer_2: 0   dense_units_2: 32   dense_layer_3: 2   dense_units_3: 32   regularizer: 0.001
11	val_accuracy: 0.8027758002281189   val_loss: 0.8015375137329102   initial_filter: 16   conv_layer_1: 1   conv_filter_1: 128   conv_layer_2: 0   conv_filter_2: 256   conv_layer_3: 2   conv_filter_3: 64   dense_layer_1: 1   dense_units_1: 128   dense_layer_2: 1   dense_units_2: 64   dense_layer_3: 0   dense_units_3: 32   regularizer: 0.005
12	val_accuracy: 0.8009990453720093   val_loss: 0.7394174337387085   initial_filter: 256   conv_layer_1: 2   conv_filter_1: 256   conv_layer_2: 1   conv_filter_2: 64   conv_layer_3: 0   conv_filter_3: 256   dense_layer_1: 0   dense_units_1: 256   dense_layer_2: 0   dense_units_2: 256   dense_layer_3: 2   dense_units_3: 64   regularizer: 0.0005
13	val_accuracy: 0.7989540100097656   val_loss: 0.8501062989234924   initial_filter: 128   conv_layer_1: 0   conv_filter_1: 256   conv_layer_2: 0   conv_filter_2: 128   conv_layer_3: 1   conv_filter_3: 128   dense_layer_1: 0   dense_units_1: 64   dense_layer_2: 2   dense_units_2: 128   dense_layer_3: 1   dense_units_3: 32   regularizer: 0.001
14	val_accuracy: 0.7964397072792053   val_loss: 0.6908913850784302   initial_filter: 64   conv_layer_1: 2   conv_filter_1: 32   conv_layer_2: 2   conv_filter_2: 32   conv_layer_3: 2   conv_filter_3: 64   dense_layer_1: 1   dense_units_1: 32   dense_layer_2: 0   dense_units_2: 128   dense_layer_3: 0   dense_units_3: 256   regularizer: 0.0005
15	val_accuracy: 0.7947299480438232   val_loss: 0.804892361164093   initial_filter: 256   conv_layer_1: 0   conv_filter_1: 16   conv_layer_2: 0   conv_filter_2: 128   conv_layer_3: 0   conv_filter_3: 256   dense_layer_1: 2   dense_units_1: 256   dense_layer_2: 2   dense_units_2: 256   dense_layer_3: 0   dense_units_3: 128   regularizer: 0.005
16	val_accuracy: 0.7871869802474976   val_loss: 0.7791540622711182   initial_filter: 64   conv_layer_1: 2   conv_filter_1: 16   conv_layer_2: 1   conv_filter_2: 256   conv_layer_3: 1   conv_filter_3: 256   dense_layer_1: 2   dense_units_1: 32   dense_layer_2: 1   dense_units_2: 256   dense_layer_3: 2   dense_units_3: 64   regularizer: 0.0005

17	val_accuracy: 0.7867511510848999   val_loss: 0.7888417840003967   initial_filter: 256   conv_layer_1: 0   conv_filter_1: 256   conv_layer_2: 0   conv_filter_2: 64   conv_layer_3: 2   conv_filter_3: 64   dense_layer_1: 2   dense_units_1: 32   dense_layer_2: 1   dense_units_2: 32   dense_layer_3: 2   dense_units_3: 128   regularizer: 0.001
18	val_accuracy: 0.7809178829193115   val_loss: 0.8763098120689392   initial_filter: 64   conv_layer_1: 0   conv_filter_1: 64   conv_layer_2: 0   conv_filter_2: 64   conv_layer_3: 1   conv_filter_3: 256   dense_layer_1: 0   dense_units_1: 64   dense_layer_2: 2   dense_units_2: 32   dense_layer_3: 0   dense_units_3: 128   regularizer: 0.0005
19	val_accuracy: 0.7669047117233276   val_loss: 0.7597916126251221   initial_filter: 16   conv_layer_1: 1   conv_filter_1: 64   conv_layer_2: 2   conv_filter_2: 16   conv_layer_3: 0   conv_filter_3: 32   dense_layer_1: 0   dense_units_1: 256   dense_layer_2: 0   dense_units_2: 128   dense_layer_3: 0   dense_units_3: 64   regularizer: 0.005
20	val_accuracy: 0.7666700482368469   val_loss: 0.7683956027030945   initial_filter: 32   conv_layer_1: 2   conv_filter_1: 16   conv_layer_2: 1   conv_filter_2: 16   conv_layer_3: 2   conv_filter_3: 16   dense_layer_1: 0   dense_units_1: 64   dense_layer_2: 1   dense_units_2: 32   dense_layer_3: 1   dense_units_3: 128   regularizer: 0.0001

## Second iteration

Model #	Parameters
1	val_accuracy: 0.8197056651115417   val_loss: 0.7174391150474548   initial_filter: 32   conv_layer_1: 2   conv_filter_1: 128   conv_layer_2: 0   conv_filter_2: 32   conv_layer_3: 1   conv_filter_3: 32   dense_layer_1: 2   dense_units_1: 256   dense_layer_2: 0   dense_units_2: 64   dense_layer_3: 1   dense_units_3: 256   regularizer: 0.0005
2	val_accuracy: 0.8150122165679932   val_loss: 0.7279840111732483   initial_filter: 16   conv_layer_1: 1   conv_filter_1: 64   conv_layer_2: 2   conv_filter_2: 64   conv_layer_3: 1   conv_filter_3: 32   dense_layer_1: 0   dense_units_1: 128   dense_layer_2: 1   dense_units_2: 256   dense_layer_3: 2   dense_units_3: 32   regularizer: 0.0005
3	val_accuracy: 0.8142076730728149   val_loss: 0.7118219137191772   initial_filter: 32   conv_layer_1: 2   conv_filter_1: 16   conv_layer_2: 0   conv_filter_2: 16   conv_layer_3: 1   conv_filter_3: 64   dense_layer_1: 1   dense_units_1: 128   dense_layer_2: 0   dense_units_2: 32   dense_layer_3: 1   dense_units_3: 256   regularizer: 0.0001
4	val_accuracy: 0.8122967481613159   val_loss: 0.7611275911331177   initial_filter: 32   conv_layer_1: 0   conv_filter_1: 128   conv_layer_2: 1   conv_filter_2: 128   conv_layer_3: 2   conv_filter_3: 128   dense_layer_1: 0   dense_units_1: 32   dense_layer_2: 0   dense_units_2: 32   dense_layer_3: 2   dense_units_3: 128   regularizer: 0.005

5	val_accuracy: 0.8102182149887085   val_loss: 0.7144971489906311   initial_filter: 16   conv_layer_1: 2   conv_filter_1: 16   conv_layer_2: 0   conv_filter_2: 256   conv_layer_3: 2   conv_filter_3: 32   dense_layer_1: 1   dense_units_1: 256   dense_layer_2: 0   dense_units_2: 128   dense_layer_3: 0   dense_units_3: 256   regularizer: 0.0005
6	val_accuracy: 0.806698203086853   val_loss: 0.7357587218284607   initial_filter: 256   conv_layer_1: 0   conv_filter_1: 128   conv_layer_2: 1   conv_filter_2: 16   conv_layer_3: 1   conv_filter_3: 32   dense_layer_1: 2   dense_units_1: 128   dense_layer_2: 2   dense_units_2: 64   dense_layer_3: 2   dense_units_3: 128   regularizer: 0.005
7	val_accuracy: 0.8060277104377747   val_loss: 0.7321142554283142   initial_filter: 32   conv_layer_1: 0   conv_filter_1: 32   conv_layer_2: 2   conv_filter_2: 128   conv_layer_3: 0   conv_filter_3: 256   dense_layer_1: 1   dense_units_1: 128   dense_layer_2: 2   dense_units_2: 256   dense_layer_3: 1   dense_units_3: 256   regularizer: 0.0001
8	val_accuracy: 0.801535427570343   val_loss: 0.7104081511497498   initial_filter: 128   conv_layer_1: 2   conv_filter_1: 16   conv_layer_2: 0   conv_filter_2: 256   conv_layer_3: 1   conv_filter_3: 256   dense_layer_1: 1   dense_units_1: 64   dense_layer_2: 2   dense_units_2: 128   dense_layer_3: 1   dense_units_3: 128   regularizer: 0.0001
9	val_accuracy: 0.7991887331008911   val_loss: 0.8399752378463745   initial_filter: 16   conv_layer_1: 2   conv_filter_1: 256   conv_layer_2: 1   conv_filter_2: 256   conv_layer_3: 1   conv_filter_3: 128   dense_layer_1: 0   dense_units_1: 256   dense_layer_2: 2   dense_units_2: 256   dense_layer_3: 2   dense_units_3: 32   regularizer: 0.001
10	val_accuracy: 0.792819082736969   val_loss: 0.7686797976493835   initial_filter: 256   conv_layer_1: 2   conv_filter_1: 32   conv_layer_2: 2   conv_filter_2: 256   conv_layer_3: 1   conv_filter_3: 128   dense_layer_1: 2   dense_units_1: 32   dense_layer_2: 0   dense_units_2: 64   dense_layer_3: 1   dense_units_3: 64   regularizer: 0.0005
11	val_accuracy: 0.7901371121406555   val_loss: 0.8707634210586548   initial_filter: 16   conv_layer_1: 1   conv_filter_1: 128   conv_layer_2: 0   conv_filter_2: 32   conv_layer_3: 1   conv_filter_3: 128   dense_layer_1: 2   dense_units_1: 256   dense_layer_2: 1   dense_units_2: 256   dense_layer_3: 2   dense_units_3: 64   regularizer: 0.005
12	val_accuracy: 0.7899024486541748   val_loss: 0.7642438411712646   initial_filter: 128   conv_layer_1: 2   conv_filter_1: 32   conv_layer_2: 2   conv_filter_2: 64   conv_layer_3: 2   conv_filter_3: 128   dense_layer_1: 1   dense_units_1: 32   dense_layer_2: 2   dense_units_2: 128   dense_layer_3: 2   dense_units_3: 32   regularizer: 0.0001
13	val_accuracy: 0.7898018956184387   val_loss: 0.7580240964889526   initial_filter: 16   conv_layer_1: 2   conv_filter_1: 32   conv_layer_2: 2   conv_filter_2: 64   conv_layer_3: 1   conv_filter_3: 128   dense_layer_1: 2   dense_units_1: 32   dense_layer_2: 0   dense_units_2: 64   dense_layer_3: 1   dense_units_3: 32   regularizer: 0.0001

14	val_accuracy: 0.786851704120636   val_loss: 0.7991734147071838   initial_filter: 32   conv_layer_1: 2   conv_filter_1: 32   conv_layer_2: 0   conv_filter_2: 128   conv_layer_3: 0   conv_filter_3: 128   dense_layer_1: 2   dense_units_1: 128   dense_layer_2: 0   dense_units_2: 32   dense_layer_3: 2   dense_units_3: 32   regularizer: 0.005
15	val_accuracy: 0.7856783866882324   val_loss: 0.7295673489570618   initial_filter: 64   conv_layer_1: 2   conv_filter_1: 256   conv_layer_2: 2   conv_filter_2: 32   conv_layer_3: 2   conv_filter_3: 16   dense_layer_1: 0   dense_units_1: 32   dense_layer_2: 2   dense_units_2: 64   dense_layer_3: 0   dense_units_3: 64   regularizer: 0.005
16	val_accuracy: 0.785007894039154   val_loss: 0.8131935000419617   initial_filter: 16   conv_layer_1: 1   conv_filter_1: 64   conv_layer_2: 2   conv_filter_2: 128   conv_layer_3: 0   conv_filter_3: 256   dense_layer_1: 1   dense_units_1: 64   dense_layer_2: 2   dense_units_2: 64   dense_layer_3: 0   dense_units_3: 64   regularizer: 0.005
17	val_accuracy: 0.7811525464057922   val_loss: 0.72502201795578   initial_filter: 256   conv_layer_1: 1   conv_filter_1: 64   conv_layer_2: 2   conv_filter_2: 32   conv_layer_3: 1   conv_filter_3: 32   dense_layer_1: 0   dense_units_1: 256   dense_layer_2: 0   dense_units_2: 256   dense_layer_3: 0   dense_units_3: 64   regularizer: 0.001
18	val_accuracy: 0.775688111782074   val_loss: 0.7885745167732239   initial_filter: 256   conv_layer_1: 1   conv_filter_1: 32   conv_layer_2: 2   conv_filter_2: 16   conv_layer_3: 2   conv_filter_3: 32   dense_layer_1: 1   dense_units_1: 32   dense_layer_2: 2   dense_units_2: 32   dense_layer_3: 2   dense_units_3: 256   regularizer: 0.005
19	val_accuracy: 0.7719668745994568   val_loss: 0.748751163482666   initial_filter: 256   conv_layer_1: 2   conv_filter_1: 32   conv_layer_2: 1   conv_filter_2: 16   conv_layer_3: 0   conv_filter_3: 64   dense_layer_1: 0   dense_units_1: 128   dense_layer_2: 0   dense_units_2: 64   dense_layer_3: 0   dense_units_3: 128   regularizer: 0.0005
20	val_accuracy: 0.7655637264251709   val_loss: 0.8058105111122131   initial_filter: 32   conv_layer_1: 2   conv_filter_1: 16   conv_layer_2: 1   conv_filter_2: 256   conv_layer_3: 0   conv_filter_3: 16   dense_layer_1: 0   dense_units_1: 128   dense_layer_2: 0   dense_units_2: 128   dense_layer_3: 2   dense_units_3: 32   regularizer: 0.0005

### Third iteration

Model #	Parameters
1	val_accuracy: 0.8209460377693176   val_loss: 0.6701213121414185   initial_filter: 256   conv_layer_1: 2   conv_filter_1: 16   conv_layer_2: 0   conv_filter_2: 16   conv_layer_3: 1   conv_filter_3: 256   dense_layer_1: 0   dense_units_1: 64   dense_layer_2: 1   dense_units_2: 256   dense_layer_3: 2   dense_units_3: 64   regularizer: 0.001
2	val_accuracy: 0.8193368911743164   val_loss: 0.7029942870140076   initial_filter: 32   conv_layer_1: 1   conv_filter_1: 128   conv_layer_2: 2



	conv_filter_2: 64   conv_layer_3: 0   conv_filter_3: 32   dense_layer_1: 1   dense_units_1: 256   dense_layer_2: 1   dense_units_2: 64   dense_layer_3: 2   dense_units_3: 128   regularizer: 0.0005
3	val_accuracy: 0.8149116635322571   val_loss: 0.7142208218574524   initial_filter: 64   conv_layer_1: 0   conv_filter_1: 256   conv_layer_2: 0   conv_filter_2: 64   conv_layer_3: 2   conv_filter_3: 32   dense_layer_1: 0   dense_units_1: 128   dense_layer_2: 1   dense_units_2: 256   dense_layer_3: 2   dense_units_3: 64   regularizer: 0.001
4	val_accuracy: 0.8058265447616577   val_loss: 0.900017261505127   initial_filter: 64   conv_layer_1: 0   conv_filter_1: 128   conv_layer_2: 1   conv_filter_2: 256   conv_layer_3: 0   conv_filter_3: 128   dense_layer_1: 2   dense_units_1: 256   dense_layer_2: 2   dense_units_2: 64   dense_layer_3: 2   dense_units_3: 128   regularizer: 0.001
5	val_accuracy: 0.8050554990768433   val_loss: 0.684587299823761   initial_filter: 32   conv_layer_1: 2   conv_filter_1: 16   conv_layer_2: 2   conv_filter_2: 16   conv_layer_3: 0   conv_filter_3: 32   dense_layer_1: 0   dense_units_1: 32   dense_layer_2: 1   dense_units_2: 128   dense_layer_3: 1   dense_units_3: 256   regularizer: 0.0005
6	val_accuracy: 0.8005967140197754   val_loss: 0.9978362917900085   initial_filter: 32   conv_layer_1: 0   conv_filter_1: 64   conv_layer_2: 0   conv_filter_2: 64   conv_layer_3: 1   conv_filter_3: 256   dense_layer_1: 1   dense_units_1: 256   dense_layer_2: 1   dense_units_2: 128   dense_layer_3: 2   dense_units_3: 64   regularizer: 0.005
7	val_accuracy: 0.7932884097099304   val_loss: 0.7681880593299866   initial_filter: 32   conv_layer_1: 2   conv_filter_1: 64   conv_layer_2: 0   conv_filter_2: 256   conv_layer_3: 1   conv_filter_3: 256   dense_layer_1: 1   dense_units_1: 32   dense_layer_2: 2   dense_units_2: 256   dense_layer_3: 1   dense_units_3: 32   regularizer: 0.005
8	val_accuracy: 0.782191812992096   val_loss: 0.7846401929855347   initial_filter: 128   conv_layer_1: 2   conv_filter_1: 32   conv_layer_2: 1   conv_filter_2: 64   conv_layer_3: 1   conv_filter_3: 64   dense_layer_1: 2   dense_units_1: 32   dense_layer_2: 0   dense_units_2: 256   dense_layer_3: 2   dense_units_3: 128   regularizer: 0.005
9	val_accuracy: 0.7809178829193115   val_loss: 0.7320871353149414   initial_filter: 32   conv_layer_1: 0   conv_filter_1: 256   conv_layer_2: 2   conv_filter_2: 16   conv_layer_3: 1   conv_filter_3: 32   dense_layer_1: 0   dense_units_1: 256   dense_layer_2: 1   dense_units_2: 32   dense_layer_3: 0   dense_units_3: 128   regularizer: 0.0001
10	val_accuracy: 0.747728705406189   val_loss: 0.9129321575164795   initial_filter: 32   conv_layer_1: 0   conv_filter_1: 32   conv_layer_2: 0   conv_filter_2: 32   conv_layer_3: 1   conv_filter_3: 128   dense_layer_1: 0   dense_units_1: 64   dense_layer_2: 0   dense_units_2: 64   dense_layer_3: 0   dense_units_3: 256   regularizer: 0.001

Fourth iteration

Model #	Parameters
1	val_accuracy: 0.8183311820030212   val_loss: 0.6953126192092896   initial_filter: 256   conv_layer_1: 0   conv_filter_1: 128   conv_layer_2: 0   conv_filter_2: 32   conv_layer_3: 2   conv_filter_3: 64   dense_layer_1: 2   dense_units_1: 128   dense_layer_2: 0   dense_units_2: 64   dense_layer_3: 0   dense_units_3: 256   regularizer: 0.0005
2	val_accuracy: 0.8179959058761597   val_loss: 0.6887810826301575   initial_filter: 32   conv_layer_1: 1   conv_filter_1: 32   conv_layer_2: 1   conv_filter_2: 16   conv_layer_3: 1   conv_filter_3: 128   dense_layer_1: 0   dense_units_1: 64   dense_layer_2: 2   dense_units_2: 256   dense_layer_3: 1   dense_units_3: 32   regularizer: 0.0001
3	val_accuracy: 0.812095582485199   val_loss: 0.8223552703857422   initial_filter: 128   conv_layer_1: 1   conv_filter_1: 256   conv_layer_2: 2   conv_filter_2: 256   conv_layer_3: 0   conv_filter_3: 256   dense_layer_1: 1   dense_units_1: 128   dense_layer_2: 0   dense_units_2: 256   dense_layer_3: 2   dense_units_3: 32   regularizer: 0.005
4	val_accuracy: 0.809950053691864   val_loss: 0.7144642472267151   initial_filter: 256   conv_layer_1: 0   conv_filter_1: 16   conv_layer_2: 1   conv_filter_2: 16   conv_layer_3: 1   conv_filter_3: 256   dense_layer_1: 0   dense_units_1: 128   dense_layer_2: 0   dense_units_2: 128   dense_layer_3: 2   dense_units_3: 64   regularizer: 0.0001
5	val_accuracy: 0.8067987561225891   val_loss: 0.7662034630775452   initial_filter: 32   conv_layer_1: 0   conv_filter_1: 256   conv_layer_2: 1   conv_filter_2: 64   conv_layer_3: 0   conv_filter_3: 256   dense_layer_1: 2   dense_units_1: 256   dense_layer_2: 2   dense_units_2: 128   dense_layer_3: 1   dense_units_3: 128   regularizer: 0.005
6	val_accuracy: 0.8038486242294312   val_loss: 0.7643226385116577   initial_filter: 64   conv_layer_1: 0   conv_filter_1: 16   conv_layer_2: 2   conv_filter_2: 256   conv_layer_3: 0   conv_filter_3: 256   dense_layer_1: 2   dense_units_1: 128   dense_layer_2: 0   dense_units_2: 256   dense_layer_3: 2   dense_units_3: 64   regularizer: 0.005
7	val_accuracy: 0.7996915578842163   val_loss: 0.9513117074966431   initial_filter: 128   conv_layer_1: 1   conv_filter_1: 128   conv_layer_2: 0   conv_filter_2: 256   conv_layer_3: 0   conv_filter_3: 32   dense_layer_1: 1   dense_units_1: 256   dense_layer_2: 0   dense_units_2: 64   dense_layer_3: 2   dense_units_3: 256   regularizer: 0.005
8	val_accuracy: 0.7987529039382935   val_loss: 0.7738701105117798   initial_filter: 128   conv_layer_1: 2   conv_filter_1: 256   conv_layer_2: 1   conv_filter_2: 256   conv_layer_3: 0   conv_filter_3: 128   dense_layer_1: 1   dense_units_1: 32   dense_layer_2: 1   dense_units_2: 64   dense_layer_3: 1   dense_units_3: 256   regularizer: 0.001
9	val_accuracy: 0.7978812456130981   val_loss: 0.7402510046958923   initial_filter: 128   conv_layer_1: 1   conv_filter_1: 32   conv_layer_2: 0   conv_filter_2: 32   conv_layer_3: 1   conv_filter_3: 64   dense_layer_1: 1   dense_units_1: 128   dense_layer_2: 0   dense_units_2: 64   dense_layer_3: 1   dense_units_3: 256   regularizer: 0.0005

10	val_accuracy: 0.7978141903877258   val_loss: 0.6967147588729858   initial_filter: 256   conv_layer_1: 0   conv_filter_1: 128   conv_layer_2: 2   conv_filter_2: 16   conv_layer_3: 1   conv_filter_3: 128   dense_layer_1: 1   dense_units_1: 32   dense_layer_2: 0   dense_units_2: 256   dense_layer_3: 2   dense_units_3: 128   regularizer: 0.0001
11	val_accuracy: 0.7977806925773621   val_loss: 0.7775613069534302   initial_filter: 256   conv_layer_1: 2   conv_filter_1: 256   conv_layer_2: 0   conv_filter_2: 16   conv_layer_3: 1   conv_filter_3: 256   dense_layer_1: 0   dense_units_1: 256   dense_layer_2: 1   dense_units_2: 32   dense_layer_3: 0   dense_units_3: 256   regularizer: 0.0005
12	val_accuracy: 0.7972778081893921   val_loss: 0.6859294772148132   initial_filter: 32   conv_layer_1: 2   conv_filter_1: 128   conv_layer_2: 1   conv_filter_2: 32   conv_layer_3: 0   conv_filter_3: 16   dense_layer_1: 0   dense_units_1: 128   dense_layer_2: 0   dense_units_2: 64   dense_layer_3: 1   dense_units_3: 64   regularizer: 0.001
13	val_accuracy: 0.7957357168197632   val_loss: 0.6842451691627502   initial_filter: 64   conv_layer_1: 2   conv_filter_1: 32   conv_layer_2: 1   conv_filter_2: 32   conv_layer_3: 0   conv_filter_3: 256   dense_layer_1: 1   dense_units_1: 32   dense_layer_2: 2   dense_units_2: 256   dense_layer_3: 0   dense_units_3: 128   regularizer: 0.0001
14	val_accuracy: 0.7945288419723511   val_loss: 0.7235530018806458   initial_filter: 256   conv_layer_1: 1   conv_filter_1: 64   conv_layer_2: 1   conv_filter_2: 16   conv_layer_3: 1   conv_filter_3: 128   dense_layer_1: 2   dense_units_1: 64   dense_layer_2: 0   dense_units_2: 64   dense_layer_3: 0   dense_units_3: 256   regularizer: 0.005
15	val_accuracy: 0.7924167513847351   val_loss: 0.744037389755249   initial_filter: 32   conv_layer_1: 0   conv_filter_1: 64   conv_layer_2: 2   conv_filter_2: 16   conv_layer_3: 0   conv_filter_3: 128   dense_layer_1: 1   dense_units_1: 256   dense_layer_2: 1   dense_units_2: 128   dense_layer_3: 1   dense_units_3: 64   regularizer: 0.0005
16	val_accuracy: 0.7877233624458313   val_loss: 0.7196337580680847   initial_filter: 256   conv_layer_1: 0   conv_filter_1: 64   conv_layer_2: 0   conv_filter_2: 256   conv_layer_3: 1   conv_filter_3: 64   dense_layer_1: 2   dense_units_1: 64   dense_layer_2: 0   dense_units_2: 128   dense_layer_3: 0   dense_units_3: 128   regularizer: 0.005
17	val_accuracy: 0.7860136032104492   val_loss: 0.719147264957428   initial_filter: 16   conv_layer_1: 1   conv_filter_1: 256   conv_layer_2: 2   conv_filter_2: 32   conv_layer_3: 2   conv_filter_3: 32   dense_layer_1: 1   dense_units_1: 32   dense_layer_2: 1   dense_units_2: 128   dense_layer_3: 0   dense_units_3: 128   regularizer: 0.0001
18	val_accuracy: 0.7817559838294983   val_loss: 0.7723918557167053   initial_filter: 16   conv_layer_1: 1   conv_filter_1: 128   conv_layer_2: 1   conv_filter_2: 32   conv_layer_3: 0   conv_filter_3: 32   dense_layer_1: 1   dense_units_1: 32   dense_layer_2: 1   dense_units_2: 64   dense_layer_3: 1   dense_units_3: 256   regularizer: 0.001

19	val_accuracy: 0.7800797820091248   val_loss: 0.7519169449806213   initial_filter: 256   conv_layer_1: 0   conv_filter_1: 128   conv_layer_2: 2   conv_filter_2: 64   conv_layer_3: 2   conv_filter_3: 16   dense_layer_1: 0   dense_units_1: 64   dense_layer_2: 2   dense_units_2: 32   dense_layer_3: 0   dense_units_3: 256   regularizer: 0.001
20	val_accuracy: 0.7456166744232178   val_loss: 0.8871795535087585   initial_filter: 16   conv_layer_1: 0   conv_filter_1: 256   conv_layer_2: 0   conv_filter_2: 64   conv_layer_3: 0   conv_filter_3: 64   dense_layer_1: 0   dense_units_1: 256   dense_layer_2: 2   dense_units_2: 32   dense_layer_3: 1   dense_units_3: 32   regularizer: 0.0001

# I User Testing

The user testing is translated from Norwegian to English.

## Person A

### Tasks

Send a new song to analysis

1. "Add song"
2. Paste in a YouTube-link in the input-feltet
3. Click «Submit»

Check the result of the newly added song

1. Click "Results"

Find out if any songs failed the analysis

1. Click «Status»
2. Look at the «Failed songs» section

### Questions

Did you encounter any challenges during the execution of the tasks?

- Could have been redirected to the «Results» page when the song was uploaded.

How was your experience finding information if you encountered something you did not understand?

- There is no way to know what the purpose of the «Status» page is. That this is the place where failed songs show up.

What is your overall impression?

- Looks fine except for the points mentioned about the «Status» page.

Do you have any general feedback? (Design, navigation, layout, functionality, etc.)

- API Status does not make sense for people who do not know what it is. The songs should be displayed at the top of the page, as this is the main point.

## Person B

### Tasks

Send a new song to analysis

1. Find a YouTube link
2. Paste it in the input field on «Add song» page
3. Click «Submit»

Find out if any songs failed the analysis

1. Click «Results»
2. Would assume that the song at the top is the newest one. If not: search for it

Finn ut om noen sanger har feilet

1. Search for the songs on the «Results» page
  - If no results occurred I would assume they were not uploaded
2. Possibly go to the «Status» page if nothing happened

## Questions

Did you encounter any challenges during the execution of the tasks?

- If I did not accidentally click the «Status» button, I would have never seen the «Failed songs» section. Usually I would have tried to input the song again.

How was your experience finding information if you encountered something you did not understand?

- Only the things mentioned above.

What is your overall impression?

- Most things make sense.

Do you have any general feedback? (Design, navigation, layout, functionality, etc.)

- There is a bit too much going on on the «Status» page. Maybe a closing functionality could be added so you could collapse an entire category. If there are 100 processing songs, and you want to get to the bottom of the page it is a bit time consuming. Sorting the results would also have been nice, but searching is also good.

## Person C

### Tasks

Send a new song to analysis

1. Paste a YouTube link in the input field
2. Click «Submit»

Check the result of the newly added song

1. Click «Results»

Find out if any songs failed the analysis

1. Check the «Status» page
  - Would assume they were located at the bottom of the «Results» page

### Questions

Did you encounter any challenges during the execution of the tasks?

- The only thing I can think of would be the «Status» page. I would not have checked that page if the last task was not to find the failed songs. I would have assumed they were located at the bottom of the «Results» page.

How was your experience finding information if you encountered something you did not understand?

- The things mentioned earlier. I thought «Pending» and «Processing» were the same thing because of the yellow color that is used. I believe it makes more sense to give

«Pending» a more orange color.

What is your overall impression?

- Somewhat confusing because of the lack of information.

Do you have any general feedback? (Design, navigation, layout, functionality, etc.)

- The layout of the filtering feature is not like expected. It only looks like a description of the colors, and not something you can click. Maybe «Approved» and «Pending» could be placed at the bottom of the page as an explanation, and a bigger version of the squares at the top. Alphabetical or chronological sorting of the results would have been helpful.

## Person D

### Tasks

Send a new song to analysis

1. Input a link in the input field
2. Click «Submit»

Check the result of the newly added song

1. Click «Results»

Find out if any songs failed the analysis

1. Click «Status»
2. Look at «Failed songs»

### Questions

Did you encounter any challenges during the execution of the tasks?

- I would have liked it if «Failed songs» were at the top of the «Status» page, or on the «Results» page. I would not have thought about visiting the «Status» page. Maybe an alert could be displayed on the «Results» page.

How was your experience finding information if you encountered something you did not understand?

- The only thing I found difficult was the «Status» page.

What is your overall impression?

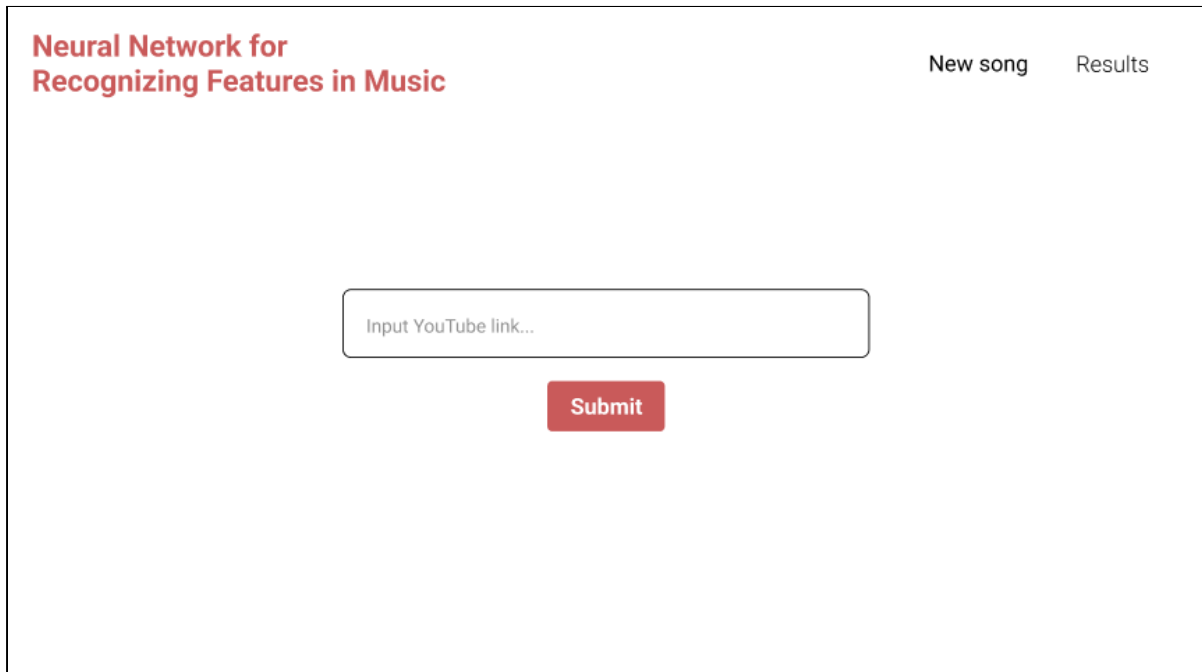
- It is ok and simple.

Do you have any general feedback? (Design, navigation, layout, functionality, etc.)

- It is a bit confusing that the color of «Pending» and «Processing» is the same. One of them should have been orange.

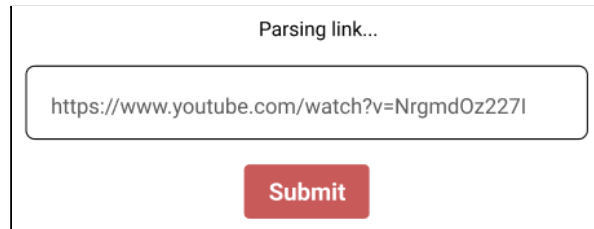
# J Prototypes of Web Application

Before user testing



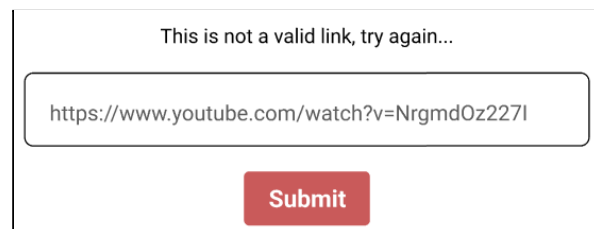
The initial web application interface features a header with the title "Neural Network for Recognizing Features in Music" on the left and navigation links "New song" and "Results" on the right. The main content area contains a text input field with the placeholder text "Input YouTube link..." and a red "Submit" button centered below it.

Input page



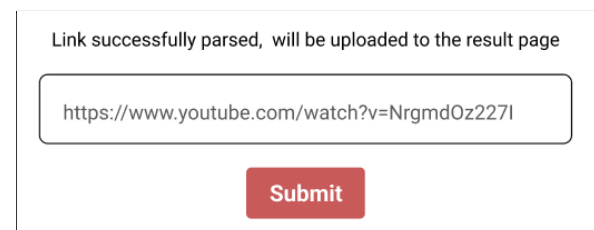
This prototype shows the input page after the user has clicked the "Submit" button. The text "Parsing link..." is displayed above the input field. The input field now contains the URL "https://www.youtube.com/watch?v=NrgmdOz2271". The "Submit" button remains visible below the input field.

Input page - When user clicks submit



This prototype shows the input page with an error message. The text "This is not a valid link, try again..." is displayed above the input field. The input field contains the same URL "https://www.youtube.com/watch?v=NrgmdOz2271". The "Submit" button is still present below the input field.

Input page - The link is not valid



This prototype shows the input page with a success message. The text "Link successfully parsed, will be uploaded to the result page" is displayed above the input field. The input field contains the URL "https://www.youtube.com/watch?v=NrgmdOz2271". The "Submit" button is still present below the input field.

Input page - The link was successfully parsed



**Neural Network for Recognizing Features in Music** Add song **Results** Status

Search for something... Filter: ■ Approved | ■ Pending

■ 100 geecs - money machine (Official Music Video) ▼

---

■ Yesterday (Remastered 2009) Approve ▲

```

{
  "beats": [],
  "parts": {
    "length":,
    "bars": {
      "length":,
      "chords": {
        "chord":,
        "minor":,
        "length":,
      }
    }
  },
  "arrangement": {
    "part":,
    "repetitions":,
  },
  "startTime":,
}

```

---

■ Kanye West - Praise God (Audio) ▼

Output page

**Neural Network for Recognizing Features in Music** Add song **Results** Status

Search for something... Filter: ■ Approved | ■ Pending

**Error: An error has occurred. See status page for details**

Output page - Database error

**Status: 200 OK**

**API Status**

API Connection: 200 OK

Model Connection: 200 OK

Database Connection: 200 OK

**Processing songs**

Lana Del Rey - Watercolor Eyes

**Failed songs**

The Rolling Stones - Paint It, Black (Official Lyric Video)

Radiohead - Karma Police

Status page

## After user testing

**Neural Network for Recognizing Features in Music** Add song Results Status

If you can't find the song you are looking for, check out the status page.

Search for something... Filter: Approved | Pending

- 100 gecs - money machine (Official Music Video) [dropdown arrow]
- Yesterday (Remastered 2009) Approved [dropdown arrow]

```
{
  "beats": [],
  "parts": [{
    "length": 1,
    "bars": [{
      "length": 1,
      "chords": [{
        "chord": "C",
        "minor": false,
        "length": 1
      }]
    }]
  }],
  "arrangement": {
    "part": "verse",
    "repetitions": 1
  },
  "startTime": 0
}
```

- Kanye West - Praise God (Audio) [dropdown arrow]

Output page

**Neural Network for Recognizing Features in Music** Add song Results Status

**Status: 200 OK**

- Processing songs** [dropdown arrow]

  - Lana Del Rey - Watercolor Eyes

- Failed songs** [dropdown arrow]

  - The Rolling Stones - Paint It, Black (Official Lyric Video)
  - Radiohead - Karma Police

- API Status**

  - API Connection: 200 OK
  - Model Connection: 200 OK
  - Database Connection: 200 OK

Status page

# K Meeting Minutes

## 07.01.2022 - Group meeting

Participants:

- All group members

Content:

- Scheduled meeting with client.
- Created a list of questions to ask both client and supervisor.

## 11.01.2022 - Group meeting

Participants:

- All group members

Content:

- Initial setup of project planning phase.
- Discussed tools to use for project management (time tracking, gantt).

## 13.01.2022 - Meeting with EC-Play

Participants:

- All group members
- EC-Play

Content:

- Got an overview of the task.
- Scheduled future meetings: Thursdays every other week at 10. In the planning phase, we will also have an additional meeting 20.01 to get more time to discuss the planning.
- Looked over their dataset and got the format explained.
  - We are getting a better example of the format later.
- Asked about their expectations in relation to software security:
  - Not extremely important as the product we are creating is only going to be used in-house, and is not connected to an important database.
- Discussed the UI:
  - Input: YouTube link
  - The design can be simple, but ultimately we can choose the complexity.
  - What technologies are going to be used: They use React, and it is therefore great if we use it.
- Decided channel for meetings and communication: Discord

## 14.01.2022 - Meeting with supervisor

Participants:

- All group members
- Ali Shariq Imran

Content:

- Scheduled future meetings: Friday every week at 10.
  - We need to create an event in our calendar and invite Ali.
- Asked about the poster point in the follow-up document posted on Blackboard. Ali will ask Tom.
- Got clarification about public vs private repository.

- Important to get requirements from the client at the beginning. Make an agreement, so we can get the functional requirements to get a working project. Also get the non functional requirements for additional features. For example which file types the AI will handle.
  - High level overview of data flow
  - Interaction between the different components
  - Should be part of the architecture part of the development, put it in the final report
- Got feedback on the network model:
  - Keep it high level, it is too complicated and not easy to understand
  - Clear starting and ending point
  - One level in, and divide into blocks. For example pre-processing. What happens in this block?
  - We can make a simpler model before, so this is more understandable
- Got feedback on the risk assessment:
  - Break down into categories:
    - Team dynamics
      - If we get sick
      - If the project is delayed
      - What happens if the leader is sick
    - Other categories related to development
  - Explain why we are not making a mitigation plan for each risk
- Asked about how to get access to SkyHiGh:
  - We should ask Tom.

## 20.01.2022 - Meeting with EC-Play

Participants:

- All group members
- EC-Play

Content:

- Clarification about the repository: it can be public.
  - We are going to create it, and invite them to it.
  - After the meeting we decided to buy Github Pro. This allows us to use features like “Projects” and “Wiki” even though the repository is private. The cost was split among all group members.
- We are going to fix the cooperation agreement and send it to them.

Because of illness, the meeting was kept short.

## 25.01.2022 - Meeting with supervisor

Participants:

- All group members
- Ali Shariq Imran

Content:

- Got feedback on Gantt chart
  - We need to justify why it is correct
  - Is it ok if the blocks are not connected?

- Makes sense because of all the subtitles
- Accuracy/precision in the “Objectives” section
  - Do not recommend to use a specific number unless we support it with a source or if it is a request from the client
- What should we include in the “Delimitation” section?
  - Depends on the client, if they have not given any delimitations, it is up to us.
  - UI
    - Folder, file or link as the input.
    - “We only need a minimalistic UI”
- Public or private repo
  - Should keep it private until we are finished with the project.
- License
  - It is not important to use EC-Play’s license.
- Should we add the product backlog to the project plan?
  - Not required in the project plan.
- Should we keep a log separate from meeting minutes and time tracking?
  - Can be risky to have a lot of appendices, more important to have a more extensive final report.
  - Spend little time on things that are not important for the project.

## 27.01.2022 - Meeting with EC-Play

Participants:

- All group members
- EC-Play

Content:

- Standard agreement
  - We need to find out if the person who signs it (client) also needs to be set as the contact person:
    - If so: change contact person from Kristoffer til Joakim.
- Model performance:
  - Clarified what the % means.
  - Not sure if we should define the performance?
- Million Song Dataset:
  - Not great that it is only one bpm value.
- Should the input take several YouTube links?
  - Not necessary.
- Should be a page where all the results from all inputted links are shown
  - If you do not have time to correct it right away, it is still there later.
- 1 bpm per tempo.
- Time limit when training on corrected songs:
  - If there are, for example, only 7 songs waiting for a long time it should happen anyway.

## 04.02.2022 - Meeting with EC-Play

Participants:

- All group members
- EC-Play

## Content:

- Nines notation - what is the goal?
  - We can decide.
- Output (JSON format):
  - {
    - beats: [number (timestamps in seconds)],
    - parts - description of a part: []
      - {
        - bars: bokser [{
          - length: antall slag
          - chords: [{
            - chord: 0-11
            - minor: bool
            - length: antall slag
          - }]
        - }]
      - }
        - arrangement: the order of the parts, and how many times they are repeated each time.
        - startTime: first timestamp in beats.
    - }
- We go feedback on the UI:
  - We need to change the output page so it corresponds with the correct JSON (discussed earlier in the meeting).
  - Status page is good:
    - However, adding a "Processing songs" section where all songs that are currently analyzed would be nice.
    - If an error happens (for example problems with saving results to the database), this can also be shown here.
- Feedback on use case
  - Can expand "See YouTube video".

## 07.02.2022 - Meeting with supervisor

### Participants:

- All group members
- Ali Shariq Imran

### Content:

- Status on the agreement and project report:
  - Everything OK
- Licensing of libraries used - can we only use libraries with license for commercial use?
  - As long as the project is not used commercially, we can use it.
  - If the client wants to use it after the project, we should be more concerned about it.
- Status report and meeting notice:
  - We can decide the dates and how many meetings.
- Got feedback on requirements specification.

- Got feedback on architecture.
- Clarification on weekly meeting:
  - Friday at 10 is still good.

## 11.02.2022 - Meeting with supervisor

Participants:

- All group members
- Ali Shariq Imran

Content:

- Discussed the requirement specifications:
  - The starting point of the architecture is not clear.
- We should create a list of all features we are going to implement.
- Talked about dataset:
  - We should ask EC-Play for theirs.

## 14.02.2022 - Group meeting

Participants:

- All group members

Content:

- Looked at code reviews on Github.
- Invited all members to the Firebase project.
- Fixed Gantt chart.
- Planned API endpoints.

## 17.02.2022 - Meeting with EC-Play

Participants:

- Sindre Eiklid
- Rickard Loland
- EC-Play

Content:

- Talked about changes in plans (MVP).
- Talked about current status (What we have done):
  - Rickard shared screen showcasing beat algorithm and chromagram.
  - Kristoffer said that they don't know exactly how many beats they need, but no point in being super accurate.
  - Wonders how BPM is calculated (found the way the algorithm does it to be good).
  - Kristoffer said that we should compare the chords with the beats to verify
    - Chords often switch with beats.
    - Chordify assumes 4-beat.
- Talked about needing their dataset:
  - Kristoffer will try to copy over the dataset into a new project that we can get access to.
  - Might be some mistakes: they suggested double checking
  - Newest songs are definitely correct (tested through metronome)
    - Date created label
    - last updated



- For pattern recognition we should look at beats + chords:
  - The same pattern can happen at two different tempos.
  - 4 beats per bar - akkord varer en hel bar (Ikke alltid):
    - Patterns often happen within a length of 4 (or 8, 12, 16) bars
    - hvis pattern 1 følger alltid etter pattern 2, bør det legges sammen? må finne en gunstig lengde
      - 4 - 12 bars er en god løsning
      - chords that was left over was added to its own pattern
      - hvis bare en takt er forskjellig blir det sin egen pattern
      - if only one chord is different then it will be added to its own pattern
      - leftovers: its own pattern
  - They suggested using the EC-Play player to look for easy songs.
  - They said that grouping them together like they have done programmatically would be difficult and said that human correction would be fine here.
- We informed them about the incoming status meeting next week.

## 18.02.2022 - Meeting with supervisor

Participants:

- All group members
- Ali Shariq Imran

Content:

- Validate algorithm to check if it is consistent: check if the beats are occurring on the same timestamps every time it is run
  - Find another algorithm and look at the output.
- We should try to visualize the beat data.
- Informed him about the status meeting next week.

## 21.02.2022 - Group meeting

Participants:

- All group members

Content:

- Created status report 1.
- Sent a meeting notice (with the status report attached) to the client and our supervisor.

## 24.02.2022 - Status meeting with EC-Play

Participants:

- All group members
- EC-Play

Content:

- Informed client about the current status:
  - Added additional information to what was written in the status report, as we have made progress since it was written.
- Asked about their dataset again:
  - They are going to send it.

## 25.02.2022 - Status meeting with supervisor

Participants:

- All group members
- Ali Shariq Imran

Content:

- Feedback on status report and list of requirements:
  - Everything is fine for now.
- Status on algorithms
  - The research has to be documented:
    - Write down the different algorithms
    - Weaknesses and strengths
    - Try to understand how they are working
  - Find an additional dataset.
- We should show Ali some of our outputs and visualizations next week.

## 28.02.2022 - Group meeting

Participants:

- All group members

Content:

- Updated Gantt chart.
- Status on algorithms
  - Beats algorithm ready for feedback:
    - Aubio and librosa comparison
    - Potentially use outputs for NN training
    - Ask how to handle offbeats detected
  - Chord algorithm working, but want to improve it:
    - Currently getting chord data for every frame, want only on beats
    - Getting the raw PCP data from pyACA - how?
    - Simpler but worse alternative is just getting the calculated chord for the timestamps.
- Status on API
  - Good progress so far.
  - On hold for now.

## 03.03.2022 - Meeting with EC-Play

Participants:

- All group members
- EC-Play

Content:

- Showed and discussed the visualization of both beats and chords.
- Got feedback on issues with chords - problem is different chords in vocals and instrumental.
- New idea: filtering out specific frequencies to get the chords we want.

## 03.03.2022 - Meeting with supervisor

Participants:

- Rickard Loland

- Sindre Eiklid
- Ali Shariq Imran

Content:

- Showed and discussed the visualization of both beats and chords.
- Feedback on beats - get MSE, test on songs that change tempo, research the algorithms for report.
- Feedback on chords - filter out frequencies to get right chords, use preprocessing to pass in only timestamps on beats.

## 28.02.2022 - Group meeting

Participants:

- All group members

Content:

- Updated Gantt chart.
- Status on what everyone has done since last time:
  - Rickard has worked on separating vocals and instruments.
  - The first version of the input page has been developed.

## 03.03.2022 - Meeting with EC-Play

Participants:

- All group members
- EC-Play

Content:

- Status on algorithms:
  - The chord part is going great, but there is a problem with differentiating some chords.
  - Some problems with beat algorithm:
    - Når de ikke er samme over hele, går det ikke.
    - Problems with numpy
      - Splitting of songs (vocals and instruments) requires another version of numpy than Aubio does. It has the worst result, but if we do not use it we only have Librosa for testing at the moment.
      - Should probably split the songs even more:
        - Bass and drums have the most even rhythm, so these might be interesting.

## 18.03.2022 - Meeting with supervisor

Participants:

- All group members
- Ali Shariq Imran

Content:

- Started on neural network:
  - No point in doing it for beats, probably not gonna get a better result with the algorithm.
  - Process of chords.

## 21.03.2022 - Group meeting

Participants:

- All group members

Content:

- We need to test the Dockerfile on Windows.
- What everyone is going to work on this week:
  - Rickard: continue working on batch processing.
  - Sindre: neural network.
  - Maren: continue developing the API.
- Created initial outline of the report.

## 24.03.2022 - Meeting with EC-Play

Participants:

- All group members
- EC-Play

Content:

- Status on batch processing:
  - Discussed problem with beat timestamps being after the song is over. We might just have to cut the end of the array.
- How to evaluate beat algorithm performance?
- Training set: has to be the same matrix size. But the x-axis will be varying. Have to normalize for the same length.
  - Add empty frames to get the length of the biggest one for now.
  - Exclude things over 50, threshold.

## 25.03.2022 - Meeting with supervisor

Participants:

- All group members
- Ali Shariq Imran

Content:

- Algorithm comparison:
  - Pretty good accuracy for chords, but don't know how to compare beat performance. If it finds offbeats between the beats, not a problem in terms of what they need, and it is also not inaccurate. But it would only be 50% really.
    - Make assumptions. Manually labeled.
    - If new song, it is hard to separate offbeat
      - Threshold?
        - Can vary a lot between songs and in one song
- Practical vs academic neighboring chords:
  - Analysis point of view, the more we dive down into these kinds of things, the more value in terms of research. If we think it is worth exploring, we should go for it.
- 4 chords that will be trained a lot:
  - Talked about transposing the data with EC-Play. Will that help or make the neural network worse?
    - Neural network might overtrain on those particular chords.
    - Either train model as is, and see the result.

- Or oversample the data. make them balanced by reusing the chords that are missing.
  - See side by side result.
- In the report: write about the different solutions we have looked at. Even if they did not work.

## 01.04.2022 - Meeting with supervisor

Participants:

- All group members
- Ali Shariq Imran

Content:

- Training set for model finished, run through once. Accuracy improved with preprocessing. Still issues with overfitting.
- Plot the data as heatmaps so it is easier to identify.
- Chord algorithm result:
  - Average is around 60%.
- When should we try to have the first edition of the report finished?
  - Ali wants to see the structure/outline, and how we plan to fill them, as soon as possible. The end of Easter is ok.
  - Whatever we are doing, put it in the report. Write about/rewrite it later.
- Comparison of chord and beat algorithms.

## 07.04.2022 - Meeting with EC-Play

Participants:

- Rickard Loland
- Maren Skårestuen Grindal
- EC-Play

Content:

- Status on the neural network.
- Transposing of the songs:
  - Looked at it, but prioritized preprocessing for now.
- Planned status meeting after easter break:
  - April 28th does not work for EC-Play.

## 08.04.2022 - Meeting with supervisor

Participants:

- Rickard Loland
- Maren Skårestuen Grindal
- Ali Shariq Imran

Content:

- Issue with latest issue of tensorflow:
  - Problems with numpy arrays, so had to use lists.
- Dynamically change the number of layers or kind of layers?
  - Ali will take a look at it.
  - Automatically change the architecture with configuration file.
- Important to test different architectures for research purposes.

## 22.04.2022 - Meeting with supervisor

Participants:

- All group members
- Ali Shariq Imran

Content:

- Random search model:
  - Tried different combinations.
  - Do not need to do anything with the activation functions.
  - Not much details, number of layers and nodes are enough.
  - I henhold til rapporten er det som har blitt gjort bra nok.
- Status:
  - Important with literature.

## 25.04.2022 - Group meeting

Participants:

- All group members

Content:

- Update on random search.
- Send out status report in preparation for the status meetings this week.
- Plan for upcoming week, we are trying to finish all code related tasks by sunday.
- Next week we are going to update all figures in the report.

## 26.04.2022 - Status meeting with EC-Play

Participants:

- All group members
- EC-Play

Content:

- Random search: best model with 82% accuracy
  - Change of dataset to improve model (or transposing, but no time).
- This week will mostly be used for deployment.

## 28.04.2022 - Group meeting

Participants:

- Rickard Loland
- Sindre Eiklid

Content:

- Discussed progress on neural network code and results analysis.
- Plan second meeting 29.04 to coordinate work for the following weekend.
- Starting to look at deployment options, potential barriers.
- Plan to look closer at licensing to see if non-commercial license on repo can be used.

## 29.04.2022 - Status meeting with supervisor

Participants:

- All group members
- Ali Shariq Imran

Content:

- Ali will read the report draft soon.
- Updated him on the neural network.
- Presentations:
  - More emphasis on scope, conclusions and work that have been done.
  - Be short and precise.
  - Should a demo be included?
    - Maybe not too interesting.
    - Create an interface in python and showcase some of the functionality.  
For example the different algorithms with an image.

## 02.05.2022 - Group meeting

Participants:

- Rickard Loland
- Sindre Eiklid
- Maren Skårestuen Grindal

Content:

- Merged the branches containing the remaining dockerfiles to main.
- Created docker-compose file.
- Set a final deadline for coding: thursday 5th of May
- Created issues for all remaining parts of the report.

## 05.05.2022 - Meeting with EC-Play

Participants:

- All group members.
- EC-Play

Content:

- Rickard showed result of naboanalyse.
- Description of EC-Play in the report:
  - They are gonna write some notes and send it to us.

## 13.05.2022 - Meeting with supervisor

Participants:

- All group members
- Ali Shariq Imran

Content:

- The flow does not make sense all of the time (section 8 and 10).
- Everything, even the simple things, should be explained with references and arguments.
- The table of attachments can be put near the table of content.
- Try to have less subsections, especially in the “Further work” section.
  - We have written more since the version we sent Ali, so he will look at it again.
- When PDFing our report, make sure the links are still there. Do not use the print version.
- Glossary - are we supposed to have the source to the common definitions?
  - May not need to put a source there, as we are not claiming it is our own definition.

- When he is done going through the report, we can change the things and send it to him again.

## 15.05.2022 - Group meeting

Participants:

- Rickard Loland
- Sindre Eiklid
- Maren Skårestuen Grindal

Content:

- Went through the comments on section 1 to 5 from Ali.
- Wrote down all things to be done before, and on, monday.



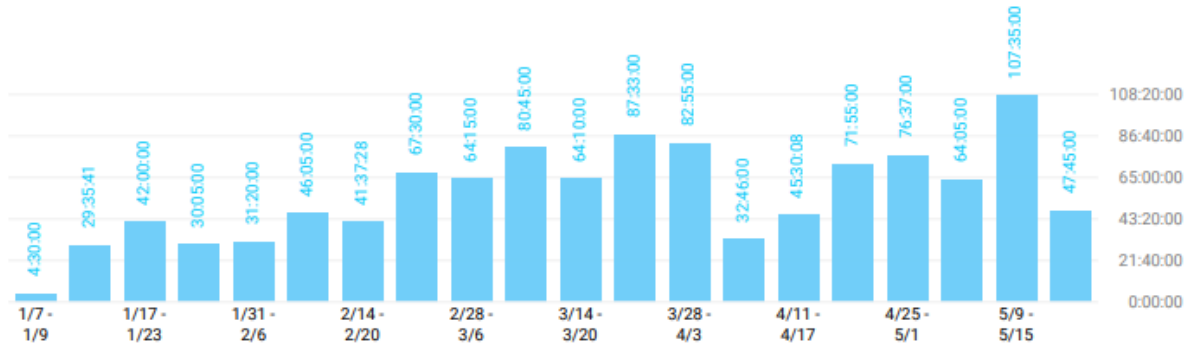
# L Time Log

## Summary Report



01/07/2022 – 05/20/2022

TOTAL HOURS: 1118:34:17



- USER
- SE Sindre Eiklid
  - RI Rickarl
  - MA Mareng

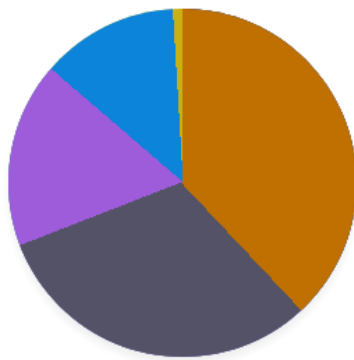
- DURATION
- 381:24:40
  - 370:37:37
  - 366:32:00



- TIME ENTRY
- Report
  - Back-end: AI
  - Competence Acquisition
  - Meeting
  - Front-end
  - Back-end: Cloud
  - Testing

- DURATION
- 344:23:57
  - 314:38:00
  - 167:12:28
  - 131:17:52
  - 96:35:00
  - 39:05:00
  - 25:22:00

## Rickard Loland



TIME ENTRY
Report
Back-end: AI
Competence Acquisition
Meeting
Testing

DURATION
142:14:53
116:30:00
64:40:00
47:32:44
0:20:00

## Maren Grindal

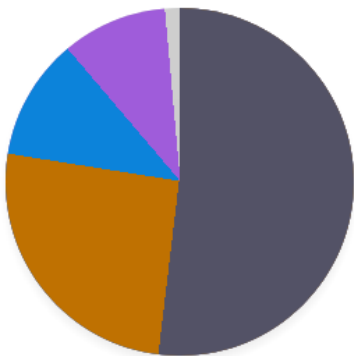


TIME ENTRY
Report
Front-end
Competence Acquisition
Meeting
Back-end: Cloud
Testing

DURATION
105:30:00
94:20:00
65:20:00
40:42:00
38:35:00
22:05:00

## Sindre Eiklid

Other time entries include: *Testing 2:57:00*, *Front-end 2:15:00*, and *Back-end: Cloud 0:30:00*



TIME ENTRY
Back-end: AI
Report
Meeting
Competence Acquisition
Other time entries

DURATION
198:08:00
97:19:04
43:03:08
37:12:28
5:42:00

