

Mood Classification

Classification of Songs Into Moods Using Convolutional Neural Network

Master Thesis

Philippe Bucheli

University of Bern

August 2022

“Productivity is never an accident. It is always the result of a commitment to excellence, intelligent planning, and focused effort.”

- *Paul J. Meyer*

Abstract

The motive of this dissertation was to analyze the performance and features of the most common mood classifier of songs. Different datasets and moods were tested. Furthermore, the two different approaches of single and multi-label classification were compared. In the end, the listening data was taken as an input for the constructed spectrogram-based convolutional neural network (CNN). Moreover, the four moods happy, sad, calm and energetic were taken.

In addition, a web application was implemented which provides the service of retrieving the Spotify tracks of a user's favorite anime (Japanese TV shows) titles in a playlist. This was achieved by connecting anime TV shows to their songs with the help of the API of MyAnimeList and Spotify. The frontend was implemented in HTML and JavaScript, while the backend was written in Python. Hence, the micro web framework Flask was utilized to host it later with Firebase and Google Cloud Run. Continuous Deployment was also implemented to simplify the update and deployment process as soon as changes were made on GitHub.

In order to create the Spotify playlists of the favorite anime titles, the Spotify OAuth login was implemented. This resulting playlist could then be categorized by moods and accessed via a new playlist saved in the user's Spotify account.

Another neural network using KerasClassifier was taken as a comparison and evaluation of this prototype by using the same dataset split into a training and a validation set. However, for one model the audio features were given as input whereas the other model only received the frequency in Hertz. Thus, the accuracy of this model using KerasClassifier, namely 72.75 %, can be compared to some extent to the results of the self-implemented model which has a slightly higher accuracy of 78.57 %. Hence, a hybrid solution using both systems was used for the prototype to solve the nuances between few moods.

The second evaluation was done by taking some existing Spotify playlists labeled with a mood as the input of the model and filtering it by the same mood. Thus, the number of titles added to the new playlist categorized by this mood indicated the success rate. The mood energetic provided the best predictive results with a rate of 82.14 %. Generally, it was observed that anime songs are mostly predicted as energetic.

Keywords: Mood, Spotify, Song, Anime, Convolutional Neural Network, Image Classification, Multi-Label Classification, Flask, Firebase, Google Cloud Run, Continuous Deployment

Prof. Dr. Jacques Pasquier-Rocha, Software Engineering Group, University of Fribourg, Supervisor

Preamble

Acknowledgements

First of all, I would like to express my gratitude to Prof. Dr. Jacques Pasquier-Rocha, who gave me the chance to write this master thesis. Next, I would like to thank Kevin De Keyser. He gave me tips on how to overcome many obstacles and thus guided me to a successful result. Moreover, he provided crucial advice as an expert of convolutional neural networks, which made it more efficient.

Table of Contents

1 Introduction	1
1.1 Problem Statement.....	1
1.2 Research Questions.....	1
1.3 Objectives	2
1.4 Outline	2
1.5 Technologies Used.....	2
1.6 Time Schedule	2
2 Anime Titles and Spotify Songs	3
2.1 Dataset	3
2.2 Web Scraping.....	3
3 Mood Prediction	4
3.1 Dataset	4
3.1.1 AllMusic.....	4
3.1.2 Spotify and GraceNote	5
3.1.3 MTG-Jamendo	5
3.1.4 Spotify	5
3.2 Moods	5
3.3 Image Classification	6
3.3.1 Convolutional Neural Networks.....	6
3.3.2 MP3 to Spectrograms	9
3.3.3 Construction of the CNN.....	9
3.3.4 Fast AI.....	10
3.3.5 Multi-Label Classification.....	10
3.3.6 Train a Model	11
3.3.7 CPU vs. GPU.....	13
3.3.8 Results	14
3.3.9 Comparison to KerasClassifier.....	16
4 Prototype	17
4.1 Software Architecture	17
4.2 Spotify OAuth.....	18
4.3 MyAnimeList API	19
4.4 Anime Songs Playlist.....	20

Table of Contents

4.5	Mood Classification.....	20
4.6	Hosting Firebase	21
4.7	Continuous Deployment	24
4.8	Evaluation	25
5	Conclusion	27
	References	30
	Referenced Web Resources	32

List of Figures

Figure 1 Time Schedule	2
Figure 2 Traditional Model of Mood	5
Figure 3 Layer 1	7
Figure 4 Layer 2	7
Figure 5 Layer 3	8
Figure 6 Layer 4	8
Figure 7 Spectrogram	9
Figure 8 Architecture of CNN	9
Figure 9 Order of Functions Called by DataBlock	10
Figure 10 Plot of Loss and Learning Rate	11
Figure 11 Structure of FP16	11
Figure 12 CUDA Out of Memory Error	13
Figure 13 Results of Training and Validation	14
Figure 14 Scores of Predictions	14
Figure 15 Results for Taking the Middle Part of Songs	14
Figure 16 Predictions for Guren no Yumiya TV Size Version	15
Figure 17 Accuracy of Model Trained With the Spotify Dataset	15
Figure 18 Predictions of Two Chart Songs	15
Figure 19 Confusion Matrix	16
Figure 20 Spectrograms with their Label and Prediction	16
Figure 21 Software Architecture	18
Figure 22 Landing Page of the Web Application	19
Figure 23 Spotify OAuth Login	19
Figure 24 Provided Link of the New Playlist	20
Figure 25 Selection of Moods to Filter the Playlist	21
Figure 26 The Structure of the Project	22
Figure 27 Docker File Part 1	22
Figure 28 Docker File Part 2	23
Figure 29 Build Summary of the Continuous Deployment	24

List of Figures

Figure 30 Finished Step Two With the Link to Access the Hosted Web Application.....	25
Figure 31 Songs Predicted as Sad	25
Figure 32 Songs Predicted as Energetic	26
Figure 33 Songs Predicted as Calm.....	26

1

Introduction

1.1 Problem Statement	1
1.2 Research Questions	1
1.3 Objectives	2
1.4 Outline	2
1.5 Technologies Used	2
1.6 Time Schedule	2

1.1 Problem Statement

Currently, there is no service available to find the songs of favorite anime titles. Hence, the idea of a website matching these titles rated on MyAnimeList [1] with the songs on Spotify [2] came about. By comparing other music streaming platforms, like YouTube or Apple Music, the requirement of a mood classifier was discovered. There are many ways to classify the mood of a song. Thus, the most suitable method must be found. Furthermore, the quality of the classifier depends also on the choice of moods and its dataset.

1.2 Research Questions

The following research questions will be addressed in this master thesis:

- Where and how can the connection of anime titles and their songs be made?
- Which mood classifier method is the most efficient one?
- How can audio data be used for image classification and what needs to be considered?
- Which moods are relevant for a user to classify his or her playlist and how should they be displayed?
- What is the most cost-effective and robust way to host a static and dynamic website

1.3 Objectives

The goal of this project is to create a functional and robust mood classifier of songs by using an innovative method. This will be achieved by accessing different data of many songs provided online and analyzing the ways to classify moods of songs. The accuracy of an image classification using spectrograms of songs will be compared to the classification with the help of audio features.

Ultimately, the results of the final prototype will also be evaluated by predicting moods of Spotify playlists labeled with a specific mood. More than 66.66 % of the playlist's songs should be predicted as the mood mentioned in the title of the playlist.

1.4 Outline

This master thesis was written in three parts. The first part consists of the preparation where the data was obtained by using Selenium and saved efficiently in a text file. The second one is about the choice of the dataset with the moods as labels and how the image classification was implemented and trained.

The final section belongs to the more practical part containing the different steps completed to create the prototype. At the end of this section, the way to host this prototype and the conducted evaluation are described.

1.5 Technologies Used

The frontend was implemented with HTML and JavaScript whereas the backend was written in Python. Hence, the micro web framework Flask was used to host it later with Firebase and Google Cloud Run. Continuous Deployment was also set up to improve the update and deployment process as soon as changes were made on GitHub. The API of MyAnimeList and Spotify were needed to access the users' anime titles and store the new created playlists. The fast AI library was utilized to implement the CNN model, which enhances training of fast and accurate neural networks using modern best practices.

1.6 Time Schedule

The following work plan in Figure 1 was created and followed during this thesis.

<p><i>June 2022 :</i></p> <ul style="list-style-type: none">• Personal thinking, background work and some testing on the thesis matter• Introduce master thesis to Prof. Jacques Pasquier-Rocha• Inform about several infrastructure and plan the architecture• Install all the requirements to be able to start programming with continuous integration and delivery• Scrape data from websites and prepare the dataset <p><i>July 2022:</i></p> <ul style="list-style-type: none">• Implement the algorithm for classifying moods of songs and evaluate it• Implement frontend and backend of application	<p><i>August 2022:</i></p> <ul style="list-style-type: none">• Beginning of August: First Demo of the System• Implement frontend and backend of application• Evaluate Prototype• Finalize Master Thesis Report <p><i>1. September 2022:</i></p> <ul style="list-style-type: none">• Hand-in Master Thesis• Evaluation of Thesis by Prof. Jacques Pasquier-Rocha <p><i>16. September 2022:</i></p> <ul style="list-style-type: none">• Hand-in Master Thesis to the Decan
--	--

Figure 1 Time Schedule

2

Anime Titles and Spotify Songs

2.1 Dataset	3
2.2 Web Scraping	3

2.1 Dataset

In order to build the connection between the anime titles from MyAnimeList and their songs from Spotify, the website AnimeMusic [3] was scraped. The new dataset consists of 2'106 lines with each an anime title, URL to MyAnimeList, song title, artist, URL to the song on Spotify, URL to the artist page on Spotify and the kind of song (opening, edition or ending). This information was saved in a text file called "animeSongs.txt" since it will only be processed by the computer and must not be readable for programmers. The different columns were separated by two semicolons since anime titles tend to have special characters. The Spotify URL consists of a keyword, like track or playlist, followed by a colon with the corresponding identifier.

2.2 Web Scraping

The python library Selenium [4] was used to obtain the dataset mentioned above. It is mostly used for automated tests of web applications but is also a practical tool to scrape websites since it is simpler to debug with the user interface and runs rapidly. The code below was used to loop through the table row by row for each anime song and extract the information in the columns. Lastly, it was saved in a text file. The full code can be found on the GitHub repository [5] in the prepareDataset folder and saved as the animeSongsScraping.py file.

```
1      # Open file to append anime songs
2      file = open('animeSongs.txt', 'a', encoding='utf8')
3
4      for t_row in range(1, (rows + 1)):
5
6          for t_column in range(1, (columns + 1)):
7              FinalXPath = before_XPath + str(t_row) + aftertd_XPath +
8                  str(t_column) + aftertr_XPath
9              cell_text = driver.find_element_by_xpath(FinalXPath).text
10
11          # Write the Anime Title to the text file
12          if t_column == 3:
13              file.write(";;ANIMENAME;;" + cell_text)
```

3

Mood Prediction

3.1 Dataset	4
3.1.1 AllMusic	4
3.1.2 Spotify and GraceNote	5
3.1.3 MTG-Jamendo.....	5
3.1.4 Spotify	5
3.2 Moods	5
3.3 Image Classification	6
3.3.1 Convolutional Neural Networks	6
3.3.2 MP3 to Spectrograms	9
3.3.3 Construction of the CNN.....	9
3.3.4 Fast AI.....	10
3.3.5 Multi-Label Classification	10
3.3.6 Train a Model.....	11
3.3.7 CPU vs. GPU	13
3.3.8 Results.....	14
3.3.9 Comparison to KerasClassifier	16

3.1 Dataset

There are several datasets available to predict the mood of a song. The following datasets are sorted as they were tried.

3.1.1 AllMusic

In [Kor20] a subset of the Million Song Dataset with 67'000 tracks containing expert annotations of 188 different moods was taken. It was provided by the music collection called AllMusic [6]. Unfortunately, only the model with the song ID of the titles on AllMusic was provided, but not the mood of the titles themselves. When asked, the authors of the paper confirmed that they were not authorized to provide these labels. The only approach would have

been to scrape the moods of all 67'000 tracks on AllMusic. However, finding the moods of every song was only possible with its title by using the search bar of the website because no ID was provided in the model. Due to the fact that these titles did not always match and this approach is forbidden according to AllMusic, another dataset had to be found.

3.1.2 Spotify and GraceNote

In [7] a dataset with over 21'000 songs, 69 potential features and 25 moods was created by using the APIs of Spotify and GraceNote [8]. This big dataset with recent song titles would have been very useful, but could only be accessed by reaching out to the author. Unfortunately, the author did not respond to any request. Hence, the construction of the dataset was tried by using the provided code. The information of the Spotify playlists was accessed in a simple way. However, GraceNote does no longer offer a free way to receive the important mood labels of the songs through their API. Thus, the company was contacted to obtain the dataset, but again there was no response and another dataset was required.

3.1.3 MTG-Jamendo

In [Tan21] the open dataset for music auto-tagging called MTG-Jamendo [9] is provided by this paper. It consists of 55'000 full audio tracks distributed in 320kbps MP3 format with 56 mood/theme tags. However, only 18'487 titles were labeled with the mood tags. The song IDs and corresponding moods are available as .csv file. Due to the huge number of moods and tracks, this dataset was used as the first approach with a multi-label classification.

3.1.4 Spotify

In [10] another dataset is available with four moods each with 200 tracks including their labels. The features of instrumentality, liveness, valence, length, danceability, energy, loudness, speechiness, acoustics and tempo were used in this paper to classify the tracks. This dataset was applied after the first approach with the poor result of the MTG-Jamendo approach. Not only does it provide an accurate prediction for those four moods, but it also enables a good comparison between the spectrogram- and feature-based neural networks.

3.2 Moods

Finally, only four moods were considered according to [Nuz21], which highlights the best way of classifying music by mood. The moods of songs are divided regarding to psychologist Robert Thayer's traditional model of mood. This can be seen in Figure 2, which divides tracks along the y-axis representing energy and x-axis for stress, from calm to energetic and happy to sad. In the MTG-Jamendo dataset there were more moods available, but less suitable ones for this mood classifier such as Christmas, documentary, space and

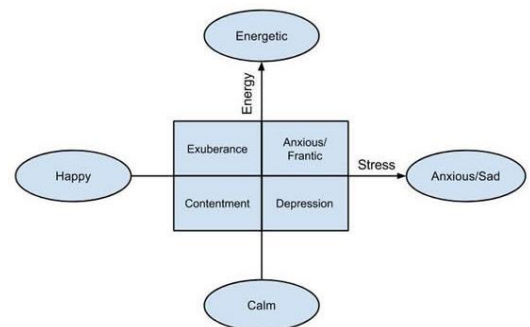


Figure 2 Traditional Model of Mood

others alike. Furthermore, some moods are identical like soft and calm or powerful and energetic.

3.3 Image Classification

Unfortunately, there is no pure solution for classifications like in software architecture. First, a state of the art of the different machine learning methods was searched. Afterwards, the trial and error approach was used to identify the most suitable solution.

The chosen approach was to use listening data for the mood classification, since these features outperform content-based features, as mentioned in [Kor20]. The results of matrix factorization of listening data are more helpful to identify the mood than those regarding its audio content. Moreover, a content-based model, for instance analyzing the lyrics of a song, could not have been taken into account, since the focus of this thesis was to classify Japanese anime songs. Thus, it would be difficult to accomplish a successful transfer learning from English to Japanese.

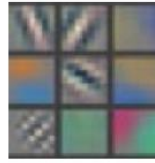
On the other hand, the paper suggests using spectrogram-based convolutional neural network (CNN), which achieves state-of-the-art results. Thus, the same approach was applied with the same dataset, but another CNN was created.

According to [GAV19] the image classification problem is well known in computer vision. It consists of finding visual structures of an image given as input by analyzing the numerical properties of different features of an image and organizing them into categories. In the past, the classification approaches like fuzzy-sets, artificial neural networks and expert systems have been mostly used for image classification. However, all of them have some problems and compared to the CNN their accuracy is lower. The CNN has recently achieved impressive classification performance on the ImageNet benchmark [Kri12].

3.3.1 Convolutional Neural Networks

In [Zei14] a novel visualization technique is introduced to get a clear understanding of why the large convolutional network models perform so well, which was not known before. This technique offers insights into the function of intermediate feature layers and the operation of the classifier. It also visualizes the weight learned in each layer of a model. The following explanations were gathered from [11].

The first layer of parameters has many features and may look like the image in the upper left corner of Figure 3 drawn as a picture with 9 images. One of them has the ability to recognize diagonal lines from the top left to bottom right as can be seen in Figure 3 in the image in the upper left corner. The other image on the right could have diagonal lines from bottom left to top right and the one below could have a gradient that flows from top orange to bottom blue. Each of these nine images are called features or filters. In actual images like in the lower image of Figure 3, specific parts of photos that match that filter were found. For instance, for this filter on the top left, there are nine actual patches from real photos that match this filter as they are all diagonal lines. Thus, the first layer is very simple, but useful for other computer vision tasks as well.



Layer 1

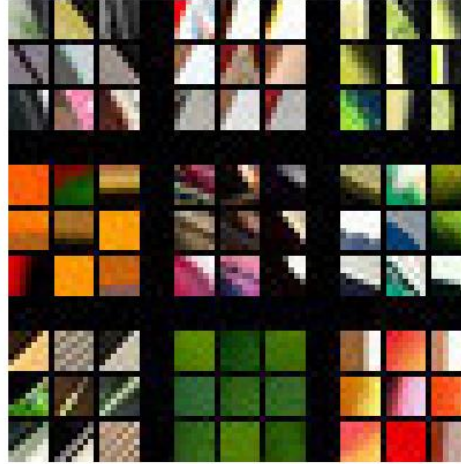


Figure 3 Layer 1

Layer two takes the features of layer one and combines them. Hence, it cannot just find edges but also corners, semi-, full-circles or repeating curving patterns as it can be seen on the left side of Figure 2. On the right side, there are parts of photos that this layer 2, for example the circular filter, has activated on. Objects with circles were found in the block of the second row from above and second column from the left. Another example is the block with the orange gradient on the left side in Figure 4 in the second row and first column, which seems to be good at finding sunsets according to the right side in Figure 4 in the second row and first column.

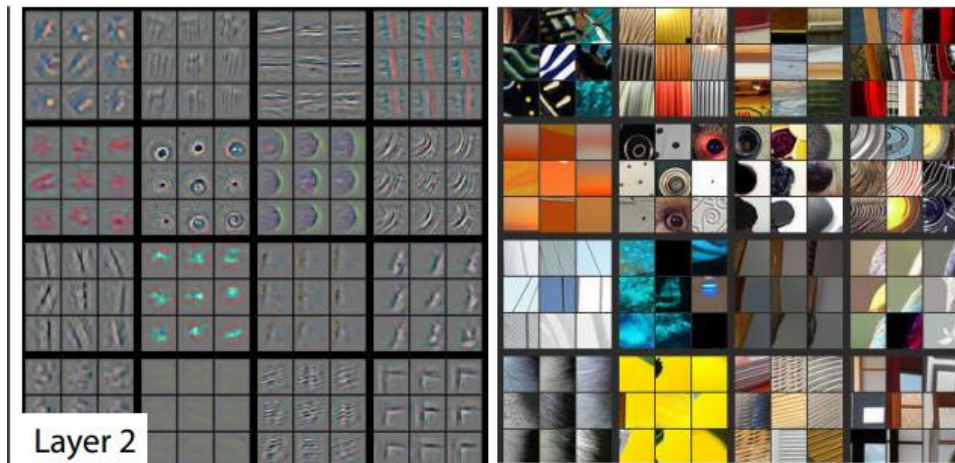


Figure 4 Layer 2

Layer three can combine all kinds of features in layer two which can already find text. Furthermore, repeating geometric patterns can also be found like in the first block on the right side of Figure 5. It is not just matching specific pixel pattern, but a semantic concept finding repeating circles, hexagons, or squares. Thus, it is really computing and not just matching a template.

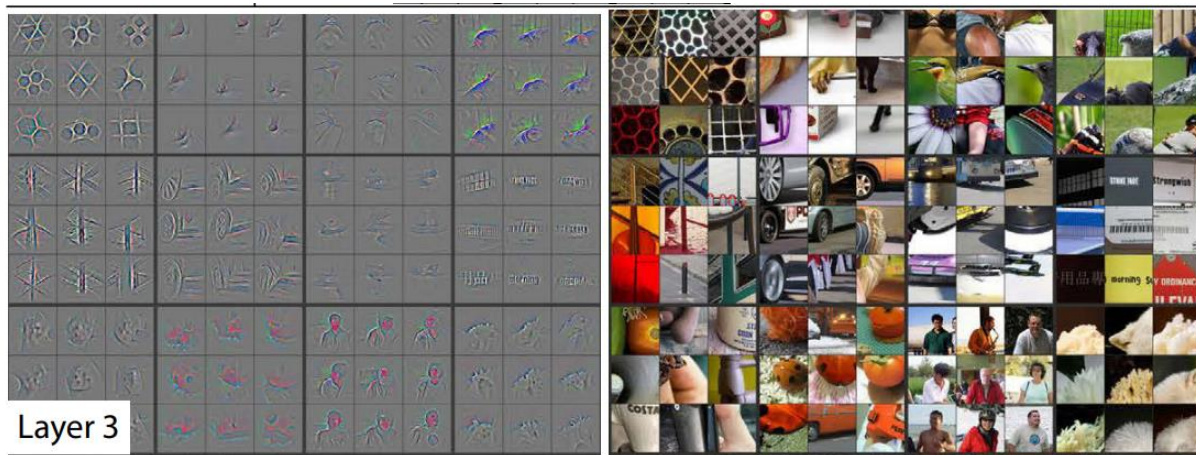


Figure 5 Layer 3

Layer four combines all the filters from layer three as represented in Figure 6, which can now find for instance the face of dogs. For each layer, more sophisticated features are obtained, which explains why deep neural networks can be very powerful. That is the reason why transfer learning works very well, as you can take advantage of all these pre-learned features to find objects that are combinations of these existing features. Moreover, transfer learning can be performed much faster and with much less data than traditional approaches.

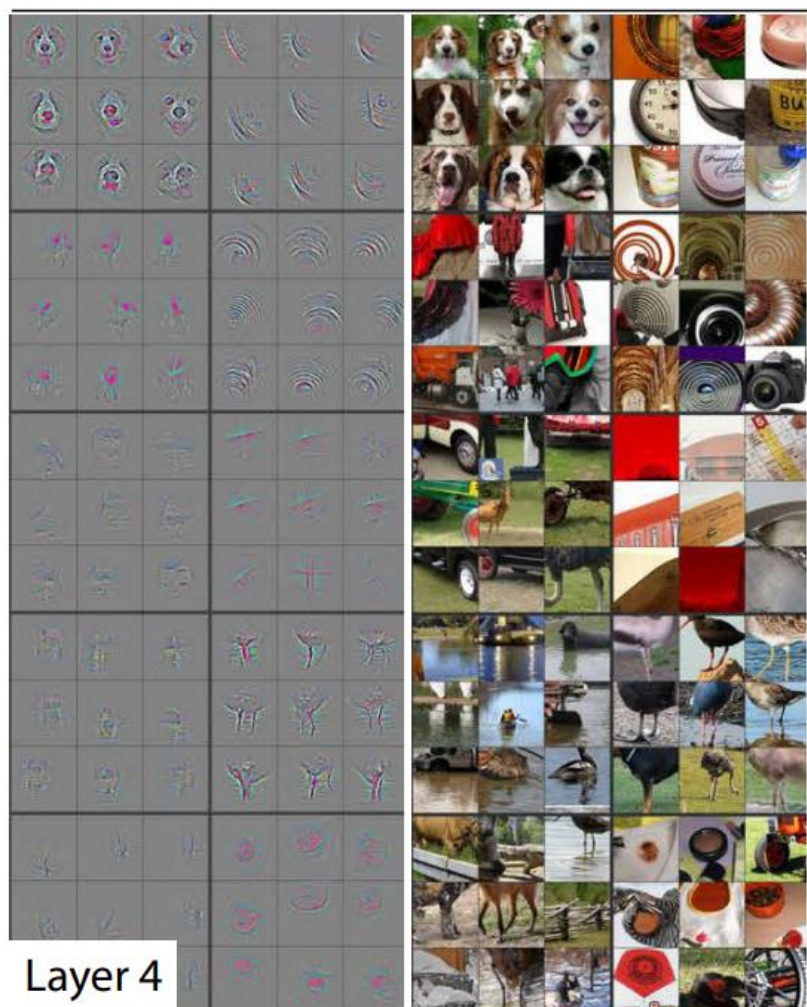


Figure 6 Layer 4

3.3.2 MP3 to Spectrograms

Importantly, these techniques for computer visions are not only useful for recognizing images since there are many areas which can be turned into pictures. For instance, sounds can be displayed pictorially by representing their frequency over time as in Figure 7, which also provides state-of-the-art results at sound detection. The code mp3ToSpectrogram.py on Github [5] was used for the transformation of mp3 data into spectrograms which is an adapted version of [12]. First, the mp3 file will be read using the function audioSegment from the pydub [13] library, 30 seconds extracted and converted to a wav file. Afterwards, the data can be read from this wav file which is an array of length (2500,2). Thus, the left channel was taken since it corresponds to the frequency in Hertz. Finally, the data was plotted using the specgram function of the library matplotlib [14].

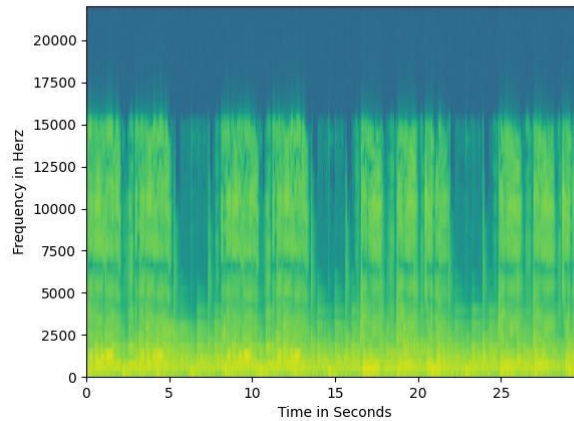


Figure 7 Spectrogram

3.3.3 Construction of the CNN

The following standard approach was used for the construction of the CNN. There is an architecture like Figure 8 which contains parameters and data as inputs. The architecture and the parameters are the model. With the inputs, they calculate the predictions, which are then compared to the labels with the loss function to update the parameters many times and improve them until the loss becomes very low.

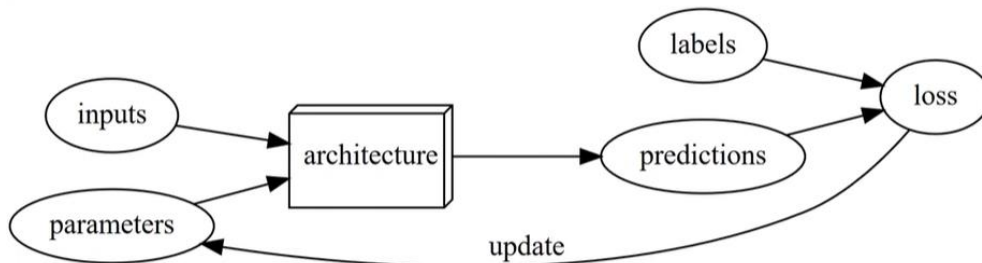


Figure 8 Architecture of CNN

3.3.4 Fast AI

The fast AI [15] library was used to create the CNN model, which simplifies training fast and accurate neural networks using modern best practices. The flexible way to work with any data format is using their DataBlock API, which is represented in Figure 9. First of all, the independent and dependent variables will be given as input. In this case, the images of the song titles are the independent input data and the moods the dependent categories alias labels.

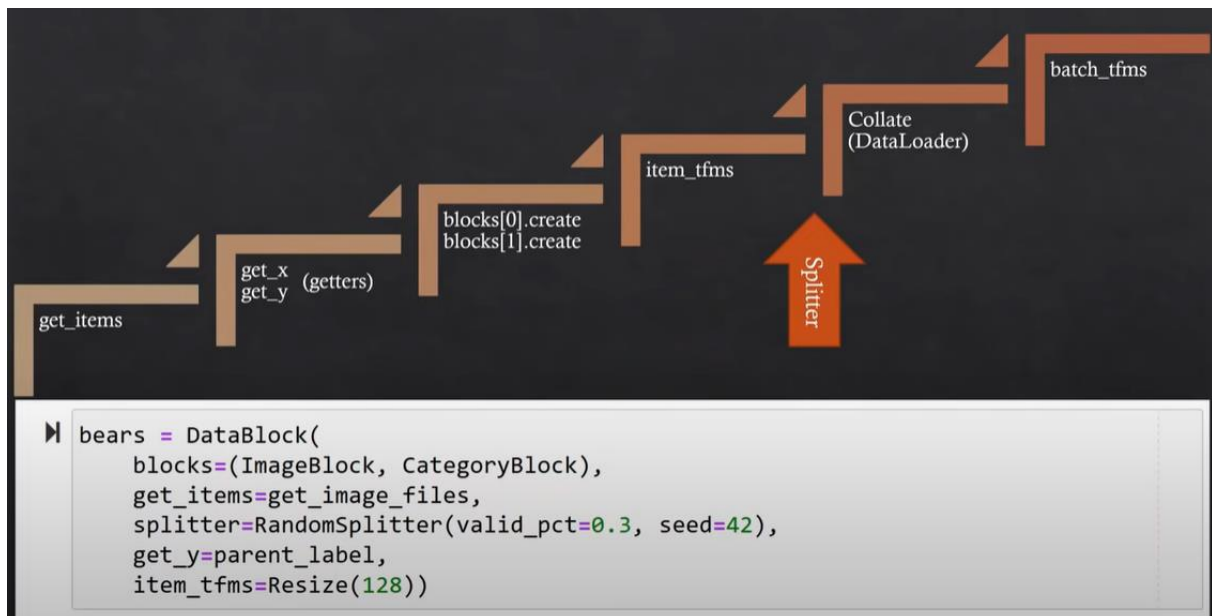


Figure 9 Order of Functions Called by DataBlock

The `get_items` function provides the path to the list of images. The .csv file from the MTG-Jamendo dataset was used to indicate the location of the downloaded images. Normally, the data will be labeled by the parent folder of the image. Since there are more than one mood for each track, the corresponding moods were taken from the column “moods” next to the column “file path” of the .csv file. Afterwards, the split of the data into validation and training set must be given. Here it will be randomly split and 30 % assigned to the validation set. The seed of 42 is to ensure that every time the model is run, the validation set will be the same. The final parameter item transformation would resize the image, which does not make sense for the similar plotted spectrograms. At the end, a data loader will be created, which takes 64 images and put them into a batch to pass them all to the model at once. This procedure allows to speed up the computation.

3.3.5 Multi-Label Classification

As mentioned before, the MTG Jamendo dataset contains for each track multiple moods. Thus, multi-label classification was applied according to [16]. One difference is the way the labels are obtained from the dataset. First, lambda functions were utilized, but an error occurred during the saving process of the model using pickle called “PicklingError: Can't pickle <function <lambda> “. Hence, the ColReader functions solved the problem by reading the space separated moods of the .csv dataset.

The main difference is the evaluation criteria and loss function. The most common way to evaluate traditional classification, such as multi-class problems, is using accuracy according to [Sor10]. Other standard evaluation metrics exist including recall, F-measure, and ROC area that are applied to single label multi-class classification problems. However, in multi-label classification the predictions contain a set of labels which result in fully correct, partially correct (with different levels of correctness) or fully incorrect predictions. Thus, the evaluation of a multi-label classifier is more challenging than one of a single label classifier and these existing evaluation metrics do not capture such notion originally.

The loss function according to [16] was implemented with the use of the package `accuracy_multi` for multi-classification. Furthermore, the loss function called `BCEWithLogitsLossFlat` was utilized since the results of the model are distributed on a scale with a threshold instead of providing one exact prediction. This provides the possibility to retrieve many mood predictions for a song based on sigmoidal activation in the loss function.

3.3.6 Train a Model

First of all, the learning rate must be found using the `learn.lr_find()` function, which provides a graph of loss as a function of the training steps. Then, one can identify a learning rate by observing the graph and finding the section where the loss is decreasing the fastest. Thus, the learning rate that was used at that training step will be taken. For example, in Figure 10, the proposed learning rate is around 0.00005, which should be the best suited learning rate for the underlying data.

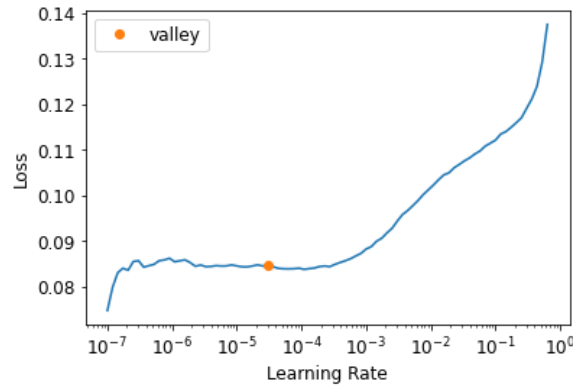


Figure 10 Plot of Loss and Learning Rate

In order to reduce memory usage, the function `learn.to_fp16()` ensures to use 16-bits floats for the arrays representing the inputs, activations and weights instead of the single precision with 32-bit floats of the standard computation in neural nets. According to [17] the FP16 is structured like in Figure 11. Firstly, there is the sign bit providing +1 or -1. Secondly, 5 bits are available to code an exponent between -14 and 15 and the fraction part contains the remaining 10 bits.

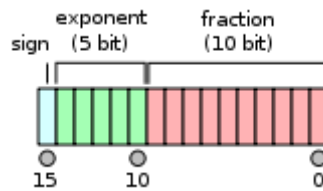


Figure 11 Structure of FP16

There is a smaller range of possible values (approximately $2e-14$ to $2e15$) compared to $2e-126$ to $2e127$ for FP32, but also a smaller offset. For instance, between 1 and 2, the FP16 format only represents the number 1, $1+2e-10$, $1+2*2e-10$... which means that $1 + 0.0001 = 1$ in half-precision. Thus, there exist three problems with this half-precision that can occur and negatively affect the training process.

1. The gradients can underflow. They can simply be changed to 0 due to the fact that they are too low in FP16.
2. Every weight of the network computes $w = w - lr * w.grad$ in order to optimize. The $w.grad$ is often many orders of magnitude below w , which causes the problem to occur. Moreover, the learning rate equals a small number. Thus, the case of $w = 1$ and $lr*w.grad$ is 0.0001 (or lower) is very common, but the update has no impact in those cases. Hence, the weight update is imprecise.
3. The activations or losses may overflow since it is easier to reach nan (or infinity) in FP16 precision. Thus, the training might rather diverge.

The solution is mixed precision training by doing some of the operations in FP16 precision, others in FP32, which solves the problem of imprecise weight updates. For the other two problems, the forward pass and gradient computation are done with half-precision to go fast, but the update is with single precision to improve accuracy. It is acceptable if w and $grad$ are both half floats. However, the operation $w = w - lr*grad$ must be computed in FP32. This ensures that $1 + 0.0001$ will be 1.0001. Thus, a copy of the weights must be kept and saved as a master model. Finally, the training process below will be used.

1. The output with the FP16 model will be computed after the loss
2. The gradients will be backpropagated in half-precision
3. The gradients will be copied in FP32 precision
4. The update on the master model will be completed in FP32 precision
5. Lastly, the master model will be copied in the FP16 model

As a next step, the model was trained by using the function `learn.fit_one_cycle(5, slice(lr))` and according to the steps from fast AI [18]. The first parameter indicates the number of epochs, which is one complete pass through the input data. The second parameter is used to enable different learning rates for each parameter group. This technique is called discriminative layer training and is widely applied in computer vision. The goal is to train the model as long as the validation loss decreases and the accuracy increases. The loss takes a prediction of a song and compares it to the actual value, more specifically the label, to compute the performance. Thus, it works with data not used for training, called a validation set. On the other hand, the training loss checks how well the model fits the data from the training set. By decreasing validation loss, overfitting is prevented, where a model is trained to remember specific features of the data instead of generalizing well to data that was not used in the training process such as the validation set.

It is also important to unfreeze the layers to make them trainable according to [Zei14]. Hence, all the weights from frozen layers of the model can now get updated from their pre-trained state according to the loss function. As mentioned before, the first layers let the model learn low-level features like shapes, curves and other patterns. While the later layers are mainly for capturing high-level features on current dataset for example animals. These are fully connected layers which identify features like the shape of a lion or bird in its entirety. These layers possess

composite or aggregated information from previous layers according to the current data. The information maintained by these layers is improved during the training process of the model and the optimization of the loss according to the target labels.

Afterwards, the model will be trained again with the same function `learn.fit_one_cycle(2, max_lr=slice(1e-6,1e-4))`, but using discriminative learning rates with the parameter `max_lr`. This parameter holds a low learning rate for initial layers since they require less tuning and gradually increases the learning rate for later layers, which require higher tuning. Again `lr_find()` must be used before `fit_one_cycle()` to receive the best suited learning rate for the underlying data.

The weights of all layers will then be saved as stage-1 using `learn.save('stage-1')`. However, the architecture is not saved. Thus, the same architecture must be defined again to use these weights another time, which is clearly not necessary if the training process in a Jupyter Notebook.

3.3.7 CPU vs. GPU

Since there are around 18'000 music titles as input, one epoch took around three hours. Thus, a GPU machine was rented from the website Paperspace [19] to make the model faster and simpler. Furthermore, the CUDA out of memory error in Figure 12 was solved using a GPU because more memory was now available. One epoch was later executed in only six minutes.

```
571
572     def compute_should_use_set_data(tensor, tensor_applied):

/opt/conda/lib/python3.7/site-packages/torch/nn/modules/module.py in _apply(self, fn)
591     # 'with torch.no_grad()':
592     with torch.no_grad():
--> 593         param_applied = fn(param)
594         should_use_set_data = compute_should_use_set_data(param, param_applied)
595         if should_use_set_data:

/opt/conda/lib/python3.7/site-packages/torch/nn/modules/module.py in convert(t)
895         return t.to(device, dtype if t.is_floating_point() or t.is_complex() else None,
896                non_blocking, memory_format=convert_to_format)
--> 897         return t.to(device, dtype if t.is_floating_point() or t.is_complex() else None, non_blocking)
898
899     return self._apply(convert)

RuntimeError: Exception occurred in `TrainEvalCallback` when calling event `before_fit`:
  CUDA out of memory. Tried to allocate 20.00 MiB (GPU 0; 7.94 GiB total capacity; 7.19 GiB already allocated; 12.19 MiB free; 7.21 GiB
reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.  See documentation
for Memory Management and PYTORCH_CUDA_ALLOC_CONF
```

Figure 12 CUDA Out of Memory Error

In order to use the already programmed algorithm in Paperspace, a *ssh* key had to be generated to link the GitHub account by using the code below.

```
1    cd ~/.ssh
2    ssh-keygen -o -t rsa -C name@mail.ch
3    cat id_rsa.pub
```

3.3.8 Results

First of all, the dataset MGT-Jamendo was taken as input with around 18'000 titles and 56 moods. Then the optimal learning rate was found to train the model. Due to the high number of titles, the time for one epoch was around three hours, but it was worth regarding the accuracy of 0.9704 in epoch 4 as shown in Figure 13. It shows the training process of the data in epochs where each image from the training set is trained, and the total loss computed. In the same epoch, the trained model is validated using the validation dataset resulting for example in the accuracy of 0.969765 in the epoch zero.

```
learn.fit_one_cycle(5, slice(lr, 0.007585775572806597/5))
```

✓ 1029m 22.3s

epoch	train_loss	valid_loss	accuracy_multi	time
0	0.113349	0.117693	0.969765	3:06:38
1	0.111569	0.114875	0.969976	3:08:23
2	0.106215	0.111388	0.970246	3:59:54
3	0.095477	0.110839	0.970384	3:21:36
4	0.085647	0.111176	0.970434	3:32:48

Figure 13 Results of Training and Validation

However, this model was then tested with other songs than in the dataset which resulted in bad predictions. All the scores of the moods were below 0.01. Hence, no mood was predicted as it can be seen in Figure 14, which shows the prediction of every mood in an array saved in an TensorBase object.

```
, TensorBase([1.6971e-03, 5.0088e-03, 1.9155e-02, 4.0882e-02, 2.3818e-02, 3.4183e-03, 2.8934e-02, 3.6567e-03, 6.7118e-03, 2.0408e-03, 7.1755e-03, 3.6150e-03, -03, 9.0874e-03, 1.2149e-02, 2.3353e-04, 7.0347e-02, 9.7050e-04, 4.6949e-04, 1.7335e-03, 4.0860e-04, 6.1853e-03, 3.3807e-04, 1.1025e-03, 9.6938e-03, 1.3184e-02, 1.3810e-02, 2.7709e-03, 4.3061e-02, 3.8977e-03, 1.5930e-02, 1.3470e-02, 1.0553e-03, 2.7112e-03, 2.9393e-03, 2.6242e-01, 3.4881e-03, 1.3565e-02, 3.3509e-02, 6.5159e-02, 7.7285e-02, 2.1818e-04, 9.5578e-04, 1.7083e-03, 3.7403e-03, 1.5283e-03, 9.9158e-03]))
```

Figure 14 Scores of Predictions

The assumption was that the first 30 seconds of each song are always similar according to their sound. Therefore, taking the middle part could improve the results and needs to be considered when using audio data for image classification. The accuracy of the predictions increased slowly to 0.9705 as in Figure 15 but the predicted labels were still empty due to the low scores.

```
learn.fit_one_cycle(5, slice(lr))
```

✓ 1007m 33.8s

epoch	train_loss	valid_loss	accuracy_multi	time
0	0.816875	0.531380	0.819134	3:21:03
1	0.231455	0.133001	0.970563	3:19:53
2	0.144988	0.125057	0.970586	3:20:11
3	0.131492	0.121643	0.970585	3:22:50
4	0.127144	0.120032	0.970590	3:23:34

Figure 15 Results for Taking the Middle Part of Songs

The next step was to try to build the model with less data. The accuracy dropped to around 0.55 but the prediction scores were much higher. For instance, for the anime song “Guren no Yumiya TV Size Version” the moods dark (in red with a prediction of 0.9819) and slow (in blue with a

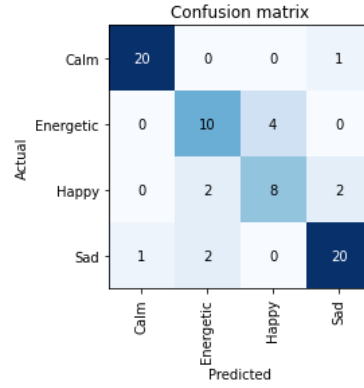


Figure 19 Confusion Matrix

After looking closer to the spectrograms in Figure 20, their resemblance can be recognized. For instance, the density of energetic is clearly higher than the one of happy. While the density of the mood sad is also higher than calm. However, there are still nuances which were removed by continuing training the model.

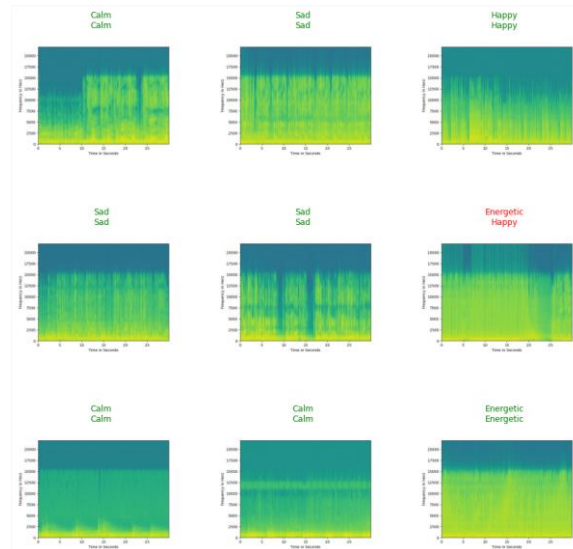


Figure 20 Spectrograms with their Label and Prediction

The reason for the better predictions could be that the new dataset contains fewer but more distinctive labels. Moreover, the dataset MTG-Jamendo also includes acoustic titles, which are seldom present in anime or other songs.

3.3.9 Comparison to KerasClassifier

In [10] a neural network with an input of ten features, one layer with eight nodes, and four outputs with the output layer was constructed. The KerasClassifier was utilized with an activation function corresponding to a Rectified Linear Unit (Relu), the loss function was a logistic function and Adam Gradient Descent algorithm was taken as the optimizer. The same dataset was used to train and validate the data. However, audio features were taken as input of the model instead of the frequency in Hertz. Thus, the accuracy of this model using KerasClassifier, namely 72.75 %, can be compared to some extent to the results from above with a slightly higher accuracy of 78.57 %. Therefore, a hybrid solution using both systems will be needed for the prototype to solve the mentioned nuances between two moods in 3.3.8.

4

Prototype

4.1 Software Architecture	17
4.2 Spotify OAuth	18
4.3 MyAnimeList API	19
4.4 Anime Songs Playlist	20
4.5 Mood Classification	20
4.6 Hosting Firebase	21
4.7 Continuous Deployment	24
4.8 Evaluation	25

4.1 Software Architecture

At the beginning, the software architecture was designed like in Figure 21. The user can log in to their MyAnimeList account via OAuth on the website implemented in Flutter. Then the anime titles will be sent as a JSON object to the backend where the user needs to login to Spotify also via OAuth. Afterwards, the user can get all the songs of their list of favorite anime by pressing on a button. Then a Spotify link will be provided to access the playlist directly on their Spotify account. If the user wants to classify this playlist by mood, one can be chosen and a new playlist containing only songs regarding the desired mood will be returned. Everything will be hosted on Firebase since it is free and compatible with Flutter.

However, at the end, the frontend was also implemented in Flask since there is no need for a complex user interface. Plain HTML and JavaScript can be directly implemented in Flask, which is convenient to render the result of the model. Thus, anime titles are accessed directly in Flask via the MyAnimeList API. Before permission is granted to add a Spotify playlist. The reason why the Spotify authentication occurs first is to also let users without a MyAnimeList account classify their titles by moods in the future. The code of the implemented prototype can be found on the GitHub repository [5].

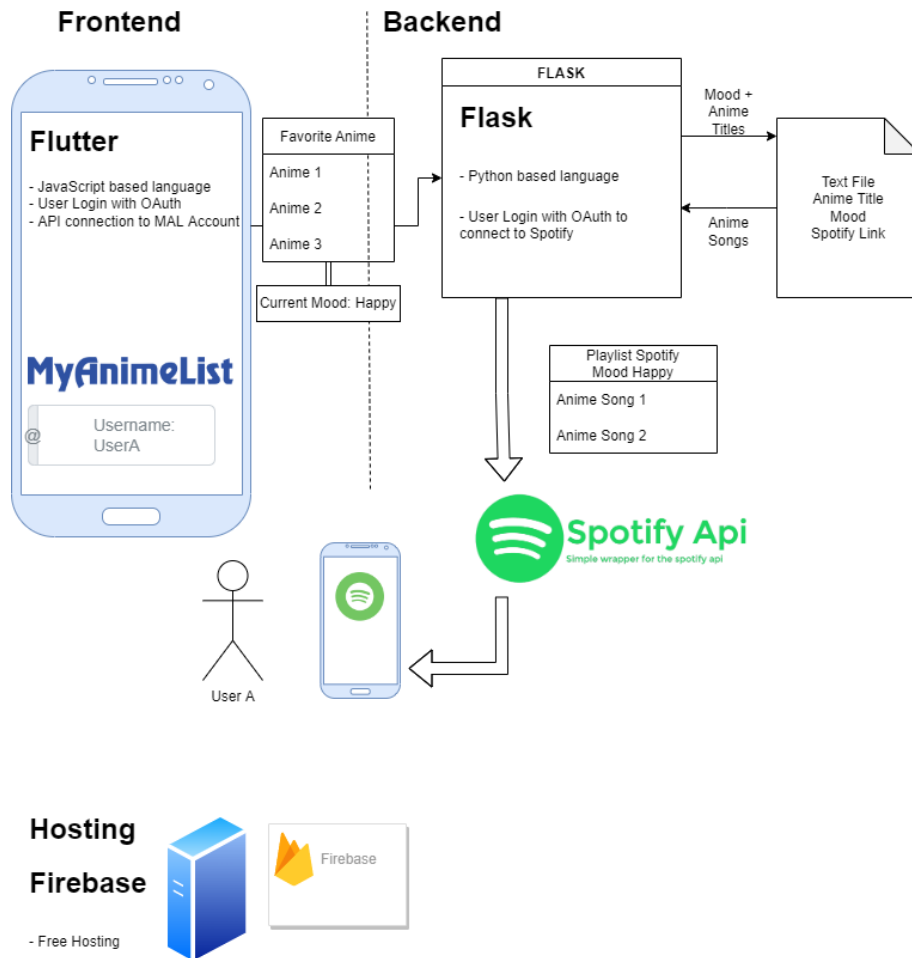


Figure 21 Software Architecture

4.2 Spotify OAuth

As mentioned before, the user will first press the login button to sign in via Spotify OAuth like in Figure 22. The `spotipy` [20] library provides the function `SpotifyOAuth` to let the user log in with their account by using the Authorization Code Flow. This flow was used since it is optimal for long-running applications in which the user grants permission only once. As soon as the user is logged in according to Figure 23, the system offers an access token that can be refreshed. For the Authorization Code Flow a redirect URI must be added to the application in the Spotify developer dashboard.

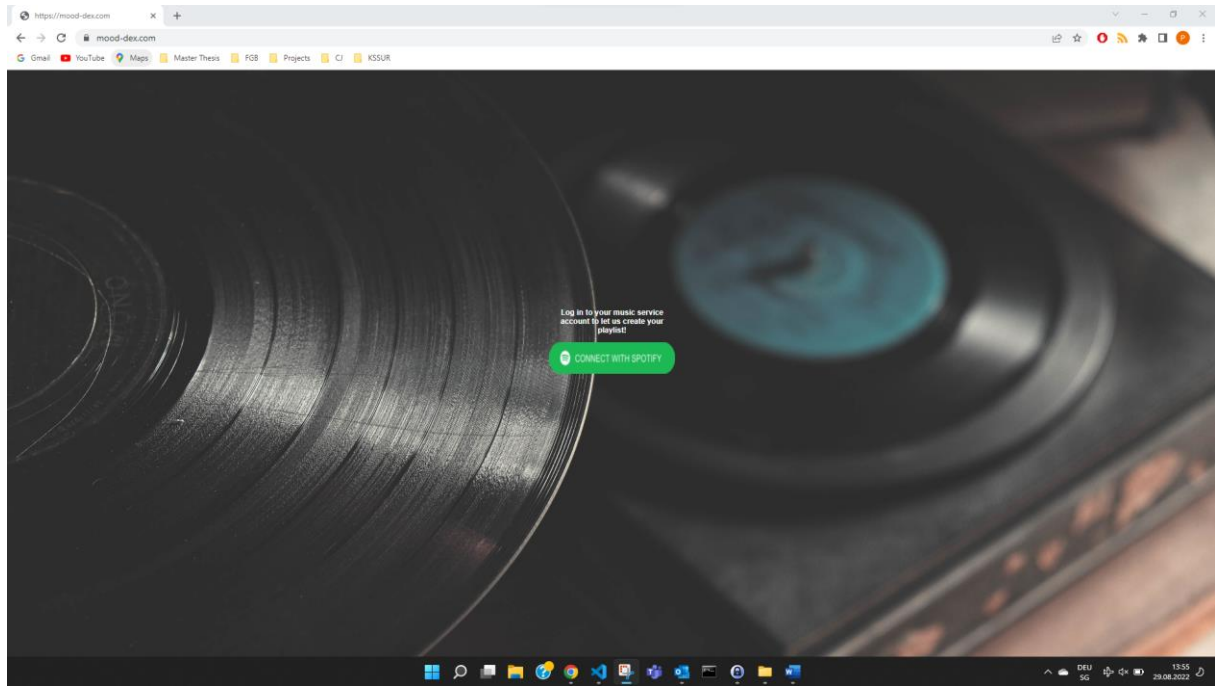


Figure 22 Landing Page of the Web Application

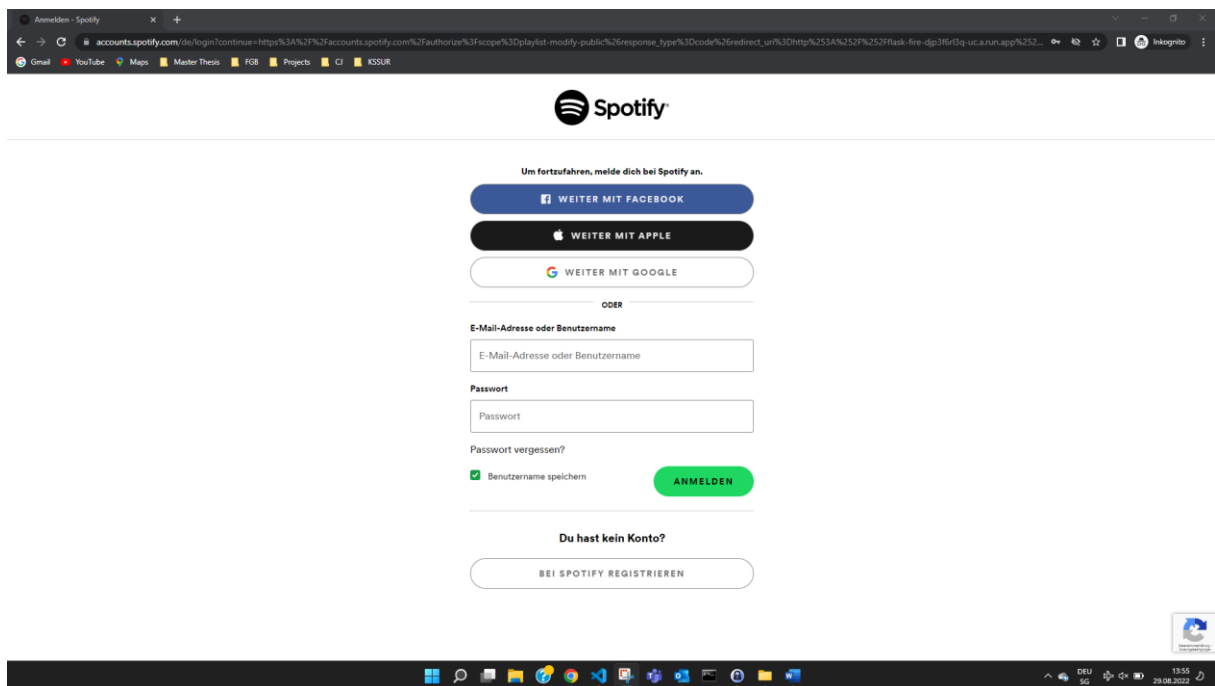


Figure 23 Spotify OAuth Login

4.3 MyAnimeList API

As a next step, the user enters their username on MyAnimeList to get songs of their favorite anime. The API of MyAnimeList [21] provides the ability to get the list of anime watched by a user. Then the IDs of every title will be saved as a list to compare it with the text file

animeSongs.txt, which was created in 2.1, containing the same IDs linked to the Spotify IDs of the corresponding anime songs.

The first intended approach to let the user sign in via OAuth to their MyAnimeList account was discarded since no modification of their list of anime watched is needed. Thus, no additional authentication is necessary. The only disadvantage of this approach lies in the fact that no private anime list can be accessed, but by default they are public. Furthermore, the goal of this website is to share the watched and rated anime titles which requires to have a public profile.

4.4 Anime Songs Playlist

When the user submits their MyAnimeList username, the IDs of the matching anime songs will be saved in a list. After the creation of a new Spotify playlist in the user's account, all these titles of the list will be added to this playlist. Importantly, the program must sleep for a few milliseconds to wait for the creation of the list. Otherwise, the titles cannot be added due to the fact that the playlist ID does not exist yet. The link of the new playlist will be shown like in Figure 24. By clicking on it, the user will be redirected to their new playlist on Spotify where the titles can be listened to immediately.

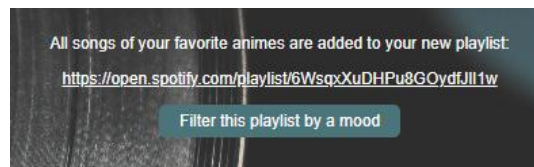


Figure 24 Provided Link of the New Playlist

4.5 Mood Classification

After the user has received their new playlist containing all the anime songs of their list of favorite anime titles, they can classify it by a mood by clicking on the button “Filter this playlist by a mood”. The circumplex model of affect from [Pos05] was taken as an inspiration for the representation of the mood selection because it is more consistent with many recent findings from behavioral, cognitive neuroscience, neuroimaging, and developmental studies of affect. In addition, [Pos05] indicates that basic emotion theories no longer adequately explain the large number of empirical observations from studies in affective neuroscience. The model shows that all affective states result from two fundamental neurophysiological systems where one is related to valence (x-axis) and the other to arousal (y-axis). Thus, each emotion is a linear combination of these two dimensions. Since a hybrid solution was taken, two moods can be proposed for one song. Thus, the moods were displayed as in Figure 25 instead of strictly classifying every song in one of the four moods. For instance, if the image classifier proposes the mood happy for a song and the KerasClassifier suggests calm, then both moods will be saved for this title. Now if the user desires to listen to a happy and calm song, only titles containing these moods will be provided.

As soon as the user clicks on the mood, they want to use to filter the new playlist, the tracks will be saved as a list using the spotipy function `playlist_tracks` with the playlist ID as a parameter. In order to avoid classifying the same title multiple times, a dictionary called `dictTrackMood`, containing all classified titles with the Spotify ID as key and moods as value, was created. Thus, all titles of the current playlist will be searched in this dictionary. If the mood

of one song is already available, the track ID will be saved to a new list `listSpotifyTrackID` for all titles matching the wished mood.

Otherwise, the song must be classified. First, the song details will be downloaded using the track function of `spotipy`. To avoid an expired token, the token will be refreshed after 30 tracks. Then the preview will be downloaded with the help of the preview URL received before with the song details.

If the download is successful, it is saved in the corresponding list `downloaded_successfully`. The mp3 data will then be converted to a spectrogram and given as input to the predict function from the fast AI library. Before, the model and parameters were loaded. The predicted mood will then be saved to the dictionary with the prediction of the `KerasClassifier` using the features of the song. Seldom, a song of Spotify does not provide a preview URL. In this case, the song will be saved to the dictionary `dictTrackMood` with only one mood. If the predicted mood is like the one desired by the user, the track ID will be saved in the list called `listSpotifyTrackID`. Finally, these titles will be added to the Spotify playlist and provided to the user as a link.



Figure 25 Selection of Moods to Filter the Playlist

4.6 Hosting Firebase

Firebase [22] will be used to host the Flask server because it is free for this scale of work and offers a content delivery network (CDN). It is generally the most cost-effective and robust way of hosting, as data is only read and synchronized if necessary. In addition, it is easy to scale. In the past, Firebase did only allow static websites. Today, cloud functions of Google Cloud Run allow hosting of dynamic pages by running containers serverless. Thus, if they can fit into a stateless Docker container, it can be combined with Firebase Hosting. The tutorial of [23] and [24] were followed for the following steps.

First of all, the project structured as in Figure 26 was created where the division of the server and static part is essential because Firebase will only serve the static part while Google Cloud Run is responsible for the dynamic one.

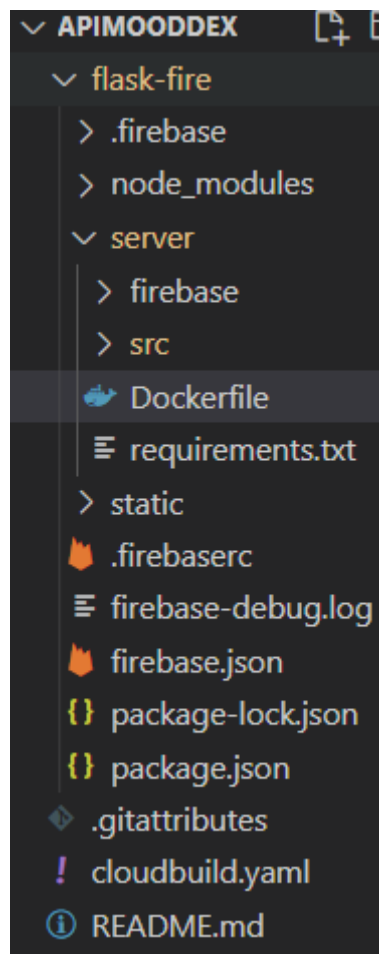


Figure 26 The Structure of the Project

As a next step, a Docker file is created in the folder server, which acts like a recipe for the server environment. The first entry corresponds to the base image of the container. In this case, it is alpine 3.6 to install gcloud and change the limited runtime of gcloud of 600 to 3600 seconds as it can be seen in Figure 27. The reason why the runtime had to be updated was the long duration of the package installations.

The second base image of the container set is python 3.7 like in Figure 28. The packages will be installed with the commands RUN pip install. Then the server's source files will be copied into a folder within the container named app and the working directory set to this app folder. Subsequently, an environment variable for the port will be set for 5000 and gunicorn bound to this port.

```
1 FROM alpine:3.6
2
3 # Change the limited timeout of Gcloud to 3600s
4 RUN apk add --update \
5 python \
6 curl \
7 which \
8 bash
9 RUN curl -sSL https://sdk.cloud.google.com | bash
10 ENV PATH $PATH:/root/google-cloud-sdk/bin
11
12 RUN gcloud config set app/cloud_build_timeout 3600s
```

Figure 27 Docker File Part 1

```

14 # Install packages
15
16 FROM python:3.7
17
18 RUN pip install Flask gunicorn
19
20 RUN pip install spotipy pandas pydub matplotlib scipy fastbook ipywidgets
21 RUN pip install numpy tensorflow sklearn keras ffmpeg-python
22
23 COPY src/ /app
24 WORKDIR /app
25
26 ENV PORT 5000
27
28 CMD exec gunicorn --bind :$PORT --workers 1 --threads 8 app:app

```

Figure 28 Docker File Part 2

Then a project must be created on Firebase to execute the container with Cloud Run. For this, the container must be stored in Google Cloud's Container Registry (GCR). It will be sent to Cloud Build where it will be built and uploaded to GCR. This approach is much simpler than uploading the local container, which requires Docker, and is less suited to the continuous deployment discussed in the next chapter. The following commands will be used to initialize, build, and deploy the container inside the server directory.

```

4 gcloud init
5 gcloud builds submit --tag gcr.io/<project-id>/flask-fire
6 gcloud beta run deploy --image gcr.io/<project-id>/flask-fire

```

The Firebase Hosting needs to be set up by installing Firebase CLI via npm. Then the following commands can be run to create a package.json file and install the Firebase tools.

```

7 npm init -y
8 npm install -D firebase-tools

```

Within the new created folder called node modules, the next command starts the set-up of the Firebase Hosting within the project. After the completion of the set-up, there are two files in the root called .firebaserc and firebase.json in the root of the project.

```

9 ./node_modules/.bin/firebase init hosting

```

Firebase.json helps to connect to Cloud Run for every incoming request matching the provided URL in this file, which is defined as everything. Thus, it will invoke Cloud Run, which returns the container. To deploy it, the command below will be used.

```

10 firebase deploy

```

4.7 Continuous Deployment

The continuous deployment was added to automatically deploy the container set up above every time the code is updated and pushed to Github. [25], [26] and [27] were taken as guide to set up the continuous deployment. Continuous integration was not used since the application was implemented by only one person. Thus, it is not needed since the developer has made a habit of running tests before committing and the code functionality of the code is not complex enough to require automated tests.

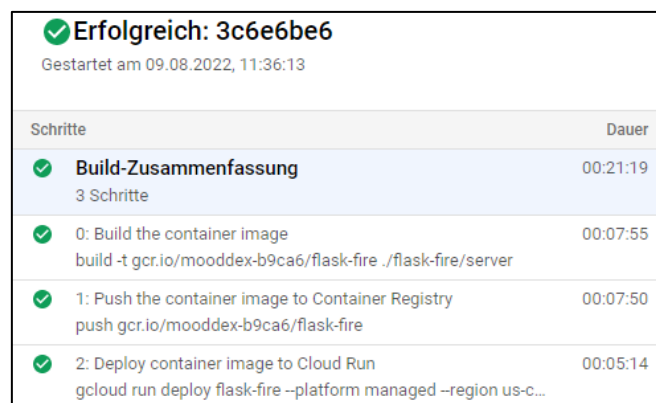
First of all, a unique firebase *ci* token is required to deploy the web app automatically. Since this token cannot be stored safely inside the code, the Google Cloud Secret Manager will be used to provide the token on the runtime of Cloud Build trigger. The token will be accessed with the following code.

```
11  firebase login:ci
```

A secret was then created in the Google Cloud Console with the provided unique Firebase token as the secret value. The same Docker file as in Chapter 4.6 was taken to build the Docker image only once and push it to Google Cloud Run. Later, this Docker image was always used to start a container and run the necessary commands.

A yaml file was created acting as a build configuration file and defining the fields that are necessary for Cloud Build to perform the instructions of the continuous deployment. This configuration file starts with a timeout field to ensure that the building time can be longer as the maximum of Google Cloud Run of 600 seconds. The file is structured in three steps. For each step, Cloud Build spin up and later spin down the Docker container once the task completed. Every step contains a unique identifier and a name to specify cloud-builders or Docker container images. Furthermore, the args field receives a list of arguments and passes them to the builder referenced by the name field. The field entrypoint specifies an entry point. Per default bash is installed. As it can be seen in Figure 29, the first step consists of building the container image. The next one is pushing it to the container registry to store the built image there. Finally, it will be deployed on Google Cloud Run.

Then a trigger must be added by linking Github with the Google Cloud project using Cloud Build. The Google Cloud platform must provide the authorization to access the Github account in order to select the desired repository. The file cloudbuild.yaml will be given as the location for the configuration file.



The screenshot shows a successful build and deployment summary. At the top, it says 'Erfolgreich: 3c6e6be6' with a green checkmark and 'Gestartet am 09.08.2022, 11:36:13'. Below this is a table with two columns: 'Schritte' (Steps) and 'Dauer' (Duration). The table lists four steps, all marked with green checkmarks. The first step is 'Build-Zusammenfassung' (3 Schritte) with a duration of 00:21:19. The second step is '0: Build the container image' with a duration of 00:07:55. The third step is '1: Push the container image to Container Registry' with a duration of 00:07:50. The fourth step is '2: Deploy container image to Cloud Run' with a duration of 00:05:14.

Schritte	Dauer
✓ Build-Zusammenfassung 3 Schritte	00:21:19
✓ 0: Build the container image build -t gcr.io/mooddex-b9ca6/flask-fire ./flask-fire/server	00:07:55
✓ 1: Push the container image to Container Registry push gcr.io/mooddex-b9ca6/flask-fire	00:07:50
✓ 2: Deploy container image to Cloud Run gcloud run deploy flask-fire --platform managed --region us-c...	00:05:14

Figure 29 Build Summary of the Continuous Deployment

As soon as the last step is finished, the container image is deployed and a link [28] for the web application is provided as it can be seen in Figure 30.

```

15128 Step #2 - "Deploy container image to Cloud Run": Deploying container to Cloud Run service [flask-fire] in project [mooddex-b9ca6] region [us-central1]
15129 Step #2 - "Deploy container image to Cloud Run": Deploying...
15130 Step #2 - "Deploy container image to Cloud Run": Creating Revision.....done
15131 Step #2 - "Deploy container image to Cloud Run": Routing traffic.....done
15132 Step #2 - "Deploy container image to Cloud Run": Done.
15133 Step #2 - "Deploy container image to Cloud Run": Service [flask-fire] revision [flask-fire-00007-kiy] has been deployed and is serving 100 percent of traffic.
15134 Step #2 - "Deploy container image to Cloud Run": Service URL: https://flask-fire-djp3fer13q-uc.a.run.app
15135 Finished Step #2 - "Deploy container image to Cloud Run"
15136 PUSH
15137 DONE

```

Figure 30 Finished Step Two With the Link to Access the Hosted Web Application

4.8 Evaluation

The final application was evaluated by taking anime songs from a human made playlist on Spotify categorized by a mood, for instance “Anime sad vibes song” [29]. Then this playlist’s ID was given as input in the backend, which categorized 14 of 24 titles as sad. Thus, 58.33 % of the songs were predicted correctly. However, not all titles can actually be categorized as sad after listening to them. Thus, another Spotify playlist with more songs was tested called “Sad Anime Soundtracks” [30] which resulted in 47 of 187 songs as shown in Figure 31. This result is worse with a success rate of 25.13 %, which can be explained by the fact that the more songs are added, the less carefully they are selected.

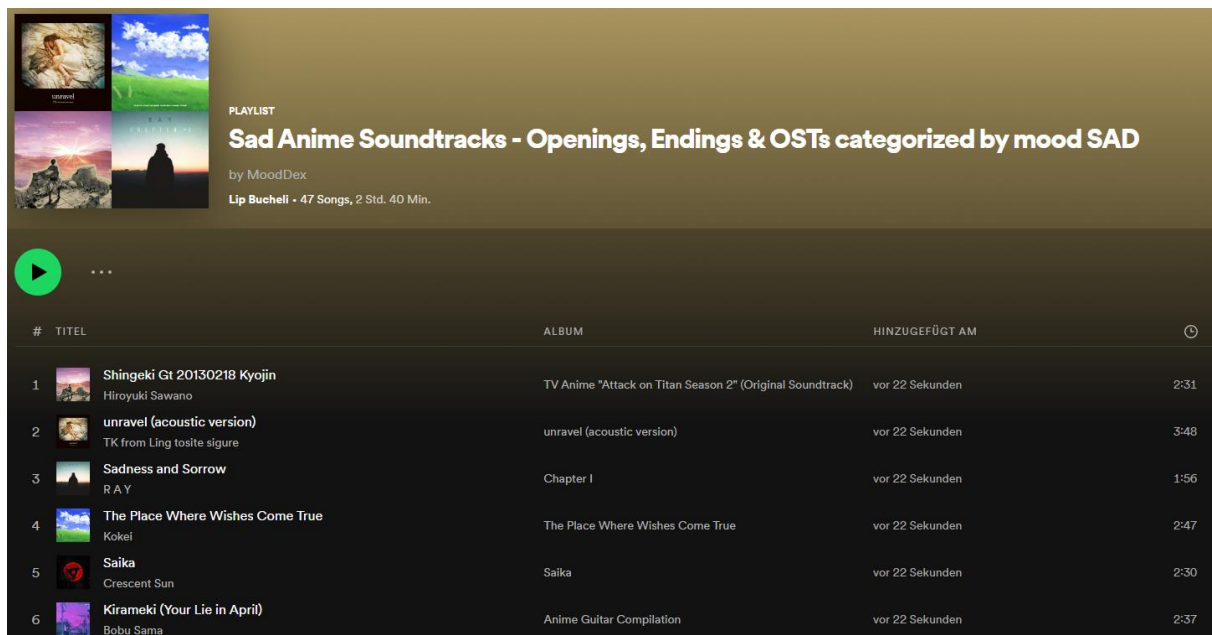


Figure 31 Songs Predicted as Sad

The same procedure was repeated for the other moods. For the playlist “Happy anime songs” [31] 13 of 30 songs were categorized as happy which is a success rate of 43.34 %. For the next playlist “Anime openings that make me happy” [32] a much lower rate of 22.73 % was achieved with 10 of 44 anime songs predicted properly.

For the energetic playlist called “Energetic Anime Songs” [33] 19 of 26 songs were predicted correctly leading to a success rate of 73.07 %. Another playlist called “Energetic AF Anime Songs” [34] had a success rate of 82.14 % where 23 of 28 titles were predicted as energetic as it can be seen in Figure 32. Generally, it was observed that anime songs are mostly predicted as energetic.

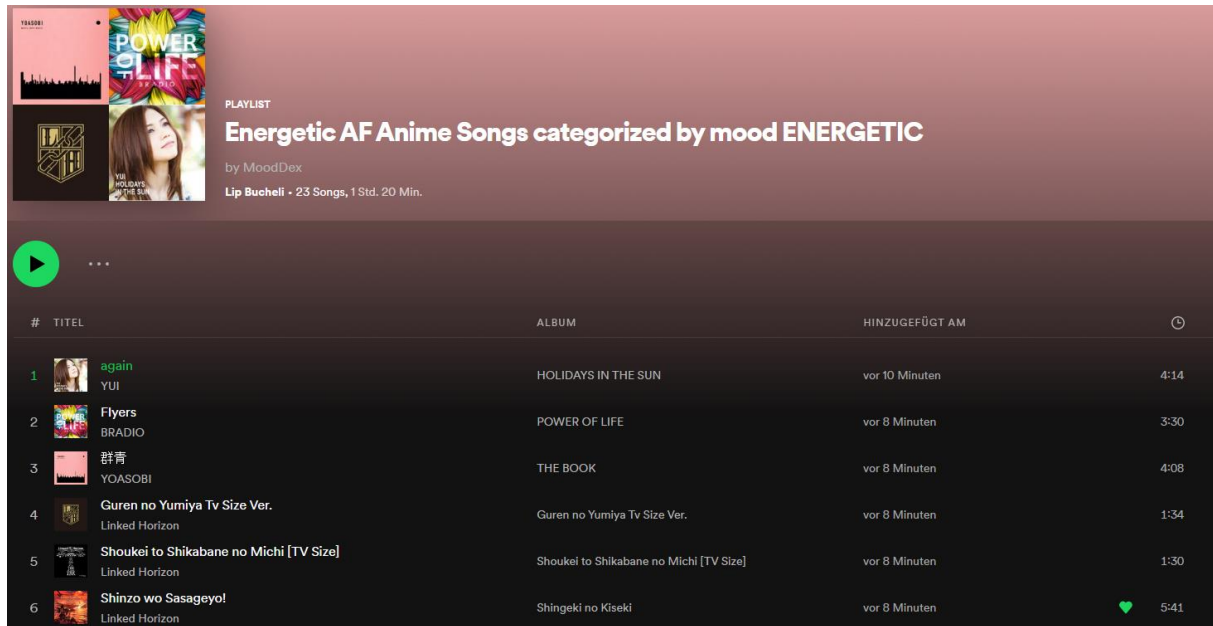


Figure 32 Songs Predicted as Energetic

The mood calm was tested with the playlist “Calm Anime songs” [35] where 9 of 25 songs were predicted as calm leading to a success rate of 36 %. Another playlist called “Calm Anime Mix” [36] provided 25 of 50 songs giving a solid rate of 50 % as shown in Figure 33.

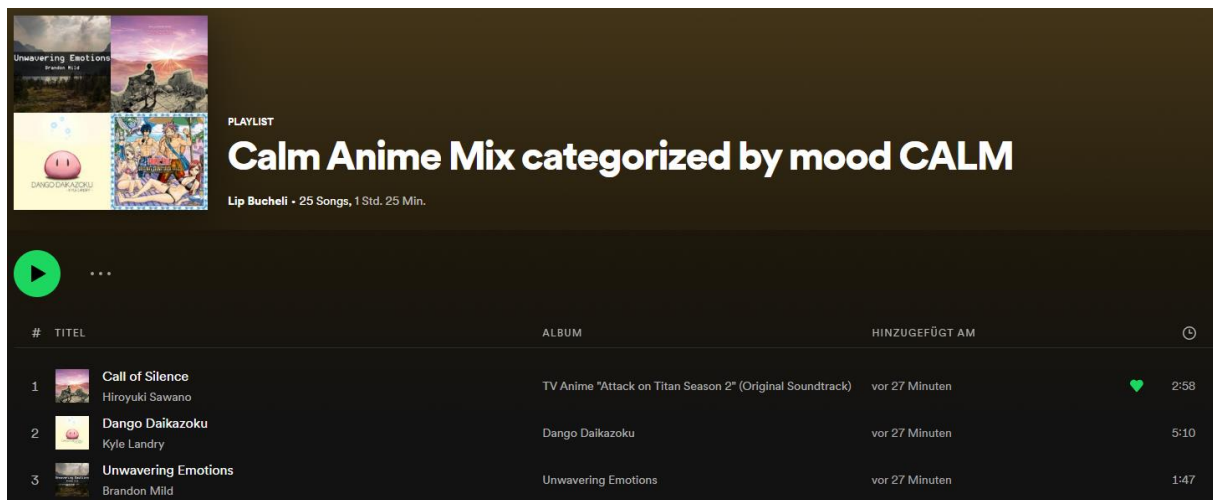


Figure 33 Songs Predicted as Calm

To sum up, the energetic mood was predicted the most accurate, followed by sad, calm and then happy.

5

Conclusion

The first chapter described how the connection between the anime titles and their songs from Spotify was built. Furthermore, the process of the web scraping was defined.

The second section dealt with the mood prediction. Four different datasets were prepared, but only two were fully accessible. The first dataset did not provide moods and the second one did not provide song features. While the third dataset included multi-labels, the fourth consisted of four single-labeled moods. As a next step, the moods were chosen according to their usefulness.

For the mood classification, the image classification using convolutional neural networks was considered as the best approach after reading several papers. Thus, the way the CNN works was described and then applied by converting songs into spectrograms. For the construction of the CNN, the library fast AI was taken which improves the training process and results in accurate neural networks. Their DataBlock API allows working with any dataset. As mentioned before, the two different approaches for single- and multi-labeled classification were discussed and tested. Furthermore, the main differences between their implementation were pointed out.

Not only the construction of the model was explained, but also the training process. First, the learning rate had to be found by observing the graph of loss against training steps and finding the part where the loss is decreasing the fastest. In addition, the function `learn.to_fp16()` was applied to use 16-bits floats for the arrays representing the inputs, activations, and weights instead of the single precision with 32-bit floats of the standard computation in neural nets to reduce memory usage. As a next step, the model was trained regarding the learning rate. This process was repeated many times until the best possible accuracy was reached. Since the datasets consisted of many song titles and the epochs lasted for hours, the usage of a GPU was required. Moreover, the CUDA out of memory error was one more reason to rent a GPU machine on Paperspace.

The prediction scores of the model trained with multi-labeled data and tested with a different test dataset was not higher than 0.01. Therefore, the assumption that the first 30 seconds of a song always sound similar for every song was tested by taking the middle part which improved

the results slightly. However, the predictions were still not accurate and their score too low. Hence, a smaller dataset with single-labeled songs was taken into consideration. The results of the multi-labeled model with the validation set were better than the ones of the single-labeled model. However, the latter one achieved much better results with a different test dataset. At the end, this result was compared to another already implemented neural network using KerasClassifier with audio features. The same training and validation set was used to compare these models. The accuracy of the image classifier was 78.57 %, thus, slightly better than the KerasClassifier's 72.75 %. Hence, a hybrid solution using both systems was implemented for the prototype.

In the last section, the prototype was constructed and described step by step. The first step describes how the Spotify OAuth was implemented to receive the authorization to create new playlists of the favorite anime titles. Afterwards, the MyAnimeList API was accessed to receive the list of anime titles. Here it was also foreseen to use the OAuth of MyAnimeList, but no modifications of these lists were necessary, and the API was preferred to avoid a second time-consuming authentication of the user. The new playlist of the songs of the favorite anime titles can be accessed in the user's Spotify account over a provided link. Thereafter, this playlist can be classified by combinations of four moods represented like the circumplex model of affect. The implementation using the hybrid solution mentioned before was explained in this section. Firebase was chosen to host the web application. Since it allows hosting only static sites, Google Cloud Run was also utilized to run the dynamic part. A Docker file was created to build and deploy the container. Continuous deployment was also implemented to enable deployment automatically as soon as the code gets updated on GitHub using a yaml file for the instructions.

Finally, the prototype was evaluated by taking anime songs from a human made playlist on Spotify categorized by a mood, for instance "Energetic Anime Songs". Then this playlist's ID was given as input in the backend, which categorized 19 of 26 titles as energetic leading to a success rate of 73.07 %. In general, it was recognized that anime songs were mostly predicted as energetic. In summary, the energetic mood was predicted the most accurate, followed by sad, calm and lastly happy.

The future work would involve first re-evaluating this prototype with test users in order to gather a broader evaluation. In addition, this model was not fully tested with other songs but anime titles. Thus, if the success rates of other tracks are much higher, transfer learning to anime songs can be applied by generating a self-made dataset of anime titles labeled with the combinations of the four moods. The service of classifying all playlists of a Spotify user can be done in the future by accessing all playlists and displaying them as selection. Then, the user can choose which one to filter by mood. Moreover, the functionality of a comparison of the anime

or Spotify profiles of two users can be added to the web application. In addition, the same titles can then be provided as a new list to share.

References

[Kor20]

Filip Korzeniowski, Oriol Nieto, Matthew McCallum, Minz Won, Sergio Oramas, Erik Schmidt. *Mood Classification Using Listening Data*. Paperswithcode, 2020. [Retrieved July 01, 2022, from <https://paperswithcode.com/paper/mood-classification-using-listening-data>].

[Tan21]

Hao Tan. *Semi-supervised music emotion recognition using noisy student training and harmonic pitch class profiles*. Paperswithcode, 2021. [Retrieved July 10, 2022, from <https://paperswithcode.com/paper/semi-supervised-music-emotion-recognition>].

[Nuz21]

Michael Nuzzolo. *Music Mood Classification*. Tufts, 2021. [Retrieved July 25, 2022, from <https://sites.tufts.edu/eeseniordesignhandbook/2015/music-mood-classification/>].

[Gav19]

Pralhad Gavali, J. Saira Banu. *Chapter 6 - Deep Convolutional Neural Network for Image Classification on CUDA Platform, Deep Learning and Parallel Computing Environment for Bioengineering Systems*, Academic Press, 2019, ISBN 9780128167182. [Retrieved July 26, 2022, from <https://www.sciencedirect.com/science/article/pii/B9780128167182000130>].

[Kri12]

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012. [Retrieved July 28, 2022, from <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>].

[Zei14]

Zeiler, M.D., Fergus, R. *Visualizing and Understanding Convolutional Networks*, Springer, 2014, ISBN 978-3-319-10589-5. [Retrieved July 27, 2022, from https://link.springer.com/chapter/10.1007/978-3-319-10590-1_53].

[Sor10]

Sorower, Mohammad S. "A literature survey on algorithms for multi-label learning." Oregon State University, Corvallis 18. 2010. [Retrieved July 29, 2022, from <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.364.5612&rep=rep1&type=pdf>].

[Pos05]

Posner J, Russell JA, Peterson BS. "The circumplex model of affect: an integrative approach to affective neuroscience, cognitive development, and psychopathology." Dev Psychopathol. 2005. [Retrieved July 30, 2022, from <https://pubmed.ncbi.nlm.nih.gov/16262989/>].

Referenced Web Resources

- [1] MyAnimeList. <https://myanimelist.net/> (accessed on June 07, 2022).
- [2] Spotify. <https://open.spotify.com/> (accessed on June 07, 2022).
- [3] AnimeMusic. <https://open.spotify.com/> (accessed on June 07, 2022).
- [4] Selenium. <https://open.spotify.com/> (accessed on June 07, 2022).
- [5] Github Repository of this Project called API mooddex.
<https://open.spotify.com/> (accessed on June 07, 2022).
- [6] AllMusic. <https://www.allmusic.com/> (accessed on June 08, 2022).
- [7] GitHub: Audio Music Mood Classification.
<https://neokt.github.io/projects/audio-music-mood-classification/> (accessed on June 08, 2022).
- [8] Gracenote. <https://www.gracenote.com/web-api> (accessed on June 08, 2022).
- [9] MTG dataset. <https://github.com/MTG/mtg-jamendo-dataset> (accessed on June 08, 2022).
- [10] Predicting the Music Mood of a Song with Deep Learning.
<https://towardsdatascience.com/predicting-the-music-mood-of-a-song-with-deep-learning-c3ac2b45229e> (accessed on July 01, 2022).
- [11] Fast AI: Deep Learning for Coders.
<https://course20.fast.ai/videos/?lesson=2> (accessed on July 03, 2022).
- [12] Stackoverflow: Another very simple way of plotting spectrogram of mp3 file.
<https://stackoverflow.com/questions/15311853/plot-spectrogram-from-mp3> (accessed on July 10, 2022).
- [13] Pypi Pydub Library. <https://pypi.org/project/pydub/> (accessed on July 10, 2022).
- [14] Matplotlib. <https://matplotlib.org/> (accessed on July 10, 2022).
- [15] Fast ai. <https://docs.fast.ai/> (accessed on July 11, 2022).
- [16] Walk with Fast AI: Lesson 3 - Multi-Label Classification.
https://walkwithfastai.com/Multi_Label (accessed on July 13, 2022).
- [17] Fast AI: Mixed precision training.
<https://docs.fast.ai/callback.fp16.html> (accessed on July 15, 2022).

-
- [18] Forum Fast ai. <https://forums.fast.ai/t/why-do-we-need-to-unfreeze-the-learner-everytime-before-retarining-even-if-learn-fit-one-cycle-works-fine-without-learn-unfreeze/41614/5> (accessed on July 15, 2022).
- [19] Paperspace. <https://www.paperspace.com/> (accessed on July 16, 2022).
- [20] Spotipy. <https://spotipy.readthedocs.io/en/master/> (accessed on July 16, 2022).
- [21] MyAnimeList API. https://myanimelist.net/apiconfig/references/api/v2#operation/users_user_id_animelist_get (accessed on July 26, 2022).
- [22] Firebase. <https://firebase.google.com/> (accessed on July 28, 2022).
- [23] Hosting Flask servers on Firebase from scratch. <https://medium.com/firebase-developers/hosting-flask-servers-on-firebase-from-scratch-c97cfb204579> (accessed on July 28, 2022).
- [24] Deploy Python on Firebase Hosting with Cloud Run - Firecasts. https://www.youtube.com/watch?v=t5EfITuFD9w&ab_channel=Firebase (accessed on July 28, 2022).
- [25] Firebase CI/CD Using Google Cloud Build. https://www.youtube.com/watch?v=MijHRE5h6Gg&ab_channel=CodingCatDev (accessed on July 30, 2022).
- [26] Firebase Continuous Integration and Deployment (CI/CD) with Google Cloud Build. <https://faun.pub/firebase-continous-integration-and-deployment-ci-cd-with-google-cloud-build-78ada9333cb9> (accessed on July 31, 2022).
- [27] Deploying to Cloud Run using Cloud Build. <https://cloud.google.com/build/docs/deploying-builds/deploy-cloud-run> (accessed on July 31, 2022).
- [28] The link of the implemented web app. <https://flask-fire-djp3f6rl3q-uc.a.run.app/> (accessed on July 31, 2022).
- [29] Spotify Playlist “Anime sad vibes song”. <https://open.spotify.com/playlist/6bxDvjDWiuWw8wapvZXawp> (accessed on August 01, 2022).
- [30] Spotify Playlist “Sad Anime Soundtracks”. <https://open.spotify.com/playlist/75lQcEWNNNoaTbRunjchEot> (accessed on August 01, 2022).
- [31] Spotify Playlist “Happy anime songs”. <https://open.spotify.com/playlist/0iMDhYAmc2nlIgk7kNVKAF> (accessed on August 01, 2022).
- [32] Spotify Playlist “Anime openings that make me happy”. <https://open.spotify.com/playlist/4p6HdSdmjrDOlRZh20YNey> (accessed on August 01, 2022).

-
- [33] Spotify Playlist “Energetic Anime Songs”.
<https://open.spotify.com/playlist/2ProTVhOAtFFhTLVTUF6tu>
(accessed on August 01, 2022).
- [34] Spotify Playlist “Energetic AF Anime Songs”.
<https://open.spotify.com/playlist/20qKzLiRmzTn5qXepMWtOh>
(accessed on August 01, 2022).
- [35] Spotify Playlist “Calm Anime songs”.
<https://open.spotify.com/playlist/5pJtsiZrHw7BF1Hm59s25y>
(accessed on August 01, 2022).
- [36] Spotify Playlist “Calm Anime Mix”.
<https://open.spotify.com/playlist/37i9dQZF1EIfHze1wC0geS>
(accessed on August 01, 2022).