CS4347 Final Project - Transonify (Group 1)

Akanksha Bansal
National University of Singapore
Singapore
akanksha_bansal@u.nus.edu

Justin Tzuriel Krisnahadi
National University of Singapore
Singapore
justin.tzuriel@u.nus.edu

Max Ng Kai Shen
National University of Singapore
Singapore
maxngkaishen@gmail.com

ABSTRACT

Music plays a significant role in today's day and age. It has found various uses from clinical treatments to being used as a recreational activity by people. Automatically transcribing lyrics and producing the melody transcription for songs are becoming tasks of paramount importance. An application which can transcribe lyrics and melody can be useful in many real-world applications. It can be used by songwriters and musicians, or for creating automatic Karaoke applications. With this objective in mind, our team experimented and implemented models which can be used to automatically generate lyrics, melody transcription and music score. In this academic report, we provide our work, which is evaluated using metrics like Word Error Rate and F1-score.[1]

KEYWORDS

automatic lyrics transcription, automatic melody transcription, neural networks, deep learning, web application

ACM Reference Format:
Akanksha Bansal, Justin Tzuriel Krisnahadi, and Max Ng Kai Shen. 2022. CS4347 Final Project - Transonify (Group 1). In *Proceedings of Final Project (CS4347 AY21/22 Semester 2).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

1.1 Motivation

With technological advancements, tasks which were previously handled manually by humans are getting automated. Automation is turning out to be the need of the hour. One such area where a lot of progress has been made with advancements in technology has been music. Tasks like automatic music score generation, automatic lyrics transcription, emotion recognition in music are challenges which are of paramount importance and are being actively worked on. We focus on two areas in this academic report: Automatic Lyrics Transcription (ALT) and Automatic Melody Transcription (AMT). An application which takes as input a song and produces the lyrics, melody and score can prove to be useful to songwriters and singers. It can be used to quickly transcribe musical inspirations into lyrics and melody transcriptions and singers can also use it to identify the notes they are singing.

1.2 Problem Statement

Studies show that recognition of sung words from audio samples is a challenging task even for humans, and this task is more difficult than speech recognition. Singing and speech differ in the pitch, duration of phonemes etc. and hence Automatic Speech Recognition (ASR) models perform badly on singing data. This creates a need to adapt the ASR systems for singing data. Training of ASR models is a computationally expensive and intense task. In this report, we experiment with fine-tuning existing ASR models on a monophonic singing dataset to achieve the task of Automatic Lyrics Transcription.

Automatic melody transcription for singing voice remains a challenging problem due to the wide variability in genre and vocal timbre; another layer of complexity is introduced when background sounds interfere with the clarity of the vocal track. Given an excerpt of music in audio form, the task is to produce a melody transcription and music score in symbolic form. In this paper, we propose our solution to address AMT for monophonic and polyphonic typical Western music in the English language.

1.3 Plan of Paper

The remainder of this paper is structured as follows. Section 2 synthesises the related work in this domain. Section 3 gives an overview of the empirical models implemented in this paper. Section 4 provides in-depth details related to the methodology and experiments carried out for the purpose of Automatic Lyrics Transcription. Section 5 provides in-depth details related to the experimental setup and implementation for the task of Melody Transcription. Section 6 presents the approach used for creating a web application. Section 7 summarises the contribution, discusses the limitations of our application and indicates the main directions for future work. Finally, Section 8 highlights the roles and responsibilities for each of the team members for this project and Section 9 acknowledges individuals who contributed towards the successful completion of this project.

2 LITERATURE REVIEW

Automatic lyrics transcription is a well researched area. A lot of work has already been done to propose solutions for this problem. We look at some of the most recent works which have achieved milestone results in this area. We reviewed a lot of papers for gaining an understanding of the current relevant work, but present the two papers most closely associated with our problem of ALT.

In their paper, Demirel et al [4] propose an approach for monophonic and polyphonic music lyrics transcription using a compact variant of Multistreaming Time-Delay Neural Network (MTDNN)

[1]Link to the codebase: https://github.com/justintzuriel/transonify

CS4347 AY21/22 Semester 2, National University of Singapore, Singapore

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$00.00
https://doi.org/XXXXXXX.XXXXXXX

referred to as MSTRE-Net. The streams of an audio sample are diversified by having different number of layers with the goal of reducing the number of trainable parameters. They also propose to train the system on both monophonic and polyphonic music simultaneously. They also leverage a lexicon model to identify words in singing data which are out of vocabulary. The model proposed by these authors is built from scratch and requires a lot of computational resources. In our approach, we propose to make use of pre-trained ASR models which have proven to perform very well for the task of ASR. This helps reduce the computational cost associated with training robust models for the purpose of ALT.

Chitralekha Gupta et al [7] in their paper "Automatic Lyrics Alignment and Transcription in Polyphonic Music: Does Background Music Help?" proposed a system that automatically provides word-level alignment of a given lyrics text when the input is a polyphonic music audio file. They train acoustic models using vocal extracted data and polyphonic data to test whether background music help with the Automatic Lyrics Alignment and Transcription task. The ASR system employed by the authors for the purpose of their research is based on the Kaldi ASR toolkit which is deemed to be complicated to use and setup. Due to computational constraints the approach could not be reproduced by us as execution of the code needs around 20 GB RAM.

When it comes to melody transcription for singing voice, there are a variety of papers and solutions utilizing different approaches to solve this problem. While we reviewed quite a few papers for inspiration for our final solution, we list the two most important papers here for brevity.

Jui-Yang Hsu, & Li Su [3] proposed a method to utilize semi-supervised learning (SSL) to process unlabelled data to train a specialized singing voice transcription model. In addition, the paper illustrates the utilization of singing voice separation tools (demucs) to remove accompaniments of the input before inference. However, we found that the SSL process to be rather complex, therefore we chose to adapt their dataset to utilize TONAS [2] in our training and validation stage as it is annotated. In addition, the demucs tool is also very complex and our team chose to focus on the underlying principles of demucs – Deep U-Net; we therefore utilized a variation of the Deep U-Net CNN model [1] in our final solution for polyphonic input inference.

Matthias Mauch and Simon Dixon [5] proposed the probabilistic YIN (pYIN) algorithm, which augments the YIN method by extracting multiple pitch candidates with associated probabilities for each frame. The pitch candidates are then passed to a hidden Markov model (HMM) which is decoded using an efficient Viterbi algorithm. While the results showed promising pitch detection over the YIN algorithm, the result is mostly for synthesized data. In addition, the paper concludes that the pYIN algorithm manages to reduce octave errors in the pitch track. However, we believe that this solution is not sufficient for our problem statement since real human singing carries different timbres and intonations, which results in an accurate, yet unintended, fundamental pitch notation even when the singing is slightly off pitch. We explore this more in our results section.

3 SYSTEM OVERVIEW

Figure 1 shows a high-level overview for the approach taken in this paper to solve the problem of Automatic Lyrics Transcription and Automatic Melody Transcription. Voice recordings are inputs for the training of the models and the web application. The system input of voice recording is taken as input for the purpose of training. Each model uses the input based on the expected format, with some of the models doing the required pre-processing. The web application takes the system input and passes it to the trained models for the purpose of inference and then displays the output on the web application.

The sections that follow, explain the individual parts of the system in more detail.

4 LYRIC TRANSCRIPTION

In this section, we cover our approach for Automatic Lyrics Transcription. The task of transcribing lyrics is similar to the task of automatic speech recognition with some noticeable and important differences. When a person sings a song, they tend to prolong the pronunciation of the vowels in the words. This makes the identification of words in songs difficult as compared to detecting words in speech. For building an application which can successfully output the lyrics in an a song audio note, three methods were mainly experimented with. Section 4.2 introduces the three methods and Section 4.3 provides implementation details for the three.

4.1 Data

For the purpose of the automatic lyrics transcription task, we used the multimodal N20EM dataset which has been generated at National University of Singapore. Along with a .wav file, the facial movement of a singer is recorded by a video camera and a single eSense earbud captures the IMU signal to head movement. All the songs in this dataset have been recorded on a professional condensor microphone. This dataset comprises of monophonic songs. The dataset provides the text transcription, the split of the data (whether it belongs to test, validation or train set) along with the path of the audio sample.

In this project, we choose to focus on using the audio sensory data for training and testing our models.

4.2 Method

To create a model which can successfully transcribe text from a given audio sample, there are three components which are essential. A language model, a tokenizer and an Automatic Speech Recognizer. As part of our experiment, for training models on singing data, we used a tokenizer and ASR.

We explored three different ways for generating lyrics from an input audio song sample.

4.2.1 Google Speech-to-Text API. This approach uses the Cloud Speech to Text API[2] provided by Google. Google's most advanced deep-learning neural network algorithms are applied for Automatic Speech Recognition. Easy to use APIs are provided by Google which enable a user to configure a model of choice for their own ASR task.

[2]https://cloud.google.com/speech-to-text

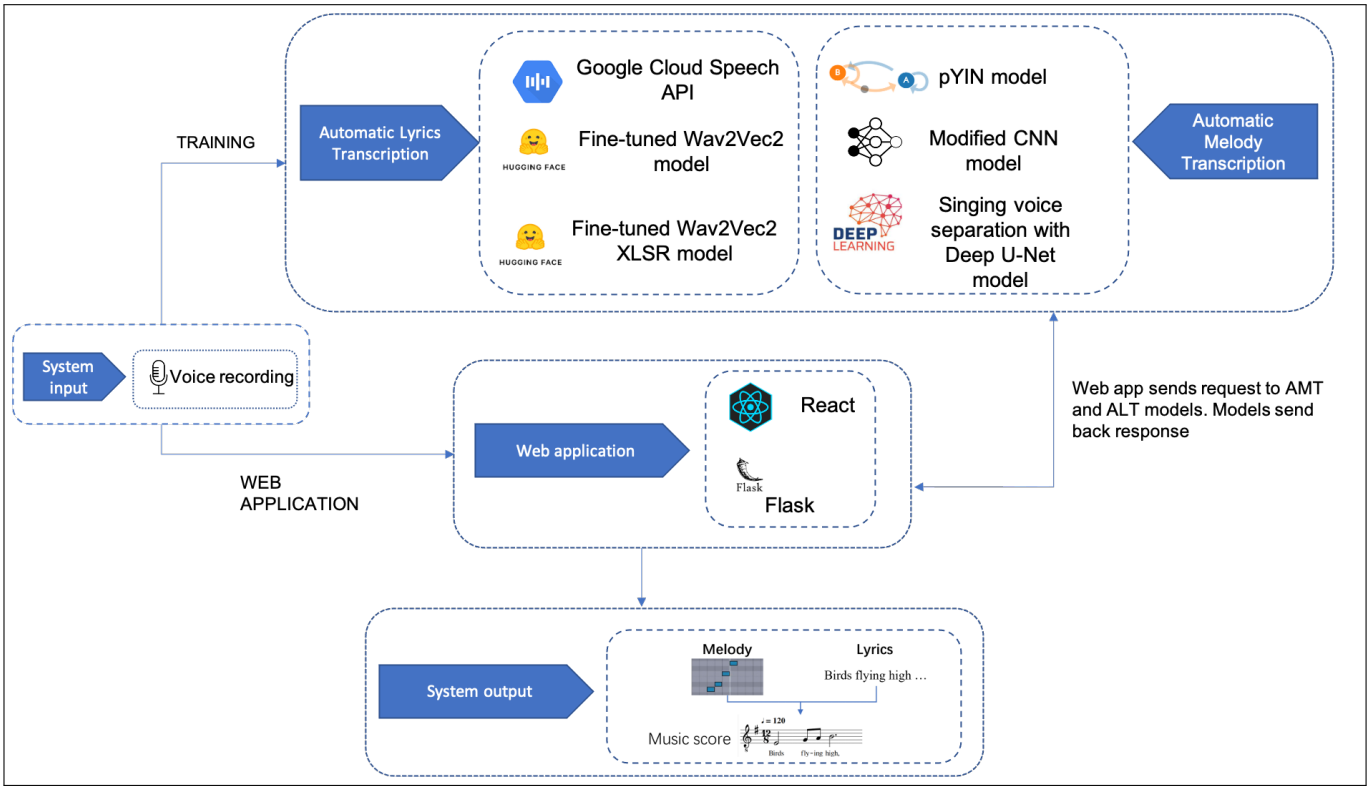

Figure 1: Process Flow.

4.2.2 Fine-Tuning pre-trained Wav2Vec2 Base model. Wav2Vec2 model [10] was trained by Facebook in 2020 and it's the first model which showed that learning powerful representations from speech audio alone, followed by fine-tuning on transcribed data can outperform the best semi-supervised methods. It is also simpler in concept as compared to the semi-supervised methods. Wav2Vec2 achieved impressive results for ASR.

4.2.3 Fine-tuning pre-trained Wav2Vec2 Large XLSR. The XLSR model [11] is a advanced version of the Wav2Vec2 model. This also has been trained by Facebook. This model has been trained on cross-lingual datasets as compared to the mono-lingual dataset on which Wav2Vec2 was trained.

4.3 Experiments

4.3.1 Google Speech-to-Text API. The first approach aimed to explore the performance of the current state-of-the-art ASR technologies on song data. For the same, Google's Speech-to-Text API was leveraged. Google's Speech-to-Text API has various different models trained for the purpose of a Phone Call, Meeting, Video etc. For our experiment, we used the default model available.

The sample audio which had to be transcribed was converted into a mono channel audio using ffmpeg library. The converted audio was passed to Google's API. The sample frequency rate was set to 44.1 KHz and the target language in which the songs would be encoded was set to English. For long audio samples, the result of the transcription is split across different chunks. To output the complete transcript from the input sample, the transcribed results are appended together.

4.3.2 Wav2Vec2 Base Model. Post testing the open-source SoTA models available, we focused on building a model which is trained on singing data. While models which are trained on singing data are not easily available, there is a plethora of models which have been trained on thousands of hours of speech data. With this motivation, we fine-tune the Wav2Vec2 model on the N20EM dataset. This model is fine-tuned without a language model and fine-tuned using Connectionist Temporal Classification (CTC). CTC is an algorithm that is used to train neural networks for sequence to sequence tasks. The original Wav2Vec2 model is trained on 960 hours of LibriSpeech dataset and it can be fine-tuned on any English ASR dataset.

Prior to training our own model, the original ASR model is passed a sample audio song to evaluate it's performance. A sample having the following annotation is given as input to the original Wav2Vec2 model: *Let's start at the very beginning*. The audio sample chosen has non-prolonged words in the beginning of the audio sample, but the words *very beginning* have vowels which have been sung for longer duration as opposed to the duration they would have in normal speech. Wav2Vec2 model generates the following transcription for this audio input: *Let's start at the feiry beaining*. As it can be seen, it fails to identify the prolonged vowel pronunciation in the words.

To fine-tune the pre-trained model, a feature-extractor and a tokenizer is needed. The feature extractor is needed to process the audio sample to the model's input format and the tokenizer

is needed to process the model's output to text format. The pre-trained model maps the speech signal to a sequence of context representations. The fine-tuned model needs to map these context representations to the transcription. For the same purpose, a linear layer is added on top of the pre-trained model.The size of the linear layer is equivalent to the number of tokens available in the dataset on which fine-tuning is going to happen. The count of the tokens in the dataset is calculated using the complete dataset (train and test). To segregate the different words from each other, the space token (" ") is retained in the tokens vocabulary. As a final step in constructing the tokenizer, we added a padding token that corresponds to CTC's *"blank token"*.The complete vocabulary has a size of 29 and hence the linear output layer added on top of the current model had an output dimension of 29.

The input to the Wav2Vec2 model consists of the annotation and a one dimensional vector which represents the audio signal and we use this for all the samples in the training set and train the Wav2Vec2 model for the N20EM dataset. Wav2Vec2 expects all the audio samples to be of 16 KHz frequency and hence before passing the data for training, we resample all the input data to 16 KHz frequency. The pre-trained checkpoint of Wav2Vec2 is loaded and it is trained for 30 epochs. CTC loss reduction is set to mean. Below is a list of the hyperparameters used for training purpose:

```
model = Wav2Vec2ForCTC.from_pretrained(
    "facebook/wav2vec2-base"
    if not last_checkpoint else last_checkpoint,
    attention_dropout=0.1,
    hidden_dropout=0.1,
    feat_proj_dropout=0.1,
    mask_time_prob=0.05,
    layerdrop=0.1,
    gradient_checkpointing=True,
    ctc_loss_reduction="mean",
    pad_token_id=processor.tokenizer.pad_token_id,
    vocab_size=processor.tokenizer.vocab_size)
```

The trained model and processor is saved in a repository on HuggingFace[3]. This repository is a private repository and an access token needs to be provided to load the model. For transcribing the lyrics for an unseen audio sample, the saved model and processor can be loaded from HuggingFace (after providing an access token). HuggingFace also provided hosted infernece APIs, which allows for easy integration of the saved models with our own application.

Once the results are obtained from the model, we use an autocorrect[4] on the result to further improve the results. We ackowledge the lack of a language model might result in errors in the transcribed results. We propose autocorrect as a solution to compensate for the lack of a language model.

4.3.3 Wav2Vec2-Large-XLSR Pre-trained model. In this approach also we use the pre-trained Wav2Vec2-Large-XLSR model which has been trained by Facebook. XLSR learns cross-lingual speech representations by pre-training a single model from the raw waveform of speech in multiple languages. It has been built on top of the base Wav2Vec2 model. The motivation behind choosing this model to fine-tune was that it had been proven that cross-lingual training outperforms mono-lingual training. A tokenizer is constructed using the complete N20EM dataset and the resultant vocabulary has a size of 29. The linear layer added on top of the existing XLSR model has an output dimension of 29. Similar to the Wav2Vec2 base model, the audio samples are resampled to 16 KHz and converted into a one dimensional array using torchaudio. We trained for 30 epochs.

Below is a list of the hyperparameters used for training purpose:

```
model = Wav2Vec2ForCTC.from_pretrained(
"facebook/wav2vec2-large-xlsr-53"
if not last_checkpoint else last_checkpoint,
attention_dropout=0.1,
hidden_dropout=0.1,
feat_proj_dropout=0.1,
mask_time_prob=0.05,
layerdrop=0.1,
gradient_checkpointing=True,
ctc_loss_reduction="mean",
pad_token_id=processor.tokenizer.pad_token_id,
vocab_size=processor.tokenizer.vocab_size
)
```

The trained model and processor is saved in a repository on HuggingFace[5]. This repository is a private repository and an access token needs to be provided to load the model. For the purpose of inference, the saved model and processor can be loaded from the HuggingFace repository by providing an access token.

To correct some minor errors which might happen because of not having a language model, we used autocorrect on the transcribe result of a test data for this model also.

4.4 Results

To evaluate the performance of the trained models on the song dataset, we calculated the Word Error Rate (WER) on the validation dataset. The validation dataset has been used as the test dataset.

4.4.1 Google Speech-to-Text. On testing on different audio samples, Google Speech-to-Text performs well most of the time. For some samples where the dialect is not clear and the pronunciation has an elongated effect, the accuracy of the words predicted is very poor.

4.4.2 Wav2Vec2 Base. Table 1 shows the validation and training loss along with the word error rate for fine-tuning the Wav2Vec2 base model. On the training set, pre-training the Wav2Vec2 model achieves a WER of 53.81.

On evaluating on the validation dataset, which was not used for training purpose, this model achieved a WER of 45.42.

4.4.3 Wav2Vec2 Large XLSR Model. Table 2 shows the validation and training loss along with the word error rate during training. On the training data, pre-training the Wav2Vec2-Large-XLSR model achieves a WER of 48.85.

On evaluating on the validation dataset, which was not used for the purpose of training, this model achieved a WER of 39.4.

[3]https://huggingface.co/akanksha-b14/songs_transcription_wav2vec_base2
[4]https://pypi.org/project/autocorrect/
[5]https://huggingface.co/akanksha-b14/songs-transcription

Table 1: Training and Validation Loss for base model

Step	Training Loss	Validation Loss	WER
100	13.3721	3.630211	1
500	3.0533	3.033746	1
1000	2.9486	2.95589	1
1500	1.0646	1.061442	0.679716
2000	0.5591	0.963065	0.565106
2500	0.2814	0.955017	0.540993
3000	0.2215	0.984059	0.544681
3500	0.2058	1.010484	0.538156

Table 2: Training and Validation loss for XLSR model

Step	Training Loss	Validation Loss	WER
100	19.1819	17.32152	1.001135
500	2.9995	2.963588	1
1000	1.5807	1.164744	0.89078
1500	0.5846	0.806974	0.558014
2000	0.3494	0.832786	0.520851
2500	0.2535	0.821912	0.489645
3000	0.2358	0.815781	0.492482
3500	0.2062	0.836054	0.488511

Table 3 shows the transcriptions obtained by passing an audio sample to the three proposed solutions. This provides an insight into the comparative performance of the three approaches. The original song has the following lyrics: *I have a dream, a song to sing. To help me cope with anything. If you see the wonder of a fairytale, you can take the future.*

As it can be seen, for this particular sample, the XLSR model gives the best transcription. This result is seen for a particular example, we can not generalize and state that the trained XLSR model always performs better than Google Speech to Text.

5 MELODY TRANSCRIPTION

In a bid to identify the best approach to the automatic melody transcription problem, we experimented and tested with a variety of solutions and modifications. Each solution's performance is then measured against the base CNN model from Assignment 2.

5.1 Data

Our dataset is in the MIR-ST500 where the annotations are in the form (onset, offset, midi_note), along with the .mp3 audio file. This dataset is used to train and verify the CNN model during our experiments. We also attempted to utilize the TONAS [2] dataset since it is annotated with onset, duration and midi_note as well. However, due to time constraints, we did not have enough time to repeat the experiments with the TONAS dataset.

5.2 Method

5.2.1 pYIN [5]. This approach estimates a set of fundamental frequencies, along with their assigned probabilities, for each of the frames in the audio input. These probabilities are then used "as observations in a hidden Markov model (HMM), which is Viterbi-decoded" (dynamic programming algorithm for HMM as shown in CS4347 lecture).

5.2.2 Modified CNN. The base CNN model is varied based on the number of convolution and linear layers, to find improvements on the precision and speed of the model. Other modifications are listed in the experiment section below.

5.2.3 VOCANO [3]. This state-of-the-art framework utilizes semi-supervised learning with PyramidNet with ShakeDrop for note segmentation and PatchCNN for pitch extraction, before applying temporal decoding to determine the onset, offset and pitch for each note.

5.2.4 Singing voice separation with Deep U-NET [1]. The pYIN and modified CNN works better with monophonic inputs. By passing a polyphonic input through the Deep U-NET vocal separator, we can then augment solutions 1 and 2 to work with polyphonic inputs with a better result.

5.3 Experiment

For the pYIN model, we utilized librosa's[6] pYIN library to first determine the likely sequence of fundamental frequencies before being converted into a .mid file. Since this solution requires no training, the results of the verification of pYIN will be described in the section below.

For the modified CNN, we investigated the following variables: The number of songs used during the training of each model, the number of convolution and linear layers in the model, the size of the convolution layer. During the first round of experiment, a preliminary test will be conducted by varying each variable first to obtain a trend for the evaluation score (more details in the section below under results) against each variable. In the second round of experiments, promising variables will be selected and put together in combination to get an optimal model.

Lastly, for the Deep U-Net CNN model for vocal separation and vocano framework, we used a pre-trained model for our project due to computation resource limitations as well as time constraints.

5.4 Result

5.4.1 Evaluation metrics. We utilized the mir_eval library [8] and followed the evaluation metrics and notations. For a melody transcription of the singing voice, the onset and pitch is much more important than the offset in order to get a clear .mid output. Therefore, our results will be focused on the COnP for Precision, Recall, and most importantly, F1-score. For the purposes of our experiment, we set the onset threshold of the mir_eval library to be 0.4 and the onset tolerance to be 0.05.

5.4.2 VOCANO (rejected). Firstly, the only model that we chose not to include is the VOCANO framework. Despite the high precision and accuracy of the model, we found it difficult to train, due to computational resource limitations. Coupled with a highly complex model which is difficult to fully understand, our team decided to drop this framework from our solution as a pre-trained model does

[6] https://librosa.org/

Table 3: Example of Transcribed results from an audio sample for all three models

Model	Transcription
Wav2Vec2-base	i have a dream a song to using to help mecrewith anything you see the wonder ove a far are you can take the future
Wav2Vec2-large-xlsr-53	i have a dream a song to sing to help me copewith anything if you see the wonder of a fairytale you can take the future
Google Speech to Text	I Have a Dream song to say, To help me call. With anything. if you see the Wonder, OVA Fairy Tail, you can take the future.

not warrant much merit in comparison to our own modified CNN model.

5.4.3 pYIN (partially rejected). After performing the preliminary check for pYIN against the monophonic input, we determined that while the fundamental frequency determination was accurate, this solution was not suitable for melody transcription of the singing voice. This is due to vocal contours and grace notes during singing which results is many unintended notes to be transcribed. This is supported by our test when we input audio samples with no contours.

5.4.4 Modified CNN (accepted). After performing the first round of experiments against each variable, we found that changing the number of linear layers, and number of songs used for training either results in a drop or negligible change in the COnP score for Precision, Recall and F1-scores. However, increasing the size of the convolution layers (Table 4) showed a general improvement in Precision; as for the Recall and F1-score, both showed increases, before dropping slightly when the convolution layer size reaches (512, 512).

Table 4: Increasing size of convolution layer

Conv Layer Size	Precision	Recall	F1-Score
(32, 32)	0.579395	0.654123	0.612902
(64, 64)	0.603982	0.630198	0.616174
(128, 128)	0.591794	0.671782	0.628184
(256, 256)	0.603963	0.737339	0.634734
(512, 512)	0.623733	0.694237	0.627100

In addition, increasing the number of convolution layers (Table 5) showed improvements in the COnP scores for Recall and F1-score. Therefore, the second round of experiments involved a combined increase of size and number of convolution layers (Table 6) to attain our optimal model.

Table 5: Increasing number of convolution layer

Number of Conv Layer	Precision	Recall	F1-Score
3	0.579395	0.654123	0.612902
5	0.569031	0.670003	0.614723
6	0.632374	0.681283	0.655286

Our final result (Table 6, Iteration 3) comprised of 6 convolution layers with approximately half having size (256, 256) and the other half having (128, 128). Overall, the modified model showed an improvement in F1-score of 0.690575 compared to the Base model's F1-score of 0.612902.

Table 6: Combined modification of convolution layer size and number

Iterations	Precision	Recall	F1-Score
1	0.579395	0.654123	0.612902
2	0.632374	0.681283	0.655286
3	0.674835	0.709568	0.690575

5.4.5 Deep U-NET (accepted). We extracted multiple songs from the MIR-ST500 dataset and analyzed the output manually by listening to the separated track. This solution proved to be effective in solving the polyphonic audio problem. The specific results of the Deep U-Net solution can be found in Jansson et al.'s paper [1].

6 WEB APPLICATION

In this section, the details of the Transonify web application are elaborated, and design considerations for both the frontend and the backend systems are discussed. The web application was built using React for the frontend and Flask for the backend. Figure 2 shows a screenshot of the final application.

6.1 Frontend

The React frontend was bootstrapped using create-react-app. The whole homepage is contained inside the src/Home.jsx file as a functional component. The application allows users to upload a .wav file of their choice from their file system. The application also allows the selection of the models for both lyrics transcription and melody transcription. There is also a switch for whether the user wants to do polyphonic separation, which would be useful for transcribing polyphonic audio more accurately.

States are used to keep track of the current selected model and whether polyphonic separation is to be performed. When the user clicks on the *Transonify* button, the audio file gets sent to the backend for transcription. During this process, the web application displays a loading animation. After all the backend processes are done, the loading animation is replaced with the transcription results from the backend.

The results are divided into several collapsible containers. The lyrics transcription results are returned in a single "Lyrics" container, while the melody transcription results are returned as two representations in the "Piano Roll" and "Score" containers. There is also an audio player that can control the playback of the melody transcription results. The containers can be collapsed so that users can choose to only see the results they are interested in.

To visualize the melody transcription results a web component called "html-midi-player" [9] is used. The web component consists of a midi player and multiple midi visualizers. For Transonify, the "piano roll" and "staff" visualizations are used for the piano roll and

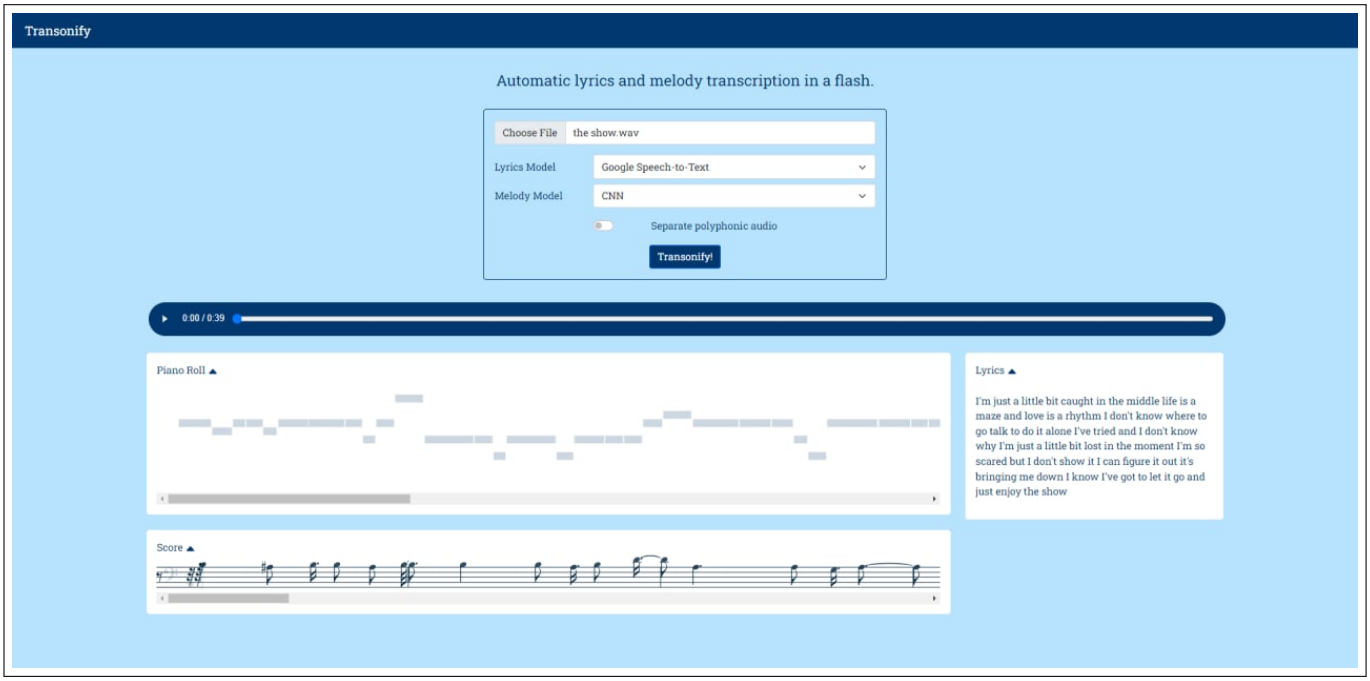

Figure 2: Web Application

score representations respectively. This is done so users without a music background can still understand the results using the piano roll while allowing those with a music background to make use of the provided music score. Some of the styles of the web component can be customized using additional CSS, which is how the animations for the active notes in the piano roll representation is done.

6.2 Backend

The Flask backend consists of several endpoints. These endpoints are listed in the server/server.py file, which is also the entry point for the Flask application. The "/" endpoint serves the homepage of the web application. The "/save" endpoint saves the audio sent from the frontend. The "/separate" endpoint separates the audio into vocal and instrumental tracks, and the vocal track is taken for further processing. This endpoint is called if the user switches on polyphonic separation.

There are three models that can be chosen for the lyrics transcription: Google Speech-to-Text, XLSR-Wav2Vec2, and Wav2Vec2 Base. The endpoints for these models are "/lyrics", "/lyrics_xlsr", and "/lyrics_base" respectively. There are two models that can be chosen for the melody transcription: CNN and pYIN. The endpoints are "/melody" and "/melody_v2m" respectively. If called, the endpoints for the models take the saved audio file and perform either lyrics or melody transcription. For lyrics transcription, the predicted lyrics string is returned, whereas for melody transcription, a midi file is returned.

6.3 Deployment

Deployment of the application using the free platform Heroku was considered. Due to the number and size of the Python dependencies, the slug size of the application was too big to be pushed directly by Heroku. An alternative to this was to use Docker, as Heroku supports pre-built Docker images. Docker was used to containerize the application, then the container image was pushed to Heroku. The application is hosted at http://transonify.herokuapp.com/. This approach is also not ideal because Heroku has a request time-out after 30 seconds, which is not enough to perform the melody transcription for most songs. Thus, only very short songs (such as 'do_re_mi.wav' in /samples) are supported. Other than that, the deployed Heroku application could not support the polyphonic separation feature, as the process requires a lot of memory but the free dynos (containers) available in Heroku only have 512MB of RAM, which is not enough for the process. Thus, it is better to pull the Docker image for the application from justintzuriel/transonify in Docker Hub to run the application without worrying about dependency issues and facing the limitations of the Heroku-hosted application.

7 CONCLUSION

As part of this project, we were able to build a robust Automatic Lyrics Transcription system which performs well on singing data. The ALT system has comparative performance with the open-source SoTA models available for ASR and sometimes outperforms the open-source SoTA models. The best performing ALT model has a WER of 39.4 on the test dataset and 48.85 on the training dataset. This metric is computed prior to passing the transcribed

text to autocorrect. Use of autocorrect has proven to be an effective method for improving the transcription results.

The AMT system supports both monophonic and polyphonic inputs for both the pYIN and modified CNN options. The modified CNN model offered accurate melody transcription (especially for mandopop since the model was trained with MIR-ST500) with an improved mir_eval COnP F1-score of 0.69. For users with synthetic vocal inputs, the pYIN solution can potentially offer a more accurate melody transcription due to the lack of vocal contours and varying vocal timbre. The Deep U-Net model for vocal separation of polyphonic inputs also proves to be an effective solution.

We have also developed and deployed a web application where users can make use of these models by uploading a song of their choice, and having the lyrics and melody transcribed. An interactive visualization of the melody transcription results is available with two different representations to appeal to all users regardless of their musical background.

7.1 Limitations

Currently, the ALT system has been trained without a language model, the lack of which results in erroneous predictions some times. Inclusion of a language model in the end-to-end ALT system can provide better results. The ALT system has been trained on only monophonic data, hence when the background score and vocal score are separated and the vocal score is passed for transcription, the performance deteriorates. We hypothesized the presence of reverberation in the separated vocal score can be a possible reason for the poor performance of the ALT system.

As for the AMT, polyphonic inputs have to be pre-processed by passing the audio through the Deep U-Net model to extract a vocals-only track. This implies that polyphonic inputs will require more time to process as compared to monophonic inputs.

Other than that, the web application only supports .wav files with a sampling rate of 44100 Hz and a duration of at most 1 minute. Another limitation is that the lyrics transcription models "XLSR-Wav2Vec2" and "Wav2Vec2 Base" are hosted on Hugging Face and need to be instantiated the first time they are called in a while. Thus, the application will place an error message instead of the lyrics transcription result in the "Lyrics" container if these models have not been instantiated yet. When this occurs, users will have to click the "Transonify" button again after a few seconds for the application to reattempt the transcription.

7.2 Future Work

In future, we plan to train the current ALT models on vocal scores of polyphonic data. We also plan to provide our users with an option to record a song on the fly and see the results on our web application.

8 TEAM CONTRIBUTIONS

This section aims to clearly state the individual contributions from each team member in the following subsections. The team contributed equally to the writing of this report.

8.1 Akanksha

Akanksha was responsible for the Automatic Lyrics Transcription part in this project.

- Literature review for Automatic Lyrics Transcription
- Backend API for Google Speech to Text.
- Trained Wav2Vec2_base model on N20EM dataset.
- Trained Wav2Vec2_large_xlsr on N20EM dataset.
- Wrote backend APIs to use trained models for the purpose of inference.

8.2 Justin

Justin was responsible for developing the Transonify web application and integrating the models into the web application in this project.

- Developed the full-stack application using React and Flask.
- Integrate backend APIs into web application.
- Set up and augment Base CNN model from Assignment 2.
- Prepared prototype demonstration videos.
- Containerized application with Docker and deployed to Heroku.

8.3 Max

Max was responsible for the Automatic Melody Transcription (AMT) in this project.

- Literature review for melody transcription.
- Backend API for pYIN model.
- Improve modified CNN model through experiment and training on MIR-ST500 dataset.
- Vocal separation for polyphonic music input using Deep U-Net model.
- Experiment with VOCANO for potential final solution.

9 ACKNOWLEDGMENTS

We would like to thank Professor Wang Ye and Liu Hongfu for their guidance and support in carrying out this research study. We would also like to acknowledge the use of the Base CNN model from Assignment 2 as a reference when coding our own CNN model.

REFERENCES

[1] Jansson et al., "Singing Voice Separation with Deep U-Net Convolutional Networks"
[2] TONAS - Mora, J., Gomez, F., Gomez, E., Escobar-Borrego, F.J., Diaz-Banez, J.M. (2010). Melodic Characterization and Similarity in A Cappella Flamenco Cantes. 11th International Society for Music Information Retrieval Conference (ISMIR 2010)
[3] Jui-Yang Hsu, & Li Su. (2021). VOCANO: A note transcription framework for singing voice in polyphonic music
[4] Demirel, E., Ahlbäck, S., & Dixon, S. (2021). MSTRE-Net: Multistreaming Acoustic Modeling for Automatic Lyrics Transcription.
[5] M. Mauch and S. Dixon, "PYIN: A fundamental frequency estimator using probabilistic threshold distributions," 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2014, pp. 659-663, doi: 10.1109/ICASSP.2014.6853678.
[6] Molina, E., Barbancho, A.M., Tardón, L.J., & Barbancho, I. (2014). Evaluation Framework for Automatic Singing Transcription. ISMIR.
[7] C. Gupta, E. Yılmaz and H. Li, "Automatic Lyrics Alignment and Transcription in Polyphonic Music: Does Background Music Help?," ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 2020, pp. 496-500, doi: 10.1109/ICASSP40776.2020.9054567.
[8] Raffel, Colin et al. "MIR_EVAL: A Transparent Implementation of Common MIR Metrics." ISMIR (2014).
[9] cifkao. html-midi-player. https://github.com/cifkao/html-midi-player.
[10] Baevski, Alexei & Zhou, Henry & Mohamed, Abdelrahman & Auli, Michael. (2020). wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations.

[11] Conneau, Alexis & Baevski, Alexei & Collobert, Ronan & Mohamed, Abdelrahman & Auli, Michael. (2020). Unsupervised Cross-lingual Representation Learning for Speech Recognition.