Singing Vocal Source Separation with Deep Neutral Networks

Kevin George kgeorge38@gatech.edu
Kelian Li kli421@gatech.edu
Guangyu Min gmin8@gatech.edu
Jieun Seong jseong8@gatech.edu

Abstract

Music source separation is the task of decomposing music signal into separated stems (e.g., vocals, bass, and drums). In this paper, we explore using deep neural networks for separation vocals from mixed multi-track recordings in a supervised setting. We proposed new models by changing to time domain, replacing the the BiLSTM with the transformer, replacing the MSE Loss with the L1 Loss and the Dual Loss, and compared their Source-to-Distortion Ratios (SDRs).

1. Introduction

The cocktail party effect [3] discovered in 1950's describes how the humans can focus and separate a conversation out of a room full of people chatting and other environmental noises. Music recordings are mixtures of individual instruments and music elements called stems which are mixed and mastered into the final songs through signal processing controlled by audio engineers. The goal of music source separation is then to recover the individual components from the mixed audio signal. Because the publicly available datasets are limited, in most published studies the individual elements are grouped into 4 categories: vocals, bass, drums, and other. And due to computational resources and the scope of this paper, we only focus on singing vocal separation. The model can be re-trained for other source separation tasks with no modification on the model but only different training data.

In the 50-year history of music source separation, traditional signal processing with handcrafted algorithms were the only method in the field until the recent development of deep learning and the freely available datasets released in the past decade. The traditional methods include kernel additive models, nonnegative matrix factorization, and sinusoidal modeling [2]. These methods are developed based on domain knowledge with pre-assumptions on the characteristics of the music signals.

Source separation suffers from the lack of datasets because individual recordings for each instrument in a mixture are rarely available. The data-driven approaches are only possible after some public datasets released in the past decade. Recent success based on deep learning train source separation models in a supervised manner using data sets where both the professional mix and the clean isolated tracks are needed. these approaches train the neural network parameters to minimize the reconstruction error of the model's output (spectrogram, filter mask, or time signal) based on the isolated audio inputs.

The earliest and the most common deep learning models take a given frame of the spectrogram as the input and output the corresponding frame for each of the targeted music sources. Over the years, the architecture developed from fully-connected neural networks to recurrent neural networks (RNN) and convolutional neural networks (CNN) [2]. The fully-connected neural networks suffer from very short time span of each spectrogram size due to the large number of parameters needed. RNN and CNN can both reduce the size of the model, and RNN especially performs well as it suits better for time series signals. Some models take an end-to-end approach using time signal as input and outputs waveform directly [10]. The state-of-the-art model Demucs [4] incorporates both frequency domain and time domain together.

Vocal source separation has wide applications in both downstream academia research and real-world use case. source separation can improve the performance of automatic music transcription and other computational musicology tasks. Many data-driven music Information retrieval studies like genre classification [12], lyrics transcription [7], pitch detection [8] can be benefited from separated tracks instead of mixed recordings as the input data. The ability of isolate the vocals from any commercial recordings has great potential in music education for singers, producers, and sound engineers to learn the newest trend in pop music or some old tricks in historical recordings. Source separation by itself has already been part of some commercial products in Audionamix XTRAX STEMS, IZOTOPE RX 9, and AudioSourceRE. Sound engineers utilize these tools to isolate vocals for remixes or to keep the accompaniment tracks for karaoke [1].

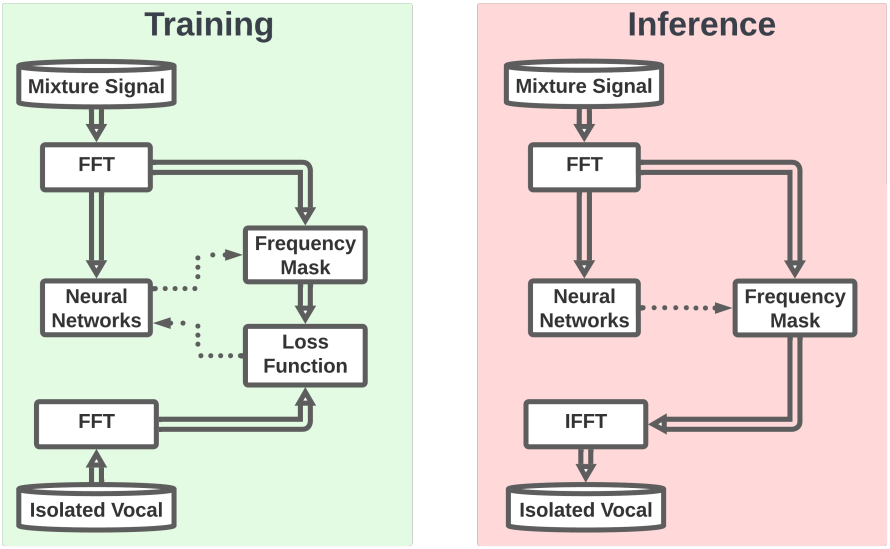

Figure 1. Workflow of spectrogram music source separation in training and inference

2. Approach

Our research is based on Open-Unmix[11] with PyTorch, an open source reference implementation serves as a benchmark model for many music source separation studies. Figure 1 is a diagram showing the workflow of the training and inference of source separation on frequency domain. After applying the short-time Fourier transform to the windowed input audio segments, the signal is transformed into the complex spectrogram of magnitude and phase. Open-Unmix only take the magnitude spectrogram into accord because phase is often considered less influential to human perception. A deep learning model is trained to predict a spectrogram mask with the ground truth isolated signal and the mixture spectrogram input. The spectrogram mask is able to remove the residual component in the mixture signal and returns the magnitudes of the separated target individual stems. The estimated magnitudes and the original phases of the mixture signal are converted back to the time signal by the inverse short-time Fourier transform.

The mask is applied through element-wise multiplying. So if a spectrogram mask $M \in [0.0, 1.0]^{T,F}$ represents the music source S, and $|Y| \in \mathbb{R}^{T,F}$ is the magnitude spectrogram of the mixture signal,

$$S = M \odot |Y| \tag{1}$$

Instead of a binary mask on the magnitude spectrogram, soft masks are allowed to take any value in the interval [0.0,1.0]. The benefit is that the energy of one frequency bin can be distributed to different music sources. In reality, music signals are commonly overlapped in frequency bins. Assigning 0.0 or 1.0 in the soft mask is equivalent to a binary mask. Previous experiments show that soft mask results in better audio quality [6].

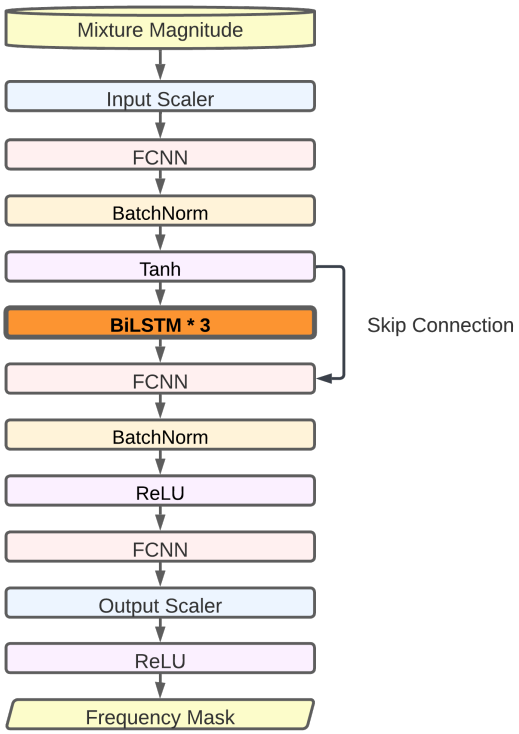

Figure 2. Open-Unmix model architecture

2.1. Open-Unmix

The detailed model architecture of Open-Unmix is displayed in Figure 2. The core of the Open-Unmix source model is based on a three-layer bidirectional long short-term memory (BiLSTM) [5] block firstly introduced to source separation in 2017 by Uhlich et al.[13]. LSTM is a neural network structure that is suited for time series information while minimzing the gradient vanishing problem in traditional RNNs. Due to its recurrent nature, the model can be trained and evaluated on arbitrary length of audio signals. BiLSTM consists two LSTMs, one takes the input from the regular forward direction and one from a backward direction of the future input. Since the model takes information from past and future simultaneously, the model cannot be used in real-time without latency. The model is optimized in the magnitude domain using mean squared error.

2.2. Proposed Time Domain Model

Models for audio source separation usually operate on the magnitude spectrum, which ignores phase information and makes separation performance dependent on hyper-parameters for the spectral front-end. Therefore, we investigate end-to-end source separation in the time domain, which allows modeling phase information and avoids fixed spectral transformations.

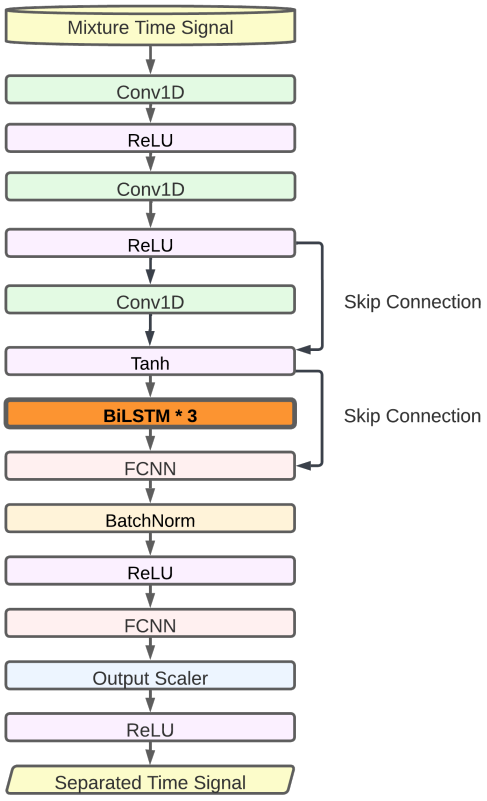

Figure 3. Proposed time domain model architecture

2.3. Proposed Transformer Model

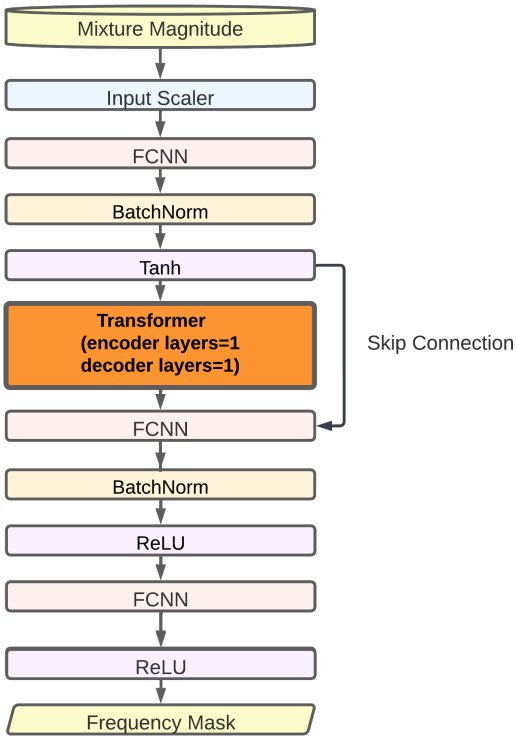

Figure 4. Transformer model architecture

The transformer architecture, unlike LSTM, ingests inputs concurrently instead of sequentially. We replaced LSTMs in the original model with the transformer, expecting the training and inferring times to be shorter and the performance to be higher. However, this proposed model was still taking as much time as the one with LSTMs (completing one epoch for both of them took around 4 hours on Colab GPU). Another advantage of the transformer is that it uses positional embedding which encodes a specific position of a word in a sentence. This prevents the training process from forgetting the past information and also helps learning the relationship between the words and their positions.

2.4. Proposed Model With Different Loss Functions

The original model uses MSE Loss:

$$l(x, y) = \text{mean}\left(\{(x_1 - y_1)^2, \cdots, (x_N - y_N)^2\}^T\right),$$

where N is the batch size.

We wanted to experiment with different loss functions, so we kept the original structure, but replaced MSE Loss with the L1 Loss and the Dual Loss.

2.4.1 Proposed L1 Loss Model

L1 Loss measures the mean absolute error (MAE) between each element in x and y and it is defined by

$$l(x, y) = \text{mean}\left(\{|x_1 - y_1|, \cdots, |x_N - y_N|\}^T\right),$$

where N is the batch size.

The MSE Loss is known to converge faster than the L1 Loss and tends to over-smooth. We wanted to replace the MSE loss with the L1 Loss to see the effect of smoothing on the result.

2.4.2 Proposed Dual Loss Model

Considering that the magnitude spectrogram error does not directly reflect the perceptual difference of two audio signals, the time domain loss is incorporated along with the magnitude loss. Although either the magnitude error nor the time domain error are directly correlated perceived audio quality. The time signal is obtained by the inverse short-time Fourier transform with the estimated magnitude spectrogram and the input mixture phase spectrogram. The mean squared error of magnitude and time are computed independently with the targeted isolated signal, and then added to form the total loss

$$L_{Total} = \alpha L_{mag} + \beta L_{time} \tag{2}$$

where L_{Total} is the total loss, L_{mag} is the magnitude loss, and L_{time} is the time loss. α and β are the weighting coefficients where in our case α is 0.9 and β is 0.1.

3. Experiments and Results

3.1. Dataset

We use the MUSDB dataset for both training and evaluation [9]. The dataset consists of 150 full lengths music tracks and nearly 10-hour duration in total. The music is primarily in genres of Western popular music. The dataset provides isolated clean vocals, drums, bass, and other stems as well as the mixdown, and we only use the vocals and the mixture. The official dataset is split into "train" and "test" at a 2:1 split respectively. The training set contains 100 songs, and the test set contains 50 songs. All signals are stereophonic and encoded at 44.1kHz. All files from the musdb18 dataset are encoded in the Native Instruments stems format (.mp4), composed of 5 stereo streams, each one encoded in AAC at 256kbps. The result of the compression is a bandwidth limited to 16 kHz.

The mixture track composed of the individual sources is loaded using the stempeg library. The audio is loaded as chunks of duration 6s, sampled at a rate 44.1kHz and starting from a random position in the track.

3.2. Metric

Source-to-Distortion Ratio (SDR) is the most common evaluation metric from source separation. The estimated signal output from the model S_{est} is measured as the follows,

$$S_{\text{est}} = S_{\text{target}} + e_{\text{infer}} + e_{\text{noise}} + e_{\text{arti}} \tag{3}$$

where S_{target} is the isolated signal, and e_{infer}, e_{noise}, e_{arti} are the error terms added during model inference. SDR measures the ratio between the target signal and the error terms in dB as

$$\text{SDR} = 10\log_{10}\left(\frac{\|S_{\text{target}}\|^2}{\|e_{\text{infer}} + e_{\text{noise}} + e_{\text{arti}}\|^2}\right) \tag{4}$$

3.3. Training Parameter Settings

Generally, we chose a set of parameters for training: the path to the compressed music dataset folder MUSDB18 ("–root"), a fixed number 42 for the random seed as the need of consistency ("–seed"), the chosen architecture of the model ("–arch", followed by lstm or transformer), the selected loss function ("–loss_func", followed by L1Loss or MSELoss), the folder for the output models ("–output"), the training batch size ("–batch-size"), the chosen number of training epochs ("–epochs") and the number of dataloaders ("–nb-workers").

Initially we considered to utilize the Google Colab Runtime, which is free for basic usage and consists of abundant RAM and VRAMs, to train the modified models. However, the shared free resource seems not to be as powerful as that we need, where it takes more that 3 hours to finish just one training epoch of the original model, then we transferred all the runtime environment, codes and datasets to a local machine, which consists of an AMD Ryzen7 5800H as the CPU and a GeForce RTX 3060 Laptop as the GPU, which shows a more powerful computation performance, while the machine is also short on RAM and VRAM, which are 8GB and 6GB respectively. That makes it difficult to load the deep model with a huge number of parameters. In the beginning, we could not even finish its first training epoch, when the machine threw out an OOM(Out of Memory) error. After many times of experiments, we found that tuning the batch size, the number of dataloaders and the sampling rate could be a way to avoid such conflicts between model and local resources. However, low sampling rate may significantly affect the performance of training process and the final trained models, so we initially decided to reduce the size of batches from 32 to 16, and successfully passed the whole training processes without any OOM error.

As we changing the structure of models, we found that the training time is still a bit long for so many times of training. After several times of attempts, we discovered that the

training script is highly CPU-intensive so that it wastes a bunch of time to read the training data sequentially on CPU and causes low GPU usage in each cycle of training, while the dataloaders can prepare the training data on RAM and VRAM before computing on GPU and make it faster for computing units to find the next job to do, thus we tried to change the batch size and number of dataloaders at the same time to create a balance between the RAM/VRAM usage and the speed of data-loading. Finally, as we decreased the batch size from 16 to 1 and increased the number of dataloaders from 0 to 32, we actually improved the usage of GPU, which increased from lower that 10% to 40% on average, speeded up the training time from 20min/epoch to 13min/epoch and at the same time kept the allocation of memory at a reasonable level.

3.4. Evaluation Results

3.4.1 Time Domain Model

The time domain model performs well in terms of loss metric. However, separating the vocals from a sound track using the model is not productive as it was trained on data at a lower sample rate.

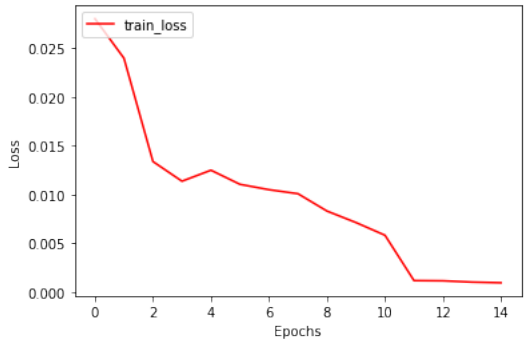

Figure 5. Training loss for the Time Domain Model

3.4.2 Transformer Model

We ran into several problems in training the transformer model. We have been training other models on one of our teammates' GPU, because using the GPU wasn't accelerating the training on Colab, while it was on the personal machine. We suspect it could be because of the environment configuration related to GPU and CUDA. We did not have issues with other models, but for the transformer architecture, we realized it needs much more RAM than other models and the amount of memory needed for it was beyond the capacity of a personal machine. Thus, for this model, we tried to train this on Colab. One epoch was taking more than 6 hours even with the GPU. We tried several times, but every time, the session randomly disconnected, unable to obtain any valid result.

3.4.3 L1 Loss Model

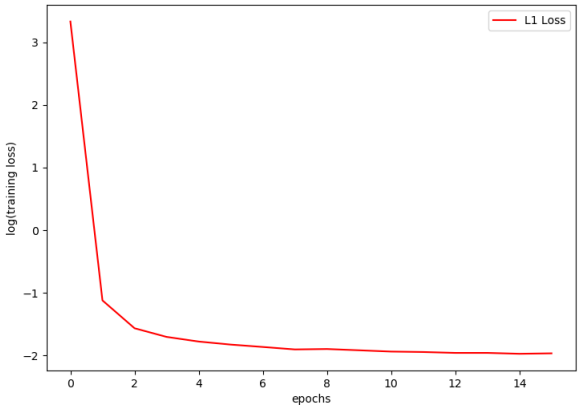

Figure 6. Training loss of the model with the L1 Loss.

One interesting result from this experiment was that the audio quality using MSE Loss was better than L1 Loss, but L1 Loss was better at more exclusively separating the vocals from other instruments than MSE Loss. We think the reason for this is that MSE Loss tends to smooth out more than the L1 Loss, so it blends more the vocals and other instruments than L1 Loss.

3.4.4 Dual Loss Model

The computational cost of dual loss model is about the same as the baseline Open-Unmix model. The final loss of one epoch is 167 for training and 13.1 for validation. The training loss started from 13559.297 in the first batch. Because the loss function is different, it's not comparable with other models.

3.4.5 Comparing the results

Our goal was to compare the SDRs between the original and the proposed models. However, the `museval` library on Open Unmix was not well documented, so we were not able to run it at all.

Method	SDR
Open-Unmix	N/A
Time Domain	N/A
Transformer	N/A
L1 Loss	N/A
Dual Loss	N/A

Table 1. Results

4. Discussion

Although we experimented with different loss functions which may have more correlations with human percep-

tion, a well-rounded loss function is still in need for many audio generation tasks. Some possibilities include Mel-frequency cepstral coefficients (MFCCs), Mel-spectrogram, Bark-spectrogram, and equal-loudness contours.

The dataset consists of almost entirely western popular music, and many music gneres and cultures are ignored. The musical texture of western popular music is melody-dominated homophony (singing melody and instrumental chords), but the music of other genres have different textures as polyphony or hetetrophony. Furthermore, it is uncommon in western popular music for instruments playing the same melodic line and the lead vocal, but this accompaniment style exists in other cultures especially in folk music. All these differences suggest that our vocal separation model may perform poorly on music outside of the dataset.

5. Conclusion

Music source separation is a task that separate individual music elements from the mixture signal. We trained several models to separate singing vocals from the songs, and then tried to evaluate their performances and compare them to the original model, but because of insufficient computational resources and lack of documentation on the evaluation library, we could not train models beyond one epoch and could not compute the SDRs. However, we were still able to separate vocals using the proposed models wi1 epoch of training, as uploaded on the repository.

References

[1] Kevin Brown. *Karaoke Idols: Popular music and the performance of identity*. Intellect Books, 2015. 1

[2] Estefania Cano, Derry FitzGerald, Antoine Liutkus, Mark D Plumbley, and Fabian-Robert Stöter. Musical source separation: An introduction. *IEEE Signal Processing Magazine*, 36(1):31–40, 2018. 1

[3] E Colin Cherry. Some experiments on the recognition of speech, with one and with two ears. *The Journal of the acoustical society of America*, 25(5):975–979, 1953. 1

[4] Alexandre Défossez, Nicolas Usunier, Léon Bottou, and Francis Bach. Demucs: Deep extractor for music sources with extra unlabeled data remixed. *arXiv preprint arXiv:1909.01174*, 2019. 1

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 3

[6] Ethan Manilow, Prem Seetharman, and Justin Salamon. *Open Source Tools & Data for Music Source Separation*. https://source-separation.github.io/tutorial, 2020. 2

[7] Annamaria Mesaros and Tuomas Virtanen. Automatic recognition of lyrics in singing. *EURASIP Journal on Audio, Speech, and Music Processing*, 2010:1–11, 2010. 1

[8] Lawrence Rabiner, Md Cheng, A Rosenberg, and C McGonegal. A comparative performance study of several pitch detection algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 24(5):399–418, 1976. 1

[9] Zafar Rafii, Antoine Liutkus, Fabian-Robert Stöter, Stylianos Ioannis Mimilakis, and Rachel Bittner. The MUSDB18 corpus for music separation, Dec. 2017. 4

[10] Daniel Stoller, Sebastian Ewert, and Simon Dixon. Wave-u-net: A multi-scale neural network for end-to-end audio source separation. *arXiv preprint arXiv:1806.03185*, 2018. 1

[11] Fabian-Robert Stöter, Stefan Uhlich, Antoine Liutkus, and Yuki Mitsufuji. Open-unmix-a reference implementation for music source separation. *Journal of Open Source Software*, 4(41):1667, 2019. 2

[12] George Tzanetakis and Perry Cook. Musical genre classification of audio signals. *IEEE Transactions on speech and audio processing*, 10(5):293–302, 2002. 1

[13] Stefan Uhlich, Marcello Porcu, Franck Giron, Michael Enenkl, Thomas Kemp, Naoya Takahashi, and Yuki Mitsufuji. Improving music source separation based on deep neural networks through data augmentation and network blending. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 261–265. IEEE, 2017. 3

Student Name	Contributed Aspects	Details
Kevin George	Designing, Implementation and Training	Designed the time domain model by combining open-unmix and Wave-U-Net. Trained and tested the model. Edited the musdb package library to accommodate lower sample rates.
Kelian Li	Designing, Literature review, and Implementation	Reviewed the current publishing on music source separation. Provided research directions based on the review. Set up the initial development environment in Google Colab and Github. Implemented the the Dual Loss Model
Guangyu Min	Training, Hyperparameter tuning and Test	Deployed codes on local machine, trained different implemented models with the RTX3060 Laptop, overcame most of the existed conflicts between the codes and resources, increased the efficiency of training, plotted the training loss and obtained the separated parts of a sample music by the trained models.
Jieun Seong	Design, Implementation, and Training	Designed and implemented different loss functions and Transformer instead of BiLSTM in the original model to improve results. Trained the models on Colab.

Table 2. Contributions of team members.