Improving Urban Safety Using Scream Sound Detection Based on Convolutional Recurrent Neural Networks Framework

Project Report

Huhyun Lee
Sltask0222@gmail.com

Abstract

Scream detection can be used as an appropriate monitoring tool for crime prevention in urban areas. In this project, I developed a scream detection model by building a CRNN architecture that combines CNN and LSTM. In addition, a dataset was directly collected to construct the model suitable for detecting scream sounds from the alleys and secluded vacant lots in the city. The constructed model showed suitable performance for screaming sound classification and is sufficient as a baseline for the development of more complex models.

1 Introduction

Screams are one of the main audio signals that represent abnormal situations (Clavel et al., 2005). Scream detection can be used as part of an appropriate city surveillance system to prevent crimes that occur in CCTV blind spots or in secluded spaces.

In this project, I developed a CRNN model for detecting screams in alleys or remote places in the city, and collected datasets for training and evaluation of the model. To solve the data shortage, data augmentation techniques were applied. As a result, a test accuracy of 97.56 and a recall of 1 for scream data were obtained.

2 Background

Sound Event Detection (SED) has a goal of recognizing what happens in an audio signal and when it is happening. In general, the goal is to recognize what temporal instances different sounds exist for a given audio signal (Mesaros et al., 2021). Label containing all temporal information of an event occurring in given signal is called strong label, and that containing only the presence or absence of a class without temporal information is called weak label. In this work, according to the specific purpose of detecting a scream, and assuming that data will be given in a fixed length of 1 second, only binary classification problem with weakly labeled data - whether scream is included or not - will be tackled. This will serve as a basic baseline for the development of future complex works.

Scream Sound Detection in the city has been steadily researched for use in crime prevention and accidents. Jung and Chung (2017) developed a Gaussian Mixture Model (GMM) with improved performance using the UBM (Universal Background Model)-GMM method for scream detection. Saeed et al. (2021) developed a Support Vector Machine model and a Long Short-Term Memory model for scream detection at a fire site and compared their performances. My work differs from these works in that it aims to collect screams that occur in alleys and vacant lots in urban areas at night, and also it leverages Convolutional Recurrent Neural Network Framework for detection.

Convolutional Recurrent Neural Network (CRNN) is a deep neural network framework that extracts features from data by leveraging both Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) architecture. Through CNN Filters, model extracts contextual spatial information of the data, and through RNN, it also gets temporal context information to identify sequential characteristics of data. This type of model architecture combining CNN and RNN has been steadily developed in the field of automatic speech recognition or music classification in recent (çakır et al., 2017). In this work, to detect scream

Label	Audio Type	#Data (Files)	#Augmented Data (Files)	Total
Non-scream	Calm background	70	3,751	4,092
	Car	62		
	Conversation	50		
	Electric kickboard	40		
	Walk	42		
	Wind	72		
	Total	341		
Scream	Scream	50	3,550	3,600

Table 1: Custom dataset summary for training and validation.

Step	Augmentation	# Augmented Data (Files)	
		Non-scream data	Scream data
-	Original data	341	50
1	Pitch change	-	100 (× 3)
2	Volume change	682 (× 3)	300 (× 3)
3	Stretch	-	450 (× 2)
4	Mix with non-scream	1023 (× 2)	2700 (× 4)
5	Add random noise	2046 (× 2)	-
Total		4092	3600

Table 2: Data augmentation summary.

sounds, an RCNN model was built using two CNN sub-models with LSTM layers, that is a type of RNN.

The Long Short-Term Memory (LSTM) is a kind of an RNN architecture, and was designed to solve the problem that RNNs do not properly reflect inputs that are far from their outputs (long-term dependencies). Through the cell state, forget gate, input gate, and output gate, the model determines what to discard and what to keep among past information and updates it in its layers.

3 Background

To build a model suitable for the assumed urban environment such as alleys and vacant lots, scream and non-scream sounds of such place were directly collected. The model was trained and evaluated using these collected data.

3.1 Data Collection

Data was collected at night (from 8 pm to 3 am) from December 17th to 20th, 2022, at the plaza in front of the Pohang University of Science and Technology Institute of Artificial Intelligence. The plaza has an environment similar to our assumed urban areas, and the presence of echoes between building also provides environmental similarity to an alley in the city. The reason for not directly collecting data from the urban area is that this research had to be performed inside Pohang University of Science and Technology. The types of collected audio and the number of data are shown in Table 1. The frequency of all sound data is 44100 Hz and was saved as a wav file. The device used for recording is the K050 microphone model of FIFINE.

Each data has a length of 1 second and is weakly annotated. For the non-scream data, I chose six category of sounds – *calm background, car, conversation, electric kickboard, walk,* and *wind* – that are prevalent in the collected place and considered to be also prevalent in the target urban areas. In the case of *electric kickboard*, 20 data were initially collected, but 20 more were augmented by randomly adding white noise to the original data separately before data augmentation, to match the ratio with other data. In the case of scream sound, it was collected at a distance of about 5m from the microphone, but the volume was reduced by half because there was a large difference in volume from other audio types. A total of 12 speakers (7 men and 5 women) participated in the scream recording, and 6 data were collected form each speaker. Of the total 72 screams, 50 were used for training and validation, and the remaining 22 were used for the test. This ratio of 0.7:0.3 was also applied to non-scream data, and as a result, of the total 483 non-screams, 341 were used for training and validation, and 142 were used for the test.

3.2 Data Augmentation

Since the number of data that can be used for training is not enough, data augmentation was implemented using the python `librosa` library (version 0.10.0). Each step of augmentation and the number of augmented data are shown in Table 2. Both non-scream data and scream data were augmented along the steps.

For non-scream data, each file was imported by `librosa` as an array of numbers and multiplied by a random variable X. X follows normal

Audio Feature	# Features	Input Model
MFCCs	16	CNN1
SC	1	
ZCR	1	CNN2
RMSE	1	
Total	19	

Table 3: Features summary.

distribution that has 1 as the mean and 0.04 as the variance. For each ith data x^i, we generate two values x_1^i and x_2^i from X where x_1^i is smaller than 1 and x_2^i is bigger than 1. By multiplying each of them to x^i, we can generate two more data sets where one sets have a smaller volume and the other sets have a larger volume than the pair in the original sets (Step 2). As a next step, for each file, another non- scream file was randomly selected and the two data were overlayed (Step 4). This step was performed under the assumption that the actual data in urban area may include two or more sounds overlayed together. Finally, additional pair was created by adding the randomly generalized white noise to each data (Step 5).

For scream data, pitch change was firstly adapted. Similar to the step 2 for non-scream data, a dataset with an increased pitch and another dataset with a reduced pitch were created for the scream files. For each data, I extracted two random numbers from the normal distribution $N(0, 4)$ and used them as the number of fractional shift steps (Step 1). The volume change was implemented in exactly the same way as for the non-scream data (Step 2). For the stretch step, a random number was drawn from the distribution of $N(0.8, 0.0025)$ for each file and applied as the stretch rate. Applying a stretch rate less than 1 serves to increase the length and lower the pitch of each scream. Since the scream length of the original data was often short rather than long enough, only length-increased sets were additionally created (Step 3). Step 4 proceeded in the same way as for the non-scream data, but the difference is that three non-scream files were selected for each scream and overlayed to it, unlike before where only one was selected. At this time, half of the non-scream files had already been added to the white noise, so step 5 of additionally applying noise was not implemented.

4 Methodology

4.1 Audio Feature Extraction

CNN1			
Input Size : (1, 16, 44)			
Layer	**Kernel Size**	**Stride**	**Ouput Size**
Conv2d_1	(3, 3)	(1, 1)	(32, 16, 44)
Maxpool2d_1	(2, 1)	(2, 1)	(32, 8, 44)
Conv2d_2	(3, 3)	(1, 1)	(64, 8, 44)
Maxpool2d_2	(2, 1)	(2, 1)	(64, 4, 44)
Conv2d_3	(3, 3)	(1, 1)	(128, 4, 44)
Maxpool2d_3	(2, 1)	(2, 1)	(128, 2, 44)

CNN2			
Input Size : (1, 3, 44)			
Layer	**Kernel Size**	**Stride**	**Ouput Size**
Conv2d_4	(3, 3)	(1, 1)	(32, 3, 44)
Conv2d_2	(3, 3)	(1, 1)	(64, 3, 44)
Conv2d_3	(3, 3)	(1, 1)	(128, 3, 44)

LSTM			
Input Size : (44, 128 × 5)			
Layer	**Hidden Size**	**#Layers**	**Ouput Size**
LSTM	32	**2**	(44, 32)
Flatten	-	1	(44 × 32)
FC_1	-	1	16
FC_2	-	1	1
Sigmoid	-	1	1

Table 4: Model architecture. The overall model consists of three sub-models, two of CNNs and one of a LSTM.

To classify the scream sound, the features shown in Table 3 were extracted from each data file. For the types of audio features, Mel- frequency cepstral coefficients (MFCCs), Spectral centroid (SC), Zero crossing rate (ZCR), and Root-mean-square energy (RMSE), which are considered suitable for scream classification, were selected by referring to saeed et al. (2021). However, Saeed et al. extracted 13 MFCCs and used total 16 audio features as LSTM inputs and it differs from mine where 16 MFCCs and 3 other audio features became input for different CNN models, respectively. The outputs of CNN models were combined and used in LSTM.

4.2 Model Architecture

The summary of model architecture is shown in Table 4. It was motivated by the architecture example of Mesaros et al. (2021), but differs from it in the way of using two different CNN sub-models to get the input features for LSTM.

CNN1 receives MFCCs as input and extracts 128 convolutional features. At this time, since the frequency of the sound data is 44100Hz and the data length is 1 second, each input has sequential

Result Summary	
Training accuracy	98.02%
Validation accuracy	98.18%
Test accuracy	97.56%
Precision for scream	0.85
Recall for scream	1
F1-Score for scream	0.91

Table 5: Performance metrics for the model

information of 44 lengths. And by setting the kernel size and stride of the `maxpooling` layer to (2, 1), the shape of the sequential axis is maintained as 44 until the last output. After the Maxpool2d_3 layer, I used dropout with a rate of 0.2.

CNN2 receives SC, ZCR, and RMSE as input and extracts 128 convolutional features as well. In CNN2, `maxpooling` layer was not used in order to maintain the appropriately enough feature numbers for each data. So as a result, the shape of the audio feature axis and sequential axis were maintained as 3 and 44.

Finally, LSTM receives the output of the two CNNs as input. At this time, the outputs of the two CNNs are concatenated based on the audio features axis so there exist total 5 audio features. Considering the 128 convolutional features extracted from CNN models, each sequence has a total of 128×5 features as an input. The hidden size of LSTM was set to 32 and two layers were used. Also, dropout rate of 0.2 was used. Total number of the parameters are 302,625.

Since the problem was set to binary classification of weak labels of the sound data, a sigmoid function was applied after going through two Fully Connected Layers for the output of LSTM and the Binary Cross Entropy Loss was used as the loss function.

5 Experiment

For the experiment, I used `pytorch` version 2.0.0 with `cuda` version 11.8.

5.1 Hyper-parameter Tuning

I used Adam as an optimizer, and as the final hyper-parameter of the model, the batch size of 512, and the learning rate of 0.001 were applied. Early stopping was applied during the training so the model trained with 160 epochs.

5.2 Results

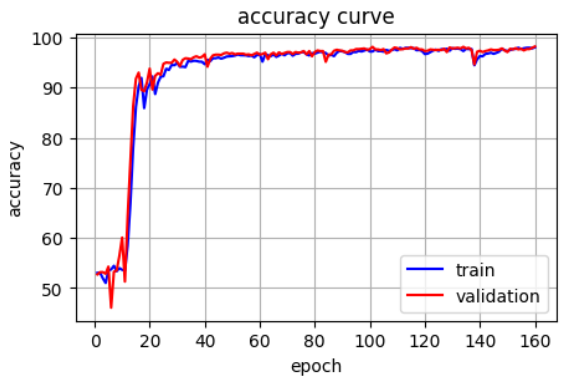

Figure 1: Accuracy curve of the model.

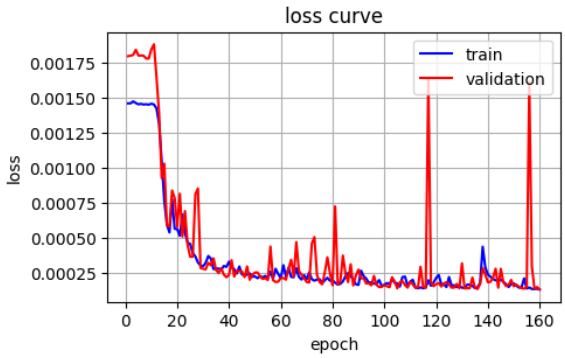

Figure 2: Loss curve of the model.

Table 5 summarizes the results of each evaluation metrics for the final model. The final test accuracy is 97.56%, and the precision and recall for scream data are 0.85 and 1, respectively. It can be expected that the model will perform the role of an appropriate detector when scream occurs, but considering that there are only 22 screams among the test data, further research will be helpful to test the performance of the model using additional datasets in the future.

5.3 Analysis

The accuracy curve is shown in Figure 1, and the loss curve is shown in Figure 2. Overall, the loss does converge but it still shows some sharp peaks in places. Therefore, further research such as building more deep layers or training with additional epochs will be possible in the future work.

Figure 3 is a confusion matrix of the test results of the model. Among 164 test data, 4 non-scream data were misclassified as scream, but the others showed good prediction ability of the model. However, as mentioned earlier, unlike training

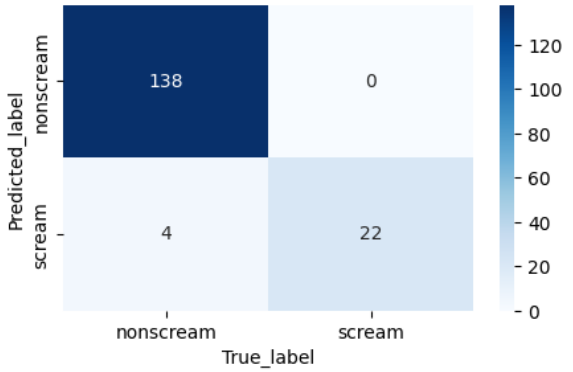

Figure 3: Confusion matrix for the model evaluation with test dataset.

process, the number of non-scream data and scream data is imbalanced during the test, so using additional test data will be helpful in evaluating the additional performance of the model.

6 Conclusion

For screaming sound detection on weakly labeled data, in this study, I developed an CRNN model that performs binary classification of city sounds by combining CNNs and LSTM. In addition, to build a specific model which detects scream sounds in urban areas at night time, screams and other background sounds were directly collected in the analogous environment, and model training was performed with a total of 7,692 data using a data augmentation technique. The final test performance of the model was 97.56%, showing a high recall of 1. It is expected that this study will be able to serve as a basic baseline in the development of a sound event detection model for urban safety.

References

Chloé Clavel, T. Ehrette, and Gaël Richard. 2005. Events Detection for an Audio-Based Surveillance System. In *2005 IEEE International Conference on Multimedia and Expo.*

Annamaria Mesaros, Toni Heittola, Tuomas Virtanen, and Mark D. Plumbley. 2021. Sound Event Detection: A Tutorial. In *IEEE Signal Processing Magazine.*

Sukhwan Jung and Yongjoo Chung. 2017. Screaming Sound Detection based on UBM-GMM. In *International Journal of Grid and Distributed Computing.*

Fairuz Samiha Saeed, Abdullah Al Bashit, Vishu Viswanathan, and Damian Valles. 2021. An Initial Machine Learning-Based Victim's Scream Detection Analysis for Burning Sites. In *MDPI Applied Sciences.*

Emre Çakır, Giambattista Parascandolo, Toni Heittola, Heikki Huttunen, and Tuomas Virtanen. 2017. Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection. In *IEEE Transactions on Audio, Speech and Language Processing..*