Deep Neural Network for 3D Audio

Multichannel Sound Event Detection Using 3D Convolutional Neural Networks for Learning Inter-channel Features

Neural Networks 2018-2019

Andrea Carlesimo 1618599, Simone Faricelli 1647406

May 13, 2019

1 Introduction

The paper assigned for the project proposes a stacked convolutional and recurrent neural network (CRNN) with a 3D convolutional neural network (CNN) in the first layer for the multichannel sound event detection (SED) task. The 3D CNN enables the network to simultaneously learn the inter- and intra-channel features from the input multichannel audio. The proposed method learns to recognize overlapping sound events from multichannel features faster and performs better SED with a fewer number of training epochs. We worked on the real-life recording TUT-SED 2017 development dataset, that consist of manual annotations for sound event of six classes. We implemented two main scripts: one for the feature extraction and one for the train of the model. The programming language used is Python2 due to the presence of *Keras* framework and some helpful libraries to analyze the audio files.

2 Dataset analysis

This dataset was recorded in the street context using a binaural in-ear microphone at 24 bit and 44100 Hz sampling rate. Each of the recordings is of the length 3-5 minutes, amounting to a total length of 70 minutes. This dataset consists of manual annotations for sound event classes such as brakes squeaking, car, children, large vehicle, people speaking, and people walking. The appearances of each class is showed in Tab. 1. It has four-folds of training and testing splits. The dataset has two channels, so we extracted only binaural features such as *Log Mel Band Energy* and *Generalized Cross Correlation with Phase Transform*. Also it would have been possible working on the single channel version taking

Event label	**Dev. set**
brakes squeaking	52
car	304
children	44
large vehicle	61
people speaking	89
people walking	109
total	659

Table 1: Event instances per class.

the mean of the binaural channel excluding the gcc features from the model. In order to assess the performance of SED for more than two channels of audio they proposed a synthetic datasets that it has not been taken into account in this project.

3 Feature extraction

As mentioned before, we extracted two types of features: the log of the mel-band energy (*mbe*) and the generalized cross correlation with phase transform (*gcc*). The extraction of *mbe* was fast and generally simple thanks to the functions in the library *librosa*. Instead there were no libraries or sample code providing the *gcc* generation, so we made it on our own and, in terms of computation it was really heavy.

3.1 Log mel-band energy

We extract mbe for each of two channels in 40 ms windows with 50% overlap, so the hop-lenght parameter is half of nfft. We use 40 mel-bands in the frequency range of 0-22050 Hz. (The above mentioned value was wrong in [1]) For a sequence length of T frames, the mbe feature has a general dimension of T $\times$ 40 $\times$ C, where C is the number of channels, in our case 2. Below the Python function to extract mbe: it takes in input the audio y, the sampling rate $sr = 44100\ Hz$, the FFT windows size $nfft = 2048$ and the number of mel bands $nb_{mel} = 40$. The graphic visualization of a sample of the extracted mbe in Fig. 1.

```
#--------------------
# EXTRACT MEL BAND
#--------------------
#hop_lenght = nfft/2 means overlap 50%
#power = 1 is for energy mel
def extract_mbe(_y, _sr, _nfft, _nb_mel):
    spec, n_fft = librosa.core.spectrum._spectrogram(
        y=_y, n_fft=_nfft, hop_length=_nfft//2, power=1)
    mel_basis = librosa.filters.mel(sr=_sr, n_fft=_nfft,
        n_mels=_nb_mel)
    return np.log(np.dot(mel_basis, spec))
```

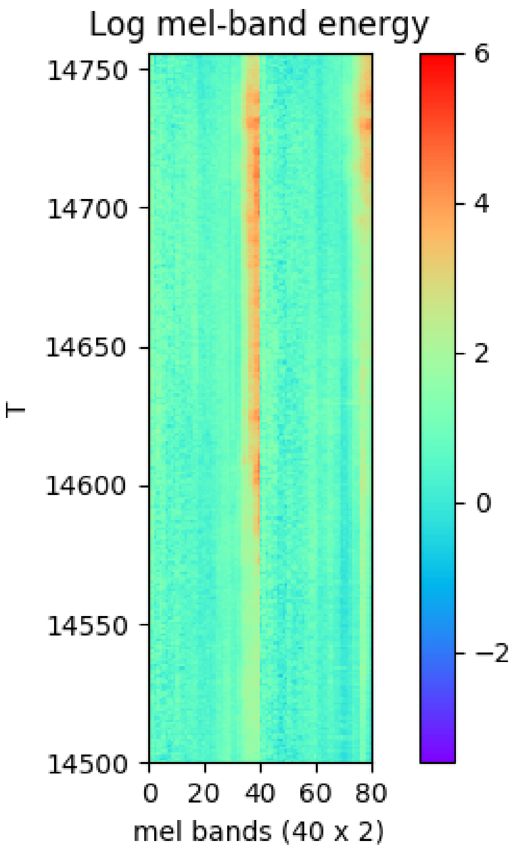

Figure 1: Log mel-band energy (Tx40xC)

3.2 Generalized cross correlation with phase transform

The SED methods can benefit with gcc for overlapping sound events. We extract gcc in three resolutions, 120, 240, and 480 ms as:

$$R(\Delta_{12}, t) = \sum_{k=0}^{K-1} \frac{X_1(k,t) \cdot X_2^*(k,t)}{|X_1(k,t)||X_2(k,t)|} \exp \frac{i2\pi k \Delta_{12}}{K} \tag{1}$$

where, X_1 and X_2 are the FFT coefficients of the two-channels between which the gcc is calculated. $X_1(k,t)$ is the coefficient at time frame t and k-th frequency bin, of the total K bins. gcc per frame given by $R(\Delta_{12}, t)$ is extracted for delays Δ_{12} in the range $[-\tau_{\max}\ , \tau_{\max}]$ where $\tau_{\max}$ is the maximum sample delay for a sound wave to travel between the pair of microphones recording audio. In order to have a factorisable feature length or max pooling in the neural network, 60 gcc values are chosen in the range $\Delta_{12} \in [-29, 30]$ lag for each of the three multi-resolution. For a sequence length of T frames, the gcc feature is dimension T x 60 x 3. The graphic explanation of the gcc, corresponding to the above mentioned sample of mbe, is showed in Fig. 2.

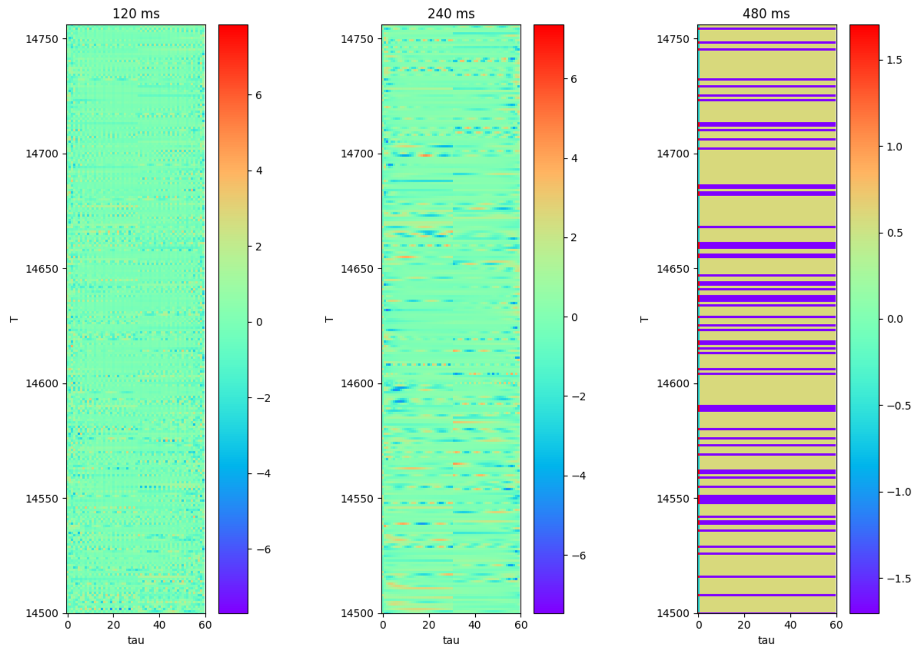

Figure 2: Generalized cross-correlation (Tx60x3)

First of all we compute the FFT coefficients for the two channels in three resolutions (120, 240 and 480 ms) varying the windows length size, which is expressed as 8, 4 and 2 frames. (frames = 1/sec).

```
#--------------------
# Extract FFT for gcc
#--------------------
FFT_120 = librosa.core.stft(
    y=_y, n_fft=_nfft, hop_length=_nfft//2, win_length=8) # 1/0.12
FFT_240 = librosa.core.stft(
    y=_y, n_fft=_nfft, hop_length=_nfft//2, win_length=4) # 1/0.24
FFT_480 = librosa.core.stft(
    y=_y, n_fft=_nfft, hop_length=_nfft//2, win_length=2) # 1/0.48
```

Because eq. 1 is very heavy in terms of computational complexity, it has been needed to optimize the code implementing multiprocessing to exploit the full CPU power, using all the logic cores. For each of the 24 audio file, the axis relative to *time* dimension is divided, for each resolution, in 4 batches, for a total of 12 parallel call to the function *extract_gcc*. The extraction time is then reduced by 12 for the designed launch on a 12 cores CPU. At the end of all the 12 processes the full time dimension is then reconstructed.

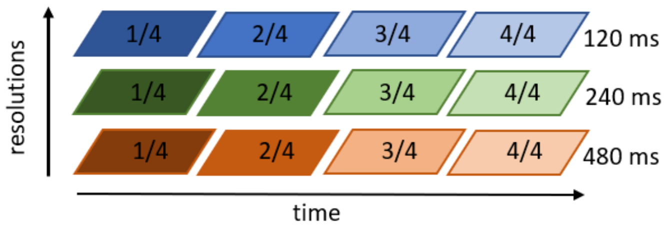

Figure 3: Multiprocessing sample division.

```
#--------------------
# MULTIPROCESSING
#--------------------
multiprocess_diz =
        {1 :FFT_120, 2 :FFT_240, 3 :FFT_480,   #1/4
        4 :FFT_120, 5 :FFT_240, 6 :FFT_480,   #2/4
        7 :FFT_120, 8 :FFT_240, 9 :FFT_480,   #3/4
        10 :FFT_120, 11 :FFT_240, 12 :FFT_480}  #4/4
output = mp.Queue()
processes = [mp.Process(target=extract_gcc, args=(multiprocess_diz[res],res,output,
        tqdm(total=60,position=res)))for res in range(1,13)]
# Run processes
for p in processes:
    p.start()
results =  [output.get() for p in processes]
# Exit the completed processes
for p in processes:
    p.join()
# Get process results from the output queue
results.sort()
results = [r[1] for r in results]
```

Finally all the features extracted from the 24 audio files were divided in 4 folders, as mentioned in [5], and each folder was split in train and test data. The test data is around the 20%. The result of the GCC extraction is composed by a real and an imaginary part, but due to lack of information in [1], we decided to keep only the real part as input for the model. That's also because in presence of complex inputs the model would need also a branch for the imaginary part, eventually merging it with the real branch. But that's not taken into account in [1].

```
#--------------------
# EXTRACT GCC
#--------------------
def extract_gcc(_FFT,_res, _output,_bar):
    time_cut = _res
    #time = np.arange(0,_FFT.shape[0],1)
    partial_index = time.shape[0]//4
    if time_cut <= 3:
       time = time[0:partial_index]
    elif time_cut <= 6:
        time = time[partial_index:partial_index*2]
    elif time_cut <= 9:
        time = time[partial_index*2:partial_index*3]
    else:
        time = time[partial_index*3:]
    TAU = np.arange(-29,31,1)
    gcc = np.zeros((time.shape[0],len(TAU)),dtype=np.complex_)
    N = _FFT.shape[1]//2
    for delta in TAU: #-29, -28, ..., 28, 29, 30
        for i,t in enumerate(time):
            gcc_sum = 0
            for freq in range(N):
                X_1 = _FFT[t][freq]
                X_2 = _FFT[t][freq+N-1]
                fraction_term = (X_1 * np.conjugate(X_2))/(abs(X_1) * abs(X_2))
                exp_term = np.exp((2j*math.pi*freq*delta)/N)
                #exp_term = np.real(np.exp(np.complex((2*math.pi*freq*delta))/40))
                gcc_sum += fraction_term*exp_term
            gcc[i][delta] = gcc_sum
        _bar.update(1)
    _bar.close()
    _output.put((_res,gcc))
```

Input Preprocessing the extracted features:

- Each different type of training data (mbe and gcc in three resolutions, for a total of 4 sets) is normalized, and the corresponding testing data is scaled using its training data weights.

```
scaler = preprocessing.StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

- We sample again the dataset in subsets of 256 frames with a *Reshape*. The final shape is than samples x 256 x 40 x 2 for mbe and samples x 256 x 60 x 3 for gcc.

- One last reshape is done in order to match the 3D convolutional layer input requirement. This particular input of features along channel-time-frequency enables the network to learn both inter- and intra-channel features simultaneously.

Output The output vector Y is filled with a one-Hot encoded vector for each frame. Each vector has length equal to the number sound classes. For each frame we set 1 the element in position correspondent to the value of the sound category. The dimension is n_frames x $m\_SoundClasses$.

$$\begin{aligned} & SoundClasses & \\ Y = [& [0,0,1,0,0,1] & frame\_0 \\ & [0,1,1,0,1,1] & frame\_1 \\ & \ldots & \\ & [0,1,0,0,0,0]] & frame\_n \end{aligned} \tag{2}$$

4 Neural Network

The paper proposed a baseline CRNN and a new approach, they define "C3RNN", in which with the first layer is 3D convolutional. We develop a model of the above mentioned approach, but also a baseline CRNN, tested on the mbe features only. The first part of the network is a sequence of 3 convolutional layers, the first 3D and the others 2D, each one followed by Batch normalization, max-pooling and a Dropout as regularizer. This architecture is replicated in another branch and learned in parallel for the mbe and gcc features, see Fig.4. The CNN activations from the two branches are concatenated along feature axis and are fed to layers of bi-directional gated recurrent units (GRU), to learn long-term temporal activity patterns. This is followed by a layer of time-distributed fully-connected (dense) network. The final prediction layer has six sigmoid units as the number of sound event labels in the dataset.

As mentioned in [1] we setup all the parameters of the network as follow:

- Kernels size: 64
- Filters dimension: 2x3x3, 3x3 and 3x3 mbe
- Filters dimension: 3x3x3, 3x3 and 3x3 gcc
- Max pooling on mel-bands: 5,2,2
- Max pooling on delay: 5,3,2
- GRU units: 64
- Suggested dropout rate: 0.2
- Actually used dropout rate: 0.5

4.1 Training

The training is performed for 1000 epochs using Adam optimizer, learning rate of 0.0001, and binary cross-entropy loss between the reference sound class activities and the predicted ones. Early stopping is used to avoid overfitting the network to training data. A threshold of 0.5 is used to obtain the binary decision from the sigmoid activations in the final prediction layer. Was needed to implement a personalized keras class to compute the required custom metric, frame-wise error rate and F1 score at the end of each epoch, as mentioned in [1], using these results also for early stopping. As first approach the training is stopped if the frame-wise error rate on the validation does not improve for 100 epochs, later on we check the mean error rate over the epochs, to avoid local improvement on the error.

Custom Class Metric The development of a personalized class to use particular metric computation was fundamental. We have used the keras callback *on_train_begin* to initialize some variables and *on_epoch_end* to compute our frame-wise metric and storage all the data for future analysis.

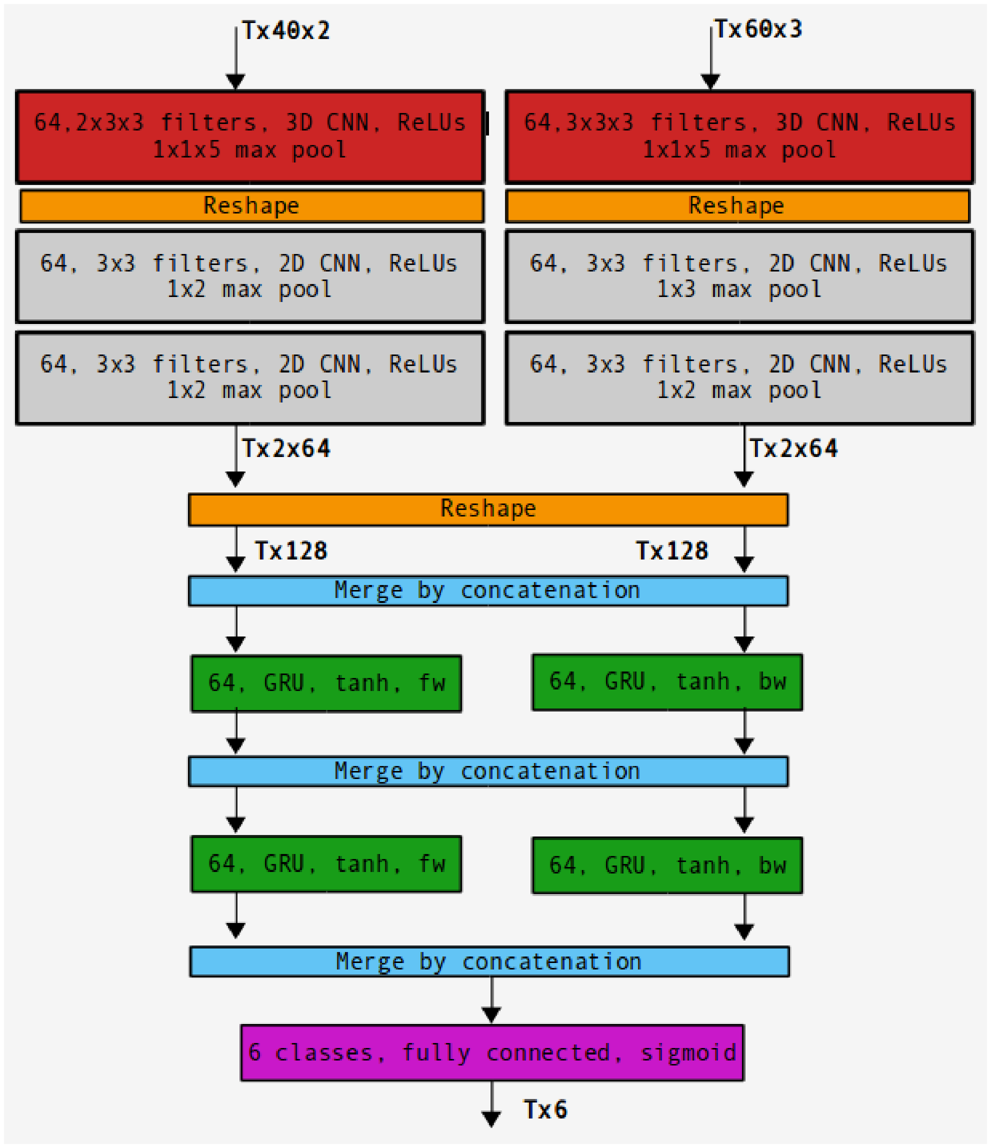
Tx40x2
Tx60x3
64,2x3x3 filters, 3D CNN, ReLUs 1x1x5 max pool
64,3x3x3 filters, 3D CNN, ReLUs 1x1x5 max pool
Reshape
Reshape
64, 3x3 filters, 2D CNN, ReLUs 1x2 max pool
64, 3x3 filters, 2D CNN, ReLUs 1x3 max pool
64, 3x3 filters, 2D CNN, ReLUs 1x2 max pool
64, 3x3 filters, 2D CNN, ReLUs 1x2 max pool
Tx2x64
Tx2x64
Reshape
Tx128
Tx128
Merge by concatenation
64, GRU, tanh, fw
64, GRU, tanh, bw
Merge by concatenation
64, GRU, tanh, fw
64, GRU, tanh, bw
Merge by concatenation
6 classes, fully connected, sigmoid
Tx6

Figure 4: C3RNN Network architecture.

```python
class Metrics(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self._er= 0
        self._f1 = 0
        #   ...
        #other variables
    def on_epoch_end(self,epoch, batch, logs={}):
        X_val,X_val_gcc, y_val = self.validation_data[0],
            self.validation_data[1],self.validation_data[2]
        pred = model.predict([X_val,X_val_gcc])
        pred_thresh = pred > 0.5
        #print pred_thresh
        score_list = metrics.compute_scores(
            pred_thresh, y_val, frames_in_1_sec=frames_1_sec)
        self._f1 = score_list['f1_overall_1sec']
        self._er = score_list['er_overall_1sec']
        #error rate over epoch
        self.er_mean_batch += self._er
        self.er_mean= float(self.er_mean_batch) / (epoch+1)
        if self.er_mean > self.er_mean_prev:
            self._fail_count+=1
            if self._fail_count >= 100:
                self.model.stop_training = True
        else:
            self._fail_count = 0
        self._er_prev = self._er
        self.er_mean_prev = self.er_mean
        self._er_list.append(self._er)
        self._f1_list.append(self._f1)
        self.er_mean_list.append(self.er_mean)
        return
    def get_data(self):
        return DATA
```

Regarding the model with both mbe and gcc branches the weight parameters are around 500 k, instead 290 k for mbe only. So the network is pretty big, we achieved a batch size of 128 (as suggested in [1]) to train only the mbe and 64 for the full features network. The previous batch sizes worked on a NVIDIA Tesla K80 12 GB hosted by Google Colab. On our NVIDIA 940MX 2 GB the maximum batch size supported was 8. A full training is repeated for each of the 4 folders [5].

GPU	Training epoch ETA
NVIDIA 940MX 2 GB	$\sim$ 120 s
NVIDIA RTX 2080 8 GB	$\sim$ 32 s
NVIDIA Tesla K80 12 GB	$\sim$ 29 s

Table 2: Training epoch ETA for the tested hardware with the gcc and mbe branches.

4.2 Metric

The proposed SED method is evaluated using the poly-phonic SED metrics proposed in [4]. Particularly we use segment wise error rate (ER) and F-score calculated in one-second length segments. The F-score is calculated as

$$F = \frac{2 \cdot \sum_{k=1}^{K} TP(k)}{2 \cdot \sum_{k=1}^{K} TP(k) + \sum_{k=1}^{K} FP(k) + \sum_{k=1}^{K} FN(k)} \tag{3}$$

where for each one-second segment k, TP(k) is the number of true positives i.e., the number of sound event labels active in both predictions and ground truth. FP(k) is the number of false positives i.e., the number of sound event labels active in predictions but inactive in ground truth. FN(k) is the number of false negatives i.e., the number of sound event labels active in the ground truth but inactive in the predictions. The error rate is measured as

$$ER = \frac{\sum_{k=1}^{K} S(k) + \sum_{k=1}^{K} D(k) + \sum_{k=1}^{K} I(k)}{\sum_{k=1}^{K} N(k)} \tag{4}$$

where N(k) is the total number of active sound events in the ground truth of segment k. The number of substitutions S(k), deletions D(k) and insertions I(k) is measured using the following equations for each of the K one second segments:

$$\begin{aligned} S(k) &= \min(FN(k), FP(k)) \\ D(k) &= \max(0, FN(k) - FP(k)) \\ I(k) &= \max(0, FP(k) - FN(k)) \end{aligned} \tag{5}$$

5 Results

Several trainings has been performed in the attempt to reach the state-of-the-art values of **F1 Score** and **Error Rate** mentioned in [1]. In table 3 are listed the best values from the training of the four different folders for each one of the different configuration. In Table 3 and 4 are reported the various configurations and their relative F1 score and Error rate, computed with the metric proposed in [4]. Besides, plots of the above-mentioned values over epochs are showed in the figures below. As highlighted from the graph regarding the dropout value = 0.2, the best F1 score is obtained around 100 epochs. Similar results are gathered from the training mentioned in [1].

For a matter of personal satisfaction, a last training has been performed with the following parameters, gathered from [2]:

- Kernel Size: 128
- GRU Units: 32

Scoring the following results:

- F1 Score: 52.3
- Error Rate: 73.1

Which are aligned with the other results but are in certain way optimized with respect to the other because the f1 best score match exactly the f1 score related to the minimum error rate value.

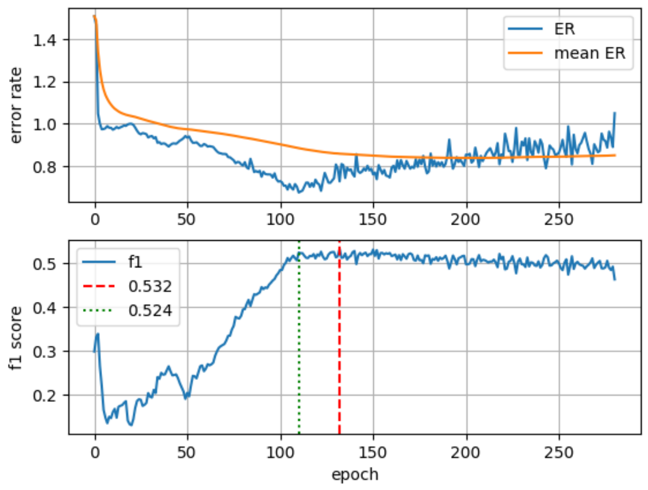

Figure 5: C3RNN mbe features only with dropout = 0.2, best f1 (- - - -), f1 for the best ER (⋯⋯).

Following the guidelines reported in [1], we build the model of a C3RNN training it with both configurations of mbe and mbe+gcc. After observing the model performing overfitting, we decided to test the model on different dropout. (0.35 and 0.5). The highest dropout gave us the best results. We also tested the CRNN version of the model feeding it with the mbe only. In that way we managed to have a comparison of the results with others papers [3], [2]. Scores are listed in Table 4. We checked that, as mentioned in [1], the use of gcc in addition to mbe brings to slightly improved results with respect to mbe only.

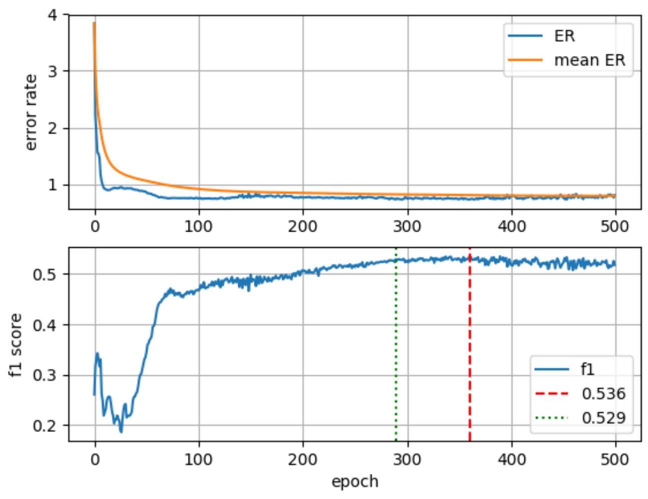

Figure 6: C3RNN mbe features only with dropout = 0.5, best f1 (- - - -), f1 for the best ER (······).

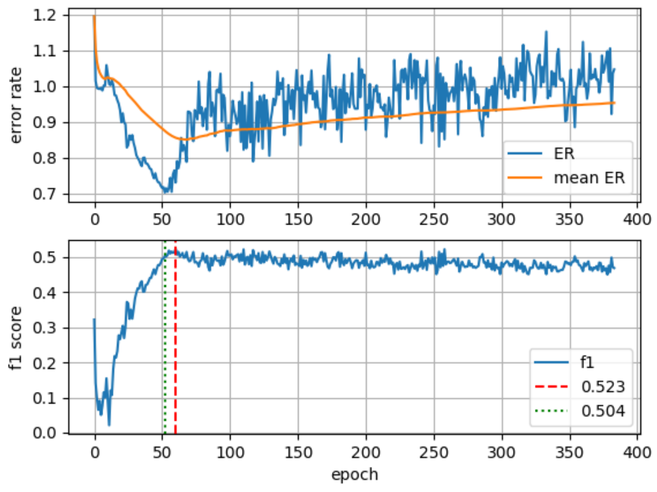

Figure 7: C3RNN gcc and mbe features with dropout = 0.2, best f1 (- - - -), f1 for the best ER (······).

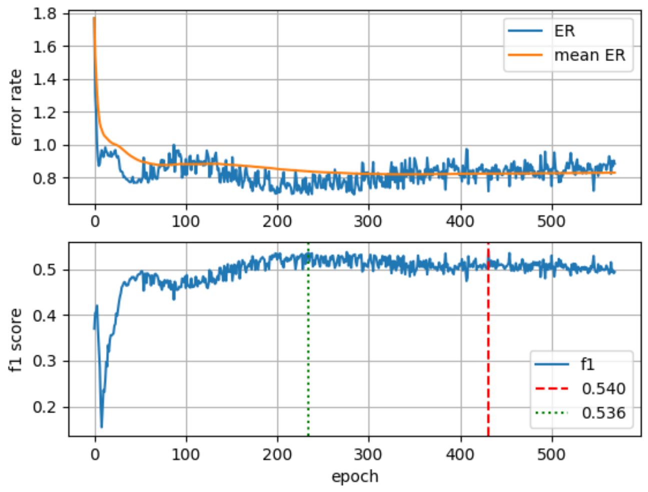

Figure 8: C3RNN gcc and mbe features with dropout = 0.5, best f1 (- - - -), f1 for the best ER (······).

C3RNN	**ER**	**F1**
dropout = **0.2**		
mbe-gcc	70.6	51.9
mbe	70.2	51.2
dropout = **0.5**		
mbe-gcc	72.3	**53.2**
mbe	74.7	52.4

Table 3: Evaluation metric scores for SED using the C3RNN.

CRNN	**ER**	**F1**
dropout = **0.2**		
mbe	70.5	50.3
dropout = **0.5**		
mbe	71.4	51.9

Table 4: Evaluation metric scores for SED using the CRNN with mbe features only.

6 Conclusion

Although our results are really close to the goal, we failed in reaching the state-of-the-art. One possible way to improve the score could be through the use of **minibatch** as validation for each epoch, instead of the entire batch. This could avoid overfitting as mentioned in [3].
Another unclear point in the paper is about the imaginary part of the complex gcc, which correspond to the Phase. We just discarded that part and focused on the real one, but a way to improve the results could have been creating another branch fed with a matrix filled by complex part, and merging it to the real part at the end of the convolutional block of the model.
The small improvement given through the use of gcc does not worth the huge increment in trainable parameters (30% more), also due to the heavy computation for the extraction of the gcc.
The major issue in reproducing the results given in the paper is the lack of information about the model, the use of the features and hyper-parameters that brought us to guess some values and try several configurations taking into account some other different papers cited in references. We actually manage to reach the results cited in [5].

References

[1] Sharath Adavanne, Archontis Politis, and Tuomas Virtanen. "Multichannel Sound Event Detection Using 3D Convolutional Neural Networks for Learning Inter-channel Features". In: *CoRR* abs/1801.09522 (2018). arXiv: `1801.09522`. URL: `http://arxiv.org/abs/1801.09522`.

[2] Sharath Adavanne and Tuomas Virtanen. "A report on sound event detection with different binaural features". In: *CoRR* abs/1710.02997 (2017). arXiv: `1710.02997`. URL: `http://arxiv.org/abs/1710.02997`.

[3] Il-Young Jeong et al. "Audio Event Detection Using Multiple-Input Convolutional Neural Network". In: (Nov. 2017), pp. 51–54.

[4] Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen. "Metrics for polyphonic sound event detection". English. In: *Applied Sciences* 6.6 (2016). ISSN: 2076-3417. DOI: `10.3390/app6060162`.

[5] Annamaria Mesaros et al. "DCASE2017 Challenge Setup: Tasks, Datasets and Baseline System". In: *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2017 Workshop (DCASE2017)*. Nov. 2017, pp. 85–92.