Speaker Diarization On Telephony Audio Using Deep Learning Neural Networks

Bernie Armour
Machine Learning Engineer Nanodegree Program
Capstone Project

Contents

Project Overview 3
Background 3
Problem Statement 3
Metrics 3
Analysis 5
Data Exploration 5
Audio Data Preprocessing and Visualization 6
Visualizing the Superframes 7
Algorithms and Techniques 9
Methodology 10
Implementation: 2D-CNN Only Solution 10
Development of a convolutional neural network for speaker classification 10
Development of a speaker change function 15
Evaluation of the 2D-CNN Classifier 15
Alternative Speaker Change Function: Gaussian Mixture Clustering 18
Test Results for 2D-CNN Diarization of 2-Speaker Recordings 18
Implementation: Recurrent-CNN Approach 19
R-CNN Architecture Implemented 20
Results 29
Conclusion 31
Source Code 33
Training Functions 34
Diarization Function 34
References 35

List of Tables

Table 1**:** 2-D CNN Development Test Results..... 11
Table 2**:** 2D-CNN Diarization Results Using 260 Class Feature Vector SWBCell2 Tests of 595 Balanced Recordings, 161,501 superframes diarized 19
Table 3: R-CNN Model Development Test Results 22
Table 4: R-CNN (2D-CNN + Bidirectional GRU) Diarization (Model Test 4) Diarization Results Using 260 Class Feature Vector SWBCell2 Tests of 595 Balanced Recordings, 161,501 superframes diarized 28
Table 5: Accuracy Test Results using the SWBPhase3 data set to compare R-CNN and 2-D CNN Models. 29
Table 6: SWBCell2 tests with recordings containing one dominant speaker. 30
Table 7: SWBCell2 tests with near equal amounts of speech from each of the 2 speakers. 30
Table 8: Comparison of the Datasets and Features in the work in **[Cyrta,2017]** and the in our project..... 31

List of Figures

Figure 1: Cepstrum superframes. Each row is a mel cepstrum of 32 coefficients. Coefficient 0 (coefficient 0 is set to 0 in all cepstrums since it a measure of energy which is not useful for speaker differentiation. Time increases from top to bottom. "mel frequency" increases from left to right. 8
Figure 2: Model Accuracy: Training history over 1000 epochs for the final model 2D-CNN Network (Test 6) ... 14
Figure 3: Model Loss: Training history over 1000 epochs for the final model 2D-CNN Network (Test 6) 14
Figure 4: Typical 2D-CNN classification results using the 260 class softmax activation layer to classify each superframe in this recoding of length 520 superframes. 17
Figure 5: Model Accuracy: Training history over 1000 epochs for the final model: R-CNN Model 4 26
Figure 6: Model Loss: Training history over 1000 epochs for the final model: R-CNN Model 4 26
Figure 7: Model Accuracy: Training history over 1000 epochs for the final model: R-CNN Model 4 with Batch Normalization after each Convolution layer..... 27
Figure 8: Model Loss: Training history over 1000 epochs for the final model: R-CNN Model 4 with Batch Normalization after each Convolution layer..... 27

Project Overview

Background

Speaker diarization is the process by which an audio recording is divided into segments where each segment is assigned a label according to the speaker's identity. Therefore, speaker identification answers the question of "who spoken when" in an audio recording. The actual identity of the speakers is not determined. Instead speaker diarization seeks to distinguish between each unknown speaker in the audio recording. Applications of speaker diarization include:

- analysis of phone conversations, meetings, legal proceedings and transcripts
- speaker identification
- speech to text with speaker annotations

Speaker diarization has been an active area of research over the last 20 years. In particular the NIST Rich Transcription Evaluations in 2003 and 2004 yielded a variety of approaches for speech detection and segmentation (see Rich Transcription Evaluation). These approaches were based on conventional algorithms using spectral feature extraction, change detection, and feature clustering. A significant advancement was achieved based on the application of joint factor analysis to generate an i-vector feature which compactly encompasses the differentiating information for speaker identification and speaker change detection [Kenny, 2010]. This work focused on diarization of telephony audio.

More recently researchers have been experimenting with neural networks to achieve lower error rates in diarization. In [Cyrta, 2017], the authors developed a "recurrent convolutional neural network" (R-CNN) that they trained to classify short audio segments in a large training set of annotated speaker segments. The resulting classifier has activation layers which are said to contain "speaker embeddings". These speaker embeddings are considered to be sufficiently generalized such that they can be applied to other sources of audio recordings and speakers who are not part of the training set. It is similar to the networks developed to achieve high accuracy with ImageNet, but they are also useful for transfer learning to other related image classification tasks. Speaker diarization is a very difficult task that continues to attract new research work. Unfortunately, even the best solutions are still achieving 10% or higher error rates.

Problem Statement

In this project, the goal is to follow the approach used in [Cyrta, 2017] to design, implement, and test a speaker diarization solution based on a deep neural network solution. Instead of working with high quality broadcast audio, regular land-line and mobile (cellular) telephone audio will be the audio source. To measure the results in experiments a Diarization Error Rate is defined and applied to the diarization tests.

Metrics

Evaluation Metrics

It was initially proposed that a diarization error rate combining speech activity detection and speaker assignment would be used, however, when working with the telephone recordings it soon became apparent that a simpler metric can be defined.

Speech activity detection simply determines in which parts of a recording there is someone speaking. All the other parts are either silence or some non-speaker noise. This is quite easy to detect and not the focus of this

project. Therefore, we define the diarization error metric as the accuracy to which we classify each speech unit of analysis.

After removing the non-voice parts of a recording, the audio is sliced into a sequence of 2-second segments. Each 2-second segment is labeled by a speaker label. If the segment contains less than 75% of speech by one speaker, then it is labeled as double talk.

The Diarization Error Rate (DER) is defined as follows:

DER = Count of segments correctly labeled / total number of segments

All double talk segments are ignored since they contain more than one speaker.

Analysis

Data Exploration

Datasets

- Switchboard Cellular Part 2 Audio https://catalog.ldc.upenn.edu/LDC2004S07

  "Total of 2,020 calls, or 4,040 sides (2,950 cellular) from 419 participants (2,405 female speakers, 1,635 male speakers) under varied environmental conditions."

- Switchboard-2 Phase III Audio https://catalog.ldc.upenn.edu/LDC2002S06

  "Total of 2,728 calls, or 5,456 sides, from 640 participants (292 Male, 348 Female), under varied environmental conditions."

The two datasets listed above contain stereo channel audio recordings of telephone conversations. In each recording there are two persons speaking: one speaker per channel. A voice activity detector (VAD) is used to mark the speech segments in each channel. These segments are then used to create speaker labels: speaker A, speaker B, speaker A+B (double talk). In addition, speakers A and B are assigned PIN numbers uniquely identifying the person speaking.

The original 2-channel audio files with PIN identifiers can be directly used for training a speaker classifier. We simply use one channel at a time and use a VAD to only process the speech parts in the recordings.

For speaker diarization testing, we of course require that the source audio consists of only one channel containing multiple speakers. To achieve this, we will simply sum the two channels in the test audio set. The segments for speaker A, B and A+B that were generated using the 2-channel recordings are used as the truth for the diarization outcome. We use this info to calculate the Diarization Error Rate (DER) for the test dataset.

Switchboard-2 Phase III Audio dataset consists of only landline telephone call recordings. In comparison, Switchboard Cellular Part 2 Audio contains a mix of landline, CDMA, GSM, TDMA cellular phone recordings. Since there are over 4000 audio recordings and 419 participants, the Switchboard Cellular Part 2 Audio data set on its own is sufficiently rich for training a robust model for speaker identification and speaker separation.

We will use the Switchboard-2 Phase III Audio data set as a test set to evaluate the final diarizer solution.

The audio recordings are formatted as PCM samples, u-law, 8000 samples/second, two channels per recording. Each recording has associated metadata of:

- personal identification number (PIN) for each of the two speakers
- male/female
- phone technology (landline, CDMA, GSM, TDMA).

Each recording contain 5 minutes of audio with each speaker taking turns speaking in natural conversation.

Given that this is a NIST trial dataset, it has been cleaned and preprocessed to remove bad samples such as bad audio quality or recordings with no audio at all. The audio recordings are realistic in that some audio is clipped, some audio has low power, and most of the recordings contain some double talk. Audio preprocessing is discussed in the Methodology section.

Audio Data Preprocessing and Visualization

The SWBCELL2 dataset contains 4040 telephone conversation recordings. The telephony technologies represented in this dataset consists of landline, CDMA, GSM, TDMA systems.

To prepare the data, the data set was organized into two groups: male speakers, and female speakers. In total, there are 377 speakers. The speakers were sorted by number of recordings per speaker and the speakers with the highest number of recordings were selected. This resulted in a list of the top 130 male and top 130 female speakers. There are at least 8 telephone recordings for each individual speaker.

For each speaker, a personal identifier number (PIN) is assigned. In this project, the digital audio samples are not used directly for model training or testing. Instead, each recording is preprocessed to extract spectral magnitude data. The software to do this as well as the programs to organize the audio files for processing was developed using C++ and C#. These programs are not included in the project submission, however, the resulting preprocessed data is included as this is the training and test data used in the neural networks.

Each recording channel contains a single speaker and is 5 minutes in length. The recording is in fact one side of a phone conversation in each channel. Each audio recording channel is processed into a sequence of features as follows:

- Apply voice activity detection to identify speech parts.
- For the speech parts only, calculate the mel-spaced cepstrum coefficients for each frame of audio samples using a frame length of 64ms. The result is a sequence audio frames. Each frame represents the audio spectrum of the speech in a 64ms interval. The frame contains 32 cepstrum coefficients.
- Starting from the time-ordered list of cepstrum frames, select (in time ordered sequence sets) 64 frames (2.048s of cepstrum frames) and store these as *superframes*.
- This superframe is the audio unit which we attempt to classify as having originated from a single speaker. The 64 cepstrum frames in a superframe represent the audio spectrum as it varies over 2.048 seconds of speech.
- A superframe is created with a step advance of 25%. That is, each successive superframe overlaps the previous one by about 0.5s.
- For 2D-CNN use, we require a 2-D array similar to an image. The superframe gives us this.

For audio, our superframe is by analogy an image of 64 rows by 32 columns. Each row is one frame containing the spectral content for 64ms of speech. Each of the 32 values in the frame is a measure of audio frequency in a narrow band. Each new row in the 2-D array is a new spectrum measure taken 32ms after the previous frame. Therefore, successive rows hold a short time evolution of the spectral content.

The above processing results in approximately 500 to 2000 superframes per individual speaker. These are divided into three subsets per speaker as follows:

For each speaker PIN:

- 80% of the superframes are used for training
- 10% are used for validation
- 10% are retained for testing

In total there are 260 speaker PINs:

- 130 male speakers
- 130 female speakers.

Visualizing the Superframes

The resulting superframes were rescaled to fall in the range 0.0 to 1.0. This was calculated by finding the global min value and max value over all the cepstrum coefficients in the entire dataset. The results were:

min_feature_val = -13.288

max_feature_val = 9.952

(See python code: GetMinMaxVals.py)

These values were then used to rescale all the data to be in the range of [0.0, 1.0].

A selection of superframes for one speaker was then plotted as heatmaps. Each heatmap plot consists of 64 rows with each row containing one mel cepstrum. Time increases from top to bottom in each picture. "mel frequency" increases from left to right.

(See python code SampleMelSpectrogram.py)

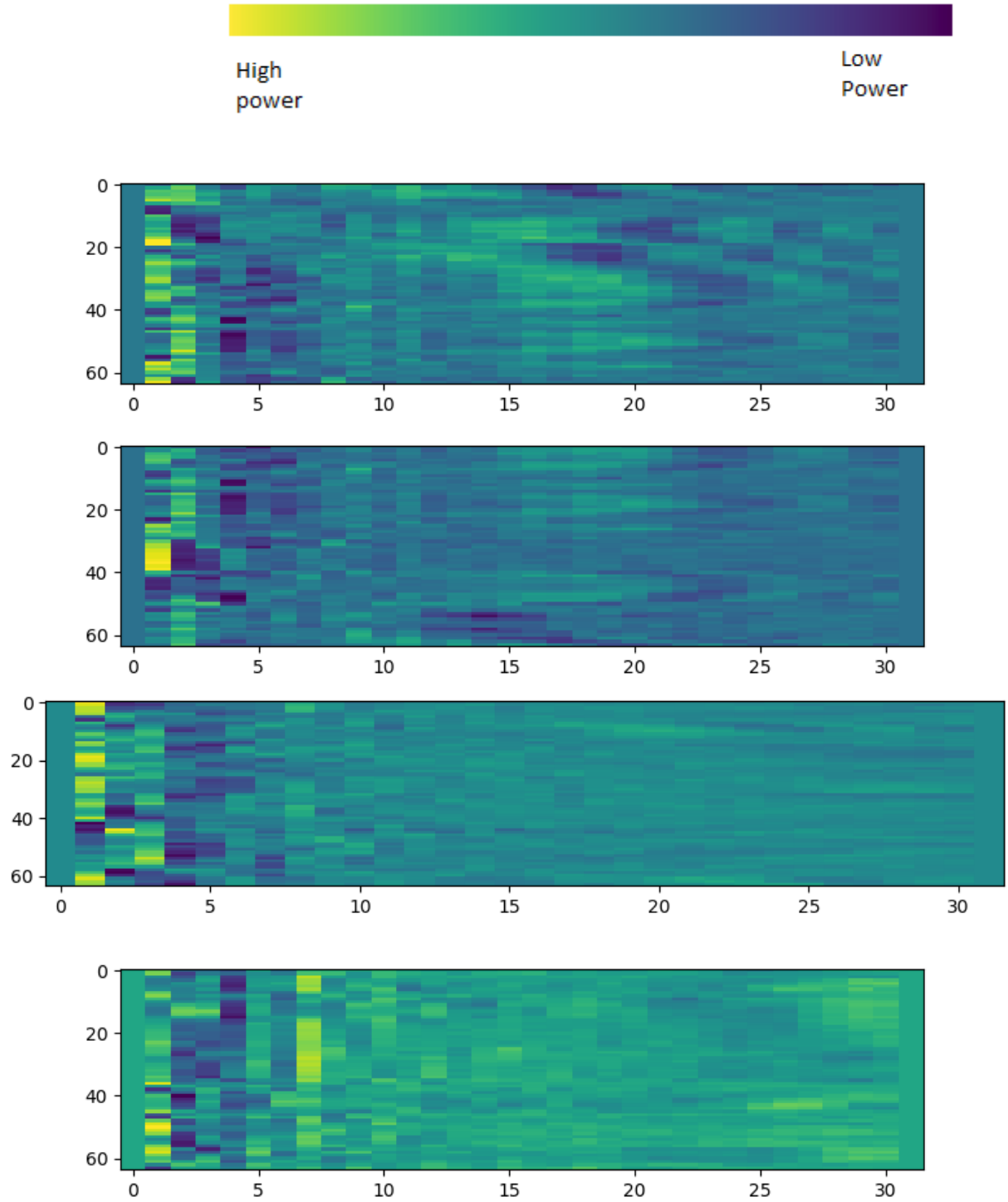


Figure 1: Cepstrum superframes. Each row is a mel cepstrum of 32 coefficients. Coefficient 0 (coefficient 0 is set to 0 in all cepstrums since it a measure of energy which is not useful for speaker differentiation. Time increases from top to bottom. "mel frequency" increases from left to right.

If the visualization we can see there is both vertical and horizontal structure in the superframe. Most analysis methods use only the horizontal variations which represents a snapshot of spectral energy from low to high frequencies in a 64ms window. The superframe however shows vertical structure. This represents the time variation of spectral power in one frequency band as time increases by 32ms per row for a total of 2.048s from top to bottom.

This two-dimensional structure helps to explain why the authors in the reference paper [Cyrta, 2017] chose to use 2-D CNNs instead of the more common practice of one-dimensional convolution of the time series audio signal or spectrum.

Algorithms and Techniques

The proposed solution is to follow the implementation published in [Cyrta, 2017]. Their R-CNN design was a combination of 2-D CNN layers and GRU layers. This network was trained to perform speaker identification. The final layer containing the softmax output was then used as the "embedded speakers" feature vector for speaker change detection. The authors split the voice audio into 3-second segments. They trained their network on a variety of broadband audio recordings including broadcast news and YouTube recordings.

In [Cyrta, 2017], the R-CNN was implemented with Keras and is described as follows for training the speaker classifier:

1. Input
2. 4 convolutions layers.
    a. Layer 1: 3X3 convolution, batch-normalization layer, max pooling layer 2X2 non-overlapping stride, ELU
    b. Layer 2: 3X3 convolution, batch-normalization layer, max pooling layer 3X3 non-overlapping stride, ELU
    c. Layer 3: 3X3 convolution, batch-normalization layer, max pooling layer 4X4 non-overlapping stride, ELU
    d. Layer 4: 3X3 convolution, batch-normalization layer, max pooling layer 4X4 non-overlapping stride, ELU
3. 2 recurrent layers with GRU gating
4. 1 fully connected layer with presumably softmax and C outputs (C = number of speaker classes).

They do not describe the details of the GRU layers. They favored the ELU activation function over RELU.

To detect speaker changes, they used a speaker change function that assigned the highest scoring classified speaker to each 3-second segment.

In our project, we are working with narrow-band audio (8000 samples/second) and conversational content. Accordingly, two-second segments were chosen to achieve finer time granularity for the speaker labelling. We also chose to evaluate a CNN-only solution first and then add in the GRU layers and evaluate any performance improvements.

Given that we are working with a one-dimensional audio signal converted to a two-dimensional 64x32 superframe with time by frequency units, it is not at all clear how 2-D convolution should be applied. This is not an image after all. So a series of experiments with increasing number of CNN layers and convolutions sizes were evaluated.

Methodology

Implementation: 2D-CNN Only Solution

In this approach we use only the 2D-CNN to classify speakers and extract the diarization result.

Following the methodology given in the reference paper [Cyrta, 2017] and excluding the data preprocessing , there are two parts to be developed:

Part 1: Develop a convolutional neural network for speaker classification. This classifier is in itself a 2D-CNN with a classification hopefully a low error rate. It might be used for speaker identification. In our case, however, we use the final activation layer network as a source of time-dependent features called *speaker embeddings*.

Part 2: Develop a speaker change function. For diarization, we care only about speaker changes. Is it currently Speaker A, or is it a new speaker starting, say Speaker B? We use the speaker embedding as a source of time ordered features (one per super frame) and detect changes from one feature to another using a cosine similarity measure. A large change indicates a new speaker starting, a small change indicates the current speaker is continuing. If a speaker change is detected, we compare the new speaker with the previous speakers seen and decide whether this is a new speaker or a previously seen speaker.

Development of a convolutional neural network for speaker classification

Development Data:

- 230 single-speaker data sets consisting of 200 superframes (64 X 32).
- 230 speakers (130 male, 130 female)
- Superframes allocation per speaker:
    - 150 superframes for training
    - 25 superframes for validation
    - 25 superframes for testing

Development of the CNN network progressed in steps starting from a simple network and progressing in stages. (See source code for training the CNN at:

https://github.com/BFrancisA/Diarize/blob/master/Train-cnn.py

Table 1: 2-D CNN Development Test Results

2D-CNN
 	
In all tests:
- Number of speaker targets 260
- input_shape=(64, 32, 1)
- All Conv2D use: padding='same', activation= 'relu'
- Model.compile use:
  o loss='categorical_crossentropy',
  o metrics=['accuracy']
- Training 2000 epochs
	Result

Test 1: Conv2D(filters=32, kernel_size=3, …) Conv2D(filters=64, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Flatten() Dense(speaker_target_names_count, activation='softmax') model.compile(optimizer='rmsprop')	Test accuracy: 47.7% Total params: 2,148,996
Test 2: 3 conv layers Conv2D(filters=32, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Conv2D(filters=64, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Conv2D(filters=64, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Flatten() Dense(speaker_target_names_count, activation='softmax') model.compile(optimizer='rmsprop')	Test accuracy: 55.0% Total params: 568,004
Test 3: 3 conv layers with dropouts Conv2D(filters=32, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Dropout(0.1) Conv2D(filters=64, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Dropout(0.1) Conv2D(filters=64, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Dropout(0.1) Flatten() Dense(speaker_target_names_count, activation='softmax') model.compile(optimizer='rmsprop')	Test accuracy: 54.9% Total params: 557,604
Test 4: 3 conv layers with dropouts, dropout added to dense layer Conv2D(filters=32, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Dropout(0.1)	Test accuracy: 63.7 Total params: 560,804

Conv2D(filters=32, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Dropout(0.1) Conv2D(filters=64, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Dropout(0.1) Flatten() Dropout(0.3) Dense(speaker_target_names_count, activation='softmax') model.compile(optimizer='rmsprop')	
Test 5: 4 conv layers with dropouts after all layers Conv2D(filters=32, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Dropout(0.2) Conv2D(filters=32, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Dropout(0.2) Conv2D(filters=32, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Dropout(0.2) Conv2D(filters=64, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Dropout(0.2) Flatten() Dropout(0.3) Dense(speaker_target_names_count, activation='softmax') model.compile(optimizer='rmsprop')	Test accuracy: 62.3% Total params: 170,692
Test 6: same as Test5, but removed first max pooling layer and reduced learning rate Conv2D(filters=32, kernel_size=3, …) ~~MaxPooling2D(pool_size=2, strides=(2, 2))~~ Dropout(0.2) Conv2D(filters=32, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Dropout(0.2) Conv2D(filters=64, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Dropout(0.2) Conv2D(filters=64, kernel_size=3, …) MaxPooling2D(pool_size=2, strides=(2, 2)) Dropout(0.2) Flatten() Dropout(0.3) Dense(speaker_target_names_count, activation='softmax') rmsprop_slow = optimizers.RMSprop(lr=0.0001, rho=0.9, epsilon=None, decay=0.0) model.compile(optimizer=rmsprop_slow, …)	Test accuracy: 70.4% Total params: 597,732

The results for the network in Test 6 are quite good considering that there are 260 possible speakers and the test audio is only 2 seconds long. A random guess would be 1/260*100 = 0.38% accurate. The final network achieved about 70% accuracy. Though it is possible that the results could be incrementally improved with more experiments, it is likely that the improvements will be small given that the amount of audio per superframe is so small. Furthermore, our objective is to create a feature vector to be used in detecting speaker changes from one superframe to the next. The actual speaker identification is not required.

The network final 2D-CNN Network (Test 6) has the following model summary:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 64, 32, 32)        320
_________________________________________________________________
dropout_11 (Dropout)         (None, 64, 32, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 64, 32, 32)        9248
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 32, 16, 32)        0
_________________________________________________________________
dropout_12 (Dropout)         (None, 32, 16, 32)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 32, 16, 64)        18496
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 16, 8, 64)         0
_________________________________________________________________
dropout_13 (Dropout)         (None, 16, 8, 64)         0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 16, 8, 64)         36928
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 8, 4, 64)          0
_________________________________________________________________
dropout_14 (Dropout)         (None, 8, 4, 64)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 2048)              0
_________________________________________________________________
dropout_15 (Dropout)         (None, 2048)              0
_________________________________________________________________
dense_5 (Dense)              (None, 260)               532740
=================================================================
Total params: 597,732.0
Trainable params: 597,732.0
Non-trainable params: 0.0
_________________________________________________________________
```

The training history for the Test 6 CNN is plotted in Figure 2 and Figure 3. It can be seen that the model accuracy and loss have levelled out after about 800 epochs. There is no sign of overfitting.

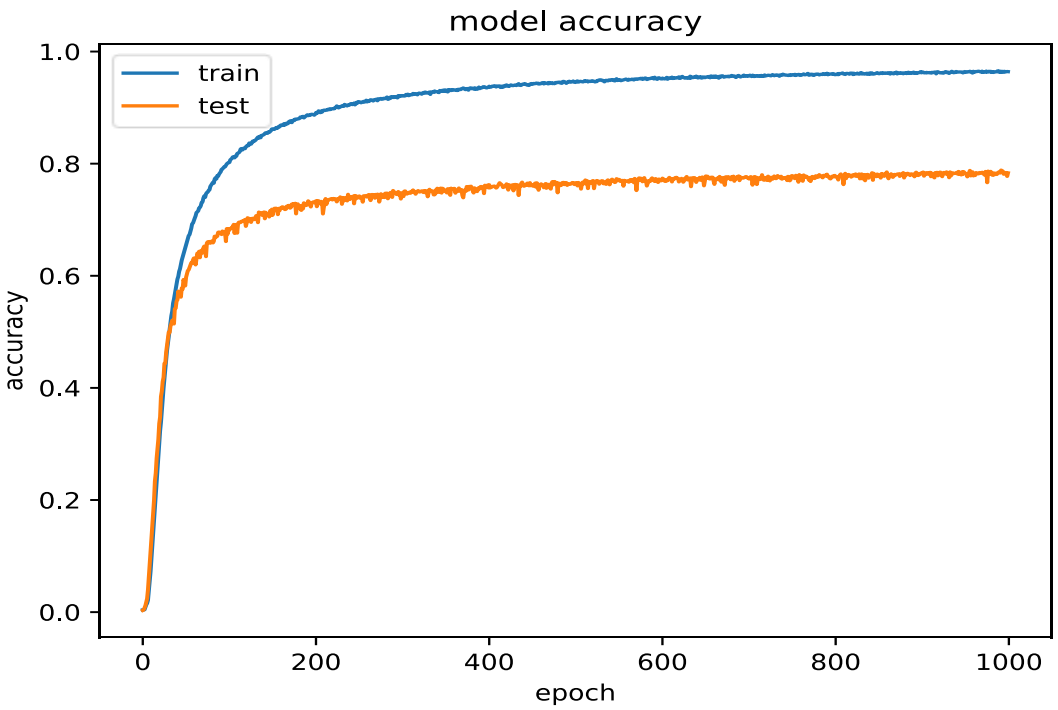


Figure 2: Model Accuracy: Training history over 1000 epochs for the final model 2D-CNN Network (Test 6)

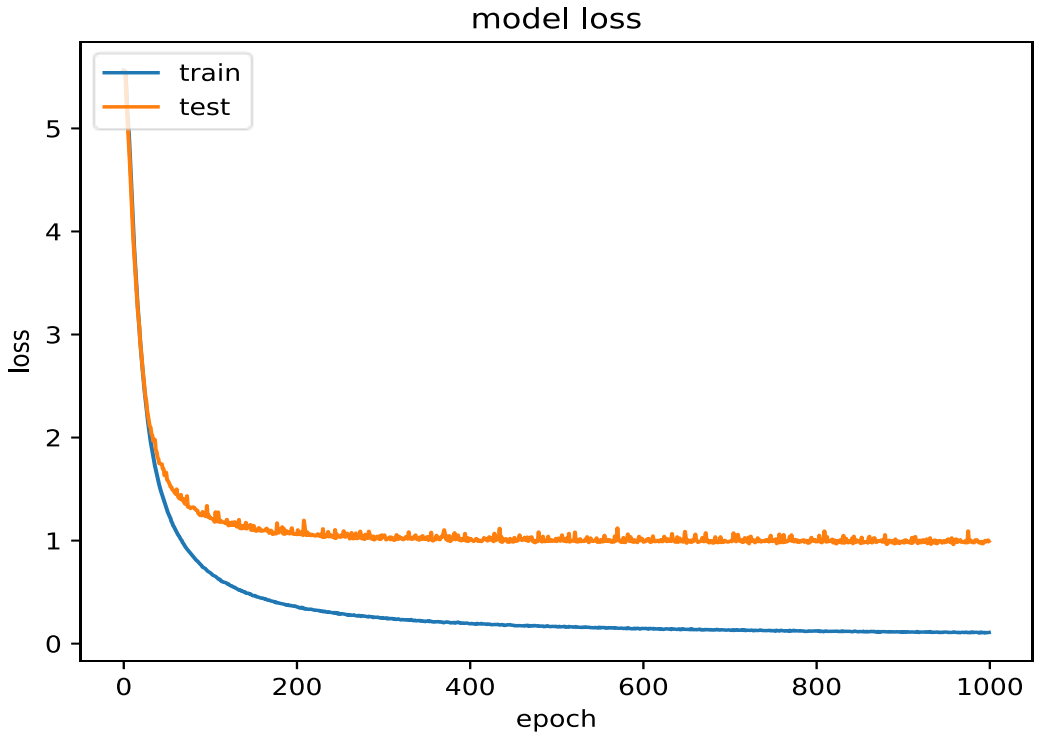


Figure 3: Model Loss: Training history over 1000 epochs for the final model 2D-CNN Network (Test 6)

Development of a speaker change function

Cosine Similarity Speaker Change Function

Given an audio recording converted to a sequence of superframes:

1. For each superframe: use the 2D-CNN to calculate the softmax output of speaker probabilities. This results in a 1 X 260 "speaker embedding" vector. There is one such vector superframe.
2. Sum and normalize the speaker embeddings vectors for the audio recording:
    a. Sum all vectors to get 1 X 260 sum vector.
    b. Determine the max value in the sum vector. Use this to normalize all the vector coefficients in all the speaker embeddings feature vectors.

    The above normalization is outlined in the reference paper [Cyrta, 2017]. The normalization is necessary because we use a constant threshold value for detecting speaker changes as described in the next step.

3. Classify each normalized feature vector, v_t, in sequence, for the whole recording. This is done as follows. The first feature vector in the sequence is assigned the label Speaker1. For each vector, v_t:
    a. Calculate the cosine similarity between v_{t-1} and v_t.
    b. If the change exceeds a threshold:
        i. Test if v_t is similar to a previously assigned speaker. If so assign the previous speaker.
        ii. Otherwise, assign a new speaker (e.g. speaker 2).
    c. Else change is less than the threshold. Assign the current speaker.

The output of the speaker change function is therefore just a sequence of speaker labels. One speaker label per superframe of the original audio recording.

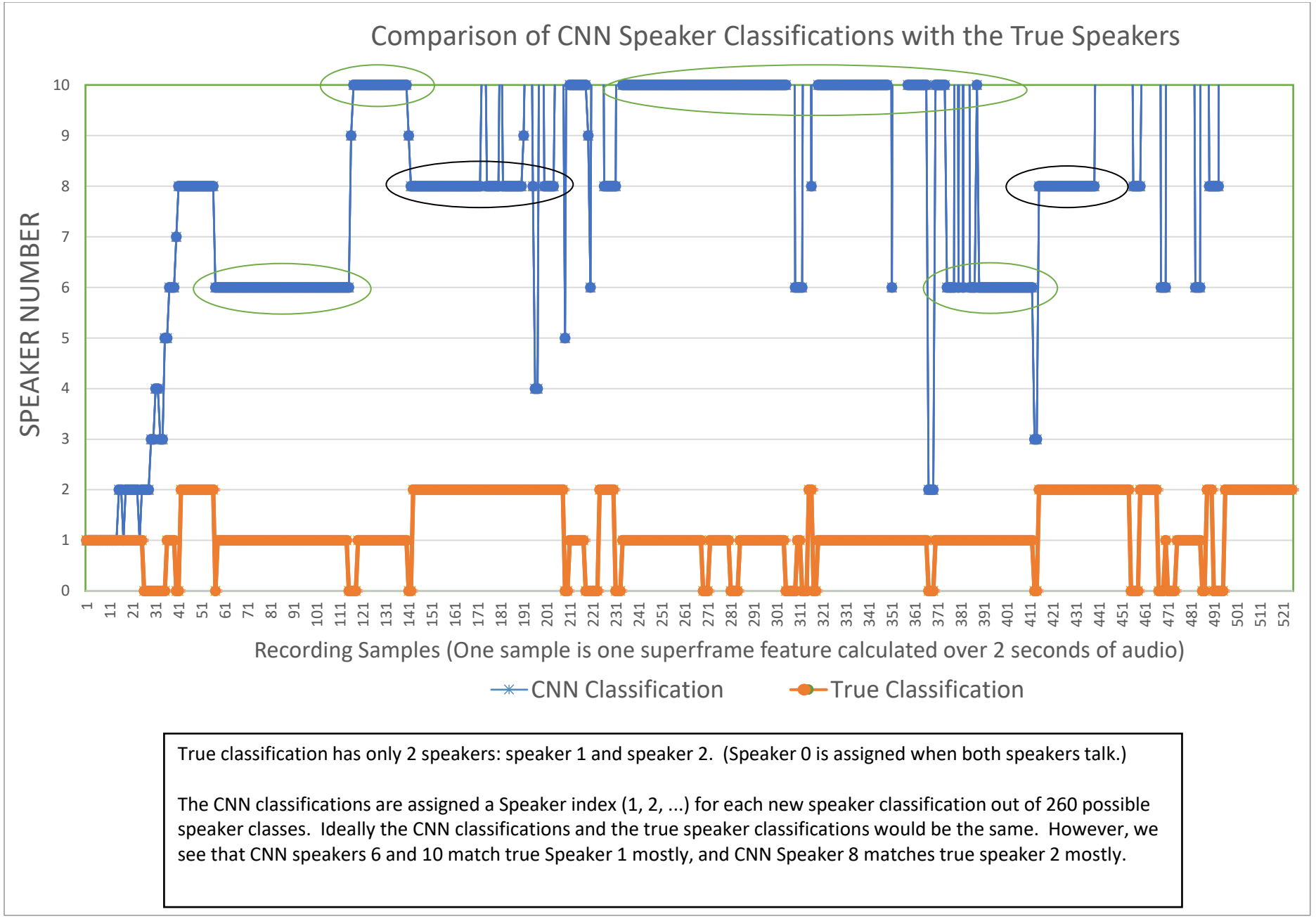
Evaluation of the 2D-CNN Classifier

Using the 260 class *softmax* as a classifier in the Speaker Change Function resulted in many errors in the speaker class assignments. A typical example is shown in Figure 4: Typical 2D-CNN classification results using the 260 class softmax activation layer to classify each superframe in this recoding of length 520 superframes.Figure 4. We see that the CNN does detect speaker changes well enough, but it fails to consistently classify all of the true speaker's samples to the same speaker index. We see that the true Speaker 1 is classified as Speaker 6 and Speaker 10 among other speakers. In the figure, the samples where the true speaker is 0 (points right on the x-axis) indicate that there was double talk for these samples. So, at these points we can expect that there will be somewhat random results.

What we would like is for the CNN to classify most of the true speaker's samples to just one speaker class (ignoring the double talk cases). To do this, it must determine not just when there is a speaker change, but also if the new speaker has been seen before. Clearly the CNN classifier is failing to consistently recognize the speakers that it has detected earlier in the recording.

Remember that we are not trying to classify the current speaker to one speaker class. We are instead using the softmax activation as a feature vector and we are using the cosine distance function to compare a series of feature vectors. Nevertheless, it seems possible that 260 speaker classes is too large a number of classes for

this task. There maybe very little difference between two classes for some speakers. As a result, one speaker may randomly be assigned to any of 5 or more classes.



True classification has only 2 speakers: speaker 1 and speaker 2. (Speaker 0 is assigned when both speakers talk.)

The CNN classifications are assigned a Speaker index (1, 2, ...) for each new speaker classification out of 260 possible speaker classes. Ideally the CNN classifications and the true speaker classifications would be the same. However, we see that CNN speakers 6 and 10 match true Speaker 1 mostly, and CNN Speaker 8 matches true speaker 2 mostly.

Figure 4: Typical 2D-CNN classification results using the 260 class softmax activation layer to classify each superframe in this recoding of length 520 superframes.

Alternative Speaker Change Function: Gaussian Mixture Clustering

An alternative speaker change function is to use clustering to discover the speaker classes for a recording and then assign each feature to one of the classes. A simple approach was taken as follows:

1. Start with the feature sequence for one recording.
2. Run Gaussian Mixture (GM) clustering (sklearn.mixture.GaussianMixture) using n_components=2.
3. Calculate the silhouette score (sklearn.metrics.silhouette_score).
4. In steps of 1, increase n_components and recalculate the silhouette score.
5. Stop increasing n_components when the silhouette score hits a minimum.
6. Use the GM model to classify each superframe to a cluster.
7. The cluster assignments are the speaker labels.

K-Means was also tested for clustering however it gave very poor fitting results for a low number of clusters. The Gaussian Mixture model was found to fit the features quite accurately with silhouette scores < 1.0 for all recordings. Furthermore, the optimal number of components was reliably 2 or 3. This is in fact accurate. Each test recording has 2 speakers and there are many cases of double-talk. The third speaker discovered by the clustering is likely the double-talk case.

Testing was also run using PCA fitting with the transformation applied to the feature sequence prior to Gaussian Mixture clustering. The results however were quite poor. It is likely that there is insufficient data available per audio recording for accurate PCA eigenvector estimation. The number of superframes per audio recording is typically in the range of 200 to 400.

Test Results for 2D-CNN Diarization of 2-Speaker Recordings

The test results shown below were obtained by running the 2D-CNN Model and Speaker Change Function on 595 audio recordings. These recording were selected from a total of 1,885 recordings. Each recording contains a five-minute telephone conversation with exactly two speakers. The 595 recordings for the evaluation were selected using the requirement that both speakers have at least 37.5% of the superframes assigned to them. This data set is referred to as a balanced data set since it has significant presence of both speakers.

In the test results in Table 2, all feature vectors that correspond to double-talk superframes are excluded from the error calculation. These superframes contain both speakers talking at the same time.

Table 2: 2D-CNN Diarization Results Using 260 Class Feature Vector
SWBCell2 Tests of 595 Balanced Recordings, 161,501 superframes diarized

Change Detection Function	**Total Overall Diarization Error**
Cosine Similarity Speaker Change Function	74.8 %
Gaussian Mixture Clustering with Cluster Count Detection	18.3 %
Gaussian Mixture Clustering with Number of Clusters Fixed at 2	12.6%

The speaker change detector based on Gaussian Mixture clustering is clearly required to get reasonable results. Using the silhouette score to detect the number of clusters resulted in an error of 18.3%. In comparison, using the prior knowledge that there are exactly 2 speakers per recording, the error is reduced to 12.6%. For telephone conversations, it is usually the case that there are only 2 speakers, however, conference calls or skype meetings may contain multiple speakers. Therefore, the use of cluster count detection is the more general solution.

Implementation: Recurrent-CNN Approach

The reference paper *Speaker Diarization using Deep Recurrent Convolutional Neural Networks for Speaker Embeddings* [Cyrta, 2017] used the following network architecture:

1. Input
2. 4 convolutions layers.
    a. Layer 1: 3X3 convolution, batch-normalization layer, max pooling layer 2X2 non-overlapping stride, ELU
    b. Layer 2: 3X3 convolution, batch-normalization layer, max pooling layer 3X3 non-overlapping stride, ELU
    c. Layer 3: 3X3 convolution, batch-normalization layer, max pooling layer 4X4 non-overlapping stride, ELU
    d. Layer 4: 3X3 convolution, batch-normalization layer, max pooling layer 4X4 non-overlapping stride, ELU
3. 2 recurrent layers with GRU gating

4. 1 fully connected layer with presumably softmax and C outputs (C = number of speaker classes).

The authors, Cyrta *et al*, do not describe how the output of the final convolution layer is converted to a sequence of features to input to the recurrent layers. Using references [Zuo, 2015], [Cakir, 2017], [Adavanne, 2017], it appears that the following process should be used to convert the output of the final 2D-CNN layer to a sequence that is input to the recurrent layer.

Given a R X C X N_f (rows by columns by number of filters) output from the last 2D-CNN layer:

- Convert the R X C matrix of N_f filters to a sequence of R*C X N_f where the rows are concatenated to a one-dimensional array.

For example, if the output of the final CNN layer is 8 X 4 X 64 filters, then the sequence is 32 vectors where each vector 64 has coefficients.

A superframe consists of a time sequence of 64 frequency spectrums. Over the process of CNN filtering this time sequence becomes a higher order encoding of feature change in both time and mel-cepstrum "frequency". To capture sequence info within the superframe, the recurrent layer is expected learn time or frequency order patterns.

Given that the output of the R-CNN network is still a softmax vector of $N_{targets}$ classes, we can use the same clustering speaker change function as was used with the output of the 2D-CNN model.

R-CNN Architecture Implemented

The architecture implemented follows the example implemented by Adavanne [Adavanne, 2017] which can be found on GitHub as multichannel-sed-crnn.

As can be seen in the results tabled below, we start with:

Test 1:

- 4 2-D Conv layers (These are the same as used in our 2-D CNN model)
- 2 Bidirectional GRU layers
- 2 Time Distributed Dense layers
- 1 Dense layer with softmax output

The recurrent layers (GRU) operate on a sequence of vectors within the multidimensional sample that is the output of the last convolutional layer. Perhaps the best sequence of vectors is one that has leading dimension being the time dimension. The superframe is 64 X 32 where the row dimension is the time dimension, i.e. 64 frequency spectrums. Therefore, we lead the sequence with the row dimension.

The bidirectional wrapper around each recurrent layer has the affect of doubling the amount of data available for GRU training. It trains on the forward sequence and also the backward sequence. It is unclear what is the best way to combine the outputs from each direction. In the reference paper and network [Adavanne, 2017], they use value-be-value multiplication. This is not intuitive. It seems like summing or concatenating the two outputs would preserve any learned modelling.

The reference-based design [Cyrta, 2017] also uses two TimeDistributed layers. A TimeDistributed layer simply applies a fully connected dense layer to each temporal slice (a row in our case) of the input. It is hard see how this could improve results given that 2-dimensional convolution filtering was applied in four layers. By the output of the fourth convolutional layer, the time and frequency dimensions will have been extensively mixed together. As we will see in the test results, the TimeDistributed layers do not improve the classification results.

The development and test results are summarized in Table 3. In the results below, instead of building up a network we actually start with the reference network and then remove parts. After each modification, we measure the change in speaker classification accuracy.

Table 3: R-CNN Model Development Test Results

R-CNN Model
 	
In all tests:
- Number of speaker targets 260
- input_shape=(64, 32, 1)
- All Conv2D use: padding='same', activation= 'relu'
- Model.compile use:
  o loss='categorical_crossentropy',
  o metrics=['accuracy']
- Training 2000 epochs
	Result

Test 1: R-CNN Model 1
input_1 (InputLayer) (None, 64, 32, 1) 0
 	
Conv2D(filters=32, kernel_size=3, padding='same')
Dropout(0.2)
 	
Conv2D(filters=32, kernel_size=3, padding='same')
MaxPooling2D(pool_size=2, strides=(2, 2))
Dropout(0.2)
 	
Conv2D(filters=64, kernel_size=3, padding='same')
MaxPooling2D(pool_size=2, strides=(2, 2))
Dropout(0.2)
 	
Conv2D(filters=64, kernel_size=3, padding='same')
MaxPooling2D(pool_size=2, strides=(2, 2))
Dropout(0.2)
 	
# Convert 8 x 4 x 64 to 32 x 64 by concatenating rows 	
Reshape((-1, 64))
# Is multiplying the output of forward and reverse directions the right thing to do here? 	
# Sum appears to be best 	
Bidirectional(GRU(32, activation='tanh', dropout=0.2,
recurrent_dropout=0.2_rate, return_sequences=True),
merge_mode='sum')
Bidirectional(GRU(32, activation='tanh', dropout=0.2,
recurrent_dropout=0.2_rate, return_sequences=True),
merge_mode='sum')
 	
TimeDistributed(Dense(32))
Dropout(0.2)
TimeDistributed(Dense(32))
Dropout(0.2)
 	
Flatten()
Dropout(0.3)
out = Dense(speaker_target_names_count, activation='softmax', name='output')
model = Model(inputs=spec_input, outputs=out)
	Test accuracy: 71.4%
	
	Total params: 364,708

Test 2: R-CNN Model 2
Changes from Test 1:

- **Input shape to bidirectional GRU changed from (none 32, 16) in Test 1, to (none, 8, 256) in Test 2. This makes more sense given that the first dimension is the time dimension in the superframe).**
- **Replaced GRU with CuDNNGRU and added Dropput layers after each bidirectional GRU.**

The changes are highlighted below.

```
input_1 (InputLayer)        (None, 64, 32, 1)        0

Conv2D(filters=32, kernel_size=3, padding='same')
Dropout( 0.2)

Conv2D(filters=32, kernel_size=3, padding='same')
MaxPooling2D(pool_size=2, strides=(2, 2))
Dropout(0.2)

Conv2D(filters=64, kernel_size=3, padding='same')
MaxPooling2D(pool_size=2, strides=(2, 2))
Dropout(0.2)

Conv2D(filters=64, kernel_size=3, padding='same')
MaxPooling2D(pool_size=2, strides=(2, 2))
Dropout(0.2)

# Convert 8 x 4 x 64 to 8 x 256
Reshape((8, -1)

# Is multiplying the output of forward and reverse directions the right thing to do here?
# Sum appears to be best
 Bidirectional(CuDNNGRU(32, return_sequences=True), merge_mode='sum')
Dropout(dropout_rate)(spec)

Bidirectional(CuDNNGRU(32, return_sequences=True), merge_mode='sum')
Dropout(dropout_rate)(spec)

TimeDistributed(Dense(32))
Dropout(0.2)
TimeDistributed(Dense(32))
Dropout(0.2)

Flatten()
Dropout(0.3)
out = Dense(speaker_target_names_count, activation='softmax', name='output')
model = Model(inputs=spec_input, outputs=out)
```

Test accuracy: 69.7%
Total params: 202,276

Test 3: R-CNN Model 3
Changes from Test 1 and 2 :

- **Still using Input shape to bidirectional GRU set to (none, 8, 256).**
- **Replaced GRU with CuDNNGRU and added Dropout layers after each Bidirectional(GRU).**
- **Removed both TimeDistributed layers**

The changes are highlighted below.

input_1 (InputLayer) (None, 64, 32, 1) 0

Conv2D(filters=32, kernel_size=3, padding='same')
Dropout(0.2)

Conv2D(filters=32, kernel_size=3, padding='same')
MaxPooling2D(pool_size=2, strides=(2, 2))
Dropout(0.2)

Conv2D(filters=64, kernel_size=3, padding='same')
MaxPooling2D(pool_size=2, strides=(2, 2))
Dropout(0.2)

Conv2D(filters=64, kernel_size=3, padding='same')
MaxPooling2D(pool_size=2, strides=(2, 2))
Dropout(0.2)

Convert 8 x 4 x 64 to 8 x 256
Reshape((8, -1)

Is multiplying the output of forward and reverse directions the right thing to do here?
Sum appears to be best

Bidirectional(CuDNNGRU(32, return_sequences=True), merge_mode='sum')
Dropout(dropout_rate)(spec)

Bidirectional(CuDNNGRU(32, return_sequences=True), merge_mode='sum')
Dropout(dropout_rate)(spec)

~~TimeDistributed(Dense(32))~~
~~Dropout(0.2)~~
~~TimeDistributed(Dense(32))~~
~~Dropout(0.2)~~

Flatten()
Dropout(0.3)
out = Dense(speaker_target_names_count, activation='softmax', name='output')
model = Model(inputs=spec_input, outputs=out)

Test accuracy: 74.4%
Total params: 200,164

Test 4: R-CNN Model 4
Changes from Test 1, 2 and 3 :
- **Changed Bidirectional merge_mode from sum to concat**
The changes are highlighted below.
 	
input_1 (InputLayer) (None, 64, 32, 1) 0
 	
Conv2D(filters=32, kernel_size=3, padding='same')
Dropout(0.2)
 	
Conv2D(filters=32, kernel_size=3, padding='same')
MaxPooling2D(pool_size=2, strides=(2, 2))
Dropout(0.2)
 	
Conv2D(filters=64, kernel_size=3, padding='same')
MaxPooling2D(pool_size=2, strides=(2, 2))
Dropout(0.2)
 	
Conv2D(filters=64, kernel_size=3, padding='same')
MaxPooling2D(pool_size=2, strides=(2, 2))
Dropout(0.2)
 	
# Convert 8 x 4 x 64 to 8 x 256 	
Reshape((8, -1)
 	
# Is multiplying the output of forward and reverse directions the right thing to do here? 	
# Sum appears to be best 	
 	
Bidirectional(CuDNNGRU(32, return_sequences=True), merge_mode='concat')
Dropout(dropout_rate)(spec)
 	
Bidirectional(CuDNNGRU(32, return_sequences=True), merge_mode='concat')
Dropout(dropout_rate)(spec)
 	
Flatten()
Dropout(0.3)
out = Dense(speaker_target_names_count, activation='softmax', name='output')
model = Model(inputs=spec_input, outputs=out)
	Test accuracy: 75.1%
	Total params: 272,868

Test 5: R-CNN Model 4 with Batch Normalization **Same as Test 4 but with an added a batch_normalization layer after each of the Conv2D layers.**	Test accuracy: 74.4% Total params: 273,636

We switched from the GRU to the CuDNNGRU in order speed up the training. In all cases, 2000 epochs were run. Evidently the best results were achieved by using concatenation of the bidirectional outputs from the CuDNNGRU. The TimeDistributed layers added nothing to the accuracy of the classifications. In fact, for the same number of training iterations, the results deteriorated.

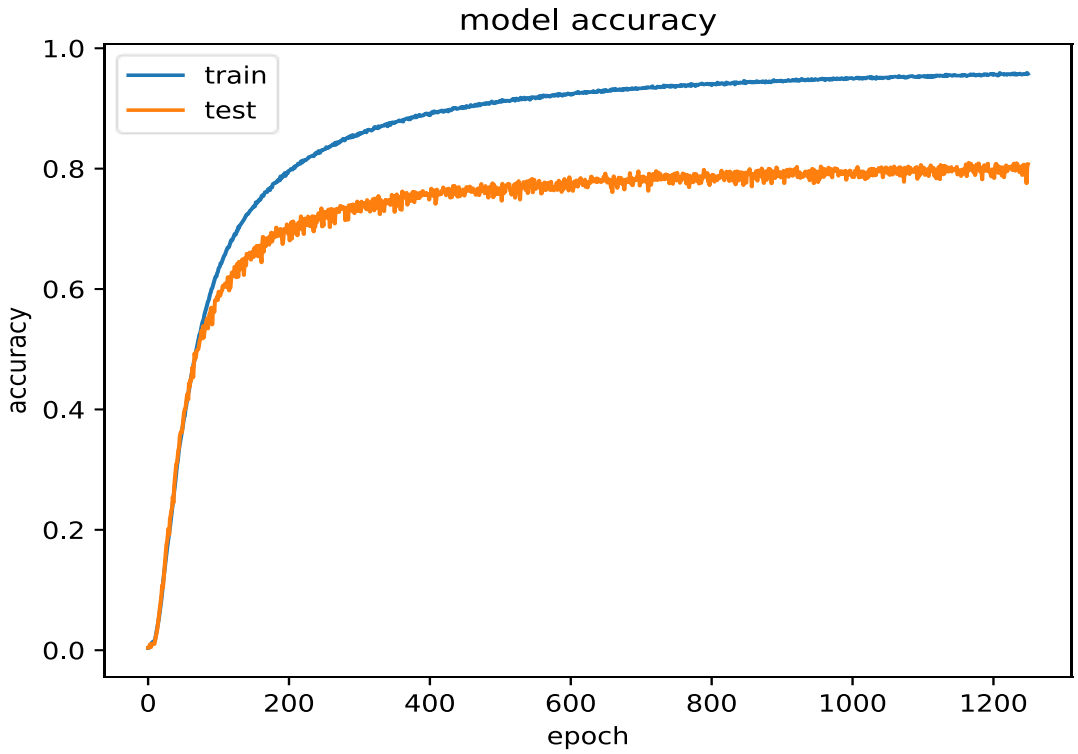


Figure 5: Model Accuracy: Training history over 1000 epochs for the final model: R-CNN Model 4

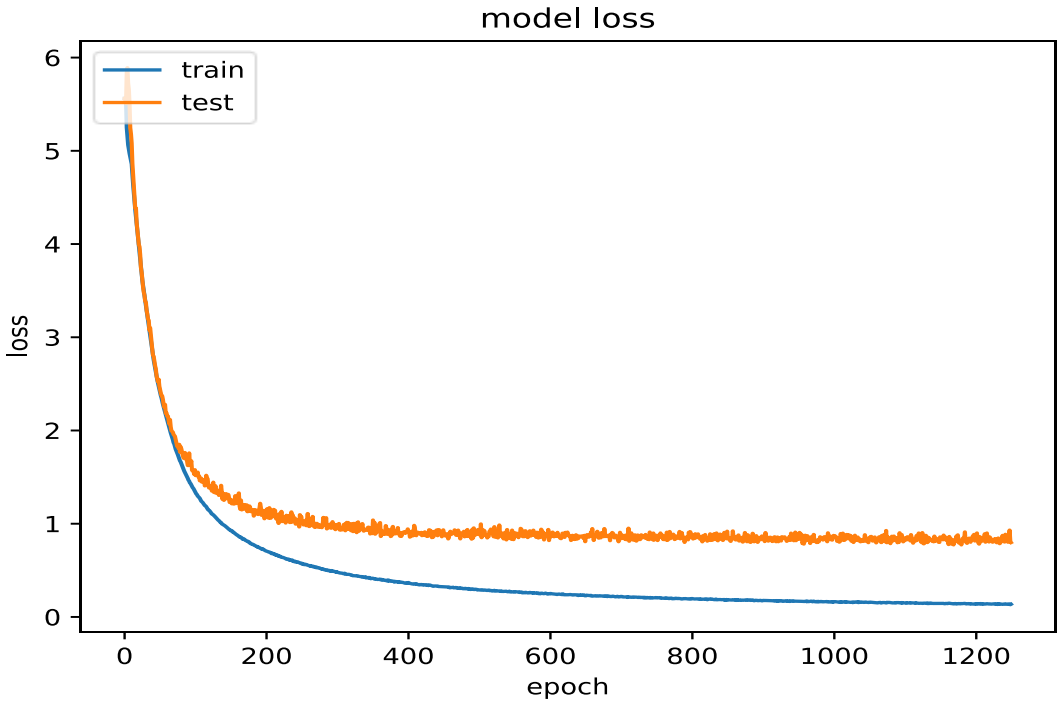


Figure 6: Model Loss: Training history over 1000 epochs for the final model: R-CNN Model 4

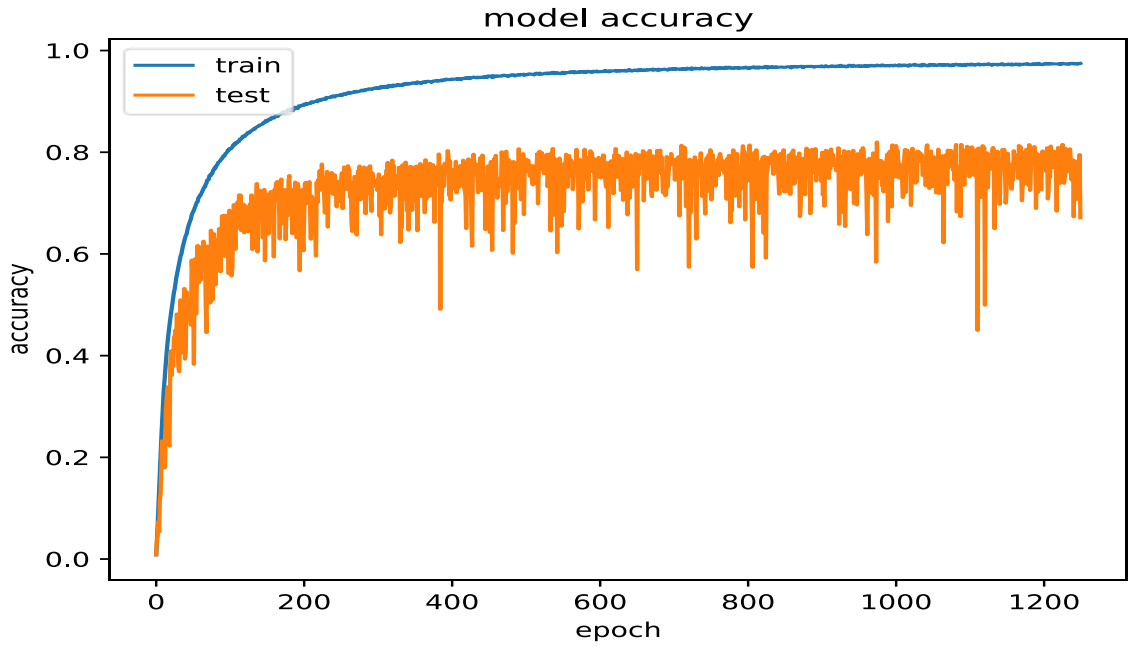


Figure 7: Model Accuracy: Training history over 1000 epochs for the final model: R-CNN Model 4 with Batch Normalization after each Convolution layer

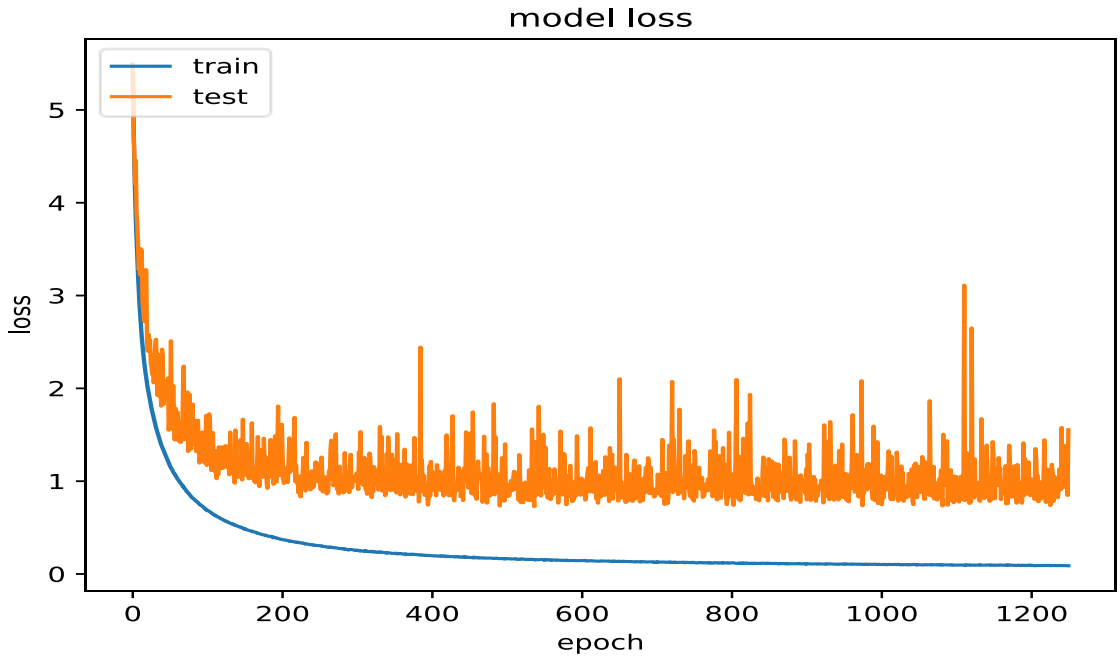


Figure 8: Model Loss: Training history over 1000 epochs for the final model: R-CNN Model 4 with Batch Normalization after each Convolution layer

The training history for the final model, R-CNN Model 4, is shown in Figure 5 and Figure 6. Both the model accuracy and loss traces plateau around 1000 epochs. There is no sign of overfitting.

One additional test was run using the R-CNN Model 4 with batch normalization inserted after each of the convolution layers. There were no significant improvements in the model accuracy or the apparent rate of convergence as can be seen in the training history of plots in Figure 7 and Figure 8. For this reason, we chose the R-CNN Model 4 without batch normalization as the final model for evaluations.

Table 4: R-CNN (2D-CNN + Bidirectional GRU) Diarization (Model Test 4)
Diarization Results Using 260 Class Feature Vector
SWBCell2 Tests of 595 Balanced Recordings, 161,501 superframes diarized

SWBCell2 Test Results, 595 recordings tested		
Change Detection Function	**Total Diarization Error** R-CNN (2D-CNN + Bidirectional GRU)	**Total Diarization Error** 2-D CNN Only
Gaussian Mixture Clustering with Cluster Count Detection	17.3 %	18.3 %
Gaussian Mixture Clustering with Number of Clusters Fixed at 2	12.8%	12.6%

The diarization results in Table 4 are very similar to those for the CNN-only network. Essentially there is a small decrease in error for the case where the number of speakers is not known *a priori*. Otherwise, the addition of the recurrent network layers did not make much of an improvement.

Results

The two networks developed 2D-CNN and R-CNN (2D-CNN + GRU-RNN) were trained using the Switchboard Cellular Part 2 (SWBCell2) data. In addition, the diarization accuracy was measured using SWBCell2 recordings selected such that both speakers' voices are present in at least 37.5% of the recording duration.

In order to get an independent measurement of accuracy, the Switchboard-2 Phase III (SWB-Phase3) dataset was used. This data set contains landline telephone call recordings collected independently of SWBCell2 data set that was used in training.

The SWB-Phase3 dataset contains a total of 2,728 phone calls from 640 participants (292 Male, 348 Female). Of these calls 118 recordings were randomly selected for the evaluation.

The test results are shown in Table 5. For the case where the number of speakers is to be determined by the Gaussian mixture clustering, we see that the 2D-CNN model achieved a lower error by nearly 3 percentage points. When the number of speakers is fixed at the true number of speakers (i.e. two speakers), the results are the same at 15.5%. This suggests that the 2-D CNN embedded speaker embeddings features cluster a little more accurately compared to the features from the R-CNN model. Possibly the distribution of the 2-D CNN clusters has a lower variance or a slightly more Gaussian shape.

Table 5: Accuracy Test Results using the SWBPhase3 data set to compare R-CNN and 2-D CNN Models.

SWBPhase3 Data Test Results 118 recordings tested		
Change Detection Function	**Total Diarization Error** R-CNN (2D-CNN + Bidirectional GRU)	**Total Diarization Error** 2-D CNN Only
Gaussian Mixture Clustering with Cluster Count Detection	22.1 %	19.2 %
Gaussian Mixture Clustering with Number of Clusters Fixed at 2	15.5 %	15.5 %

Additional tests were run using the SWBCell2 data set where the recordings were chosen such that there is one dominant speaker speaking for the majority of the recoding time. These recordings are the so-called "uneven" data set. The goal here was to determine what effect

this has on the diarization error. In particular, how does one speaker talking for the majority of the recording affect the clustering accuracy?

Table 6: SWBCell2 tests with recordings containing one dominant speaker.

SWBCell2 Data Test Results Uneven Data Set: One Dominant Speaker 1279 recordings tested		
Change Detection Function	**Total Diarization Error** R-CNN (2D-CNN + Bidirectional GRU)	**Total Diarization Error** 2-D CNN Only
Gaussian Mixture Clustering with Cluster Count Detection	22.8%	23.2 %
Gaussian Mixture Clustering with Number of Clusters Fixed at 2	17.5 %	17.6 %

Table 7: SWBCell2 tests with near equal amounts of speech from each of the 2 speakers.

SWBCell2 Data Test Results Near Equal Content of Both Speakers 595 balanced recordings tested		
Change Detection Function	**Total Diarization Error** R-CNN (2D-CNN + Bidirectional GRU)	**Total Diarization Error** 2-D CNN Only
Gaussian Mixture Clustering with Cluster Count Detection	17.3 %	18.3 %
Gaussian Mixture Clustering with Number of Clusters Fixed at 2	12.8%	12.6%

The results are shown in Table 6 and Table 7. For the case of one dominant speaker, we see in Table 6 that diarization error is approximately 5 percentage points higher (i.e. worse) compared with the accuracy in Table 7 where the two speakers have about the same amount of speaking time in each recording.

Notice also that the accuracies achieved with the SWBPhase3 data set in Table 5 are not as good as the results with the SWBCell2 in Table 7. A subset of the SWBCell2 recordings were used to train both the 2-D CNN and the R-CNN models. This means that some of the speakers

that were used to train the model as a classifier are appearing in the test recordings. For this reason, the SWBCell2 data set results are not a valid evaluation of accuracy. Therefore, we have only used it to evaluated the change in results comparing the uneven and balanced data subsets.

Conclusion

As mentioned in the introduction, diarization is still an unsolved problem. An error rate of 10% is considered very good. The authors of the paper that this project is based on [Cyrta, 2017], achieved error rates in the range of 13.8% to 19.6% for their best model (R-CNN with CQT). It is not clear if the number of speakers was fixed or estimated in these results.

In comparison, our best results using the SWBPhase3 test-only dataset are 19.2% for the case where the number of speakers is unknown, and 15.5% where the speaker count is known. These results are roughly similar to those obtained in the reference paper [Cyrta, 2017], however, there are some important differences in the features and modelling used in our project. These are listed in Table 8.

*Table 8: Comparison of the Datasets and Features in the work in **[Cyrta,2017]** and the in our project.*

	Reference Paper [Cyrta,2017]	**This Project**
Audio Data	Wideband broadcast audio 16k samples/second	Narrow band telephone/mobile phone audio 8k samples/second
Audio Recordings	Broadcast news and meeting recordings. Tends to have longer segments of speech from one speaker. Few cases of double talk.	Telephone conversations with short turns of speakers. Speech segments for one speaker can be as short as 300ms. Frequent cases of double talk.
Superframe	96 X 96 96 frames Each frame containing 96 bins over a 0 to 16kHz spectrum	64X32 64 frames Each frame has 32 bins over a 0 to 4kHz spectrum
Superframe length	3 seconds	2 seconds

The main difference in the audio data is the reduced bandwidth of the telephone quality audio. The majority of the voice information is below 4kHz, however, there is additional voice content

in the higher frequencies that we are missing and which is likely useful differentiating between speakers.

The second major difference is in the length of the superframe. In [Cyrta,2017], they used a three second superframe. This is very long and not suited to casual conversations where speakers may alternate several times in three seconds. For the speaker classification network, we rely on there only being one speaker present in the superframe. In our development we reduced this to two seconds to reduce the number of double-talk frames.

Clearly, the shorter the superframe is, the harder the task becomes for speaker classification. It may well be however that we could obtain a useful feature vector by using a one second or even half-second super frame. This would make a good follow-on project.

In terms of modelling, 1-D convolution of a 1-dimensional frame followed by recurrent networks and time-distributed networks would be interesting for comparison. Would the results be inferior in this approach? All analytical algorithms create a feature vector base on one spectral frame. This is a 1-D approach and these algorithms do quite well in published results. This is also an interesting avenue for experimenting.

Source Code

Source Code: https://github.com/BFrancisA/Diarize

All development was carried out using:

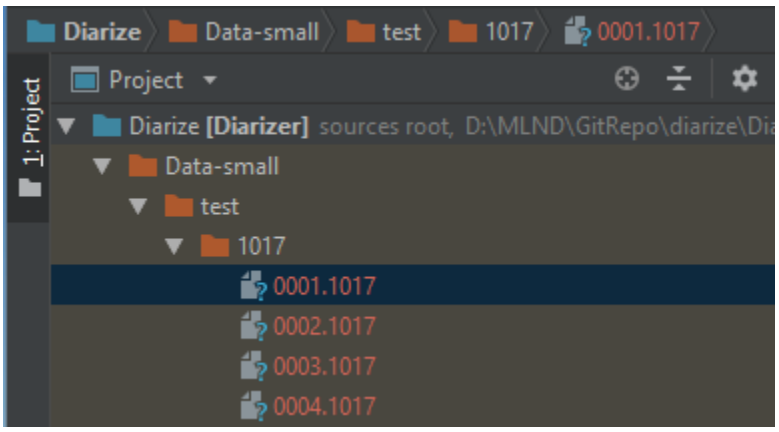
PyCharm 2018.2 (Community Edition)
Build #PC-182.3684.100, built on July 24, 2018
JRE: 1.8.0_152-release-1248-b8 amd64
JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o
Windows 10 10.0

The easiest way to get started with this project is to clone the repository and then load the project into PyCharm. All the dependent libraries will be loaded by PyCharm using Conda.

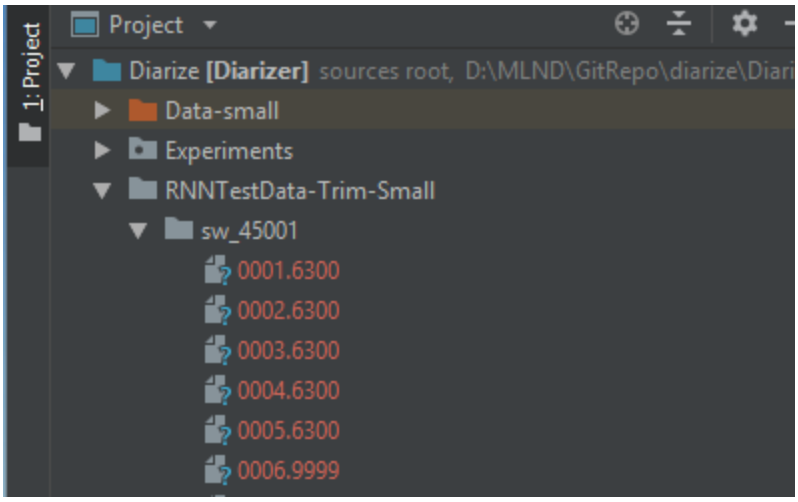
The programs developed to convert the audio data files to mel-frequency cepstrums and build the superframes is prior work developed in C++. This software is not included.

The SWBCell2 and SWBPhase3 datasets are copyrighted and cannot be distributed, however, a small dataset of derived superframe data is included in the Diarize project on github.

Samples of training data are supplied in the directory: **Data-small**. This contains example data of 5 speakers taken from the SWBCell2 dataset. Each file contains one superframe. The file naming convention follows the familiar format of <file number>.label:



Diarization test data is supplied in the directory: **RNNTestData-Trim-Small**. This small dataset contains 5 full length audio recordings converted to superframes. Each superframe file is named using the format <frame number>.<speaker pin number>:



Double-talk frames are assigned a pin of 9999. These are not included in the accuracy scoring.

Training Functions

Train-cnn.py	Used to train the 2-D CNN model.
Train-R-CNN.py	Used to train the R-CNN model.

Diarization Function

Diarize-Model-Clustering.py ： Used to diarize recordings represented as superframes. Produces a diarization accuracy score measuring the percentage of superframes correctly diarized as either speaker 1 or speaker 2 in these two-speaker recordings.

References

[Barras, 2006] Claude Barras, Xuan Zhu, Sylvain Meignier, Jean-Luc Gauvain. Multi-stage speaker diarization of broadcast news. IEEE Transactions on Audio, Speech and Language Processing, Institute of Electrical and Electronics Engineers, 2006, 14 (5) .

[Kenny, 2010] P. Kenny, D. Reynolds and F. Castaldo, "Diarization of Telephone Conversations Using Factor Analysis," in IEEE Journal of Selected Topics in Signal Processing, vol. 4, no. 6, pp. 1059-1070, Dec. 2010. doi: 10.1109/JSTSP.2010.2081790

The Reference Paper for this Project

[Cyrta, 2017] Pawel Cyrta, Tomasz Trzciński, Wojciech Stokowiec. Speaker Diarization using Deep Recurrent Convolutional Neural Networks for Speaker Embeddings. arXiv:1708.02840v2 [cs.SD] 10.1007/978-3-319-67220-5_10, (https://arxiv.org/abs/1708.02840)

[Zuo, 2015] Z. Zuo, B. Shuai, G. Wang, X. Liu, X. Wang, B. Wang, and Y. Chen, “Convolutional recurrent neural networks: Learning spatial dependencies for image representation,” in CVPR, 2015

[Cakir, 2017] E. Cakir, S. Adavanne, G. Parascandolo, K. Drossos, and T. Virtanen, “Convolutional recurrent neural networks for bird audio detection,” in ICASSP, 2017 (http://www.cs.tut.fi/~cakir/publications/convolutional-recurrent-bird-eusipco2017.pdf)

[Adavanne, 2017] S. Adavanne, D. Konstantinos, E. Cakir, and T. Virtanen, “Stacked convolutional and recurrent neural networks for bird audio detection,” in European Signal Processing Conference (EUSIPCO), 2017.