**REPORT**

1. **Audio Preprocessing**
2. To deal with the raw data, we build Audio class, containing various functions for audio processing, where the audio is extracted using the read method, SAMPLE\_RATE is set to 16000, and the trim\_silence method removes silence at the beginning and end of a sample. use the FBank method provided by python\_speech\_features for MFCC feature extraction for the input of the model. The process of MFCC feature extraction is as follows：

Pre-emphasis: amplify the high frequence.

Frame blocking and windowing: split the signal into short-time frames.

Fourier-Transform and Power Spectrum.

Filter Banks.

Mel-frequency Cepstral Coefficients(MFCC): discrete cosine transform.

Mean normalization: MFCC(mean and var).

1. In the stage of data augmentation, we mix the audio from the noise dataset and the train dataset to generate a new training dataset with noise.
2. Decompose the training dataset into training and test samples, and create the KerasFormatConverter class to convert the audio format data into numpy array format, where NUM\_FRAMES indicates the number of frames per second panning in frame blocking, set to 160.
3. Save the converted file to the corresponding folder.
4. **Model Description**

This neural network architecture is mainly based on the idea of ResNet, which makes it easier to train deep networks，the CNN is composed of a number of stacked residual blocks (ResBlocks), Figure 1 illustrates the architecture of resBlock. A convolution block Conv 3×3 is parameterized by the filter size 3×3, the zero padding 1 in both directions and the consecutive striding 1×1. There is also a BatchNorm between each convolution and activation function.



Figure 1: Detailed view of ResBlock.

Table 1 shows the details of the proposed neural network architecture. The second column in Table 1 is mainly for each convolution, indicating the parameters of the corresponding module. Each layer of convolution doubles the number of channels, but halves the dimensionality of the features, thus keeping the fourth column dim unchanged, the last column indicates the number of parameters of each module. “Average” denotes the temporal pooling layer and “ln” denotes the length normalization layer.

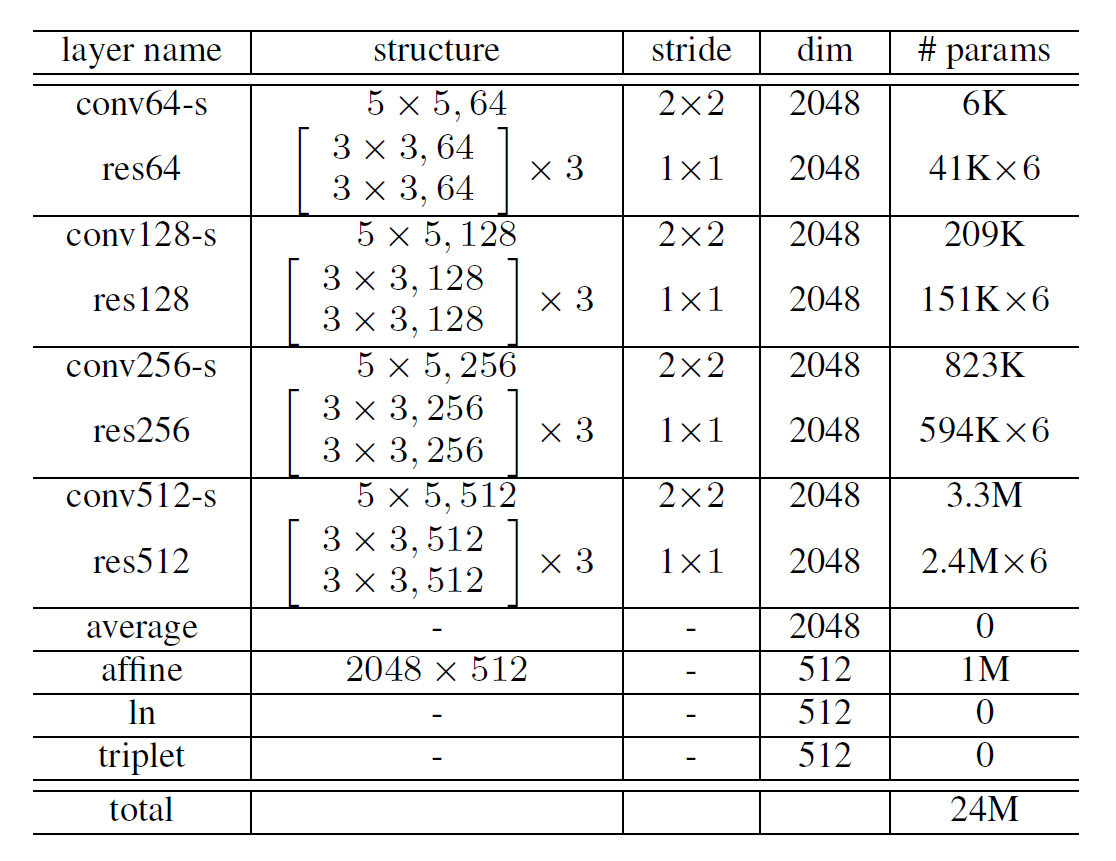


Table 1: Architecture of ResCNN

The output of this model is a 512-dimensional embedding, then the layer activation h is computed as follows:

where, T is the number of frames in the utterance. The cosine similarity can be calculated through length normalization.

This cosine similarity will be used to calculate triplet loss.

1. **Experimental Details**

**3.1 Loss Fuction**

After the model is constructed, the training of the model is started. First, we use triplets loss as the loss function. Triplet loss has three inputs samples, an anchor (an utterance from a specific speaker), a positive example (another utterance from the same speaker), and a negative example (an utterance from another speaker). These tree inputs are speaker embedding. The cosine similarity between the anchor and the positive example is larger than the cosine similarity between the anchor and the negative example. The condition is as follows:

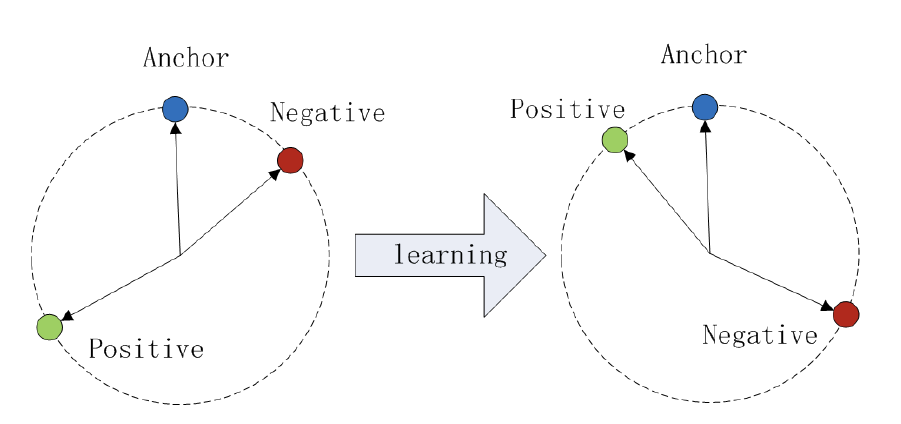


Figure 2: The Triplet loss in cosine similarity.

According to this constraint, the cost function for N triplets can be written as

**3.2 Train the model**

To avoid suboptimal local minima early-on in training, we use a softmax and cross entropy loss to pre-train the model. The main approach is contained in fit\_model\_softmax method. In this method, max\_epochs is set to 1000 and batch size is set to 32\*3 which is a multiple of 3. To make sure the training process will perform well, we set EarlyStopping and ReduceLROnPlateau. If the accuracy does not increase by 0.1% over 20 epochs, we stop the training and if the accuracy does not increase over 10 epochs, we reduce the learning rate by half. The initial learning rate and the learning rate in adam optimizer are consistent. The batch input shape include two hyperparameters, they are NUM\_FRAME and NUM\_FBANKS. NUM\_FRAME is mentioned before, which is set to 160. NUM\_FBANKS is set to 64 .To start with softmax pre-training, we use KerasFormatConverter class which is mentioned in the stage of Audio Preprocessing to generate train data and test data as the input of this model. The pretraining weight will be saved in 'checkpoints-softmax'.

After finishing the softmax pre-training, we continue to use triplet loss as the loss function to train model and save the weight to 'checkpoints-triplets'. The epoch in this phase is also set to 1000 .The fit\_model method in this training process is different from the pre-training process. In fit\_model method, we need to get the batcher for training, the barcher belongs to LazyTripletBatcher class, this class is used to get a batch samples, and it mainly calls get\_random\_batch method to randomly extract anchor utterances, positive utterances, negative utterances from the dataset. Then the MFCC feature values of these three types of samples will be the input of model. From the experimental results, softmax pre-training does help to converge faster.

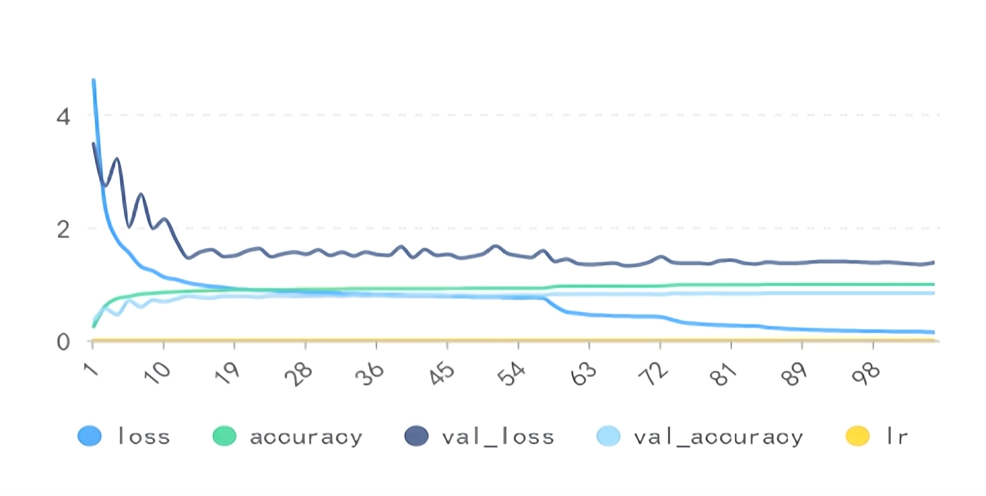


Figure 3: Pretrainlog

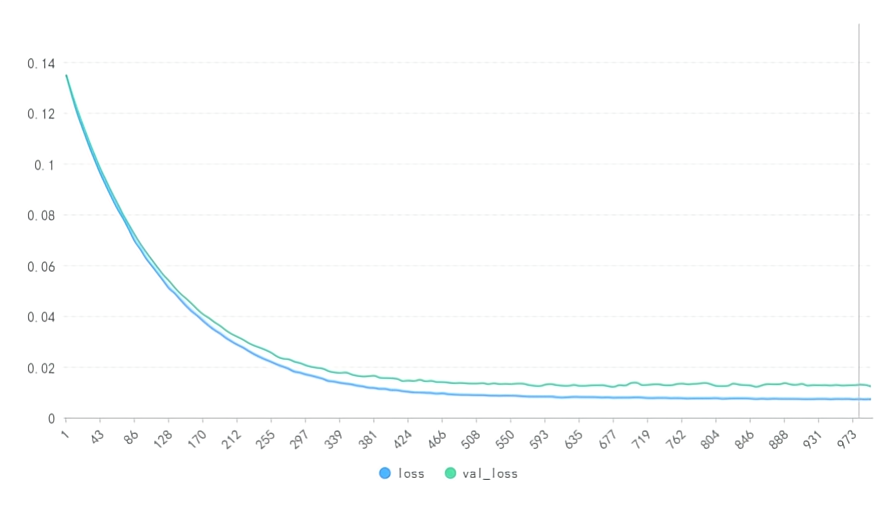


Figure 4: Trainlog

**3.3 Evaluation and Test**

We set four evaluation metrics to evaluate the performance of model, they are f-measure, true positive rate, accuracy and equal error rate.

In pre-train phase, the performances of models rapidly improves as the epoch increases. In train phase, the performance of the model is improving at a slower rate.

At last, we choose the model with epoch 401 for prediction, which has the best f-meature 0.6875 and accuracy 0.9948.

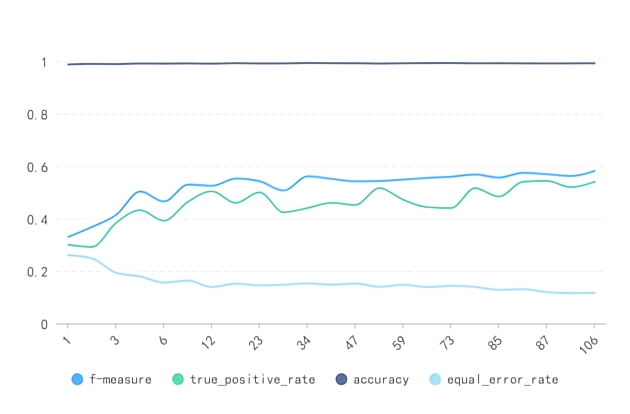


Figure 5: Pretrain model evaluation

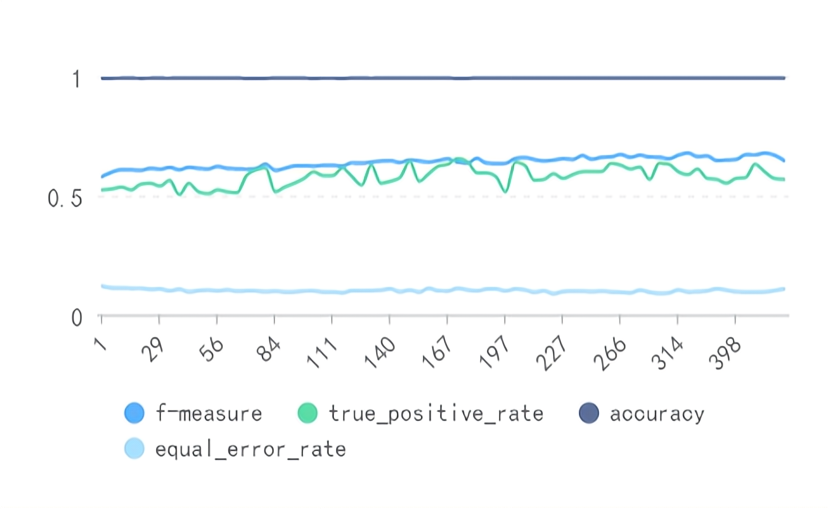


Figure 6: Train model evaluation

1. **Member Contribution**

**Liang Shuyun**:

1. Write part of report:
2. Evaluation and Test.
3. Code:
4. Data preprocessing module:

Audio\_preprocessing.py, audio.py, Mix\_train\_set\_and\_noise.py

1. Batch sample module:

batcher.py, Build\_keras\_inputs.py

1. Model module:

conv\_models.py, train.py

1. Evaluation module:

eval\_metrics.py, test.py, eval\_models.py

1. Util module:

Constant.py, triplet\_loss.py, utils.py

1. Predict module:

predict.py