A Convolutional Model for Tone Classification

FinalProject of MS318, Fall 2016

XUEYUAN ZHAO, HAOMING LU, YUNQI LI

January 15, 2017

Abstract

To improve the tone classification on Mandarin Chinese individual characters, we proposed a model with data processing and convolutional model which focuses on reducing the noise in the dataset. Given a specific character, this approach excludes noise frames by both frequency and energy information, regenerates the frequency sequence by spline fitting, captures the feature and finally acts the classification with a convolutional neural network (CNN). We evaluated our model on the provided datasets *test* and *test_new*, achieved the accuracy of $\mathbf{100}\%$ and $\mathbf{98.25}\%$ respectively within a short period of time.

1 Introduction

Chinese language is a tonal language while the tonal system of Mandarin Chinese has four lexical tone patterns including (1) flat and high, (2) rising, (3) low and dipping, and (4) falling, which can be reflected through the fundamental frequency(F0) variation over time.

Some previous works on this problem tend to extract feature vector of the data and then deal with it by support vector machine (SVM)[6] or some even more direct apporaches. However, considering the course mostly focuses on deep learning, we decided to keep the preprocessed input sequence and construct a neural network to finish the rest of the work, which turned out to be an effective idea. We will introduce how we preprocess the input data in **Chapter 2**, the network model in **Chapter 3**. The experiment results and comparation among different toolkits will be presented in **Chapter 4**. And **Chapter 5** will be a summary to our work.

2 Data Processing

After reading the dataset and referring some previous works, we noticed that most mistakes in this problem are caused by environmental noise and pronunciation differences among people. In order to deal with these disturbance, we executed *invalid frame exclusion*, *difference normalization* and *spline fitting* successively. The network designed for this way of processing turned out to perform best among all the ways we had tried. We will then discuss the three parts in detail.

2.1 Invalid Frame Exclusion

Both the environmental noise and pronunciation patterns lead to invalid frames. From the frequency input of training dataset, we can see that each sequence has a series of zero frames in the front and in the tail. It is not hard to see that these frames are useless in distinguishing tones, so we set a constant $threshold_{f0}$, all the frames where the value is less than $threshold_{f0}$ will be ignored. And from energy input of training dataset we discovered that, the values of some frames are extremely small which indicates that the corresponding frames in the frequency sequence also make no sense. These frames are also excluded according to another constant $threshold_{en}$. In the end, considering that the pronouncing process may not be stable in the beginning or in the end, we exclude a few frames at both ends.

The whole process can be implemented with the following Python code:

```
def exclude_invalid_frame(f0, en):
  f0 = f0[numpy.where(en < en_threshold)[0]]
  f0 = f0[numpy.where(f0 < f0_threshold)[0]]
  f0 = f0[cut_length:-cut_length]
  return f0
```

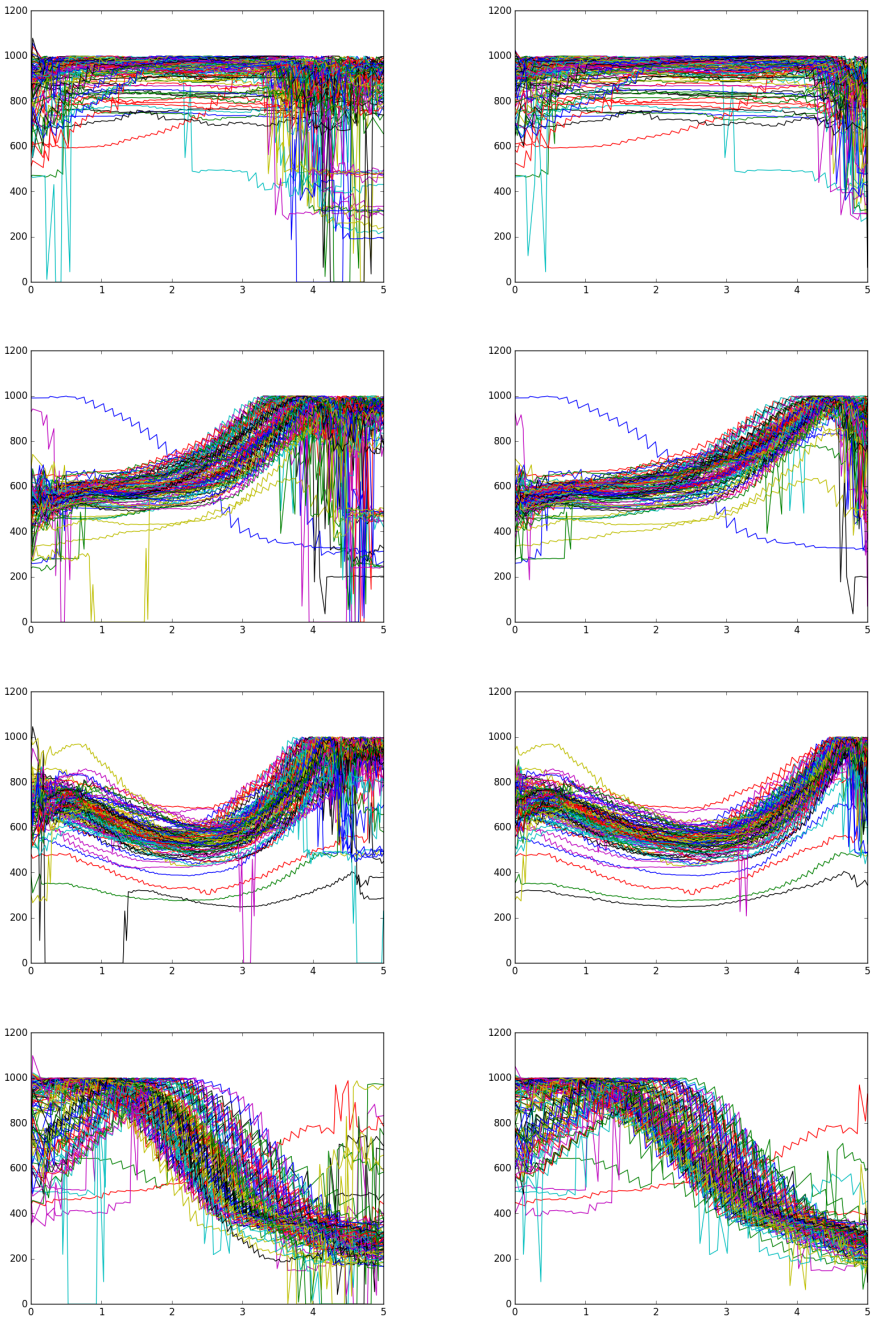

Figure 1: The comparation between original frequency sequences(left) and excluded sequences(right). The data are captured from training dataset.

2.2 Difference Normalization

After observing the input fundamental frequencies of the training dataset (as shown in Figure 1), we found that, most lines with the same label already share similar patterns except some variance in the y direction. So we decided to do the first order difference along the x-axis as

$$f0_{dif}(i) = f0(i+1) - f0(i)$$

We then noticed that most of the difference sequences with the same label have similar tendencies over time. However, the distribution of actual numerical values of first order differences has high variance which might have a negative influence on the performance of network models. In order to reduce the variance of first order difference sequences, we set a threshold $bound$ and replaced all the frames where the absolute value is beyond the threshold with it. This method can be described as

$$f0_{norm\_dif}(i) = \begin{cases} bound, & f0_{dif}(i) > bound \\ -bound, & f0_{dif}(i) < -bound \\ f0_{dif}(i), & otherwise \end{cases} \tag{1}$$

Figure 2 shows the comparison between original difference and the differences after normalization.

2.3 Spline Fitting

The replacement above indeed excluded abnormal frames, but it broke the continuity of the sequence in some ways. In order to reconstruct the continuity and eventually smooth the frequency sequences, we applied interpolations here to regenerate the sequence. After attempting different kinds of interpolations including polynomial interpolation, Hermite interpolation and spline interpolation, cubic spline interpolation, as defined below, turned out to have the best performance among them in testing.

Definition 1 *The cubic spline interpolation of n points (x_i, y_i) is defined as n cubic polynomials $p_i(x)$, where*

$$p_i(x_i) = y_i$$
$$p_i(x_{i+1}) = y_{i+1}$$
$$p_i'(x_{i+1}) = p_{i+1}'(x_{i+1})$$
$$p_i''(x_{i+1}) = p_{i+1}''(x_{i+1})$$

More narrowly, we first set all the values uniformly on a interval with fixed length, for example $[0, 100]$, and used their new coordinates as the input data. After excuting interpolation, we then resampled on a fixed numer of positions in this intervel, for example, all the integer points, now all the frequency sequences will have the same length.

The Figure 3 following gives comparation between the original frequency sequences and frequency sequences recovered from the preprocessed differences. The left column came from the original data while the right column are recovered from the preprocessed difference, with the start value fixed to zero. We can see that features of different labels became much more distinct after data processing.

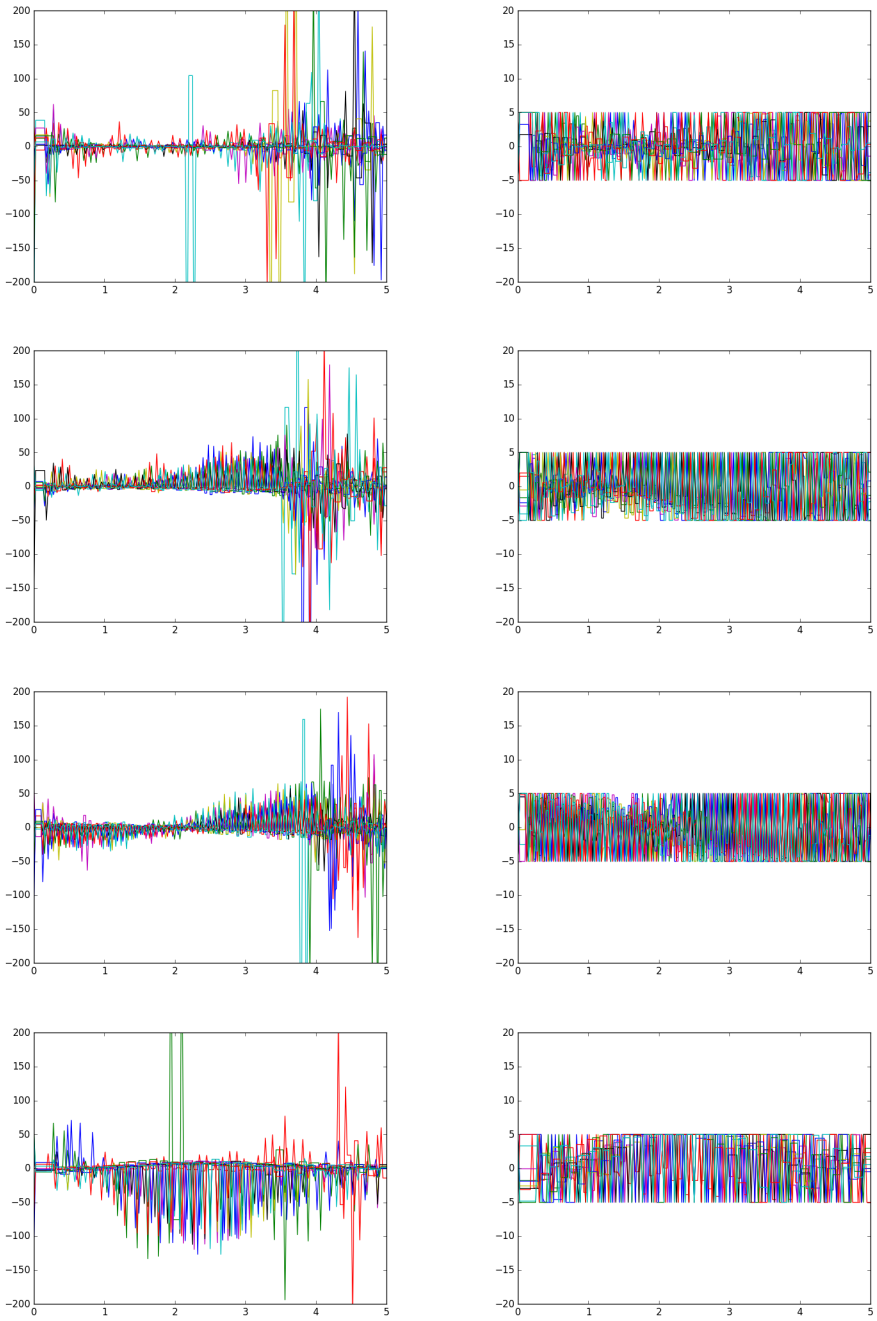

Figure 2: Comparation between original differences and normalized differences. 10 examples per label are picked. Notice that the axis ranges on the left side are ten times those on the right side.

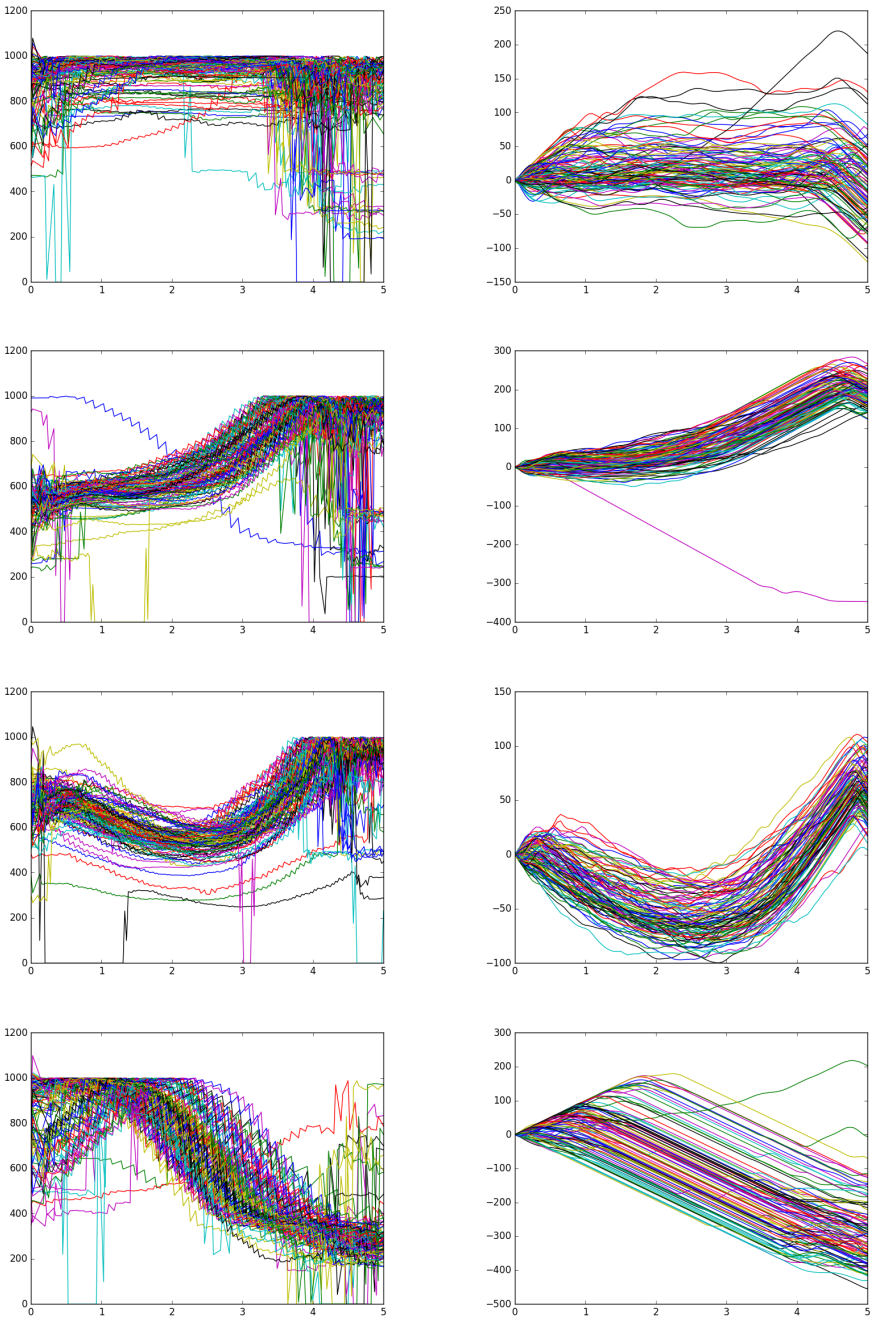

Figure 3: Comparation between the original and preprocessed frequency sequences on training dataset. Notice that the subfigures in the right column are suitably scaled to make the tendency more evident.

3 Network Model

We implemented several neural network models on this problem. Our first attempt with deep neural network (DNN) including two fully-connected hidden layers achieved 100% accuracy on dataset *test*, but it turned out to perform poor on dataset *test_new*. After further experiments with recurrent neural networks (RNN)[4] and convolutional neural networks (CNN), we found that CNN performed better no matter from the consideration on time or on accuracy, and eventually proposed a convolutional net model with 8 hidden layers. Figure 4 desribes the architecture of the network.

The input of the network is a 1-D vector of length 100, preprocessed with $threshold_{f0} = 50.0$., $threshold_{en} = 50.0$, $cut\_len = 3$ and $bound = 5.0$. The kernel size of convolutional layers and max-pooling layers are $[10,]$ and $[4,]$ respectively. The performance of the model increases until 3 Conv-Pool blocks are piled up. The convolutional layers are initialized with uniform distribution on $[-0.97, 0.97]$ and padded with zero. Two fully connected layers each with 2048 units and use $sigmoid$ for activation are foillowing.

We trained the model with 10 batchsize Stochastic Gradient Descent (SGD)[5] and learning rate 0.005. The model converges after about 60 epochs. All the parameters were adjusted according to the architecture and input data, some processes of adjustment will be introduced in the next section.

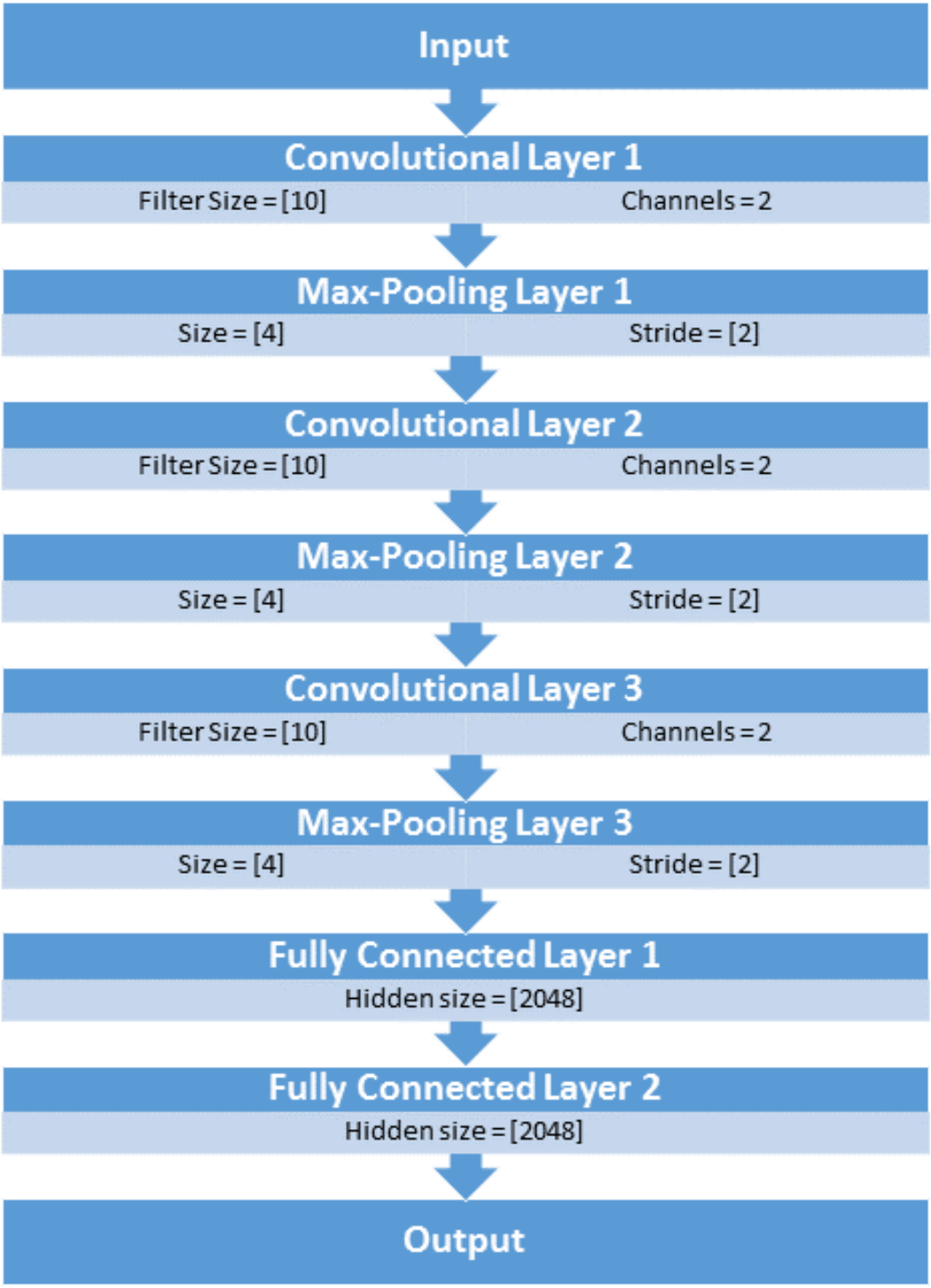

Figure 4: The Architecture of Network Model

4 Experiments

In this section, we present our experiments in details, including processing, experimental setup, model comparison, toolkit comparison, and corresponding analysis. Since almost all of our models can reach the accuracy of more than **99**% on the easy data set *test*, we will only discuss the performances on the hard data set *test_new*. In our experiments, our model reached the accuracy of **98.25**% on *test_new*.

4.1 Datasets

The dataset consists of three parts: *train*, *test* and *test_new*. The *train* and *test* are the voices of standard Mandarin recorded in the quite environment. The *text_new* are the voices of Mandarin (may have accent) recorded in the normal office environment.

4.2 Evaluation Metrics

The main evaluation metric in our experiment is classification accuracy, which is the ratio of correct classification predictions :

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100\%$$

4.3 Comparison over Data Process Methods

In our experiments, we tested the three data process methods mentioned in **Chapter 2**. In this section, we call the corresponding three methods *Exclude*, *Normalize* and *Fit*. The relationship between accuracy and data process methods are shown in Figure 5.

4.4 Comparison over Model Architectures

In this section, we discuss the impact of model architecture. For our convolutional models, we take three hyperparameters into consideration:

1. *Width of Network*: We explore the impact of network width by adjusting the width of fully connected layers in our model. We compared different widths of fully connected layers: 2^8, 2^9, 2^{10}, 2^{11}, 2^{12} and 2^{13}. Figure 6 (Left) shows the performance as network width grows. Generally speaking, the networks with fully connected layers of width 2^{11} works better generalization on the test set.
2. *Network Depth*: We explore the impact of network depth by adjusting the depth of fully connected layers in our model. We compared different depths

of fully connected layers: 1, 2, 3 and 4. Figure 6 (Right) shows the performance as network depth grows. Generally speaking, the networks with fully connected layers of depth 2 works better generalization on the test set.

3. *Activation Functions*: In our experiments, we compared three mainstream activation functions: $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$, $\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$ and $\text{relu}(x) = \max(0, x)$. Figure 7 (right) compares these activation functions on our model. From this figure we find that sigmoid has better performance than tanh and relu.

4. *Number of Convolution Filters*: In our model, there are 3 convolutional layers, we explore the impact of different number of convolutional filters by comparing following parameters (x-y-z means the first convolutional layer has x filters, the second convolutional layer has y filters, the third convolutional layer has z filters): 1-2-4, 2-4-8, 4-8-16 and 8-16-32. The results are shown in the Figure 7 (left). The results show that the set of parameters 2-4-8 outperforms other sets.

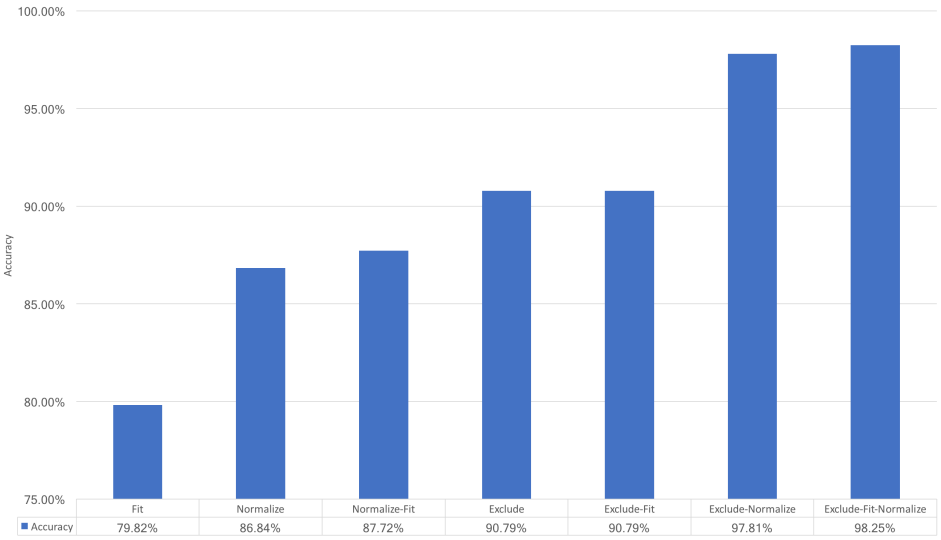

Figure 5: Accuracy Comparison over Different Data Process Methods

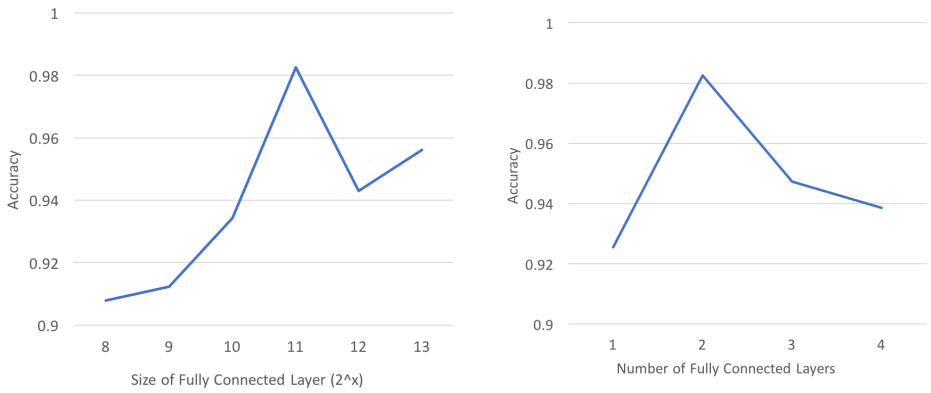

Figure 6: Accuracy Comparison over Different Network Widths(Left); Accuracy Accuracy Comparison over Different Network Depths(Right)

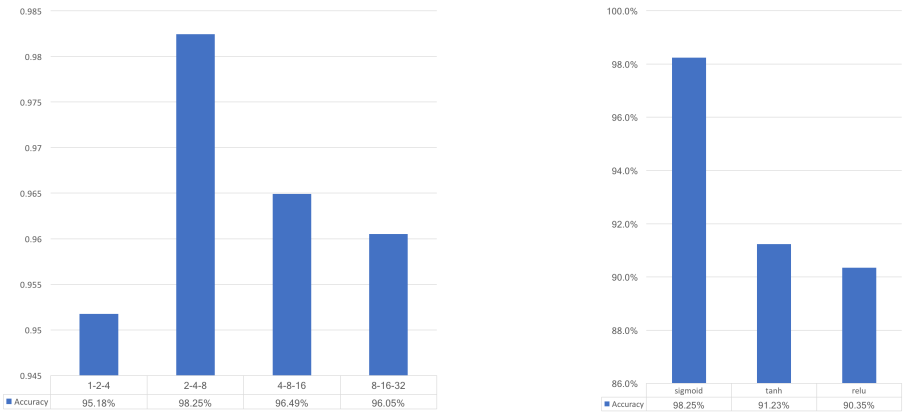

Figure 7: Accuracy Comparison over Different Number of Filters(Left); Accuracy Comparison over Different Activation Functions(Right);

4.5 Compare over Different Deep Learning Toolkits

In our experiments, we compared the efficiency, accuracy and stability on GPU computation of three mainstream deep learning toolkits: CNTK[3] , mxnet[1] and torch[2] .

1. *Efficiency*: In our experiments, we implemented our model in three toolkits and evaluated the training time of 60 epochs with mini-bash size of 10. Figure 8 shows the comparison of training time over three toolkits. Table 1 shows the comparison of GPU utilization over three toolkits. From the figure we find that torch and mxnet are significantly more efficient than CNTK, and torch is a little better than mxnet. Possible reasons include:

   (a) Torch has a significantly better ability to make full use of GPU.

(b) mxnet has better algorithms so can reach a better efficiency with less GPU resources.

2. *Accuracy*: We evaluate the same model implemented in three toolkits and compared their accuracy. Figure 8 shows the comparison of accuracy over three toolkits and the relationship between accuracy and efficiency. From the figure we find that CNTK has a better performance than mxnet and torch. We discover that the frame work with higher efficiency tends to have a lower accuracy, the possible reason is that the different toolkits have different balances of efficiency and accuracy.
3. *Stability*: During our experiments, we find that CNTK and mxnet has better numerical stability than torch. For the same toolkit, the numerical stability on CPU is better than that on GPU.

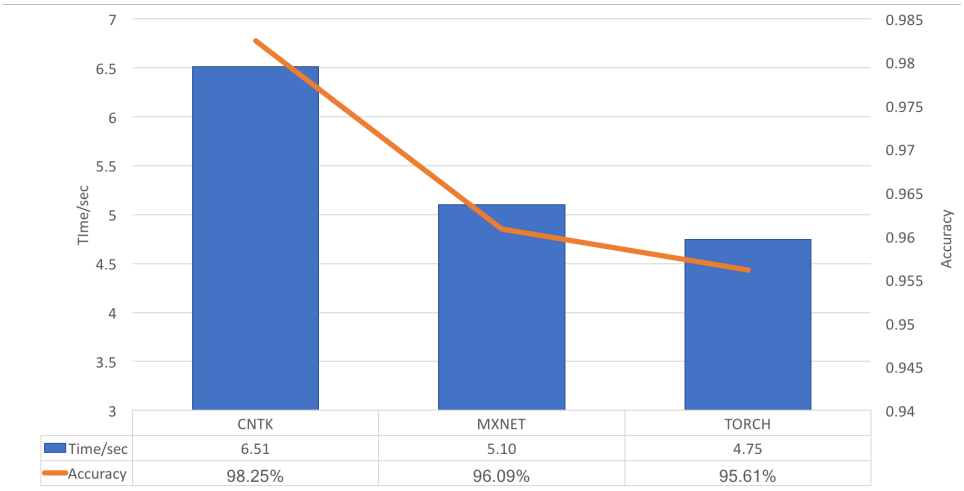

Figure 8: Comparison of Training Time and Accuracy over Different Toolkits

	CNTK	mxnet	torch
GPU-Util	∼67%	∼57%	∼80%

Table 1: GPU Utilization Comparison over Different Toolkits

5 Conclusion

In this work, we proposed a CNN-based model for Chinese Mandarin tone classification and a series of methods to process input data. We chose the classification accuracy as our main evaluation metric and explored the impacts of different data processing methods, network architectures and deep learning toolkits. Furthermore, we explored the performances of three mainstream deep learning toolkits: CNTK, mxnet and torch in the aspect of efficiency, accuracy, stability and availability. [**?**]

References

1. Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

2. Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

3. William J Dally. Cntk: An embedded language for circuit description, dept. of computer science, california institute of technology, display file.

4. Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, pages 347–352. IEEE, 1996.

5. Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

6. Shui-ping Wang, Zhen-ming Tang, Ying-nan Zhao, and Sai Ji. Tone recognition of continuous mandarin speech based on binary-class svms. In *2009 First International Conference on Information Science and Engineering*, pages 710–713. IEEE, 2009.