KEYBOARD SOUND RECOGNITION

Carlos Arenas Gallego *Daniel Muñoz Puerta* *Oriol Vila Clarà*

Universitat Politècnica de Catalunya, Barcelona, Spain

ABSTRACT

The present project aims to perform a research study about two different approaches used for keyboard sound recognition, specifically, convolutional neural networks (CNN) based on spectrogram features versus other special-purpose designs methods based on Mel frequency cepstral coefficients features (MFCC).

To do this, it is intended to compare and contrast the results obtained by these two methods with a SVM classifier, and finally, perform an unsupervised learning (clustering), to be able to predict written text through the sound of a keyboard with high accuracy. Thanks to this analysis, we can know the strengths and weaknesses of the proposed methods.

1. INTRODUCTION

Over the years, people rely more on the Internet use, making that their identity and personal data are overexposed to other users.

Nowadays, we can do almost everything with our smartphone or computer, such as managing our bank account, buying online, sharing data in the cloud or getting in touch through social networks. Therefore, it is increasingly difficult to protect the security and privacy of users from intrusive attacks. Although cyber-security field has evolved over the last years, hackers always go one step further.

The last progress in the field of pattern recognition, both in image and sound, can be beneficial or harmful to society, depending on the use made of it. Tus, the main reason for this project is to check whether it is possible to guess written text through the sound of a keyboard. This will allow us to know the vulnerability of the system against intruders.

The complete source code as well as the whole dataset is given in the following link:

GitHub repository

1.1 Related Work

Several authors [1] [2] have focused on this topic, using different feature extraction techniques, as well as classification algorithms as we can see in figure 1.

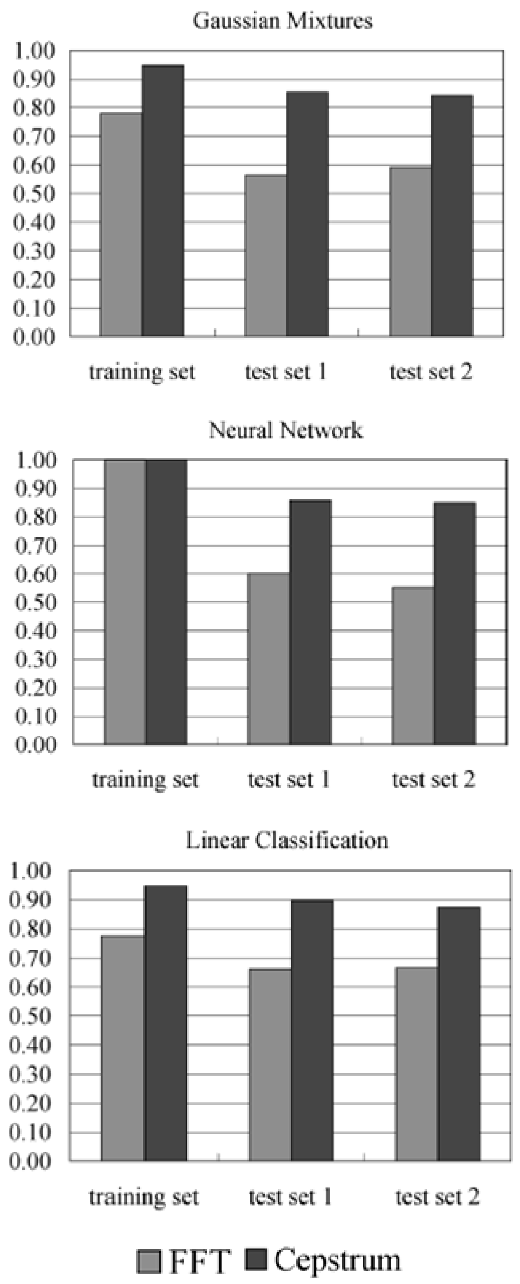


Fig. 1 Recognition rates using three different classification methods on the same sets of FFT and cepstrum features.

In these graphs, we can see a comparative in which the cepstrum and FFT features are trained on a Gaussian Mixture, Neural Network (NN) and a Linear Classifier. In all of them we see how cepstrum features gets a better correct rate against FFT. However, knowing which is the best classifier is not so obvious.

The training set performance is not very representative since it does not show a clear idea of the method generalization. When we have a very complex model (high capacity) we are in overfitting, which means that the model fits extremely well to a particular training set (low bias), although it is not robust against the change of the training set (high variance) as with NN. On the other hand, when the model is too simple, we are in underfitting (high bias and low variance). Therefore, we focus on seeing which model have a better performance on test sets. As we see, the best of them is the linear classifier.

Unlike the present project, the authors of the work "Keyboard Acoustic Emanations Revisited" [2] has been helped using language models to improve their results.

1.2 Contributions

With this study, we want to reach several objectives that allow us to approach the keyboard sound recognition topic, creating an algorithm that try to offer better results than previous studies.

Specifically, we start generating a reliable dataset under controlled conditions, which supports the experiments that will be carried out later on (section 2). Next, we analyse the audio signals from our dataset to extract the frames that present more information (section 3). With our segmented dataset, we extract features using both MFCC and CNN methods to reliably identify each sound (section 4 and 5). Later, we compare the effectiveness of both methods by classifying the extracted information with SVM (section 6). Once this is done, we perform an unsupervised learning on the extracted features to map sounds to keys (section 7). Finally, we analyse our results and show some conclusions (section 8 and 9).

2. DATASET GENERATION

Working with a solid database is essential to maximize the possibility of achieving good results at the end of the tests. So, we have intended to generate our dataset using the best conditions available.

All the recordings were done in an isolated room to minimize the impact of external noises, obtaining the clearest possible signals. The recording chain is represented in figure 2.

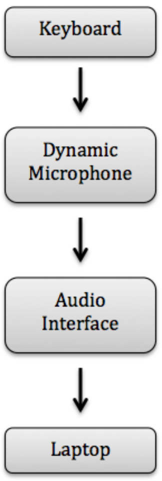


Fig. 2 Dataset Recording Chain.

The first step was selecting the best keyboard in order to record the keystrokes. After testing different types of keyboards, we decided to use an external Logitech K120 keyboard, since the loudness of the keys was greater than in the other cases. Then, we used a dynamic Shure SM58 microphone located at the center of the keyboard at a height of about 10 centimeters over the keyboard. The microphone was connected to an external Focusrite Scarlett 2i2 audio interface and then, the interface was connected to a Macbook Pro via USB. The dataset files were recorded at 44.1 KHz and 16 bits, exported into WAV files using Logic Pro recording software.

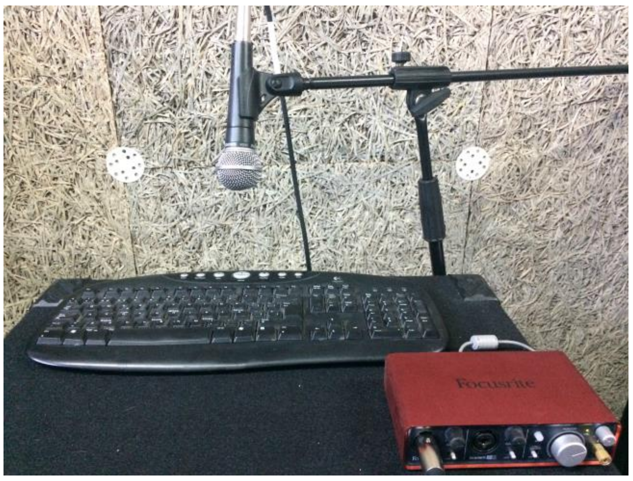


Fig.3 Working environment used for the dataset generation..

Having the recording chain ready to use, an independent dataset was generated for each method. The first dataset, used with the MFCC method, is composed of 100 keystrokes for 7 different keys (A, C, M, O, R, Enter, Space) and 15 repetitions of 3 different words including previously recorded letters (CAMA, MARCO, ROMA).

For the CNN method, we recorded around 1000 keystrokes for two different letters (A, M) to have enough data for training and testing the system, obtaining exactly 2103 samples distributed on this two different classes.

The recorded keys were selected taking into account the distance between the letters and the shape of the keys, trying to maximize differences in the sound of the keystrokes.

3. SEGMENTATION

Since each audio file obtained from the dataset generation contains all the keystrokes of a particular key, the main task of the segmenter is to generate independent audio files for each keystroke. To do this, the function must detect the instant in which each keystroke occurs and once it is detected, extract only the relevant information. In order to accomplish this task, we must first analyse the audio signal produced by a key when is pressed.

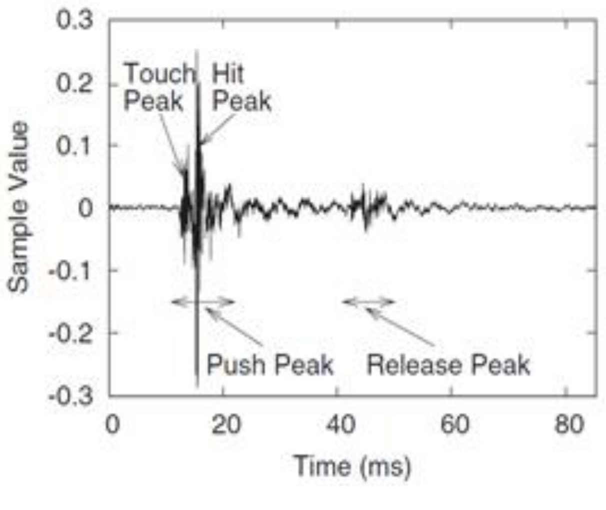


Fig.4 The audio signal of a keystroke.

In figure 4 we see that the keystroke duration is about 100 ms. This duration depends on the speed at which the user writes. In our case, the typical duration is greater to avoid problems when we record and segment. In each keystroke we can differentiate two essential parts: push peak and release peak.

Push peak, in turn, consists of the touch peak (moment when the finger touches the key) and the hit peak (when the key hits the supporting plate). In our dataset, the push duration is about 40ms.

Release peak occurs after the push peak, and is the sound generated when the key returns to its original position. In our experiments we consider that the relevant information is implicit in the push peak, so the release peak is omitted.

segmentation.m function starts by loading the audio file of the key that we want to segment. Later, it checks sample by sample the energy level of the signal, so that when it exceeds a 0.01 threshold, a hit peak is detected. When the hit peak is detected, the push peak is extracted with the aid of a 40ms window, delayed 150 samples to contain the touch peak. Then, we advanced the search pointer with a 250ms window to avoid detecting a release peak as a hit peak, in addition to speeding up the process. Finally, it generates an independent .WAV audio file for each 40ms push peak extracted.

4. FEATURE EXTRACTION WITH MFCC METHOD

Feature extraction offers very good results in the frequency domain due to our audio signals are of short duration (40ms). Cepstrum features are widely used in speech analysis and recognition. In order to represent the previously segmented audio files using the minimum number of characteristics, without losing relevant information, we use the Mel frequency cepstral coefficients (MFCC).

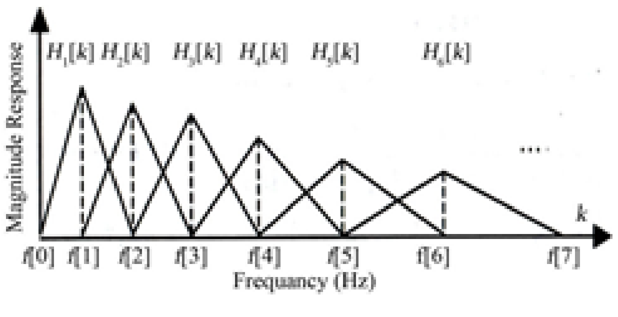


Fig. 5 Energy calculation in triangular bands.

MFCC adopts the human ear behaviour thanks to the use of the Mel scale, in which all frequency bands are determined by an overlapped band-pass filter bank with a logarithmic frequency spacing (decibels) as shown in figure 5. The filter bandwidth increases as the frequency increases. This technique consists of the following steps:

1. Calculate the DFT to the input signal.
2. Define a bank of M triangular filters.
3. Define f_{low} and f_{high} (in Hz) to determine the frequency range of the filter bank, as well as the sampling frequency F (in Hz).
4. Compute the logarithm to the output signal after applying each filter.
5. Obtain the C cepstral coefficients after computing the discrete cosine transform (DCT).

To make the system more robust and avoid information loss, in each 40ms audio signal we calculate the cepstral coefficients in frames of 10ms with a displacement of 2.5ms. This gives 14 frames/push peak, to which we calculate their cepstral coefficients.

We defined the following parameters for the MFCC operations:

- Frame duration: Tw = 10ms
- Frame shift: Ts = 2.5ms
- Number of filter bank channels: M = 32
- Number of cepstral coefficients: C = 13
- Lower frequency limit: LF = 400 Hz
- Upper frequency limit: HF = 12000 Hz

Therefore, a 14x13 = 182 feature vector defines each push peak audio signal.

This process is performed for the 100 keystrokes of the 7 different keys (A, C, M, O, R, Enter, Space).

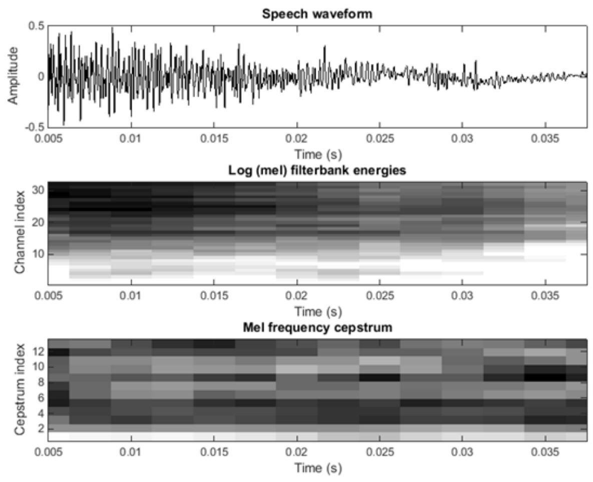


Fig. 6 Different stages of MFCC feature extraction.

To conclude, Figure 5 shows three graphs with different stages for calculating the feature vector of an audio signal belonging to the push peak of the space bar. In the last graph, we can see 13 cepstral coefficients for each frame.

Thus, feature vector is achieved by sequentially storing the 13 cepstral coefficients from the 14 frames.

5. FEATURE EXTRACTION WITH CNN METHOD

Nowadays, deep neural networks are extensively used in audio recognition applications, obtaining performances far beyond other conventional methods. That's why we thought it could be very interesting to try them in our task and see by ourselves how good are CNN for sound recognition problems. Here we propose a CNN architecture made of 2 convolutional layers which is able to achieve 99.43% accuracy on a test dataset using two different classes.

The purpose of this implementation, is to train a CNN that is able to extract the most important features of different key sounds. In order to achieve that we have decided to train the system using only two classes. mainly because of time and resources constraints. Although more than two classes should have been better, we believe it’s not necessary to train the net with all the classes, and once trained for a specific keyboard it should still be able to extract features from other models.

5.1 System set up

The first step on this work was to install and setup the system on which this project has been developed. Although this may seem as a trivial step, this has been one of the most time-consuming ones.

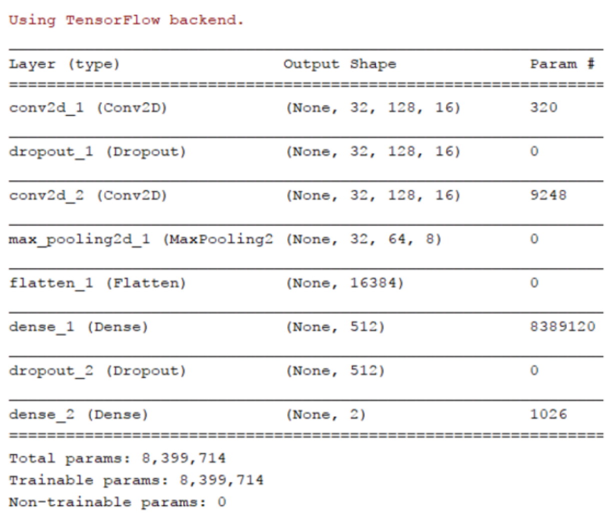
The first step was to install anaconda 3.5 to have the last python environment with all its scientific libraries. Then, install the CUDA drivers along the cuDNN libraries necessary to train the CNN on GPU mode using a single GTX 1050 with 2GB, more than enough to train simple CNN. Regarding the neural network API, we decided to go with Keras, using tensorflow as backend. As this seems to be the most popular configuration nowadays for such kind of tasks and has a lot of documentation online.

Fig.7 Spectrogram of the audio segment corresponding to the A key.

As said before, we have used a total dataset of 2103 samples distributed on two different classes, each one corresponding to a different keystroke. In order to train and evaluate the system, this dataset has been split into 1928 training samples and 175 test samples. For all the different sounds of the dataset, a spectrogram has been computed to obtain images of 16x128x1 pixels that can be feed directly to the CNN input.

5.2 CNN architecture and training

As the two classes used for training seem to be distinctive enough and we only have a very small dataset, we have decided to use a very simple CNN architecture of only 8 layers as this seems the most feasible option. In this case, the structure is the following:



```
Using TensorFlow backend.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 128, 16)       320
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 128, 16)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 128, 16)       9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 64, 8)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               8389120
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 1026
=================================================================
Total params: 8,399,714
Trainable params: 8,399,714
Non-trainable params: 0
```

Fig.8 CNN architecture layout

We have a first convolutional input layer which consists of 32 feature maps with a size of 3x3 pixels using a ReLU activation function. Then, this layer is followed by a dropout stage (set to 20%) and another convolutional layer with the same shape. After this stage we use a max-pooling layer of size 2x2, a flatten layer and a fully connected layer with 512 units using ReLU again, as activation function. Finally, we have 50% dropout layer followed by a fully connected layer with two units as the final stage. Which is going to provide us with the classification scores.

A logarithmic loss function is used with the stochastic gradient descent optimization algorithm configured with a large momentum and weight decay start with a learning rate of 0.01.

We have seen experimentally that with a mini-batch size of 32 and using only 4 epochs we obtain a 99.43% training accuracy.

6. METHOD COMPARISON WITH SVM

In order to compare how reliable are the feature vectors that we have extracted from the two previous methods, we have trained a multi class classifier based on SVM. We used 700 feature vectors for each method (100 keystrokes for the 7 key classes). The number of features (vector length) in MFCC is 182, as we already explained in section 4.

In the CNN case, we have 512 features per vector. These features were extracted before the last network layer, when we have introduced the spectrograms of the 700 samples by the model already trained with the 2103 keystrokes of the A and M keys.

To make a fair comparison, two SVMs are trained with both sets of vectors using cross-validation with Kfold 10, so that in each of the 10 iterations, SVM is trained with 630 vectors and tested with the remaining 70 vectors randomly selected.

7. CLUSTERING

As the real task of keyboard sound recognition is an unsupervised task, we have decided to try at least one algorithm to show the viability of such methods. In this case, for mapping sounds to physical letters finding clusters on the data.

In order to do that, we first have to extract the feature vectors of every keystroke, either by using MFCC or with CNN. Then, after we have such information, the next step is to find some structure in the data that allows us to divide it in clusters by means of similarity/distance, on for each letter. With that procedure, we will be able to group the most similar sounds by a same label, that will finally correspond to a specific letter using prior information and language models.

The purpose of this section is only to show that this is possible using clustering techniques, so, we have not tried to optimize the results. There are lots of different clustering algorithms suited to different tasks, but in our case, we have only tested the hierarchical clustering as it is easy and reliable to implement.

We have used a dataset with 7 different keys, containing 100 feature vectors for each class. By doing so we have been able to separate the data between the different classes quite well.

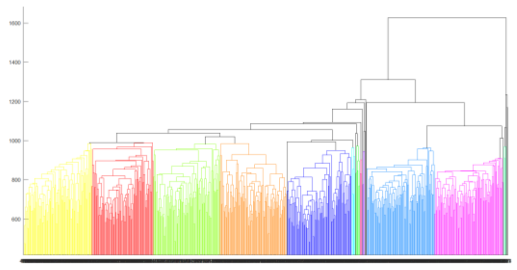


Fig.10 Hierarchical Clustering on the dataset. All the seven classes have more or less been detected

We must say that we only used supervised methods in order to separate the data into clusters after the dendrogram as it the simplest way. In further developments, methods such as Calinski Harabasz, based on variance reduction criterion should be used.

8. RESULTS

8.1 CNN results

After training the model with all the training data we have performed an evaluation with the test dataset, obtaining a 99.43% of accuracy. Which is surprisingly good considering our limited dataset and the low system complexity.

8.2 SVM results

With MFCC we obtain a 98% of correct rate, while in the case of CNN it is 96.4%. Although in this case MFCC obtains the best results, the CNN performance is very hopeful because it shows that, having trained only two key classes, the network is good enough to know which features are essential for each sound. If we had more time to generate a more complete database and trained the CNN with more than two classes (e.g. 10 classes), the results would have been much better.

Despite obtaining a 98% of correct rate with the classifier previously trained with the MFCC feature vectors, the results are not very good when we want to predict words from the dataset (ROMA, CAMA, MARCO). For instance, in the case of MARCO it detects MOMCO (60% of correct characters). Since our study has focused on analysing specific parts of a sound recognizer rather than evaluating the whole system, we do not get very good predictions. However, we have similar results to those obtained in [2] before adding correction methods based on language models, among others.

8.3 Clustering results

After computing the dendrogram with all the data we have obtained a cophenetic correlation coefficient of 0.8529, which is not bad considering we have not optimized all the hyperparameters and we are using very simple descriptors like MFCC.

After cutting of the tree with a fixed threshold to separate the data into different clusters, we have obtained an accuracy of 95.1429%.

9. CONCLUSIONS

The first conclusion we can draw from all the obtained results is that is possible to classify keyboard sounds depending on which key has been pressed. Furthermore, it has been shown that this can be achieved with very high accuracy rates and very simple implementations.

Another important remark is that it is also possible to map the key sounds to each key class using unsupervised approaches as clustering.

However, these tests have been performed under controlled conditions, using a single keyboard, and being used by the same person. The method is quite sensitive to noise or the use of different keyboard models or writers, so its use for intrusive purposes is very limited.

Regarding the CNN feature extraction system we can say it is a very feasible option for that kind of tasks and requires just a minimal setup. With only few layers and a very small dataset we have obtained very good results.

10. REFERENCES

[1] Dmitri Asonov and Rakesh Agrawal. Keyboard acoustic emanations. In 2004 IEEE Symposium on Security and Privacy (S&P 2004), 9-12 May 2004, Berkeley, CA, USA, pages 3–11, 2004.

[2] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. ACM Trans. Inf. Syst. Secur., 13(1):3:1–3:26, 2009.