Voice Commands Recognition For Virtual Reality Environments Using Convolutional Neural Networks

Jose Carlos de Almeida Machado* Cristina Nader Vasconcelos Esteban Walter Gonzalez Clua

Universidade Federal Fluminense, Computer Science Department, Brazil

ABSTRACT

Virtual Reality (VR) environments provide interactivity freedom that can be exploited for reducing natural barriers found by physically challenged people. The use of applications whose interaction is captured by typing or by gesture recognition may be prohibitive. The use of audio recognition is one of the possible alternatives that may overcome such issues and is investigated in the present work. This paper proposes voice command recognition that is performed by a Deep Learning (DL) based approach that can be used alongside VR games. In general, classical speech recognition solutions are based on the construction of a specialist system in which invariance such as ambient noise or accent should be modeled specifically. The proposed model is based on the use of Convolutional Neural Networks (CNN) that are expected to learn a hierarchy of features from a database. In that sense, a new annotated database was constructed aiming to cover a variety of environmental noise, type of voice, accent, intonation, and other aspects that enriched the information presented to the network for training. It covers 326 audio samples, of twenty-three different speakers, and five commonly used game commands. The methodology achieved 66% of accuracy rate in experiments with a single CNN.

Keywords: audio recognition, neural networks, convolutional neural networks, virtual reality.

1 INTRODUCTION

Voice recognition is currently being used daily on applications such as GPS and television controllers. It may became a frustrating experience when a command word is not understood by the program or the user's speaking manner needs to follow a certain pattern to be comprehended.

Identifying voice commands is a complex task. One word can be poorly identified for various reasons, such as accents, dialects, context, agglutinations, finesse, multiple ways the sound could be written, recording failure, and so on.

In a VR environment, the attempt to mimic the physical world is critical and the interaction with it should be as immersive as possible. In order to achieve a more fluid sequence of actions, speaking is an alternative to gestural interaction, making VR more accessible for physically challenged people, and creating a new option for the regular usability.

The current recognizers use syllabic identification [1], comparing the input with a phonetic database given an error tolerance, place the information together to form words and try to identify what was said. Even though this approach performs well in many scenarios, it demonstrates difficulties in highly diverse speaking possibilities, since they must be previously modeled. The use of DL could be a valid solution for the future of this technology, since it can identify patterns from provided data of each word, instead of previously modeling it.

*e-mail: jcamachado@id.uff.br

This project aims at creating a command recognition system using CNN capable of learning and classifying voice commands with higher precision independently of speaking variation. For this, it is necessary to have an audio database with commonly used commands present in these applications. These commands are represented as spectrograms that serve as input for a CNN to learn patterns of each command.

2 METHODOLOGY

2.1 Neural Networks' Selection

In this project, two neural networks are being used to train a model to classify the dataset. The first is AlexNet due to its relatively low error rate on classifying ImageNet's dataset, using data augmentation techniques [4]. The second one is GoogLeNet that contrasts with the "simplicity" of AlexNet, having a lower error rate on ImageNet Large-Scale Visual Recognition Challenge (ILSVRC), deeper and more complex architecture and higher computational cost [7].

Both were top competitors at ILSVRC at different years. Additionally, the use of NVIDIA DIGITS and its practicality in giving a graphical user interface for dataset and manipulation on both networks were reasons to choose them.

2.2 Converting Audio to Image

As said before, an open and free audio database was necessary, with specific commands simply and intuitively annotated in classes to be used with NVIDIA DIGITS. From these audio (.wav) files, grayscale spectrograms were created. The spectrogram is a graphic representation of the sound waves [2], mapping frequency as the vertical axis, time as the horizontal axis, and power represented as the pixel color on the resulting image, where stronger patches are darker and silence is white.

Converting audio to image is proposed as an alternative approach, since the networks mentioned in section 2.1 have a predefined support for image input.

The spectrograms were created using Python, based on [8] which uses short-time Fourier transforms to help extract the frequency information from the audio signal [5]. The resulting images' axes are dependent on the frequency and time.

In order to reach the requirements for the intended networks, the input data must be an image with 256x256. Processing these spectrograms were necessary to fit these dimensions. It was decided to normalize the Y-axis and making slices along the X-axis. This process was an attempt to minimize data loss while comprising the image, keeping the frequency range within 256 samples in all images.

Homogenizing the images' height demands extracting a division factor **F** from the image with highest frequency. Dividing the maximum height on the database **hmax=480** by the target size, **256**. So, **F=hmax/256** and our factor is **F = 1.875**. By dividing each image's height by this factor, the resultant images are smaller than or equal to 256 pixels on the Y-axis. When the result -was smaller than 256 (On Y or X-axis), the image was filled with absolute silence, white pixels, as noted in Figures 2 and 3.

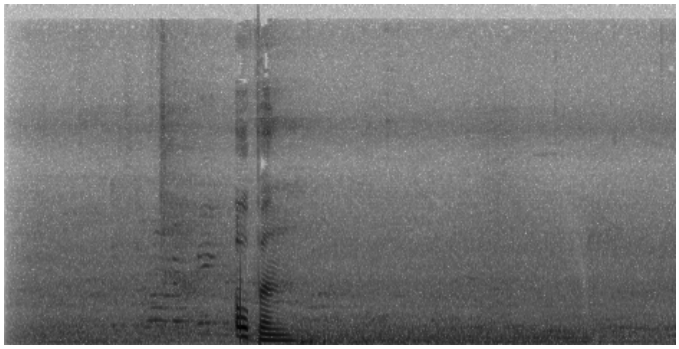

Figure 1: Spectrogram example: Command "Open". Dimensions: 468x237.

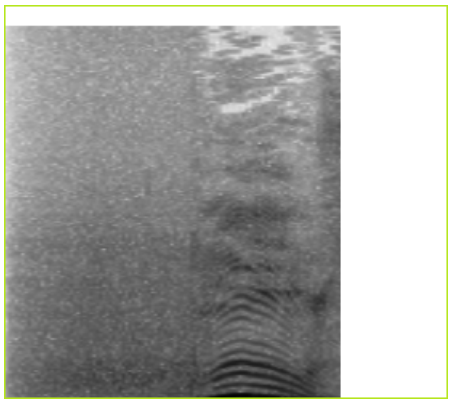

Figure 2: Spectrogram example: Command "Close". Dimensions smaller than 256x256.

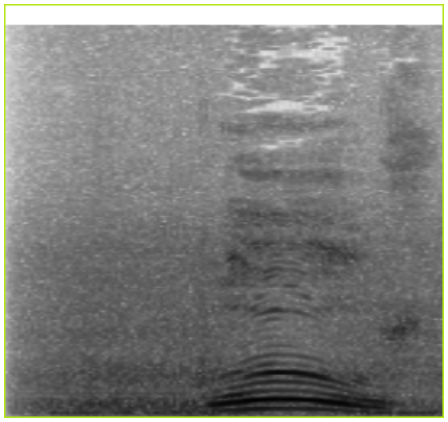

Figure 3: Spectrogram example: Command "Close". Y dimension smaller than 256 pixels.

Another case is that time is not being compressed to 256 pixels in order to avoid sample loss over time. Images using less than 256 time samples are filled with silence as mentioned before. But when the time has more than 256 pixels, the image is sliced. This method also helps to increase the amount of data in the database.

The spectrogram is represented as a 256xT picture, where **T** is the total value of pixels on the X-axis. For every spectrogram where T<256, the image will be filled with silence on the gap until it reaches the aimed size of 256 pixels. T=256 means that no manipulation needs to be done. When T>256, T-255 images will be generated from the original spectrogram. This process starts by the leftmost pixel in (0,0).

The Y-axis goal was achieved in a previous step. Now, the image will be sliced vertically in x and x+255, starting at x=0. This slice is a new 256x256 image in the database. When x is zero, we have the leftmost sub-image from the spectrogram. This process repeats itself increasing the x's value by 1, until it reaches a value less than 256 pixels apart from the original image size (x < T-255), in order to not access invalid memory position. Using this process, the total of images at the database increased from 326 images to 57972 images.

This approach for data augmentation provides two advantages. One is to reinforce the learning of an audio pattern multiple times since the same spectrogram could several new slices containing its data signal. The second advantage is that to make the classifier more robust to displacement through time, the slices from the same spectrogram were translated by one pixel from each other on the time axis (X-axis).

$$Slices = \begin{cases} \text{if T >= 256} & : T - 255 \\ \text{otherwise} & : 1 \end{cases} \quad (1)$$

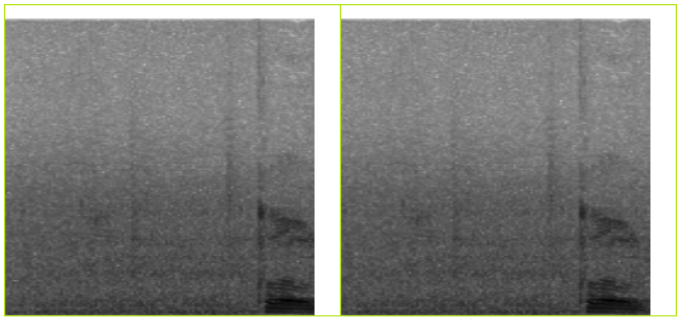

Figure 4: Note that 4-a) has the same source than 4-b). However 4-a) is a slice that comes chronologically before 4-b).

2.3 Annotation

The annotation process is simply defined, but demands manual work and could take time. Given that the training is supervised, it is necessary to provide information about the relationship between each image and the correspondent command

This project's indexation works on the original spectrogram. Each column on the spectrogram represents the smaller audio sample represented. The information provided on each column (time sample) is used to identify where the command is inside the picture. This process was performed manually. Given where the command is in the original picture, it becomes possible to map if the spectrogram's slice does or does not have a relevant amount of command samples in it. And if it does not, the sub-image is considered as noise.

The information about each image column is placed in text files (.txt) that were created using the same name as the correspondent

original image. The content of each file starts with a collection of characters. Each character corresponds to a command and its position on the String maps the pixel column on the original image (i.e. string[0] maps image's first column). The characters used to index the commands in the file are: Close **c**, Open **o**, Pick **i**, Pause **p**, Throw **t** and Noise **s**. Thus, this String has T identifiers, one for each sample. Any other text after the T-th column must be ignored. In this case, it was used as a comment session indicating where to index each command.

```
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssscccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccsssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssss

0-204
205-311
312-380
```

Figure 5: Annotation file's content example. The first sequence of character is the reference String. The text that follows this sequence is not used in code. It was used as comment session, recording meaningful indexes to favor the String construction.

After finishing annotating the samples, it is required to define the criteria to consider a minimum amount of samples in a sub-image that are not noise in order to consider the image as having a command. The decision used to choose the minimum number of samples is based on the spectrogram with the lowest number of command samples, in other words, the image where the command was said faster. The spectrogram with fastest spoken command gave us 32 samples, corresponding to the Close command. Thus, for a sub-image to be considered having a command, it needs at least 32 samples from the correspondent command. In this case, it will be tagged and allocated to the correspondent class of commands, otherwise, it will be classified as Noise. A name identifier is given to compose the final name of the sub-image file. After this process, a spectrogram' slice is generated using the identifiers in its file name, accordingly to its creation process.

3 Experiment

3.1 System Requirements and Technology

The machine where this experiment was performed is a personal computer with relatively accessible components. The physical components of this computer include an Intel i3-2120 3.3GHz, two main memories 8GB DDR3 RAM, and one GPU GTX-1070 with 8GB RAM.

The system's software used in this experiment was Linux-Ubuntu Operational System 16.04.2; NVIDIA DIGITS 5, for a simple interface manipulation of Neural Network's hyper-parameters. CUDA version 8.0 was used alongside DIGITS. Additionally, Caffe was chosen to work with AlexNet e GoogLeNet. As stated in section 2.2, the database was generated using Python 2.7.

3.2 Hyper-parameters

Choosing which hyper-parameters would be used in the learning experiment is a complex task where promising values can be ignored by inexperience or lack of knowledge on the experiment subject. From this point, it may be necessary to experiment by trial and error. The hyper-parameters used in this experiment did not follow any specific method. These were defined mainly on manual fine-tuning, an approach similar to [3].

Depending on the hyper-parameters (e.g. **Epoch**, **Learning Rate** and **Batch Size**), the learning time varies. The network with the fastest experiment execution was AlexNet, which within 70 epochs took 30 minutes to execute. Oppositely, GoogLeNet took more time and resources, as expected due to its complexity.

3.3 Dataset Learning

Works with Neural Networks using audio are not widely explored and open datasets may not be as specific enough for every issue. Because of that, it was necessary to create a more suitable database for this problem called **AudiUFF** (Audio UFF).

It is worth mentioning that AudiUFF was used in two versions. The first is a smaller version containing with 46138 images and the second version was composed of 57972. This difference is due to an addition of 51 audio files (.wav) on the database, from 271 to 326, which increased the difference after the data augmentation.

The database is divided into 6 command classes, chosen as a representation for application commands. They are: **Open**, **Close**, **Pause**, **Pick**, **Throw** and **Noise**. Each class represents one voice command that has the same name, except Noise. Noise's class represents audios that do not fall on the aforementioned classes. This includes silence, background noise or misunderstood words. Each file has only one valid non-noise command.

According to this data classification, the Noise class could contain other words and serve as a reservoir for spare commands that could be used in other applications.

Twenty-three people participated as speakers in this database, sixteen men, and seven women. To increase the amount of data and its variation, they spoke between one and five times each command, in different tones and surroundings. In order to increase the variety of vocal patterns, the participants are from different parts of the world, being one from the United States of America, another from Argentina, and the remaining twenty-one are from Brazil, from which at least six speakers had known experience as exchange student in different regions of the United States for at least one year. This provides diversity in signal recording and pronunciation of each command since the speakers have different fluency in English and used different devices to capture the audio, given the reduced amount of people and data recorded.

3.4 Dataset Configuration

Hypothetically, a scenario where all classes, 6 in this case, have the same number of samples, the database percentage of files per class would be approximately 16%, so any accuracy rate above this number on the learning process can be considered as better than randomly guessing. However, in AudiUFF, similarly to what happens to phone calls [6], a high percentage of audio time is composed of silence or background noise. The noise class represents 40% of the whole database, which includes augmented data. Considering this, it is fundamental to be aware of misleading accuracy.

Following this idea, three database settings were used in this project, considering different scenarios. The first one, Noisy setting, used a noisy database, where 40% of it was composed of noise class files, while the other classes ranged from 12.5% to 10.9% of the training set. Validation and test data followed a similar ratio between classes. The training set represented 88% of the whole data, while validation and testing represented 6% each.

The second setting, Balanced setting, is similar to the first, but unlike the previous one, there is a regular distribution between the classes, leaving the most numerous class with 17.3% of the training set, and the least numerous with 15.1%. When this distribution was applied, the percentage of each set changed (training, validation, and test sets). Training set's percentage changed to 80% of the total data, validation to 11% and test to 9%. It is important to note that 40% of the test and validation data included audio from people that participated in the training set, but using different audio files. On this approach, in order to achieve an approximate distribution, the files from the noise class were pseudo-randomly deleted. The intention was to remove files affecting all original spectrograms.

The variation of database's classes distribution had the objective of avoiding inaccurate results. For example, using the noisy database, a 40% accuracy rate could indicate the false idea of a

good result. However, if the model tried to guess every sample as Noise, it would give us the same result, since 40% of the entire database was composed by noise files.

On the third configuration, DIGITS selected randomly 25% of the training set as validation and test sets. It was defined using the Noisy database.

4 Results

4.1 Accuracy Rate

Starting with the third configuration. Since the data origin could be very similar, due to the data augmentation technique, the accuracy of 99% suggested a problem. The interpretation is that samples from the same spectrogram were used in test, validation and training sets, giving a poor diversity range on the prediction. Taking this in consideration, this configuration was discarded.

The Noisy database was composed by approximately 40% of Noise's class samples. Considering this, the results could have been biased. The accuracy rate ranged from 35% to 66%, which seems to be a good result in an equal distribution between classes. Although, if the results were around 40% accurate, it is valid to assume that the network learned how to predict only one class, choosing Noise as a safe guess guaranteeing 40% of hit rate. In other words, the worst decision could be a lucky guided guess.

The best result GoogLeNet achieved in this configuration was 50%, using the bigger version of AudiUFF and: **learning rate = 0,001**, **batch size = 5**, and the other parameters as the current standard from DIGITS. It took 8 hours on the specified machine. AlexNet achieved the mark of 66 % rate on the smaller version of AudiUFFF using the following hyper-parameters: **learning rate = 0,01**, **batch size = 15**, **batch accumulation = 5**, $\gamma = 0{,}4$, **step down = 20 epochs**, the remaining parameters were DIGITS standard values. It took around 1,5 hours to complete the training.

Even though the results were cheering, they could be dubious. It was necessary to check in a second and more balanced configuration. The Balanced setting had class proportions ranging within 2.2% (17.3%-15.1%). Consequently, the targeted accuracy must be at least 20% to be considered as better than randomly choosing.

However, the experiments in the Balanced setting had accuracy between 35% and 62%. Three times better than the worst significant case, and worse than the Noisy configuration. This result was achieved by AlexNet on the bigger version of AudiUFF, using: **learning rate = 0,02**, $\gamma = 0{,}4$, **batch size = DIGITS' default**, **batch accumulation = DIGITS' default**, and **step down = 20 epochs**. Its execution took 1.5 hours.

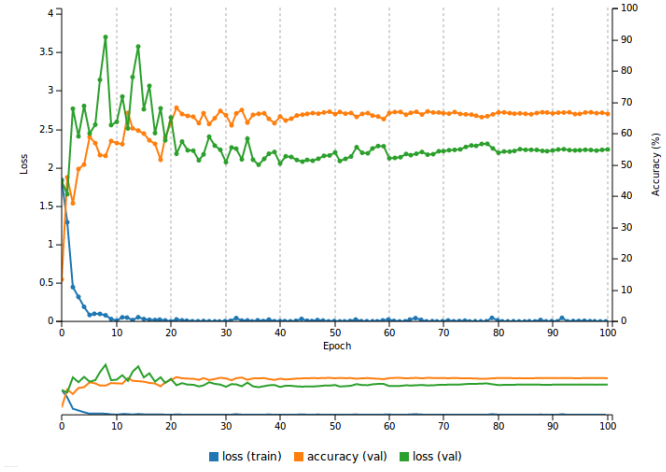

Figure 6: Graphic displaying the learning of AlexNet through 100 epochs on the Noisy configuration using a smaller version o AudiUFF.

5 Conclusion

The results can be considered positive given the collected data in this experiment. Even though the database has a considerable size, it is not truly diverse, considering that it was augmented from a few files, even though it was recorded in different situations. Even using the data augmentation techniques, a more heterogeneous dataset could help the network to increase its pattern detection.

Additionally to expanding and enriching the database, experimenting with different parameters, finetuning using bigger datasets and using network arrangements (such as committees) could also contribute to improve the results.

Ultimately, following the above considerations, the average accuracy rate of 40% in the first two settings shows an increase of 2.5 times the accuracy rate of selecting randomly a command. As a first experiment using this simply defined methodology, it indicates a potential for improvement, allowing a projection of better results in the future.

References

[1] M. Cravero, R. Pieraccini, and F. Raineri. Definition and evaluation of phonetic units for speech recognition by hidden markov models. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'86.*, volume 11, pages 2235–2238. IEEE, 1986.

[2] O. Gencoglu, T. Virtanen, and H. Huttunen. Recognition of acoustic events using deep neural networks. In *Signal Processing Conference (EUSIPCO), 2014 Proceedings of the 22nd European*, pages 506–510. IEEE, 2014.

[3] S. E. Kahou, X. Bouthillier, P. Lamblin, Ç. Gülçehre, V. Michalski, K. R. Konda, S. Jean, P. Froumenty, Y. Dauphin, N. Boulanger-Lewandowski, R. C. Ferrari, M. Mirza, D. Warde-Farley, A. C. Courville, P. Vincent, R. Memisevic, C. J. Pal, and Y. Bengio. Emonets: Multimodal deep learning approaches for emotion recognition in video. *CoRR*, abs/1503.01800, 2015.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[5] M. Müller. Short-time fourier transform and chroma features. 2015.

[6] J. Ramirez, J. M. Górriz, and J. C. Segura. Voice activity detection. fundamentals and speech recognition system robustness. In *Robust Speech Recognition and Understanding*. InTech, 2007.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6 2015.

[8] F. Zalkow. Create audio spectrograms with Python. http://www.frank-zalkow.de/en/code-snippets/create-audio-spectrograms-with-python.html?i=3. Accessed: 2017-05-22.