Speech Command Recognition with Convolutional Neural Network

Xuejiao Li
xjli1013@stanford.edu

Zixuan Zhou
zixuan95@stanford.edu

Abstract—**This project aims to build an accurate, small-footprint, low-latency Speech Command Recognition system that is capable of detecting predefined keywords. Using the Speech Commands Dataset provided by Google's TensorFlow and AIY teams, we have implemented different architectures using different machine learning algorithms. Our models include: Vanilla Single-Layer softmax model, Deep Neural Network and Convolutional Neural Network. The Convolutional Neural Network proves to outperform the other two models and can achieve accuracy of 95.1% for 6 labels.**

Keywords—**Keyword Spooting (KWS), Deep Neural Network (DNN), Convolutional Neural Network(CNN)**

I. INTRODUCTION

Thanks to the rapid development of mobile devices, interacting with machines using voice technology has become increasing popular. Related products like Google Now or iPhone's Siri both exploit speech command technology. Google has also offered the service to search by voice [1] on Android phones and a fully hands-free experience called "Ok Google"[2]. As a matter of fact, keyword spotting (KWS) technology is a potential technique to provide fully hands-free interface, and this is especially convenient for mobile devices compared to typing by hands. And it is also the desired technique for situations like driving or some emergency cases. Since speech command recognition system usually runs on smartphones or tablets, it therefore must be low-latency, and must have a very small memory footprint, and require only very small computation. Thus the motivation of our project is to build a keyword spotting system that is capable of detecting predefined keywords and helps device to interact differently based on what the command asks for. Specially for this task, our dataset is provided by Google's TensorFlow and AIY teams, which contains 65,000 WAVE audio files of people saying thirty different words [3]. Each of the audio clip lasts for one second and contains one single word. According to different requirements, different predefined keywords are required, such as *"yes"/"no" or "up"/"down"/"left"/"right"* or *"stop"/"go"*. The predefined keywords can be reconfigurable, thus enabling our system to work for different labels with high flexibility. The system tries to classify a one second audio clip as either "*silence*", an "*unknown word*" or one of our predefined keywords. We then use a single-layer softmax model, a DNN model and a CNN model to calculate the probability that the input audio belongs to each of the labels and finally output the predicted label that the machine believes the input audio clip belongs to. The task is very meaningful and can be configured and run in an Android application.

We describe related work in section Ⅱ, dataset and preprocessing method in section Ⅲ, and three models, namely, Vanilla, DNN, and CNN, in Section Ⅳ. The experiment setup, results and some discussion follow in Section Ⅴ. Section Ⅵ closes with the conclusions.

II. RELATED WORK

Machine learning has been proved to have powerful ability for classification task. A commonly use technique for KWS is the Key-word/Filler Hidden Markov Model (HMM) [4, 5, 6, 7, 8]. In this generative approach, for each of the keyword, an HMM model is trained, and a filler model HMM is trained from the non-keyword segment of the speech signal (filters)[9]. This method is very computational expensive, since HMM requires Viterbi decoding. Other recent work explores some discriminative models based on large-margin formulation [10,11] or recurrent neural network [12,13]. These techniques show some improvement over HMM approach, but have relatively long-latency, since they either require to process over the whole speech to find the region of the keyword or take inputs from a long period of time to predict the keyword. The current KWS system at Google [9] uses a DNN, which outperforms the traditional HMM system and is also very simple and requires relatively lower computation. However, we believe that a CNN model can provide further improvement over a DNN model in a variety of small and large vocabulary tasks [14,15,16].

CNNs are better than DNNs for KWS task for mainly 2 reasons. First, DNNs just ignore the input topology and resize it into column vectors. However, for audio signals, the spectrum representations show very strong correlations in time and frequency. So modeling local correlations with CNNs will be beneficial and is expected to have much better performance than DNNs. Second, thanks to the parameter sharing quality of CNNs, CNNs can have far fewer parameters compared to DNNs for the same task, thus reducing memory footprint and computational requirement. So CNNs will have improved performance and reduced model size over DNNs and is thus the state-of-the-art technique for KWS task.

III. Dataset and Features

A. Dataset preparation

We are provided with the Speech Commands Dataset from Google's TensorFlow and AIY teams, which consist of 65,000 WAVE audio files of people saying thirty different words, each of which lasts for one second. The data set has been separated into different categories like numbers, animals, directions or person names. By doing so the system can be trained with more specific purposes. We divided the data set into three part, including 80% training set, 10% validation set and 10% test set, and each subset of speech audio is classified as either *silence*, *unknown word*, or *predefined keywords*, which are attached different labels respectively.

Key word. The key word class, labeled separately, contains a set of concerned words. In this project, "up", "down", "left", "right" are chosen for this speech recognition task.

Unknown word. The unknown word audio clips capture the words, which are not concerned about. This class is differed from the key word class, which can motivate the network to learn which speech should be ignored or captured.

Silence. Only relatively silent situation can be recorded, this class contains audios over a variety of quiet environment.

In addition, to obtain a more robust set of parameters, the background noise is attached to the training set proportionally. In more specific, the noise audio, captured from machinery and household activities, is divided into small segments, which is randomly mixed into the training audio clip with an adjustable lower volume. In this project, the background volume is 0.1 and its frequency is 0.8.

B. Feature Extraction with MFCC

We calculated Mel-Frequency Cepstral Coefficients (MFCC) to extract spectral features. Based on human perception experiments, Mel-Frequency analysis is employed to re-weight dimension of frequency and gain more perceptually-relevant representation of speech audio [17].

The diagram of feature exaction is shown in the Fig. 1. We firstly define a 30-ms analysis window, and divide the speech signal into different time frames by shifting the window (shifting stride = 10 ms). Since audio signal sample is 1s each, we will have (1000-30)/10+1=98 time frames, as shown in Fig. 2. After windowing, Fast Fourier Transformation (FFT) is calculated for each frame to obtain the frequency features, and the logarithmic Mel-Scaled filter bank is applied to the Fourier transformed frames. The last step is to calculate Discrete Cosine Transformation (DCT) to obtain the 40-dimentional coefficients vector. In this project, we finally obtained a [98×40] 2D matrix desired to feed the successive neural network.

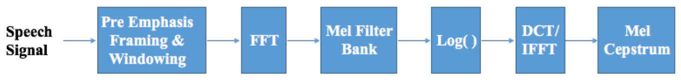


Fig. 1. Diagram of MFCC Derivation Process

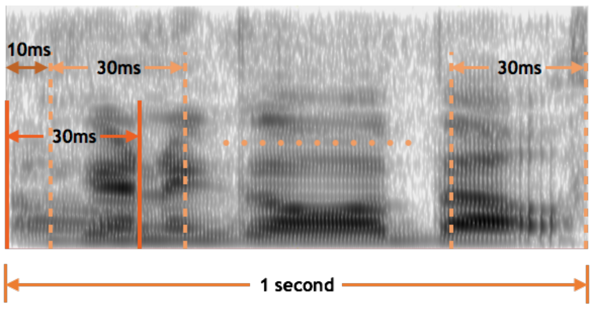


Fig. 2. Feature Exaction Window

IV. Methods

A. Vanilla Single Layer Softmax

We firstly built a model with a single hidden fully-connected layer and a softmax output layer.

This simple model has only one matrix multiplication and bias, and the number of the output nodes is the same as the labels. As expected, Vanilla can't produce very accurate results, but can work very fast.

B. Deep Neural Network

Our second model is a standard feed-forward fully connected neural network with 3 hidden layers and 128 hidden nodes per layer, as shown in Fig. 3. We use 3 hidden layers because in practice, a 3 hidden layers fully-connected neural network usually outperforms DNNs with 1 or 2 hidden layers, but only slightly worse with DNNs with 4 or more hidden layers. Another empirical experience is to use more hidden nodes per layer to achieve higher accuracy, although it may lead to overfitting. In this project, dropout technique is used to prevent overfitting.

For the hidden layers, we use rectified linear unit (ReLU) as activation functions for computing reduction, the weighted sum of the output from previous layer. Compared to Vanilla Single Layer, this model is expected to give a more accurate result at the cost of more memory footprint and higher computational cost. Apart from that, DNN model is desirable for device, as its size can be easily adjusted via altering the number of parameters in the network [18].

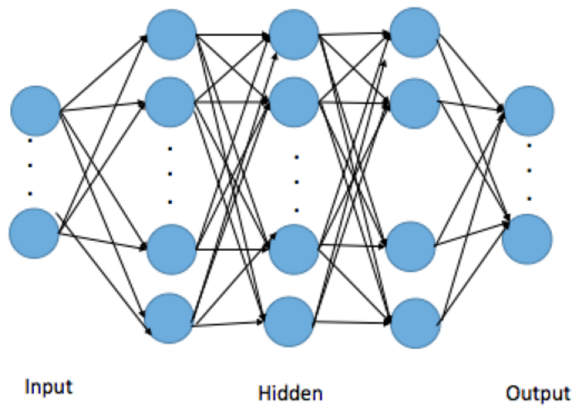


Fig. 3. Fully-Connected DNN structure

C. Convolutional Neural Network

As stated in section Ⅱ, Convolutional Neural Networks (CNNs) are attractive for keyword spotting (KWS) task, so we have implemented a convolutional architecture with two convolutional layers. For our interest, some key layers are: Convolutional (Conv) layer (multiple convolution filters to obtain different features), Pooling layer (down-sampling by taking max operation to reduce the amount of parameters and computation in the network, and hence control overfitting), Dropout layer (only keep a neuron active with some probability p, or set it to zero otherwise to control overfitting), Linear low-rank (Lin) layer (perform linear multiplication and addition to transfer the output of Conv layer to discrete nodes, reduce parameters and computation, control overfitting), and Fully-connected (FC) layer (preserve full information, or make the final softmax prediction). Our CNN model is cascaded as in Fig. 4. The reason why we choose only to apply 2 Conv layers instead of the state-of-art very deep and big CNNs is to limit the number of parameters at the sacrifice of some accuracy. We have successfully kept the number of parameters below 250K, which is feasible for small-footprint KWS tasks on mobile devices where memory footprint is limited. The number of parameters we need is shown as in Table 1.

In section Ⅴ, we will show the benefit of this architecture for KWS compared to a DNN and a Vanilla single-layer network.

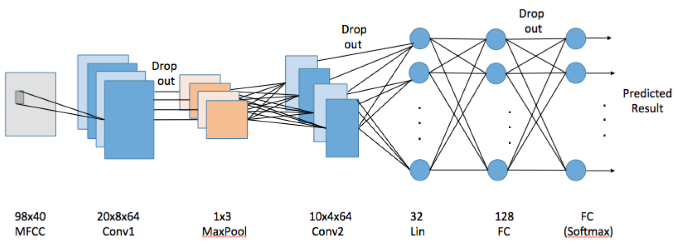


Fig.4. Structure of Convolutional network architecture

Table 1: CNN architecture

Type	Ht.	Wd.	Depth	Stride Ht.	Stride Wd.	Par.
Conv1	20	8	64	1	3	10.2k
Conv2	10	4	64	1	1	164.8k
Lin	-	-	32	-	-	65.5k
DNN	-	-	128	-	-	4.1k
Softmax	-	-	6	-	-	0.7k
Total	-	-	-	-	-	244.4k

V. EXPERIMENTS AND RESULTS

A. Training Details

Training Environment

We choose to use a GPU to train our network, since the required computation power of neural network is huge. Our training environment is shown in table 2.

Table 2: Training Environment

Language	Python3.5
Framework	Google TensorFlow 1.4.0
GPU	Nvidia GeForce GTX 960M
GPU Memory	4044MB

Initialization

Weights are initialized randomly from a truncated normal distribution with zero mean and specified standard deviation for symmetry breaking.

Since we do not know the final value of every weight in the trained network, but with proper data normalization it is reasonable to assume that approximately half of the weights will be positive and half of them will be negative. Therefore, we want the weights to be very close to zero, but not identically zero, because if every neuron in the network computes the same output, then they will also all compute the same gradients during back-propagation and makes the exact same parameter updates.

Batch size

Batch size for gradient descent is equal to 100. For each step, we randomly choose 100 training samples, which can break the correlation among them and make the network learn more efficiently.

We choose batch size not equal to 1 to avoid overfitting. However the batch size can also not be too large, since training a neural network can be extremely computational consuming. So this value is a trade-off between performance and hardware limitation.

Learning rate

The learning rate is 0.001 for the first 5/6 of total steps followed by 0.0001 towards the end. The learning rate for the latter is relatively smaller, since we are fine-tuning the model for the latter steps. We tried several combinations and find this one can obtain both high efficiency and good convergence.

Update Method

Since one drawback of stochastic gradient descent is that updating direction depends completely on current batch, momentum method is introduced to stabilize updating process. We applied Nesterov Momentum update, which in practice works slightly better than standard momentum. First, we update one step along original direction, and then compute its gradient, according to which we correct the final updating direction, expressed as:

$$x_{t+1} = x_t + \Delta x_t \qquad (4.1)$$

$$\Delta x_t = \rho x_{t-1} - \eta \Delta f(x_{t-1} + \rho x_{t-1}) \qquad (4.2)$$

where, η is learning rate, and ρ represents momentum momentum value. In the beginning stage, the gradient is relatively large, for which initial value of ρ is 0.5, while a large ρ is chosen for small gradient. In the project, momentum value ρ is assigned with 0.5, 0.9, 0.95, 0.99 with regard to the increasing steps [19].

Cross-entropy

For each model, last layer employs a softmax classifier with cross-entropy loss L to estimate the posterior of each output label, expressed as:

$$L_i = -log\left(\frac{e^{f_{yi}}}{\sum_j e^{f_j}}\right) \qquad (4.3)$$

where, $f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$ is softmax function, which means the score for j-th element.

Regularization

We applied dropout method as regularization technique, which can reduce overfitting in neural networks by preventing complex co-adaptations on training data [20]. In the experiment, drop probability is equal to 0.5, as it can maximize number of randomly-generated network structures.

B. *Result and Discussion*

Our 6 labels for this KWS task is: *"up"/"down"/"left"/"right"/"silence"/"unknown"*.

We performed 33,000 training steps for there 3 models individually, and every 400 steps, we perform a test on validation-set. After training we perform a single prediction on our test-set to obtain test-accuracy.

Definition

We introduce accuracy, precision, recall and loss as our performance metrics. We also plot the ROC/AUV curve, precision-recall curve to demonstrate the performance of there 3 models. The definitions of these metrics are shown in Fig.5. The ROC/AUV curve uses false positive rate (FPR) as its x-axis and true positive rate (TPR) as its y-axis.

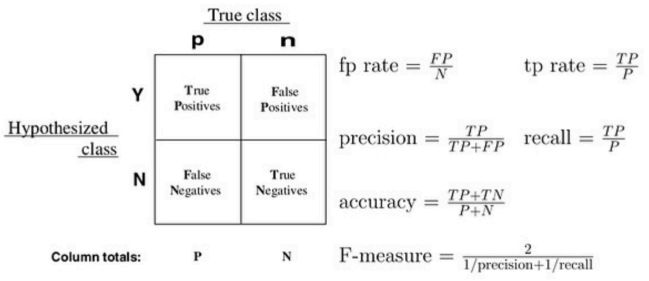


Fig.5. Confusion matrix and common performance metrics

Accuracy and Loss

We calculated the accuracy over validation-set and plot the accuracy trend (fitting) of 3 different models in Fig. 6. It is clear that compared to DNN and Vanilla, CNN model performs a smoother curve has a higher final accuracy on validation set.

After training we perform a single prediction on our test-set to obtain test-accuracy. We plot Table 3 to illustrate the final accuracy and loss on test set.

The first observation is that Vanilla is not good at all with only 56% accuracy overall and highest loss, but also not surprising given that it only performs linear computation. Another observation is that DNNs achieve roughly 72% accuracy and smaller loss, indicating that the non-linear combination of 3 hidden layers can improve the result significantly. Finally a simple 2-ConvLayer CNN network is already outperforming the Vanilla and DNN and achieves 31.43% and 66.67% relative improvement with regard to DNN and Vanilla in test-accuracy and 82% and 94.8% in loss. What's more, CNN also proves to have faster convergence speed.

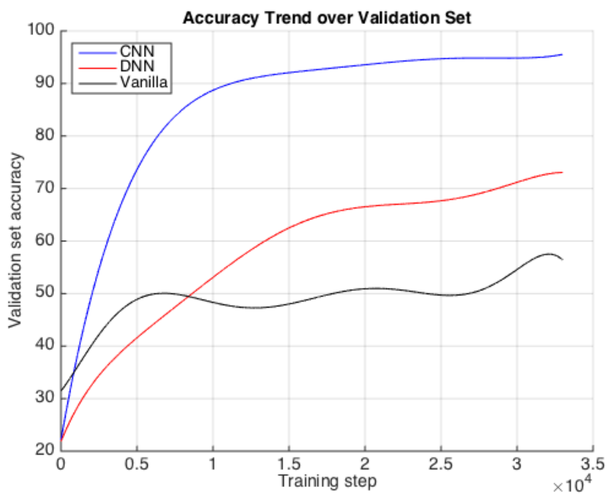
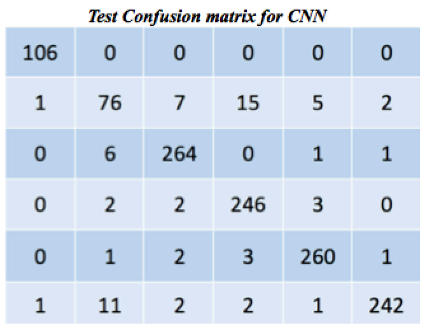


Fig.6. Accuracy trend over validation set

Table 3: Comparison between accuracy loss

Models	Validation -Accuracy	Test-Accuracy	Loss
CNN	95.1%	94.5%	0.190
DNN	72.5%	71.9%	1.048
Vanilla	57.3%	56.7%	3.640



Test Confusion matrix for CNN

106	0	0	0	0	0
1	76	7	15	5	2
0	6	264	0	1	1
0	2	2	246	3	0
0	1	2	3	260	1
1	11	2	2	1	242

Fig.7. Test Confusion Matrix for CNN

From Table 3 we also find that the differences between test-accuracy and validation-accuracy are relatively small, indicating we haven't got overfit to our training-set.

Finally we show a confusion matrix for the CNN model for the test-set in Fig 7. The columns of the confusion matrix represent a set of samples that were predicted as *"silence"*, *"unknown"*, *"up"*, *"down"*, *"left"*, *"right"* respectively. All of the entries are very small apart from the diagonal lines through the center, which indicates the CNN model makes very few mistakes. We can also see that the biggest confusion for CNNs now is to identify *"unknown"* and *"up"*.

Recall, Precision, ROC and AUC

We calculated precision value and recall value using confusion matrix for test-set and we plot Table 4 to illustrate the recall and precision values on test set. We also plot the

Precision-Recall curve illustrated in Fig. 8, 9 and 10 and the ROC/AUC curve in Fig. 11.

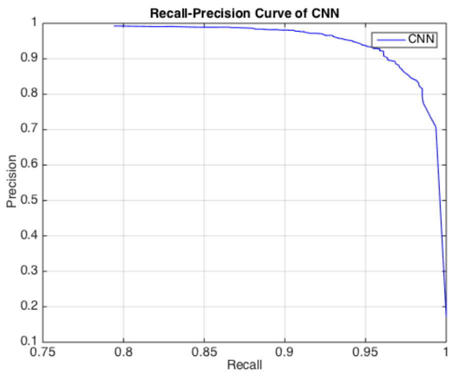


Fig.8. Recall-Precision curve for CNN

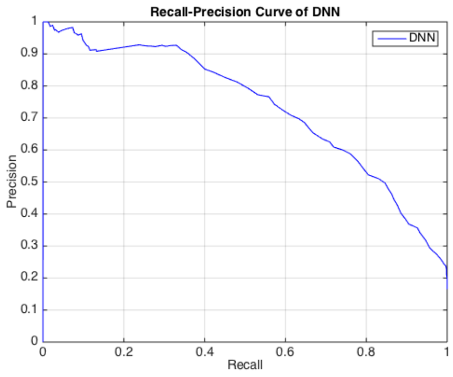


Fig.9. Recall-Precision curve for DNN

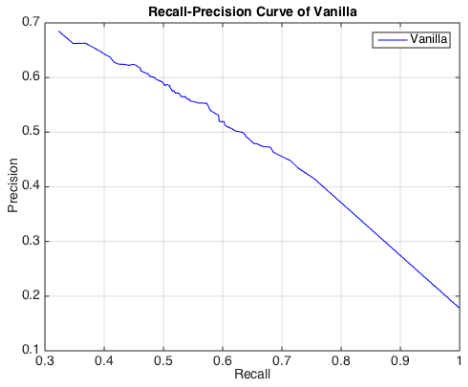


Fig.10. Recall-Precision curve for Vanilla

Table 4: Precision Value and Recall Value for 3 models

	CNN	DNN	Vanilla
Precision Value	0.9330	0.7866	0.5416
Recall Value	0.9280	0.6754	0.5734

These results indicate that CNN is more effective than DNN and Vanilla, giving 18.6% relative improvement over DNN and 72.3% over Vanilla on precision value; 37.4% over DNN and 61.8% over Vanilla on recall value, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate.

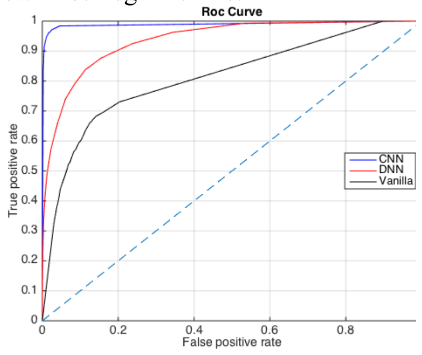


Fig.11. ROC/AUC Curves comparing 3 models

The precision-recall curve shows the trade-off between precision and recall for different threshold. CNN has the highest area under the curve, representing both high recall and high precision. This means that our CNN model is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).

ROC/AUC curve shows that CNN model achieves a much higher AUC, and gains dramatically at a very low false positive rate, which is a desirable property for speech command system.

Reconfiguration Ability

We have also tested other labels besides "up"/"down"/"left"/"right", in order to test the reconfiguration ability of our CNN models. The result is illustrated in Table 5, indicating our CNN model can maintain high accuracy with regard to different labels and different number of labels.

Table 5: Accuracy for different labels

Labels	Yes/No	Up/Down/ Left/Right	0-9
Accuracy	95.7%	94.5%	93%

VI. CONCLUSION AND FUTURE WORK

In this project, we used 3 models for KWS task: Vanilla, DNN and CNN. All of these 3 models exploit softmax classifier and MFCC feature extraction. The experiment results show that CNN model outperforms the other two models and achieves 31.43% and 66.67% relative improvement with regard to DNN and Vanilla in accuracy; 82% and 94.8% in loss; 18.6% and 72.3% in precision value; and 37.4% and 61.8% in recall value. One limitation of our CNN model is the huge number of multiplies in the second ConvLayer because of the 3-dimentional inputs spanning across time, frequency and feature maps. So our next step is to dive into some new CNN architectures with fewer multiplies and thus make it feasible to work on some power-constrained devices.

CONTRIBUTIONS

Zixuan dived into researches about feature extraction and implemented Vanilla and DNN models, Xuejiao dived into training methodology, trained the hyper-parameters, implemented CNN models and evaluated the performance metrics. The team worked together to run the training, process result data, plot curves and tables, as well as make poster and write the final report. We share ideas through frequent discussions and cooperate each other.

REFERENCES

[1] Johan Schalkwyk, Doug Beeferman, Françoise Beaufays, Bill Byrne, Ciprian Chelba, Mike Cohen, Maryam Kamvar, and Brian Strope, "“Your word is my command”: Google search by voice: A case study," in Advances in Speech Recognition, pp. 61–90. Springer, 2010.

[2] G. Chen, C. Parada, and G. Heigold, "Small-footprint Keyword Spotting using Deep Neural Networks," in Proc. ICASSP, 2014.

[3] https://research.googleblog.com/2017/08/launching-speech-commands-dataset.html

[4] J.R. Rohlicek, W. Russell, S. Roukos, and H. Gish, "Continuous hidden Markov modeling for speaker-independent wordspotting," in Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 1990, pp. 627–630.

[5] Richard C Rose and Douglas B Paul, "A hidden Markov model based keyword recognition system," in Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 1990, pp. 129–132.

[6] JG Wilpon, LG Miller, and P Modi, "Improvements and applications for key word recognition using hidden Markov modeling techniques," in Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 1991, pp. 309–312.

[7] Marius-Calin Silaghi and Hervé Bourlard, "Iterative posteriorbased keyword spotting without filler models," in Proceedings of the Automatic Speech Recognition and Understanding Work shop (ASRU). IEEE, 1999, pp. 213–216.

[8] Marius-Calin Silaghi, "Spotting subsequences matching an HMM using the average observation probability criteria with application to keyword spotting.," in Proceedings of the National Conference on Artificial Intelligence. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005, vol. 20, p. 1118.

[9] Chen, Guoguo, Carolina Parada, and Georg Heigold. "Small-footprint keyword spotting using deep neural networks." Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on. IEEE, 2014.

[10] David Grangier, Joseph Keshet, and Samy Bengio, "Discriminative keyword spotting," Automatic speech and speaker recognition: large margin and kernel methods, pp. 175–194, 2009.

[11] Shima Tabibian, Ahmad Akbari, and Babak Nasersharif, "An evolutionary based discriminative system for keyword spotting," in International Symposium on Artificial Intelligence and Signal Processing (AISP). IEEE, 2011, pp. 83–88.

[12] KP Li, JA Naylor, and ML Rossen, "A whole word recurrent neural network for keyword spotting," in Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 1992, vol. 2, pp. 81–84.

[13] Santiago Fernández, Alex Graves, and Jürgen Schmidhuber, "An application of recurrent neural networks to discriminative keyword spotting," in Artificial Neural Networks–ICANN 2007, pp. 220–229. Springer, 2007.

[14] O. Abdel-Hamid, A. Mohamed, H. Jiang, and G. Penn, "Applying Convolutional Neural Network Concepts to Hybrid NN-HMM Model for Speech Recognition," in Proc. ICASSP, 2012.

[15] L. Toth, "Combining Time-and Frequency-Domain Convolution in Convolutional Neural Network-Based Phone Recognition," in Proc. ICASSP, 2014.

[16] T. N. Sainath, A. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep Convolutional Neural Networks for LVCSR," in Proc. ICASSP, 2013.

[17] Dave, Namrata. (2013). Feature extraction methods LPC, PLP and MFCC in speech recognition. International Journal For Advance Research in Engineering And Technology(ISSN 2320-6802). Volume 1.

[18] Sainath, Tara N., and Carolina Parada. "Convolutional neural networks for small-footprint keyword spotting." Sixteenth Annual Conference of the International Speech Communication Association. 2015.

[19] http://cs231n.github.io/neural-networks-3/

[20] Hinton, Geoffrey E., et al. "Improving nerural networks by preventing co-adaptation of feature detectors." arXiv preprint arXiv:1207.0580 (2012)