Wake-word Detection for Resource Constrained Hardware using Convolutional Neural Networks

Dillon Pulliam[1]*, Hashim Saeed*[1]

[1]Carnegie Mellon University

jpulliam@andrew.cmu.edu, hashims@andrew.cmu.edu

Abstract

In this work we explore and profile various Convolutional Neural Network (CNN) architectures and their application for wake-word detection engines. We specifically experiment with resource-constrained architectures such as MobileNets, ShuffleNets, and EfficientNets and their usefulness in this task while developing a custom, more efficient, variant of both ShuffleNet and EfficientNet. Model evaluation is performed by measuring F1 scores over the *Google Speech Commands Dataset* which is used for training where a specific word is chosen as the keyword. Finally we implement a novel training pipeline involving various data transformation/augmentation techniques for speech data and measure the effectiveness of each.

Index Terms: wake-word detection, keyword spotting, convolutional neural networks, resource constrained hardware, data augmentation

1. Introduction

Wake-Word (WW) Detection involves an algorithm which monitors a stream of audio signal and activates the system upon recognition of a keyword in it. The wake-word detection problem has been prevalent in Speech Recognition due to its applications in real world systems [1]. Integrating a wake-word system is expensive and takes a lot of time. This makes this unaffordable for upcoming startups who don't possess the resources to create such systems. Furthermore, the way wake-word engines are trained nowadays involves collecting hundreds of random samples of the same "word" by different people in different environments which is both time-consuming and costly.

Keeping in mind these limitations, our work focuses on creating a Wake-Word Detection Framework on a pre-existing publicly available Google Speech Commands Dataset [2] which not only focuses on accurate results but also the computational complexity and hardware resources. In this paper, we look at several preceding Keyword-Spotting and Wake-Word Detection Frameworks and infer the limitations of each. Next, we outline our own experimental setup in which we discuss several transformations such as Amplitude Multiplication, Speech and Pitch Modulation, STFT Spectrogram, Time Shift, and Background Noise Addition which we applied on the Dataset to create the input to our Deep Neural Networks. Furthermore, we observe the performances of various classifiers using our framework and test their F1 scores as well as the size of the model and its inference time. Finally, we propose our novel network design using ShuffleNets [3] and EfficientNets [4] which not only provides high accuracy but also an approximately 20 fold deduction in size of the model and a great decrease in the CPU inference time.

2. Related Work

Before we describe our experimental setup and how we proceeded with our objective of increasing the performance of our network to acceptable standards while decreasing the hardware resources at the same time, first lets look at some of the related literature to this problem as well as some already pre-established Wake Word Detection Systems.

When looking at interactions through current speech recognition devices, wake-word detection is the first step in the process [1]. Many different approaches have been outlined for this task. One of the frameworks consists of using a DNN-HMM encoder at first, followed by a light-classifier to perform the final decision of whether the wake-word was present [5]. The second classifier improves the performance of the HMM based decoder and also preserves computational time. Furthermore, to better distinguish between the wake-word and other noisy segments, monophone based units are introduced to model the non-chosen keywords which further help enhance the classification accuracy [5]. Expanding more upon the two-stage system, the first stage of the new framework involves expanding speech and non-speech segments into monophones. After introducing the new units, the background HMM becomes similar to phone-level unigram FST. The second stage now uses this HMM with more states to compute the *MatchScore* which is the measure of degree of wake-word segments and background monophone.

Another wake-word detection framework more similar to our approach involves using end-to-end Deep NN framework with the absence of HMMs. Most Wake-Word Detectors use signal processing techniques such as DFT and Short Time Fourier Transfrom to obtain compact feature representation such as log-mel filterbank energy (LFBE) [6] which can then be fed to our system [7]. In our case, we are focused towards dealing with raw audio instead of such transforms. One of such primary works which involves audio-input DNN modelling from single channel audio uses Convolutional Neural Networks (CNN) layers with long short-term memory (LSTM) [8]. These convolutional layers model time-frequency feature of audio well and other layers represent temporal characters of speech input.

One of the most significant works in using raw audio as input for Wake-Word Detection system also used in Alexa uses a Time-Delay Neural Network (TDNN) [9]. The basic architechure of the network follows the same design architecture as that of the baseline Wake-Word DNN which uses LFBE Features as input [10] but has some significant differences in the training process. These features, directly from the raw audio, are stacked on top of each other and then fed to the DNN. The new stage-wise training step introduced in [9] significantly improves the accuracy of Wake-Word detection compared to baseline model [10].

In our framework, we delve into the different convolutional

networks that can be used for our task of wake-word detection. The metrics in mind are not only the classification accuracy but also the size and inference time of the network. For this purpose, one of the main networks we looked at was ShuffleNet V2 [3]. With the arrival of CNNs, many networks have been introduced which focus on increasing the classification accuracy of the network which include VGG [11], ResNet[12], etc. However, such networks do not take into account the speed required for a network to infer the result. One of the breakthroughs in this regard comes in the form of ShuffleNet V2 [3] which improves on the concept of ShuffleNet [13] and takes into account the speed of the network as a metric. ShuffleNet was designed for low-end architecture such as mobile phones whereas ShuffleNet V2 strengthens the concept by introducing the concept of Channel Splitting, where initially, the input features are divided into two channels and each of the channel receives only a specific number of features to it. At the end, two branches are concatenated. ShuffleNet V2 performs comparable to the other high end models but with a significant decrease in computational time.

The second network we delved further into was EfficientNet [4]. This work introduced the idea of model scaling and its effect on the performance of the network. The idea behind the framework was to scale various parameters of the network such as depth, width, and resolution and see the effect of compounding various scales of these parameters together on the accuracy and the size of the network. Such compounded scaled networks were named EfficientNets and it was conclusively shown that with proper scaling of these parameters, we can achieve a high accuracy compared to the other traditional CNN's with a reduced size of parameters and thus reduced size of the network itself. This is an important concept which we utilized in our framework for scalability of our model in real-world WW systems.

3. Experimental Setup

3.1. Dataset

To build a wake-word detection engine a variant of the *Google Speech Commands Dataset* [2] is used. This dataset contains 65,000 one-second long utterances of 30 short words spoken by thousands of different people. It also contains various background noises that can easily be added to the utterances to increase the generalization of results as the utterances themselves were collected in a silent room.

For these experiments we select a single word as the wake-word. Primary experimentation is done using "Marvin" as the keyword. In order to spit the data 50-50 in terms of wake-word and non-wake-word examples we use all examples of "Marvin" from the training set as positive examples. For negative examples 10% of this is silence. The remaining 90% is selected by randomly choosing one of the other 29 words from the data set and then randomly choosing a training sample. This same process is repeated for building the validation and test sets.

In order to augment the training set and achieve better wake-word detection generalization we apply a number of transformations to the data. These transformations are applied to the training set only and are described in more detail in the following section. Since each transformation is applied with some randomly probability we augment the training data by applying different transformations across the same sample. Thus each true wake-word example is used five times in the training set; negative examples are selected randomly.

All in all using the *Google Speech Commands* data to build a "Marvin" wake-word detection engine ended with 14,240 training set samples, 320 validation set samples, and 324 test set samples. Each is split 50-50 in terms of wake-word non-wake-word examples.

3.2. Transformations

To increase model generalization and augment the data set a number of transformations are applied to training data only as suggested by [14]. Two of these transformations are applied directly to WAV files while the other three are applied after computing the Short-Time Fourier Transform (STFT) spectrogram of the audio. Note that each transformation is applied at a 50% probability individually, thus it is possible that all or none of these transformations are applied to a single audio file.

3.2.1. Amplitude Multiply (WAV file)

Audio samples multiplied by some value between 0.7 and 1.1

3.2.2. Speed and Pitch Modulation (WAV file)

Audio clip scaled by some factor between 0.833 and 1.25

3.2.3. Frequency Domain Stretch (STFT spectrogram)

Spectrogram frequency scaled between 0.8 and 1.2 using Librosa's phase vocoder

3.2.4. Time Shift (STFT spectrogram)

Spectrogram shifted either left or right between 0 and 8 frames

3.2.5. Background Noise Addition (STFT spectrogram)

Background noise randomly sampled, STFT spectrogram computed, noise added according to:

$$y = (1 - \lambda)x + \lambda n \tag{1}$$

x is the original audio spectrogram, n is the noise spectrogram, λ is some factor between 0 and 0.45, and y is the resulting spectrogram

3.3. Training Pipeline

The data transformation results described are pre-processed and saved as PyTorch [15] tensors to increase the speed of model training and avoid application in real-time. The training pipeline setup is to first read in the WAV file using Librosa [16] as a NumPy [17] array. After the audio file is read transformations *3.2.1* and *3.2.2* are applied directly to the WAV file before fixing the audio length to 1-second by cutting or padding.

Once the audio length is fixed the STFT is computed using Librosa across 2048 samples with a hop of 512. Transformations *3.2.3* and *3.2.4* are then applied to the STFT spectrogram. After applying these transformations the STFT data dimensionality is corrected based on if stretching or time-shifting was used. Finally transformation *3.2.5* is applied before computing the Mel Spectrogram using Librosa and saving the result as a PyTorch tensor which can then be input to the model.

3.4. CNN Architectures

Profiling is performed across a number of different Convolutional Neural Network (CNN) architectures from larger scale

networks to more optimized hardware efficient ones. The input to each model is a 32x32 image of the Mel Spectrogram of the audio. Note that many of the CNN architectures profiled were originally designed for ImageNet [18] like data, 3-channel 256x256 images, thus there is a lot of efficiency lost during computation as our input is significantly smaller. Additionally each model's final linear layer is modified from 1000 output units (for ImageNet) to 2 output units representing a score value for true/false wake-word detection.

The first architectures profiled were VGG [19] and ResNet [20]. The specific variant of each of these architectures tested were VGG-19 with Batch Normalization and ResNet-18. These architectures were used to get a baseline performance while developing and training more hardware efficient CNN models.

After profiling these models a number of more highly optimized architectures were evaluated. These include MobileNetV2 [21], ShuffleNet V2 [22] and its variants, as well as EfficientNet [23] and its variants. A custom version of ShuffleNet V2 was also built and evaluated with a stage-in factor of [2,2,2] and stage-out factor of [12,24,48,96,512]. Likewise a custom EfficientNet architecture was built and tested with a width, depth, residual, dropout multiplier of 0.1, 0.1, 32. and 0.2 respectively.

3.5. Model Training and Evaluation

All CNN architectures are trained by minimizing the Cross-Entropy loss between the true and predicted wake-word. This loss function can be described as follows:

$$Loss = -\sum_{i=1}^{C} t_i * log(f_i(a_)) \tag{2}$$

Here C is the number of classes, in this case two (wake-word/non-wake-word), t is a one-hot vector indicating whether or not the audio is a wake-word example, f is the model, and a is the audio sample.

For architecture evaluation precision, recall, and F1 score are measured. These are more informative and important metrics for this task compared to accuracy. Precision is defined as the fraction of words detected that are true wake-words while recall is the fraction of true wake-words detected from the total instances. The F1 score can then be computed from precision and recall as follows:

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{3}$$

4. Results

4.1. Model Training Results

A comparison of VGG-19, ResNet-18, MobileNetV2, ShuffleNet V2 (x0.5 version) and EfficientNet (B0 version) over training can be seen in *Figure 1*. Note that training is performed over 75 epochs using stochastic gradient descent with a batch size of 64, weight decay of 0.01, and an initial learning rate of 0.0001 that is decayed at plateaus by 0.5 for all models. Results are the average validation set F1 score across five runs of training.

As can be seen hardware efficient Convolutional Neural Network architectures perform nearly as well as large-scale ones due to the simplicity of the input and wake-word detection task.

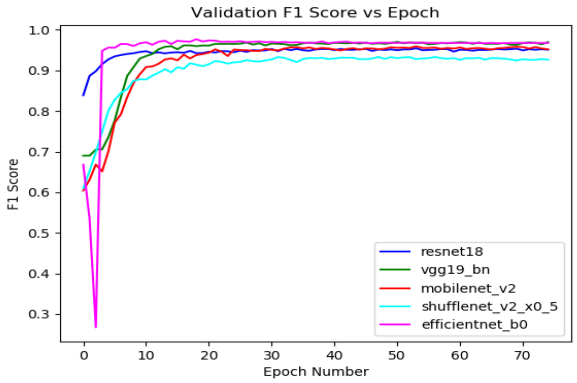

Figure 1: *Validation F1 score comparison.*

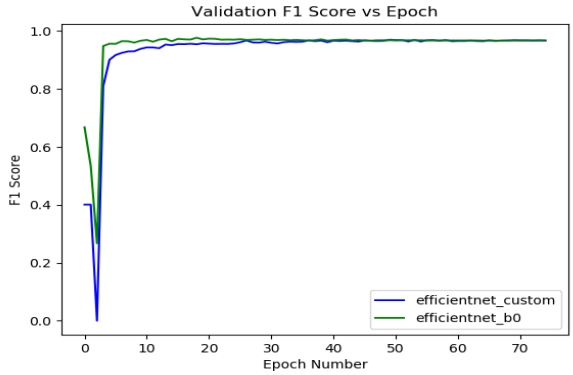

Figure 2: *EfficientNet F1 score comparison.*

4.2. Custom ShuffleNet/EfficientNet Results

In *Figure 2* we compare the training results of EfficientNet_B0, the smallest variant of EfficientNet originally published, with our custom EfficientNet designed for wake-word detection. Note that EfficientNet_B0 contains 4.010 million parameters (16.3MB) while the custom version only has 0.033 million (0.203MB). This is a reduction of over 80x for virtually the same performance showing that consistent model scaling also holds as model size is decreased.

Results are similar in the case of a comparison of the x0.5 version of ShuffleNet V2 with our custom ShuffleNet V2 variant. Here we decrease the parameter count of the network by nearly 5x from 0.343M (1.5MB) to 0.070M (0.346MB) with virtually the same performance.

4.3. Test Set Results

Table 1 contains the test set results from the various CNN architectures profiled. Here the model size, CPU inference time for a single 1-second long audio utterance, and test set F1 score are reported. Note that larger variants of EfficientNet (B2, B3, etc.) as well as ShuffleNet are not reported. Also note that CPU inference times are measured using an Intel Core i7-8750H - 2.20 GHz processor with 8GB of system RAM.

The custom version of EfficientNet implemented is actually one of the better models trained receiving a test set F1 score of 0.9527. In fact the only models trained that performed better were the larger-scale EfficientNet version, B0 (0.9659) and B1 (0.9598).

Table 1: *Test results*

Model	Size	CPU Time	F1 Score
Custom E-Net	0.203MB	7.805ms	0.9527
Custom S-NetV2	0.346MB	4.635ms	0.9297
S-NetV2 x0.5	1.5MB	15.892ms	0.9231
S-NetV2 x1.0	5.2MB	37.843ms	0.9292
MobileNetV2	9.1MB	119.021ms	0.9354
E-Net B0	16.3MB	162.532ms	0.9659
E-Net B1	26.5MB	246.932ms	0.9598
ResNet-18	44.8MB	6.466ms	0.9448
VGG-19	155.8MB	25.300ms	0.9477

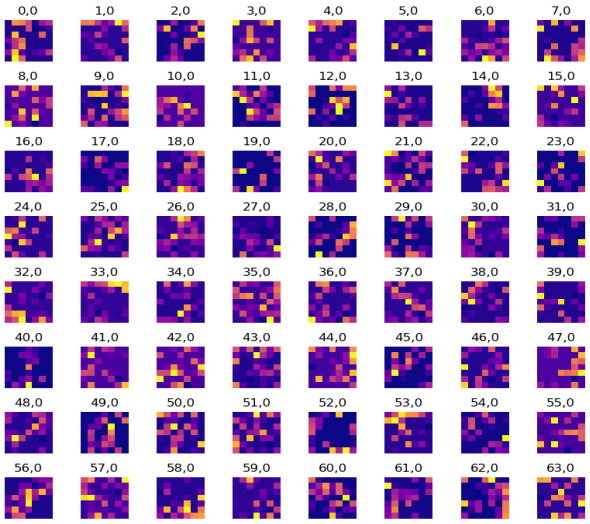

Figure 3: *ResNet-18 filters.*

These results are interesting because CNN detector parameter counts in [24] range from 20.0k to 244.2k while HMM-DNN phone-based detectors in [25] run from 3.02M to 3.99M. Our EfficientNet-based CNN architecture contains 33.0k, in the same ballpark range or smaller while still receiving good performance.

4.4. CNN Filter Visualization

A visualization of ResNet18's first layer filters can be seen in *Figure 3*. This is visualized as ResNet18 uses 7x7 filters in layer 1 whereas all other networks use 3x3. One interesting filter is 56 which appears to place significance on the $x = y$ diagonal of Mel spectrograms.

In *Figure 4* a visualization of the 3x3 filters in our custom EfficientNet model can be seen.

4.5. Wake-Word Generalization

To ensure performance wasn't dictated on a specific wake-word, in this case "Marvin", the same training pipeline was performed using four others words from *Google Speech Commands* as the true wake-word. Test set results (F1 score) across the custom EfficientNet and EfficientNet_B0 models can be seen in *Table 2*.

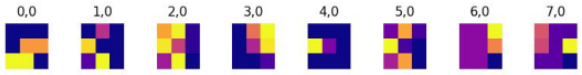

Figure 4: *Custom EfficientNet filters.*

Table 2: *Wake-word F1 scores*

Wake-Word	Custom E-Net	E-Net B0
Cat	0.9527	0.9645
Four	0.9570	0.9621
Happy	0.9669	0.9806
Marvin	0.9527	0.9659
Three	0.9436	0.9346

4.6. Data Transformation/Augmentation Effects

Here we analyze the effect of each transformation described in *Section 3.2*. Data transformations are applied by themselves at 50% probability five times across the data to build the training set as well as no transformations being applied at all. Analysis is also performed on the effect of data augmentation when all transformations are applied a single time to build the training set. Results can be seen in *Table 3* where test set F1 scores are reported.

Table 3: *Data Transformation/Augmentation F1 scores*

Transformation/Augmentation	Custom E-Net
All Transformations (5 runs)	0.9527
All Transformations (1 run)	0.9012
No Transformations (5 runs)	0.9438
Amplitude Multiply Only (5 runs)	0.9397
Speed/Pitch Mod. Only (5 runs)	0.9565
Frequency Stretch Only (5 runs)	0.9530
Timeshift Only (5 runs)	0.9438
Background Noise Only (5 runs)	0.9256

5. Conclusions and Future Work

In this paper, we demonstrate a novel framework for a Wake-Word Detection System and its effectiveness compared to other renowned Neural Networks. Our experimental results conclusively show that by using our modified CNN we can reduce the hardware resources including the size of the model as well as the inference time on CPU while keeping the F1 score of the network as high as possible. From our experiments, the best model in terms of hardware resources and generalization is **Custom EfficientNet** which has a size of 0.203 MegaBytes, CPU Inference Time of 7.805 ms, and an F1 Score of 0.9527.

To deploy this system in the real world, further improvements in the framework can be made. One way to make this framework more robust is by introducing more negative samples compared to the 29 samples we used in our experiment. Having real-time audio samples for various instances as negative samples can make our classifier more suited for real-world applications. Profiling the trained network in real time and evaluating the F1 Score from the live version of our framework can help us to further assess the quality of our classifier and how we can fine-tune it further to make it deployable.

6. References

[1] K. Kumatani, J. McDonough, and B. Raj, "Microphone array processing for distant speech recognition: From close-talking microphones to far-field sensors," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 127–140, 2012.

[2] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *CoRR*, vol. abs/1804.03209, 2018. [Online]. Available: http://arxiv.org/abs/1804.03209

[3] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 116–131.

[4] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

[5] M. Wu, S. Panchapagesan, M. Sun, J. Gu, R. Thomas, S. N. P. Vitaladevuni, B. Hoffmeister, and A. Mandal, "Monophone-based background modeling for two-stage on-device wake word detection," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5494–5498.

[6] M. Sun, A. Raju, G. Tucker, S. Panchapagesan, G. Fu, A. Mandal, S. Matsoukas, N. Strom, and S. Vitaladevuni, "Max-pooling loss training of long short-term memory networks for small-footprint keyword spotting," in *2016 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2016, pp. 474–480.

[7] M. Sun, B. Hoffmeister, S. N. P. Vitaladevuni, and V. K. Nagaraja, "Model shrinking for embedded keyword spotting," Mar. 21 2017, uS Patent 9,600,231.

[8] T. N. Sainath, R. J. Weiss, A. Senior, K. W. Wilson, and O. Vinyals, "Learning the speech front-end with raw waveform cldnns," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[9] K. Kumatani, S. Panchapagesan, M. Wu, M. Kim, N. Strom, G. Tiwari, and A. Mandai, "Direct modeling of raw audio with dnns for wake word detection," in *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2017, pp. 252–257.

[10] M. Bhargava and R. Rose, "Architectures for deep neural network based acoustic models defined over windowed speech waveforms," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[13] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[14] R. Tang and J. Lin, "Honk: A pytorch reimplementation of convolutional neural networks for keyword spotting," *CoRR*, vol. abs/1710.06554, 2017. [Online]. Available: http://arxiv.org/abs/1710.06554

[15] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[16] B. McFee, V. Lostanlen, M. McVicar, A. Metsai, S. Balke, C. Thomé, C. Raffel, D. Lee, F. Zalkow, K. Lee, O. Nieto, J. Mason, D. Ellis, R. Yamamoto, E. Battenberg, , R. Bittner, K. Choi, J. Moore, Z. Wei, S. Seyfarth, nullmightybofo, P. Friesch, F.-R. Stöter, D. Hereñú, Thassilo, T. Kim, M. Vollrath, A. Weiss, and A. Weiss, "librosa/librosa: 0.7.1," Oct. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3478579

[17] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, "SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python," *arXiv e-prints*, p. arXiv:1907.10121, Jul 2019.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[21] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018. [Online]. Available: http://arxiv.org/abs/1801.04381

[22] N. Ma, X. Zhang, H. Zheng, and J. Sun, "Shufflenet V2: practical guidelines for efficient CNN architecture design," *CoRR*, vol. abs/1807.11164, 2018. [Online]. Available: http://arxiv.org/abs/1807.11164

[23] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019. [Online]. Available: http://arxiv.org/abs/1905.11946

[24] T. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Interspeech*, 2015.

[25] M. Wu, S. Panchapagesan, M. Sun, J. Gu, R. Thomas, S. Vitaladevuni, B. Hoffmeister, and A. Mandal, "Monophone-based background modeling for two-stage on-device wake word detection," 04 2018, pp. 5494–5498.