

Keyword Spotting on Arduino Nano 33 BLE Sense

Aidan Cowan

Dept. of Electrical and Computer Engineering
University of North Carolina at Charlotte
Charlotte, NC 28223
acowan8charlotte.edu

Hunter Burnett

Dept. of Electrical and Computer Engineering
University of North Carolina at Charlotte
Charlotte, NC 28223
hburnet7@charlotte.edu

GitHub Repository: https://github.com/holleman-classes/proj2_kws-aidan-c/tree/main

Abstract—This paper presents the development and evaluation of a keyword-spotting system for the Arduino Nano 33 BLE Sense micro-controller. The system is designed to recognize two target words: the custom word "imposter" and the word "down" from the Speech Commands dataset. We explore different neural network architectures, including convolutional neural networks (CNNs) and long short-term memory (LSTM) networks, to find the most suitable model for this application. The models are trained on a dataset consisting of the mini speech dataset combined with custom recorded audio for the word "imposter". Various data augmentation techniques are applied to increase the diversity of the training data. The trained models are then deployed and evaluated in a streaming context on the Arduino device. The CNN model achieves an accuracy of 93.4

Index Terms—keyword spotting, Arduino, CNN, LSTM, embedded systems

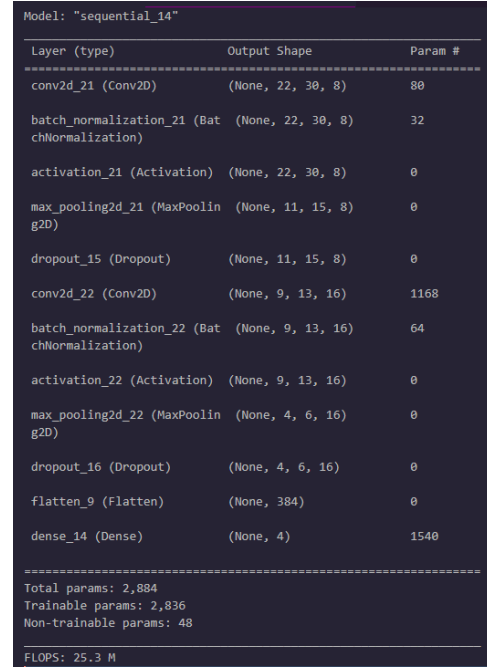
I. INTRODUCTION

This project aims to train and deploy a keyword-spotting algorithm using the Arduino Nano 33 BLE Sense. The system will be designed to recognize at least two words, one of which will be the custom word "imposter". The project will collect training data, train the model, and deploy it in a streaming scenario to detect the target words in a continuous audio stream.

II. MODEL ARCHITECTURE

We chose the topology of our neural network by considering models that could recognize two target words from a custom dataset and be deployed to an embedded device with limited resources. The network size was constrained by the number of parameters and runtime due to the limited memory and the window length for capturing audio on the Arduino Nano 33 BLE. If the models took too long to infer, it would produce a delayed response, increase false positives due to processing older audio that doesn't match the current environment or user speaking into the microphone, and it would no longer be 'real-time'.

The input features were spectrograms from audio files. These spectrograms were generated by applying the Fast Fourier Transform to the audio wav files and then converting the frequency domain representation into a 2D image. This representation could be processed by both the convolutional neural network and the long-short-term memory network.



Layer (type)	Output Shape	Param #
conv2d_21 (Conv2D)	(None, 22, 30, 8)	80
batch_normalization_21 (Batch Normalization)	(None, 22, 30, 8)	32
activation_21 (Activation)	(None, 22, 30, 8)	0
max_pooling2d_21 (MaxPooling2D)	(None, 11, 15, 8)	0
dropout_15 (Dropout)	(None, 11, 15, 8)	0
conv2d_22 (Conv2D)	(None, 9, 13, 16)	1168
batch_normalization_22 (Batch Normalization)	(None, 9, 13, 16)	64
activation_22 (Activation)	(None, 9, 13, 16)	0
max_pooling2d_22 (MaxPooling2D)	(None, 4, 6, 16)	0
dropout_16 (Dropout)	(None, 4, 6, 16)	0
flatten_9 (Flatten)	(None, 384)	0
dense_14 (Dense)	(None, 4)	1540

Total params: 2,884
Trainable params: 2,836
Non-trainable params: 48
FLOPs: 25.3 M

Fig. 1. CNN Model Plot

We performed various experiments to determine the best model approach to use. Multiple convolutional layers were tested with minimal dense layers and vice versa for the convolutional model. Originally, we used three conv layers with batch normalization and dropout followed by two dense layers which results in test accuracy of 89% however the inference time when deployed was terrible. For the LSTM model, multiple LSTM layers were tested with varying dense layers. On top of experimenting with different layer amounts, dropout, and batch normalization were applied to see if it made any difference to inference without compromising the runtime. It was concluded that applying dropout after each layer increased the validation accuracy and stabilized it, however applying dropout with a probability higher than 30% to the dense layers staggered the validation accuracy greatly and never learned.

- A. CNN Model Plot
- B. CNN Model Architecture
- C. LSTM Model Plot
- D. LSTM Model Architecture

III. DATA AND TRAINING

For this project, the primary dataset used was the mini speech dataset. This dataset contains 9 different keywords, including the custom word "imposter" and the chosen word "down" from the Speech Commands dataset. Each keyword has 1000 one-second audio WAV files, for a total of 9,000 files in the dataset. The data for the custom word 'imposter' was gathered by recording multiple speakers saying the target words and applying various data augmentation techniques to increase the diversity of the training data. To balance the training dataset, the number of samples for each non-target command (i.e., all commands except "imposter" and "down") was limited to 1000 samples. This ensured that the target commands had an equal number of samples compared to the non-target commands, resulting in a balanced training dataset.

To ensure the training data covered natural variation, the custom "imposter" WAV files were collected by recording multiple speakers from diverse backgrounds. This helped to introduce variation in accent, voice, and environment into the dataset.

The data augmentation process involved the following steps:

- 1) Normalizing the audio data to the range [-1, 1].
- 2) Applying time shifting to the audio clips to create longer and shorter versions.
- 3) Applying pitch shifting to create higher and lower pitch versions.
- 4) Trimming the silent parts of the audio clips.
- 5) Ensuring all audio clips were of the same length (1 second) by padding or trimming as necessary.

We did not use any specific training techniques such as distillation, transfer learning, or synthetic data generation nor did we estimate how much data we would need. The focus was on using a subset of the public Speech Commands dataset, combined with the custom 'imposter' audio files and various data augmentation techniques, to train the keyword spotting model.

IV. RESULTS

The CNN model when deployed in a streaming context performs better than expected as it does respond to the target words however does not accurately detect silence, this is due to the silence generated during training is quieter than the background silence/ambience of the room we tested in. It also does exhibit false positives on a few other sounds such as near-homophones, delay in speech, and certain annunciations of words. The inference time took 34.491 ms and the sampling rate was 29 frames per second.

A. Summary Table

B. Model Comparisons

Overall the CNN model we ended deploying had a decent accuracy when training, testing and evaluating. When calculating false alarms per hour we played Hit 'Em Up by 2Pac as it was a 5 minute 12 second song and it covered various sound levels. We recorded 35 false alarms over the 5 minute period which is calculated to be 420 false alarms per hour. We found that the model would trigger on the "st" portion of a word that was not "imposter". We found similarities in the start of the spectrograms between imposter and words that have a phrase that starts with "st". We also found that the model could not recognize silence at all. You can see a reference to the spectrograms in Figure 5 and Figure 6. This is because we did not include any ambient noises in our training so it just recognizes silence as "unknown" This can be resolved by recording the ambient sound of a quiet room and including this in our dataset.

It would also be helpful to have a larger dataset so we can build a more generalized model. The word imposter was sampled from YouTube videos manually so it was hard to get a large dataset for our chosen wake word. We found that we would have to really enunciate the word "imposter" for our model to recognize the wake word. We chose "Down" to be our other wake word which was from the Speech Commands dataset. We found that the model was able to detect this word better. This could be due to the fact that the Speech Commands dataset includes a wide range of voices than our custom dataset.

V. CONCLUSION

When creating a wake word detection model it is important for us to consider a wide range of factors. We need a well rounded dataset for our wake word that includes a balance between female and male voices so the model is not biased one way or the other. We also found that the CNN model performs better when detecting wake words than the LSTM model. The CNN has many advantages over the LSTM in this use case. The CNN is well suited for fixed-length input frames and can efficiently process audio. The CNN also results in a smaller size which is very important in TinyML applications as we are working with a small amount of memory.

REFERENCES

- [1] Shawn Hymel, "Custom Wake Word Part 1: Capturing Data — TinyML," Uploaded June 30th, 2020, YouTube video, <https://www.youtube.com/watch?v=DdFBMtq2jso>
- [2] jschlattLIVE, "I went on an Among Us date.", Uploaded October 16th, 2020, YouTube video, https://www.youtube.com/watch?v=VAQgB6pMEI&t=462s&ab_channel=jschlattLIVE
- [3] WilburSoot, "Wilbur Soot VOD (Oct 12th) - among us with celebrities and fundy", Uploaded October 12th, 2020, Twitch VOD, https://www.youtube.com/watch?v=uiCiltj2YVE&list=PLtP_4y041z4jCrbVhJFcWoXGeItYxlRX&ab_channel=LittleBitOfEverything
- [4] Jawsh unofficial Vods, "Among Us VR [Jawsh Among us VR Vod]", Uploaded November 10th, 2022, Twitch VOD, https://www.youtube.com/watch?v=tnMPtGnSoss&t=5s&ab_channel=JawshunofficialVods

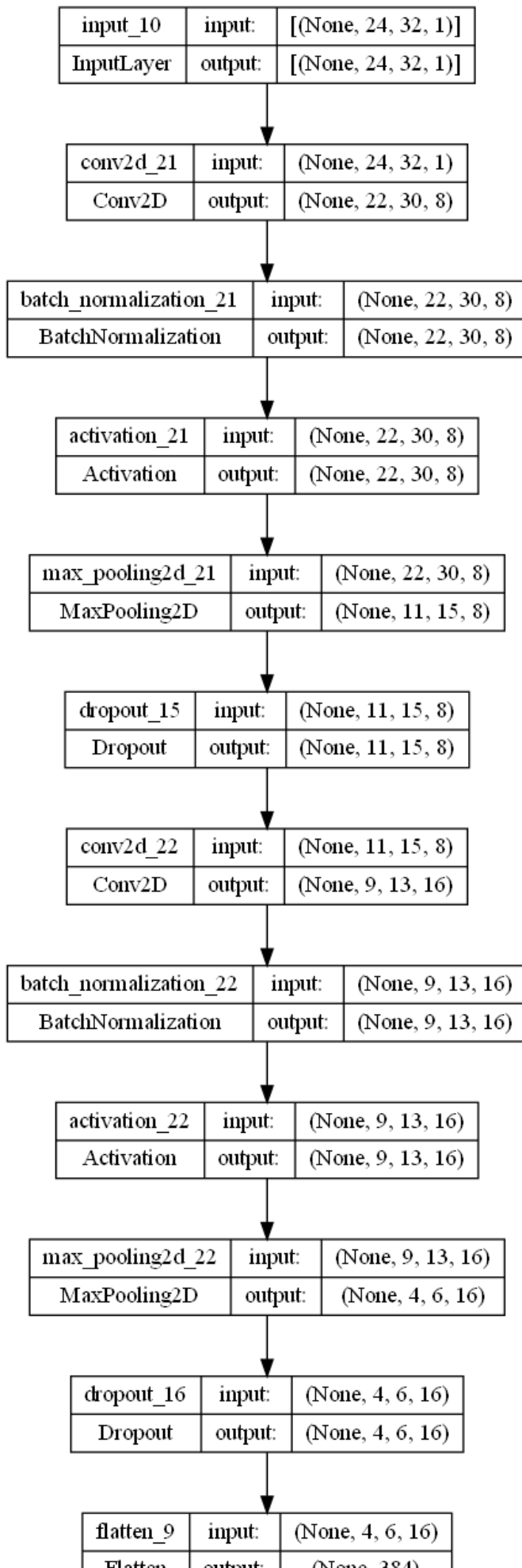
TABLE I
CNN SUMMARY TABLE

Model	Accuracies (Train, Val, Test)	False Rejection Rate for 'imposter'	False Rejection Rate for 'down'	Parameters	MACs	Input Shape
CNN	92.36%, 90.75%, 93.4%	0.010989	0.0686275	2,884	25.3 megaFLOPS	(24, 32, 1)
		Sampling Rate (FPS)	False Alarm Rate (FA/hour)			
		29 frames per second	420 false alarms per hour			

TABLE II
LSTM SUMMARY TABLE

Model	Accuracies (Train, Val, Test)	False Rejection Rate for 'imposter'	False Rejection Rate for 'down'	Parameters	MACs	Input Shape
LSTM	72.80%, 87.93%, 51.2%	0.321	0.987	198,660	0.0671 M	(24, 32, 1)
		Sampling Rate (FPS)	False Alarm Rate (FA/hour)			
		-	-			

- [5] Poki's VODs,"Pokimane VOD—Among Us with valkyrae, toast fuslie and more amazing people", Uploaded October 30th, 2022, Twitch VOD, https://www.youtube.com/watch?v=_ame_-mH7iA&ab_channel=Poki%27sVODs
- [6] Benedikt Strobel, "YTKS.app", YouTube Keyword Seacher, Benedikt Strobel c/o COCENTER Koppoldstr. 186551 AichachGerman, <https://ytk.s.app/>
- [7] 2Pac, Outlawz, "Hit Em Up", Death Row Greatest Hits. Death Row Records, November 26, 1996



Model: "sequential_13"		
Layer (type)	Output Shape	Param #
reshape_4 (Reshape)	(None, 768, 1)	0
lstm_8 (LSTM)	(None, 768, 64)	16896
lstm_9 (LSTM)	(None, 64)	33024
dense_13 (Dense)	(None, 4)	260
Total params: 50,180		
Trainable params: 50,180		
Non-trainable params: 0		

Fig. 3. LSTM Model Plot

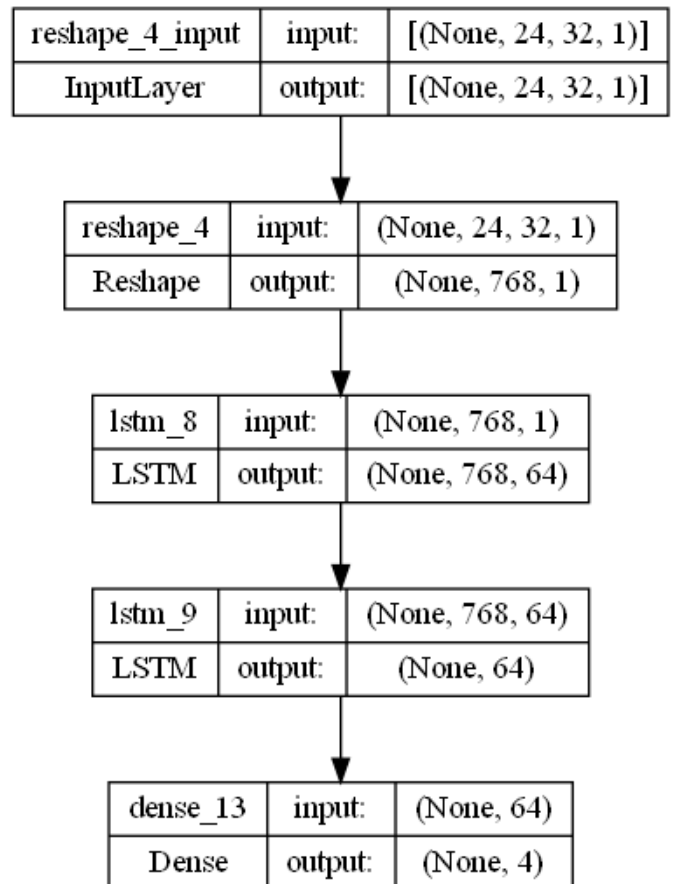


Fig. 4. LSTM Model Architecture

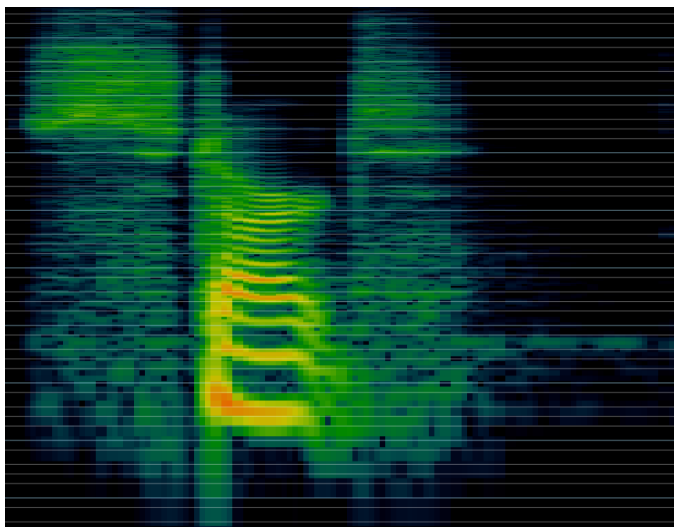


Fig. 5. Spectrogram of "Start"

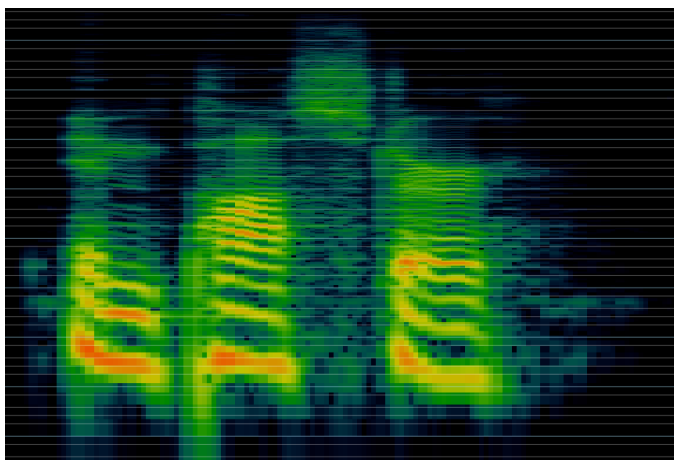


Fig. 6. Spectrogram of "Imposter"