Wake Word Decection via Transfer Learning

Burchill
University of Florida
Gainesville, United States of America
lburchill@ufl.edu

Heffernan
University of Florida
Gainesville, United States of America
shaunheffernan@ufl.edu

Gelli
University of Florida
Gainesville, United States of America
yuri.gelli@ufl.edu

I. Abstract

This report is a culmination of experiments carried out to train a model to detect the presence of the wake words “Go Gators” in audio samples with a variety of background noises. There is a total of 450 samples with hidden, easy, samples reserved for testing. Of the 450 total samples, 360 were available for training. Each researcher conducted their own experiments in pursuit of finding the final model, and their findings will be covered in the experiments section. The implementation section will focus on the development of the final submitted model to be used for testing.

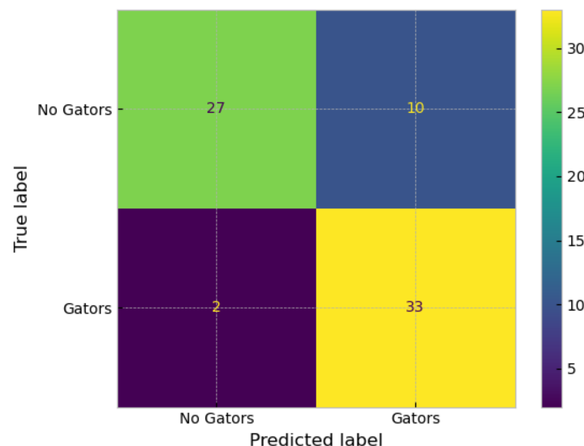
II. Introduction

Each member conducted individual experiments leading to the derivation of the final model. Both convolutional neural networks and auto encoders paired with classifiers were investigated as potential solutions. Additionally, Google Speech commands from the TensorFlow dataset were investigated as a way to train an autoencoder which would later be frozen and used in a classifier. Both up sampling and down sampling were investigated as methods to handle the different frequencies of the training data and Google dataset. l1 and l2 regularizers as well as well as dropout were investigated for reducing overfitting and overall simplification of the model. Experiments were also conducted for both convolutional neural networks and autoencoders examining the benefits of reducing the decision threshold.

III. Experiments

A. *Experiments conducted by Landon*

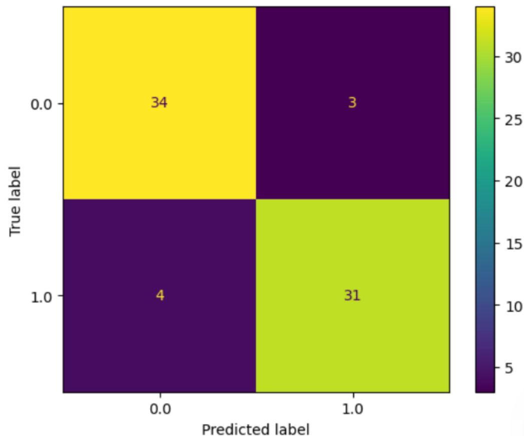
A series of experiments were performed across a convolutional neural network with a random initialization and a one with an initialization derived from an auto encoder. To frame the basis of the experiments both models when initially ran showed signs of overfitting and a produced more false negatives than positives. The first experiment performed was comparing both l1 and l2 regularizers in both models, this was an attempt to address the models overfitting. Both models performed worse when using a l1 regularizer as it was setting weights to zero and losing useful information completely. The scores slightly improved when using a l2 regularizer with a value of .01. At this point it was obvious both models were still significantly overfit earning accuracies in the high 90s in training but only high 70s to low 80s in validation. In order to address this, the size of the dense layer was decreased from 256 to 32, with 128 and 64 tried as well. The highest accuracy achieved in training fell for both models, however the f1 scores on the validation set increased for both. A dense layer of size 128 performed best for the first model, and a dense layer of size 32 performed best for the second model. After reducing the size of the dense layer and applying the l2 regularizer, both models had roughly double the number of false negatives compared to false positives still. To address this multiple threshold values below .5 were tried. A threshold of .4 performed best for the first model with a f1 score of .8. The second model earned f1 scores between .86 and .88 with thresholds of .4, .45 and .5. At .5 the f1 score was .857, at .45 it was .861, and at .4 it was .877. Lowering the threshold decreased the number of false negatives significantly while only slightly hurting the number of false positives. At this point it became clear that the second model, with an initialization based on the auto encoder, was the better performing model. Through these experiments it was observed that a smaller dense layer and use of a l2 regularizer helped the model become less overfit. Additionally running these initial experiments on two different models helped determine the best approach, not just improve the parameters.



B. *Experiments conducted by Shaun*

I first wanted to try transfer learning, so I found a dataset from tensorflow that had a lot of command words which are similar to detecting wake words. The first problem I ran into was the audio samples being 16khz instead of 48khz like our recorded data was. So I originally upsampled the google speech commands to all be 48khz and trained the

encoder that way. I then froze the encoder and attached a classifier trained on the wake words. I got f1 scores in the 60s for the validation set, and when adding drop out and regularization it still was overfitting on the training and had low validation f1 scores. I decided to try downsampling the wake words to 16khz so they could match the google libraries because it seemed my upsampling was just adding noise and not very good. I trained the encoder on the full google data set and attached classifier with softmax for each class to validate the encoders embeddings. After training the encoder on the google data set, I froze the weights and attached a classifier head for the wake word detection. I had a lot of overfitting again, so I added 2 dropouts of 0.5 and l2 regularization of 0.001. This made the model much more generalizable and the validation f1 score higher. I wanted to try and fine tune the model so I trained it with the encoder weights unfroze, which performed significantly better then when frozen. But after adjusting regularization each run started to perform worse and I couldn't figure out why. I realized that since the weights were unfrozen they were getting modified and it takes very long to retrain the encoder every time. So I realized I could save the encoder weights and load them in, so every time I adjust the wake word classifier head I do not need to retrain the encoder. This was the performance on the fine tuned model with unfrozen weights on the held out test set below.



C. *Experiments conducted by Yuri*

To make the model more realistic, we tested background-noise augmentation using our own recorded noise. We recorded a continuous 30-s background clip at the same sampling rate as the dataset (48 kHz). During recording, we played several different background audios from YouTube on speakers and also made additional room noise, so the recording captured mixed real-world conditions.

At first, we applied this augmentation to the entire dataset to expand it. However, this produced near $F_1 \approx 1$ behavior, which was a strong red flag for data leakage. After detecting this issue, we corrected the pipeline by applying augmentation to the training split only, while keeping validation and test sets untouched.

Even after fixing leakage, performance still dropped slightly compared to the non-augmented setup. Since the improvement in robustness did not outweigh the marginal decrease in metrics, we removed this augmentation strategy in the final submission.

We also tested a signal-cleaning step where low-amplitude samples were forced to zero based on an absolute-value sensitivity threshold (i.e., values with $|x| < \tau$ were set to 0). The goal was to suppress low-level background fluctuations and keep only stronger waveform components. τ = 0.10, 0.05, 0.03, .01 were tested.

In practice, this thresholding removed weak signal details that were still useful for classification, especially in quiet or transitional parts of speech. After training with this preprocessing, we observed worse model performance (including degraded overall detection quality) in all tested τ values, so this method was not kept in the final pipeline.

IV. Implementation

The model works by creating an encoder trained on a large dataset called google speech commands. Then transfer learning is used to attach a classifier head and fine tune the model with the wake word training dataset. We chose the google speech commands dataset because they were relatively short audio clips and contained words similar to our wake word. So this encoder would be able to learn features such as noise and words being spoken. To improve this model we added a Mel Spectrogram layer which better approximates human speech frequencies by compressing high frequencies and expanding lower ones. This dataset is only in 16Khz so we decided that instead of upsampling it, we would downsample the training dataset from 48Khz to 16Khz because human voice recognition is still good on that frequency range. The CNN itself is composed of 3 convolution blocks made up of a convolution and a pooling. The convolutions use a small kernel size of 3 because we wanted to get small features out and not lose on any relational information because speaking commands are close together in time. The filter count starts at 32 and doubles until the final convolution block at 128. This is because the pooling layer is downsampling the data each time, so when combined with more filters each block in the CNN will be able to learn more abstract features. So the first block would learn closely relational features; the deeper blocks could learn relations farther apart in the spectrogram. To train this classifier the google speech dataset is used and then an attached classifier is added. There is a dense layer of 500 to understand the learned weights which is then surrounded by a dropout of 0.3 to add regularization. Then a dense layer of 12 with softmax is used to classify each speech word belonging to each of its classes. Once fitting this model on the speech commands using the attached classifier, the weights of the encoder are saved and ready to be fine tuned.

For the wake word detection we take this encoder and attach a classifier head. It has a dense layer of 32 because we want the final prediction to not be made off of a lot of noise, and that dense layer is surrounded by dropouts of 0.5. There is heavy dropout because we have such a small training set so we want it

to be as generalizable as possible. After that the final dense layer has 1 output with sigmoid, to output the probability of the data being a wake word. We used the threshold of 0.5 throughout training and validation, but that threshold can be tuned in test.ipynb depending on whether you prioritize false positives or false negatives.

V. Conclusion

The process of building the final model is outlined in the implementation section. Throughout the development of the final model multiple experiments lead to useful insights. Each team member identified issues with models overfitting in their respective experiments. The use of l2 regularizers and smaller dense layers helped address this. Additionally, 48khz audio data is incredibly dense, downsampling the 48khz data to match the 16khz from the library ended up performing significantly better then upsampling the library to 48khz, which makes sense as that adds significant noise. Additionally multiple experiments led to increasing dropout to reduce overfitting. Even after downsampling the model was still significantly overfit and these methods to reduce overfitting still proved necessary. Most of these experiments were conducted with the goal of reducing overfitting, which is especially important when creating a model on a hidden test set. The final submitted model, to be used in testing, was developed and validated with a train test split of the provided data, however the final model to be used is trained on the entirety of the dataset provided for development.

VI. References

[1] Google Speech Commands, "speech_commands," TensorFlow. [Online]. Available: https://www.tensorflow.org/datasets/catalog/speech_commands. [Accessed: April. 18, 2026]