Smithers

Final Presentation

Jacob Jones, ECE 551

Outline

- Introduction – *What's this all about?*
- Motivation – *Why am I doing this?*
- Background – *Contemporary methods*
- Method – *How I did this*
- Results – *Project results*
- Proposed Research – *Further improvements*
- Conclusions – *Wrap up this presentation*

Introduction

- *Smithers* is a machine learning model used to detect a wake word
- This model uses mel-frequency cepstral coefficients and a convolutional neural net to make predictions on real-time audio
- This presentation reports on the progress made towards this effort

Motivation

- Smart devices are everywhere
- Smart devices respond to voice commands
- Three big ones:
  - Google Assistant, Alexa, Siri
- I am interested in embedded devices
  - How does voice recognition work on resource-constrained hardware?
- Are these devices always listening?
  - Short answer: No
  - Long answer: Sort of…

Background

- Are these devices always listening?
- How do you respond to commands without always listening?
  - Voice command recognition too computationally intensive
- Amazon, Apple, and Google have researched this topic extensively
- Common solution: “wake words”
- Idea: train model to recognize only one command
  - “Hey Alexa”, “Hey Siri”, “Hey Google”
  - Once command is recognized, activate larger model
- Some devices place model on separate dedicated processor

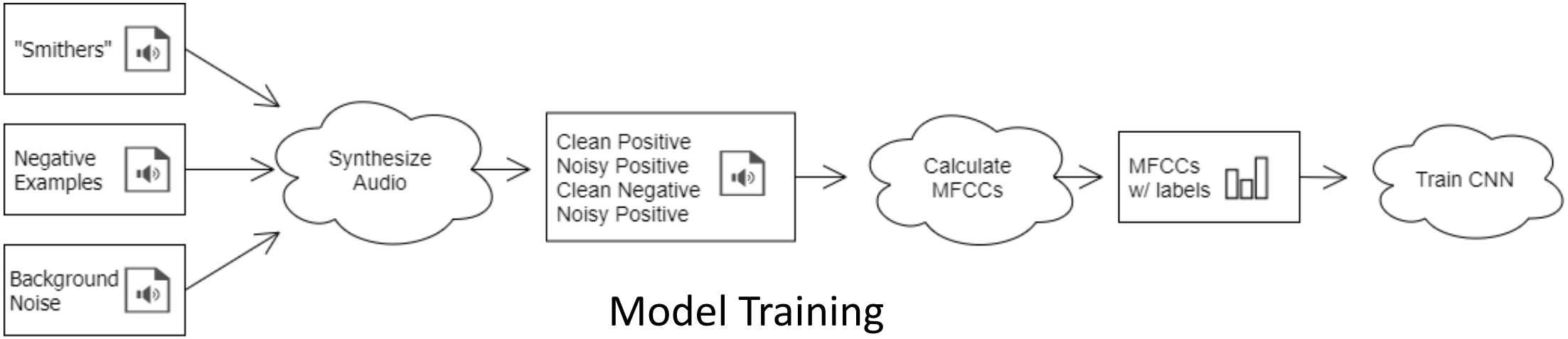
Background (cont.)

- Common approaches:
  - Convolutional Neural Nets (CNNs)
  - Mel-frequency cepstral coefficients (MFCC) – more on these later
- Papers referenced:
  - C. Jose, Y. Mishchenko, T. Senechal, A. Shah, A. Escott, and S. Vitaladevuni, “Accurate detection of wake word start and end using a cnn,” 2020 (Amazon)
  - G. Chen, C. Parada, and G. Heigold, “Small-footprint keyword spotting using deep neural networks,” in 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2014, pp. 4087–4091 (Google)
  - S. Sigtia, R. Haynes, H. Richards, E. Marchi, and J. Bridle, “Efficient voice trigger detection for low resource hardware,” in INTERSPEECH, 2018 (Apple)
  - T. -H. Tsai, P. -C. Hao and C. -L. Wang, "Self-Defined Text-Dependent Wake-Up-Words Speaker Recognition System," in IEEE Access, doi: 10.1109/ACCESS.2021.3117602.

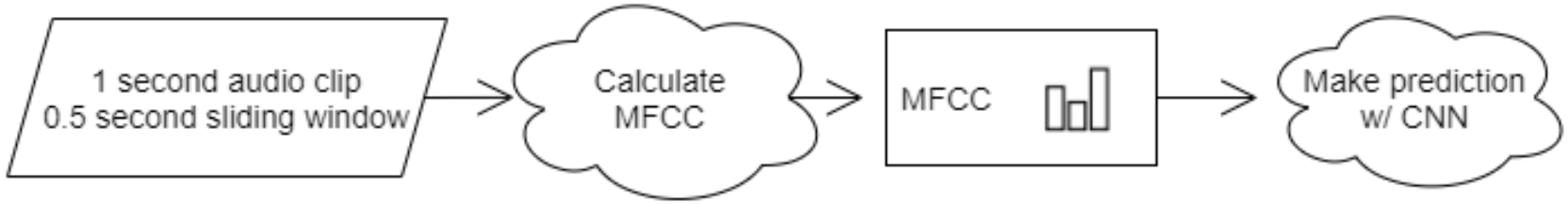
Method – "Smithers"

- Develop my own CNN capable of recognizing the wake word phrase "Hey Smithers"
- Why "Hey Smithers"? Smithers is the faithful assistant to the crotchety Mr. Burns in *The Simpsons*
- Use methods proposed by the referenced research paper

Method (cont.)



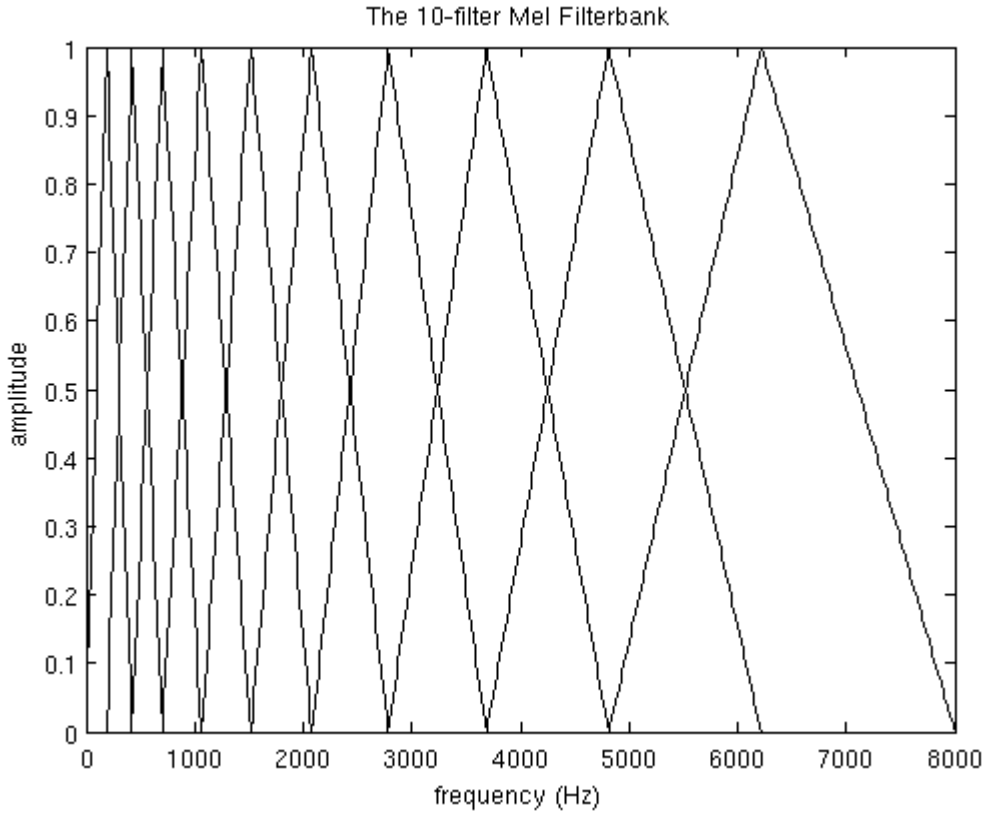
Model Training



Implementation

Method (cont.) - MFCCs

- Human ear frequency response is not linear
- Mel scale better mimics human hearing
- MFCCs – take FFT, apply mel filter bank, take logs of powers at each mel frequency, take discrete cosine transform
- MFCCs are amplitudes of spectrum
- Idea: distil spectrum into only relevant information
- Train CNN with mel-frequency cepstral coefficients (MFCCs)



Source: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/

Method (cont.) - CNN

Training:

- Training audio will be trimmed into 1 second clips – important!
- MFCC coefficients will be calculated for each clip
- Convolutional neural net will train on the MFCC coefficients

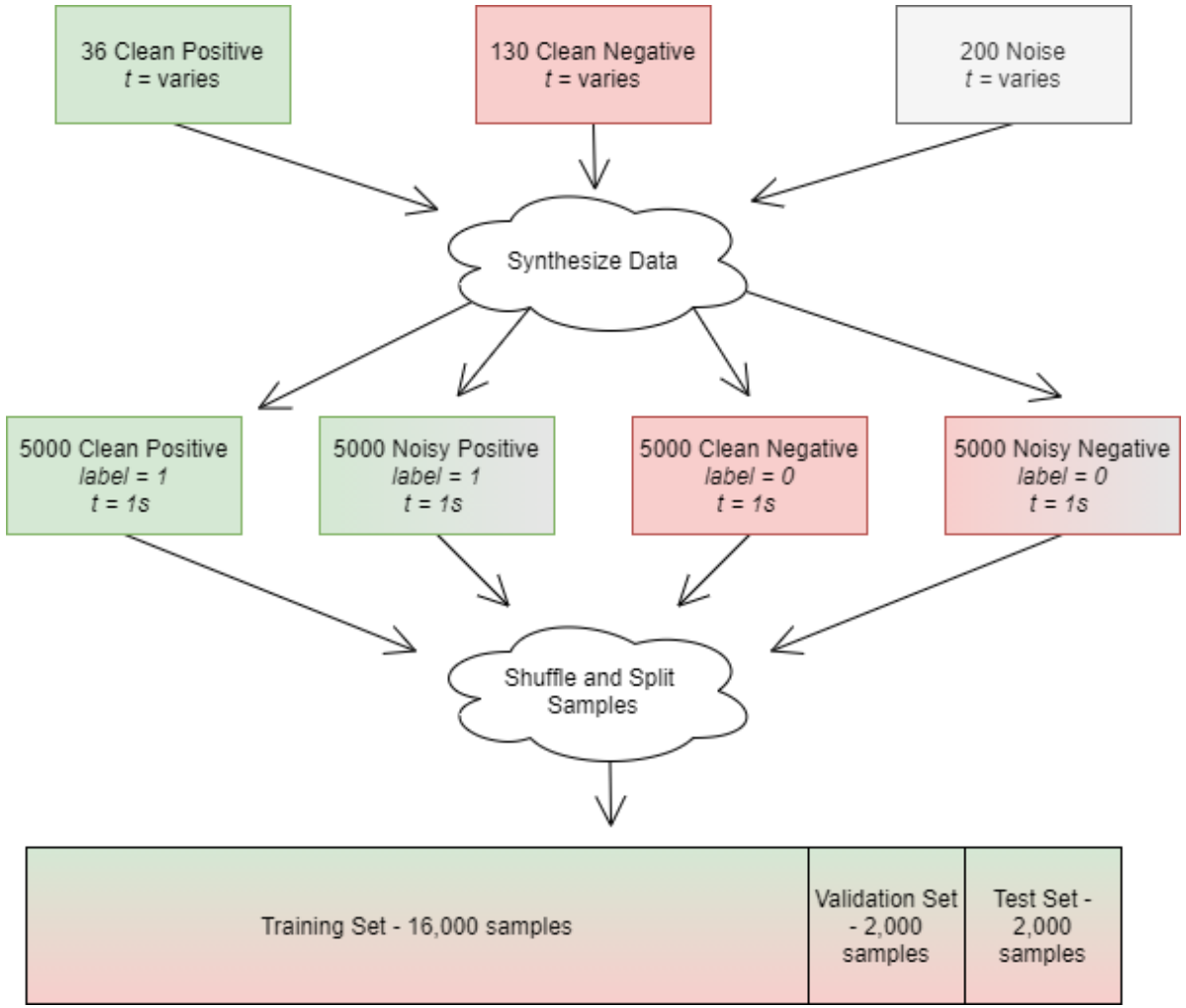
Implementation:

- Two 1-second input audio buffers will be used, spaced 0.5 seconds apart
- This sliding window will help when trigger word is split between two 1-second intervals

Results – Data Synthesis

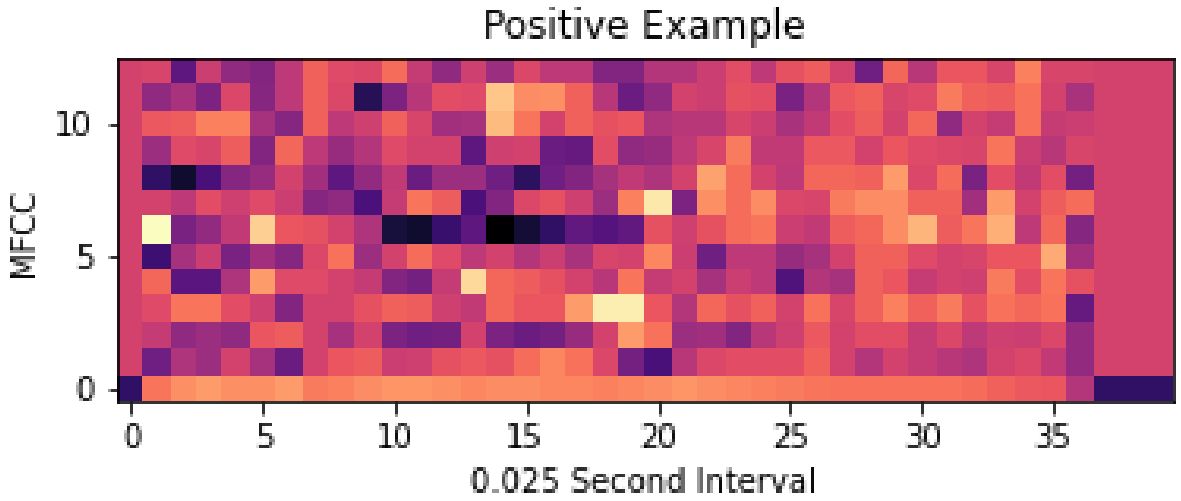
- Recorded 36 examples of the wake word phrase “Hey Smithers”
- Used 130 negative examples from Google’s Speech Commands Dataset
- Used 200 noise clips from Harvard’s ESC-50 dataset
- Synthesized 10000 examples each of positive and negative examples
  - For each loop, choose one positive example, one negative example, and one background noise
  - Randomly select 1 second interval of noise
  - Insert each positive example, negative example into random start point of background noise
  - Save noisy positive, clean positive, noisy negative, and clean negative
  - Half of dataset is clean, half is noisy
- Split data into 80% train, 10% validation, and 10% test sets
  - 16,000 train, 2,000 validation, and 2,000 test samples

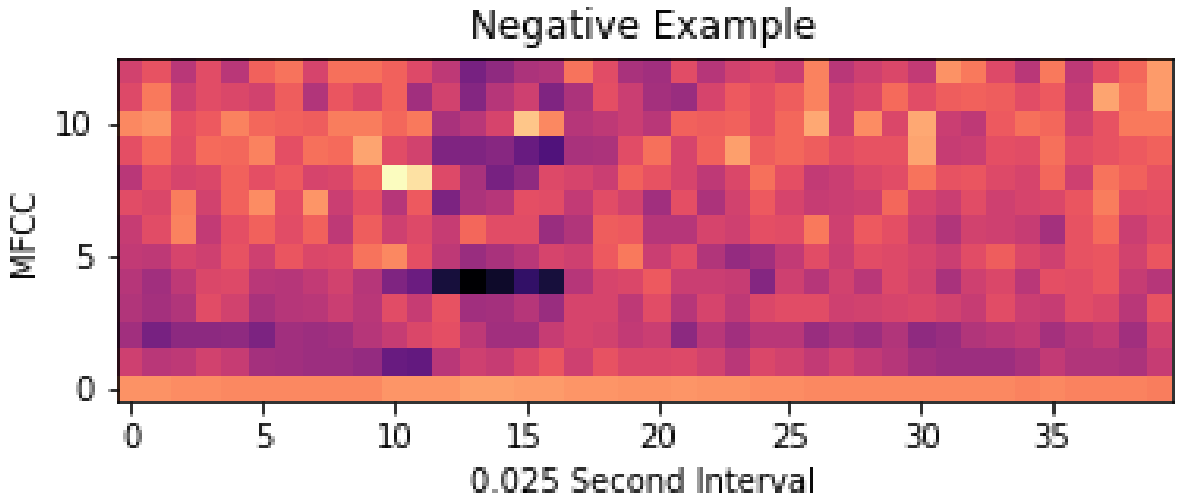
Results (cont.) – Data Synthesis



Results (cont.) - MFCCs

- Audio was downsampled to 10khz
- MFCCs were calculated at 0.025 second intervals – 40 total for each 1 second clip
- MFCCs casted from float64 to int8 to reduce computational complexity





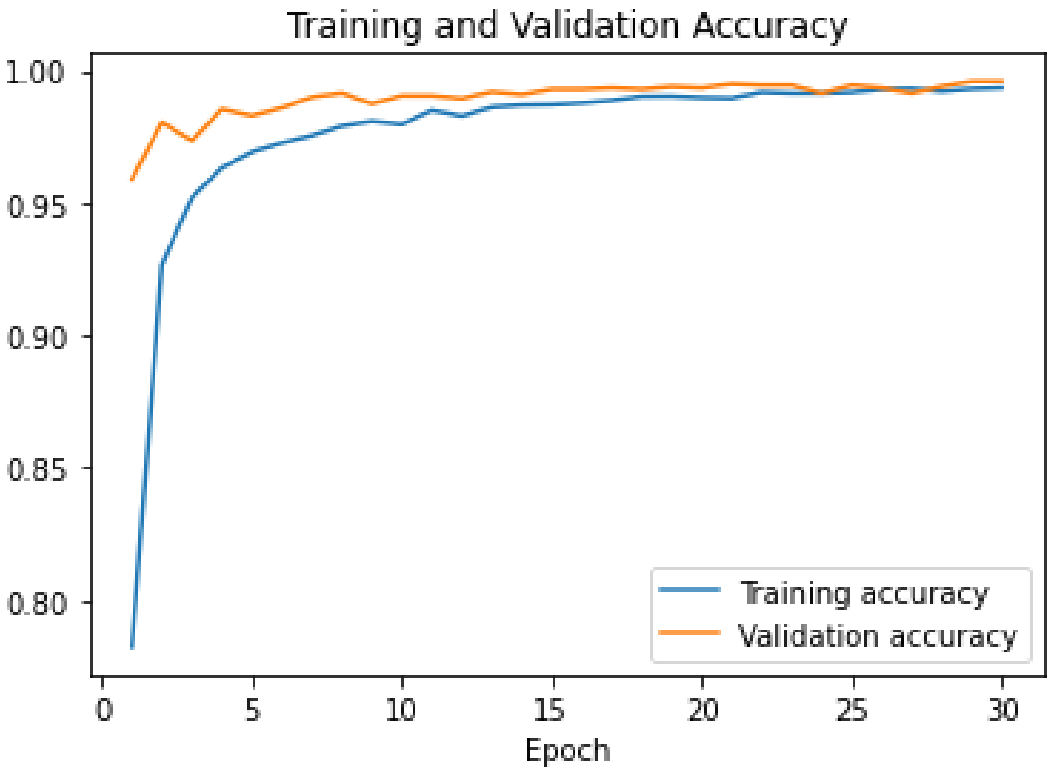
Results (cont.) - CNN

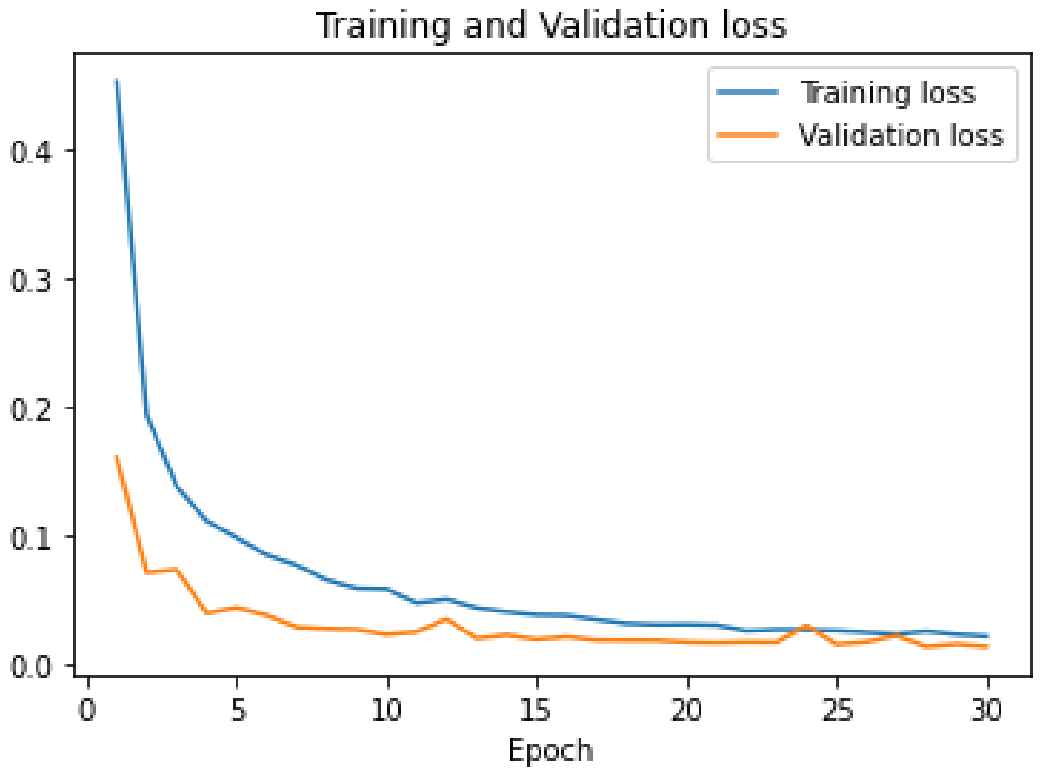
- Image processing CNN architecture
- Reduced number of trainable parameters
- RelU activation as suggested by Apple
- Sigmoid output for [0,1] prediction range
- 100 sample batch size, 30 epochs for training
- Binary cross entropy loss function, SGD optimizer for validation

```
Model: "Smithers"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_20 (InputLayer)        [(None, 13, 40, 1)]       0
_________________________________________________________________
conv2d_57 (Conv2D)           (None, 12, 39, 32)        160
_________________________________________________________________
max_pooling2d_57 (MaxPooling (None, 6, 20, 32)         0
_________________________________________________________________
conv2d_58 (Conv2D)           (None, 5, 19, 32)         4128
_________________________________________________________________
max_pooling2d_58 (MaxPooling (None, 3, 10, 32)         0
_________________________________________________________________
conv2d_59 (Conv2D)           (None, 2, 9, 32)          4128
_________________________________________________________________
max_pooling2d_59 (MaxPooling (None, 1, 5, 32)          0
_________________________________________________________________
flatten_19 (Flatten)         (None, 160)               0
_________________________________________________________________
dense_38 (Dense)             (None, 16)                2576
_________________________________________________________________
dropout_19 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_39 (Dense)             (None, 1)                 17
=================================================================
Total params: 11,009
Trainable params: 11,009
Non-trainable params: 0
_________________________________________________________________
```

Results (cont.) - Validation

- Model works!
- Model evaluation on test set: loss: 0.0147 - acc: 0.9960





Results (cont.) - Deployment

- Saved model as TensorFlow Lite model
- TensorFlow Lite – stripped down framework for “on-device inference”
- Model can be deployed on resource-constrained hardware
  - Requirement – Python
  - Includes embedded Linux devices like RaspberryPi, custom hardware, etc.
- Tested TensorFlow Lite model on development computer
  - Will load on RaspberryPi if I can get the hardware

Results (cont.) - Discussion

- Optimizations for resource-constrained hardware:
  - Reduced number of CNN parameters
  - MFCC's quantized to 8-bit signed ints
  - Reduced sample rate from 20kHz to 10kHz
- Other improvements:
  - Increased number of samples from 400 to 20,000
  - Changed wake word phrase from "Smithers" to "Hey Smithers"
    - 3-syllable wake words suggested by Tsai et. al.

Proposed Research

- Further optimizations
  - Reduce sample rate
  - Reduce number of MFC coefficients?
  - Reduce number of CNN parameters
- Hardware deployment
- Device responds to wake word – now what?

Conclusions

- *Smithers* is a machine learning model used to detect a wake word
- MFCCs and CNNs are the key components to this model
- Model works well!