Google TensorFlow Speech Recognition Competition

# Project

<https://www.kaggle.com/c/tensorflow-speech-recognition-challenge>

This project is the Google sponsored TensorFlow Speech Recognition competition.

# Goal

To build an algorithm that understands simple spoken commands using the Speech Commands Dataset as input to the algorithm. Improve on the current baseline accuracy of 0.88 achieved by the TensorFlow Speech tutorial.

# Data

<https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/data>

TensorFlow released the Speech Commands Dataset that consist of 65,000 one-second long utterances of 30 short words by thousands of different people. The training data consists of a few informational files and a folder of audio files. The audio folder contains subfolders with 1 second clips of voice commands, with the folder name being the label of the audio clip. The training data has 12 possible labels: yes, no, up, down, left, right, on, off, stop, go, silence, unknown. The ‘unknown’ label is used for any utterance that is not one of the first 10 labels or that is not ‘silence’. There are more labels in the training set than what the algorithm is to be trained to predict. There is also a folder of ‘\_background\_noise\_’ that contains longer clips of ‘silence’. This can be used for additional model training.

The test data consists of 150,000+ .wav files.

## Data acquisition

The data to test and train the algorithm is provided by the kaggle competition.

Note: This is a multiclass classification problem.

## Data Transformation

The initial assumption is that all the test data is of a 16000-sample rate. After introspection some are less while some are more. To counter this I padded the files that have a smaller sampling rate with additional 0s. Files with a greater than 16000 sampling rates were chopped into smaller chunks. For my initial runs the algorithm halves the sampling rate to decrease model train/predict time (current model training time: from 1hr 40min to 31min)

The model uses a Convolutional Neural Net to predict the labels for the .wav files. The .wav files are transformed into image data by converting them into a spectrogram. A spectrogram is a visual representation of the spectrum of frequencies of sound as they vary with time. The algorithm uses spectrograms to identify spoken words phonetically. This is normal practice in the field of speech processing. The scipy.signal.spectrogam api is used to generate the spectrogram.

The input matrix, x\_train, consists of the set of spectrograms generated from the .wav files. The prediction matrix, y\_train, consists of the 12 labels (yes, no, up, down, left, right, on, off, stop, go, silence, unknown). The ‘\_background\_noise\_’ label is converted to ‘silence’ prior to processing.

X\_train.shape: (64,841, 99, 161, 1)

Y\_train.shape:

(64,841, 12)

64, 841 rows (spectrograms) by 12 columns (labels)

# Model Selection

A Convolutional Neural Network (CNN) model was chosen for spectrogram analysis. This is a common practice in speech analysis. i.e. converting audio data to image data and running it through a CNN. The TensorFlow Speech tutorial runs a CNN on the Speech Commands dataset as a starting point for the kaggle competition participants. Keras, a high-level neural networks API capable of running on TensorFlow, was chosen for the initial model construction. See the table below:

## Topology

|  |  |
| --- | --- |
| **Layer** | **Description** |
| Input | First layer – tuple of integers |
| Batch Normalize | Maintains mean activation close to 0 & stdev close to 1. (allows higher learning rates) |
| Convo2D | 8 filters, 2x2 convo window, relu activation |
| Convo2D | 8 filters, 2x2 convo window, relu activation |
| Pooling2D | 2x2 pooling window |
| Dropout | 0.2 rate, prevent overfitting |
| Convo2D | 16 filters, 3x3 convo window, relu activation |
| Convo2D | 16 filters, 3x3 convo window, relu activation |
| Pooling2D | 2x2 pooling window |
| Dropout | 0.2 rate, prevent overfitting |
| Convo2D | 32 filters, 3x3 convo window, relu activation |
| Pooling2D | 2x2 pooling window |
| Dropout | 0.2 rate, prevent overfitting |
| Flatten | Flattens the input |
| Dense/Batch Normalize | 128 units, relu activation |
| Dense/Batch Normalize | 128 units, relu activation |
| Dense | 12 final output classes, relu activation |
|  |  |

Multiple dense layers is generally accepted practice – first layers learn edge detectors and subsequent layers learn more complex features and higher level layers encode more abstract features. Additionally, a dropout layer is applied after each pooling layer.

## Hyperparameters

Two dropout rates were compared, 0.2 & 0.5, for each dropout layer of the model. The competition score decreased dramatically with a dropout rate of 0.5. When fitting the model to the training data two epochs were compared, 5 & 10. Training the model for 10 epochs produced a higher accuracy score on the kaggle competition data. More testing is required to determine at which epoch the training accuracy no longer improves.

Initially, the model used a binary\_crossentropy loss function. Since this is a multi-class classification problem this has been switched to a categorical\_crossentropy loss function. The difference in accuracy that can be related to the categorical\_crossentropy loss function is not clear.

The activation function for the final dense layer is ‘softmax’ – this is standard for classification models.

The optimizer used for the model is the ‘Adam’ optimizer. The default parameters to the Adam optimizer are currently used. The default learning rate is ‘0.001’. (Future option: customize the optimizer parameters, try different learning rates, and compare against the ‘SGD – Stochastic Gradient Descent’ optimizer.)

## Metrics

During model compilation ‘accuracy’ metrics were gathered. Example summary output from a training run consisting of: epochs=10, optimization=Adam, loss=categorical\_crossentropy, Dropout=0.2)

Train on 58356 samples, validate on 6485 samples

Epoch 1/10

- 591s - loss: 1.0806 - acc: 0.6862 - val\_loss: 0.5846 - val\_acc: 0.8122

Epoch 2/10

- 606s - loss: 0.5370 - acc: 0.8237 - val\_loss: 0.3786 - val\_acc: 0.8790

Epoch 3/10

- 592s - loss: 0.4166 - acc: 0.8643 - val\_loss: 0.2889 - val\_acc: 0.9115

Epoch 4/10

- 593s - loss: 0.3542 - acc: 0.8840 - val\_loss: 0.2357 - val\_acc: 0.9243

Epoch 5/10

- 592s - loss: 0.3101 - acc: 0.8989 - val\_loss: 0.2555 - val\_acc: 0.9192

Epoch 6/10

- 591s - loss: 0.2814 - acc: 0.9079 - val\_loss: 0.2151 - val\_acc: 0.9291

Epoch 7/10

- 591s - loss: 0.2592 - acc: 0.9154 - val\_loss: 0.1932 - val\_acc: 0.9399

Epoch 8/10

- 592s - loss: 0.2410 - acc: 0.9226 - val\_loss: 0.1830 - val\_acc: 0.9443

Epoch 9/10

- 592s - loss: 0.2272 - acc: 0.9255 - val\_loss: 0.1818 - val\_acc: 0.9403

Epoch 10/10

- 592s - loss: 0.2121 - acc: 0.9307 - val\_loss: 0.1642 - val\_acc: 0.9503

Training set loss decreases and accuracy increases thru all epochs. Validation loss decreases and accuracy increases thru all epochs.

# Future model modifications

Current best score on the competition leader board with this model is 0.73. This is 504th place out of 975 entries. The best submitted score is 89th. The following model changes will be incorporated to increase the current accuracy score on the Kaggle competition data.

1. Construct the model w/o Keras – using TensorFlow api’s directly to have greater access to model tuning.
2. Construct more ‘silence’ audio samples from the ‘\_background\_noise\_’ wave files. This will better train the model on ‘silence’.
3. Construct a deeper CNN, i.e. each subsequent layer (or filter) learns more complex representations.
4. Vary the hyperparameters:
   1. Train for additional epochs
   2. Try dropout rates btw 0.2 – 0.35.
   3. Try SGD and other optimization functions
   4. Vary the learning rate (lower learning rate = reliable but long training. Higher learning rate = training may not converge or may overshoot but training will be faster.)
5. Use Transfer Learning – take a Neural net that has been trained on a similar dataset, and retrain the last few layers of the network for new categories. The idea is that the beginning layers of the network are solving problems like edge detection and basic shape detection, and that this will generalize to other categories. (See ‘Transfer Learning for Sound Classification’)