Winning Model Documentation

Name: Tom O’Malley

Location: New Jersey

Email: omalleyt@bu.edu

Competition: Tensorflow Speech Recognition Challenge

1. Background on you/your team

* What your academic/professional background?

B.S Industrial Engineering, Rutgers University

M.A. Economics, Boston University

* Did you have any prior experience that helped you succeed in this competition?

I work as a data scientist in the insurance industry, but have no particular experience with acoustic data analysis.

* What made you decide to enter this competition?

I relished the opportunity to participate in a challenge sponsored by the Google Brain team, and the opportunity to explore the application of deep learning techniques to a somewhat non-traditional domain.

* How much time did you spend on the competition?

I ramped up in the last month of the competition and tried to spend 1-3 hours a day working on it at that point

2. Summary

I used an ensemble of Convolution Neural Networks. The models were mostly slight variations of my best model. My best single model used 120 Log Mel Filterbanks and achieved a private LB score.of 90.9% All WAV files were standardized to have the same peak volume. Vocal tract length perturbation, speed distortion, and background noise were used for data augmentation. Some models in the ensemble used files from the test set that other models classified as silence or unknown for pseudo-labels for semi-supervised learning. The model was trained in Tensorflow using the ADAM optimizer. Each model takes 6-10 hours to train on a K80 GPU.

3. Features Selection / Engineering

* What were the most important features?

The features that seemed to work the best were 120 Log Mel Filterbanks. I also used a model with 40 Log Mel Filterbanks and a model with 13 MFCCs in the winning ensemble. My best model used a window size of 30ms and a stride of 10ms. In the ensemble I also added models that used a window size of 16ms and a stride of 10ms, and this improved the overall score.

* How did you select features?

I initially tried MFCCs but found Mel Filterbanks to perform better with my model architecture, and Log Mels performed slightly better than Mels.

* Did you make any important feature transformations?

One of the models in the ensemble transformed the Log Mel Filterbanks so that every frame of the spectrogram had equal energy. This was not my best model, but was used in the ensemble.

* Did you use external data?

I did not use any external data.

4. Training Method(s)

* What training methods did you use?

I trained the models using the ADAM optimizer. The initial learning rate was set to 0.01, and decreased over time down to 1e-6

* Did you ensemble the models?

Yes.

* If you did ensemble, how did you weight the different models?

I averaged the square roots of the class probabilities from each model, as this seemed to provide a slightly better result than simply averaging the probabilities

5. Interesting findings

**Model Architecture:**

I used 120 log-mel filterbanks for my best model. Given this, I thought it was important to create a model that treated time and frequency differently. Specifically, I didn't do any downsampling in the time domain until the very end. With time as the first dimension and frequency as the second, my model architecture was:

1) Conv2d(64, [7,3] )

I thought of this as a "denoising" and basic feature extraction step

2) MaxPool( [1,3] )

Getting back down to the standard 40 frequency features

3) Conv2d(128, [1,7] )

Look for local patterns across frequency bands

4) MaxPool( [1,4] )

Allow for speaker variation, similar to what worked here: <https://link.springer.com/content/pdf/10.1186%2Fs13636-015-0068-3.pdf>

5) Conv2d(256 [1,10], padding="VALID")

This allows it to treat each remaining freq band very differently, and compress the frequency dimension entirely. I think of this as detecting phoneme-level features

6) Conv2d(512,[7,1])

I think of this as looking for connected components of a short keyword at different points in time

7) GlobalMaxPool in time

Collect all the components

8) Dropout + Fully Connected 256

Because why not, and seemed to work well

**Data Augmentation/Standardization:**

In addition to time stretching (which gave a boost of +1% LB), there were two techniques I applied that I haven't seen mentioned yet here that I think really helped.

1) Standardize Peak (Windowed) Volume

Basically, I took every clip, split it into 20 to 50 chunks, and then standardized the volume of the clips so that every clip had the same max chunk volume. Why this approach? Well, standardizing by average volume would be fine, but since some keywords were longer than others, very short . keywords would have a low average volume even though the part of the WAV containing the actual word could be quite loud.

When I implemented this, my LB score went from 82% to 84%

My theory is that by doing this, the convnet didn't have to deal with as many issues in terms of different scales for the same feature, since the volumes of the WAVs spanned orders or magnitude. Obviously using Log Mels helped with this too.

2) Vocal Tract Length Perturbation

I used VTLP as described in this paper <https://pdfs.semanticscholar.org/3de0/616eb3cd4554fdf9fd65c9c82f2605a17413.pdf>

This perturbation could be applied when creating the weight matrix to convert a spectrogram into log mels, so it was a very fast augmentation method

This increased my LB score +1%, and I saw the greatest benefit using the same VTLP factor within a batch, along the line of reasoning described here:

<https://arxiv.org/abs/1707.00722>

6. Simple Features and Methods

Many customers are happy to trade off model performance for simplicity. With this in mind:

* Is there a subset of features that would get 90-95% of your final performance? Which features?

Using just the single best model would give almost the same accuracy as my overall ensemble. My best single model achieved a 90.9% private LB, and a 90.4% score on the public LB. This model used 120 Log Mel Filterbanks, but I believe that using 40 Log Mel Filterbanks would probably achieve only slightly lower accuracy, and would enable one to remove the first layer on my model architecture. I also tried a depthwise-separable version of the CNN that resulted in a much smaller model and achieved approximately 89% accuracy. Also, a model with fewer convolutional channels at each level achieved 89% accuracy and ran in 155ms on the Raspberry Pi.

* What model that was most important?

The model that used 120 Log Mel Filterbanks was the most important, this model if referred at as “overdrive\_full\_bn” in the models.py file

* What would the simplified model score?

Without ensembling, my best single model achieved a 90.9% private LB, and a 90.4% score on the public LB

Appendix

A1. Model Execution Time

* What software did you use for training and prediction?

Tensorflow

* What hardware (CPUS spec, number of CPU cores, memory)?

1 K80 GPU, 6 CPU cores

* How long does it take to train your model?

6-10 hours per single model

* How long does it take to generate predictions using your model?

The best single model ran in ~250ms on a Raspberry Pi, the ensemble take approximately 10x as long

* How long does it take to train the simplified model (referenced in section 4)?

6-10 hours

* How long does it take to generate predictions from the simplified model?

A simplified model with fewer convolutional channels ran in 155ms on the Raspberry Pi

A2. Dependencies

* programming language/statistical tool

Python

* libraries or packages

bleach==1.5.0

enum34==1.1.6

html5lib==0.9999999

Keras==2.1.3

llvmlite==0.21.0

Markdown==2.6.11

numba==0.36.2

numpy==1.14.0

pandas==0.22.0

protobuf==3.5.1

python-dateutil==2.6.1

pytz==2017.3

PyYAML==3.12

scipy==1.0.0

six==1.11.0

tensorflow-gpu==1.4.1

tensorflow-tensorboard==0.4.0

toolz==0.9.0

Werkzeug==0.14.1

* operating system

Ubuntu

A3. How To Generate the Solution (aka README file)

The commands to generate the submission file are contained in run.sh in the zip file provided. This guide assumes that the training files have been placed in a directory called “train” under this directory and that the test files are in a directory called “test”.

The commands (contained in run.sh) are also listed below:

#!/usr/bin/env bash

python train.py --features log-mel --model overdrive --save overdrive --train --no-val

python train.py --features log-mel --model overdrive --save overdrive\_pl1 --pseudo\_labels overdrive --train --no-val

python train.py --features log-mel --model overdrive --save overdrive\_pl2 --pseudo\_labels overdrive\_pl1 --train --no-val

python train.py --features equal-log-mel --model overdrive --save overdrive\_frame\_eq --train --no-val --pseudo\_labels overdrive

python train.py --features log-mel-40 --model okconv --save okconv --train --no-val

python train.py --features mfcc-13 --model mfccnet --save mfccnet --train --no-val

python train.py --features mfcc-13 --model mfccnet --save mfccnet\_pl1 --train --no-val --pseudo\_labels mfccnet

python train.py --features short-log-mel --model overdrive --save overdrive\_short --train --no-val

python train.py --features short-log-mel --model overdrive --save overdrive\_short\_pl1 --pseudo\_labels overdrive\_short --train --no-val

python train.py --features short-log-mel --model overdrive --save overdrive\_short\_pl2 --pseudo\_labels overdrive\_short\_pl1 --train --no-val

cp models/overdrive\_comp.pickle ensemble\_models/

cp models/overdrive\_pl1\_comp.pickle ensemble\_models/

cp models/overdrive\_pl2\_comp.pickle ensemble\_models/

cp models/overdrive\_frame\_eq\_comp.pickle ensemble\_models/

cp models/okconv\_comp.pickle ensemble\_models/

cp models/overdrive\_short\_comp.pickle ensemble\_models/

cp models/overdrive\_short\_pl1\_comp.pickle ensemble\_models/

cp models/overdrive\_short\_pl2\_comp.pickle ensemble\_models/

python ensemble\_sqrt.py