CS230

Automated Speaker Verification (2019) for CS-230 Winter 2020

Daniel J. Evert*
Department of Computer Science
Stanford University
dje334@stanford.edu

Abstract

The Automated Speaker Verification (ASV) Spoof Challenge of 2019 is an international modeling challenge with the purpose of detecting spoofed audio, specifically text-to-speech (TTS), voice conversation (VC), and replay spoofing attacks. The task was to design automatic systems capable of discriminating between bona fide and spoofed speech aimed to encourage the design of generalized countermeasures. In order to meet this challenge, this project sought a variety of deep learning architectures which combined 1-dimensional convolutional neural networks (CNNs) and long-short term memory (LSTM) recurrent neural networks (RNNs) using the resources in Amazon's Cloud Service (AWS). For this architecture to succeed, all audio was processed with Mel-frequency cepstral coefficients (MFCC) into what's commonly known as Mel spectrograms, and hence, was transformed into an image.

1 Introduction

The rise and accessibility of (deep) fake audio poses a modern risk to financial institutions as it grapples with a variety of social engineering attack vectors, specifically one in which cybercriminals disguise their voice using synthetic speech to access consumer accounts. If a financial institution was equipped with knowledge at the point when fraudsters disguise their voice, it can potentially mitigate this attack vector which attempts to acquire credentials from unsuspecting consumers saving these companies millions of dollars. The ASVspoof 2019 Challenge was set up to deal with this challenge of detecting synthetic audio by providing variety of spoofing scenarios injected into the datasets. Due to computational memory and time limitations, this project's scope focused on the Logical Access (LA) dataset.

The purpose of this project was to minimize tandem decision cost function (t-DCF)[3], a cost function combining the countermeasure (CM) and automatic speaker verification (ASV) systems. The ASVspoof Challenge provided scores for an ASV system, thus all modeling efforts were poised to detect "bonafide" and "spoof" speech of the CM system. The t-DCF was based on three scenarios: "bonafide target", "bonafide non-target" and "spoof", where target is defined as whether the ASV-system can verify a specific speaker, alongside scenarios for costs of misclassifications.

The input for the the CNN-LSTM, and henceforth will be called Conv1D-LSTM, architecture was a grayscale Mel spectrogram, an image which was converted from the provided audio clips with column frequency, time domain and channels as a 128XnX1 numpy array, respectively. The binary classifier would output a score and be combined with the ASV-system to compute the t-DCF.

*LinkedIn profile: https://www.linkedin.com/in/daniel-evert-68a52623/

2 Related work

Much inspiration for this project's approach came from an ASVspoof 2017 submission which used a light-CNN and a bidirectional gated recurrent network (GRU)[8], the state-of-the-art solution. A significant difference between their approach and mine was in scale (i.e. 864 vs 128 columns) and pooling activation (i.e. Max-Feature-Map vs Max Pooling).

Previous ASVspoof challenges[10] highlighted several different feature extraction methodologies for the audio, including Short-term Power Spectrum Features (e.g., log-spectrum, Cepstrum, Delta-Cepstrum, etc.), Short-Term Phase Features (e.g. All-pole group delay function (APGDF)), and Spectral Features with Long-term Processing (e.g. Modulation spectrum (ModSpec)). This project focused on only the grayscale Mel Spectrogram, though these additional conversions are likely to add value.

Other methods existed which did not utilize a deep learning architecture, specifically the use of Guassian-Mixture Models (GMMs) classifiers[11]. The 2015 ASVspoof challenge highlighted this classifier to be in the top-ranking and most used[12]. The classification with a GMM-based method supposes that the acoustical parameter repartition for a sound class may be modeled with a sum of Gaussian distributions[13]. The strength of this approach makes sense, frequencies are sine waves and may likely be modeled by gaussian shapes. However, a weakness to this approach is the added hyper-parameterization of choosing a-priori the number of distributions, likely leading to lower sample size clusters. In contrast, deep learning architectures are likely to provide a richer set of features, though at risk to over-fitting. In my hypothesis, if constructed properly, the right deep learning architecture would yield maximum results if tuned appropriately, given the Universal Approximation theorem of neural networks.

3 Dataset and Features

All data was received from the ASVspoof challenge[1]. Table 1 describes high level the LA dataset. For further details about each spoofing attack, Table 2 shows the representation, the type of spoof, the algorithm which spoofed, and the number of records. Table 2 further shows that each partition yielded 89% spoofed data.

Table 1: High Level Details of Each Partition

Partition	Record Count	Average(s)	Minimum(s)	Maximum(s)
Training	25,380	3.426	0.652	13.188
Development	24,844	3.478	0.695	11.594
Evaluation	71,237	3.108	0.467	13.026

Table 2: Audio Attack Vector Breakdown of Training Partition for LA Group

Partition	Data Label	Spoof Type	Algorithm Used to Spoof	Record Count
Training	A01	TTS	neural waveform model	3,799
Training	A02	TTS	vocoder	3,799
Training	A03	TTS	vocoder	3,799
Training	A04	TTS	waveform concatenation	3,799
Training	A05	VC	vocoder	3,799
Training	A06	VC	spectral filtering	3,799
Training	-	Real	Bonafide	2,579

All data was processed with Mel-frequency cepstral coefficients (MFCC) which are based on processing audio with a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency. In effect, this pre-processing of the audio files created a spectrogram, a feature representation from an audio signal to an image. This translation requires the use of parameters (i.e. Fourier-Transform, Hop-length, the mel coefficient, etc.), thus the conversion to an image for a richer set of features comes at a cost of reducing information about the audio file. The log-frequency (y-axis) and time

domain (x-axis) are affected by parameter choice. Only one set of parameters was chosen for this project due to time limitations.

Once spectrograms were created the program would slice each spectrogram according to the number of timesteps (i.e. rows) of the image. This was used as a hyper-parameter. In an ideal state, each sliding window would be included in the training dataset; however, the yield of offsetting by 1 would be over 8 million images. Instead, various timesteps (76, 64, 48) and were chosen with an two different offsets (16, 12 and 7)]. Figure 1 shows two Mel Spectrograms, one a "bonafide" example and the other a "spoof" example. The final Conv1D-LSTM model achieved a training set size of 787,967 and a validation set of 215,875 records using a 48 time-step bidirectional LSTM with an offset of 7, achieving an overlap of 14.6% per audio file for the sliding window. Note, the validation set was reduced so as to fit both datasets into memory on AWS.

Two types of standardization was used: 0-1 scaling and z-score scaling. The results of the first was to divide all data by 80, given the maximum value in decibles for each Mel-spectrogram was capped at 80. The z-score scaling resulted in a mean of -53.15 with a standard deviation of 17.33 for the training dataset. The idea to standardize the data is supported by empirical evidence that the model converges faster, as this project discovered non-standardized inputs converging slower. No data augmentation was considered given the volume of training records, which exceeded memory in AWS.

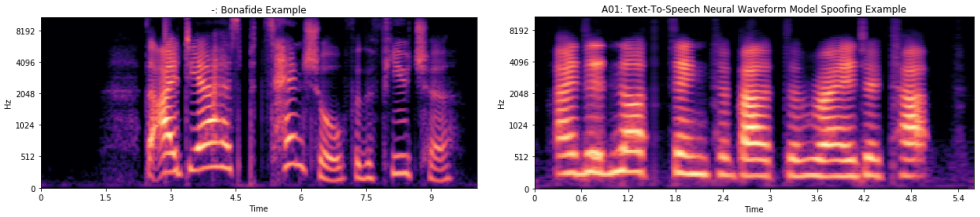

Figure 1: Mel Spectrograms

4 Methods

The first step to audio classification was to consider removing "empty" leading and trailing audio segments. To achieve this reduction in empty noise, this project utilized a common approach in speech science which took the the root mean square (RMS) of the amplitude on the waveform. The calculation for RMS is shown below.

$$x_{rms} = \sqrt[2]{x_1^2 + x_2^2 + x_3^2 + ...x_n^2}$$

A threshold (0.08) for each RMS value would then loop through each Mel-spectrogram, removing all audio segments less than or equal to the threshold until the first value was greater.

The next step was to determine how to score variable-length audio. A sliding window approach (i.e. 48 time stemps) was chosen, being mindful of the size so as to capture 99% of all records. Spectrograms less than 48 were removed (< 0.1% of records). To illustrate the inputs into the Conv1D-lSTM, suppose a the dimensions of the spectrogram were 300X128X1 with a sliding window of 48 timesteps. This would yield 252 samples available for training if an offset of 1 was chosen. The pro of this approach are generating millions of records, while the con then over-samples, and perhaps, overfits to, longer audio files. No compensation of this imbalance of samples was considered due to time constraints, but future efforts would attempt at balancing each audio file. Subsequently, all scores for an audio file explored taking the maximum, minimum, average and median of the values for each offset per audio file as the final score used for prediction.

The final deep learning architecture was a 4-layers of 1-dimensional Convolutions and two fully connected (FC) layers with their output as input into a timestep of a many-to-one bidirectional LSTM, as detailed in Table 3. The framework mimicked the success of other convolutional layers where channels increase over time while the kernel size decreases. Attempts to regularize the model were done via dropout; attempts to converge faster with batch normalization; and attempts to reduce kernel sizes via max pooling. Given the binary output of "bonafide" and "spoof", a final logistic function was used with the chosen loss function using binary cross entropy, with the equation shown below.

$$BinaryCrossEntropy = -\frac{1}{m} * \sum y^i * log(y^i) + (1 - y^i) * log((1 - y^i))$$

To understand what each convolution is doing, it's best to go through an example. For the first convolution, a kernel size of 65, a stride of 1, and 18 channels were chosen. The effect is taking a 128X1 slice and converting it into 65X18 dimensions, where the 1st dimension on the 1st element of the 65 kernel represents the 1st to 63rd elements of the 128X1 slice. Subsequently, the 2nd element of the 65 kernel would represent the 2nd - 64th elements of the 128X1 slice, and so on. A max pooling would then be applied taking the maximum values of the 1st two dimensions of this newly created 65 dimensional kernel, in effect reducing the slice from 65 to 32 dimensions. Batch Normalization would then be applied as an attempt to increase convergence by normalizing the outputs of the convolutional layer, as scaling the weights generally aids in faster convergence.

Layer (type)	Output Shape	Param #
TimeDistributed(Conv1D)	(None, 48, 65, 18)	1,170
TimeDistributed(MaxPooling(2))	(None, 48, 32, 18)	0
TimeDistributed(BatchNorm)	(None, 48, 32, 18)	72
TimeDistributed(Conv1D)	(None, 48, 16, 20)	380
TimeDistributed(MaxPooling(2))	(None, 48, 16, 20)	0
TimeDistributed(BatchNorm)	(None, 48, 16, 20)	80
TimeDistributed(Conv1D)	(None, 48, 8, 22)	462
TimeDistributed(MaxPooling(2))	(None, 48, 8, 22)	0
TimeDistributed(BatchNorm)	(None, 48, 8, 22)	88
TimeDistributed(Conv1D)	(None, 48, 8, 24)	552
TimeDistributed(MaxPooling(2))	(None, 48, 4, 24)	0
TimeDistributed(flatten())	(None, 48, 4, 24)	0
TimeDistributed(FC)	(None, 48, 96)	9,312
Dropout(0.3)	(None, 48, 96)	0
TimeDistributed(BatchNorm)	(None, 48, 96)	384
TimeDistributed(FC)	(None, 48, 96)	9,312
Dropout(0.3)	(None, 48, 96)	0
bidirectional(LSTM)	(None, 96)	55,680
Dense(1)	(None, 1)	97

Table 3: Table of Final Neural Network Architecture

5 Experiments/Results/Discussion

Table 4 shows the list of hyper-parameters explored for this project, though not all combinations were tried and tested, given time considerations (15 minutes per epoch). Experiments with modifying time-steps yielded less training records which resulted in wild swings on validation accuracy during each epoch (20% to 95% range as epoch grew). Experiments with batch-sizes showed GPU's train slowly with size of 32 as compared with 128 to 512; however, using CPU's approximately the same. Experiments with learning rates showed 0.1 to have wild swings early in the epochs on validation sets. Experiments with convolutional layers slowed down training each epoch considerably (+5 mins/conv layer per epoch). Experiments with dropout showed significant decease in training accuracy (99% to 95% with increasing dropout), though solution could not prevent overfitting (i.e. training AUC of 0.91 with evaluation 0.82). Architecture likely too deep and complex. Experiments to solve class imbalance showed slower training convergence the more imbalanced the dataset. Experiments with optimizer showed little difference between Adam and Nadam. Most epochs did not reach 100 given in most architectures would would require 36 - 96 hours to train on 4 GPUs. Experiments with LeakyReLU showed faster convergence than ReLU (likely due to neurons with non-0 gradients). Stacked LSTM's & GRU's were attempted, but pushed back due to increased time/epoch. All results are in Figure 2.

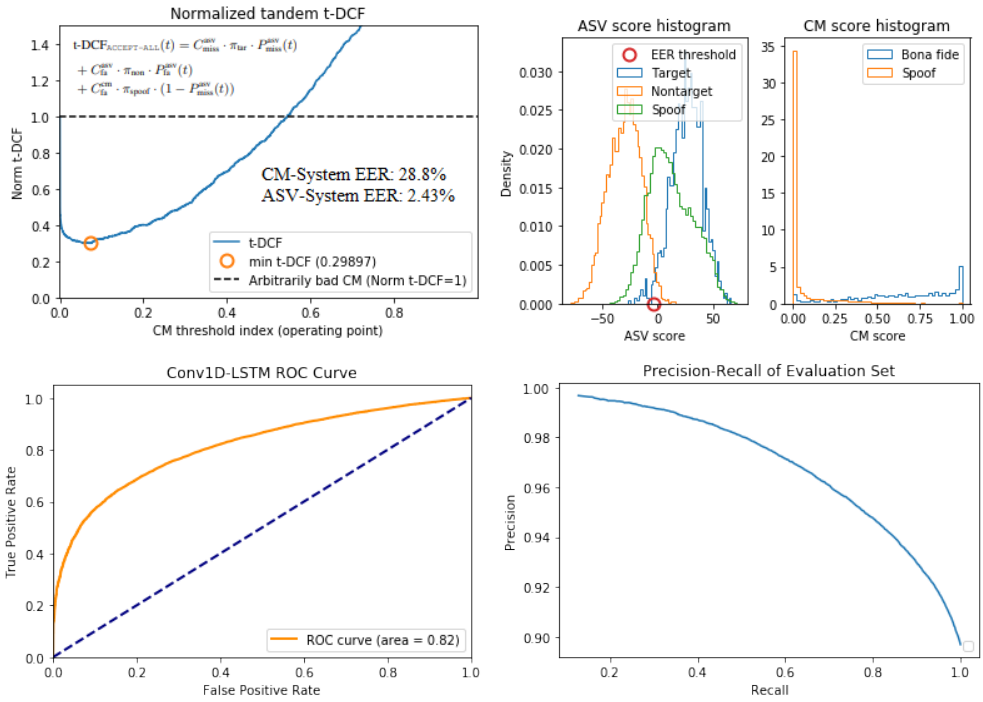

Figure 2: Eval t-DCF Graph with equal error rate (EER) (top-left),Eval histogram by score (top-right), Eval ROC-curve (bot-left), and Eval Precision/Recall Graph (bot-right)

Hyper-parameter Name	Hyper-parameter Value	Best Hyper-parameter Combination
Activation	[ReLU, LeakyReLU]	LeakyReLU
Class Weights	[1, 4, 9]	9
learning rate	[0.1,0.01,0.001]	0.001
Optimization Algorithm	[Adam, Nadam]	Adam
batch size	[32, 64, 128, 256, 512]	128
Sliding Window Offset	[16, 12, 7]	7
Number of LSTM Inputs	[76, 64, 48]	48
Number of Convolutional Layers	[4, 3, 2, 1]	4
Number of Channels	[6, 12, 18, 20, 22, 24]	18,20,22,24 combination
Number of epochs	[<100, 100]	20

Table 4: Table of Hyper-parameters

6 Conclusion/Future Work

In summary, lot's of failed attempts on data pre-processing, setting up AWS, scoring, and waiting for convergence per attempt, yet yielding a solution markedly better (i.e. t-DCF of 0.82) than guessing randomly by expanding the number of convolutions and using LeakyReLU's. This work was a starting ground for my own interests in audio classification. Future work will explore a non-TimeDistributed convolutional layer with FC outputs equal to the number of inputs of the LSTM, perhaps stacked. Teaming up would also have greatly benefited so that multiple EC2 instances could explore faster, deeper and wider, plus expand on the feature extractor. Also, code more developed to customized functions in Keras to determine a more robust generalization to the validation set. Lastly, future work would focus on externalizing the algorithm to fit in a real-time framework, so as to create real-world value.

7 Contributions

Much kudos to my TA - Shahab Mousavi - for the back and forths! And, the Department of Computer Science for the $500 AWS credits! It was great to learn about GPU's and utilizing the cloud.

8 References

To Get ASVspoof Data

[1] Automatic Speaker Verification (ASV): Spoofing And Countermeasures Challenge. 2019. https://www.asvspoof.org/

Code & Libraries

Tensorflow, Keras, Librosa

[1] Evert, D. 2020. Code for CS230 Project. Github page. https://github.com/TrickBoarder/cs230

Papers

[1] Kinnuen, et al. t-DCF: a Detection Cost Function for the Tandem Assessment of Spoofing Countermeasures and Automatic Speaker Verification. Odyssey 2018. https://arxiv.org/abs/1804.09618

[2] M. Sahidullah, T. Kinnunen and C. Hanilci. A comparison of features for synthetic speech detection. Proc. INTERSPEECH 2015, pp. 2087–2091

[3] Phan, H. et al. Audio Scene Classification with Deep Recurrent Neural Networks. Institute for Signal Processing. https://arxiv.org/pdf/1703.04770.pdf

[4] Hershey S., et al. CNN Architectures For Large-Scale Audio Classification. Google, Inc., New York, NY, and Mountain View, CA, USA https://arxiv.org/pdf/1609.09430v2.pdf

[5] Here's how much cybercrime can cost your company https://www.fm-magazine.com/news/2019/may/cybercrime-costs-201920981.html

[6] Lavrentyeva1, G., et al. 2017. Audio replay attack detection with deep learning frameworks. ITMO University, St.Petersburg, Russia. INTERSPEECH2017 *https://pdfs.semanticscholar.org/a2b4/c396dc1064fb90bb5455525733733c761a7f.pdf*

[7] Chettri, B., et al. 2018. A Study On Convolutional Neural Network Based End-To-End Replay. Queen Mary University of London, United Kingdom.

[8] Sahidullah, Md., et al. 2015. A Comparison of Features for Synthetic Speech Detection. School of Computing, University of Eastern Finland, Finland. INTERSPEECH2015.

[9] Bekir, B., et al. 2018. An Experimental Study on Audio Replay Attack Detection Using Deep Neural Networks. IEEE.

[10] Zhizheng, W., et al. 2016. ASVspoof: the Automatic Speaker Verification Spoofing and Countermeasures Challenge. IEEE. https://www.asvspoof.org/papers/IEEE$_{JS}TSP_{A}SVspoof.pdf$

[11] Vacher, M., et al. 2007. Sound Classification in a Smart Room Environment: An approach using GMM and HMM methods.Team GEOD, UMR CNRS-INPG-UJF 5524, 385, rue de la Bibliothèque,BP 53, 38041 Grenoble cedex 9. http://www-clips.imag.fr/geod/User/michel.vacher/Pdf/2007$_{S}ped_{V}acher.pdf$