Speaker Spoofing Detection with Convolutional Neural Networks

April 20, 2018

1 Introduction

The ASVspoof initiative was created to promote the development of countermeasures which aim to protect automatic speaker verification (ASV) from spoofing attacks. The first community-led, common evaluation held in 2015 focused on countermeasures for speech synthesis and voice conversion spoofing attacks. Arguably, however, it is replay attacks which pose the greatest threat. Such attacks involve the replay of recordings collected from enrolled speakers in order to provoke false alarms and can be mounted with greater ease using everyday consumer devices. ASVspoof 2017, the second in the series, hence focused on the development of replay attack countermeasures. In this project you will use the data for the ASVspoof 2017 challenge [3, 2]. You will learn the weights of a convolutional neural network (CNN) from the train data and you will use this CNN to classify the test data as *genuine* or *spoof*.

2 Data

You can get the data from the Edinburgh DataShare at:
https://datashare.is.ed.ac.uk/handle/10283/3035
Also, you may get a baseline system from http://www.asvspoof.org/. The baseline system, which is also included in this project zip file, uses as features the Constant Q Cepstral Coefficients [6] and trains two Gaussian Mixture Models [1] one for genuine and one for spoof utterances. In order to decide,

whether a test utterance is genuine or spoof, the log-likelihoods of the Cepstral Coefficients with respect to the two trained models is calculated. Then the decision is based on the comparison of the two log-likelihoods, taking into account possible bias thresholds. For example in the competition the evaluation was based on Equal Error Rate (EER) [3]. An implementation of this measure is included in the baseline code. To simplify the project we will not consider a bias threshold.

Download all project files in: `http://www.telecom.tuc.gr/patreco/res/projects/03_SpeakerSpoofing/`

3 Project description

In ASVSpoof 2017 competition many classification algorithms have been investigated [4]. In this project you will try the challenge with a deep convolutional neural network (CNN). The input to CNN will be filter banks. A Python script is provided to help you calculate filter banks from the .wav files.

- Open the python script *make_fbanks.py* and set the *base_dir* directory path. Then run the script three times; one for each *train_phase*: *train*, *dev* and *eval*. The filter banks are saved as a matrix of dimensions $n\_frames \times n\_fbanks$, where n_frames is the number of frames and $n\_fbanks = 64$ is the number of filter banks.

- Note that, in our CNN implementation the filter bank features are needed as the transpose *numpy* matrix of dimension $n\_fbanks \times n\_frames$.

A frame of 64 filter banks is produces every 5 ms. Speech is a sequence in which current samples are correlated with the past and future samples. This correlation is also reflected in the derived sequence of filter bank frames. In order to encode the time correlation, a block of frames, in this project 17 frames (8 past frame + current frame + 8 future frames), is spliced (stacked) together to form a 2D image of size (64). This image is used in the place of the current frame. Figure **??** shows how this is done for an example with 3 context frames. You should create the 64×17 images when you read the utterances. Note that, at the beginning and at the end of the utterance you repeat the current frame. For each image you should keep an integer label, which takes the value 1 when the utterance is "genuine" and the value

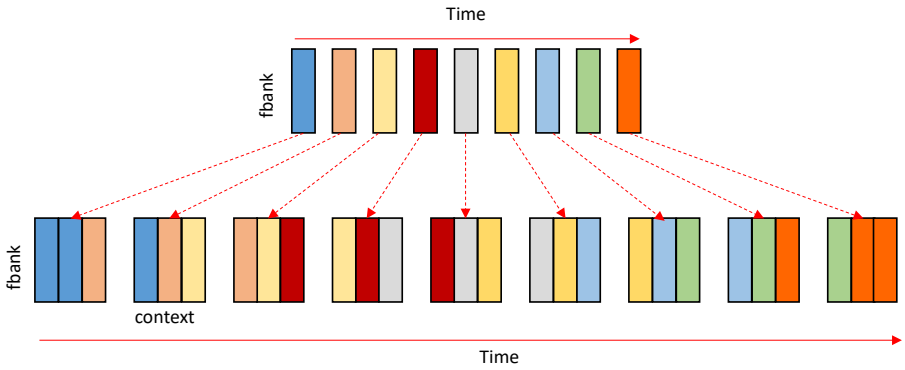

Figure 1: Example: Two dimensional splicing of three consecutive frames. At the beginning of a utterance the first frame is repeated to the left and at the end of a utterance the last frame is repeated to the right.

0 when the utterance is "spoof". All images of a utterance will have the same label. In order to improve the classification score the images should be presented to the classifier in randomized order. You may do this by collecting all images of all train utterances in one huge list L. For example the train list will contain 1587420 images. Then at the start of each new epoch you must shuffle (randomly permute the elements of) the list (see Fig. **??**). Note that the corresponding labels should also be shuffled the same way so that to preserve the labelling of images. There is no need to shuffle the validation images and the test images.

You may implement the CNN in any language and library you want. However, in order to help you, part of a working TensorFlow code is provided. You may complete this code according to the following instructions:

- For each *train* utterance, read the utterance, create the 64×17 images, and append these images to a *train_params* list.
- For each *dev* (validation) utterance, read the utterance, create the 64×17 images, and append these images to a *valid_params* list.
- The input to the CNN should be a batch of $batch\_size = 256$ images and a batch of the corresponding labels.
- Let $n\_elements$ be the number of images in train list L, then $n\_batches = \lfloor n\_element/batch\_size \rfloor$ is the number of batches. We do not consider

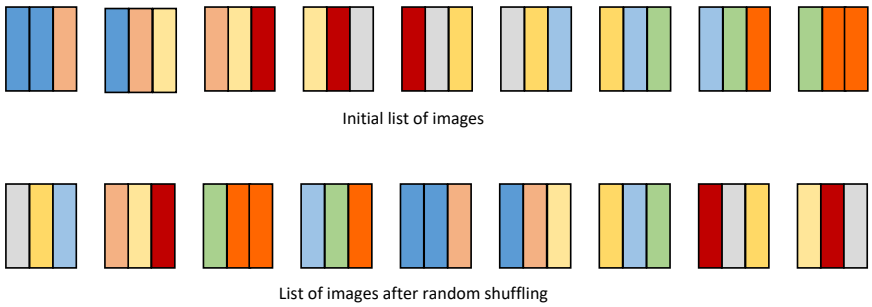

Figure 2: At the beginning of each epoch the list of images is randomly shuffled.

possible remaining images ($remainder = n\_elements \mod batch\_size$). For example, there are 6200 batches in the training set.

- Similarly for the validation list.
- When you define the computation graph the filter bank input should be a *tf.placeholder* of type *ft.float32* and shape ($batch\_size \times height \times width \times channels$), where $batch\_size = 256$, $height = 64$, $width = 17$ and $channels = 1$. The input to labels should be a *tf.placeholder* of type *ft.int32* and shape ($batch\_size$,).
- There should be separate place-holders for train and validation data.
- When you evaluate the computation graph through a session you should iterate through all train batches and feed each one (together with the batch of corresponding labels) into the network using a feeding dictionary (*feed_dict*). An iteration through all data is called an epoch.
- The same when you evaluate the validation part of the graph.
- Use the validation score to decide if the and you should stop the iterations.

3.1 Network Architecture

You will define the network architecture in *inference* function. A useful reference is the work of Qian et al. [5].

- Define a few blocks consisting of one or more convolutional layers, followed by a max pooling layer.
- All convolutions will use 3×3 filters and $padding =' SAME'$.
- The input image has one channels. The convolution layers will progressively increase the number of output channels up to value 64.
- All max pooling will have $padding =' VALID'$.
- Some max pooling layers will have $ksize = [1, 2, 1, 1]$ and $strides = [1, 2, 1, 1]$. These layers will reduce the height of the layer's input by 2.
- Some max pooling layers will have $ksize = [1, 2, 2, 1]$ and $strides = [1, 2, 2, 1]$. These layers will reduce both the height and the width of the layer's input by 2.
- When the width and the height of the output of the above blocks are both ≤ 2 and the number of channels is 64, insert a flatten layer.
- Then insert one or more dense layers.
- Use the ReLU non-linearity for all the layer but the flatten and the last one.
- Note that the last layer will not use a non-linearity function. Instead, its output will pass through the softmax function in other parts of the code.

3.2 Apply a Trained Model to Test Samples

When a network is trained its weights are saved to disk. In evaluation time the weights are reloaded from the disk.

- Iterate through all test utterances. Convert each test utterance to a sequence of 64 images.
- In a session run the inference to calculate the output of the network, which will be a matrix $utterance\_length \times n\_classes$, where $n\_classes = 2$.

- Apply the softmax to each line to convert the output to a matrix of probabilities.
- Then take the log of each line.
- Then compute the sum of first column, l_0, and the sum of second column, l_1.
- Decide *spoof* if $l_0 \geq l_1$, else decide *genuine*.
- Since you have the correct labels of the test utterances you can calculate the accuracy and F_1 measures.

3.3 Bonus

- Normalize the filter bank features by element-wise subtracting the mean and divide with the standard deviation of the train utterances.
- Note that, you should use the mean and standard deviation of train utterances to normalize the train, dev and eval utterances.
- Since the first 3 and last 3 frames of a utterance may contain noise, do not use these frames in the calculation of the mean and standard deviation.
- Try to use dropout in the CNN architecture
- Try to use batch normalization in the CNN architecture.

4 Project deliverables

You must provide your own implementation of CNNs. You must also write a report where you provide all your results (plots, tables, etc.) and comments on the project questions. Include an introduction where you explain the mathematical background of your implementations. Your report should provide sufficient details that are needed to future readers in case they want to repeat your experiments. Provide explanations about your empirical design decisions (if any), the insights you learn in intermediate analysis steps, as well as your final results.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[2] T. Kinnunen, M. Sahidullah, M. Falcone, L. Costantini, R. G. Hautamäki, D. Thomsen, A. Sarkar, Z. H. Tan, H. Delgado, M. Todisco, N. Evans, V. Hautamäki, and K. A. Lee. Reddots replayed: A new replay spoofing attack corpus for text-dependent speaker verification research. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5395–5399, March 2017.

[3] Tomi Kinnunen, Nicholas Evans, Junichi Yamagishi, Kong Aik Lee, Md Sahidullah, Massimiliano Todisco, and Héctor Delgado. ASVspoof 2017: Automatic Speaker Verification Spoofing and Countermeasures Challenge Evaluation Plan. Technical report, 2017.

[4] Tomi Kinnunen, Md Sahidullah, Héctor Delgado, Massimiliano Todisco, Nicholas Evans, Junichi Yamagishi, and Kong Aik Lee. The ASVspoof 2017 challenge: Assessing the limits of replay spoofing attack detection. In *INTERSPEECH 2017, Annual Conference of the International Speech Communication Association, August 20-24, 2017, Stockholm, Sweden*, Stockholm, SWEDEN, 08 2017.

[5] Y. Qian, M. Bi, T. Tan, and K. Yu. Very deep convolutional neural networks for noise robust speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(12):2263–2276, 2016.

[6] Massimiliano Todisco, Hector Delgado, and Nicholas Evans. A new feature for automatic speaker verification anti-spoofing: Constant Q cepstral coefficients. In *ODYSSEY 2016, The Speaker and Language Recognition Workshop, June 21-24, 2016, Bilbao, Spain*, Bilbao, SPAIN, 06 2016.