Vysoké učení technické v Brně
Fakulta informačních technologií

SUR

Target detection from audio and image data

April 20, 2024 Vojtěch Eichler (xeichl01), Adam Zvara (xzvara01)

Target detection from audio

For the target detection from audio files we have decided to use techniques discussed in lectures, namely calculating MFCC coefficients, which are then utilized in two Gaussian mixture models to classify target and non-target speakers. While the availability of a small dataset for training isn't inherently problematic (since we are using generative models), we aim to improve accuracy on new and potentially transformed data with simple data augmentation techniques.

Data preprocessing and augmentation

Data preprocessing and augmentation tasks have been facilitated by the `torchaudio` library, which offers a straightforward API[1] for applying transformations on data and extracting features. The basic augmentation pipeline contains the following steps:

1. Resampling: if audio file has different sample rate than expected (16kHz), resample it to this sample rate.
2. Removing silence: recordings sometimes contain large portion of completely silent passages or parts with quiet background noise (especially at the beginning and the end), which are not useful for target detection, as the model might even learn to identify target based on it's surroundings. Therefore a fixed window of 1.8 seconds is removed from the beginning of the recording and the ending is dynamically trimmed based on an amplitude threshold.
3. Applying effects (optional): each effect is applied stochastically, and multiple effects can be simultaneously applied to the same recording.
4. Extracting MFCC coefficients

During training (and evaluation) we have opted for the following effects (the parameters and probabilities were chosen randomly):

- low frequency filter in range $\langle 200, 1000 \rangle$
- high frequency filter in range $\langle 800, 3000 \rangle$
- tempo change in range $\langle 0.7, 1.3 \rangle$
- reverb
- gain in range $\langle -5, 20 \rangle$ (asymmetric as the recordings are already quiet)

Model, training and evaluation

Initially, both GMMs were trained on the training data as provided by the assignment. However, it quickly became apparent that using a different subset for training returns significantly different results. Therefore, we have decided to evaluate each model on multiple samples – first, we concatenate all target and non-target datasets together and utilize `StratifiedShuffleSplit`[2] to split the datasets into training (60%) and validation subsets (40%). For each configuration, the model was trained five times and the overall accuracy of the model was then calculated as the average of all accuracies obtained. In each batch, the model underwent evaluation twice. Firstly, it was assessed on the default validation dataset provided within the batch. Secondly,

[1] `https://pytorch.org/audio/stable/generated/torchaudio.sox_effects.apply_effects_tensor`

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit`

it was evaluated on the same validation batch, but with effects applied as described in the previous section.

The overall parameters of the model include: number of components for the target and non-target GMM (TC and NTC), how many times we replicate target training dataset and non-target training dataset (TR and NTR – the usage of this will be explained later) and whether to use augmentation on the training datasets (*Effects*). The parameters used to calculate the MFCC coefficients are identical to those presented in lectures: $n\_fft = 256$, $window\_size = 200$, $window\_overlap = 100$, $mel\_banks = 23$, $mfcc\_coefficients = 13$.

GMM parameters					**Validation data**		**Augmented validation data**	
TC	*NTC*	*TR*	*NTR*	*Effects*	*Target*	*Non-target*	*Target*	*Non-target*
3	10	1	1	no	1.0	0.95	0.66	0.83
3	10	1	1	yes	0.73	0.98	0.66	0.95
3	10	4	2	yes	0.86	0.97	0.81	0.96
7	15	4	2	yes	0.93	0.94	0.86	0.93
15	30	4	2	yes	1.0	0.98	0.93	0.94
15	30	4	3	yes	0.96	0.91	0.88	0.92
15	30	5	3	yes	1.0	0.93	0.93	0.95
20	40	4	2	yes	1.0	0.90	0.96	0.93

Table 1: Evaluation of audio classifier with different parameters

Table 1 presents the results of training GMMs with different parameters. We started the experiments with a lower number of components for each GMM ($TC = 3$, $NTC = 10$, $NTC > TC$ as there are more non-target speakers). In the first experiment, the training data were not augmented, resulting in a drop in accuracy for the augmented validation data. In the second experiment, we maintained the same parameters while applying augmentation to the training dataset. Notably, the target accuracy on the non-augmented dataset dropped, while on the augmented dataset, it remained consistent. This occurred because the effects were applied stochastically, meaning that if we read the training dataset only once, we obtained relatively few normal and augmented samples. To address this issue, in the third iteration the training dataset is read multiple times ($TR = 4$) to diversify the data, as different effects were applied with each reading. The non-target training file was read twice ($NTR = 2$) to achieve a similar effect, considering there are generally more non-target samples. However, despite these efforts, the target accuracy still did not improve. In the next two iterations, we increased the number of components in each GMM and the models with $TC = 15$ and $NTC = 30$ presented the best results so far. In the following experiments, we tried to increase number of components in both GMMs as well as reading training files multiple times but it did not show significantly better results.

Target detection from images

As with audio, target detection involves data augmentation and training the classifier on the augmented data. While the task itself isn't very complicated, the fact that we have quite few training examples (especially for the target) makes this challenging. The obvious choice for a classifier is convolutional neural network, which was our choice. For the CNN to learn the underlying patterns it would be very useful to have as much data as possible and as previously stated, this was not the case and that's where data augmentation comes in.

Data augmentation

Data augmentation plays a key role in our approach. Not only does it allow us to train bigger CNN and for more epochs by reducing (or even avoiding) overfitting, it also allows the model to adapt to variances in data (e.g. different lighting, different position of a face in the image, ...). We've gone through many experiments with most of the transformations that we thought would be relevant to our task and some general data augmentations which help with learning underlying features. These are some of the transformations applied and the reasoning behind using them, the illustration of these effects is illustrated in pytorch documentation[3]:

1. Random Erasing: This is the only transformation which isn't inherently inspired by the task on hand, but is generally useful, because by erasing parts of the images, as it erases parts of the images by stochastically filling rectangles in the image with zeroes, it forces the model to learn from different parts of the images.
2. Random Resized Crop: The effect of this augmentation is quite straightforward as the name suggests, the image is cropped and resized to fill the original size, which not only focuses on different parts of the images, but effectively changes the resolution as well.
3. Random Vertical and Rotation: Flips are some of the simplest augmentations for images, which primarily reduce overfitting, as we don't really expect the new images to be flipped, but even if they were, these augmentations train the model on such cases. Rotation is very similar, the image is simply randomly rotated up to 10 degrees, which we imagine could very likely happen in new images as the setup for taking photos might be tilted a bit, or the subject might tilt their head.
4. Color Jitter and Gaussian Blur: What we expect from these techniques is again to force the classifier to learn more diverse patterns rather than lighting and colors of the original images. Gaussian Blur we expect to be somewhat useful for new images as they might be distorted if the setup fails to focus on the subject's face.

Model, training and evaluation

We built a basic CNN according to the standard CNN recipe, which means Convolutional layers followed by Batch normalization, ReLU activations and Dropout, this sequence is repeated 4 times. After getting the feature maps we flattened them and fed them into first fully connected layers, followed by ReLU, dropout and one more fully connected layer which outputs a single scalar. The model has roughly 2.2M parameters. Training such model on our limited data for 150 epochs takes few minutes if ran on GPU and evaluation of the test data only requires few seconds of runtime.

We used Adam optimizer with default configuration and slightly lowered learning rate, which was experimentally shown to perform better and we achieved more stable learning that way.

The detection task is very similar to binary classification so we used BCEWithLogitsLoss, which is Binary Cross Entropy loss, which also combines sigmoid layer. This function is supposed to be more numerically stable than using plain sigmoid followed by regular BCELoss as mentioned in the pytorch documentation[4] .

The last thing that should be explained is detection threshold. In order to decide whether the image belongs to target or nontarget we had to apply sigmoid on the outputs of the model (remember the sigmoid is combined into the loss function rather than the model itself). Deciding based on whether the value is higher or lower than 0.5 wasn't feasible as this would result in

[3]https://pytorch.org/vision/0.11/auto_examples/plot_transforms.html
[4]https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

no detections at all (or very few). The chosen threshold is 0.2, this value was decided on based on many experiments.

The model was trained for 250 epochs, which was possible thanks to the already mentioned data augmentation. It's important to note that only the training data was augmented. Each epoch goes through all the training images and then evaluates the model on all validation data. The fact that the data augmentation is stochastic means that the training data is gonna vary in each epoch and that's why training over many epochs is beneficial. For this purpose, the model also contains dropout layers, which further reduces the overfitting, allowing for even more training epochs.

How to run

Make sure that all dependencies listed in requirements.txt are installed. The implementation has been verified on Python version 3.11.7. For the torchaudio library to function properly, the sox library must be installed (on Ubuntu, you can install it with the command `sudo apt install libsox-dev`). Training of audio and image models was implemented as standalone applications `train_audio_gmm.py` and `train_image_cnn.py`. After running the training scripts, the trained models are (by default) saved as `audio_classifier.pkl` and `image_classifier.pkl`. After obtaining the models it is possible to run them on the evaluation dataset with `evaluate.py`. The evaluation script can be run with `--audio` to evaluate the audio model and `--image` to evaluate the image model.

Before running the scripts, make sure that the directory structure is in the following format: