Summplementary Materials

1 ABSTRACT

Advanced Generative Adversarial Networks (GANs) are remarkable in generating intelligible audio from a random latent vector. In this paper, we examine the task of recovering the latent vector of both synthesized and real audio. Previous works recovered latent vectors of given audio through an auto-encoder inspired technique that trains an encoder network either in parallel with the GAN or after the generator is trained. With our approach, we train a deep residual neural network architecture to project audio synthesized by WaveGAN into the corresponding latent space with near identical reconstruction performance. To accommodate for the lack of an original latent vector for real audio, we optimize the residual network on the perceptual loss between the real audio samples and the reconstructed audio of the predicted latent vectors. In the case of synthesized audio, the Mean Squared Error (MSE) between the ground truth and recovered latent vector is minimized as well. We further investigated the audio reconstruction performance when several gradient optimization steps are applied to the predicted latent vector. Through our deep neural network based method of training on real and synthesized audio, we are able to predict a latent vector that corresponds to a reasonable reconstruction of real audio. Even though we evaluated our method on WaveGAN, our proposed method is universal and can be applied to any other GANs.

2 SETUP

All code is written for Tensorflow 2. WaveGAN models are stored in Tensorflow 1 format but are converted at run-time to be compatible with Tensorflow 2.

2.1 WaveGAN

In order to generate fake audio we use WaveGAN pre-trained on SC09. WaveGAN code and pre-trained models are publicly available and can be downloaded from the waveGAN's author github in this link. Save the downloaded checkpoint to */checkpoint*. The WaveGAN model is frozen to run in Tensorflow 2 at run time. The model receives as input a random vector of size 100 with a uniform distribution between -1 and 1 to generate spoken digits. During training the generated audio is created at run-time. The below command can be ran to generate example audio.

```
python generate.py
```

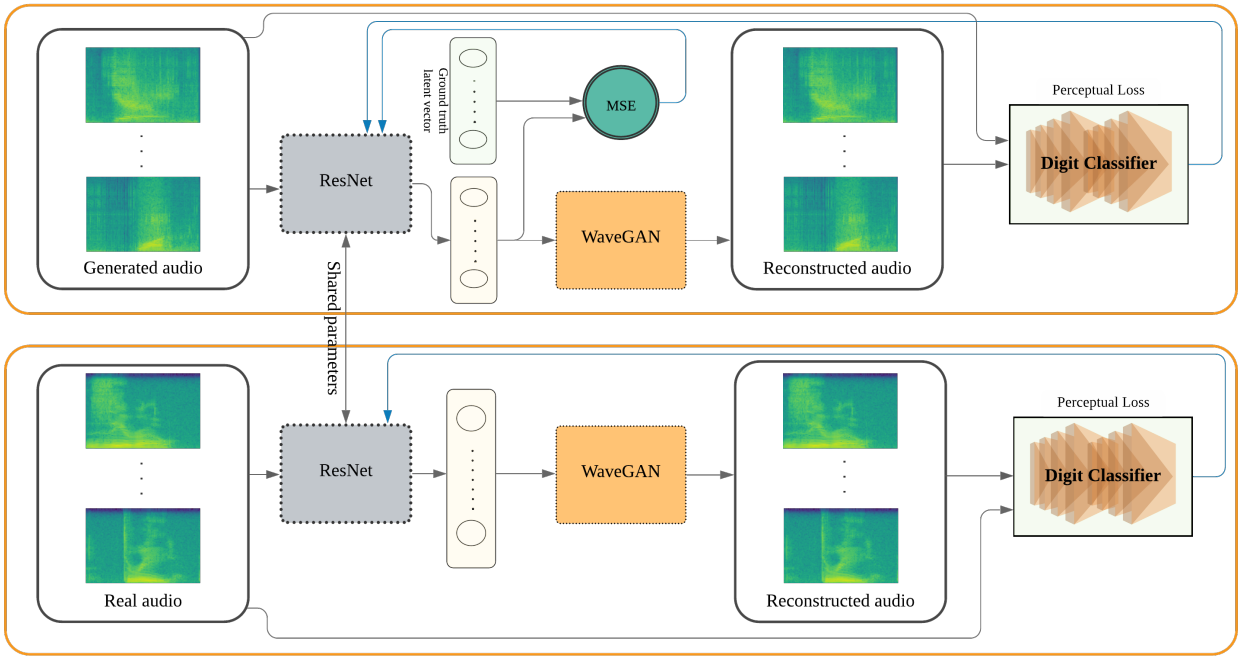
2.2 Speech Commands Digits Dataset

The dataset of spoken digits is publicly available and can be downloaded from this link. Decompress the file and save under */datasets*. This dataset is used as input to train the inverse mapping model on real data. The dataset is also used as input for calculating the metrics for real data reconstruction.

2.3 Spoken Digit Classifier

To train the classifier for spoken digit classification use the below command. The accuracy reached was 95.41% on the validation set. A pre-trained classifier is also available at this link. Once downloaded, copy the model to */classifier_checkpoint*.

```
python train_sc09_classifier.py
```



2.4 Inverse Mapping Model

The inverse mapping model can be trained as detailed in the training section or the pre-trained model can be downloaded here. Copy the decompressed files to */inverse_mapping_checkpoint*.

3 TRAINING

Train the inverse mapping model with the command below. The training logs are saved in tensorboard format in */logs*. The inverse mapping model is saved at the end of each epoch in */inverse_mapping_model_checkpoint*. Sample audio with the corresponding reconstruction is saved in */waves* at the end of each epoch to measure progress qualitatively.

```
python train.py
```

The inverse mapping model is trained using two objective functions. A diagram for how the model is trained is shown above. For all audio, the perceptual loss between the original audio and reconstructed audio is minimized. The perceptual loss is quantified as the MAE between the output activations at each residual block of the Spoken Digit Classifier. For synthesized audio, the MSE between the true latent vector and the predicted latent vector is also minimized to learn the inverse mapping.

4 TESTING

4.1 Inverting Audio

To invert audio of 1 second, use the command below. In the file latent_vector_recovery.py change the files variable to point towards the directory where audio data is stored. The reconstruction will be saved in */waves*. To use the hybrid method, uncomment the line that uses scipy.optimize.minimize.

```
python latent_vector_recovery.py
```

4.2 Metrics

The metrics shown in the paper can be regenerated with the command below. To alter which metric is being calculated, change the flags "gradient", "inverse_mapping", and "synth". Synth controls whether real or synthesized audio is used. Inverse_mapping controls whether the inverse mapping model is used to predict the latent vector. Gradient is used to apply gradient-based methods to the prediction. Gradient and Inverse_mapping can both be set to true to use a hybrid method. The maximum number of steps can be set in the optimize.minimize function call with max_iter.

```
python calculate_metrics.py
```

5 EXAMPLES

Examples of reconstructed audio using each method for real and synthesized audio can be found in the */Audio_Examples* folder. These examples are the same as shown in the paper.