Neural Networks, Data Augmentation, and Speech Recognition

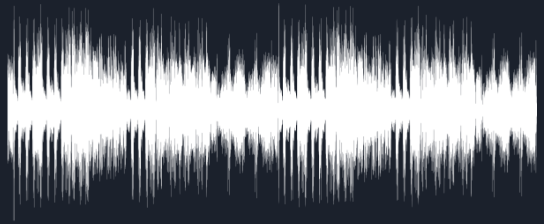
Grayson Cordell
Jesse Gailbreath
Noah Norrod
Jacob Swindell

Background

- Speech recognition through data augmentation
- Data Augmentation - taking a set of data and changing it somehow
- 3,000 files into over 30,000
- Will augmentation make the network train better?

Sound Explained

- Sound is made by changes in air pressure
- Compression and rarefaction
- How fast this happens is called frequency, which determines pitch
- How extreme the compression and rarefaction is determines amplitude

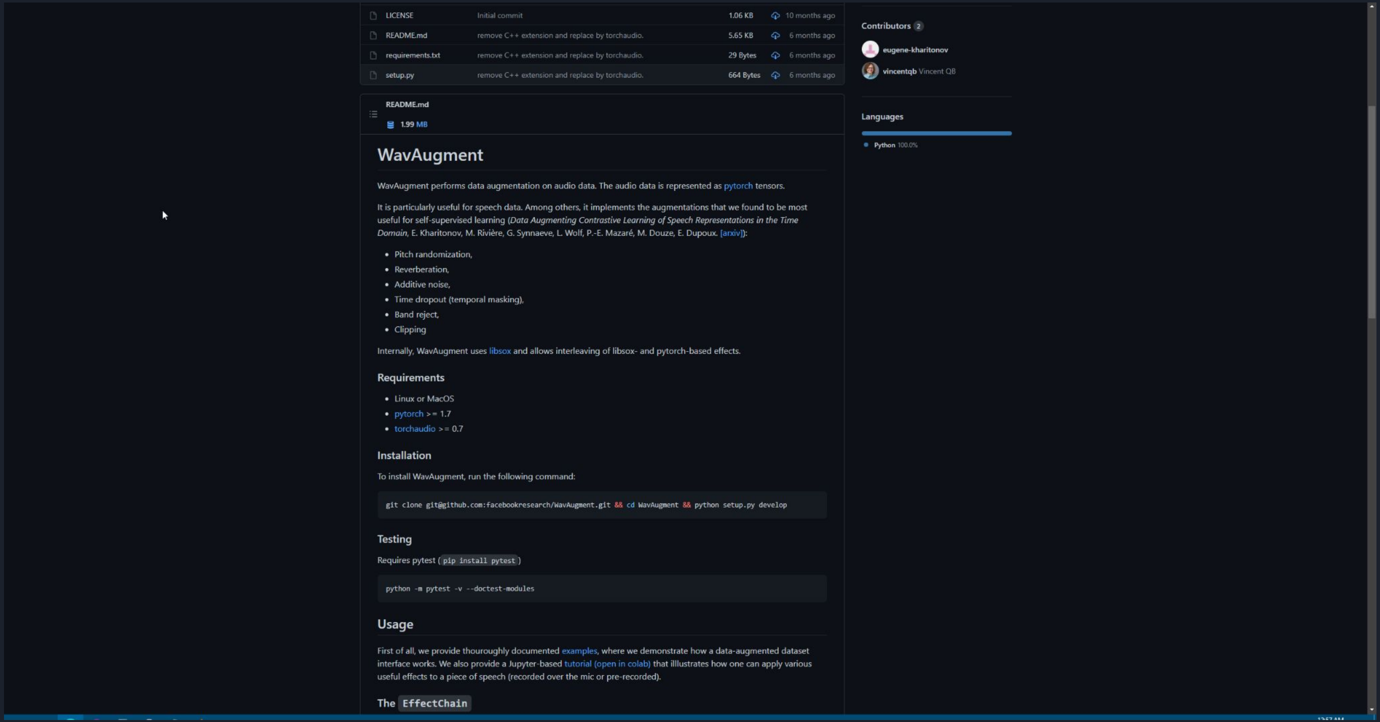


Example drawing of a sound wave

Sound Processes Used

- Multiple processes used to augment the data set
- Pitch shift: altering frequency to make higher or lower sounds
- Background noise: gives slight distinction from original sound
- Reverb: the sound of a given room

Our Library



LICENSE	Initial commit	1.06 KB	10 months ago
README.md	remove C++ extension and replace by torchaudio.	5.65 KB	6 months ago
requirements.txt	remove C++ extension and replace by torchaudio.	29 Bytes	6 months ago
setup.py	remove C++ extension and replace by torchaudio.	664 Bytes	6 months ago

Contributors 2

eugene-kharitonov

vincentqb Vincent QB

Languages

Python 100.0%

README.md

1.99 MB

WavAugment

WavAugment performs data augmentation on audio data. The audio data is represented as pytorch tensors.

It is particularly useful for speech data. Among others, it implements the augmentations that we found to be most useful for self-supervised learning (*Data Augmenting Contrastive Learning of Speech Representations in the Time Domain*, E. Kharitonov, M. Rivière, G. Synnaeve, L. Wolf, P.-E. Mazaré, M. Douze, E. Dupoux. [arxiv]):

- Pitch randomization,
- Reverberation,
- Additive noise,
- Time dropout (temporal masking),
- Band reject,
- Clipping

Internally, WavAugment uses libsox and allows interleaving of libsox- and pytorch-based effects.

Requirements

- Linux or MacOS
- pytorch >= 1.7
- torchaudio >= 0.7

Installation

To install WavAugment, run the following command:

```
git clone git@github.com:facebookresearch/WavAugment.git && cd WavAugment && python setup.py develop
```

Testing

Requires pytest (`pip install pytest`)

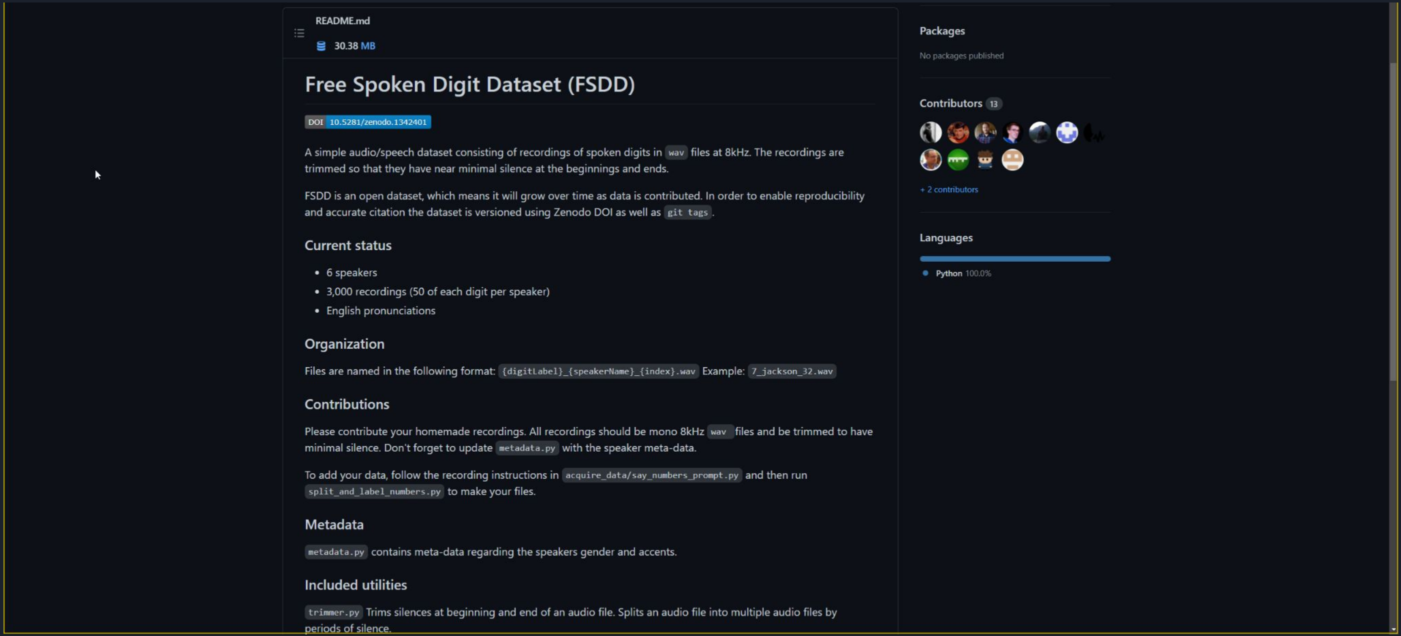
```
python -m pytest -v --doctest-modules
```

Usage

First of all, we provide thouroughly documented examples, where we demonstrate how a data-augmented dataset interface works. We also provide a Jupyter-based tutorial (open in colab) that illustrates how one can apply various useful effects to a piece of speech (recorded over the mic or pre-recorded).

The `EffectChain`

Our Dataset



README.md

30.38 MB

Free Spoken Digit Dataset (FSDD)

DOI 10.5281/zenodo.1342401

A simple audio/speech dataset consisting of recordings of spoken digits in `wav` files at 8kHz. The recordings are trimmed so that they have near minimal silence at the beginnings and ends.

FSDD is an open dataset, which means it will grow over time as data is contributed. In order to enable reproducibility and accurate citation the dataset is versioned using Zenodo DOI as well as `git tags`.

Current status

- 6 speakers
- 3,000 recordings (50 of each digit per speaker)
- English pronunciations

Organization

Files are named in the following format: `{digitLabel}_{speakerName}_{index}.wav` Example: `7_jackson_32.wav`

Contributions

Please contribute your homemade recordings. All recordings should be mono 8kHz `wav` files and be trimmed to have minimal silence. Don't forget to update `metadata.py` with the speaker meta-data.

To add your data, follow the recording instructions in `acquire_data/say_numbers_prompt.py` and then run `split_and_label_numbers.py` to make your files.

Metadata

`metadata.py` contains meta-data regarding the speakers gender and accents.

Included utilities

`trimmer.py` Trims silences at beginning and end of an audio file. Splits an audio file into multiple audio files by periods of silence.

Packages

No packages published

Contributors 13

+ 2 contributors

Languages

Python 100.0%

All Augmentations
(and Their Spectrograms)

1. Clipped
2. Pitch -200 cents
3. Pitch +200 cents
4. Random pitch 1 (-400 to +400 cents)
5. Random pitch 2 (-400 to +400 cents)
6. Random reverb 1
7. Random reverb 2
8. Dropout
9. Additive noise
10. Band stop filter

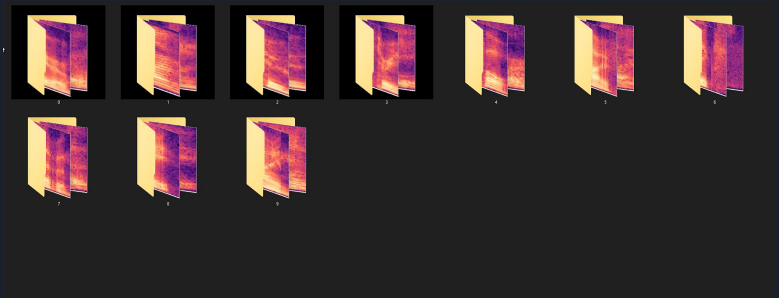
Developing the Spectrograms

```
[ ] def spectrogram(samples, sample_rate, stride_ms = 10.0,
                          window_ms = 20.0, max_freq = 8192, eps = 1e-14):

        stride_size = int(0.001 * sample_rate * stride_ms)
        window_size = int(0.001 * sample_rate * window_ms)

        # Extract strided windows
        truncate_size = (len(samples) - window_size) % stride_size
        samples = samples[:len(samples) - truncate_size]
        nshape = (window_size, (len(samples) - window_size) // stride_size + 1)
        nstrides = (samples.strides[0], samples.strides[0] * stride_size)
        windows = np.lib.stride_tricks.as_strided(samples,
                                              shape = nshape, strides = nstrides)

        assert np.all(windows[:, 1] == samples[stride_size:(stride_size + window_size)])

        # Window weighting, squared Fast Fourier Transform (fft), scaling
        weighting = np.hanning(window_size)[:, None]

        fft = np.fft.rfft(windows * weighting, axis=0)
        fft = np.absolute(fft)
        fft = fft**2

        scale = np.sum(weighting**2) * sample_rate
        fft[1:-1, :] *= (2.0 / scale)
        fft[(0, -1), :] /= scale

        # Prepare fft frequency list
        freqs = float(sample_rate) / window_size * np.arange(fft.shape[0])

        # Compute spectrogram feature
        ind = np.where(freqs <= max_freq)[0][-1] + 1
        specgram = np.log(fft[:ind, :] + eps)
        return specgram
```

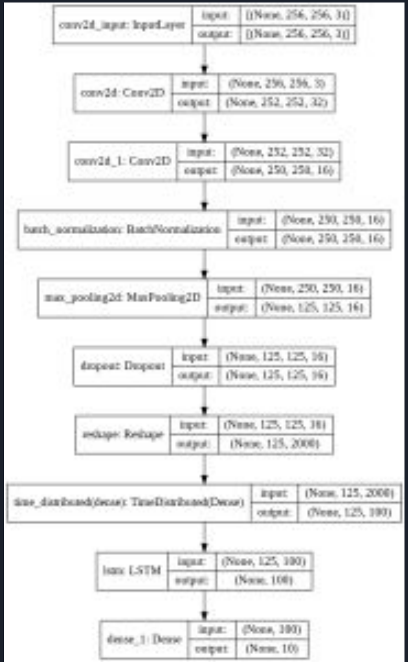
```
[ ] def SpecSaver(name):
      spec=spectrogram(x2, sr2)
      librosa.display.specshow(spec)
      plt.savefig(name, bbox_inches='tight', transparent=True, pad_inches=0.0 )
```

```
directory = r'/content/s21-team7-project/free-spoken-digit-dataset-master/free-spoken-digit-dataset-master/recordings'
for filename in os.listdir(directory):
  if filename.endswith(".wav"):
    d=os.path.join(directory, filename)
    Base=os.path.basename(filename)
    x2, sr2 =librosa.load(d, sr=None)
    newone = "sImage"+Base +".png"
    SpecSaver(newone)
```

- Our code for generating spectrograms and the folders in which they are stored



Model Architecture




```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 252, 252, 16)      1216
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 250, 250, 16)      2320
_________________________________________________________________
batch_normalization (BatchNo (None, 250, 250, 16)      64
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 125, 125, 16)      0
_________________________________________________________________
dropout (Dropout)            (None, 125, 125, 16)      0
_________________________________________________________________
reshape (Reshape)            (None, 125, 2000)         0
_________________________________________________________________
time_distributed (TimeDistri (None, 125, 50)           100050
_________________________________________________________________
lstm (LSTM)                  (None, 50)                20200
_________________________________________________________________
dense_1 (Dense)              (None, 10)                510
=================================================================
Total params: 124,360
Trainable params: 124,328
Non-trainable params: 32
```

Training the Network

- Original Results vs. Augmented Results

```
# now test the model against the unseen images from the original dataset
ods_history_test = model.evaluate(ods_test, verbose = 1)
print("Using only original samples:")
print("Accuracy: ", ods_history_test[1])
print("Loss: ", ods_history_test[0])
```

```
90/90 [==============================] - 4s 38ms/step - loss: 0.6467 - categorical_accuracy: 0.8033
Using only original samples:
Accuracy:  0.8033333420753479
Loss:  0.6467224955558777
```

```
# now let's test with unseen images from the augmented plus original dataset
plusAugmented_history_test = model.evaluate(plusAugmented_ds_test, verbose = 1)
print("Using the augmented plus original samples:")
print("Accuracy: ", plusAugmented_history_test[1])
print("Loss: ", plusAugmented_history_test[0])
```

```
990/990 [==============================] - 40s 41ms/step - loss: 0.1218 - categorical_accuracy: 0.9619
Using the augmented plus original samples:
Accuracy:  0.9619191884994507
Loss:  0.12183058261871338
```

- In this example, we got .80 accuracy with .64 loss against .96 accuracy with .12 loss

Thank You for Listening!

Grayson Cordell - Architecture Lead

Jesse Gailbreath - Code Lead

Noah Norrod - Documentation Lead

Jacob Swindell - Presentation Lead

We will now take any questions!