laughr

...

jeffgreenca

Summary

Data Preprocessing

librosa (features)

Keras (model)

Results

Fails and Futures

About Me

software engineer, IT

musician, experimenter,

fan of Frasier, new dad

Talk Focus

- Audio data preprocessing
- High level libraries:

  Keras, LibROSA

- Lessons Learned

Material Availability

Slides, code, notes...

jeffgreenca @ GitHub

jeff.green.ca@gmail.com

The Dinner Party


AT THE END OF THIS STORY,
WILL I ROLL MY EYES?

The Question

Does the audience laugher

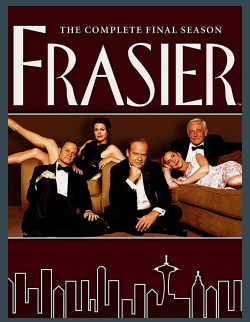
enhance or detract?

Goal: Mute audience laughter

Motivation

ML

"Nail"



Key Consideration

Live studio audience,

laughing,

during the performance.

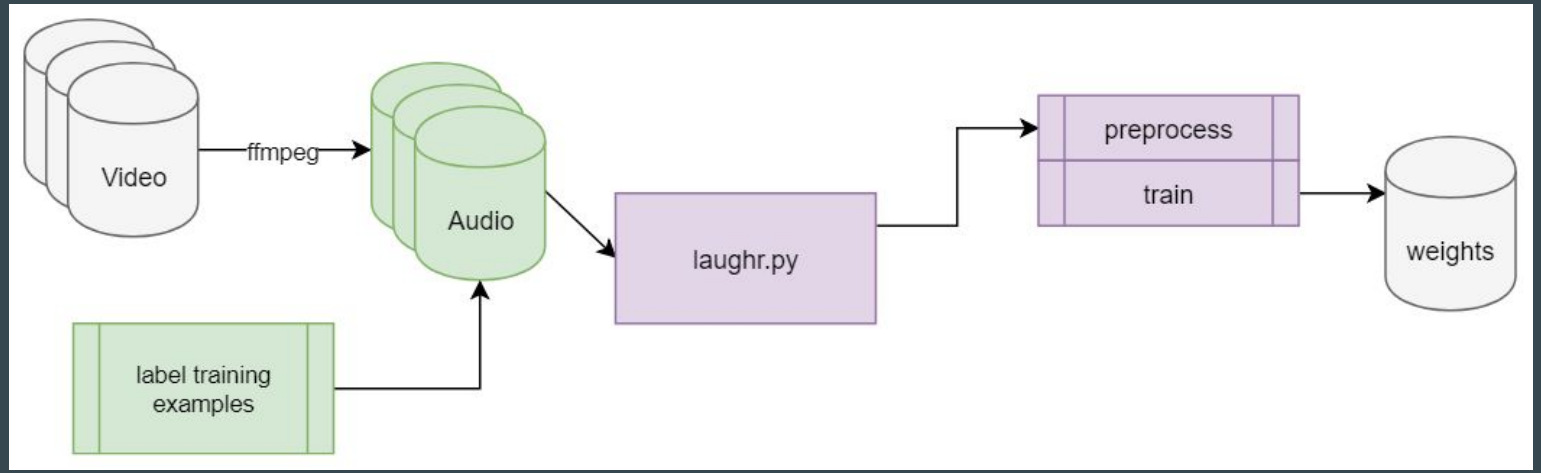
Limits

- Classify and just mute
- vs. "noise cancellation"
- Fun is mandatory

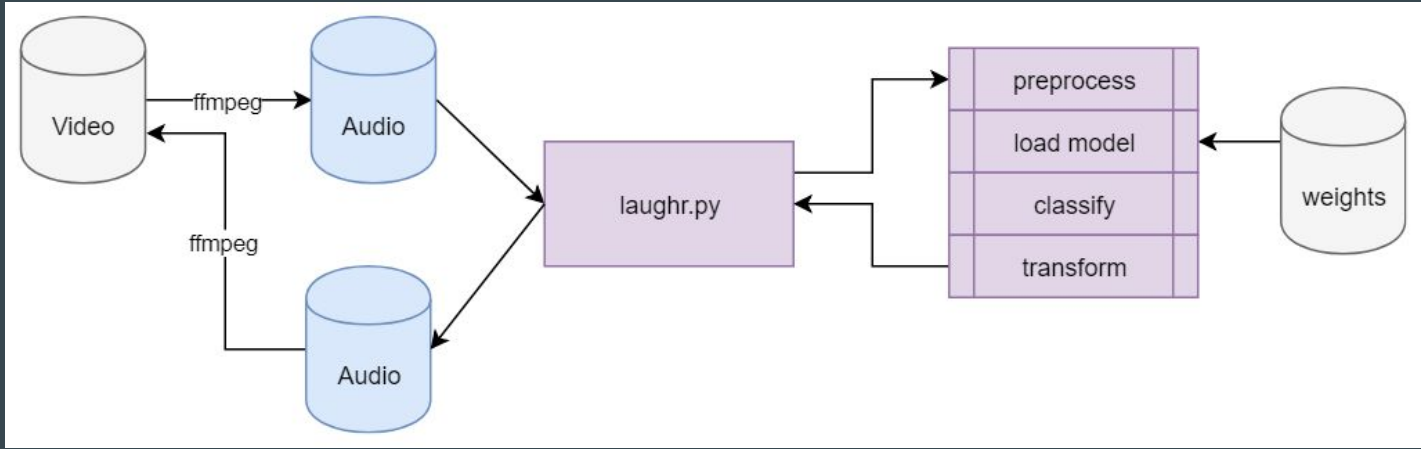
Tools

- Python
- Jupyter
- Open source libraries
- Google (so much)

Training Mode



Mute Laughs Mode (Apply)



Workflow

1. Start with domain-specific tools
2. Search for libraries
3. Write linear code (notebook)
4. Group into functions, classes (Repeat 1-4)
5. Finally, wrap in a CLI

laughr.py

```
usage: laughr.py [-h] --model MODEL.h5
                 [--train /path/to/L*.wav /path/to/D*.wav]
                 [--mute-laughs SOURCE.wav OUTPUT.wav]
```

Summary

Data Preprocessing

librosa (features)

Keras (model)

Results

Fails and Futures

Dataset Basics

Collection of MP4 videos

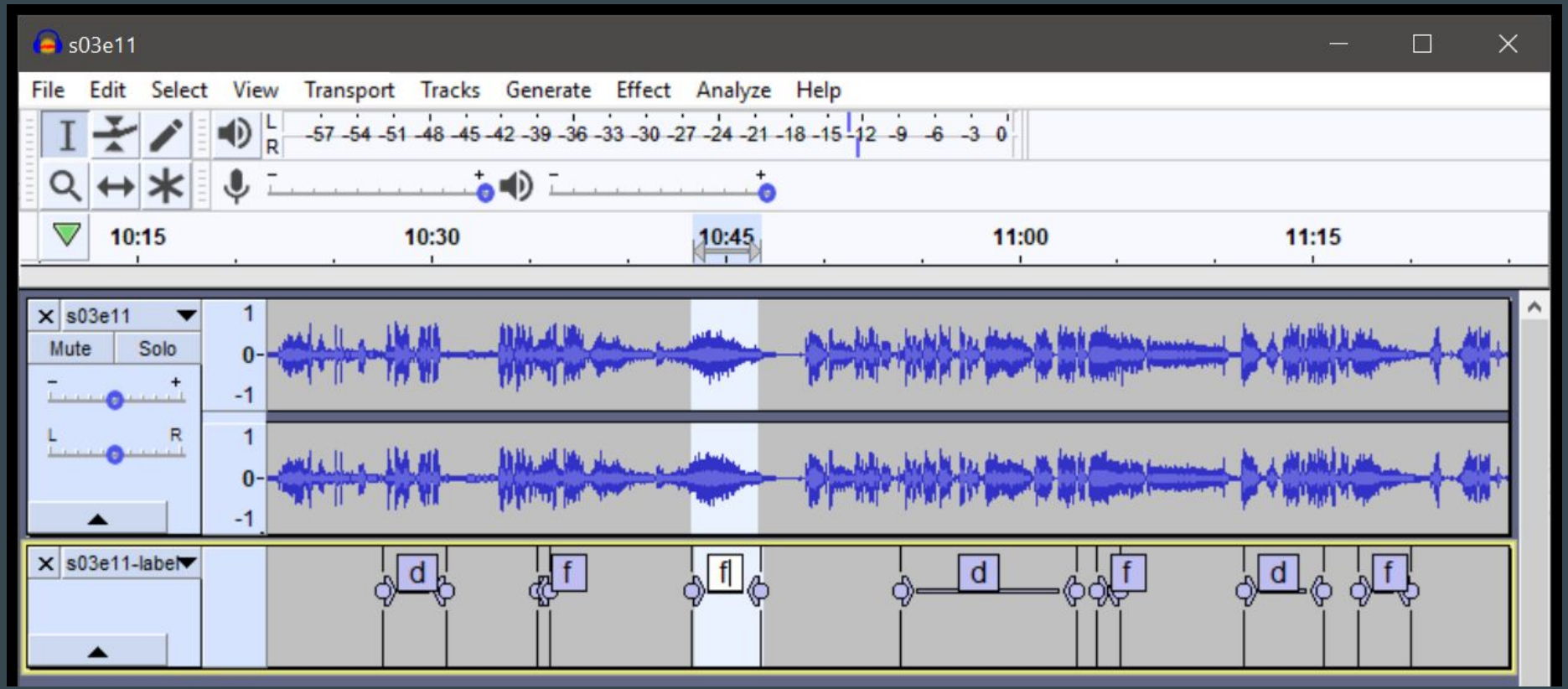
MP4 > ffmpeg > WAV

Split into clips

Training Set

Using Audacity...

1. Load WAV file
2. Define clips
3. Export to files


s03e11
File Edit Select View Transport Tracks Generate Effect Analyze Help
-57 -54 -51 -48 -45 -42 -39 -36 -33 -30 -27 -24 -21 -18 -15 -12 -9 -6 -3 0
10:15
10:30
10:45
11:00
11:15
s03e11
Mute
Solo
s03e11-label
d
f
f
d
f
d
f

Training Set

Selected across episodes

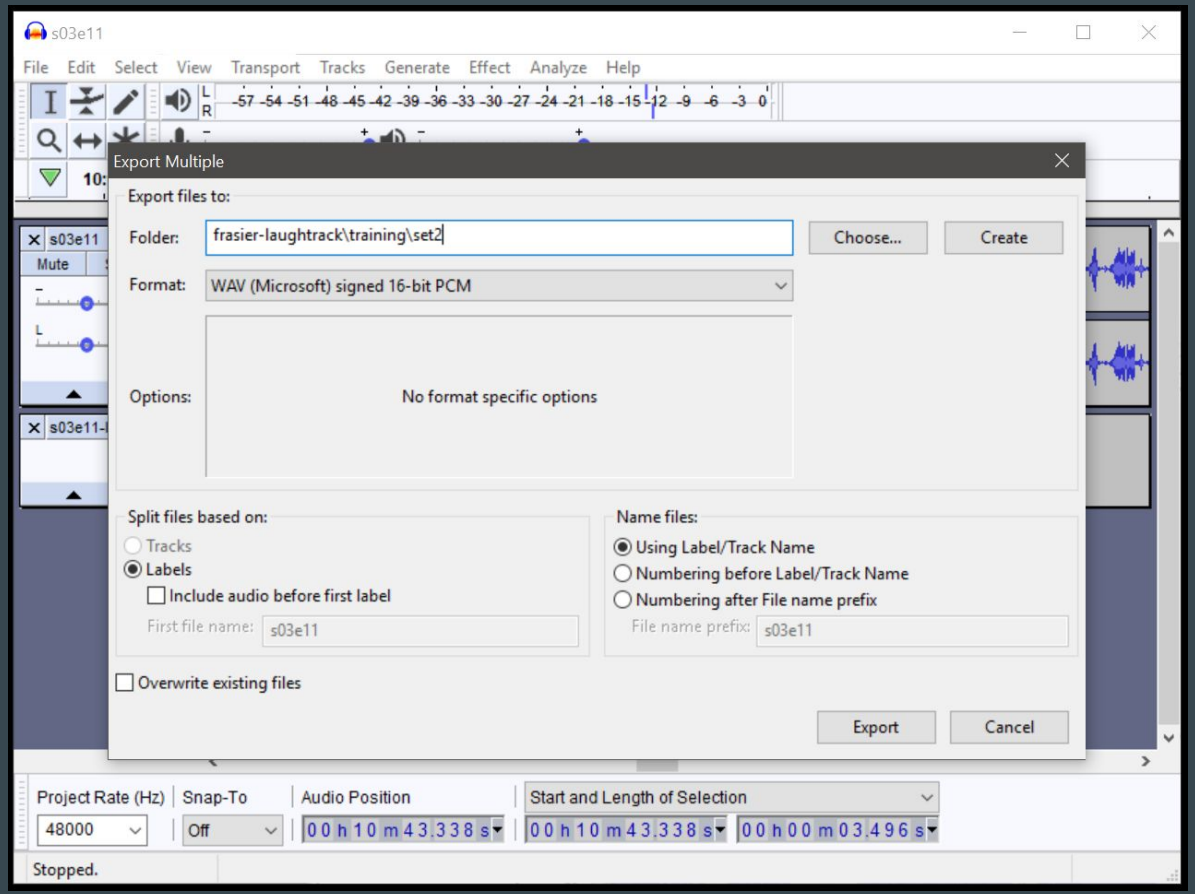
Was not super systematic

Mildly fun!

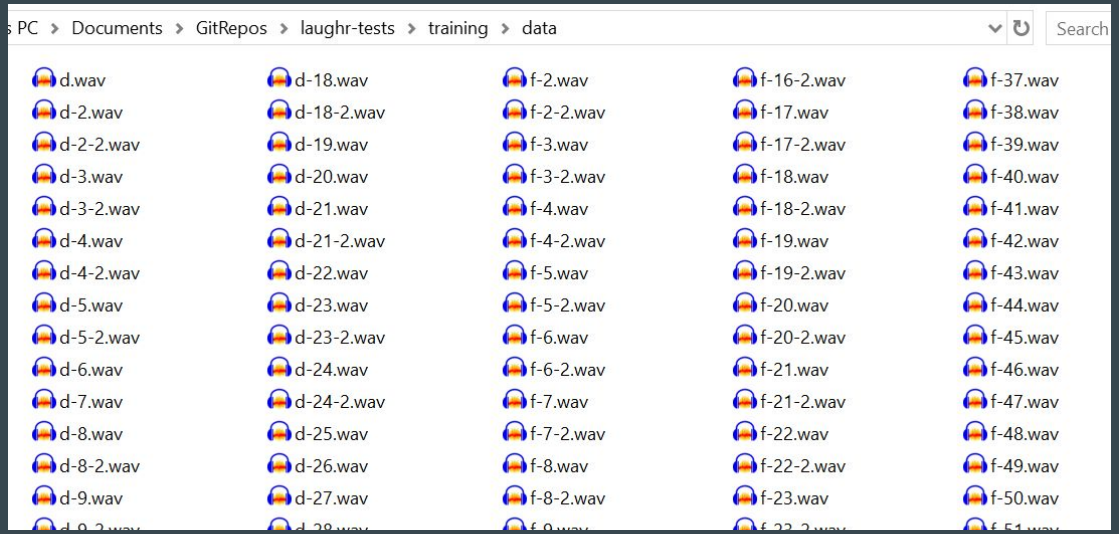
To go faster

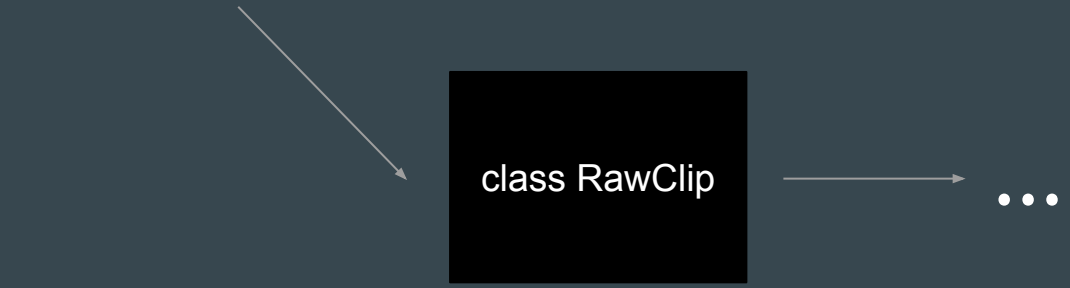
Use Shortcut

Use "Export Multiple"


s03e11
File Edit Select View Transport Tracks Generate Effect Analyze Help
Export Multiple
Export files to:
Folder: frasier-laughtrack\training\set2
Choose...
Create
Format: WAV (Microsoft) signed 16-bit PCM
Options: No format specific options
Split files based on:
Tracks
Labels
Include audio before first label
First file name: s03e11
Name files:
Using Label/Track Name
Numbering before Label/Track Name
Numbering after File name prefix
File name prefix: s03e11
Overwrite existing files
Export
Cancel
Project Rate (Hz)
Snap-To
Audio Position
Start and Length of Selection
48000
Off
00 h 10 m 43.338 s
00 h 10 m 43.338 s
00 h 00 m 03.496 s
Stopped.

Result:


PC › Documents › GitRepos › laughr-tests › training › data
Search
d.wav
d-2.wav
d-2-2.wav
d-3.wav
d-3-2.wav
d-4.wav
d-4-2.wav
d-5.wav
d-5-2.wav
d-6.wav
d-7.wav
d-8.wav
d-8-2.wav
d-9.wav
d-18.wav
d-18-2.wav
d-19.wav
d-20.wav
d-21.wav
d-21-2.wav
d-22.wav
d-23.wav
d-23-2.wav
d-24.wav
d-24-2.wav
d-25.wav
d-26.wav
d-27.wav
f-2.wav
f-2-2.wav
f-3.wav
f-3-2.wav
f-4.wav
f-4-2.wav
f-5.wav
f-5-2.wav
f-6.wav
f-6-2.wav
f-7.wav
f-7-2.wav
f-8.wav
f-8-2.wav
f-16-2.wav
f-17.wav
f-17-2.wav
f-18.wav
f-18-2.wav
f-19.wav
f-19-2.wav
f-20.wav
f-20-2.wav
f-21.wav
f-21-2.wav
f-22.wav
f-22-2.wav
f-23.wav
f-37.wav
f-38.wav
f-39.wav
f-40.wav
f-41.wav
f-42.wav
f-43.wav
f-44.wav
f-45.wav
f-46.wav
f-47.wav
f-48.wav
f-49.wav
f-50.wav


laugh.wav
class RawClip
...

Digital Recording

Audio signals are converted into a stream of discrete numbers, representing changes over time in air pressure

- Wikipedia

Two Considerations

Bit Depth

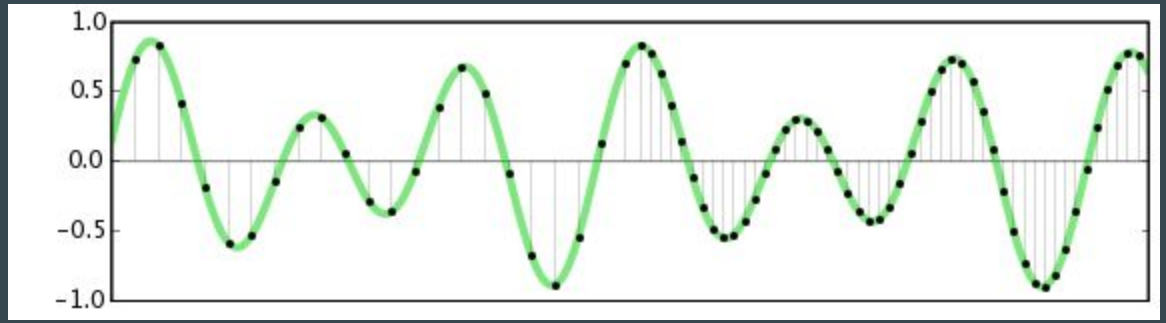
Sample Rate

Bit Depth

...I ignored this

Sample Rates

Hz = samples per second



http://ipthsc.wikifoundry.com/page/Characteristics+of+Multimedia+Systems

Sample Rates

48kHz == 48000 samples / sec

y = 1 second of audio

len(y) == 48000

PySoundFile

Audio File to NumPy array

```
import soundfile as sf

data, sample_rate = sf.read()
```

Example

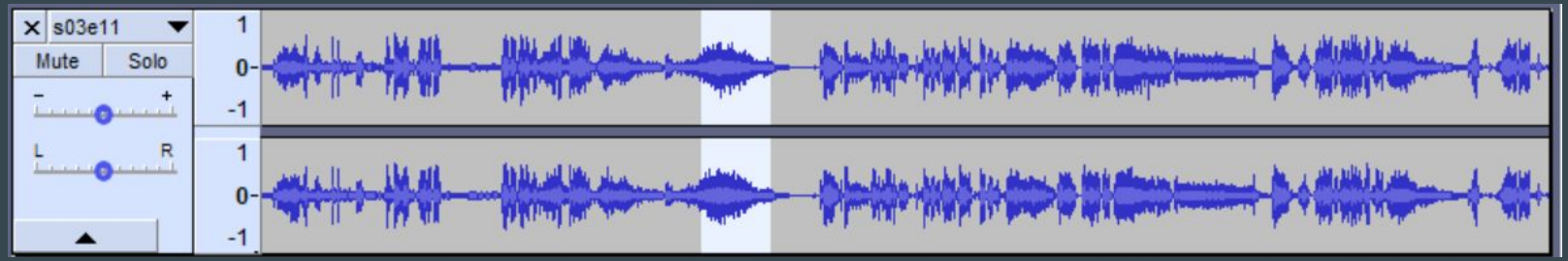
```
y, sr = sf.read("laugh.wav")
```

y.shape == (63218, 2)

sr == 48000 (48kHz)

~1.3 seconds

Discard stereo right



Discard stereo right

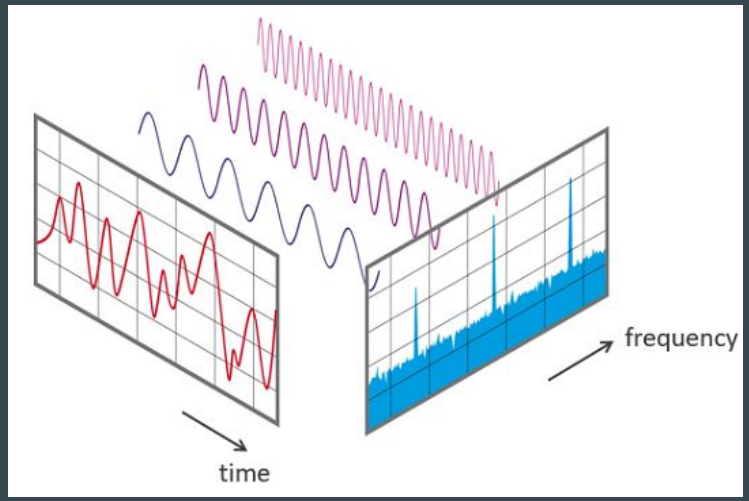
y.shape = (63218, 2)

y.T[0].shape = (63218)

Why discard one track?

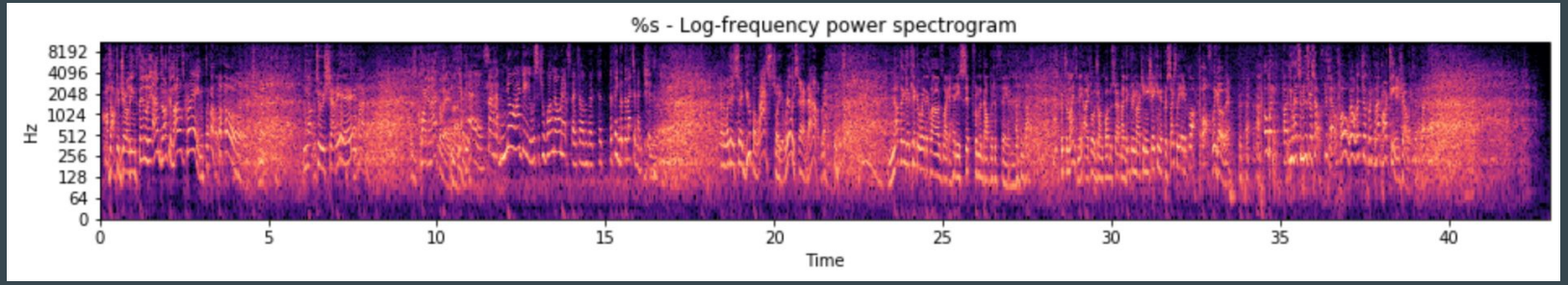
laugh.wav

Plotting Spectra



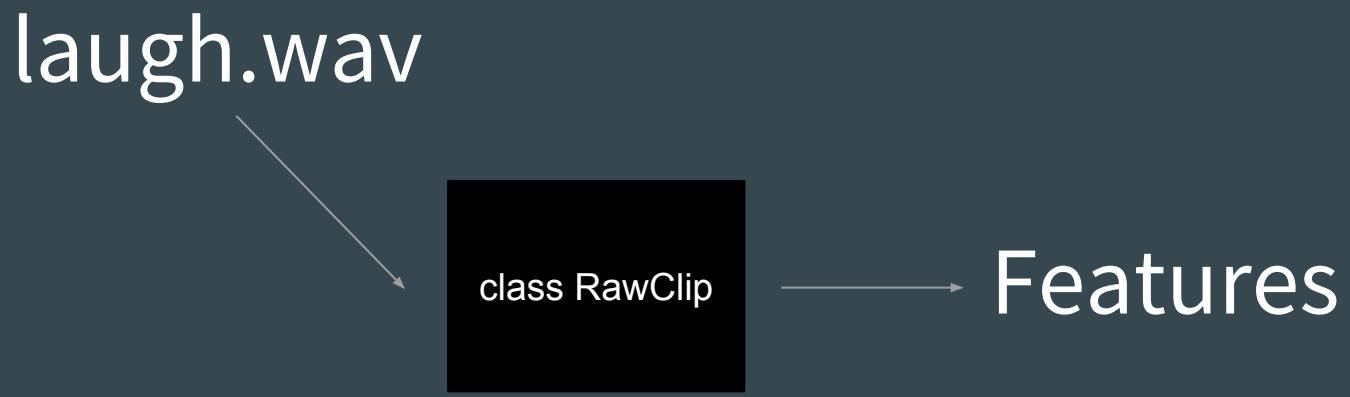
Wikipedia - Fast Fourier transform

Spectrogram



Time scale...!

Summary
Data Preprocessing
librosa (features)
Keras (model)
Results
Fails and Futures


laugh.wav
class RawClip
Features

librosa

0.6

Search docs

- Installation instructions
- Tutorial
- Core IO and DSP
- Display
- Feature extraction
- Onset detection
- Beat and tempo
- Spectrogram decomposition
- Effects

Docs » LibROSA View page source

LibROSA

LibROSA is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems.

For a quick introduction to using librosa, please refer to the Tutorial. For a more advanced introduction which describes the package design principles, please refer to the librosa paper at SciPy 2015.

Goal: Build a feature array

suitable for our model

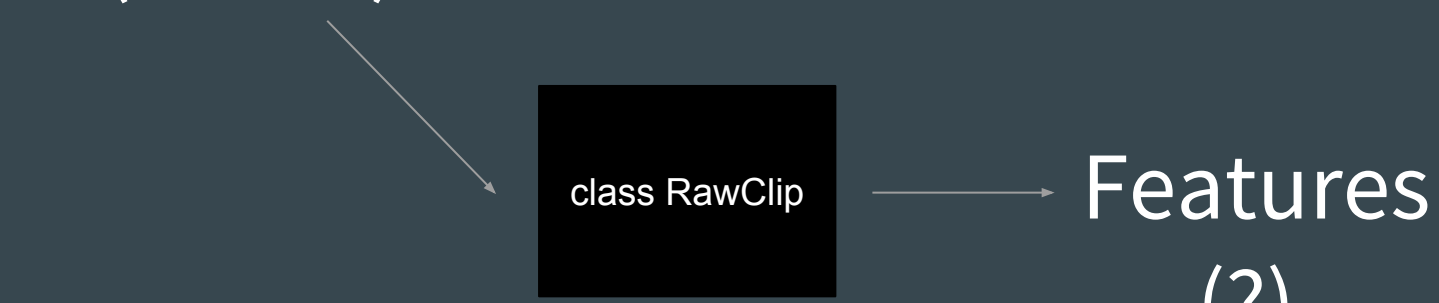
Spectral features

`chroma_stft` ([y, sr, S, norm, n_fft, ...])	Compute a chromagram from a waveform or power spectr
`chroma_cqt` ([y, sr, C, hop_length, fmin, ...])	Constant-Q chromagram
`chroma_cens` ([y, sr, C, hop_length, fmin, ...])	Computes the chroma variant “Chroma Energy Normalized
`melspectrogram` ([y, sr, S, n_fft, ...])	Compute a mel-scaled spectrogram.
`mfcc` ([y, sr, S, n_mfcc, dct_type, norm])	Mel-frequency cepstral coefficients (MFCCs)
`rmse` ([y, S, frame_length, hop_length, ...])	Compute root-mean-square (RMS) energy for each frame,
`spectral_centroid` ([y, sr, S, n_fft, ...])	Compute the spectral centroid.
`spectral_bandwidth` ([y, sr, S, n_fft, ...])	Compute p’th-order spectral bandwidth:
`spectral_contrast` ([y, sr, S, n_fft, ...])	Compute spectral contrast [R6ffcc01153df-1]
`spectral_flatness` ([y, S, n_fft, hop_length, ...])	Compute spectral flatness
`spectral_rolloff` ([y, sr, S, n_fft, ...])	Compute roll-off frequency.
`poly_features` ([y, sr, S, n_fft, hop_length, ...])	Get coefficients of fitting an nth-order polynomial to the c
`tonnetz` ([y, sr, chroma])	Computes the tonal centroid features (tonnetz), following
`zero_crossing_rate` (y[, frame_length, ...])	Compute the zero-crossing rate of an audio time series.

Use all the features (mostly)

```
featureFuncs =  ['tonnetz',
                 'spectral_rolloff',
                 'spectral_contrast',
                 'spectral_bandwidth',
                 'spectral_flatness',
                 'mfcc',
                 'chroma_cqt',
                 'chroma_cens',
                 'melspectrogram']
```

```
# For this chunk, run all of our feature extraction functions
# Each returned array is in the shape (features, steps)
# Use concatenate to combine (allfeatures, steps)
chunkFeatures = np.concatenate(
    list(
        map(self._extract_feature, self.featureFuncs)
    )
)
```

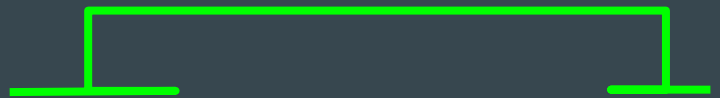

samples / length
(63218)
class RawClip
Features
(?)

Dimensions

(63218) > f1() -> (6, 124)

(63218) > f2() -> (1, 124)

Samples to Steps



(63218) > f1() -> (6, 124)

(63218) > f2() -> (1, 124)

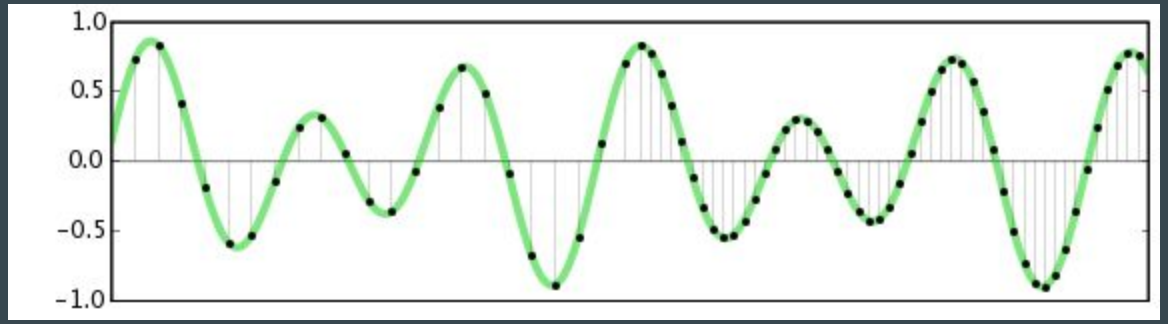
About 10 milliseconds per step

Features per Func

(63218) > f1() -> (6, 124)

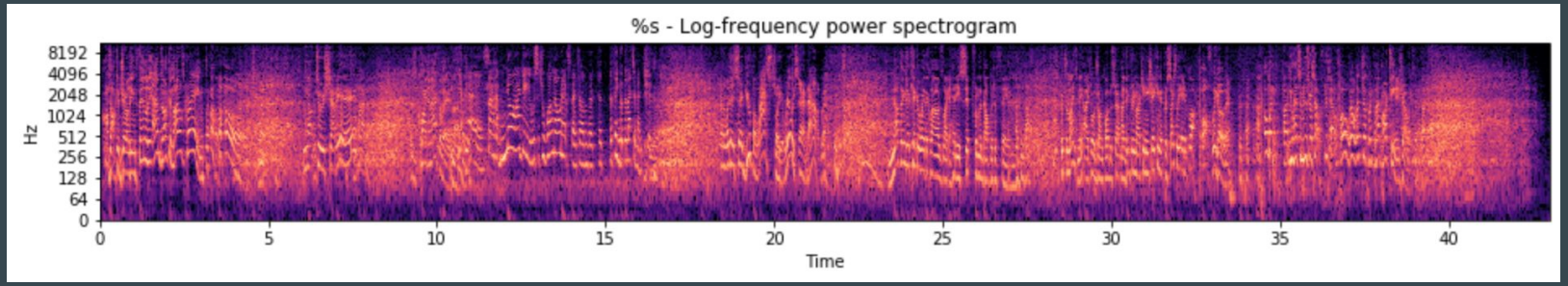
(63218) > f2() -> (1, 124)

From time, pressure:



http://ipthsc.wikifoundry.com/page/Characteristics+of+Multimedia+Systems

...to time, frequency, amplitude:

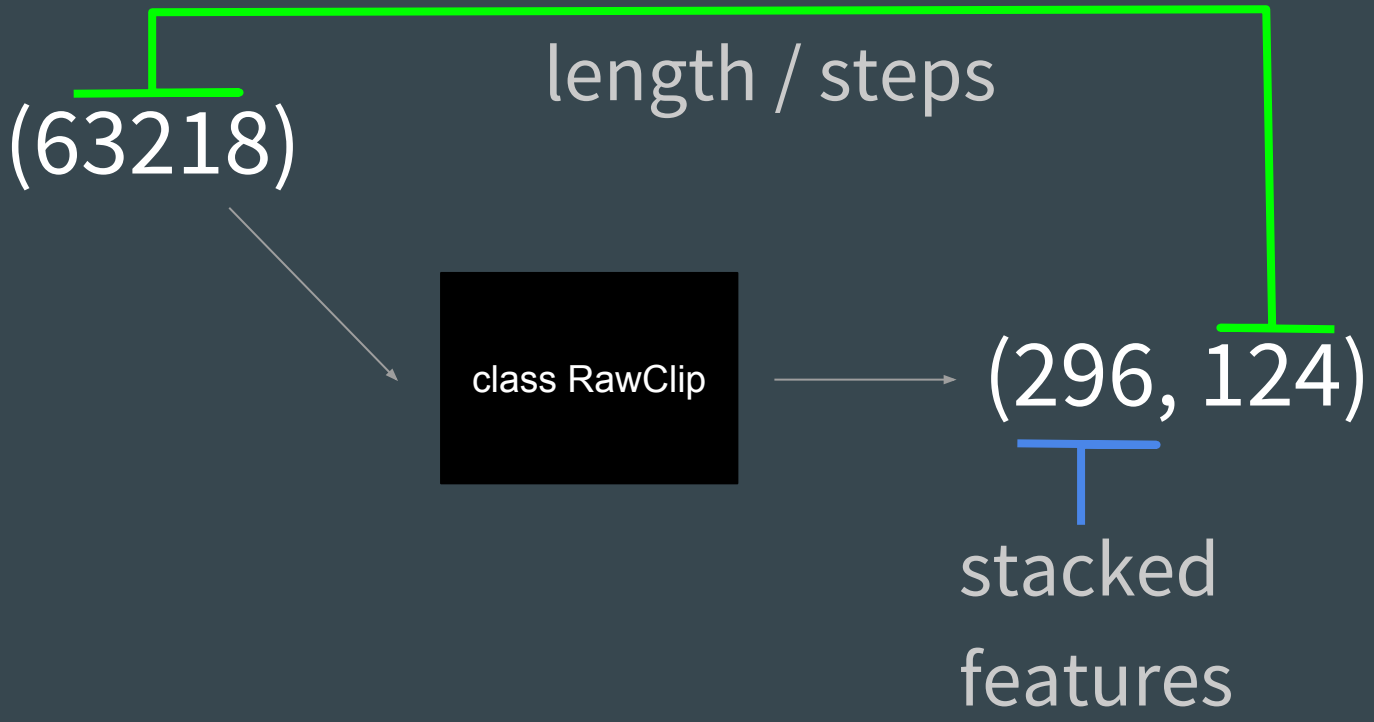


Stack Features

(63218) > f1() -> (6, 124)

+ (63218) > f2() -> (1, 124)

+ (63218) > f3() -> (100, 124)


length / steps
(63218)
class RawClip
(296, 124)
stacked
features

Tweak

x.shape = (features, steps)

x.T.shape = (steps, features)

From clips to a dataset

One hot encoding

Laugh: [1, 0]

not Laugh: [0, 1]

A Granularity Question

Length of clips varies

What time length do we classify?

A Granularity Question...

Remember,

63218 samples --> 124 steps

About 10ms per step

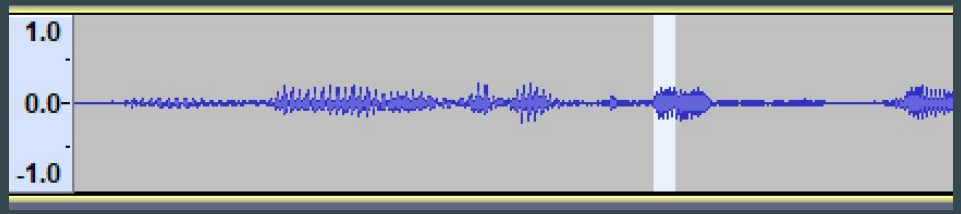
...A Granularity Question...

10ms is very short

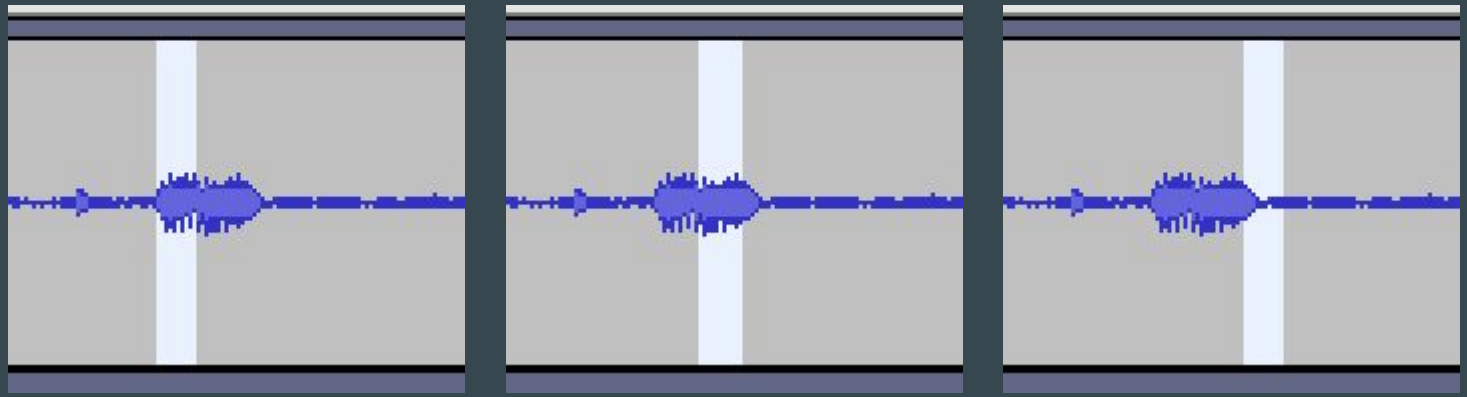
to identify sound...

Window Size

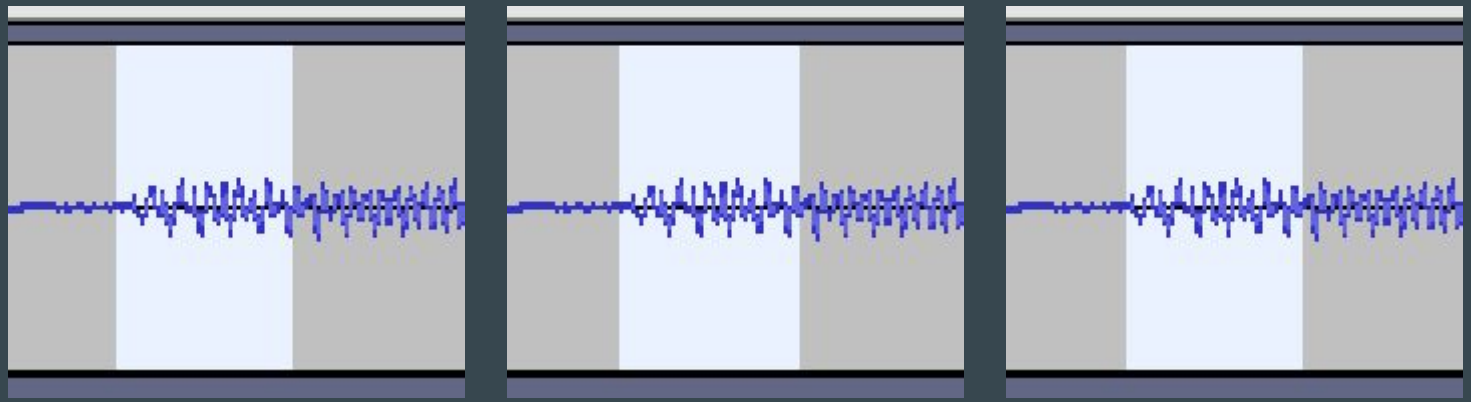
30 step ~ 300ms



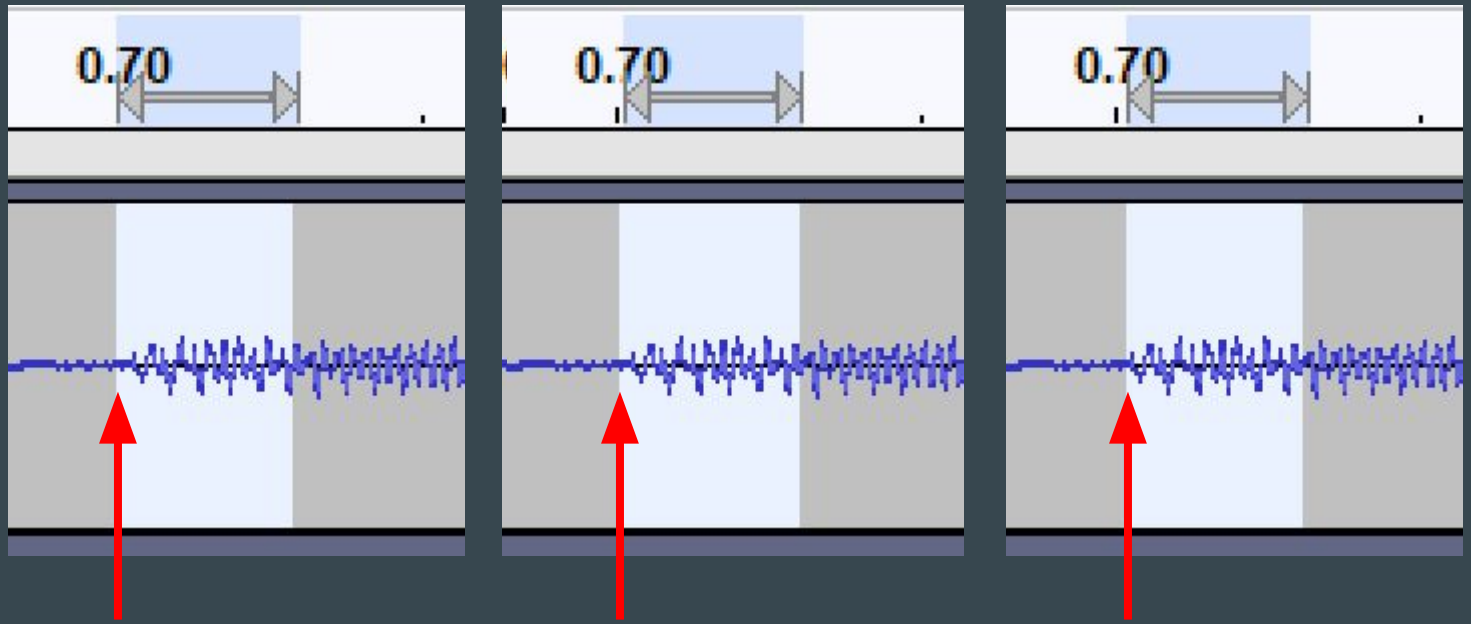
Sequential Windows?



Rolling Windows!



Classify 1ms: window start

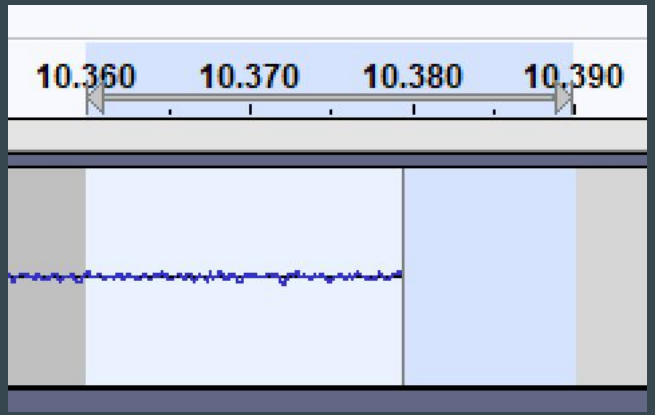


Window Size Considerations

Smaller - better granularity

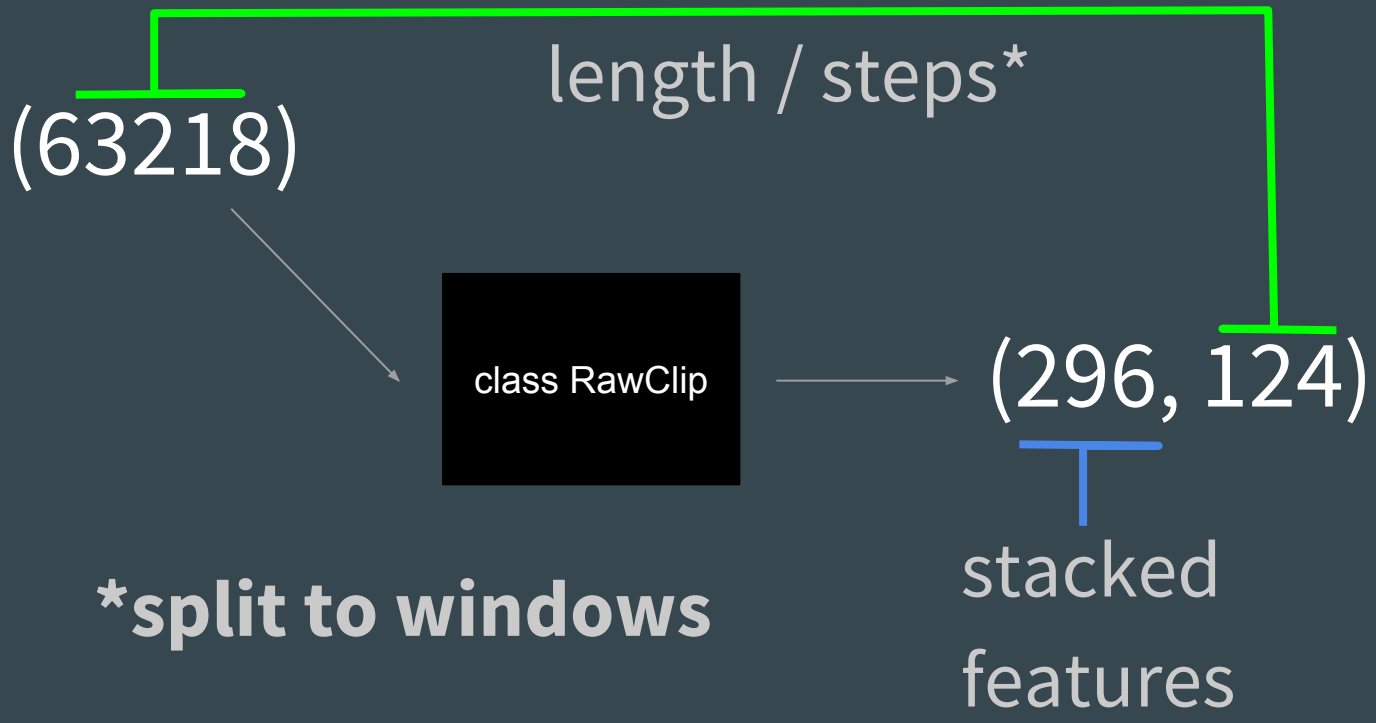
Larger - easier to classify?

The last window...?

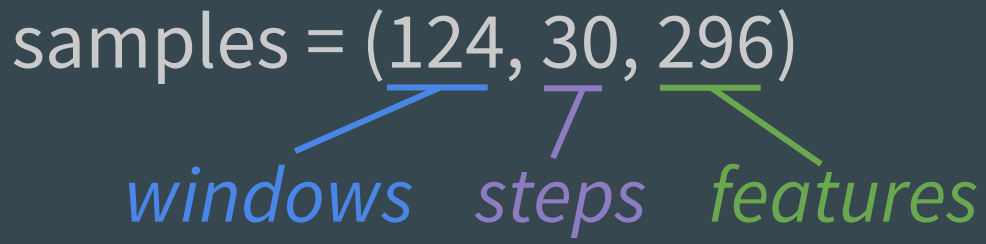


Don't do this, probably

```
# Pad the rightmost edge by repeating frames, simplifies stretching
# the model predictions to the original audio later on.
data = np.pad(data, [[0, duration], [0, 0]], mode='edge')
```


length / steps*
(63218)
class RawClip
(296, 124)
stacked features
*split to windows

RawClip Feature Array



Concat the clips

X == (11245, 30, 296)

windows *steps* *features*

Y_class == one hot for 11245 examples

```
[        _get_samples()] [DEBUG] number of clips=13
[      build_features()] [DEBUG] called on training/set2/ff-6.wav
[      build_features()] [DEBUG] numChunks=1
[      build_features()] [DEBUG] self.raw shape=150143
[    _extract_feature()] [DEBUG] extracting feature tonnetz
[    _extract_feature()] [DEBUG] feature dimensions=(6, 294)
[    _extract_feature()] [DEBUG] extracting feature spectral_rolloff
[    _extract_feature()] [DEBUG] feature dimensions=(1, 294)
[    _extract_feature()] [DEBUG] extracting feature spectral_contrast
[    _extract_feature()] [DEBUG] feature dimensions=(7, 294)
[    _extract_feature()] [DEBUG] extracting feature spectral_bandwidth
[    _extract_feature()] [DEBUG] feature dimensions=(1, 294)
[    _extract_feature()] [DEBUG] extracting feature spectral_flatness
[    _extract_feature()] [DEBUG] feature dimensions=(1, 294)
[    _extract_feature()] [DEBUG] extracting feature mfcc
[    _extract_feature()] [DEBUG] feature dimensions=(128, 294)
[    _extract_feature()] [DEBUG] extracting feature chroma_cqt
[    _extract_feature()] [DEBUG] feature dimensions=(12, 294)
[    _extract_feature()] [DEBUG] extracting feature chroma_cens
[    _extract_feature()] [DEBUG] feature dimensions=(12, 294)
[    _extract_feature()] [DEBUG] extracting feature melspectrogram
[    _extract_feature()] [DEBUG] feature dimensions=(128, 294)
[      build_features()] [DEBUG] chunkFeatures shape=(296, 294)
[      build_features()] [DEBUG] shape of np.concatenate(features, axis=1).T=(294, 296)
[_split_features_into_windows()] [DEBUG] shape of data: (294, 296)
[_split_features_into_windows()] [DEBUG] shape of data (padded): (324, 296)
[_split_features_into_windows()] [DEBUG] number of windows: 294
[_split_features_into_windows()] [DEBUG] retval windows_array shape=(294, 30, 296)
[      build_features()] [DEBUG] shape of features after _split_features_into_windows=(294, 30, 296)
```

```
[            __init__()] [DEBUG] Y_class=[1.0, 0.0]
[            __init__()] [DEBUG] sourcefile=training/set2/ff-5.wav
[            __init__()] [DEBUG] self.y.shape=(137986, 2)
[            __init__()] [DEBUG] sr=48000
[            __init__()] [DEBUG] Y_class=[1.0, 0.0]
[            __init__()] [DEBUG] sourcefile=training/set2/ff-7.wav
[            __init__()] [DEBUG] self.y.shape=(133730, 2)
[            __init__()] [DEBUG] sr=48000
[            __init__()] [DEBUG] Y_class=[1.0, 0.0]
[            __init__()] [DEBUG] sourcefile=training/set2/ff-3.wav
[            __init__()] [DEBUG] self.y.shape=(107592, 2)
[            __init__()] [DEBUG] sr=48000
[            __init__()] [DEBUG] Y_class=[1.0, 0.0]
[            __init__()] [DEBUG] sourcefile=training/set2/ff.wav
[            __init__()] [DEBUG] self.y.shape=(100906, 2)
[            __init__()] [DEBUG] sr=48000
[            __init__()] [DEBUG] Y_class=[1.0, 0.0]
[            __init__()] [DEBUG] sourcefile=training/set2/dd-2.wav
[            __init__()] [DEBUG] self.y.shape=(1188226, 2)
[            __init__()] [DEBUG] sr=48000
[            __init__()] [DEBUG] Y_class=[0.0, 1.0]
```

Break

Summary
Data Preprocessing
librosa (features)
Keras (model)
Results
Fails and Futures

High Level Libraries

Keras: The Python Deep Learning library

You have just found Keras.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.*

Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

Read the documentation at Keras.io.

Keras is compatible with: **Python 2.7-3.6**.

model = Sequential()

"The Sequential model is a linear stack of layers."

https://keras.io/getting-started/sequential-model-guide/

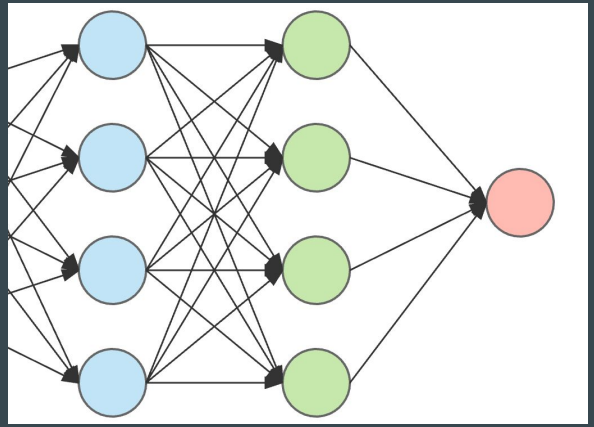
model.add(layer)

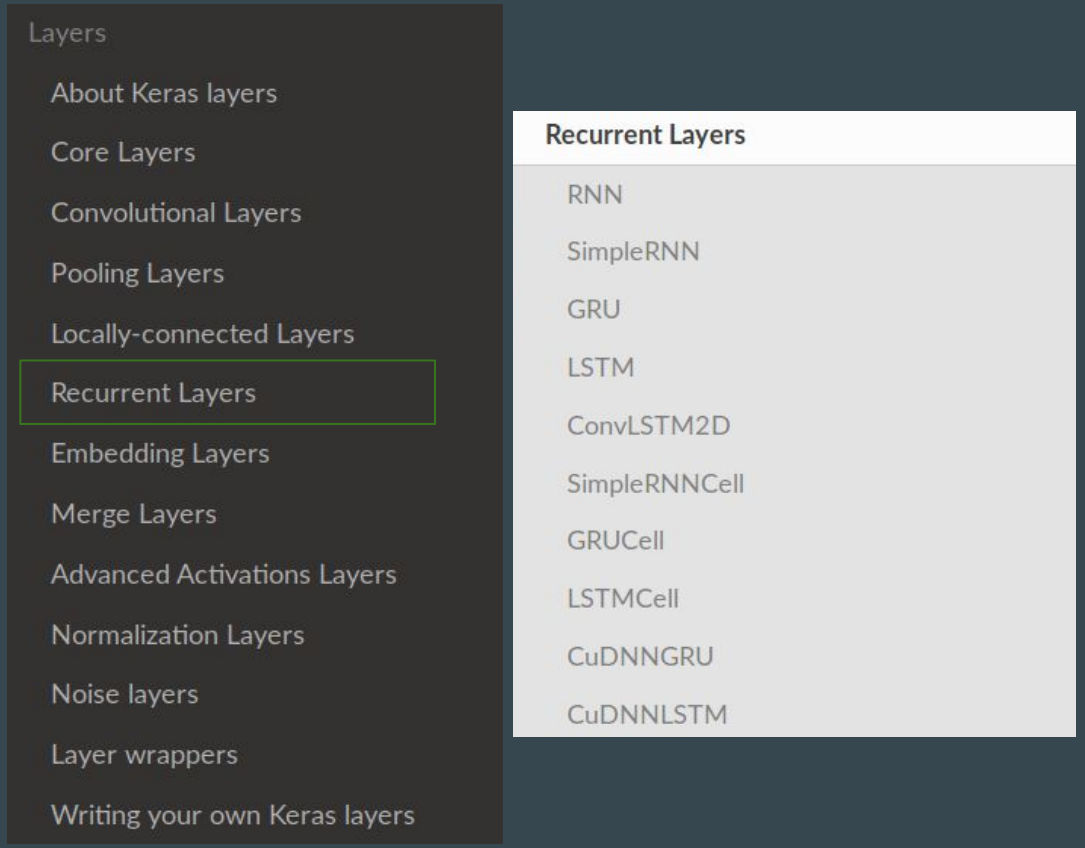
So simple.

keras.layers.Dense(units)

For example:

```
model.add(Dense(4))
model.add(Dense(4))
model.add(Dense(1))
```




Layers
About Keras layers
Core Layers
Convolutional Layers
Pooling Layers
Locally-connected Layers
Recurrent Layers
Embedding Layers
Merge Layers
Advanced Activations Layers
Normalization Layers
Noise layers
Layer wrappers
Writing your own Keras layers
Recurrent Layers
RNN
SimpleRNN
GRU
LSTM
ConvLSTM2D
SimpleRNNCell
GRUCell
LSTMCell
CuDNNGRU
CuDNNLSTM

LSTM [source]

```
keras.layers.LSTM(units, activation='tanh', recurrent_activation='hard_sigmoid', use_bias=True, kerne
```

Long Short-Term Memory layer - Hochreiter 1997.

Arguments

- **units**: Positive integer, dimensionality of the output space.
- **activation**: Activation function to use (see activations). Default: hyperbolic tangent (`tanh`). If you pass `None` , no activation is applied (ie. "linear" activation: `a(x) = x`).
- **recurrent_activation**: Activation function to use for the recurrent step (see activations). Default: hard sigmoid (`hard_sigmoid`). If you pass `None` , no activation is applied (ie. "linear" activation: `a(x) = x`).
- **use_bias**: Boolean, whether the layer uses a bias vector.
- **kernel_initializer**: Initializer for the `kernel` weights matrix, used for the linear transformation of the inputs. (see initializers).
- **recurrent_initializer**: Initializer for the `recurrent_kernel` weights matrix, used for the linear transformation of the recurrent state. (see initializers).
- **bias_initializer**: Initializer for the bias vector (see initializers).
- **unit_forget_bias**: Boolean. If True, add 1 to the bias of the forget gate at initialization. Setting it to true will also force `bias_initializer="zeros"` . This is recommended in Jozefowicz et al.
- **kernel_regularizer**: Regularizer function applied to the `kernel` weights matrix (see regularizer).
- **recurrent_regularizer**: Regularizer function applied to the `recurrent_kernel` weights matrix (see regularizer).
- **bias_regularizer**: Regularizer function applied to the bias vector (see regularizer).
- **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation"). (see regularizer).
- **kernel_constraint**: Constraint function applied to the `kernel` weights matrix (see constraints).
- **recurrent_constraint**: Constraint function applied to the `recurrent_kernel` weights matrix (see constraints).
- **bias_constraint**: Constraint function applied to the bias vector (see constraints).
- **dropout**: Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs.
- **recurrent_dropout**: Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.
- **implementation**: Implementation mode, either 1 or 2. Mode 1 will structure its operations as a larger number of smaller dot products and additions, whereas mode 2 will batch them into fewer, larger operations. These modes will have different performance profiles on different hardware and for different applications.
- **return_sequences**: Boolean. Whether to return the last output in the output sequence, or the full sequence.
- **return_state**: Boolean. Whether to return the last state in addition to the output.
- **go_backwards**: Boolean (default False). If True, process the input sequence backwards and return the reversed sequence.
- **stateful**: Boolean (default False). If True, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.
- **unroll**: Boolean (default False). If True, the network will be unrolled, else a symbolic loop will be used. Unrolling can speed-up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences.

LSTM Network

well-suited to **classifying, processing and making predictions based on time series data**, since there can be lags of unknown duration between important events in a time series

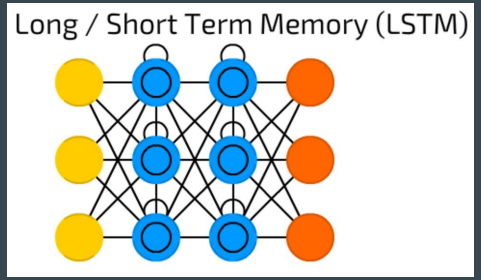
- https://en.wikipedia.org/wiki/Long_short-term_memory

LSTM Network

- Popular for speech recognition, natural language
- Google, Apple, Amazon - speech recognition, typing prediction, Echo

  - https://en.wikipedia.org/wiki/Long_short-term_memory

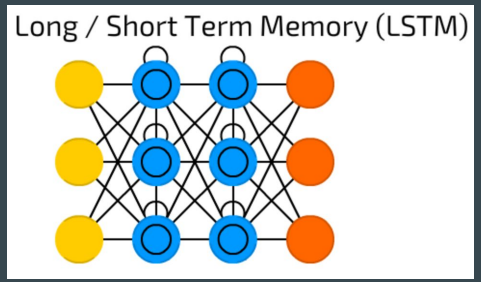
LSTM Network



"This type introduces a memory cell, a special cell that can process data when data have time gaps (or lags). RNNs can process texts by “keeping in mind” ten previous words, and LSTM networks can process video frame “keeping in mind” something that happened many frames ago."

https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

LSTM Network

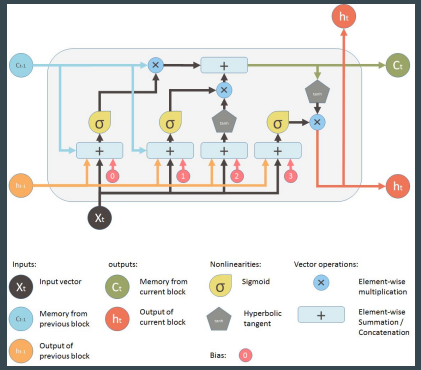


"[gates] have they own weights and sometimes activation functions. On each sample they decide whether to pass the data forward, erase memory and so on"

https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

For LSTM internals...

https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714



Input Dimensions - Time Steps

LSTM is dealing with sequences

Each sample needs "state_at_t" time

Our "30 steps" per example

```
import keras_metrics
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, SimpleRNN, GRU

laugh_precision = keras_metrics.precision(label=0)
laugh_recall    = keras_metrics.recall(label=0)
metrics = ['accuracy', laugh_precision, laugh_recall]

model = Sequential()

model.add(LSTM(8, recurrent_dropout=0.20, input_shape=input_shape,
                  return_sequences=True))
model.add(LSTM(8, recurrent_dropout=0.20,
                  return_sequences=True))
model.add(LSTM(4, recurrent_dropout=0.20))

model.add(Dense(2, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=metrics)
```

Don't use Dropout layer in RNN

Dropout [source]

```
keras.layers.Dropout(rate, noise_shape=None, seed=None)
```

Applies Dropout to the input.

Dropout consists in randomly setting a fraction `rate` of input units to 0 at each update during training time, which helps prevent overfitting.

return_sequences=true

For multiple LSTM layers

Needs the whole 30-step seq

RMSprop

[source]

```
keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)
```

RMSProp optimizer.

It is recommended to leave the parameters of this optimizer at their default values (except the learning rate, which can be freely tuned).

This optimizer is usually a good choice for recurrent neural networks.

loss = categorical_crossentropy

Note: when using the `categorical_crossentropy` loss, your targets should be in categorical format (e.g. if you have 10 classes, the target for each sample should be a 10-dimensional vector that is all-zeros except for a 1 at the index corresponding to the class of the sample). In order to convert *integer targets* into *categorical targets*, you can use the Keras utility `to_categorical`:

softmax activation

- The "I Google'd it" method
- Goes with categorical cross-entropy loss function
- Worked well enough for my data

Summary
Data Preprocessing
librosa (features)
Keras (model)
Results
Fails and Futures

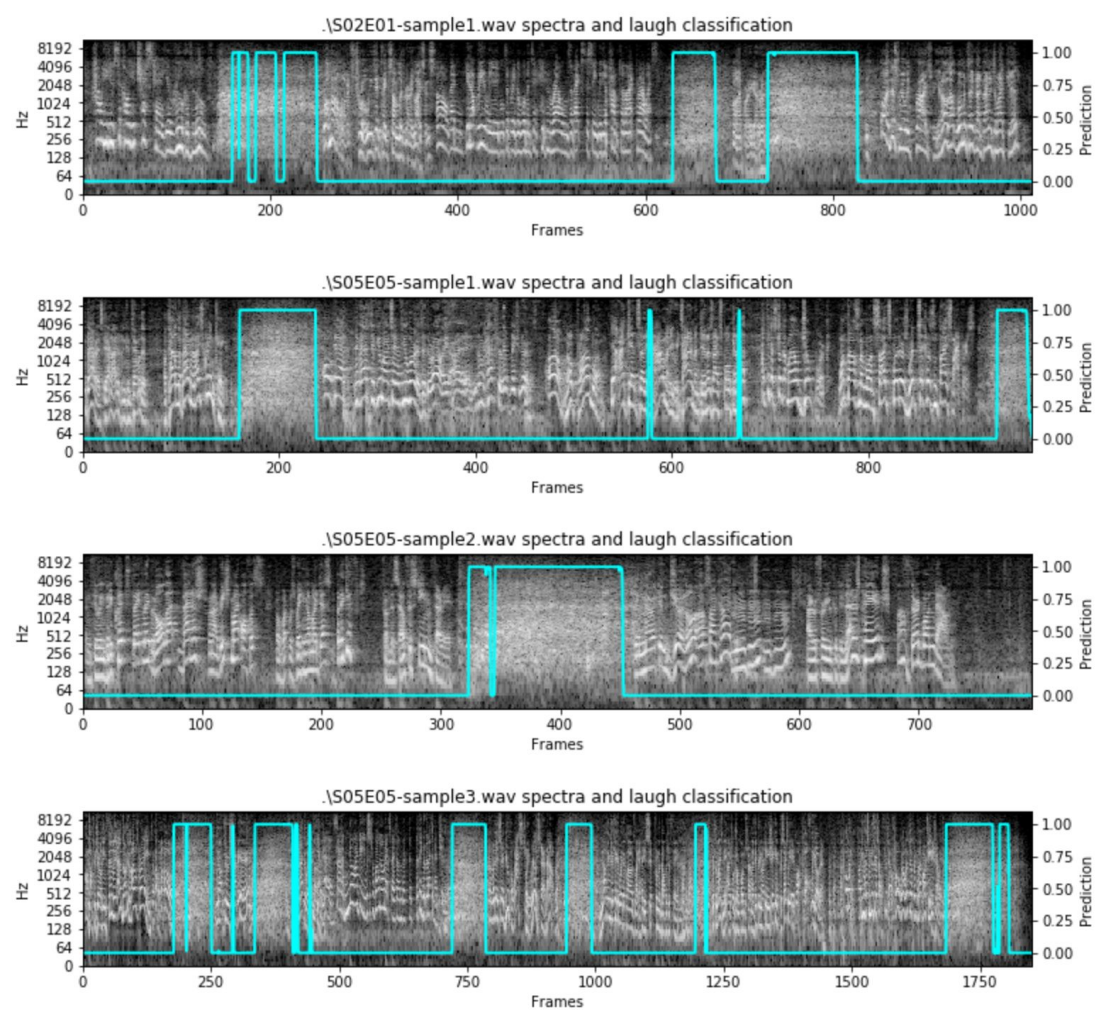
Processed Results

https://youtu.be/DeTQBiKzmYc

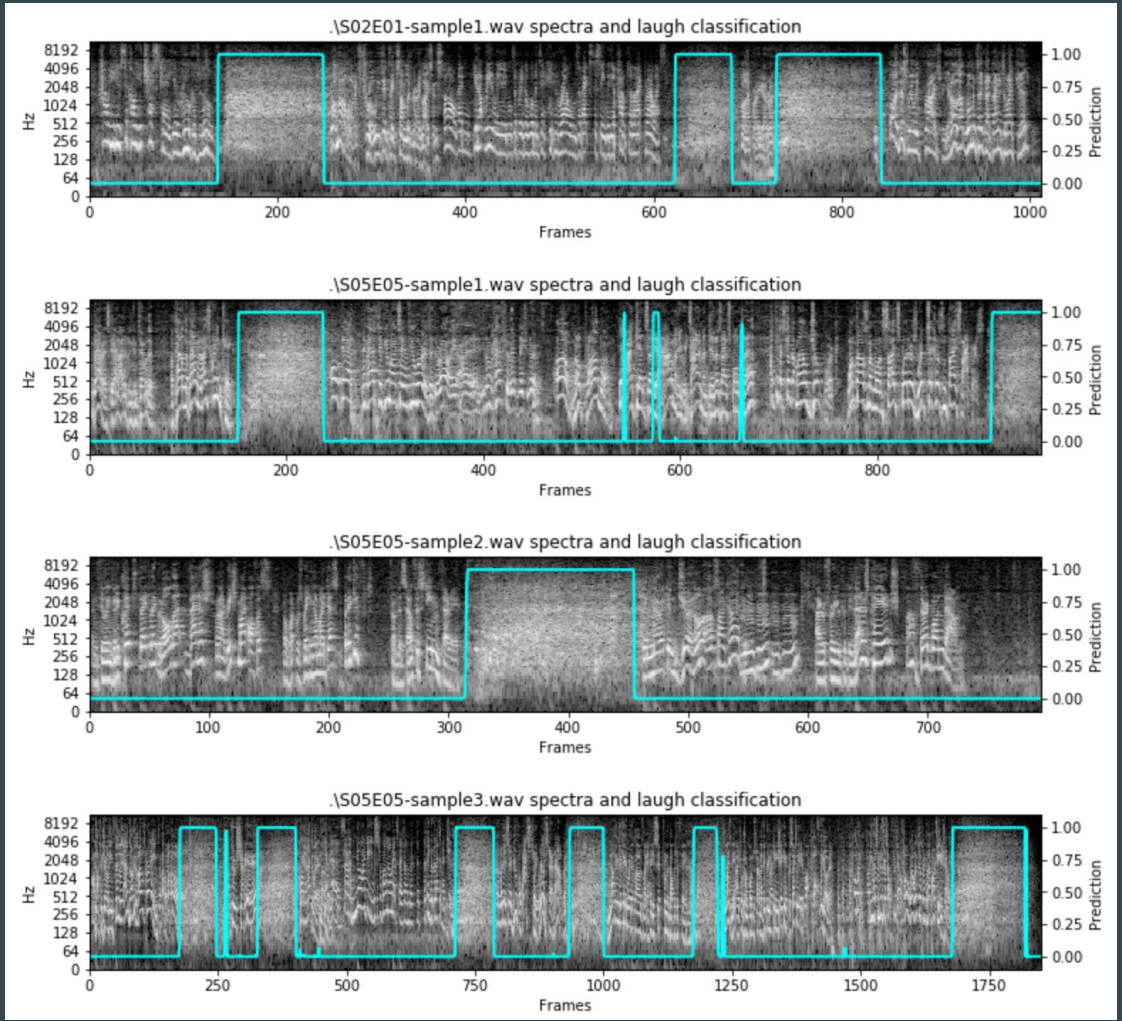
Visualizing Classifications

More training data is better

Less


.\S02E01-sample1.wav spectra and laugh classification
.\S05E05-sample1.wav spectra and laugh classification
.\S05E05-sample2.wav spectra and laugh classification
.\S05E05-sample3.wav spectra and laugh classification
Hz
Frames
Prediction

More


.\S02E01-sample1.wav spectra and laugh classification
.\S05E05-sample1.wav spectra and laugh classification
.\S05E05-sample2.wav spectra and laugh classification
.\S05E05-sample3.wav spectra and laugh classification
Hz
Frames
Prediction

Visualizing Features

over time

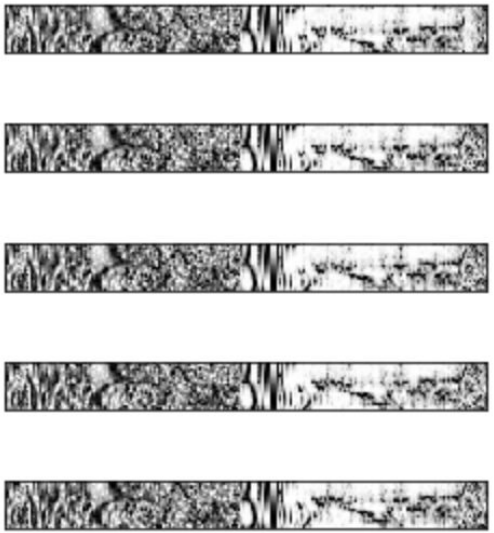
30 step x 296 feature, normalized.

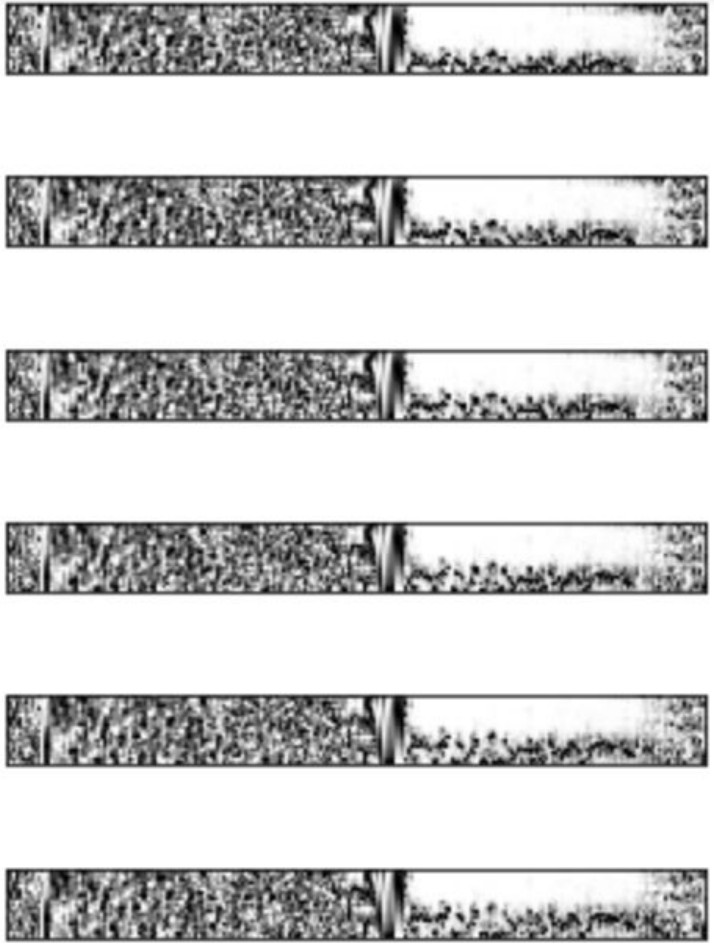
as grayscale image

In [6]:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
for i in range(features.shape[0]):
    scaler.fit(features[i])
    f2 = scaler.transform(features[i])
    plt.figure(figsize=(4,1))
    plt.yticks([])
    plt.xticks([])
    plt.imshow(f2, cmap='binary')
```





Benchmarking Models

```
for name in models:
    m, s = do_train(ds[0], models[name])
    r    = do_eval(ds[1], m)
```

benchmark.py

generate_models()

...iterate over params

Visualize results

```python
def do_train(ds, model):
    s = time.time()
    model.fit(ds.X, ds.Y_class, epochs=15, batch_size=1000, verbose=0)
    duration = time.time() - s
    return model, round(duration, 2)
```

```python
def do_eval(ds, model):
    results = model.evaluate(ds.X, ds.Y_class, verbose=0)
    return dict(zip(model.metrics_names, [round(r, 6) for r in results]))
```

generate_models():

Type - LSTM, GRU

Topologies, Dropout

Window size

F_1 Score

The general formula for positive real β is:

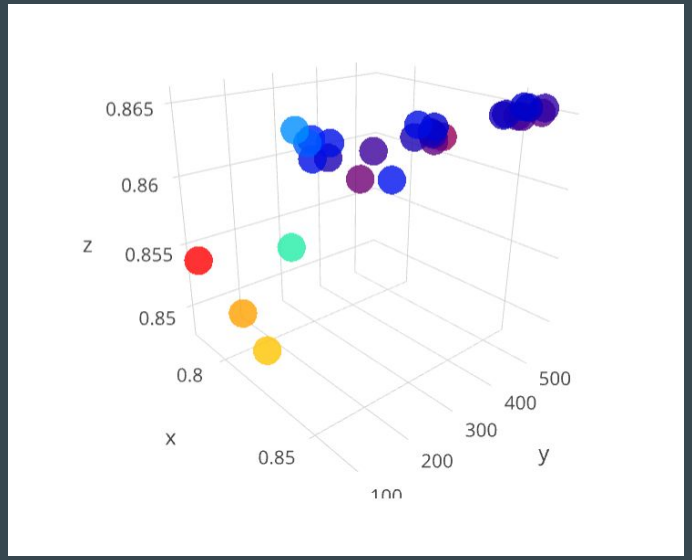
$$F_\beta = (1 + \beta^2) \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{(\beta^2 \cdot \mathrm{precision}) + \mathrm{recall}}.$$

https://en.wikipedia.org/wiki/F1_score

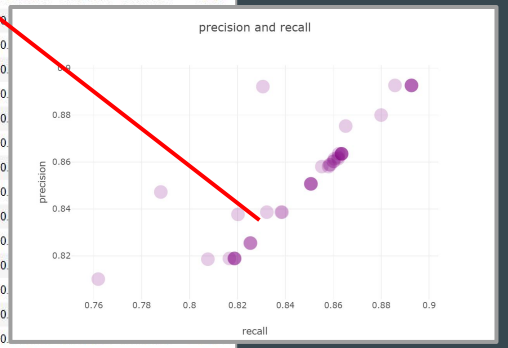
$F_{0.5}$ Score (*Precision*)

F0.5 score				
	0.1	0.8	1	recall
0.1	0.1	0.12	0.12	
0.8	0.33	0.8	0.83	
1	0.36	0.95	1.0	
precision				

Visualizing Benchmarks

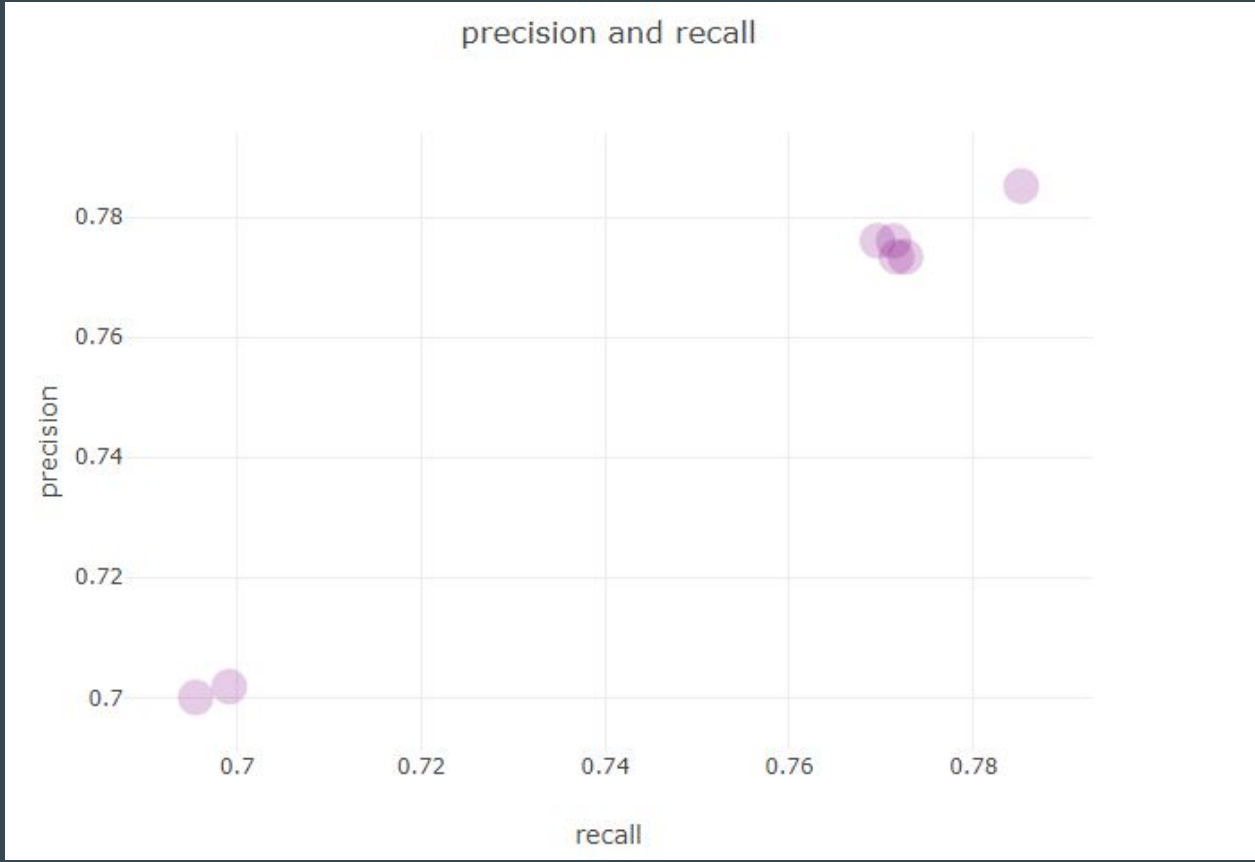


	f0.5	windowSize	type	topology	r_drop	drop	acc	loss	precision	recall	traintime
0	0.892629	w_80	LSTM	8x8x4	0.2	0	0.999723	0.004681	0.892629	0.892629	519.75
1	0.892629	w_80	LSTM	24x16x8	0.2	0	0.999251	0.003554	0.892629	0.892629	731.9
2	0.892629	w_80	GRU	8x8x4	0.2	0	0.998003	0.011586	0.892629	0.892629	419.4
3	0.892629	w_80	GRU	24x16x8	0.2	0	0.997421	0.020911	0.892629	0.892629	511.96
4	0.891245	w_80	SimpleRNN	24x16x8	0.2	0	0.993067	0.028671	0.892629	0.885752	321.7
5	0.879986	w_80	GRU	24x16x8	0	0.05	0.999941	0.000440	0.879992	0.879962	401.79
6	0.879107	w_80	SimpleRNN	8x8x4	0.2	0	0.838279	0.236128	0.892127	0.930618	276.29
7	0.873258	w_80	GRU	8x8x4	0	0.05	0.982040	0.073901	0.875384	0.865167	331.3
8	0.863478	w_40	LSTM	8x8x4	0.2	0	0.996071	0.019764	0.863478	0.863478	214.6
9	0.863478	w_40	GRU	8x8x4	0.2	0	0.994864	0.031579	0.863478	0.863478	187.14
10	0.863472	w_40	LSTM	24x16x8	0.2	0	0.995369	0.021086	0.863478	0.863450	263.4
11	0.863472	w_40	SimpleRNN	24x16x8	0.2	0	0.992971	0.037390	0.863478	0.863450	152.03
12	0.863467	w_40	GRU	24x16x8	0.2	0	0.995004	0.023813	0.863478	0.863422	218.38
13	0.863231	w_40	SimpleRNN	8x8x4	0.2	0	0.976900	0.106944	0.863478	0.862243	140.0
14	0.862053	w_20	LSTM	24x16x8	0	0.05	0.997787	0.011622	0.862072	0.861979	142.43
15	0.861551	w_20	LSTM	8x8x4	0	0.05	0.996602	0.017400	0.861426	0.862049	119.77
16	0.861150	w_20	SimpleRNN	24x16x8	0	0.05	0.996132	0.0			
17	0.860268	w_30	GRU	24x16x8	0	0.05	0.998344	0.			
18	0.860213	w_30	GRU	8x8x4	0	0.05	0.997365	0.			
19	0.858802	w_20	GRU	24x16x8	0	0.05	0.996856	0.			
20	0.858723	w_20	GRU	8x8x4	0	0.05	0.993131	0.			
21	0.858039	w_40	GRU	8x8x4	0	0.05	0.999790	0.			
22	0.857470	w_40	GRU	24x16x8	0	0.05	0.995642	0.			
23	0.850679	w_80	SimpleRNN	24x16x8	0	0.05	0.997372	0.			
24	0.850679	w_80	LSTM	24x16x8	0	0.05	0.996722	0.			
25	0.850679	w_80	SimpleRNN	8x8x4	0	0.05	0.991553	0.			
26	0.850655	w_80	LSTM	8x8x4	0	0.05	0.990372	0.			
27	0.842082	w_30	LSTM	8x8x4	0.2	0	0.998590	0.			
28	0.842082	w_30	GRU	24x16x8	0.2	0	0.996638	0.			
29	0.842077	w_30	LSTM	24x16x8	0.2	0	0.996719	0.			
30	0.842044	w_30	GRU	8x8x4	0.2	0	0.994577	0.025881	0.842082	0.841893	181.0
31	0.841555	w_30	SimpleRNN	24x16x8	0.2	0	0.995201	0.017946	0.842082	0.839452	170.74
32	0.840506	w_30	SimpleRNN	8x8x4	0.2	0	0.974783	0.101084	0.842082	0.834263	117.58
33	0.838619	w_40	LSTM	24x16x8	0	0.05	0.996292	0.022837	0.838619	0.838619	225.62



Benchmark data (pandas df)

	f0.5	windowSize	type	topology	r_drop	drop	acc	loss	precision	recall	traintime
0	0.775102	w_30	LSTM	24x16x8	0.2	0	0.991542	0.040502	0.776051	0.771328	181.55
1	0.774746	w_30	GRU	24x16x8	0.2	0	0.991585	0.029332	0.776051	0.769568	159.51
2	0.773235	w_80	GRU	24x16x8	0.2	0	0.995340	0.037712	0.773393	0.772606	542.32
3	0.773041	w_80	LSTM	24x16x8	0.2	0	0.993947	0.034538	0.773393	0.771638	845.67


precision and recall
0.78
0.76
precision
0.74
0.72
0.7
0.7
0.72
0.74
0.76
0.78
recall

Chunking Large "Clips"

RawClip helps limit memory

for very big clips

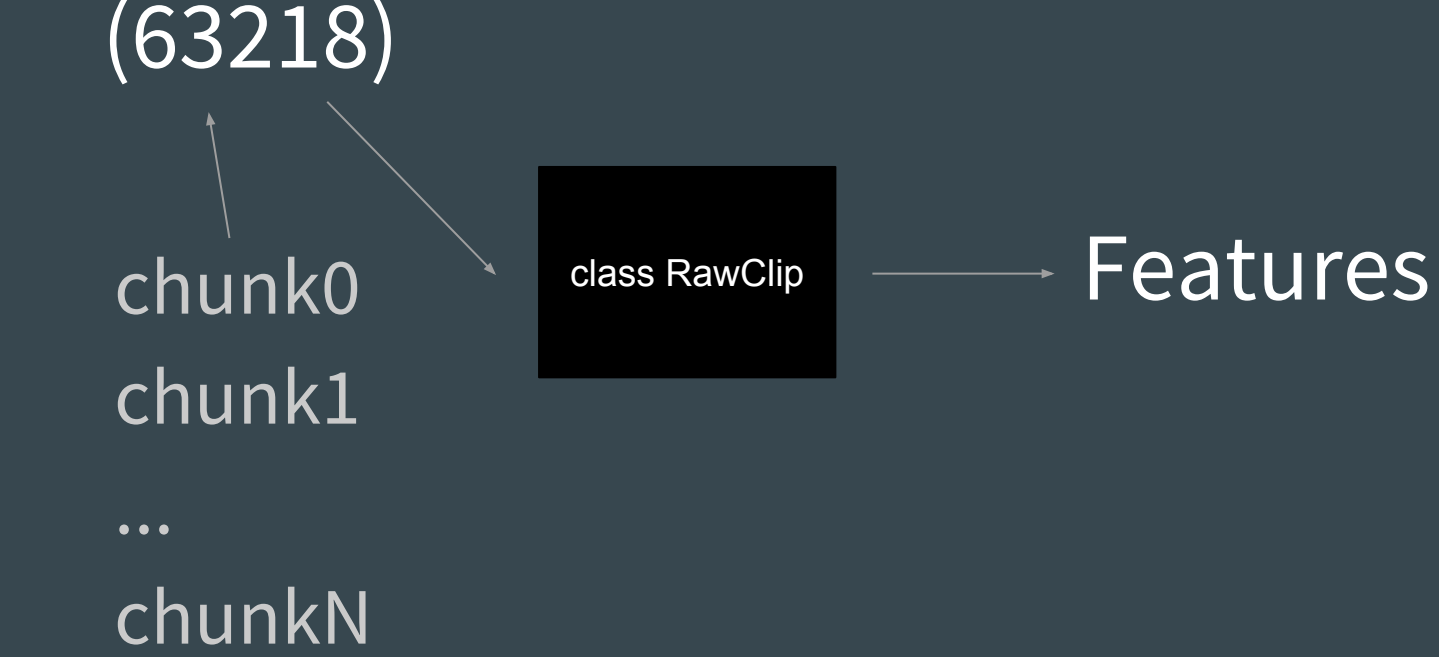
(~20 minute episode)

Method for Chunking

librosa funcs can hog memory

Clumsy (generator / yield ?)

But, it works.


samples / length
(63218)
chunk0
chunk1
...
chunkN
class RawClip
Features

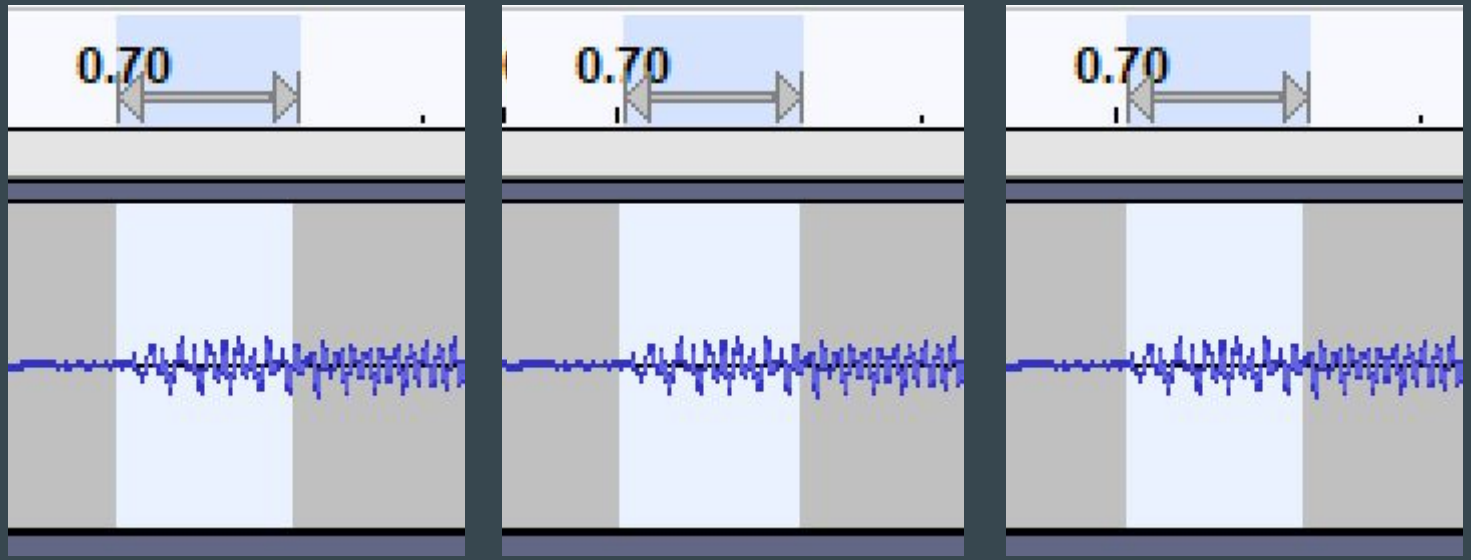
Summary
Data Preprocessing
librosa (features)
Keras (model)
Results
Fails and Futures

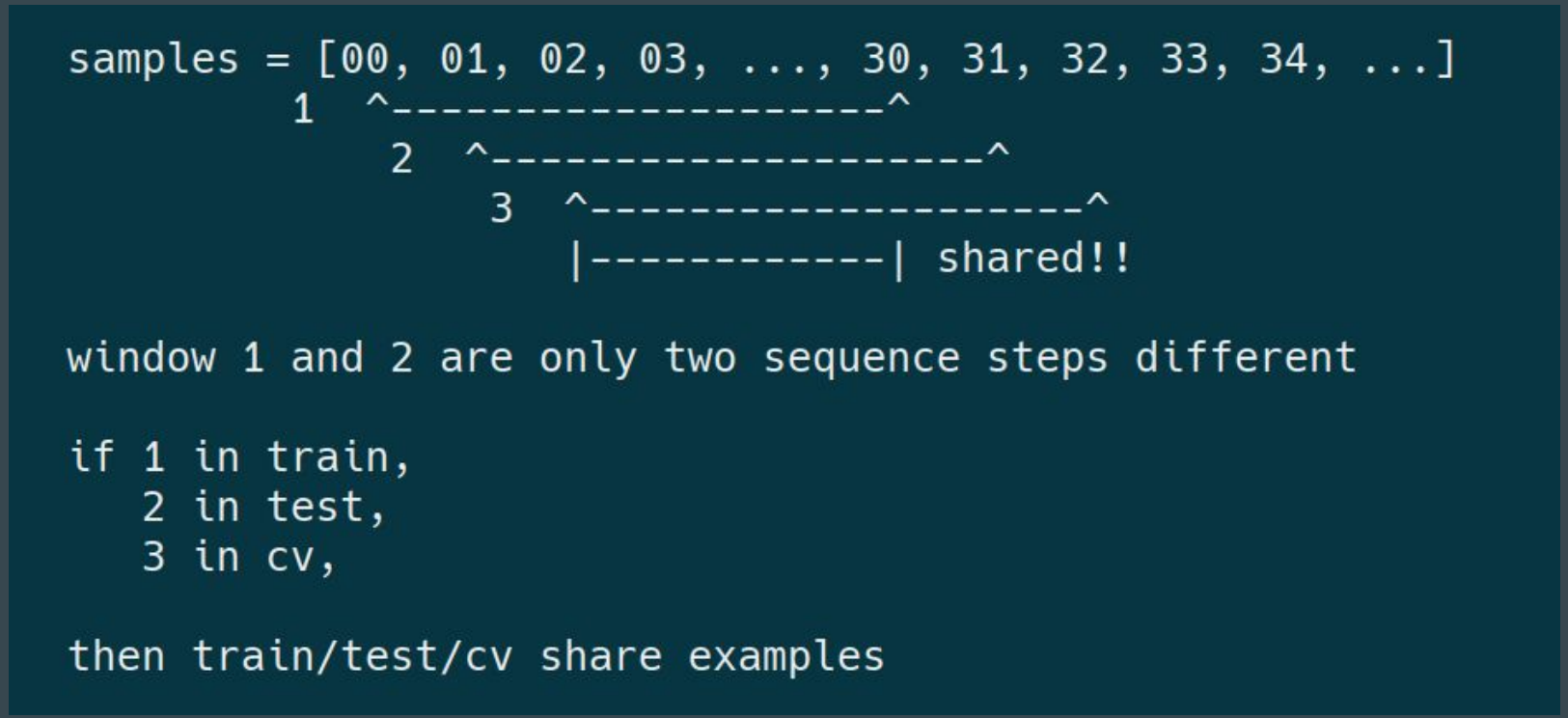
Windowing Problem

Test and CV - not good

Because of the windowing

Major Problem!




samples = [00, 01, 02, 03, ..., 30, 31, 32, 33, 34, ...]
1
2
3
shared!!
window 1 and 2 are only two sequence steps different
if 1 in train,
2 in test,
3 in cv,
then train/test/cv share examples

CV/Test Solution:

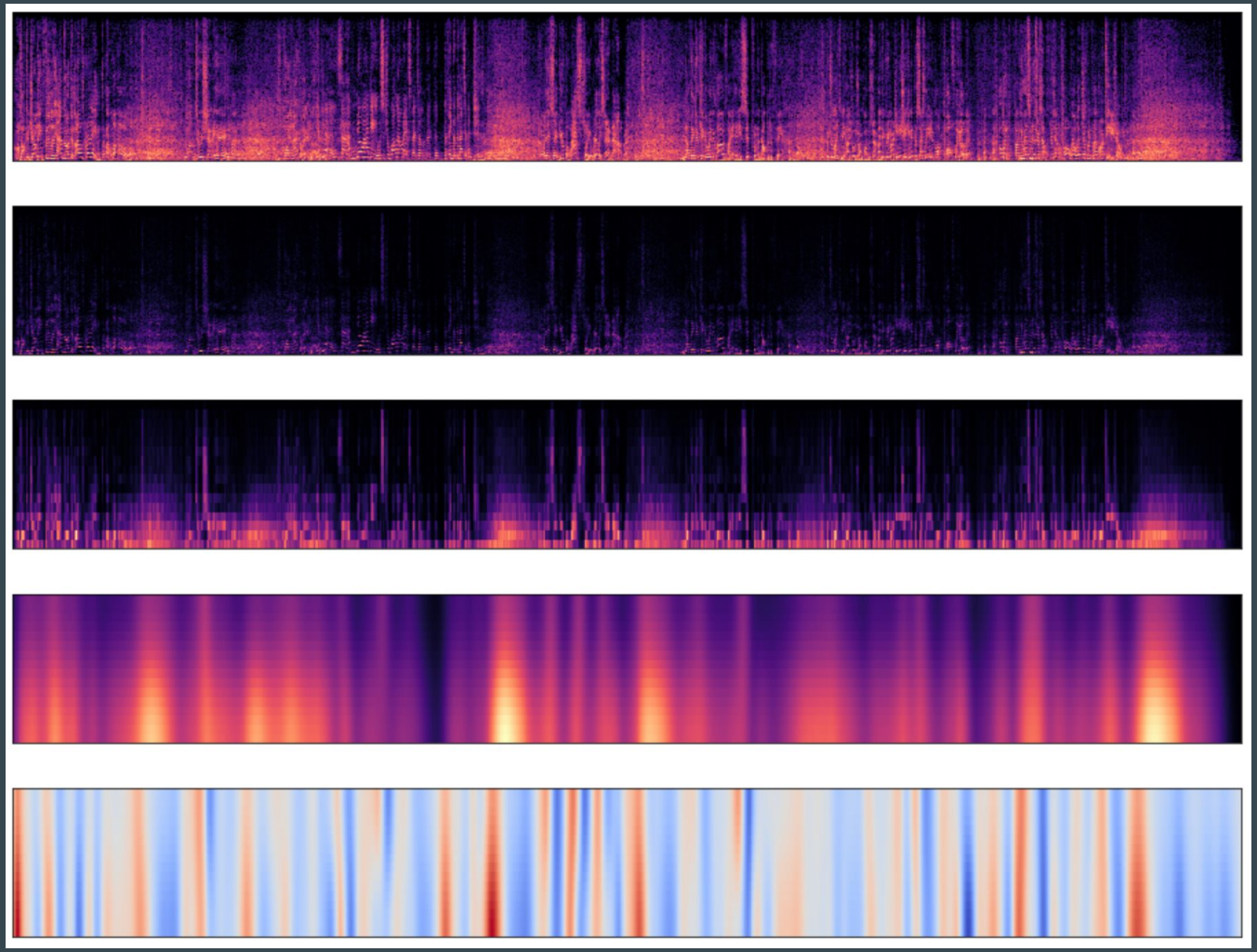
Created different labelled

example .wav clips.

2 DataSet() instances

"Image" Analysis Attempt

Gaussian blurring based on spectrogram plot



Visual Method Results

Did not generalize well.

Tune parameters to each clip.

Not good at quiet laughter.

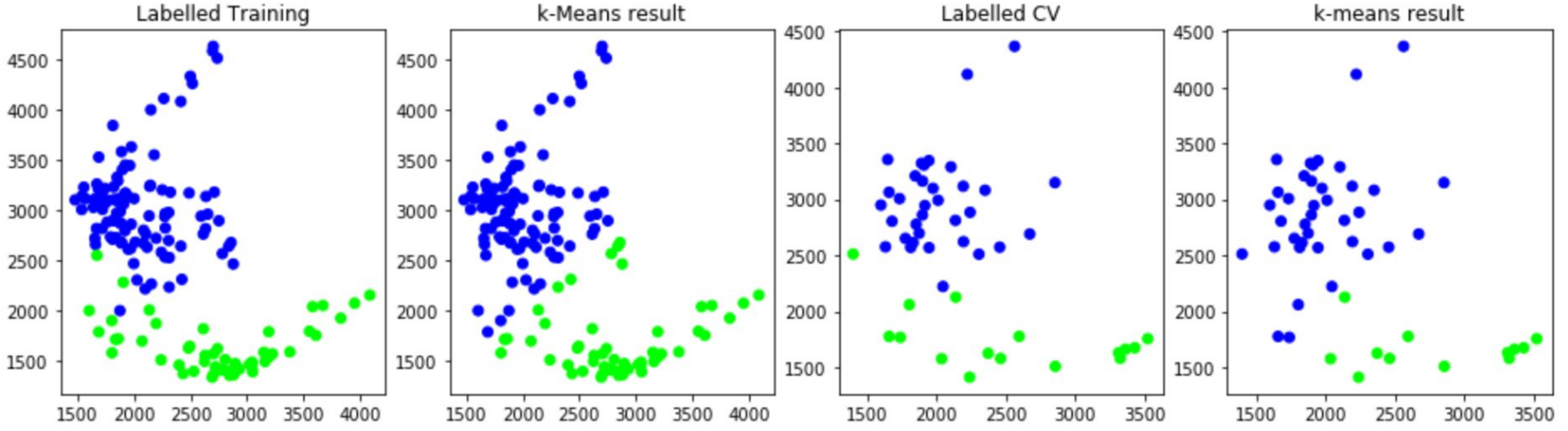
k-Means clustering

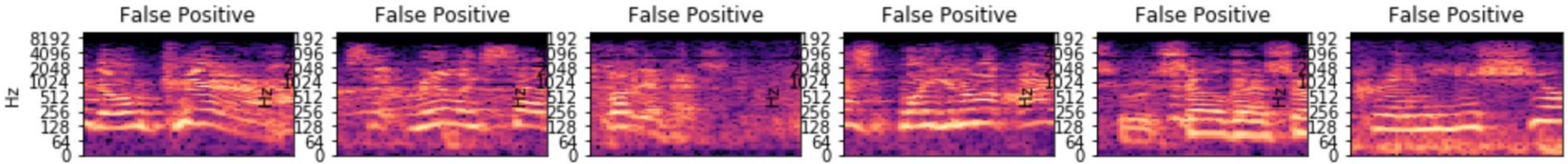
Based on spectrogram data

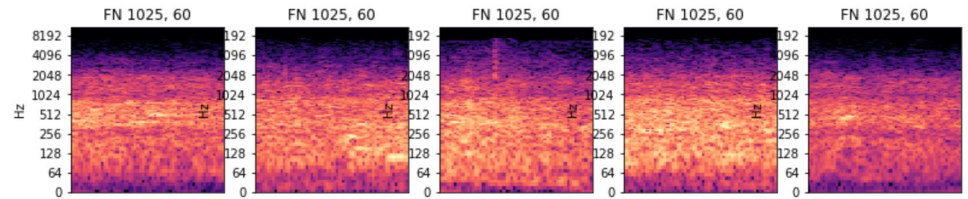
Maybe better results with

better data?

<matplotlib.text.Text at 0x1fa93dadef0>







Simple NN attempt

```
model.add(Dense(...))
model.add(Dense(...))
model.add(Dense(...))
```

Lesson Learned - Features

Too much time analyzing features

Lots of plotting...

...Not very helpful ultimately

python -m cProfile -o outfile laughr.py

```
set2.cprofile.dump% stats 25
Sat Aug 18 06:51:49 2018    set2.cprofile.dump

         2438705 function calls (2366295 primitive calls) in 15.836 seconds

   Ordered by: cumulative time
   List reduced from 6773 to 25 due to restriction <25>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   1155/1    0.041    0.000   15.841   15.841 {built-in method builtins.exec}
        1    0.000    0.000   15.841   15.841 laughr.py:2(<module>)
        1    0.000    0.000   14.939   14.939 laughr.py:160(load_dataset)
        1    0.070    0.070   14.939   14.939 laughr.py:94(__init__)
        1    0.005    0.005   14.826   14.826 laughr.py:116(_get_samples)
       13    0.010    0.001   14.503    1.116 laughr.py:60(build_features)
      117    0.041    0.000   14.150    0.121 laughr.py:33(_extract_feature)
       39    0.037    0.001   10.235    0.262 spectral.py:976(chroma_cqt)
       39    0.070    0.002   10.181    0.261 constantq.py:23(cqt)
      260    0.032    0.000    4.762    0.018 audio.py:209(resample)
      260    0.010    0.000    4.579    0.018 core.py:14(resample)
       13    0.001    0.000    3.944    0.303 spectral.py:1218(tonnetz)
       39    0.169    0.004    3.659    0.094 pitch.py:16(estimate_tuning)
      130    1.066    0.008    3.588    0.028 spectrum.py:1491(_spectrogram)
      390    0.783    0.002    3.543    0.009 spectrum.py:30(stft)
      260    3.516    0.014    3.516    0.014 interpn.py:7(resample_f)
       39    0.776    0.020    3.425    0.088 pitch.py:165(piptrack)
       13    0.002    0.000    3.151    0.242 spectral.py:1094(chroma_cens)
     4434    1.754    0.000    1.791    0.000 basic.py:185(fft)
     1507    0.131    0.000    1.777    0.001 arraypad.py:1094(pad)
```

https://www.stefaanlippens.net/python_profiling_with_pstats_interactive_mode/

Thanks!

github.com/jeffgreenca