NUS-ISS
Real Time Audio-Visual Sensing and Sense Making

Module 7 - Workshop on real time audio recognition, part 1

Dr. Tan Jen Hong
Lecturer & Consultant
Institute of System Science
National University of Singapore
issjht@nus.edu.sg

Machine Learning for Audio

Some applications ...

Machine learning + audio

- **Audio classification**
- Audio fingerprinting
- Audio segmentation
- Audio source separation
- Speech recognition
- Automatic music tagging
- Automatic music transcription
- Beat tracking
- Music recommendation
- Music retrieval
- Onset detection

Source: https://icon-library.net/icon/audio-icon-transparent-28.html

Audio classification

The most explored topic

- Audio classification: A fundamental problem in audio processing, and the most explored topic, plenty papers published

- Useful applications based on audio classification:
  - Genre classification
  - Instrument recognition
  - Artist identification
  - Voice recognition

- Common approach to solve the task:
  1. Preprocess audio inputs
  2. Extract useful features
  3. Perform classification

Source: https://www.steinway.com/artists/vladimir-horowitz

Audio fingerprinting

Think Shazam

- Objective of audio fingerprinting: produce a digital 'summary' of a piece of audio

- Shazam (founded in 1999) is the leading example on the use of fingerprinting:

- Match a recoded audio with a database of over 8 million songs / audio files

This is the one

Not him!

Source: https://itunes.apple.com/us/movie/shazam/id1456295500

Audio fingerprinting

Think Shazam

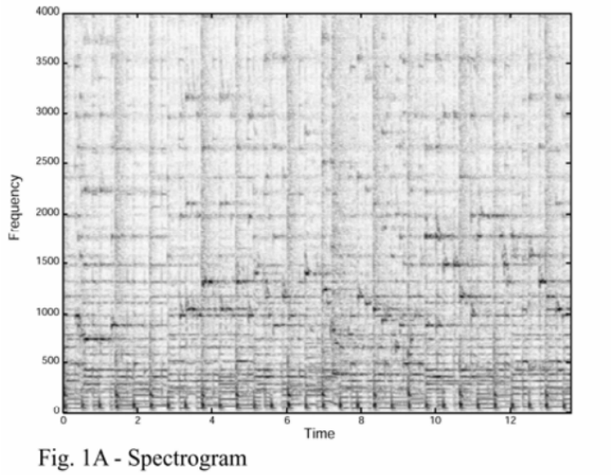


Fig. 1A - Spectrogram

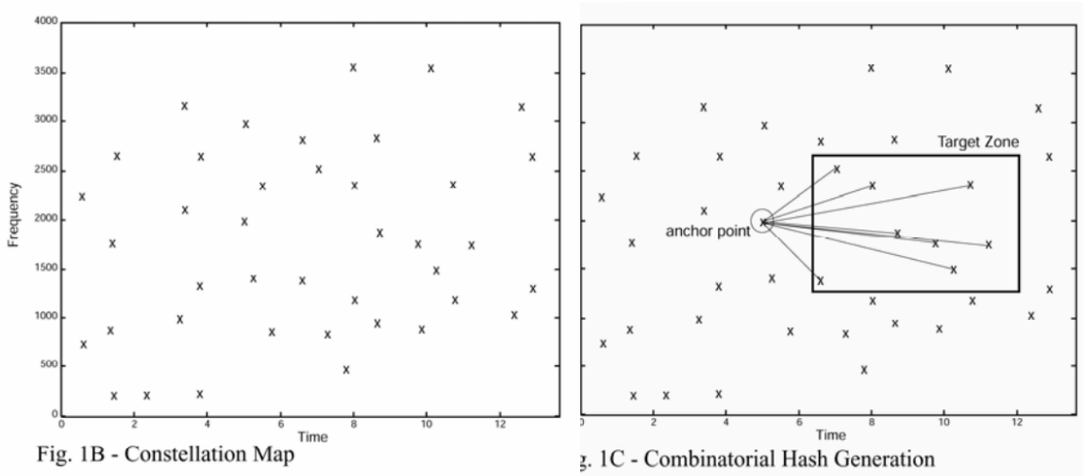


Fig. 1B - Constellation Map

g. 1C - Combinatorial Hash Generation

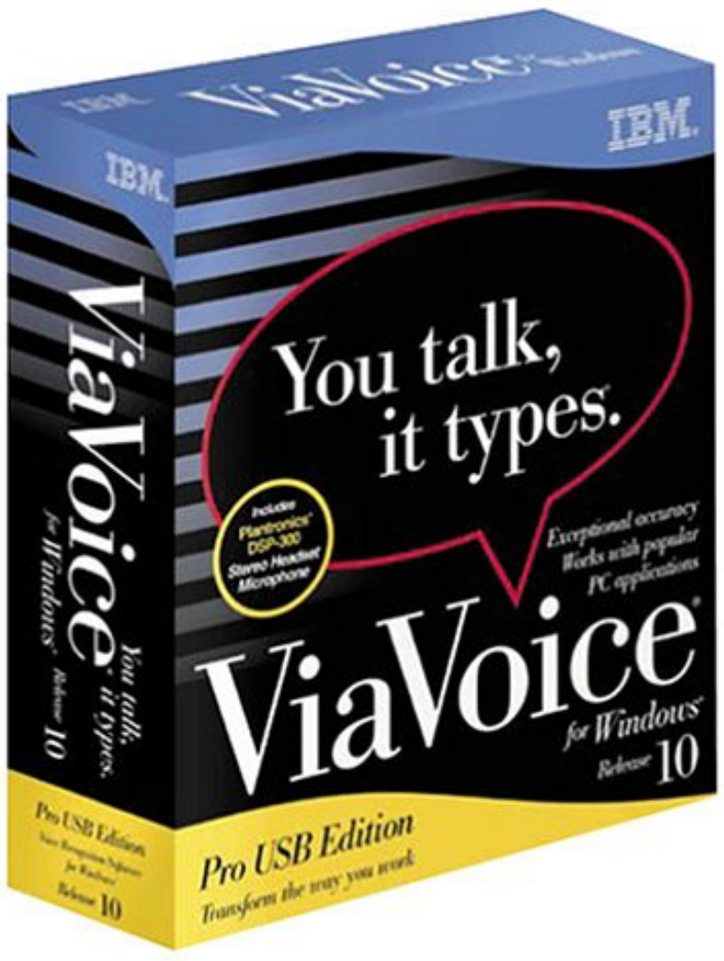
Source: https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf

- 3 steps are involved to generate audio fingerprint
- Step 1. Convert an audio file into spectogram. Y-axis represents frequency, X-axis represents time, intensity of shading represents amplitude
- Step 2. For each section of an audio file, choose the strongest peaks and spectogram is reduced to scatter plot (constellation map)
- Step 3. Generate combinatorial hashes based on every pair of anchor-point to point in the map
- Go to the below link if you want to know more about this

Source: https://medium.com/@treycoopermusic/how-shazam-works-d97135fb4582

Audio Segmentation

Separation by time



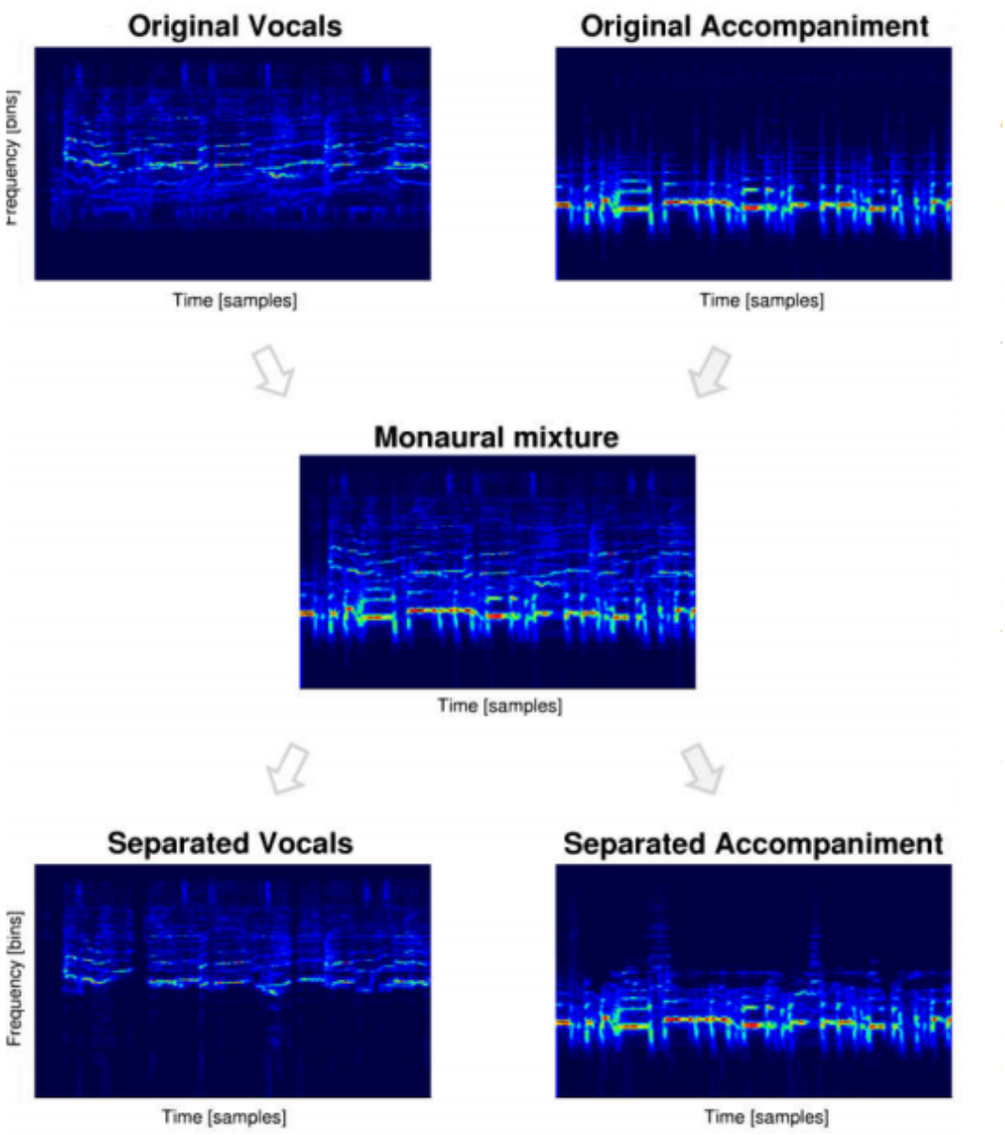
Source: https://www.amazon.ca/IBM-ViaVoice-Pro-USB-Edition/dp/B0000A58IX

- Audio segmentation: Segment audio stream into homogeneous regions
- Purpose: separate regions of different nature, e.g. music/noise, speech/non-speech, male/female
- Often a preprocessing step for further classification of the segments:
  - Speaker identification/tracking
  - Automatic transcription
  - Segmentation in news broadcast
  - Automatic speech recognition
- Two steps invovled: feature extraction, statistical analysis (to find segment boundaries

Source: http://www.music.mcgill.ca/~ich/classes/mumt611_07/presentations/shiyong/shiyong07audio.pdf

Audio Source Separation

A need to focus

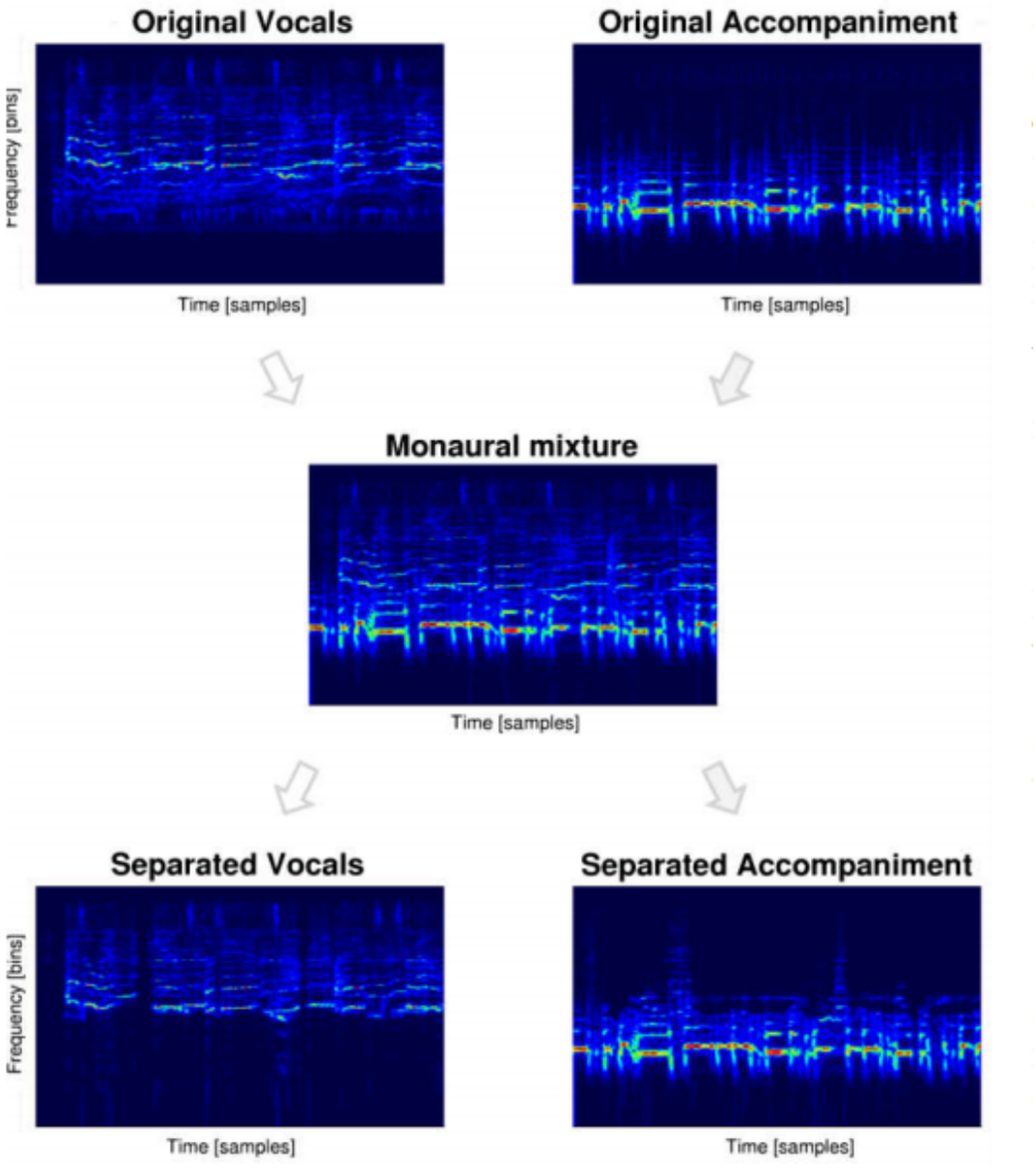


Source: https://www.technologyreview.com/s/537101/deep-learning-machine-solves-the-cocktail-party-problem/

- Cocktail party effect: The ability to focus on a specific human voice while filtering out other voices or background noise (source separation)
- Human can do this easily, but very hard to achieve digitally
- Application: Singer identification for the effective management and exploration of large amount of music data
- Application: Separate vocal from music accompaniment in songs (see next slide)

Audio Source Separation

A need to focus

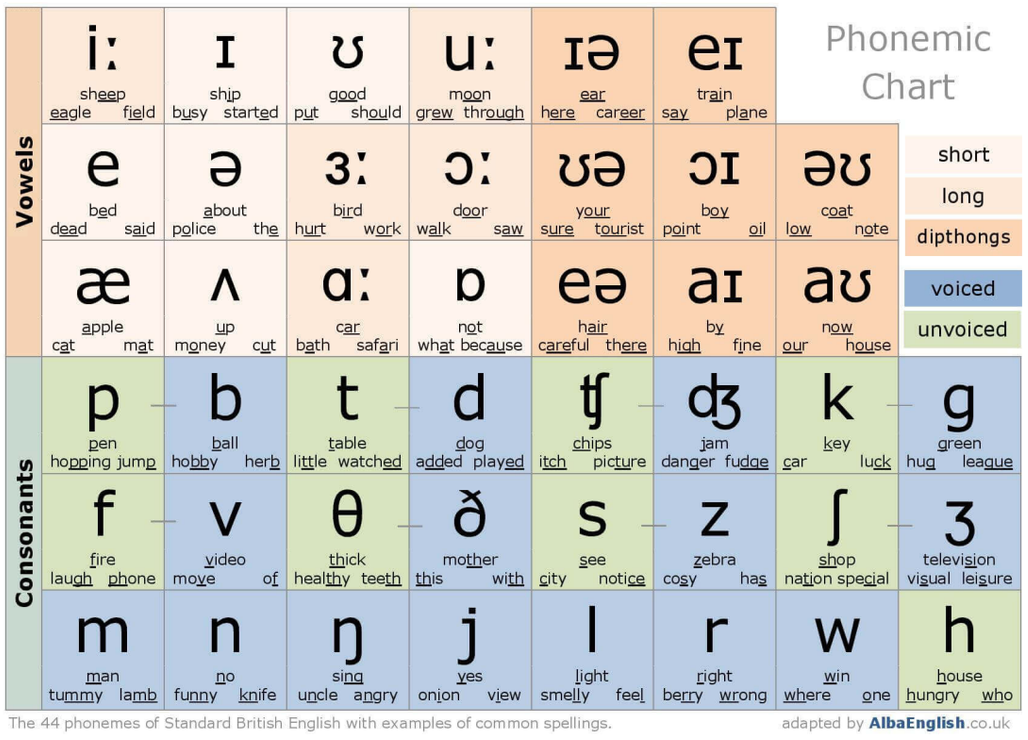


Source: https://arxiv.org/pdf/1504.04658.pdf

- Convert a song (consists of vocals and accompaniment) into a spectogram. Feed a 20-sample spectogram (along time) into a deep neural network.
- The output of the net is a mask (the same size as the input) that will be used to retain values in the input that belonged to vocal
- Get the next 20-sample spectogram (stride of 1, not 20) and repeat the steps until the end
- The masked spectogram, together with the phase components of the original spectogram are used to get the separated audio

Speech recognition

Think Siri / Google voice

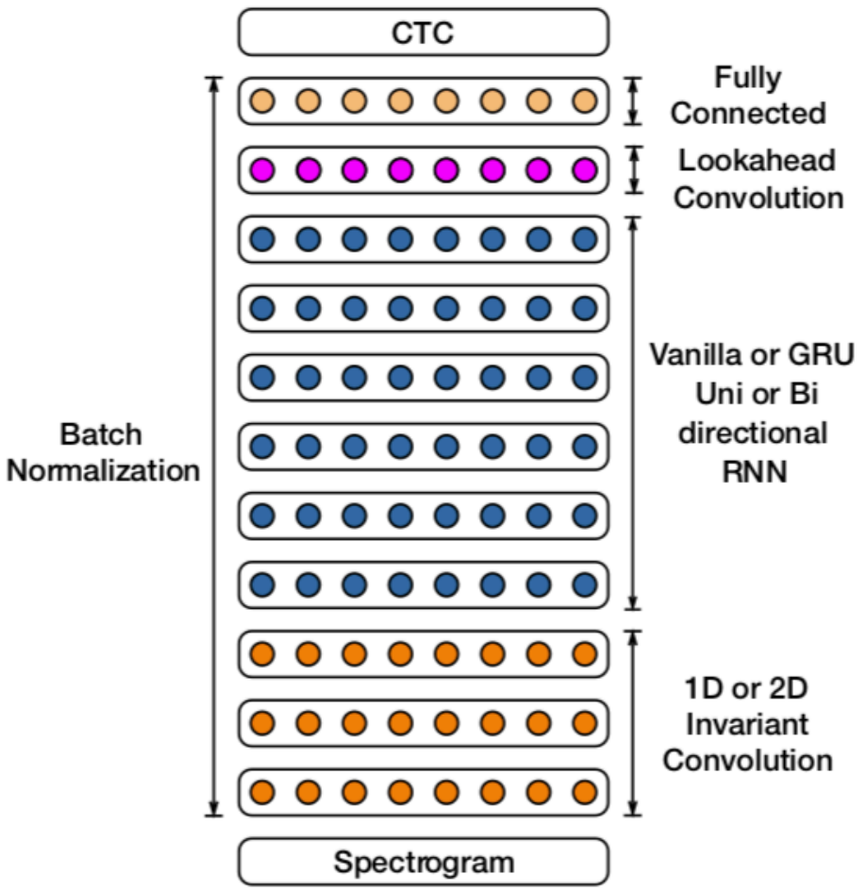


Source: https://albaenglish.co.uk/blog/improve-english-pronunciation-phonemic-chart

- In early days, speech recognition systems consists of mainly 3 components
- Acoustic model: Maps segments of audio (usually 10 milliseconds) to phonemes
- Pronunciation model: Connect phonemes together to form words
- Language model: Express the likelihood of given phrases
- Since 2014, researchers started to focus on training a single neural network that could map an input audio waveform to an output sentence

Speech recognition

End-to-end model



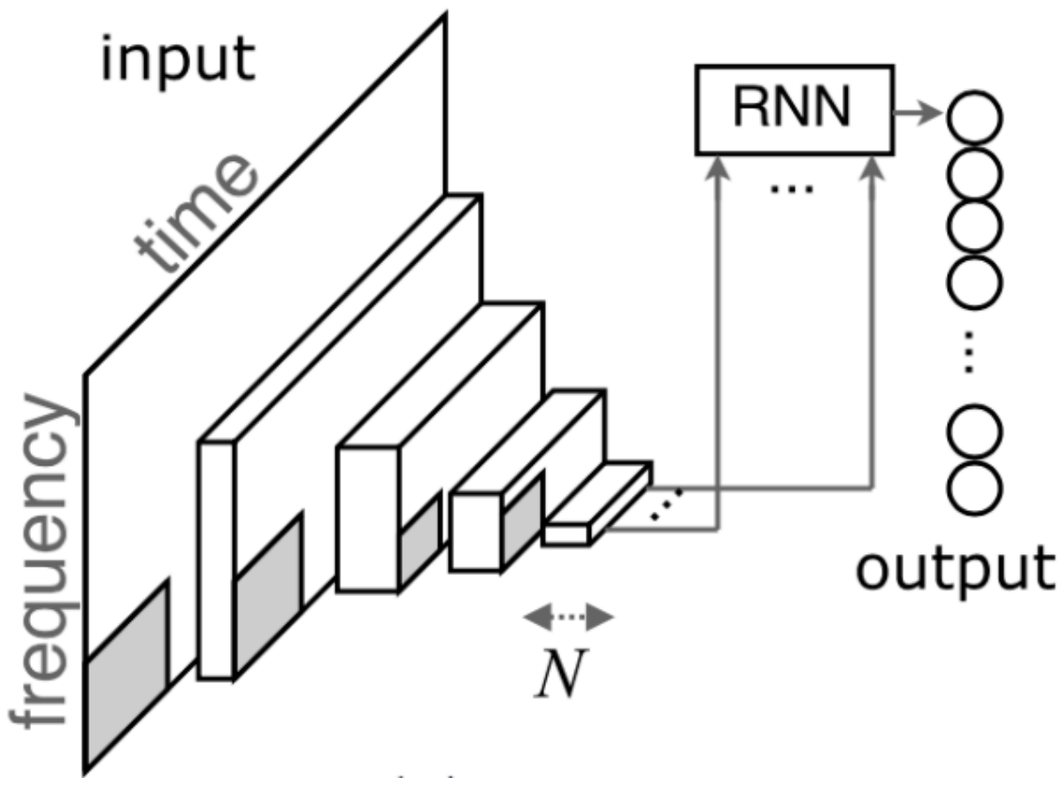
Source: https://pdfs.semanticscholar.org/8e0e/acf11a22b9705a262e908f17b1704fd21fa7.pdf

- Example: Baidu has come out an End-to-end model that works for English and Chinese
- Inputs: a sequence of log-spectograms of power normalized audio segments, calculated on 20ms windows
- Outputs: At each output time-step *t*, in English, the output is probability prediction on alphabets plus space, apostrophe and blank
- For Chinese, the output is probability prediction on 6000 characters + Roman alphabet
- Note: Connectionist Temporal Classification (CTC) is a type of network output and associated scoring function

Automatic music tagging

Spotify?

- Tags: Text-based labels that encode semantic information related to sound (Panagakis & Kotropoulos, 2011)
- Music tagging: Add tags to song
- Example of tags: rock, pop, jazz, 60s, sad and etc ...
- Music tagging is a ***multi-label*** classification problem (should not use softmax in the last layer, use sigmoid instead)
- Input: Mel-spectograms of a song

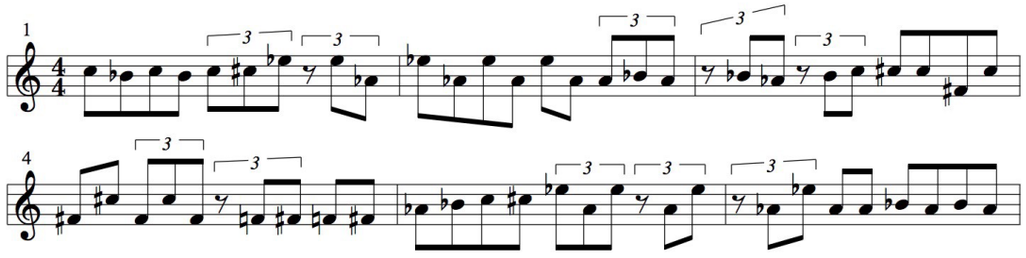


Source: https://github.com/keunwoochoi/music-auto_tagging-keras

Automatic music transcription

Think Siri / Google voice

- Any idea about perfect pitch?
- Music transcription: Turn a piece of music into musical score (and make it playable by electronic instrument)
- Convert song into spectogram, split the spectogram into segments with a length equivalent to 1/8 second
- Model: Convolutional neural network with sigmoid used for last layers
- Output: Notes to be played with specified velocities and time delay



Source: https://medium.com/@dhruvverma/music-transcription-using-a-convolutional-neural-network-b115968829f4

Let's do some coding

Speech command dataset

by google

- About 65,000 one-second long utterances of 30 short words, by thousands of people

- Can go to the website and contribute your speaking: https://aiyprojects.withgoogle.com/open_speech_recording

- Only 10 words are used in this example:

yes
no
on
off
left
right
up
down
stop
go

Source: https://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html

Import the necessary

A few utility libraries

```
> import os
> import librosa          for audio processing
> import warnings         remove unwanted warnings

> import matplotlib.pyplot as plt
> import numpy as np
> import pandas as pd     read csv file
> import sklearn.metrics as metrics

> from scipy.io import wavfile                                  another way to load audio file
> from sklearn.preprocessing import LabelEncoder                Generate labels from data
> from sklearn.model_selection import train_test_split          Split data to train and test set
```

- Librosa: A python package for music and audio analysis; provide the necessary tools to create a music retrieval system

- Librosa can be used to create mel-scaled spectrogram, mel-frequency cepstral coeeficients and etc

Import the necessary
tensorflow.keras

- Import the layers and functions to build, train and test model

```
> from tensorflow.keras.layers import Dense
> from tensorflow.keras.layers import Dropout
> from tensorflow.keras.layers import Flatten
> from tensorflow.keras.layers import Conv1D
> from tensorflow.keras.layers import Input
> from tensorflow.keras.layers import MaxPooling1D
> from tensorflow.keras.models import Model
> from tensorflow.keras.callbacks import ModelCheckpoint
> from tensorflow.keras.callbacks import CSVLogger
> from tensorflow.keras.utils import to_categorical
> from tensorflow.keras.utils import plot_model

> warnings.filterwarnings("ignore")
```

Use Conv1D, not Conv2D

Use MaxPooling1D, not MaxPooling2D

Remove unnecessary warnings from python

Some basic setup

- Set the directory, re-sampling rate, input length, model name and plot style

```
> audioPth       = 'speechsub'
> resmpRate      = 8000
> inputLength    = 8000

> modelname      = 'speechRV1'
```

The folder that stores all the audio data

The sampling rate we are going to use for training and testing

The length of the input; the input is equivalent to 1-second audio, since the input length and the sampling rate is 8000

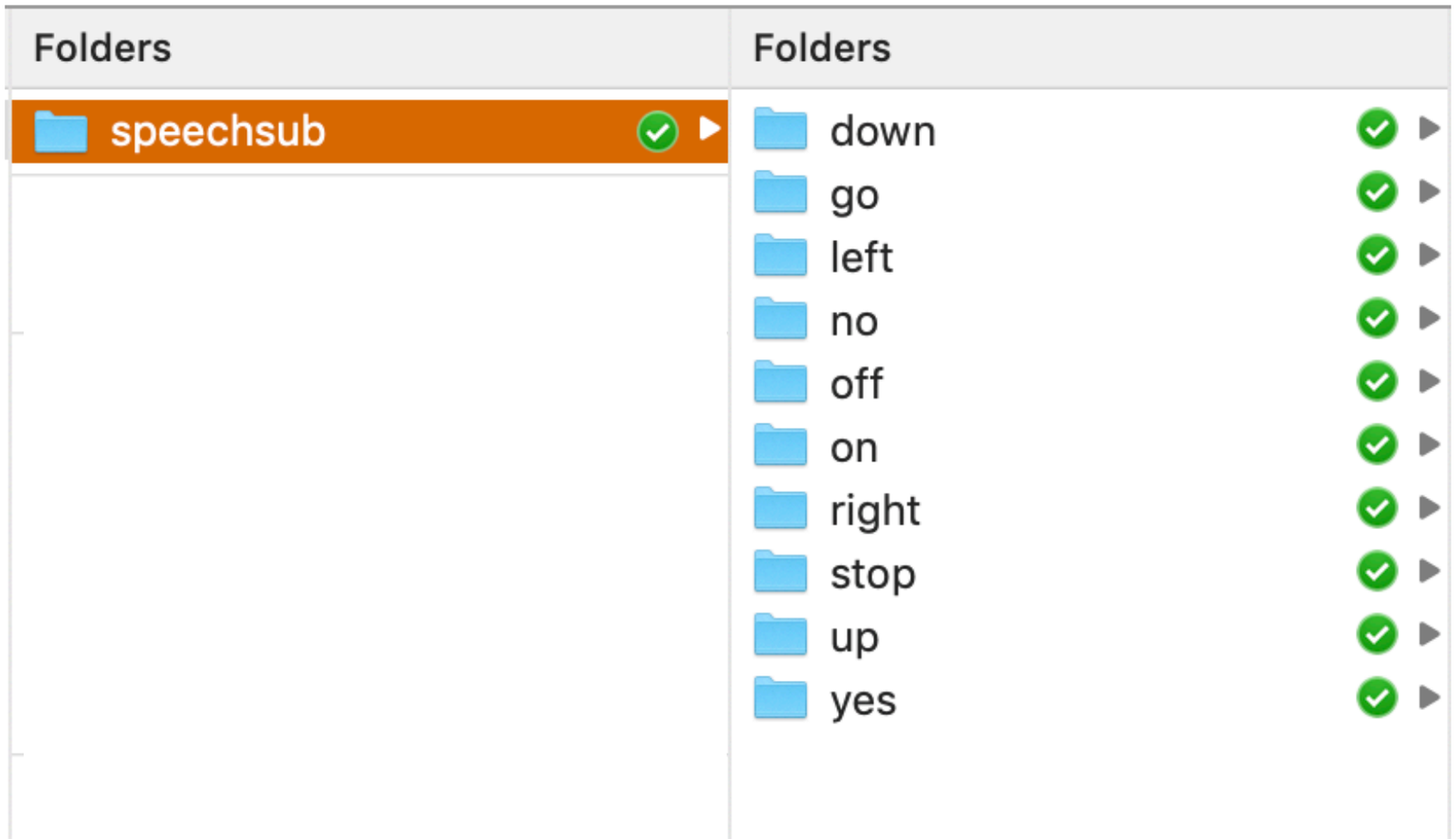
The name for our deep learning model

```
> plt.style.use('ggplot')
> plt.rcParams['ytick.right']     = True
> plt.rcParams['ytick.labelright']= True
> plt.rcParams['ytick.left']      = False
> plt.rcParams['ytick.labelleft'] = False
> plt.rcParams['font.family']     = 'Arial'
```

Setting up the style of our plot

Some basic setup

- The subfolders under ‘speechsub’



Data exploration

Let's take a look at one audio

- A wave file in which a 'go' is pronounced

```
> goSample    = os.path.join(audioPth,
                             'go',
                             '00f0204f_nohash_1.wav')
> (smp,smpR)  = librosa.load(goSample,sr=16000)
```

Get the path for the particular wave file

The sampling rate of the wave file is 16000. Use 'None' for 'sr' to preserve the native sampling rate, else the default will be 22050.

```
> plt.figure(figsize=(8,4))
> plt.plot(np.linspace(0,
                       len(smp)/smpR,
                       len(smp)),
           smp)
> plt.title('A sound of "go" ...')
> plt.xlabel('time in seconds')
> plt.ylabel('amplitude')
```

create a figure and specify the (width, height) in inches

the start value of the series for X-axis

the end value of the series for X-axis (convert sample into second)

the number of points

title of the figure

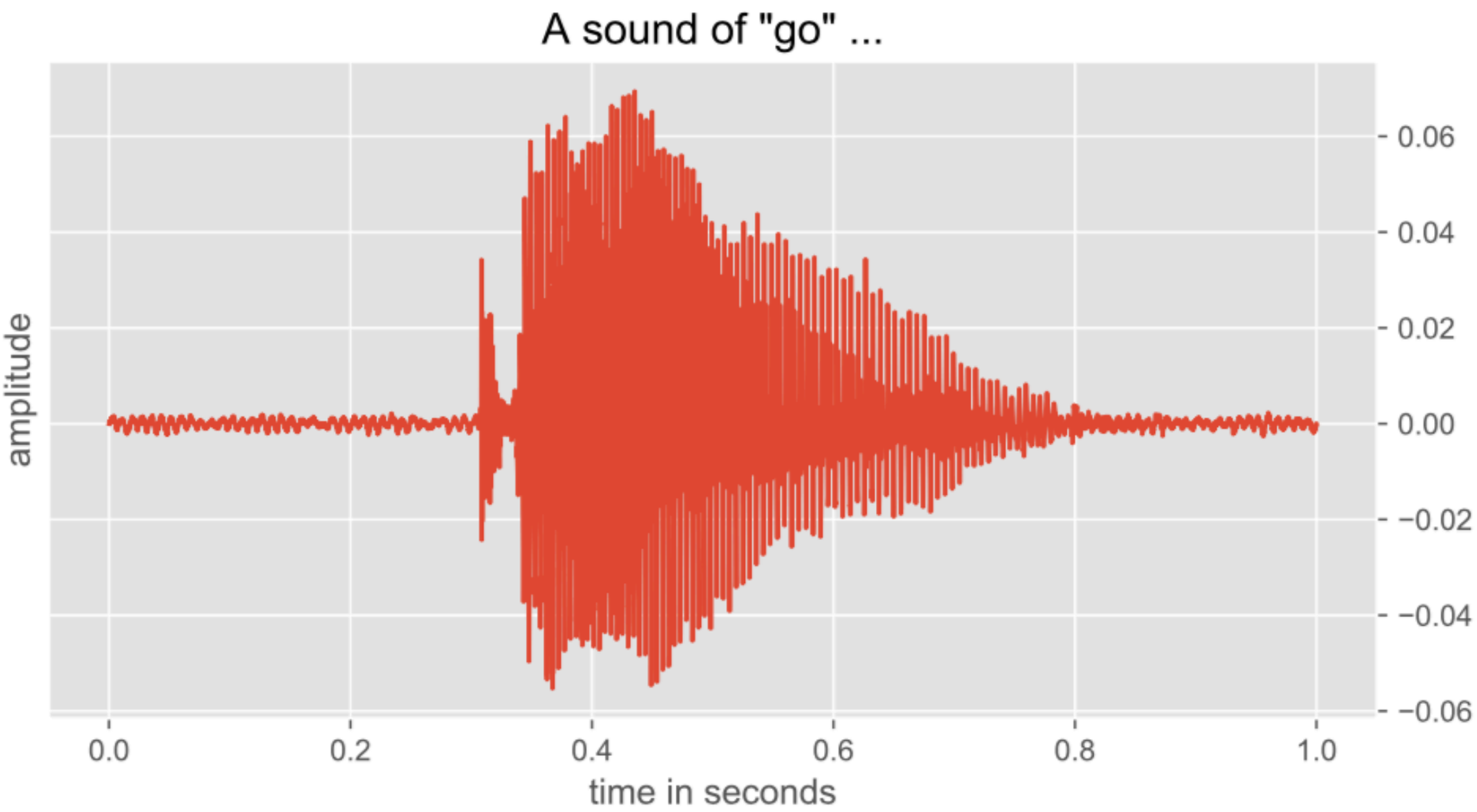
label for X-axis

label for Y-axis

Data exploration

Let's take a look at one audio

- A sound of "go" ...
- What do you observe from the plot?



Data exploration

Inspecting data

- Check the amount of record we have

```
> labels       = ["go",
                  "stop",
                  "yes",
                  "no",
                  "up",
                  "down",
                  "left",
                  "right",
                  "on",
                  "off"]
```

The labels we are going to classify in this example

```
> numOfRecords= []
```

To hold the number of records in each directory

```
> for lbl in labels:
      pth     = os.path.join(audioPth,lbl)
      records = [f for f in os.listdir(pth) if f.endswith('.wav')]

      numOfRecords.append(len(records))
```

The path of each directory

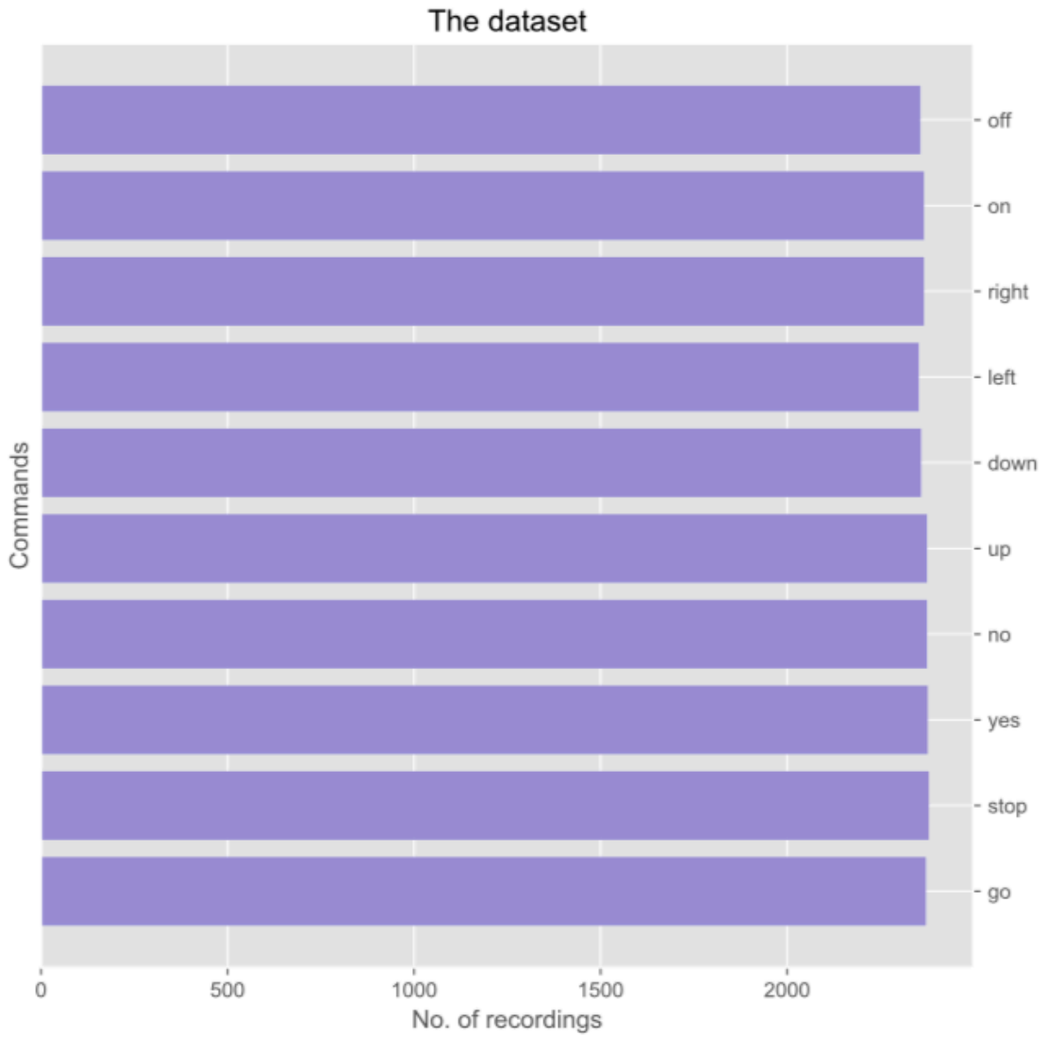
The list to hold all the paths with .wav for one directory

Get the number of records in each directory and append the value to the list

Data exploration

Inspecting data

- From the plot, we can tell the dataset is great
- Why?




```
> plt.figure(figsize=(8,8))
> plt.barh(np.arange(len(labels)),
           numOfRecords,
           color="C2")
> plt.xlabel('No. of recordings')
> plt.ylabel('Commands')
> plt.yticks(np.arange(len(labels)), labels)
> plt.title('The dataset')
> plt.show()
```

Data exploration

How about the distribution of duration?

```
> durations   = []

> for lbl in labels:
      pth     = os.path.join(audioPth,lbl)
      records = [f for f in os.listdir(pth) if f.endswith('.wav')]
      for rcd in records:
          (smpR,smp)  = wavfile.read(os.path.join(pth,rcd))

          durations.append(float(len(smp)/smpR))

> plt.figure(figsize=(8,8))
> plt.title('The distribution of duration')
> plt.ylabel('The number of recordings')
> plt.xlabel('Time in seconds')
> durationHist    = plt.hist(durations)
```

The path of each directory

The list to hold all the paths with .wav for one directory

For each wavefile, we use scipy function 'wavfile' to load the audio

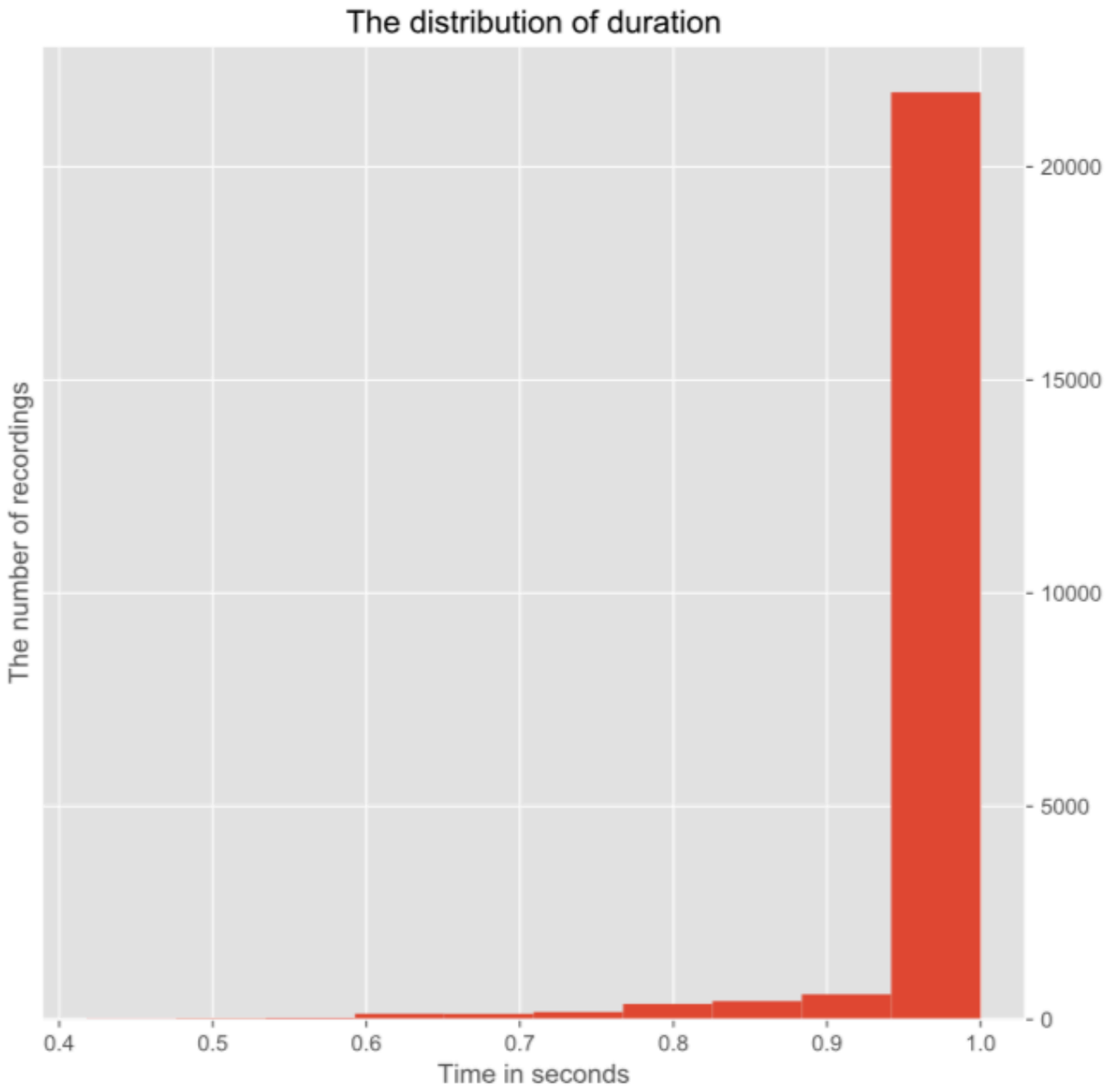
Calculate the duration of each wave: number of samples divided by sample rate

Plot a histgram based on duration

Data exploration

How about the distribution of duration?

- Your observation? Next action?



Data preparation

Get only sample with a duration of 1 second

```
> allRecords  = []
> allLabels   = []

> for lbl in labels:
      pth     = os.path.join(audioPth,lbl)
      records = [f for f in os.listdir(pth) if f.endswith('.wav')]

      for rcd in records:
          (smp,smpR)  = librosa.load(os.path.join(pth,rcd),sr=16000)
          smp         = librosa.resample(smp,
                                         smpR,
                                         resmpRate)
          if(len(smp)== inputLength) :
              allRecords.append(smp)
              allLabels.append(lbl)

> allRecords  = np.array(allRecords).reshape(-1,inputLength,1)
```

Resample each audio from a rate of 16000 to 8000

Only accept a sample with a length 8000, which is equivalent to 1 second

Convert a list to a numpy array; use '-1' in the reshape( ) to get numpy to infer the number for that dimension

Data preparation

Intermediate outputs

Name ▲	Type	Size	Value
allLabels	list	21312	['go', 'go', 'go', 'go', 'go', 'go', 'go', 'go', 'go', 'go', ...]
allRecords	float32	(21312, 8000, 1)	[[[1.6428283e-04] [3.1998314e-04]
audioPth	str	1	speechsub
durationHist	tuple	3	(Numpy array, Numpy array, silent_list)
durations	list	23682	[0.8359375, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, ...]
goSample	str	1	speechsub/go/00f0204f_nohash_1.wav
inputLength	int	1	8000
labels	list	10	['go', 'stop', 'yes', 'no', 'up', 'down', 'left', 'right', 'on', 'off' ...
lbl	str	1	off
modelname	str	1	speechRV1
numOfRecords	list	10	[2372, 2380, 2377, 2375, 2375, 2359, 2353, 2367, 2367, 2357]
pth	str	1	speechsub/off
rcd	str	1	c6389ab0_nohash_0.wav
records	list	2357	['37e8db82_nohash_1.wav', '439c84f4_nohash_1.wav', '6a27a9bf_nohash_0. ...
resmpRate	int	1	8000
smp	float32	(8000,)	[7.7158975e-04 2.0734037e-04 -5.8851430e-05 ... -1.11795… -4.0 ...
smpR	int	1	16000

Data preparation

Prepare label for training

- LabelEncoder encodes the labels to integer
- Since we have 10 classes, the integer starts from 0 to 9

```
> le      = LabelEncoder()
> lbls    = le.fit_transform(allLabels)
```

Encode the labels into integer

```
  lbls      | int64    | (21312,)  | [1 1 1 ... 4 4 4]
```

```
> classes = list(le.classes_)
```

Find out what each integer stands for which class, but do take note, str_ is not string

```
  classes   | list     | 10        | [str_ object of numpy module, str_ ...
```

```
> classes = [str(c) for c in classes]
```

convert str_ to string, so 0 stands for 'down', 1 stands for 'go' .....

```
  classes   | list     | 10        | ['down','go','left','no','off',...
```

```
> lbls    = to_categorical(lbls,num_classes=len(classes))
```

One-hot encode the integer

```
  lbls      | float32  | (21312,10)| [[0. 1. 0. ... 0. 0. 0.]
                                     [0. 1. 0. ... 0. 0. 0.]
```

Data preparation

Split train test set

```
> (trDat,
   vlDat,
   trLbl,
   vlLbl) = train_test_split(allRecords,
                             lbls,
                             stratify=lbls,
                             test_size=0.2,
                             random_state=229,
                             shuffle=True)
```

Make sure the split is applied on each class

20% of the total samples are used for validation

The random number to determine the way the function shuffles and splits the dataset

trDat	float32	(17049, 8000, 1)
trLbl	float32	(17049, 10)
vlDat	float32	(4263, 8000, 1)
vlLbl	float32	(4263, 10)

Create model

Conv1D, Maxpooling1D

```
> def createModel(inputSize):
    ipt = Input(shape=(inputSize,1))
    x   = Conv1D(8, 11, padding='valid', activation='relu')(ipt)
    x   = MaxPooling1D(4)(x)
    x   = Dropout(0.25)(x)

    x   = Conv1D(16, 11, padding='valid', activation='relu')(x)
    x   = MaxPooling1D(4)(x)
    x   = Dropout(0.25)(x)

    x   = Conv1D(32, 11, padding='valid', activation='relu')(x)
    x   = MaxPooling1D(4)(x)
    x   = Dropout(0.25)(x)

    x   = Conv1D(64, 11, padding='valid', activation='relu')(x)
    x   = MaxPooling1D(4)(x)
    x   = Dropout(0.25)(x)

    x   = Flatten()(x)
    x   = Dense(256, activation='relu')(x)
    x   = Dropout(0.5)(x)
    x   = Dense(128, activation='relu')(x)
    x   = Dropout(0.5)(x)
    x   = Dense(len(classes), activation='softmax')(x)

    model = Model(ipt, x)
    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    return model
```

Create model

Conv1D, Maxpooling1D

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 8000, 1)           0
_________________________________________________________________
conv1d (Conv1D)              (None, 7990, 8)           96
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 1997, 8)           0
_________________________________________________________________
dropout (Dropout)            (None, 1997, 8)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 1987, 16)          1424
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 496, 16)           0
_________________________________________________________________
dropout_1 (Dropout)          (None, 496, 16)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 486, 32)           5664
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 121, 32)           0
_________________________________________________________________
dropout_2 (Dropout)          (None, 121, 32)           0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 111, 64)           22592
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 27, 64)            0
_________________________________________________________________
dropout_3 (Dropout)          (None, 27, 64)            0
_________________________________________________________________
flatten (Flatten)            (None, 1728)              0
_________________________________________________________________
dense (Dense)                (None, 256)               442624
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 506,586
Trainable params: 506,586
Non-trainable params: 0
_________________________________________________________________
```

Training setup

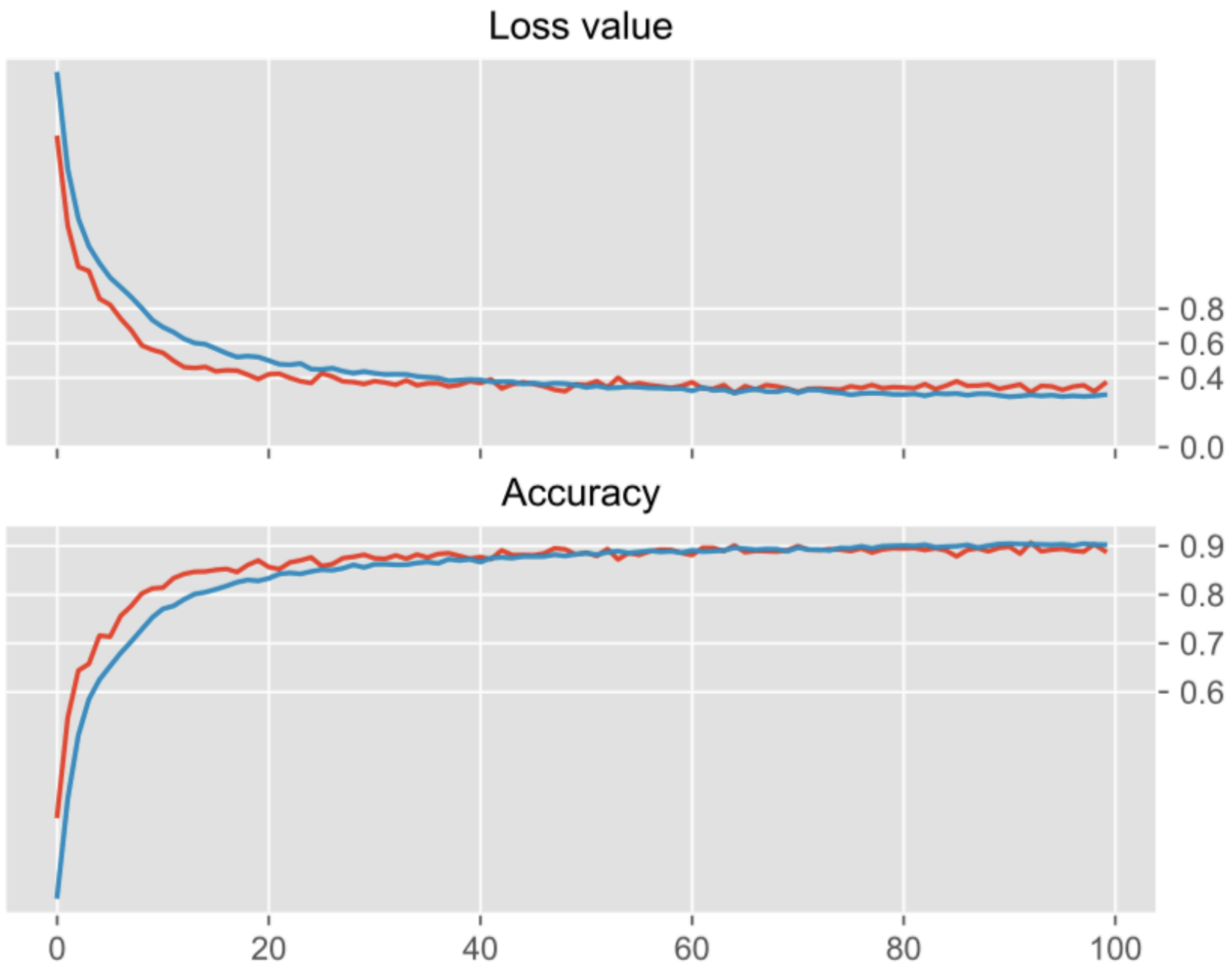
And also result

```
> model.fit(trDat,
            trLbl,
            validation_data=(vlDat, vlLbl),
            epochs=100,
            batch_size=32,
            shuffle=True,
            callbacks=callbacks_list)
```

```
Best accuracy (on testing dataset): 90.10%
              precision    recall  f1-score   support

        down     0.9398    0.8721    0.9047       430
          go     0.9062    0.8976    0.9019       420
        left     0.8953    0.8891    0.8922       433
          no     0.8753    0.9190    0.8966       420
         off     0.8878    0.8671    0.8774       429
          on     0.9386    0.9074    0.9227       421
       right     0.9519    0.9188    0.9351       431
        stop     0.9319    0.9126    0.9222       435
          up     0.7983    0.9029    0.8474       412
         yes     0.9007    0.9236    0.9120       432

    accuracy                         0.9010      4263

[[375  19   3   8   0   6   1   8   3   7]
 [  9 377   0  12   1   0   1   3  10   7]
 [  0   1 385  13   0   1   5   3   7  18]
 [  6   9   4 386   1   0   2   0   7   5]
 [  0   1   1   4 372  14   1   6  29   1]
 [  4   1   0   2  16 382   4   1  11   0]
 [  1   1  18   2   0   2 396   1   6   4]
 [  3   4   5   0   6   0   0 397  19   1]
 [  0   3   2   2  23   2   1   6 372   1]
 [  1   0  12  12   0   0   5   1   2 399]]
```

Training setup

And also result



training

testing

Simple prediction

on a single wave file

- Create a function to make prediction on audio

```
> def commandPred(file):
      (smp,smpR)  = librosa.load(file,sr=16000)
      smp         = librosa.resample(smp,
                                     smpR,
                                     resmpRate)
      smp         = smp.reshape(-1,inputLength,1)
      pred        = modelGo.predict(smp)
      pred        = np.argmax(pred,axis=1)
      return classes[pred[0]]
```

```
> wfile       = 'voice01.wav'
> pred        = commandPred(wfile)
> print("")
> print("The command predicted from '%s' is '%s'." % (wfile,pred) )
```

```
:The command predicted from 'voice01.wav' is 'right'.
```