trigger_word_detection

March 28, 2020

Project created during the Deep Learning Specialization course on www.coursera.or

1 Trigger word detection

[1]:
```
import numpy as np
from pydub import AudioSegment # Audio files editing
import random
import sys
import io
import os
from scipy.io import wavfile
import glob # Finds pathnames matching a pattern
import IPython # Interactive Python
import matplotlib.pyplot as plt
%matplotlib inline
```

1.1 Data preparation

[2]:
```
# Exampleo of a trigger word
IPython.display.Audio('./raw_data/activates/1.wav')
```

[2]:
```
<IPython.lib.display.Audio object>
```

[3]:
```
# Example of a non-trigger word
IPython.display.Audio('./raw_data/negatives/4.wav')
```

[3]:
```
<IPython.lib.display.Audio object>
```

[4]:
```
# Example of a background noise
IPython.display.Audio('./raw_data/backgrounds/1.wav')
```

[4]:
```
<IPython.lib.display.Audio object>
```

[5]:
```
# Load raw audio clips

def load_raw_audio():
```

```
    # Initialize lists to store clips
    activates = []
    backgrounds = []
    negatives = []

    # Load audio files in a list

    # Number of timesteps varies depending on a file
    for filename in os.listdir("./raw_data/activates"):
        if filename.endswith("wav"):
            activate = AudioSegment.from_wav("./raw_data/activates/"+filename)
            activates.append(activate)
    for filename in os.listdir("./raw_data/backgrounds"):

        # (10 s clip / 0.001 s) (1 ms discretization of pydub) => 10000 timesteps
        if filename.endswith("wav"):
            background = AudioSegment.from_wav("./raw_data/backgrounds/
↪"+filename)
            backgrounds.append(background)

    # Number of timesteps varies depending on a file
    for filename in os.listdir("./raw_data/negatives"):
        if filename.endswith("wav"):
            negative = AudioSegment.from_wav("./raw_data/negatives/"+filename)
            negatives.append(negative)

    return activates, negatives, backgrounds
```

[6]:
```
# Load audio segments using pydub
activates, negatives, backgrounds = load_raw_audio()
```

1.2 Data synthesis

[7]:
```
# Helper function: select random time segment of background noise audio

def get_random_time_segment(audio_length):

    # Define start of the segment (has to fit in 10 s backgroud audio)
    start = np.random.randint(low=0, high=(10000-audio_length))

    # Define end of the segment
    end = start + audio_length - 1

    return (start, end)
```

[8]:
```
# Helper function: check for the overlap between selected time segments

def overlap(time_segment, previous_segments):

    # Retrieve start and end of the new segment
    start, end = time_segment

    # Initialize overlap variable
    overlap = False

    # Iterate over previous segments tuples to check for overlap
    for previous_start, previous_end in previous_segments:
        if (start <= previous_end) and (end >= previous_start):
            overlap = True

    return overlap
```

[9]:
```
# Insert audio clip (trigger or non-triger word) to background audio

def insert_audio_clip(background, audio_clip, previous_segments):

    # Get random time segment
    time_segment = get_random_time_segment(len(audio_clip))

    # Check for overlap
    while overlap(time_segment, previous_segments):
        time_segment = get_random_time_segment(len(audio_clip))

    previous_segments.append(time_segment)

    # Insert audio into background clip
    new_background = background.overlay(audio_clip, position=time_segment[0])

    return new_background, time_segment
```

[10]:
```
# Helper function: prepare labels - set to 1 segments following the trigger word

def insert_ones(y, segment_end):

    # Convert timesteps into output format
    segment_end_conv = int(segment_end * Ty / 10000.)

    # Take 50 timesteps that follow the end of the segment and set them to 1
    for i in range(segment_end_conv+1, segment_end_conv+51):
        if i < y.shape[1]:
            y[0, i] = 1
```

```
    return y
```

1.3 Synthesized data transformation (Fourier transform)

[11]:
```
# Load sample audio

IPython.display.Audio("./audio_examples/example_train.wav")
```

[11]:
```
<IPython.lib.display.Audio object>
```

[12]:
```
_, data = wavfile.read('./audio_examples/example_train.wav')
```

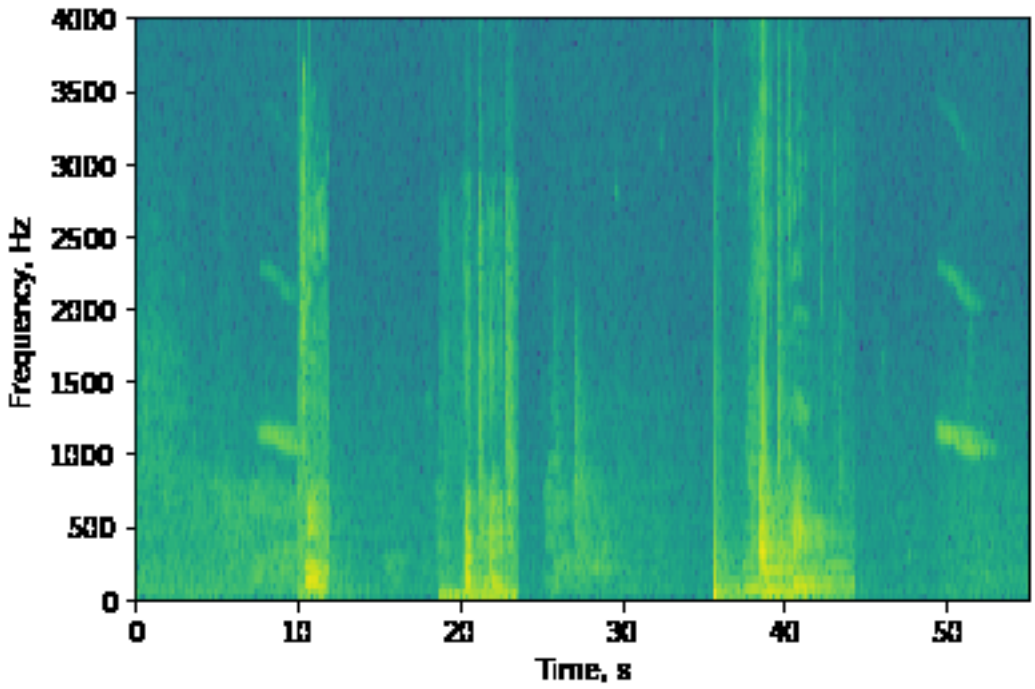
[13]:
```
# Timesteps in audio recording before the transform: audios were recorded at␣
↪sample rate of 44100 Hz
# 44100 1/sec x 10 sec = 441000
data[:, 0].shape
```

[13]:
```
(441000,)
```

[14]:
```
# Plot spectogram for a wav audio file: how active each frequency is over time

def graph_spectrogram(wav_file):

    rate, data = wavfile.read(wav_file)

    # Define length of each window segment (for Fast Fourier Transform)
    nfft = 200

    # Define sampling frequency
    fs = 8000

    # Define amount of (points) overlap of each window segments
    noverlap = 120

    # Retrieve number of channels
    nchannels = data.ndim

    # Plot spectrogram
    if nchannels == 1:
        spectrum, freqs, bins, im = plt.specgram(data, nfft, fs,␣
↪noverlap=noverlap)

    elif nchannels == 2:
        spectrum, freqs, bins, im = plt.specgram(data[:, 0], nfft, fs,␣
↪noverlap=noverlap)
```

```
    # Add labels
    plt.xlabel('Time, s')
    plt.ylabel('Frequency, Hz')

    return spectrum
```

[15]:
```
# Transform a sample audio
spectrum = graph_spectrogram("./audio_examples/example_train.wav")
```



[16]:
```
'''
Spectrum:
columns: periodograms of successive segments (timesteps after the transform)
rows: frequencies
'''
# Retrieve number of frequencies in the input
n_freq = spectrum.shape[0]
print('n_freq = ', n_freq)

# Retrieve number of timesteps in the input (out of 10 second audio)
Tx = spectrum.shape[1]
print('Tx = ', Tx)

# Define number of timesteps in the output
Ty = 1375
```

```
print('Ty = ', Ty)
```

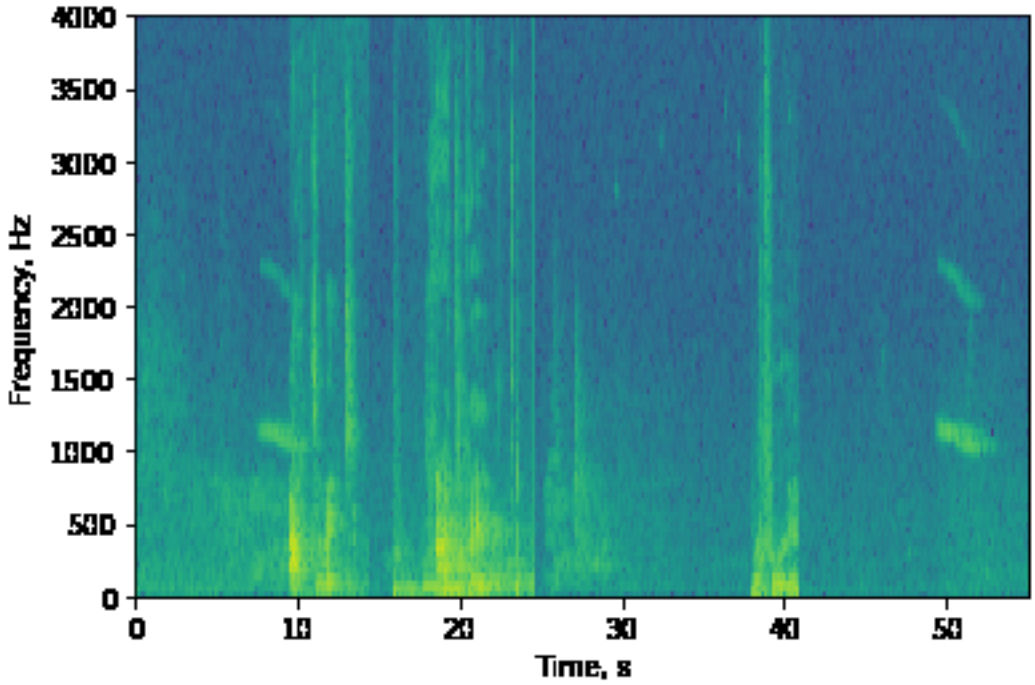
```
n_freq =  101
Tx =  5511
Ty =  1375
```

1.4 Training set preparation

[17]:
```
# Create a training example

def create_training_example(background, activates, negatives):

    np.random.seed(18)

    # Make background quieter
    background = background - 20

    # Initialize output vector and list of inserted segments
    y = np.zeros((1, Ty))
    previous_segments = []

    # Select 0-4 random "activate" audio clips from the entire list of␣
↪"activates" recordings
    number_of_activates = np.random.randint(0, 5)
    random_indices = np.random.randint(len(activates), size=number_of_activates)
    random_activates = [activates[i] for i in random_indices]

    # Iterate over randomly selected activate clips
    for random_activate in random_activates:

        # Insert "activate" clips to background clip
        background, segment_time = insert_audio_clip(background,␣
↪random_activate, previous_segments)

        # Update label vector
        y = insert_ones(y, segment_time[1])

    # Select 0-2 random negatives audio recordings from the entire list of␣
↪"negatives" recordings
    number_of_negatives = np.random.randint(0, 3)
    random_indices = np.random.randint(len(negatives), size=number_of_negatives)
    random_negatives = [negatives[i] for i in random_indices]

    # Iterate over randomly selected activate clips
    for random_negative in random_negatives:

        # Insert "negative" clips to background clip
```

```
        background, _ = insert_audio_clip(background, random_negative,␣
↪previous_segments)

    # Standardize the volume of the audio clip
    target_dBFS = -20.0
    change_in_dBFS = target_dBFS - background.dBFS
    background.apply_gain(change_in_dBFS)

    # Export new training example
    file_handle = background.export("new_training_example" + ".wav",␣
↪format="wav")

    # Get and plot spectrogram of the new recording (background with␣
↪superposition of positive and negatives)
    x = graph_spectrogram("./new_training_example.wav")

    return x, y
```

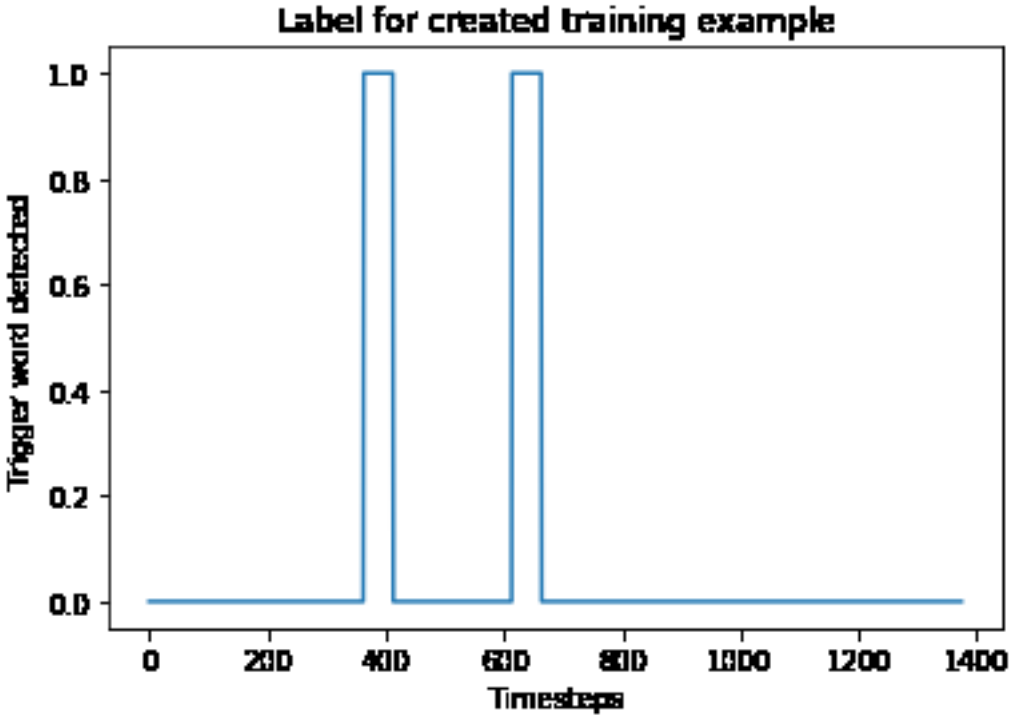
[18]:
```
# Create a training example
x, y = create_training_example(backgrounds[1], activates, negatives)
```



[19]:
```
# Load created audio
IPython.display.Audio("new_training_example.wav")
```

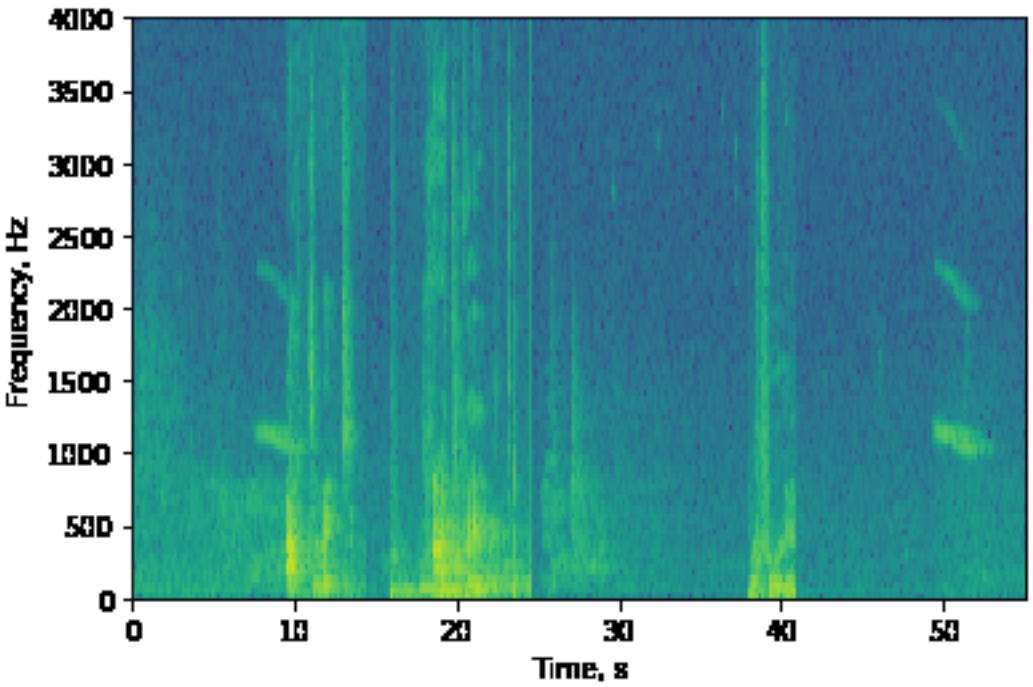
[19]: <IPython.lib.display.Audio object>

[20]:
```
# Plot the graph showing the
plt.plot(y[0])

# Add axis labels
plt.title('Label for created training example')
plt.xlabel('Timesteps')
plt.ylabel('Trigger word detected')
plt.show()
```



[21]:
```
# Create dataset

X_train = np.zeros((26, 5511, 101))
Y_train = np.zeros((26, 1375, 1))

for i in range(26):
    background = backgrounds[0]
    if i > 13:
        background = backgrounds[1]
    x, y = create_training_example(background, activates, negatives)
    X_train[i, :, :] = x.T
    Y_train[i, :, :] = y.T
```

```
/Applications/anaconda3/lib/python3.6/site-
packages/matplotlib/axes/_axes.py:7564: RuntimeWarning: divide by zero
encountered in log10
  Z = 10. * np.log10(spec)
```



[22]:
```
print('X_train', X_train.shape)
print('Y_train', Y_train.shape)
```

```
X_train (26, 5511, 101)
Y_train (26, 1375, 1)
```

1.5 Model architecture

[23]:
```
from keras.models import Model, load_model
from keras.layers import Dense, Activation, Dropout, Input, TimeDistributed,␣
 ↪Conv1D, GRU, BatchNormalization
from keras.optimizers import Adam
```

```
/Applications/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36:
FutureWarning: Conversion of the second argument of issubdtype from `float` to
`np.floating` is deprecated. In future, it will be treated as `np.float64 ==
np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
/Applications/anaconda3/lib/python3.6/site-
```

```
packages/tensorflow/python/framework/dtypes.py:458: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/Applications/anaconda3/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:459: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/Applications/anaconda3/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:460: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/Applications/anaconda3/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:461: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/Applications/anaconda3/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:462: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/Applications/anaconda3/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:465: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

[24]:
```
def model(input_shape):

    X_input = Input(shape=input_shape)

    X = Conv1D(filters=196, kernel_size=15, strides=4)(X_input)
    X = BatchNormalization()(X)
    X = Activation('relu')(X)
    X = Dropout(0.8)(X)

    X= GRU(128, return_sequences=True)(X)
    X = Dropout(0.8)(X)
    X = BatchNormalization()(X)

    X= GRU(128, return_sequences=True)(X)
    X = Dropout(0.8)(X)
    X = BatchNormalization()(X)
    X = Dropout(0.8)(X)
```

```
    X = TimeDistributed(Dense(1, activation='sigmoid'))(X)

    model = Model(input=X_input, outputs=X)

    return model
```

[25]:
```
model = model(input_shape=(Tx, n_freq))
```

```
/Applications/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:21:
UserWarning: Update your `Model` call to the Keras 2 API:
`Model(outputs=Tensor("ti..., inputs=Tensor("in...)`
```

[26]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 5511, 101)         0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 1375, 196)         297136
_________________________________________________________________
batch_normalization_1 (Batch (None, 1375, 196)         784
_________________________________________________________________
activation_1 (Activation)    (None, 1375, 196)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1375, 196)         0
_________________________________________________________________
gru_1 (GRU)                  (None, 1375, 128)         124800
_________________________________________________________________
dropout_2 (Dropout)          (None, 1375, 128)         0
_________________________________________________________________
batch_normalization_2 (Batch (None, 1375, 128)         512
_________________________________________________________________
gru_2 (GRU)                  (None, 1375, 128)         98688
_________________________________________________________________
dropout_3 (Dropout)          (None, 1375, 128)         0
_________________________________________________________________
batch_normalization_3 (Batch (None, 1375, 128)         512
_________________________________________________________________
dropout_4 (Dropout)          (None, 1375, 128)         0
_________________________________________________________________
time_distributed_1 (TimeDist (None, 1375, 1)           129
=================================================================
Total params: 522,561
Trainable params: 521,657
Non-trainable params: 904
```

```
---------------------------------------------------------------
```

[27]:
```
# Load pretrained model (provided on coursera hub)
model = load_model('./model/tr_model.h5')
```

[28]:
```
# Requirements for the model to load correctly
import keras
keras.__version__
```

[28]:
```
'2.0.7'
```

[29]:
```
import tensorflow
tensorflow.__version__
```

[29]:
```
'1.2.1'
```

1.6 Training

[30]:
```
# Define optimizer
optimizer = Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, decay=0.01)
```

[31]:
```
# Compile the model
model.compile(loss='binary_crossentropy', optimizer=optimizer,␣
↪metrics=['accuracy'])
```

[32]:
```
# Train the model
model.fit(X_train, Y_train, batch_size=5, epochs=1)
```

```
Epoch 1/1
26/26 [==============================] - 14s - loss: 0.2951 - acc: 0.9144
```

[32]:
```
<keras.callbacks.History at 0xd32abbda0>
```

1.7 Validation

[33]:
```
## Load dev set (found here: https://gitlab-cw9.centralesupelec.fr/
↪codingweeksstaff/cs_codingweek_audio/tree/master/trigger_word_detection/
↪XY_val)
X_dev = np.load('./dev/X_dev.npy')
Y_dev = np.load('./dev/Y_dev.npy')
```

[34]:
```
# Evaluate model on dev set
loss, acc = model.evaluate(X_dev, Y_dev)
print(acc)
```

```
25/25 [==============================] - 2s
0.9499345421791077
```

1.8 Testing

[35]:
```
# Helper function: make predictions

def detect_triggerword(filename):

    # Plot spectrogram on the first subplot
    plt.subplot(2,1,1)

    # Compute spectrum of the file
    x = graph_spectrogram(filename)

    # Convert spectrum format to coincide with model input format
    x = x.T
    x = np.expand_dims(x, axis=0)

    # Make predictions
    predictions = model.predict(x)

    # Plot predictions on the second subplot
    ax2 = plt.subplot(2,1,2)
    ax2.plot(predictions[0, :, 0])
    plt.ylabel('probability')
    plt.ylim(0, 1)
    plt.show()

    return predictions
```

[36]:
```
# Helper function: add the sound of a chime when the trigger word is detected

chime_file = './audio_examples/chime.wav'

def chime_on_activate(filename, predictions, threshold):

    # Open the audio file and chime file
    audio_clip = AudioSegment.from_wav(filename)

    chime = AudioSegment.from_wav(chime_file)

    # Retrieve amount of timesteps
    Ty = predictions.shape[1]

    # Initialize
    consecutive_timesteps = 0

    for i in range(Ty):
```

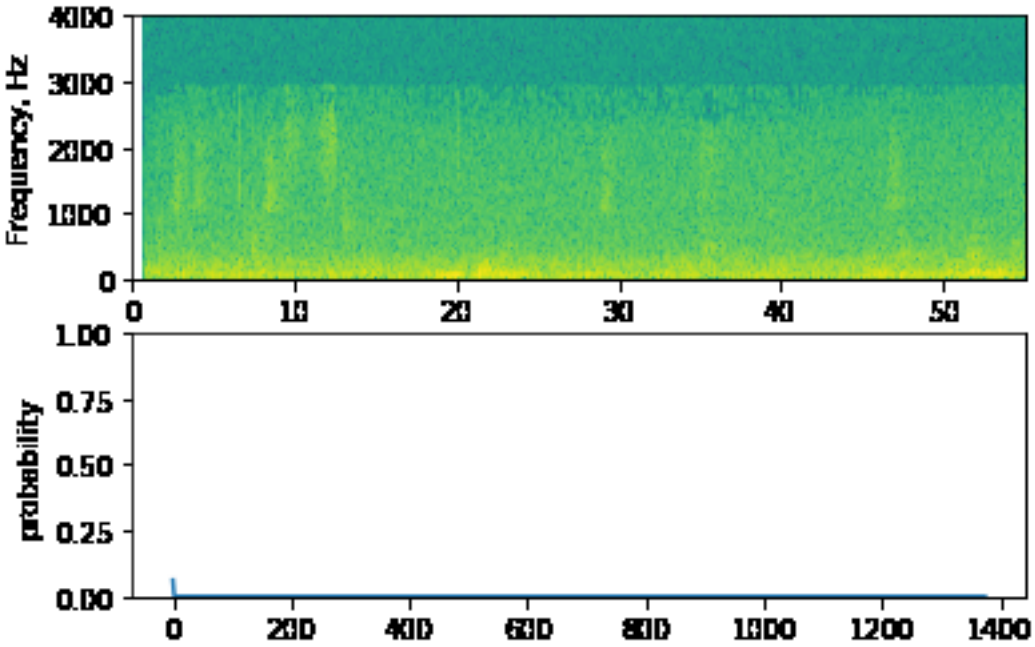
```
            consecutive_timesteps += 1

            # Add sound of a chime, prediction prob is higher than the threshold
            if predictions[0, i, 0] > threshold:
                if consecutive_timesteps > 150:

                    # Avoid adding multiple chimes for the same trigger word
                    audio_clip = audio_clip.overlay(chime, position=((i / Ty) *␣
↪audio_clip.duration_seconds)*1000)

                    consecutive_timesteps = 0

        # Export modified audio
        path, file_extension = os.path.splitext(filename)
        audio_clip.export(path + '-output.wav', format='wav')
```

[37]:
```
# Helper function: preprocess the audio to the correct format

def preprocess_audio(filename):

    # Trim or pad audio segment to 10000ms
    padding = AudioSegment.silent(duration=10000)
    segment = AudioSegment.from_wav(filename)[:10000]
    segment = padding.overlay(segment)

    # Set frame rate to 44100
    segment = segment.set_frame_rate(44100)

    # Export as wav
    segment.export(filename, format='wav')
```

[38]:
```
# Perform inference

def inference(path):

    prediction = detect_triggerword(path)
    chime_on_activate(path, prediction, 0.5)
    path, file_extention = os.path.splitext(path)
    return IPython.display.Audio(path + '-output.wav')
```

[39]:
```
inference('./test/1.wav') # silent audio, no chime
```

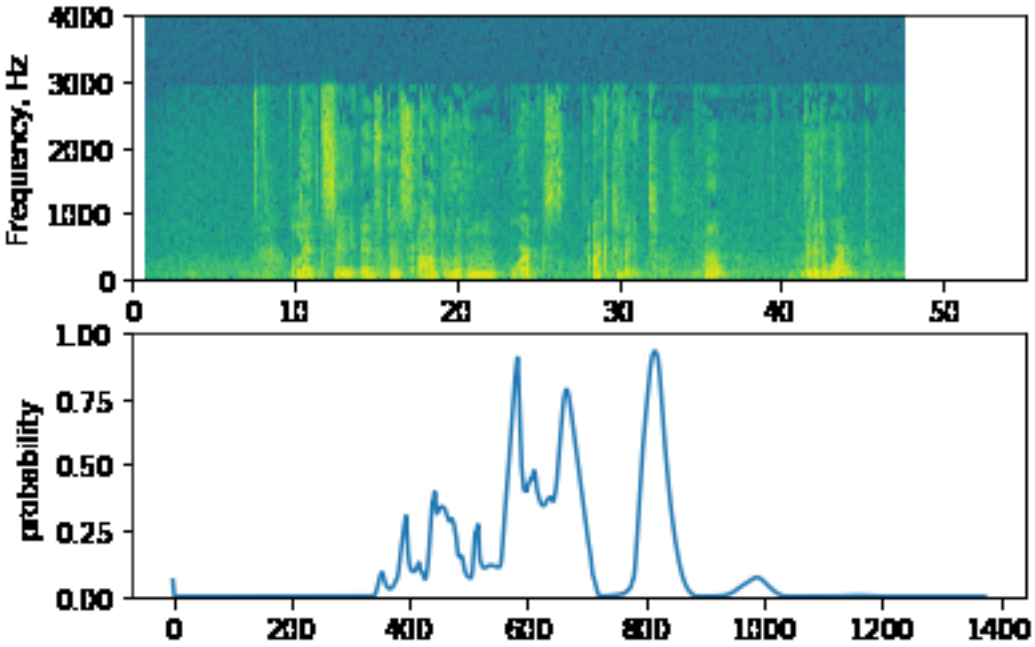
```
/Applications/anaconda3/lib/python3.6/site-
packages/matplotlib/axes/_axes.py:7564: RuntimeWarning: divide by zero
encountered in log10
  Z = 10. * np.log10(spec)
```



[39]: <IPython.lib.display.Audio object>

[40]:
```
inference('./test/2.wav') # chimed on 'activate' and 'algorithm'
```

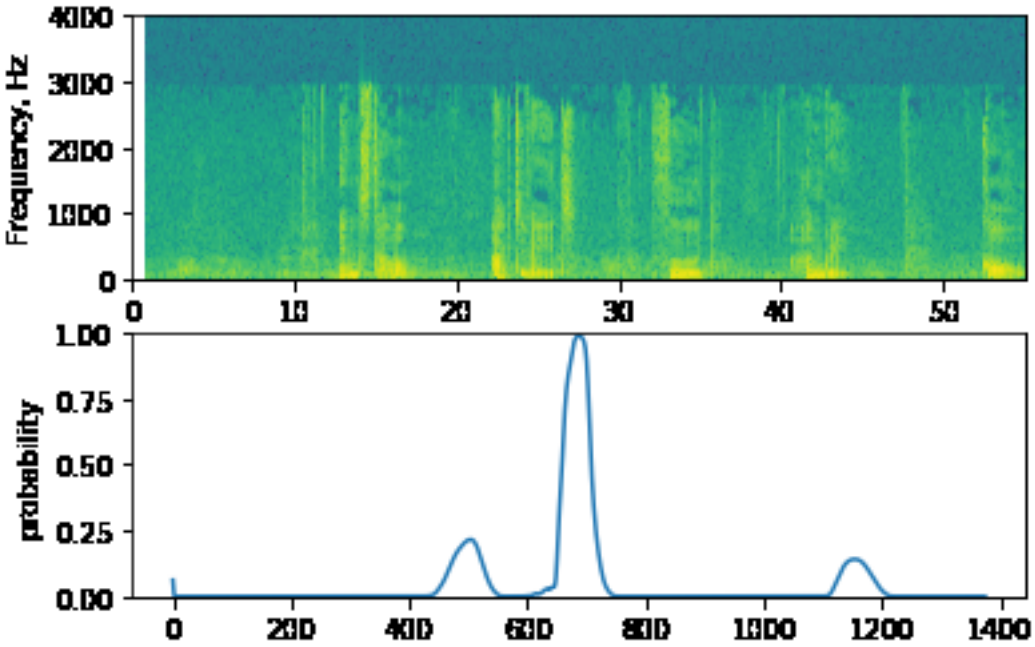
```
/Applications/anaconda3/lib/python3.6/site-
packages/matplotlib/axes/_axes.py:7564: RuntimeWarning: divide by zero
encountered in log10
  Z = 10. * np.log10(spec)
```



[40]: <IPython.lib.display.Audio object>

[41]:
```
inference('./test/3.wav') # chimed on 'activate'
```

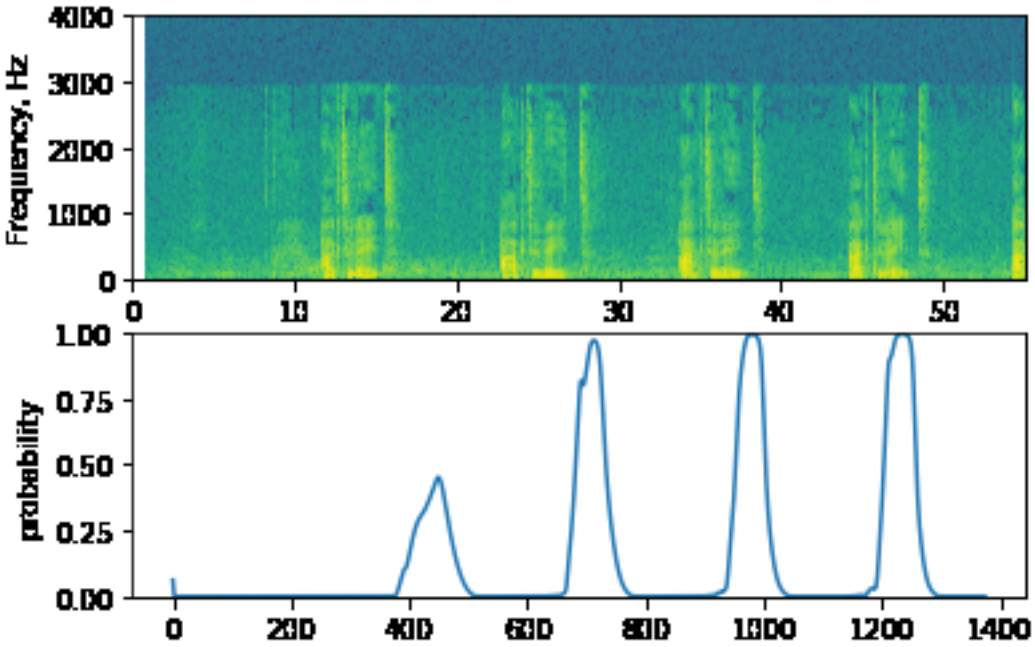
```
/Applications/anaconda3/lib/python3.6/site-
packages/matplotlib/axes/_axes.py:7564: RuntimeWarning: divide by zero
encountered in log10
  Z = 10. * np.log10(spec)
```



[41]: <IPython.lib.display.Audio object>

[42]:
```
inference('./test/4.wav') # chimed 3 times on 'activate', missed the first one
```

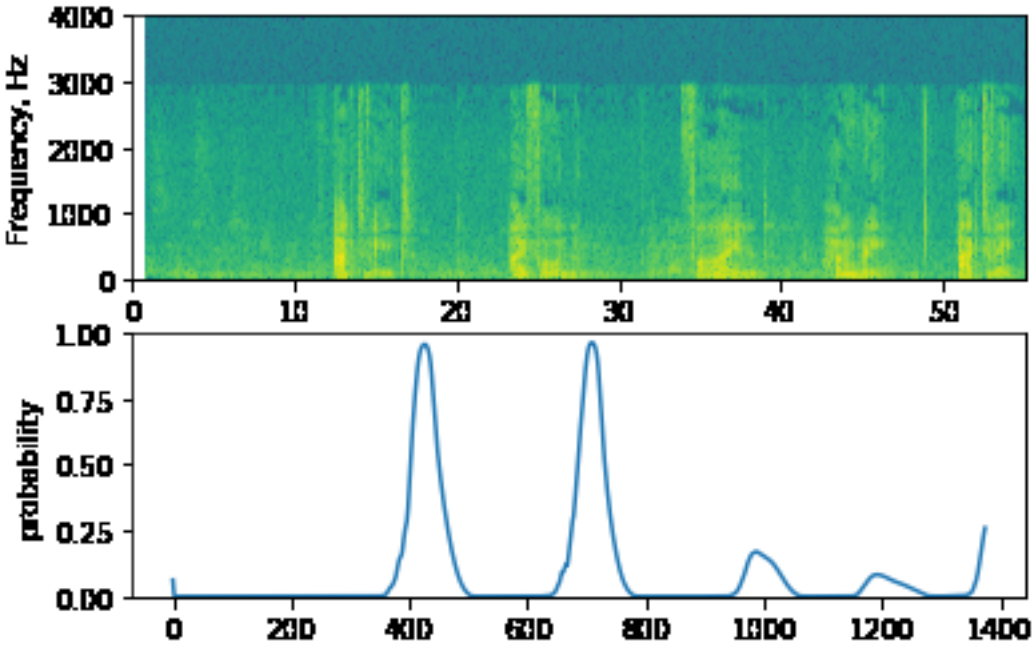
```
/Applications/anaconda3/lib/python3.6/site-
packages/matplotlib/axes/_axes.py:7564: RuntimeWarning: divide by zero
encountered in log10
  Z = 10. * np.log10(spec)
```



[42]: <IPython.lib.display.Audio object>

[43]:
```
inference('./test/5.wav') # chimed on 'activate' and 'iPhone'
```

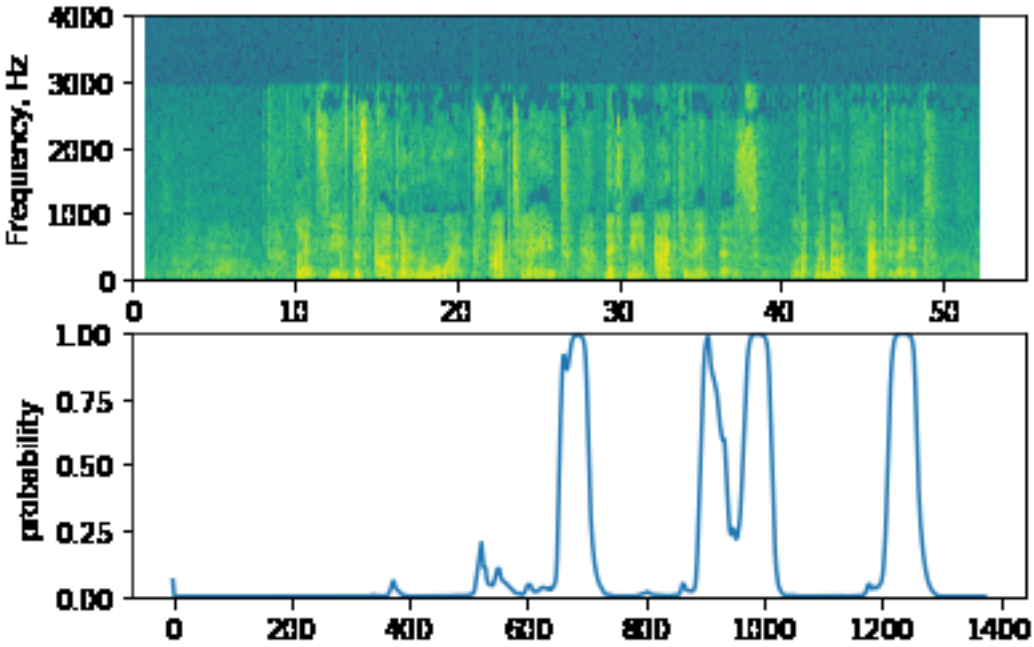
```
/Applications/anaconda3/lib/python3.6/site-
packages/matplotlib/axes/_axes.py:7564: RuntimeWarning: divide by zero
encountered in log10
  Z = 10. * np.log10(spec)
```



[43]: <IPython.lib.display.Audio object>

[44]:
```
inference('./test/6.wav') # chimed three times on 'activate'
```

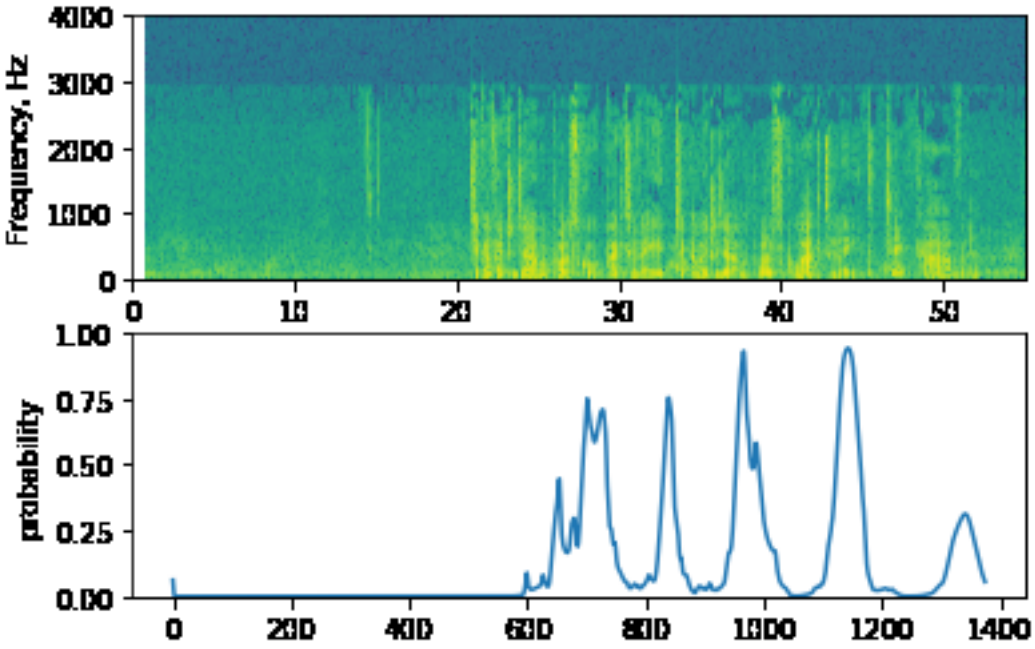
```
/Applications/anaconda3/lib/python3.6/site-
packages/matplotlib/axes/_axes.py:7564: RuntimeWarning: divide by zero
encountered in log10
  Z = 10. * np.log10(spec)
```



[44]: <IPython.lib.display.Audio object>

[45]:
```
inference('./test/7.wav') # chimed on 'działa' and twice on 'activate'
```

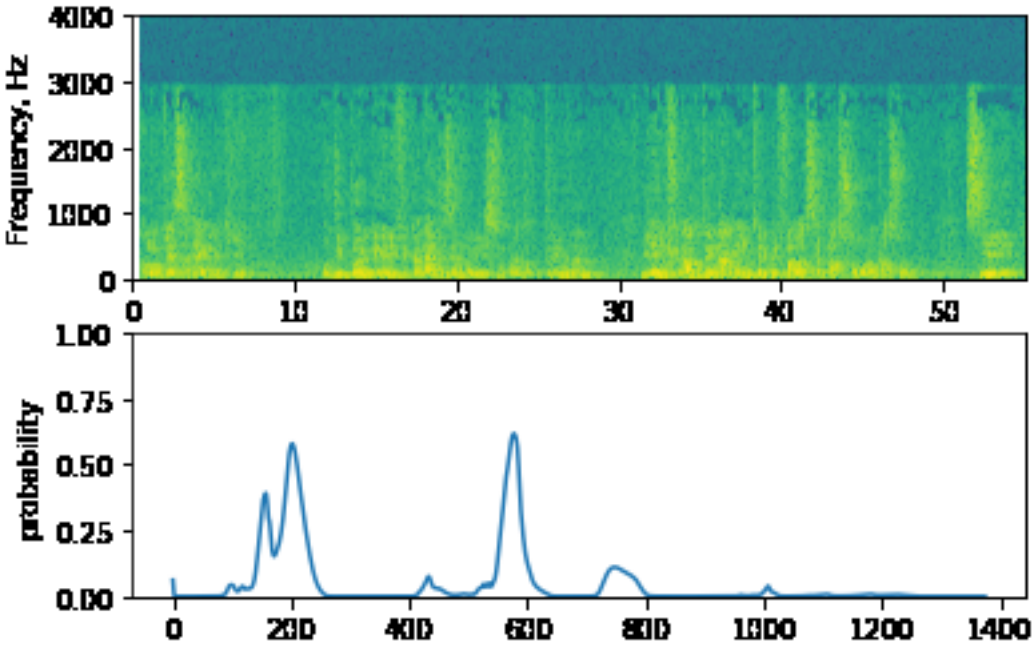
```
/Applications/anaconda3/lib/python3.6/site-
packages/matplotlib/axes/_axes.py:7564: RuntimeWarning: divide by zero
encountered in log10
  Z = 10. * np.log10(spec)
```



[45]: <IPython.lib.display.Audio object>

[46]:
```
inference('./test/8.wav') # background noise, chimes twice
```

```
/Applications/anaconda3/lib/python3.6/site-
packages/matplotlib/axes/_axes.py:7564: RuntimeWarning: divide by zero
encountered in log10
  Z = 10. * np.log10(spec)
```



[46]: <IPython.lib.display.Audio object>

[ ]: