

27-29주차

Tensorflow Speech Recognition

2017010698
수학과 오서영

0. Overview



Featured Prediction Competition

TensorFlow Speech Recognition Challenge

Can you build an algorithm that understands simple speech commands?

\$25,000

Prize Money



Google Brain · 1,314 teams · 3 years ago

Competition

- you're challenged to use the Speech Commands Dataset to build an algorithm that understands simple spoken commands.
- There are only 12 possible labels for the Test set :
 - yes, no, up, down, left, right, on, off, stop, go, silence, unknown

1. Visualization

check!

- There are two theories of a human hearing - place and temporal In speech recognition, I see two main tendencies - to input spectrogram (frequencies), and more sophisticated features MFCC - Mel-Frequency Cepstral Coefficients, PLP. You rarely work with raw, temporal data.

```
# Wave and spectrogram  
train_audio_path = 'train/audio/'  
filename = '/yes/0a7c2a8d_nohash_0.wav'  
samples, sample_rate = librosa.load(str(train_audio_path)+filename)
```

check!

- Note, that we are taking logarithm of spectrogram values.
It will make our plot much more clear, moreover, it is strictly connected to the way people hear.
We need to assure that there are no 0 values as input to logarithm.

Spectrogram : [소리](#)나 [파동](#)을 시각화하여 파악하기 위한 도구

1. Visualization

Define a function that calculates spectrogram.

```
def log_spectrogram(audio, sample_rate, window_size=20,
                    step_size=10, eps=1e-10):
    nperseg = int(round(window_size * sample_rate / 1e3))
    noverlap = int(round(step_size * sample_rate / 1e3))
    freqs, times, spec = signal.spectrogram(audio,
                                             fs=sample_rate,
                                             window='hann',
                                             nperseg=nperseg,
                                             noverlap=noverlap,
                                             detrend=False)
    return freqs, times, np.log(spec.T.astype(np.float32) + eps)
```

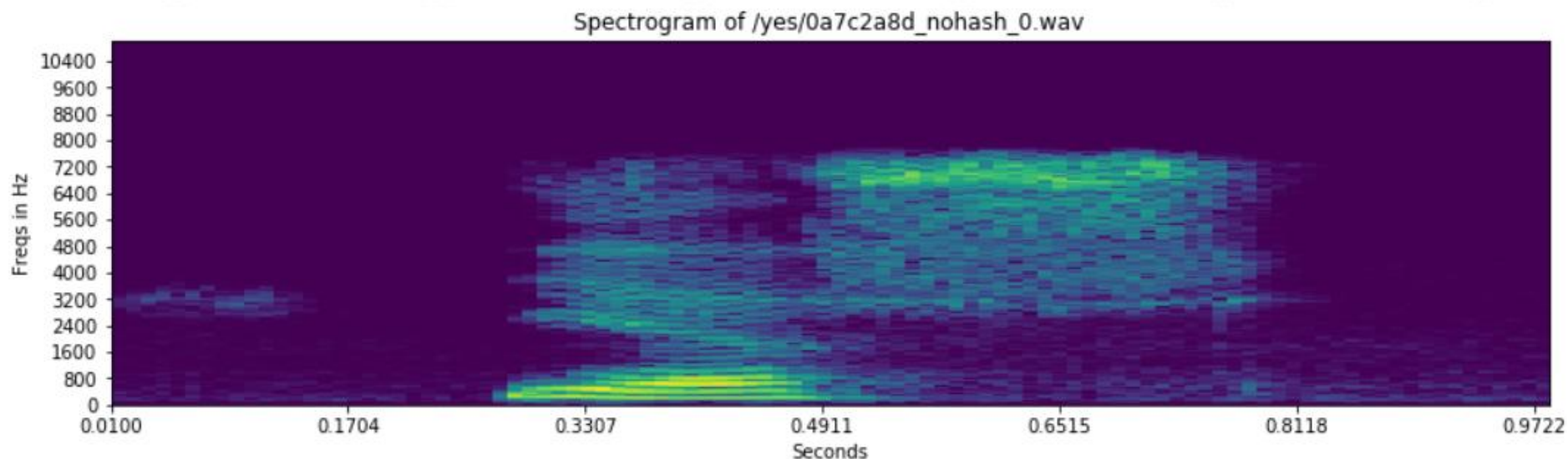
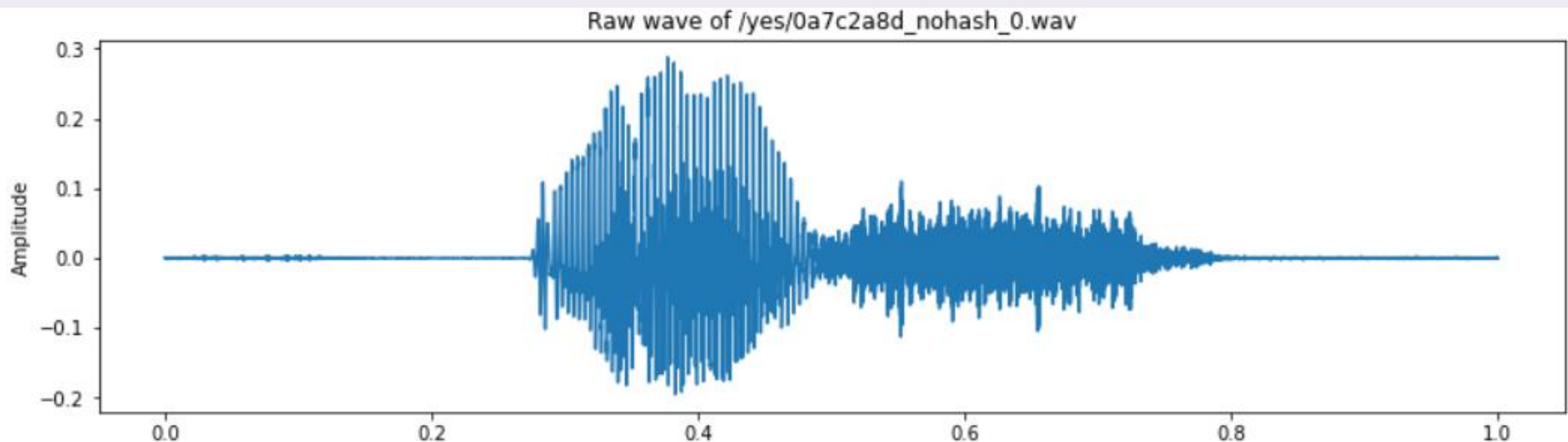
```
freqs, times, spectrogram = log_spectrogram(samples, sample_rate)

fig = plt.figure(figsize=(14, 8))
ax1 = fig.add_subplot(211)
ax1.set_title('Raw wave of ' + filename)
ax1.set_ylabel('Amplitude')
ax1.plot(np.linspace(0, sample_rate/len(samples), sample_rate), samples)

ax2 = fig.add_subplot(212)
ax2.imshow(spectrogram.T, aspect='auto', origin='lower',
           extent=[times.min(), times.max(), freqs.min(), freqs.max()])
ax2.set_yticks(freqs[::16])
ax2.set_xticks(times[::16])
ax2.set_title('Spectrogram of ' + filename)
ax2.set_ylabel('Freqs in Hz')
ax2.set_xlabel('Seconds')

plt.show()
```

1. Visualization



1. Visualization

check!

- If we use spectrogram as an input features for NN, we have to remember to normalize features.

```
# normalize  
mean = np.mean(spectrogram, axis=0)  
std = np.std(spectrogram, axis=0)  
spectrogram = (spectrogram - mean) / std
```

1. Visualization

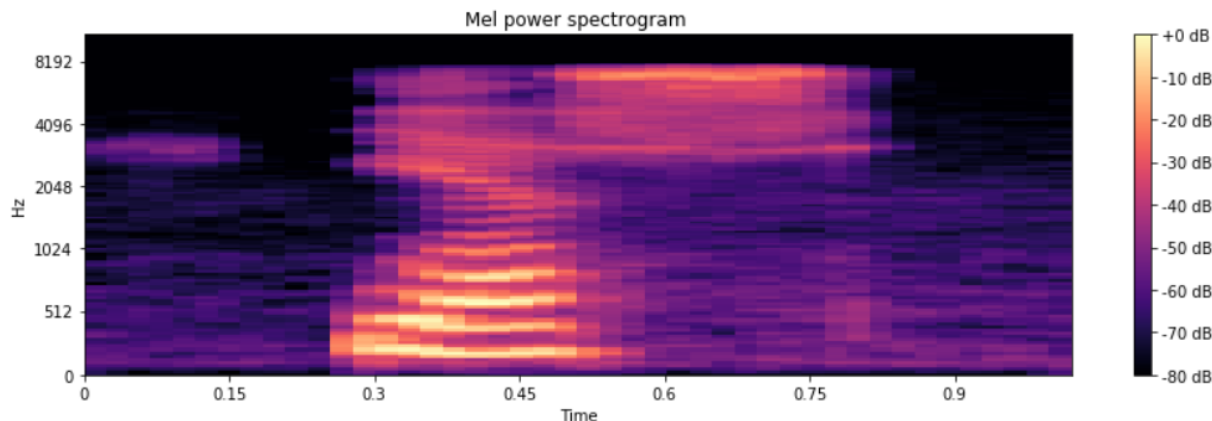
MFCC

- MFCC explained You can see, that it is well prepared to imitate human hearing properties. You can calculate Mel power spectrogram and MFCC using for example librosa python package.

```
# From this tutorial
# https://github.com/librosa/librosa/blob/master/examples/LibROSA%20demo.ipynb
S = librosa.feature.melspectrogram(samples, sr=sample_rate, n_mels=128)

# Convert to log scale (dB). We'll use the peak power (max) as reference.
log_S = librosa.power_to_db(S, ref=np.max)

plt.figure(figsize=(12, 4))
librosa.display.specshow(log_S, sr=sample_rate, x_axis='time', y_axis='mel')
plt.title('Mel power spectrogram ')
plt.colorbar(format='%+02.0f dB')
plt.tight_layout()
```



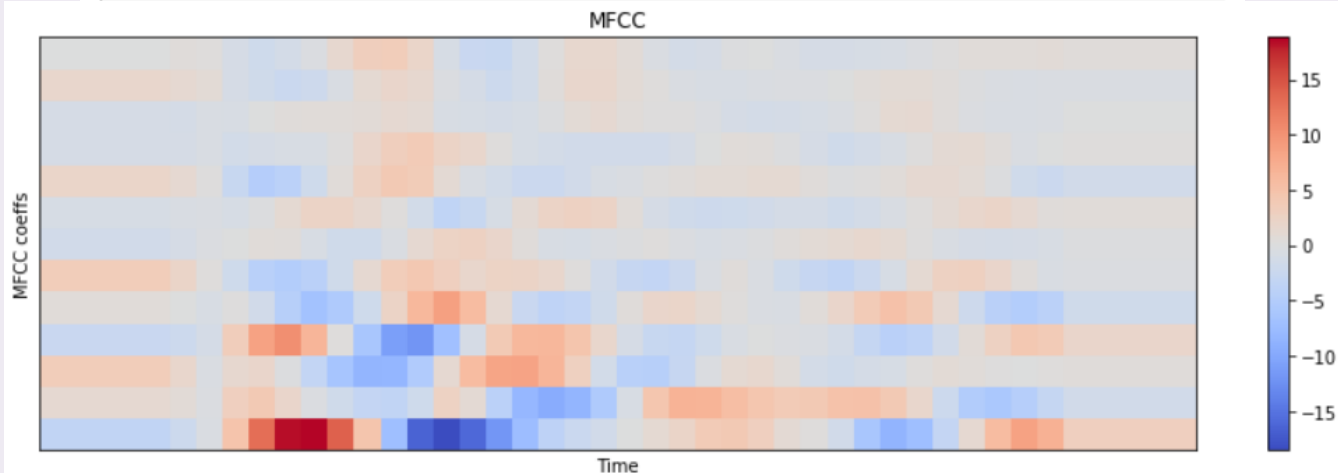
1. Visualization

MFCC : 소리의 특징을 추출하는 기법
입력된 소리 전체를 대상으로 하는 것이 아니라, 일정 구간(Short time)씩 나누어, 이 구간에 대한 **스펙트럼을 분석**

```
mfcc = librosa.feature.mfcc(S=log_S, n_mfcc=13)

# Let's pad on the first and second deltas while we're at it
delta2_mfcc = librosa.feature.delta(mfcc, order=2)

plt.figure(figsize=(12, 4))
librosa.display.specshow(delta2_mfcc)
plt.ylabel('MFCC coeffs')
plt.xlabel('Time')
plt.title('MFCC')
plt.colorbar()
plt.tight_layout()
```

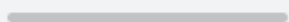



1. Visualization

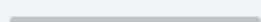

Silence removal

- I consider that some VAD (Voice Activity Detection) will be really useful here. Although the words are short, there is a lot of silence in them. A decent VAD can **reduce training size** a lot, accelerating training speed significantly. Let's cut a bit of the file from the beginning and from the end. and listen to it again (based on a plot above, we take from 4000 to 13000):

```
# Silence removal  
ipd.Audio(samples, rate = sample_rate)
```

▶ 0:00 / 0:00  

```
# after remove  
samples_cut = samples[4000:13000]  
ipd.Audio(samples_cut, rate=sample_rate)
```

▶ 0:00 / 0:00  

1. Visualization

check!

- It is impossible to cut all the files manually and do this basing on the simple plot. But you can use for example webrtcvad package to have a good VAD.

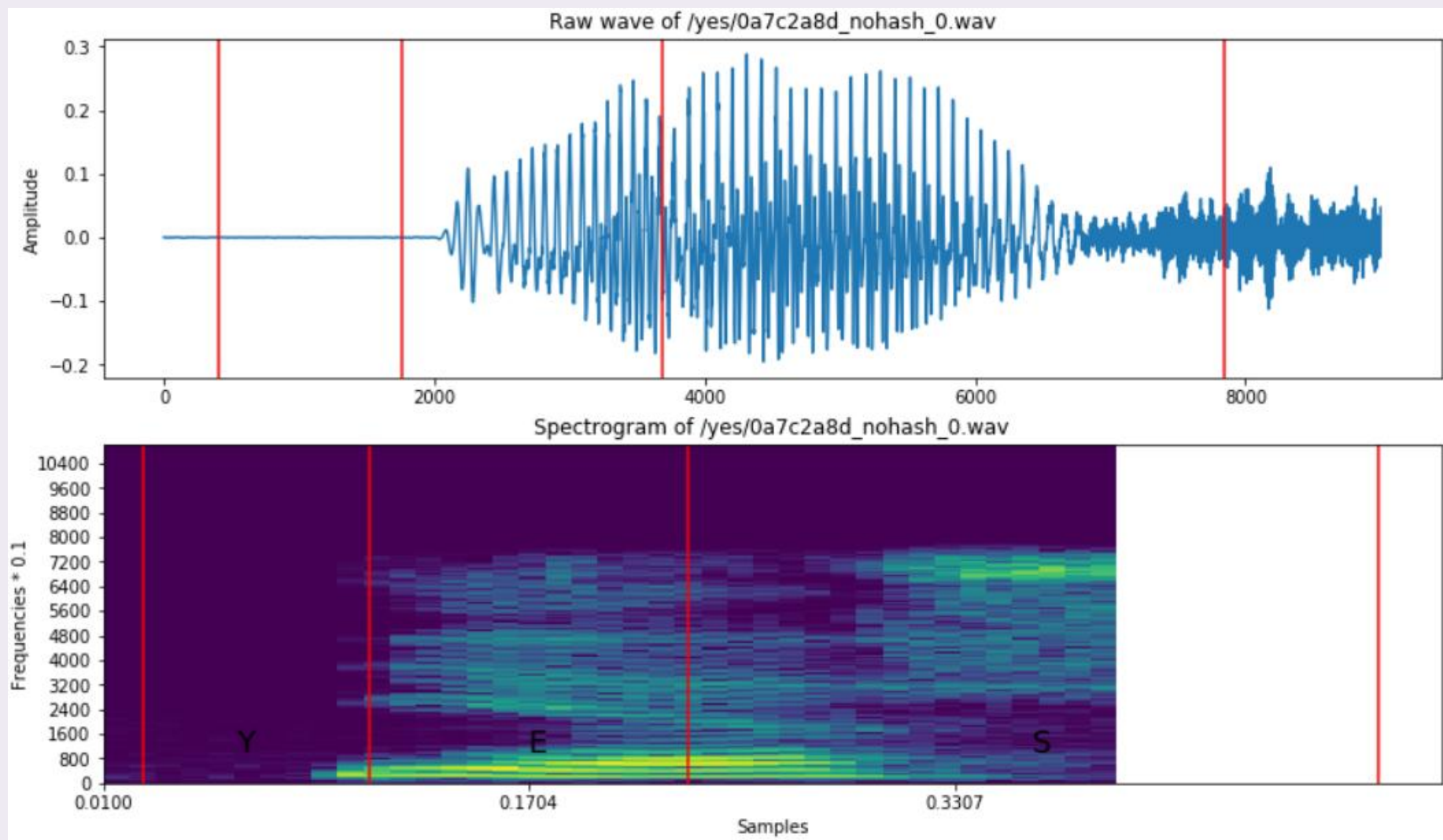
```
# plot it again, together with guessed alignment of 'y' 'e' 's' graphemes
freqs, times, spectrogram_cut = log_specgram(samples_cut, sample_rate)

fig = plt.figure(figsize=(14, 8))
ax1 = fig.add_subplot(211)
ax1.set_title('Raw wave of ' + filename)
ax1.set_ylabel('Amplitude')
ax1.plot(samples_cut)

ax2 = fig.add_subplot(212)
ax2.set_title('Spectrogram of ' + filename)
ax2.set_ylabel('Frequencies * 0.1')
ax2.set_xlabel('Samples')
ax2.imshow(spectrogram_cut.T, aspect='auto', origin='lower',
           extent=[times.min(), times.max(), freqs.min(), freqs.max()])
ax2.set_yticks(freqs[::16])
ax2.set_xticks(times[::16])
ax2.text(0.06, 1000, 'Y', fontsize=18)
ax2.text(0.17, 1000, 'E', fontsize=18)
ax2.text(0.36, 1000, 'S', fontsize=18)

xcoords = [0.025, 0.11, 0.23, 0.49]
for xc in xcoords:
    ax1.axvline(x=xc*16000, c='r')
    ax2.axvline(x=xc, c='r')
```

1. Visualization



1. Visualization

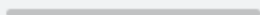
Resampling - dimensionality reduction

- Another way to reduce the dimensionality of our data is to **resample recordings**.
we could resample our dataset to 8k. We will discard some information that shouldn't be important, and we'll reduce size of the data.
experiments can be done much faster with smaller training size.
We'll need to calculate FFT (Fast Fourier Transform).

```
def custom_fft(y, fs):  
    T = 1.0 / fs  
    N = y.shape[0]  
    yf = fft(y)  
    xf = np.linspace(0.0, 1.0/(2.0*T), N//2)  
    vals = 2.0/N * np.abs(yf[0:N//2])  
    # FFT is simmetrical, so we take just the first half  
    # FFT is also complex, so we take just the real part (abs)  
    return xf, vals
```

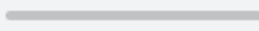

```
filename = '/happy/0b09edd3_nohash_0.wav'  
new_sample_rate = 8000  
  
sample_rate, samples = wavfile.read(str(train_audio_path) + filename)  
resampled = signal.resample(samples, int(new_sample_rate/sample_rate * samples.shape[0]))
```

```
ipd.Audio(samples, rate=sample_rate)
```

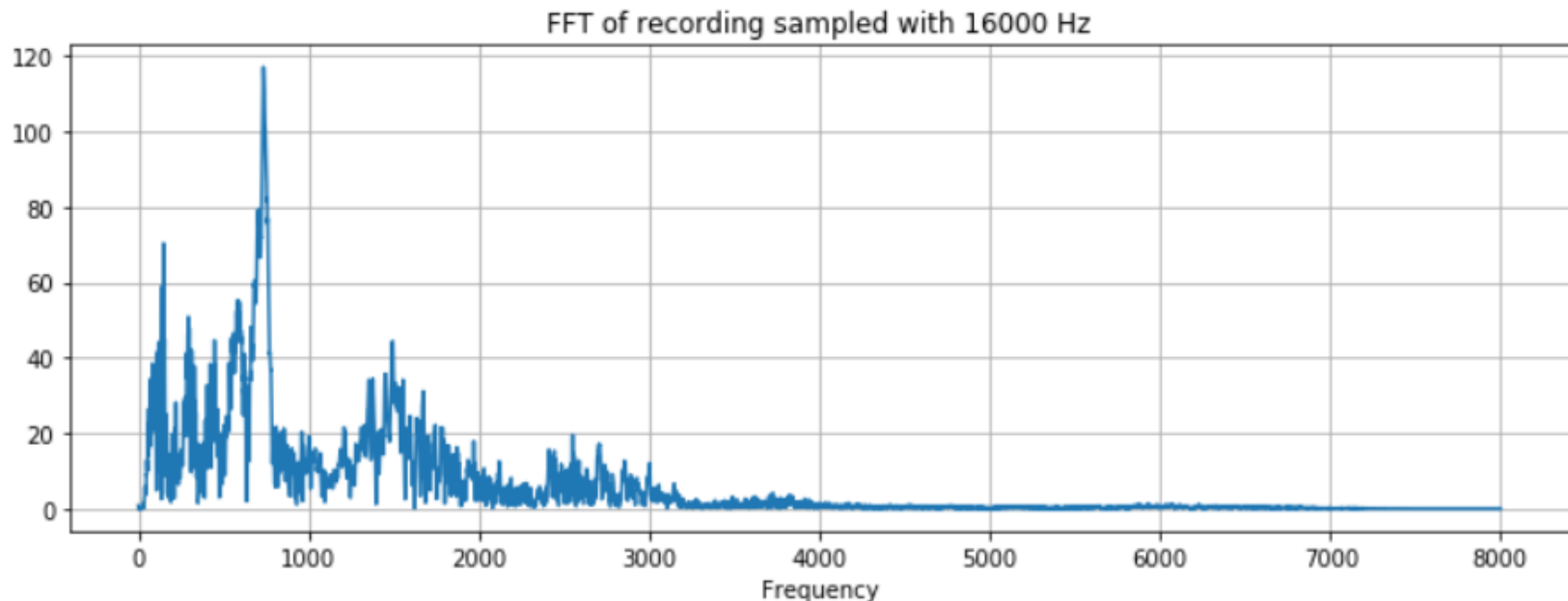
▶ 0:00 / 0:00  🔊

1. Visualization

```
ipd.Audio(resampled, rate=new_sample_rate) # Almost no difference!
```

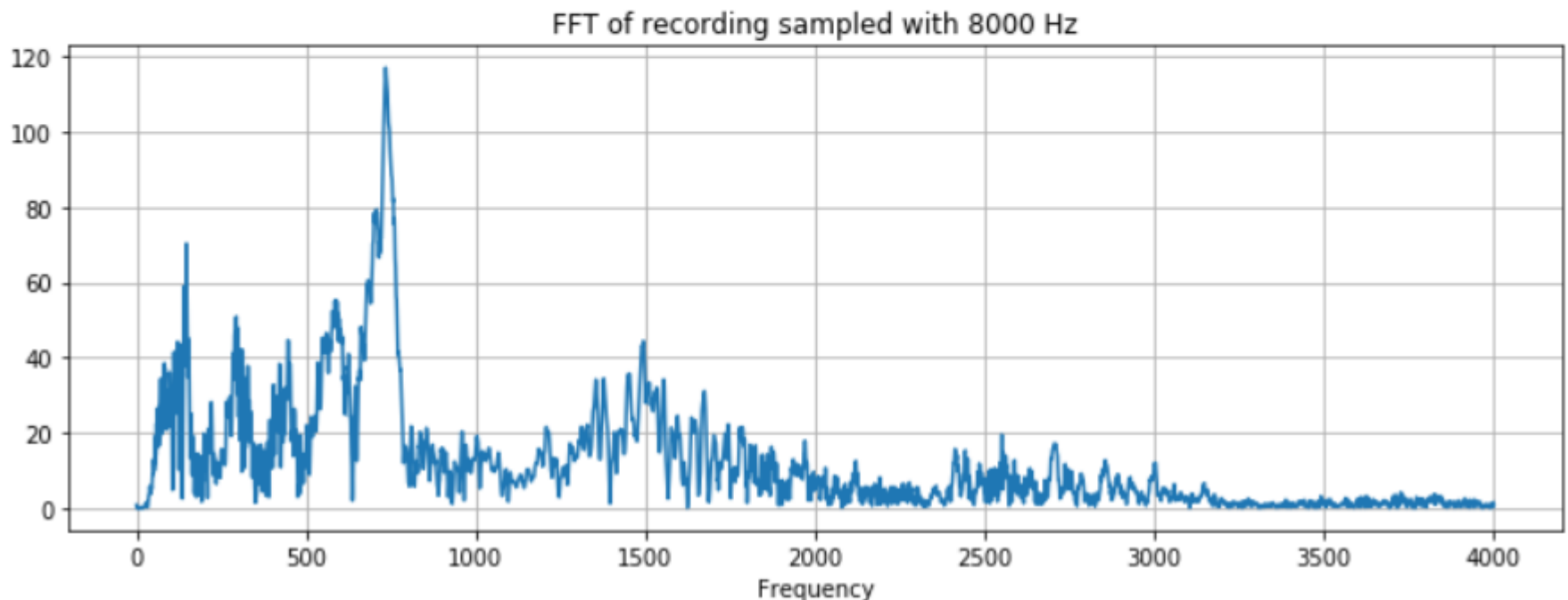
▶ 0:00 / 0:00  

```
xf, vals = custom_fft(samples, sample_rate)
plt.figure(figsize=(12, 4))
plt.title('FFT of recording sampled with ' + str(sample_rate) + ' Hz')
plt.plot(xf, vals)
plt.xlabel('Frequency')
plt.grid()
plt.show()
```



1. Visualization

```
xf, vals = custom_fft(resampled, new_sample_rate)
plt.figure(figsize=(12, 4))
plt.title('FFT of recording sampled with ' + str(new_sample_rate) + ' Hz')
plt.plot(xf, vals)
plt.xlabel('Frequency')
plt.grid()
plt.show()
```



2. Dataset investigation

```
# Number of records
dirs = [f for f in os.listdir(train_audio_path) if isdir(join(train_audio_path, f))]
dirs.sort()
print('Number of labels: ' + str(len(dirs)))
```

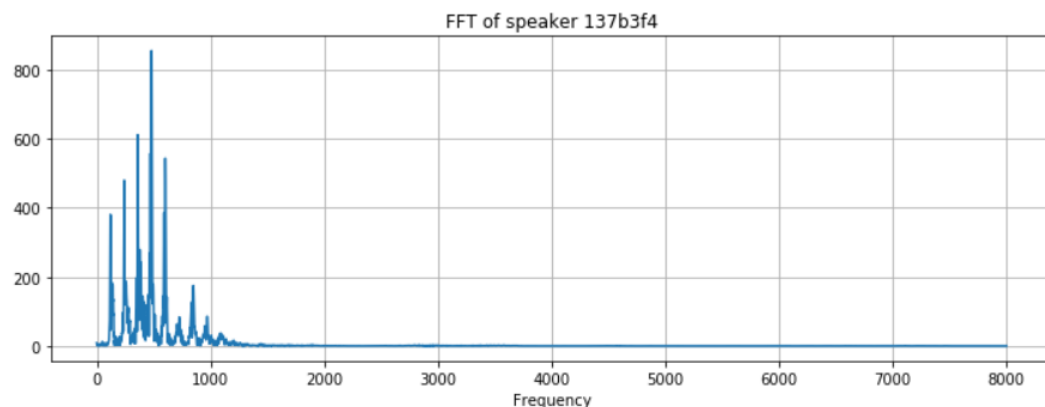
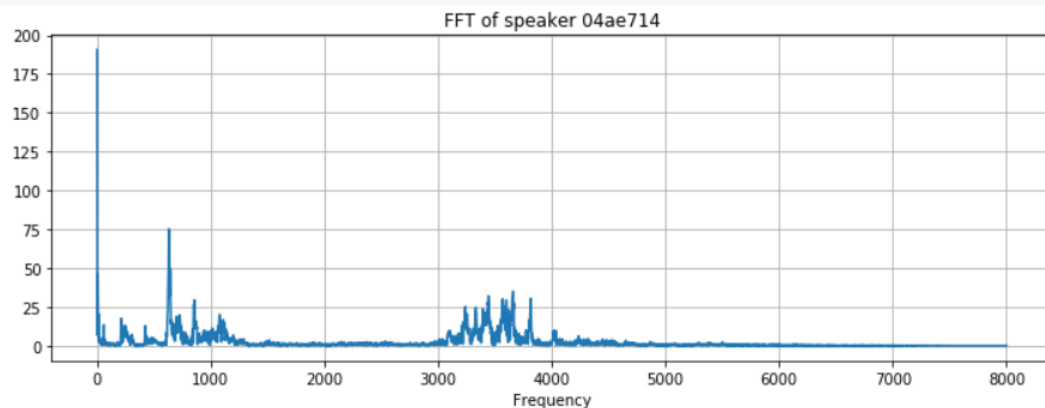
Number of labels: 31

Deeper into recordings

- Recordings come from very different sources. As far as I can tell, some of them can come from mobile GSM channel. Nevertheless, it is extremely important to split the dataset in a way that one speaker doesn't occur in both train and test sets. Just take a look and listen to this two examples:

2. Dataset investigation

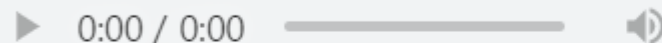
```
filenames = ['on/004ae714_nohash_0.wav', 'on/0137b3f4_nohash_0.wav']
for filename in filenames:
    sample_rate, samples = wavfile.read(str(train_audio_path) + filename)
    xf, vals = custom_fft(samples, sample_rate)
    plt.figure(figsize=(12, 4))
    plt.title('FFT of speaker ' + filename[4:11])
    plt.plot(xf, vals)
    plt.xlabel('Frequency')
    plt.grid()
    plt.show()
```



2. Dataset investigation

```
print('Speaker ' + filenames[0][4:11])  
ipd.Audio(join(train_audio_path, filenames[0]))
```

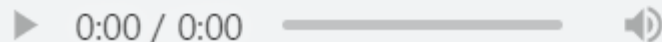
Speaker 04ae714



```
print('Speaker ' + filenames[1][4:11])  
ipd.Audio(join(train_audio_path, filenames[1]))
```

There are also recordings with some weird silence

Speaker 137b3f4

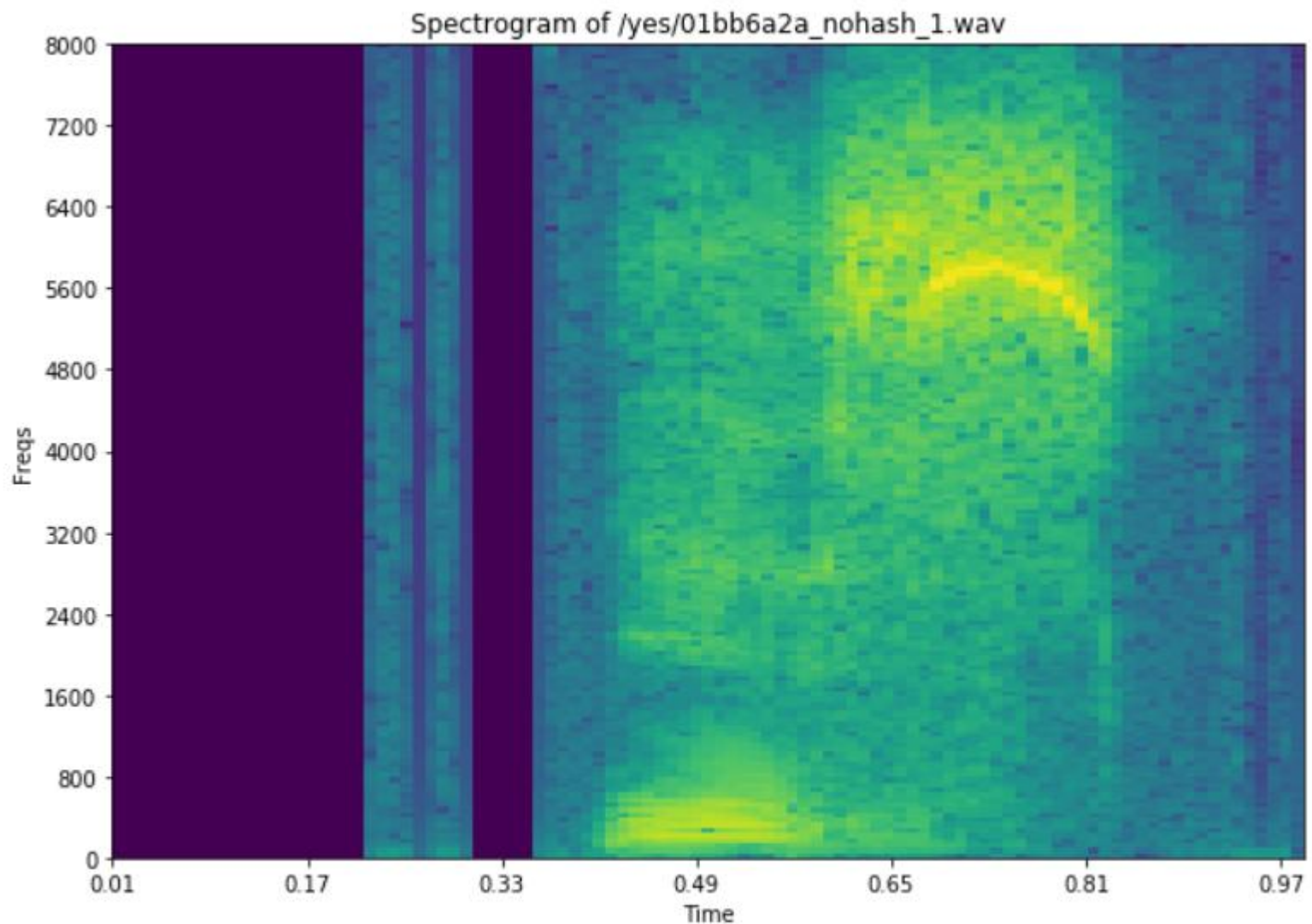


2. Dataset investigation

```
filename = '/yes/01bb6a2a_nohash_1.wav'
sample_rate, samples = wavfile.read(str(train_audio_path) + filename)
freqs, times, spectrogram = log_spectrogram(samples, sample_rate)

plt.figure(figsize=(10, 7))
plt.title('Spectrogram of ' + filename)
plt.ylabel('Freqs')
plt.xlabel('Time')
plt.imshow(spectrogram.T, aspect='auto', origin='lower',
           extent=[times.min(), times.max(), freqs.min(), freqs.max()])
plt.yticks(freqs[::16])
plt.xticks(times[::16])
plt.show()
```

2. Dataset investigation



- It means, that we have to prevent overfitting to the very specific acoustical environments.

2. Dataset investigation

```
# Recordings length
num_of_shorter = 0
for direct in dirs:
    waves = [f for f in os.listdir(join(train_audio_path, direct)) if f.endswith('.wav')]
    for wav in waves:
        sample_rate, samples = wavfile.read(train_audio_path + direct + '/' + wav)
        if samples.shape[0] < sample_rate:
            num_of_shorter += 1
print('Number of recordings shorter than 1 second: ' + str(num_of_shorter))
```

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:6: WavFileWarning:

Chunk (non-data) not understood, skipping it.

Number of recordings shorter than 1 second: 6469

- That's suprising, and there is a lot of them. We can pad them with zeros.

2. Dataset investigation

```
# Mean spectrograms and FFT
to_keep = 'yes no up down left right on off stop go'.split()
dirs = [d for d in dirs if d in to_keep]

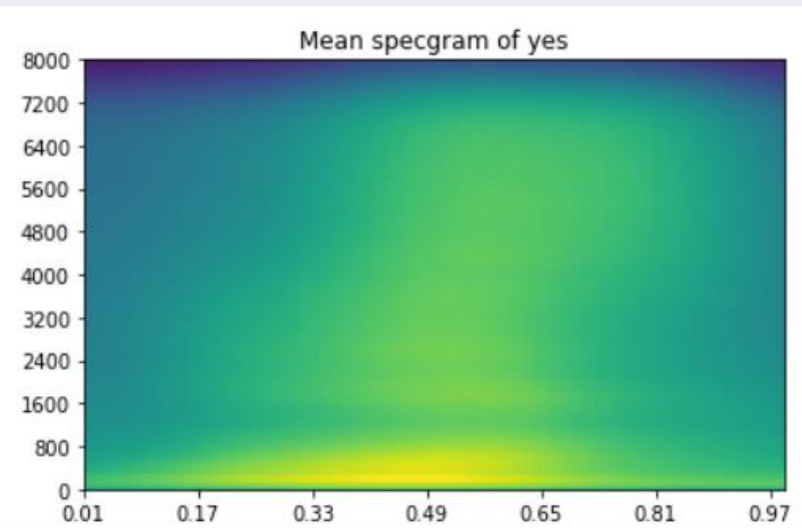
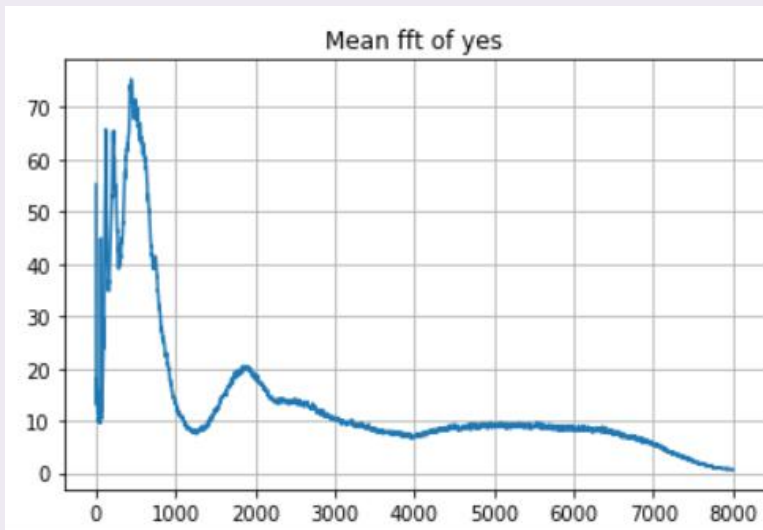
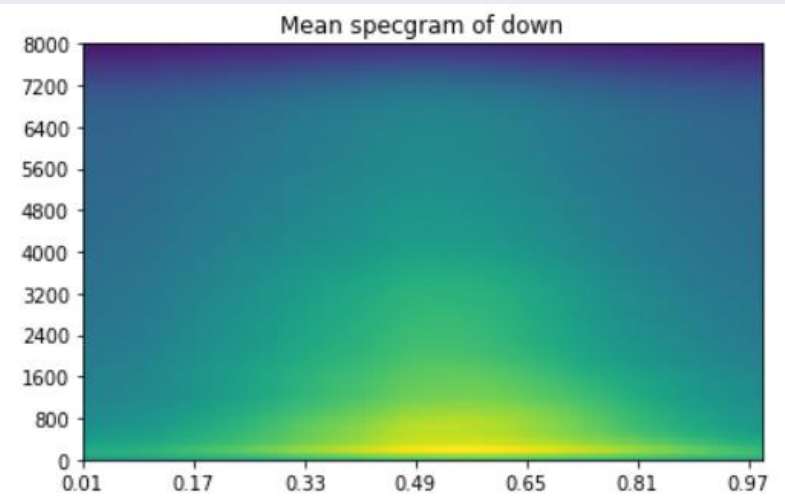
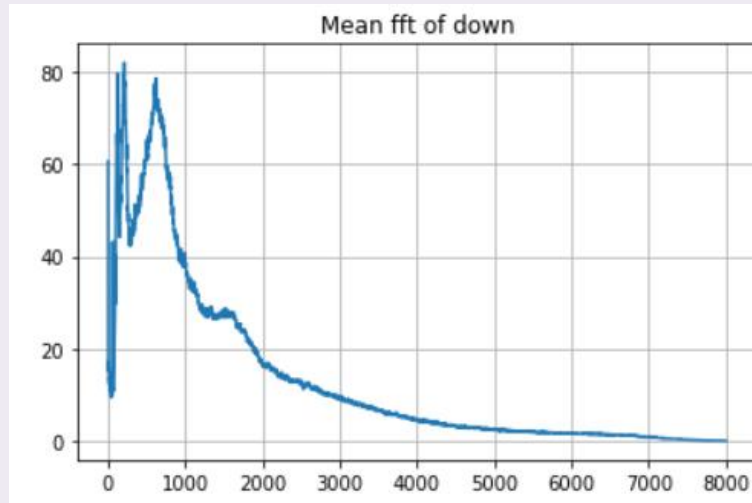
print(dirs)

for direct in dirs:
    vals_all = []
    spec_all = []

    waves = [f for f in os.listdir(join(train_audio_path, direct)) if f.endswith('.wav')]
    for wav in waves:
        sample_rate, samples = wavfile.read(train_audio_path + direct + '/' + wav)
        if samples.shape[0] != 16000:
            continue
        xf, vals = custom_fft(samples, 16000)
        vals_all.append(vals)
        freqs, times, spec = log_spectrogram(samples, 16000)
        spec_all.append(spec)

    plt.figure(figsize=(14, 4))
    plt.subplot(121)
    plt.title('Mean fft of ' + direct)
    plt.plot(np.mean(np.array(vals_all), axis=0))
    plt.grid()
    plt.subplot(122)
    plt.title('Mean specgram of ' + direct)
    plt.imshow(np.mean(np.array(spec_all), axis=0).T, aspect='auto', origin='lower',
               extent=[times.min(), times.max(), freqs.min(), freqs.max()])
    plt.yticks(freqs[::16])
    plt.xticks(times[::16])
    plt.show()
```

2. Dataset investigation



2. Dataset investigation

```
# Frequency components across the words
def violinplot_frequency(dirs, freq_ind):
    """ Plot violinplots for given words (waves in dirs) and frequency freq_ind
    from all frequencies freqs."""

    spec_all = [] # Contain spectrograms
    ind = 0
    for direct in dirs:
        spec_all.append([])

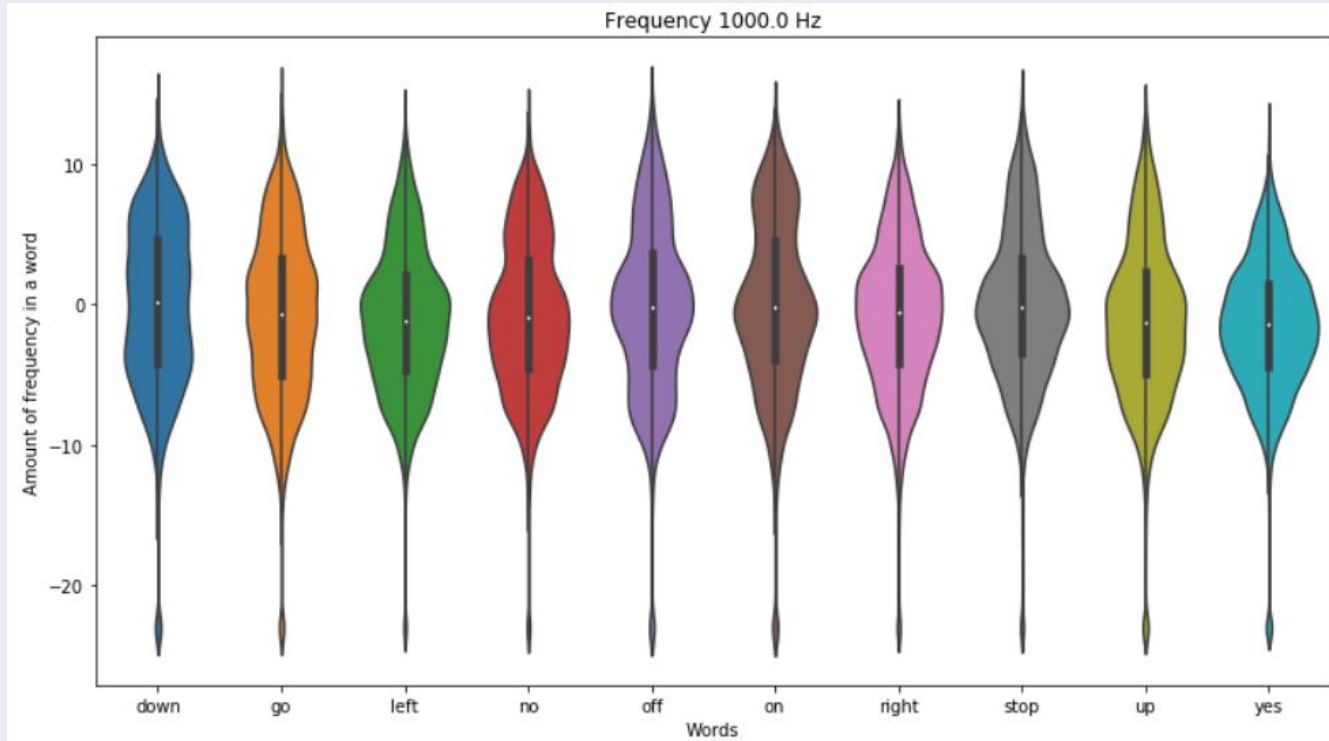
        waves = [f for f in os.listdir(join(train_audio_path, direct)) if
                  f.endswith('.wav')]
        for wav in waves[:100]:
            sample_rate, samples = wavfile.read(
                train_audio_path + direct + '/' + wav)
            freqs, times, spec = log_specgram(samples, sample_rate)
            spec_all[ind].extend(spec[:, freq_ind])
            ind += 1

    # Different lengths = different num of frames. Make number equal
    minimum = min([len(spec) for spec in spec_all])
    spec_all = np.array([spec[:minimum] for spec in spec_all])

    plt.figure(figsize=(13,7))
    plt.title('Frequency ' + str(freqs[freq_ind]) + ' Hz')
    plt.ylabel('Amount of frequency in a word')
    plt.xlabel('Words')
    sns.violinplot(data=pd.DataFrame(spec_all.T, columns=dirs))
    plt.show()
```

```
violinplot_frequency(dirs, 20)
```

2. Dataset investigation



2. Dataset investigation

Anomaly detection

- We should check if there are any recordings that somehow stand out from the rest.
We can lower the dimensionality of the dataset and interactively check for any anomaly. -> **PCA**

```
fft_all = []
names = []
for direct in dirs:
    waves = [f for f in os.listdir(join(train_audio_path, direct)) if f.endswith('.wav')]
    for wav in waves:
        sample_rate, samples = wavfile.read(train_audio_path + direct + '/' + wav)
        if samples.shape[0] != sample_rate:
            samples = np.append(samples, np.zeros((sample_rate - samples.shape[0], )))
        x, val = custom_fft(samples, sample_rate)
        fft_all.append(val)
        names.append(direct + '/' + wav)

fft_all = np.array(fft_all)

# Normalization
fft_all = (fft_all - np.mean(fft_all, axis=0)) / np.std(fft_all, axis=0)

# Dim reduction
pca = PCA(n_components=3)
fft_all = pca.fit_transform(fft_all)

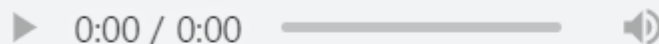
def interactive_3d_plot(data, names):
    scatt = go.Scatter3d(x=data[:, 0], y=data[:, 1], z=data[:, 2], mode='markers', text=names)
    data = go.Data([scatt])
    layout = go.Layout(title="Anomaly detection")
    figure = go.Figure(data=data, layout=layout)
    py.iplot(figure)

interactive_3d_plot(fft_all, names)
```

2. Dataset investigation

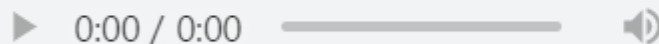
```
print('Recording go/0487ba9b_nohash_0.wav')  
ipd.Audio(join(train_audio_path, 'go/0487ba9b_nohash_0.wav'))
```

Recording go/0487ba9b_nohash_0.wav



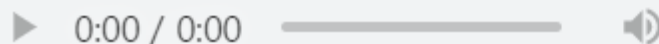
```
print('Recording yes/e4b02540_nohash_0.wav')  
ipd.Audio(join(train_audio_path, 'yes/e4b02540_nohash_0.wav'))
```

Recording yes/e4b02540_nohash_0.wav



```
print('Recording seven/e4b02540_nohash_0.wav')  
ipd.Audio(join(train_audio_path, 'seven/b1114e4f_nohash_0.wav'))
```

Recording seven/e4b02540_nohash_0.wav



3. Light-Weight CNN

```
input_shape = (99, 81, 1)
nclass = 12
inp = Input(shape=input_shape)
norm_inp = BatchNormalization()(inp)
img_1 = Convolution2D(8, kernel_size=2, activation=activations.relu)(norm_inp)
img_1 = Convolution2D(8, kernel_size=2, activation=activations.relu)(img_1)
img_1 = MaxPooling2D(pool_size=(2, 2))(img_1)
img_1 = Dropout(rate=0.2)(img_1)
img_1 = Convolution2D(16, kernel_size=3, activation=activations.relu)(img_1)
img_1 = Convolution2D(16, kernel_size=3, activation=activations.relu)(img_1)
img_1 = MaxPooling2D(pool_size=(2, 2))(img_1)
img_1 = Dropout(rate=0.2)(img_1)
img_1 = Convolution2D(32, kernel_size=3, activation=activations.relu)(img_1)
img_1 = MaxPooling2D(pool_size=(2, 2))(img_1)
img_1 = Dropout(rate=0.2)(img_1)
img_1 = Flatten()(img_1)

dense_1 = BatchNormalization()(Dense(128, activation=activations.relu)(img_1))
dense_1 = BatchNormalization()(Dense(128, activation=activations.relu)(dense_1))
dense_1 = Dense(nclass, activation=activations.softmax)(dense_1)

model = models.Model(inputs=inp, outputs=dense_1)
opt = optimizers.Adam()

model.compile(optimizer=opt, loss=losses.binary_crossentropy)
model.summary()

x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train, test_size=0.1, random_state=2017)
model.fit(x_train, y_train, batch_size=16, validation_data=(x_valid, y_valid), epochs=3, shuffle=True, verbose=2)

model.save(os.path.join(model_path, 'cnn.model'))
```

4. WavCeption V1: just a 1-D Inception approach

- The WavCeption V1 network seems to produce impressive results compared to a regular convolutional neural network, but in this competition it seems that there is a hard-work on the pre-processing and unknown tracks management. It is based on the Google's inception network, the same idea. By running the model for 12h without struggling too much I achieved 0.76 in the leaderboard (with 0.84 in local test).

```
# Noise generation functions
def ms(x):
    """Mean value of signal `x` squared.
    :param x: Dynamic quantity.
    :returns: Mean squared of `x`.
    """
    return (np.abs(x)**2.0).mean()

def normalize(y, x=None):
    """normalize power in y to a (standard normal) white noise signal.
    Optionally normalize to power in signal `x`.
    #The mean power of a Gaussian with  $\mu=0$  and  $\sigma=1$  is 1.
    """
    #return y * np.sqrt( (np.abs(x)**2.0).mean() / (np.abs(y)**2.0).mean() )
    if x is not None:
        x = ms(x)
    else:
        x = 1.0
    return y * np.sqrt( x / ms(y) )
#return y * np.sqrt( 1.0 / (np.abs(y)**2.0).mean() )
```

4. WavCeption V1: just a 1-D Inception approach

```
def white_noise(N, state=None):
    state = np.random.RandomState() if state is None else state
    return state.randn(N)

def pink_noise(N, state=None):

    state = np.random.RandomState() if state is None else state
    uneven = N%2
    X = state.randn(N//2+1+uneven) + 1j * state.randn(N//2+1+uneven)
    S = np.sqrt(np.arange(len(X))+1.) # +1 to avoid divide by zero
    y = (irfft(X/S)).real
    if uneven:
        y = y[:-1]
    return normalize(y)

def blue_noise(N, state=None):
    """
    Blue noise.

    :param N: Amount of samples.
    :param state: State of PRNG.
    :type state: :class:`np.random.RandomState`

    Power increases with 6 dB per octave.
    Power density increases with 3 dB per octave.

    """
    state = np.random.RandomState() if state is None else state
    uneven = N%2
    X = state.randn(N//2+1+uneven) + 1j * state.randn(N//2+1+uneven)
    S = np.sqrt(np.arange(len(X))) # Filter
    y = (irfft(X*S)).real
    if uneven:
        y = y[:-1]
    return normalize(y)
```

4. WavCeption V1: just a 1-D Inception approach

```
def brown_noise(N, state=None):
    """
    Violet noise.

    :param N: Amount of samples.
    :param state: State of PRNG.
    :type state: :class:`np.random.RandomState`

    Power decreases with -3 dB per octave.
    Power density decreases with 6 dB per octave.
    """
    state = np.random.RandomState() if state is None else state
    uneven = N%2
    X = state.randn(N//2+1+uneven) + 1j * state.randn(N//2+1+uneven)
    S = (np.arange(len(X))+1)# Filter
    y = (irfft(X/S)).real
    if uneven:
        y = y[:-1]
    return normalize(y)
```

```
def violet_noise(N, state=None):
    """
    Violet noise. Power increases with 6 dB per octave.

    :param N: Amount of samples.
    :param state: State of PRNG.
    :type state: :class:`np.random.RandomState`

    Power increases with +9 dB per octave.
    Power density increases with +6 dB per octave.

    """
    state = np.random.RandomState() if state is None else state
    uneven = N%2
    X = state.randn(N//2+1+uneven) + 1j * state.randn(N//2+1+uneven)
    S = (np.arange(len(X)))# Filter
    y = (irfft(X*S)).real
    if uneven:
        y = y[:-1]
    return normalize(y)
```

4. WavCeption V1: just a 1-D Inception approach

Architecture building blocks

- Inception-1D (a.k.a wavception) is a module I designed some weeks ago for this problem. It substantially enhances the performance of a regular convolutional neural net.

[illegible]

4. WavCeption V1: just a 1-D Inception approach

```
def inception_1d(x, is_train, depth, norm_function, activ_function, name):
    """
    Inception 1D module implementation.
    :param x: input to the current module (4D tensor with channels-last)
    :param is_train: it is intended to be a boolean placeholder for controlling the BatchNormalization behavior (0D tensor)
    :param depth: linearly controls the depth of the network (int)
    :param norm_function: normalization class (same format as the BatchNorm class above)
    :param activ_function: tensorflow activation function (e.g. tf.nn.relu)
    :param name: name of the variable scope (str)
    """
    with tf.variable_scope(name):
        x_norm = norm_function(name="norm_input")(x, train=is_train)

        # Branch 1: 64 x conv 1x1
        branch_conv_1_1 = tf.layers.conv1d(inputs=x_norm, filters=16*depth, kernel_size=1,
                                           kernel_initializer=tf.contrib.layers.xavier_initializer(),
                                           padding="same", name="conv_1_1")
        branch_conv_1_1 = norm_function(name="norm_conv_1_1")(branch_conv_1_1, train=is_train)
        branch_conv_1_1 = activ_function(branch_conv_1_1, "activation_1_1")

        # Branch 2: 128 x conv 3x3
        branch_conv_3_3 = tf.layers.conv1d(inputs=x_norm, filters=16, kernel_size=1,
                                           kernel_initializer=tf.contrib.layers.xavier_initializer(),
                                           padding="same", name="conv_3_3_1")
        branch_conv_3_3 = norm_function(name="norm_conv_3_3_1")(branch_conv_3_3, train=is_train)
        branch_conv_3_3 = activ_function(branch_conv_3_3, "activation_3_3_1")

        branch_conv_3_3 = tf.layers.conv1d(inputs=branch_conv_3_3, filters=32*depth, kernel_size=3,
                                           kernel_initializer=tf.contrib.layers.xavier_initializer(),
                                           padding="same", name="conv_3_3_2")
        branch_conv_3_3 = norm_function(name="norm_conv_3_3_2")(branch_conv_3_3, train=is_train)
        branch_conv_3_3 = activ_function(branch_conv_3_3, "activation_3_3_2")
```


4. WavCeption V1: just a 1-D Inception approach

```
# Load and prepare Data
# Synthetic and provided noise addition
filepaths_noise = glob.glob(os.path.join(get_train_audio_path(), "_background_noise_", "*.wav"))

noise = np.concatenate(list(map(lambda x: read_wav(x, False)[0], filepaths_noise)))
noise = np.concatenate([noise, noise[::-1]])
synthetic_noise = np.concatenate([white_noise(N=16000*30, state=np.random.RandomState(655321)),
                                   blue_noise(N=16000*30, state=np.random.RandomState(655321)),
                                   pink_noise(N=16000*30, state=np.random.RandomState(655321)),
                                   brown_noise(N=16000*30, state=np.random.RandomState(655321)),
                                   violet_noise(N=16000*30, state=np.random.RandomState(655321)),
                                   np.zeros(16000*60)])

synthetic_noise /= np.max(np.abs(synthetic_noise))
synthetic_noise = np.concatenate([synthetic_noise, (synthetic_noise+synthetic_noise[::-1])/2])
all_noise = np.concatenate([noise, synthetic_noise])
```

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:10: WavFileWarning:

Chunk (non-data) not understood, skipping it.

4. WavCeption V1: just a 1-D Inception approach

```
# Classes processing
cardinal_classes = list(set(map(lambda fp:os.path.split(os.path.split(fp)[0])[1], filepaths)))
le_classes = LabelEncoder().fit(cardinal_classes)
Counter(map(lambda fp:os.path.split(os.path.split(fp)[0])[1], filepaths))
```

```
Counter({'no': 2375,
        'yes': 2377,
        'stop': 2380,
        'nine': 2364,
        'left': 2353,
        'dog': 1746,
        'wow': 1745,
        'up': 2375,
        'one': 2370,
        'six': 2369,
        'zero': 2376,
        'two': 2373,
        'sheila': 1734,
        'tree': 1733,
        'silence': 8000,
        'four': 2372,
```

4. WavCeption V1: just a 1-D Inception approach

```
def define_core_model(self):
    with tf.variable_scope("Core_Model"):
        x = inception_1d(x=self.placeholders.wav_in, is_train=self.placeholders.is_train,
                        norm_function=BatchNorm, activ_function=tf.nn.relu, depth=1,
                        name="Inception_1_1")
        x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                        activ_function=tf.nn.relu, depth=1, name="Inception_1_2")
        x = tf.layers.max_pooling1d(x, 2, 2, name="maxpool_1")
        x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                        activ_function=tf.nn.relu, depth=1, name="Inception_2_1")
        x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                        activ_function=tf.nn.relu, depth=1, name="Inception_2_3")
        x = tf.layers.max_pooling1d(x, 2, 2, name="maxpool_2")
        x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                        activ_function=tf.nn.relu, depth=2, name="Inception_3_1")
        x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                        activ_function=tf.nn.relu, depth=2, name="Inception_3_2")
        x = tf.layers.max_pooling1d(x, 2, 2, name="maxpool_3")
        x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                        activ_function=tf.nn.relu, depth=2, name="Inception_4_1")
        x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                        activ_function=tf.nn.relu, depth=2, name="Inception_4_2")
        x = tf.layers.max_pooling1d(x, 2, 2, name="maxpool_4")
        x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                        activ_function=tf.nn.relu, depth=3, name="Inception_5_1")
        x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                        activ_function=tf.nn.relu, depth=3, name="Inception_5_2")
        x = tf.layers.max_pooling1d(x, 2, 2, name="maxpool_5")
        x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                        activ_function=tf.nn.relu, depth=3, name="Inception_6_1")
        x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                        activ_function=tf.nn.relu, depth=3, name="Inception_6_2")
        x = tf.layers.max_pooling1d(x, 2, 2, name="maxpool_6")
        x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                        activ_function=tf.nn.relu, depth=4, name="Inception_7_1")
        x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                        activ_function=tf.nn.relu, depth=4, name="Inception_7_2")
```

4. WavCeption V1: just a 1-D Inception approach

```
x = tf.layers.max_pooling1d(x, 2, 2, name="maxpool_7")
x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                 activ_function=tf.nn.relu, depth=4, name="Inception_8_1")
x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                 activ_function=tf.nn.relu, depth=4, name="Inception_8_2")
x = tf.layers.max_pooling1d(x, 2, 2, name="maxpool_8")
x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                 activ_function=tf.nn.relu, depth=4, name="Inception_9_1")
x = inception_1d(x=x, is_train=self.placeholders.is_train, norm_function=BatchNorm,
                 activ_function=tf.nn.relu, depth=4, name="Inception_9_2")
x = tf.layers.max_pooling1d(x, 2, 2, name="maxpool_9")
x = tf.contrib.layers.flatten(x)
x = tf.layers.dense(BatchNorm(name="bn_dense_1")(x, train=self.placeholders.is_train),
                    128, activation=tf.nn.relu, kernel_initializer=tf.contrib.layers.xavier_initializer(),
                    name="dense_1")
output = tf.layers.dense(BatchNorm(name="bn_dense_2")(x, train=self.placeholders.is_train),
                        self.class_cardinality, activation=None, kernel_initializer=tf.contrib.layers.xavier_initializer(),
                        name="output")
return({"output": output})
```

```
def define_losses(self):
    with tf.variable_scope("Losses"):
        softmax_ce = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=tf.squeeze(self.placeholders.target),
                                                                    logits=self.core_model.output,
                                                                    name="softmax")

        return({"softmax": softmax_ce})

def define_optimizers(self):
    with tf.variable_scope("Optimization"):
        op = self.optimizer.minimize(self.losses.softmax)
        return({"op": op})
```

Reference

[1] Speech representation and data exploration,
<https://www.kaggle.com/davids1992/speech-representation-and-data-exploration>

[2] Light-Weight CNN LB 0.74, <https://www.kaggle.com/alphasis/light-weight-cnn-lb-0-74>

[3] WavCeption V1: a 1-D Inception approach (LB 0.76),
<https://www.kaggle.com/ivallesp/wavception-v1-a-1-d-inception-approach-lb-0-76>