Spectrogram Implementation

Mohammad Ghorbani 9447023

Dependencies

- `pip install matplotlib`
- `pip install numpy`

Code

```
def overlapped_window(stream, window_size, overlap_rate):
    X = []
    while True:
        read_size = window_size if len(X) == 0 else window_size
        x = stream.read(read_size)
        if len(x) != read_size:
            raise StopIteration()
        X.extend(x)
        yield np.array(X)
        X = X[window_size//overlap_rate:]

input_file='maybe-next-time.wav'
window_size=1024
scale="log"

w = wavfile(input_file)

if w.get_param('nchannels') > 2:
    raise Exception('error: only mono and stereo tracks are supp

if w.get_param('nframes') < window_size:
    raise Exception('error: audio file is shorter than configure

# Hann window function coefficients.
hamming = 0.54 - 0.46 * np.cos(2.0 * np.pi * (np.arange(window_s

# Hann window must have 4x overlap for good results.
```

```
overlap = 4

# Y will hold the DFT of each window.
Y = []
# Process each window of audio.
for x in overlapped_window(monowrapper(w), window_size, overlap)
    y = np.fft.rfft(x * hamming)[:window_size//2]
    Y.append(y)

# Normalize data and convert to dB.
Y = np.column_stack(Y)
Y = np.absolute(Y) * 2.0 / np.sum(hamming)
Y = Y / np.power(2.0, (8 * w.get_param('sampwidth') - 1))
Y = (20.0 * np.log10(Y)).clip(-120)

# Time domain: We have Y.shape[1] windows, so convert to seconds
# by window size, dividing by sample rate, and dividing by the o
t = np.arange(0, Y.shape[1], dtype=np.float) * window_size / w.g

# Frequency domain: There are window_size/2 frequencies represen
# by dividing by window size and multiplying by sample frequency
f = np.arange(0, window_size / 2, dtype=np.float) * w.get_param(

# Plot the spectrogram.
ax = plt.subplot(111)
plt.pcolormesh(t, f, Y, vmin=-120, vmax=0)

# Use log scale above 100 Hz, linear below.
if scale == 'log':
    yscale = 0.25

    if matplotlib.__version__[0:3] == '1.3':
        yscale = 1
        warnings.warn('You are using matplotlib 1.3.* (and not >
    plt.yscale('symlog', linthreshy=100, linscaley=yscale)
    ax.yaxis.set_major_formatter(matplotlib.ticker.ScalarFormatt

# Set x/y limits by using the maximums from the time/frequency a
plt.xlim(0, t[-1])
plt.ylim(0, f[-1])

# Set axis labels.
plt.xlabel("Time (s)")
plt.ylabel("Frequency (Hz)")

# Show legend and set label.
cbar = plt.colorbar()
cbar.set_label("Intensity (dB)")
```

```
# Display spectrogram.
plt.show()
```

Output