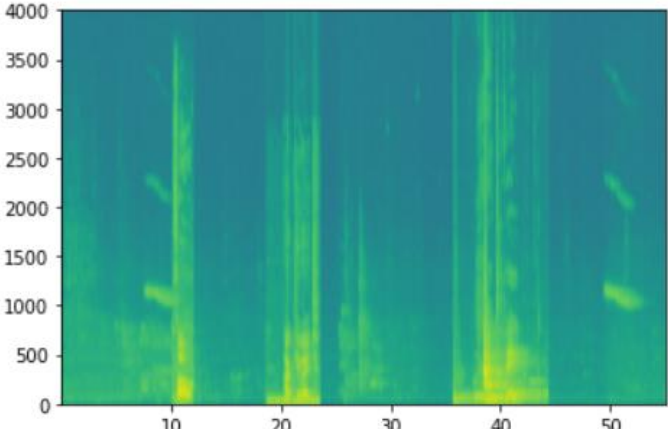
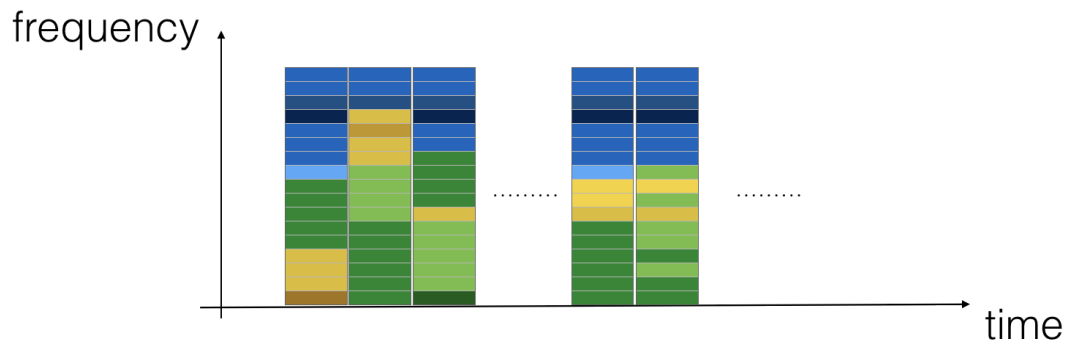


计算机科学与技术学院神经网络与深度学习课程实验报告

实验题目: Trigger word detection		学号: 201900130143
日期: 12/16	班级: 智能班	姓名: 吴家麒
Email: wjq_777@126.com		
<p>实验目的:</p> <p>You will implement a model which will beep every time you say "activate". After the model is completed, you will be able to record your own speech clips and trigger a prompt tone when the algorithm detects that you say "activate".</p>		
<p>实验软件和硬件环境:</p> <p>Anaconda3 + Jupyter notebook</p>		
<p>实验步骤: (不要求罗列完整源代码)</p> <p>1、Data synthesis: Creating a speech dataset</p> <p>现有数据:</p> <ul style="list-style-type: none">· 在不同环境下的背景噪音· 包含 "positive / negative" 词的音频片段 (包括不同的方言)· 总的来说就是有三种音频片段<ul style="list-style-type: none">“background noise”“positive words”“negative words” <p>我们将利用以上三种不同的片段来合成音频数据集.</p> <p>输出: 一批图片</p> <p>1.1 From audio recordings to spectrograms</p> <p>音频实际上是由麦克风记录气压变化产生的, 因此我们可以将音频转换为记录气压变化的数据, 也就是相对应的频谱。我们使用的音频数据频率为 44100Hz。</p> 		



上图表示了每个频率(y轴)在多个时间步(x轴)中的活跃程度。蓝色代表出现频率较小,绿色代表出现频率较大。

声谱图将作为网络的输入 x , $T_x = 5511$ (有 5511 个时间步), 一个时间步中的频率数为 101。同时定义 GRU 网络的输出 $T_y = 1375$ 。这意味着我们利用 GRU 将一个十秒的音频分成 1375 个时间段, 并且尝试从每一个时间段中来判断, 该段是否含有 “activate”

```
Time steps in audio recording before spectrogram (441000,)
Time steps in input after spectrogram (101, 5511)
```

- 441000 (raw audio)
- 5511 = T_x (spectrogram output, and dimension of input to the neural network).
- 10000 (used by the `pydub` module to synthesize audio)
- 1375 = T_y (the number of steps in the output of the GRU you'll build).

1.2 Generating a single training example

合成一个训练样本可以分为以下三步:

- 选一个十秒的背景音频
- 随机插入 0 - 4 段 “activate” 的音频片段
- 随机插入 0 - 2 段 “negative words” 的音频片段

因为我们是插入的音频片段, 所以我们知道 “activate” 片段的位置, 这样就很容易进行标注。

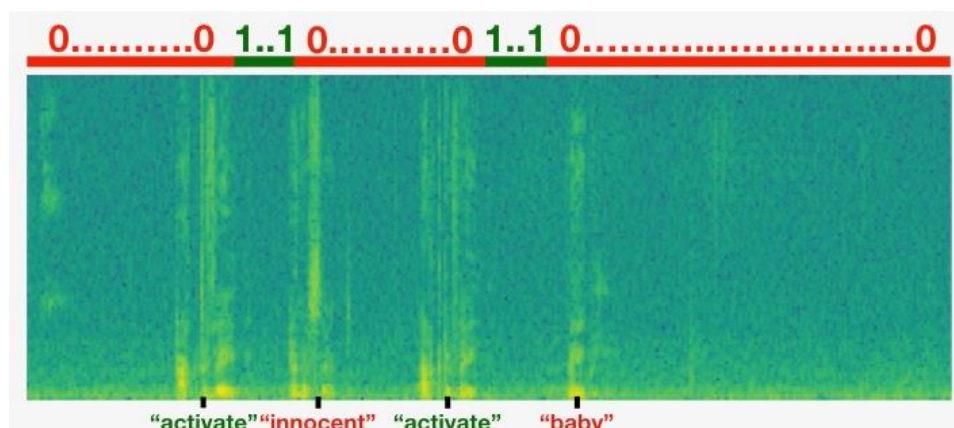
我们利用 `pydub` 包来处理音频, `pydub` 将 1ms 作为一个离散的时间间隔 (10s = 10000ms), 这也是我们为什么将一个十秒的片段表示为 10000 个 step

```
# Load audio segments using pydub
activates, negatives, backgrounds = load_raw_audio()

print("background len: " + str(len(backgrounds[0])))
print("activate[0] len: " + str(len(activates[0])))
print("activate[1] len: " + str(len(activates[1])))
```

```
background len: 10000
activate[0] len: 721
activate[1] len: 731
```

背景音频的标签全设为 0，当插入一段 “activate” 音频时，我们将后面的 50 个 step 的标签都设为 1。



要实现训练集成过程，您将使用以下辅助函数。所有这些函数都将使用 1ms 离散化间隔，所以 10 秒的音频将被离散化为 10,000 步。

1. `get_random_time_segment(segment_ms)` gets a random time segment in our background audio
2. `is_overlapping(segment_time, existing_segments)` checks if a time segment overlaps with existing segments
3. `insert_audio_clip(background, audio_clip, existing_times)` inserts an audio segment at a random time in our background audio using `get_random_time_segment` and `is_overlapping`
4. `insert_ones(y, segment_end_ms)` inserts 1's into our label vector `y` after the word "activate"

第一个函数 `get_random_time_segment(segment_ms)` 用于随机产生一段音频的起始位置和终止位置，

输入：随机产生的片段长度

输出：随机产生片段的起始位置和终止位置

```
def get_random_time_segment(segment_ms):  
    """  
    Gets a random time segment of duration segment_ms in a 10,000 ms audio clip.  
  
    Arguments:  
    segment_ms -- the duration of the audio clip in ms ("ms" stands for "milliseconds")  
  
    Returns:  
    segment_time -- a tuple of (segment_start, segment_end) in ms  
    """  
  
    segment_start = np.random.randint(low=0, high=10000-segment_ms) # Make sure segment doesn't run past the 10sec background  
    segment_end = segment_start + segment_ms - 1  
  
    return (segment_start, segment_end)
```

第二个函数 `is_overlapping(segment_time, existing_segments)` 用来判断即将新加入的片段是否和已经加入的片段有重合。

输入：插入音频的起始位置和终止位置

输出：判断结果

```
# GRADED FUNCTION: is_overlapping

def is_overlapping(segment_time, previous_segments):
    """
    Checks if the time of a segment overlaps with the times of existing segments.

    Arguments:
    segment_time -- a tuple of (segment_start, segment_end) for the new segment
    previous_segments -- a list of tuples of (segment_start, segment_end) for the existing segments

    Returns:
    True if the time segment overlaps with any of the existing segments, False otherwise
    """

    segment_start, segment_end = segment_time

    ### START CODE HERE ### (~ 4 line)
    # Step 1: Initialize overlap as a "False" flag. (~ 1 line)
    overlap = False

    # Step 2: loop over the previous_segments start and end times.
    # Compare start/end times and set the flag to True if there is an overlap (~ 3 lines)
    for previous_start, previous_end in previous_segments:
        if (segment_start >= previous_start and segment_start <= previous_end) or \
            (segment_end >= previous_start and segment_end <= previous_end):
            overlap = True
    ### END CODE HERE ###

    return overlap
```

```
overlap1 = is_overlapping((950, 1430), [(2000, 2550), (260, 949)])
overlap2 = is_overlapping((2305, 2950), [(824, 1532), (1900, 2305), (3424, 3656)])
print("Overlap 1 = ", overlap1)
print("Overlap 2 = ", overlap2)
```

```
Overlap 1 = False
Overlap 2 = True
```

第三个函数 `insert_audio_clip()` 用来向背景音频中插入其他类型的音频。

输入：背景音频、插入音频片段、之前的音频片段

输出：合并后的新音频、新插入片段的起始位置和终止位置

可以分为四个步骤，

- 首先利用之前的函数，创建一段随机的插入位置
- 判断插入位置与之前存在的片段是否产生重叠，产生随机片段一直到没有重叠区域为止
- 将新插入的片段插入到已存在的片段中
- 进行音乐片段的剪辑

```
### START CODE HERE ###
# Step 1: Use one of the helper functions to pick a random time segment onto which to insert
# the new audio clip. (~ 1 line)
segment_time = get_random_time_segment(segment_ms)

# Step 2: Check if the new segment_time overlaps with one of the previous_segments. If so, keep
# picking new segment_time at random until it doesn't overlap. (~ 2 lines)
while is_overlapping(segment_time, previous_segments):
    segment_time = get_random_time_segment(segment_ms)

# Step 3: Add the new segment_time to the list of previous_segments (~ 1 line)
previous_segments.append(segment_time)
### END CODE HERE ###

# Step 4: Superpose audio segment and background
new_background = background.overlay(audio_clip, position = segment_time[0])

return new_background, segment_time
```

```

np.random.seed(5)
audio_clip, segment_time = insert_audio_clip(backgrounds[0], activates[0], [(3790, 4400)])
audio_clip.export("insert_test.wav", format="wav")
print("Segment Time: ", segment_time)
IPython.display.Audio("insert_test.wav")

```

Segment Time: (2915, 3635)

▶ 0:00 / 0:10 🔊 ⋮

第四个函数 `insert_ones()` 将 `activate` 样本的标签设为 1.

输入：当前标签向量 `y`、插入音频的结束位置

输出：修改后的标签向量 `y`

```

# GRADED FUNCTION: insert_ones

def insert_ones(y, segment_end_ms):
    """
    Update the label vector y. The labels of the 50 output steps strictly after the end of the segment
    should be set to 1. By strictly we mean that the label of segment_end_y should be 0 while, the
    50 following labels should be ones.

    Arguments:
    y -- numpy array of shape (1, Ty), the labels of the training example
    segment_end_ms -- the end time of the segment in ms

    Returns:
    y -- updated labels
    """

    # duration of the background (in terms of spectrogram time-steps)
    segment_end_y = int(segment_end_ms * Ty / 10000.0)

    # Add 1 to the correct index in the background label (y)
    ### START CODE HERE ### (~ 3 lines)
    for i in range(segment_end_y+1, segment_end_y+51):
        if i < Ty:
            y[0, i] = 1.0
    ### END CODE HERE ###

    return y

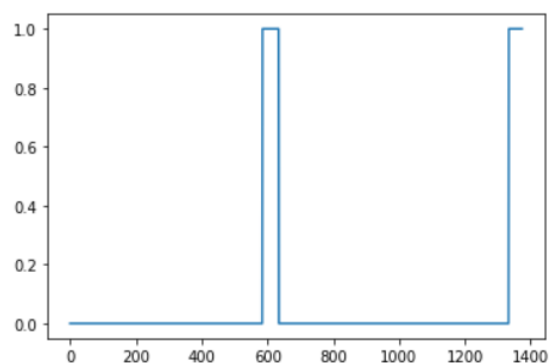
```

```

arr1 = insert_ones(np.zeros((1, Ty)), 9700)
plt.plot(insert_ones(arr1, 4251)[0,:])
print("sanity checks:", arr1[0][1333], arr1[0][634], arr1[0][635])

```

sanity checks: 0.0 1.0 0.0

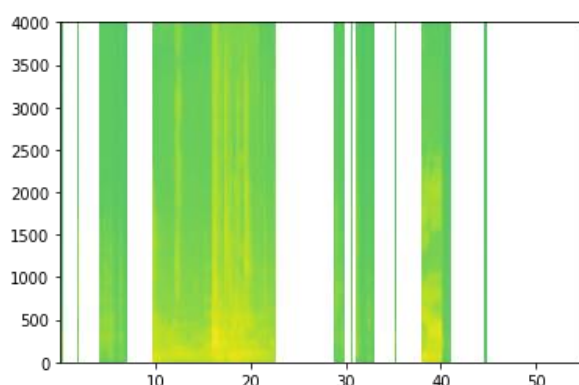


2、利用上述函数，构建训练集样例

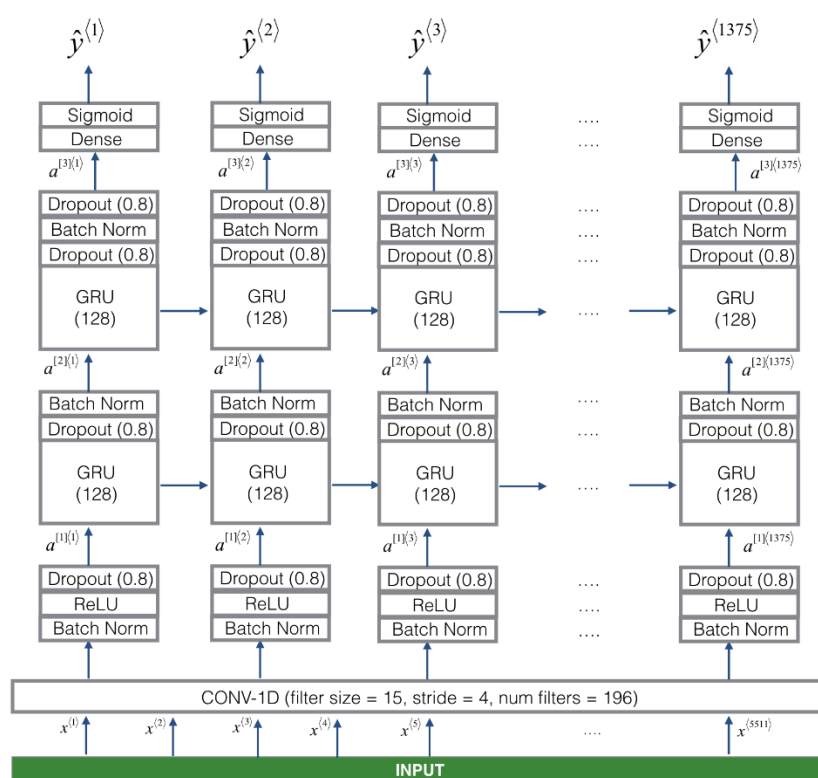
- 将 “activate” 音频和 “negative” 音频都插入 `background` 中。
- 将向量 `y` 初始化为大小为 $(1, T_y)$ 的 0 向量
- 将现存的片段集合初始化为空

- 随机插入 0 — 4 条 “activate” 音频片段，并且将标 y 的对应位置设置为 1
 - 随机插入 0 — 2 条 “negative” 音频片段
- (代码略)

调用结果：



3. 构建模型



CONV-1D 的输入是 5511 个时间步的声谱，输出 1375 个时间步，用于判断这 1375 个时间步是否含有 “activate”。

构建模型的过程可以分为四步。

- 实现卷积，利用 `Conv1D()` 函数，其中有 196 filters, filter size = 15 ,

```
stride = 4
```

- 产生第一个 GRU 层, 通过 `X = GRU(units = 128, return_sequences = True)(X)`

`return_sequences = True` , 代表 GRU 的 hidden states 都会被传递给下一层

- 产生第二个 GRU 层, 和上一层类似, 只不过多了一层 Dropout 层

- 创建一个时间分布的致密层, 通过 `X = TimeDistributed(Dense(1, activation = "sigmoid"))(X)`

(代码略)

3.1 Fit the model

```
Epoch 1/1
```

```
26/26 [=====] - 10s 388ms/step - loss: 0.0726 - acc: 0.9805
```

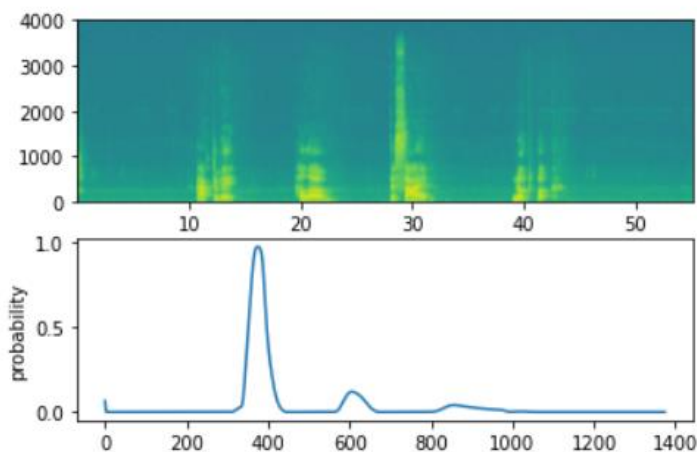
3.2 Test the model

```
loss, acc = model.evaluate(X_dev, Y_dev)
print("Dev set accuracy = ", acc)
```

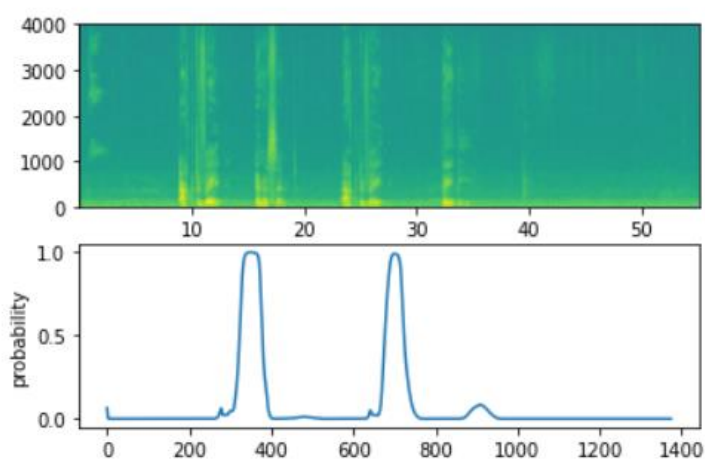
```
25/25 [=====] - 2s 73ms/step
Dev set accuracy = 0.9451636075973511
```

4. 模型预测

```
filename = "./raw_data/dev/1.wav"
prediction = detect_triggerword(filename)
chime_on_activate(filename, prediction, 0.5)
IPython.display.Audio("./chime_output.wav")
```



```
filename = "./raw_data/dev/2.wav"
prediction = detect_triggerword(filename)
chime_on_activate(filename, prediction, 0.5)
IPython.display.Audio("./chime_output.wav")
```



结论分析与体会：

- 1、从本次实验的结果中，实现了对音频唤醒词检测的算法。
- 2、通过本次实验对音频处理和训练的过程进行了学习和运用，对于 Conv 和 GRU 结合的复杂神经网络的训练过程有了更深入的理解。
- 3、对于神经网络在实际应用中的过程有了更为深入的理解和学习。