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：201900130151
日期：2021.12.20	班级： 人工智能	姓名： 莫甫龙
Email：m1533979510@163.com		

实验目的：

您将实现一个模型，该模型将在您每次说“激活”时发出嘟嘟声。之后模型完成后，您将能够录制自己的语音片段并触发算法检测到您说“activa”时发出提示音.

实验软件和硬件环境：

VS code

Win11

实验步骤：（不要求罗列完整源代码）

is_overlapping

```python
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

    return overlap
```

insert_audio_clip

```
    # Get the duration of the audio clip in ms
    segment_ms = len(audio_clip)

    ### 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
```

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 followinf 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
    ### END CODE HERE ###

    return y
```

create_training_example

```
    ### START CODE HERE ###
    # Step 1: Initialize y (label vector) of zeros (≈ 1 line)
    y = np.zeros((1, Ty))

    # Step 2: Initialize segment times as empty list (≈ 1 line)
    previous_segments = []
    ### END CODE HERE ###

    # 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]

    ### START CODE HERE ### (≈ 3 lines)
    # Step 3: Loop over randomly selected "activate" clips and insert in background
    for random_activate in random_activates:
        # Insert the audio clip on the background
        background, segment_time = insert_audio_clip(background, random_activate, previous_segments)
        # Retrieve segment_start and segment_end from segment_time
        segment_start, segment_end = segment_time
        # Insert labels in "y"
        y = insert_ones(y, segment_end)
    ### END CODE HERE ###
```

```
    # 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]

    ### START CODE HERE ### (≈ 2 lines)
    # Step 4: Loop over randomly selected negative clips and insert in background
    for random_negative in random_negatives:
        # Insert the audio clip on the background
        background, _ = insert_audio_clip(background, random_negative, previous_segments)
    ### END CODE HERE ###

    # Standardize the volume of the audio clip
    background = match_target_amplitude(background, -20.0)

    # Export new training example
    file_handle = background.export("train" + ".wav", format="wav")
    print("File (train.wav) was saved in your directory.")

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

    return x, y
```

Model

```
X_input = Input(shape = input_shape)

### START CODE HERE ###

# Step 1: CONV layer (≈4 lines)
X = Conv1D(196, 15, strides=4)(X_input)                                 # CONV1D
X = BatchNormalization()(X)                                 # Batch normalization
X = Activation('relu')(X)                                 # ReLu activation
X = Dropout(0.8)(X)                                 # dropout (use 0.8)

# Step 2: First GRU Layer (≈4 lines)
X = GRU(128, return_sequences=True)(X)                                 # GRU (use 128 units and return the sequences)
X = Dropout(0.8)(X)                                 # dropout (use 0.8)
X = BatchNormalization()(X)                                 # Batch normalization

# Step 3: Second GRU Layer (≈4 lines)
X = GRU(128, return_sequences=True)(X)                                 # GRU (use 128 units and return the sequences
X = Dropout(0.8)(X)                                 # dropout (use 0.8)
X = BatchNormalization()(X)                                 # Batch normalization
X = Dropout(0.8)(X)                                 # dropout (use 0.8)

# Step 4: Time-distributed dense layer (≈1 line)
X = TimeDistributed(Dense(1, activation = "sigmoid"))(X) # time distributed  (sigmoid)

### END CODE HERE ###

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

return model
```

结论分析与体会：

该实验运用 python 自带的库很好地实现了关键字的检测，效果十分好。

就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟 1—3 道问答题：

在做这个实验的时候，刚开始一条条地运行，发现后面生成的音频没有变化，但是在重新运行全部代码以后，生成的音频又符合实验要求了。