Detectarea cuvantului (expresiei) Trigger

Keyword spotting / Trigger Word Detection

- Detectarea cuvintelor trigger (declanșatoare) este tehnologia care permite dispozitivelor precum Amazon Alexa, Google Home, Apple Siri și Baidu DuerOS să se trezească/pornească la auzirea unui anumit cuvânt sau expresie.
- In sistemul nostru toate cuvintele sunt cuvinte cheie (dintre care unul este trigger) si lucreaza in felul urmator: unul are menirea sa trezeasca sistemul, iar celelalte sa execute comenzi vocale.
- Se poate de rulat pe sistemul de operare Windows, astfel încât de fiecare dată când cineva „sistem” să pornească aplicația dvs. preferată sau să aprindă o lampă conectată la rețea în casa dvs. sau să declanșeze un alt eveniment.
- Avem cuvintele cheie: sistem, deschide, ..., ... In total sunt *n* cuvinte cheie.
- In teoria noastra avem un cuvant cheie de baza care le activeaza pe celelalate. Il denumim cuvant super-cheie/trigger. In acest context cuvantul super-cheie este **sistem**. Atunci cand aplicatia va auzi cuvantul sistem noi vom putea folosi si celelelate cuvinte cheie.

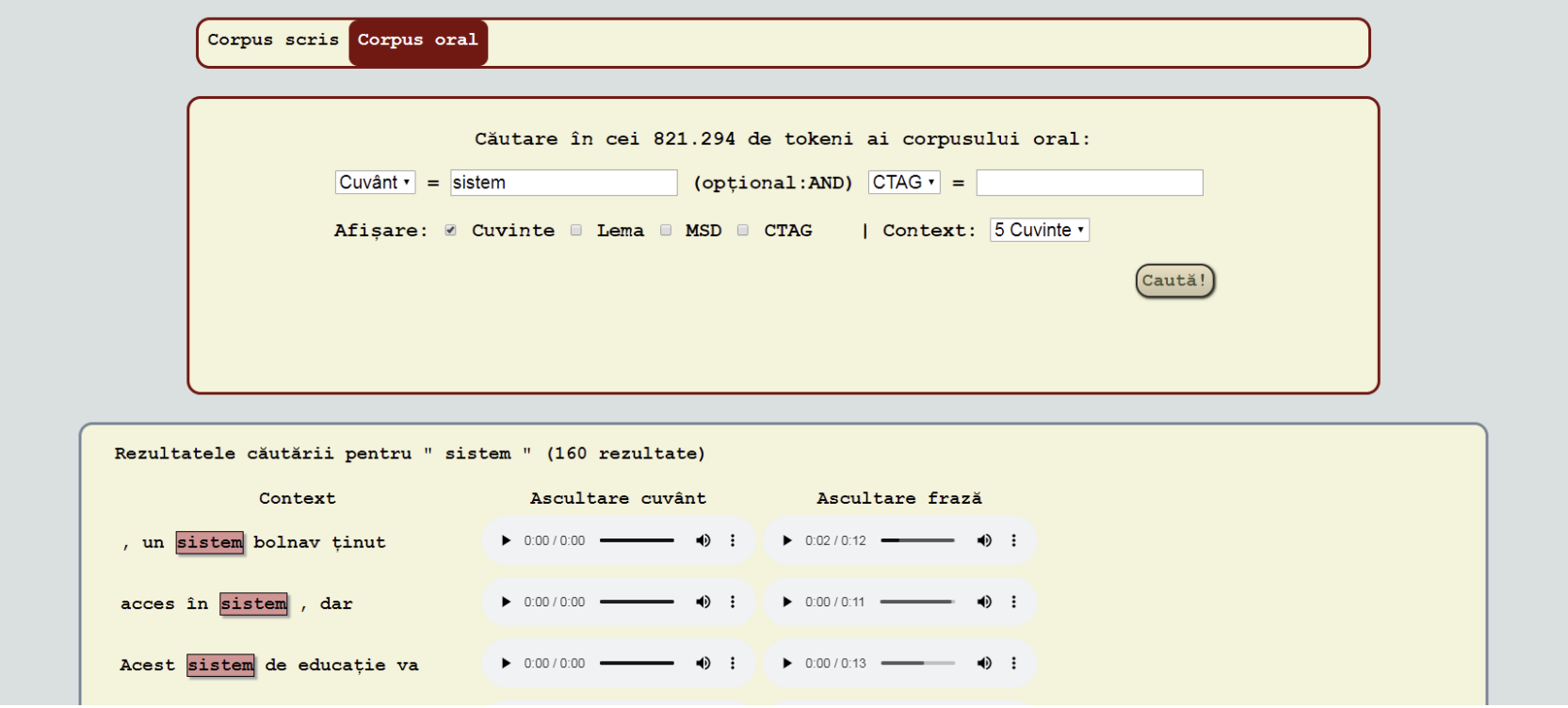
In [6]:

```
import numpy as np ## lucru cu tablouri
from pydub import AudioSegment ## lucru cu procesarea fisierelor audio si extragerea caracteristicilor
import random ## pt a inializa o oarecare structura de date cu valori aleatorii
import sys ## pt a apela diferite metode din sistem
import io ## citire - afisare
import os ## ... din sistemul de operare
import glob ## pt a putea incarca mai multe fisiere dintr-un dosar
import IPython ## pt a rula fisiere media in jupyter notebook
from td_utils import *  ## anumite utilitati pt prelucrarea datelor audio ca materie prima
%matplotlib inline
```

0 - Pregatirea resurselor prime: Colectarea fisierelor audio in limba romana

- Primul pas din lucrarea practica constituie crearea unei liste de cuvinte cheie pt sistemul nostru. Aceasta lista este creata, deocamdata, la discretia noastra si contine: 10 cuvinte cheie dintre care un cuvant este cuvant-pornire (cuvant de trezire a aplicatie).
- sistem (cuvant-trigger)
  - deschide
    - dosar
      - "Colectii"
    - aplicatie (denumire)
      - denumire aplicatie
    - fisier (nume fisier)
    - calculator
    - skype
  - sterge
  - modifica

    - inchide
    - stinge
    - aprinde
- Urmatoarea etapa este colectarea fisierelor audio care redau cuvintele cheie in limba romana. La aceasta etapa am folosit o resursa importanta creata de lingvisti si inormaticieni din Romanaia, si anume proiectul: Corola - Corpus computațional de referință pentru limba română contemporană (http://corola.racai.ro/ (http://corola.racai.ro/))
- Aici, in Corola, este Oral Corpus Query Platform (OCQP): Platformă ce permite identificarea modalității de pronunțare a unui cuvânt în corpusul oral, în diferitele contexte de apariție ale acestuia. Căutarea se poate face pornind de la forma sau lema cuvîntului și se poate asocia cu filtre în funcție de adnotările morfo-sintactice (MSD sau CTag). Puteți vedea o descriere detaliată a platformei aici. http://89.38.230.23/corola_sound_search/index.php (http://89.38.230.23/corola_sound_search/index.php)



1 - Crearea unui set de date pentru detectarea vorbirii

- Un set de date pentru recunoașterea vorbirii ar trebui să fie în mod ideal cât mai aproape posibil de aplicația noastră.
- În acest caz, dorim să recunoaștem cuvintele cheie în mediile de lucru cu un PC care ruleaza pe Windows (deschide, folder, browser, spații deschise...).
- Prin urmare, trebuie să creăm înregistrări (recordings) cu un amestec de cuvinte pozitive ("deschide", "inchide", "sistem") și cuvinte negative (cuvinte aleatorii, altele decât cuvintele cheie) si diferite sunete de fundal. Să vedem cum putem crea un astfel de set de date.

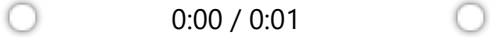
1.1 - Ascultarea datelor audio

- Unul dintre prietenii ne ajută în acest proiect și s-a dus la biblioteci, cafenele, restaurante, universități, case și birouri din întreaga regiune pentru a înregistra zgomote de fundal, precum și fragmente audio ale oamenilor care spun cuvinte pozitive/negative. Acest set de date include persoane care vorbesc cu o varietate de accente. Dar in realitate am folosit datele dintr-un corpus deja colectat.
- În folderul `raw_data` , vom pastra fișiere audio brute ale cuvintelor pozitive, cuvintelor negative și zgomotului de fundal. Veți folosi aceste fișiere audio pentru a sintetiza un set de date pentru a antrena modelul.
  - Folderele de cuvinte cheie conțin exemple pozitive de oameni care rostesc cuvintele ... .
  - Folderul „negative” conține exemple negative de oameni care spun cuvinte aleatorii, altele decât cuvintele cheie.
  - Există un cuvânt pentru fiecare înregistrare audio.
  - Directorul „background” conține clipuri audio de 10 secunde de zgomot de fundal în diferite medii.

Vom rula celulele de mai jos pentru a asculta câteva exemple.

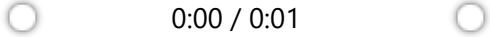
In [15]:
```
1 IPython.display.Audio("./raw_data/positive/sistem/13.wav")
```

Out[15]:



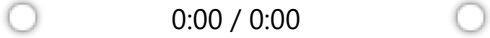
In [16]:
```
1 IPython.display.Audio("./raw_data/positive/calculatorul/2.wav")
```

Out[16]:



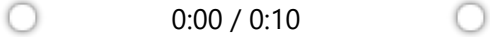
In [17]:
```
1 IPython.display.Audio("./raw_data/negative/12.wav")
```

Out[17]:



In [18]:
```
1 IPython.display.Audio("./raw_data/background/7.wav")
```

Out[18]:



Vom folosi aceste trei tipuri de înregistrări (pozitive/negative/background) pentru a crea un set de date etichetat.

1.2 - De la înregistrări audio la spectrograme

Ce este de fapt o înregistrare audio?

- Un microfon înregistrează mici variații ale presiunii aerului de-a lungul timpului și aceste mici variații ale presiunii aerului sunt acelea pe care urechile noastre le percepe și ca sunet.

- Vă puteți gândi că o înregistrare audio ca fiind o listă lungă de numere care măsoară micile modificări ale presiunii aerului detectate de microfon.
- Vom folosi audio de 44100 Hz (sau 44100 Herți).
  - Aceasta înseamnă că microfonul ne oferă 44.100 de numere pe secundă.
  - Astfel, un clip audio de 10 secunde este reprezentat de 441.000 de numere (= 10×44.100).

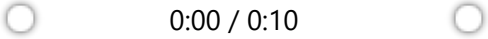
Spectrograma

- Este destul de dificil să ne dăm seama din reprezentare „raw” (brută) a unui clip audio dacă a fost spus un cuvânt pozitiv.
- Pentru a ajuta modelul nostru secvențail să învețe mai ușor să detecteze cuvintele cheie, vom calcula o *spectrogramă* a sunetului, adică vom converti sunetul intr-o imagine a sunetului.
- Spectrograma ne spune cât de multe frecvențe diferite sunt prezente într-un clip audio în orice moment timp.
- Dacă ați avut vreun curs avansat de procesare a semnalului sau de transformări Fourier:
  - O spectrogramă este calculată prin glisarea unei ferestre peste semnalul audio brut și calculând cele mai active frecvențe din fiecare fereastră folosind o transformată Fourier.
  - Dacă nu înțelegeți propoziția anterioară, nu vă faceți griji.

Să ne uităm la un exemplu.

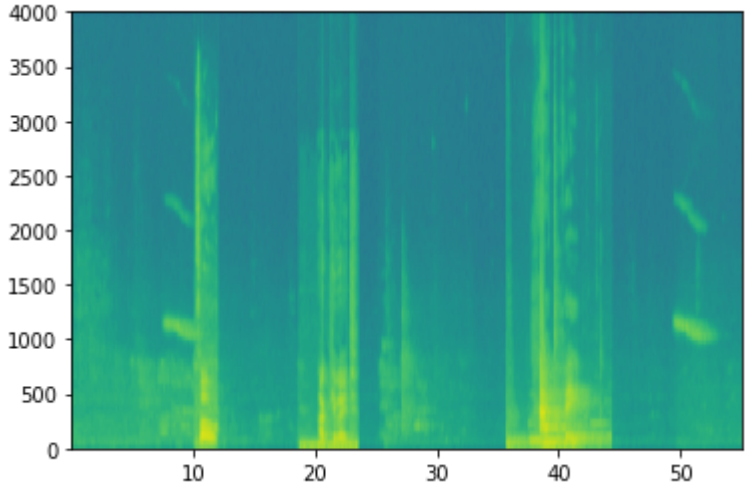
In [19]:
```
1 IPython.display.Audio("audio_examples/sistem.wav")
```

Out[19]:



In [7]:

```
1 x = graph_spectrogram("audio_examples/example_train.wav")
```



Graficul de mai sus reprezintă cât de activă este fiecare frecvență (axa y) pe un număr de pași de timp (axa x).

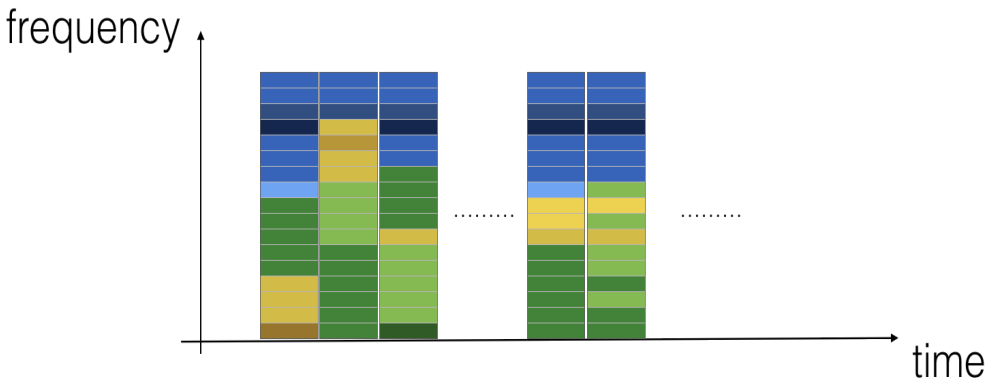


Figura 1: Spectrograma unei înregistrări audio cu cuvantul trigger "sistem".

- Culoarea din spectrogramă arată gradul în care diferite frecvențe sunt prezente în audio în diferite momente de timp.
- Verde înseamnă că o anumită frecvență este mai activă sau mai prezentă în clipul audio (mai tare).

- Pătratele albastre indică frecvențe mai puțin active.
- Dimensiunea spectrogramei de ieșire depinde de hiperparametrii software-ului de spectrogramă și de lungimea intrării.
- În acest proiect, vom lucra cu clipuri audio de 10 secunde ca „lungime standard” pentru exemplele noastre de antrenament.
  - Numărul de pași de timp al spectrogramei va fi 5511.
  - Veți vedea mai târziu că spectrograma va fi intrarea x în rețea și deci $T_x = 5511$.

In [8]:

```
1 _, data = wavfile.read("audio_examples/example_train.wav")
2 print("Pași de timp în înregistrarea audio înainte de spectrogramă", data[:,0].shape)
3 print("Pași de timp în intrare după calcularea spectrogramei", len(x[0]))
4
```

```
Pași de timp în înregistrarea audio înainte de spectrogramă (441000,)
Pași de timp în intrare după calcularea spectrogramei 5511
```

Acum putem defini:

In [9]:

```
1 Tx = len(x[0]) # numărul de pași de timp introduși în model din spectrogramă
2 n_freq = len(x[:,0]) # numărul de frecvențe introduse în model la fiecare pas de timp al spectrogramei
3
4 print("Numărul de pași: ", Tx, "\nNumarul de frecventa per pas: ", n_freq)
5
6 x[0][0]
```

```
Numărul de pași:  5511
Numarul de frecventa per pas:  101
```

Out[9]:

```
9.208503132631792e-08
```

Împărțirea în intervale de timp

Rețineți că putem împărți un interval de timp de 10 secunde cu diferite unități (pași).

- Audio brut împarte 10 secunde în 441.000 de unități.
- O spectrogramă împarte 10 secunde în 5.511 unități.
  - $T_x = 5511$
- Vom folosi un pachet Python numit `pydub` pentru a sintetiza audio și a împarte 10 secunde în 10.000 de unități.
- Ieșirea modelului nostru va împărți 10 secunde în 1.375 de unități.
  - $T_y = 1375$
  - Pentru fiecare dintre cei 1375 de pași de timp, modelul prezice dacă cineva a terminat recent de rostit cuvântul cheie.
- Toți aceștia sunt hiperparametri și pot fi modificați (cu excepția 441000, care este o funcție a microfonului).
- Am ales valori care se încadrează în intervalul standard utilizat pentru sistemele de vorbire.

In [10]:
```
Ty = 1375 # Numărul de pași de timp din rezultatul modelului nostru
```

1.3 - Generarea unui singur exemplu de antrenament

Beneficiile sintetizării datelor

Deoarece datele de vorbire sunt greu de achiziționat și de etichetat, vom sintetiza datele de antrenament utilizând clipurile audio ale cuvintelor cheie, negative și audio de fundal.

- Este destul de lent să înregistrezi o mulțime de clipuri audio de 10 secunde cu cuvinte cheie aleatorii.
- În schimb, este mai ușor să înregistrați o mulțime de cuvinte pozitive și negative și să înregistrați separat zgomotul de fundal (sau descărcați zgomotul de fundal din surse online gratuite).

Proces pentru sintetizarea unui clip audio

- Pentru a sintetiza un singur exemplu de antrenament, veți:
  - Alegeți un clip audio de fundal aleatoriu de 10 secunde
  - Inserați aleatoriu 0-3 clipuri audio de cuvinte pozitive în acest clip de 10 secunde
  - Inserați aleatoriu 0-2 clipuri audio de cuvinte negative în acest clip de 10 secunde
- Pentru că vom sintetiza cuvântul „sistem” în clipul de fundal, vom ști exact când în clipul de 10 secunde își face apariția „sistem”.
  - Acest lucru facilitează generarea etichetelor $y^{\langle t \rangle}$.

Pydub

- Vom folosi pachetul pydub pentru a procesa fisiere audio.
- Pydub convertește fișierele audio brute în liste de structuri de date Pydub.
  - Nu vă faceți griji cu privire la detaliile structurilor de date.
- Pydub folosește 1 ms ca interval de discretizare (1 ms este 1 milisecundă = 1/1000 secunde).
  - Acesta este motivul pentru care un clip de 10 secunde este întotdeauna reprezentat folosind 10.000 de pași.

In [11]:

```
# Load audio segments using pydub
# sistem, deschide, inchide, sterge, porneste, browser, dosar, negatives, backgrounds = load_raw_audio(
backgrounds = load_raw_audio("raw_data/background/")
sistem = load_raw_audio("raw_data/positive/sistem")
print(len(sistem))
```

```
15
```

Suprapunerea clipurilor audio pozitive/negative deasupra sunetului de fundal

- Având în vedere un clip de fundal de 10 secunde și un scurt clip audio care conține un cuvânt pozitiv sau negativ, trebuie să putetem „adăuga” cuvântul clip audio deasupra sunetului de fundal.
- Vom insera mai multe clipuri de cuvinte pozitive/negative în sunetul de fundal, dar nu vom dori să inserați un cuvânt pozitiv sau un cuvânt aleatoriu undeva unde se va suprapune cu un alt clip pe care l-ați adăugat anterior.
  - Pentru a ne asigura că segmentele audio pozitive/negative nu se suprapun atunci când sunt introduse, vom ține cont de timpii clipurilor audio introduse anterior.
- Pentru a fi clar, atunci când introducem un cavant „sistem” de 1 secundă într-un clip de 10 secunde de zgomot dintr-o cafenea, **nu veți ajunge cu un clip de 11 secunde.**
  - Clipul audio rezultat are tot 10 secunde!

Etichetarea cuvintele pozitive/negative

- Sa ne amintim că etichetele $y^{\langle t \rangle}$ reprezintă dacă cineva tocmai a terminat de rostit „sistem” sau nu.
  - $y^{\langle t \rangle} = 1 \Rightarrow [1, 0, 0, 0, \dots]$ când acel clip a terminat de rostit „sistem”.
  - $y^{\langle t \rangle} = 2 \Rightarrow [0, 1, 0, 0, \dots]$ când acel clip a terminat de spus „deschide”.

- Având în vedere un clip de fundal, putem inițializa $y^{\langle t \rangle} = 0$ pentru toți t, deoarece clipul nu conține niciun „cuvânt cheie”.

- Când inseram sau suprapunem un cuvant cheie, vom actualiza și etichetele pentru $y^{\langle t \rangle}$.
  - În loc să actualizăm eticheta unui singur pas de timp, vom actualiza 50 de pași ai rezultatului pentru a avea eticheta țintă 1.
- Vom antrena o retea neurala recurenta pentru a detecta când cineva a **terminat** să spună „sistem”.

Exemplu

- Să presupunem că clipul audio „sistem” sintetizat se termină la a 5-a secunda în audio de 10 secunde - exact la jumătatea clipului.
- Ne amintim că $T_y = 1375$, deci pasul de timp $687 = \texttt{int(1375*0.5)}$ corespunde momentului de 5 secunde din clipul audio.
- Setați $y^{\langle 688 \rangle} = 1$.
- Vom permite GRU să detecteze cuvantul „sistem” oriunde într-un timp scurt intern **după** acest moment, așa că de fapt **setăm 50 de valori consecutive** ale iesirii $y^{\langle t \rangle}$ cu valoarea 1.
  - Mai exact, avem $y^{\langle 688 \rangle} = y^{\langle 689 \rangle} = \cdots = y^{\langle 737 \rangle} = 1$.

Datele sintetizate sunt mai ușor de etichetat

- Acesta este un alt motiv pentru sintetizarea datelor de antrenament: este relativ simplu să generam aceste etichete $y^{\langle t \rangle}$ așa cum este descris mai sus.
- În schimb, dacă avem 10 secunde de audio înregistrate pe un microfon, este nevoie de mult timp pentru o persoană să îl asculte și să marcheze/eticheteze manual exact când cuvantul „sistem” s-a terminat.

Vizualizarea etichetelor

- Iată o figură care ilustrează etichetele $y^{\langle t \rangle}$ într-un clip.

- Am introdus „sistem”, „nevinovat”, „sistem”, „Mircea”.
- De retinut că etichetele pozitive „1” sunt asociate numai cu cuvintele pozitive.

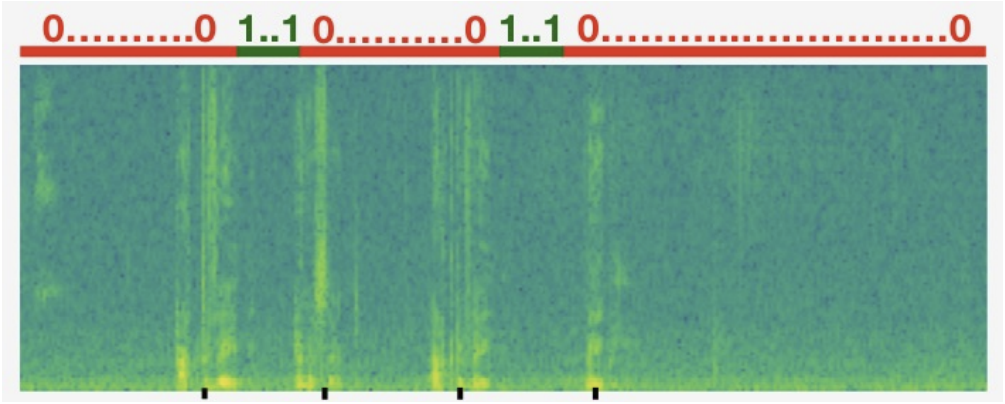


Figura 2

Funcții de ajutor (helpers.py)

Pentru a implementa procesul de sinteză a setului de antrenament, vom folosi următoarele funcții de ajutor.

- Toate aceste funcții vor folosi un interval de discretizare de 1 ms
- Cele 10 secunde de sunet sunt întotdeauna discretizate în 10.000 de pași.

1. `get_random_time_segment(segment_ms)`
    - preia un segment de timp aleatoriu din sunetul de fundal.
2. `is_overlapping(segment_time, existing_segments)`
    - verifică dacă un segment de timp se suprapune cu segmentele existente
3. `insert_audio_clip(background, audio_clip, existing_times)`
    - inserează un segment audio la un moment aleator în sunetul de fundal
    - folosește funcțiile `get_random_time_segment` și `is_overlapping`
4. `insert_label(y, segment_end_ms)`
    - inserează 1 în vectorul de etichetă y după cuvântul „sistem”, 2 pentru a eticheta cuvântul cheie

„deschide”, ...

Obțineți un segment de timp aleatoriu

- Funcția `get_random_time_segment(segment_ms)` returnează un segment de timp aleatoriu în care putem insera un clip audio de durata `segment_ms` .

In [12]:

```
def get_random_time_segment(segment_ms):
    """
     Obține un segment de timp aleatoriu cu durata segment_ms într-un clip audio de 10.000 ms.

     Argumente:
     segment_ms -- durata clipului audio în ms ("ms" înseamnă "milisecunde")

     Returneaza:
     segment_time -- un tuplu de (segment_start, segment_end) în ms
    """

    segment_start = np.random.randint(low=0, high=10000-segment_ms)   # Asigurați-vă că segmentul nu tr
    segment_end = segment_start + segment_ms - 1

    return (segment_start, segment_end)
```

Verificam dacă clipurile audio se suprapun

- Să presupunem că am introdus clipuri audio la segmente (1000,1800) și (3400,4500).
  - Primul segment începe la pasul 1000 și se termină la pasul 1800.
  - Al doilea segment începe la 3400 și se termină la 4500.
- Dacă ne gândim dacă să inserăm un nou clip audio la (3000,3600), acesta se suprapune cu unul dintre

segmentele inserate anterior?
    - În acest caz, (3000,3600) și (3400,4500) se suprapun, așa că ar trebui să decidem să nu inserăm un clip aici.
- Mai jos vom implementa `is_overlapping(segment_time, existing_segments)` pentru a verifica dacă un nou segment de timp se suprapune cu oricare dintre segmentele anterioare.
- Va trebui să efectuam 2 pași:

1. Creați un flag „False”, pe care îl vom seta ulterior cu „True” dacă descoperim că există o suprapunere.
2. Vom compara timpul dat cu timpul de început și de sfârșit ale segmentului. Dacă există o suprapunere, vom seta indicatorul definit în (1) ca True.

Există suprapunere dacă:

- Noul segment începe înainte ca segmentul anterior să se încheie **și**
- Noul segment se termină după ce începe segmentul anterior.

In [13]:

```python
def is_overlapping(segment_time, previous_segments):
    """
    Verifică dacă timpul unui segment se suprapune cu timpul segmentelor existente.

    Argumente:
    segment_time -- un tuplu de (segment_start, segment_end) pentru noul segment
    previous_segments -- o listă de tupluri ale (segment_start, segment_end) pentru segmentele existent

    Returneaza:
    True dacă segmentul de timp se suprapune cu oricare dintre segmentele existente, Fals în caz contra
    """

    segment_start, segment_end = segment_time

    # Pasul 1: Inițializam suprapunerea ca un flag „False”.
    overlap = False

    # Pasul 2: treceți într-un ciclu peste previous_segments.
    # Comparam timpurile de început/sfârșit și setam True dacă există o suprapunere
    for previous_start, previous_end in previous_segments:
        if segment_start <= previous_end and segment_end >= previous_start:
            overlap = True

    return overlap
```

In [14]:

```
overlap1 = is_overlapping((950, 1430), [])
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
```

Inserați clipul audio

- Vom folosim funcțiile de ajutor anterioare pentru a insera un nou clip audio pe fundalul de 10 secunde la un moment dat.
- Ne vom asigura că orice segment nou introdus nu se suprapune cu segmentele inserate anterior.
- Vom implementa functia insert_audio_clip() in 4 pași:

1. Obținem lungimea clipului audio care urmează să fie introdus.
    - un segment de timp aleatoriu a cărui durată este egală cu durata clipului audio care urmează să fie inserat.
2. Ne asiguram că segmentul de timp nu se suprapune cu niciunul dintre segmentele de timp anterioare.
    - dacă se suprapune, atunci revenim la pasul 1 și alegem un nou segment de timp.
3. Adăugam noul segment de timp la lista de segmente de timp existente
    - aceasta va ține evidența tuturor segmentelor pe care le-am inserat.
4. Suprapunem clipul audio peste fundal folosind pydub.

In [15]:

```
def insert_audio_clip(background, audio_clip, previous_segments):
    """
     Introducem un nou segment audio peste zgomotul de fundal la un pas de timp aleatoriu
     aigurandu-ne ca segmentul audio nu se suprapune cu segmentele existente.

     Argumente:
     background -- o înregistrare audio de fundal de 10 secunde.
     audio_clip -- clipul audio care urmează să fie inserat/suprapus.
     previous_segments -- segmentele de timp în care segmente audio au fost deja plasate

     Returneaza:
     new_background -- sunetul de fundal actualizat
    """

    # Obținem durata clipului audio în ms
    segment_ms = len(audio_clip)

    # Pasul 1: Alege un segment de timp aleatoriu in care să introducem noul audio_clip
    segment_time = get_random_time_segment(segment_ms)

    # Pasul 2: Verificam dacă noul segment_time se suprapune cu unul dintre segmentele_anterioare.
    # Atata timp cat se suprapun noile segmente generate vom alege un segment_time la intamplare
    # până când nu se suprapune.
    while is_overlapping(segment_time, previous_segments):
        segment_time = get_random_time_segment(segment_ms)

    # Pasul 3: Adăugam noul segment_time la lista de segmente_anterioare
```

```
    previous_segments.append(segment_time)

    # Pasul 4: Suprapunem segmentul audio pe fundal
    new_background = background.overlay(audio_clip, position = segment_time[0])

    return new_background, segment_time
```

In [16]:

```
np.random.seed(5)
audio_clip, segment_time = insert_audio_clip(backgrounds[0], sistem[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, 3294)
```

Out[16]:

0:00 / 0:10

Introducem valoarea 1 pentru etichetele pozitive (target)

- Vom implementa codul pentru a actualiza etichetele $y^{\langle t \rangle}$, presupunând că tocmai am inserat un clip audio „sistem”.
- În codul de mai jos, `y` este un vector unidimensional `(1,1375)`, deoarece $T_y = 1375$.
- Dacă clipul audio „sistem” se termină la pasul de timp t, atunci setam $y^{\langle t+1 \rangle} = 1$ și setați, de asemenea, următoarele 49 de valori consecutive suplimentare la 1.
  - Sa observam că, dacă cuvântul țintă apare aproape de sfârșitul întregului clip audio, este posibil să nu existe 50 de pași de timp suplimentari de setat la 1.

- Ne asiguram că nu ajungem la sfârșitul matricei și încercam să actualizam `y[0][1375]` , deoarece indicii corecti sunt `y[0][0]` până la `y[0][1374] ]` deoarece $T_y = 1375$.
- Deci, dacă cuvantul "sitem" se termină la pasul 1370, vom seta `y[0][1371] = y[0][1372] = y[0][1373] = y[0][1374] = 1`

Vom implementa functia `insert_ones()` .

In [17]:

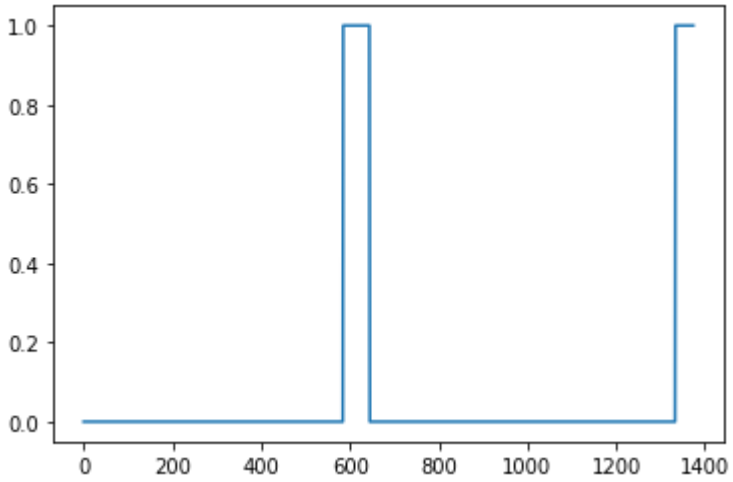
```
def insert_label(y, segment_end_ms):
    """
     Se va actualiza vectorul etichetă y. Etichetele celor 50 de pași de ieșire sunt strict după
     sfârșitul segmentului si ii vom seta la 1.

     Argumente:
     y -- matrice numpy de forma (1, Ty), etichetele exemplului de antrenare
     segment_end_ms -- timpul de sfarsit al segmentului în ms

     Returneaza:
     y -- etichete actualizate
    """

    # durata sunteului de background (dupa ce a fost convertit in spetrograma)
    segment_end_y = int(segment_end_ms * Ty / 10000.0)

    # Adauga valaoarea 1 la indexul corect din eticheta de fundal (y)
    for i in range(segment_end_y + 1, segment_end_y + 61):
        if i < Ty:
                y[0, i] = 1

    return y
```

In [18]:

```
1 arr1 = insert_label(np.zeros((1, Ty)), 9700)
2 plt.plot(insert_label(arr1, 4251)[0,:])
```

Out[18]:

```
[<matplotlib.lines.Line2D at 0x7fc544c30390>]
```



Crearea setului de antrenare

În cele din urmă, putem folosi `insert_audio_clip` și `insert_label` pentru a crea exemple de antrenare.

Vom implementa `create_training_example()` . Va trebui să facem următorii pași:

1. Inițializa vectorul clasa y ca o matrice numerică de zerouri și forma $(1, T_y)$.
2. Inițializa setul de segmente existente într-o listă goală.
3. Selecta aleatoriu de la 0 până la 4 clipuri audio „sistem” și le vom introduce în clipul de 10 secunde. De asemenea, vom introduce etichete în poziția corectă în vectorul clasa y.
4. Selecta aleatoriu până la 2 clipuri audio negative și le vom introduce în clipul de 10 secunde.

In [19]:

```
def create_training_example(background, positive, negative):
    """
     Creează un exemplu de antrenament cu un fundal dat, positive și negative.

     Argumente:
     background -- o înregistrare audio de fundal de 10 secunde
     positive -- o listă de segmente audio ale cuvântului „sistem”
     negative -- o listă de segmente audio de cuvinte aleatorii care nu sunt positive

     Returneaza:
     x -- spectrograma exemplului de antrenare
     y -- eticheta la fiecare pas de timp al spectrogramei
    """

    np.random.seed(18)

    # Facem fundalul mai silențios
    background = background - 30

    # Pasul 1: Inițializam y cu zerouri
    y = np.zeros((1, Ty))

    # Pasul 2: Inițializam segmentele de timp cu o listă goală
    previous_segments = []

    # Selecteaza 0-4 clipuri audio aleatorii din întreaga listă de înregistrări ale cuv „sistem”.
    number_of_positive = np.random.randint(0, 5)
```

```
    random_indices = np.random.randint(len(positive), size=number_of_positive)
    random_positive = [positive[i] for i in random_indices]

    # Pasul 3: Ciclam peste clipurile „sistem” selectate aleatoriu și le inseram în fundal
    for random_positiv in random_positive:
        background, segment_time = insert_audio_clip(background, random_positiv, previous_segments)
        segment_start, segment_end = segment_time
        # Inseram etichetele in "y"
        y = insert_label(y, segment_end_ms=segment_end)

    # Selectam aleatoriu 0-2 cuvinte negative
    number_of_negatives = np.random.randint(0, 3)
    random_indices = np.random.randint(len(negative), size=number_of_negatives)
    random_negatives = [negative[i] for i in random_indices]

    # Pasul 4: Ciclam peste clipurile negative selectate aleatoriu și inseram în fundal
    for random_negative in random_negatives:
        background, _ = insert_audio_clip(background, random_negative, previous_segments)

    # Standardizam volumul clipului audio
    background = match_target_amplitude(background, -30.0)

    # Exportam un nou exemplu de antrenare
    file_handle = background.export("train" + ".wav", format="wav")
    print("Fisierul (train.wav) a fost salvat in folderul background.")

    # Obținem spectrograma noii înregistrări (impreuna cu suprapunerea pozitive și negative)
    x = graph_spectrogram("train.wav")
```

```
56
57      Tx = len(x[0]) # numărul de pași de timp introduși în model din spectrogramă
58      n_freq = len(x[:,0]) #
59      x = x.reshape(Tx, n_freq)
60
61      return x, y
```

In [ ]:

```
1  IPython.display.Audio("train.wav")
```

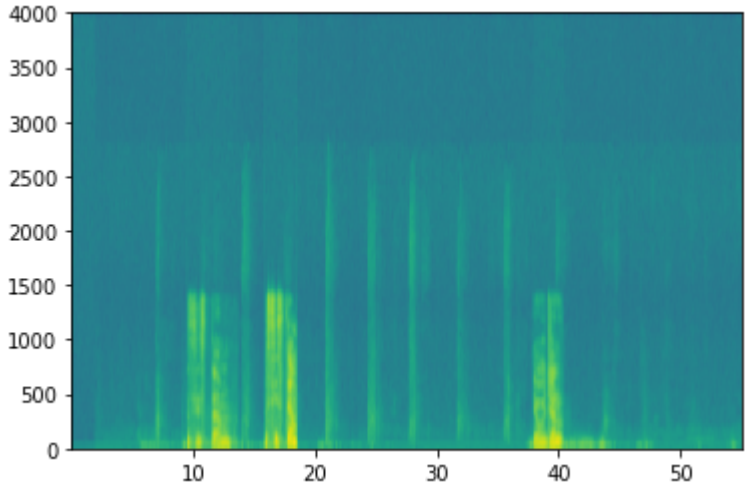
In [22]:

```
backgrounds = load_raw_audio("raw_data/background/")
positive = load_raw_audio("raw_data/positive/sistem")
negative = load_raw_audio("raw_data/negative")
x, y = create_training_example(backgrounds[0], positive, negative)
# y
# x
y[0][:]
```

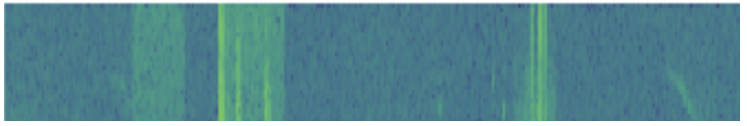
```
Fisierul (train.wav) a fost salvat in folderul background.
```

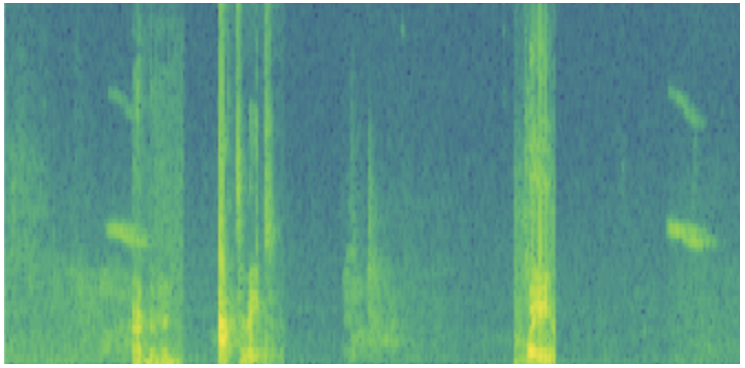
Out[22]:

```
array([0., 0., 0., ..., 0., 0., 0.])
```



Expected Output





In [ ]:
```
1 IPython.display.Audio("train.wav")
```

În cele din urmă, putem afisa etichetele asociate pentru exemplul de antrenament generat.

In [ ]:
```
1 plt.plot(y[0])
```

1.4 - Set complet de antrenament

- Acum am implementat codul necesar pentru a genera un singur exemplu de antrenament.
- Am folosit acest proces pentru a genera un set mare de antrenament.
- Pentru a economisi timp, am generat deja un set de exemple de antrenare.

In [33]:
```
1 # Incarcam exemple de antrenare
2 X = np.load("XY_train/X_300_sistem.npy")
3 Y = np.load("XY_train/Y_300_sistem.npy")
4
```

In [45]:
```
# print(Y[0])
```

```
[[0. 0. 0. ... 0. 0. 0.]]
```

1.5 - Set de testare

- Pentru a testa modelul nostru, am înregistrat un set de testare de 25 de exemple.
- Luam fraze din corola care contin cuvintele cheie pt a testa sa vedem cum se descurca modelul
- În timp ce datele noastre de antrenament sunt sintetizate, dorim să creăm un set de dezvoltare folosind aceeași distribuție ca și intrările reale.
- Astfel, am înregistrat 25 de clipuri audio de 10 secunde cu oameni care spuneau „sistem” și alte cuvinte aleatorii și le-am etichetat manual.

2 - Model

- Acum că am construit un set de date, să scriem și să antrenăm un model de detectare a cuvintelor trigger!
- Modelul va folosi straturi convoluționale 1-D, straturi recurente GRU și straturi dense.
- Să încărcăm pachetele care vă vor permite să utilizați aceste straturi în Keras. Acest lucru poate dura un minut pentru a încărca.

In [23]:
```
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import Model, load_model, Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout, Input, Masking, TimeDistributed, LSTM,
from tensorflow.keras.layers import GRU, Bidirectional, BatchNormalization, Reshape
from tensorflow.keras.optimizers import Adam
```

2.1 - Construirea modelului

Scopul nostru este să construim o rețea care va primi la intrare o spectrogramă și va emite un semnal atunci când detectează cuvântul trigger. Această rețea va folosi 4 straturi: * Un strat convoluțional * Două straturi GRU * Un strat dens.

Iată arhitectura pe care o vom folosi.

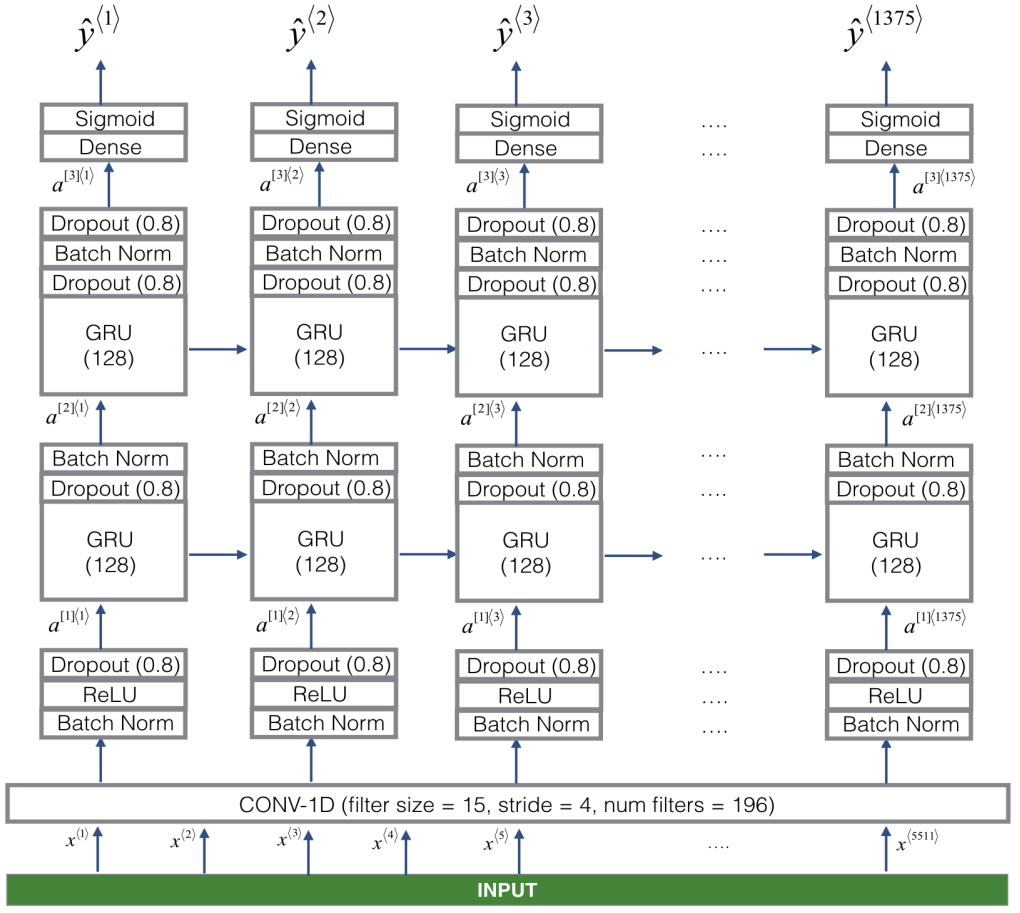


Figure 3

Strat convoluțional 1D

Un strat cheie al acestui model este stratul convoluțional 1D (aproape de partea de jos a figurii 3).

- Introduce spectrograma cu 5511 pasi. Fiecare pas este un vector de 101 unități.
- Emite o ieșire de 1375 de pași
- Această ieșire este procesată în continuare de mai multe straturi pentru a obține rezultatul final $T_y = 1375$.
- Acest strat convoluțional 1D joacă un rol similar cu convoluțiile 2D pe care le-ați văzut în Cursul despre CNN, de a extrage caracteristici de nivel scăzut și apoi, eventual, de a genera o ieșire de o dimensiune mai mică.
- Din punct de vedere computațional, stratul de conversie 1-D ajută și la accelerarea modelului, deoarece acum GRU poate procesa doar 1375 de pași de timp, mai degrabă decât 5511 de pași de timp.

GRU, dens și sigmoid

- Cele două straturi GRU citesc secvența intrărilor de la stânga la dreapta.
- Un strat dens plus sigmoid face o predicție pentru $y^{\langle t \rangle}$.
- Deoarece y este o valoare binară (0 sau 1), folosim o ieșire sigmoid la ultimul strat pentru a estima șansa ca rezultatul să fie 1, corespunzând utilizatorului care tocmai a spus „sistem”.

RNN unidirecțional

- Rețineți că folosim un **RNN unidirecțional** mai degrabă decât un RNN bidirecțional.
- Acest lucru este foarte important pentru detectarea cuvântului declanșator, deoarece dorim să putem detecta cuvântul declanșator aproape imediat după ce este rostit.
- Dacă am folosi un RNN bidirecțional, ar trebui să așteptăm ca toate cele 10 secunde de audio să fie înregistrate înainte de a putea spune dacă a fost spus „sistem” în prima secundă a clipului audio.

Implementarea modelului

Implementarea modelului se poate face în patru etape:

Pasul 1: Stratul Convolutional. Vom utiliza un strat `Conv1D()` din Keras cu 196 de filtre, o dimensiune a filtrului de 15 (`kernel_size=15`) și un pas de 4. conv1d (https://keras.io/layers/convolutional/#conv1d)

```
output_x = Conv1D(filters=...,kernel_size=...,strides=...)(input_x)
```

- Vom folosi f-tia de activare ReLu.

```
output_x = Activation("...")(input_x)
```

- Vom aplica dropout, folosind o rată de păstrare de 0,8

```
output_x = Dropout(rate=...)(input_x)
```

Pasul 2: Primul strat GRU (un strat recurent). Pentru a genera stratul GRU, vom utiliza 128 de unități.

```
output_x = GRU(unități=..., return_sequences = ...)(input_x)
```

- Returnarea secventelor inseamna returnarea outputului la ultimul pas pentru a ne asigura că toate stările ascunse ale GRU sunt trimise spre stratul următor.
- Vom folosi iarasi dropout, folosind o rată de păstrare de 0,8.
- Urmați acest lucru cu normalizarea lotului. Nu trebuie setati parametri.

```
output_x = BatchNormalization()(input_x)
```

Pasul 3: Al doilea strat GRU. Acesta are aceleași specificații ca primul strat GRU.

Pasul 4: Vom crea un strat dens distribuit în timp, după cum urmează:

```
X = TimeDistributed(Dens(1, activare = „sigmoid”))(X)
```

Acest cod creează un strat dens activarea utilizand sigmoid, astfel încât parametrii utilizați pentru stratul dens sunt aceiași pentru fiecare pas de timp. Documentație:

- Documentația Keras despre wrappers (https://keras.io/layers/wrappers/).

- Pentru a afla mai multe, puteți citi această postare de blog Cum se utilizează stratul TimeDistributed în Keras (https://machinelearningmastery.com/timedistributed-layer-for-long-short-term-memory-networks-in-python/).

In continuare vom implementa functia `model()` conform arhitecturii prezentate în Figura 3.

In [27]:

```
def model(input_shape):
    """
    Funcție de creare a arhitecturii modelului în Keras.

    Argument:
    input_shape -- forma datelor de intrare ale modelului (folosind convențiile Keras)

    Returneaza:
    model -- Instanță de model Keras
    """


    X_input = Input(shape = input_shape)

    # Pas 1: Strat convolutional
    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)

    # Pas 2: Primul strat GRU
    X = GRU(units = 128, return_sequences=True)(X)          # GRU (use 128 units and return the sequ
    X = Dropout(0.8)(X)                                     # dropout (use 0.8)
    X = BatchNormalization()(X)                             # Batch normalization

    # Pas 3: Al doilea GRU
    X = GRU(units = 128, return_sequences=True)(X)          # GRU (use 128 units and return the sequ
    X = Dropout(0.8)(X)                                     # dropout (use 0.8)
```

```
    X = BatchNormalization()(X)                                    # Batch normalization
    X = Dropout(0.8)(X)                                            # dropout (use 0.8)

    # Pas 4: Strat dens distribuit in timp
    X = TimeDistributed(Dense(1, activation = "sigmoid"))(X) # time distributed  (sigmoid)

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

    return model
```

In [24]:

```
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import Model, load_model, Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout, Input, Masking, TimeDistributed, LSTM,
from tensorflow.keras.layers import GRU, Bidirectional, BatchNormalization, Reshape
from tensorflow.keras.optimizers import Adam
```

In [28]:

```
Tx = len(x[0]) # numărul de pași de timp introduși în model din spectrogramă
n_freq = len(x[:,0]) # numărul de frecvențe introduse în model la fiecare pas de timp al spectrogramei

model = model(input_shape = (Tx, n_freq))
model.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 101, 5511)]       0
_________________________________________________________________
conv1d (Conv1D)              (None, 22, 196)           16202536
_________________________________________________________________
batch_normalization (BatchNo (None, 22, 196)           784
_________________________________________________________________
activation (Activation)      (None, 22, 196)           0
_________________________________________________________________
dropout (Dropout)            (None, 22, 196)           0
_________________________________________________________________
gru (GRU)                    (None, 22, 128)           125184
_________________________________________________________________
dropout_1 (Dropout)          (None, 22, 128)           0
_________________________________________________________________
batch_normalization_1 (Batch (None, 22, 128)           512
_________________________________________________________________
gru_1 (GRU)                  (None, 22, 128)           99072
_________________________________________________________________
dropout_2 (Dropout)          (None, 22, 128)           0
_________________________________________________________________
batch_normalization_2 (Batch (None, 22, 128)           512
_________________________________________________________________
dropout_3 (Dropout)          (None, 22, 128)           0
_________________________________________________________________
time_distributed (TimeDistri (None, 22, 1)             129
=================================================================
Total params: 16,428,729
Trainable params: 16,427,825
Non-trainable params: 904
_________________________________________________________________
```

Expected Output:

Total params	522,561
Trainable params	521,657

Non-trainable params 904

Ieșirea rețelei este de formă (None, 1375, 1), în timp ce intrarea este (None, 5511, 101). Conv1D a redus numărul de pași de la 5511 la 1375.

2.2 - Antrenarea modelului

- Detectare a cuvintelor trigger durează mult timp pentru a se antrena.
- Pentru a economisi timp, am antrenat deja un model timp de aproximativ 3 ore pe un GPU folosind arhitectura pe care am construit-o mai sus și un set mare de antrenament de aproximativ 4000 de exemple.
- Să încărcăm modelul.

In [29]:
```
1 import tensorflow
2 tensorflow.compat.v1.disable_v2_behavior()
3 model = tensorflow.compat.v1.keras.models.load_model('./models/tr_model.h5')
```

```
WARNING:tensorflow:From /home/ubuntu/tudor-env/lib/python3.7/site-packages/tensorflow/python/compat/v2_compat.py:96: disable_resource_vari
ables (from tensorflow.python.ops.variable_scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

You can train the model further, using the Adam optimizer and binary cross entropy loss, as follows. This will run quickly because we are training just for one epoch and with a small training set of 26 examples.

In [30]:
```
1 opt = Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, decay=0.01)
2 model.compile(loss='binary_crossentropy', optimizer=opt, metrics=["accuracy"])
```

In [47]:

```
X = X.reshape(300,5511,101)
Y = Y.reshape(300,1375,1)
Y.shape
model.fit(X, Y, batch_size = 10, epochs=1)
```

```
Train on 300 samples
300/300 [==============================] - 67s 222ms/sample - loss: 0.1319 - acc: 0.8656
```

Out[47]:

```
<tensorflow.python.keras.callbacks.History at 0x7fc508135890>
```

2.3 - Testam modelul

În cele din urmă, să vedem cum funcționează modelul tău pe setul de testare.

In [48]:

```
X_test = np.load("./XY_test/X_70.npy")
X_test = X_test.reshape(70,5511,101)
Y_test = np.load("./XY_test/Y_70.npy")
Y_test = Y_test.reshape(70,1375,1)


loss, accuracy = model.evaluate(X_test, Y_test)
print("Accuratete pe setul de testare", accuracy)
```

```
WARNING:tensorflow:From /home/ubuntu/tudor-env/lib/python3.7/site-packages/tensorflow/python/keras/engine/training_v1.py:2048: Model.state
_updates (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatically.
Accuratete pe setul de testare 0.88343894
```

Arată destul de bine!

Cu toate acestea, acuratețea nu este o măsură bună pentru această sarcină Deoarece etichetele sunt puternic denaturate la 0, o rețea neuronală care scoate doar 0 ar obține o precizie puțin peste 90%. Am putea defini valori mai utile, cum ar fi scorul F1 sau Precizie/Reamintire. Să nu ne deranjam aici, ci doar să vedem empiric cum se descurcă modelul cu unele predicții.

3 - Efectuarea de predicții

Acum că am construit un model pentru detectarea cuvintelor trigger, să-l folosim pentru a face predicții. Acest fragment de cod rulează audio (salvat într-un fișier wav) prin rețea.

In [50]:

```
def detect_triggerword(filename):
    plt.subplot(2, 1, 1)

    x = graph_spectrogram(filename)
    x  = x.swapaxes(0,1)
    x = np.expand_dims(x, axis=0)
    predictions = model.predict(x)

    plt.subplot(2, 1, 2)
    plt.plot(predictions[0,:,0])
    plt.ylabel('probabilitate')
    plt.show()
    return predictions
```

In [5]:

```
# model = tensorflow.compat.v1.keras.models.load_model('./models/tr_model.h5')
```

In [54]:

```
IPython.display.Audio("train.wav")
```

Out[54]:

0:00 / 0:10

In [55]:

```
1 filename = "train.wav"
2 prediction = detect_triggerword(filename)
```