Università Politecnica delle Marche

Digital Adaptive Circuits And Learning Systems

Sound Event Detection con la tecnica del "*few-shot learning*"

Matteo Orlandini e Jacopo Pagliuca

Prof. Stefano Squartini
Dott.ssa Michela Cantarini

15 luglio 2021

Indice

1 Introduzione

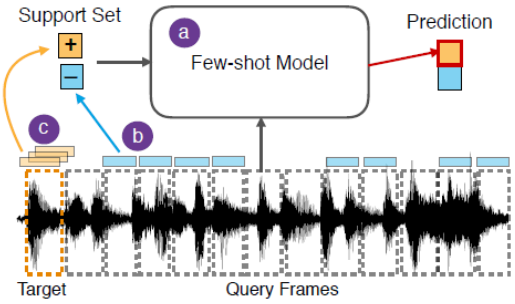


Figura 1: Metodo proposto per il few-shot sound event detection. (a) Applicazione del modello few-shot, (b) costruzione del set di esempi negativi, in blu, e (c) data augmentation per la generazioni di più esempi positivi, in arancione. [4]

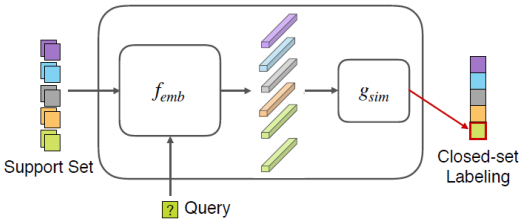


Figura 2: Modello few-shot learning nel caso 5-*way* 2-*shot*. [4]

2 Convolutional Neural Network (CNN)

Una rete neurale convoluzionale è una rete feedforward pensata per applicazioni che richiedono l'elaborazione di immagini o di grandi moli di dati. L'ingresso della rete è costituito da una o più matrici che attraversano delle fasi di convoluzione che ne riducono le dimensioni. I filtri di convoluzione sono detti kernel o di pooling. Infine, il risultato viene passato a una rete fully connected che ha il compito di classificazione.

La prima parte del filtro consiste nella convoluzione dell'immagine in ingresso con una serie di kernel i cui parametri vengono allenati. Il filtro viene traslato in entrambe le dimensioni della matrice producendo una serie di feature bidimensionali. Il parametro *stride* definisce il passo con cui il kernel viene traslato sulla matrice di ingresso. Solitamente, viene effettuato uno zero-padding sui contorni del volume in modo da poter caratterizzare anche i valori che si trovano ai bordi delle matrici. In seguito alla convoluzione viene applicata una funzione non-lineare come ad esempio una sigmoide o una ReLu con lo scopo di aumentare la proprietà di non linearità.

Il pooling layer è uno strato della rete che ha lo scopo di ridurre le dimensioni della matrice prodotta dal convolutional layer. Combinando gruppi di elementi della matrice (solitamente 2x2) restituisce il valore massimo tra essi, che andrà a sostituire il blocco stesso.

3 Area under precision-recall curve (AUPRC)

Nella fase di test, la rete elabora una predizione dell'output a partire da un ingresso noto che poi viene confrontata con il valore effettivo. Nel nostro caso è presente un positive set e un negative set, si tratta quindi di classificazione binaria. Il confronto tra predizione e label può produrre quattro risultati:

- Vero Negativo (TN): il valore reale è negativo e il valore predetto è negativo;
- Vero Positivo (TP): il valore reale è positivo e il valore predetto è positivo;
- Falso Negativo (FN): il valore reale è positivo e il valore predetto è negativo;
- Falso Positivo (FP): il valore reale è negativo e il valore predetto è positivo;

Questi valori vanno a comporre la confusion matrix. Nel nostro caso siamo interessati a Precision e Recall.

Il parametro Precision rappresenta quanti tra i casi predetti come positivi sono realmente positivi.

Il parametro Recall indica quanti tra i casi realmente positivi è stato predetto in modo corretto.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

4 Few-shot learning

(https://www.borealisai.com/en/blog/tutorial-2-few-shot-learning-and-meta-learning-i/)

4.1 Introduzione

Gli esseri umani possono riconoscere nuove classi di oggetti partendo da pochissimi esempi. Tuttavia, la maggior parte delle tecniche di machine learning richiedono migliaia di esempi per ottenere prestazioni simili a quelle umane. L'obiettivo del *few-shot learning* è classificare i nuovi dati dopo aver visto solo pochi esempi di training. Nel caso estremo, potrebbe esserci solo un singolo esempio per ogni classe (*one shot learning*). In pratica, il few-shot learning è utile quando è difficile trovare esempi di training (ad es. casi di una malattia rara) o quando il costo dell'etichettatura dei dati è elevato.

L'apprendimento few-shot viene solitamente studiato utilizzando la classificazione *N-way-K-shot*. L'obiettivo è quello di discriminare le N classi composte da K esempi ciascuna. Una tipica dimensione del problema potrebbe essere quella di discriminare tra $N = 10$ classi con solo $K = 5$ campioni ciascuno di training. Non possiamo allenare un classificatore usando metodi convenzionali; qualsiasi algoritmo di classificazione moderno dipenderà da molti più parametri rispetto agli esempi di addestramento e generalizzerà male.

Se i dati non sono sufficienti per ridurre il problema, una possibile soluzione è acquisire esperienza da altri problemi simili. A tal fine, la maggior parte degli approcci caratterizza l'apprendimento a breve termine con un problema di meta-apprendimento.

4.2 The meta learning framework

Nel framework di apprendimento classico, impariamo come classificare dai dati di training e valutiamo i risultati utilizzando i dati di test. Nel quadro del meta-apprendimento, *impariamo come imparare* a classificare in base a una serie di *training task* e valutiamo utilizzando una serie di test task; In altre parole, usiamo un insieme di problemi di classificazione per aiutare a risolvere altri insiemi non correlati.

Qui, ogni attività imita lo scenario few-shot, quindi per la classificazione N-way-K-shot, ogni attività include N classi con K esempi ciascuna. Questi sono noti come support set per il task e vengono utilizzati per apprendere come risolvere il task. Inoltre, esistono ulteriori esempi delle stesse classi, note come set di query, utilizzate per valutare le prestazioni del task corrente. Ogni task può essere completamente unico; potremmo non vedere mai le classi di un task in nessuno degli altri. L'idea è che il sistema veda ripetutamente istanze (task) durante l'addestramento che corrispondono alla struttura dell'attività finale di few-shot, ma che contengono classi diverse.

Ad ogni fase del meta-apprendimento, aggiorniamo i parametri del modello in base ad un training task selezionato casualmente. La funzione di loss è determinata dalle prestazioni di classificazione sul set di query del task, in base alla conoscenza acquisita dal relativo set di supporto. Poiché la rete viene sottoposta ad un compito diverso in ogni fase temporale, deve imparare a discriminare le classi di dati in generale, piuttosto che un particolare sottoinsieme di classi.

Per valutare le prestazioni in pochi colpi, utilizziamo una serie di attività di test. Ciascuno contiene solo classi invisibili che non erano in nessuna delle attività di formazione. Per ciascuno, misuriamo le prestazioni sul set di query in base alla conoscenza del loro set di supporto.

4.3 Approcci al meta-apprendimento

Gli approcci al meta-apprendimento sono diversi e non c'è consenso sull'approccio migliore. Tuttavia, esistono tre famiglie distinte, ognuna delle quali sfrutta un diverso tipo di conoscenza a priori:

- Conoscenze a priori sulla somiglianza: apprendiamo degli embedding nei training task che tendono a separare classi diverse anche quando non sono visibili.
- Conoscenze a priori sull'apprendimento: utilizziamo le conoscenze a priori per vincolare l'algoritmo di apprendimento a scegliere parametri che generalizzino bene da pochi esempi.

- Conoscenza a priori dei dati: sfruttiamo le conoscenze a priori sulla struttura e la variabilità dei dati e questo ci consente di apprendere modelli praticabili da pochi esempi.

5 Prototypical Network

Nella classificazione few-shot per ogni episodio viene fornito un support set di N esempi etichettati $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ dove $\mathbf{x}_i \in \mathbb{R}^D$ rappresenta il vettore D-dimensionale della feature (nel nostro caso spettrogrammi di dimensione 128×51) e $y_i \in \{1, \ldots, K\}$ la rispettiva label. S_k denota il set di esempi etichettati con la classe k.

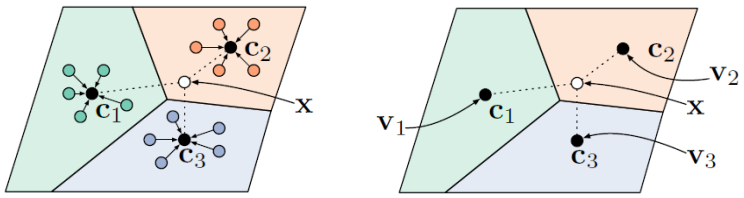
La rete Prototypical calcola una rappresentazione M-dimensionale $\mathbf{c}_k$, o *prototipo*, di ogni classe tramite una funzione di embedding $f_\phi : \mathbb{R}^D \to \mathbb{R}^M$ con parametri da allenare ϕ. La funzione di embedding è rappresentata da una rete convoluzionale. Ogni prototipo è il vettore media tra gli embedding delle istanze della stessa classe.

$$\mathbf{c}_k = \frac{1}{|S_k|} \sum_{(\mathbf{x}_i, y_i) \in S_k} f_\phi(\mathbf{x}_i) \tag{3}$$

Data una funzione distanza $d : \mathbb{R}^M \times \mathbb{R}^M \to [0, +\infty)$, la rete Protypical calcola la relazione di una query $\mathbf{x}$ rispetto ai prototipi tramite la funzione softmax delle distanze prese con segno negativo.

$$p_\phi(y = k|\mathbf{x}) = \frac{\exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_k))}{\sum'_k \exp(-d(f_\phi(\mathbf{x}), \mathbf{c}'_k))} \tag{4}$$

Il processo di training avviene minimizzando il negativo del logaritmo della probabilità $J(\phi) = -\log(p_\phi(y = k|\mathbf{x}))$ considerando la distanza fra query e il prototipo della sua classe. Gli episodi di training sono formati campionando C classi di parole da un lettore e selezionando per ognuna casualmente K istanze e Q query.



(a) Few-shot (b) Zero-shot

Figura 3: Reti Prototypical negli scenari few-shot e zero-shot. (a): i prototipi few-shot $\mathbf{c}_k$ sono calcolati come la media degli embedding del support set per ogni classe. (b): i prototipi zero-shot $\mathbf{c}_k$ sono prodotti facendo l'embedding dei metadati $\mathbf{v}_k$. In entrambi i casi i punti degli embedding delle query sono classificati facendo il softmax sulle distanze del prototipo delle classi: $p_\phi(y = k|\mathbf{x}) \propto \exp\left(-d(f_\phi(\mathbf{x}), \mathbf{c}_k)\right)$. [2]

6 Relation Network

La Relation Network è costituita da due moduli: un modulo di *embedding* f_φ (equivalente a quello nella Prototypical) e un modulo di *relation* g_ϕ, come illustrato in figura 4. Le istanze x_i del query set $\mathcal{Q}$ e quelle x_j del support set $\mathcal{S}$ vengono date in ingresso al modulo di embedding producendo dei vettori (feature maps) $f_\varphi(x_i)$ e $f_\varphi(x_j)$. Questi ultimi vengono poi dati all'operatore $\mathcal{C}(\cdot,\cdot)$ che ne fa la concatenazione: $\mathcal{C}(f_\varphi(x_i), f_\varphi(x_j))$. Le feature map concatenate passano poi attraverso il modulo di decisione che restituisce uno scalare da 0 a 1, il quale rappresenta la somiglianza tra x_i e x_j. Per il caso C-way one-shot, viene concatenata la query con le istanze delle C classi producendo C punteggi di somiglianza.

$$r_{i,j} = g_\phi(\mathcal{C}(f_\varphi(x_i), f_\varphi(x_j))), \qquad i = 1, 2, \ldots, C \tag{5}$$

Nel caso C-way K-shot invece, la query viene concatenata con la somma elemento per elemento degli embedding di ogni istanza delle classi. Quindi, in entrambi i casi i confronti $r_{i,j}$ sono C per ogni query.

Per allenare il modello viene usato l'errore quadratico medio (MSE) in modo che l'uscita del modulo di decisione produca 1 se i vettori concatenati sono della stessa classe e 0 altrimenti.

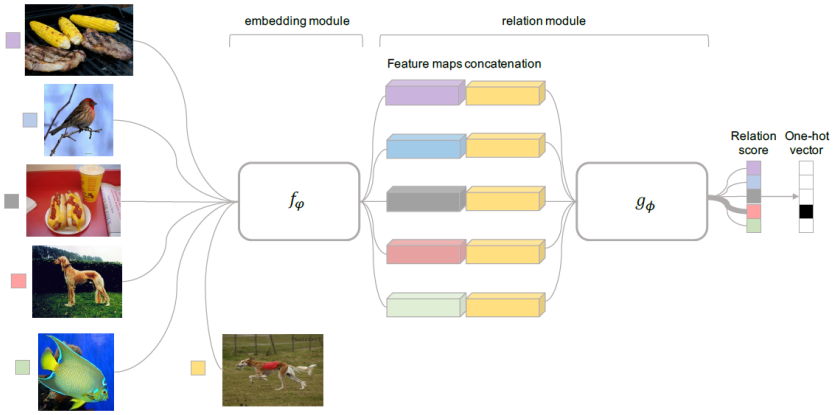
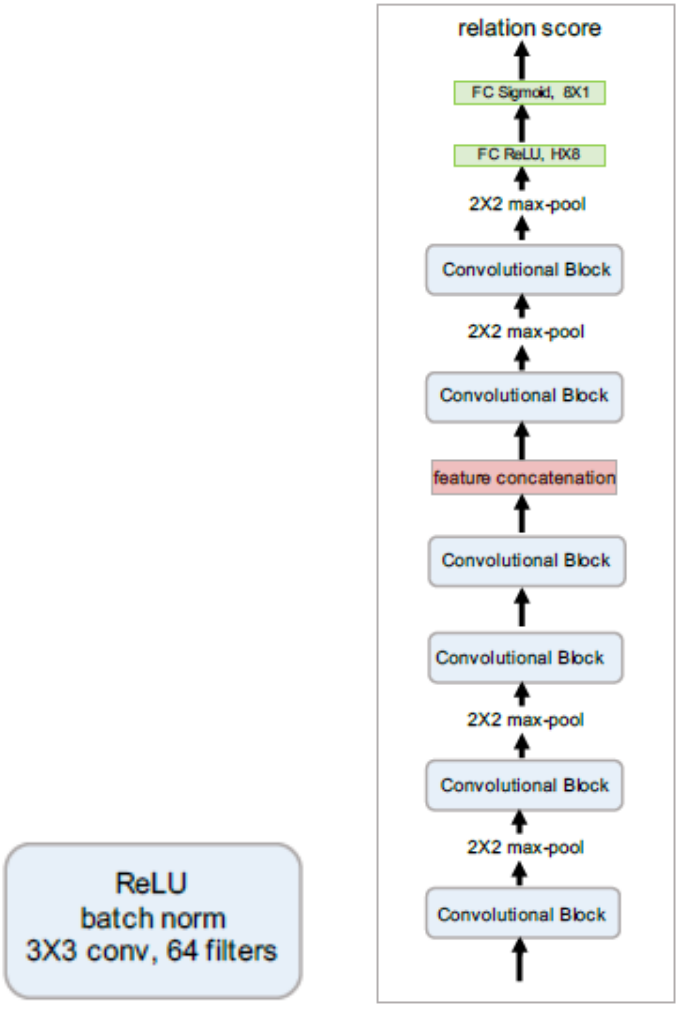


Figura 4: Architettura della Relation Network nel caso 5-*way* 1-*shot* con un esempio di query. [3]



(a) Blocco convoluzionale

(b) Relation Network per few-shot learning

Figura 5: Architettura della Relation Network per few-shot learning (b) composta dagli elementi inclusi nel blocco convoluzionale (a). [3]

7 Dataset Spoken Wikipedia Corpora

Il progetto Spoken Wikipedia unisce lettori volontari di articoli di Wikipedia. Sono disponibili centinaia di articoli in inglese, tedesco e olandese per gli utenti che non sono in grado o non vogliono leggere la versione scritta dell'articolo. Il dataset trasforma i file audio in un corpus, cioè una raccolta ordinata e completa di opere o di autori, allineato nel tempo, rendendolo accessibile per la ricerca.[1]

Questo corpus ha diverse caratteristiche importanti:

- centinaia di ore di audio allineato
- lettori eterogenei
- diversi argomenti
- genere testuale ben studiato
- le annotazioni delle parole possono essere mappate all'html originale
- allineamenti a livello di fonema

Ogni articolo è suddiviso in sezioni, frasi e token. Ogni token è normalizzato e la normalizzazione è allineata all'audio.

7.1 Annotazioni

Ogni annotazione è racchiusa in un tag chiamato "`article`", che contiene una sezione "`meta`" per i metadati, come si può vedere in 1 e una sezione "`d`", come in 2, contenente l'articolo e le relative annotazioni.

```
<article>
  <meta>
    <link key="DC.conformsto" value="http://nats.gitlab.io/swc/schema/swc-1.0.rnc"/>
    <prop key="DC.creator" value="Spoken Wikipedia Corpus Collection Software"/>
    <prop key="DC.publisher" value="Universität Hamburg"/>
    <link key="DC.reference" value="http://nbn-resolving.de/urn:nbn:de:gbv:18-228-7-2209
        "/>
    <prop key="DC.type" value="dataset"/>
    <prop key="DC.license" value="CC-BY-SA"/>
    <prop key="DC.title" value="Limerence"/>
    <prop key="DC.language" value="en"/>
    <prop key="DC.identifier" value="Limerence"/>
    <prop key="DC.date.read" value="2005-04-29 00:00:00"/>
    <link key="DC.source" value="https://en.wikipedia.org/wiki/Limerence"/>
    <prop key="DC.source.wikiID" value="154147"/>
    <prop key="DC.source.revision" value="791627969"/>
    <link key="DC.source.text" value="https://en.wikipedia.org/w/index.php?title=
        Limerence&oldid=13811989"/>
    <link key="DC.source.audio" value="https://upload.wikimedia.org/wikipedia/commons/a/
        aa/Limerence.ogg" group="audio1"/>
    <prop key="DC.source.audio.offset" value="0.0" group="audio1"/>
    <link key="DC.source.audio.page" value="https://en.wikipedia.org/w/index.php?title=
        File%3aLimerence.ogg" group="audio1"/>
    <link key="DC.source.audio.date" value="2007-09-14 19:23:05" group="audio1"/>
    <prop key="reader.name" value="the Epopt"/>
    <prop key="processing.step" value="tokenize" group="tokenize"/>
```

```
    <prop key="processing.step.date" value="2017-08-10T07:44:44.095+02:00[Europe/Berlin]
        " group="tokenize"/>
    <prop key="processing.step.options" value="Namespace(output=articles/Limerence/
        tokenized.swc, all_sections=false, null_normalize=false, raw_output=null,
        subparser_name=tokenize, lang=en, no_introduction=false, article_dir=articles/
        Limerence)" group="tokenize"/>
    <prop key="processing.step.git.commit.id" value="7431
        dbf93f212ad828208abaf8f518fb8de11ff3" group="tokenize"/>
    <prop key="processing.step.git.commit.time" value="09.08.2017 @ 15:21:55 CEST" group
        ="tokenize"/>
    <prop key="processing.step" value="align" group="align"/>
    <prop key="processing.step.date" value="2017-08-11T16:12:18.423+02:00[Europe/Berlin]
        " group="align"/>
    <prop key="processing.step.options" value="Namespace(output=articles/Limerence/
        aligned.swc, transcript=articles/Limerence/tokenized.swc, g2p=../model_en/model.
        fst.ser, phone=false, subparser_name=align, dict=../model_en/empty.dic,
        acoustic_model=../model_en/, audio=articles/Limerence/audio.wav)" group="align"
        />
    <prop key="processing.step.git.commit.id" value="7431
        dbf93f212ad828208abaf8f518fb8de11ff3" group="align"/>
    <prop key="processing.step.git.commit.time" value="09.08.2017 @ 15:21:55 CEST" group
        ="align"/>
  </meta>
```

Codice 1: Metadati delle annotazioni delle parole in un audio

Il documento, contrassegnato dal tag "`d`", può contenere parti diverse, ciascuna spiegata di seguito. La sezione "`extra`" contiene il testo che abbiamo incluso ma non fa parte dell'articolo, "`ignored`" contiene ciò che fa parte del testo ma viene ignorato per l'allineamento, "`section`" contiene un titolo e un contenuto, "`p`" contiene un paragrafo e "`s`" una frase che a sua volta contiene dei token "`t`". In quest'ultimo è contenuta la singola parola originale e le normalizzazioni. Per esempio, la punteggiatura non ha annotazioni di normalizzazione in quanto non è pronunciata, ma il numero 500 ne ha due - "cinque" e "cento". un token stesso non ha allineamento, solo la sua normalizzazione "`n`" è allineata. La normalizzazione ha una "`pronunciation`" e può avere un tempo di "`start`" ed "`end`", se è allineata. La normalizzazione, a sua volta, può contenere dei fonemi "`ph`".

```
<d>
  <extra text="LimerenceFrom wikipedia, the free encyclopedia at e n dot wikipedia dot
      org.">
    <s text="LimerenceFrom wikipedia, the free encyclopedia at e n dot wikipedia dot org
        .">
      <t text="LimerenceFrom">
        <n pronunciation="LimerenceFrom" start="140" end="1190"/>
      </t>
      <t text="wikipedia">
        <n pronunciation="wikipedia" start="1250" end="1950"/>
      </t>
      <t text=","/>
      <t text="the">
        <n pronunciation="the" start="1950" end="2070"/>
      </t>
      <t text="free">
        <n pronunciation="free" start="2070" end="2300"/>
      </t>
      <t text="encyclopedia">
        <n pronunciation="encyclopedia" start="2300" end="3220"/>
      </t>
      <t text="at">
        <n pronunciation="at"/>
      </t>
      <t text="e">
        <n pronunciation="e" start="3490" end="3710"/>
      </t>
```

Codice 2: Annotazioni delle parole in un audio

7.2 Lettori

Il dataset Spoken Wikipedia Corpora contiene un totale di 1340 audio di diversi lettori, ma nel progetto vengono presi solo gli audio che contengono annotazioni a livello di parola. In questo modo vengono presi solo 208 lettori e partizionati come in [4] con un rapporto 138 : 15 : 30 tra lettori di training, validation e test. I lettori e le parole sono state estratte dai file "aligned.swc" contenuti in ogni audio e, successivamente, salvati in diversi file json. Per la gestione dei json sono state create due funzioni utili per la lettura e scrittura come mostrato nel codice 3.

```
import json

def write_json_file(filename, list_of_dict, indent = 0):
  f = open(filename, "w")
  f.write(json.dumps(list_of_dict, indent = indent))
  f.close

def read_json_file(filename):
  f = open(filename, "r")
  list_of_dict = json.load(f)
  f.close
  return list_of_dict
```

Codice 3: json_manager.py

Il codice 4 salva un file json chiamato "readers_paths.json" formato dalle chiavi "reader_name" e "folder" i cui valori sono rispettivamente il nome del lettore e le cartelle in cui sono salvati i file audio registrati dal relativo lettore, come si può vedere nel json 5. Questo codice, usando la libreria `xml.etree.ElementTree` e la funzione `ET.parse`, rappresenta l'intero documento XML come un albero. La funzione `getroot` ne trova la radice e, successivamente, si scorre l'albero iterando finché non si trova il tag "`prop`" in cui è contenuta la chiave "`reader.name`". Inoltre, si effettua un controllo sul nome del lettore perché può capitare che viene salvato nel file "aligned.swc" con nomi diversi. Ad esempio, in alcuni file si può trovare nella chiave "reader.name" il valore ":en:user:alexkillby|alexkillby", mentre in altri solo "alexkillby" oppure si può trovare "user:popularoutcast", mentre in altri solo "popularoutcast". Una volta noto il nome del lettore si vede se è già presente nella lista di dizionari e se è un lettore nuovo lo si aggiunge con il relativo nome del file audio, mentre se il lettore era già presente si aggiunge solamente il nome del file audio.

```
import xml.etree.ElementTree as ET
from tqdm import tqdm
import os
import json
from json_manager import *

# Initialize list with empty dictionaries
```

```
readers = []

source_path = "./Dataset/English spoken wikipedia/english/"
filename = "aligned.swc"

# search for readers for each folder
for audio_path in os.scandir(source_path):
  # save only folder name from entire path
  folder =  os.path.basename(audio_path)
  if (os.path.exists(source_path + "/" + folder + "/" + filename)):
    # parse the xml file "aligned.swc"
    tree = ET.parse(source_path + "/" + folder + "/" + filename)
    # getroot returns the root element for this tree
    root = tree.getroot()
    #root.iter creates a tree iterator with the current element as the root.
    # The iterator iterates over this element and all elements below it, in
    # document (depth first) order.
    for property in root.iter(tag = 'prop'):
      # if the key "reader.name" exists
      if (property.attrib['key'] == 'reader.name'):
        # save the reader name taking the value of the attribute
        reader_name = property.attrib['value'].lower()
        # fix readers names that contain "user:"
        if ("user:" in reader_name):
        # fix readers names that contain "|""
          if ("|" in reader_name):
            # example reader_name = [[:en:user:alexkillby|alexkillby]] ->
            # reader_name = alexkillby
            reader_name = reader_name[reader_name.find("user:")+5:
                reader_name.find("|")]
            # fix readers names that contain "|""
          elif ("]]" in reader_name):
            # example reader_name = [[user:popularoutcast]] ->
            # reader_name = popularoutcast
            reader_name = reader_name[reader_name.find("user:")+5:
                reader_name.find("]]")]
        # if the reader is not yet on the list create a dict and append to
        # the readers list
        if not any(reader['reader_name'] == reader_name for reader in
            readers):
          dictionary = {'reader_name': reader_name, 'folder' : [folder]}
          readers.append(dictionary)
        else:
          # if the reader is already on the list add the folder name
          for reader in readers:
            if (reader['reader_name'] == reader_name):
              reader['folder'].append(folder)
# print the number of the readers
print("The readers are:", str(len(readers)))

# save a "readers_paths.json" with the name of the readers and the relative
# file audio folders
write_json_file("readers_paths.json", readers, indent = 4)
```

Codice 4: xml_parser_readers.py

```
{
    "reader_name": "the epopt",
    "folder": [
        "(I_Can%27t_Get_No)_Satisfaction",
        "Ceremonial_ship_launching",
```

```
            "Limerence",
            "Revolt_of_the_Admirals",
            "Ship_commissioning"
        ]
    },
    {
        "reader_name": "wodup",
        "folder": [
            "0.999..%2e",
            "Execution_by_elephant",
            "Hell_Is_Other_Robots",
            "Tom_Bosley",
            "Truthiness"
        ]
    },
```


Codice 5: Formato JSON dei lettori

7.3 Parole


```
[
    {
        "word": "i",
        "frequency": 12,
        "start": [
            660,
            8800,
            115050,
            116300,
            117910,
            228560,
            273900,
            497150,
            518270,
            534740,
            543420,
            589280
        ],
        "end": [
            870,
            8940,
            115240,
            116360,
            118020,
            228690,
            274000,
            497340,
            518520,
            534860,
            543700,
            589360
        ]
    },
```


Codice 6: Formato JSON delle parole allineate

Riferimenti bibliografici

[1] Arne Köhn, Florian Stegen e Timo Baumann. «Mining the Spoken Wikipedia for Speech Data and Beyond». Inglese. In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)* (23–28 mag. 2016). A cura di Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk e Stelios Piperidis. Portorož, Slovenia: European Language Resources Association (ELRA). ISBN: 978-2-9517408-9-1.

[2] Jake Snell, Kevin Swersky e Richard S. Zemel. *Prototypical Networks for Few-shot Learning*. 2017. arXiv: 1703.05175 [cs.LG].

[3] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H. S. Torr e Timothy M. Hospedales. «Learning to Compare: Relation Network for Few-Shot Learning». In: *CoRR* abs/1711.06025 (2017). arXiv: 1711.06025. URL: http://arxiv.org/abs/1711.06025.

[4] Yu Wang, Justin Salamon, Nicholas J. Bryan e Juan Pablo Bello. *Few-Shot Sound Event Detection*. 2020, pp. 81–85. DOI: 10.1109/ICASSP40776.2020.9054708.