Faculty of Mathematics and Informatics

VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
MODELLING AND DATA ANALYSIS
MASTER'S STUDY PROGRAMME

ACCENT IDENTIFICATION USING MACHINE LEARNING

Master's thesis

Author: Justina Grigaliūnaitė
VU email address: justina.grigaliunaite@mif.stud.vu.lt
Supervisor: Dr. Gerda Ana Melnik-Leroy

Vilnius

2022

Contents

Abstract

Accent identification can offer performance improvements for many Automatic Speech Recognition (ASR) systems. This thesis focuses on phoneme-based accent identification and proposes a classification approach for two speaker groups: native French and American speakers, detecting accent in pronounced French words and cut out /u/-/y/ sounds. The classification is carried out on a new psycholinguistic dataset derived from experimental research. Our developed approach could be used as a pre-processing step in complex ASR systems. Moreover, this issue is of great relevance for the development of methodology in linguistics and cognitive psychology as they lack objective ways to efficiently identify and assess foreign accent. Underlying idea for class difference is based on linguistic research and the fact that it is extremely difficult to reproduce some language-specific sounds for non-native speakers which results in foreign accent.

Scientific experiment presented in the thesis consists of several parts: classification of vowel and word samples represented as spectrograms using Residual Neural Network (ResNet), comparison with baseline Support Vector Machine (SVM) model employing Mel-frequency Cepstral Coefficients (MFCCs) and suggested SVM classification improvement using spectrogram data. ResNet classification results show up to 81.6% accuracy for accent identification from vowel samples and up to 93.06% accuracy using word samples. The results are significantly better than those produced by a baseline model of 66.46% and 67.34% accuracy for vowel and word respectively. In addition, the thesis suggests possible performance improvement for the baseline SVM model by using spectrogram representation of a signal instead of widely used MFCCs.

Thesis provides introductory analysis of interdisciplinary accent identification topic which requires linguistic basics, sound signal processing and supervised learning knowledge. The results of this study shows the suitability of 2D spectrogram representation and ResNet model for sound data classification. It also raises some further questions that can be researched in order to improve accent identification even more. In addition to this, the thesis succeeded in solving an accent classification problem, that traditional acoustic methods used by psycholinguists failed to tackle. This indicates that Machine Learning based methods can improve research methodology in psycholinguistics and other fundamental research fields.

Keywords: Accent Identification, Spectrogram, ResNet, Support Vector Machine

Santrauka

Akcento identifikavimas gali ženkliai pagerinti automatinio kalbos atpažinimo sistemų pasiekiamus rezultatus. Šis darbas nagrinėja foneminiu pagrindu paremtą akcento atpažinimą ir pateikia klasifikavimo metodą, skirtą diferencijuoti dvi kalbėtojų grupes: gimtąja kalba šnekančius prancūzus bei prancūziškai šnekančius amerikiečius. Klasifikavimui naudojami atskiri prancūziški žodžiai bei iškirpti balsiai /u/-/y/. Šie duomenys gauti psicholingvistinio eksperimento metu ir niekada nebuvo naudojami automatiniam klasifikavimui. Sukurtas metodas galėtų būti taikomas kaip išankstinio apdorojimo žingsnis kompleksiškose automatinio kalbos atpažinimo sistemose. Be to, ši akcento identifikavimo problema yra svarbi lingvistinės ir kognityvinės psichologijos metodologijos vystymui, kadangi šiuo metu naudojamiems metodams trūksta objektyvių būdų atpažinti kalbos akcentą. Pamatinė grupių skirtingumo įdėja paremta lingvistiniais tyrimais ir tuo, jog kitakalbiams yra be galo sunku atkartoti garsus, kurių nėra jų gimtojoje kalboje, todėl atsiranda kalbėtojo akcentas.

Darbe pateiktas mokslinis eksperimentas susideda iš kelių dalių: balsio ir žodžio, atvaizduoto spektrograma, klasifikavimas naudojant liekamuosius neuroninius tinklus (ResNet), palyginimas su baziniu atraminių vektorių klasifikatoriaus modeliu pritaikant melų skalės kepstro koeficientus bei pasiūlytas atraminių vektorių klasifikatoriaus metodo pagerinimas naudojant spektrogramas. Rezultatai parodo ResNet modelio pasiekiamą tikslumą iki 81.6% klasifikuojant balsius bei iki 93.06% tikslumą klasifikuojant žodžius ir taip identifikuojant kalbėtojo akcentą. ResNet rezultatai yra ženkliai geresni nei gaunami su baziniu metodu, kuris pasiekia 66.46% ir 67.34% tikslumą balsiui ir žodžiui, atitinkamai. Be to, yra pasiūlomas bazinio metodo pagerinimas naudojant spektrogramas vietoje melų skalės kepstro koeficientų, kurie yra dažniausiai naudojami panašiose užduotyse.

Taip pat yra trumpai pristatoma trumpa tarpdisciplininė analizė, reikalinga pilnai suprasti akcento identifikavimo užduotį. Ši analizė įtraukia lingvistikos pagrindus, garsų apdorojimo bei prižiūrimo (mašininio) mokymosi žinias. Darbo rezultatai parodo, jog dviejų dimensijų spektrogramos yra tinkamas kalbos signalų atvaizdavimo metodas bei liekamųjų neuroninių tinklų pritaikomumą šių garsų klasifikavimui. Be to, yra keliami papildomi klausimai, kurie galėtų padėti patobulinti automatinį akcento identifikavimą ateityje. Atliktas darbas sėkmingai išsprendė akcento klasifikavimo užduotį, kurios nepavyko išspręsti tradiciniais akustiniais metodais, kuriuos naudoja psicholingvistikos atstovai. Tai parodo, jog mašininiu mokymusi paremti metodai gali pagerinti psicholingvistikos ir kitų fundamentaliųjų mokslų tyrimus.

Raktiniai žodžiai: akcento identifikavimas, spekrograma, ResNet, atraminių vektorių klasifikatorius

List of abbreviations

ASR - Automatic Speech Recognition

AUC - Area Under the Curve

CNN - Convolutional Neural Network

DFT - Discrete Fourier Transform

DNN - Deep Neural Network

FFT - Fast Fourier Transform

FN - False Negative

FPR - False Positive Rate

FP - False Positive

GMM - Gaussian Mixture Model

HMM - Hidden Markov Model

LID - Automatic Language Identification

MCC - Matthews Correlation Coefficient

MFCC - Mel-frequency Cepstral Coefficient

ML - Machine Learning

RNN - Recurrent Neural Network

ROC - Receiver Operating Characteristics

ResNet - Residual Neural Network

SGD - Stochastic Gradient Descent

STFT - Short-time Fourier transform

SVM - Support Vector Machine

TNR - True Negatie Rate

TN - True Negative

TPR - True Positive Rate

TP - True Positive

Introduction

Automatic Speech Recognition (ASR) is an important topic in technology today given the increased interest in smart appliances that respond to human voice commands. First step in any ASR is to identify the language. Therefore, much research focuses on Automatic Language Identification (LID) task which identifies a language from random spoken utterance. However, it is not sufficient to identify the language. Identifying idiosyncratic differences in speech production is also important for improving the robustness of existing speech analysis systems such as speech recognition, speaker identification, voice conversion, etc. For example, ASR systems evaluated on foreign accented speech shows lower performance compared to the speech of native speakers. These systems can be modified by developing pre-processing algorithms that identify the accent of a speaker and then customizing the recognition algorithm to the particular accent [7].

This thesis focuses on phoneme-based accent identification. More specifically, classifying two speaker groups: native French and American speakers pronouncing French words. Even though it is quite narrow topic in terms of accent identification it employs modern methods for classification and sound data representation. Moreover, this issue is of great relevance for the development of methodology in such fields as psycholinguistics, cognitive psychology and neuroscience. Specifically, these fields lack objective ways to efficiently identify and assess foreign accent [11]. The dataset used in this thesis was previously collected and examined using traditional acoustic measurements by a team of psycholinguists, who did not succeed in accurately classifying the sounds according to their spoken accent. Thus, machine learning-based methods have a potential to solve this problem.

Previously, acoustic properties of a speech signal were mostly modelled using statistical techniques such as Hidden Markov Model (HMM) and Gaussian Mixture Model (GMM). However, those require numerical representation of a signal, deliberate selection of parameters, appropriate choice of acoustic analysis and supervised learning. In more recent works we notice Machine Learning (ML) based models like Deep Neural Network (DNN) or Convolutional Neural Network (CNN) gaining more popularity in the field. They are much more adaptable given assumptions on non-linear speech signal properties. Application of neural network based architectures led to major breakthrough in speech analysis.

This research focuses on evaluating the performance of special case of CNN - Residual Neural Network (ResNet) for speech signal classification on a new psycholinguistic dataset derived from experimental research. It is known problem that deeper neural networks are more difficult to train, therefore, for performance optimisation residual learning framework can be used to ease the training of networks [5]. ResNet has an advantage compared to CNN because it overcame the "vanishing gradient" problem. That allowed to construct networks with up to thousands of convolutional layers, which outperform shallower networks. Classifier described in this thesis is based on pre-trained ResNet-18 model that was implemented in Torchvision library which is part of the PyTorch project and fine tuned for new dataset.

Datasets based on sound recordings require particular rigorous pre-processing before ML techniques can be applied to them. First step is analysing acoustic parameters that are time domain descriptors such as pitch, formants, loudness, energy and frequency domain parameters that are most often derived from the Fourier transform and describe the shape of the spectrum [8]. Classical approaches usually

require 1D input, thus one of the most popular methods is calculating Mel-frequency Cepstral Coefficients (MFCCs) and passing it to the model. Speech signal can be also represented as 2D time-variant feature spaces that correspond to audio features of speech signals. Then we can employ ML models that show good performance in image classification to classify audio signals.

In this thesis, speech signals were represented as 2D mel-scaled spectrograms. They were preprocessed and normalised to be all of the same length. Model was trained on two types of data: cut out vowels /u/ and /y/ that are very distinct to French language and whole words. Obtained results were compared with classical method - Support Vector Machine (SVM) predictions as well as comparing both models' performance on cut out vowels and whole words.

The aim of this thesis is to evaluate application of ResNet model on 2D spectrogram data to classify native French and American speakers based on differences in their pronunciation of French. Thesis consists of literature review, analytical part and scientific experiment. The first part explores state of the art methods for sound representation, sound classification and accent identification. The second part shows in depth analysis of why chosen methods are selected for particular problem and methodology behind them. Third part describes the experiment on new dataset applying ResNet model for sound classification and comparing results to chosen benchmark method. Finally, conclusion summarises the most important results and mentions which parts of research can be improved.

1. Literature review

There are number of studies dedicated to topics in ASR while accent identification is relatively less researched field. Nevertheless, both of those topics employ similar methods for audio data pre-processing, feature extraction and classification tasks. Neural networks have been investigated exhaustively in the area of language identification but there are much fewer studies evaluating their performance in accent identification. In this section we will review most popular approaches in ASR and accent identification as well as discuss the best practices for audio data pre-processing and representation.

There are multiple ways to improve ASR systems for accented speech such as introducing pre-processing algorithms or ent-to-end training which allows multiple parameters to be trained jointly, rather than step by step. The example of the latter is presented in the article [18]. Authors explored multi-task training and accent embedding in the context of end-to-end ASR. Many current techniques mainly focus on neural networks and are sensitive to the generalization potential of the data. Usually training dataset does not include various accents which means that a system may operate well for seen accents and become unusable for unseen accents. Authors use 40-dimensional MFCCs as input features and combination of convolutional and recurrent layers. Then ReLU activation function is used which is known to perform well in the context of speech recognition. The final layer uses a softmax activation function that produces a probability distribution over the set of characters. During the backpropagation Adam optimization algorithm was used. It is an efficient algorithm based on the Stochastic Gradient Descent (SGD) and has the advantage to use an evolving learning rate, also known to be robust and fast. The authors hypothesized that by modelling jointly the accent class and the sequence of symbols in the utterance, the model will get more robust to unseen accents. The results showed significant increase in model accuracy comparing to the baseline trained with conventional time-delay neural networks.

Differences in accent are due to both prosodic and articulation characteristics. Therefore, [6] proposed to take advantage of both long-term and short-term features. Individual speech samples were processed into multiple speech segments of equal length. For each segment DNN was trained on long-term statistical features (mean, standard deviation, kurtosis of MFCC, RASTA, etc.) and Recurrent Neural Network (RNN) trained on short-term acoustic features (39th-order mel-scale filterbank features with logarithmic compression). The final prediction was obtained by linearly fusing the results from both neural networks. The performance of the proposed system greatly surpassed the provided baseline system built with SVM. Article was based on the INTERSPEECH 2016 Native Language Sub-Challenge which goal was to identify the corresponding native language from the accented speech. The challenge was particularly difficult for two reasons: all of the speech samples were recorded with babel background noise and a large number of the speakers were not perfectly fluent in English. Which, in addition to accent differences, led to number of pauses and linguistic fillers in the speech. However, such pauses might be picked up by the model and help to classify native and accented speech.

One of the popular ways to represent audio data is strictly based on acoustical features and signal is converted to MFCCs. Article [2] presented CNN application to differentiate between the Arabic, Korean, Italian, and Japanese accents in spoken English. Data was first preprocessed into uniform

30 second length audio files by padding (via duplication) or trimming. To reduce number of features (frequencies) MFCCs were extracted from each file to represent the short-term spectrum of sounds. This process involved breaking down the audio clip into several windows and extracting the frequency information in each window. The number of final features (13) was selected based on the number of frequencies corresponding to the human vocal range. Later those features were used to train CNN model. The authors experimented with various number of convolutional/pooling layers but 2-layer CNN showed the best balance between accuracy and model complexity. The same ratio for each language was maintained within the test and training sets which was necessary to evaluate accuracy of the model for each accent.

Article [9] provided an evaluation of suitability of 2D audio signal feature maps for speech recognition based on CNN, which is a class of deep, feed-forward neural network. The authors analysed audio signal feature maps, namely spectrograms, chromagrams, linear and mel-scale cepstrograms. The choice to use such audio data representation was made upon the fact that CNNs perform well in 2D data-oriented processing contexts. This approach was assessed on Lithuanian word recognition task. Chosen CNN architecture included convolution and pooling layers that detect specific patterns in the analysed data and downsample it for the next convolution-pooling layer block. At the end of the network there is fully connected layer, which maps the extracted features to an n-dimensional vector which is used for making decision on class. In the article authors also claimed that wavelet-adapted time-frequency and cepstral representations have successfully been applied to many speech related issues with spectrograms and mel-scale cepstrograms being among the most popular choices. This might be due to the fact that signal representations with low energy level can capture the core structure of a signal and can improve the recognition accuracy. Based on this research using mel-scale spectrograms and CNN seems like an appropriate choice to solve a task described in this thesis.

Implementation of hybrid Convolutional Recurrent Neural Network (CRNN) that operates on spectrogram images of the provided audio snippets was presented in [1]. The goal of the experiment was to classify the spoken language from a given audio sample that is converted to an image. Extensive experiments showed, that suggested model is applicable to a range of noisy scenarios and can easily be extended to previously unknown languages, while maintaining its classification accuracy. This task is part of LID systems that usually are first step for many spoken language processing tasks such as ASR. This field is also closely related to the accent identification since similar methodology and reasoning can be applied to both.

When evaluating the performance of the model it is important to compare it with the baseline method. Article [8] focused on finding a way of highlighting the acoustic differences between consonant phonemes of the Polish and Lithuanian languages. This was achieved by using similarity matrices based on speech acoustic parameters combined with a CNN. The results were compared with two baseline classifiers: k-nearest neighbours and SVM, both classifying audio data represented as MFCCs. In the experiments authors computed the performance scores of proposed methods for Lithuanian, Polish, and English consonant classification. Since consonants contain language-related information and they are necessary to identify lexical meaning it is important to recognise and highlight language specific differences. For the experiment, data was represented as similarity matrices an as spectrograms obtained from the same dataset. Resulting images are derived from the acoustic characteristics of the

speech signals (e.g. the location of the formant frequencies, articulation, voicing). The experiment showed that DNN performs better with more extensive data from which it extracts useful information automatically. It also showed that combining spectrograms and CNN performs reasonably well for classification of consonants and achieving high level of accuracy.

Another important part of successful accent identification is selecting appropriate type of data based on linguistic characteristics. Many popular approaches in speech signal classification include using single sounds (such as consonants [8] or vowels) or whole words as model inputs. Article [14] showed an example of using whole English words for automated classification of words containing aspirated and unaspirated allophones produced by Polish L2 speakers. Different signal representations, such as low-level audio features and spectrograms were tested in the context of the detection of aspiration. Employing whole words represented in such way returned satisfying results when classified with CNN model. The approach also used zero-padding procedure to unify the length of all data inputs which is required for CNN (in this case, specifically, to the length of the longest recording). Various types of frame lengths and the overlap factors employed in spectrogram calculation were checked for the purpose of hyper-parameter selection. Finally, spectrogram values were scaled to interval [0, 1]. Article [15] focused on classifying allophones /p, t, k/ that were extracted from whole English words to aspirated and non-aspirated employing spectrogram representation of a speech signal. CNN model showed good results with 84% accuracy.

2. Analytical part

In this section we will describe methodological basis that were chosen for the scientific experiment and explain why they are fit to solve the problem at hand.

Linguistic basis

To understand how to work with human language data it is important to know how people produce language and what are the differences between learning your mother tongue versus learning a foreign language later in life. According to the book [4] there are two branches of linguistics that analyses speech sounds: phonetics and phonology. Phonetics studies speech sounds from three viewpoints:

- Production - focuses on how sounds are made in the human vocal tract;
- Acoustics - studies the speech waveforms which are transmitted through the atmosphere;
- Perception - examines how the incoming acoustic signal is processed to detect the sound sequence originally intended by the speaker.

Phonology involves more formal analysis and abstract theorizing. It aims to understand the implicit system of rules that the speaker uses in perceiving and manipulating the sounds. Phonological theory rests on phonetic data - observations of the phonetic form of utterances.

Thousands of audibly distinct sounds can be produced by the human vocal tract. It is common in almost all languages to segregate sounds into vowels and consonants. Vowels are made with a relatively open vocal tract and therefore are highly sonorous sounds. Consonants involve some constriction in the vocal tract and are quieter than vowels. They often are detectable not so much by their own sound as by the transitional acoustic events that occur at the boundaries of consonants and vowels. For both types of sounds phonetic description involves assigning a phonetic symbol to each sound.

This thesis mostly focuses on vowels, which differ from consonants in that they do not have "places of articulation" (points of major constriction in the vocal tract). It is well known that vowels carry on an essential part of the speech and language characteristics. When pronouncing a vowel the vocal tract acts as a resonating chamber. Its shape can be modified by using movements of the lips, tongue and jaw. That causes different timbres in the basic sound produced at the vocal cords. Vowels are described by specifying modification that one can apply to shape the vocal tract.

The basic unit of phonology is phoneme which represents the smallest unit of speech sound. It may cause a change of meaning within a language but does not have the meaning of its own. In most studies the uttered signal is represented by phonemes. However, same phonemes may have different articulation. To detect such differences ML methods can be applied [8].

Accents

According to [2] an accent is defined to be a distinct mode of pronunciation of a language, especially one associated with a particular locality, nation or social class. Usually it comes from the articulation habits of the speaker in his/her own native language. When learning a foreign language, the speaker has to learn a modification in the patterns of intonation, lexical stress and grammar together with

the use of additional distinctive phonemes. Such modification leads to both acoustic and articulation differences [7].

Linguistic methodology

It is difficult to objectively evaluate speech acoustical characteristics, therefore, accent identification problem raises some methodological challenges. Researchers use two types of approach: acoustic and perceptual methods. Perceptual methods require professional to evaluate records which requires a lot of time and lacks objectivity. Moreover, individual sounds cut out from recordings are often too short for specialists to evaluate. Including the sounds' context may increase bias in the judges' evaluation when assessing the global accent of the L2 speaker. While acoustic measurements might seem more objective, the definition of what is to be measured (distinctiveness of L2 sounds vs. differences between L1 and L2 productions) or the choice of a measure (Euclidian vs. Mahalanobis) is not straightforward either. Finally, foreign accent often encompasses a mixture of acoustic cues, which can hardly be measured in isolation. Another approach is more data driven and can be used in algorithms - acoustic methods focus on audio signal measurements. However, it is difficult obtain all the measurements and differentiate which ones are the most definitive. In this study we explore possibilities of employing ML to bypass these issues. Recent research showed encouraging results in the automated classification and/or evaluation of consonant pronunciation [14], [8].

The current study focuses on vowels and uses a supervised ML algorithm (CNN) to re-analyse data from a previous study. The dataset consisted of recordings of the French contrast /u/-/y/ pronounced by American L2 speakers and by French natives (this contrast has been shown to be difficult for English learners. Article [11] focused on the French vowel /u/–/y/ contrast and tested proficient English-speaking L2 learners of French. The research aimed at obtaining more precise and comparable measures for perception and production accuracy. It showed that the native French participants produced significantly shorter tokens of /u/ and /y/ compared to the late learners. Native speakers pronounce words much faster and this means that the distance between those vowels in French productions was shorter, as vowels typically became more central at increased speech rates.

Audio data processing

Probably most of the human speech research done today is based on digital sound representation. Therefore, it is important to understand how the signal is represented and how can it be processed to in a meaningful way.

Audio signal

In general, a signal is a variation in certain quantity over time. Audio signal is the varying quantity of air pressure. We can take samples of the air pressure over time to capture this information digitally. The total number of waves produced in one second is called the frequency of the wave. The number of vibrations counted per second is called frequency. Captured waveform for the signal can be interpreted, modified, and analysed with computer software.

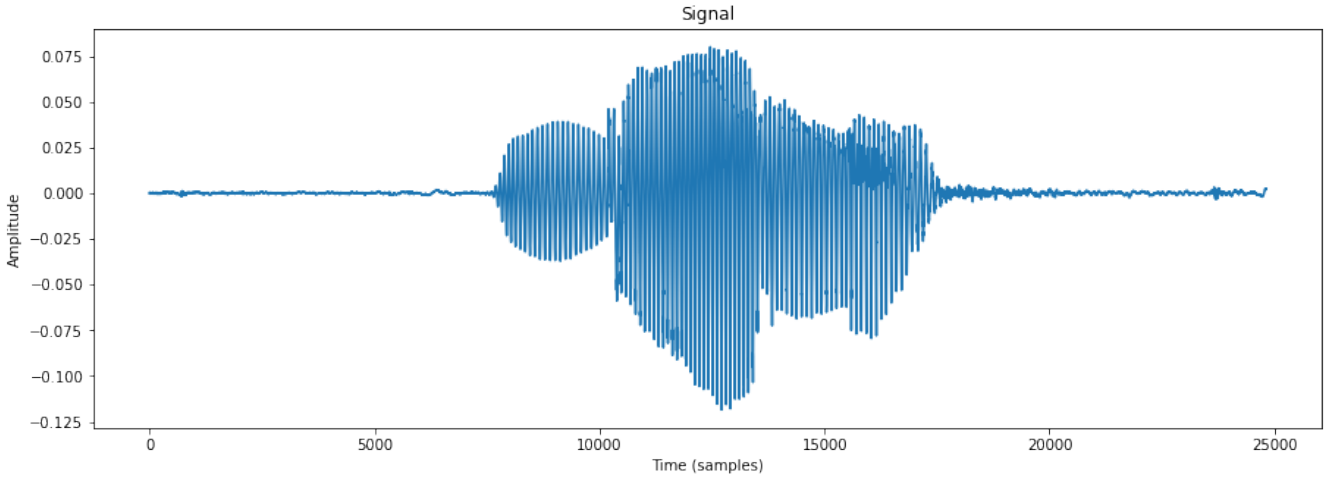


Figure 1: Single word "bulle" (eng. bubble) represented as a sound wave.

Fourier Transform

Audio signal can be represented in frequency domain. Here parameters are most often derived from the Fourier transform and describe the shape of the spectrum. The Fourier transform is a mathematical formula that allows to decompose a signal into it's individual frequencies and the frequency's amplitude. It converts the signal from the time domain into the frequency domain. The result is called a spectrum.

For digital (and discrete) application it is possible to use Fast Fourier Transform (FFT) that can efficiently compute the Discrete Fourier Transform (DFT). It is widely used in signal processing. However, in speech signal analysis it is important to examine how signal changes over time. For that purpose we can use Short-Time Fourier Transform (STFT) algorithm which represents a signal in the time-frequency domain by computing DFTs over short overlapping windows, whereas the standard Fourier transform provides the frequency information averaged over the entire signal time interval.

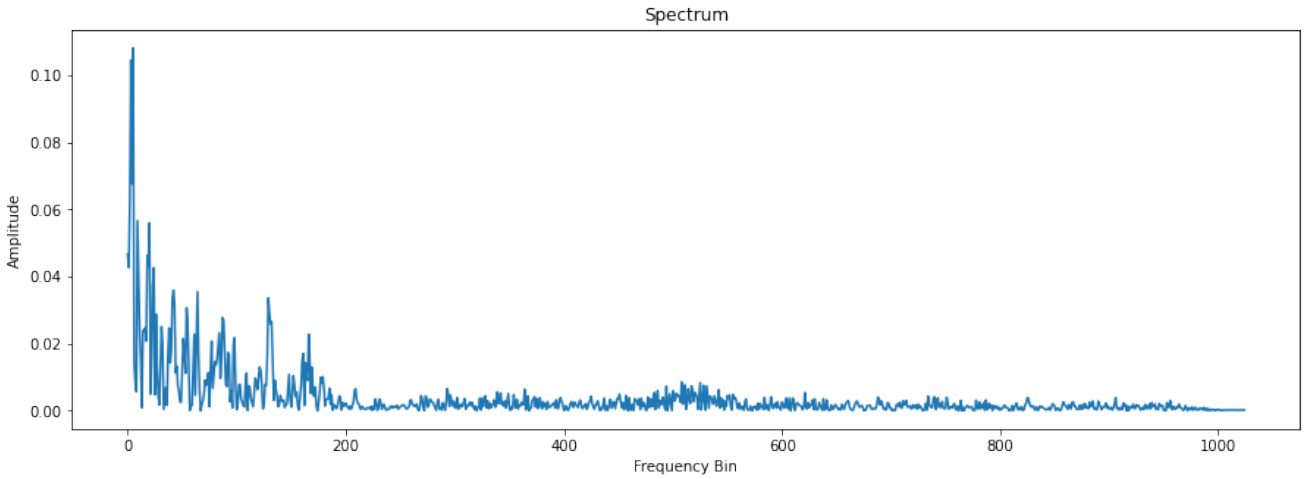


Figure 2: Sound wave of a word "bulle" after STFT.

Spectrograms

Speech signals are non-periodic signals. To represent the spectrum of these signals as they vary over time we can compute several spectrums by performing FFT on several overlapping windowed segments of the signal which is known as STFT. Obtained result is called the spectrogram. It, essentially, is a heat-map where the X-axis represents the time, Y-axis represents the frequency, with lower frequency at the bottom. The colour determines the amplitude of how loud the sound is. It is a way to visually represent a signal's loudness or amplitude, as it varies over time at different frequencies.

Mel-scale

Studies have shown that humans do not perceive frequencies on a linear scale. We are better at detecting differences in lower frequencies than higher frequencies. For example, we can easily tell the difference between 500 Hz and 1000 Hz, but we will hardly be able to tell a difference between 10,000 Hz and 10,500 Hz, even though the distance between the two pairs is the same. In 1937, Stevens, Volkmann, and Newmann proposed a unit of pitch such that equal distances in pitch sounded equally distant to the listener. This is called the mel-scale. A spectrogram where the frequencies are converted to the mel-scale is called a Mel-spectrogram. Widely used formula for converting frequency (f) to mel-scale can be written as follows:

$$M(f) = 2595 \log_{10}(1 + \frac{f}{700}).$$

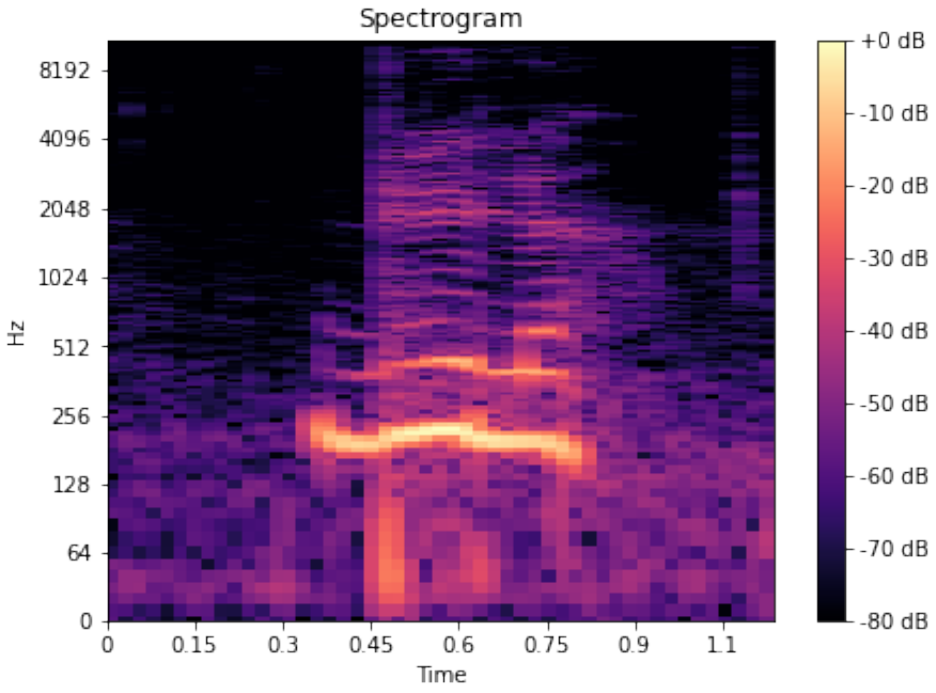


Figure 3: Spectrogram of a word "bulle".

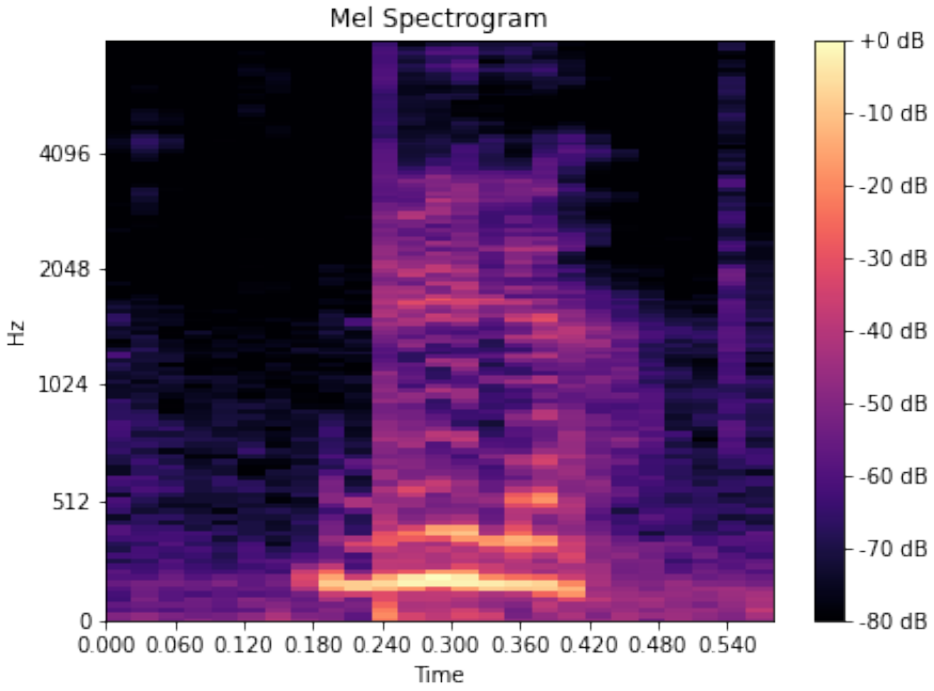


Figure 4: Mel-spectrogram of a word "bulle".

Mel-frequency Cepstral Coefficients

Mel-frequency Cepstral Coefficients can be a more compressible way to represent audio data. They are calculated by computing discrete cosine transformation on Mel-spectrograms, thus uses the mel-scale. MFCCs are widely used spectral feature extraction method [3], especially for classical models like GMM or SVM due to low number of coefficients or for neural networks.

Classical methods for classification

There is a broad set of methods that are considered ML methods. Classical algorithms include: SVM, nearest neighbour, decision tree, naive Bayes classifier and others. In this study we only consider SVM since it is a classical, widely-used method and is suitable for benchmarking in speech signal classification tasks.

Support Vector Machine

Support Vector Machine (SVM) is a supervised learning technique which is a classical, simpler and older method that can be applied for classification, regression and outliers detection tasks. It can solve linear and non-linear problems which works well for many practical examples.

The idea behind SVM classification [17] is simple: the algorithm constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space which separates the data into classes. A good separation is achieved by the hyper-plane that has the largest distance to the nearest training data point of any class. The distance from the decision surface to the closest data point determines the

margin of the classifier. The larger the margin - the lower the generalization error of the classifier. Data samples on the margin boundaries, called "support vectors".

In case of the binary SVM classification, given a training set of input and label pairs $(x_i, y_i),\ i = 1, ..., l$, where $x_i \in \mathbb{R}^n$ and $y \in \{1, -1\}^l$ we want to find the "maximum-margin hyper-plane" that divides the group of points x_i into two classes and maximises the distance between the hyperplane and the nearest point x_i from either group. Let's denote the hyper-plane in question as $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$ such that the prediction given by $sign(w^T\phi(x) + b)$ is correct for most samples, here b is a bias term and $\phi(x)$ denotes a fixed feature-space transformation. SVM classification requires the solution of the following optimization problem:

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{l} \zeta_i,\ \textit{subject to}\ y_i(w^T\phi(x_i) + b) \geq 1 - \zeta_i,\ \zeta_i \geq 0,\ i = 1, ..., l.$$

The method tries to maximize the margin by minimizing $\|w\|^2 = w^T w)$, while having a penalty when a sample is misclassified or within the margin boundary. ζ_i is the allowed sample distance from its correct margin boundary. The penalty term C controls the strength of this penalty.

Deep Learning methods

Deep learning is based on the idea of capturing hierarchical features: low, mid and high-level. This follows the thinking pattern that the algorithm would be able to first recognise the edges, then shapes and finally whole objects. Adding more layers would mean the ability to capture more features. However, this is not the case due to well-known "vanishing gradient" problem. Thus, new methods try to solve this problem by introducing different types of layers and calculations.

Convolutional Neural Network

Models based on convolutional network are invariant to certain transformations of the inputs. They perform well when the identity of the represented object is invariant under translations, scaling and small rotations [17]. CNN also exploits key property of images - nearby pixels are more strongly correlated than those which are more distant. This allows to extract local features that depend only on small subregions of the image. In later stages of processing such observations can be merged to detect higher-order features and produce information about the image as whole.

The units in convolutional layer are organized into planes (feature maps). In each feature map a unit takes inputs only from a small subregion of the image. All of the units in a feature map are constrained to share the same weight. Input values from a patch are linearly combined using weights and bias, the result transformed by a non-linear function (e.g. sigmoid). Units can be thought of as feature detectors that detect the same pattern but at different locations in the image. Since we need to detect multiple features in order to build well-performing model there will be multiple feature maps in the convolutional layer, each having a separate set of weight and bias parameters.

The outputs of the convolutional units then goes through the subsampling layer (e.g. pooling) which role is to downsample the output of a convolution layer together with the spatial dimensions of height and width. This reduces chance of over-fitting and thereby increases the overall performance

and accuracy of the network. The network will contain multiple pairs of convolutional and subsampling layers until the final layer that is fully connected and adaptive layer with a non-linear function (e.g. softmax). The network training involves error minimization using back-propagation to evaluate the gradient of the error function.

Residual Network

It is known that deep CNNs perform well on image data due to their ability to capture various features in the image. More layers give ability to represent more features. However, after certain number of layers model's performance starts to decrease due to vanishing gradient problem. Repeated multiplication may make the gradient infinitely small as the gradient is back-propagated to previous layers. Network's performance gets saturated or even starts degrading rapidly as it goes deeper. Residual learning gives ability to create very deep neural network models without the degradation problem. The layers are reformulated as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. [5] provides comprehensive empirical evidence showing that these residual networks have lower complexity, they are easier to optimize and can gain accuracy from considerably increased depth. An ensemble of these residual nets achieves 3.57% error on the ImageNet test set. ResNet holds as one of the top performing networks for computer vision tasks.

When constructing a residual neural network instead of underlying mapping between layers with a non-linear $H(x)$ function from input to output authors suggest multiple layers fit a residual mapping. Formally, stacked layers residual mapping can be noted as $F(x) := H(x) - x$, then the original mapping $H(x) = F(x) + x$. Authors hypothesize that it is easier to optimize the residual mapping than to optimize the unreferenced mapping. The latter formulation can be realised by feed forward neural networks with "shortcut connections" (by skipping one or more layers). Such "shortcut connections" simply perform identity mapping and their outputs are added to the outputs of the stacked layers. They don't add extra parameter or computational complexity and entire network still can be trained end-to-end by SGD with backpropagation. These deep residual networks retain gains from greatly increased depth and produce results substantially better than previous networks.

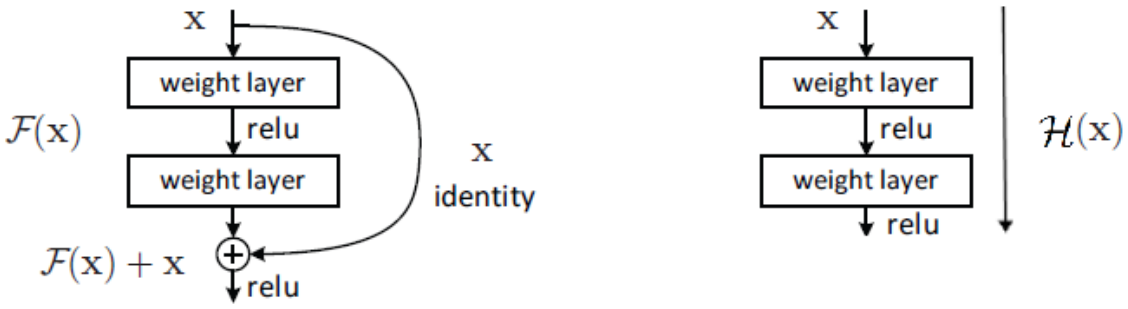


Figure 5: Residual block on the left and regular convolutional block on the right.

Let's consider $H(x)$ as an underlying mapping between a few stacked layers with x denoting the inputs to the first of these layers. If multiple non-linear layers can asymptotically approximate complicated functions, then it is equivalent to hypothesize that they can asymptotically approximate the

residual functions such as $H(x) - x$ (assuming that the input and output are of the same dimensions). So instead of stacked layers to approximating $H(x)$, these layers will approximate a residual function $F(x) := H(x) - x$. The original function thus becomes $F(x) + x$. If the added layers can be constructed as identity mappings, a deeper model should have training error no greater than its shallower counter part. With the reformulation of residual learning: if identity mappings are optimal, the solvers may simply drive the weights of the multiple non-linear layers toward zero to approach identity mappings. [5] shows by experiments that the learned residual functions in general have small responses which suggests that identity mappings provide reasonable preconditioning.

The baseline implementation of ResNet model consists of 18 weighted layers. The convolutional layers mostly have $3x3$ filters and have the same number of filters for the same output feature map size and if the feature map size is halved, the number of filters is doubled so as to preserve the time complexity per layer. Downsampling is performed directly by convolutional layers that have a stride of 2. The network ends with a global average pooling layer and a 1000-way fully-connected layer with softmax.

Performance metrics

To objectively evaluate model's performance we must use various metrics. For classification models it is popular to use measurements such as precision, recall, accuracy, F1-score. However, it is important to choose correct metrics for specific dataset. E.g. accuracy can be a misleading metric for imbalanced datasets. These measures may propagate the underlying marginal prevalences and biases. They also fail to take into account the chance level performance [16]. Let's consider four possible cases of sample to be classified:

- True Positive (TP) - correctly classified sample with label 1;
- True Negative (TN) - correctly classified sample with label 0;
- False Positive (FP) - incorrectly classified sample with label 1;
- False Negative (FN) - incorrectly classified sample with label 0.

Recall (or sensitivity) is the proportion of TP cases that are correctly predicted out of all relevant positive cases. In computational linguistics and machine translation context recall has been shown to have a major weight in predicting the success of word alignment tasks. In Medical context it is moreover regarded as primary measure as the aim is to identify all TP cases. It is also part of Receiver Operating Characteristics (ROC) calculation which has been borrowed from signal processing to become a standard for evaluation comparing True Positive Rate (TPR or recall) and False Positive Rate (FPR or fall-out).

$$Recall = \frac{TP}{TP + FN}.$$

Precision (confidence) denotes the proportion of how many predicted samples are relevant.

$$Precision = \frac{TP}{TP + FP}.$$

F1 metric is the harmonic average of the precision and recall and is a good choice for the imbalanced dataset classification.

$$F1\text{-}score = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}.$$

These measures only focus on the positive examples and predictions, while still capturing some information about the rates and kinds of errors made. However, neither of them captures any information about how well the model handles negative cases. Thus, specificity is the proportion of TN cases that are correctly predicted negative and is also known as the True Negative Rate (TNR).

$$Specificity = 1 - \frac{FP}{FP + TN}.$$

Accuracy calculates how many of predicted values were correct out of all values.

$$Accuracy = \frac{TP + TN}{l},$$

here l is the number of observations.

More complex performance evaluation for binary classification is based on ROC analysis which gives geometric insights into the nature of the measures and their sensitivity to skew. ROC is created by plotting the fraction of TPR on the Y-axis and the FPR on the X-axis. It is a probability curve that essentially separates the "signal" from the "noise". The ROC-AUC metric is an Area Under the Curve (AUC) obtained by ROC curve. It is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better performance of the model distinguishing between the positive and negative classes.

$$ROC\text{-}AUC = \frac{TPR + TNR}{2}.$$

Another good metric for the imbalanced dataset classification is Matthews Correlation Coefficient (MCC). It can be safely used even when classes are very different in sizes. MCC takes into account all cases: TN, TP, FN and FP. Thus, is reliable measure which produces high scores only if the prediction returns good rates for all four of these categories. It can return values between -1 and 1 (1 showing perfect classification, 0 random classification and -1 "opposite" classification).

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}}.$$

3. Practical Application

This section provides detailed description of practical experiment on untested data for accent identification. Experiment results are presented at the end of the section with some comments on how could this be improved in the future. Performance of suggested model was evaluated by comparing its produced metrics to another classical method's. Specifically, in this study we compared Deep Learning model ResNet model with standard SVM approach for binary classification.

Dataset

Experiment was done on new data that was not previously assessed by ML methods. Dataset contains 2738 recordings of single French words. The data was obtained by using a picture naming task. The naming task is ecologically more suitable to obtain naturalistic productions than reading tasks, as it is more spontaneous and prevents a possible interference of orthography in the performance. 30 images that represent various nouns containing /u/ and 30 containing /y/ were chosen. All nouns were likely to be familiar to all participants. Pictures representing the nouns were presented one by one on the screen in a pseudo-random order, such that no more than three objects with the same target vowel in their name appeared in a row. Participants were asked to name the object they saw and to press a button to proceed to the next picture.

All items were recorded in a soundproof booth at the Laboratory of Cognitive Science and Psycholinguistics (Paris, France), at 16 bits mono with a sampling rate of 44.1 kHz. The waveform and the wideband spectrogram of the production data were visualized by professional phoneticians using the software Praat (version 6.0.17) and an annotation text was added for each audio file to segment and label the target vowels. To manually segment the vowels, boundaries were set at zero crossings. Therefore, same amount of cut out single vowel recordings were obtained in the process.

All participants were living in Paris at the time of the test. The non-native speakers were native speakers of American or British English who had started to learn French at school. Speakers of only these two dialects were chosen, as some vowels vary across dialects and it is important to avoid this variability. The participants were students staying in France for at least one year and they all were medium-to proficient speakers of French. Additional few native French speakers were recruited as control participants. None of the participants had hearing or language problems.

The number of samples for each phoneme was different, while for each phoneme, the same number of samples from each language was taken. However, not all of the recordings were suitable for the experiment due to technical problems so they were removed from the set. In the final dataset there are 1458 words containing vowel /u/ and 1280 containing vowel /y/.

The ratio between French and American speakers' recordings is approximately 2:3 (1136 French and 1602 American), therefore, the final dataset is slightly imbalanced. For that purpose, special metrics are employed to evaluate model's performance.

Value	American	French
Total count of samples	3204	2272
Whole word samples	1602	1136
Single vowel samples	1602	1136
/u/ samples	870	588
/y/ samples	732	548
Female recordings (whole word and single vowel each)	1284	762
Male recordings (whole word and single vowel each)	318	374
Whole word max length	3.3524 (s)	6.6336 (s)
Single vowel max length	0.6474 (s)	0.3795 (s)
Whole word average length	1.1264 (s)	0.8372 (s)
Single vowel average length	0.1168 (s)	0.0937 (s)

Table 1: Dataset summary table.

Data pre-processing

All the recordings were in Waveform Audio File Format (WAV) and they needed to be pre-processed and converted to a format suitable for selected model. Since ResNet is based on CNN architecture which is fitted for image data, we needed to represent speech signals as images. For that purpose we used Mel-spectrograms, described in Analytical part. To convert WAV files to 2D Mel-spectrogram data we used Python library *librosa* (version 0.8.1) [10] which is dedicated to audio analysis and provides the building blocks necessary to create audio information retrieval systems.

First step was loading audio files into variables each of which is an array of floating point time series. Each file also contains sampling rate data (number of samples per second), in this case it is 44100 Hz. Since all of the recordings have different duration, the audio data had to be processed to have the same length. This was achieved by padding the arrays from both sides with constant value of 0 (zero-padding). The fixed length of 0.3 (s) for single vowel and 2.4 (s) for whole word was chosen based on approximated value at 99^{th} quantile. The recordings that were longer than specified fixed value were trimmed (from the right). Result arrays were used to calculate mel-scaled spectrograms (which operates on a power spectrum) using STFT.

When computing an STFT number of parameters have to be selected. *n_fft* indicates the length of short segments on which FFT is calculated. Usually, these segments overlap (to avoid information loss) and the distance between two segments is *hop_length* which should be less than *n_fft*. To optimize the speed of the FFT calculation *n_fft* should be set to a power of two. *n_mels* indicates number of mel bands to generate. The spectrogram calculation output is in the shape of (*n_mels*, t) where t is record duration or *win_length*. Final parameters selected for the pre-processing:

- *n_fft = 2048*,
- *hop_length = 512*,
- *n_mels = 128*.

After obtaining power spectrograms (amplitude squared) we can convert them to decibel (dB) units. Then the scale becomes logarithmic. This limits the numerical range and the intensity of colours. When this is plotted colour intensity corresponds more closely to what we hear than if one used a linear scale.

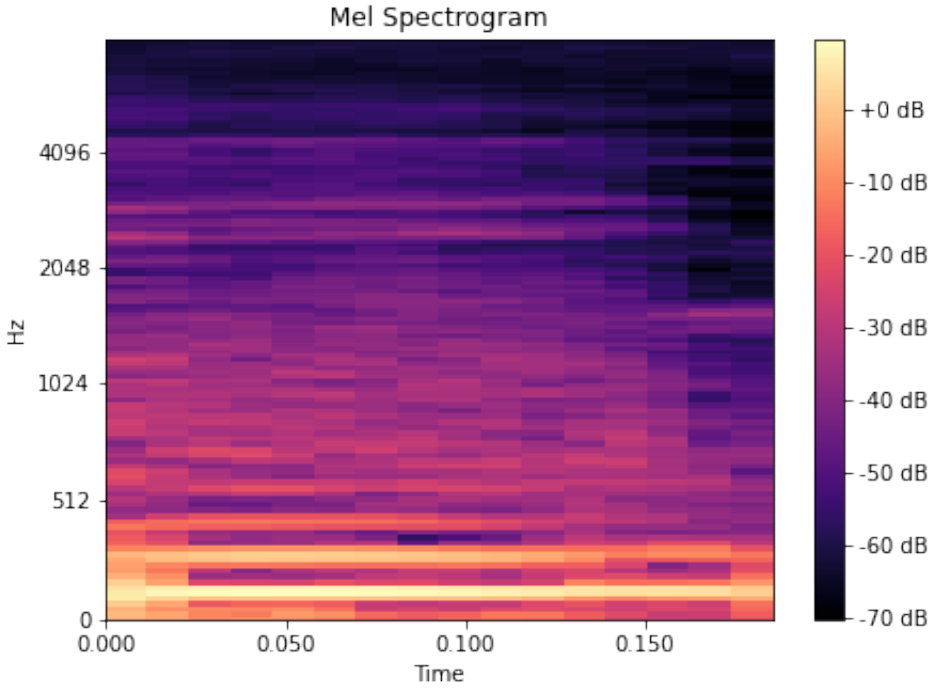


Figure 6: Mel-spectrogram of a cut out vowel /u/ in word "bambou".

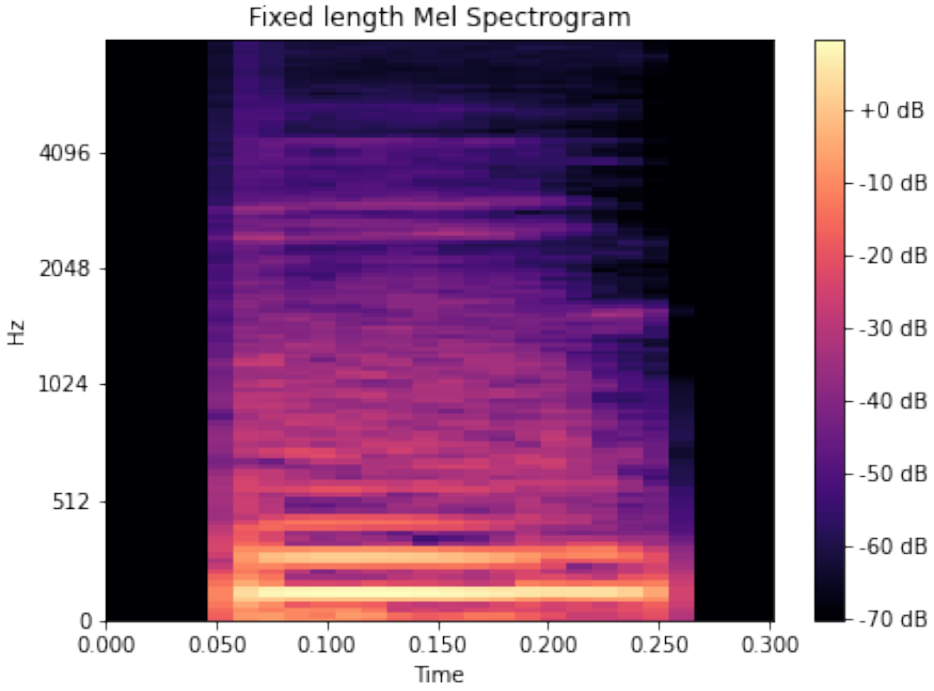


Figure 7: Mel-spectrogram of a cut out vowel /u/ in word "bambou" after padding.

When spectrogram values were calculated we applied two step normalization: Z-normalization and value scaling. Z-normalization normalizes every value in a dataset such that the mean of all values is 0 and the standard deviation is 1 by applying formula: $x_{new} = (x - \mu)/\sigma$. Then spectrogram scaling maps the values to the range between 0 and 1. It is known that CNN models perform well on data in this range and it also improves performance by applying neural network's calculations on small numbers. Scaling was done on each column of a 2D array by aggregating over the rows. This highlights the energy peaks over time.

After normalization, we appended additional dimension to the data, which corresponds to a colour channel. In this case we only have single channel data (due to mono audio). However, adding this dimension is necessary for the ResNet network which expects 4 dimensions (batch, colour channel, height and weight). Then arrays are converted to tensors and audio data is combined with the appropriate label (0 - French speaker and 1 - American speaker).

Finally, the data was split into 80% train, 10% validation and 10% test, maintaining class ratio in each split. Validation dataset was used to track and evaluate model's performance during training and to notice if the model starts to overfit the training data. Testing data was used at the end of the training cycle to obtain the final performance metrics. Data was split in this way 5 different times and re-shuffled. Such 5-fold split was used to apply model Cross Validation and better evaluate performance of suggested approach.

Method description

This study tried to solve an accent identification problem that can be formulated as binary classification task. Chosen model is a special case of CNN which is ResNet. This model suppose to offer better performance in terms of computation and network depth. For this study used ResNet implementation that is offered in Python library *torchvision* (version 0.11.1), which is part of the PyTorch project [12]. More specifically, 18-layer pre-trained model with Cross Entropy Loss and Adam optimizer. It was chosen to use 18-layer model instead of one with more layers due to the fact that dataset used in this study is relatively small. It is known, that more complex neural networks tend to overfit training data and it is recommended to use networks with less layers for small datasets.

First layer of the model was re-defined to take 1-channel input, with $7x7$ kernel, $2x2$ stride and $3x3$ padding (kernel size, stride and padding parameters were chosen according to original implementation in [5]). Last layer was changed to map previous layer outputs to 2 output features (equivalent to 2 classes).

Cross Entropy Loss was chosen because it is widely used for classification problems and is particularly useful for unbalanced training set. Calculation formula for binary classification can be written as follows:

$$\mathcal{L}_{CE}(\hat{y}, y) = -\Sigma_{i=1}^{l}(y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})).$$

To better understand how this approach fits the task 5-fold Cross Validation was used for model training. Each time model's weights were reset and training process started from the beginning. After each fold, metrics for obtained model were calculated.

Hyperparameter fine-tuning

To improve model's performance we evaluated few hyperparameters: different batch size, learning rate and number of epochs. These values are important since they directly impact on how fast the model learns, calculation time and overall result. Each iteration only one hyperparameter was changed (e.g. batch size) to obtain comparable results.

Batch size	Epoch	Train loss	Train accuracy	Val loss	Val accuracy
4	10	0.5484	0.7167	0.4816	0.7717
8	10	0.4745	0.7828	0.5557	0.7679
16	10	0.4040	0.8100	0.4844	0.7535
32	10	0.4230	0.8036	0.7156	0.7052
64	10	0.3094	0.8591	0.5814	0.7753

Table 2: Model performance with different batch sizes.

Table 2 shows the results with different batch sizes. Higher numbers than 64 were not considered due to relatively small dataset where validation and testing sets contain only 273 and 274 samples, respectively. There was no high increase in performance among different batch sizes, therefore it was selected to use batch size 16 due to low loss value for both training and validation sets and high accuracy for both sets. Also this value yields reasonably small difference between training and validation accuracy.

Learning rate	Epoch	Train loss	Train accuracy	Val loss	Val accuracy
0.0001	10	0.1874	0.9241	0.5708	0.8056
0.001	10	0.4086	0.8123	0.5208	0.7465
0.1	10	0.5489	0.7129	0.5963	0.7361

Table 3: Model performance with different learning rates.

Table 3 shows the performance results with different learning rates (with batch size 16). Lowest loss in both sets was produced when learning rate is equal to 0.001. It also produced more stable training and validation convergence, therefore this learning rate was selected for final model.

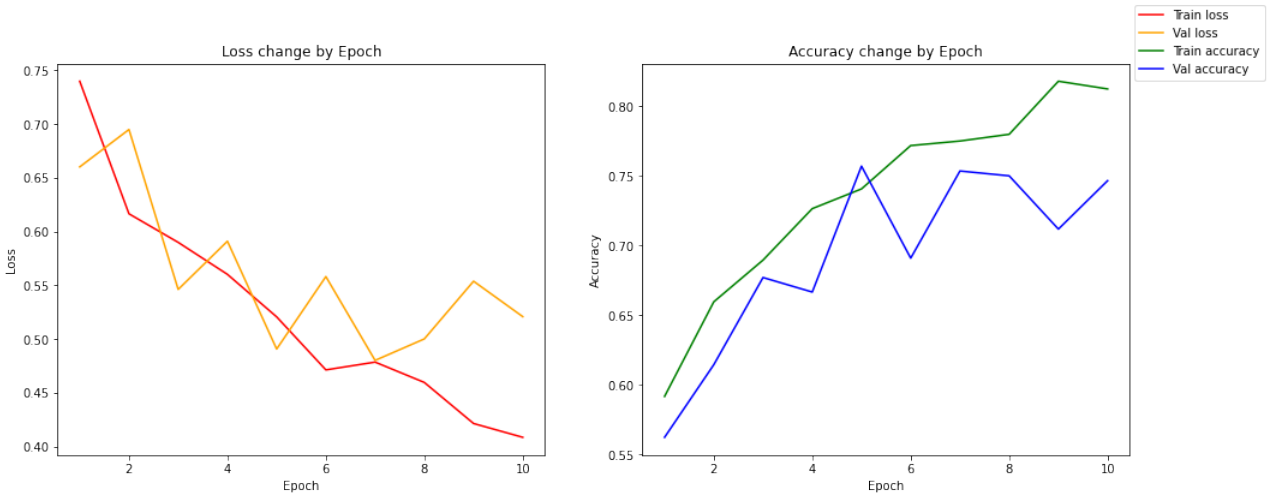


Figure 8: Loss and accuracy change over time when learning rate is 0.001.

Last parameter that was evaluated is number of epochs. Since the model has a tendency to overfit due to lack of data, one of the ways to prevent that is an early stop. To evaluate that, model was tested on 30 epochs (with batch size 16 and learning rate 0.001). Looking at loss and accuracy curves from both training and validation sets we can notice that those curves start to clearly deviate from each other around 15th epoch. This indicates that model starts to learn the training set instead of general features of the class and thus, overfits. Therefore, for the final model we selected 15 epochs.

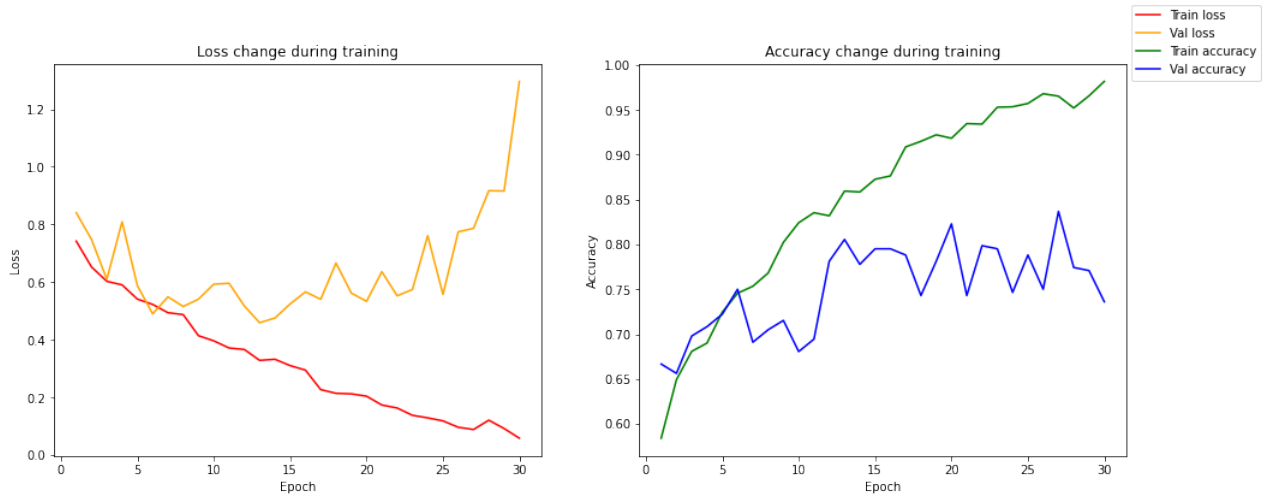


Figure 9: Loss and accuracy change over epochs.

After experimentation with different hyperparameters the following ones were chosen for the final model:

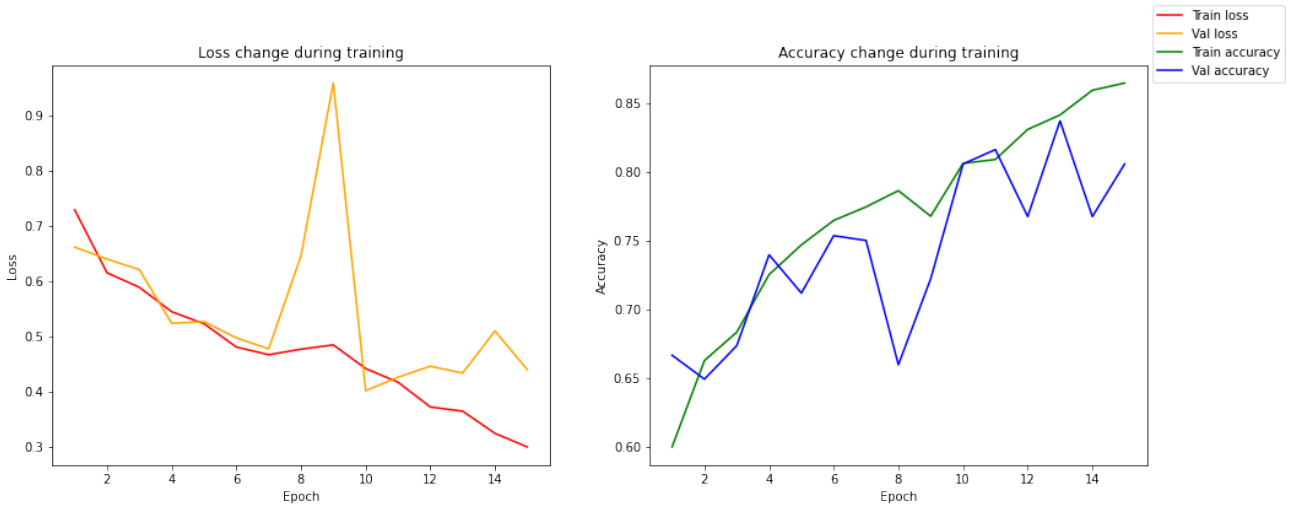
- *batch_size = 16*,
- *learning_rate = 0.001*,
- *epochs = 15*.

Evaluation and results

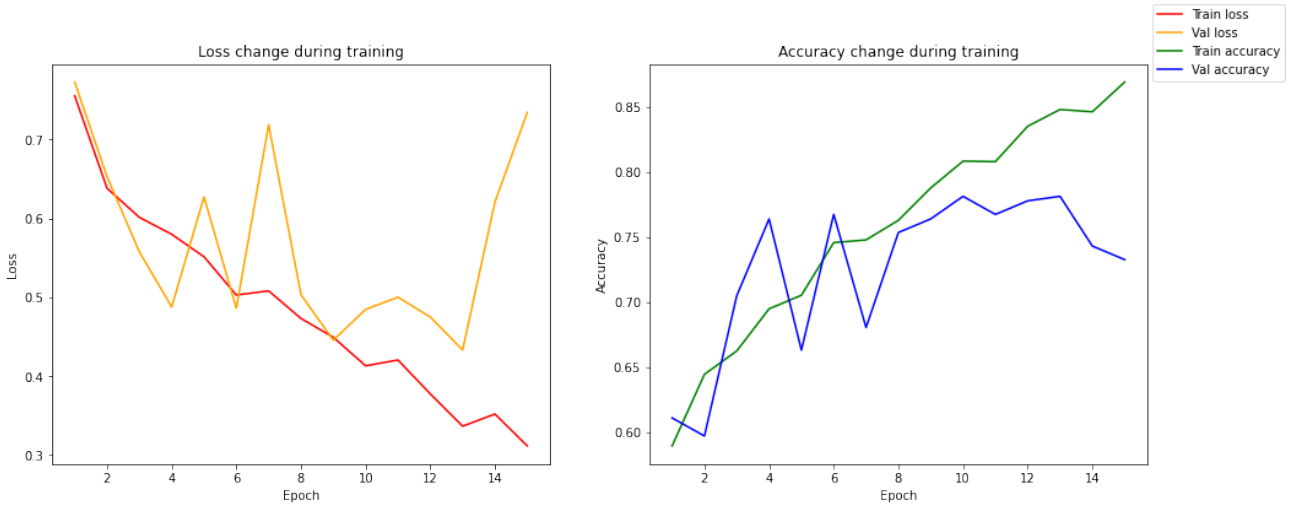
In this subsection we will describe the experiment results and will present obtained performance metrics to better understand the method's fit for the given problem.

Vowel classification

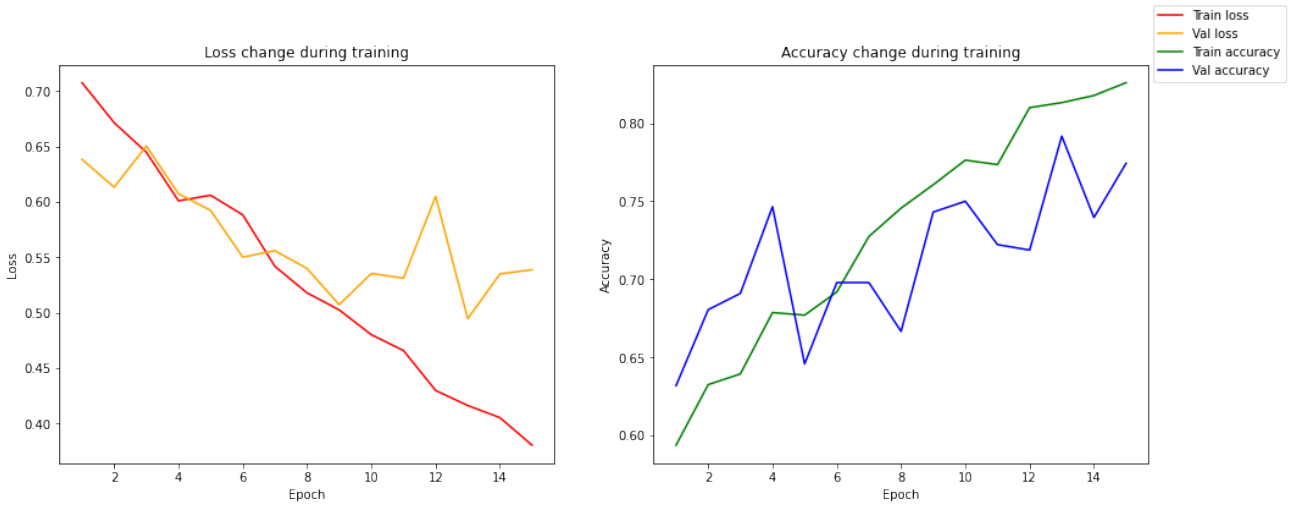
The goal of a single vowel classification was to classify cut out sounds of /u/ and /y/ to American and native French speakers. For better understanding of how well ResNet model classifies such data we employed model Cross Validation. Since there is some randomness involved in calculating model's weights we must repeat this process multiple times to ensure that acceptable results can be reproduced each training. The graphs bellow show each fold's training and validation process loss and accuracy change over epochs. Each time data was reshuffled into different batches.



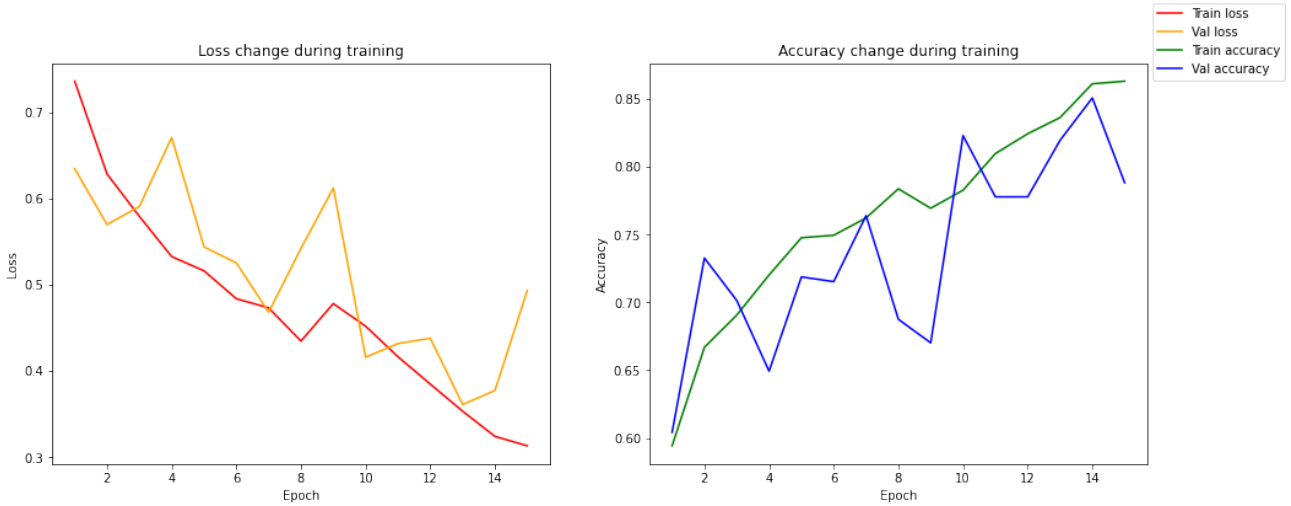
(a) 1^{st} Fold



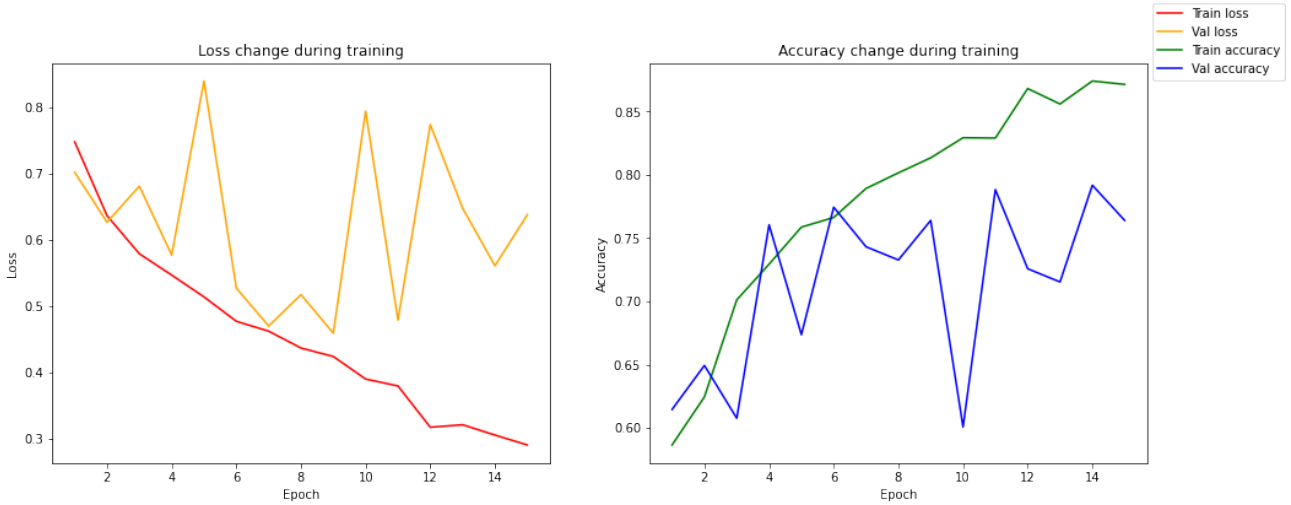
(b) 2^{nd} Fold



(c) 3^{rd} Fold



(d) 4^{th} Fold



(e) 5^{th} Fold

Figure 10: Accuracy and loss change during Cross Validation training process vowel classification.

From the graphs we observe that validation curves are relatively close to the training, although we still see some fluctuation in loss and accuracy values produced in validation process. However, these graphs only take into account two measurements and, as mentioned previously, accuracy is not always the best performance metric. Since we applied 5-fold model Cross Validation it means that 5 different models were created in each iteration. Each of those models was trained for 15 epochs. At the end of 15th epoch in each fold we evaluated model's performance on unseen data (test set). This showed the more detailed picture of how well model was performing.

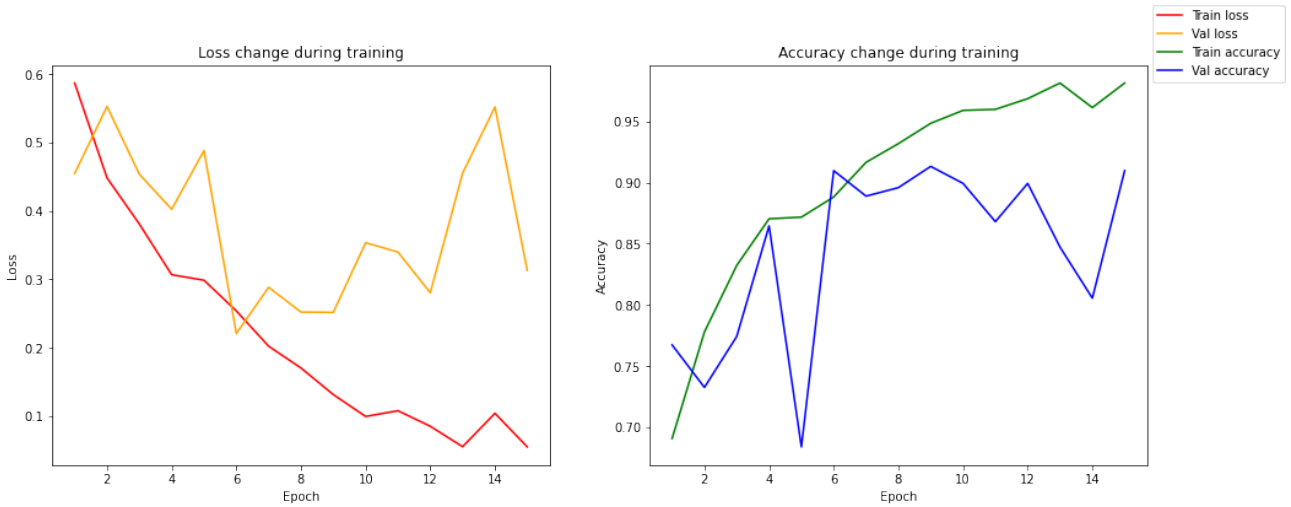
Fold	1	2	3	4	5	Average
Train loss	0.2999	0.3116	0.3804	0.3131	0.2901	0.3190
Train Accuracy	0.8643	0.8690	0.8259	0.8630	0.8713	0.8587
Val loss	0.4397	0.7345	0.5389	0.4933	0.6372	0.5687
Val Accuracy	0.8056	0.7326	0.7743	0.7882	0.7639	0.7729
Test loss	0.6576	0.5566	0.4366	0.4609	0.5019	0.5227
Test Accuracy	0.7500	0.8125	0.7813	0.8160	0.7569	0.7833
Recall	0.8876	0.9485	0.8151	0.8002	0.9095	0.8722
Fall-out	0.4400	0.3685	0.3358	0.1446	0.4534	0.3484
Precision	0.7300	0.7776	0.7854	0.8746	0.7386	0.7813
Specificity	0.5600	0.6315	0.6642	0.8554	0.5466	0.6516
F1 score	0.7914	0.8466	0.7956	0.8252	0.8003	0.8118
ROC-AUC	0.7238	0.7900	0.7397	0.8278	0.7280	0.7619
MCC	0.4740	0.5740	0.4258	0.5908	0.4830	0.5095

Table 4: Cross Validation results for vowel classification.

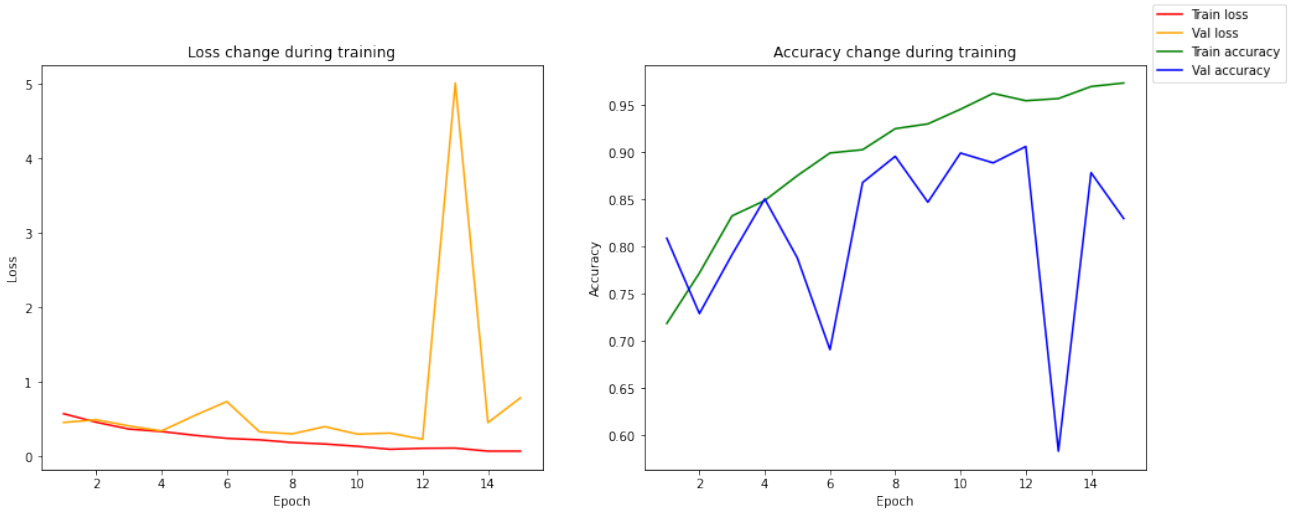
From the 4 table we see that all 5 models reached an average of 78.33% accuracy on test (unseen) data at the end of 15^{th} epoch with the highest accuracy of 81.6% in 4th fold. However, it is apparent that models tend to correctly classify American speakers' samples (where label is 1) more than French speakers. This is evident from relatively high average recall (or TPR) score of 87.22% and only 65.16% specificity (or TNR). This might have been due to slightly imbalanced dataset containing more samples of American speakers. The difference between recall and specificity is approximately 20% which is also the difference between class samples in the dataset.

Word classification

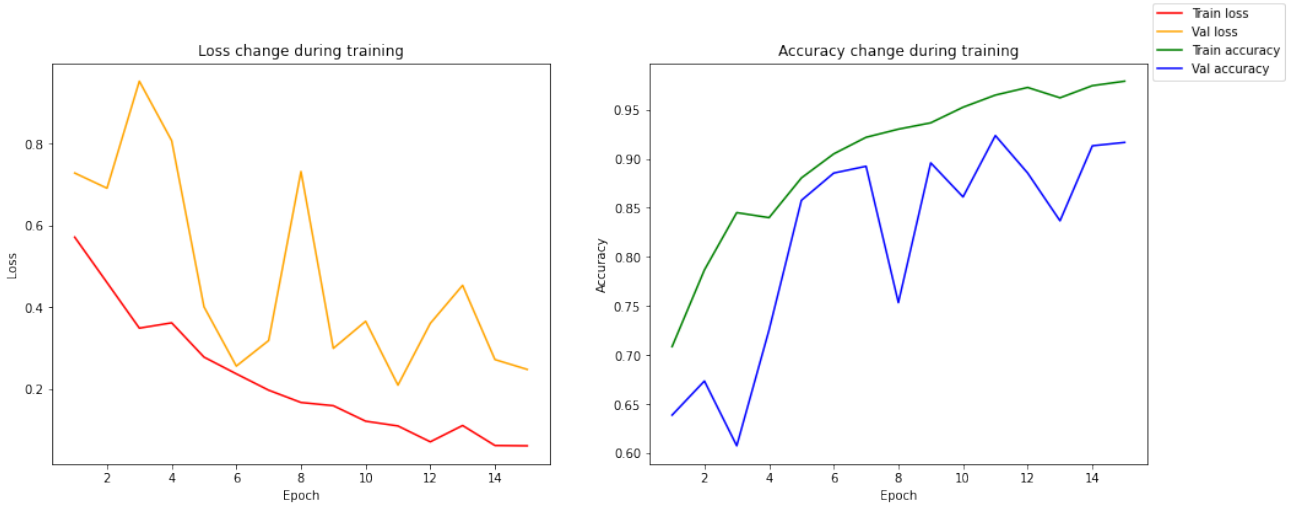
Next part of the experiment was whole word classification. It was implemented using the same pre-processing techniques and using the same pre-trained 18-layer ResNet model. However, it is important to note, that each sample in this case contains much more information than cut out vowel. Specifically, each word contained either the target vowel /u/ or /y/, but also other sounds that potentially are pronounced differently by native and non-native speakers. So the results of the classification were not surprising and showed acceptable performance. As in vowel case there were 5 separate models generated using Cross Validation and 80% train, 10% validation and 10% test sets. Following graphs show accuracy and loss changes over epochs.



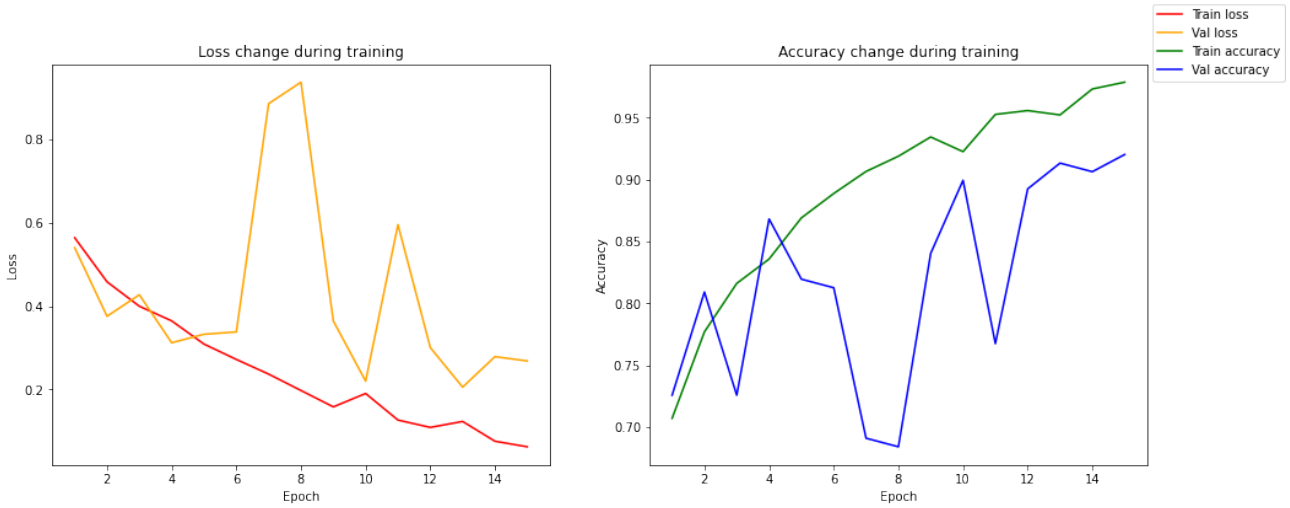
(a) 1^{st} Fold



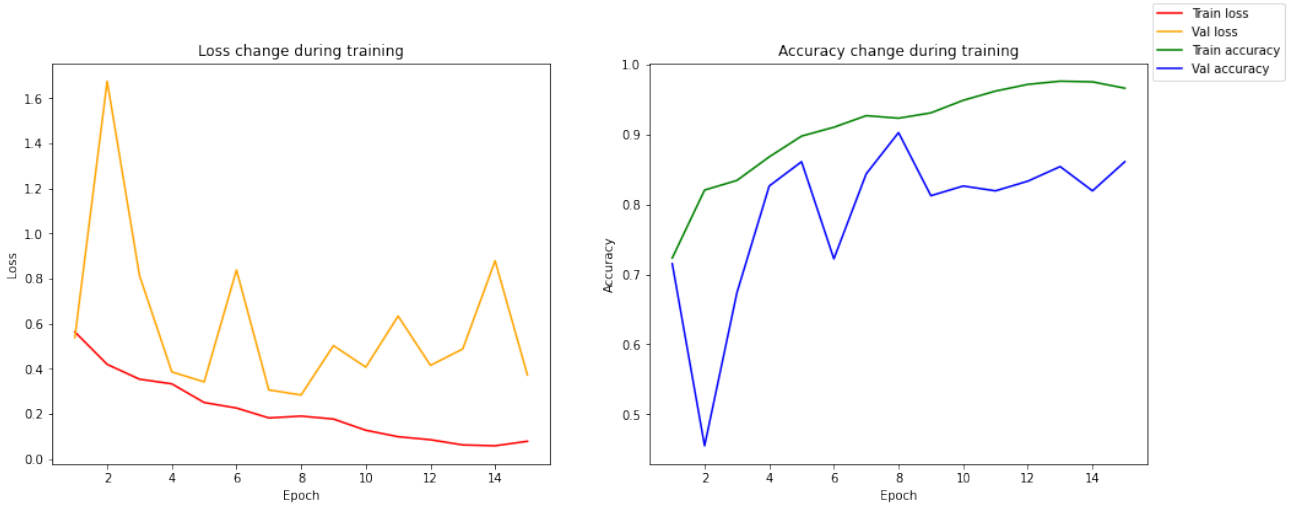
(b) 2^{nd} Fold



(c) 3^{rd} Fold



(d) 4^{th} Fold



(e) 5^{th} Fold

Figure 11: Accuracy and loss change during Cross Validation training process for word classification.

The graphs show similar fluctuations to vowel classification in validation process measurements but both curves remain close to each other at the end of the training process. This means that the model is not overfitting the data and produces similar results on unseen data as on training data.

Fold	1	2	3	4	5	Average
Train loss	0.0552	0.0735	0.0600	0.0625	0.0779	0.0658
Train Accuracy	0.9813	0.9735	0.9790	0.9786	0.9662	0.9757
Val loss	0.3132	0.7901	0.2472	0.2686	0.3722	0.3983
Val Accuracy	0.9097	0.8299	0.9167	0.9201	0.8611	0.8875
Test loss	0.2647	0.5150	0.2096	0.2381	0.5354	0.3526
Test Accuracy	0.9236	0.8750	0.9306	0.9271	0.8681	0.9049
Recall	0.9301	0.7313	0.9396	0.8862	0.9857	0.8946
Fall-out	0.0791	0.0000	0.0797	0.0187	0.3250	0.1005
Precision	0.9327	0.9444	0.9430	0.9758	0.8180	0.9228
Specificity	0.9209	1.0000	0.9203	0.9813	0.6750	0.8995
F1 score	0.9273	0.8193	0.9393	0.9216	0.8888	0.8993
ROC-AUC	0.9255	0.8657	0.9299	0.9337	0.8303	0.8970
MCC	0.8423	0.7235	0.8552	0.8022	0.6883	0.7823

Table 5: Cross Validation results for word classification.

Table 5 show that word classification shows more than 10% increase in test accuracy (comparing average fold value) than vowel classification. This might be due to the previously mentioned fact, that word contains more information, therefore, model can learn more distinct features. Also, in word classification case we see much higher metrics' values that are useful for imbalanced classification (ROC-AUC and MCC). Best performing model offers more than 93% accuracy on test dataset. In the word classification case we see low difference between recall and specificity metrics which means that both classes are classified correctly at around the same rate. Note that in vowel classification, the difference between the recall and the specificity score could be explained by the imbalance of groups in the data (i.e., there was more American data, thus the model tended to classify items as American productions). However, in the word classification, where data is still unbalanced in the same way, this tendency is no more visible. Which leads to a consideration that effect of slight class imbalance can be reduced by using samples with more information. Nonetheless, in both cases models could be improved by reducing fluctuations of validation curves.

Comparison with SVM

To evaluate performance of suggested approach we need to compare its results to a standard method. In this study we chose to use SVM. Classical SVM implementations only take 1D input. Therefore, comparison can be done in two ways: comparing coefficient values that are used for spectrogram creation or converting 2D spectrograms into 1D arrays. Those arrays can be fed to the SVM model together with class labels. SVM classification was implemented using Python library *scikit-learn* (version 1.0.1) [13].

Based on relevant speech signal classification cases that were presented in literature review it was chosen to implement benchmarking using MFCCs and SVM. Same dataset was used as in the previous parts. However, it was split into 80% training and 20% sets. It was decided to omit validation set for simplicity reasons because we did not intend to implement optimal parameter search and only use this model as a baseline. In such cases, only simple and standard model is considered as can be seen in one of reviewed articles [8]. Samples in the dataset were pre-processed to have the same length and then we used a pre-defined method to calculate MFCCs from an audio file (even though actual calculations would involve same steps as calculating Mel-spectrograms and then applying discrete cosine transformation). We chose to use *n_mfcc* (number of MFCCs to return) value of 13 as it is widely used in many applications.

	Vowel	Word
Accuracy	0.6642	0.6734
Precision	0.7882	0.9128
Recall	0.6856	0.6599
F1 score	0.7333	0.7660
ROC-AUC	0.6529	0.6953
MCC	0.2911	0.3110

Table 6: SVM MFCC-based classification metrics.

Table 6 shows SVM MFCC-based classification model metrics obtained on the test set. It achieved approximately 66.42% accuracy for single vowel and 67.34% accuracy for whole word. It shows low results for MCC metric and is heavily impacted by class imbalance in the dataset.

Since SVM classification results using MFCCs were significantly lower than those obtained with ResNet it was interesting to see if SVM performance can be improved converting 2D spectrogram data into 1D arrays (even though it is not a common approach). Surprisingly, results were higher and outperformed SVM with MFCCs, although did not surpass ResNet approach. This might indicate that spectrograms are generally better representation of sound signals.

	Vowel	Word
Accuracy	0.7336	0.7938
Precision	0.7788	0.8006
Recall	0.7692	0.8399
F1 score	0.7740	0.8198
ROC-AUC	0.7254	0.7877
MCC	0.4496	0.5801

Table 7: SVM spectrogram-based classification metrics.

Table 7 shows that model reached 73.36% accuracy for single vowel and 79.38% accuracy for whole word and increased performance metrics for imbalanced data. We can conclude that our selected baseline SVM model performs better using the same pre-processing steps as in our main approach.

Conclusions

Accent identification is an important problem and solving it can improve various ASR systems. In this work a special focus was made on the signal pre-processing technique. The thesis showed novelty application of ResNet model to classify accented speech when sound signals are represented as 2D spectrograms. As expected, it gives great results in comparison to classical approach and with reasonable computation time.

- Compared to MFCC-based classification with SVM algorithm new suggested approach leads to more than 10% increase in model performance measurements with 78.33% accuracy, 87.22% recall and 78.13% precision for single vowel and 90.49% accuracy, 89.46% recall and 92.28% precision for whole word at 15th epoch based on 5-fold Cross Validation average.

- SVM model made use of acoustical parameters - MFCCs. The baseline model produced accuracy of 66.42% for a single vowel classification and 67.34% for the whole word.

- Interestingly, SVM classification performance was increased to 73.36% accuracy for vowel and 79.38% for word by using spectrogram data as input instead of MFCCs. This could have been determined by the fact that spectrograms provide more information than it is obtained by calculating MFCCs.

- Classifying words produced better results than vowels due to more information in each sample. Word samples, in addition to /u/ or /y/ vowel, contained more sounds that could have helped to identify the accent. More research needs to be done on how those vowels, that are known to be difficult to pronounce correctly for non-native speakers, impact word classification and if we would be able to produce similar results with other French words. Vowel classification can be improved by searching and focusing more on vowel-specific differences in pronunciation and on the extraction of these particular features.

- All vowel signals were either trimmed or padded with zeros to length of 0.3 (s) and all word signals to 2.4 (s). It is known that native speakers usually pronounce words faster due to the fact that non-native speakers require more time to process information and produce words. Algorithm may have used these differences to classify sounds according to the length of meaningful information in the spectrogram.

- Even though full dataset is slightly imbalanced with 60% of American speakers and 40% French speakers suggested approach still produced fairly good imbalanced data metrics such as Matthews Correlation Coefficient (MCC), especially for word classification with 0.7823 5-fold average and a bit lower for single vowel with value of 0.5095 which is still significantly better than benchmark model results.

- This thesis succeeded in solving an accent classification problem, that traditional acoustic methods used by psycholinguists failed to tackle [11]. This points to the prospects of employing ML-based methods to improve research methodology in psycholinguistics and other fundamental

research fields. Possibilities to use feature extraction for the identification of acoustic correlates, specific to particular accents, could be further examined.

To sum up, suggested approach could be helpful to solve accent identification problems or sound classification problems using 2D spectrograms. Further research could include classifying more than 2 accent groups, focusing on different sounds or including broader set of words. It would also be useful to develop an automatic algorithm for data preparation since manually cutting out vowels is a lengthy process. Suggested approach relies on specific case of CNN so it would be possible to investigate other models or different number of layers in the network. It would be also interesting to explore computational differences between ResNet and CNN which usually takes quite a long time to train on image data.

References

[1] Christian Bartz, Tom Herold, Haojin Yang, and Christoph Meinel. Language identification using deep convolutional recurrent neural networks. In *International conference on neural information processing*, pages 880–889. Springer, 2017.

[2] Keven Chionh, Maoyuan Song, and Yue Yin. Application of convolutional neural networks in accent identification. *Project Report, Carnegie Mellon University, Pittsburgh, Pennsylvania*, 2018.

[3] Himanish Das and Pinki Roy. *A Deep Dive into Deep Learning Techniques for solving Spoken Language Identification Problems in Speech Signal processing*, pages 81–100. 12 2018.

[4] Bruce Hayes. *Introductory phonology*, volume 32. John Wiley & Sons, 2011.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Yishan Jiao, Ming Tu, Visar Berisha, and Julie Liss. Accent identification by combining deep neural networks and recurrent neural networks trained on long and short term features. pages 2388–2392, 09 2016.

[7] Liu Wai Kat and P. Fung. Fast accent identification and accented speech recognition. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*, volume 1, pages 221–224 vol.1, 1999.

[8] Gražina Korvel, Povilas Treigys, and Bożena Kostek. Highlighting interlanguage phoneme differences based on similarity matrices and convolutional neural network. *The Journal of the Acoustical Society of America*, 149(1):508–523, 2021.

[9] Gražina Korvel, Povilas Treigys, Gintautas Tamulevicus, Jolita Bernataviciene, and Bozena Kostek. Analysis of 2d feature spaces for deep learning-based speech recognition. *Journal of the Audio Engineering Society*, 66(12):1072–1081, 2018.

[10] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, 2015.

[11] Gerda Ana Melnik-Leroy, Rory Turnbull, and Sharon Peperkamp. On the relationship between perception and production of l2 sounds: Evidence from anglophones' processing of the french/u/–/y/contrast. *Second Language Research*, page 0267658320988061, 2021.

[12] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[14] Magdalena Piotrowska, Andrzej Czyżewski, Tomasz Ciszewski, Gražina Korvel, Adam Kurowski, and Bożena Kostek. Evaluation of aspiration problems in l2 english pronunciation employing machine learning. *The Journal of the Acoustical Society of America*, 150(1):120–132, 2021.

[15] Magdalena Piotrowska, Grazina Korvel, Adam Kurowski, Bozena Kostek, and Andrzej Czyzewski. Machine learning applied to aspirated and non-aspirated allophone classification–an approach based on audio "fingerprinting". In *Audio Engineering Society Convention 145*. Audio Engineering Society, 2018.

[16] David MW Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*, 2020.

[17] Markus Svensén and Christopher M Bishop. Pattern recognition and machine learning, 2007.

[18] Thibault Viglino, Petr Motlicek, and Milos Cernak. End-to-end accented speech recognition. In *Interspeech*, pages 2140–2144, 2019.

Appendix A

Code Listing 1: This appendix provides the Python code that was written to obtain study results.

```
# Mounting Google Drive
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)

# Reading the data
import glob
import numpy as np
import pandas as pd
from tqdm import tqdm

recordings_path = 'gdrive/My Drive/Accent Identification/All Recordings'
images = glob.glob(recordings_path + '/*/*/*/*/*.wav')
df = pd.DataFrame(data={'filepath': list(images)})
df.shape

# Assign labels and features
def assign_vowel(x):
    if 'Cut_u' in x:
        return 1
    elif 'Cut_y' in x:
        return 0
    else:
        return np.nan

df['label'] = df['filepath'].apply(lambda x: 0 if 'French' in x else 1)
df['sound_u'] = df['filepath'].apply(lambda x: assign_vowel(x))
df.head()

tqdm.pandas()
df['duration'] = df['filepath'].progress_apply(
    lambda x: librosa.get_duration(filename=x))
df['sex'] = df['filepath'].progress_apply(lambda x: 0 if 'Male' in x else 1)
df.head()

# Data pre-processing
import librosa
import librosa.display
import matplotlib.pyplot as plt

def get_fixed_length_audio(wav, sampling_rate, audio_length):
    """
    This function returns the wav variable of desired length specified with the
    argument audio_length by padding with zeros or trimming the recording
    """
    if wav.shape[0] < audio_length * sampling_rate:
        wav = np.pad(wav,
                     int(np.ceil((audio_length * sampling_rate - wav.shape[0]) / 2)),
```

```
                        mode='constant')  # pads with zeros
    wav = wav[:int(np.ceil(audio_length * sampling_rate))]

    return wav

def get_melspectrogram(wav, sampling_rate, n_fft=2048, hop_length=512, n_mels=128):
    """
    This function returns mel-scaled spectrogram
    """
    spectrogram = librosa.feature.melspectrogram(wav,
                                                 sr=sampling_rate,
                                                 n_fft=n_fft,
                                                 hop_length=hop_length,
                                                 n_mels=n_mels)
    # Convert a power spectrogram to decibel units
    spectrogram = librosa.power_to_db(spectrogram)

    return spectrogram

def spectrogram_normalization(spectrogram):
    """
    This function returns spectrogram afte Z-score normalization and Min-Max scaling
    """
    epsilon = 1e-6

    # Z-score normalization
    mean = spectrogram.mean()
    std = spectrogram.std()
    spectrogram_normalized = (spectrogram - mean) / (std + epsilon)

    # Normalize each column of a 2D array by aggregating over the rows
    spectrogram_normalized = librosa.util.normalize(spectrogram)

    return spectrogram_normalized

def load_data(filepath, audio_length):
    """
    This function returns stacked dataset with labels
    """
    wav, sampling_rate = librosa.load(filepath, sr=None)
    wav = get_fixed_length_audio(wav, sampling_rate, audio_length)
    melspectrogram = get_melspectrogram(wav, sampling_rate)
    melspectrogram_normalized = spectrogram_normalization(melspectrogram)
    melspectrogram_expanded = np.expand_dims(melspectrogram_normalized, axis=0)

    return melspectrogram_expanded

# ResNet model with Cross Validation
import torch
import torch.utils.data as data_utils
import torch.nn as nn
```

```
import torch.nn.functional as F
import torch.optim as optim
from torchvision.models import resnet18

# Model parameters
BATCH_SIZE = 16
EPOCHS = 15
LEARNING_RATE = 0.001

from sklearn.model_selection import StratifiedKFold, train_test_split
stratified_k_fold = StratifiedKFold(shuffle=True, random_state=1)

def evaluate_model(model, test_loader):
    """
    This function calculates loss function, accuracy and other performance metrics
    on a test dataset
    """
    model.eval()
    batch_count = len(test_loader)
    epsilon = 1e-6

    with torch.no_grad():
        test_loss = 0
        test_accuracy = 0

        tpr = np.zeros(batch_count)
        fpr = np.zeros(batch_count)
        prec = np.zeros(batch_count)
        tnr = np.zeros(batch_count)
        f1 = np.zeros(batch_count)
        auc = np.zeros(batch_count)
        mcc = np.zeros(batch_count)

        for i, batch in enumerate(tqdm(test_loader)):
            data, labels = batch
            if torch.cuda.is_available():
                data, labels = data.cuda(), labels.cuda()
            outputs = model(data)
            _, predicted = torch.max(outputs, dim=1)
            loss = loss_function(outputs, labels)
            test_loss += loss.item()

            total = labels.size(0)
            correct = (predicted == labels).sum().item()
            accuracy = correct / total
            test_accuracy += accuracy

            # True positive
            tp = ((predicted == labels) & (predicted == 1)).sum().item()
            # False positive
            fp = ((predicted != labels) & (predicted == 1)).sum().item()
```

```
            # True negative
            tn = ((predicted == labels) & (predicted == 0)).sum().item()
            # False negative
            fn = ((predicted != labels) & (predicted == 0)).sum().item()

            # True positive rate (sensitivity or recall)
            tpr[i] = tp / (tp + fn + epsilon)
            # False positive rate (fall-out)
            fpr[i] = fp / (fp + tn + epsilon)
            # Precision
            prec[i] = tp / (tp + fp + epsilon)
            # True negatvie tate (specificity)
            tnr[i] = 1 - fpr[i]
            # F1 score
            f1[i] = 2 * tp / (2 * tp + fp + fn + epsilon)
            # ROC-AUC for binary classification
            auc[i] = (tpr[i] + tnr[i]) / 2
            # MCC
            mcc[i] = (tp * tn - fp * fn) / (np.sqrt((tp + fp) * (tp + fn) * (tn + fp) *
                                                    (tn + fn)) + epsilon)

    metrics = (np.average(tpr), np.average(fpr), np.average(prec), np.average(tnr),
               np.average(f1), np.average(auc), np.average(mcc))

    return test_loss / batch_count, test_accuracy / batch_count, metrics

# Vowel classification
df_v = df[df['sound_u'].notna()]
df_v = df_v.sample(frac=1).reset_index(drop=True)
df_v.shape

df_v['duration'].quantile(q=0.99) # value at 99th quantile

tqdm.pandas()
df_v['audio_data'] = df_v['filepath'].progress_apply(lambda x: load_data(x, 0.3))

result_df = pd.DataFrame(columns=['Fold', 'Epoch', 'Recall', 'Fall-out', 'Precision',
                                  'Specificity', 'F1', 'ROC-AUC', 'MCC',
                                  'Train loss', 'Train Accuracy',
                                  'Val loss', 'Val Accuracy',
                                  'Test loss', 'Test Accuracy'])

# Vowel classification [with Test Metrics]
fold = 0

for train_index, test_index in stratified_k_fold.split(df_v, df_v['label']):
    print('---------------- Fold ' + str(fold + 1) + ' ----------------')

    # Splitting data into 80% train, 10% validation and 10% test
    train_df = df_v.iloc[train_index]
    val_test_df = df_v.iloc[test_index]
```

```
val_df, test_df = train_test_split(val_test_df, stratify=val_test_df['label'],
                                   test_size=0.5, random_state=1, shuffle=True)

train_audio_data = torch.Tensor(np.stack(train_df['audio_data'], axis=0))
train_labels = torch.Tensor(np.stack(train_df['label'], axis=0)).long()

val_audio_data = torch.Tensor(np.stack(val_df['audio_data'], axis=0))
val_labels = torch.Tensor(np.stack(val_df['label'], axis=0)).long()

test_audio_data = torch.Tensor(np.stack(test_df['audio_data'], axis=0))
test_labels = torch.Tensor(np.stack(test_df['label'], axis=0)).long()

train_data = data_utils.TensorDataset(train_audio_data, train_labels)
val_data = data_utils.TensorDataset(val_audio_data, val_labels)
test_data = data_utils.TensorDataset(test_audio_data, test_labels)

train_loader = data_utils.DataLoader(train_data, batch_size=BATCH_SIZE, shuffle=
                                     True)
val_loader = data_utils.DataLoader(val_data, batch_size=BATCH_SIZE, shuffle=True)
test_loader = data_utils.DataLoader(test_data, batch_size=BATCH_SIZE, shuffle=True)

resnet = resnet18(pretrained=True)

# Changing the input and output layers to fit the binary classification task
resnet.conv1 = nn.Conv2d(in_channels=1, out_channels=64, kernel_size=(7, 7),
                         stride=(2, 2), padding=(3, 3), bias=False)
resnet.fc = nn.Linear(in_features=512, out_features=2)

if torch.cuda.is_available():
    resnet = resnet.to('cuda')

# Defining loss function and optimizer
loss_function = nn.CrossEntropyLoss()
optimizer = optim.Adam(resnet.parameters(), lr=LEARNING_RATE)

train_batch_count = len(train_loader)
val_batch_count = len(val_loader)

train_loss_history = []
train_accuracy_history = []
val_loss_history = []
val_accuracy_history = []

for epoch in range(EPOCHS):
    print('Epoch ' + str(epoch + 1))

    ############################################################
    #  Training
    ############################################################
```

```
train_loss = 0
train_accuracy = 0

resnet.train()
for i, batch in enumerate(train_loader):
    data, labels = batch
    if torch.cuda.is_available():
        data, labels = data.cuda(), labels.cuda()

    optimizer.zero_grad()
    outputs = resnet(data)
    loss = loss_function(outputs, labels)

    loss.backward()
    optimizer.step()
    train_loss += loss.item()

    # Calculating accuracy
    total = labels.size(0)
    _, predicted = torch.max(outputs, 1)
    correct = (predicted == labels).sum().item()
    accuracy = correct / total
    train_accuracy += accuracy

train_loss_history.append(train_loss / train_batch_count)
train_accuracy_history.append(train_accuracy / train_batch_count)

################################################################
#  Validation
################################################################

validation_loss = 0
validation_accuracy = 0

resnet.eval()
for i, batch in enumerate(val_loader):
    data, labels = batch
    if torch.cuda.is_available():
        data, labels = data.cuda(), labels.cuda()

    outputs = resnet(data)
    loss = loss_function(outputs, labels)
    validation_loss += loss.item()

    # Calculating accuracy
    total = labels.size(0)
    _, predicted = torch.max(outputs, dim=1)
    correct = (predicted == labels).sum().item()
    accuracy = correct / total
    validation_accuracy += accuracy
```

```
        val_loss_history.append(validation_loss / val_batch_count)
        val_accuracy_history.append(validation_accuracy / val_batch_count)

        result_df = result_df.append({'Fold': fold + 1,
                                      'Epoch': epoch + 1,
                                      'Train loss': train_loss_history[epoch],
                                      'Train Accuracy': train_accuracy_history[epoch],
                                      'Val loss': val_loss_history[epoch],
                                      'Val Accuracy': val_accuracy_history[epoch]},
                                     ignore_index=True)

        print('Training loss: {:.4f}, training accuracy: {:.4f}'.format(
            round(train_loss_history[epoch], 4),
            round(train_accuracy_history[epoch], 4)))
        print('Validation loss: {:.4f}, validation accuracy: {:.4f}'.format(
            round(val_loss_history[epoch], 4),
            round(val_accuracy_history[epoch], 4)))
        print('\n')

    # Plotting loss and accuracy change in train and validation over epochs
    fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))

    ax[0].plot(range(1, EPOCHS + 1), train_loss_history, c='red', label='Train loss')
    ax[0].plot(range(1, EPOCHS + 1), val_loss_history, c='orange', label='Val loss')
    ax[0].set_ylabel('Loss')
    ax[0].set_xlabel('Epoch')
    ax[0].set_title('Loss change during training')

ax[1].plot(range(1, EPOCHS + 1), train_accuracy_history,
           c='green', label='Train accuracy')
ax[1].plot(range(1, EPOCHS + 1), val_accuracy_history,
           c='blue', label='Val accuracy')
    ax[1].set_ylabel('Accuracy')
    ax[1].set_xlabel('Epoch')
    ax[1].set_title('Accuracy change during training')
    fig.legend()
    plt.show()

    ################################################################
    #  Testing
    ################################################################
    test_loss, test_accuracy, metrics = evaluate_model(resnet, test_loader)
    tpr, fpr, prec, tnr, f1, auc, mcc = metrics
    print('\nTest loss: {:.4f}, test accuracy: {:.4f}'.format(
        round(test_loss, 4), round(test_accuracy, 4)))
    print('True positive rate (recall): {:.4f}'.format(tpr))
    print('False positive rate: {:.4f}'.format(fpr))
    print('Precision: {:.4f}'.format(prec))
    print('True negative rate: {:.4f}'.format(tnr))
    print('F1: {:.4f}'.format(f1))
    print('ROC-AUC: {:.4f}'.format(auc))
```

```
    print('MCC: {:.4f}'.format(mcc))
    print('\n')
    result_df = result_df.append({'Fold': fold + 1,
                                  'Epoch': epoch + 1,
                                  'Recall': tpr,
                                  'Fall-out': fpr,
                                  'Precision': prec,
                                  'Specificity': tnr,
                                  'F1': f1,
                                  'ROC-AUC': auc,
                                  'MCC': mcc,
                                  'Train loss': train_loss_history[-1],
                                  'Train Accuracy': train_accuracy_history[-1],
                                  'Val loss': val_loss_history[-1],
                                  'Val Accuracy': val_accuracy_history[-1],
                                  'Test loss': test_loss,
                                  'Test Accuracy': test_accuracy},
                                 ignore_index = True)
    fold = fold + 1

# Word classification
df_w = df[df['sound_u'].isna()]
df_w = df_w.sample(frac=1).reset_index(drop=True)
df_w.shape

df_w['duration'].quantile(q=0.99) # value at 99th quantile

tqdm.pandas()
df_w['audio_data'] = df_w['filepath'].progress_apply(lambda x: load_data(x, 2.4))

result_df = pd.DataFrame(columns=['Fold', 'Epoch', 'Recall', 'Fall-out', 'Precision',
                                  'Specificity', 'F1', 'ROC-AUC', 'MCC',
                                  'Train loss', 'Train Accuracy',
                                  'Val loss', 'Val Accuracy',
                                  'Test loss', 'Test Accuracy'])

# Word classification [with Test Metrics]
fold = 0

for train_index, test_index in stratified_k_fold.split(df_w, df_w['label']):
    print('---------------- Fold ' + str(fold + 1) + ' ----------------')

    # Splitting data into 80% train, 10% validation and 10% test
    train_df = df_w.iloc[train_index]
    val_test_df = df_w.iloc[test_index]

    val_df, test_df = train_test_split(val_test_df, stratify=val_test_df['label'],
                                       test_size=0.5, random_state=1, shuffle=True)

    train_audio_data = torch.Tensor(np.stack(train_df['audio_data'], axis=0))
    train_labels = torch.Tensor(np.stack(train_df['label'], axis=0)).long()
```

```
val_audio_data = torch.Tensor(np.stack(val_df['audio_data'], axis=0))
val_labels = torch.Tensor(np.stack(val_df['label'], axis=0)).long()

test_audio_data = torch.Tensor(np.stack(test_df['audio_data'], axis=0))
test_labels = torch.Tensor(np.stack(test_df['label'], axis=0)).long()

train_data = data_utils.TensorDataset(train_audio_data, train_labels)
val_data = data_utils.TensorDataset(val_audio_data, val_labels)
test_data = data_utils.TensorDataset(test_audio_data, test_labels)

train_loader = data_utils.DataLoader(train_data, batch_size=BATCH_SIZE, shuffle=
                                                    True)
val_loader = data_utils.DataLoader(val_data, batch_size=BATCH_SIZE, shuffle=True)
test_loader = data_utils.DataLoader(test_data, batch_size=BATCH_SIZE, shuffle=True)

resnet = resnet18(pretrained=True)

# Changing the input and output layers to fit the binary classification task
resnet.conv1 = nn.Conv2d(in_channels=1, out_channels=64, kernel_size=(7, 7),
                         stride=(2, 2), padding=(3, 3), bias=False)
resnet.fc = nn.Linear(in_features=512, out_features=2)

if torch.cuda.is_available():
    resnet = resnet.to('cuda')

# Defining loss function and optimizer
loss_function = nn.CrossEntropyLoss()
optimizer = optim.Adam(resnet.parameters(), lr=LEARNING_RATE)

train_batch_count = len(train_loader)
val_batch_count = len(val_loader)

train_loss_history = []
train_accuracy_history = []
val_loss_history = []
val_accuracy_history = []

for epoch in range(EPOCHS):
    print('Epoch ' + str(epoch + 1))

    ##############################################################
    #  Training
    ##############################################################

    train_loss = 0
    train_accuracy = 0

    resnet.train()
    for i, batch in enumerate(train_loader):
        data, labels = batch
```

```
        if torch.cuda.is_available():
            data, labels = data.cuda(), labels.cuda()

        optimizer.zero_grad()
        outputs = resnet(data)
        loss = loss_function(outputs, labels)

        loss.backward()
        optimizer.step()
        train_loss += loss.item()

        # Calculating accuracy
        total = labels.size(0)
        _, predicted = torch.max(outputs, 1)
        correct = (predicted == labels).sum().item()
        accuracy = correct / total
        train_accuracy += accuracy

    train_loss_history.append(train_loss / train_batch_count)
    train_accuracy_history.append(train_accuracy / train_batch_count)

    ###################################################################
    #  Validation
    ###################################################################

    validation_loss = 0
    validation_accuracy = 0

    resnet.eval()
    for i, batch in enumerate(val_loader):
        data, labels = batch
        if torch.cuda.is_available():
            data, labels = data.cuda(), labels.cuda()

        outputs = resnet(data)
        loss = loss_function(outputs, labels)
        validation_loss += loss.item()

        # Calculating accuracy
        total = labels.size(0)
        _, predicted = torch.max(outputs, dim=1)
        correct = (predicted == labels).sum().item()
        accuracy = correct / total
        validation_accuracy += accuracy

    val_loss_history.append(validation_loss / val_batch_count)
    val_accuracy_history.append(validation_accuracy / val_batch_count)

    result_df = result_df.append({'Fold': fold + 1,
                                  'Epoch': epoch + 1,
                                  'Train loss': train_loss_history[epoch],
```

```
                                        'Train Accuracy': train_accuracy_history[epoch],
                                        'Val loss': val_loss_history[epoch],
                                        'Val Accuracy': val_accuracy_history[epoch]},
                                       ignore_index=True)

        print('Training loss: {:.4f}, training accuracy: {:.4f}'.format(
            round(train_loss_history[epoch], 4),
            round(train_accuracy_history[epoch], 4)))
        print('Validation loss: {:.4f}, validation accuracy: {:.4f}'.format(
            round(val_loss_history[epoch], 4),
            round(val_accuracy_history[epoch], 4)))
        print('\n')

    # Plotting loss and accuracy change in train and validation over epochs
    fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))

    ax[0].plot(range(1, EPOCHS + 1), train_loss_history, c='red', label='Train loss')
    ax[0].plot(range(1, EPOCHS + 1), val_loss_history, c='orange', label='Val loss')
    ax[0].set_ylabel('Loss')
    ax[0].set_xlabel('Epoch')
    ax[0].set_title('Loss change during training')

    ax[1].plot(range(1, EPOCHS + 1), train_accuracy_history,
               c='green', label='Train accuracy')
    ax[1].plot(range(1, EPOCHS + 1), val_accuracy_history,
               c='blue', label='Val accuracy')
    ax[1].set_ylabel('Accuracy')
    ax[1].set_xlabel('Epoch')
    ax[1].set_title('Accuracy change during training')
    fig.legend()
    plt.show()

    ###################################################################
    #  Testing
    ###################################################################
    test_loss, test_accuracy, metrics = evaluate_model(resnet, test_loader)
    tpr, fpr, prec, tnr, f1, auc, mcc = metrics
    print('\nTest loss: {:.4f}, test accuracy: {:.4f}'.format(
        round(test_loss, 4), round(test_accuracy, 4)))
    print('True positive rate (recall): {:.4f}'.format(tpr))
    print('False positive rate: {:.4f}'.format(fpr))
    print('Precision: {:.4f}'.format(prec))
    print('True negative rate: {:.4f}'.format(tnr))
    print('F1: {:.4f}'.format(f1))
    print('ROC-AUC: {:.4f}'.format(auc))
    print('MCC: {:.4f}'.format(mcc))
    print('\n')
    result_df = result_df.append({'Fold': fold + 1,
                                  'Epoch': epoch + 1,
                                  'Recall': tpr,
                                  'Fall-out': fpr,
```

```
                                        'Precision': prec,
                                        'Specificity': tnr,
                                        'F1': f1,
                                        'ROC-AUC': auc,
                                        'MCC': mcc,
                                        'Train loss': train_loss_history[-1],
                                        'Train Accuracy': train_accuracy_history[-1],
                                        'Val loss': val_loss_history[-1],
                                        'Val Accuracy': val_accuracy_history[-1],
                                        'Test loss': test_loss,
                                        'Test Accuracy': test_accuracy},
                                       ignore_index = True)
    fold = fold + 1

# SVM model with MFCC
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score,
                                            roc_auc_score, matthews_corrcoef

param_grid = {'C': [0.1, 1, 10, 100],
              'gamma': [0.0001, 0.001, 0.1, 1],
              'kernel': ['rbf', 'poly']}

svc = svm.SVC(probability=True)
model = GridSearchCV(svc, param_grid)

# Vowel classification
train_df, val_df = train_test_split(df_v, stratify=df_v['label'], test_size=0.2,
                                    random_state=1, shuffle=True)

x_train = []
y_train = []
for index, row in tqdm(train_df.iterrows(), total=train_df.shape[0]):
    wav, sr = librosa.load(row['filepath'])
    wav = get_fixed_length_audio(wav, sr, 0.3)
    mfcc = librosa.feature.mfcc(wav, n_mfcc=13)
    x_train.append(np.hstack(mfcc))
    y_train.append(row['label'])

x_val = []
y_val = []
for index, row in tqdm(val_df.iterrows(), total=val_df.shape[0]):
    wav, sr = librosa.load(row['filepath'])
    wav = get_fixed_length_audio(wav, sr, 0.3)
    mfcc = librosa.feature.mfcc(wav, n_mfcc=13)
    x_val.append(np.hstack(mfcc))
    y_val.append(row['label'])

clf = svm.SVC(kernel='rbf', probability=True)
clf.fit(x_train, y_train)
```

```
print('Accuracy: {:.4f}'.format(accuracy_score(clf.predict(x_val), y_val)))
print('Precision: {:.4f}'.format(precision_score(clf.predict(x_val), y_val)))
print('Recall: {:.4f}'.format(recall_score(clf.predict(x_val), y_val)))
print('F1 score: {:.4f}'.format(f1_score(clf.predict(x_val), y_val)))
print('ROC-AUC: {:.4f}'.format(roc_auc_score(clf.predict(x_val), y_val)))
print('MCC: {:.4f}'.format(matthews_corrcoef(clf.predict(x_val), y_val)))

# Word classification
train_df, val_df = train_test_split(df_w, stratify=df_w['label'], test_size=0.2,
                                    random_state=1, shuffle=True)

x_train = []
y_train = []
for index, row in tqdm(train_df.iterrows(), total=train_df.shape[0]):
    wav, sr = librosa.load(row['filepath'])
    wav = get_fixed_length_audio(wav, sr, 0.3)
    mfcc = librosa.feature.mfcc(wav, n_mfcc=13)
    x_train.append(np.hstack(mfcc))
    y_train.append(row['label'])

x_val = []
y_val = []
for index, row in tqdm(val_df.iterrows(), total=val_df.shape[0]):
    wav, sr = librosa.load(row['filepath'])
    wav = get_fixed_length_audio(wav, sr, 0.3)
    mfcc = librosa.feature.mfcc(wav, n_mfcc=13)
    x_val.append(np.hstack(mfcc))
    y_val.append(row['label'])

clf = svm.SVC(kernel='rbf', probability=True)
clf.fit(x_train, y_train)

print('Accuracy: {:.4f}'.format(accuracy_score(clf.predict(x_val), y_val)))
print('Precision: {:.4f}'.format(precision_score(clf.predict(x_val), y_val)))
print('Recall: {:.4f}'.format(recall_score(clf.predict(x_val), y_val)))
print('F1 score: {:.4f}'.format(f1_score(clf.predict(x_val), y_val)))
print('ROC-AUC: {:.4f}'.format(roc_auc_score(clf.predict(x_val), y_val)))
print('MCC: {:.4f}'.format(matthews_corrcoef(clf.predict(x_val), y_val)))

# SVM model with spectrograms
svc = svm.SVC(probability=True)
model = GridSearchCV(svc, param_grid)

# Vowel classification
train_df, val_df = train_test_split(df_v, stratify=df_v['label'], test_size=0.2,
                                    random_state=1, shuffle=True)

x_train = []
y_train = []
for index, row in tqdm(train_df.iterrows(), total=train_df.shape[0]):
```

```
    x_train.append(np.hstack(np.hstack(row['audio_data'])))
    y_train.append(row['label'])

x_val = []
y_val = []
for index, row in tqdm(val_df.iterrows(), total=val_df.shape[0]):
    x_val.append(np.hstack(np.hstack(row['audio_data'])))
    y_val.append(row['label'])

clf = svm.SVC(kernel='rbf', probability=True)
clf.fit(x_train, y_train)

print('Accuracy: {:.4f}'.format(accuracy_score(clf.predict(x_val), y_val)))
print('Precision: {:.4f}'.format(precision_score(clf.predict(x_val), y_val)))
print('Recall: {:.4f}'.format(recall_score(clf.predict(x_val), y_val)))
print('F1 score: {:.4f}'.format(f1_score(clf.predict(x_val), y_val)))
print('ROC-AUC: {:.4f}'.format(roc_auc_score(clf.predict(x_val), y_val)))
print('MCC: {:.4f}'.format(matthews_corrcoef(clf.predict(x_val), y_val)))

# Word classification
train_df, val_df = train_test_split(df_w, stratify=df_w['label'], test_size=0.2,
                                    random_state=1, shuffle=True)

x_train = []
y_train = []
for index, row in tqdm(train_df.iterrows(), total=train_df.shape[0]):
    x_train.append(np.hstack(np.hstack(row['audio_data'])))
    y_train.append(row['label'])

x_val = []
y_val = []
for index, row in tqdm(val_df.iterrows(), total=val_df.shape[0]):
    x_val.append(np.hstack(np.hstack(row['audio_data'])))
    y_val.append(row['label'])

clf = svm.SVC(kernel='rbf', probability=True)
clf.fit(x_train, y_train)

print('Accuracy: {:.4f}'.format(accuracy_score(clf.predict(x_val), y_val)))
print('Precision: {:.4f}'.format(precision_score(clf.predict(x_val), y_val)))
print('Recall: {:.4f}'.format(recall_score(clf.predict(x_val), y_val)))
print('F1 score: {:.4f}'.format(f1_score(clf.predict(x_val), y_val)))
print('ROC-AUC: {:.4f}'.format(roc_auc_score(clf.predict(x_val), y_val)))
print('MCC: {:.4f}'.format(matthews_corrcoef(clf.predict(x_val), y_val)))
```

Appendix B

Implemented code with outputs can be found in Google Colab notebook: https://colab.research.google.com/drive/1X7pxeLbroBa6sQwBW5doFcGnJWB1R4qJ?usp=sharing.