**UNIVERSITY OF VAASA**

**SCHOOL OF TECHNOLOGY AND INNOVATIONS**

**WIRELESS INDUSTRIAL AUTOMATION**

Stavros Grigoriadis

# CONVOLUTIONAL NEURAL NETWORKS FOR ACCENT CLASSIFICATION

Master's thesis for the degree of Master of Science in Technology; left for assessment on 1 Feb. 2019 in Vaasa.

Supervisor Professor Mohammed Elmusrati

Instructor Professor Mohammed Elmusrati

ACKNOWLEDGEMENTS

First of all, I would like to thank deeply Professor Mohammed Elmusrati of the School of Technology and Innovations at the University of Vaasa for his guidance in choosing the topic of my thesis, his motivation and for being one of my mentors during my studies.

Secondly, I want to express my gratitude to the University of Vaasa for believing in me and giving me the chance to study again after so many years and expand my horizons. It was really an honour to me.

Last, but not least, I want to thank my partner and my parents for providing me with support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them. From the bottom of my heart, thank you.

TABLE OF CONTENTS

TABLE OF FIGURES AND TABLES

ABBREVIATIONS

| | |
|---|---|
| ANN | Artificial Neural Network |
| ASR | Automatic Speech Recognition |
| BP | Back-Propagation |
| CNN | Convolutional Neural Network |
| FIR | Finite Impulse Response |
| FNN | Feed-forward Neural Network |
| GPU | Graphics Processing Unit |
| MFCC | Mel-Frequency Cepstral Coefficient |
| MLP | Multilayer Perceptron |
| ReLU | Rectified Linear Unit |
| WAV | Windows Wave Audio Format |

**UNIVERSITY OF VAASA**
**School of Technology and Innovations**
**Author:** Stavros Grigoriadis
**Topic of the Thesis:** Convolutional Neural Networks for Accent Classification
**Supervisor:** Professor Mohammed Elmusrati
**Instructor:** Professor Mohammed Elmusrati
**Degree:** Master of Science in Technology
**Major of Subject:** Wireless Industrial Automation
**Year of Entering the University:** 2016
**Year of Completing the Thesis:** 2019 **Pages:** 143

## ABSTRACT

Speech recognition systems have been extensively improved over the years. However, accent classification remains a highly challenging task. Accent classification technology can be a great benefit to automatic speech recognition applications, telephony based service centres, immigration offices and in military operations. The application of convolutional neural networks has been an efficient and effective way to solve the accent recognition problem.

In this thesis the accent classification task is approached by the application of two convolutional neural networks. The difference between them can be seen at their activation functions. The work includes a dataset of native speakers of four different languages (Chinese, Spanish, English, Arabic) who read a certain elicitation paragraph in English. The chosen paragraph contains common English words which cover in majority the sounds of English language. The feature extraction is based on the Mel-Frequency Cepstral Coefficients, in particular the first 13 coefficients are used. The MFCC has proved to be one of the best representations of human voice in terms of audio signal processing. The convolutional neural networks manipulate the audio signals of the speakers in the form of 2 dimensional images, making them an effective approach for accent classification. The thesis contains an extensive presentation of the accuracy, validation loss and confusion matrices of each cases between training and test samples and the results of each model for the reader to compare and decide which model to apply for a similar application. Appendix 1 contains the original and modified source code for the implementation of the proposed convolutional neural networks in order to solve the accent classification problem.

**KEYWORDS:** Accent Classification, CNN, Machine Learning, MFCC, Python.

# 1. INTRODUCTION

Speech is one of the most important media of communication between humans. Humans use it to express their opinions as well as their feelings and moods. The adaptation, usage, processing and understanding of human speech by computers can be considered a significant challenge in modern societies. Although many achievements and improvements have been made in the automatic speech recognition (ASR) application area, and more specific with its application in Apple's Siri, Google's Assistant and Amazon's Alexa, the issue of accent recognition seems to be a problem for these programs as they can only understand the American English accent. Specifically the above applications can recognise speakers of American English with high accuracy but may fail in recognising speakers of English with Scottish or Irish accent (Najafian, Safavi, Hanani & Russell 2014). The problem seems to be more apparent when the speakers are not native English.

The problem of distinguishing the accent of the speaker can be called accent recognition and the applications of using the technology to identify the origin of a speaker are implementing algorithms in order to achieve the accent classification of the speakers. The accent classification task is quite challenging because each speaker has his own speaking style, for example depending on the place where the speaker was born and his environment, and his accent would have the same characteristics with the citizens living in the same region.

The application of accent classification systems is significant and quite useful in speech technology and can be seen in other areas including speech recognition systems. This technology can be applied in telephone centre systems and services; by identifying the origin of the speaker, a certain employee with a similar accent can provide his services to the caller. Another area that can benefit from accent classification systems is at boarders of countries and immigration offices; the agencies will be able to recognise in high accuracy the origins of the immigrants by their speech.

The main purpose of this thesis is to tackle a part of the above problem, being the accent recognition and the estimation of the origin of the speaker, who reads a specific text in English. A system using machine learning algorithms and more specific two convolutional neural network architectures was implemented and it is proposed in this thesis in order to classify and accomplish as accurately as possible the accent recognition of a speaker. There have been implemented different approaches during the implementation of the accent recognition system, concerning mainly the tuning of parameters of the network and adjusting the percentage between the training and test samples. Each of the above approaches follows a certain systematic way with their advantages and disadvantages and they will be presented and discussed in this work.

It is noteworthy mentioning that the proposed system is text dependent and it can recognise speakers whose native language is Chinese, Spanish, English or Arabic. Therefore, the type of the classification that is used in this thesis is multi-classification. In addition, the approach of supervised learning is applied, where each input sample of a speaker has a certain output label which corresponds to his accent. Besides, feature extraction seemed to be an important process in order to represent in the best possible way the human voice.

The approach to solve the accent classification problem was based on two different convolutional neural networks. One may be confused by the above approach because convolutional neural networks are efficient and effective in image classification. The interesting part of the thesis is that the audio signals of each speaker in the system are treated like a two-dimensional image.

Moreover, for each approach experiments and their results are discussed. The accuracy, validation and confusion matrices of every possible combination between the training and test samples are presented. The reader can focus on each case and have a general idea of the effectiveness of the current case.

The thesis is organised as follows: In Chapter 2 various machine learning applications are presented and the role of machine learning and its techniques are discussed. The fea-

ture extraction process and the Mel-Frequency Cepstral Coefficients that have been used are examined. The chapter also contains the usage of a confusion matrix, the terms of generalisation, memorisation and overfitting. In Chapter 3, neural networks and convolutional neural networks are discussed. In particular, the chapter contains theory about feed-forward neural networks, multilayer perceptrons, back-propagation and activation functions. Terms of consisting a convolutional neural network are also presented. Topics such as the convolutional layer, the 2D convolutional layer, the receptive field, the pooling layer and the fully connected layer are analysed. The chapter also contains the terms of dropout, loss functions in CNNs and the soft-max loss. The system architecture and the implementation are presented in Chapter 4. Specifically, the reader can find information about the architecture of the proposed system, the dataset that is used and the stages of pre-processing and feature extraction. Moreover, the architecture of the convolutional neural networks that are used and the program implementation are explained. Chapter 5 consists of the experiments and the results of the proposed system. The experiment setup is discussed while the experiments of the two different CNNs for each case of training and test samples are presented in detail. Finally, the conclusion and the future work are considered in Chapter 6 and the source code of the project is included in Appendix 1.

# 2. MACHINE LEARNING

Machine learning is a term used in the broader area of Artificial Intelligence and it is referred to the usage of algorithms and statistical techniques of a system in order to "*learn*" or acquire knowledge through mapping inputs and outputs of a series of data without explicitly being programmed (Bishop 2006:2). The term was conceived by the American pioneer in computer gaming and artificial intelligence Arthur Lee Samuel in 1959.

Machine learning can be described as the process of finding the best possible approximation that can be used as a solution to a problem. Based on a model defined by an expert human the aim of machine learning is to propose as much as accurately solutions to given problems. The system using machine learning algorithms is provided with inputs as datasets and desired outputs. Examples of using machine learning techniques can be found in everyday life such as recommender systems for online shopping, e-mail filtering such as defining which e-mail is spam and which is not, fraud detection in transactions bank systems, speech recognition, hand written recognition, computer vision, medical diagnosis, smart systems and more. In this thesis the area of machine learning concerned the ability of a computer program to recognize and classify the four different accents of speakers reading a certain text written in English.

One of the key elements of machine learning is the information and its capacity concerning each problem field. Almost any material in this world can be represented as a series of numbers which contains information in fields such as economic, social and biological informatics, thermodynamics and quantum information, etc. Information theory is an important term in machine learning area. Cloud Shannon proposed that the information content could be considered as a function in its uncertainty in 1948. More specific the information content of an event is estimated to be high if the event has a low probability to occur.

According to Flach (2012: 3) "machine learning is the systematic study of algorithms and systems that improve their knowledge or performance with experience". The experience is referred to the correct labelled input data of the system and the term performance to the ability of the system to classify the data in a classification problem for instance.

Figure 1 depicts an overview of the process that is used from machine learning to address a task. The objects in this thesis are represented by the audio files of the speakers reading a certain text in English. Each speaker has its own accent and the features of the speeches can be represented by taking the Mel-Frequency Cepstral Coefficients (MFCC). Next, the training data are fed to the system and into the learning algorithm which then produce the model. The model addresses the task of the system and this is the place where a mapping between the features and the desired output will be achieved.

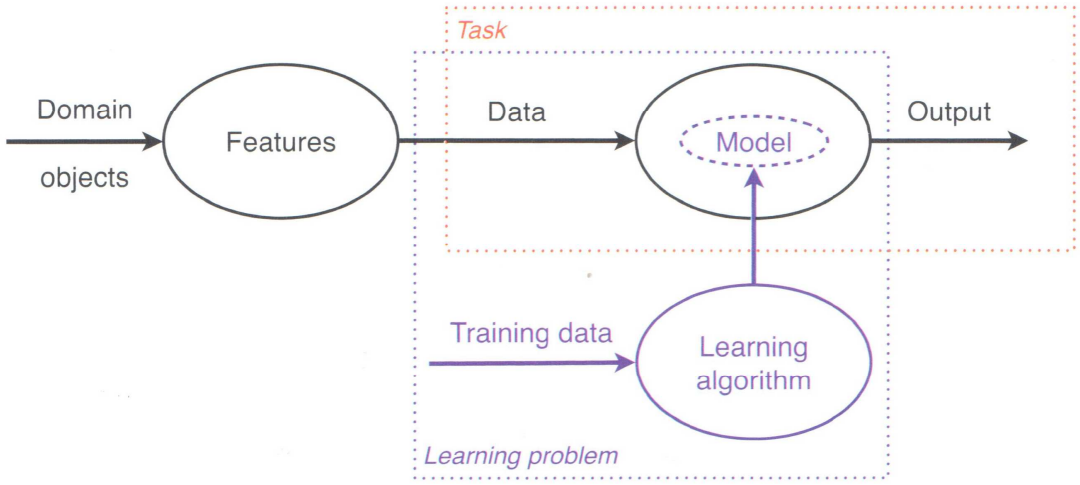


**Figure 1.** Machine Learning process to address a task (Flach 2012: 11).

## 2.1. Machine learning applications

The range of the applications of machine learning is wide. In this section a part of the applications is presented. Firstly, machine learning algorithms can be applied in banking

processes. Banks refer to their data to build models in order to use them in fraud detection, loan plans for customers, credit application as well as stock market.

Secondly, machine learning techniques can be used in fields of manufacturing, medical and autonomy machines. Specifically in manufacturing, processes can be optimized, controlled and can be used in troubleshooting. In medicine, medical diagnosis and drug manufacturing can be applied. Concerning autonomy machines, the application of autonomous cars is popular nowadays as well as air drones.

Last but not least, there are applications of machine learning with smart systems such as smart building, smart cities and smart grids as well as in telecommunication networks where patterns are analysed for network and quality of service optimization. The applications in pattern recognition are also important which include speech recognition, handwritten recognition, biometric recognition, etc.

## 2.2. The role of big data

The term *big data* can be considered as a large capacity of data or more specific information generated from different sources such as mobile devices, microphones, cameras, radio-frequency identification readers, wireless sensor networks, software logs, etc (Hellerstein 2008). The acquisition of big data and its appropriate usage and analysis of the owners can be powerful. More specific, companies that hold big data use machine learning algorithms to analyse consumer behaviour and in extension to adapt their production plans in order to maximise their profits.

An example could be the data collected by a supermarket chain about its customers' needs and information. At first customers' behaviour in general may seem random but on a second thought it can be predicted, on the basis of past purchases. Through this phase the company can have valuable information about its customers concerning their preferences, which may have correlations between specific products. On the other hand, customers find the recommendations of the companies' systems about products that

were bought from other customers with similar preferences helpful. The above examples show that both producers and consumers can benefit from machine learning applications. The process of applying machine learning algorithms on big datasets is called *data mining*.

In conventional computer programs the programmer should build and follow a certain algorithm and program in order to solve the given problem. In contrast, in the machine learning field a programmer cannot follow a certain algorithm to solve a problem, but he or she has to find as much data as possible and create a system, which uses the data as inputs that correspond to a specific labelled output. This method is used mainly in supervised learning and in the proposed system in this thesis. The various machine learning techniques will be presented in following section.

Following the above logic, the most important ingredient of a successful classification system is the number of the input data. Given sufficient input data and the mapping between input and output, a system can be modelled and trained in order to predict as accurately as possible a good approximation answer or output for a given input. The approximation of the system is usually not 100% accurate, depending on the field of the problem, but a rule of thumb is that the system will be able to detect specific patterns and regularities (Alpaydin 2014: 2). These patterns can give the programmer some hints of the elements of the algorithm used by the system. If the model under training provides high accuracy then it can be assumed that depending on the input data gathered from the near past, a good approximation and prediction can be made from the system for future input data.

## 2.3. Types of machine learning techniques

There are a few machine learning techniques that are used in various domains to solve specific problems. In this section these machine learning techniques will be presented.

### 2.3.1. Supervised Learning

Supervised learning is a machine learning technique in which the desired outcome is to find a function that maps specific inputs with outputs with the use of labelled training data. For example if the input is *X* and the output is *Y*, then the aim of supervised learning is to learn the mapping from the input *X* to the output *Y*. Usually the model that is followed has the form:

$$y = g(x \mid \theta), \tag{1}$$

*g* is the model and $\theta$ are its parameters. It is important to note that regression and classification belong to this type of machine learning. *Y* is a class if classification is used or a number if regression is used. The machine learning application should optimize the parameters *theta* in such way that the approximation error is minimised and the estimations are close enough to the correct values of the training set (Alpaydin 2014: 9). An example of a regression problem is represented in Figure 2 where the fitted function has the form:

$$y = wx + w_0, \tag{2}$$

the training dataset corresponds to used cars where the input attribute is the mileage of the car and the output is its price.

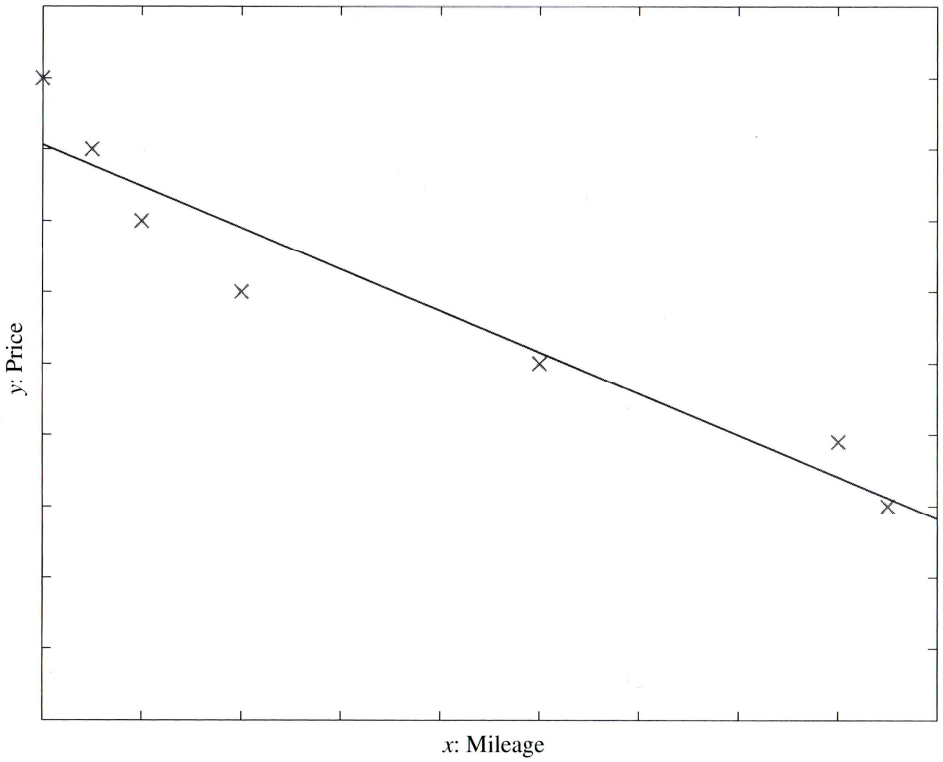


**Figure 2.** Dataset of used cars and their mileage (Alpaydin 2014: 10).

Supervised learning with classification is the type of machine learning technique that is used in the current thesis.

### 2.3.2. Unsupervised Learning

Unsupervised learning on the other hand is a machine learning technique that learns from data that has no labels. The supervisor in this learning is the input data and the goal is to find similarities and regularities in the input. Usually the term of density estimation is used and there can be identified a structure in the input space that contains certain patterns. In this technique the term of clustering is used. The aim is to find clusters or groupings of input. Clustering can be applied in many fields such as customer segmentations in companies, customer relationship management, image compression, document clustering as well as in bioinformatics (Alpaydin 2014: 11–13).

### 2.3.3. Reinforcement Learning

Reinforcement learning is the type of machine learning applications where the output of the system is a sequence of actions and a policy with sequence of correct actions is desired to be followed. The machine learning applications can learn from previous good policies and try to adapt their policies in that manner.

The application of reinforcement learning is wide and it can be seen in game theory, control theory, information technology, multi-agents systems, genetic algorithms, etc. For example in chess the number of rules are small but the number of possible moves of each player is large. Another example could be the navigation of a robot in an environment in order to search for a goal location. The robot can move to any direction, but the selection of the policy of the sequence of moves that accomplish this goal as quickly as possible is important (Alpaydin 2014: 13).

## 2.4. Inductive and deductive learning

Humans can learn or be taught based on two types of methods, induction and deduction. Induction can be observed when a person has in his possession training examples, labels or terms of a certain event and then construct an outcome. For example when a parent wants to teach to his kid that playing with the fire is very dangerous, he can show it and use photographs, videos or any evidence of fire accidents or burnt persons in order for the kid to understand the danger of playing with the fire.

On the other hand, the deductive learning can be achieved in the opposite way of induction. In deduction the person can learn an outcome of an event through his own experience. In the same example with the parent and the kid that was mentioned before, in case of deduction the parent would not act as a supervisor but he would let the kid to play with the fire and get burnt. When the kid will experience the outcome of its action it will learn and remember that playing with fire is dangerous and should be avoided.

In most machine learning applications, especially the ones that apply supervised learning, the type of learning that is used is inductive learning. The systems have training examples as input data with labelled and specific output.

## 2.5. Feature Extraction

Features in machine learning applications are considered to be one of the essential parts of a system and they can contribute to a large extent to the accuracy and successful prediction of the applications. The features of a system represent the measurements of the input data and these measurements in the proposed machine learning application are numerical and more specific real numbers.

Therefore, the process of feature extraction is a fundamental part of the application and the decision to use the correct feature extraction, or the most representative measurement of the input data, was a challenging task.

### 2.5.1. Mel-Frequency Cepstral Coefficients (MFFCs)

The input data of the proposed system consists of wave audio files of speakers reading a certain text in English. It was known that the wave audio signals could be analysed in time or in frequency domain. The analysis in time domain produces a high dimensionality in feature terms while the analysis in frequency domain with the help of feature extraction through Mel-Frequency Cepstral Coefficients can achieve a significant reduction in feature extraction of the input data.

A more detailed explanation that the dimensionality in time frequency is high is followed. If one can consider a 4 seconds of wave audio file sampled at 8kHz then it will contain 32000 samples which correspond to the number of variables that will be used in the input nodes representing the features of the current signal. The usage of MFCC is considered to be a useful feature extraction algorithm for human voice in speech recognition applications (Huang, Acero & Hon 2001: 423–426). Besides, the MFCCs can be

used to map as close as possible the human auditory perception with frequencies and they are essential elements for speech recognition systems (Elminir, Abu ElSoud & Abou El-Maged 2012).

In addition, according to Valaki & Jethva (2016), the advantages of MFCC include the good levels of discrimination and low correlation between the coefficients. They are not based on linear characteristics, which ensure the common characteristics with the human auditory system. It is significant to note also that the MFCCs can capture the important phonetic characteristics of humans.

It is known from psychological research that the human hearing does not correspond to a linear scale and each tone with a frequency *f* can be mapped to a scale in *Hz* which is called the *Mel scale*. The Mel-frequency scale is linear frequency spacing below 1 kHz and logarithmic spacing above 1 kHz. The idea of using MFCCs is that it can approximate closely the frequency response of human auditory system and the MFCCs contain the important phonetic features of human speech (Lokhande, Nehe & Vikhe 2012). Figure 3 depicts the computation process of the MFCC.

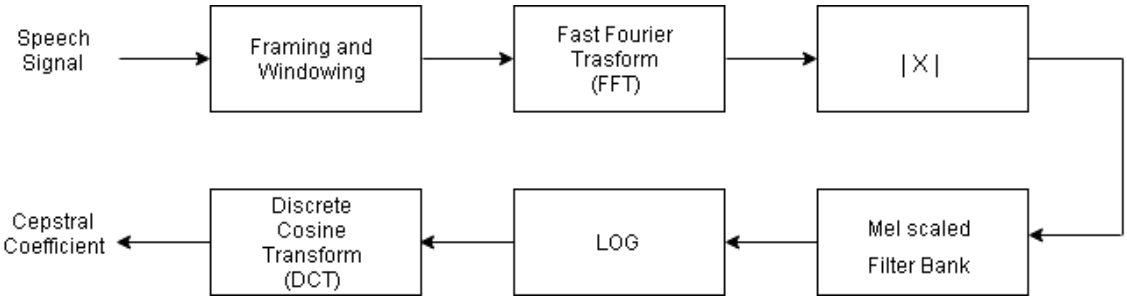


**Figure 3.** MFCC computation process.

As it can be seen from the Figure 3 the speech signal is going into a framing and windowing (usually a Hamming window) process and into a pre-emphasis filter. The next step is to take the Fast Fourier transform, which converts each frame of the input signal from time domain to frequency domain. Next follows the conversion of the scale frequency from linear to Mel scale and the logarithm of the results is calculated. The last

step is to take the discrete cosine transform of the log auditory spectrum in time domain and the result is the MFCC.

### 2.5.2. Mathematical representation of feature extraction

This section includes the mathematical representation of the feature extraction used in the proposed system with the help of MFCC. Firstly, it can be said that the pre-emphasis filtering in the previous section can be described by a kind of *finite impulse response* (FIR) that is used to provide an improvement in the energy of the high frequencies of the input signal and the following equation derives:

$$s[n] = x[n] - \alpha x[n-1], n = 1,2,..., N \quad , \tag{3}$$

where *x[n]* is the input signal at sample *n*, *s[n]* is the signal after the filtering and $\alpha$ is a parameter that adjusts the amount of filtering of the signal.

Secondly, the signal is converted from the time domain to frequency domain by using short time Fourier transform assuming that the signal over a short period of time is stationary and can be transferred to frequency domain. This can be achieved by the following expression:

$$X_\alpha[k] = \sum_{n=0}^{N-1} s[n] \cdot w_\alpha[n] \cdot e^{-i 2\pi k n / N} = \sum_{n=0}^{N-1} s[n] \cdot w_\alpha[n] \cdot e^{-i\omega k}, 0 \le k < N \quad , \tag{4}$$

where $w_\alpha$ *[n]* represents the window function and *i* an imaginary number. The window function in this case is a Hamming window and it can be expressed by the following equation:

$$w_\alpha[n] = \alpha - \beta \cos\left(\frac{2\pi n}{N-1}\right), 0 \le n < N, \alpha = 0.54, \beta = 1 - a = 0.46 \quad . \tag{5}$$

Besides, the human auditory system is more sensitive to sounds between 20 and 1000 Hz, which means that one cannot assign a signal the same scale at high frequencies as at lower frequencies. Thus, the conversion from Hertz scale to Mel scale can be achieved by the following formula:

$$mel = 2595\log_{10}\left(1+\frac{f}{700}\right), f > 1000, \tag{6}$$

and from Mel scale to Hertz scale:

$$f = 700\left(e^{mel/2595}-1\right), mel > 1000. \tag{7}$$

The next step is to define a filter bank with *M* filters (m =1,2,...,M) from the input window frame $x_\alpha[k]$. The filters are linear on Mel scale but non linear on Hertz scale and can be represented by the following expression:

$$M_m[k] = 1 - \left|\frac{k-\frac{N-1}{2}}{\frac{N-1}{2}}\right|, \tag{8}$$

where *N* is the length of the filter.

In addition the log-energy of each filter can be computed by:

$$S[m] = \ln\left[\sum_{k=0}^{N-1}\left|X_\alpha[k]\right|^2 M_m[k]\right], 0 < m \le M. \tag{9}$$

Finally, to get the Mel-frequency cepstrum coefficients the discrete cosine transform of the M filter outputs is used:

$$c[q] = \sum_{m=0}^{M-1} \left[ S[m] \cos\left( \frac{\pi q \left( m - \frac{1}{2} \right)}{M} \right) \right], 0 < m \leq M \quad . \tag{10}$$

The value of $M$ is between 24 and 40 and the first 13 MFCCs are computed, the value of $n$ is the number of window frames and $q$ is the number of MFCCs (Ma & Fokoue 2014).

## 2.6. Classification

The term classification refers to the identification of a number of categories that an observation belongs to. The classification is based on the training data and their mapping to the corresponding category they belong to. In machine learning the common classification types are the *binary classification* and the *multiclass classification*.

In *binary classification* there are examples of objects or data that are either belonging to the class or not. In this approach there are positive examples which means that the data belongs to a certain class and negative examples when the data does not belong to the class. On the other hand, in *multiclass classification* each data is mapped to a specific class. But in the same manner there are positive examples when the data (speaker's accent) belongs to a class and negative examples belonging to all other accents. The proposed system in this thesis represents a machine learning problem based on a *multiclass classification*. For instance, there are speakers that belong to one of four classes. The four classes consist of the four accents that are used (Chinese, Spanish, English and Arabic).

It is worth mentioning that the key element that describes the classification in both types of classification is the features of the data, which are derived from the feature extraction process that presented in the previous section. The features of each audio file is a matrix with 13 rows and 30 columns, which can be represented by the following matrix:

$$x = \begin{bmatrix} x_{11} & x_{21} & \dots & x_{301} \\ x_{12} & x_{22} & \dots & x_{302} \\ \dots & \dots & \dots & \dots \\ x_{113} & x_{213} & \dots & x_{3013} \end{bmatrix}. \tag{11}$$

There are 4 classes denoted by $C_i$ = 1,2,3,4. Each input instance belongs only to one of them and the training set is:

$$X = \left\{x^t, r^t\right\}_{t=1}^{N}, \tag{12}$$

where $r$ has 4 dimensions and

$$r_i^t = \begin{Bmatrix} 1, x^t \in C_i \\ 0, x^t \in C_j, j \neq i \end{Bmatrix} \tag{13}$$

(Aplaydin 2014: 22, 33)

Another approach of defining the model of accent classification is proposed by Chu, Lai & Le (2017) where "a speaker $s$ who has a native language $l_s$ in the set of all non-English languages $L$, given his n-second clip in the set of all clips $C$." The aim is to find a mapping $\Phi: C \rightarrow L$ such that the occurrence of prediction $\Phi(C_{s,n}) \neq l_s$ is minimized. The next step is to define a function $f$ that represents the number of prediction misses for all $C_{s,n} \in C_n$, for a subset $C_n \subseteq C_n$ :

$$f\left(\Phi, C_n\right) = \sum_{c_{s,n} \in C_n} \delta(\Phi(c_{s,n}), s), \tag{14}$$

where $\delta(x,y) = 1$ *if* $x \neq y$ or 0 otherwise.

According to Chu et al. (2017) the accent classification is an optimization problem where the objective is to find the mapping $\Phi^*$ for the clip set $C_n$ so that

$$\Phi^* = \arg_{\Phi}\min f(\Phi, C_n) \quad (15)$$

In the content of the proposed system given the audio clips of a speaker $s$, the goal is to classify the native language $l_s$ of the speaker $s$ to one of the four languages: Chinese, Spanish, English or Arabic.

## 2.7. Confusion matrix

There are cases where the accuracy of classification of a model is not a sufficient feature that indicates the real accuracy. In these cases the number of the observations in the input data is not an equal number in each class and also there may be more than two classes in the application. This can hold true sometimes in the proposed system. For the above reason a confusion matrix can be helpful which can show the performance of the classification used in the application.

In the case of *binary classification* a confusion matrix should have two rows and two columns. However, in this thesis a *multiclass classification* with four classes is used. Therefore, the structure of the confusion matrix consists of four rows and four columns. The confusion matrix lets the designer of a machine learning application check in detail if the algorithm used is giving good or bad results and extract information about his model. High values on the diagonal of the confusion matrix signals a successful classifier. Besides, if there are high off-diagonal elements in the matrix then this is a sign that mistakes are being made regularly in the dataset (Rogers & Girolami 2017: 200).

Each row of a confusion matrix corresponds to a predicted class and each column to an actual class. It is important to note that the total number of correct predictions regarding a class is included in the expected row for this class value and the predicted column.

Similarly, the total number of incorrect predictions for a class is included in the expected row for that class value and the predicted column.

In the section of presenting the results of the experiments of the proposed model, confusion matrices will be used in order to check the performance of the classification.

## 2.8. Generalisation, memorisation and overfitting

A system using machine learning algorithms must achieve great accuracy over the training and validating data in order to reach a good generalisation. The generalisation of the system means that the system successfully can map various inputs to correct outputs without memorisation. It is crucial for machine learning algorithms to offer high accuracy of a given model and acquire generalisation for a large and different input data.

On the other hand, the process of training a model is to aim reducing the loss function by adjusting the weights of the network and acquiring the best accuracy and generalisation. If the generalisation cannot be achieved, memorisation will take its place. Memorisation means that the system memorises the mappings between the inputs and outputs for a given set and when a different set of inputs is applied then the outputs will not be accurate. This event will result in wrong prediction and approximation of the output and overfitting of the data. In machine learning applications generalisation and avoiding overfitting are essential.

# 3. NEURAL NETWORKS

Neural networks are used in many fields of science for problem solving and their applications can be seen in translations of text, facial recognition, hand-written and speech recognition, controlling of robots, etc. Haykin (1999) has made the following definition of neural networks:

> A neural network is a massively parallel processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:
>
> 1. Knowledge is acquired from the environment through a learning process run in the network.
>
> 2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

It is important to note the connection of biological neural networks with artificial neural networks (ANN). ANNs have many similarities with the structure of the human brain. More specific a human brain contains neurons, which are connected with each other and their purpose is to process information. Figure 4 represents the neural network of two human neurons.

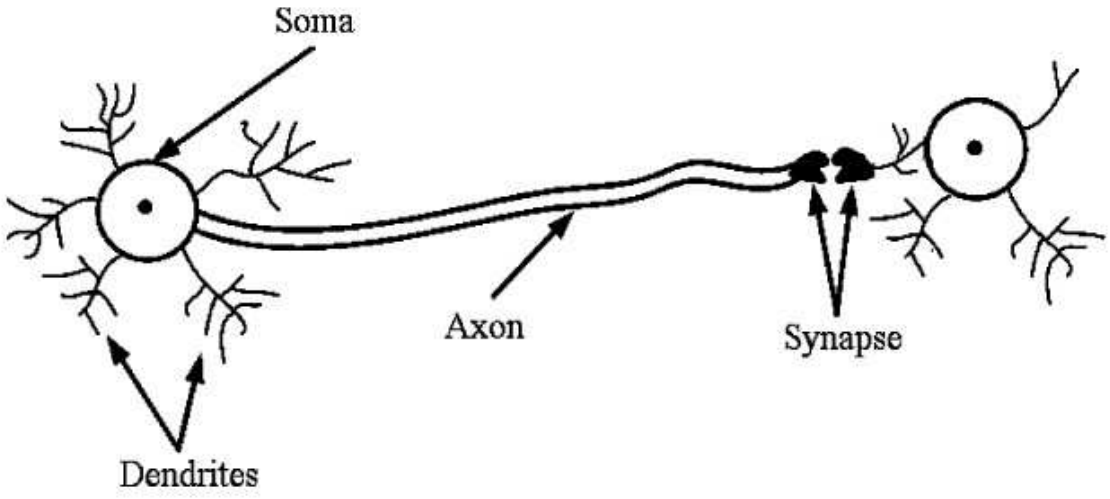


**Figure 4.** Representation of biological neural network (Deb & Dixit 2008).

Parts of a neuron are the *soma*, which is a cell body, the *dendrites*, which consist of several fibres and the *axon*, which is a single fibre. Dendrites are receivers of electrical signals that come from the axons of other neurons and the axon acts as a transmitter of electrical signals from one neuron to another through the dendrites. A *synapse* is used to connect an axon with a dendrite and it represents the place where an electrical signal is modulated by various amounts. Changes in the electrical potential in the soma can be achieved by the release of chemical substances of the synapses. An action potential is sent via the axon, which is nothing else than an electrical pulse created when the potential crosses a threshold (Deb & Dixit 2008).

The modelling of human neural networks can be achieved by artificial neural networks. In Figure 5, a diagram of an artificial neuron is represented.

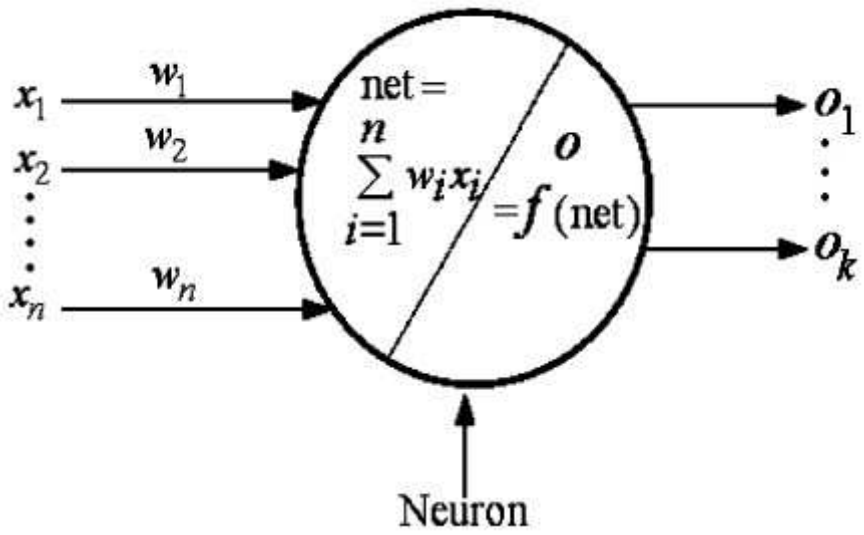


**Figure 5.** An artificial neuron (Deb & Dixit 2008).

It can be seen that the artificial neuron receives signals ($x_1$,$x_2$,.. $x_n$) from other neurons and produces signals ($o_1$, $o_2$, ...,$o_k$) that are going to be transmitted to other neurons. An artificial neural network uses numerical values for its signals rather than electricity used in human neural network. Each input signal is multiplied by a certain weight *w* and this process represents the action of the artificial synapses.

In the human brain an output signal is produced by a neuron when the input signal reaches a specific threshold. In terms of artificial neurons a summation of the inputs is calculated and an activation function, like the threshold mentioned in human neurons, is applied to the sum in order to generate the outputs of the neuron (Deb & Dixit 2008).

The simplest form of a neural network can be represented by the following summation:

$$\sum_{i=1}^{N} w_i x_i + B, \tag{16}$$

where $x_i$ is the input signal, $w_i$ is the weight and $B$ is a bias.

## 3.1. Feed-forward neural network (FNN)

Feed-forward neural networks are artificial neural networks where the connections between the nodes do not have the shape of a circle. They are the simplest form of artificial neural networks and the information travels in one direction from the input nodes (input layer) through the hidden layer to the output layer. Feed-forward neural networks can be divided into single layer and multilayer perceptrons. The single layer perceptron is the simplest kind of a neural network and consists of a single layer and output layer. In multilayer perceptrons there are multiple input and hidden layers that are interconnected in a feed-forward way. The type of neural network that is used in the proposed system is a multilayer perceptron.

## 3.2. Multilayer perceptrons (MLP) and Back-Propagation (BP)

The multilayer perceptron is a specific type of a layered feed-forward network, which consists of multiple input nodes in the input layer, multiple hidden layers (one or more hidden layers) and an output layer. The neurons in the hidden layers have the ability of

extracting important features included in the input signals. In each neuron a non-linear activation function is used. The neuron can achieve the efficient distinction of data that is not linearly separable (Cybenko 1989). The following figure depicts a fully connected feed-forward neural network with ten nodes at the input layer, one hidden layer and two nodes at the output layer.

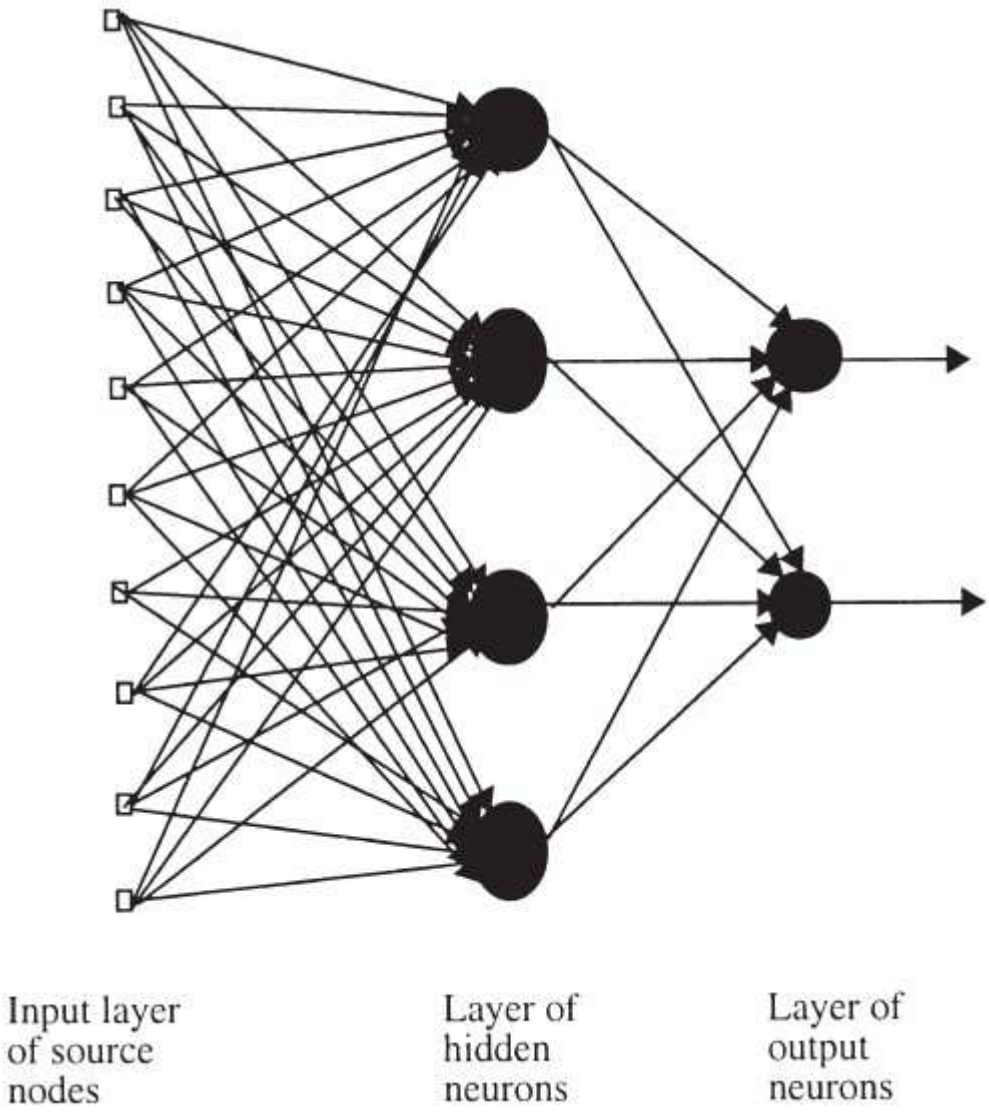


**Figure 6.** A fully connected feed forward neural network (Haykin 2004).

According to Werbos (1974) and Rumerlhart, Hinton & Williams (1986) the training of a multilayer perceptron can be achieved by a back-propagation algorithm that contains two phases:

1. *Forward Phase*: In this phase the free parameters of the network are fixed and the input signal is propagated through the network layer by layer. The phase is completed with computing the error signal:

$$e_i = d_i - y_i\,, \tag{17}$$

where $d_i$ corresponds to the desired response, $y$ to the actual output created from the network in response to the input $x$.

2. *Backward Phase*: In this phase, the error signal $e_i$, is propagated through the network following a backward direction. This phase ensures that the appropriate modification will be applied to the free parameters of the network in order to minimize the error $e_i$ (Haykin 2004).

## 3.3. Activation Functions

The activation functions are applied at the output of each node of an artificial neural network. The output of each node is used as an input to the next layer of nodes of the network. The main goal of the usage of an activation function in an ANN is to apply non-linear properties to the neural network. Non-linear properties are highly useful and beneficial in learning non-linear and complex mappings between the input and output. They can also ensure the decrease of the processing power of the system and time needed in order to find good approximations to the given problems. Similar to the function of the human brain, an activation function is used to define when a neuron should be *fired* / *activated* or not.

There are different types of activation functions used in neural networks. Some of them are presented next:

- *Sigmoid function*: It has a shape curve of the letter *S* and it is responsible to create real values between 0 and 1 that are used as the output of the nodes (Graupe 2013: 19). Sigmoid function is sometimes referred to as the logistic function and it is defined by the formula:

$$\varphi(x) = \frac{1}{1+e^{-x}} \,. \tag{18}$$

Figure 7 represents the shape of the sigmoid function.

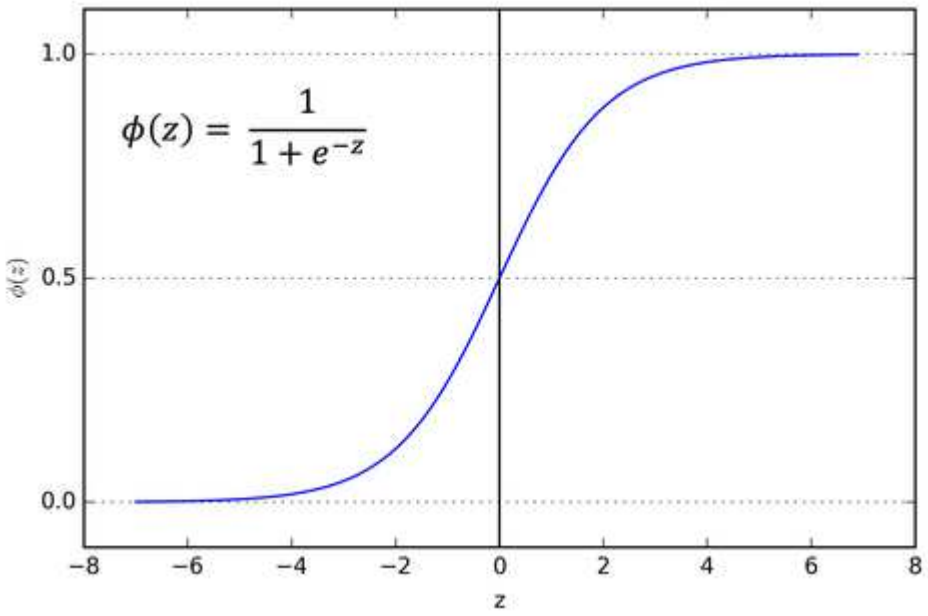


**Figure 7.** Sigmoid function.

- *Threshold function*: It refers to the function that takes the value 1 if the argument of the function exceeds a given threshold and otherwise the value 0. It is also known as the step function. It can be expressed by the following formula:

$$\varphi(x) = \begin{Bmatrix} 1, x >= 0 \\ 0, x < 0 \end{Bmatrix}. \tag{19}$$

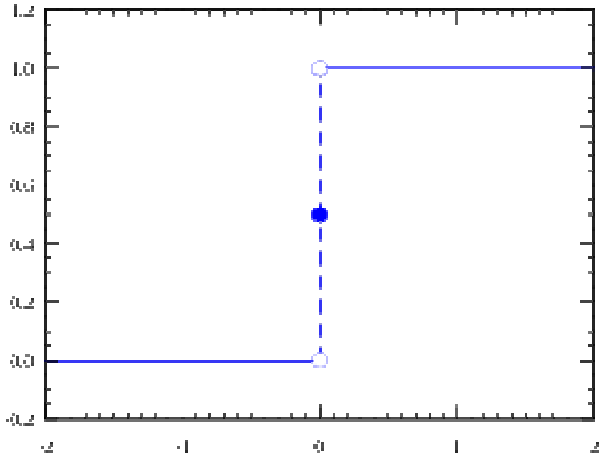

**Figure 8.** Threshold function.

- *Rectified Linear Unit (ReLU) function*: This function is used in ANNs to produce the value x as the output if x is positive and otherwise 0. It is highly recommended to use ReLU activation functions in deep neural networks because of their simplicity and efficiency. It can be expressed by the following formula:

$$\varphi(x) = \max(0, x). \tag{20}$$

Figure 9 shows the rectified linear unit function.

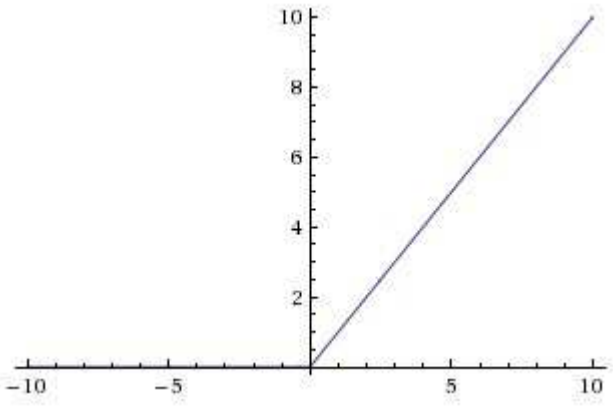


**Figure 9.** ReLU function.

- *Hyperbolic Tangent function*: The tanh function is similar to the sigmoid function. It is a non-linear function and its output values are in the range of -1 and 1. It has an s-shape curve and is smoother than the sigmoid curve (Graupe 2013: 20). It is not entirely flat and can ensure changes in its outputs depending the values of its inputs. It can be represented with the following expression:

$$\varphi(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} . \tag{21}$$

Figure 10 depicts the tanh function.

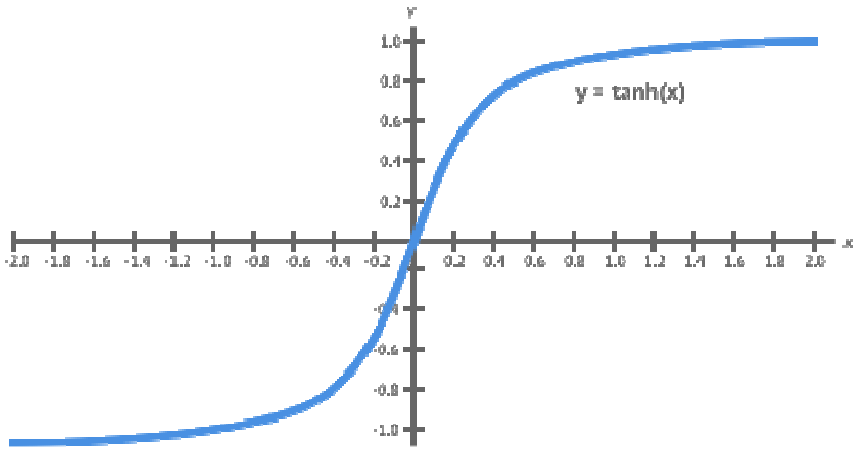


**Figure 10.** Hyperbolic Tangent function.

This section introduced some of the most important activation functions used in artificial neural networks. Their application depends on the given problem, for example the usage of sigmoid and tanh are useful and efficient, being non-linear functions, but their drawback is that they require a big amount of computation and time from the system. The ReLU function is also non-linear and it is often applied in many deep learning neural networks because of its simple form ensuring that the computations will not be a very demanding task for the system, but at the same time being able to produce useful results in the problem solving process.

It is worth noting that in this thesis the approach of achieving a good accuracy for the accent classification problem was to train two models; one model with only the Rectified linear unit activation functions and another model with combination of ReLU and sigmoid activation function at the third layer of the convolutional neural network.

## 3.4. Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) are a type of deep feed-forward artificial networks, which are used in deep learning applications such as image and video recognition, image classification, recommender systems, medical image analysis, natural language processing and speech recognition. Deep learning is a part of machine learning algorithms based on learning data representation and it is used mainly in supervised and unsupervised learning applications.

In this thesis the application of a convolutional neural network compared to a deep neural network architecture achieved better results and accuracy of the model for the accent classification problem. One reason for the difference in the results from the proposed approach using convolutional neural network may be the fact that the input audio wave files are represented by their Mel-Frequency Cepstral Coefficients and are processed like features of a two-dimensional images. Convolutional neural networks being highly efficient in the field of image recognition, can also be beneficial and useful in achieving

high accuracy and performance in the domain of accent classification compared to modern deep neural network architectures.

A CNN has input layer, hidden layers and an output layer. The hidden layer of a convolutional neural network consists of basic building blocks – layers such as the convolution layers, the pooling layers and the fully connected layers. These layers and their functions will be covered in the next sections of this chapter.

### 3.4.1. Convolutional Layer

The convolutional layer can be considered one of the most important layers of a convolutional neural network. In the convolutional layer a set of filters is applied and is convolved with the input in order to create an output. In this way the mapping between the input of the system and the output is achieved.

A filter in this layer can be seen as a matrix of numbers, which corresponds to the weights of the network. The weights are being set randomly in the beginning of the training of the CNN and after some time during training the filter weights are fine-tuned. A *2x2* filter is presented in Figure 11.

| | |
|---|---|
| 2 | 0 |
| -1 | 3 |

**Figure 11.** A 2x2 filter.

The most significant function in a convolutional layer is the convolution operation. The convolutional layer implements a convolution between the input and the filters. The function of the convolution operation can be seen with the help of Figure 12 where a 2D

convolution is shown. Specifically, a 2D input feature map size of *4x4* and a convolution filter of matrix size *2x2* are considered. The aim of the convolutional layer is to multiply the filter (matrix size *2x2*) with a section of size *2x2* of the input feature map, which is highlighted. The next step is to make a summation of all the values to create one value in the output. It is important to note that the filter has to slide through all width and height of the input (Khan, Rahmani, Shah & Mohammed 2018: 46).

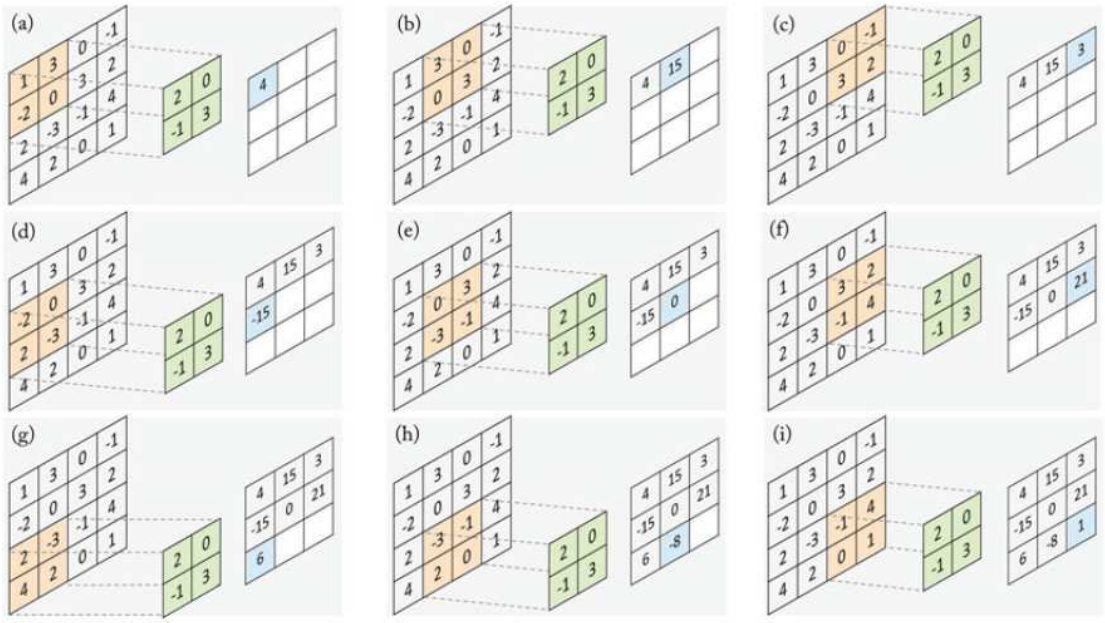


**Figure 12.** Stages of a 2D convolution operation (Khan et al. 2018: 47).

The operation described above is called *cross correlation* and in the convolution the filter is flipped before multiplication and sum-pooling. This distinction is important in the signal processing domain, but in machine learning applications both terms are used interchangeably. In the convolutional layer the correlation operation is applied in the majority of the deep learning libraries and algorithms. The main reason of following this convention is that the convergence of the network optimization will be achieved on certain weights of the filters either the operation of the correlation or convolution is used.

In the example used to describe the operation in the convolutional layer, the filter occupies a step of 1 through the horizontal and vertical axis in order to compute the value of

the output. The number of the step used is called the *stride* of the convolutional filter. It can have different values and a rule of thumb is that when the stride is increasing the dimension of the output feature map is decreasing. Given a filter of size *f x f*, an input of size *h x w* and a stride of *s*, the dimensions of the output are computed by the following expressions (Khan et al. 2018: 49):

$$h' = \frac{h - f + s}{s}, \tag{22}$$

$$w' = \frac{w - f + s}{s}. \tag{23}$$

Sometimes in order to achieve deeper networks and acquire better accuracy and performance zero-padding around the input of the network can be applied. The application of zero-padding can be effective in increasing the dimension of the output feature map and accomplishing flexibility in the process of designing the architecture of the CNN. The aim in this situation is to increase the size of the input in order to achieve an output with specific dimensions. Therefore the output feature map dimensions can be represented by:

$$h' = \frac{h - f + s + p}{s}, \tag{24}$$

$$w' = \frac{w - f + s + p}{s}, \tag{25}$$

(Khan et al. 2018: 49)

where the parameter *p* indicates the increase in the input in each dimension.

### 3.4.2. 2D Convolutional Layer

The difference between a 2D convolution layer and a 1D convolution layer is that in the case of a 2D the filter weights are handled in two dimensions. A 2D convolutional layer may contain a 2D signal such as image which in this thesis the audio signal is treated as an image of size *13x30*. The following figure depicts a 2D convolutional layer. In the figure a neuron begins from a corner of the signal, strides in one direction and ends at the opposite region. It may follow the way from the top left corner and end at the bottom-right. The next step is to apply a convolution of each neuron with every channel and feature map of the input. Concerning the outputs, they can be included in a location-wise addition in order for neuron to contain one averaged output response (Venkatesan & Li 2018: 94).

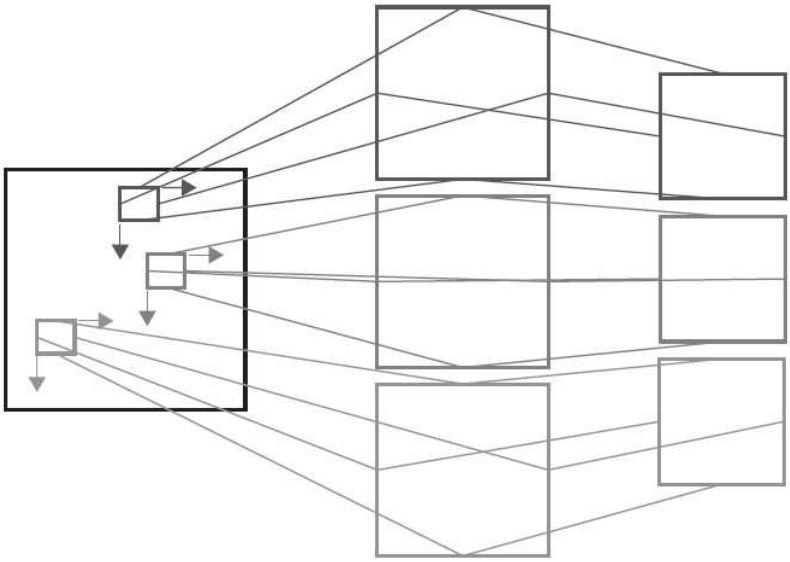

**Figure 13.** Convpool layer including three neurons (Venkatesan & Li 2018: 94).

Given the input of the layer $\alpha$ has $I$ channels and the layer has $L$ kernels with $k$ and $l$ corresponding to the element-wise activation function, the output activations of the layer can be represented by the following expression (Venkatesan & Li 2018: 95):

$$z^j = \lambda\left(\sum_{i=1}^{I} \alpha^{(i)} * k_j\right) \forall j = [1,2,...,L]. \quad (26)$$

The symbol * corresponds to the operation of convolution. The filter used in a CNN is learned by the convolutional layers from the input and the filter often detects edges or blobs. In CNNs the input in the current project represented by images are separated into small components of images that contain information in smaller parts that are mapped to the label space with the help of the neural network (Venkatesan & Li 2018: 95).

### 3.4.3. Receptive field

The inputs of convolutional neural networks in many applications are characterised by high dimensionality. In image processing and consequently in the accent classification task used in the proposed system, it is important to apply convolutional filters that have smaller size compared to the size of the input. In the current system the size of the convolutional filters are *3x3*, which are smaller than the input size of *13x30*.

Through the above approach the number of parameters to be learned from the model is decreased when the size of the kernels applied is small. In addition, the usage of small size filters can improve the learning of the system from specific patterns from the input. The term *receptive field* is referred to the size of the filter, which corresponds to a specific region that is modified at each convolution step. The receptive field is related to the dimensions of the input and in the cases of convolutional layers stacked on top of each other, the effective receptive field of each layer acts as a function of the receptive fields of all the previous convolutional layers (Khan et al. 2018: 50). The effective receptive field of a stack of $N$ convolutional layers, with a kernel size of $f$ each can be expressed by the following formula (Khan et al. 2018: 50):

$$RF_{eff}^{n} = f + n(f-1), n \in [1, N]. \quad (27)$$

### 3.4.4. Pooling Layer

Pooling layers refer to the combination of a region of the output layer into a single value of the next layer of the network (Ciresan, Meier, Masci, Gambardella & Jurgen 2011). A pooling layer can be represented as a combination operation on block of the input of the network and the feature activations. Usually this operation is defined by the average or the max function. For example if the max pooling operation is selected, then the maximum activation is also selected for the certain block of values in the layer. The operation will be applied to the input feature map by sliding this window to the input with a step size. The step size is generated by the stride. It is crucial to define the size of the pooled region and the stride that is going to be applied in the network. The output of the feature map, considering the size of the pooled region *f x f* and a stride *s*, is calculated by the following formulas:

$$h' = \frac{h - f + s}{s}, \tag{28}$$

$$w' = \frac{w - f + s}{s}. \tag{29}$$

(Khan et al. 2018: 53)

Figure 14, on the next page, depicts the operation of a max pooling layer with the pooling region being of size *2x2* and the stride being 1. The steps from *(a)* to *(i)* represent the computations at each step while the pooled region in the input (the orange region) is slid in order to calculate the value in the output (the blue value) at each step.

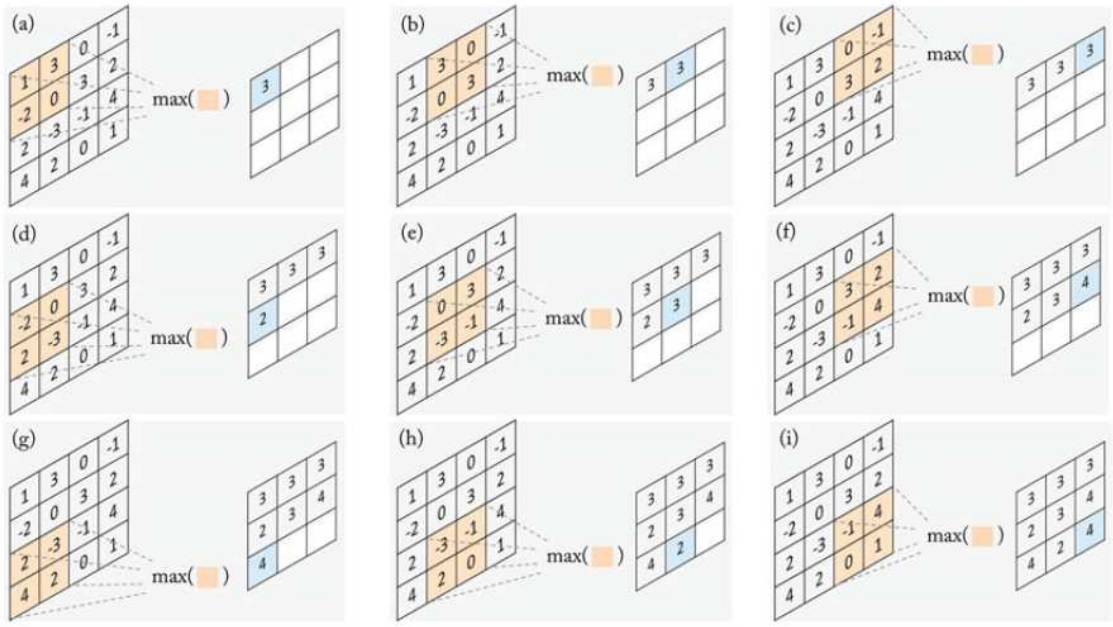


**Figure 14.** Max pooling operation with a 2x2 pool region and stride 1 (Khan et al. 2018: 53).

The main operation of the pooling layer is to down-sample the input feature map.

According to Venkatesan & Li (2018: 96) the operation of pooling reduces the data entropy by reducing the size of the activations. The reduction of size is a useful feature in a CNN because although there is a reduction in spatial information and frequency in an activation, there is a gain in activation responses. Pooling can be seen as an operation of achieving invariances among features spatially. As the neural network becomes deeper, the number of the activations increases and therefore the computations become demanding. The operation of pooling is useful in keeping the tractability in the network.

The preferred type of pooling in the domain of convolutional neural networks is that of max pool. In the case of implementing a max pool by *p*, then a sliding window and a stride of *pxp* is considered a suitable option. In each window, the maximum value is selected, which represents the entire window with this value. The max pool operation is often used because it contains the strongest response, which is an important term in the context of image processing (Venkatesan & Li 2018: 97).

### 3.4.5. Fully Connected Layers

In fully connected layers each neuron in the previous layer is connected to every neuron in the current layer. These layers can be expressed by convolution layers having filters of the size *1x1*. Most convolutional neural networks with this type of layers are placed at the end of the architecture. The operation of a fully connected layer can be expressed by a matrix multiplication and an addition of a vector of bias, implementing an element-wise nonlinear function *f* as the following:

$$y = f\left(W^T x + b\right), \tag{30}$$

where $x$ is the vector of the input and $y$ the vector of output activations, $W$ is the matrix of weights of the connections in the layer units, and $b$ is the bias vector (Khan et al. 2018: 56).

Having covered the basic elements of a CNN in the previous section it is important to include a figure of a complete CNN. Figure 15 depicts a complete convolutional neural network architecture:

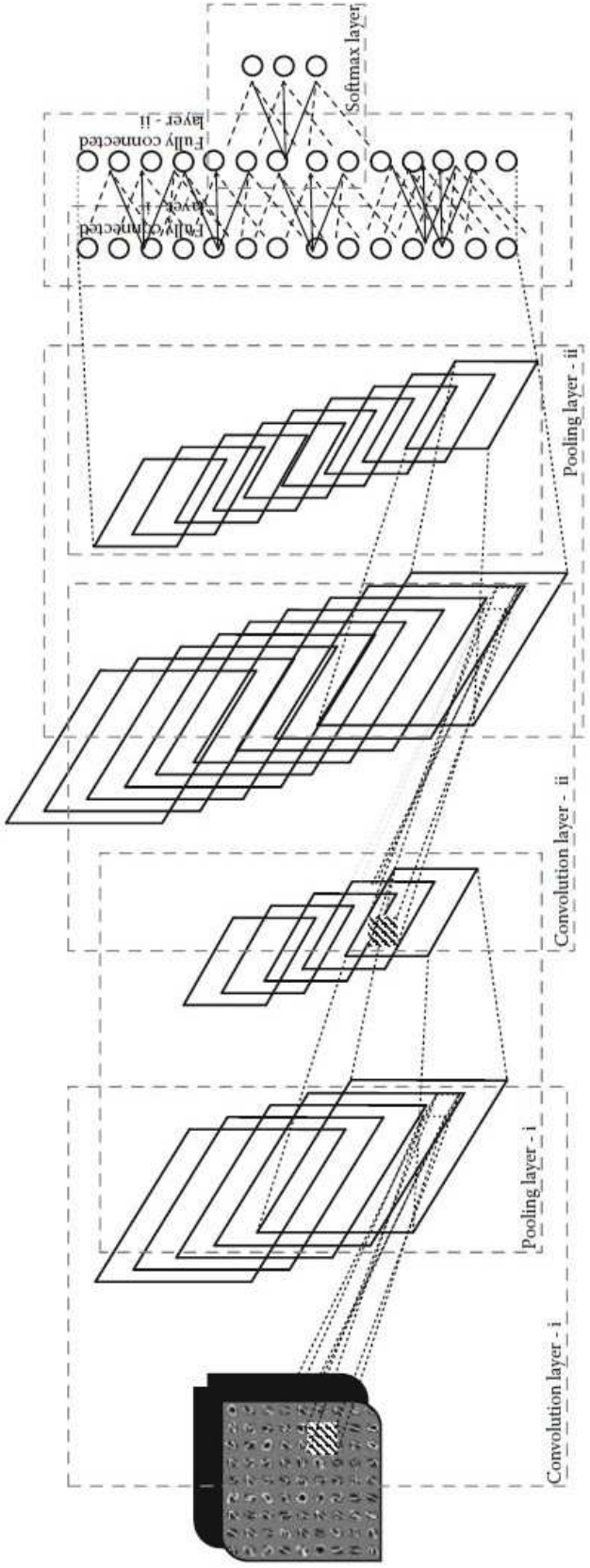


**Figure 15.** Complete convolutional network architecture (Venkatesan & Li 2018: 98).

### 3.4.6. Dropout

According to Srivastava, Hinton, Krizhevsky, Sutskever & Salakhutdivon (2014) one of the favoured approaches for neural network regularization is the technique of dropout. During the training process of the network each neuron is activated and contains a probability. The above process is characterized by randomness in the sampling of the network, which produces an ensemble effect in the testing process of the network. The dropout activation is an essential term for regularization and it is responsible for improvements in the performance on unseen data in the testing process.

Given a CNN consisting of $L$ weight layers with index $l \in \{1...L\}$ and output activations $a_{l-1}$ from the previous layer, the $a$ fully connected layer implements a transformation followed by a element-wise nonlinearity which can be represented by the following:

$$a_l = f\left(W * a_{l-1} + b_l\right), \tag{31}$$

where $a_{l-1} \in \mathfrak{R}^n$ indicates the activations and $b \in \mathfrak{R}^m$ corresponds to biases. The input and output dimension of the fully connected layer are represented by $n$ and $m$, the weight matrix is represented by $W \in \mathfrak{R}^{mxn}$ and the function $f(.)$ by the ReLU activation function. (Khan et al. 2018: 75).

### 3.4.7. Loss Functions in CNNs

The last layer in a common CNN architecture is used in the training process of the model. In order to approximate the quality of the prediction generated by the neural network on the training data, the last layer of the network utilities a loss function. The role of the loss function is to calculate the difference between the approximated output of the model and the real output.

There are different loss functions that are used in neural networks depending on the characteristic of the problem to be solved. The generic set of problems and the loss functions used can be divided into the next categories:

1. Binary Classification (SVM hinge loss, Squared hinge loss)

2. Identity Verification (Constrastive loss)

3. Multi-class Classification (Softmax loss, Expectation loss)

4. Regression (SSIM, $l^t$ error, Euclidean loss)

### 3.4.8. Soft-max Loss / Cross-Entropy Loss

The topic of the current thesis is associated with the multi-class classification therefore in the next section the Softmax loss or the cross-entropy loss will be presented.

The cross-entropy can be represented by the following formula:

$$L(p, y) = -\sum_{n} y_n \log(p_n), n \in [1, N], \tag{32}$$

where $y$ is the desired output and $p$ indicates the probability for each output category. $N$ represents the total number of neurons in the output layer, and $p,y \in \Re^n$ . By using the soft-max function $p_n = \frac{\exp(\widehat{p}_n)}{\sum_k \exp(\widehat{p}_k)}$, the probability of each class can be computed. $\widehat{p}_n$ indicates the unnormalised output from the previous layer in the network (Khan et al. 2018: 66).

# 4. SYSTEM ARCHITECTURE AND IMPLEMENTATION

This section includes the details of the system architecture and the implementation of the accent classification project using convolutional neural networks. The system has been developed on a laptop equipped with a dual Intel Core i5 2nd generation processor at 2.30 GHz each, Windows 7 64-bit operating system and 8 GB of RAM.

The programming language that was selected for the implementation of the project was Python 3.6 with the Integrated Development Environment of PyCharm (PyCharm). The pre-processing of the audio files was implemented with the open source audio processing software of Audacity 2.1.1 (Audacity) and Winamp 5.666 (Winamp). In addition, this chapter covers all the information about the dataset that was used, the architecture of the neural network, the feature extraction method, the design of the system and the explanation of the programs of the project.

## 4.1. System architecture

The system architecture consists of the processes of collecting the voice signals, pre-processing, feature extraction, classification and output in order to identify the accent of the voice signal.

The process of the acquisition of the audio files was done by a Java program created by the author. The purpose of this program was to download the audio files of selected accents from the website of "*The speech accent arch*ive". The audio files were collected by the George Mason University Department of English Speech Accent Archive (Weinberger 2015). The website contains 2775 samples of different speakers reading a certain text in English. The speakers are from around the world and the website contains accents from more than 200 languages.

The process of pre-processing was necessary because it could affect the results of the experiments. In every machine learning project the need of using clean data is crucial in order to acquire a good result and output from the system. The pre-processing procedure is divided into two stages. The first stage is to convert each mp3 audio file downloaded from the website to a WAV file of 16kHz, mono channel and 16-bit. The second stage is to normalise the maximum amplitude of each WAV file to -1.0 dB. The aim of the second stage is to ensure as much possible a universal volume level of each audio file. Figure 16 depicts the diagram of the proposed system:

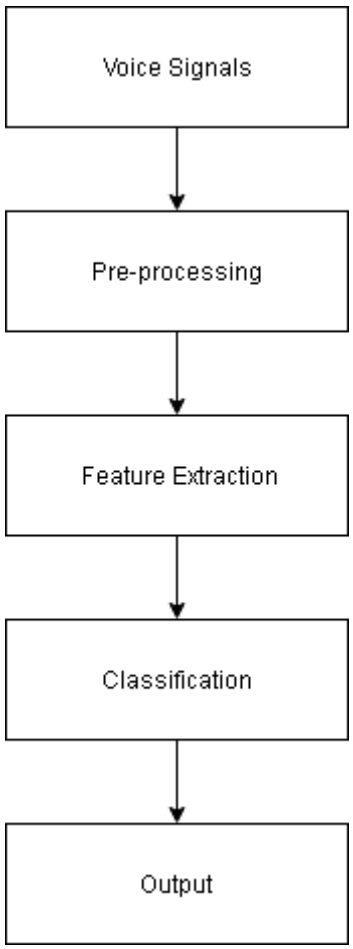


**Figure 16.** Processes of the proposed system.

The feature extraction process as it was presented in Chapter 2 is essential for the accent classification project because it can differentiate and characterise each input signal into

the convolutional neural network. It is presented in more detail also in this chapter in section 4.4.

The processes of classification and generation of output is connected with the architecture and procedures of the convolutional neural network. In this thesis two different convolutional neural networks have been examined and their structure and accuracies will be discussed in section 4.5 of this chapter.

The chapter continues with the coverage of the above elements and terms of the system in more detail.

## 4.2. Dataset

The dataset of the proposed system was downloaded from the website of "*The speech accent archive*" (Weinberger 2015). It consists of speakers who read an elicitation paragraph written in English, which contains common English words and difficult English sounds and sound sequences. The elicitation paragraph consists of English words that cover almost all the sounds of English language and contains as many as possible consonants, vowels and clusters of the standard American English. The paragraph spoken in the audio files of the system is the following:

*"Please call Stella. Ask her to bring these things with her from the store: Six spoons of fresh snow peas, five thick slabs of blue cheese, and maybe a snack for her brother Bob. We also need a small plastic snake and a big toy frog for the kids. She can scoop these things into three red bags, and we will go meet her Wednesday at the train station."*

Therefore, the approach of accent classification used in this thesis is text dependent. The number of speakers used in the proposed system is 1004 and the speakers are native speakers of Chinese, Spanish, English and Arabic. The table on the next page shows the number of audio files used in the system for each native language and the number of audio files used for prediction.

**Table 1.** Languages and number of audio files used in the system.

| Language | # of audio files | # of files for prediction |
|---|---|---|
| Chinese | 137 | 35 |
| Spanish | 166 | 42 |
| English | 376 | 94 |
| Arabic | 125 | 30 |

It is worth to note that each speaker was allowed to read the paragraph for some time and after the recoding was applied. The quality of the recordings is good and according to the website of "*The speech accent archive*" (Weinberger 2015) the recording equipment included a Sony TC-D5M with a Radio Shack 33-3001 unidirectional dynamic microphone and a Sony minidisk recorder MDR-70 with a Sony ECM-MS907 stereo microphone.

In addition, the speakers were asked to answer seven demographic questions which include the place of their birth, their native language, what other languages besides English do they know, their age, when they first began to study English, the way they learned English (academically or naturalistically) and the period they have lived in an English speaking country if applicable. All this information is included in a CSV format file. The next text represents a part of the CSV file *data_info2L.csv* used for the accent classification of Chinese and Spanish speakers:

```
href,language_num,sex,birth_place,native_language,other_languages,
age_sex,age_of_english_onset,english_learning_method,english_resid
ence,length_of_english_residence,age

http://accent.gmu.edu/browse_language.php?function=detail&speakeri
d=45,chinese1,female,"['kong,',    'china']",chinese,"['mandarin',
'']","['22,', 'female', '']",12.0,academic,['usa'],1.0,  22
```

```
http://accent.gmu.edu/browse_language.php?function=detail&speakeri
d=46,chinese2,male,"['kong,',      'china']",chinese,"['mandarin',
'']","['20,', 'male', '']",13.0,academic,['usa'],0.3,  20

http://accent.gmu.edu/browse_language.php?function=detail&speakeri
d=47,chinese3,male,"['kong,',  'china']",chinese,"['french',  'man-
darin', '']","['22,', 'male', '']",6.0,academic,['uk'],0.1,  22

.............................................................

.............................................................

http://accent.gmu.edu/browse_language.php?function=detail&speakeri
d=2230,spanish164,female,"['zacatecas,',                    'mex-
ico']",spanish,['none'],"['35,',                          'female',
'']",27.0,naturalistic,['usa'],19.0,  35

http://accent.gmu.edu/browse_language.php?function=detail&speakeri
d=2235,spanish165,male,"['puerto',    'rico']",spanish,"['german',
'italian',  '']","['62,',  'male',  '']",4.0,academic,['usa'],11.0,
62

http://accent.gmu.edu/browse_language.php?function=detail&speakeri
d=2238,spanish166,female,"['dominican',                   'repub-
lic']",spanish,['none'],"['26,',                          'female',
'']",11.0,academic,['usa'],5.0,  26
```

## 4.3. Pre-processing

The pre-processing procedure was a significant process although the dataset from the website was of a good quality. First of all, the downloaded dataset from "*The speech accent archive*" (Weinberger 2015) was in mp3 format. The proposed system uses audio file of the form of WAV files, sampled at 16kHz, mono and 16-bit. Winamp was used in order to convert the dataset in mp3 format to the desired format. The process is easy as one can load all the audio files in the playlist of the program and set its audio output to *Nullsoft Disk Writer* and its conversion to *PCM 16.000 kHz; 16 Bit: Mono*.

The second step of the pre-processing was to normalise the maximum amplitude of each WAV files of the speakers. Most of the audio files were quite good concerning their normalisation but some of them had low amplitude, which may result in inefficient us-

age of inputs for the neural network. The program used for the normalisation of the audio files was Audacity 2.1.1. and the normalisation was also not difficult to implement as one can load all the audio files and apply normalisation of the maximum amplitude to -1.0 dB using the batch programming mode of the software.

The importance of using clean data in the neural network is high. If clean input data is used, then it is most likely that a clean output can be achieved, in this case meaning a good approximation and accuracy of the model of the convolutional neural network. Usually a satisfactory sampling rate for human voice is at 16kHz and mono. But the fact of existing unnormalised audio signals is dangerous and it is an alert for the engineer to apply normalisation for the maximum amplitude of the signal. The value of -1.0 dB is the default in Audacity and often it is so in order to leave some headroom for a possible final editing of the audio signal.

## 4.4. Feature Extraction

The process of feature extraction in the proposed system consists of applying the Mel-Frequency Cepstral Coefficients as it was presented in Chapter 2, section 2.5.1. The main reason for this approach is to achieve a reduction in the dimensionality of the input. Besides, the representation of the sound signals is more useful by applying the MFCCs. By using MFCCs the audio samples are divided into small windows and chunks of time (Watanaprakornkul, Eksombatchai & Chien 2010). Furthermore, the Mel cepstrum provides a way to categorise the frequencies of the audio files in terms of effective phoneme distinction (Bryant, Chow & Li 2014).

In this thesis the first 13 coefficients were used for the feature extraction. Each audio WAV file of the speakers is divided into segments and the MFFC of the current segment is taken. The dimension of each segment is *13x30* and it is feed into the network in a way as a two-dimensional image could be represented.

There are additional features that can be used for the process of feature extraction such as Fbank and Delta (Chu et al. 2017), but the dimension of the input increases extremely. Under the current situations, using an ordinary computer and not a GPU, the strategy of following just the MFCCs suggested to be appropriate in terms of dimensionality.

## 4.5. Convolutional Neural Networks Architecture

The accent classification problem presented in this thesis was solved by the implementation of two convolutional neural networks. The difference between them is found in the activation functions of their layers.

Specifically, the first CNN consists of a 2D convolution layer with an output filter of dimension 32 in the convolution, an input shape of *13x30x1* and a ReLU activation function. The data format is set to *channels_last* which means that the ordering of the dimensions in the inputs has the form of (batch, height, width, channels). This includes the number of segmented audio files in batch, the shape of audio files after applying the MFCC with 13 coefficients and the number of channels, which is 1 in the current case. In the domain of image processing this could be seen as an image of dimension *13x30*. Next a max pooling operation with a pool size of *2x2* is applied in order to down scale the spatial dimension. The pool size refers to the vertical and horizontal factors of the process of downscaling.

Next a second 2D convolutional layer is applied with the output filter of dimension 64 and the same input shape and activation function as the first convolutional layer. The following step is to apply the same max pooling operation and a dropout operation with a rate of 0.25 in order to avoid overfitting of the network. Next an operation of flattening takes place and the input is flattened into 1 dimension without affecting the batch size. Next a regularly densely-connected layer is added with 128 units and the activation function of ReLU. Another dropout operation is applied next with a 0.5 factor in order

to avoid the cases of overfitting. The last layer is a fully connected layer with the number of accent classes used for the model and a softmax activation function.

The second CNN is almost the same as the first with the difference that on the third layer (fully connected) with 128 units instead of ReLU the sigmoid activation function was used. In addition, several other configurations have been tested. These configurations did not result in a better accuracy. However, the two proposed models gave good approximations.

Figure 17 in the next page represents the architecture of the two convolutional neural networks that were used in this thesis.

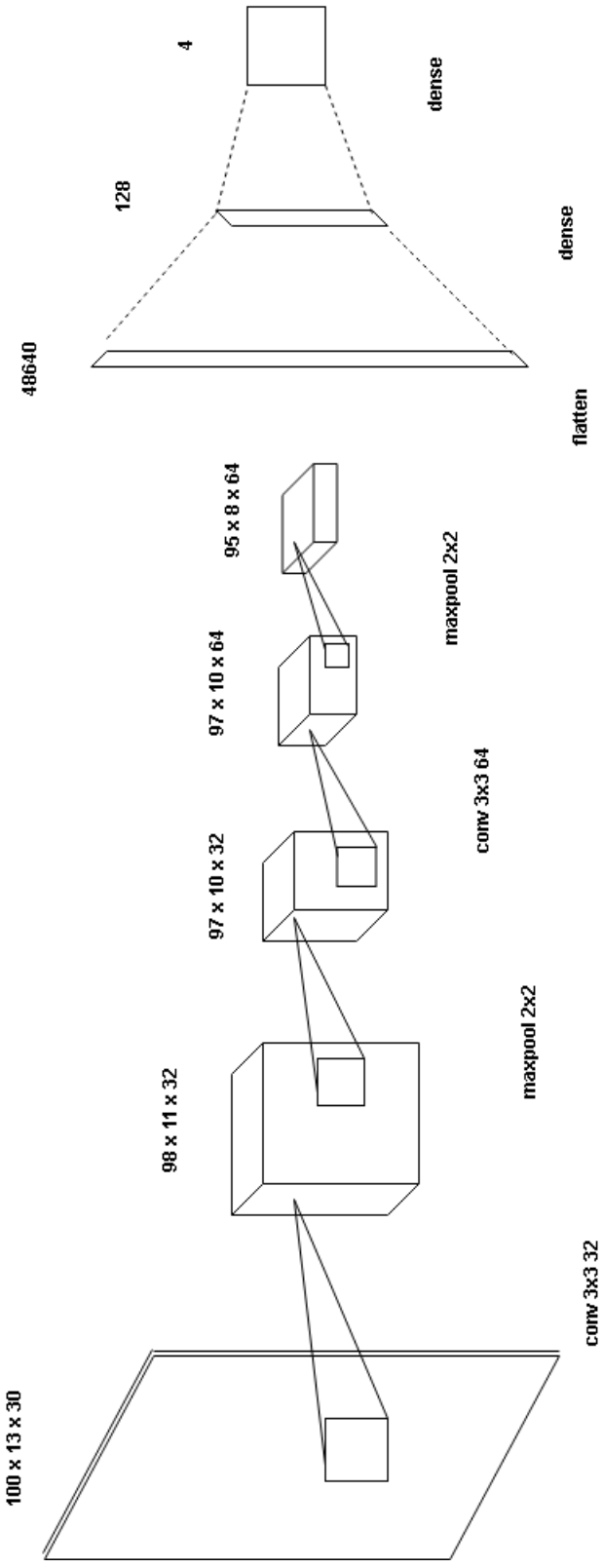


**Figure 17.** CNN architecture of the proposed system.

## 4.6. Program Implementation

The program used in the accent classification problem is based on an implementation by Yatharth Garg (2018) and it was modified and parameter tuned by the author for the requirements of the thesis. The program was created using the programming language of Python 3.6 with the help of the free version of PyCharm. The important libraries included in the program consist of Numpy, Pandas, Keras and Librosa. The implementation of the CNNs were done using Keras, which is a neural network library in Python, running on top of TensorFlow. The user friendliness, modularity, extensibility and most important being open source make it ideal for working and experimenting with deep neural networks and CNN projects.

The main python file used in the project is *trainmodel.py*. The subprograms used are *getsplit.py*, *accuracy.py* and *predict.py*. The source code of the project is included in Appendix 1.

In *trainmodel.py* the user must set the number of the MFCCs, the number of epochs for the training of each model, the CSV file that contains the data of the speakers and the name of the model that will be saved after the training of the network. The following table shows the CSV files used in the project.

**Table 2.** CSV files with the corresponding supported languages.

| CSV file | Languages |
|---|---|
| data_info2L | Chinese, Spanish |
| data_info3L | Chinese, Spanish, Arabic |
| data_info4L | Chinese, Spanish, English, Arabic |

Chinese language includes Cantonese, Mandarin, Taiwanese, Wu, and Xiang varieties of Chinese spoken language.

The training and test samples are split with the help of the *getsplit.py* program. In *getsplit.py* the user can specify the percentage of test sample size. The percentage of training sample is calculated automatically. The next step is to convert the outputs of train and test sample into a binary representation of the total number of classes using the function *to_categorical*. This conversion is essential because the loss function of the convolutional neural network is set to *categorical_crossentropy*.

Then the audio files are loaded and the MFCC for each one is calculated. The segmentation of the audio files from the MFCCs is taking place next, and the process of randomizing the training segments follows. In the next step the function of the training of the model is called with the arguments of training input and output samples and validation input and output samples. This function is called *train_model* and its purpose is to define and train the 2D CNN. The function contains an early stop of the training in case of the accuracy does not change at least 0.005 over 10 epochs.

An object of *ImageDataGenerator* is created next; it is used to fit the model on batches with real-time data augmentation and train the CNN. The following procedure is to calculate the accuracy of the predictions by calling the *predict_class_all* function and pass-

ing the arguments of the test segmented samples and the model. At the end of the *train-model.py* program some statistics are printed to the output for the user to see the number of training samples, testing samples, the confusion matrix and the accuracy of the current model. The model is saved on the disk and the final step is to output the total time needed for the program to execute and a sound is played to indicate that the program has been executed.

Finally, the prediction and classification of an audio signal can be achieved by running the program *predict.py*. The user must place the audio file for classification in the folder "*Prediction_File*" and run the above python program. The file name of the CSV file must be set by the user and then the MFCC of the input audio signal is taken. Next the output of the signal variable takes the return value from the calling of the function *predict_class_all* from the program *accuracy.py*. The last step is to check the value of the output of the signal and print out the predicted accent.

Figure 18 shows the diagram of the program *trainmodel.py*.

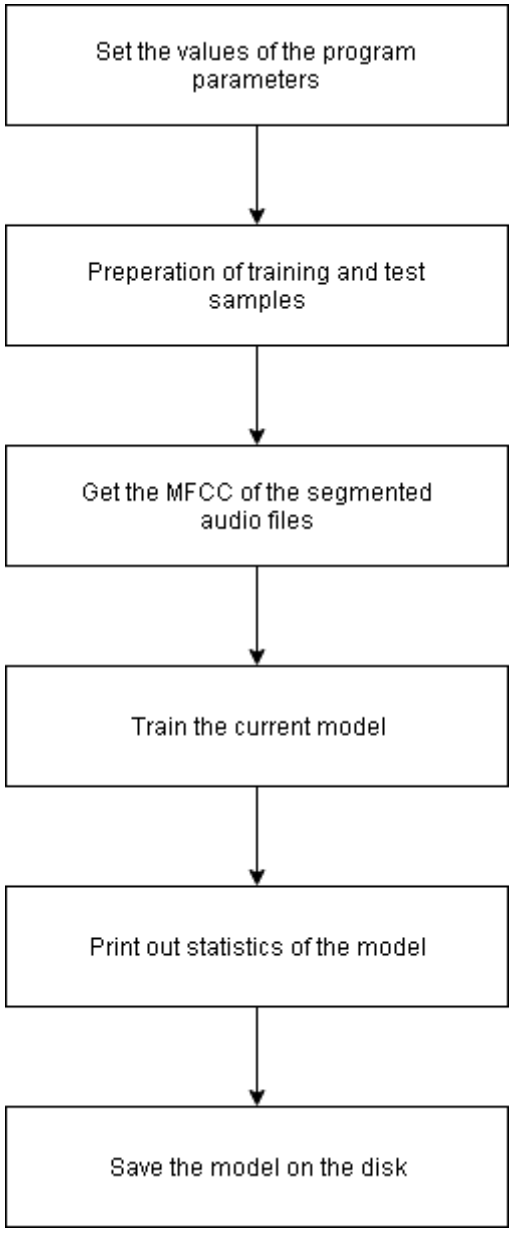


**Figure 18.** Diagram of the main program *trainmodel.py*.

# 5. EXPERIMENTS AND RESULTS

In this chapter the experiments of the two different convolutional neural networks that have been discussed in Chapter 4 are presented as well as the results of the experiments. In addition, the achieved accuracies in the form of percentage of each model and the number of epochs are discussed and shown in comparative figures.

## 5.1. Setup of the experiments

The experiments presented in this chapter refer to two convolutional neural network architectures that have been presented in the section 4.5. The difference between the proposed networks is that the first CNN uses ReLU activation functions, while the second CNN uses a sigmoid activation function on the 3rd convolutional layer.

The dataset used in the experiments includes the four languages of Chinese, Spanish, English and Arabic. The detailed structure of the dataset can be seen in section 4.2 in the previous chapter. In the case with two languages Chinese and Spanish were used, in the case with three languages Chinese, Spanish and Arabic were used and in the case with four languages Chinese, Spanish, English and Arabic were used. The experiments cover all the accuracies and the performance of the models with training and test samples of percentage from 90-10, 80-20 to 10-90 respectively.

The computer used for the training the models, as described in Chapter 4, was a laptop with a dual Intel Core i5 at 2.30Ghz each, running Windows 7 64-bit operating system and having 8 GB of RAM. In addition, the programming language of Python 3.6 was selected and the implementation of CNN was done with Keras.

The next sections of this chapter present the performance, the confusion matrix of each model with the different amount of percentage between training and testing samples, and the comparison between them.

## 5.2. Experiments with the first CNN architecture (ReLU activation functions)

In the case of 90-10 (percentages of training-test samples) the highest accuracy of the model achieved 96.72% after 83 epochs for the 2 languages. In terms of 3 languages the model scored 84.88% accuracy after 88 epochs and with 4 languages the best accuracy is at 74.27% after 35 epochs. Figures 19–23 show the accuracies for the case of 90% training samples and 10% test samples, the value of the loss function on validation samples and the confusion matrices accordingly.

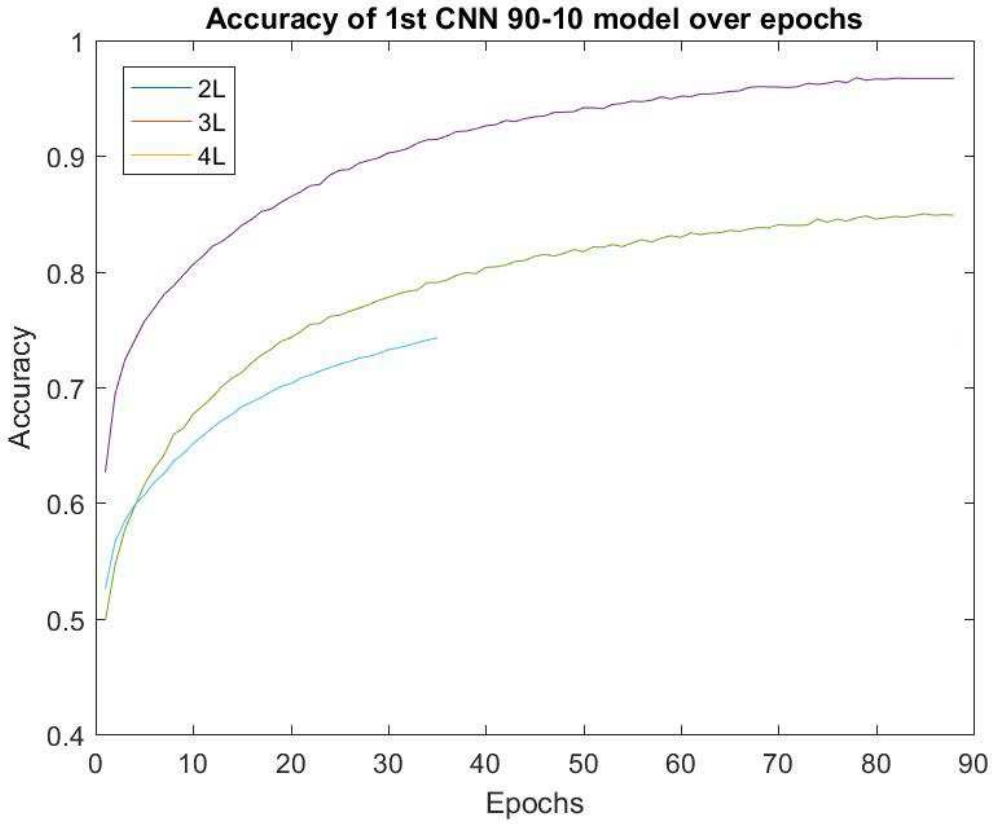


**Figure 19.** Accuracy of 1st CNN 90-10 model over epochs.

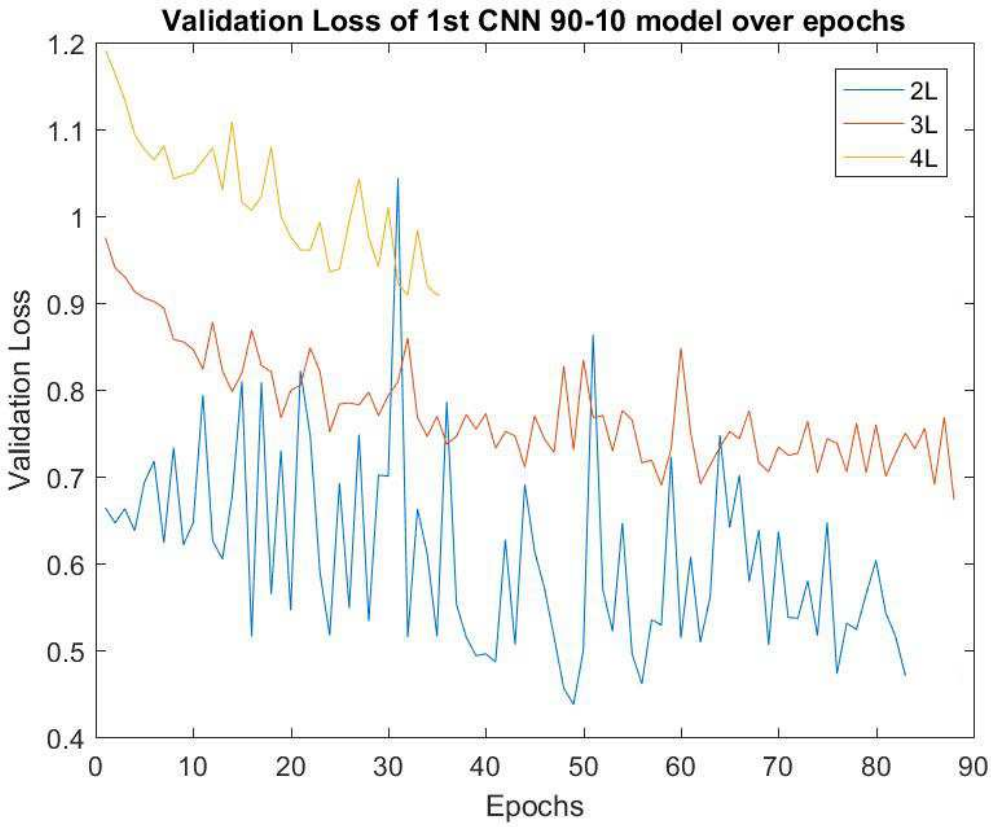


**Figure 20.** Validation loss of 1st CNN 90-10 model over epochs.

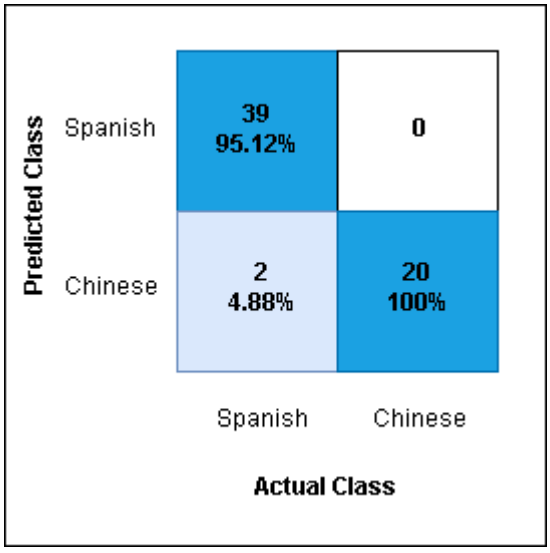


**Figure 21.** Confusion matrix of 1st CNN 90-10 model for 2 languages.

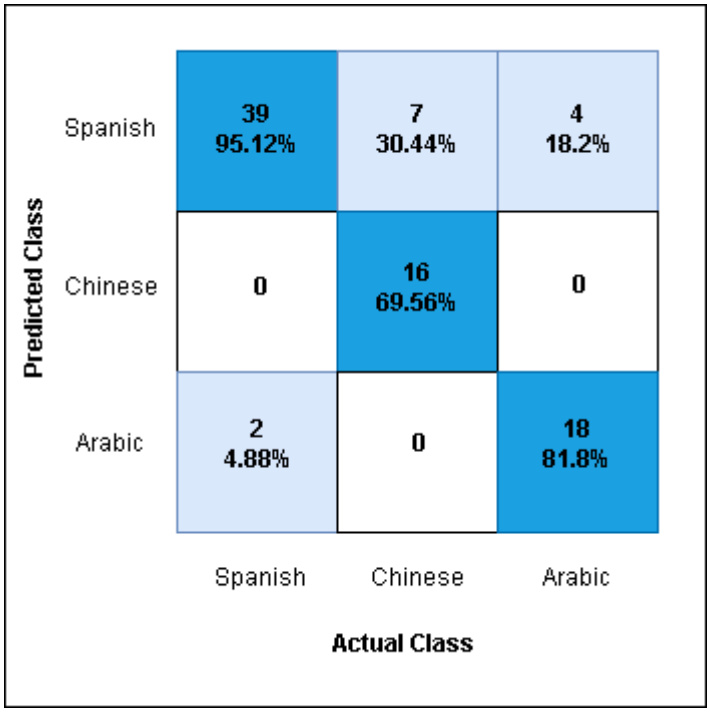


**Figure 22.** Confusion matrix of 1st CNN 90-10 model for 3 languages.

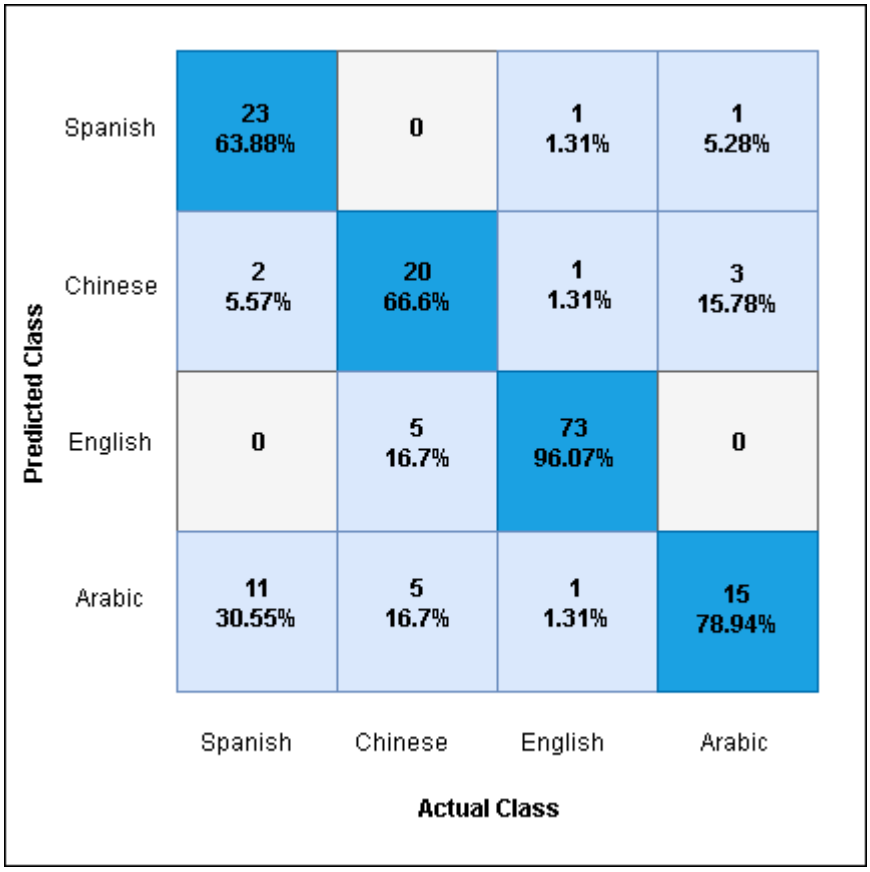


**Figure 23.** Confusion matrix of 1st CNN 90-10 model for 4 languages.

Continuing with the case of 80-20, the model for 2 languages achieved 93.44% accuracy at the 98th epoch. For 3 languages it scored the accuracy of 87.2% after 104 epochs and in the case of 4 languages it scored 75.77% accuracy after 35. The next figures represent the results for the case of 80-20 of the model.

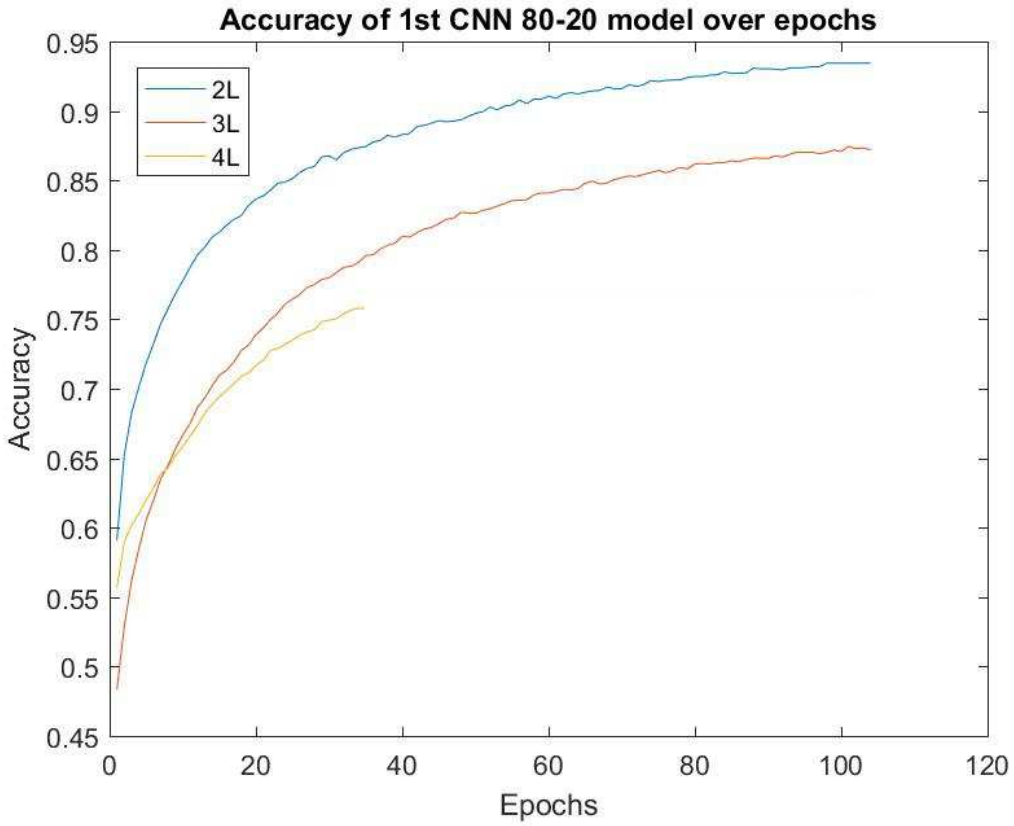


**Figure 24.** Accuracy of 1st CNN 80-20 model over epochs.

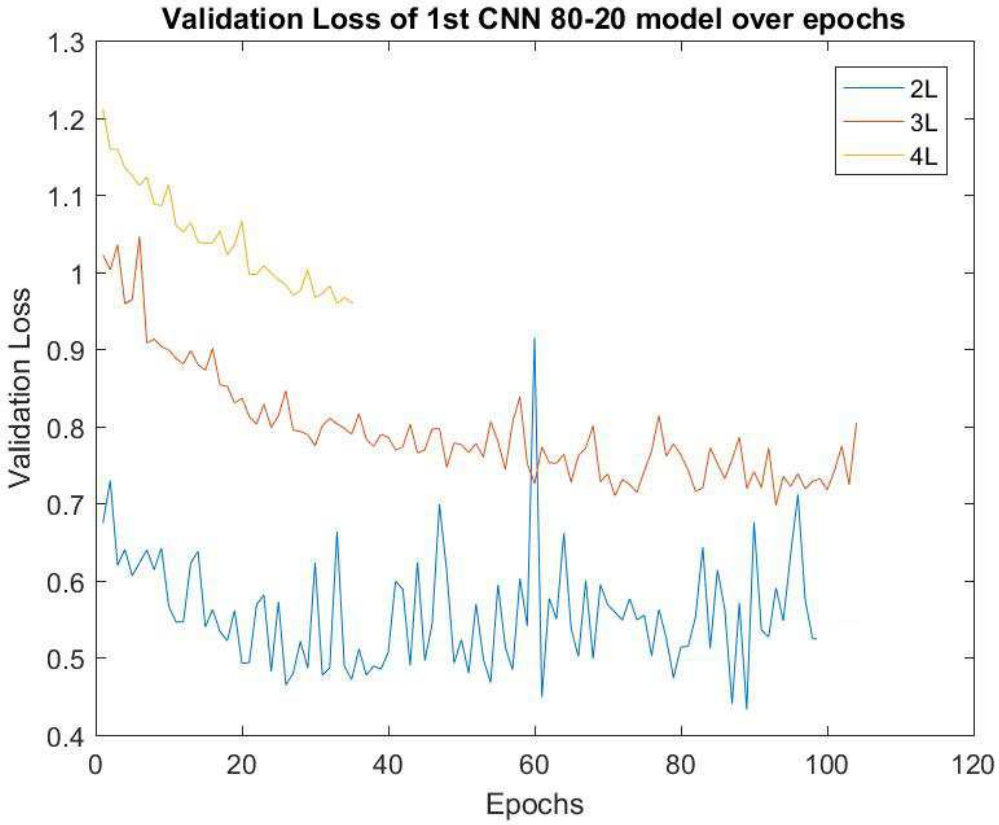


**Figure 25.** Validation loss of 1st CNN 80-20 model over epochs.

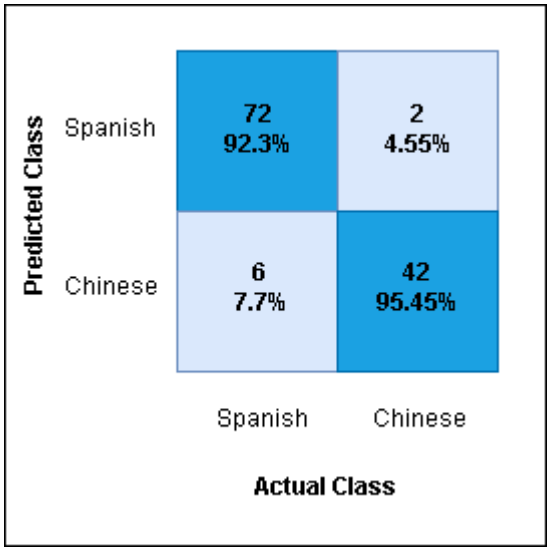


**Figure 26.** Confusion matrix of 1st CNN 80-20 model for 2 languages.

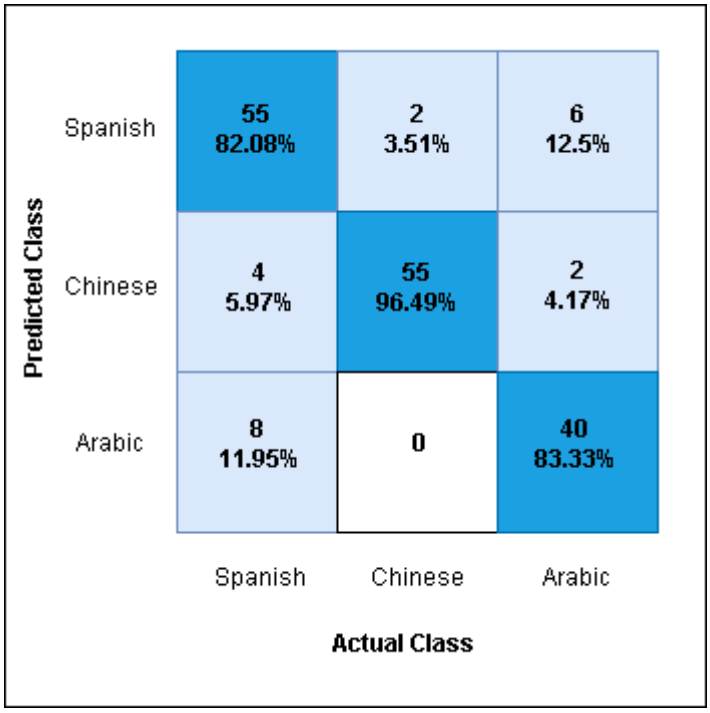


**Figure 27.** Confusion matrix of 1st CNN 80-20 model for 3 languages.

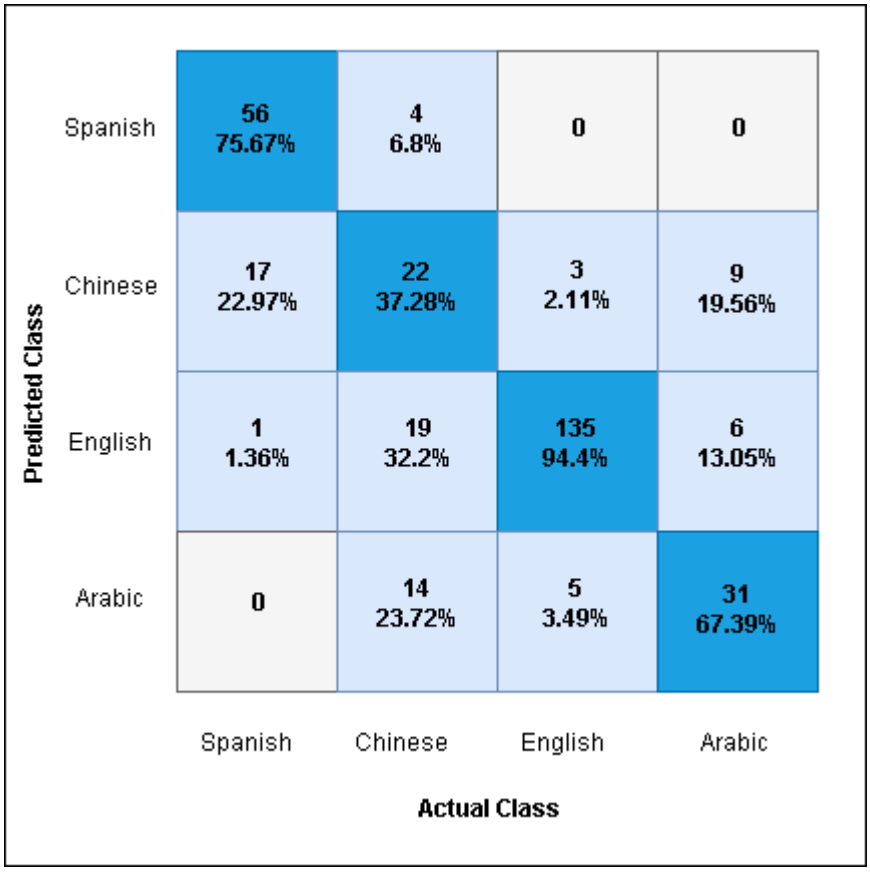


**Figure 28.** Confusion matrix of 1st CNN 80-20 model for 4 languages.

Concerning training and test samples of 70-30, in the case of 2 languages the model achieved 92.85% accuracy after 35 epochs. In the case of 3 languages the model scored 80.15% after 92 and for 4 languages the score achieved was 76.19% at the 35th epoch.

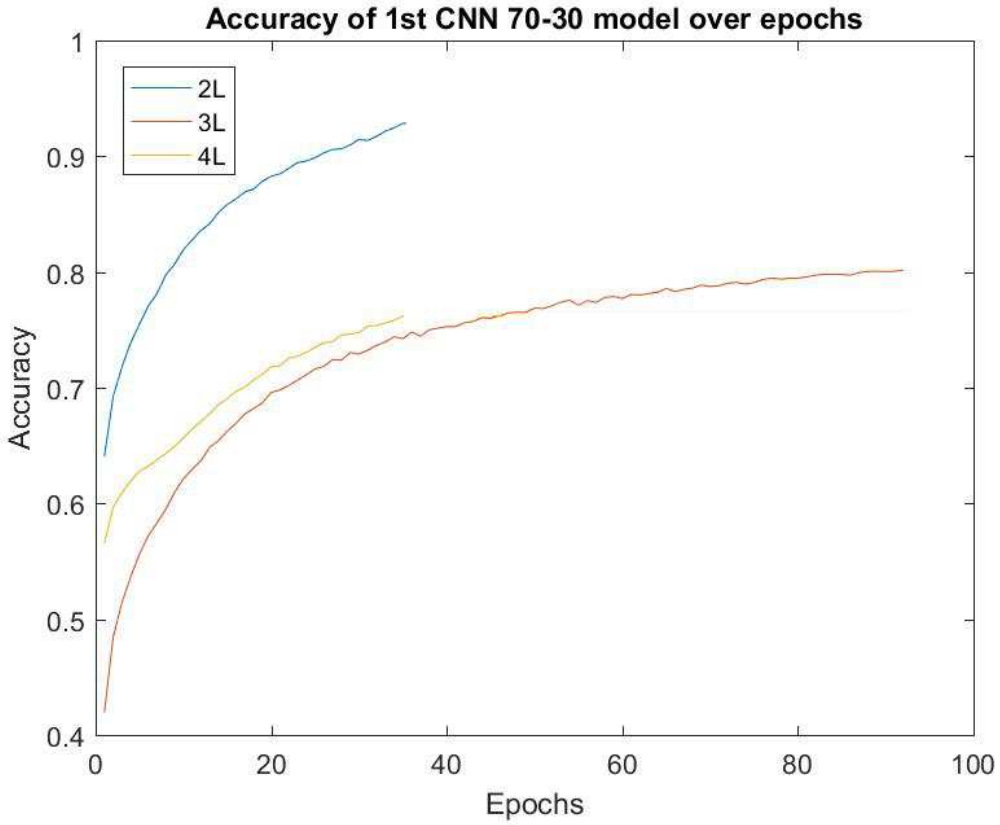


**Figure 29.** Accuracy of 1st CNN 70-30 model over epochs.

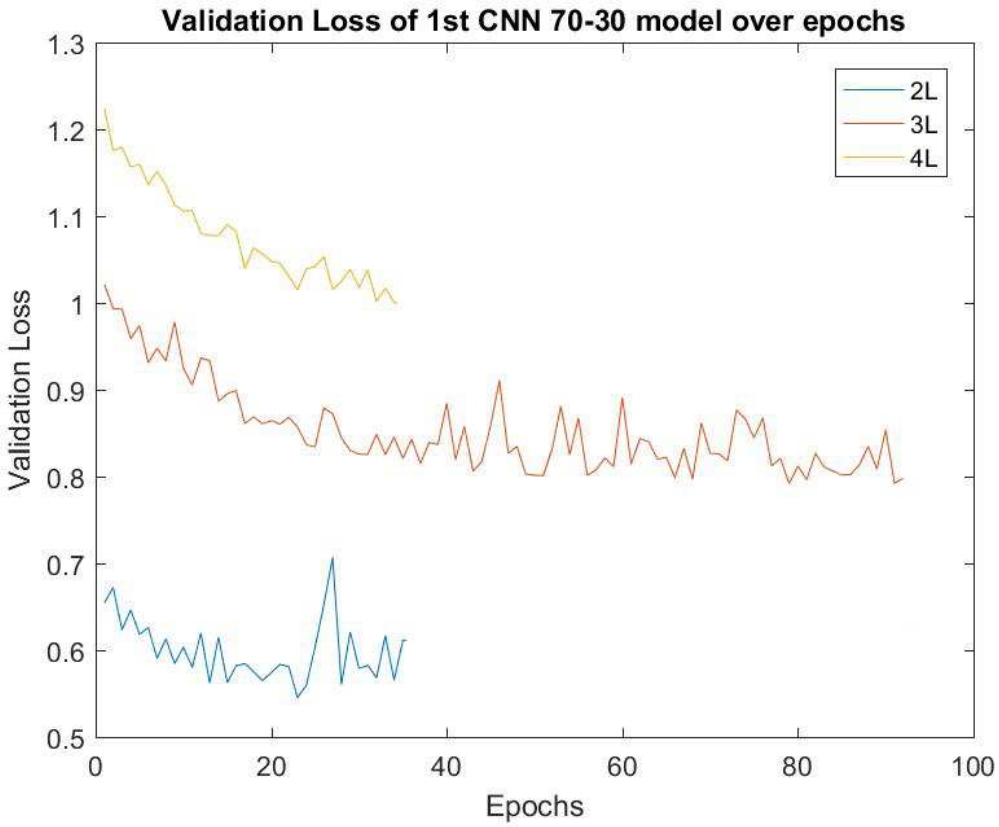


**Figure 30.** Validation loss of 1st CNN 70-30 model over epochs.

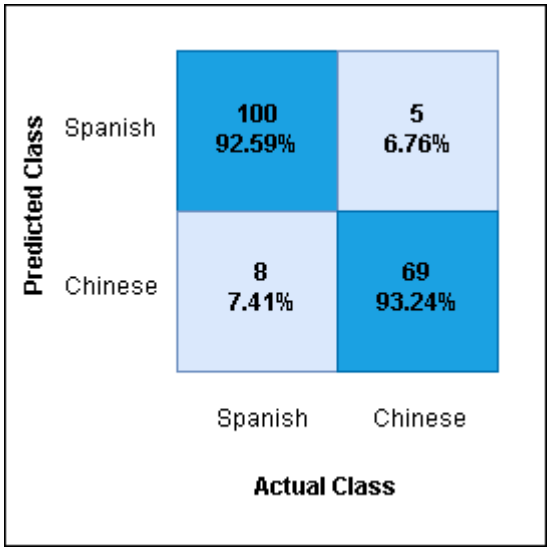


**Figure 31.** Confusion matrix of 1st CNN 70-30 model for 2 languages.

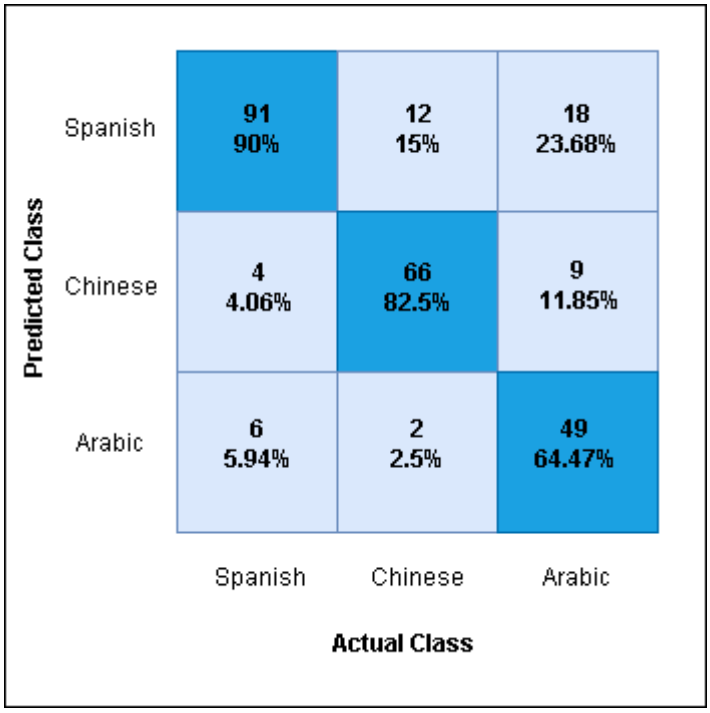


**Figure 32.** Confusion matrix of 1st CNN 70-30 model for 3 languages.

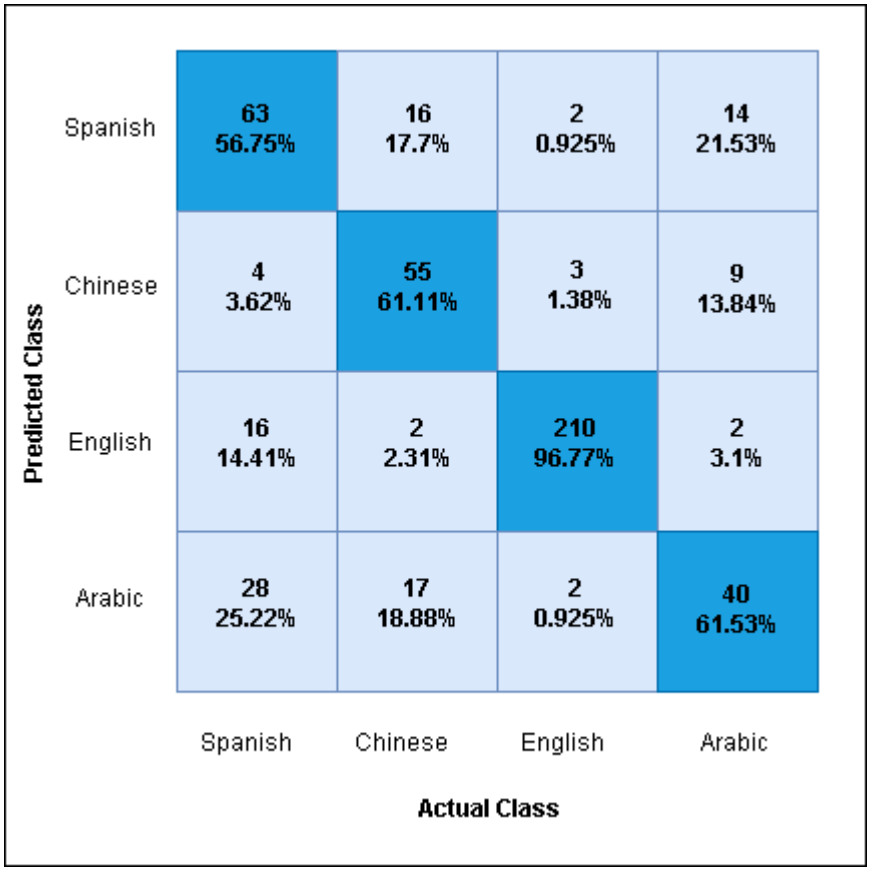


**Figure 33.** Confusion matrix of 1st CNN 70-30 model for 4 languages.

In terms of 60-40 of the current CNN, the model scored 90.53% accuracy after 35 epochs. The best accuracy for 3 languages in this category was 74.92% after 87 epochs and for 4 languages 65.06% accuracy at the 35th epoch.

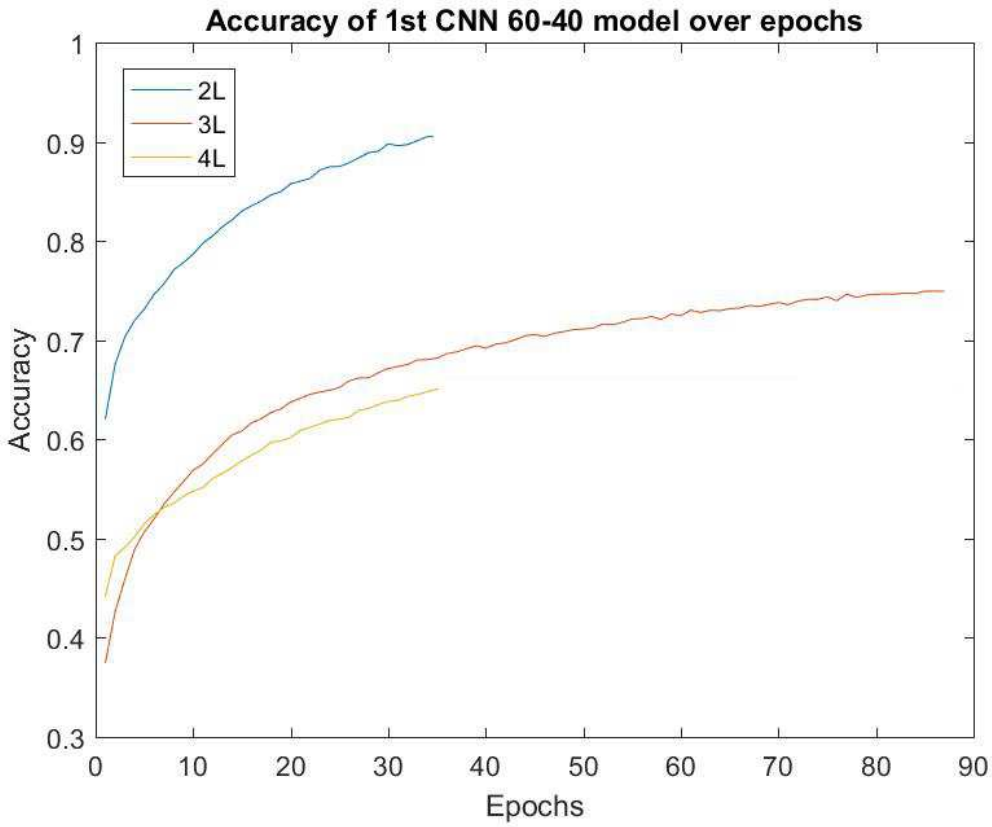


**Figure 34.** Accuracy of 1st CNN 60-40 model over epochs.

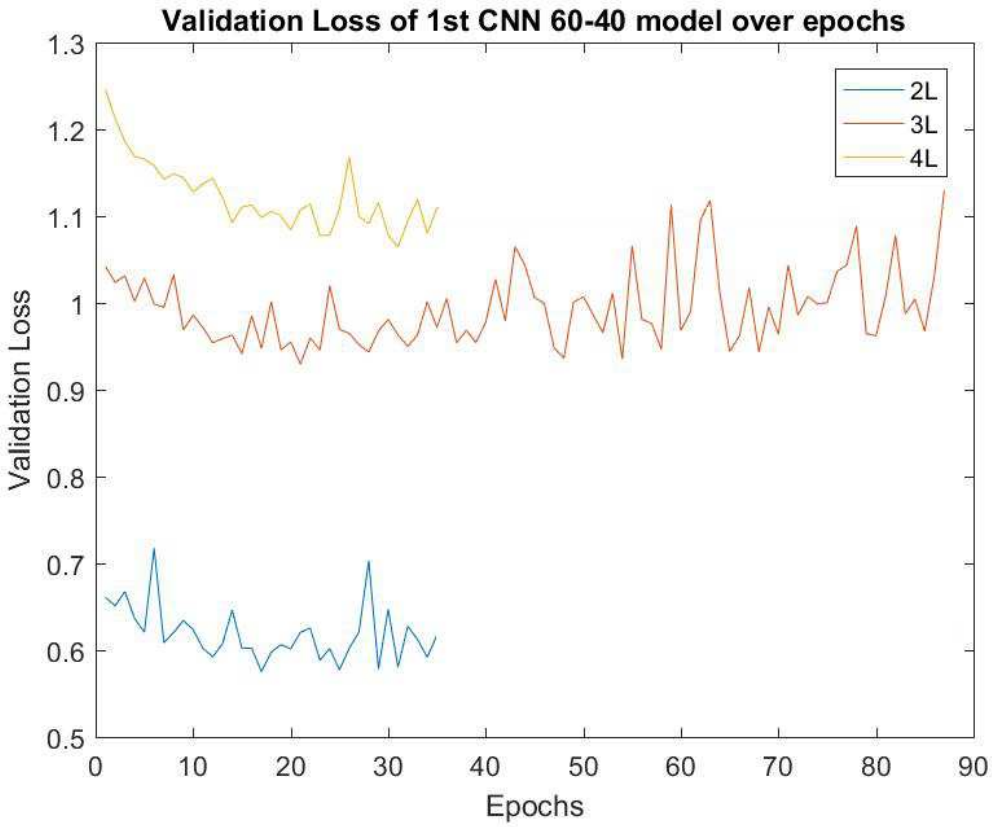


**Figure 35.** Validation loss of 1st CNN 60-40 model over epochs.

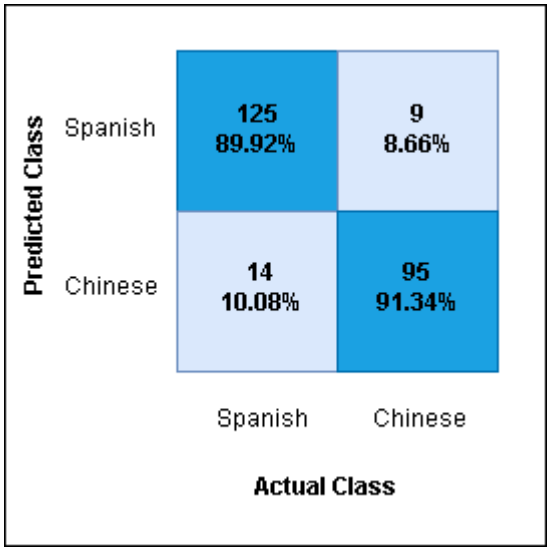


**Figure 36.** Confusion matrix of 1st CNN 60-40 model for 2 languages.

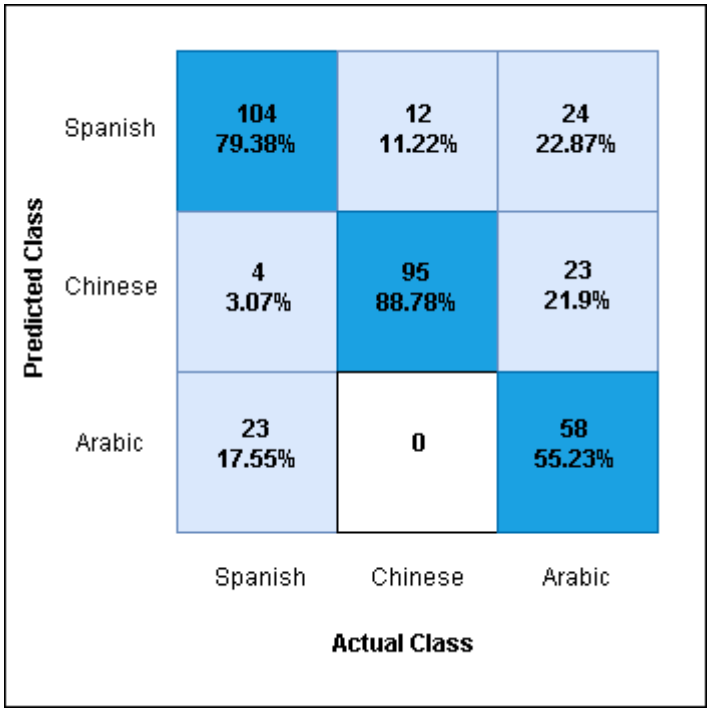


**Figure 37.** Confusion matrix of 1st CNN 60-40 model for 3 languages.

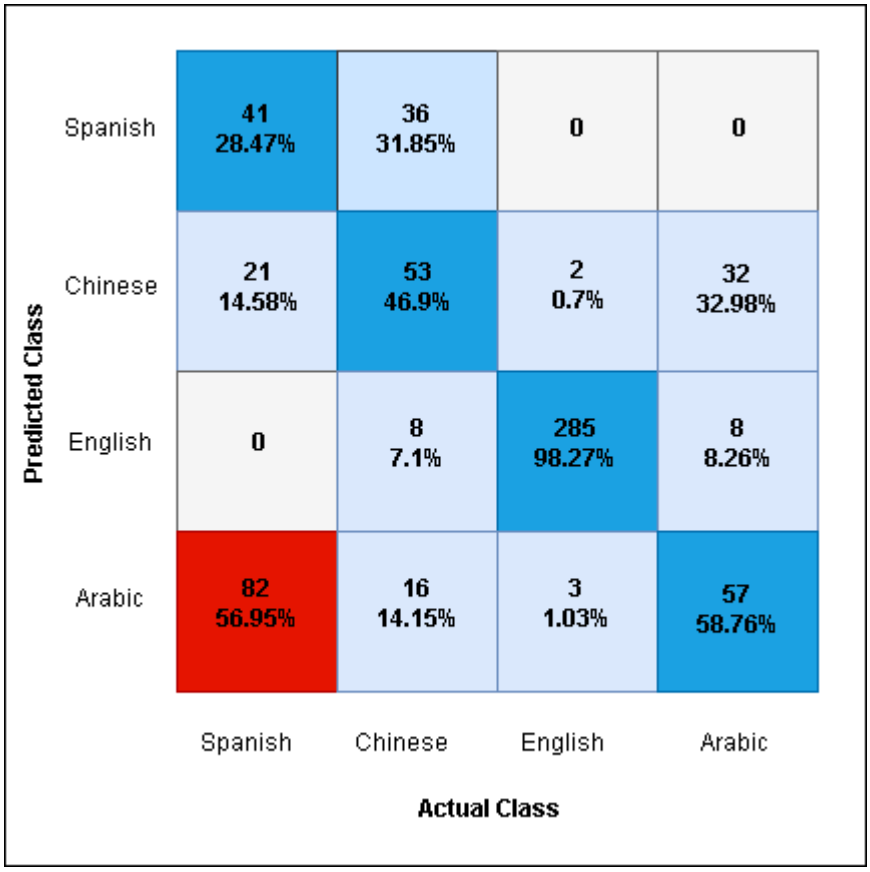


**Figure 38.** Confusion matrix of 1st CNN 60-40 model for 4 languages.

In the case of 50% training and 50% test samples the best accuracy for 2 languages is 89.43% at the 92nd epoch. The accuracy achieved for 3 languages is 71.96% after 81 epochs and 73.63% after 35 epochs for the case of 4 languages.

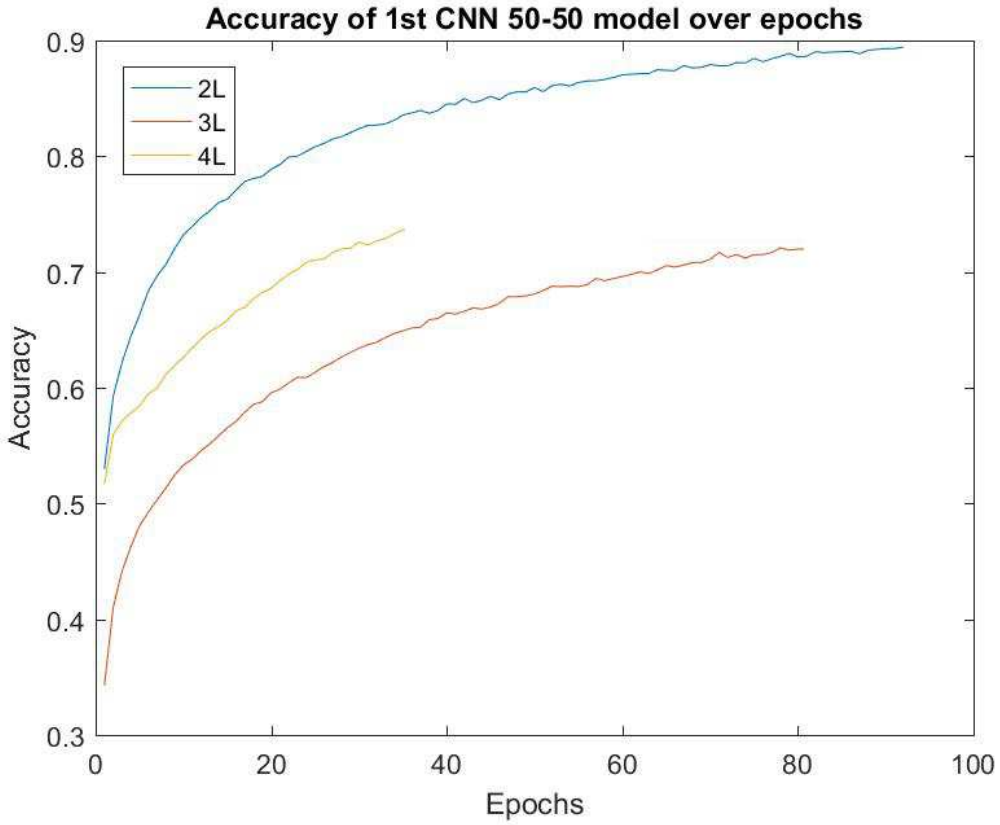


**Figure 39.** Accuracy of 1st CNN 50-50 model over epochs.

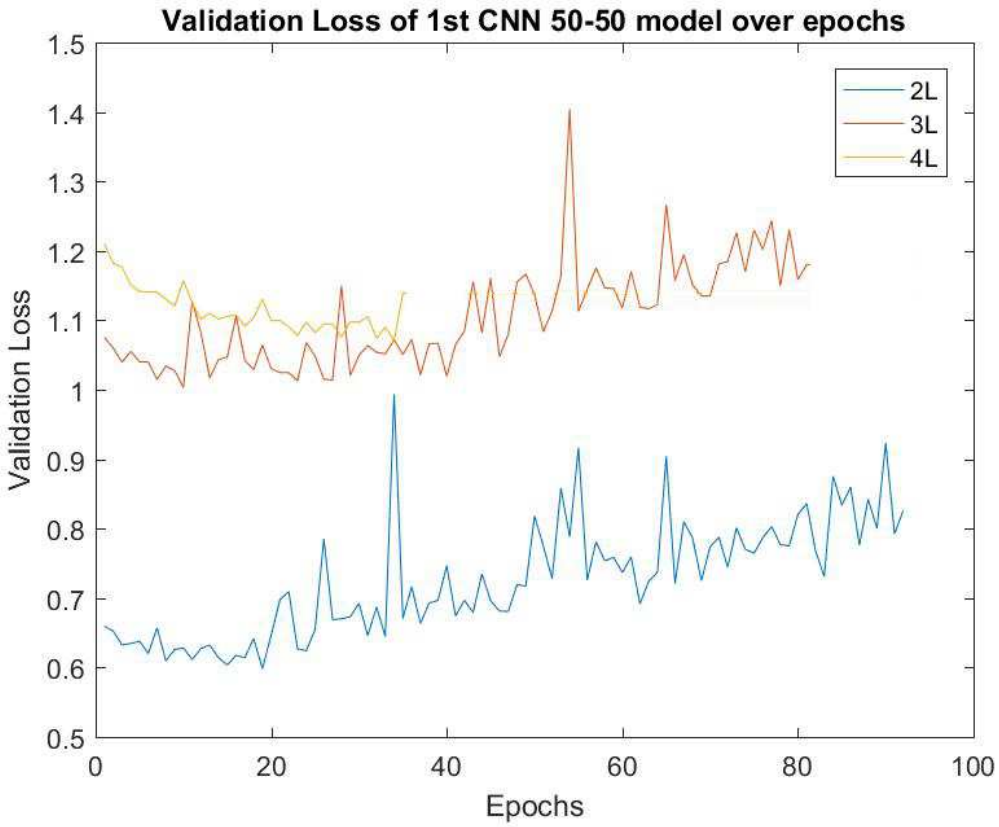


**Figure 40.** Validation loss of 1st CNN 50-50 model over epochs.

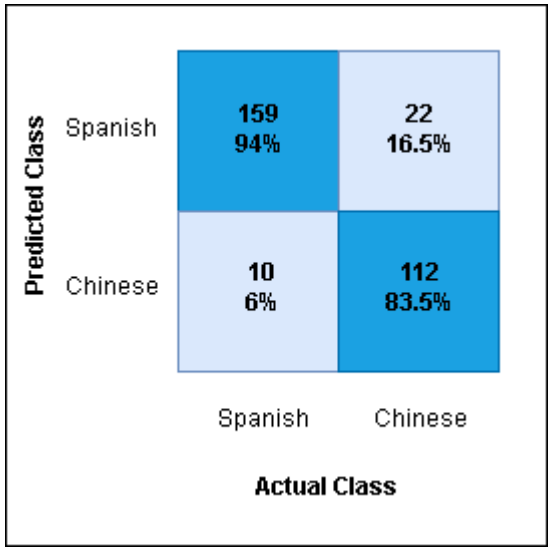


**Figure 41.** Confusion matrix of 1st CNN 50-50 model for 2 languages.

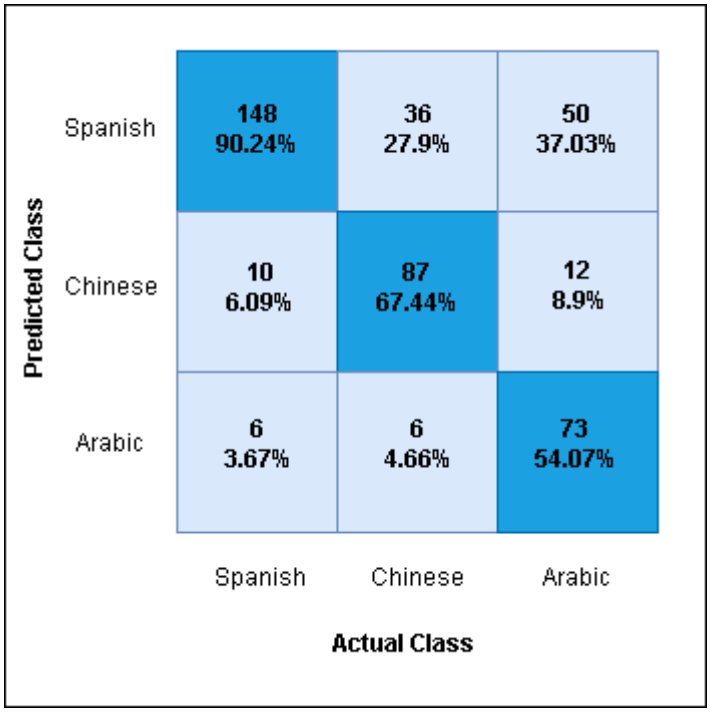


**Figure 42.** Confusion matrix of 1st CNN 50-50 model for 3 languages.

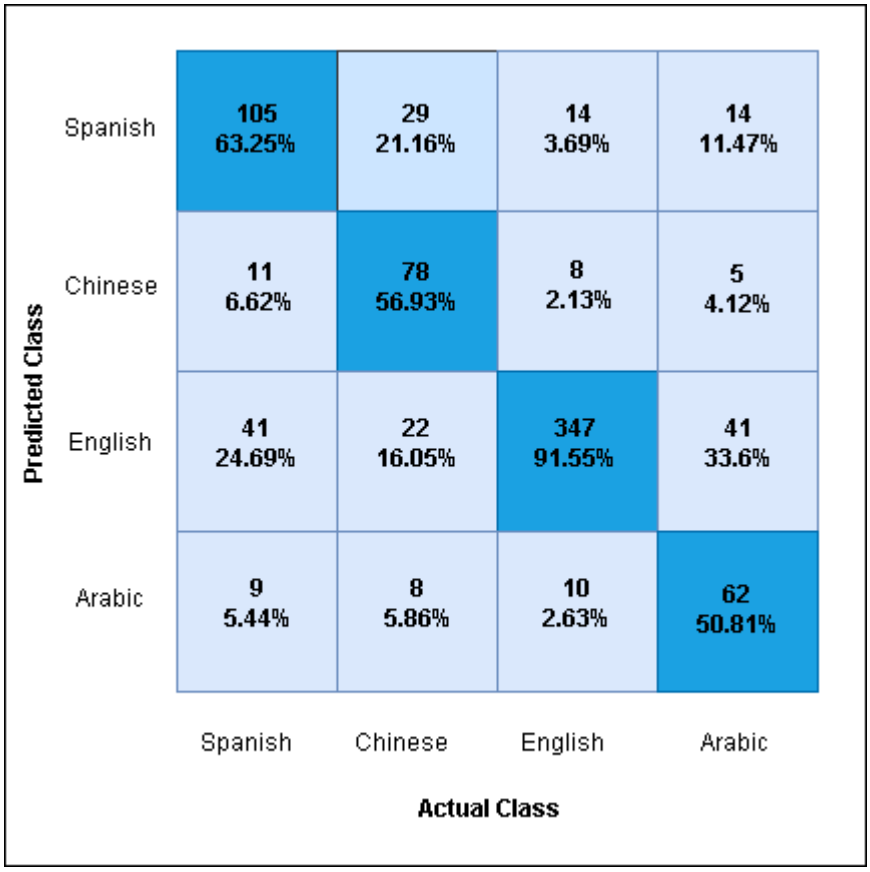


**Figure 43.** Confusion matrix of 1st CNN 50-50 model for 4 languages.

Concerning the approach of 40-60 the best accuracies are: 77.19% after 107 epochs for 2 languages; 70.81% at the 106th epoch for 3 languages and 69.53% after 96 epochs for 4 languages.

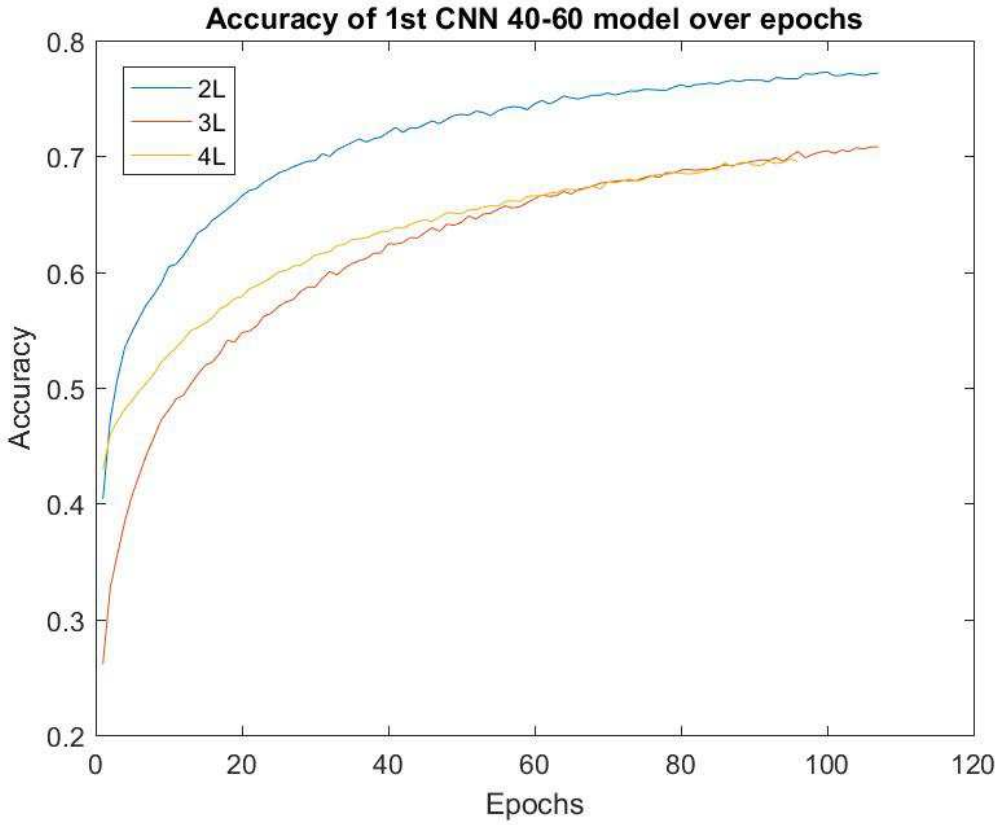


**Figure 44.** Accuracy of 1st CNN 40-60 model over epochs.

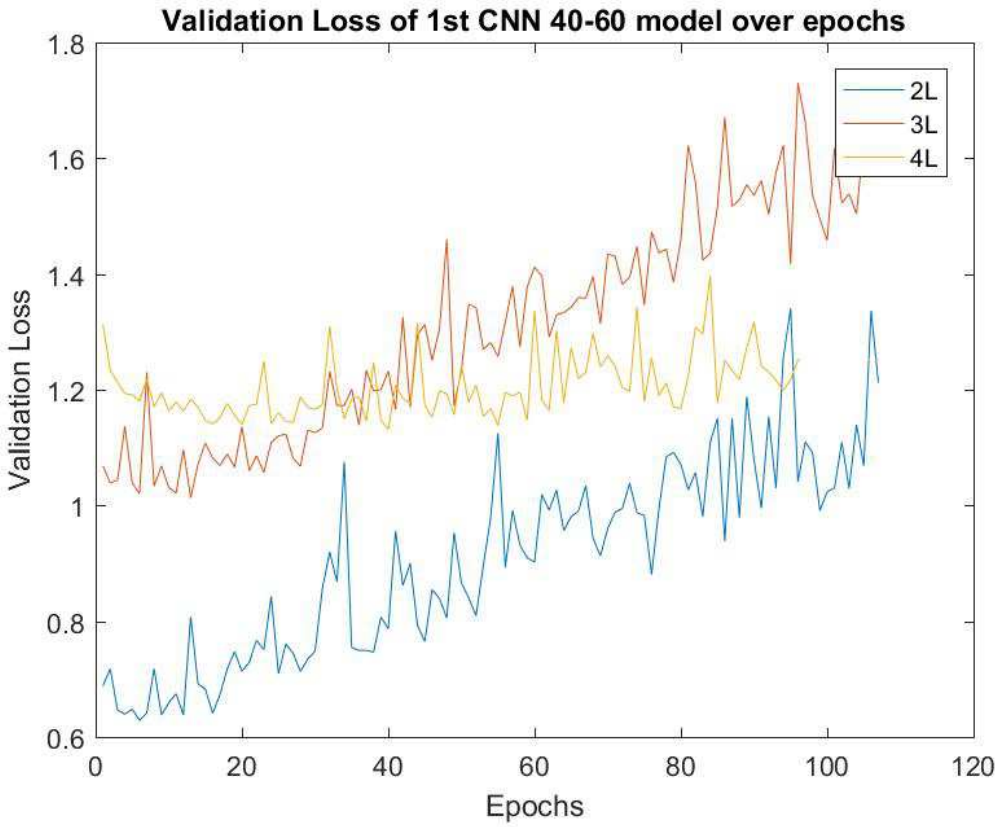


**Figure 45.** Validation loss of 1st CNN 40-60 model over epochs.

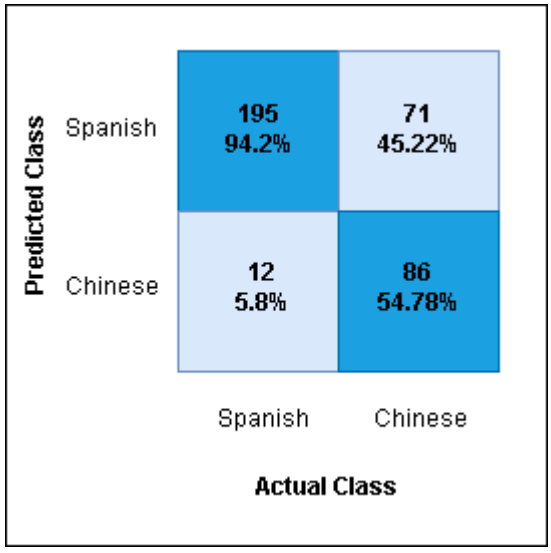


**Figure 46.** Confusion matrix of 1st CNN 40-60 model for 2 languages.

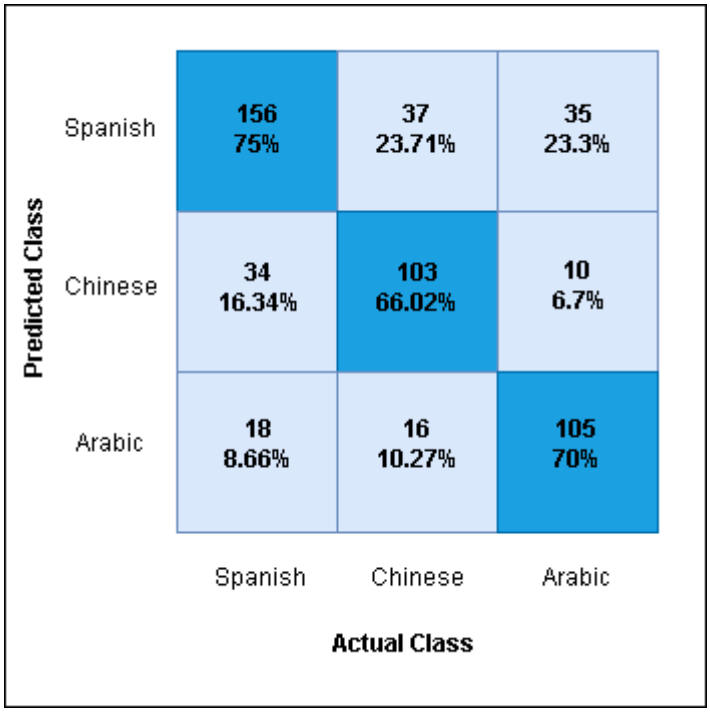


**Figure 47.** Confusion matrix of 1st CNN 40-60 model for 3 languages.

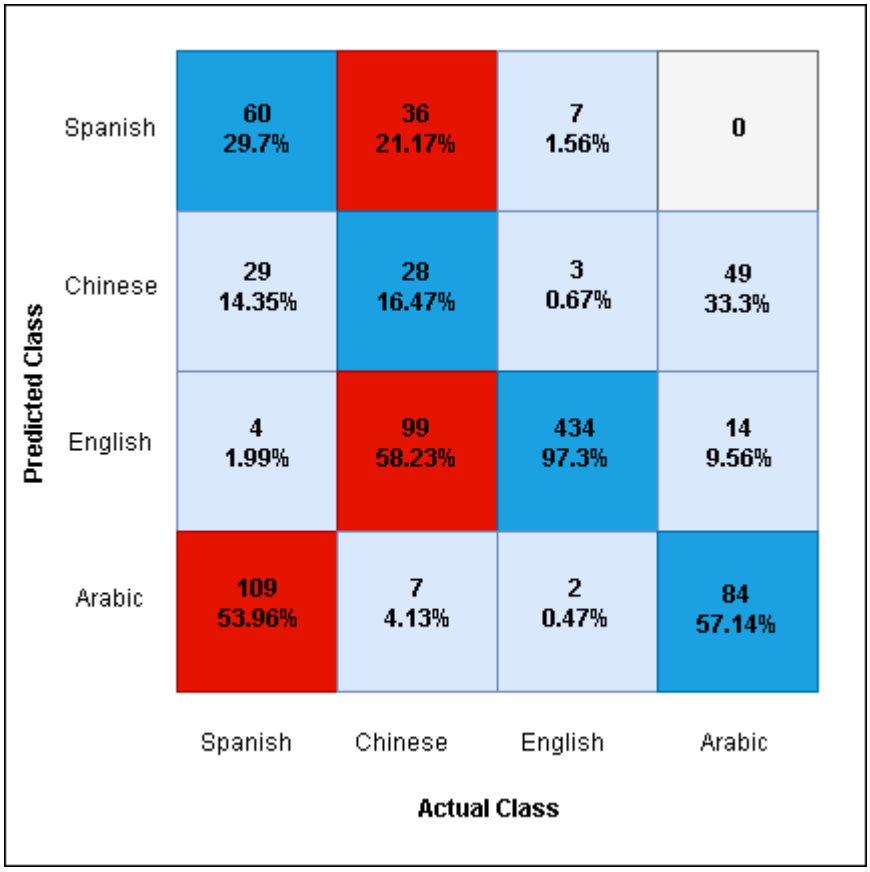


**Figure 48.** Confusion matrix of 1st CNN 40-60 model for 4 languages.

The best accuracies for the 30-70 case are: 76.94% at the 84th epoch for 2 languages; 63.66% after 129 epochs for 3 languages and 62.61% after 35 epochs for 4 languages.

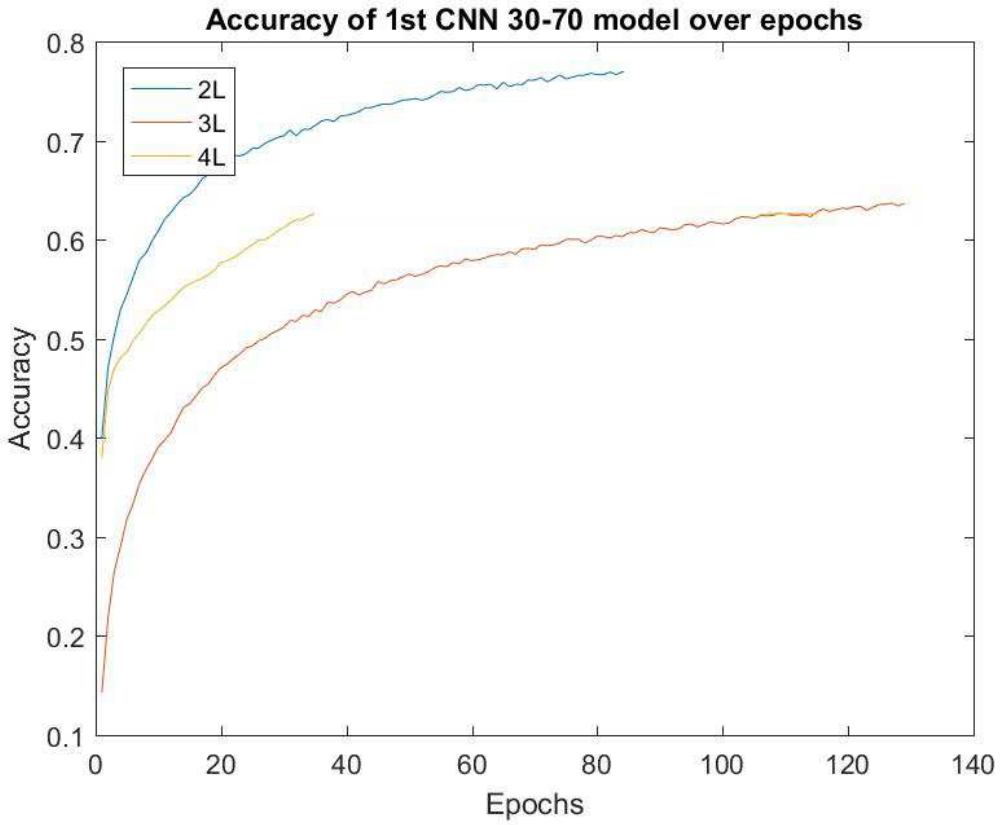


**Figure 49.** Accuracy of 1st CNN 30-70 model over epochs.

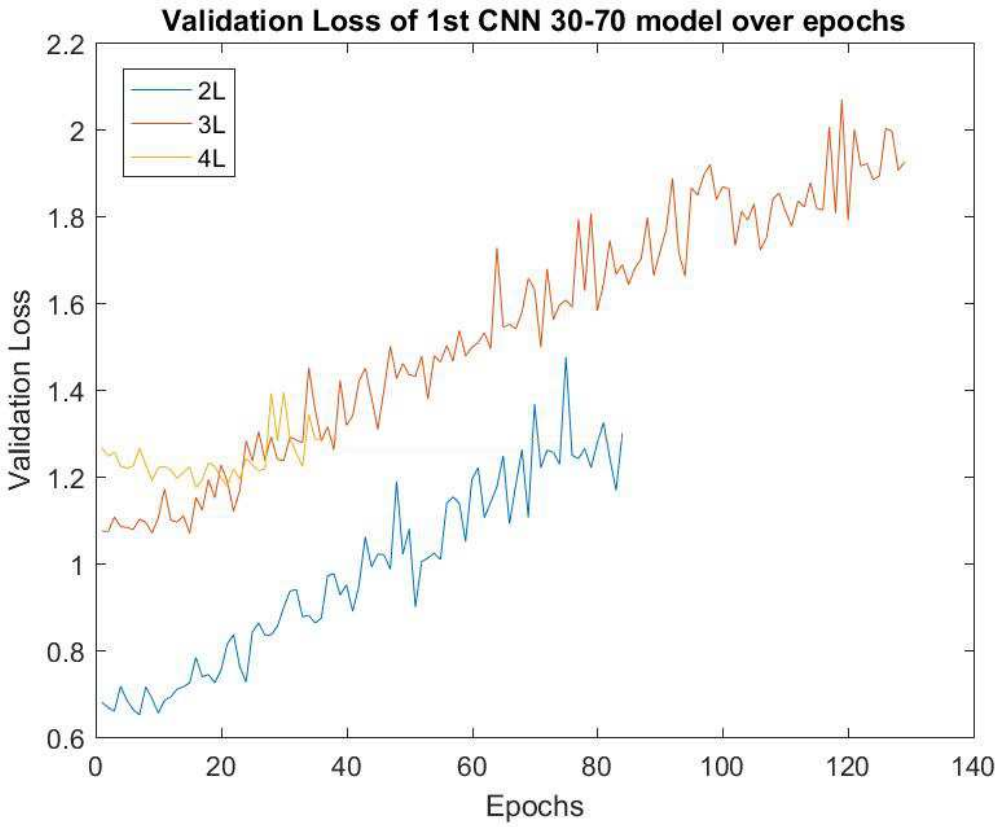


**Figure 50.** Validation loss of 1st CNN 30-70 model over epochs.

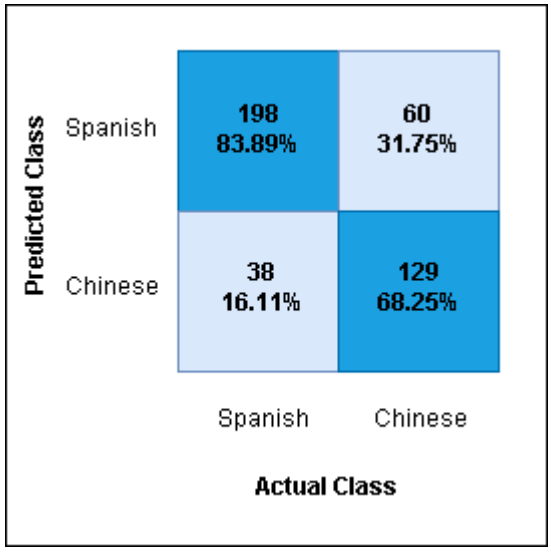


**Figure 51.** Confusion matrix of 1st CNN 30-70 model for 2 languages.

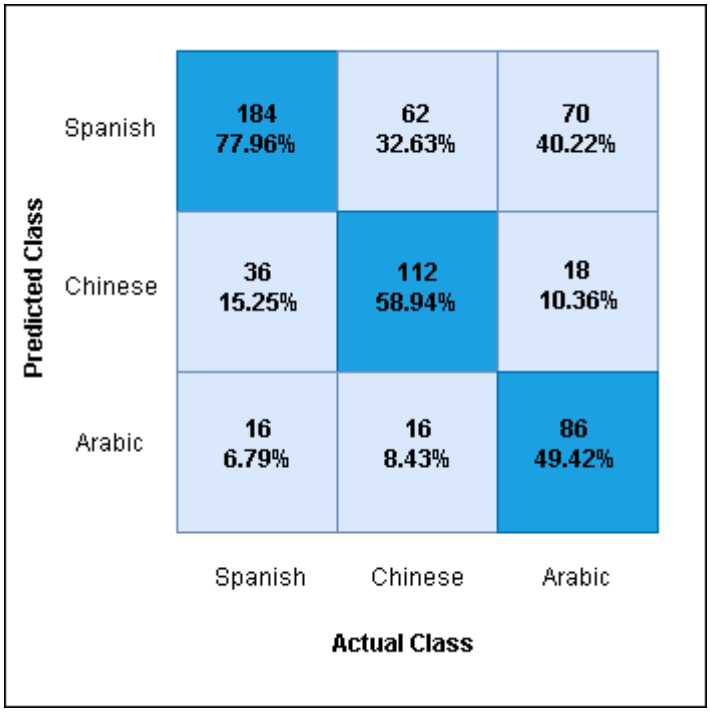


**Figure 52.** Confusion matrix of 1st CNN 30-70 model for 3 languages.

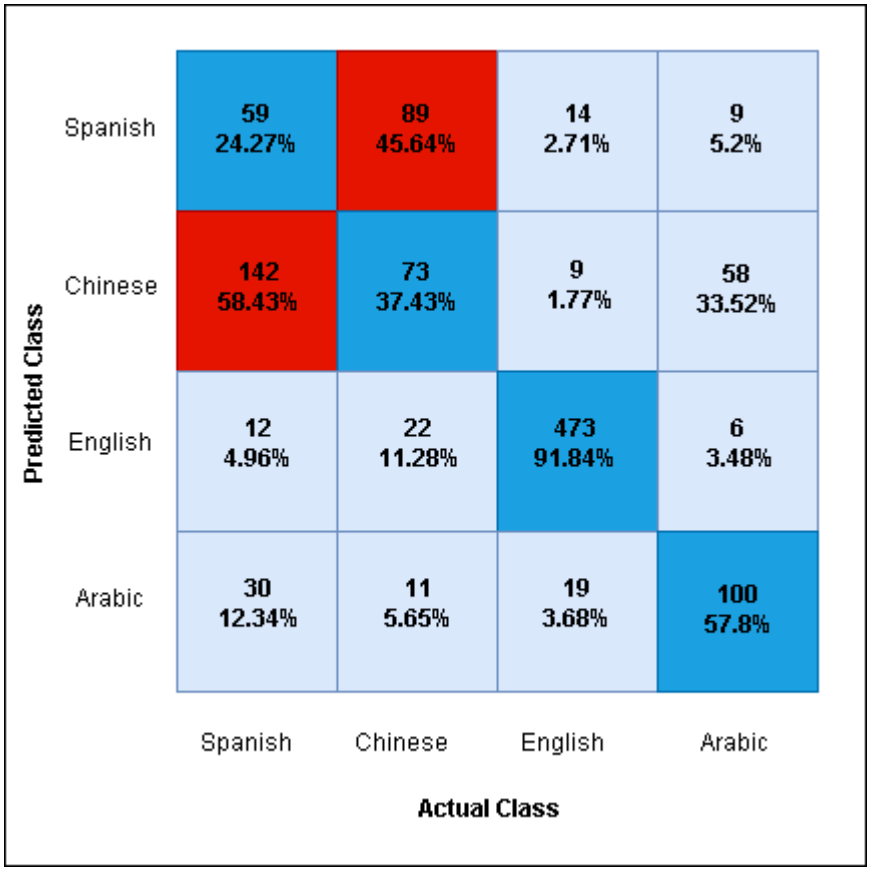


**Figure 53.** Confusion matrix of 1st CNN 30-70 model for 4 languages.

In terms of 20-80, the best accuracy for 2 languages is 73.81% after 35 epochs, for 3 languages is 56.49% at the 92nd epoch and for 4 languages 57.42% after 35 epochs.

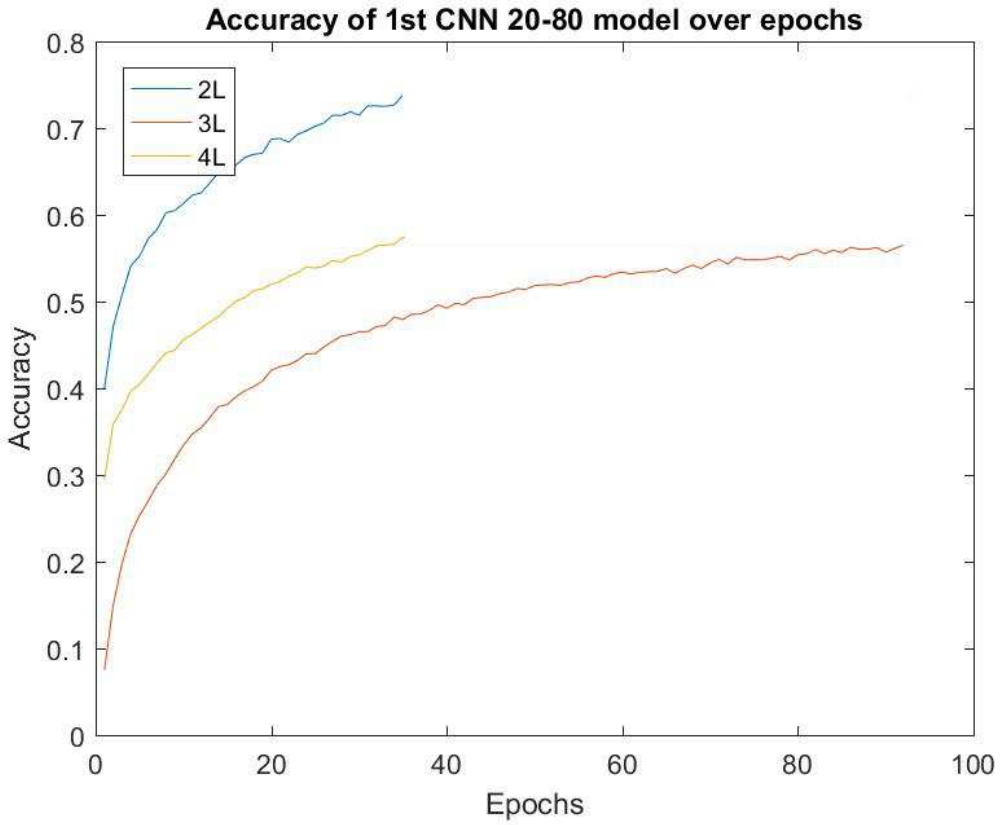


**Figure 54.** Accuracy of 1st CNN 20-80 model over epochs.

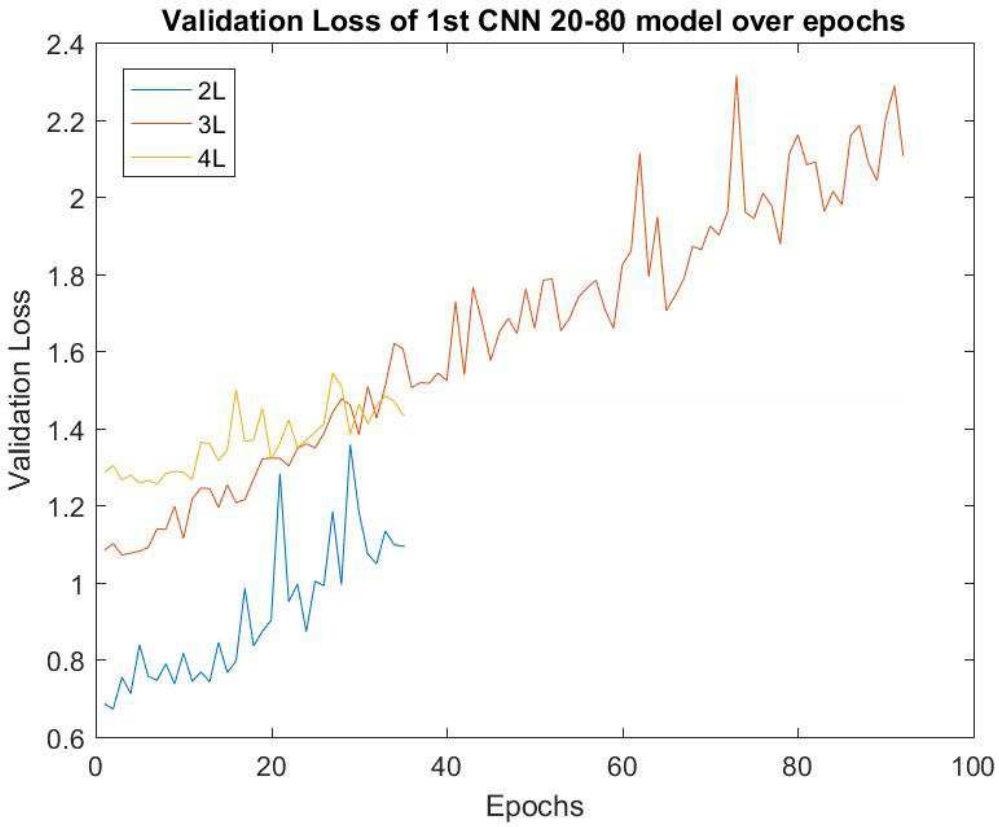


**Figure 55.** Validation loss of 1st CNN 20-80 model over epochs.

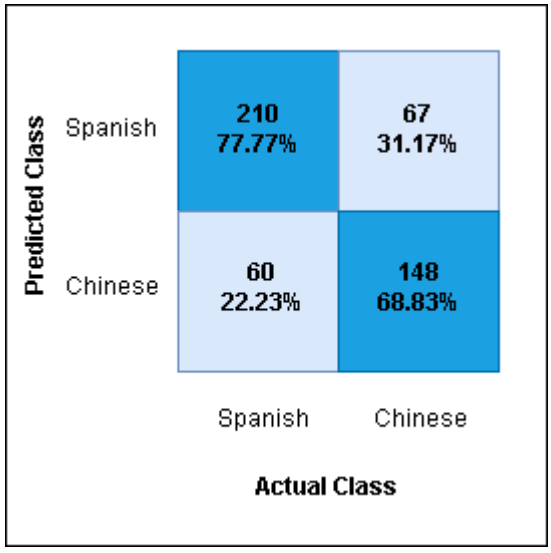


**Figure 56.** Confusion matrix of 1st CNN 20-80 model for 2 languages.

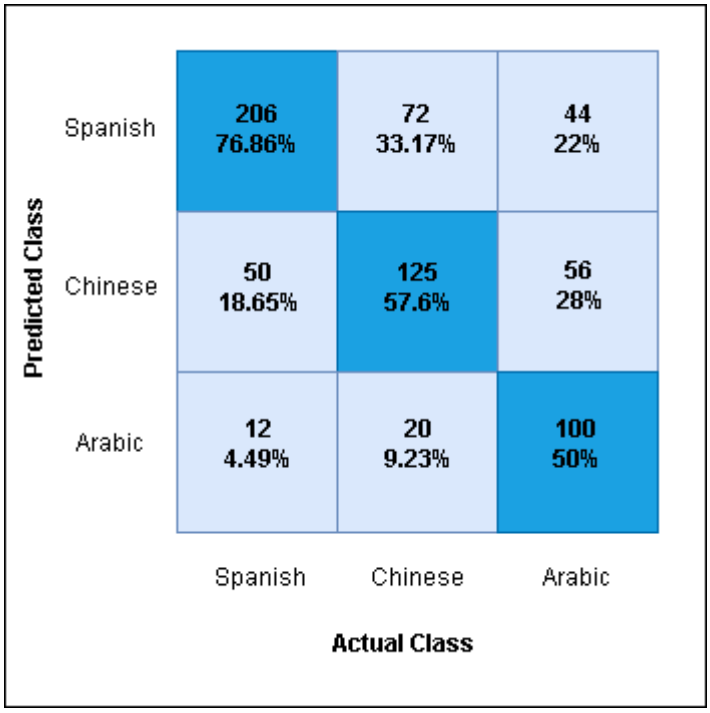


**Figure 57.** Confusion matrix of 1st CNN 20-80 model for 3 languages.

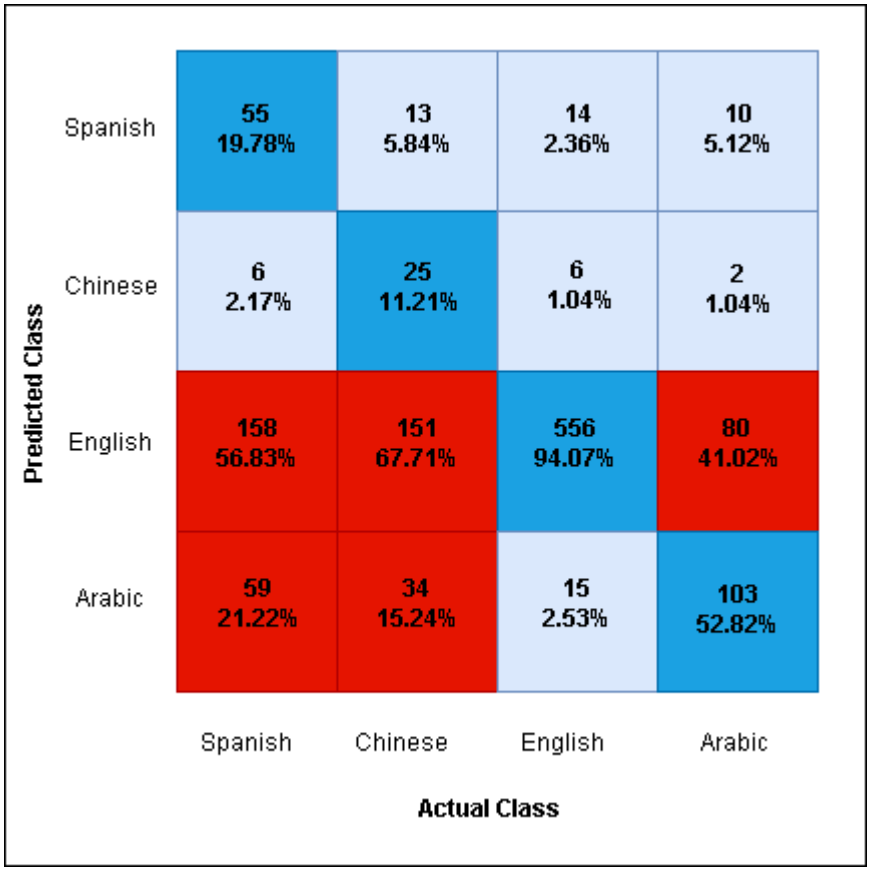


**Figure 58.** Confusion matrix of 1st CNN 20-80 model for 4 languages.

In the last case 10-90 the results are: 68.49% accuracy after 58 epochs for 2 languages; 49.28% after 35 epochs for 3 languages and 54.55% at the 110th epoch for 4 languages.

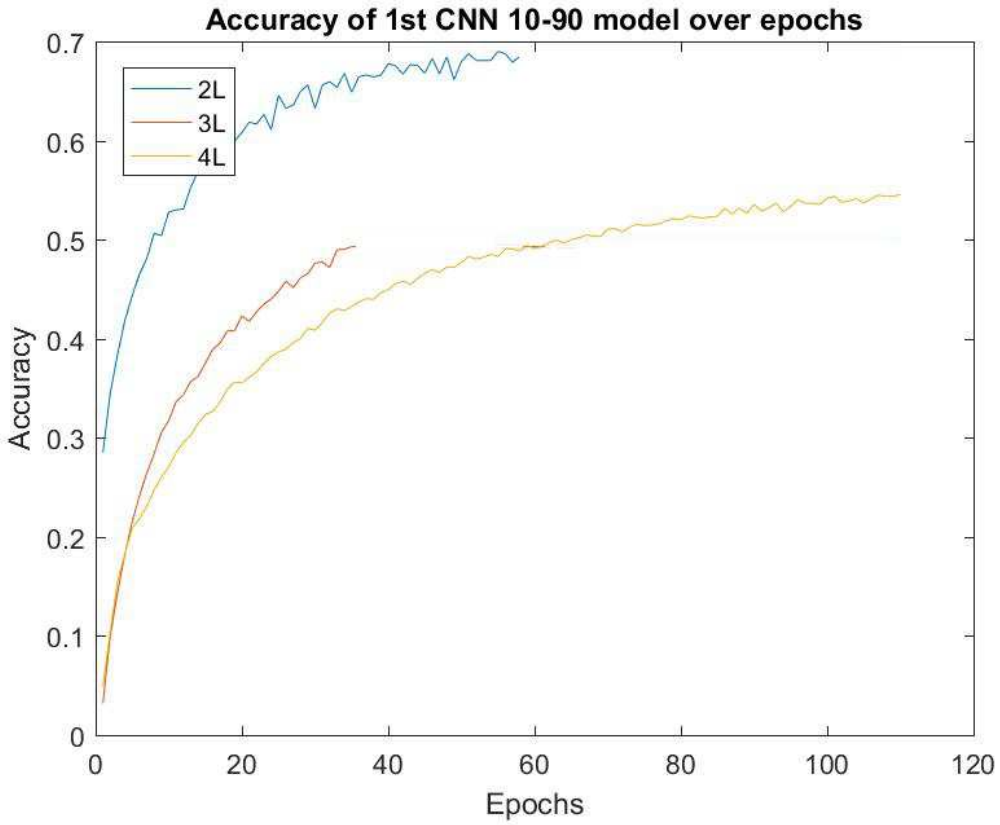


**Figure 59.** Accuracy of 1st CNN 10-90 model over epochs.

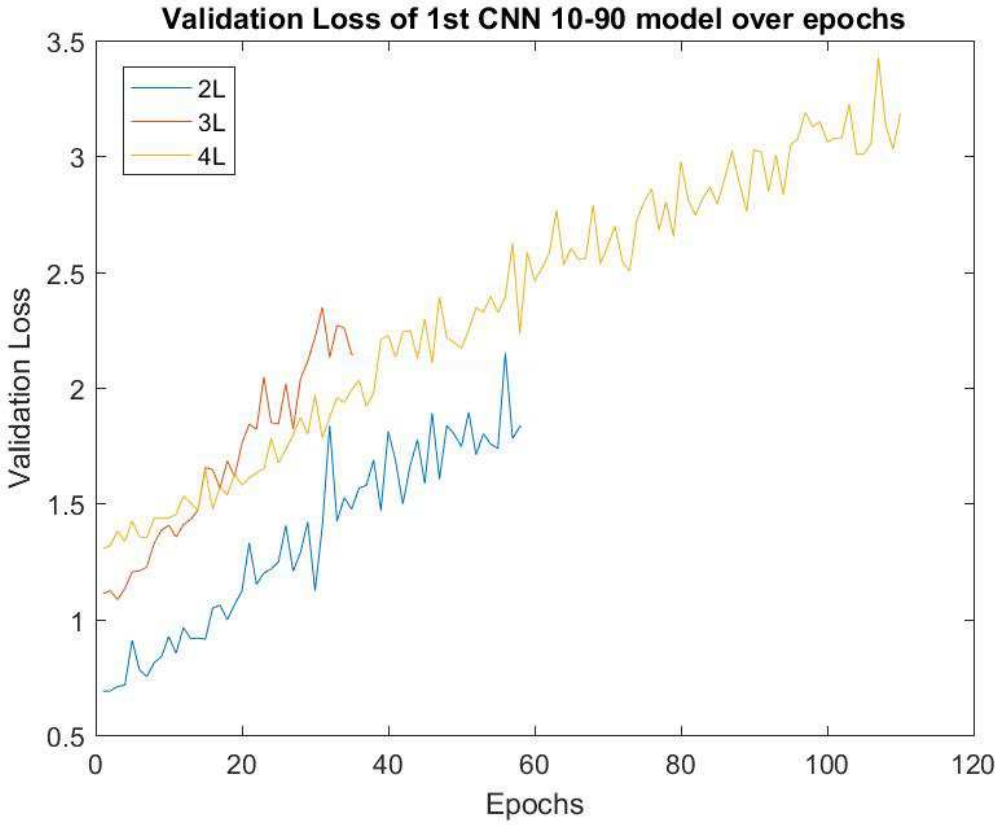


**Figure 60.** Validation loss of 1st CNN 10-90 model over epochs.

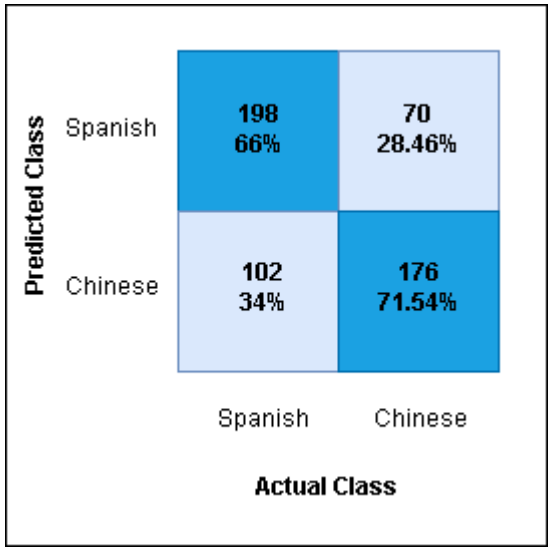


**Figure 61.** Confusion matrix of 1st CNN 10-90 model for 2 languages.

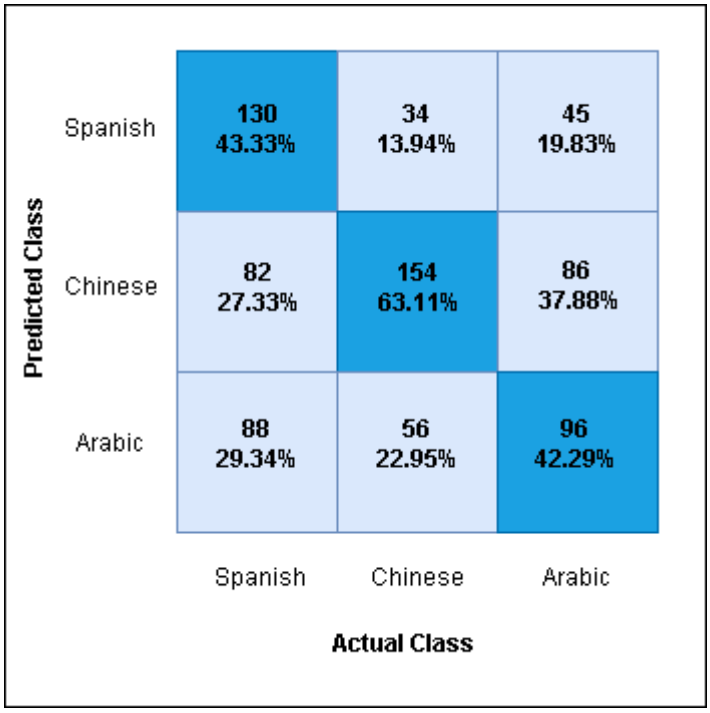


**Figure 62.** Confusion matrix of 1st CNN 10-90 model for 3 languages.

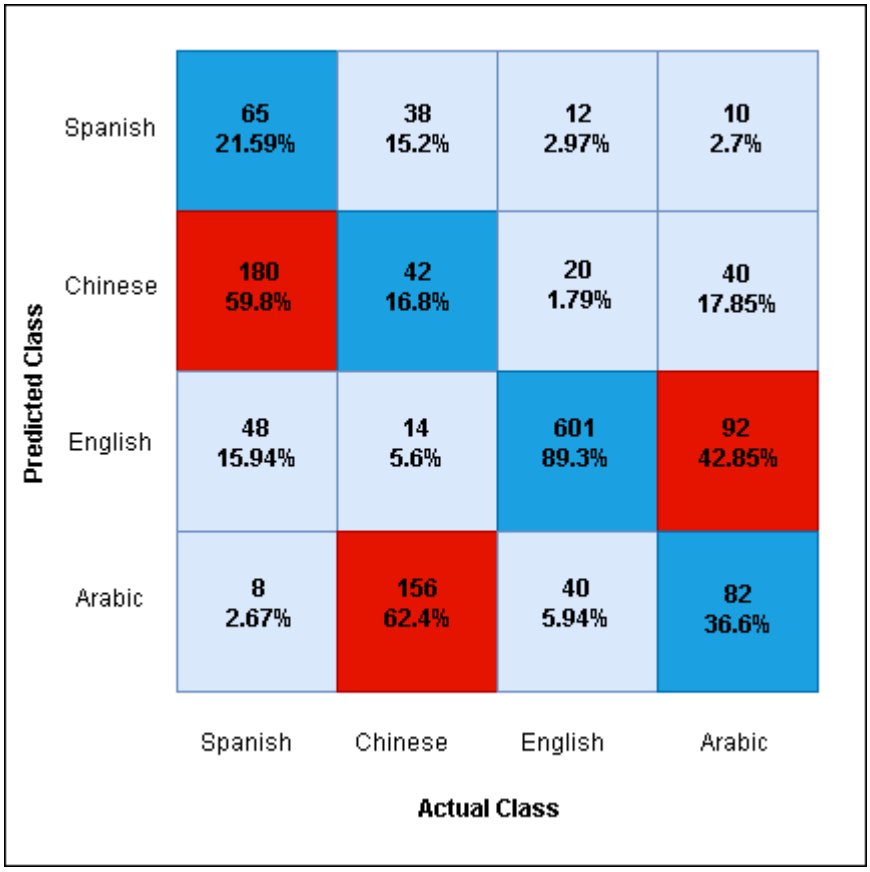


**Figure 63.** Confusion matrix of 1st CNN 10-90 model for 4 languages.

**Table 3.** Results from the experiments of the first CNN (languages, accuracy, epochs).

| **90-10** | **80-20** | **70-30** |
|---|---|---|
| 2L: 96.72% epochs = 83 | 2L: 93.44% epochs = 98 | 2L: 92.85% epochs = 35 |
| 3L: 84.88% epochs = 88 | 3L: 87.2% epochs = 104 | 3L: 80.15% epochs = 92 |
| 4L: 74.27% epochs = 35 | 4L: 75.77% epochs = 35 | 4L: 76.19% epochs = 35 |

| **60-40** | **50-50** | **40-60** |
|---|---|---|
| 2L: 90.53% epochs = 35 | 2L: 89.43% epochs = 92 | 2L: 77.19% epochs = 107 |
| 3L: 74.92% epochs = 87 | 3L: 71.96% epochs = 81 | 3L: 70.81% epochs = 106 |
| 4L: 65.06% epochs = 35 | 4L: 73.63% epochs = 35 | 4L: 69.53% epochs = 96 |

| **30-70** | **20-80** | **10-90** |
|---|---|---|
| 2L: 76.94% epochs = 84 | 2L: 73.81% epochs = 35 | 2L: 68.49% epochs = 58 |
| 3L: 63.66% epochs = 129 | 3L: 56.49% epochs = 92 | 3L: 49.28% epochs = 35 |
| 4L: 62.61% epochs = 35 | 4L: 57.42% epochs = 35 | 4L: 54.55% epochs = 110 |

As it can be seen the best accuracy for 2 languages (Chinese and Spanish) was achieved in the case of 90-10 with the value of 96.72%. The optimal accuracy for 3 languages (Chinese, Spanish and Arabic) can be seen in the case of 80-20 having a value of 87.2%. Concerning 4 languages (Chinese, Spanish, English and Arabic) the best accuracy is in the case of 70-30 with 76.19%.

The above results show that the English accents of native speakers of two different languages (Chinese and Spanish) are quite different and this difference can be seen quickly in the training of the model, achieving high accuracy in the beginning (90-10 case). While more native languages are added the difference between them seems to be a difficult task even for the computer to detect.

Concerning the statistics of the models running on the current computer it can be said that the average time for loading the WAV files is around 8 minutes for 2 languages, 11 minutes for 3 languages and 20 minutes for 4 languages. The time needed for converting the WAV files to MFCCs is 41 seconds for 2 languages, 1 minute for 3 languages and 1:39 minutes for 4 languages. Lastly, the average time needed for training the models is 55 minutes for 2 languages, 1 hour and 18 minutes for 3 languages and 2 hours for 4 languages. It must be noted that the time needed for training a model depends mainly on the number of languages it contains and the number of epochs.

## 5.3. Experiments with the second CNN architecture (Sigmoid activation function on 3rd layer)

The results of the second CNN are slightly different from the first architecture. Specifically, in the case of 90-10 the model scored the best accuracy of 95.08% after 69 epochs for 2 languages. For 3 languages the model scored 81.39% accuracy at the 100th epoch and for 4 languages 75.77% accuracy after 50 epochs. The next figures represent the accuracies for each case with 90% training samples and 10% test samples, the value of the loss function and the confusion matrices accordingly as it was shown in the previous section.

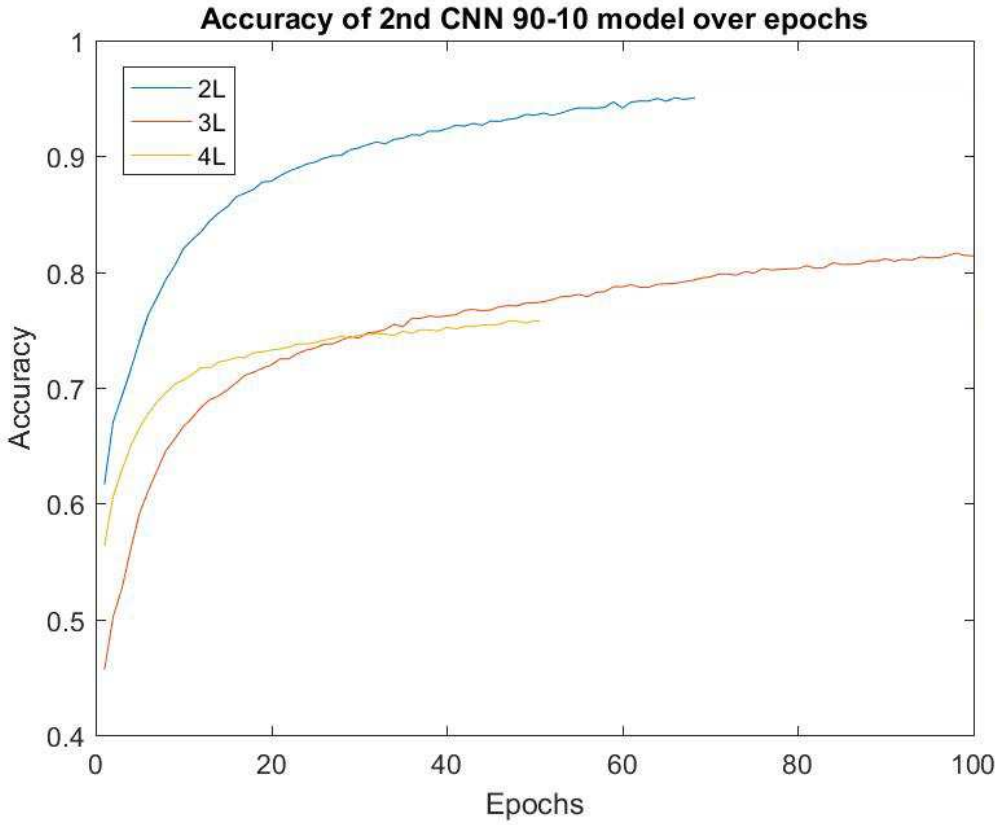


**Figure 64.** Accuracy of 2nd CNN 90-10 model over epochs.

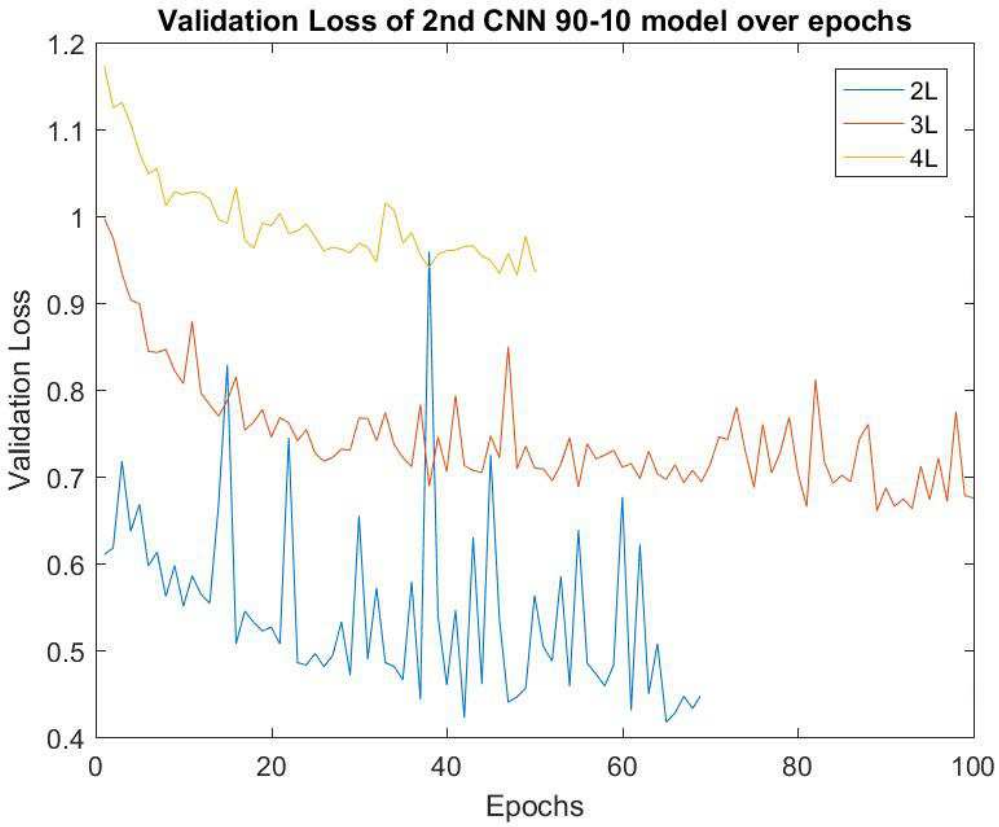


**Figure 65.** Validation loss of 2nd CNN 90-10 model over epochs.

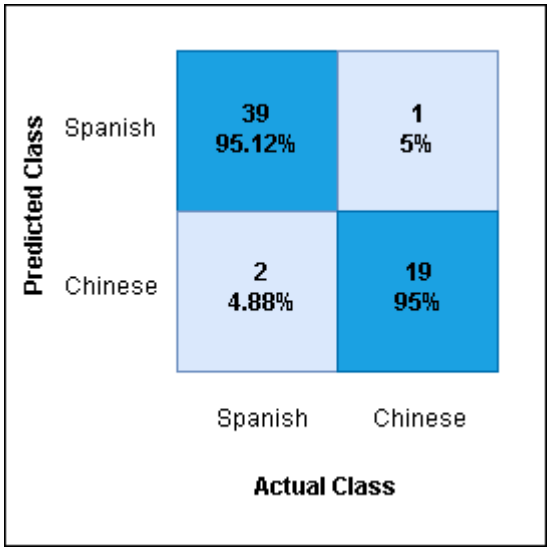


**Figure 66.** Confusion matrix of 2nd CNN 90-10 model for 2 languages.

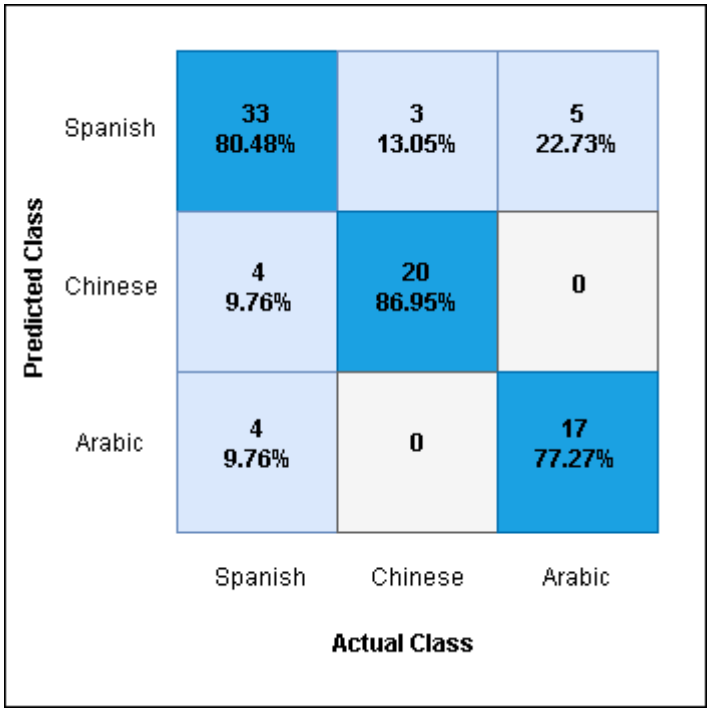


**Figure 67.** Confusion matrix of 2nd CNN 90-10 model for 3 languages.


**Figure 68.** Confusion matrix of 2nd CNN 90-10 model for 4 languages.

Concerning the case of 80-20 the best accuracy is 95.05% after 100 epochs for 2 languages; 84.3% accuracy after 87 epochs for 3 languages and 74.22% at the 50th epoch for 4 languages.

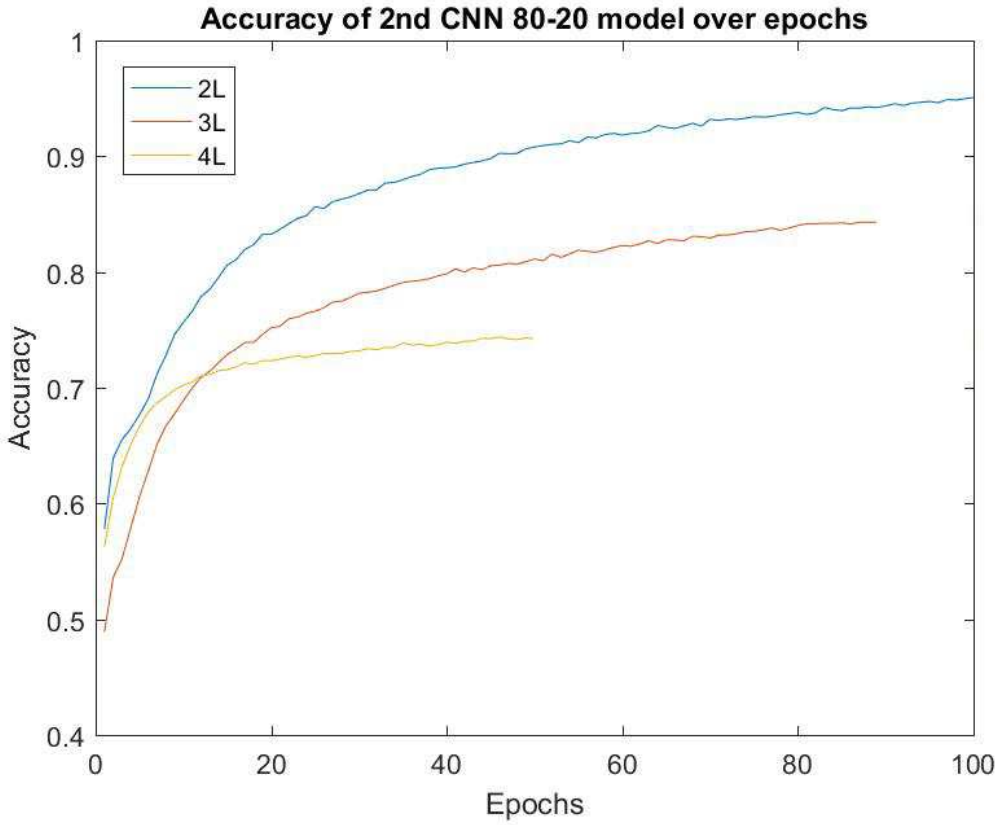


**Figure 69.** Accuracy of 2nd CNN 80-20 model over epochs.

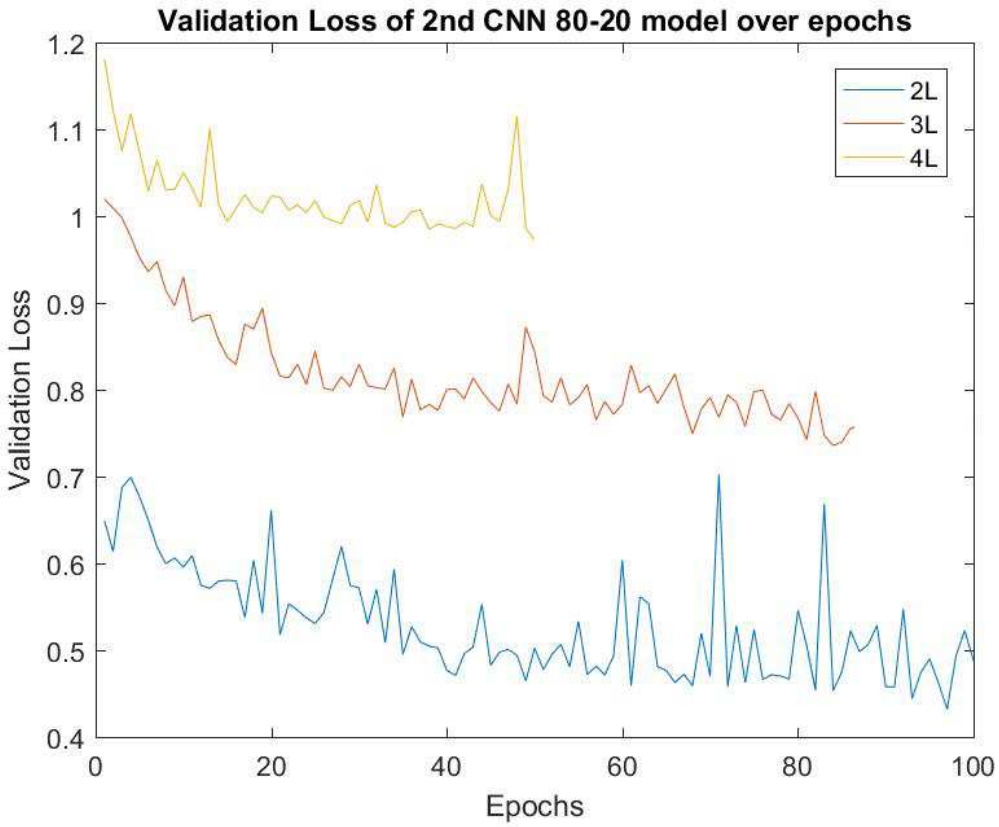


**Figure 70.** Validation loss of 2nd CNN 80-20 model over epochs.

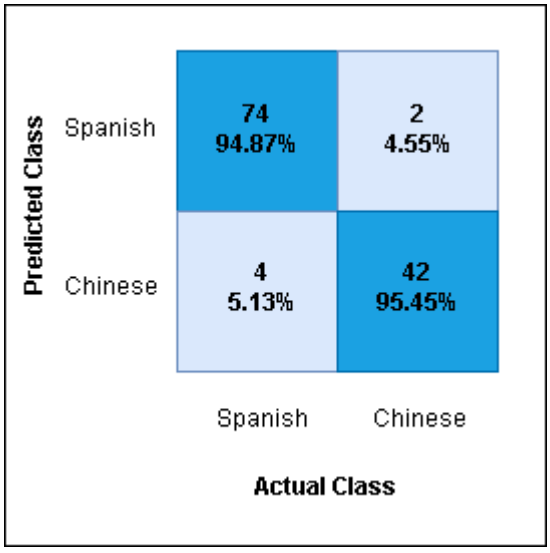


**Figure 71.** Confusion matrix of 2nd CNN 80-20 model for 2 languages.

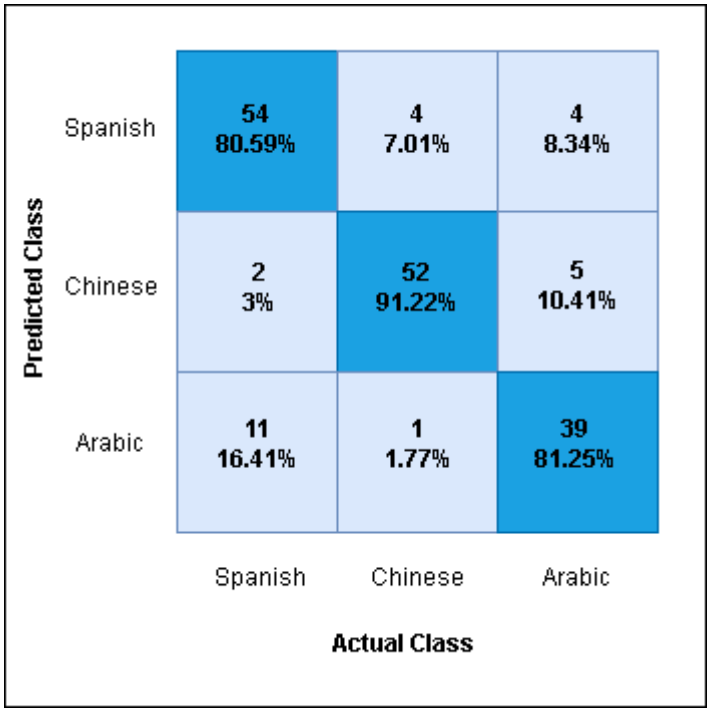


**Figure 72.** Confusion matrix of 2nd CNN 80-20 model for 3 languages.

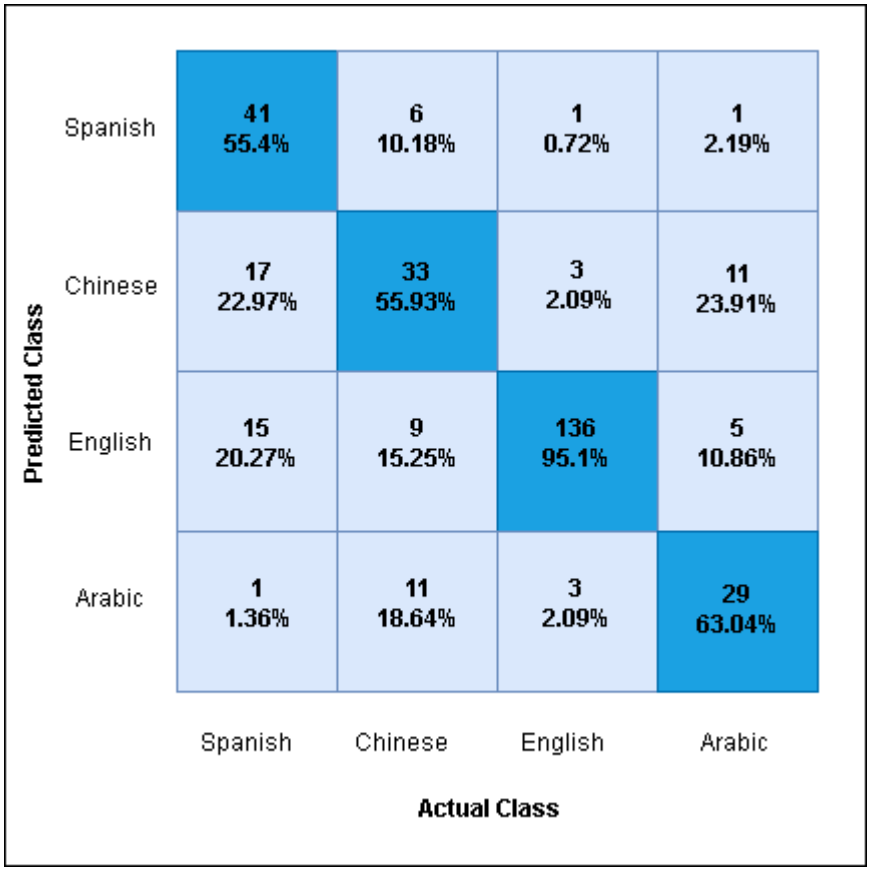


**Figure 73.** Confusion matrix of 2nd CNN 80-20 model for 4 languages.

In terms of 70% training samples and 30% test samples the model scored the best accuracy of 92.85% after 77 epochs for 2 languages. For 3 languages the best accuracy can be seen at the 85th epoch with 83.65%. The model achieved accuracy of 70.39% after 41 epochs for 4 languages.

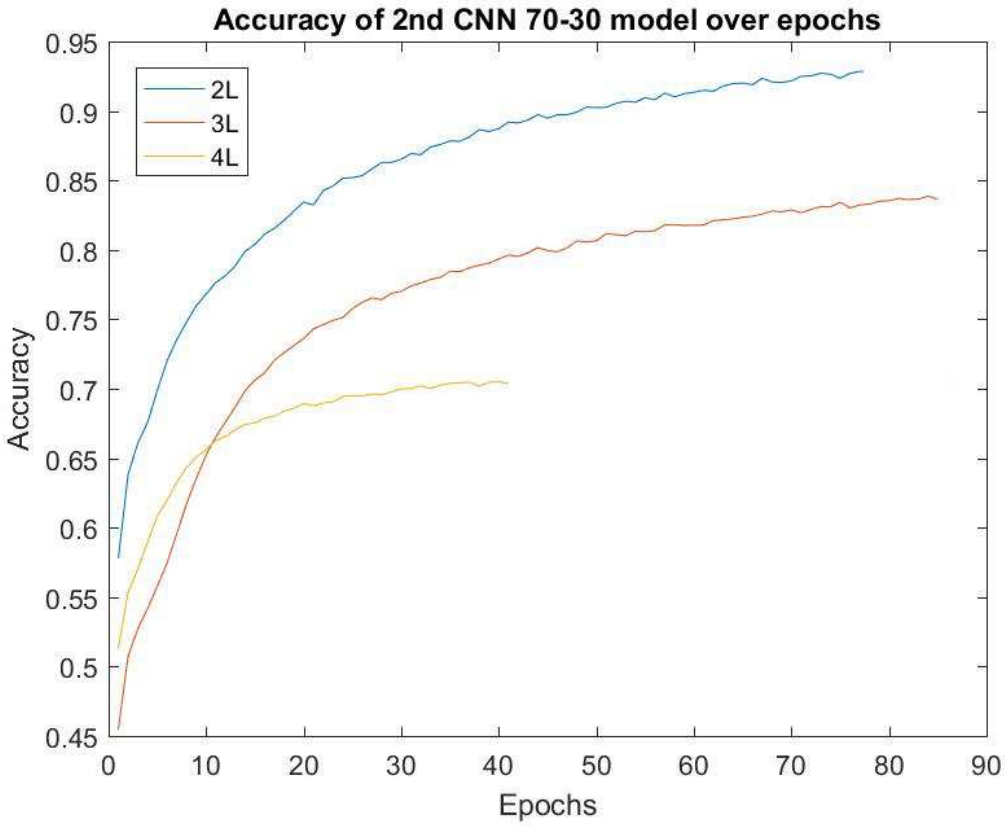


**Figure 74.** Accuracy of 2nd CNN 70-30 model over epochs.

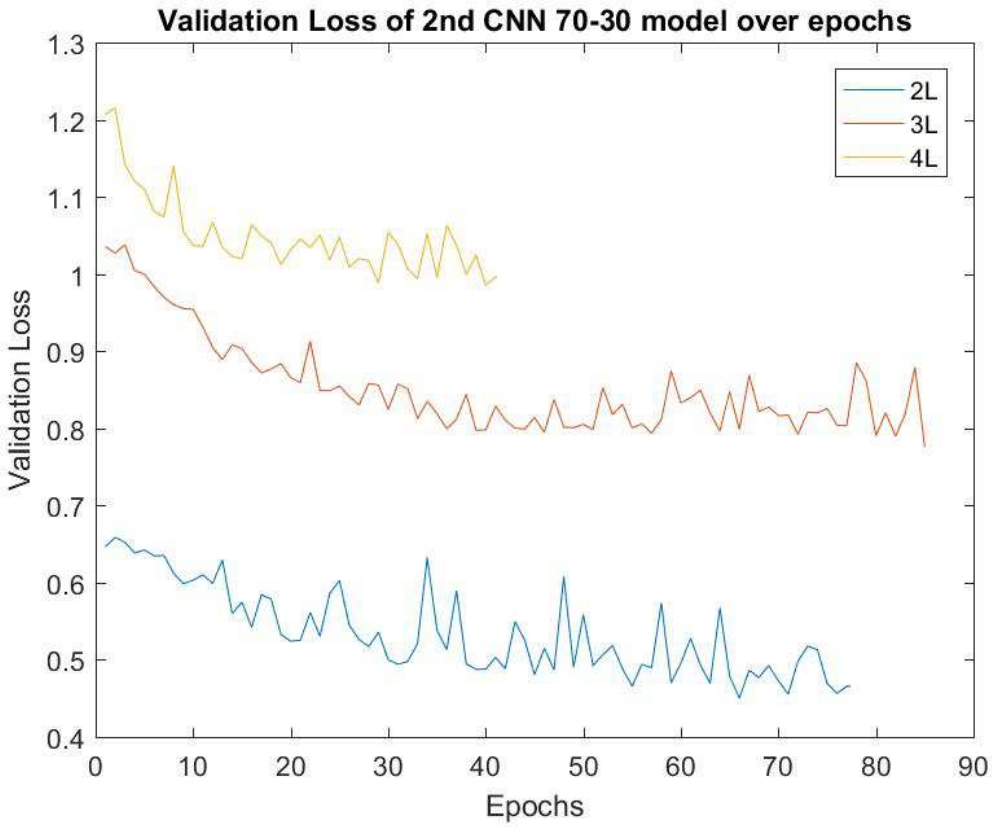


**Figure 75.** Validation loss of 2nd CNN 70-30 model over epochs.

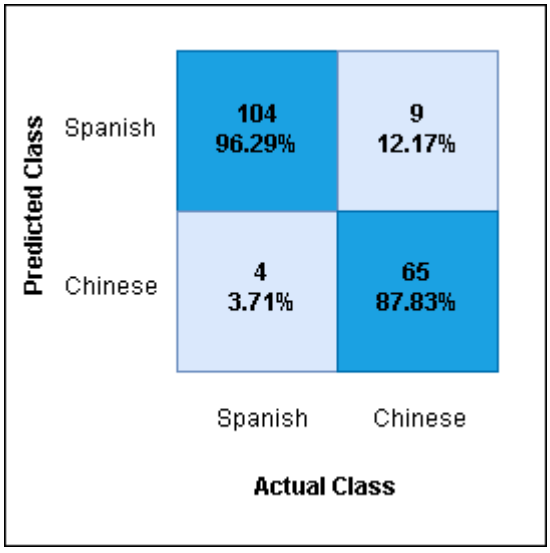


**Figure 76.** Confusion matrix of 2nd CNN 70-30 model for 2 languages.

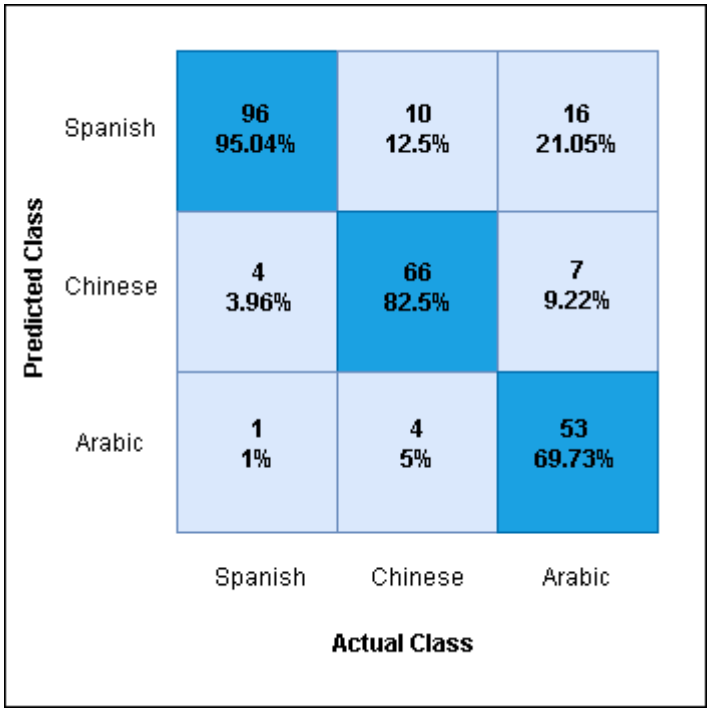


**Figure 77.** Confusion matrix of 2nd CNN 70-30 model for 3 languages.

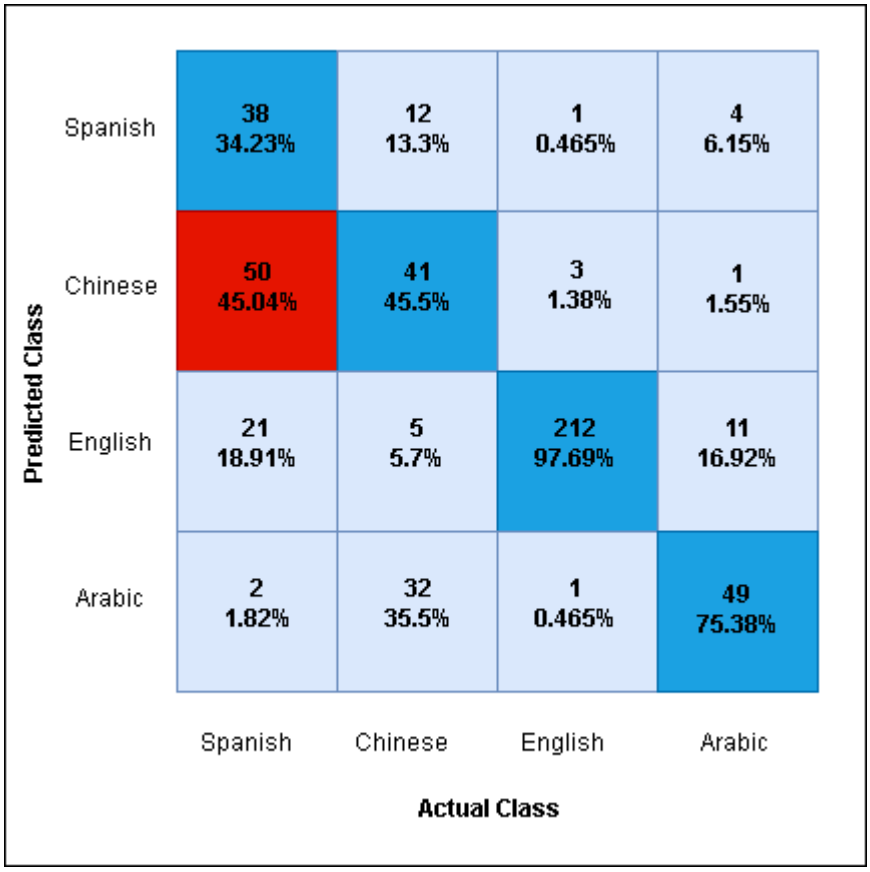


**Figure 78.** Confusion matrix of 2nd CNN 70-30 model for 4 languages.

The case of 60-40 includes 84.77% accuracy at the 74th epoch for 2 languages. In the case of 3 languages the best accuracy is 83.67% after 100 epochs. For 4 languages the accuracy is low at 19.87% after 50 epochs.

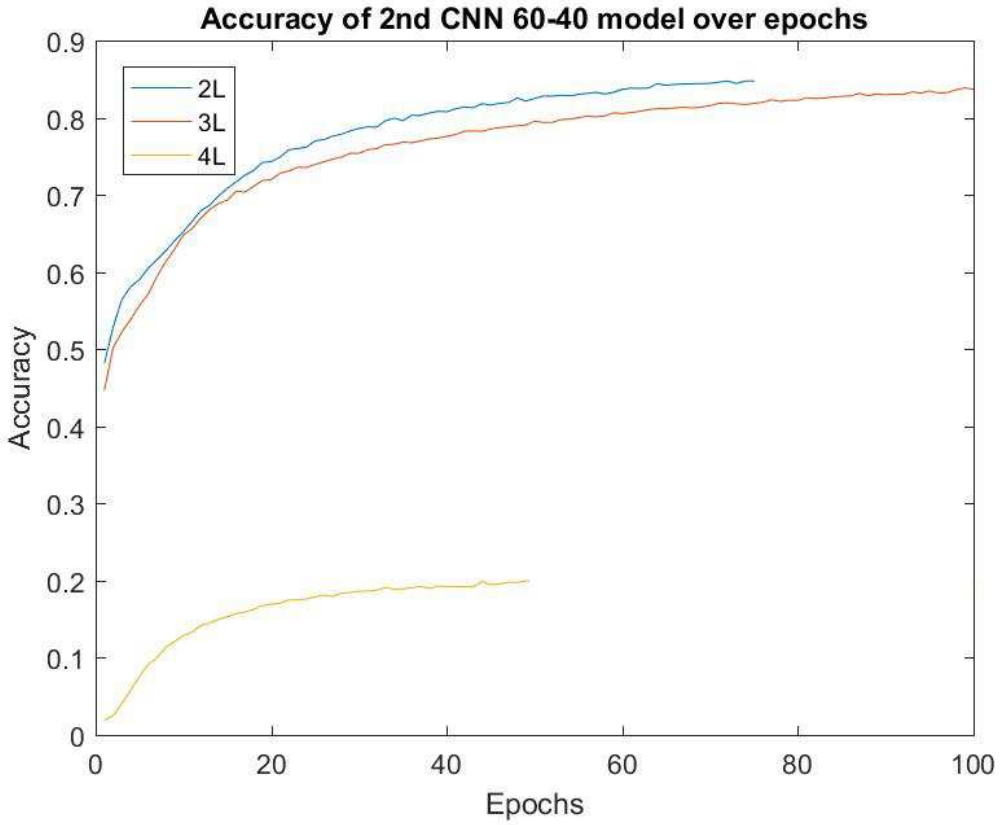


**Figure 79.** Accuracy of 2nd CNN 60-40 model over epochs.

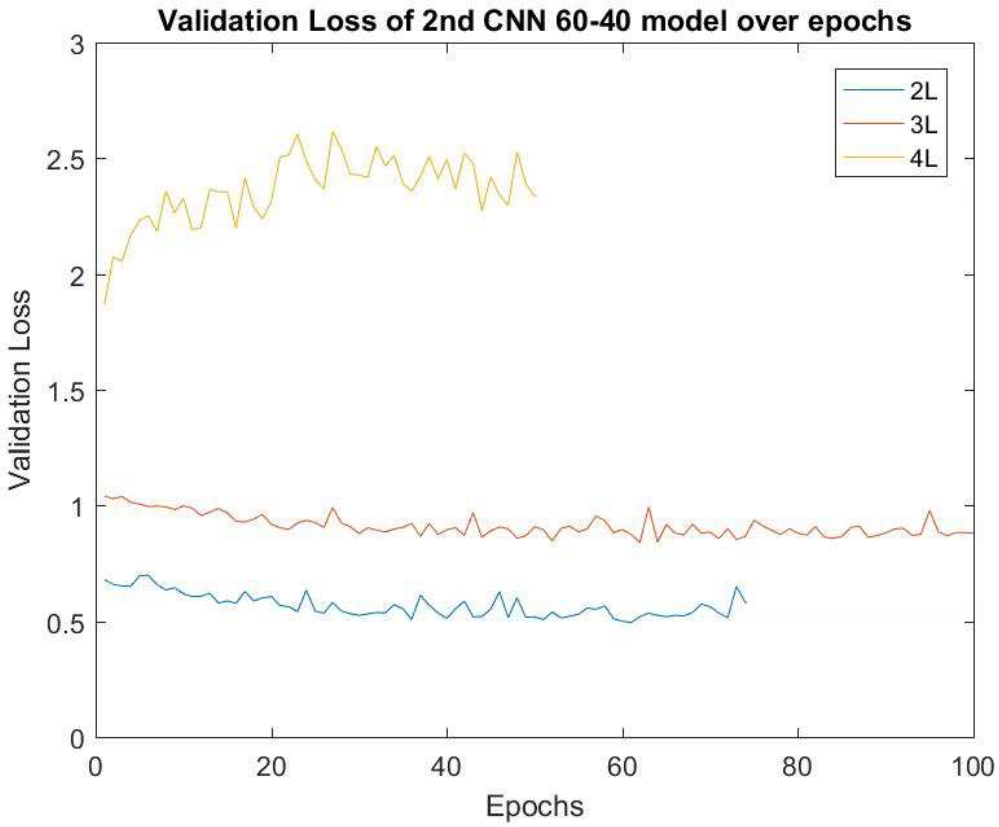


**Figure 80.** Validation loss of 2nd CNN 60-40 model over epochs.

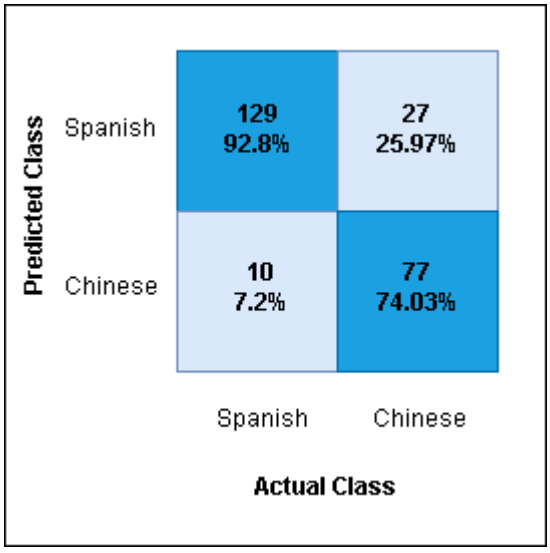


**Figure 81.** Confusion matrix of 2nd CNN 60-40 model for 2 languages.

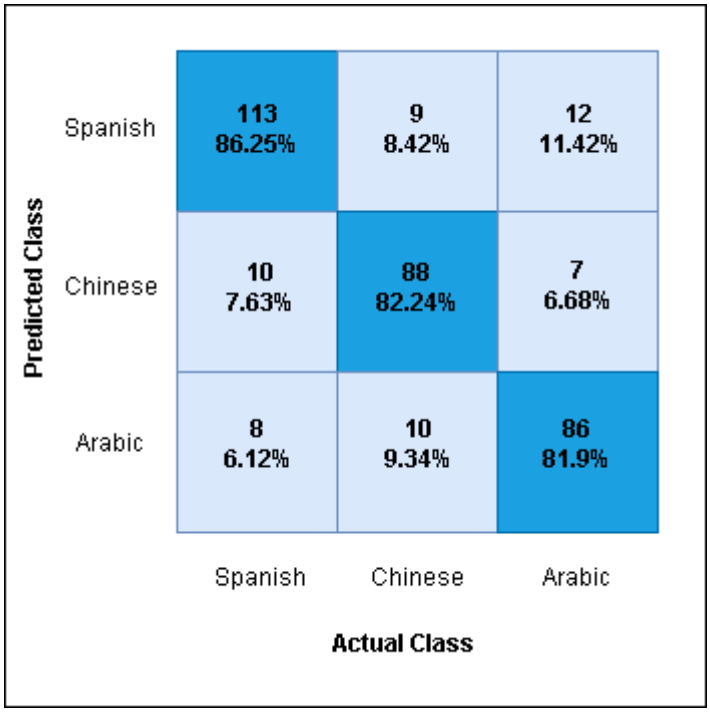


**Figure 82.** Confusion matrix of 2nd CNN 60-40 model for 3 languages.

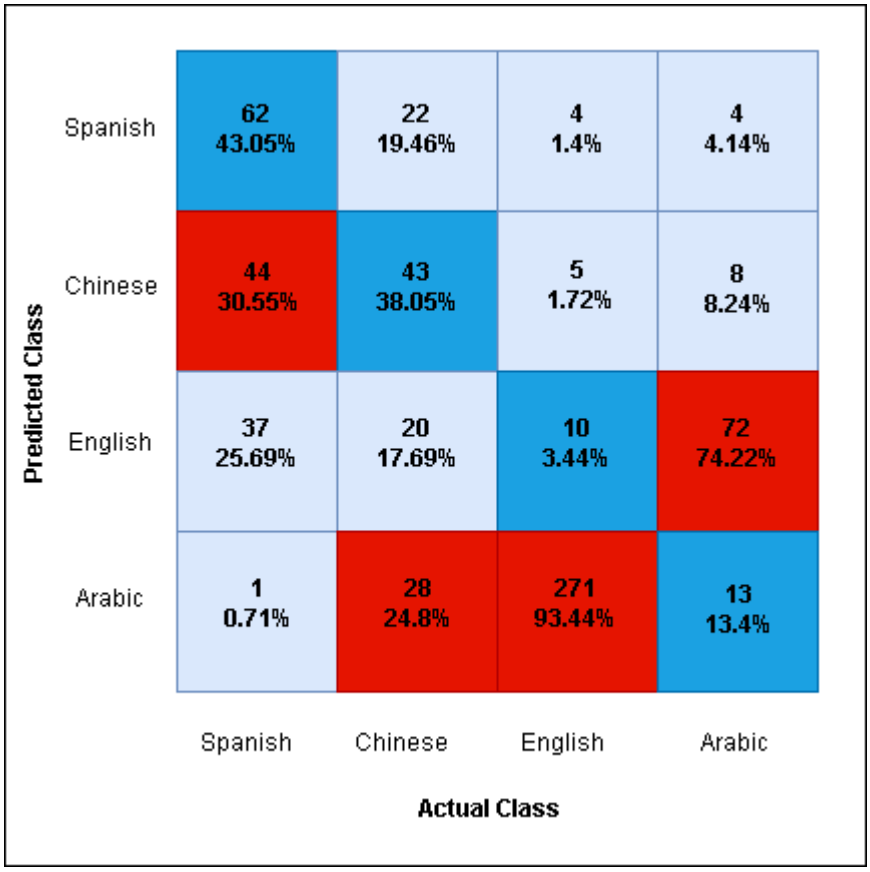


**Figure 83.** Confusion matrix of 2nd CNN 60-40 model for 4 languages.

Continuing to the case of 50-50, the best accuracy of 2 languages is 85.47% after 98 epochs; 73.36% accuracy after 100 epochs for 3 languages and 70.52% at the 50th epoch for 4 languages.

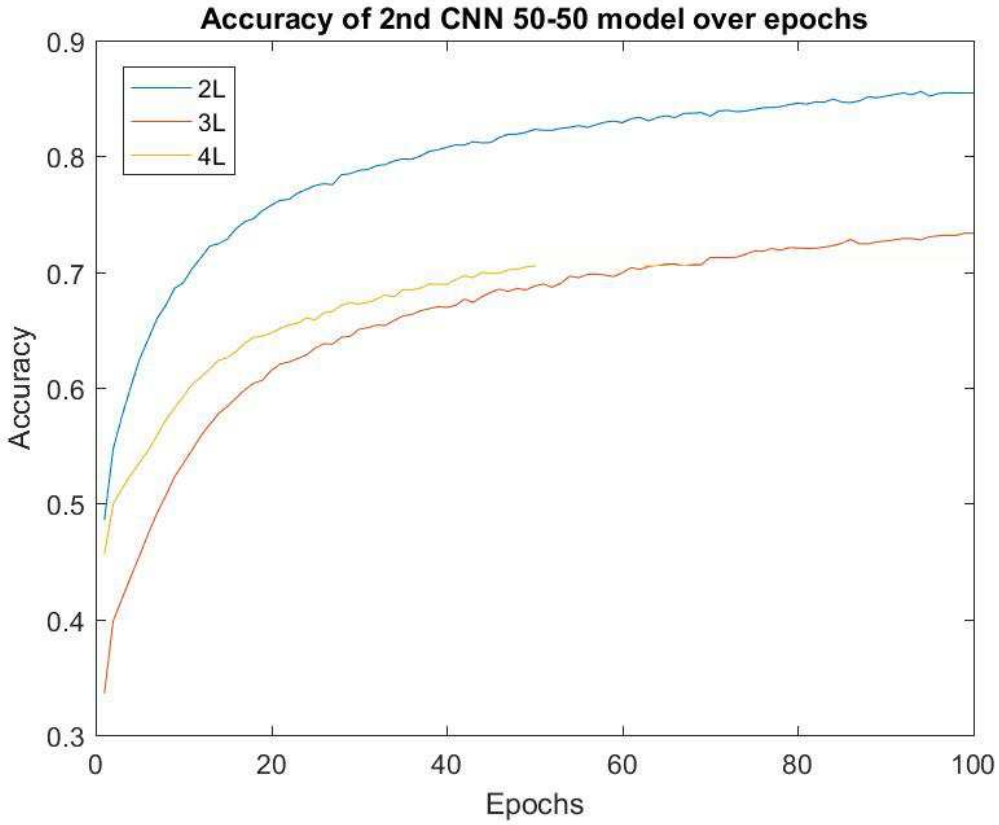


**Figure 84.** Accuracy of 2nd CNN 50-50 model over epochs.

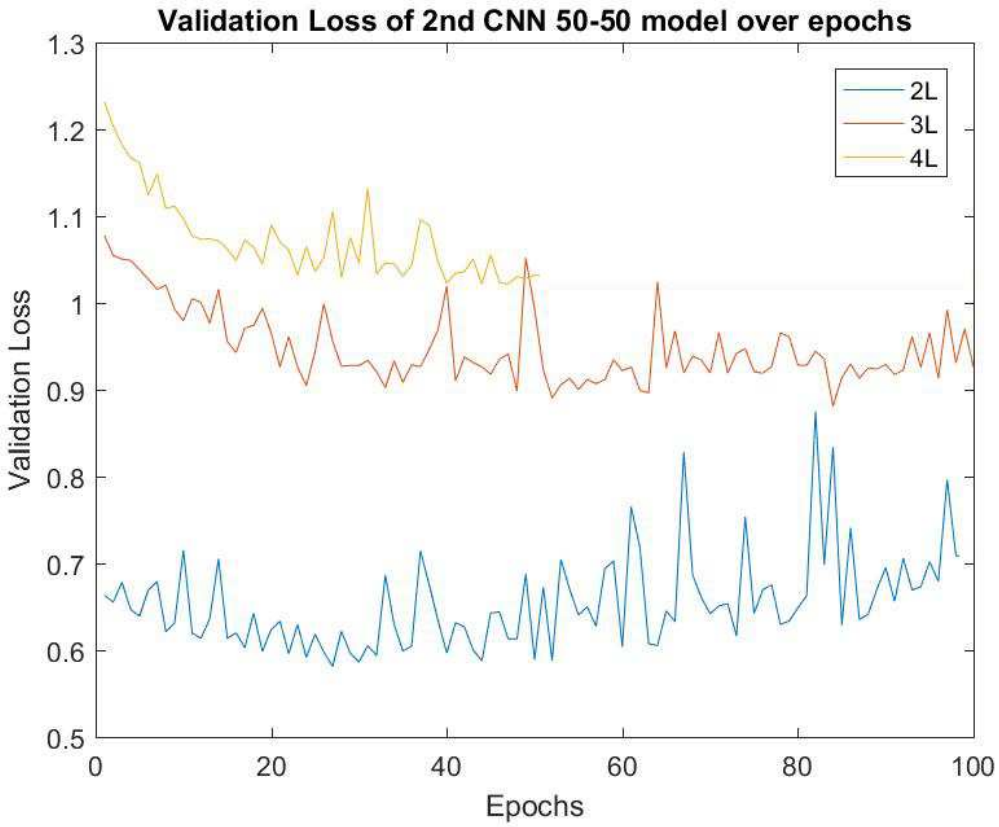


**Figure 85.** Validation loss of 2nd CNN 50-50 model over epochs.

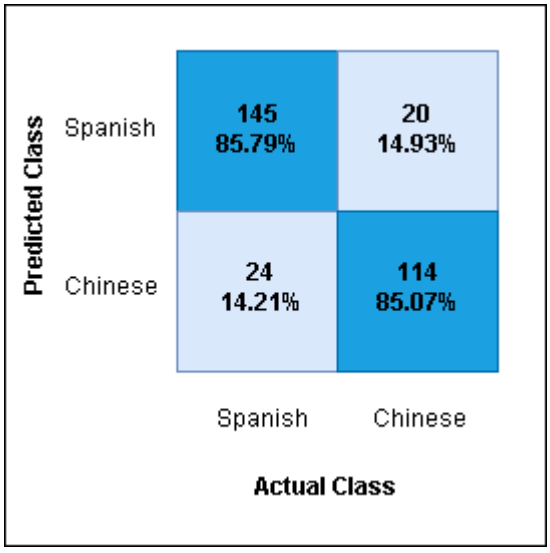


**Figure 86.** Confusion matrix of 2nd CNN 50-50 model for 2 languages.

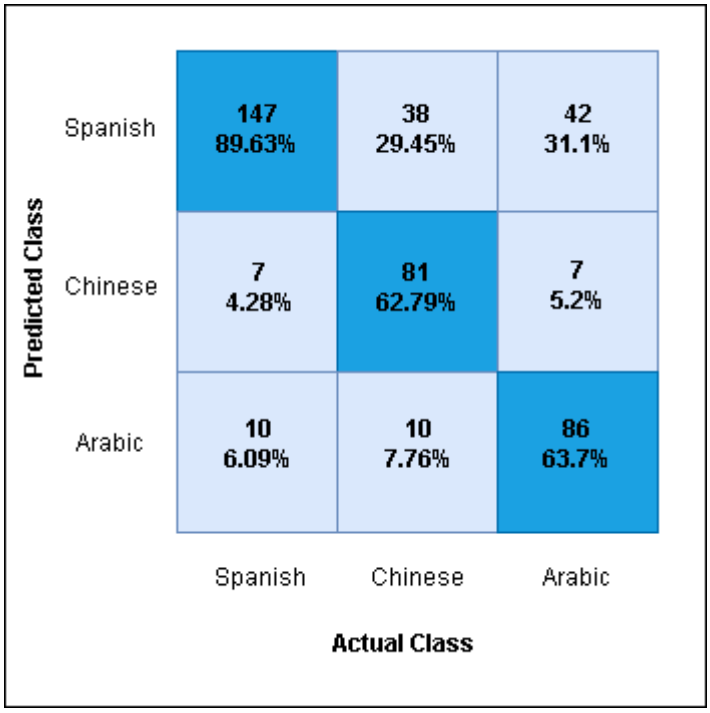


**Figure 87.** Confusion matrix of 2nd CNN 50-50 model for 3 languages.

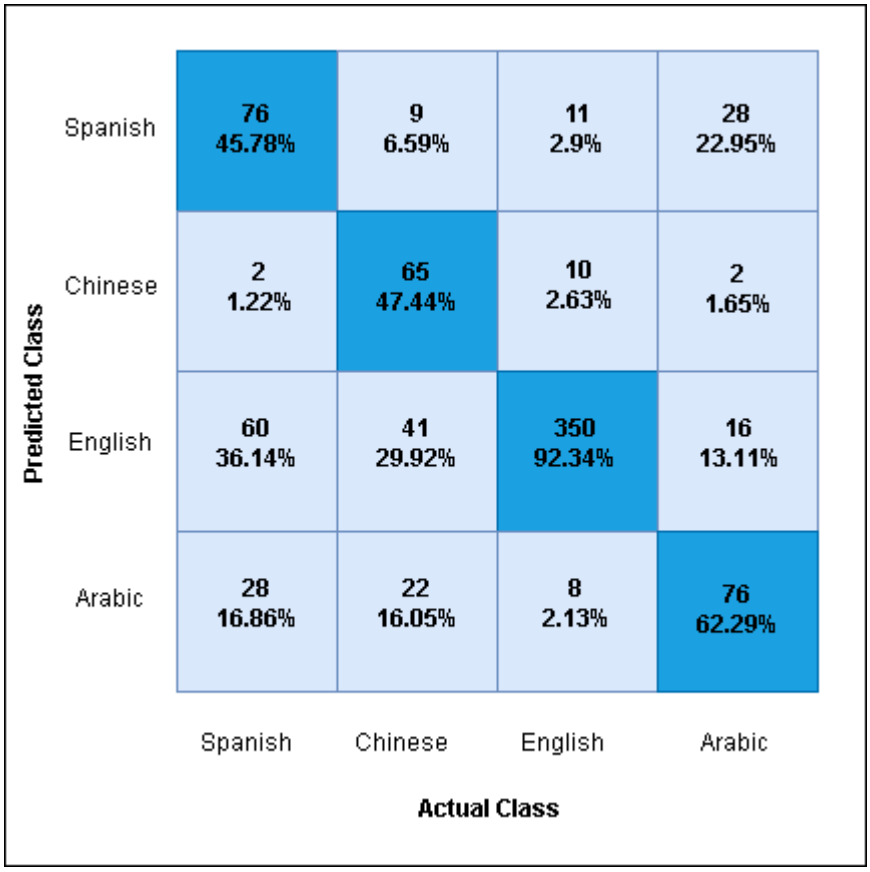


**Figure 88.** Confusion matrix of 2nd CNN 50-50 model for 4 languages.

Concerning the case of 40-60 the best accuracy for 2 languages is 75.27% at the epoch of 98; 70.81% accuracy after 100 epochs for 3 languages and 71.70% accuracy after 50 epochs for 4 languages.

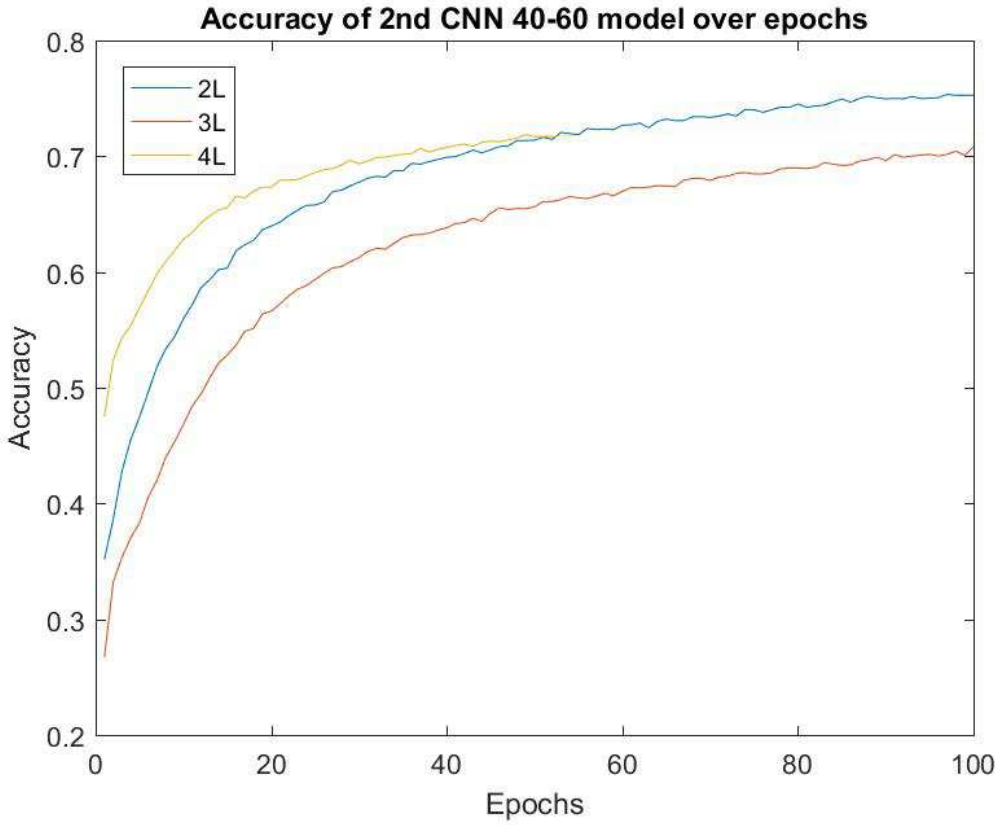


**Figure 89.** Accuracy of 2nd CNN 40-60 model over epochs.

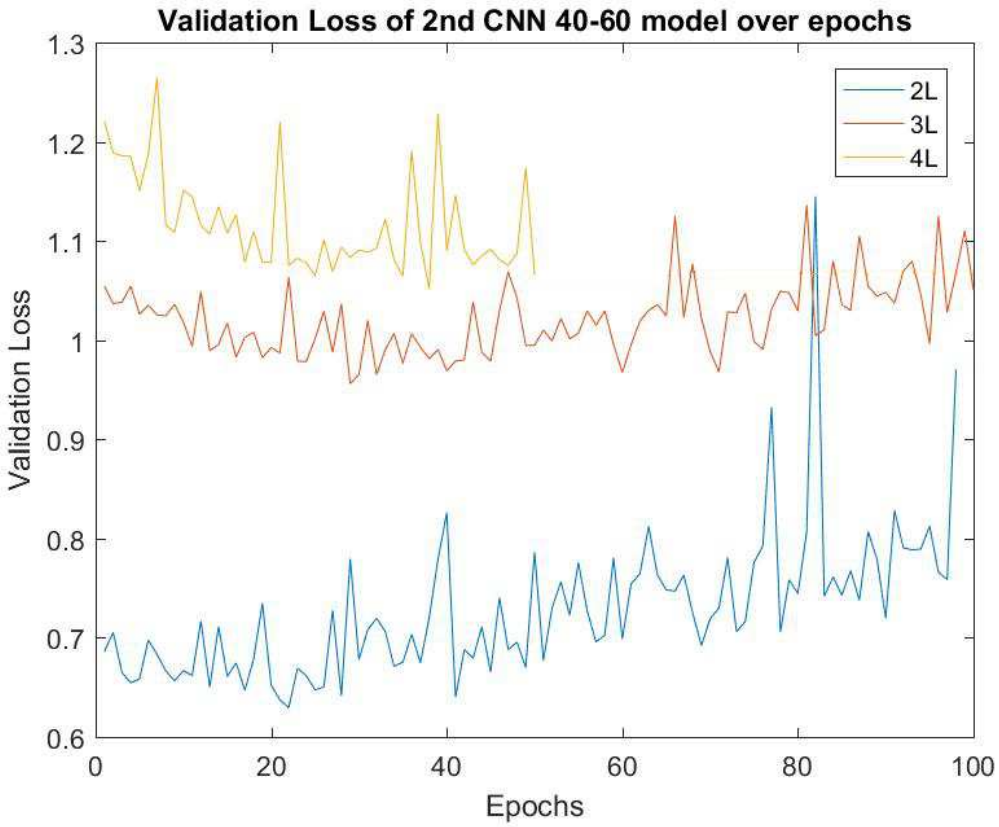


**Figure 90.** Validation loss of 2nd CNN 40-60 model over epochs.

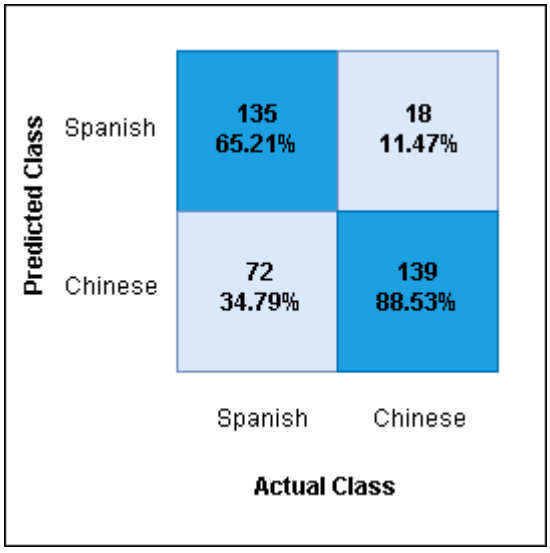


**Figure 91.** Confusion matrix of 2nd CNN 40-60 model for 2 languages.

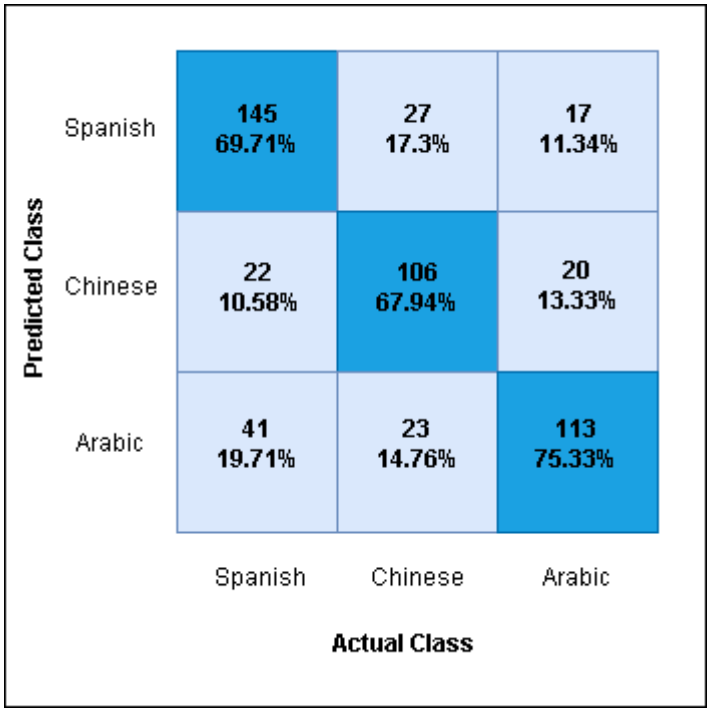


**Figure 92.** Confusion matrix of 2nd CNN 40-60 model for 3 languages.

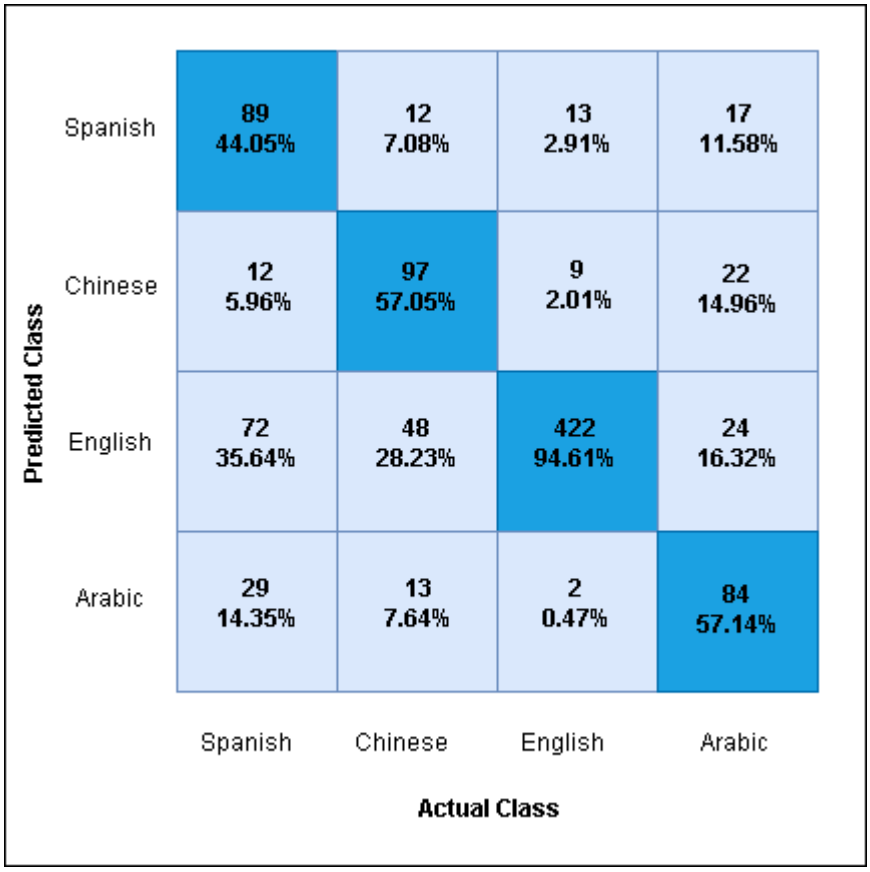


**Figure 93.** Confusion matrix of 2nd CNN 40-60 model for 4 languages.

In the matter of 30-70 the results are: 78.82% accuracy at the 100th epoch for 2 languages; 65.5% accuracy after 83 epochs for 3 languages and 65.18% accuracy at the 50th epoch for 4 languages.

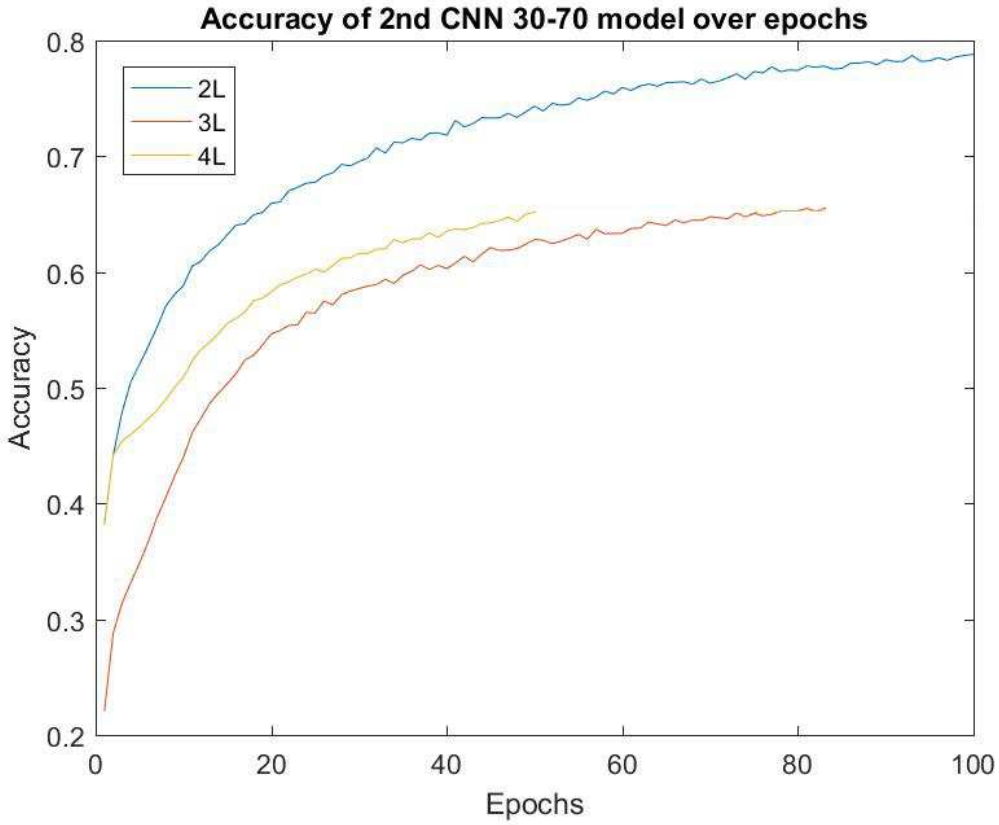


**Figure 94.** Accuracy of 2nd CNN 30-70 model over epochs.

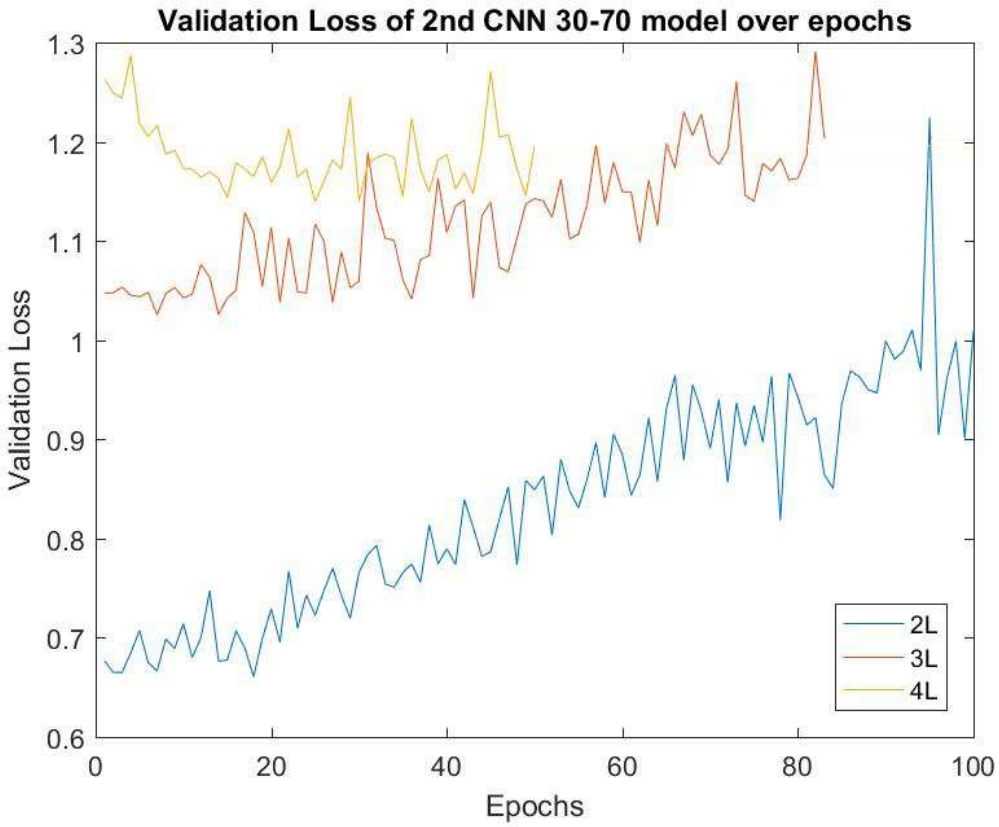


**Figure 95.** Validation loss of 2nd CNN 30-70 model over epochs.

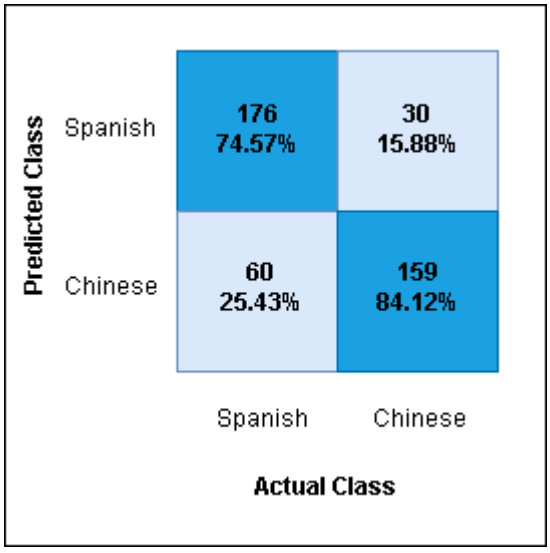


**Figure 96.** Confusion matrix of 2nd CNN 30-70 model for 2 languages.

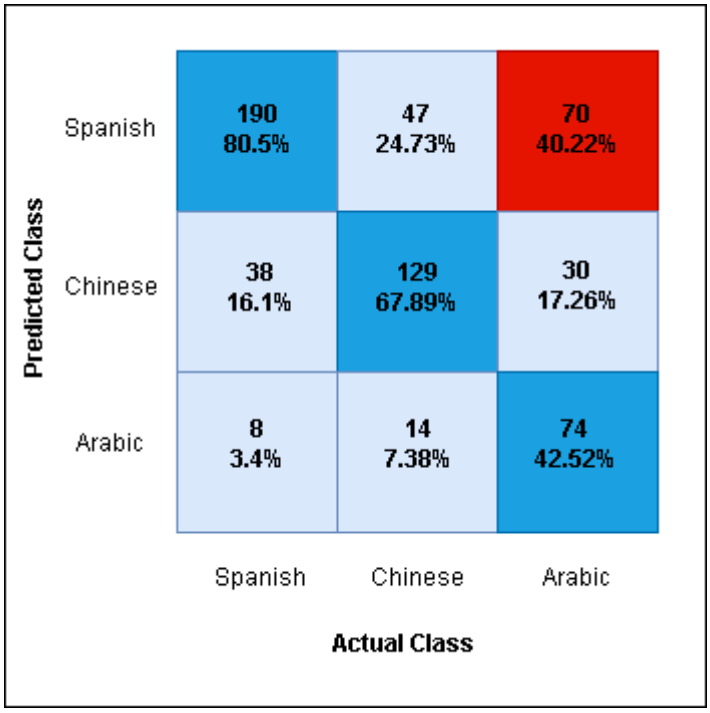


**Figure 97.** Confusion matrix of 2nd CNN 30-70 model for 3 languages.

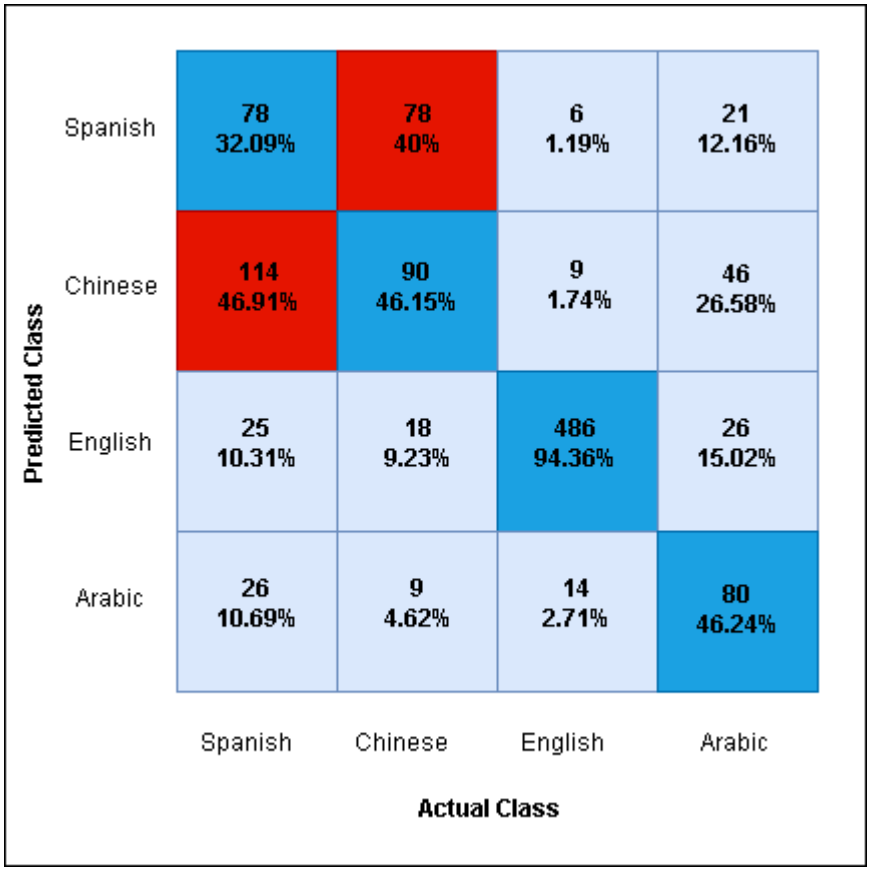


**Figure 98.** Confusion matrix of 2nd CNN 30-70 model for 4 languages.

Regarding 20% training samples and 80% test samples the best accuracy for 2 languages is 74.02% after 93 epochs. For 3 languages the model achieved 60.87% accuracy at the 90th epoch and only 11.03% accuracy after 50 epochs for 4 languages.

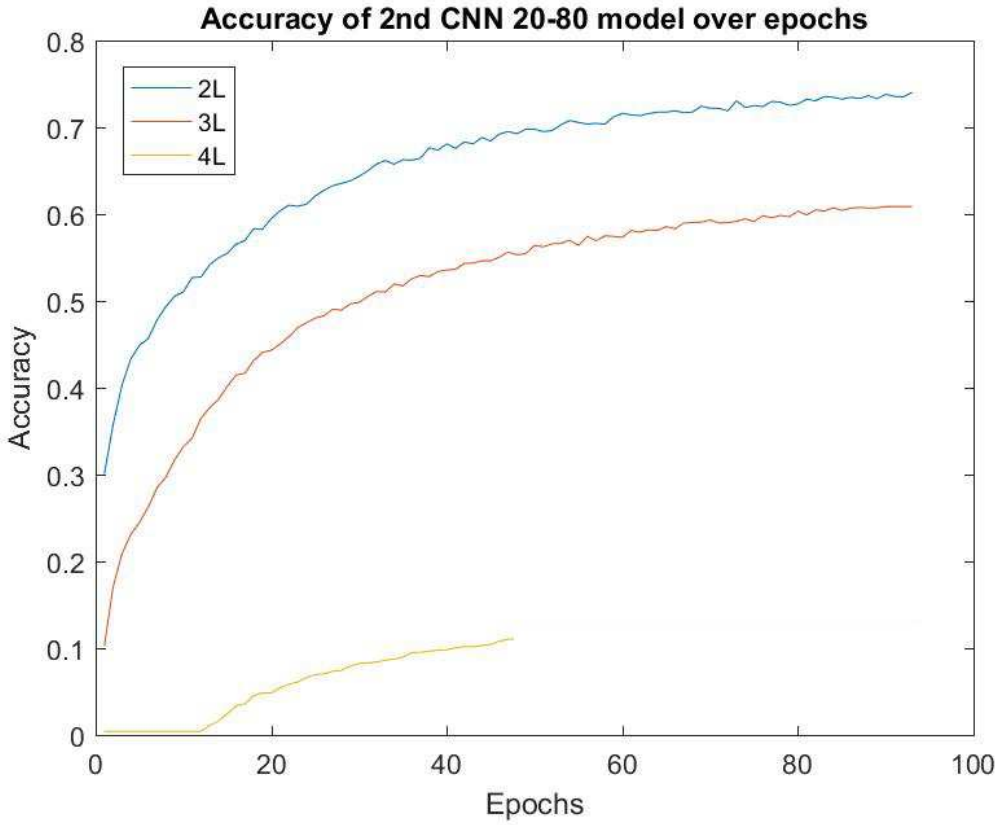


**Figure 99.** Accuracy of 2nd CNN 20-80 model over epochs.

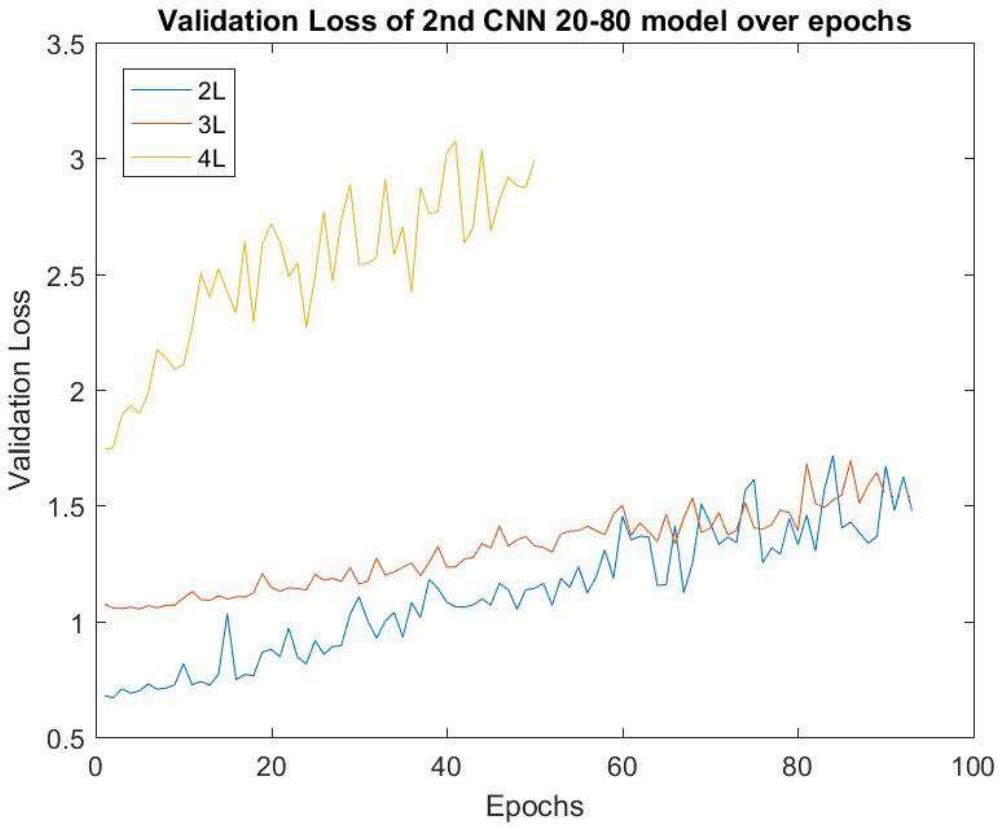


**Figure 100.** Validation loss of 2nd CNN 20-80 model over epochs.

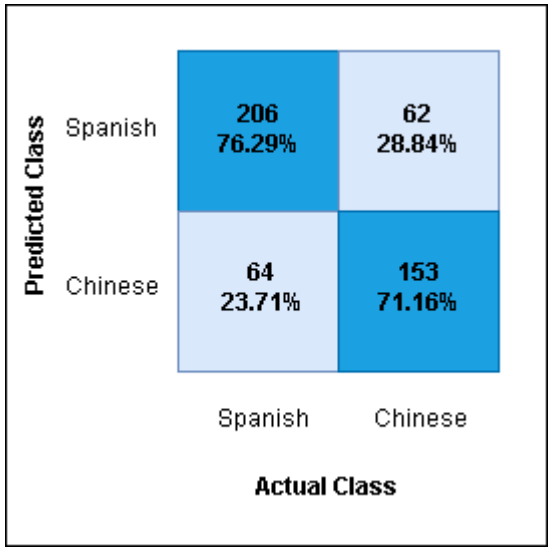


**Figure 101.** Confusion matrix of 2nd CNN 20-80 model for 2 languages.

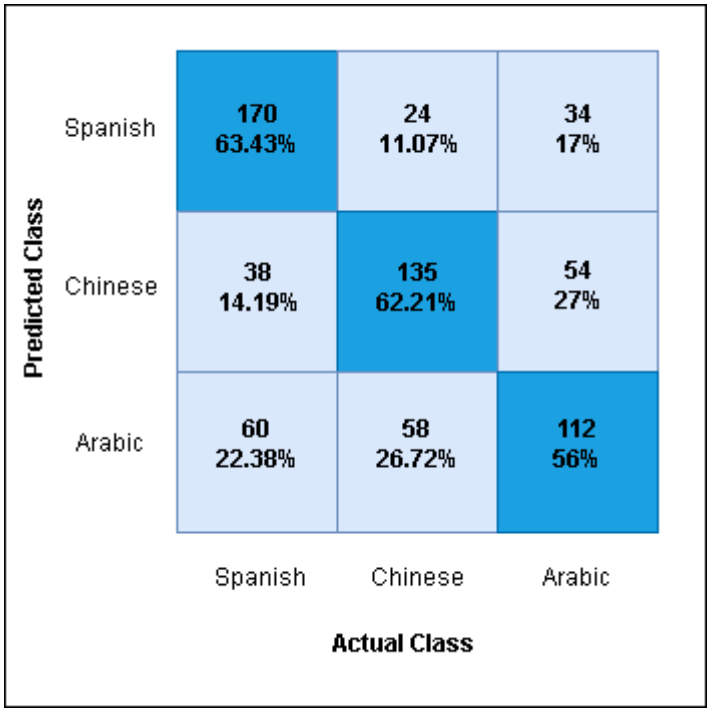


**Figure 102.** Confusion matrix of 2nd CNN 20-80 model for 3 languages.

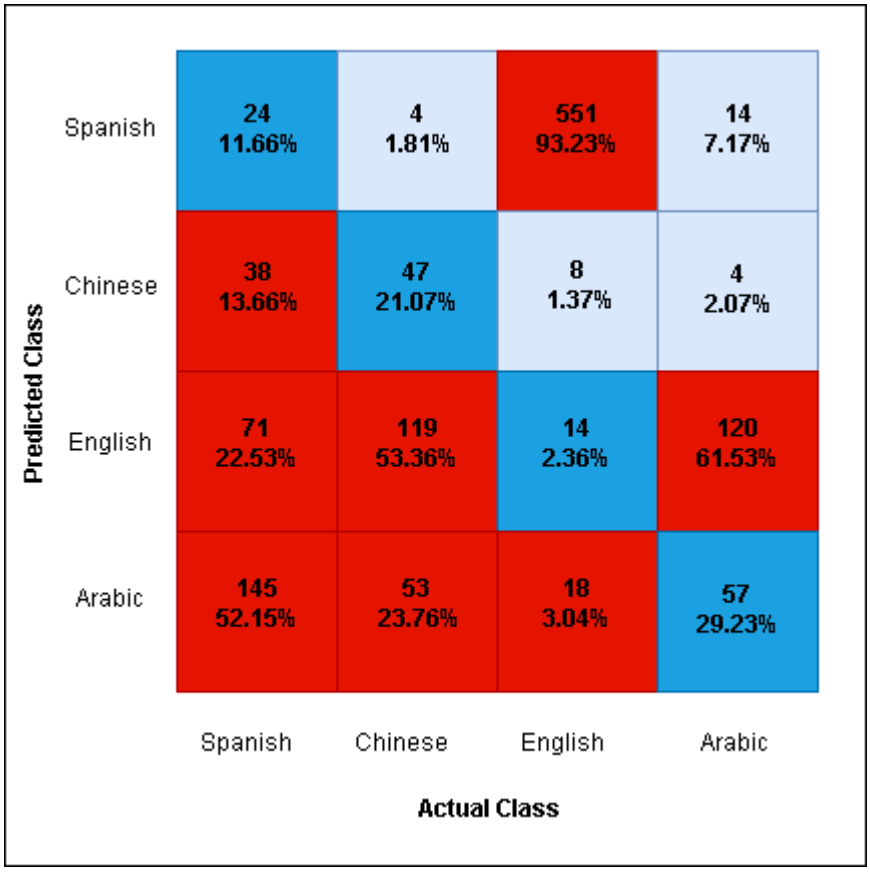


**Figure 103.** Confusion matrix of 2nd CNN 20-80 model for 4 languages.

Lastly, in the case of 10-90 the model scored 30.76% accuracy after 86 epochs for 2 languages, 48.11% after 98 epochs for 3 languages and 56.35% accuracy at the 50th epoch for 4 languages.

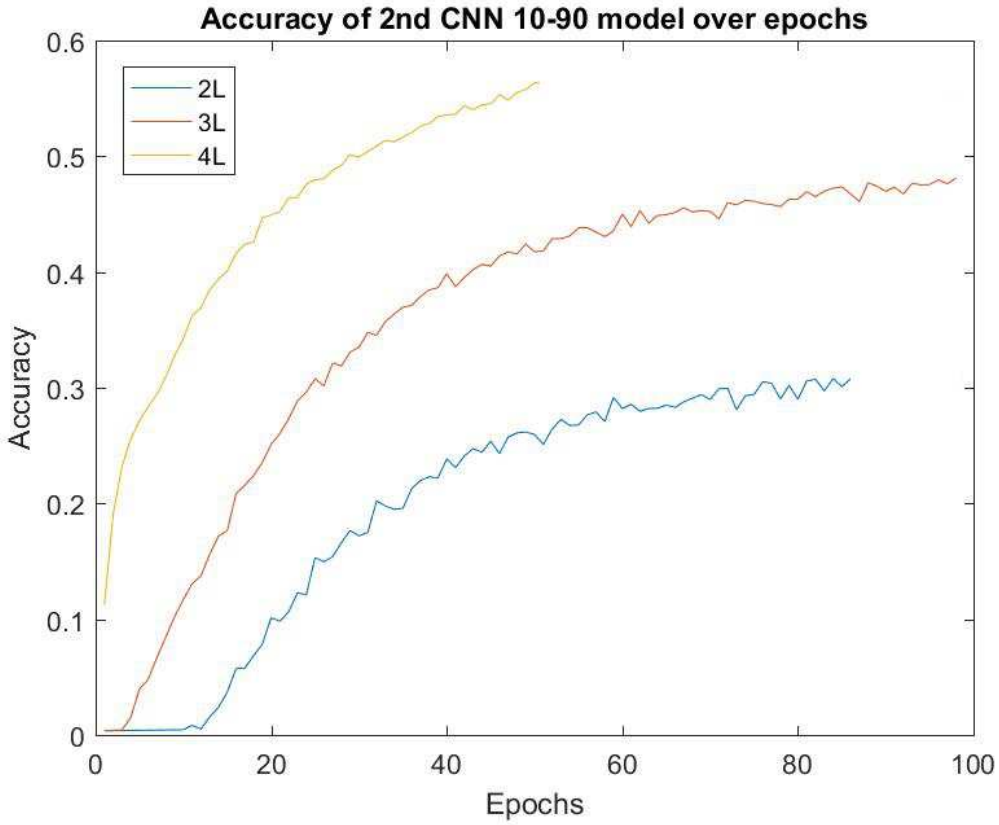


**Figure 104.** Accuracy of 2nd CNN 10-90 model over epochs.

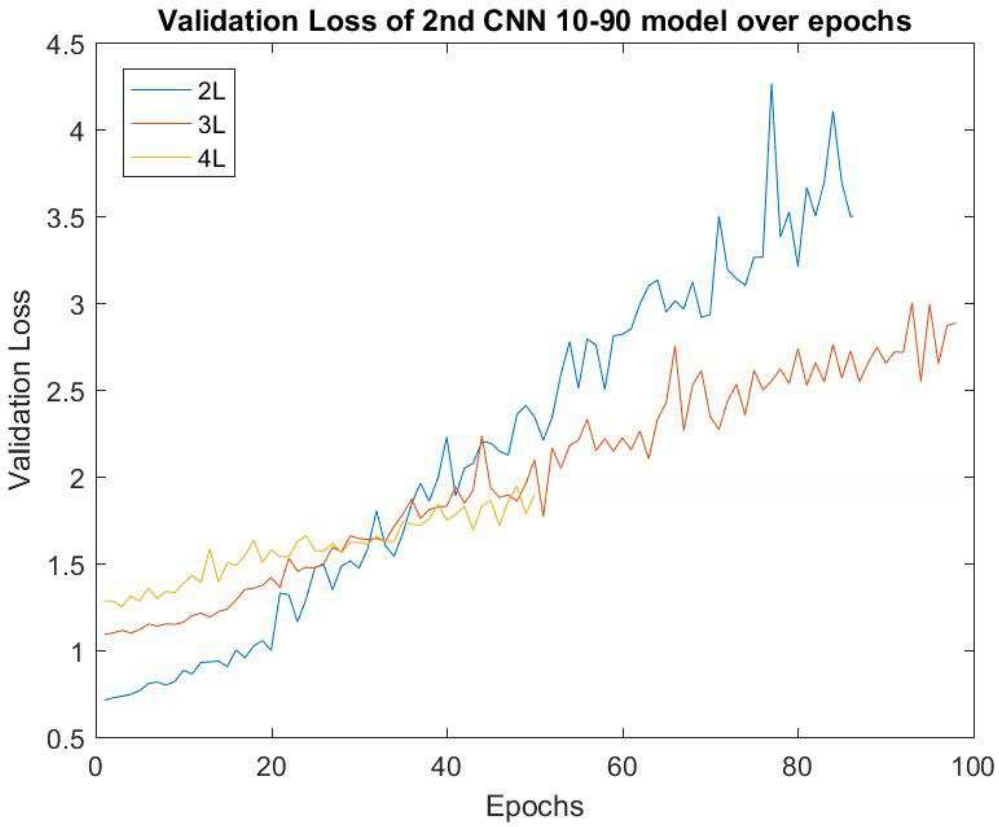


**Figure 105.** Validation loss of 2nd CNN 10-90 model over epochs.

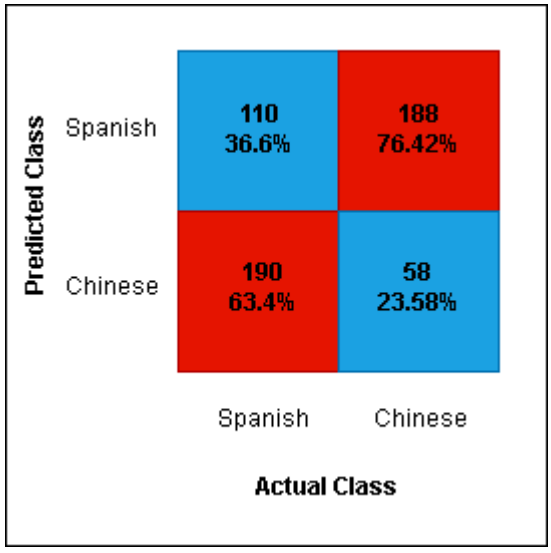


**Figure 106.** Confusion matrix of 2nd CNN 10-90 model for 2 languages.

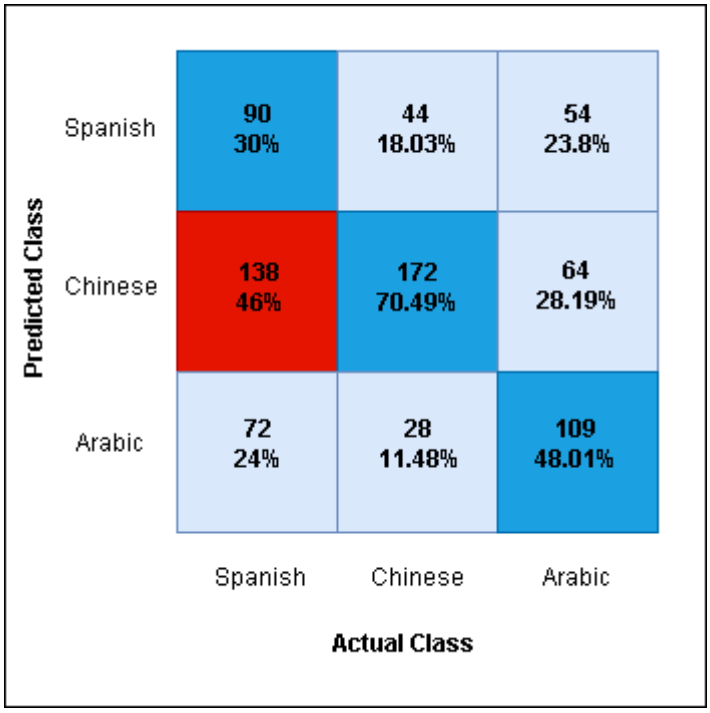


**Figure 107.** Confusion matrix of 2nd CNN 10-90 model for 3 languages.

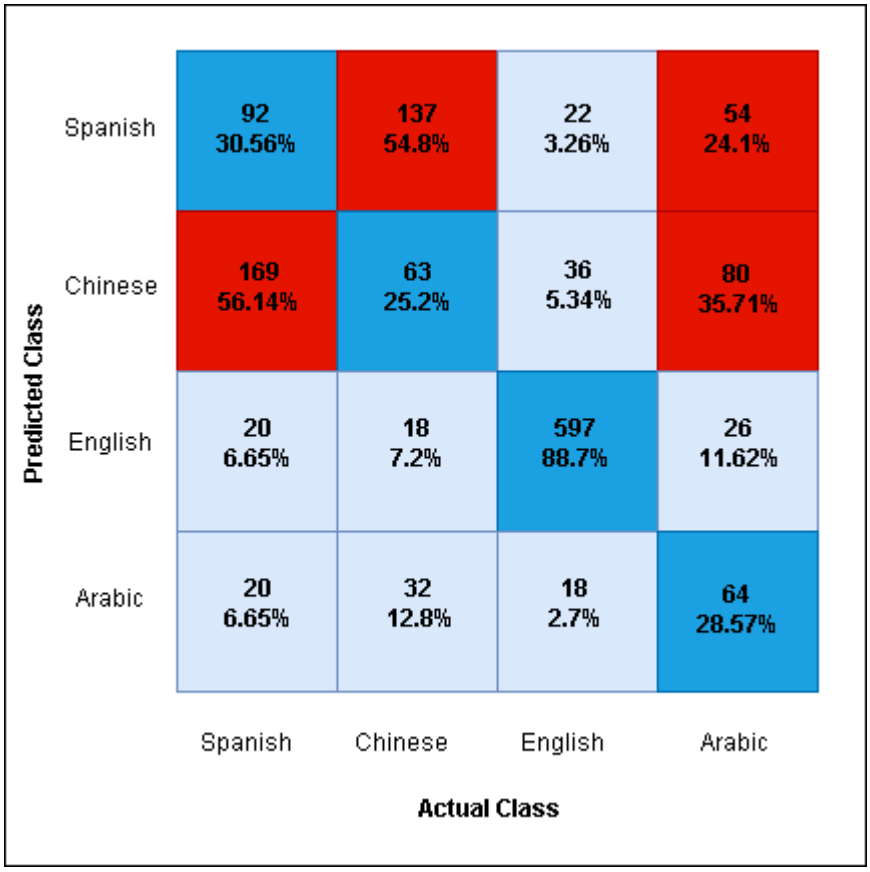


**Figure 108.** Confusion matrix of 2nd CNN 10-90 model for 4 languages.

**Table 4.** Results from the experiments of the second CNN (languages, accuracy, epochs).

| **90-10** | **80-20** | **70-30** |
|---|---|---|
| 2L: 95.08% epochs = 69 | 2L: 95.08% epochs = 100 | 2L: 92.85% epochs = 77 |
| 3L: 81.39% epochs = 100 | 3L: 84.3% epochs = 87 | 3L: 83.65% epochs = 85 |
| 4L: 75.77% epochs = 50 | 4L: 74.22% epochs = 50 | 4L: 70.39% epochs = 41 |

| **60-40** | **50-50** | **40-60** |
|---|---|---|
| 2L: 84.77% epochs = 74 | 2L: 85.47% epochs = 98 | 2L: 75.27% epochs = 98 |
| 3L: 83.67% epochs = 100 | 3L: 73.36% epochs = 100 | 3L: 70.81% epochs = 100 |
| 4L: 19.87% epochs = 50 | 4L: 70.52% epochs = 50 | 4L: 71.70% epochs = 50 |

| **30-70** | **20-80** | **10-90** |
|---|---|---|
| 2L: 78.82% epochs = 100 | 2L: 74.02% epochs = 93 | 2L: 30.76% epochs = 86 |
| 3L: 65.5% epochs = 83 | 3L: 60.87% epochs = 90 | 3L: 48.11% epochs = 98 |
| 4L: 65.18% epochs = 50 | 4L: 11.03% epochs = 50 | 4L: 56.35% epochs = 50 |

Almost similar results can be derived from the second CNN compared to the first one. The best accuracy for 2 languages can be seen in 90-10 and 80-20 cases being 95.08%, which is a little lower than the first CNN (96.72%). The best accuracy for 3 languages was at the case of 80-20 with the value of 84.3%, being a little lower than the first CNN (87.2%). Concerning 4 languages the best accuracy achieved in this model was seen in the case of 90-10 with 75.77%, while in the first CNN the accordingly accuracy was 76.19% at 70-30 category.

Again, the results show that in the case of two languages (Chinese and Spanish) the difference in the English accent is large and can be seen from the first category of 90-10 and even from the 80-20 case. When the user adds more native languages with different accents then the system is not able to achieve extremely high accuracy.

In terms of the statistics of the models running on the current computer it can be noted that the average time for loading the WAV files is almost identical to the first CNN: Specifically the system needed around 8 minutes and 38 second for 2 languages, 12 minutes and 18 seconds for 3 languages and 19 minutes and 44 seconds for 4 languages. The process of converting the WAV files to MFCCs is 48 seconds for 2 languages, 56 seconds for 3 languages and 1:39 minutes for 4 languages. The average time needed for training the models is 43 minutes for 2 languages, 1 hour and 30 minutes for 3 languages and 1 hour and 24 minutes for 4 languages. Similarly to the first neural network, the time needed for training a model depends mainly on the number of languages it contains and the number of epochs.

# 6. CONCLUSION AND FUTURE WORK

In conclusion, the process of accent classification using two different convolutional neural networks and the results of the experiments have been presented in this thesis. Specifically, the languages used in the projects were Chinese, Spanish, English and Arabic and the dataset was retrieved by "*The speech accent archive*" of George Mason University Department of English Speech Accent Archive. The audio files with the speakers contain a certain elicitation paragraph in English; therefore the proposed system is text dependent.

Moreover, the role of machine learning and the feature extraction using the Mel-Frequency Cepstral Coefficients have been discussed. The MFCCs proved to be the most accurate way to represent the energy of a human voice and to capture the characteristics of human tract. It is important to note that the first 13 MFCCs from the audio files were sufficient to be used in the system and the audio files were in WAV format sampled at 16kHz, mono channel, 16-bit.

In addition, the theory of neural networks and convolutional neural networks have been discussed in order for the reader to understand the concepts and the architecture used in the proposed system. The system architecture and the implementation have been presented in detail. Particularly the source code of the system is based on the project of Yatharth Garg and it was modified by the author to suit the purposes of the experiments. The programming language used was Python 3.6 and the importance of using this language was high in terms of extensibility, modularity and being open source. The implementation of the convolutional neural networks in this thesis was based on the open source neural network library of Keras, which runs on top of the TensorFlow framework.

The experiments with the two proposed CNN architectures using ReLU and Sigmoid activation functions and their results were presented and discussed in detail. The diagrams of the accuracy, validation loss and confusion matrices of the experiments have

shown that the proposed system can acquire relatively good results concerning the size of the dataset and the computer used to run the models.

In terms of future work and improvements it would be beneficial to use a larger dataset in order to achieve better results and accuracy of the models. An advantage of the dataset used is that is free and available to the public. Having in mind that in general datasets used for machine learning applications are licensed and expensive to acquire. However, larger datasets are crucial for achieving a more accurate prediction system.

Another aspect of improving the system would be to work independently of a speaker reading a certain text. When a system is text independent, the party using its services is more flexible to predict the origin of the speaker. On the other hand, a text independent system demands a different implementation using GPUs.

Finally, the application of different neural networks, such as recurrent neural networks and quantum neural networks, would provide a better way to solve the accent classification problem. Recurrent neural networks can be already used, but quantum neural networks may take some time to be applied. Future research will determine if recurrent or quantum neural networks will replace convolutional neural networks in accent classification.

REFERENCES

Alpaydin, Ethem (2014). *Introduction to Machine Learning*. 3rd Ed. Cambridge, Massachusetts: MIT Press. 613 p. ISBN 978-0-262-02818-9.

Audacity, Free, open source, cross-platform audio software, [cited 14 Dec. 2018]. Available from World Wide Web: <URL: https://www.audacityteam.org/>.

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. 1st Ed. New York: Springer. 738 p. ISBN 978-0387310732.

Bryant, Morgan, Amanda Chow & Sydney Li (2014). *Classification of Accents of English Speakers by Native Language* [online]. Stanford University [cited 14 Dec. 2018]. Available from World Wide Web: <URL: http://cs229.stanford.edu/proj2014/Morgan%20Bryant,%20Amanda%20Chow,%20Sydney%20Li,%20Classification%20of%20Accents%20of%20English%20Speakers%20by%20Native%20Language.pdf>.

Ciresan, Dan C., Ueli Meier, Jonathan Masci, Luca M. Gambardella & Jurgen Schmidhuber (2011). Flexible, high performance convolutional neural networks for image classification. In: *IJCAI'11 Proceedings of The Twenty-Second international joint conference on Artificial Intelligence*, Vol Two, 1237–1242. Barcelona: AAAI Press. Available from World Wide Web: <URL: https://dl.acm.org/citation.cfm?id=2283603>. ISBN 978-1-57735-514-4.

Chu, Albert, Peter Lai & Diana Le (2017). *Accent Classification of Non-Native English Speakers* [online]. [cited 22 Nov. 2018]. Available from World Wide Web: <URL: http://web.stanford.edu/class/cs224s/reports/Albert_Chu.pdf>.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. In: *Mathematics of Control, Signals and Systems* 2:4, 303–314. Springer-Verlag. Available from World Wide Web: <URL: https://doi.org/10.1007/BF02551274>. ISBN 0932-4194.

Deb, Sankha & Uday Dixit (2008). Intelligent Machining: Computational Methods and Optimization. In: *Machining*, 329–358. London: Springer. Available from World Wide Web: <URL: https://doi.org/10.1007/978-1-84800-213-5_12>. ISBN 978-1-84800-212-8.

Elminir, Hamdy K., Mohamed Abu ElSoud & L.M. Abou El-Maged (2012). Evaluation of Different Feature Extraction Techniques for Continuous Speech Recognition. In: *International Journal of Science and Technology*, 2:10, 689–695. Available from World Wide web: <URL: https://pdfs.semanticscholar.org/2d6b/4e5ae033a117a523a32e8cd8fc2c809897de.pdf>. ISSN 2224-3577.

Flach, Peter (2012). *Machine Learning, The Art and Science of Algorithms that Make Sense of Data*. New York: Cambridge University Press. 396 p. ISBN 978-1-107-42222-3.

Garg, Yatharth (2018). *Speech-Accent-Recognition* [online]. [cited 14 Nov. 2018]. Available from World Wide Web: <URL: https://github.com/yatharth1908/Speech-Accent-Recognition>.

Graupe, Daniel (2013). *Principles of Artificial Neural Networks* [online]. 3rd Ed. London: World Scientific [cited 4 Dec. 2018]. Available from Ebook Central <URL: https://ebookcentral-proquest-com.proxy.uwasa.fi/lib/tritonia-ebooks/detail.action?docID=1336559>. ISBN 978-981-4522-73-1.

Haykin, Simon (2004). *Feedforward Neural Networks: an Introduction*. 16 p. Available from World Wide Web: <URL: https://pdfs.semanticscholar.org/39b1/c4bf2409ac4fd21c611f732745329e118e0b.pdf>.

Haykin, Simon (1999). *Neural Networks: A Comprehensive Foundation*. 2nd Ed. Upper Saddle River, New Jersey: Prentice Hall. 842 p. ISBN 978-0132733502.

Hellerstein, Joe (2008). *Parallel Programming in the Age of Big Data* [online]. 9 November 2008 [cited 19 Nov. 2018]. Available from World Wide Web: <URL: https://gigaom.com/2008/11/09/mapreduce-leads-the-way-for-parallel-programming/>.

Huang, Xuedong, Alex Acero & Hsiao-Wuen Hon (2001). *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*. 1st Ed. Upper Saddle River, New Jersey: Prentice Hall. 380 p. ISBN 978-130-226167.

Khan, Salman, Hossein Rahmani, Syed Afaq Ali Shah & Mohammed Bennamoun (2018). *A Guide to Convolutional Neural Networks for Computer Vision*. Morgan & Claypool. 184 p. ISBN 978-1681730219.

Lokhande, N.N., N.S. Nehe & P.S. Vihke (2012). MFCC based robust features for English Word Recognition. In: *2012 Annual IEEE India Conference (INDICON)*, 798–801. IEEE. Available from IEEE Xplore: <URL: https://doi.org/10.1109/INDCON.2012.6420726>. ISBN 978-130-226167.

Ma, Zichen & Ernerst Fokoue (2014). A Comparison of Classifiers in Performing Speaker Accent Recognition Using MFCCs. In: *Open Journal of Statistics*, 258–266. Rochester: Rochester Institute of Technology. Available from World Wide Web: <URL: https://arxiv.org/ftp/arxiv/papers/1501/1501.07866.pdf>.

PyCharm, The Python IDE for Professional Developers by JetBrains, [cited 14 Dec. 2018]. Available from World Wide Web: <URL: https://www.jetbrains.com/pycharm/>.

Najafian, Maryam, Saeid Safavi, Abualsoud Hanani & Martin Russell (2014). Acoustic model selection using limited data for accent robust speech recognition. In: *2014 22nd European Signal Processing Conference (EUSIPCO)*, 1786-1790. IEEE. Lisbon, Portugal. Available from IEEE Xplore: <URL: https://ieeexplore.ieee.org/document/6952657>. ISBN 978-0-9928-6261-9.

Rogers, Simon & Mark Girolami (2017). *A First Course in Machine Learning*. 2nd Ed. Boca Raton: CRC Press. 397 p. ISBN 978-1-4987-38484.

Rumelhart, D.E., G.E. Hinton & R.J. Williams (1986). Learning internal representations by error propagation. In: *Parallel distributed processing: explorations in the microstructure of cognition*, vol. 1, 318–362. Cambridge, MA: MIT Press. Available from World Wide Web: <URL: http://dl.acm.org/citation.cfm?id=104279.104293>. ISBN 0-262-68053-X.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever & Ruslan Salakhutdivon (2014). Dropout: a simple way to prevent neural networks from overfitting. In: *The Journal of Machine Learning Research*, 15:1, 1929–1958. Available from World Wide Web: <URL: https://dl.acm.org/citation.cfm?id=2670313>. ISSN 1532-4435.

Valaki, Sanjay A. & Harikrishna B. Jethva (2016). A Survey on Feature Extraction and Classification Techniques for Speech Recognition. In: *International Journal of Advance Research and Innovative Ideas In Education*, 2:6, 830–837. Available from World Wide web: <URL: http://ijariie.com/AdminUploadPdf/A_Survey_on_Feature_Extraction_and_Classification_Techniques_for_Speech_Recognition_ijariie3432.pdf>. ISSN 2395-4396.

Venkatesan, Ragav & Baoxin Li (2018). *Convolutional Neural Networks in Visual Computing: A Concise Guide*. Phoenix: CRC Press. 168 p. ISBN 978-1-4987-7039-2.

Watanaprakornkul, Phumchanit, Chantat Eksombatchai & Peter Chien (2010). *Accent Classification* [online]. Stanford University [cited 14 Dec. 2018]. Available from World Wide Web: <URL: http://cs229.stanford.edu/proj2010/WatanaprakornkulEksombatchaiChien-AccentClassification.pdf>.

Weinberger, Steven (2015). *Speech Accent Archive* [online]. George Mason University [cited 7 Dec. 2018]. Available from World Wide Web <URL: http://accent.gmu.edu>.

Werbos, Paul J. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. Harvard University, Cambridge, MA. Available from World Wide Web: <URL: https://www.researchgate.net/profile/Paul_Werbos/publication/35657389_Beyond_regres-sion_new_tools_for_prediction_and_analysis_in_the_behavioral_sciences/links/576ac78508aef2a864d20964/Beyond-regression-new-tools-for-prediction-and-analysis-in-the-behavioral-sciences.pdf>.

Winamp, [cited 14 Dec. 2018]. Available from World Wide Web: <URL: https://www.winamp.com/>.

APPENDIX 1. SOURCE CODE

```
#-----------------------------------------------------------------------
# trainmodel.py
#-----------------------------------------------------------------------
# original code taken from:
# Garg, Yatharth (2018). Speech-Accent-Recognition [online].
# [cited 14 Nov. 2018].
# Available from World Wide Web:
# <URL: https://github.com/yatharth1908/Speech-Accent-Recognition>.
# modified by Stavros Grigoriadis
#-----------------------------------------------------------------------

import pandas as pd
from collections import Counter
import sys
sys.path.append('../speech-accent-recognition/src>')
import getsplit
import time
import datetime

from keras import utils
import accuracy
import multiprocessing
import librosa
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Flatten
from keras.layers.convolutional import MaxPooling2D, Conv2D
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import EarlyStopping, TensorBoard

import winsound


DEBUG = True
SILENCE_THRESHOLD = .01
RATE = 16000
N_MFCC = 13
COL_SIZE = 30
EPOCHS = 100


def to_categorical(y):
    '''
    Converts list of languages into a binary class matrix
```

```
    :param y (list): list of languages
    :return (numpy array): binary class matrix
    '''
    lang_dict = {}
    for index,language in enumerate(set(y)):
        lang_dict[language] = index
    y = list(map(lambda x: lang_dict[x],y))
    return utils.to_categorical(y, len(lang_dict))


def get_wav(language_num):
    '''
    Load wav file from disk and down-samples to RATE
    :param language_num (list): list of file names
    :return (numpy array): Down-sampled wav file
    '''

    y, sr = librosa.load('../audio/{}.wav'.format(language_num))
    return(librosa.core.resample(y=y,orig_sr=sr,target_sr=RATE,
scale=True))


def to_mfcc(wav):
    '''
    Converts wav file to Mel Frequency Ceptral Coefficients
    :param wav (numpy array): Wav form
    :return (2d numpy array: MFCC
    '''
    return(librosa.feature.mfcc(y=wav, sr=RATE, n_mfcc=N_MFCC))


def remove_silence(wav, thresh=0.04, chunk=5000):
    '''
    Searches wav form for segments of silence. If wav form values are
lower than 'thresh' for 'chunk' samples, the values will be removed
    :param wav (np array): Wav array to be filtered
    :return (np array): Wav array with silence removed
    '''

    tf_list = []
    for x in range(len(wav) / chunk):
        if (np.any(wav[chunk * x:chunk * (x + 1)] >= thresh) or
np.any(wav[chunk * x:chunk * (x + 1)] <= -thresh)):
            tf_list.extend([True] * chunk)
        else:
            tf_list.extend([False] * chunk)

    tf_list.extend((len(wav) - len(tf_list)) * [False])
    return(wav[tf_list])
```

```
def normalize_mfcc(mfcc):
    '''
    Normalize mfcc
    :param mfcc:
    :return:
    '''
    mms = MinMaxScaler()
    return(mms.fit_transform(np.abs(mfcc)))


def make_segments(mfccs,labels):
    '''
    Makes segments of mfccs and attaches them to the labels
    :param mfccs: list of mfccs
    :param labels: list of labels
    :return (tuple): Segments with labels
    '''
    segments = []
    seg_labels = []
    for mfcc,label in zip(mfccs,labels):
        for start in range(0, int(mfcc.shape[1] / COL_SIZE)):
            segments.append(mfcc[:, start * COL_SIZE:(start + 1) *
COL_SIZE])
            seg_labels.append(label)
    return(segments, seg_labels)


def segment_one(mfcc):
    '''
    Creates segments from on mfcc image. If last segments is not long
enough to be length of columns divided by COL_SIZE
    :param mfcc (numpy array): MFCC array
    :return (numpy array): Segmented MFCC array
    '''
    segments = []
    for start in range(0, int(mfcc.shape[1] / COL_SIZE)):
        segments.append(mfcc[:, start * COL_SIZE:(start + 1) *
COL_SIZE])
    return(np.array(segments))


def create_segmented_mfccs(X_train):
    '''
    Creates segmented MFCCs from X_train
    :param X_train: list of MFCCs
    :return: segmented mfccs
    '''
    segmented_mfccs = []
    for mfcc in X_train:
        segmented_mfccs.append(segment_one(mfcc))
    return(segmented_mfccs)
```

```
def train_model(X_train,y_train,X_validation,y_validation,
batch_size=128): #64
    '''
    Trains 2D convolutional neural network
    :param X_train: Numpy array of mfccs
    :param y_train: Binary matrix based on labels
    :return: Trained model
    '''

    # Get row, column, and class sizes
    rows = X_train[0].shape[0]
    cols = X_train[0].shape[1]
    val_rows = X_validation[0].shape[0]
    val_cols = X_validation[0].shape[1]
    num_classes = len(y_train[0])

    print('X_Train shape rows:',rows)
    print('X_train1 shape cols:', cols)
    print('num_classes:',num_classes)

    # input image dimensions to feed into 2D ConvNet Input layer
    input_shape = (rows, cols, 1)
    X_train = X_train.reshape(X_train.shape[0], rows, cols, 1 )
    X_validation =
X_validation.reshape(X_validation.shape[0],val_rows,val_cols,1)


    print('X_train shape:', X_train.shape)
    print(X_train.shape[0], 'training samples')

    # Initializing the CNN
    model = Sequential()

    # Add 1st Layer Convolution, input_shape = (13,30,1), MFCCs coming
in 13x30x1
    # input shape matches the data shape coming into the network
    # Output filter of dimension 32 in the convolution,
    # Kernel size: 3x3,
    # Activation ReLU,
    # Data_format = "channels_last" which means that the ordering of
the dimensions
    # in the inputs have the form of (batch, height, width, channels)
    model.add(Conv2D(32, kernel_size=(3,3), activation='relu',
                     data_format="channels_last",
                     input_shape=input_shape))

    # Max pooling operation with a pool size of 2x2 is applied
    # to down scale the spatial dimension
    model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
    # Add 2nd convolutional layer,
    # Output filter of dimension 64 in the convolution,
    # Kernel size: 3x3,
    # Activation ReLU,
    model.add(Conv2D(64,kernel_size=(3,3), activation='relu'))
    #model.add(Conv2D(64, kernel_size=(3, 3), activation='sigmoid'))

    # Max pooling operation with a pool size of 2x2 is applied
    # to down scale the spatial dimension
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Dropout operation with a rate of 0.25 to avoid overfitting
    model.add(Dropout(0.25))

    # Flattening work in a single array, 1 dimension
    model.add(Flatten())

    # Fully Connected
    # A Regularly densely-connected layer is added with 128 units
    # Activation function of ReLU
    model.add(Dense(128, activation='relu'))
    #model.add(Dense(128, activation='sigmoid'))

    # Dropout operation with a rate of 0.5 to avoid overfitting
    model.add(Dropout(0.5))

    # The last layer is a fully connected layer with the number of ac-
cent classes
    # used for the model and
    # a softmax activation function
    model.add(Dense(num_classes, activation='softmax'))

    # Compiling the CNN
    # optimizer is reverse propagation
    # readjusting the weights
    # loss how to computer the error
    model.compile(loss='categorical_crossentropy',
                  optimizer='adadelta',
                  metrics=['accuracy'])

    # Stops training if accuracy does not change at least 0.005 over
10 epochs
    es = EarlyStopping(monitor='acc', min_delta=.005, patience=10,
verbose=1, mode='auto')

    # Creates log file for graphical interpretation using TensorBoard
    tb = TensorBoard(log_dir='../logs', histogram_freq=0,
batch_size=32, write_graph=True, write_grads=True,
                     write_images=True, embeddings_freq=0, embed-
dings_layer_names=None,
```

```
                    embeddings_metadata=None)

    # Image shifting
    datagen = ImageDataGenerator(width_shift_range=0.05)

    # Fit model using ImageDataGenerator
    # Training the CNN
    model.fit_generator(datagen.flow(X_train, y_train,
batch_size=batch_size),
                        steps_per_epoch=len(X_train) / 32
                        , epochs=EPOCHS,
                        callbacks=[es,tb], valida-
tion_data=(X_validation,y_validation))

    return (model)


def save_model(model, model_filename):
    '''
    Save model to file
    :param model: Trained model to be saved
    :param model_filename: Filename
    :return: None
    '''
    model.save('../models/{}.h5'.format(model_filename))  # creates a
HDF5 file 'my_model.h5'


if __name__ == '__main__':
    '''
        Console command example:
        python trainmodel.py data_info2L.csv model2l10_9010_relu
        '''

    start = time.time()

    # Load arguments
    file_name = sys.argv[1]
    model_filename = sys.argv[2]

    # Load metadata
    df = pd.read_csv(file_name)

    # Filter metadata to retrieve only files desired
    filtered_df = getsplit.filter_df(df)

    # Train test split
    X_train, X_test, y_train, y_test = gets-
plit.split_people(filtered_df)

    # Get statistics
```

```
    train_count = Counter(y_train)
    test_count = Counter(y_test)

    print("Entering main")

    acc_to_beat = test_count.most_common(1)[0][1] /
float(np.sum(list(test_count.values())))

    # To categorical
    y_train = to_categorical(y_train)
    y_test = to_categorical(y_test)

    # Get resampled wav files using multiprocessing
    if DEBUG:
        print('Loading wav files....')
    pool = multiprocessing.Pool(processes=multiprocessing.cpu_count())

    start_loading_wavs = time.time()
    X_train = pool.map(get_wav, X_train)
    X_test = pool.map(get_wav, X_test)
    end_loading_wavs = time.time()
    print("\nTotal time needed for Loading wav files: ",
datetime.timedelta(seconds=(end_loading_wavs - start_loading_wavs)))

    # Convert to MFCC
    if DEBUG:
        print('Converting to MFCC....')

    start_converting_mfcc = time.time()
    X_train = pool.map(to_mfcc, X_train)
    X_test = pool.map(to_mfcc, X_test)

    # Create segments from MFCCs
    X_train, y_train = make_segments(X_train, y_train)
    X_validation, y_validation = make_segments(X_test, y_test)

    # Randomize training segments
    X_train, _, y_train, _ = train_test_split(X_train, y_train,
test_size=0)

    end_converting_mfcc = time.time()
    print("\nTotal time needed for Converting to MFCC: ",
datetime.timedelta(seconds=(end_converting_mfcc -
start_converting_mfcc)))


    start_training_model = time.time()

    # Train model
    model = train_model(np.array(X_train), np.array(y_train),
np.array(X_validation),np.array(y_validation))
```

```
    # Make predictions on full X_test MFCCs
    y_predicted = accu-
racy.predict_class_all(create_segmented_mfccs(X_test), model)

    end_training_model = time.time()
    print("\nTotal time needed for Training Model: ",
datetime.timedelta(seconds=(end_training_model -
start_training_model)))


    # Print statistics
    print('Training samples:', train_count)
    print('Testing samples:', test_count)
    print('Accuracy to beat:', acc_to_beat)
    print('Confusion matrix of total samples:\n',
np.sum(accuracy.confusion_matrix(y_predicted, y_test),axis=1))
    print('Confusion matrix:\n',accuracy.confusion_matrix(y_predicted,
y_test))
    print('Accuracy:', accuracy.get_accuracy(y_predicted,y_test))

    # Save model
    save_model(model, model_filename)

    end = time.time()

    print("\nTotal time needed: ", datetime.timedelta(seconds=(end -
start)))

    winsound.PlaySound("Success", winsound.SND_FILENAME)
```

```
#----------------------------------------------------------------------
# getsplit.py
#----------------------------------------------------------------------
# original code taken from:
# Garg, Yatharth (2018). Speech-Accent-Recognition [online].
# [cited 14 Nov. 2018].
# Available from World Wide Web:
# <URL: https://github.com/yatharth1908/Speech-Accent-Recognition>.
#----------------------------------------------------------------------

import pandas as pd
import sys
from sklearn.model_selection import train_test_split


def filter_df(df):
    '''
    Function to filter audio files based on df columns
    df column options:
[age,age_of_english_onset,age_sex,birth_place,english_learning_method,
    eng-
lish_residence,length_of_english_residence,native_language,other_langu
ages,sex]
    :param df (DataFrame): Full unfiltered DataFrame
    :return (DataFrame): Filtered DataFrame
    '''

    chinese = df[df.native_language == 'chinese']
    spanish = df[df.native_language == 'spanish']
    english = df[df.native_language == 'english']
    arabic = df[df.native_language == 'arabic']

    #chinese = chinese[chinese.length_of_english_residence < 10]
    #spanish = spanish[spanish.length_of_english_residence < 10]
    #arabic = arabic[arabic.length_of_english_residence < 10]


    df = df.append(chinese)
    df = df.append(spanish)
    df = df.append(english)
    df = df.append(arabic)

    return df


def split_people(df,test_size=0.1):
    '''
    Create train test split of DataFrame
    :param df (DataFrame): Pandas DataFrame of audio files to be split
    :param test_size (float): Percentage of total files to be split
into test
```

```
    :return X_train, X_test, y_train, y_test (tuple): Xs are list of
df['language_num'] and Ys are df['native_language']
    test_size = 10% train_size = 90%
    '''


    return
train_test_split(df['language_num'],df['native_language'],test_size=te
st_size,random_state=1234)


if __name__ == '__main__':
    '''
    Console command example:
    python bio_data.csv
    '''

    csv_file = sys.argv[1]
    df = pd.read_csv(csv_file)
    filtered_df = filter_df(df)
    print(split_people(filtered_df))
```

```
#-----------------------------------------------------------------------
# accuracy.py
#-----------------------------------------------------------------------
# original code taken from:
# Garg, Yatharth (2018). Speech-Accent-Recognition [online].
# [cited 14 Nov. 2018].
# Available from World Wide Web:
# <URL: https://github.com/yatharth1908/Speech-Accent-Recognition>.
#-----------------------------------------------------------------------

from collections import Counter
import numpy as np


def predict_class_audio(MFCCs, model):
    '''
    Predict class based on MFCC samples
    :param MFCCs: Numpy array of MFCCs
    :param model: Trained model
    :return: Predicted class of MFCC segment group
    '''
    MFCCs =
MFCCs.reshape(MFCCs.shape[0],MFCCs.shape[1],MFCCs.shape[2],1)
    y_predicted = model.predict_classes(MFCCs,verbose=0)
    return(Counter(list(y_predicted)).most_common(1)[0][0])


def predict_prob_class_audio(MFCCs, model):
    '''
    Predict class based on MFCC samples' probabilities
    :param MFCCs: Numpy array of MFCCs
    :param model: Trained model
    :return: Predicted class of MFCC segment group
    '''
    MFCCs =
MFCCs.reshape(MFCCs.shape[0],MFCCs.shape[1],MFCCs.shape[2],1)
    y_predicted = model.predict_proba(MFCCs,verbose=0)
    return(np.argmax(np.sum(y_predicted,axis=0)))


def predict_class_all(X_train, model):
    '''
    :param X_train: List of segmented mfccs
    :param model: trained model
    :return: list of predictions
    '''
    predictions = []
    for mfcc in X_train:
        predictions.append(predict_class_audio(mfcc, model))
        #predictions.append(predict_prob_class_audio(mfcc, model))
    return predictions
```

```
def confusion_matrix(y_predicted,y_test):
    '''
    Create confusion matrix
    :param y_predicted: list of predictions
    :param y_test: numpy array of shape (len(y_test), number of
classes). 1.'s at index of actual, otherwise 0.
    :return: numpy array. confusion matrix
    '''
    confusion_matrix =
np.zeros((len(y_test[0]),len(y_test[0])),dtype=int )
    for index, predicted in enumerate(y_predicted):
        confusion_matrix[np.argmax(y_test[index])][predicted] += 1
    return(confusion_matrix)


def get_accuracy(y_predicted,y_test):
    '''
    Get accuracy
    :param y_predicted: numpy array of predictions
    :param y_test: numpy array of actual
    :return: accuracy
    '''
    c_matrix = confusion_matrix(y_predicted,y_test)
    return( np.sum(c_matrix.diagonal()) / float(np.sum(c_matrix)))

if __name__ == '__main__':
    pass
```

```
#-----------------------------------------------------------------------
# predict.py
#-----------------------------------------------------------------------
# original code taken from:
# Garg, Yatharth (2018). Speech-Accent-Recognition [online].
# [cited 14 Nov. 2018].
# Available from World Wide Web:
# <URL: https://github.com/yatharth1908/Speech-Accent-Recognition>.
# modified by Stavros Grigoriadis
#-----------------------------------------------------------------------

import numpy as np
import accuracy
from keras.models import load_model
import librosa
import pandas as pd
import getsplit

RATE = 16000
N_MFCC = 13
COL_SIZE = 30


def get_pred_wav(language_num):
    '''
    Load wav file from disk and down-samples to RATE
    :param language_num (list): list of file names
    :return (numpy array): Down-sampled wav file
    '''

    y, sr = li-
brosa.load('../Prediction_File/{}.wav'.format(language_num))
    return(librosa.core.resample(y=y,orig_sr=sr,target_sr=RATE,
scale=True))


def create_segmented_mfccs(X_train):
    '''
    Creates segmented MFCCs from X_train
    :param X_train: list of MFCCs
    :return: segmented mfccs
    '''
    segmented_mfccs = []
    for mfcc in X_train:
        segmented_mfccs.append(segment_one(mfcc))
    return(segmented_mfccs)


def to_mfcc(wav):
    '''
    Converts wav file to Mel Frequency Ceptral Coefficients
```

```
    :param wav (numpy array): Wav form
    :return (2d numpy array: MFCC
    '''
    return(librosa.feature.mfcc(y=wav, sr=RATE, n_mfcc=N_MFCC))


def segment_one(mfcc):
    '''
    Creates segments from on mfcc image. If last segments is not long
enough to be length of columns divided by COL_SIZE
    :param mfcc (numpy array): MFCC array
    :return (numpy array): Segmented MFCC array
    '''
    segments = []
    for start in range(0, int(mfcc.shape[1] / COL_SIZE)):
        segments.append(mfcc[:, start * COL_SIZE:(start + 1) *
COL_SIZE])
    return(np.array(segments))


def create_segmented_mfccs(X_train):
    '''
    Creates segmented MFCCs from X_train
    :param X_train: list of MFCCs
    :return: segmented mfccs
    '''
    segmented_mfccs = []
    for mfcc in X_train:
        segmented_mfccs.append(segment_one(mfcc))
    return(segmented_mfccs)


# Load Model
print("============================================================")
print("Loading Model")
new_model = load_model('../models/model4182.h5')
print("============================================================")

file_name = 'data_predict.csv'

# Load metadata
df = pd.read_csv(file_name)

# Filter metadata to retrieve only files desired
filtered_df = getsplit.filter_df(df)

# Train test split
X_predict, X_test, y_train, y_test = gets-
plit.split_people(filtered_df)

X_predict = map(get_pred_wav, X_predict)
```

```
X_predict = map(to_mfcc, X_predict)
y_predicted = accu-
racy.predict_class_all(create_segmented_mfccs(X_predict), new_model)

if y_predicted == [0]:
    print ("Chinese Accent Found")
if y_predicted == [1]:
    print ("Spanish Accent Found")
if y_predicted == [2]:
    print ("English Accent Found")
if y_predicted == [3]:
    print ("Arabic Accent Found")
```