ABSTRACT

The purpose of this project is to implement an end-to-end automatic speech recognition system using recurrent neural networks, the Arabic language which ranks as the fifth most spoken language in the world has been chosen as the main language of the system. The Arabic language has been alienated from such type of projects due to its complexity, uniqueness and lack of free appropriate corpuses, but with new emerging algorithms in the domain of speech recognition such as the connectionist temporal classification, it is becoming more accessible to use unsegmented corpuses in the aim of building performant automatic speech recognition systems. The development of the project includes basic digital signal processing, exploration of the phonetic properties of the Arabic language, an adaption of a general corpus to fit the purpose of the project, feature extraction and a brief study on recurrent neural networks their performance in such a system.

The full system with its various parts is implemented in Python and TensorFlow, different models inspired from literature are trained and tested using the Arabic speech corpus, leading to a selection of a final model that shows the lowest word error rate of 35.23%.

The results encourage to explore more in depth the implementation of a speaker independent robust Arabic speech recognition system.

D E D I C A T I O N

I would like to dedicate this work to my Mother

ACKNOWLEDGMENT

I would like to thank Allah for providing me with good health, faith and courage to finish this work.

I would like to thank my supervisor Dr. AbdelHakim Dahimene, for his guidance throughout the period of this project, for providing his expertise on signal processing and speech recognition in general, and all the fruitful discussions that we had.

I would like to thank Mr. Youssouf Ismail Cherifi for his precious help throughout the full period of this project, for providing his expertise in the domain of neural networks, and for providing computation time to train the different models at the Vision Institute in Paris.

I would like to thank Dr. Nawar Halabi from the University of Southampton for providing his Arabic speech corpus free of charge, and also for answering all the questions about the corpus throughout the period of the project.

I would like to thank all my family and friends who supported me during the period of the project.

TABLE OF COTENTS

T A B L E O F C O N T E N T S

LIST OF FIGURES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ASR	Automatic Speech Recognition
GPGPUs	General Purpose Graphical Processing Units
TPUs	Tensor Processing Units
HMMs	Hidden Markov Models
DNN	Deep Neural Network
MLP	Multilayer Perceptron
GMMs	Gaussian Mixture Models
RNNs	Recurrent Neural Networks
CTC	Connectionist Temporal Classification
LSTM	Long Short-Term Memory
BRNNs	Bidirectional Recurrent Neural Networks
CNNs	Convolutional Neural Networks
MFCCs	Mel-Frequency Cepstral Coefficients
HAS	Human Auditory System
DCT	Discrete Cosine Transform
MSA	Modern Standard Arabic
WER	Word Error Rate

GENERAL INTRODUCTION

Automatic speech recognition (ASR) has been an active research area for over five decades. It has always been considered as an important bridge in fostering better human–human through machine translation and human–machine communication. In the past, however, speech never actually became an important modality in the human–machine communication.

This is partly because the technology at that time was not good enough to pass the usable bar for most real-world users under most real usage conditions, and partly because in many situations alternative communication modalities such as keyboard and mouse significantly outperform speech in the communication efficiency, restriction, and accuracy.

In the recent years, speech technology started to change the way we live and work and became one of the primary means for humans to interact with some devices. This trend started due to the progress made in several key areas. First, Moor's law continues to function. The computational power available today, through multi-core processors, general purpose graphical processing units (GPGPUs), CPU/GPU clusters, and the new tensor processing units (TPUs), is several orders of magnitude more than that available just a decade ago. This makes training of more powerful yet complex models possible. These more computation demanding models, which are the topic of this book, significantly reduced the error rates of the ASR systems. Second, we can now access to much more data than before, thanks to the continued advance of the Internet and the cloud computing. By building models on big data collected from the real usage scenarios, we can eliminate many model assumptions made before and make systems more robust. Third, mobile devices, wearable devices, intelligent living room devices, and in-vehicle infotainment systems became popular. On these devices and systems, alternative interaction is preferable [1].

CHAPTER 1: STATE OF THE ART

This chapter provides a gentle introduction to the main topic of this report, and a brief history of automatic speech recognition (ASR).

1.1 HIDDEN MARKOV MODELS (HMM)

HMM is a very powerful statistical method of characterizing the observed data samples of a discrete-time series. Not only can it provide an efficient way to build parsimonious parametric models but can also incorporate the dynamic programming principle in its core for a unified pattern segmentation and pattern classification of time-varying data sequences. The data samples in the time series can be discretely or continuously distributed; they can be scalars or vectors. The underlying assumption of the HMM is that the data samples can be well characterized as a parametric random process, and the parameters of the stochastic process can be estimated in a precise and well-defined framework [2].

The HMM had been widely used in speech recognition, natural language modelling, on-line handwriting recognition, and for the analysis of biological sequences such as proteins and DNA [3].

HMM is a good method for constructing such models because speech has a temporal structure and can be encoded as a sequence of spectral vectors inside the audio frequency range.

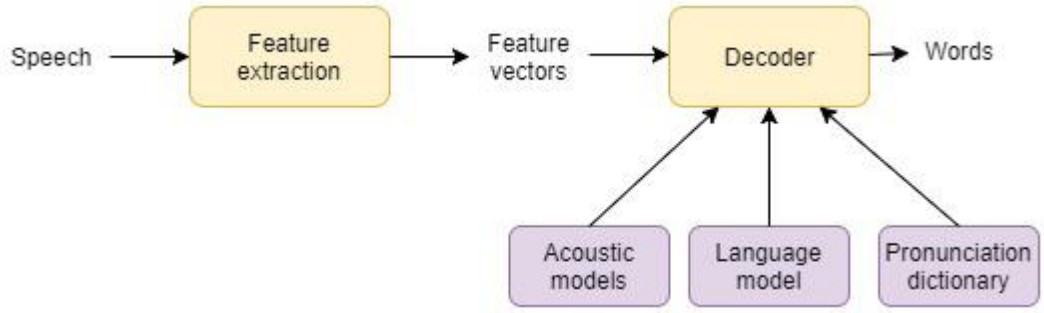

Figure 1.1-1Components of a Speech Recognizer using HMM [4]

1.2 DEEP NEURAL NETWORK - HIDDEN MARKOV MODEL (DNN-HMM)

A deep neural network (DNN) is a conventional multilayer perceptron (MLP) with many (often more than two) hidden layers, each hidden unit uses a nonlinear function to map the feature input from the layer below to the current unit. [1].

A perceptron is an algorithm for supervised learning of binary classifiers (functions that can decide whether an input, represented by a vector of numbers, belongs to some specific class or not). It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector [5].

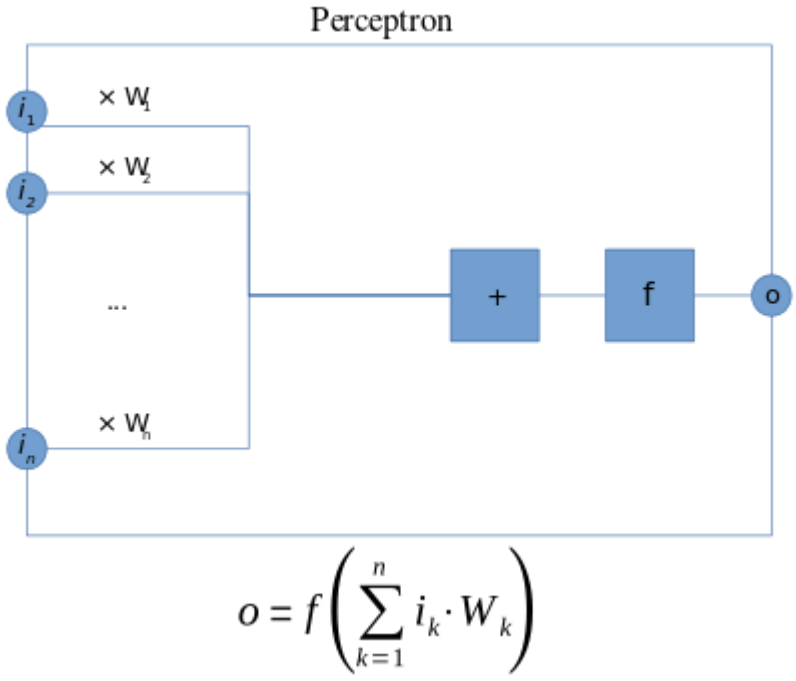

Figure 1.2-1 A diagram of a Perceptron [5]

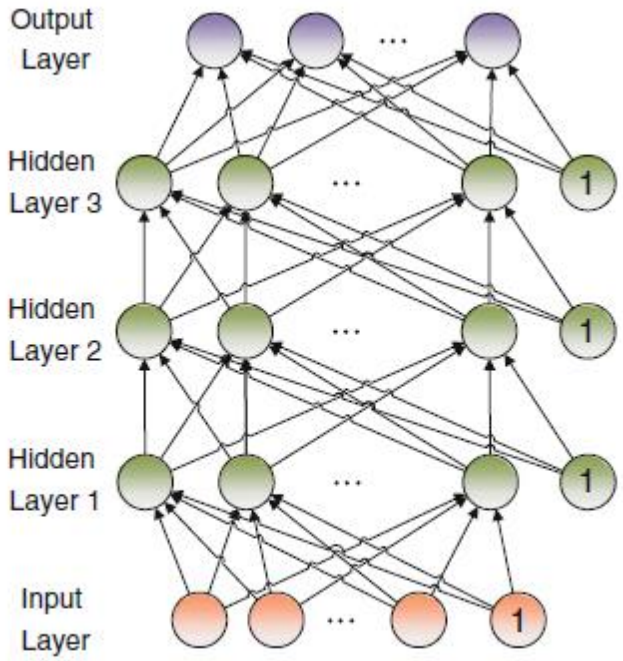

Figure 1.2-2 An example deep neural network with an input layer, three hidden layers, and an output layer [1]

Deep neural networks have become popular acoustic models for state-of-the-art large vocabulary speech recognition systems . These combinations of neural networks and statistical models are often referred to as hybrid systems. Using the new learning methods, several different research groups have shown that DNNs outperform (Gaussian Mixture Models)GMMs at acoustic modeling for speech recognition on a variety of datasets including large datasets with large vocabularies [6].

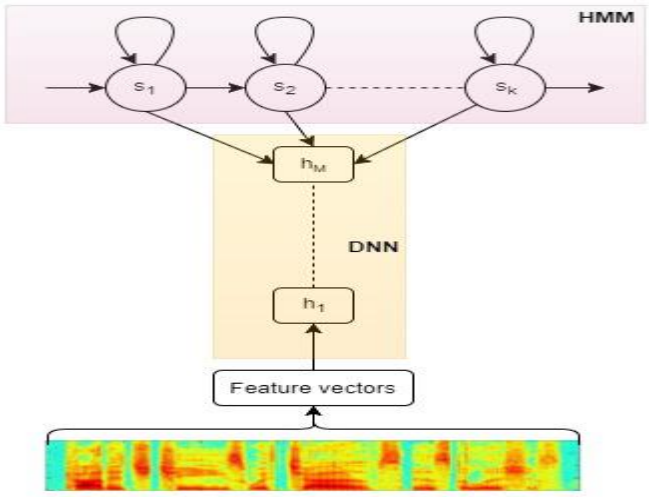

Figure 1.2-3 DNN-HMM model structure [4]

1.3 END TO END ASR

Previous methods have strong markovian-assumptions and required several separate components and training for the pronunciation, acoustic and language model.

Acoustic models take acoustic features and predict a set of sub word units (phonemes). Then the pronunciation model, which is a hand-designed lexicon, maps a sequence of phonemes produced by the acoustic model to words. Finally, the language model is in charge of assigning probabilities to word sequences.

Training independent components is complex and suboptimal compared to training all components jointly. That is why there has been a growing popularity in developing end-to-end systems over the last several years [4].

A special type of DNN called recurrent neural networks (RNNs) are a powerful model for sequential data. End-to-end training methods such as Connectionist Temporal Classification make it possible to train RNNs for sequence labelling problems where the input-output alignment is unknown. The combination of these methods with the Long Short-term Memory RNN architecture has proved particularly fruitful [7].

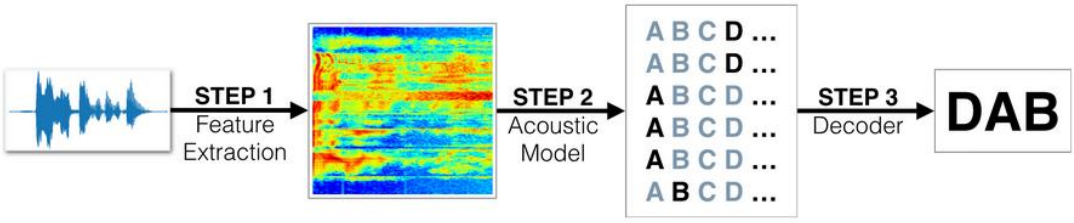

Figure 1.3-1End-to-End ASR system pipeline

1.3.1 Connectionist Temporal Classification (CTC)

CTC is a method for labelling sequence data with RNNs that removes the need for pre-segmented training data and post-processed outputs, and models all aspects of the sequence within a single network architecture. The basic idea is to interpret the network outputs as a probability distribution over all possible label sequences, conditioned on a given input sequence. Given this distribution, an objective function can be derived that directly maximizes the probabilities of the correct labeling. Since the objective function is differentiable, the network can then be trained with standard backpropagation through time [8].

CHAPTER 2: THEORITICAL BACKGROUND

The aim of this chapter is to provide the necessary theoretical background about the various methods used in this report.

2.1 DEEP NEURAL NETWORKS

Deep neural networks have proven very useful in solving many real world learning tasks such as sequence learning tasks that require the prediction of sequences of labels from noisy, unsegmented input data.

In speech recognition, for example, an acoustic signal is transcribed into words or sub-word units. RNNs are powerful sequence learners that have proven to be well suited to such tasks, why they are the focus of this section.

2.1.1 Recurrent Neural Networks

A recurrent neural network is a class of neural network models where many connections among its neurons form a directed cycle. This gives rise to the structure of internal states or memory in the RNN, endowing it with the dynamic temporal behavior not exhibited by the DNN [1].

Given an input sequence $x = (x_1, \ldots, x_T)$, a standard RNN computes the hidden vector sequence $h = (h_1, \ldots, h_T)$ and output vector sequence $y = (y_1, \ldots, y_T)$ by iterating the following equations from $t = 1$ to T:

$$h_t = H(W_{xh}x_t + W_{hh}h_t - 1 + b_h) \quad (2.1)$$

$$y_t = W_{hy}h_t + b_y \quad (2.2)$$

Where the W terms denote weight matrices (e.g. W_{xh} is the input-hidden weight matrix), the b terms denote bias vectors (e.g. b_h is hidden bias vector) and H is the hidden layer function, H is usually an elementwise application of a sigmoid function. However Long Short-Term Memory (LSTM) [9] architecture using LSTM cells shown in *figure 3.1-1*, which uses purpose-built memory cells to store information, is better at finding and exploiting long range context [7].

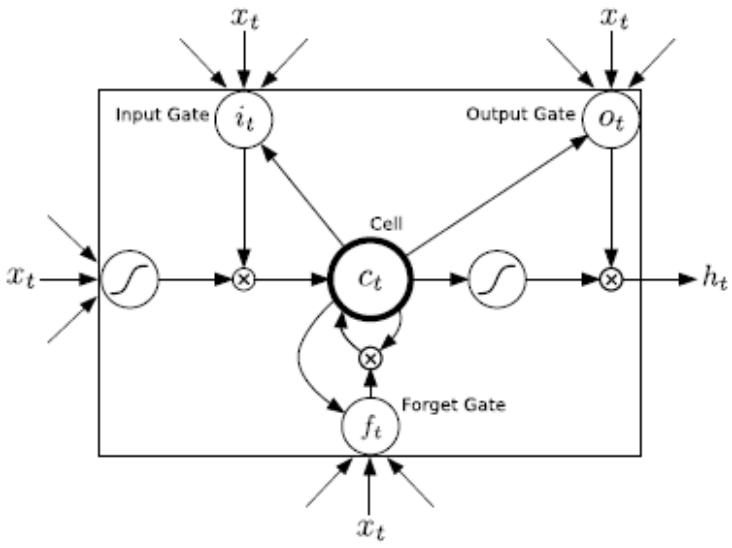

Figure 2.1-1 Long Short-term Memory Cell [7]

One shortcoming of conventional RNNs is that they are only able to make use of previous context. In speech recognition, where whole utterances are transcribed at once, there is no reason not to exploit future context as well. Bidirectional RNNs (BRNNs) do this by processing the data in both directions with two separate hidden layers, which are then fed forwards to the same output layer [7]. As illustrated in *figure 3.1-2*.

Combining BRNNs with LSTM gives bidirectional LSTM.

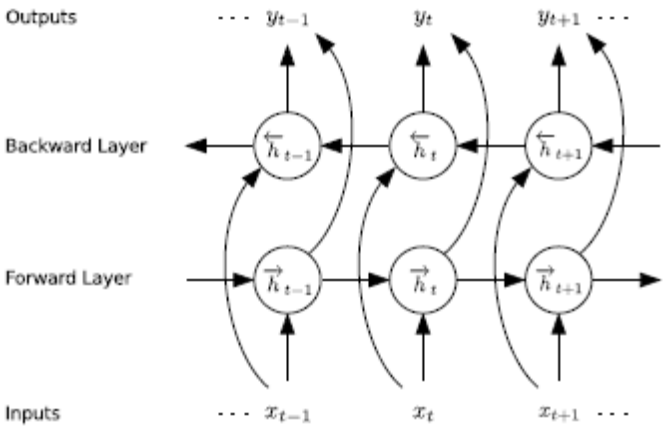

Figure 2.1-2 Bidirectional RNN [7]

A crucial element of the recent success of RNNs is the use of deep architectures that can be achieved by stacking multiple RNN hidden layers on top of each other, with the output sequence of one layer being the input sequence for the next one [7].

2.1.2 Training RNNs

Although the gradients of the RNN are easy to compute, RNNs are fundamentally difficult to train, especially on problems with long-range temporal dependencies, such as ASR, due to their nonlinear iterative nature. A small change to an iterative process can compound and result in very large effects many iterations later; this is known colloquially as "the butterfly-effect". The implication is that in an RNN, the derivative of the loss function at one time can be exponentially large with respect to the hidden activations at a much earlier time [10].

2.1.3 Connectionist temporal classification

Before CTC was proposed the most effective use of RNNs for sequence labelling was to combine them with HMMs in the so called hybrid approach. Hybrid systems use HMMs to model the long-range sequential structure of the data, and neural nets to provide localized classifications. The HMM component is able to automatically segment the sequence during training, and to transform the network classifications into label sequences [8].

Considering the mapping of input sequence X of length T, $X=[x_1,x_2,\ldots,x_T]$, such as audio, to the corresponding output sequences $Y=[y_1,y_2,\ldots,y_U]$, such as transcripts. The aim is to find an accurate mapping from X's to Y's [8].

There are challenges which get in the way of using simpler supervised learning algorithms, both X and Y can vary in length, the ratio of the lengths of X and Y can vary and most data sets don't provide and accurate alignment (correspondence of the elements) of X and Y.

The CTC algorithm overcomes these challenges. For a given X_i gives us an output distribution over all possible Y's. This distribution can be used either to infer a likely output or to assess the probability of a given output.

A CTC network has a softmax output layer with one more unit than there are labels in the (finite) alphabet L, The activations of the first |L| units are interpreted as the probabilities of observing the corresponding labels at particular times. The activation of

the extra unit is the probability of observing a ‘blank’, or no label. Together, these outputs define the probabilities of all possible ways of aligning all possible label sequences with the input sequence. The total probability of any one label sequence can then be found by summing the probabilities of its different alignments [8].

The network switches from predicting no label to predicting a label, or from predicting one label to another, using a many-to-one map B, that simply removes all blanks and repeated labels from the paths [8].

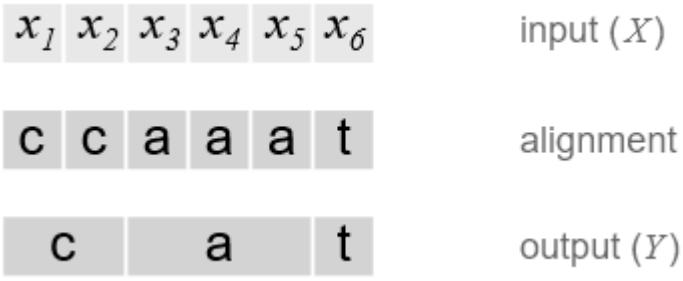

Figure 2.1-3 Connectionist temporal classification based speech recognition system

Finally B is used to define the conditional probability of a given labelling $l \in L \leq T$ as the sum of the probabilities of all the paths corresponding to it:

$$P(l \mid x) = \sum_{\pi \in B^{-1}(l)} p(\pi/x) \quad (2.3)$$

Given the above formulation, the output of the classifier should be the most probable labelling for the input sequence, then an objective function is derived from the principle of maximum likelihood. That is, minimizing it maximizes the log likelihoods of the target labeling, the same principle underlying the standard neural network objective functions. Given the objective function, and its derivatives with respect to the network outputs, the weight gradients can be calculated with standard backpropagation through time. The network can then be trained with any of the gradient-based optimization algorithms [8].

2.1.4 Convolutional Neural Networks

Convolutional neural network (CNN, or ConvNet) are a class of deep, feed-forward artificial neural networks, most commonly applied to analyzing visual imagery.

CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. They apply a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli. Although fully connected feedforward neural networks can be used to learn features as well as classify data, it is not practical to apply this architecture to images. A very high number of neurons would be necessary, even in a shallow (opposite of deep) architecture, due to the very large input sizes associated with images, where each pixel is a relevant variable. For instance, a fully connected layer for a (small) image of size 100 x 100 has 10000 weights for each neuron in the second layer. The convolution operation brings a solution to this problem as it reduces the number of free parameters, allowing the network to be deeper with fewer parameters. For instance, regardless of image size, tiling regions of size 5 x 5, each with the same shared weights, requires only 25 learnable parameters [11].

2.2 AUDIO FEATURES REPRESENTAION

Sound is a longitudinal pressure wave formed of compressions and rarefactions of air molecules [2], Audio can be represented in many ways, and which one is "best" depends on the application as well as the processing machinery. For many years, feature design and selection was a key component of many audio analysis tasks and the list includes spectral centroid and higher order statistics of spectral shape, zero crossing statistics, harmonicas, fundamental frequency, and temporal envelope descriptions [12].

Traditionally speech recognition research has largely focused on using log mel-filterbank energies or mel-frequency cepstral coefficients (MFCCs), but has been moving to raw audio recently [13] [14] [15], Recurrent neural networks such as LSTM-RNNs have been a key component in these new speech classification pipelines, because they allow for building models with long range contexts.

2.2.1 Human Speech Perception

The auditory perception system can be split in two major components: the peripheral auditory system (ears), and the auditory nervous system (brain). The received

acoustic pressure signal is processed by peripheral auditory system into two steps: firstly, it is transformed into a mechanical vibration pattern on the basilar membrane; and then, is represented by a series of pulses to be transmitted by the auditory nerve. Finally, the auditory nervous system is responsible for extracting the perceptual information. The human ear, as shown in *Figure 3.2-1*, is made up of three parts: the outer ear, the middle ear, and the inner ear. The outer ear consists of the external visible part and the external auditory canal is where sound wave travels. The length of the auditory canal is such that performs as an acoustic resonator whose principal effect is to increase the ear's sensitivity to sounds in the 3-4 KHz range. When the sound arrives at the eardrum, it vibrates at the same frequency as the incoming sound pressure wave. The vibrations are transmitted through the middle ear. The main structure of the inner ear is the cochlea which communicates with the auditory nerve, driving a representation of sound to the brain [16].

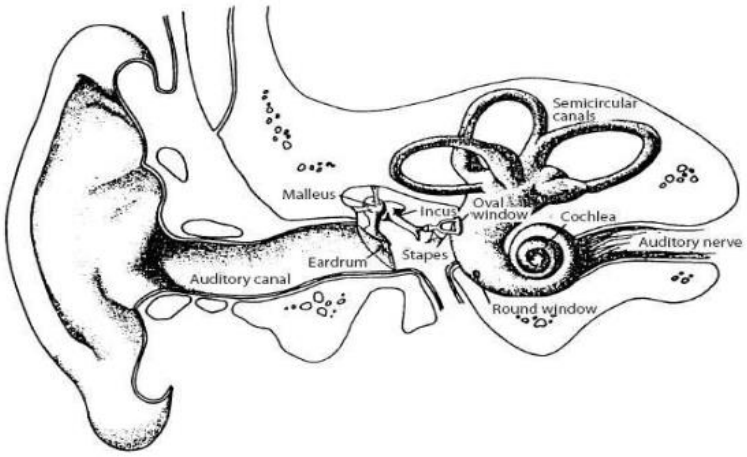

Figure 2.2-1 Peripheral auditory system [16]

There are two main competing theories about human hearing, Place theory which states that our perception of sound depends on where each component frequency produces vibrations along the basilar membrane. By this theory, the pitch of a sound, such as a human voice or a musical tone, is determined by the places where the membrane vibrates, based on frequencies corresponding to the tonotopic organization of the primary auditory neurons [17], and the other one is the temporal theory which states that human perception of sound depends on temporal patterns with which neurons respond to sound in the cochlea. Therefore, in this theory, the pitch of a pure tone is determined by the period of

neuron firing patterns either of single neurons, or groups as described by the volley theory [18].

2.2.2 Raw audio Features

Audio signals are data-intensive time-domain signals and are stored (in their basic uncompressed form) as series of numbers corresponding to the amplitude of the signal over time. Although this representation is adequate for transmission and reproduction of arbitrary waveforms it is not appropriate for analyzing and understanding audio signals. The way we perceive and understand audio signals as humans is based on and constrained by our auditory system. It is well known that the early stages of the human auditory system (HAS) decompose incoming sound wave into different frequency bands [19].

Researchers usually avoid modelling raw audio because it ticks so quickly: typically 16,000 samples per second or more, with important structure at many time-scales, but as mentioned earlier in the beginning of this section some research groups found interest in using the raw audio features as input to speech recognition models.

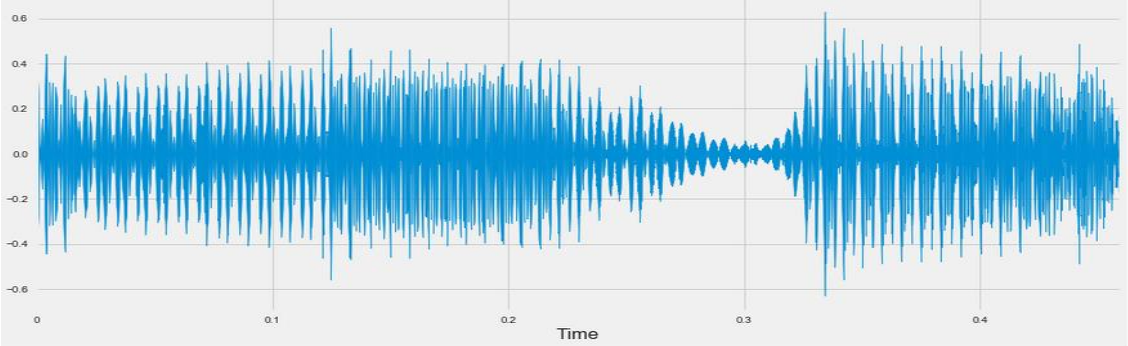

Figure 2.2-2 Raw Features of a single word

2.2.3 Spectral Shape Features

Mel-Frequency Cepstral Coefficients (MFCCs) were very popular features for a long time; but more recently, filter banks are becoming increasingly popular.

Filter Banks and MFCCs involve the same procedure, where in both cases filter banks are computed with a few more extra steps MFCCs can be obtained a signal goes through a pre-emphasis filter; then gets sliced into (overlapping) frames and a window function is applied to each frame; afterwards, a Short-Time Fourier Transform is applied and the

power spectrum is calculated; subsequently the filter banks are computed. To obtain MFCCs, a Discrete Cosine Transform (DCT) is applied to decorrelate the resulting feature vectors retaining a number of the resulting coefficients while the rest are discarded. A final step in both cases, is mean normalization.

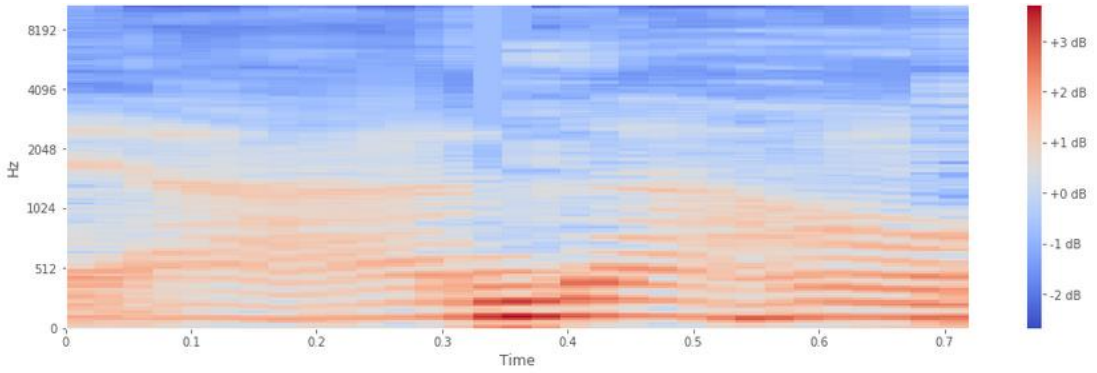

Figure 2.2-3 Spectrogram of a single word

2.3 SOFTWARE TOOLS

2.3.1 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed [20].

2.3.2 TensorFlow

TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google’s AI organization, it comes with strong support for

machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains [21].

2.3.3 Keras

Keras is a high-level neural networks library, written in Python and capable of running on top of either TensorFlow or Theano (Keras, 2017). It was developed with the purpose of enabling fast experimentation. Providing results with the least possible delay, Keras constitutes a key for good research. It is designed on the following properties: modularity, so many functions, graphs and neural layers to create new models when combined together, minimalism, that it is important each module to be short and simple and easy extensibility, the ability to add new models [22].

2.3.4 Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows the creation and sharing of documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning [23].

CHAPTER 3: IMPLEMENTATION AND RESULTS

In this chapter the implementation of a speech recognition system using RNN, LSTM and CTC, the main purpose is to implement an end to end model capable of doing phoneme level recognition without the need of a phoneme labeled database. The model is to be trained on an Arabic speech corpus that was adapted for the purpose of this project.

All the scripts in this section were written in python.

3.1 ARABIC SPEECH CORPUS

The Arabic Speech Corpus is a Modern Standard Arabic (MSA) speech corpus was mainly built for speech synthesis purposes. The corpus contains phonetic and orthographic transcriptions of more than 3.7 hours of MSA speech. The annotations include word stress marks on the individual phonemes. The corpus contains an extra separate 18 minutes of fully annotated corpus which was used for evaluations [24].

Transcription of the corpus is in a modified Buckwalter Arabic transliteration that provides a strictly one-to-one mapping, unlike the more common Romanization schemes that add morphological information not expressed in Arabic script which is undesirable for the application of this report, more information about it can be found in the **Appendix**.

The Arabic Speech Corpus was built as part of a doctoral project by Nawar Halabi at the University of Southampton funded by MicroLinkPC who own an exclusive license to commercialize the corpus, but the corpus is available for strictly non-commercial purposes through the official Arabic Speech Corpus website.

3.2 CORPUS ADAPTATION

In speech recognition each language has its own set of challenges and properties, one of the challenges faced in building an Arabic speech recognition system was the lack of specially built and tested corpuses.

The corpus at hand provided audio wav files of 1813 sentences with their transcription, in addition to 100 sentences that was used for testing and validation purposes.

Data preparation is an important task in deep learning as the data has to be in a format that allows to first layer of the model to extract the features.

Since the Arabic speech corpus wasn't tested on ASR before, it was preferred to provide at least two levels of labeling, word level labeling, and phrase level labeling that was already available.

```
"ARA NORM  0001.wav" ">atAHat lilbA}iEi lmutajaw~ili >an yakuwna jA*iban
lilmuwATini l>aqal~i daxlan"
"ARA NORM  0002.wav" ">aHrazat muntaxabAtu lbarAziyli wa>lmAnyA waruwsyA - fawzan
fiy muqAbalAtihim l<iEdAdiy~api l~atiy >uqiymat istiEdAdan linihA}iy~Ati ka>si
lEAlam - >al~atiy satanTaliqu baEda >aqal~i min >usbuwE"
"ARA NORM  0003.wav" ">axfaqa majlisu ln~uw~Abi ll~ubnAniy~u fiy xtiyAri ra}iysin
jadiydin lilbilAdi - xalafan lilr~a}iysi lHAliy~i l~a*iy tantahiy wilAyatuhu fiy
lxAmisi wAlEi$riyn - min mAyuw >ayAra lmuqbil"
"ARA NORM  0004.wav" "<i* sayaHDuru liqA'a ha*A lEAmi xamsun wa^alA^uwna minhum"
```

Figure 3.2-1 Orthographic transcription format for each audio file in the corpus

To do so an open source software for phonetic analysis called Praat was used. Praat was chosen because it was the same software used to make the corpus and for its strong scripting environment that enables the automation of tasks which was very important.

The creator of the corpus also provides Textgrid files which marked the alignment of the transcription with the audio file, this can be viewed in *figure 3.2-2*.

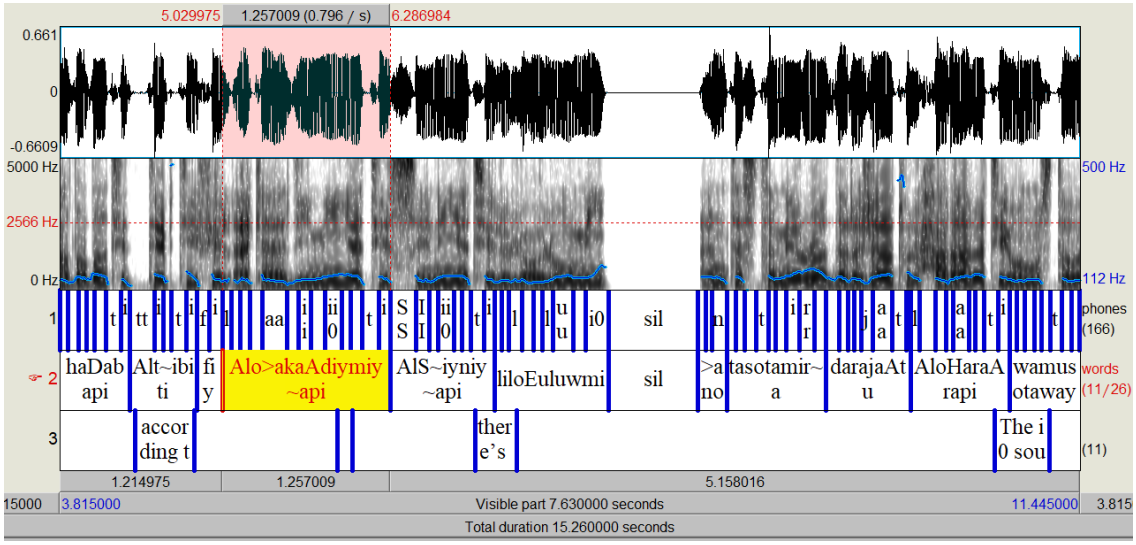

Figure 3.2-2 Praat editing interface

Based on these marks a Praat script had been developed to go through all the speech corpus segment the phrase long audio files into word with the appropriate label for each word.

The script took as in put each phrase long audio file and the corresponding Textgrid file which provided orthographic transcription and the boundaries for each word, the output

of the script was folders for each audio file that contained multiple audio wav files with a txt file that contained the transcription of each new file, this can be viewed in the *figure 3.2-3* which represents the output of the script for an input phrase audio file which contained 9 words and 2 silence files.

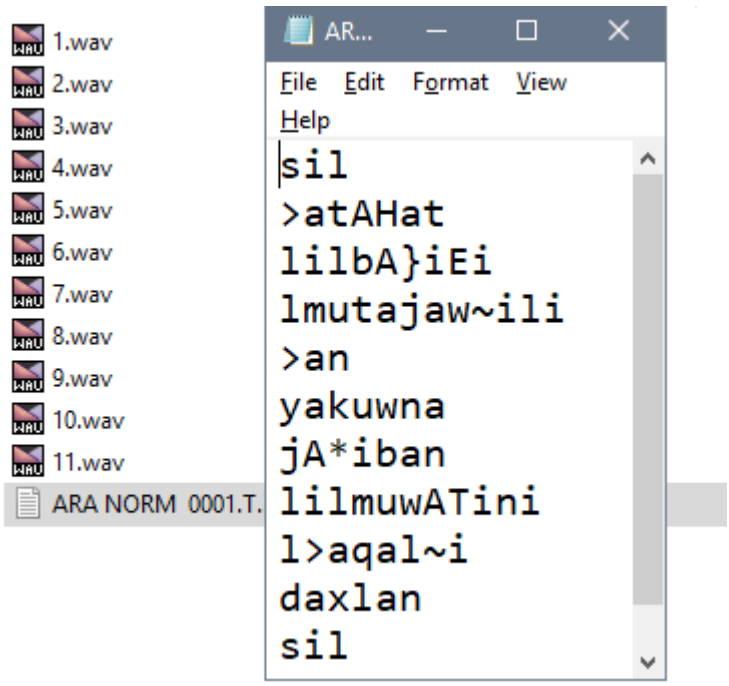

Figure 3.2-3 The output of the audio segmentation script

The script was applied on both the training the test sets and gave as output 6772 audio file from the training set and 644 audio file from the test set, with this we had at least 2 representations to test the model on, it should be also noted that the same script could have been used to get the phonemes level labeling of the corpus.

3.3 PREPROCESSING

3.3.1 Audio preprocessing

The audio signals had to be preprocessed before feeding them as in put to the neural networks, the raw normalized spectrogram was the format of choice for the work of this report.

The first step was to explore the dataset, a script that takes an audio wav file and plots its wave plot and spectrogram was developed, it was mostly useful in further steps of the project for debugging purposes.

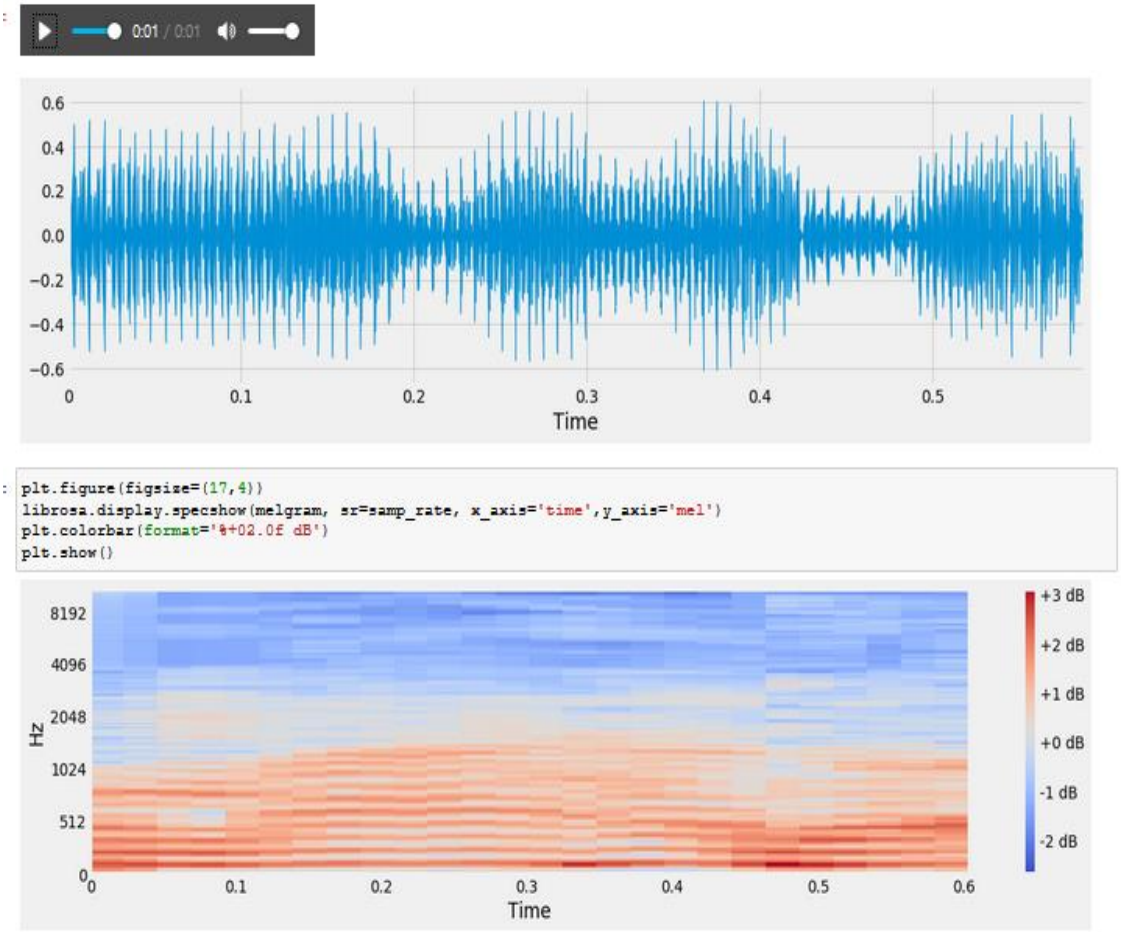

Figure 3.3-1 Mel scaled spectrogram and wave plot of an audio file for the word 'maEohadu'

After having acquired a tool to visualize the data, the main script to process the speech corpus has been developed, the script took as input an audio file, resampled it at 22050Hz, and computed its normalized spectrogram and gave an output array of shape (timesteps * 161) which has been used later on as input to the neural network.

```
print(x.shape)
print(x)
```

```
(26, 161)
[[ 1.3061853   1.60028362  1.80817141 ..., -0.44543381 -0.38910482
  -0.38373428]
 [ 1.15096253  1.39287767  1.51785927 ..., -0.63975621 -0.58615215
  -0.58109619]
 [ 1.04274239  0.90352317  0.45494975 ..., -1.3305764  -1.3305764
  -1.3305764 ]
 ...,
 [ 0.54445998  0.38925801  0.82008895 ..., -1.64685924 -1.64685924
  -1.64685924]
 [ 0.72392944  0.4789813   0.71860355 ..., -1.51006057 -1.33356244
  -1.36190009]
 [ 1.05126674  0.96342214  0.99870907 ..., -1.41249514 -1.14432962
  -1.18210355]]
```

Figure 3.3-2 Audio features array

3.3.2 Text preprocessing

The text labels were in a modified Buckwalter format which produced an alphabet of 46 character, a one-to-one char map was created each character taking an integer in the range 0 to 45, and this gave a way to transform each word into a vector based on the characters it contained.

```
{'q': 29, 'm': 32, '*': 16, 'a': 41, 'u': 42, 'H': 13, 'i': 43, 't': 10, 'f': 28, 'd': 15, '}
': 6, "'": 1, 'g': 26, '_': 27, 'Z': 24, 'D': 22, 'E': 25, 'S': 21, 'b': 8, 's': 19, 'h': 34,
'k': 30, '<SPACE>': 0, 'T': 23, 'j': 12, 'z': 18, 'N': 39, 'Y': 36, 'x': 14, 'A': 7, 'y': 37,
'&': 4, 'r': 17, 'o': 45, '|': 2, '$': 20, 'w': 35, 'l': 31, 'K': 40, '>': 3, 'n': 33, '<': 5
, '^': 11, 'p': 9, 'F': 38, '~': 44}
```

Figure 3.3-3 Speech corpus alphabet character map

Two other scripts had been developed for two functions text_to_int_sequence which took a string and gave an array of numbers using the character map, and another function int_sequence_to_text which did the inverse, the second function was used in a further step of the project to retransform the network output into text again.

```
input_ch="kamA tam~a taHsiynu wAjihAti lt~anaq~ul   wAxtiyAri wasA}ili ln~aqli"
int_sequence=text_to_int_sequence(input_ch)
word_=int_sequence_to_text(int_sequence)

print(input_ch)
print(int_sequence)
print(''.join(word_))
```

```
kamA tam~a taHsiynu wAjihAti lt~anaq~ul   wAxtiyAri wasA}ili ln~aqli
[30, 41, 32, 7, 0, 10, 41, 32, 44, 41, 0, 10, 41, 13, 19, 43, 37, 33, 42, 0, 35, 7, 12, 43, 3
4, 7, 10, 43, 0, 31, 10, 44, 41, 33, 41, 29, 44, 42, 31, 0, 0, 0, 35, 7, 14, 10, 43, 37, 7, 1
7, 43, 0, 35, 41, 19, 7, 6, 43, 31, 43, 0, 31, 33, 44, 41, 29, 31, 43]
kamA tam~a taHsiynu wAjihAti lt~anaq~ul   wAxtiyAri wasA}ili ln~aqli
```

Figure 3.3-4 Text preprocessing

3.3.3 Arabic speech corpus preprocessing

A new script had been developed using the tools developed from the previous two sections this script processed the whole Arabic speech corpus and gave back the arrays that had been used in later parts of the project to train a speech recognition model.

The script took as input the link to the Arabic speech corpus and a csv file, audio preprocessing was performed to acquire the arrays which were saved in a new directory in a .npy format, the csv file contained the duration, numpy array path and the label for each sample.

The processing was performed on both training and test set.

```
#preprocess the test set
csv_f = open('test.csv', 'a+')
process_AraSpeechCorpus(csv_f,test_dir)
csv_f.close()
```

```
ARA NORM  00019.wav
16758 22050
Arabic Speech corpus preprocessing (9 / 644) - 'E:\Lectures\Masters\FYP Masters\arabic-speech-corpus\slices\test se
t\ARA NORM  0001\10.wav']
ARA NORM  000110.wav
14774 22050
Arabic Speech corpus preprocessing (10 / 644) - 'E:\Lectures\Masters\FYP Masters\arabic-speech-corpus\slices\test s
et\ARA NORM  0001\11.wav']
ARA NORM  000111.wav
2205 22050
Arabic Speech corpus preprocessing (11 / 644) - 'E:\Lectures\Masters\FYP Masters\arabic-speech-corpus\slices\test s
et\ARA NORM  0002\1.wav']
ARA NORM  00021.wav
2426 22050
Arabic Speech corpus preprocessing (12 / 644) - 'E:\Lectures\Masters\FYP Masters\arabic-speech-corpus\slices\test s
et\ARA NORM  0002\2.wav']
ARA NORM  00022.wav
17420 22050
Arabic Speech corpus preprocessing (13 / 644) - 'E:\Lectures\Masters\FYP Masters\arabic-speech-corpus\slices\test s
et\ARA NORM  0002\3.wav']
```

Figure 3.3-5Test set preprocessing

```
#Preprocess the training set
csv_f = open('train.csv', 'a+')
process_AraSpeechCorpus(csv_f, train_dir)
csv_f.close()
```

```
Arabic Speech corpus preprocessing (0 / 6772) - 'E:\Lectures\Masters\FYP Masters\arabic-speech-corpus\slices\Traini
ng set\ARA NORM  0002\1.wav']
ARA NORM  00021.wav
2205 22050
Arabic Speech corpus preprocessing (1 / 6772) - 'E:\Lectures\Masters\FYP Masters\arabic-speech-corpus\slices\Traini
ng set\ARA NORM  0002\2.wav']
ARA NORM  00022.wav
15412 22050
Arabic Speech corpus preprocessing (2 / 6772) - 'E:\Lectures\Masters\FYP Masters\arabic-speech-corpus\slices\Traini
ng set\ARA NORM  0002\3.wav']
ARA NORM  00023.wav
14391 22050
Arabic Speech corpus preprocessing (3 / 6772) - 'E:\Lectures\Masters\FYP Masters\arabic-speech-corpus\slices\Traini
ng set\ARA NORM  0002\4.wav']
ARA NORM  00024.wav
10126 22050
Arabic Speech corpus preprocessing (4 / 6772) - 'E:\Lectures\Masters\FYP Masters\arabic-speech-corpus\slices\Traini
ng set\ARA NORM  0002\5.wav']
ARA NORM  00025.wav
```

Figure 3.3-6 Training set preprocessing

The audio and label files were of different length as they depended on the length of the audio file and transcription of the word, for this reason the next step had been to zero pad both the input arrays (audio features) and the output arrays (labels), but also with keeping the original input and output lengths in separate arrays.

The preprocessing returned 8 arrays, the_input, the_labels, input_length and label_length.

```
train_inputs = {'the_input': X_train,
           'the_labels': Y_train,
           'input_length': train_input_length,
           'label_length': train_label_length
}

test_inputs = {'the_input': X_test,
              'the_labels': Y_test,
              'input_length': test_input_length,
              'label_length': test_label_length ,
}
```

Figure 3.3-7 Dataset arrays after preprocessing

3.4 ARABIC SPEECH RECOGNITION

3.4.1 First experiment

Given their effectiveness in modeling sequential data, the first model has been chosen to be a RNN. The RNN takes the time sequence of audio features as input.

At each time step, the speaker pronounces one of 46 possible characters defined previously in the character map.

The output of the RNN at each time step is a vector of probabilities with 47 entries, where the i-th entry encodes the probability that the i-th character is spoken in the time sequence. (The extra character is an empty "character" used to pad training examples within batches containing uneven lengths.)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       (None, None, 161)         0
_________________________________________________________________
rnn (LSTM)                   (None, None, 200)         289600
_________________________________________________________________
bn_rnn_1d (BatchNormalizatio (None, None, 200)         800
_________________________________________________________________
time_distributed_7 (TimeDist (None, None, 47)          9447
_________________________________________________________________
softmax (Activation)         (None, None, 47)          0
=================================================================
Total params: 299,847
Trainable params: 299,447
Non-trainable params: 400
_________________________________________________________________
None
```

Figure 3.4-1First model summary

The sliced dataset contained a lot of silences which caused many problems in training so the silences has been removed from the dataset.

```
Training X shape: (3900, 97, 161)
Training Y shape: (3900, 30)
Number of unique words in the training set: 3157
Test X shape: (470, 66, 161)
Test Y shape: (470, 16)
Number of unique words in the test set: 434
```

Figure 3.4-2Information about the training and test data

The model was interrupted at 25 epochs of the planned 100, the training was done on the full training set (after removing the silences) and the validation was done on the full test set, the details can be viewed in the figures below.

```
Epoch 20/100
3900/3900 [==============================] - 110s 28ms/step - loss: 2.1828 - acc: 0.3010 -
val_loss: 16.1933 - val_acc: 0.0000e+00
Epoch 21/100
3900/3900 [==============================] - 112s 29ms/step - loss: 2.0088 - acc: 0.3215 -
val_loss: 16.1085 - val_acc: 0.0000e+00
Epoch 22/100
3900/3900 [==============================] - 109s 28ms/step - loss: 1.8605 - acc: 0.3490 -
val_loss: 19.5041 - val_acc: 0.0021
Epoch 23/100
3900/3900 [==============================] - 111s 28ms/step - loss: 1.7132 - acc: 0.3762 -
val_loss: 17.3116 - val_acc: 0.0021
Epoch 24/100
3900/3900 [==============================] - 111s 28ms/step - loss: 1.6154 - acc: 0.3890 -
val_loss: 16.0840 - val_acc: 0.0021
Epoch 25/100
3440/3900 [=========================>....] - ETA: 13s - loss: 1.4556 - acc: 0.4267

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
```

Figure 3.4-3Training of the first model

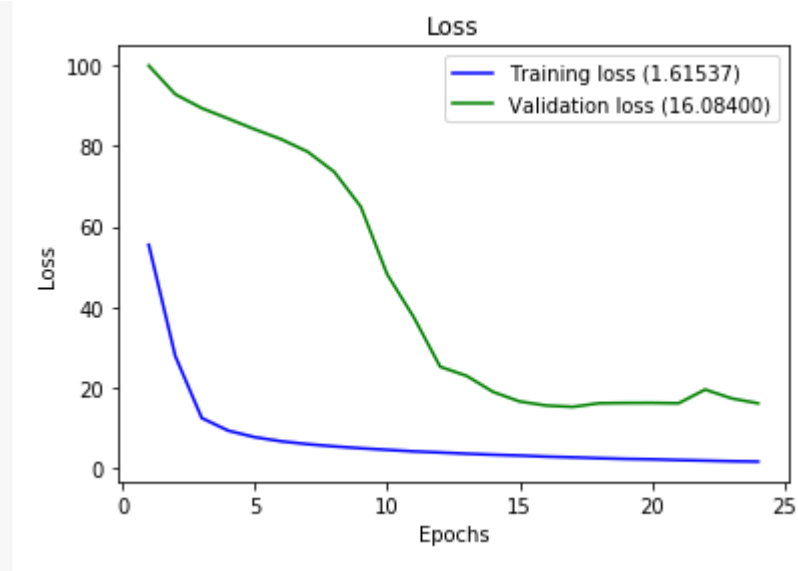

Figure 3.4-4 Train and Validation loss of the first model

The model was performing well on the training set but not so well on the test set, which suggested the model wasn't generalizing well and was going to over fit on the training data, which was behind the reason for stopping the training to try a different model.

3.4.2 Second Experiment

One shortcoming of conventional RNNs is that they are only able to make use of previous context. In speech recognition, where whole utterances are transcribed at once, future

context could help the recognition especially that how the phonemes are pronounced depend on the word itself.

For this reason the second model was chosen to be a bidirectional RNN (BRNN).

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       (None, None, 161)         0
_________________________________________________________________
bidirectional_5 (Bidirection (None, None, 400)         579200
_________________________________________________________________
time_distributed_5 (TimeDist (None, None, 47)          18847
_________________________________________________________________
softmax (Activation)         (None, None, 47)          0
=================================================================
Total params: 598,047
Trainable params: 598,047
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.4-5 Second model summary

The same training set and validation set as the previous experiment were used to train the model for 15 epochs.

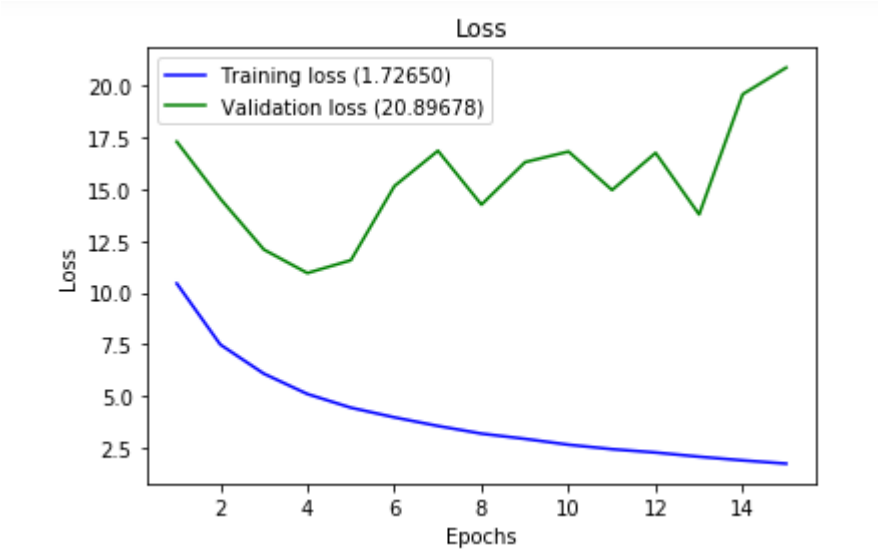

Figure 3.4-6Train and validation losses of the Second model

The model performed well on the training set, but after the 4th epoch the model started to over fit over the training set.

3.4.3 Third Experiment

Based on the first two experiments it has been decided to add a convolutional layer at the input of the model and to experiment with different type of RNN architecture, the summary of the model can be viewed in the *figure 3.4-7* .

Layer (type)	Output Shape	Param #
the_input (InputLayer)	(None, None, 161)	0
conv1d (Conv1D)	(None, None, 200)	354400
bn_conv_1d (BatchNormalizati	(None, None, 200)	800
rnn (GRU)	(None, None, 200)	240600
bn_rnn_1d (BatchNormalizatio	(None, None, 200)	800
time_distributed_13 (TimeDis	(None, None, 47)	9447
softmax (Activation)	(None, None, 47)	0

Total params: 606,047
Trainable params: 605,247
Non-trainable params: 800

Figure 3.4-7 Third model summary

The model has been trained with the same dataset for 10 epochs.

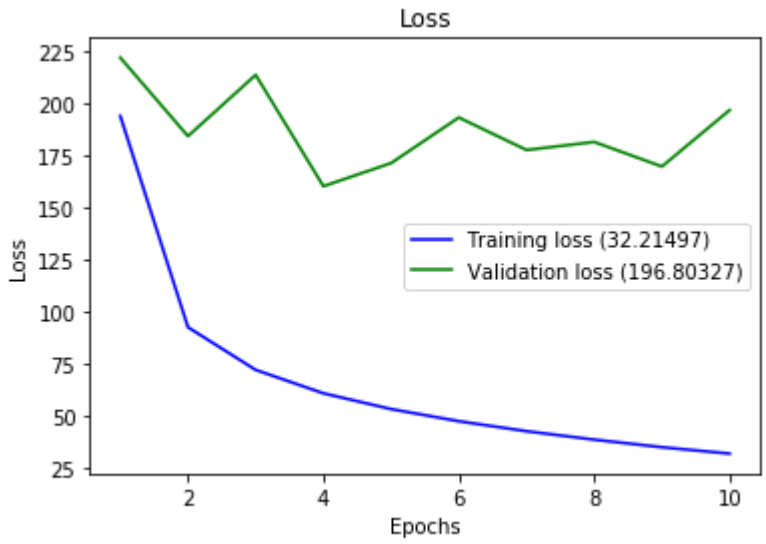

Figure 3.4-8Training and validation losses of the third model

Again, the model performed well on the training set but wasn't able to generalize well as it has been noticed on the validation loss to be fluctuating and not decreasing with the training loss.

3.4.4 Fourth Experiment

Based on the previous experiments it has been decided to try with an architecture similar to the one in the third experiment, the summary of the model can viewd in the figure below.

Layer (type)	Output Shape	Param #
the_input (InputLayer)	(None, None, 161)	0
layer_1_conv (Conv1D)	(None, None, 200)	354400
conv_batch_norm (BatchNormal	(None, None, 200)	800
rnn_1 (GRU)	(None, None, 250)	338250
bt_rnn_1 (BatchNormalization	(None, None, 250)	1000
final_layer_of_rnn (GRU)	(None, None, 250)	375750
bt_rnn_final (BatchNormaliza	(None, None, 250)	1000
time_distributed_1 (TimeDist	(None, None, 47)	11797
softmax (Activation)	(None, None, 47)	0

Total params: 1,082,997
Trainable params: 1,081,597
Non-trainable params: 1,400

Figure 3.4-9 Fourth model summary

Training of the model with the segmented data generated a lot of errors related to the CTC implementation of TesorFlow, so it has been decided to try with the full phrases as input to the model and was trained for 25 epochs, using the full 100 samples from the test set for validation.

```
Training X shape: (1813, 1557, 161)
Training Y shape: (1813, 461)
Number of unique phrases in the training set: 1813
Test X shape: (100, 1235, 161)
Test Y shape: (100, 303)
Number of unique phrases in the test set: 100
```

Figure 3.4-10 Training and validation data for the fourth model

The model performed well, which was encouraging to investigate more about the performance of the model, and testing on the combination of 4 test samples (44, 57, 100, 21) combined into one set gave the following results, and for comparison reasons the true transcription versus the predicted transcription were put side to side.

```
Test set example, samples 44,57,100,21

True transcription:

kamA tam~a taHsiynu wAjihAti lt~anaq~ul   wAxtiyAri wasA}ili ln~aqli lmunAsibapi bi$aklin kab
iyr  minhA >aqmi$apun wa>adawAtun maEdaniy~apun waxa$abiy~apun   waqinAnun blAstiykiy~apun wa
zujAjiy~apun wa>awrAqu SuHuf wayumkinuka lHuSuwlu EalY taTbiyqAtin lilt~adriybAti l>asAsiy~ap
i maj~Anan jamEu lmu&ana~^i lsa~Alimi mi^la   fAzat <iHdY lTa~AlibAti fiy musAbaqapi lqirA'At
i lqur>Aniya~pi
--------------------------------------------------------------------------------
Predicted transcription:

kaAmaA tarm~a doaAHosiyn waAjihaAti Alt~anaq~u o   waAxotiyaAri wasaA}ili Al~aqori AlmuwlaAsi
bati bi$akolay mokabiyro N minohaA aqomi$apu lw<adawAtN luEiadaniy~apN waxa$ibiy~atK    waqiy
naAnu lolaAsoiykiy~apN wzujaAjiy adpu Alowa>aworaAquSuHa ofo dm wayumobkilu kali Hu Suwlu Eal
Y tauDobi y qaAtK lilt~adolAlyibaAti Alo<i>isaAsiy~ap malj~aAnAF  DjamoE Alomu&an~a^i Als~aAl
imiymizola     faAza to<iaHoda taAlibaAti fiy muSaAbaqai AloqilaA'aAti Aloqurao|nniy~api _F
--------------------------------------------------------------------------------
```

Figure 3.4-11 True and predicted transcriptions for test samples (44, 57, 100, 21)

Getting these encouraging results called for a true metrics of ASR systems Word error rate (WER).

WER is derived from the Levenshtein distance, working at the word level instead of the phoneme level. The WER is a valuable tool for comparing different systems as well as for evaluating improvements within one system.

$$\boldsymbol{WER} = \frac{\boldsymbol{number\ of\ substitutions + number\ of\ deletions + number\ of\ insertions}}{\boldsymbol{number\ of\ words\ in\ the\ reference}} \qquad (3.1)$$

A python script was adapted to compute the WER for the model, the script took two inputs the hypothesis (predicted transcription by the model) and a reference (the true transcription), for this the model was fed the total test set and the output values were saved into a list, the true transcription was taken directly from the dataset, this gave two lists of 100 entry each, one for the predicted transcription and one for the true transcription, the WER was computed for each sample of the data set and were stored in an array of 100 entries, the final WER for the model on the test set was computed by averaging the WERs of each single sample.

This gave a WER of 0.3523 i.e. a word accuracy W_{acc} = 64.77%.

GENERAL CONCLUSION

Throughout the period of working on this project an end to end Arabic speech recognition system using recurrent neural networks was implemented; an unknown and untested Arabic speech corpus for automatic speech recognition was explored, segmented and used to train the different models. A final model was selected from the results of testing various models; this model showed good results which encouraged to go further and use a real metric for speech recognition (word error rate) which gave fairly good results of 64.77% word accuracy.

The obtained results proved the effectiveness of RNNs in dealing with temporal Data and in ASR in particular.

Buckwalter Arabic transliteration proved to be an effective representation of the Arabic text versus other Romanization schemes that add morphological information not expressed in Arabic script.

Audio features representation is an important step in ASR and finding a good representation of audio is one of the various challenges not addressed by this work.

As future work, adding a language model to increase the accuracy of the system could be explored, another method to improve the performance of the system would be better audio feature representations as it would allow to build performant speech recognition systems through general transcribed audio datasets that aren't created for the purpose of speech recognition; this could be especially helpful in languages where there is a lack in specialized datasets.

R E F E R E N C E S

[1] D. Yu and L. Deng, Automatic Speech recognition, a deep learning approach, Springer London Heidelberg New York Dordrecht, 2015.

[2] Xuedong, Acerox and Hsiao-Wuen, Spoken Language Processing, A Guide to Theory, Algorithm and System Development, Parentice Hall PTL, 2001.

[3] Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[4] Escur, "Exploring Automatic Speech recognition with tensorflow," Masters Thesis ,Barcelona, 2017.

[5] "Perceptron, Wikipedia," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Perceptron.

[6] Hinton, Deng and Yu, "Deep Neural Networks for Acoustic Modeling" *IEEE Signal Process. Mag*, 2012.

[7] Graves, AbdelRahman and Hinton, "SPEECH RECOGNITION WITH DEEP RECURRENT NEURAL NETWORKS" in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013.

[8] Graves and Al, "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks," in *23rd international conference on Machine learning*, Pittsburgh, Pennsylvania, USA, 2006.

[9] Hochreiter and Schmidhuber, "Long short-term memory," *Neural Computation*, 1997.

[10] Sutskever, "TRAINING RECURRENT NEURAL NETWORKS" , Phd Thesis, University of Toronto, 2013.

[11] "Convolutional neural networks," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network.

[12] Wyse and Lonce, “Audio spectrogram representations for processing with Convolutional Neural Networks,” Proceedings of the First International Workshop on Deep Learning and Music joint with IJCNN., 2017.

[13] Palaz and al, “Estimating phoneme class condi-tional probabilities from raw speech signal using convolutional neural networks,” in *Interspeech,* 2013.

[14] Tuske and al, “Acoustic modeling with deep neural networks using raw time signal for LVCSR,” in *Interspeech*, 2014.

[15] Sainath and al, “Learning the speech front-end with raw waveform CLDNNs” in *Interspeech*, 2015.

[16] Meseguer and Alcaraz, “Speech Analysis for Automatic Speech recognition” Norwegian University of Science and Technology, 2009.

[17] “Place theory of hearing,” 2018. [Online]. Available: https://en.wikipedia.org/wiki/Place_theory_(hearing).

[18] “Temporal Theory for human hearing,” 2018. [Online]. Available: https://en.wikipedia.org/wiki/Temporal_theory_(hearing).

[19] Tzanetakis, “Manipulation, analysis and retrival systems for audio analysis” , Phd Thesis, Princeton University, 2002.

[20] “What is Python? Executive Summary,” 2018. [Online]. Available: https://www.python.org/doc/essays/blurb/.

[21] “About TensorFlow,” 2018. [Online]. Available: https://www.tensorflow.org/.

[22] “About Keras,” 2018. [Online]. Available: https://keras.io/.

[23] “Jupyter notebook,” 2018. [Online]. Available: http://jupyter.org/.

[24] Halabi, “Modern Standard Arabic Phonetics for Speech Synthesis” ,PhD Thesis, University of Southampton 2016.

A P P E N D I X

BUCKWALTER ARABIC TRANSLITERATION

The Buckwalter Transliteration is a strict transliteration of Modern Standard Arabic orthographical symbols using only 7-bit ASCII characters. It is used for representing exact orthographical strings of Arabic in email and other environments where the display of real Arabic script is impractical or impossible. There is a strict one-to-one mapping back and forth from UNICODE to Buckwalter Transliteration, without gain or loss of ambiguity. Arabic text in ASMO 449 and ISO8859-6 can also be translated into Buckwalter Transliteration (or UNICODE), but the reverse mapping is hindered by the lack of a couple of (rare) characters.

Name (Unicode name)	**UNICODE**	**Buckwalter**	**ISO 8859-6**	**ASMO 449**	**Windows 1256**	**Glyph**
hamza-on-the-line (Arabic letter hamza)	\u0621	'	C1	A	C1	ء
madda (Arabic letter alef with madda above)	\u0622	|	C2	B	C2	آ
hamza-on-'alif (Arabic letter aleph with hamza above)	\u0623	>	C3	C	C3	أ
hamza-on-waaw (Arabic letter waw with hamza above)	\u0624	&	C4	D	C4	ؤ
hamza-under-'alif (Arabic letter aleph with hamza below)	\u0625	<	C5	E	C5	إ
hamza-on-yaa' (Arabic letter yeh with hamza above)	\u0626	}	C6	F	C6	ئ
bare 'alif (Arabic letter alef)	\u0627	A	C7	G	C7	ا

baa' (Arabic letter beh)	\u0628	b	C8	H	C8	ب
taa' marbuuTa (Arabic letter teh marbuta)	\u0629	p	C9	I	C9	ة
taa' (Arabic letter teh)	\u062A	t	CA	J	CA	ت
thaa' (Arabic letter theh)	\u062B	v	CB	K	CB	ث
jiim (Arabic letter jeem)	\u062C	j	CC	L	CC	ج
Haa' (Arabic letter hah)	\u062D	H	CD	M	CD	ح
khaa' (Arabic letter khah)	\u062E	x	CE	N	CE	خ
daal (Arabic letter dal)	\u062F	d	CF	O	CF	د
dhaal (Arabic letter thal)	\u0630	*	D0	P	D0	ذ
raa' (Arabic letter reh)	\u0631	r	D1	Q	D1	ر
zaay (Arabic letter zain)	\u0632	z	D2	R	D2	ز
siin (Arabic letter seen)	\u0633	s	D3	S	D3	س
shiin (Arabic letter sheen)	\u0634	$	D4	T	D4	ش
Saad (Arabic letter sad)	\u0635	S	D5	U	D5	ص
Daad (Arabic letter dad)	\u0636	D	D6	V	D6	ض
Taa' (Arabic letter tah)	\u0637	T	D7	W	D8	ط
Zaa' (DHaa') (Arabic letter zah)	\u0638	Z	D8	X	D9	ظ

cayn (Arabic letter ain)	\u0639	E	D9	Y	DA	ع
(Arabic letter ghain)	\u063A	g	DA	Z	DB	غ
taTwiil (Arabic letter tatweel)	\u0640	_	E0	0x60	DC	ـ
faa' (Arabic letter feh)	\u0641	f	E1	a	DD	ف
qaaf (Arabic letter qaf)	\u0642	q	E2	b	DE	ق
kaaf (Arabic letter kaf)	\u0643	k	E3	c	DF	ك
laam (Arabic letter lam)	\u0644	l	E4	d	E1	ل
miim (Arabic letter meem)	\u0645	m	E5	e	E3	م
nuun (Arabic letter noon)	\u0646	n	E6	f	E4	ن
haa' (Arabic letter heh)	\u0647	h	E7	g	E5	ه
waaw (Arabic letter waw)	\u0648	w	E8	h	E6	و
'alif maqSuura (Arabic letter alef maksura)	\u0649	Y	E9	i	EC	ى
yaa' (Arabic letter yeh)	\u064A	y	EA	j	ED	ي
fatHatayn (Arabic fathatan)	\u064B	F	EB	k	F0	ً
Dammatayn (Arabic dammatan)	\u064C	N	EC	l	F1	ٌ
kasratayn (Arabic kasratan)	\u064D	K	ED	m	F2	ٍ
fatHa (Arabic fatha)	\u064E	a	EE	n	F3	َ

Damma (Arabic damma)	\u064F	u	EF	o	F5	ُ
kasra (Arabic kasra)	\u0650	i	F0	p	F6	ِ
shaddah (Arabic shadda)	\u0651	~	F1	q	F8	ّ
sukuun (Arabic sukun)	\u0652	o	F2	r	FA	ْ
dagger 'alif (Arabic letter superscript alef)	\u0670	`	(missing)	(missing)	(missing)	
waSla-on-alif (Arabic letter alef wasla)	\u0671	{	(missing)	(missing)	(missing)	

- It should be noted that in the corpus used for this work the Arabic letter Thaa' ث was mapped to '^' instead of 'v'.