T.R.

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

VOWEL CLASSIFICATION IN TURKISH LANGUAGE

BARIŞ AYYILDIZ

SUPERVISOR
PROF. YUSUF SINAN AKGÜL

GEBZE
2023

T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

VOWEL CLASSIFICATION IN TURKISH LANGUAGE

BARIŞ AYYILDIZ

SUPERVISOR
PROF. YUSUF SINAN AKGÜL

2023
GEBZE

This study has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering on 18/01/2023 by the following jury.

JURY

Member
(Supervisor) : Prof. Yusuf Sinan AKGÜL

Member : Prof. İbrahim Soğukpınar

ABSTRACT

This project aims to address the significant issue of speech disorders, particularly among children, by utilizing the latest advancements in deep learning technology. The objective is to build a model that can effectively classify eight different vowels in the Turkish language. This is a critical subset of the larger problem of speech disorders as proper pronunciation of vowel sounds is a fundamental aspect of speech.

To achieve this goal, the project will focus on developing a deep learning model that can accurately classify the eight distinct vowel sounds in the Turkish language. The model will be trained on a large dataset of speech samples, which will be used to teach the model to recognize and differentiate between the various vowel sounds.

In addition to the deep learning model, the project will also involve the development of an application that can accept live input from users and display the predictions made by the model on an International Phonetic Alphabet chart. This application will be user-friendly and easy to use for both speech therapists and patients, allowing for real-time assessment and feedback on vowel sound production.

The ultimate goal of this project is to provide a valuable tool for speech therapists and patients to identify and address speech disorders related to vowel sounds in the Turkish language. By accurately classifying vowel sounds, the application will aid in the diagnosis and treatment of speech disorders, ultimately helping to improve the speech and communication abilities of individuals affected by these disorders.

ACKNOWLEDGEMENT

For this important research idea and their lead I am grateful to my supervisor Prof. Yusuf Sinan Akgül and Research Assistant İlhan Aytutuldu. Also I want to thank to Kahraman Arda Kılıç and Burak Yıldırım for being a participant in this research.

Barış Ayyıldız

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol or Abbreviation	:	**Explanation**
IPA	:	International Phonetic Alphabet
CNN	:	Convolutional Neural Network
LSTM	:	Long short-term memory
GCP	:	Google Cloud Platform
SGD	:	Stochastic Gradient Descent
RGB	:	Red Green Blue

CONTENTS

Abstract iv

Acknowledgement v

List of Symbols and Abbreviations vi

Contents vii

List of Figures viii

List of Tables ix

1 Introduction 1

1.1 Success Criteria 1

2 Vowel Classification 2

2.1 Data Collection 2

2.2 Preprocessing 2

2.3 Working Environment 4

2.4 Building Model 4

2.5 Model Results 6

3 Building Application 8

3.1 Frontend 8

3.2 Backend 9

4 Deployment 12

4.0.1 Response Time 12

5 Conclusions 13

5.1 Evaluation of success criteria 13

Bibliography 14

Appendices 14

LIST OF FIGURES

2.1 Sound wave of the word ”çep”. 2
2.2 A spectrogram for the vowel ’a’. 3
2.3 A spectrogram for the vowel ’e’. 3
2.4 Architecture of the model. 5
2.5 Architecture of CNN. 6
2.6 Accuracy rate in 100 epoch. 6
2.7 Train and Validation Accuracy. 7
2.8 Confusion matrix. 7

3.1 The application. 8
3.2 Prediction of the each spectrogram. 10
3.3 Normal distribution. 10
3.4 Prediction results. 10

4.1 Virtual Machine. 12

LIST OF TABLES

2.1 Dataset Table 1 3
2.2 Dataset Table 2 4

1. INTRODUCTION

In this project, I developed a model that utilizes computer vision techniques, such as CNN's and LSTM networks, to classify vowels in the Turkish language.

I also created a web application that allows users to interact with the model by providing audio input through their microphone, making a prediction using the model, and displaying the prediction visually on an IPA chart.

1.1. Success Criteria

1. Collecting at least 6000 labelled data for 8 different vowels.
2. The system will reach at least 90% of accuracy rate in terms of classifying vowels.
3. In the application a response should be returned at most in 1 second for each request.

2. VOWEL CLASSIFICATION

2.1. Data Collection

Before we could build the model, we needed to have some data. Unfortunately, there was no labelled audio data for Turkish vowels available online. Therefore, Mr. Aytutuldu and I went to Kartal Dr. Lütfi Kırdar Hospital with six different participants and recorded them saying some three-letter words with a vowel in the middle.

Such as ”çat”, ”kes”, ”kıp”, ”çip”, ”koş”, ”sök”, ”suç”, ”küs” and so on.

We took audio recordings of them saying these words, and used an ultrasound device to record the movements of their tongues. This allowed us to have both audio and video data. However, the ultrasound device was not able to capture the audio, so we had to record the sound separately.

2.2. Preprocessing

First, I used an editor to combine the separate audio and video files. Then, I used Praat software to annotate the vowels in the audio files. In the figure below, you can see the sound wave for the word ’çep’

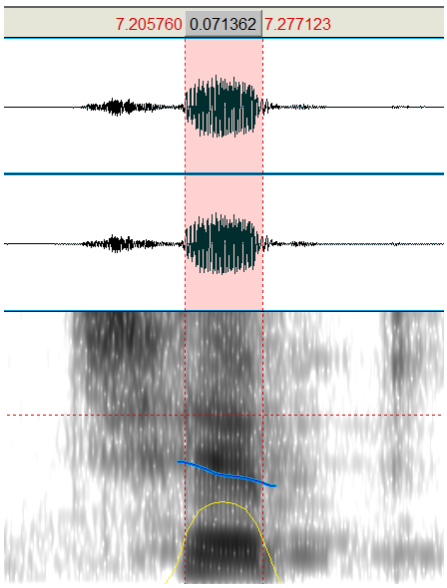

Figure 2.1: Sound wave of the word ”çep”.

As you can see, it is very easy to distinguish voiced letters from sound waves

with the human eye. This is basically what we want our model to do.

Using Praat software, I created intervals for all the vowels in all of the audio files, and used these intervals to create spectrograms for the audio data."

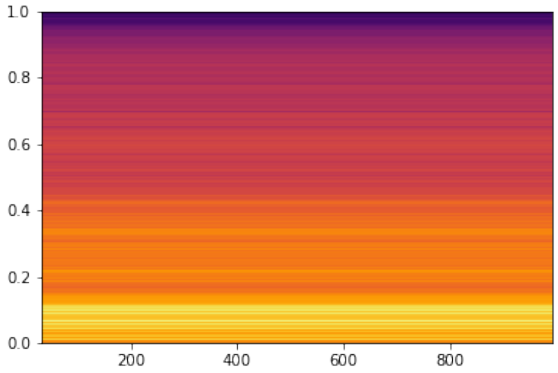

Figure 2.2: A spectrogram for the vowel 'a'.

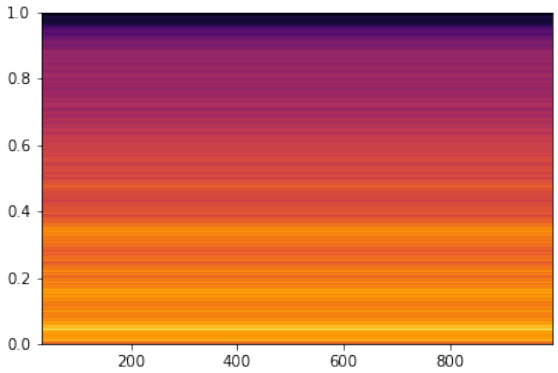

Figure 2.3: A spectrogram for the vowel 'e'.

After that, I used these intervals to collect three frames from the video files for each vowel instance. Because the movement of the tongue is important to us in ultrasound videos. Here is some information about our dataset:

Table 2.1: Dataset Table 1

	Value
Number of participants	6
Number of audio and ultrasound files	66
Total duration	1h 6m 55s

Table 2.2: Dataset Table 2

Vowels	Number of Spectrograms	Number of Frames
a	156	468
e	191	573
ı	181	543
i	157	471
o	216	648
ö	125	375
u	264	792
ü	234	702

However, at this point I decided to build my model only using the spectrograms generated from the audio files. This is because the application can only take audio input from the user.

2.3. Working Environment

I primarily used Google Colab connected to Google machines as my working environment. I used the open source libraries Tensorflow and Keras in Python to build my model.

2.4. Building Model

The model uses consecutive CNN and LSTM layers, with the SGD optimizer and a learning rate of 0.01. It runs for 100 epochs and uses the Categorical Cross Entropy loss function. Here is the architecture of the model:

The model consists of a convolutional layer wrapped in a Time Distributed layer, followed by a bidirectional LSTM layer and two fully connected dense neural network layers. The output of the model is an array of 8 elements. Each value represents the probability of a vowel sound being the actual result.

Our spectrograms has the width and the height of 96 pixels and has 3 RGB channels. That's why CNN has the input shape of (96,96,3). After that there is a convolutional layer with the kernel size of 7x7 with 128 features. Then there is another convolutional layer with the kernel size 3x3 with 64 features. Then it does max pooling with the padding of 2x2, so the size drops to 48x48 and then it does batch normalization. It basically does the same operation until it gets the shape of (1,1,32). Then the LSTM layer is added.

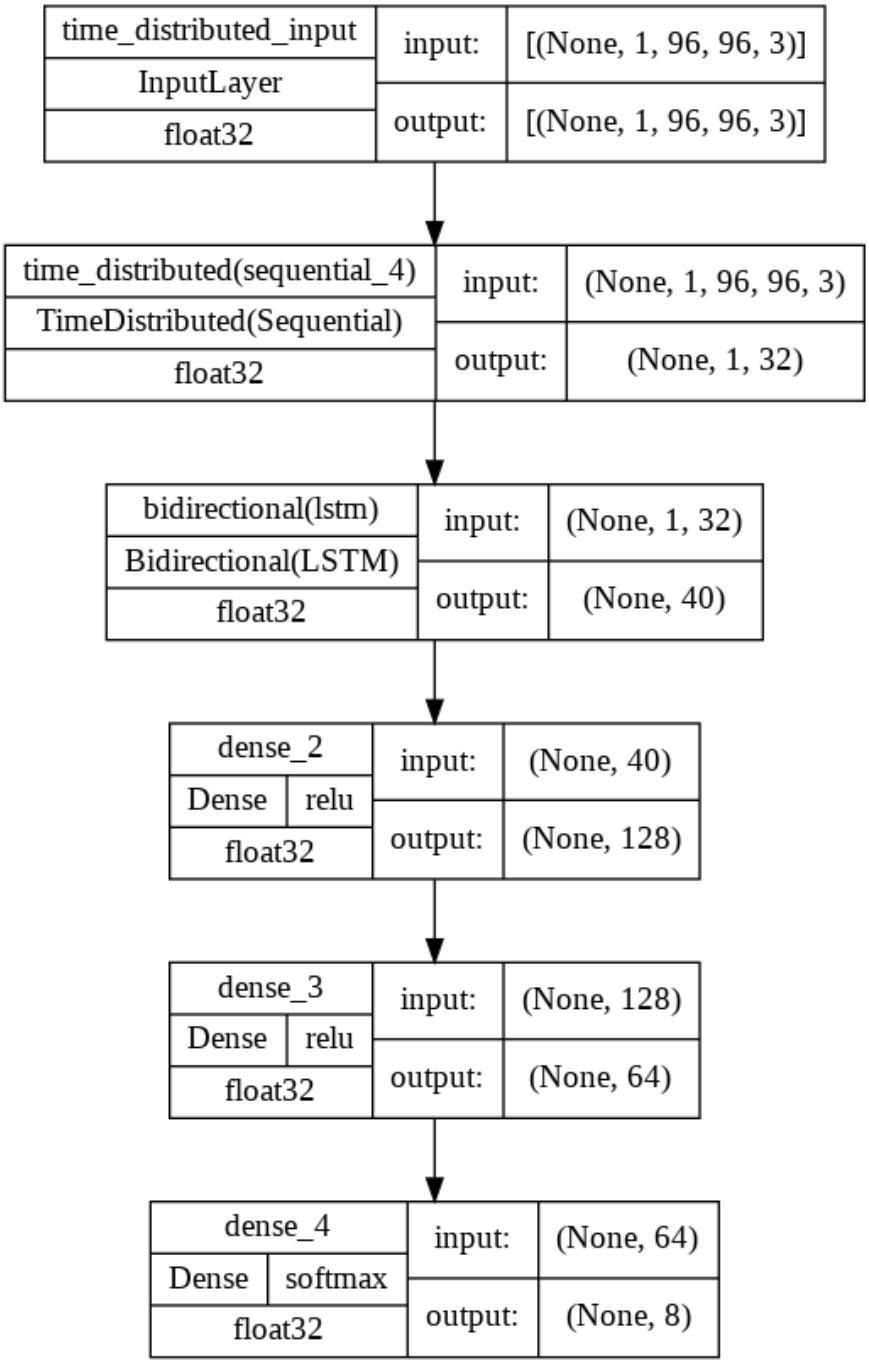

Figure 2.4: Architecture of the model.

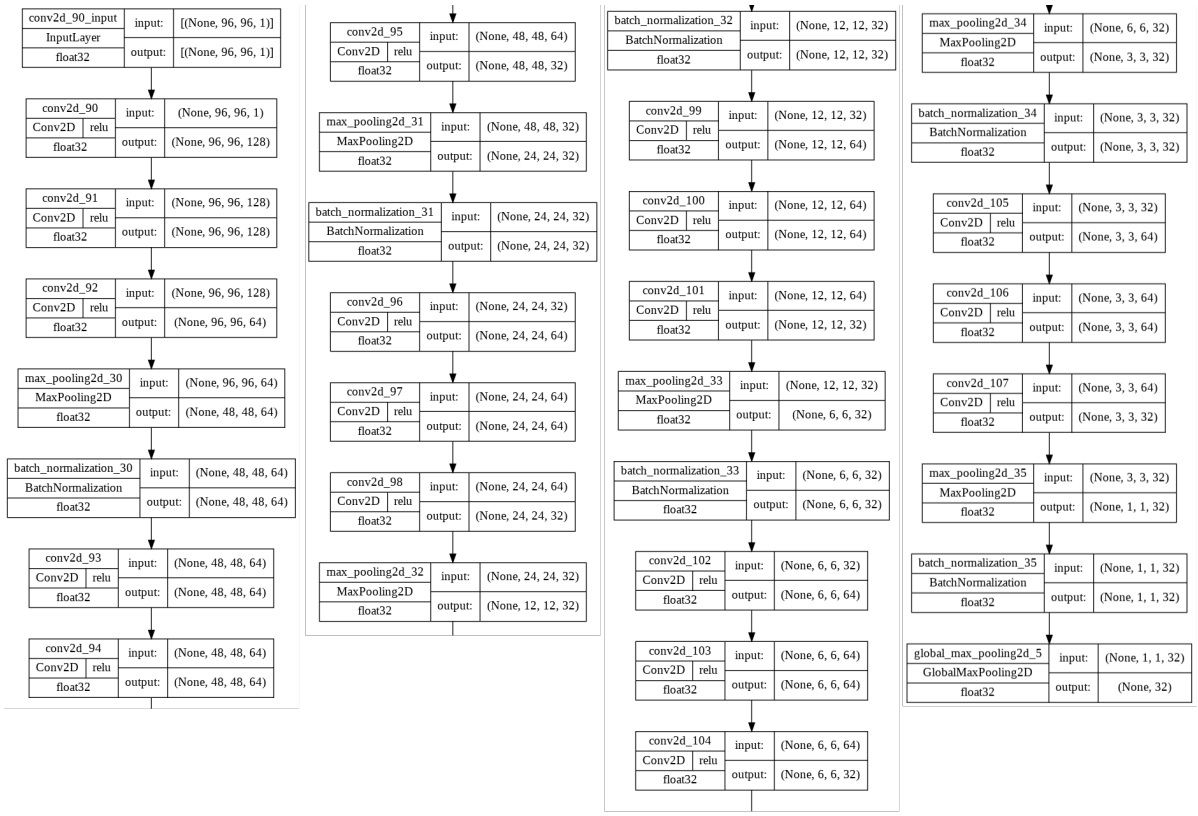

Figure 2.5: Architecture of CNN.

2.5. Model Results

Our model achieved an accuracy rate of 90%, surpassing the target I have set. The model gets over 95% of validation accuracy in 100 epoch. Here are the results:

```
Epoch 97/100
35/35 [==============================] - 5s 153ms/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 0.1759 - val_accuracy: 0.9548
Epoch 98/100
35/35 [==============================] - 5s 153ms/step - loss: 0.0189 - accuracy: 0.9937 - val_loss: 0.2082 - val_accuracy: 0.9523
Epoch 99/100
35/35 [==============================] - 5s 153ms/step - loss: 0.0218 - accuracy: 0.9955 - val_loss: 0.1956 - val_accuracy: 0.9497
Epoch 100/100
35/35 [==============================] - 5s 153ms/step - loss: 0.0165 - accuracy: 0.9955 - val_loss: 0.2261 - val_accuracy: 0.9523
<keras.callbacks.History at 0x7f8a5617bdd0>
```

Figure 2.6: Accuracy rate in 100 epoch.

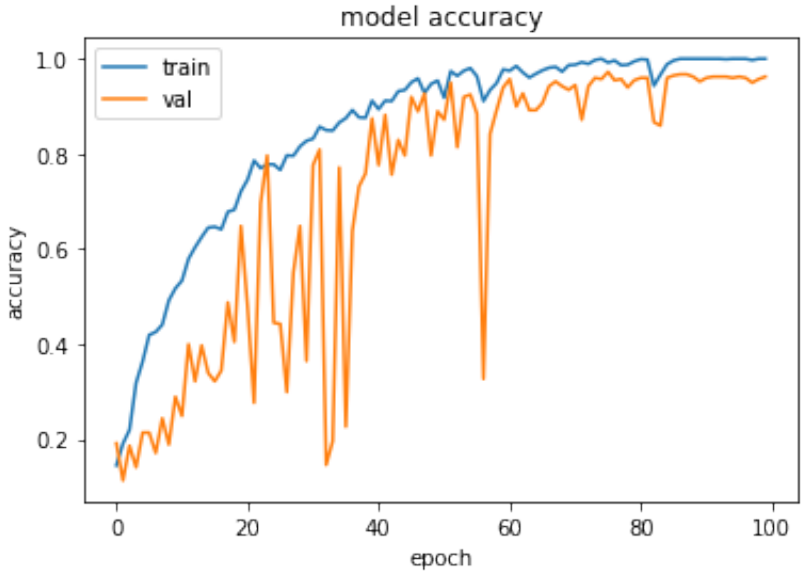

Figure 2.7: Train and Validation Accuracy.

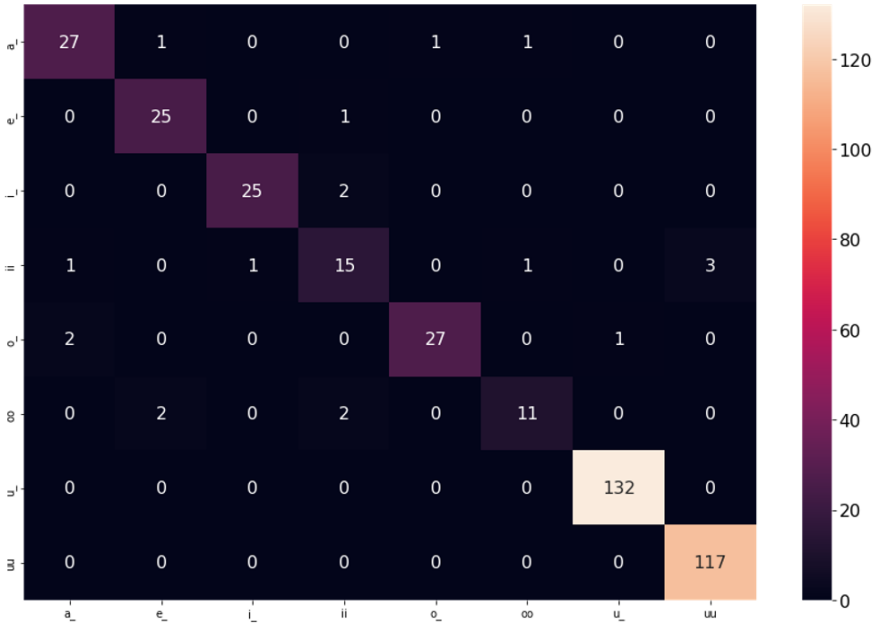

Figure 2.8: Confusion matrix.

3. BUILDING APPLICATION

3.1. Frontend

I have tried many different technology stacks to build the frontend of the application. First I have implemented it with React.js library then I switched back to vanilla JavaScript. Because the application uses some external libraries that utilize imperative programming paradigm and since React.js is a declarative library I would have use React.js for nothing.

It is a single page application, it has an IPA chart and a button. By the state of the button the system listens the audio input from user's microphone or not.

If the sound level exceeds a threshold value the system starts recording the audio and stops that recording when the sound level drops below that threshold. With this way user does not have to manually press the button every time he/she wants to use it. After that it converts the .ogg file format that is generated by the browser to a .wav file. And sends the .wav file to the backend service. When it gets the response it shows the prediction on the IPA chart. Here is the pseudo code of the algorithm

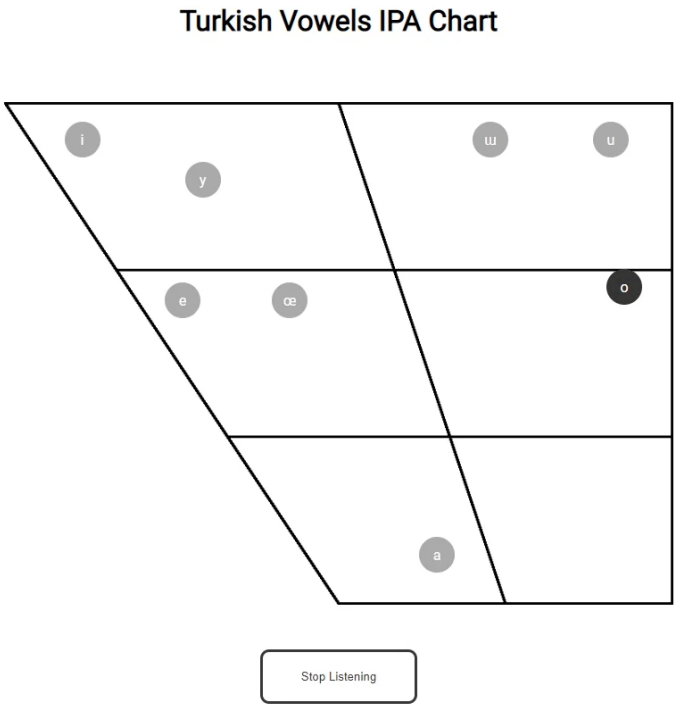

Figure 3.1: The application.

Algorithm 1 Recording algorithm

$recorder \leftarrow Recoder()$
$activeVowel \leftarrow null$
$threshold \leftarrow 0.2$
$recording \leftarrow false$
while true **do**
  $volume \leftarrow getVolume()$
  if $volume$ greater $threshold$ and $recording$ is false **then**
    $recorder.start()$
    $recording \leftarrow true$
  end if
  if $volume$ less $threshold$ and $recording$ is true **then**
    $recorder.stop()$
    $recording \leftarrow false$
    $wavfile \leftarrow convert(recorder)$
    $prediction \leftarrow ajax(wavfile)$
    $activeVowel \leftarrow prediction$
  end if
end while

3.2. Backend

I developed the backend service of the application using Python and the Fastapi library. I chose Fastapi for its simplicity and lightweight design. It made the implementation process straightforward and efficient.

I only wrote a single API. This API takes a .wav sound file from the user and converts it to a spectrogram. It then uses a trained model to predict which vowel the sound corresponds to.

The algorithm works as follows: the sound file is sent to the API as a request. The file is then divided into 30ms segments, as this is approximately the duration of a single sound. Each of these segments is converted to a spectrogram. All of the spectrograms are then fed into a trained model to make predictions.

Afterwards, the prediction values in the list are multiplied by a Gaussian distribution based on their position in the list. The prediction with the highest resulting value is returned. Because the middle values are multiplied by higher numbers, the information in the middle of the sound data becomes more significant.

I discovered through trial and error that the predictions in the middle tended to give more accurate results, and this is why I implemented this algorithm.

In the example below, the user has produced the sound of the letter 'e' and the sound file is 257ms long. Nine spectrograms are created from this sound file. As can be seen, the model made very nonsensical predictions at the beginning and end. However,

the 5th spectrogram predicted a probability of 99.1% that the sound was the letter ’e’. The predictions in this list were multiplied by the values in the Gaussian distribution, in order, as shown in Figure 3.3. This resulted in the values shown in Figure 3.4.


```
▼steps: Array(9)
  ▶0: (2) ['ö', 0.7945370078086853]
  ▶1: (2) ['o', 0.9825971126556396]
  ▶2: (2) ['o', 0.727928638458252]
  ▶3: (2) ['ö', 0.29397961497306824]
  ▶4: (2) ['e', 0.9916452169418335]
  ▶5: (2) ['ö', 0.3777584731578827]
  ▶6: (2) ['e', 0.785385251045227]
  ▶7: (2) ['ö', 0.4565572440624237]
  ▶8: (2) ['u', 0.8647940754890442]
   length: 9
```


Figure 3.2: Prediction of the each spectrogram.

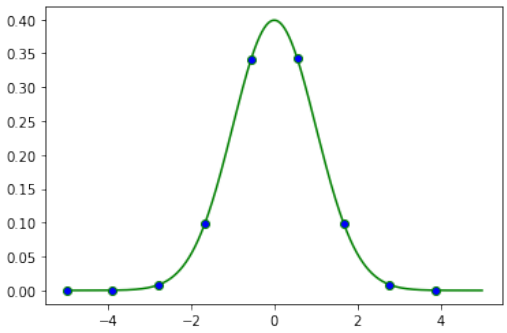

Figure 3.3: Normal distribution.


```
▼array:
    a: 0.04303236795873887
    e: 0.4536300559914604
    i: 0.03915877559996466
    o: 0.11517731466463124
    u: 0.04712191754322688
    ö: 0.15066096216185024
    ü: 0.037682039202411984
    ı: 0.013626546397522017
  ▶[[Prototype]]: Object
 prediction: "e"
```


Figure 3.4: Prediction results.

Algorithm 2 Prediction algorithm

Require: $soundFile$
Ensure: $max(predictions)$
  $audioFiles \leftarrow splitAudioFile(soundFile)$
  $spectrograms \leftarrow createSpectrograms(audioFiles)$
  $normalDist \leftarrow getNormalDist()$
  $counter \leftarrow 0$
  $predictions \leftarrow []$
  while counter<spectrograms.length() **do**
    $predictions \leftarrow predictions + predict(spectrograms[i])$
    $predictions[i] \leftarrow predictions[i] * normalDist[i]$
    $counter \leftarrow counter + 1$
  end while

However, unlike this example, the application often fails to make accurate predictions. While I am not completely sure, I have some ideas about what might be causing this.

First, we are not predicting every possible subgroup of 30ms of the sound. We are only predicting the first 30ms, then the next 30ms, and so on. The important data for the model may be in different ranges. I tried looking at every 30ms interval, but this meant that the model had to predict 70 spectrograms for a 100ms sound file. This significantly increased the response time, which is one of the success criteria for the project.

Second, there may not always be meaningful information in the middle. The model may sometimes make accurate predictions at the beginning or end of the sound file. However, my algorithm places more importance on the middle section.

Finally, the 30ms interval is a duration that I determined based on my observations. However, some sounds may take longer or shorter to pronounce. This could affect the model's ability to make accurate predictions.

4. DEPLOYMENT

I rented a virtual machine from GCP (Google Cloud Platform) and deployed both the frontend and backend code onto it. Initially, I had planned to send the frontend and backend code to separate servers, but I thought that by doing it this way I could get faster responses. I used Webpack to bundle the frontend code. And I also used nginx to start the backend server.

The virtual machine I used is the most powerful computer in the compute optimized category on GCP, as shown in the figure below.

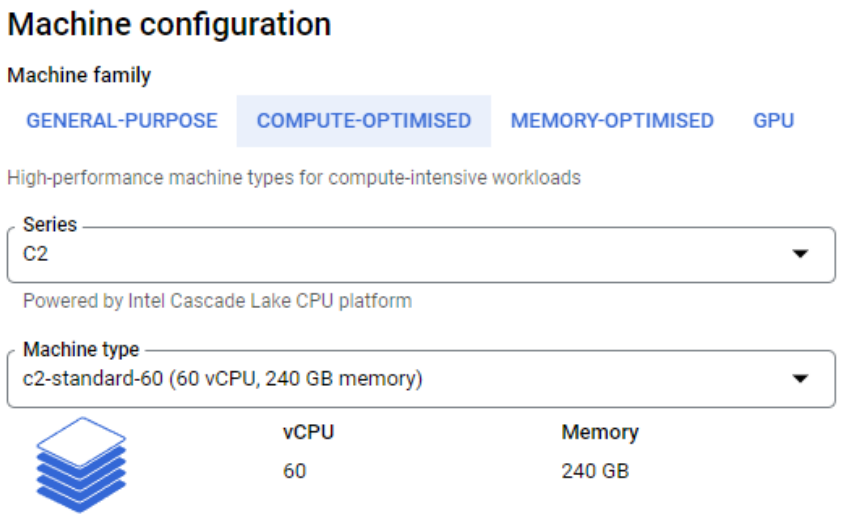

Figure 4.1: Virtual Machine.

You can test the application by clicking this link **`http://35.206.76.64/static/dist/index.html`**

4.0.1. Response Time

The response time is directly proportional to the length of the sound file, as expected. Words with three phonetic letters have a duration of between 120 and 390ms. This means that the server needs to predict an average of 3 to 12 spectrograms per request. As a result, the time required for each response varies between 850 and 2500ms.

5. CONCLUSIONS

In this project, I used computer vision techniques and convolutional neural networks (CNNs) and long short-term memory (LSTM) networks to classify vowels in Turkish language using spectrograms. The deep learning model I built had an success rate of over 90

However, no matter how I tried to analyze the sound data with different algorithms, the model did not produce good results on the real data coming from the application.

5.1. Evaluation of success criteria

1. I trained my model with only 1527 labeled data, but still managed to achieve the desired success rate
2. The system was able to achieve over 90% success in classification.
3. Response time can range from 0.8 seconds to 2.5 seconds.

I can say that I was unable to fully satisfy the first success criterion, satisfied the second success criterion, and partially satisfied the third criterion.

APPENDICES

- An Introduction to Neural Networks, Ben Kröse, University of Amsterdam, 1996
- An Introduction to Convolutional Neural Networks, Keiron O'Shea, Ryan Nash https://arxiv.org/abs/1511.08458
- Long Short-term Memory RNN, Christian Bakke Vennerød, Adrian Kjærran, Erling Stray Bugge https://arxiv.org/abs/2105.06756
- https://github.com/mattdiamond/Recorderjs