Capstone Project

Muhammad Khurram Chughtai

Machine Learning Engineer Nanodegree

June 23rd, 2020

Quran Reciter Identification (Speech Recognition) using Machine Learning

Definition

Project Overview

Speech recognition is an area that has been gaining traction for some time with evolving AI techniques. Many smart devices such as Alexa. Siri, etc. have successfully used speech recognition to aid in simple day to day activities. Speech recognition has been mainly done for the English language but there is limited information available for the Arabic language. Arabic is not only the official language of many countries around the world, but it is also the language of the Quran, the holy book for the religion of Islam.

In this project I apply a Bidirectional Long Short Term Memory (BiLSTM) Recurrent Neural Network (RNN) on Arabic Quran recitations to identify the person reciting (or speaking) the Quran. The dataset used was the MP3 Quranic recitations from this website. I used the Python Keras library to create and train the BiLSTM model and to make predictions. This study was inspired by this paper [1].

The motivation to solve this problem is to show that ML techniques can not only work for the Arabic speaker identification, but these techniques can be further enhanced to solve other problems in the Arabic language. The reason to use Quranic recitation data for this project is because Quranic recordings are readily available on the internet, whereas Arabic speech samples are not as readily available. Before proceeding further, it is important to understand how the Quran is organized and its related terminology. Quran is organized into 114 chapters (or "Sura"s), and each Sura is subdivided into verses (or "Aya"s). There are a total of 6236 Ayas in the Quran.

Problem Statement

The goal is to apply machine learning (ML) to the Quran MP3 Aya recordings of 5 reciters to correctly identify the reciter, therefore exhibiting that ML techniques also work on the Arabic language. The following process was followed to achieve this goal:

1. Download the Aya MP3 files for each reciter.
2. Preprocess and select a subset of the MP3 Aya files to extract the audio feature(s) which are split into train/validation/test data.
3. Design the RNN to work on the time-series audio data.

4. Train the RNN classifier that can identify the reciter using training & validation datasets, and then make predictions using the test dataset.

Metrics

The metrics used to measure the performance of the model are common for binary classifiers.

Accuracy

Accuracy is the measure of closeness of measurements to the true value. It takes into account both the true positives and true negatives. In this case the higher the accuracy the higher the chances of correct reciter identification.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{True\ Positives + True\ Negatives + False\ Positives + False\ Negatives}$$

Precision

Precision is the measure of fraction of true positives among the true + false positives. In this case how many Ayas identified for a particular reciter are really from that reciter.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Recall

Recall is the measure of fraction of true positives among the true positives & false negatives. In this case how many Ayas identified for a reciter **from the** overall Ayas are really from that reciter.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

F1-score

F1-score is the measure of a test's accuracy. In this case higher F1-score means the RNN is identifying the reciter more accurately.

$$F1_{score} = 2 * \frac{Precicion * Recall}{Precision + Recall}$$

Confusion Matrix

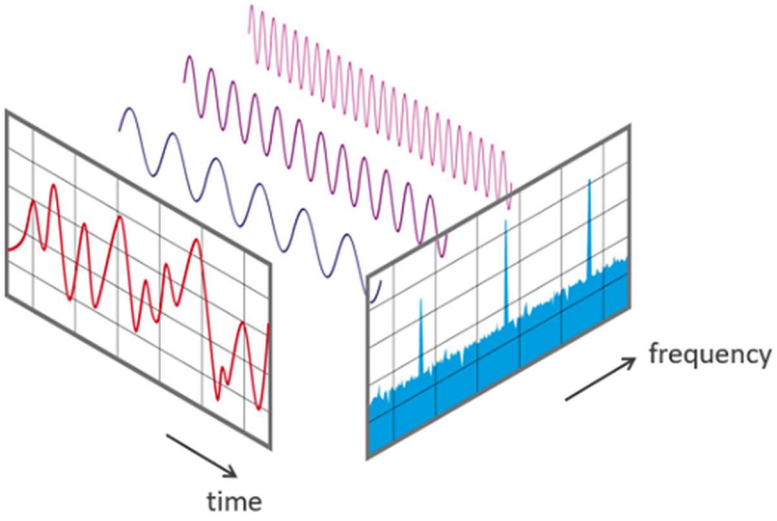
Confusion matrix is a specific table layout that helps visualize the performance of the classifier. In this case it helps show how many Ayas had the true vs predicted reciters.

Analysis

Data Exploration

What is Audio?

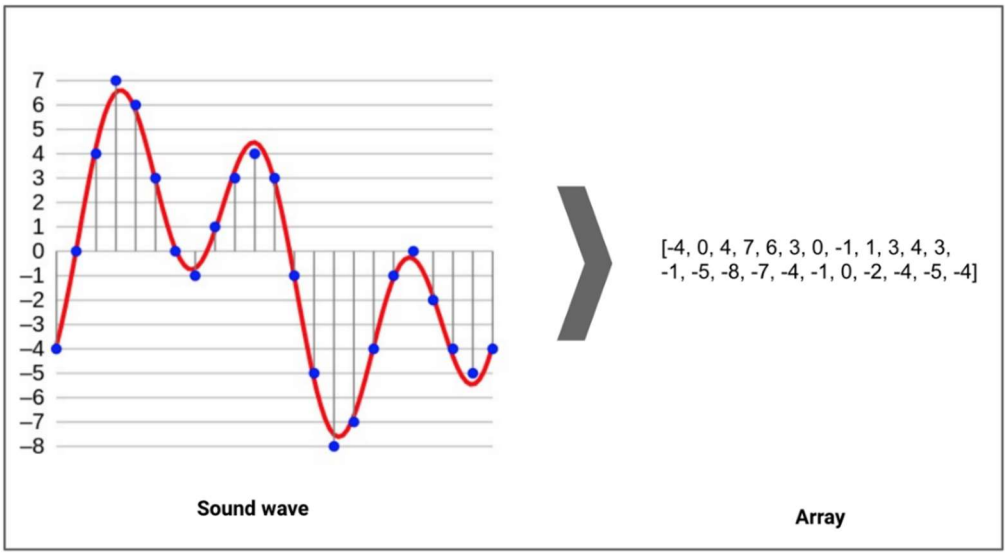
Audio or sound is represented as a signal with parameters such as time, amplitude, frequency, etc. An audio signal can be expressed as a function of time with Amplitude in "time domain" or Frequency in the "frequency domain".



Source

Digital Audio

Digital audio is sound recorded in or encoded into digital form. Sound waves are inherently analog in nature so the process of digitization is "digitally approximating" the sound waves. The analog-to-digital process results in some loss in quality which can be reduced but can never be zero. This process works by taking sound wave samples many times a second with each sample as a number. E.g. for CDs, samples are taken 44100 times per second (44.1kHz) each with 16-bit sample depth.



A sound wave, in red, represented digitally, in blue (after sampling and 4-bit quantization), with the resulting array shown on the right. Original © Aquegg | Wikimedia Commons

There are many popular digital audio formats:

- MP3 format

- WMA (Windows Media Audio) format
- WAV (Waveform Audio File) format

I will be using the MP3 format for this project. MP3 files can have different characteristics that need to be correctly accounted for in order to avoid unnecessary variations in the dataset for ML:

- Bit rate measured in kilobits per second (Kbps) – kbps is directly related to the sampling rate and quantization. The lower the Kbps the more information is discarded during analog to digital audio conversion. 128kbps is considered equivalent sound quality to what is heard on radio, and 160kps or higher is considered equivalent to CD sound quality.
- Sample rate (kHz) - The sample rate needs to be the same across all audio files so features are comparable.
- Channels – Stereo or Mono. The audio can be single or double channel. For ML this needs to be the same for all audio.
- Duration – The length of the audio in the MP3 file. The duration can be different because each Aya length is different, and how long a reciter takes to recite the Aya is also different. As will be seen that the audio duration for selected Ayas is anywhere from 1 to 185 seconds.

Exploratory Visualization

There are 6,236 verses in the Quran so each reciter has more or less as many MP3 files. For the most part I stuck with 64kbps since the voice quality is decent and the download size per reciter is in the ~500-800MB range. My research showed that the difference in kbps is not a problem.

Reciter name	Data Size	Files	MP3 Files
AbdurrahmaanAs-Sudais	584 MB	6351	6349
Ajami	1436 MB	6354	6350
Alafasy	825 MB	6352	6350
FaresAbbad	594 MB	6357	6353
Ghamadi	426 MB	6351	6349

Size of the downloaded ZIP files for each reciter

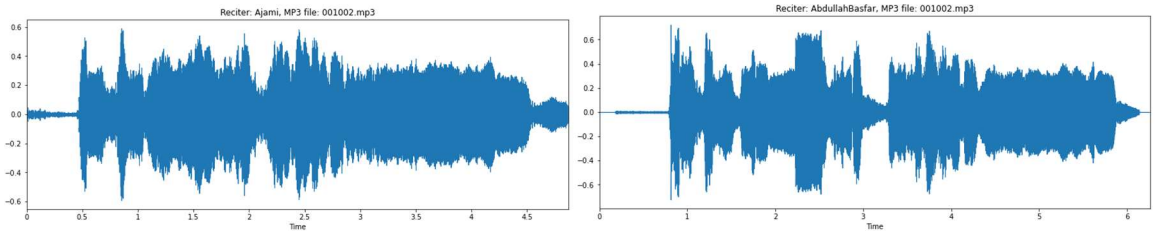
Here is a table that captures the MP3 characteristics for the first two Ayas. Librosa library will help normalize these differences among the reciters. Notice the differences in kbps, channels, and Aya durations for the reciters.

Reciter name/MP3 File	Data Size	MP3s	kbps	Ch	Mono/S	Duration(sec)
AbdurrahmaanAs-Sudais	60 KB	2				
001001.mp3	24 KB		65	2	stereo	3.0
001002.mp3	35 KB		64	2	stereo	4.4
Ajami	128 KB	2				
001001.mp3	49 KB		134	2	stereo	2.9
001002.mp3	79 KB		132	2	stereo	4.9
Alafasy	93 KB	2				
001001.mp3	48 KB		64	2	stereo	6.0
001002.mp3	45 KB		64	2	stereo	5.6
FaresAbbad	94 KB	2				
001001.mp3	47 KB		72	2	stereo	5.4
001002.mp3	47 KB		72	2	stereo	5.3
Ghamadi	53 KB	2				
001001.mp3	27 KB		40	1	mono	5.5
001002.mp3	26 KB		40	1	mono	5.3

MP3 file sizes and characteristics for first two Ayas for each reciter

One additional thing to note is the name of the MP3 File which also serves as the “label”. E.g. the file name 001001.mp3 for reciter “Ghamadi” means he recited Surah 001 Aya 001 in this MP3 file.

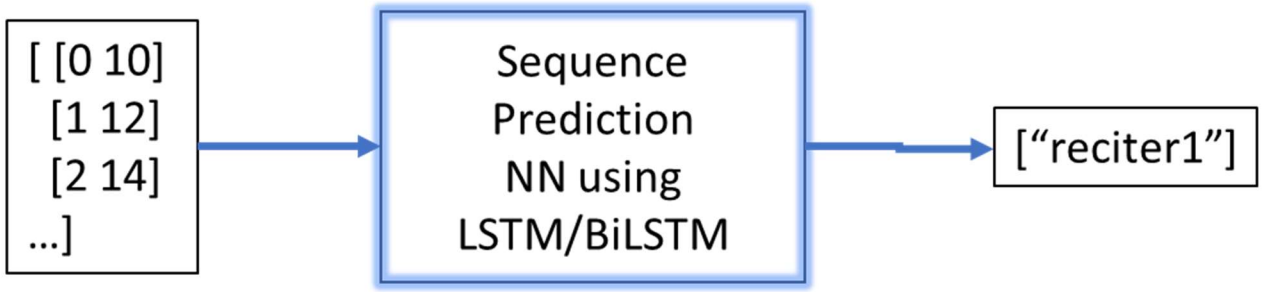
To visualize and appreciate the auditory differences among the reciters, here is the recitation of the second AyaInQuran (001002.mp3) for two reciters:



MP3 file plot for the same Aya for two reciters

Algorithms and Techniques

Audio data is a time series data as the sound wave is sampled many times a second. Even though we are using time series data, the classifier is not expected to predict what will happen at a future time. The goal for the classifier is to analyze the speech pattern over time and based on this pattern predict who the reciter (or speaker) is. This is still a supervised learning problem, but since we have a sequence (i.e. time series-based prediction) that makes it a different type of supervised learning problem compared to non-sequence problems like for example predicting house price based on number of bedrooms. The **order** of the sequence **itself** is important and must be preserved during the model training & predictions. Therefore, using a RNN made a lot of sense to solve this problem.



Sequence Prediction Model

The Long Short-Term Memory (LSTM) is a type of RNN designed for sequence-based problems. Normal NNs have layers of neurons that learn from the inputs, and feed forward the learning as one output. With RNNs we get additional “feedback” loops in the network model which act as memory or state elements, therefore giving the network the ability to learn from the ordered nature of the input sequences. This memory or state allows the RNN to get conditioned on the recent context in the input sequence, not just

the present values of the inputs. This capability helps the model with the sequence predictions. In the Keras library the LSTM layer can be created using "LSTM()" class. One important LSTM parameter is "return_sequences" which when set to "True" allows the last output to be a full sequence of outputs (each based on an input). i.e. one hidden state output for each input time step for the single LSTM cell in the layer. When set to "False" it only returns one output for the last input time step processed therefore ignoring the sequence. This parameter was set to "True" for the first two layers of the model. Another important LSTM parameter (like any other NN layer) is activation what was set to 'tanh' as per [1]. It made sense to use tanh since the input data consisted of both positive & negative numbers.

Bidirectional LSTMs (BiLSTM) are an extension of LSTMs that can further improve model performance for sequence-based problems. The BiLSTM connects two LSTM hidden layers of opposite directions to the same output; this way the network "learns" from both forward and backward sequence of the data. This additional step helps the network learn faster and provides additional context that is otherwise missing from the LSTM. In Keras any LSTM can be made Bidirectional by wrapping the LSTM layer call with "Bidirectional" e.g. "Bidirectional(LSTM(…))". One important parameter is "merge_mode" which is set to "concat" by default. It tells the layer on how to combine the outputs from the forward and backward RNNs. Other possible values for this parameter are "sum", "mul", "ave", None. I got good results with the default of "concat" so I didn't need to experiment with other values.

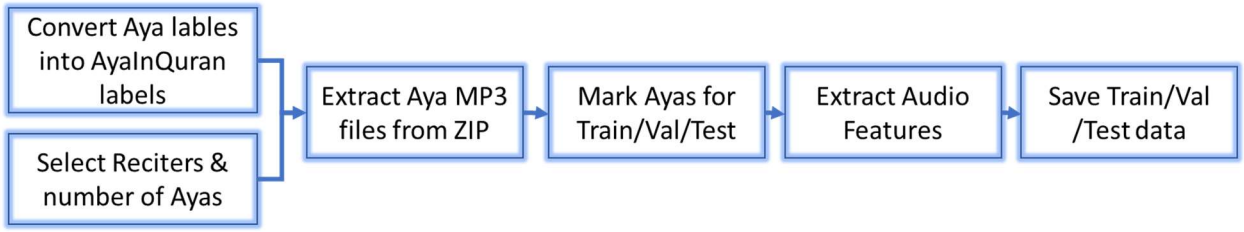
Benchmark

One benchmark was that the trained model identified the correct reciter with high accuracy. Besides this the benchmarks for accuracy/precision/recall/F1 score described in the [1] were also compared to what I was able to achieve.

Methodology

Data Preprocessing

Here is a visual diagram of the data preprocessing pipeline:



Convert Aya labels to AyaInQuran labels: As described earlier the MP3 file name also serves as the "label". E.g. the file name 001001.mp3 for reciter "Ghamadi" means he recited Surah 001 Aya 001 in this MP3 file. This naming convention was somewhat problematic as it did not facilitate the inquiry "give me only 12 Ayas for each reciter" because the naming was on a Sura boundary. E.g. to get 12 Ayas, I could get 7 Ayas from the first 001 Sura, but then had to look for another Sura that had only 5 Ayas. To make the data fetching and processing easier, the Sura+Aya label was converted to a global Aya numbering system (i.e. AyaInQuran) as follows:

- The first Aya in the first Sura was given the AyaInQuran label of “0”. i.e. 001001 -> 0
- The first Sura has 7 Ayas so the last Aya was given the AyaInQuran label of “6”, i.e. 001007 -> 6
- The second Sura’s first Aya was given the AyaInQuran label of “7”, i.e. 002001 -> 7
- Continuing in this manner, the last Sura’s last Aya was given the AyaInQuran label of “6235”, i.e. 114006 -> 6235
- This makes up the total of 6236 Ayas with AyaInQuran label ranging from 0 to 6235
- The AyaInQuran conversion was done using this info. [4]
- This made it easy to query any number of Ayas without worrying about which Sura had which Aya.

Select Reciters & number of Ayas: 5 reciters were selected and 3000 Ayas per reciters were chosen. At a certain point one of the reciters had to be changed to a different reciter due to the unusually long durations of the Ayas. The number of Ayas also changed over time as I learnt more about the data & trained the RNN.

Extract Aya MP3 files from ZIP: As mentioned earlier the reciter files downloaded were in ZIP format. I extracted only the Aya MP3 files that were needed (3000 in this case) & chose not to extract all 6236+ MP3 files. This also helped limit the MP3 characteristic analysis & audio extraction to only these files. After extraction the MP3 characteristics of these files were saved in a Pandas DataFrame.

As described earlier the MP3 files were of various characteristics that needed to be “normalized” for all the reciters. Here is how this was achieved:

- Bit rate measured in kilobits per second (Kbps) – The Librosa library doesn’t produce any different results for the different kbps MP3 files, so nothing needed to be done to account for differences in the kbps.
- Sample rate (kHz) - The sample rate needs to be the same across all audio files so features are comparable. Librosa can be given the sample rate as an input and the default is "22050". **In this project Librosa library helps sample the audio at the default sampling rate of “22050”**.
- Channels –The channels need to be the same for all the audio data. **In this project Librosa library helps read the audio with “mono” channel**.
- Duration – The duration, as will be seen later, is anywhere from 1 to 185 seconds for the Ayas. **In this project 3 second audio duration was chosen, and for Ayas less than 3 seconds, Librosa library helped pad the duration to 3 seconds**. The choice of 3 seconds came from [1].

Mark Ayas for Train/Val/Test: [1] used train/validation/test split of 60%/20%/20%, so I chose to use the same.

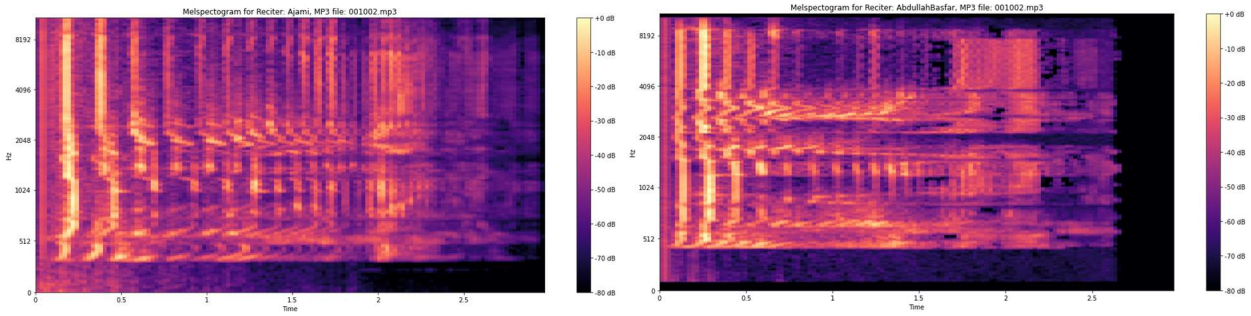
Audio Feature Extraction

Audio signals consist of many features. We need to extract the features relevant to the problem at hand so we can analyze them. Although many features were considered, the following two features were given primary focus due to the reasons noted.

Mel-spectrogram

My research turned up [3] that used Mel-spectrogram successfully to classify music genres. We can transform the audio signal from time-domain into frequency-domain using Fourier Transform to get frequency-based features (or "spectral" features). The MP3 files were converted into spectrograms which is a visual representation of spectrum of frequencies over time. A regular spectrogram is squared

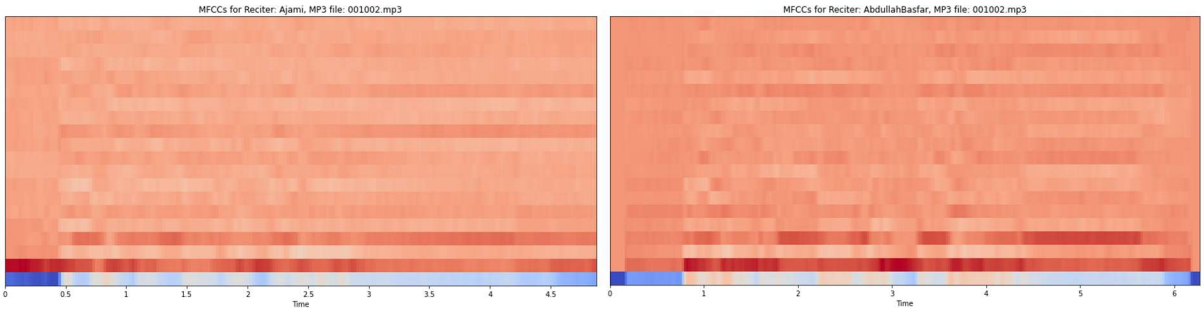
magnitude of the Short-Term Fourier Transform (STFT) of the audio signal. This regular spectrogram is squashed using Mel scale to convert the audio frequencies into something a human is more able to understand. The Librosa library can convert the audio file directly into a Mel-spectrogram. The most important parameters used in the transformation are — window length which indicates the window of time to perform Fourier Transform on and hop length which is the number of samples between successive frames. The typical window length for this transformation is 2048 which converts to about 10ms, the shortest reasonable period a human ear can distinguish. I chose hop length of 512 which is the default for Librosa. Furthermore, the Mel-spectrograms produced by Librosa were scaled by a log function. This maps the sound data to the normal logarithmic scale used to determine loudness in decibels (dB) as it relates to the human-perceived pitch. As a result of this transformation each 3-second of audio gets converted to a Mel-spectrogram of shape — 130, 128. As seen below, different reciters have noticeable differences in their Mel-spectrogram.



Mel-spectrogram for two reciters for the same Aya

Mel-Frequency Cepstral Coefficients (MFCCs)

The Mel frequency cepstral coefficients (MFCCs) of a signal are a small set of features (usually about 10–20) which concisely describe the overall shape of a spectral envelope. It models the characteristics of the human voice. The first 13 of these were used in the [1] that I was inspired by, so I went along with 13 MFCCs.



MFCCs for two reciters for the same Aya

Save Train/Val/Test data: The train/val/test data was saved for later using NumPy's "savez" method.

Implementation

The first step was to get Tensor flow running on my local machine which had GPUs. This took some trial and error as the latest Tensor flow version was not compatible with my GPU's CUDA version. After some online research and experimentation I was able to make this work. I also had to setup Anaconda Python & Python libraries & Jupyter Notebook on my machine. I also learnt the hard way that I could only train in one Jupyter Notebook using the GPUs. If I tried to start training in another Notebook it would stop at some random point giving an error message that looked as if I had a syntax error in my code.

I started developing Python code in Jupyter Notebooks for this project, but soon had to take a step back. It was a lot easier to write the code in a code editor which facilitates the writing task much better as compared to Jupyter Notebook. It was also easier to test the code outside Jupyter Notebook since the variables were always in an initial known state, whereas the variables keep getting updated inside the Notebook as you go through iterations of the code. For this purpose, I created a "helpers.py" file which has >90% of my code. To get this code into the Notebook I experimented with various online recommended methods and finally settled with the "%load helpers.py" method in the Notebook which "dumps" all the code in a cell, and then this cell is executed which "loads" everything into the Notebook.

The audio data is a time series data which gets extracted as a 2-dimensional NumPy array as (features, time) by Librosa. When all this data is aggregated at the next level for all reciters, this results in a 3-dimensional NumPy array as (reciter, features, time). This posed a problem for me because all my research and course-work had been working mostly on 1-dimensional data. This also meant that I couldn't use Sklearn classifiers which operate on 2-dimensional arrays but not 3-dimentional. It took a lot of online research to get some basic understanding of what I needed to do, and I got it wrong most of the time when my RNN would not compile or train or give meaningful results after training. I also had to open couple of Udacity Knowledge tickets to get some help! (ticket1 & ticket2).

The next step was the classifier design. I am new to NNs so it took a lot of research to get a basic understanding of NN and a basic design for it. The classifier used consisted of two Bidirectional LSTM layers of 128 & 64 neurons respectively with 'tanh' activation. These are connected with LSTM layer with 32 neurons followed by a dropout and finally dense output layer with activation 'softmax'. The design of this RNN was inspired primarily by [1] and experimentation since I didn't have access to their exact RNN design. The classifier was trained with 60% training & 20% validation dataset, and tested with remainder 20% of dataset.

Although the dataset and RNN design were inspired by [1], there are following marked differences from it:

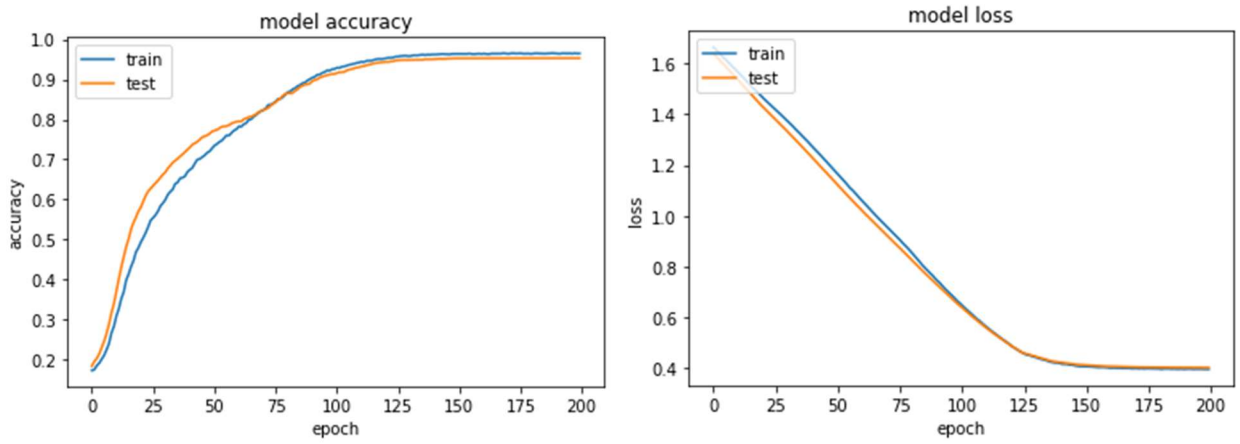
- I used different audio format (MP3 vs WAV)
- I used a different audio feature to train the RNN (Mel-spectrogram vs the 13 MFCC)
- I used audio length of 3 seconds (vs 1 & 2 & 3 seconds)

Refinement

The paper [1] was able to achieve >95% accuracy/precision/recall/F1 score on the test dataset. My initial BiLSTM RNN design would not give more than 20% accuracy on the test dataset. There were various improvements that I had to make to get to above 90%:

- Increase the dataset for training. I started with 200 Ayas for each reciter to finally end up with 3000 Ayas per reciter. More training data helped get better accuracy.
- Normalizing the extracted audio feature. I wasn't normalizing the features so I normalized them. This didn't make a difference in the results. I also researched online and it looks like normalization does not make a difference in RNNs but I need to research this more as all the examples and course work suggests normalizing the data. I chose to keep the data non-normalized.
- Change the activation from "sigmoid" to "tanh". A close look at the training & validation data showed that it mostly consisted of negative numbers so I changed to "tanh" activation which handles numbers from -1 to +1 as opposed to sigmoid's 0 to 1. Once I switched to "tanh" I didn't experiment further as I saw better results.
- Adding fully connected Dense layers between LSTM and output with Dropout layer in between. The extra Dense layers seem to degrade the metrics so I removed them.
- Changing Dropout rates. I played around with different dropout rates as high as 0.5 but eventually came down to 0.05 which worked well.
- Reducing number of neurons in each layer to reduce number of parameters to avoid overfitting.
- Changing batch size and number of epochs to improve test accuracy.
- Changing the audio duration distributions for Train/Val/Test. I initially started with 30 seconds of audio with a lot of padding for <30 second of audio which caused train/val & test duration distributions to be different. This helped a little but not a whole lot.
- Changing the audio duration to 3 seconds from 30 seconds. As I moved towards adding more and more data to get better accuracy, the 30 seconds audio feature size became a storage as well as training run time problem, so reduction to 3 seconds helped. Also with 30 seconds I hit a wall with NumPy savez method not being able to allocate ~12GB of file size.

Here are the results of the final run. As you can see the validation accuracy tracks the training very well, and the loss keeps coming down. Both plateau at around 150 epochs.

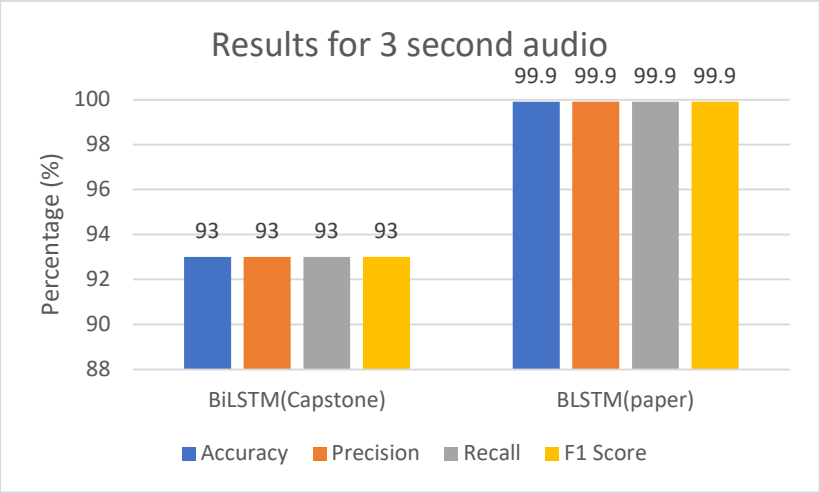


Model accuracy and loss curves through the epochs

Results

Model Evaluation and Validation

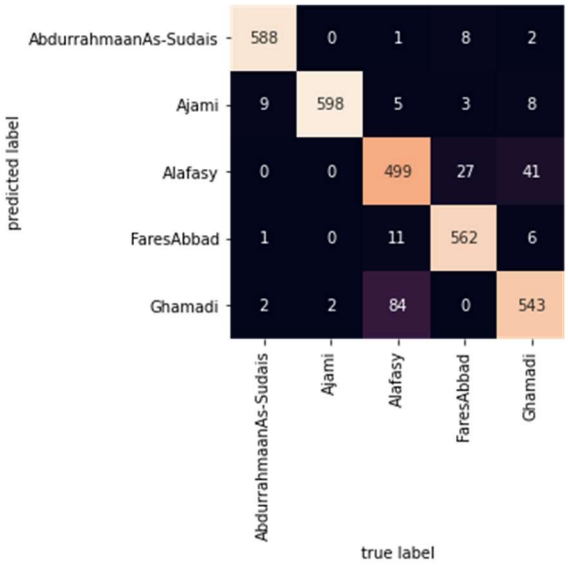
Here are the results of my Capstone project compared to [1]:



My goal was to have at least >90% of accuracy which was achieved. The results may be further improved by tweaking the RNN which requires a deeper understanding of RNNs.

Justification

Here is the confusion matrix that is reflective of the numbers above:



Confusion matrix for the test dataset

As can be observed, majority of the test Ayas are predicted correctly. The majority of the incorrect reciter identification is between Alafasy & FaresAbbad & Ghamadi who have very similar slow melodious recitation styles. As-Sudais's audio files have a lot of noise sound in them and his recitation style is also

different from the rest which is helping the model predict correctly. Ajami's recitation style is very different from the rest which is why his accuracy is the highest.

Conclusion

Free-Form Visualization

The confusion matrix shows that the RNN was able to adequately identify the reciter >90% of the time for the Arabic language, which meets the goal for this project. This proves that the English language speech recognition techniques can be applied to the Arabic language to a first order, further tweaking maybe required to further improve the results.

Reflection

The following steps summarize this project's process:

1. An initial problem was defined based on [1] and datasets were found. Not all data in [1] was available for download.
2. The data was downloaded and preprocessed.
3. The audio features were explored and the chosen feature was different from [1].
4. Tensor flow was installed to make sure GPUs would be used for training.
5. The RNN classifier was trained using the data multiple times with different reciters/durations/number of Ayas/batch sizes/epochs, until the classifier got to >90% test accuracy.

The most difficult steps were steps 3, 5, and 4 respectively. For step 3 I had to familiarize myself with 3-dimensional NumPy arrays and how to handle time series data. It was difficult to visualize the data, especially when it came to understanding the data distribution between train/test/validation sets, so I used the duration as a proxy & made sure the duration distribution was even between the sets. For step 5 the first challenge came with handling 3-dimensional dataset for the classifier where I had to determine how to start the classifier with 3-dimensional dataset and flatten it to 1-dimensional at the end. The second challenge came from lack of familiarity with NNs/RNNs and defining these in Keras library which resulted in errors that took a while to understand whether it was a syntax error or an array dimension issue. For step 4 it took a lot of online research to get it to work, and if one version (Tensor flow, graphics driver, CUDA, etc.) was off then the whole installation process had to be repeated.

As for the most satisfying parts of the project, I'm glad that I was able to make it work with >90% test accuracy despite not knowing all the steps in full detail. I have gained a new perspective on how I thought everything would work based on course work, and what actual problems are encountered during execution and what metrics to look for. I believe all these learnings will come in handy for future projects.

Improvement

There are various improvements that can be made:

1. Tweaking the RNN to get better accuracy/F1 score. [1] was able to get to 99.9%

2. Changing the audio feature to MFCC [1] which will result in smaller data size for training. Less is more!
3. Reducing audio duration from 3 seconds to 1 second as in [1]. Less is more!
4. The RNN took more than 150 epochs to train. Is there a different RNN design that can train faster like in <50 epochs?
5. Increasing the number of reciters to 10 or 20 to see if the same test accuracy is achieved.
6. Using latest CPU/GPU to see if results improve.

I see this project as a first step in getting the Arabic language speech recognition working in a similar manner to the English language speech recognition. I expect more tweaking is needed specifically for the Arabic language to further improve the results.

References

[1] "Quran Reciter Identification: A Deep Learning Approach", https://ieeexplore.ieee.org/document/8539336

[2] "How to Develop a Bidirectional LSTM For Sequence Classification in Python with Keras", https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/

[3] "Using CNNs and RNNs for Music Genre Recognition", https://towardsdatascience.com/using-cnns-and-rnns-for-music-genre-recognition-2435fb2ed6af

[4] The Quran Surah/Aya info was obtained from tanzil.net: http://tanzil.net/res/text/metadata/quran-data.xml