**Motivation:**

Voice commands are quite common nowadays, they are used on various search engines, mobile applications, and digital assistants such as Apple Siri. Nevertheless, they can only recognize the language that has been presented by the users, and for those of us who are at least bilingual, it can sometimes be inconvenient. This inconvenience made us want to improve voice recognition technologies by incorporating language recognition.

**Problem Statement:**

Language recognition, once incorporated into voice recognition, has a very wide business application. For instance, automated customer support would be able to recognize the language spoken by the customers and communicate in that language, or even connect with a customer support agent speaking that language. Language recognition can be used to transcribe meetings, lectures, etc. in the correct language. Furthermore, it can provide real-time and accurate automated translation to assist overseas business practices, monitor social media or online content, or provide better public services for people who only speak foreign languages. There are many other possible implications for automated language classification and they would assist public services, security and law enforcement.

**Methodology:** After we selected 9 languages on Kaggle, we downloaded their zip files, unzipped the files on local machines, and chose 100 samples each to test our Python codes. After thorough exploration and ensuring all codes could run smoothly, we uploaded the zip files containing all MP3 for each language onto the Shared Computing Cluster (SCC).

We have 3 different notebooks, each dedicated for different tasks and to limit RAM usage. We first ran the unzipping notebook to extract the raw data in a folder, and then ran the preprocessing notebook to preprocess them. During the preprocessing, we limited the sampling rate to 4,800 (it was 48,000 originally), normalized the audio and removed the leading and trailing silence of each audio. We then extracted the labels (language) and Mel-Frequency Cepstral Coefficients (MFCCs) and output them as Pickle for later use. We chose MFCCs because they mimic the human auditory system, are robust against noise, and can represent the entire audio file compactly and in waveform.

After preprocessing, we moved to our modeling notebook to load and read the 9 Pickle files, we only load 20,000 MFCCs and their labels to reduce the computational load and ensure a balanced sample across labels. Furthermore, we padded the data, stacked them, converted them into tensors, split them into training, validation, and testing sets, and then fed them into our models. We have Random Forest, CNN, LSTM, CRNN, one transfer learning from VGG16, and an autoencoder.

**Results and Insights:**

After running all the models, we found that CNN was able to produce the best results, with test accuracy of 0.6516 with 15 epochs. We believe that if time allows and with enough computational power, we will be able to run more epochs and achieve an even higher accuracy. Random Forest had accuracy of 0.4429, LSTM achieved testing accuracy of 0.5287, VGG16 with testing accuracy of 0.3008, CRNN with accuracy of 0.53, and autoencoder had unusually high test loss.

According to our study, CNN has, to some extent, a better accuracy than other models. This is mainly due to the fact that CNN learns automatically from the input data of the convolutional layer and captures the spatial information of the AUDIO (we used MFCCs with three dimensions), and therefore can capture the features of the different LANGUAGES more accurately. Also CNN has translation invariance, which allows CNN to recognize features very accurately even when the feature coordinates change, which is very important for audio learning. This is important for audio learning because the features of a language may shift on the scale of coordinates according to dialectal habits as well as personal habits. CNNs can eliminate these interferences and capture the language features more accurately for accurate classification.

We tried to use an autoencoder, but the complexity that autoencoder brings to the model is much higher than the usefulness it brings. Also in the current research so far, we don't need Autoencoder for dimensionality reduction, compression or denoising. Maybe in the next research, we will compress our existing MFCCs by using a suitable autoencoder to reduce the arithmetic usage.

Unfortunately, due to various limitations, we could not perform hyper-parameter tuning using W&B using our complete dataset, but were able to obtain W&B reports on our initial dataset with 100 samples from each language (Appendix 2-6).

**Challenges and Solutions:**

Our dataset size is huge, the entire dataset has 34 different languages and is around 64 gigabytes. We chose only 9 languages and it is still 17 gigabytes. It would be very computationally intensive to process the entire dataset at once, even uploading the dataset to SCC crashed our browser, and running . Therefore, we had to upload the zip files of each language and unzip them using Jupyter Notebook. Our hyper-parameter tuning is also limited due to restricted computational power, limited SCC session time.

During preprocessing, we originally wanted to process all raw data and output one file that contains preprocessed data, but even the SCC could not achieve it on 4 CPU cores and 6.0 GPU compute capability. The cell appears to be running, but the data is no longer being processed. We had to attempt different approaches, and we ended up writing a loop that processes one language folder at a time and outputs 9 different preprocessed files. Even with that being done, the SCC with 8 CPU cores could not process all 9 languages in a single run. It processed 2 to 3 languages fairly quickly, but then it would stop processing despite the cell being still running. We had to restart the kernel multiple times and run the preprocessing codes several times to obtain the preprocessed file of all languages. To avoid preprocessing the languages that have already been processed previously, we had to move the raw data folder away from the predetermined file path.

We also attempted to extract different features from raw files to improve modeling accuracy, including spectrogram, spectral centroids, spectral bandwidth, spectral roll-off, spectral flatness, zero crossing rate, pitch, chroma, root mean square, etc. However, considering the computational constraints and the amount of padding required for most features, we had no choice but to extract only MFCCs.

Through some research, we learned that HDF5 files are great for storing and handling large datasets, hence we originally chose to output our preprocessed files as HDF5. Nevertheless, it is very strict about the dimension of data, everything needs to have the same shape. Our raw audio data came in various lengths, since each unit for MFCCs is time, the length of each MFCC is different. We had to pad MFCCs before outputting them as HDF5, and our HDF5 files were as large as 5 gigabytes. Alternatively, we found that Python’s Pickle format may also work for our dataset: they do not require data to have the same shape and they are only around 100MB.

We tried to use Google’s audio-to-text API and were successful in obtaining the transcripts of each audio, but we found that the API was not very accurate and running the API for the entire dataset would exceed our monthly quota and credits from previous courses combined. Furthermore, queuing for SCC sessions was very time-consuming and it has caused some inconvenience. Nevertheless, after our presentation, we got a SCC session that we have queued for days, and it was powerful enough to successfully execute all of our codes.

#### **Future Steps:**

#### We have several unfinished tasks, including translation, we were unable to feed more different features into our models, and we could not complete W&B hyper-parameter tuning on a full-size dataset. Furthermore, the transfer learning model we have may not be very suitable for processing audio files, we should look for transformers that were trained on audio data.

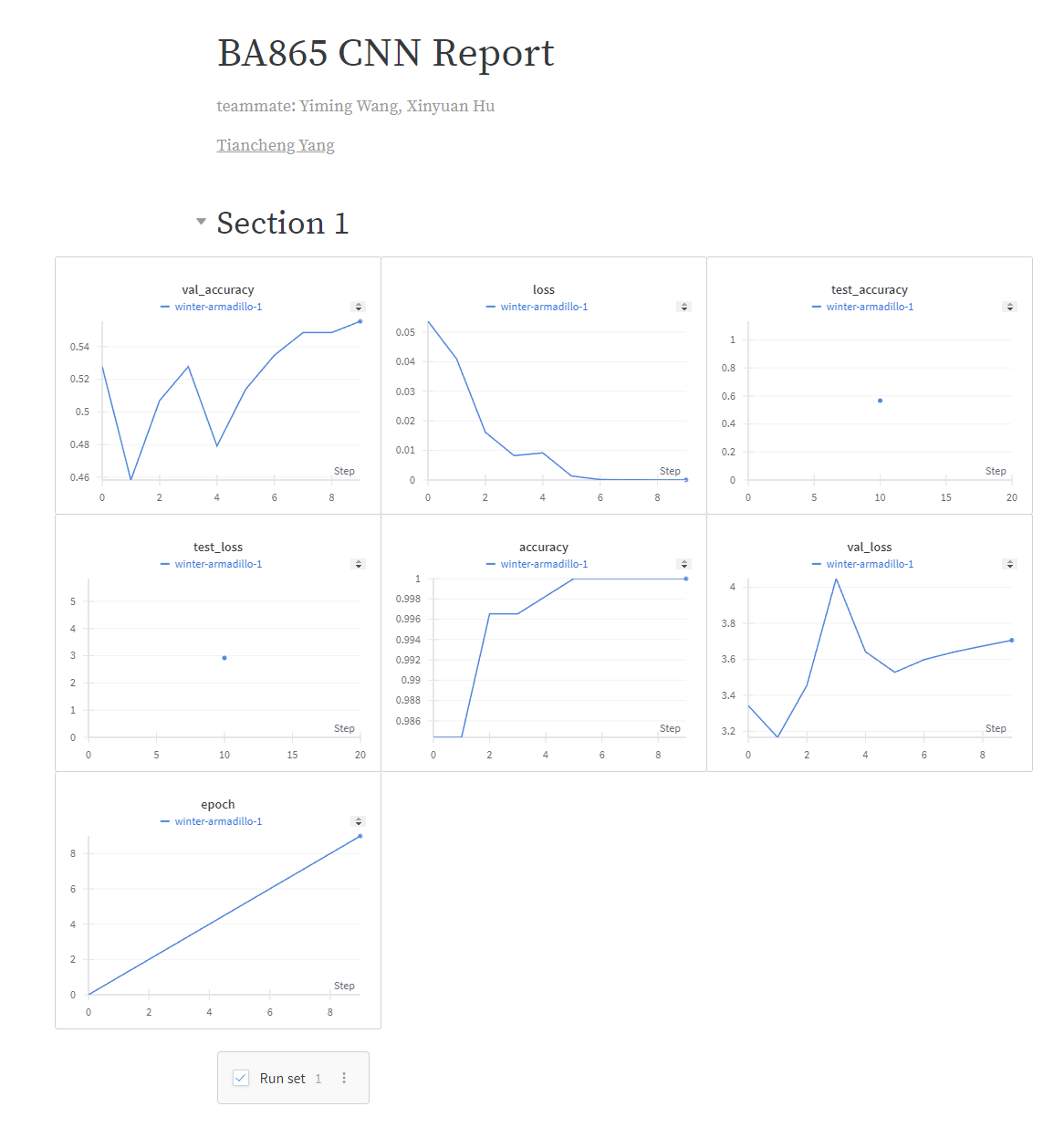
#### We originally planned to take a shot with audio translation, but due to limited resources, we could not achieve it. With enough computational power and reliable audio-to-text software/package, we can easily achieve real-time audio translation, where the audio-to-text converts audio to text, the model identifies the language spoken and uses the proper translation and text-to-audio for that language.

**Appendix:**

#### Contributions Table:

| **Name** | **Topics Coding** | **Contribution** |
| --- | --- | --- |
| **Tiancheng Yang** | Data exploration, CRNN, VGG16, interface, W&B, PPT, Model Research | 33.33% |
| **Yiming Wang** | Preprocessing, unzip, efficiency improvement, LSTM, PPT, Report, dataset handling | 33.33% |
| **Xinyuan Hu** | Preprocessing, Google Cloud Audio-to-Text API, CNN, PPT, Report, arrange meetings | 33.33% |

1. Weights and Biases report links:
   1. CNN: <https://api.wandb.ai/links/tianchengbu/dgtyq0vr>
   2. CRNN: <https://api.wandb.ai/links/tianchengbu/kca1jwci>
   3. VGG16: <https://api.wandb.ai/links/tianchengbu/m2yyyat4>
   4. LSTM: <https://api.wandb.ai/links/tianchengbu/nmj9m6sc>
2. CNN report



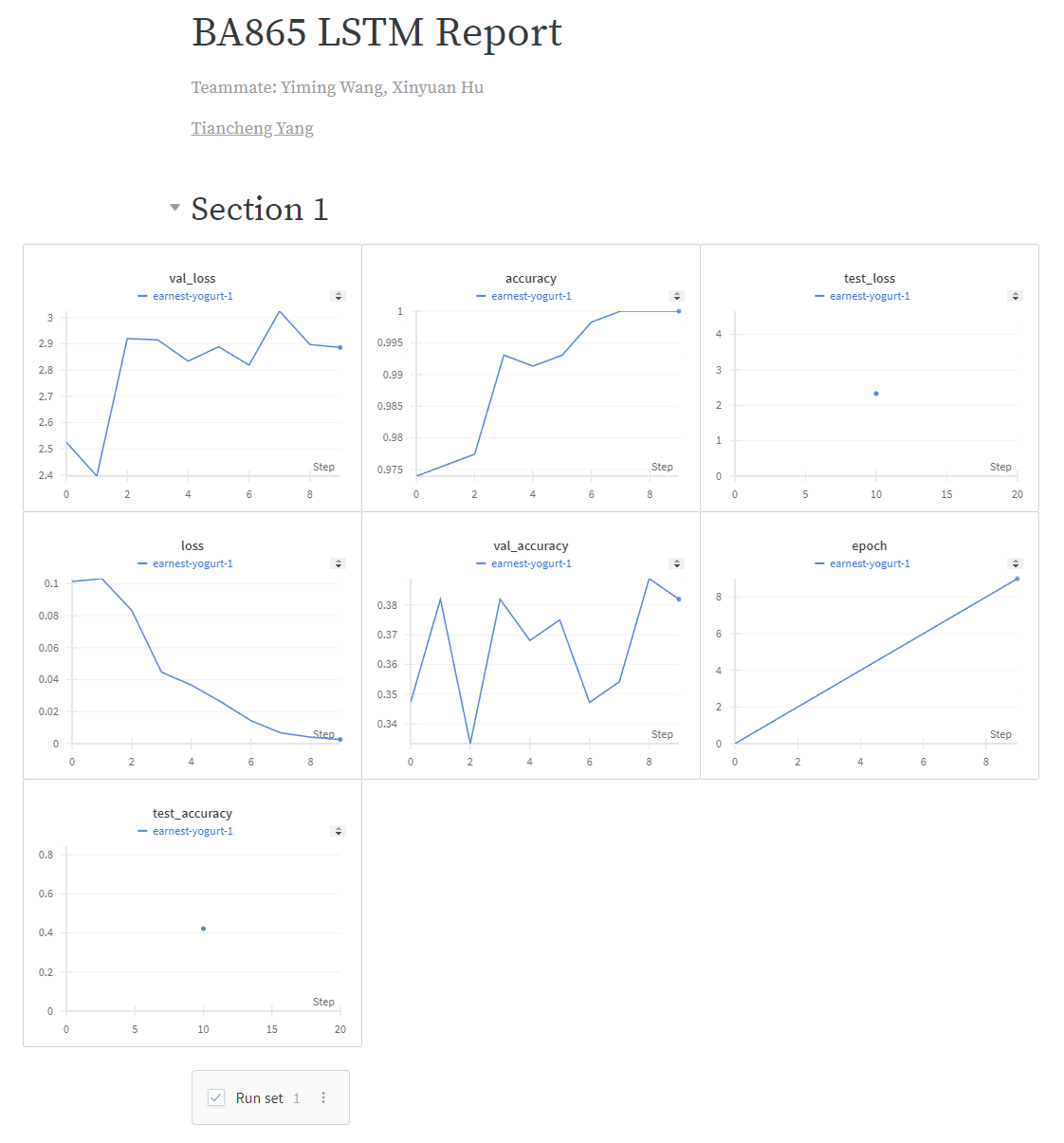
1. CRNN report



1. VGG16 report



1. LSTM report



1. References:
   1. <https://www.youtube.com/watch?v=vbhlEMcb7RQ>
   2. <https://medium.com/@karthikmandapaka/handling-audio-data-for-machine-learning-7ba225f183cb>
   3. [https://cloud.google.com/speech-to-text/docs/transcribe-api?\_gl=1\*e6i4h4\*\_ga\*NjczNjg1ODQzLjE3MTI5NzcxNTA.\*\_ga\_WH2QY8WWF5\*MTcxMjk3NzE1MC4xLjEuMTcxMjk3OTk5MC4wLjAuMA..&\_ga=2.30888056.-673685843.1712977150](https://cloud.google.com/speech-to-text/docs/transcribe-api?_gl=1*e6i4h4*_ga*NjczNjg1ODQzLjE3MTI5NzcxNTA.*_ga_WH2QY8WWF5*MTcxMjk3NzE1MC4xLjEuMTcxMjk3OTk5MC4wLjAuMA..&_ga=2.30888056.-673685843.1712977150)
   4. <https://codelabs.developers.google.com/codelabs/cloud-speech-text-python3#0>
   5. <https://cloud.google.com/speech-to-text/docs/reference/rest>