50.039 TPDL Project Report

Group 19

Muthu Ramaswamy	1007010	Training code and report
Teo Zheo Joshua	1006869	Model development and report
Du Bowei	1006633	Model development and training

Introduction

Our project explores the task of Spoken Language Identification (SLI) using deep neural networks to distinguish between various languages spoken in Southeast Asia and Europe. We explore the motivation for applying both convolutional and recurrent neural networks, present our own model that makes use of both techniques, and compare our results to other models working on the same task. We also detail the development of our model, its training process, and its limitations. Instructions to get the code and model running are found in the section README.md.

Related Work & State-of-the-Art

The performance of SLI models varies widely across publications and depends on many different factors. Before constructing a model, an appropriate dataset and format must be selected. This includes, but is not limited to, the number of languages involved in training, the type of audio feature set used, whether the closed or open-set variant of SLI is considered, and the length of audio samples. The models in [1] see an almost 8% improvement in accuracy by increasing sample lengths from 3 seconds to 30 seconds.

Two state-of-the-art models can be found in [2] and [3]. [2]'s attention-based convolutional-recurrent neural network is reported to have managed 98.7% accuracy on a dataset of Indian languages. However, [3] was unable to replicate the results of [2], and instead managed 95% on a set of 9 languages with a time-delay neural network. Both models utilised mel-frequency cepstrum coefficients (MFCCs) for audio feature sets.

Some papers including [1] use log-mel-spectrograms for their feature set instead of MFCCs. MFCCs being extracted from log-mel-spectrograms provide a more compressed audio representation than log-mel-spectrograms. However, both have been used for training various models with success.

Many models also utilised the data augmentation methods proposed by [4]. SpecAugment was originally proposed as an augmentation tool for the automatic speech recognition problem, but saw success in many other speech related tasks, including SLI as seen in [5]. The transforms proposed are implemented in PyTorch as TimeStretch, TimeMasking, and FrequencyMasking. TimeStretch and TimeMasking seem especially useful since the transforms produce audio which could still be plausibly identified as containing some language.

Dataset

For our dataset, we utilised audio samples of 9 languages from Mozilla's Common Voice datasets. We performed 2 classifications tasks: a 4-class classification between Chinese, English, French, and Indonesian and a further 9-class classification between the former 4 as well as Arabic, Hindi, Spanish, Thai, and Vietnamese. From the audio, we choose to generate log-mel-spectrograms (over MFCCs) from 3-second audio clips of speech to serve as samples.

To process and standardise the data, we reencoded all audio samples from the MP3 to WAV format. The audio was then downsampled to16kHz. A pretrained voice activity detector was then used to extract 3 second segments of voice activity at a time, from which the log-mel-spectrogram was finally generated. This allowed our model to focus on the language detection problem, rather than silent or nonsense audio clips.

This process gave us a final dataset consisting of 3177 samples across the 9 languages. Table 1 shows the distribution of samples across the languages classified. Instructions to run our preprocessing code can be found in the section README.md.

Language	Sample count
Arabic	173
Chinese	556
English	480
Hindi	525
French	442
Indonesian	169
Spanish	292
Thai	310
Vietnamese	230
Total:	3177

Table 1: Distribution of samples across languages. The first 4 languages are used for the 4-class classification problem.

Our Model

To set a baseline to which we can compare our model, we utilised a pretrained Resnet50 model, like the one in [6]. That model managed 89% accuracy across 6 languages using 3.75 seconds of audio per sample.

For our own networks, we wanted to model both local language features and temporal dependencies. This provided our motivation for constructing attention-based convolutional-recurrent neural network, like those in [1] and [2]. Specifically, our models are convolutional neural networks attached to a bi-directional long short-term memory (LSTM) layer and finally a self-attention layer, or attention-based CNN-BiLSTM. We constructed two variations of this model: one based on the architecture of Resnet34 and another on Resnet50.

In both models, spectrograms are passed through the convolutional residual blocks of the Resnet34 and Resnet50 architectures to produce encoded representations of the input. In principle, this should extract the most important local language representations from the spectrograms. This encoding is then passed through a bi-directional LSTM (BiLSTM), in order to model both forwards and backwards relationships across the encoding. The sequence is then passed through an attention layer before the final fully connected classification layer.

Our models were trained using the Adam optimiser with default hyperparameters except for an occasionally toggled learning rate. Cross entropy loss with L2 regularisation is used as the loss function to minimise.

Model Performance

The model was trained on 70% of the previously described dataset. The remaining samples were split between the validation and test sets.

Figures 1, 2, 3, and 4 show the training and validation losses and accuracies for our attention-based Resnet34-BiLSTM and Resnet50-BiLSTM for both the 4 class and 9 class SLI problems. Instructions to recreate the visualisations may be found in README.md.

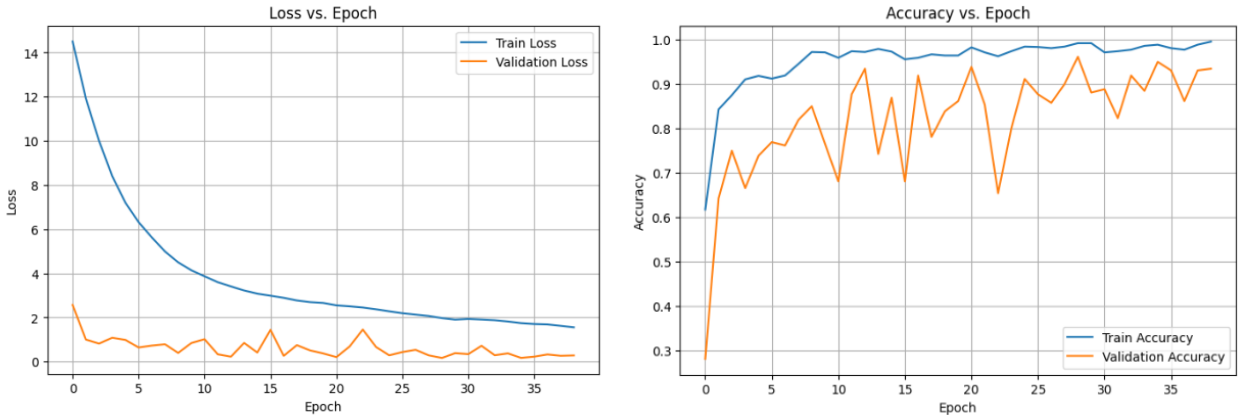

Figure 1: Attention-based Resnet34-BiLSTM training loss and accuracy over epochs for 4-class SLI.

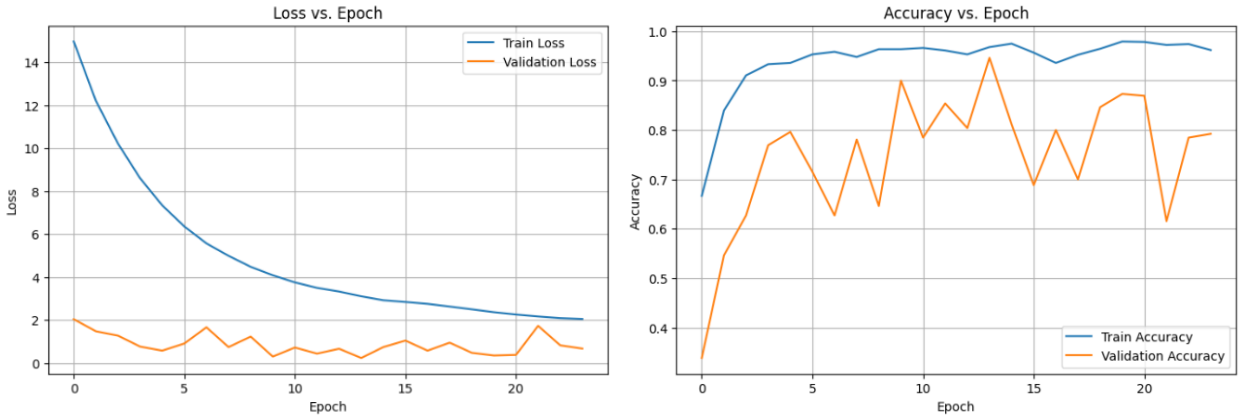

Figure 2: Attention-based Resnet50-BiLSTM training loss and accuracy over epochs for 4-class SLI.

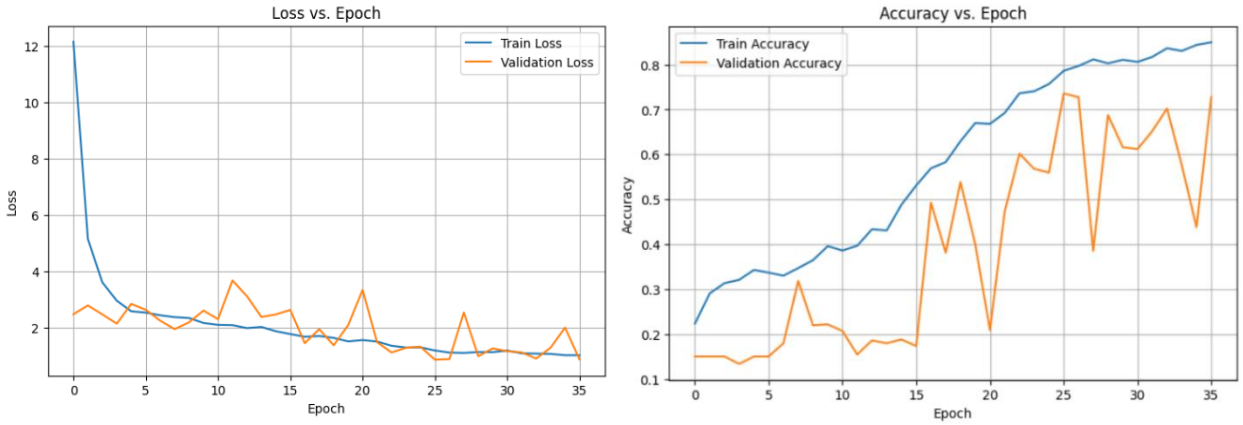

Figure 3: Attention-based Resnet34-BiLSTM training loss and accuracy over epochs for 9-class SLI.

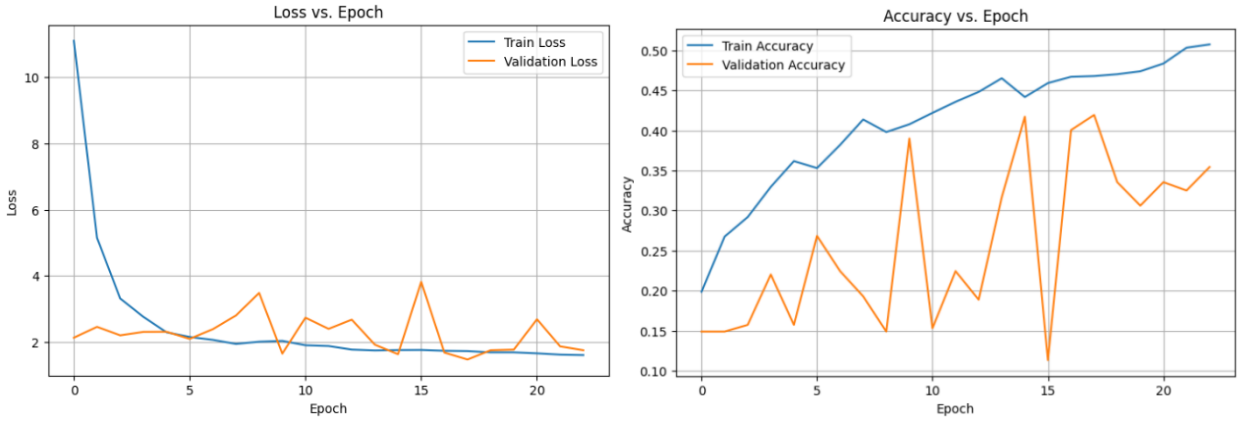

Figure 4: Attention-based Resnet50-BiLSTM training loss and accuracy over epochs for 9-class SLI.

Table 2 compares the accuracies of our models, as well as the accuracy of a baseline Resnet50.

Model	**4-class accuracy**	**9-class accuracy**
Resnet50	0.9387	0.6876
Resnet34-BiLSTM	0.9579	0.7778
Resnet50-BiLSTM	0.9349	0.4214

Table 2: Accuracy comparison of each model across both classification problems.

Compared to the baseline Resnet50 model, both the attention-based Resnet34-BiLSTM and Resnet50-BiLSTM saw minimal improvement in the 4-class SLI. In fact, the Resnet50-BiLSTM performed slightly worse. As figure 2 shows, the Resnet50-BiLSTM struggled to generalise with much fluctuation in validation accuracy over the epochs. The simpler model, Resnet34-BiLSTM, performed much better. Both these properties extend to the 9-class SLI problem as well.

In the 9-class SLI, the accuracy of all models dropped. This was expected; the number of classes was more than doubled, and many of the additional Southeast Asian languages are more similar. However, where the Resnet34-BiLSTM only had a 2% accuracy lead over the Resnet50 for 4-class, the model retained far more accuracy in the 9-class SLI with about a 9% difference between the models. The Resnet50-BiLSTM struggled to train well on the 9-class SLI data.

The attention-based Resnet34-BiLSTM shows the efficacy of introducing recurrent neural network elements to model speech's time dependency in order to identify spoken language. The failure of the attention-based Resnet50-BiLSTM to train well is more difficult to interpret. Not only did it perform worse that its lightweight Resnet34 cousin, but it struggled to train at all for the 9-class SLI. It is possible that utilising Resnet50 as a local language feature extractor destroys any useful time-dependent features the LSTM might otherwise learn. However, there may be many other reasons for this model's failure.

README.md

We run our models in Python 3.13.1. The following commands are given as bash. We recommend using pyenv to set up a virtual environment but manage your environments as seems best to you. Clone the project repository from https://github.com/r-muthu/group_19_tpdl doing the following:

```
git clone https://github.com/r-muthu/group_19_tpdl
cd group_19_tpdl
pip install requirements.txt
```

Getting the data

There are two options to acquire the spectrogram data used for our model: (1) by downloading the files from Mozilla Common Voice and running the preprocessing scripts or (2) by downloading the spectrograms directly.

To train our model with its original data, the following files need to be downloaded from https://commonvoice.mozilla.org/en/datasets:

Language	**File**
Chinese	Chinese (China) – Common Voice Corpus 2
Indonesian	Indonesian – Common Voice Corpus 5.1
Thai	Thai – Common Voice Corpus 6.1
Vietnamese	Vietnamese – Common Voice Delta Segment 10.0
Arabic	Arabic – Common Voice Corpus 5.1
English	English - Common Voice Delta Segment 15.0
French	French – Common Voice Delta Segment 20.0
Hindi	Hindi – Common Voice Delta Segment 10.0
Spanish	Spanish – Common Voice Corpus 2

Table 3: Files downloaded for each language from Mozilla's Common Voice dataset.

Create a location for the data files by running:

```
mkdir -r data/mp3/{chinese,indo,thai,viet,arabic,english,french,hindi,spanish}
```

Unzip the data into the appropriate language files within the `data/mp3` directory. For each language except English, we utilised the first 400 audio samples. For English we took the last 400 samples. To generate the log-mel-spectrograms for the model, run:

```
python3 preprocessing.py
```

This will take some time[1].

Alternatively, we provide a file containing copies of all the pre-processed spectrograms at https://drive.google.com/drive/folders/1kTJOxuvR3wXVKE_1nKetCILjwRyUGGZY. Create a location for the log-mel-spectrograms files by running:

```
mkdir -r data/mel/{chinese,indo,thai,viet,arabic,english,french,hindi,spanish}
```

Move all the data into their appropriate language files. In order to perform the 4-class task, remove the 5 language files from the data directory not associated with the 4-class SLI. The Dataset classes and functions dynamically adapt to the number of language files to classify.

[1] If `python3` does not invoke the Python interpreter, try `python` or `py`. Or just reinstall Python.

Training the models

Code for training the models from scratch can be found in `train.ipynb`. The notebook's cells are designed to be run in sequence. There is no need to modify cells marked "(no changes needed)".

The path to the dataset has to be inserted after the class definition of `SequenceDataset`. If instructions have been followed to this point, the data is probably in `./data/mel`.

Run the subsequent cells to run the training process for each model, which will also generate loss and accuracy visualisations.

Loading the models

To load our final models, retrieve the weights from the following link: https://drive.google.com/drive/folders/1uxu8qlXxQqxNDPU8P5XdMZ901_riofRt.

Running the loaded models is done in `train.ipynb` under the header 'Loading and testing a model'. Weights are loaded from a `./checkpoint/*` directory. When the directory is appropriately arranged, the subsequent cells may be run to test the models.

Limitations and Improvements

Though the Common Voice dataset is a popular one for many tasks, there are biases in the sample counts across languages. This unfortunately propagated into training, leaving us with some classes which were much more common (such as Chinese at 556 samples) while others occurred much less frequently (such as Indonesian at 169 samples). This

imbalanced dataset is likely to have made it much harder for the models to learn and identify languages with smaller sample counts.

The final prepared dataset we used for training was also incredibly small, at just 3177 spectrograms. This totals to a little over 2.6 hours of audio. In contrast, most performant models operated on hundreds of hours of audio, forcing them to learn the most important features of each language.

Furthermore, despite the frequent use of SpecAugment discussed in Related Work, we opted not to utilise the data augmentation techniques described. This could have improved the generalisability of our model, especially among low sample count languages.

Bibliography

[1] W. Cai, D. Cai, S. Huang, and M. Li, "Utterance-level end-to-end language identification using attention-based CNN-BLSTM" 2019. [Online]. Available: https://arxiv.org/abs/1902.07374

[2] A. Mandal, S. Pal, I. Duuta, M. Bhattacharya, and S. K. Naskar, "Is Attention always needed? A Case Study on Language Identification from Speech" 2021. [Online]. Available: https://arxiv.org/abs/2110.03427

[3] M. Eyceoz, J. Lee, and H. Beigi, "Modernizing Open-Set Speech Language Identification" 2022. [Online]. Available: https://www.researchgate.net/publication/360745462_Modernizing_Open-Set_Speech_Language_Identification

[4] D. S. Park, W. Chan, Y. Zhang, C. C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition" 2019. [Online]. Available: https://arxiv.org/abs/1904.08779

[5] C. Korkut, A. Haznedaroğlu and L. M. Arslan, "Spoken Language Identification with Deep Convolutional Neural Network and Data Augmentation," 2020 28th Signal Processing and Communications Applications Conference (SIU), Gaziantep, Turkey, 2020, pp. 1-4, doi: 10.1109/SIU49456.2020.9302425.

[6] S. Revay, M. Teschke, "Multiclass Language Identification using Deep Learning on Spectral Images of Audio Signals" 2019. [Online]. Available: https://arxiv.org/abs/1905.04348