University of Cape Town

Department of Statistical Sciences
Faculty of Science
Masters Minor Thesis

Triplet Entropy Loss: Improving The Generalisation of Short Speech Language Identification Systems

Author:
Ruan Henry van der Merwe

Supervisor:
Dr Şebnem Er

December 2020

Acknowledgements

I will keep it brief as everyone in this list knows how much they contributed.

- To my wife, Madelein, you have been my rock during this entire process and I could not have finished this without you.
- To Şebnem, my supervisor, thank you for all your feedback and zoom calls. Even though we could not meet nearly enough in person as we would have hoped for in this strange year, you still pushed me in the correct direction after every meeting and inspire me to ask even more questions.
- To my family, friends and Church community, you kept me sane during this time with all your distractions. Thank you!
- To both my managers which I worked under during the two years of study, Guillaume and Etienne, thank you for understanding the struggles of part time studying and allowing me the freedom to work on this project and providing support in any way you can.

Abstract

Spoken language identification systems form an integral part in many speech recognition tools today. Over the years many techniques have been used to identify the language spoken, given just the audio input, but in recent years the trend has been to use end to end deep learning systems. Most of these techniques involve converting the audio signal into a spectrogram which can be fed into a Convolutional Neural Network which can then predict the spoken language. This technique performs very well when the data being fed to model originates from the same domain as the training examples, but as soon as the input comes from a different domain these systems tend to perform poorly. Examples could be when these systems were trained on WhatsApp recordings but are put into production in an environment where the system receives recordings from a phone line.

The research presented investigates several methods to improve the generalisation of language identification systems to new speakers and to new domains. These methods involve Spectral augmentation, where spectrograms are masked in the frequency or time bands during training and CNN architectures that are pre-trained on the Imagenet dataset. The research also introduces the novel Triplet Entropy Loss training method. This training method involves training a network simultaneously using Cross Entropy and Triplet loss. Several tests were run with three different CNN architectures to investigate what the effect all three of these methods have on the generalisation of an LID system.

The tests were done in a South African context on six languages, namely Afrikaans, English, Sepedi, Setswanna, Xhosa and Zulu. The two domains tested were data from the NCHLT speech corpus, used as the training domain, with the Lwazi speech corpus being the unseen domain.

It was found that all three methods improved the generalisation of the models, though not significantly. Even though the models trained using Triplet Entropy Loss showed a better understanding of the languages and higher accuracies, it appears as though the models still memorise word patterns present in the spectrograms rather than learning the finer nuances of a language. The research shows that Triplet Entropy Loss has great potential and should be investigated further, but not only in language identification tasks but any classification task.

Contents

List of Figures

List of Tables

Nomenclature

ASR Automatic Speech Recognition, page 1

AUC Area Under the Curve, page 7

AWS Amazon Web Service, page 39

CEL Cross Entropy Loss, page 6

CNN Convolutional Neural Network, page 2

CRNN Convolutional Recurrent Neural Network, page 4

DFT Discrete Fourier Transform, page 14

FSDD Free Spoken Digit Dataset, page 80

GRS GlobalPhone Read Speech, page 8

HMM Hidden Markov Model, page 3

LID Language Identification, page 1

LSTM Long-Short-Term Memory Network, page 4

PDP phone-based dynamic programming, page 11

PRLM Phoneme Recognition followed by Language Modelling, page 4

RBS Radio Broadcast Speech, page 8

ReLU Rectified Linear Unit, page 5

RNN Recurrent Neural Network, page 4

SGD Stochastic Gradient Descent, page 27

SVM Support Vector Machine, page 3

TEL Triplet Entropy Loss, page 2

Chapter 1

Introduction

1.1 Background

Speech recognition tools have grown to form an integral part of many lives. For example, if you are reading this thesis on an electronic device, the chances of it containing one or other speech recognition tool are high, with Siri and Alexa being widely known examples.

To build a speech tool, one must first have a back-end that can perform Automatic Speech Recognition (ASR), which is a process to determine automatically what a user said purely based on the input speech signal. One of the first steps in an ASR system is to identify the spoken language using language identification (LID) systems, as the ASR systems in most cases are optimised for one language only. When the spoken language is one of the more common languages in the world, then there are already a vast majority of resources and tools that could implement this identification step.

But it is becoming more apparent that some of these tools do not generalise well to new domains [Abdullah et al., 2020]. There also are cases where the performance degrades when new speakers are introduced [Montavon, 2009]. This leads on to ask if other LID systems are also only usable for data that comes from the same domain? This can be acceptable in certain conditions, but for most applications the input data will come from unique domains and it is not feasible to train a model for each domain.

1.2 Goal of This Research

The aim of this thesis is to investigate methods that can improve the generalisation of LID systems to new domains, specifically in the case of short speech recordings (three seconds or less). The techniques will be tested on South African languages in order to test the performance on low resource languages and to test if the techniques work on languages that share the same characteristics, such as *Zulu* and *Xhosa*. The new LID techniques could then improve the ASR systems in South Africa. The new LID system could also be used as a stand-alone product. For example, if one can detect the language of a customer calling a call centre, one can then assign an agent that can speak in the customer's mother tongue.

The research will focus heavily on a novel training method being introduced, namely the Triplet Entropy Loss (TEL). It is inspired by the combination of research from [Bartz et al., 2017] and [Schroff et al., 2015]. [Bartz et al., 2017] converted the raw input speech recordings to a spectrogram which is fed into a Convolutional Neural Network (CNN) and [Schroff et al., 2015] introduced the Triplet loss function. The TEL training method then consists of training CNNs by optimising a loss function that combines the strength of Cross Entropy Loss (CEL) and Triplet loss.

The TEL method aims to improve upon the work done by [Abdullah et al., 2020] in terms of domain generalisation. Their approach involves applying domain adaption through adversarial training, which resulted in better generalisation for Slavic languages. For their technique one still requires data from the different domain during the training phase while TEL only requires one domain for training. The work done by [Abdullah et al., 2020] is explained further in Section 2.4 with TEL further discussed in Section 4.2. The other techniques that will be investigated are spectral augmentations and using CNN architectures that are pre-trained on the Imagenet dataset [Deng et al., 2009]. The training domain will be the NCHLT speech corpus and the unseen domain will be the Lwazi speech corpus. Both these domains are discussed in Chapter 3.

1.3 Outline of Thesis

The thesis comprises seven chapters, including the Introduction. The thesis will walk the reader through the history of LID systems, Chapter 2, before explaining the data that will be used during the research in Chapter 3. After this the methods to be investigated will be discussed in Chapter 4 and then the experiment results will be discussed in Chapter 5 and Chapter 6. Chapter 7 will then provide the conclusions made during the research and any future work that is required.

Chapter 2

Literature Review

2.1 LID and South African Languages

The first attempt of creating a system to predict the spoken language for all eleven official South African languages was by [Davel et al., 2012], where they implemented a Parallel Phoneme Recognition followed by Language Modeling (PPR-LM) architecture. A phoneme is a unit of sound that can change the meaning of a word, with these sounds being unique to every language. For example, in the English language the words *hat*, *cat* and *rat* are all pronounced differently by just one sound, but this sound changes the meaning of the words completely. In the *English* language there are 44 such phonemes, with *Afrikaans* and *Zulu* having 37 and 45, respectively [Henselmans et al., 2013].

For the PPR-LM system, the researchers built a phoneme extractor for all eleven languages using Hidden Markov Models (HMM) [Rabiner and Juang, 1986]. This is known as the front-end of the system. For every audio input signal, they sent the signal through the front-end to extract the phonemes whereafter the biphone frequencies are extracted from the identified phoneme strings. They then concatenate these frequencies into a single vector where it is then fed into a Support Vector Machine (SVM) [Kecman, 2005], known as the back-end, to predict what the spoken language is [Davel et al., 2012].

They trained this system on the *Lwazi* speech corpus and produced fairly good results. With audio inputs ranging from 3 to 10 seconds they achieved an accuracy of 71.72% (with the worst performing languages being *Zulu* and *Sesotho*). This drastically increased when they tried to predict the family language of the audio input (*Afrikaans*, *English*, *Nguni*, *Sotho-Tswana*, *Tswa-Ronga* or *Venda*) [Davel et al., 2012].

[Henselmans et al., 2013] attempted to predict the eleven languages using four methods, namely Phoneme Recognition followed by Language Modelling (PRLM), parallel PRLM (same as the above) and two analogous approaches based on word-level recognition. The best system was parallel word recognition followed by a language modeling approach with trigram language models. This system achieved a language ID error rate (ELID) of 0.418. According to the researchers: "ELID is the average language identification error rate compensated for evaluation priors". The technique performed the best in Germanic languages and was also built using the *Lwazi* speech corpus.

[De Wet et al., 2017] tried to solve the problem for the shortage of Afrikaans speech data by investigating if Flemish speech data can be added to the current Afrikaans data and the results simply showed that this did not have any true benefits. The researchers [Mbogho and Katz, 2010] investigated what the effect of South African accents are on English speech recognition systems and they found that models trained with accented English performed better, which indicates that there is merit in committing resources to further research accented training. Both [De Wet et al., 2017] and [Mbogho and Katz, 2010] used HMM models to conduct their research.

2.2 LID with Deep Neural Networks

From the research summarised previously, it is apparent that most of them required complex systems combined with a firm knowledge of language fundamentals. In 2017 [Bartz et al., 2017] identified a way to use a hybrid model built out of a CNN and a Recurrent Neural Network (RNN) [Lipton et al., 2015] which in this case is a Long-Short-Term Memory Network (LSTM) [Malhotra et al., 2015]. This hybrid model is known as Convolutional Recurrent Neural Network (CRNN) [Zuo et al., 2015]. For an in depth review on CNN architectures used in the literature and in this research, please refer to Section 4.1 or [Goodfellow et al., 2016].

What separated their work from previous work was that they built a CNN (based on the Inception-v3 architecture found in [Szegedy et al., 2016]) which received the image of a raw audio signal converted to a spectrogram as the input. The researchers then fed the extracted features of the CNN into an LSTM to generate the predictions. Thus the predictions are based purely on the deep learning models automatically extracting information from the spectrogram. This means that the only processing that has to be done on the input speech signal is simply to convert it to a spectrogram of a certain length. With the use of spectrograms, the extra requirement for language fundamentals are eliminated. They gathered and used data from Youtube and the European Union speech dataset for English, Russian, Dutch, French, Spanish and Mandarin resulting in close to 1508 hours of data [Bartz et al., 2017].

Table 2.1: CNN architecture used by [Bartz et al., 2017, pp. 5] to predict the spoken language based on a spectrogram image of the input audio signal.

Layer	**Kernel Size**	**Number of filters**
1	7x7	16
2	5x5	32
3	3x3	64
4	3x3	128
5	3x3	256

The architecture used in [Bartz et al., 2017] consists of five convolutional layers, with each layer having a Rectified Linear Unit (ReLU) activation function followed by Batch Normalization and a 2x2 max pooling layer (with a stride of 2). The kernel sizes and number of filters for each layer are shown in Table 2.1. As can be seen there are 5 layers where the kernel size decreases in size for the first two layers but stays constant at a size of 3x3 for the last three layers. The number of filters also increases two fold for each layer starting at 16 for layer 1 and ending at 256 for layer 5. After the convolutional layers, the architecture comprises of two LSTMs with 256 output units each. The output of both LSTMs are then concatenated into one vector of 512 dimensions and fed into a fully connected layer with 6 output neurons to classify the audio [Bartz et al., 2017].

Table 2.2: Results obtained for both the CRNN model (using the architecture in Table 2.1) and the Inception-V3 CRNN model. The values show the accuracy (%) and F1-Score for the original dataset as well as three other datasets where noise is applied.

Dataset	***CRNN***		**Inception-V3 CRNN**	
	Accuracy	*F1 Score*	*Accuracy*	*F1 Score*
No noise	91	0.91	96	0.96
White Noise added	63	0.63	91	0.91
Crackling Noise added	82	0.83	93	0.93
Background Music added	70	0.70	89	0.89

Ref: [Bartz et al., 2017, pp. 8]

To keep the audio length consistent for training and prediction, all the audio data is converted into 10 second recordings and then converted to spectrograms. The spectrograms were discretized using a Hann Window [Blackman and Tukey, 1958], using only frequencies up to $5kHz$ as most English language phonemes do not exceed $3kHz$ [Bartz et al., 2017]. Using the architecture described earlier and the 10 second

spectrograms, they achieved very promising results which are summed up in table Table 2.2. As can be seen from the table, even by adding noise to the input data they still achieved an accuracy of 82% (when the noise is crackling noise). One will also see that when the Inception-v3 CNN architecture is used, the results stay very constant over all the noise scenarios. However, using the Inception-v3 CNN requires six times more parameters to be trained. The authors did not mention what the performance was for out-of-domain data.

The researchers [Revay and Teschke, 2019] were inspired by [Bartz et al., 2017] and set out to improve the results. They achieved similar results, but by only using 4 seconds of audio data on a pre-trained ResNet [He et al., 2016] architecture. The data obtained for their research came from the VoxForge speech dataset. The languages used in the research were English, Spanish, French, German, Russian and Italian. The method resulted in accuracies above 90% for all languages except Russian and Spanish as the model seems to confuse the two with one another. The research team believe this is due to Russian being the only Slavic language trained on and that the model created a threshold to separate Russian and all other languages and that Spanish happens to be near the threshold.

In 2019, another group of researchers, [Sarthak et al., 2019] also investigated the performance between a one-dimensional (1D) and two-dimensional (2D) CNN architecture. The 2D designs consisted of a model containing an RNN structure with the other design only containing convolutional layers with feed forward layers.. The 1D model received as input the raw waveforms with the 2D model being fed spectrograms. For all the models developed, the researchers would follow the same design principle, namely that a pooling and batch normalisation layer will follow all convolutional layers and that they will use the ReLu activation function throughout the model. The two-dimensional CNN also implemented residual connections which were popularized by the ResNet architecture. The data used consisted of the VoxForge dataset. When comparing the results, the 2D CNN without an RNN structure performed the best with an accuracy of 95.4%. The next best model was then the 2D CNN with an RNN structure at 95.0% and lastly the 1D CNN with an accuracy of 93.7%, [Sarthak et al., 2019]. Again, the authors did not mention out-of-domain performance.

2.3 Applying Triplet Loss

The work in the previous section consisted of end to end deep learning systems where the model parameters are trained using Cross Entropy Loss (CEL) as the loss function. The researchers in [Margolis et al., 2018], [Bredin, 2017] and [Mingote et al., 2019] looked at using Triplet loss instead of CEL on various speech tasks such as LID and

user identification. Triplet loss is a loss function that optimises the embeddings generated by the network such that samples from the same class are close to each other in the embedding space while also being far away from other classes.

The loss works by choosing an anchor example (x_i^a) and an example from the same class as the anchor (x_i^+) and one from a different class (x_i^-). The loss to be optimised is found in Equation (2.1), [Schroff et al., 2015] , where $f()$ is the embedding function and α is a hyperparemeter for the loss function. The higher α is, the greater the separation between classes. When $\left\|f(x_i^a) - f(x_i^+)\right\|_2^2 < \left\|f(x_i^a) - f(x_i^-)\right\|_2^2 + \alpha$, the loss is zero.

$$L = \sum_{i=1}^{N} \left[\left\|f(x_i^a) - f(x_i^+)\right\|_2^2 - \left\|f(x_i^a) - f(x_i^-)\right\|_2^2 + \alpha \right]_+ \tag{2.1}$$

[Mingote et al., 2019] specifically looked at implementing an LID system based on triplet networks. Their method alters Equation (2.1) to optimise the Area Under the Curve (AUC). The researchers believe this is a more appropriate and intuitive metric than traditional metrics, since this metric is a measure of the system performance. The researchers have also previously implemented this same technique for text-dependent speaker verification systems [Mingote et al., 2020].

The expression Θ^* the researchers implement is shown in Equation (2.2), where $\sigma(s)$ is the sigmoid function, $s_\Theta\left(p_i^+\right)$ is the similarity metric of each pair of anchor-positive embeddings where $p_i^+ = \left(e, e_i^+\right)$ with $i \in \{1, \ldots, m^+\}$ and m^+ is the total number of positive examples, $s_\Theta\left(p_j^-\right)$ indicates the metric of each pair of anchor-negative embeddings where $p_j^- = \left(e, e_j^-\right)$ with $i \in \{1, \ldots, m^-\}$ and m^- is the total number of negative examples, and α is an adjustable parameter, [Mingote et al., 2019].

$$\Theta^* = \underset{\Theta}{\operatorname{argmax}} \frac{1}{m^+ m^-} \sum_{i}^{m^+} \sum_{i}^{m^-} \sigma\left(\alpha\left(s_\Theta\left(p_i^+\right) - s_\Theta\left(p_j^-\right)\right)\right) \tag{2.2}$$

By implementing this loss the researchers managed to outperform traditional methods, such as Weighted Gaussian Back-end and SVM's, in the context of a closed-set evaluation of the LRE 2009 dataset (more details about this dataset can be found in [Martin and Greenberg, 2010]), [Mingote et al., 2019].

2.4 Generalisation of LID Systems

A recent study from [Abdullah et al., 2020] has found that end to end deep learning LID systems perform severely worse when tested on out-of-domain samples. The

researchers define out-of-domain samples as data coming from datasets the model was not trained on. This research will from now on use the same definition for out-of-domain data.

The researchers found that spectral features generalise better to new domains than cepstral features, showing that networks using spectrogams as input will generalise better. To improve the generalisation from domain to domain, the researchers aimed to force the model to learn features that are domain invariant. They do this by using a technique called domain adaption through backpropagation that was introduced by [Ganin and Lempitsky, 2014] and also mentioned in [Meng et al., 2017].

The technique entails creating two fully connected network blocks, where each block has its own task, that are connected to the output of a feature extraction block. The block B_y aims to predict the language of a sample while the block B_d tries to predict from which domain a sample is sampled. Both these blocks get fed by B_f, which is the feature extractor. Each training sample in the source domain gets augmented with a label $d = 0$ and the samples from the other domain receive a label $d = 1$. The parameters θ_d, which are the parameters of the block predicting the domain, are then optimised to only minimise the loss of the domain classifier and likewise θ_y is optimised to predict the language label.

To ensure that the parameters of the feature extractor block become uninformative of the domain, the researchers seek to optimise θ_f such that the loss of the domain classifier gets maximised. This is known as adversarial training, where there is a competition between blocks in the network to optimise different losses. The loss function for this method can be found in Equation (2.3) where $\mathcal{D}_S$ is the source domain, $\mathcal{D}_\mathcal{T}$ is the target domain and λ is a parameter that controls the contribution of the domain classifier's loss [Abdullah et al., 2020].

$$\begin{aligned} J\left(\boldsymbol{\theta}_f, \boldsymbol{\theta}_y, \boldsymbol{\theta}_d\right) = \sum_{(\mathbf{X}_i, y_i) \in \mathcal{D}_S} L_y\left(B_y\left(B_f\left(\mathbf{X}_i; \boldsymbol{\theta}_f\right); \boldsymbol{\theta}_y\right), y_i\right) \\ -\lambda \sum_{(\mathbf{X}_t, d_i) \in (\mathcal{D}_\mathcal{S} \cup \mathcal{D}_\mathcal{T})} L_d\left(B_d\left(B_f\left(\mathbf{X}_i; \boldsymbol{\theta}_f\right); \boldsymbol{\theta}_d\right), d_i\right) \end{aligned} \tag{2.3}$$

The researchers ran tests with the Slavic portion of the GlobalPhone Read Speech (GRS) dataset as the source domain and the Radio Broadcast Speech (RBS) dataset as the other domain. The RBS dataset is a large collection of Slavic radio broadcasts. The researchers also used CNNs as base models. When not applying the domain adaption technique described above, the models performed very well in domain but performed poorly for out-of-domain samples, but after applying the cross domain adaption technique, the researchers could achieve improvements of up to 77.7%.

This thesis will attempt to test the possibility of generalising LID systems to new domains, without the need to have data from the other domain, as is required in

the work above. This is because there will be situations where one can not predict the domain from where new samples will come from. This thesis should also be a timely reminder that machine learning systems should be tested rigorously before deploying or publishing them. False claims of performance could lead to dangerous situations as the model could be placed in production affecting people because it was reported that it could work.

The next chapter will explore more of the NCHLT speech corpus, which will serve as the training domain, as well as the Lwazi speech corpus which will serve as the test domain.

Chapter 3

Language Domains

To create effective speech technology, it is important to have a dataset containing various examples to learn from. From a South African context, there were very few resources to get such data. Luckily, between 2006 and 2009, the Lwazi project ran to demonstrate the use of speech technology in South Africa. One output of this project was the Lwazi ASR corpus, which consisted of annotated speech data for all eleven languages. The data consisted approximately of 200 speakers with five to ten hours of data for each of the eleven languages, [Van Heerden et al., 2009].

It was realized after the release of the Lwazi Corpus that larger datasets are required for efficient tools to be developed. The Lwazi Corpus is also a telephone-based corpus, which means the audio is a recording over a telephone call. The South Africa Department of Arts and Culture recommended that higher quality recordings are required. These two requirements led to the development of the NCHLT (National Centre for Human Language Technology) speech corpus, which consists of nearly 50 hours of clean data for all the languages. Also, the dictionary of words is extended from 5 000 words to 15 000 [Barnard et al., 2014].

In order to test how the methods perform on out-of-domain data, the NCHLT and Lwazi corpora will be viewed as two different domains as they align with the definition of [Abdullah et al., 2020] for out-of-domain data, namely that out-of-domain data is data from a completely different dataset. The first section in this chapter will discuss the training domain, the NCHLT corpus, as well as how the audio will be converted to spectrograms in order to be used by the CNN models. After this the Lwazi corpus will be discussed.

3.1 NCHLT Speech Corpus

For this project, the first release and a subset of the auxiliary speech corpus is used. The reason for including the auxiliary recordings is to introduce noise and robustness to the system. It is assumed that if the model sees good examples mixed with bad examples that it will be able to generalise better to new domains, specifically where a lot of background noise is present or where the speakers are not speaking clearly.

The data can be obtained from the Sadilar website. The next sections summarise the data and explains how the data is cleaned, filtered and configured to be used by the methods explained in Chapter 4.

3.1.1 Data Cleaning

All of the languages in the Speech corpus contain an XML file describing the metadata related to each recording. The metadata consists of information on the age, gender, location, duration and the phone-based dynamic programming (PDP) score for each recording. PDP refers to a confidence scoring technique named phone-based dynamic programming. PDP matches a phone string from the prompt against a phone string extracted from the audio using dynamic programming. The scores are between -1 and 1. A +1 score is an indication that the sample is very likely to be a correct pronunciation of the target prompt and a score -1 indicates it is very unlikely. A score of +1 though does not necessarily mean it is a perfect match. Further detail can be found in [Davel et al., 2012].

Table 3.1: Summary of NCHLT data before filtering unusable data.

Language	***Unique speakers***	***Gender split [M:F]***	***Duration [H]***
Afrikaans	210	103:107	97
English	210	110:100	84
Ndebele	148	70:78	94
Sepedi	210	110:100	119
Sesotho	210	97:113	128
Siswati	197	101:96	132
Setswana	210	101:109	124
Tsonga	198	103:95	137
Tshivenda	208	125:83	147
Xhosa	209	103:106	157
Zulu	210	112:98	150

In Table 3.1 one can see a summary of each language's metadata before any filtering

or cleaning takes place. As can be seen the Germanic languages (*Afrikaans* and *English*) as well as *Ndbele* have significantly fewer hours than the other languages. Also, the gender distribution is evenly spread in all languages with the biggest difference found in *Sesotho* and *Zulu*. *Ndbele* though has substantially less speakers than the other languages.

Because of the researchers deciding to prompt non-English speakers with roughly 20% of English prompts, some of the recordings for the speakers are not in the language which is expected. Also, due to the methods used to gather text examples during the NCHLT project, some text gathered was not linked to the language it was gathered for. Some shortcomings of the methods for example are that the community generated the text and thus there was very little control. Even the text gathered from Wikipedia for *English* could contain other languages as Wikipedia articles do not always contain only one language. For example, in one instance a *Setswana* speaker was prompted to say "air traffic control" and on another occasion an *English* speaker received the prompt "Dien Bien Phu". The last example is a very good example of how an *English* participant does not specifically receive an incorrect prompt as Dien Bien Phu is a city in Vietnam which could pop up in normal conversation, but for the sake of this project pure language examples are required.

To find these sorts of recordings, each recordings' text (found in the XML metadata files) was inspected with a language detection algorithm created by Google, [Shuyo, 2010]. The algorithm simply returns the predicted language of the given text. Even though the algorithm has support for over 50 languages, the only languages supported from South Africa are *Afrikaans*, *English* and *Sesotho*. In order then to still use the algorithm to find discrepancies, it was decided to only flag recordings that are predicted to be *English* if the speakers language was not *English*.

It is assumed that the algorithm will incorrectly predict certain examples, but the decision was made that losing examples outweighs the risk of training the model on incorrect data. To reduce the risk of false negative predictions made by the algorithm, the predicted language will have to have a confidence of over 90% to be accepted. This value is assumed to be high enough to filter out any unwanted predictions. After applying this logic it was found that $65,094$ recordings (64 hours) of the total $1,251,009$ (1369 hours) recordings were flagged.

To ensure that the speech in the recordings are closely linked to the prompted text, only recordings with a high PDP are considered. Because the speech corpus that was released first is ensured to have a high PDP score all of these recordings are kept. The recordings from the auxiliary corpus are only kept if they have a PDP score

greater than 0.2. In order to further clean the data, the duration of the recordings were also limited. First, it was found that there are many recordings with long a duration, such as 20 seconds and higher. Seeing as the longest prompt would only be 5 tokens, these seemed very long and after further inspection it was found that these recordings tended to just be background noise, most likely because of the speaker not realising a recording is taking place. To get a better understanding for the spread of duration for each of the languages, the box plot in Figure 3.1 was analysed.

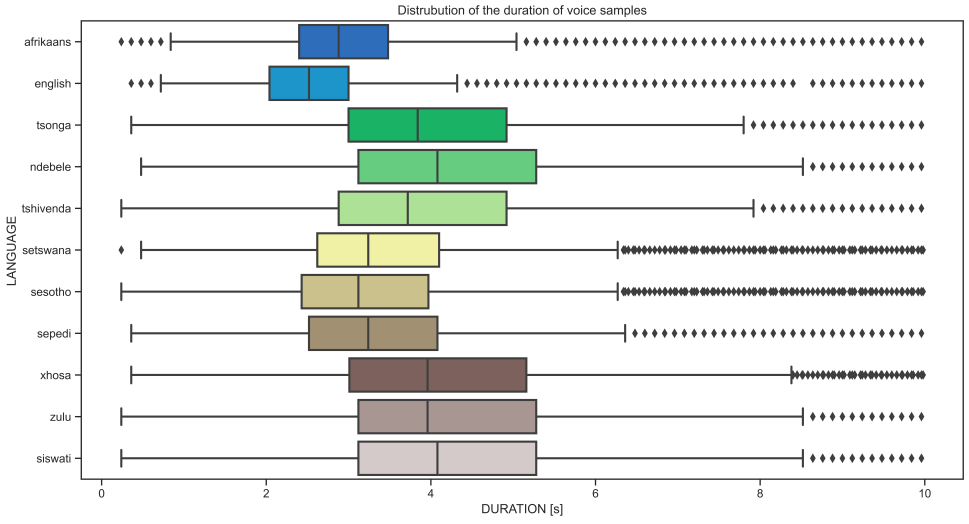


Figure 3.1: Box plots for the duration of recordings after filtering out examples based on the language of text prompts shown and PDP score.

The only recordings considered for the figure were those with a duration of less than 20 seconds. As can be seen in Figure 3.1 all the languages' largest whisker, which is a common cut-off point for outlier detection, are ten seconds or less. There are also recordings with very low durations and it was found that this was due to some prompts being "j j j" for example. Because this research will only focus on short recordings, where the shortest recordings in previous works were found to be three to four seconds in length [Abdullah et al., 2020, Revay and Teschke, 2019], it was decided that only recordings that are between three and five seconds will be kept. This is to reduce the chance of having silent examples when creating the short audio snippets that will be used to train the model.

After all the filtering techniques were applied to the dataset, the final total recording duration for every language can be found in Table 3.2. As seen the non-Germanic

languages lost a high number of hours after the filtering process. The main contribution to this is the high amount of long recordings that were present in these languages, but in the case of *English* the main contribution was the amount of short recordings. Even though these results show a huge number of hours lost, it should be noted that biggest contributions to loss are from the auxiliary set.

Table 3.2: Summary of NCHLT data after filtering out unclean data

Language	***New Duration (in hrs)***	***Difference (in hrs)***
Afrikaans	54	−42
English	31	−53
Ndebele	48	−46
Sepedi	62	−55
Sesotho	54	−73
Siswati	65	−66
Setswana	68	−55
Tsonga	77	−59
Tshivenda	80	−66
Xhosa	79	−77
Zulu	77	−74

Looking at Figure 3.2 one can see that the Germanic languages have shorter durations than the other languages with *English* having very low durations. This will affect the choice of duration for the spectrograms being generated as every language's training sample will have to be the same duration. This is required to ensure that the values in different spectrograms refer to the same amount of time being passed and in doing so will ensure the models look at the same content. In Section 3.1.2 the process of solving this problem will be discussed as well as the methods used to convert the audio data into a format that can be used to train the models.

3.1.2 Transforming Data To the Image Domain

In order for a CNN model to be used, the recording data must be converted from a mono audio input into a spectrogram image, as both [Revay and Teschke, 2019] and [Bartz et al., 2017] did. A spectrogram is a visual representation of the power present at various frequencies over time. It is most commonly displayed as a figure where the y-axis represents frequency, the x-axis represents time and a colour scale represents the magnitude. A spectrogram is calculated by splitting the signal into overlapping segments called frames and applying the Discrete Fourier Transform (DFT) to the frames in order to transform it from the time domain to the frequency domain.

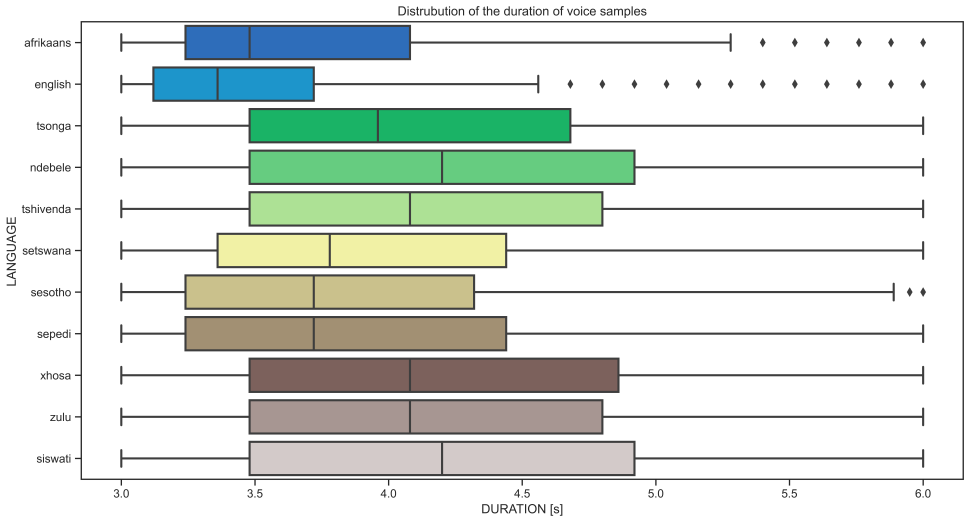


Figure 3.2: Boxplots for the duration (seconds) of recordings for the final dataset

The formula for the DFT is $X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-i2\pi}{N}kn}$, with the inverse being $x_n = \frac{1}{N}\sum_{n=0}^{N-1} X_k e^{\frac{i2\pi}{N}kn}$. For these equations, X_k is the Fourier coefficient at point k, x_n is the signal at point n and N is the number of samples. If one then obtains the complex vector X, the magnitude of the vector components shows the amount of power at each point k. To convert between frequency (ω) and k, one can use $\omega = \frac{2\pi k}{N} f_s$, where f_s is the sampling frequency x_n. For the NCHLT dataset, all audio files have a sampling frequency of $f_s = 16kHz$. By then computing the DFT of each frame and concatenating the results, the spectrogram can be constructed to show how the magnitude at each frequency changes over time. To divide a signal into frames, the signal must be multiplied by a window function to cut off the start and end points.

There are various options of windows, but for this project the window will be the default window used in the Librosa software package, namely the Hann window which is a good choice for most signal processing projects [McFee et al., 2015]. The frame length is also chosen to be 512 samples, which is the recommended value for speech processing [McFee et al., 2015]. This corresponds to 32 milliseconds of data per frame.

Because the data being used is speech data, it is common practice to convert the frequencies from a linear scale to the mel-scale as is done in [Revay and Teschke, 2019] and [Sarthak et al., 2019]. The mel-scale is a non-linear scale that is mimicking the

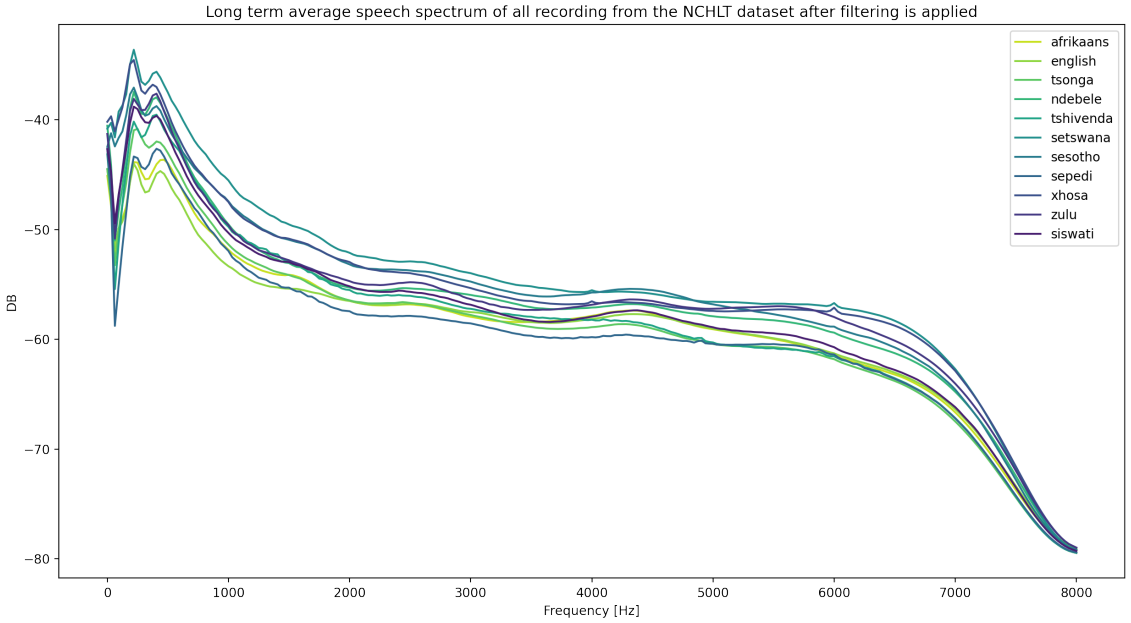


Figure 3.3: Long term average spectrum of all eleven languages showing the average power for all frequencies.

non-linear pitch perception present in the human ear. In other words the mel-scale is defined so that frequencies that sound the same also have the same mel-frequency. The mel-scale is shown in Equation (3.1), with f being frequency. A spectrogram can then be converted to a mel-spectrogram by passing it through Equation (3.1).

$$m = 2595log_{10}\Big(1 + \frac{f}{700}\Big) \tag{3.1}$$

In practice, the frequencies are binned into m mel-frequency bins within the frequency range of the signal. With the recording sampling rate being only $16kHz$ and from the Nyquist Sampling Theorem, which states that a signal can be reconstructed only if the sampling rate is at least twice as high as the maximum frequency present in the signal, we know the NCHLT can only consist of frequencies in the range $0Hz - 8kHz$, as shown in Figure 3.3. In Figure 3.3 the average power over all frequencies of a language are shown, where one can already see the different behaviours of all languages over the frequency bands. For this project, the number of mel-frequency bins will be 128. This was found to be a typical value used in various projects, namely [Han and Lee, 2016, Nam et al., 2012, Nam et al., 2013, Kinnunen et al., 2006], with [Bartz et al., 2017] using 129 bins. These previous projects consisted of deep learning techniques as well as other tech-

niques and this further indicates that 128 bins is a good value to use. Other typical values include 40 bins, but because the languages share certain traits and some share the same language family, it was decided that finer detail will be required.

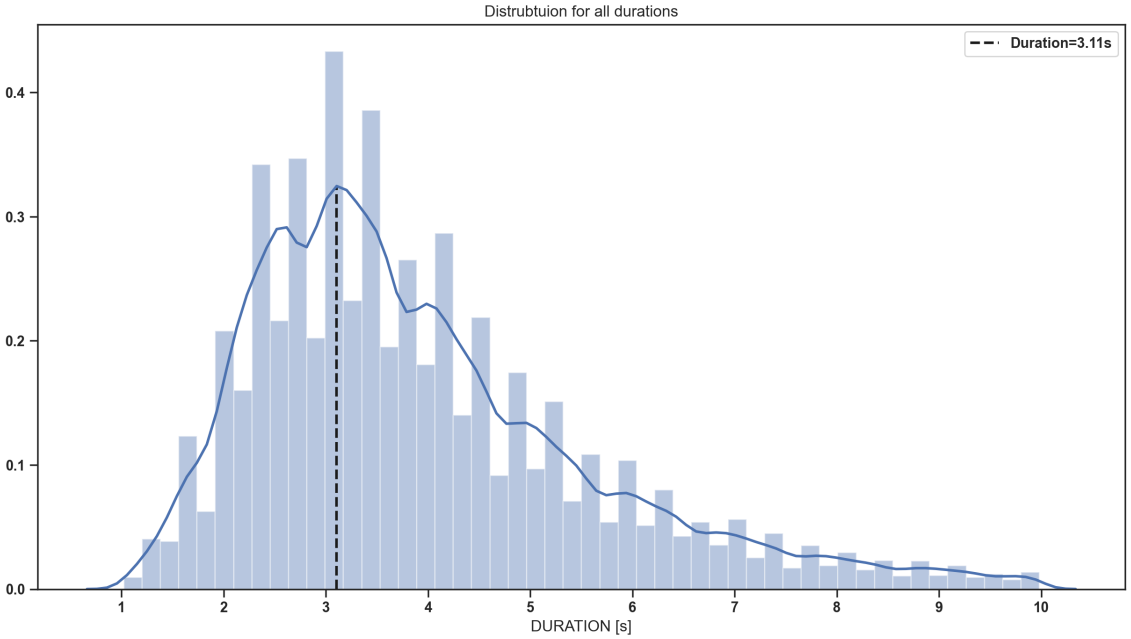


Figure 3.4: Distribution of the duration of all recordings.

In order for the spectrograms to be compared all the recordings must be sub sampled and cut to be the same length to ensure they share the same underlying structure. As previously mentioned the duration of the recordings are all of different length with *Afrikaans* and *English* having the lowest average duration. In Figure 3.4 one can see the duration distribution of all languages combined. It can be seen from the figure that the distribution is skewed to right, with the vast majority of recordings being between 2.5 and 4 seconds. Seeing as all the previous researchers chose one length and developed their models around it [Bartz et al., 2017, Revay and Teschke, 2019, Sarthak et al., 2019], it was decided that for this project only one length will also be used.

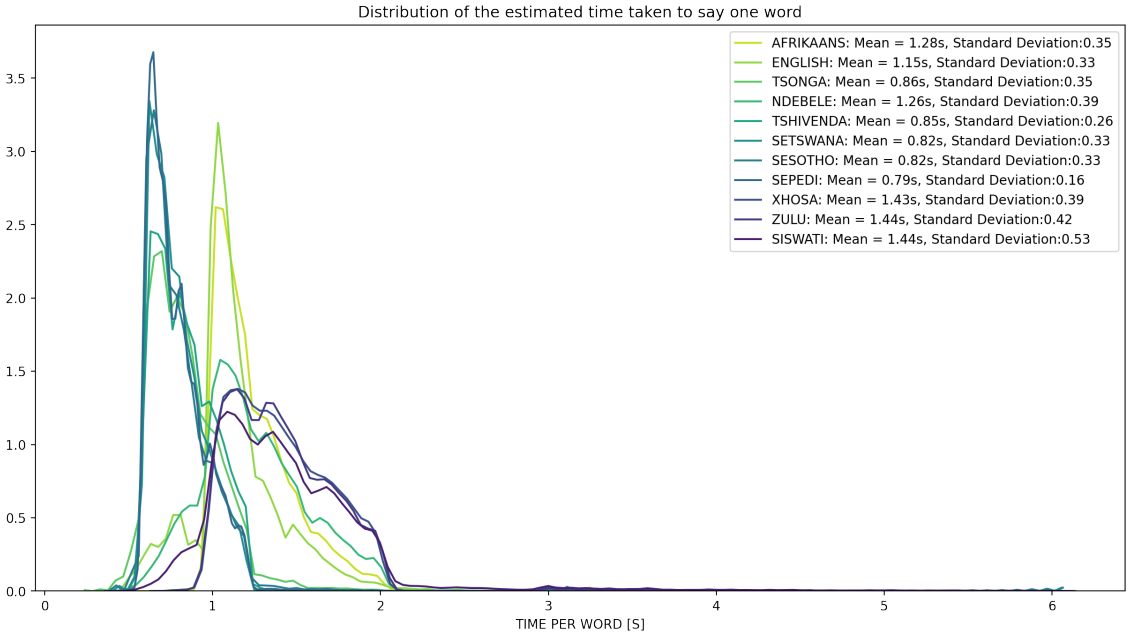


Figure 3.5: Distribution of the estimated time taken to utter one word.

Because the minimum length of the durations after cleaning the data is three seconds, it is decided that all the spectrograms will be limited to the first three seconds of the audio. This choice can be justified by looking at the table contained in Figure 3.5 which shows the average time it takes for a speaker from each language to say one word. One can see that the longest average time is 1.44 seconds, which means that on average the spectrograms will contain at least two words spoken.

If one also thinks about the practical implications for choosing the length of data required, choosing the smallest required time makes the most sense. For example, if a length of six seconds is chosen, the customer facing system that uses this model would require substantially long recordings from the customer to predict the spoken language accurately. This could cause the system having to ask the customer to say something longer instead of just allowing an ordinary sentence to be said. For these reasons, three seconds was chosen as lenght. An example mel-spectrogram, which from now onwards will just be referred to as a spectrogram, can be seen in Figure 3.6. As can be seen the spectrogram length is only 3 seconds and the pixilated visualisation shows the 128 bins of frequency.

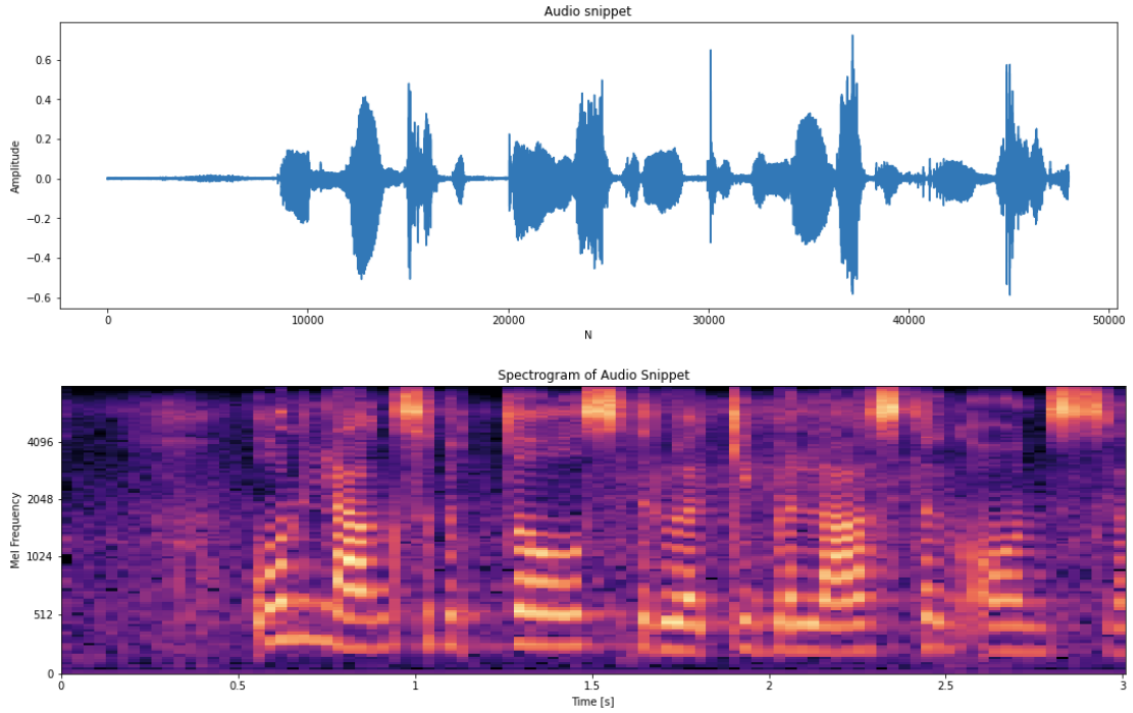


Figure 3.6: Example of a spectrogram created from an audio snippet

3.1.3 Splitting Data Into Train, Validation and Test sets

To test whether the model is overfiting during training, the data will be divided into a train, validation and test set. No cross-validation is performed in this project due to computational limitations. It is assumed here that because the dataset being used is a relatively big and diverse dataset, that the model will be able to generalise to this domain and this will be verified by looking at the results on the validation and test datasets. To further ensure the model does not overfit, regularisation techniques will be applied to the model and these techniques will be further discussed in Chapter 4 and Chapter 5.

To ensure the model does not simply learn what language each individual user speaks, the dataset is divided such that the same speakers are not found in different datasets, as was done by [Mateju et al., 2018, Revay and Teschke, 2019] and [Montavon, 2009]. The splits are also done in such as a way as to ensure that the ratio of men to women stay constant throughout, as well as the distribution of languages. This is to ensure that there is no data bias within the data to ensure that all languages and genders are treated fairly. This results in a training set containing roughly 80% of the data with the validation and test set both consisting of roughly 10% each.

Table 3.3: Languages to be used to train LID models. SB refers to Southern Bantu.

Language	***Total speakers (Million)***	***Percentage***	***Language Family***
Zulu	11.6	23.38	SB:Nguni
Xhosa	8.2	16.53	SB:Nguni
Afrikaans	6.9	13.91	Germanic
English	4.9	9.87	Germanic
Sepedi	4.6	9.27	SB:Sotho-Tswana
Setswana	4.0	8.06	SB:Sotho-Tswana
Total	40.2	81.02	

Ref: [Barnard et al., 2014, pp. 195]

Because the South African corpus of languages contain languages that share the same family language, there is a big overlap between languages and what is spoken. To ensure that this does not interfere with the research aim of this thesis, namely finding a method that generalises better to new domains for language identification, only a subsample of the languages will be investigated. The languages that are chosen is shown in Table 3.3, with the respective number of speakers each of these languages have in South Africa, based on data gathered in 2014 [Barnard et al., 2014, pp 195]. As can be seen from the table, these six languages represent close to 82% of the speakers in South Africa. There are two language families not represented by the chosen languages, namely Tswa-Ronga and Venda. These two families represent 4.63% and 2.41% of speakers in South Africa, [Barnard et al., 2014, pp 195].

Table 3.4 shows the summary of how the data was split. As can be seen the data is fairly evenly split across both genders for all languages in the training set. One can also see that for the validation and test sets, there is a pattern of having more samples for males. This will not impact the conclusions, as the results will be interpreted together as well as per gender to ensure the model is able to perform equally for all genders and languages.

Table 3.4: Summary of train, validation and test datasets.

Language	***Unique speakers***			***Gender split [Female:Male]***		
	Train set	*Validation Set*	*Test Set*	*Train set*	*Validation Set*	*Test Set*
Afrikaans	168	21	21	83:85	12:9	12:9
English	168	21	21	84:84	9:12	7:14
Sepedi	168	21	21	81:87	14:7	5:16
Setswana	168	21	21	86:82	11:10	12:9
Xhosa	167	21	21	92:75	7:14	7:14
Zulu	168	21	21	83:85	5:16	10:11

3.2 Lwazi Speech Corpus

As was mentioned in the previous section, the Lwazi speech corpus was created before the NCHLT speech corpus and it was the first corpus to include all the South African Languages. It is substantially smaller than the NCHLT corpus, as it only contains between four to ten hours per language. Whereas the NCHLT corpus was gathered with the Woezella application, the Lwazi dataset was captured by recording either a landline or cellphone call. The NCHLT dataset recordings were also sampled at $16kHz$ [Barnard et al., 2014] while the Lwazi recordings are only sampled at $8kHz$ [Van Heerden et al., 2009].

According to the Nyquist Sampling Theorem, the highest voice frequency present in the Lwazi recordings are only $4kHz$, compared to the $8kHz$ in the NCHLT dataset. To get the spectrograms generated from the Lwazi dataset into the same range as the spectrograms generated from the NCHLT dataset, the audio will be resampled at $16kHz$ when read in for processing. This will ensure the same frequency bins and time steps are present in the spectrograms, but the data present in these bins will not be 100% accurate. For more information on the artefacts that could occur from upsampling, please refer to [Pons et al., 2020]. This will have to be taken into consideration when the results on the Lwazi dataset are investigated in Chapter 6. Future work, as mentioned in Section 7.1 also, will need to investigate the possibility of finding two South African domains with the same sampling rate.

In Table 3.5 one can find a summary of the dataset, [van Heerden et al., 2016]. As can be seen, the amount of unique speakers are close to the same amount as the NCHLT dataset with the gender split nearly down the middle for all of the languages.

Table 3.5: Summary of the Lwazi Speech Corpus

Language	***Unique speakers***	***Gender split [M:F]***
afr	200	101:99
eng	196	92:104
nbl	200	99:101
nso	190	92:98
sot	202	90:112
tsn	203	96:107
tso	214	103:111
ssw	196	103:111
ven	198	98:100
xho	210	101:109
zul	199	98:101

Whereas the NCHLT corpus was filtered and cleaned in order to provide good examples for the model to learn from, the Lwazi corpus will be used as is when testing the model. This is to simulate a real world environment where the data is not always perfect. The only filtering technique applied is to only consider recordings that are between three and eight seconds in length, so that no recordings are fed into the system that do not contain speech. The assumption is that the labels associated with the recordings are correct. In Table 3.6, one can find the final numbers corresponding to the Lwazi data that will be used. As mentioned in Section 3.1.3, only six languages will be used in this project. The dataset can again be obtained from the Sadilar website. One will find various versions of the dataset on the website, but for this project the Lwazi project was used and not the Lwazi II or Lwazi III corpora.

Table 3.6: Summary of the Lwazi Speech Corpus used to test the model trained on the NCHLT speech corpus.

Language	***Unique Speakers***	***Duration [H]***	***Number of Samples***
afr	200	2.00	1931
eng	195	2.58	2380
nso	185	1.00	898
tsn	200	1.61	1456
xho	210	1.62	1470
zul	197	1.20	1089

The next chapter will explain the various methods that will be introduced in to create an LID system that can be trained on only one domain but still generalise to new domains.

Chapter 4

Methodology

This chapter will discuss the methodology behind the methods used to generalise an LID system to new domains. The chapter will consist of three sections, where the first section gives an overview of how CNN architectures are able to extract features from an image. This will be followed by an overview of the novel Triplet Entropy Loss (TEL) training method where the last section discusses the additional methods introduced to improve the generalisation of LID systems.

4.1 Convolutional Neural Networks explained

This section will explain the architecture and methods used to extract features from spectrograms, but the work is based on prior knowledge of neural network architectures and techniques. If the reader wishes to learn more about other architectures and techniques (such as regularisation and overfitting), more information can be found in [Goodfellow et al., 2016].

Firstly, it is important to realise that an image is a matrix of numbers, where the numbers show the intensity of a pixel. Typically, these values range between 0 and 255. When the image is gray scale, the matrix is one-dimensional, but when the image has colour, the matrix is three dimensional. These three dimensions represent the intensity of each pixel within in the red, green and blue colour channels. These dimensions are also known as the channels of the image.

Whereas in feed forward neural networks [Rumelhart et al., 1985] the data goes through various matrix multiplications until it reaches the end, the CNN rather performs convolution operations on the data as it moves through the network. Along with these convolution operations, the fundamental building blocks of a CNN model are convolution layers, non linearities, pooling layers and fully connected layers (the

prediction layer). These layers are differentiable, which allow the CNN architecture to be trained using backpropagation.

The most important layers in a CNN, and from where it gets its name, are the convolution layers. It is in this layer where the CNN is extracting features from an image. This is done by sliding a kernel (or filter) over the input matrix. For every position of the sliding process, the element wise multiplication is calculated for the overlapping regions between the matrix and the kernel and then these multiplication results are added together. After the kernel has moved through the entire input, the resultant matrix is known as a feature map. If more than one kernel is applied to the input, the resulting feature maps are concatenated depth wise. Thus, if a 3 channel image is convolved by p kernels, the feature map will have p channels.

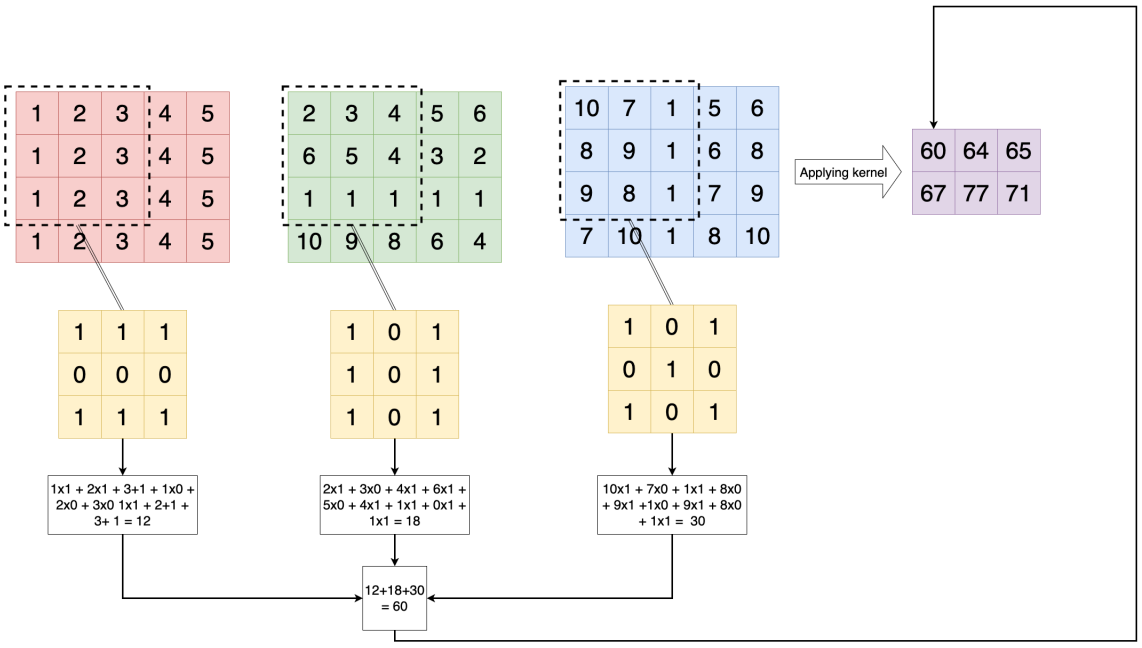


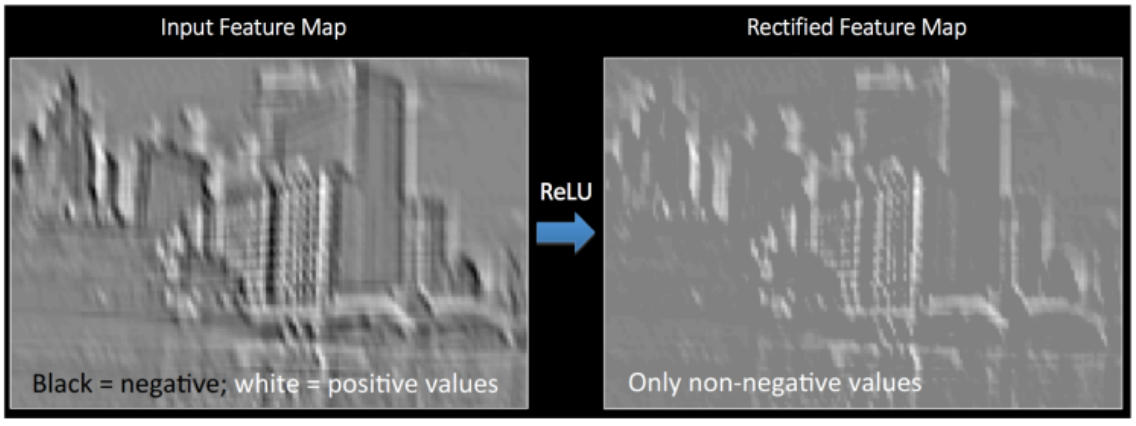
Figure 4.1: An example of a three channel matrix of shape being convolved with a kernel of shape $3 \times 3 \times 3$.

The kernel will always have the same number of channels as the input matrix, but will be substantially smaller than the input, with typical sizes being 3×3 up until 7×7. For every convolution layer, the model designer will have to decide what the size of the kernel is and how many kernels are to be used. One of the other hyperparameters in this layer is the number of elements that a kernel should move, for instance one can go move one column/row at a time (stride=1) or any other number. The kernel will also move from left to right, and then move down and

continue from left to right again. The kernel weights, which are the values of the kernel matrix, are learned by the model and these weights extract features from the image.

In Figure 4.1 one can see an example of a three channel matrix of shape 4×5 being convolved with a kernel of shape $3 \times 3 \times 3$. The matrices are broken up into their separate channels to make it more clear what elements are multiplied with what elements. The figure shows in the bottom how one of the new features in the feature map is calculated and the final feature map is shown to the right. In this example, not all the weights in the kernel will be seen by some of the input elements as the size of the input prohibits that movement. A technique to avoid this is to apply padding to the input to increase the size of the input. In most cases the padding values will be zero.

This convolution layer makes CNNs a good fit for spatial data. The fact that a kernel moves over the entire image with the same parameters means a CNN is spatial invariant. For instance, if the kernel is structured in a way to pick up cat ears, it will not matter if the image contains ears in the middle of the image or in the corners as the kernel will still pick them up. Also, seeing as the kernel uses the same parameters over the entire input, it can be said that CNN uses parameter sharing. This reduces the amount of training required substantially.



Source: [Fergus, 2015]
Figure 4.2: An example of ReLU being applied to a feature map.

The next step in the CNN pipeline is to add non-linearity to the system. Because the real world is non-linear and convolution layers are a linear action, non-linearity must be introduced to the system. This can be achieved by applying any non-linear function to the feature maps created by the kernels. Typical functions include

Rectified Linear Unit (ReLU) [Nair and Hinton, 2010], Tanh and Sigmoid. But in practice it has been found that ReLU tends to give better results. The ReLU makes all negative values zero and keeps positive values the same. An example of ReLU being applied to a feature map can be found in Figure 4.2.

As the network grows bigger, more and more computations will be required which can make the model slow and take up a lot of memory. This impact can be reduced by pooling the feature maps. This involves specifying a window (usually a square shape) and sliding this over each feature map channel separately and reducing the elements in the window to just one element. Typically, the maximum value or the average of the window is taken. Pooling also ensures that minor changes in an image will not affect the predictions, acting as a regularizer. For deep In figure Figure 4.3 one can see an example of max pooling, with a window of size 2×2 being applied to a matrix with a stride of two. The matrix is coloured to show the locations where the window will be applied, with the resulting pooled value shown in the same colour.

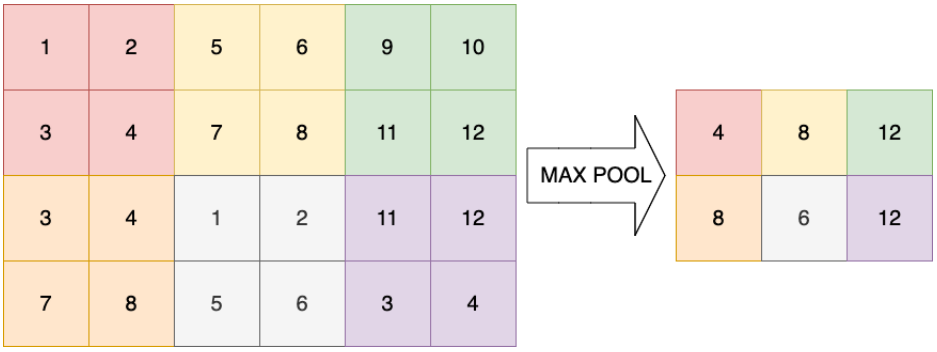


Figure 4.3: An example of max pooling, with a window of size 2×2 being applied to a matrix.

The previous three steps are usually repeated and grouped together for a few layers. At the end of all these layers, the final feature map is flattened into one column vector and passed through a feed forward neural network where these newly created features will be used to make a prediction.

A CNN is trained using the same backpropagation algorithm that is used in fully connected neural networks. This algorithm consists of repeating a forward and backward phase repeatedly until convergence or another criteria is met. In the forward phase the loss of the model is calculated for a given input and in the backward phase the weights, biases and kernel values are updated using gradient descent.

To train the model one will first initialise all parameters by assigning random values drawn from a distribution, with most cases using the Gaussian distribution. After

this, the model is fed an input image that propagates through the network and produces a result, $\hat{y}$. The loss for a given input is calculated using the loss function in Equation (4.1). In this case, the loss function is the cross-entropy function where y_i will be 1 for the true class and 0 for other classes, with $\hat{y}_i$ being the predicted probability of the class i. In some literature the cost function is used to refer to the average loss for all inputs, but for the sake of clarity in this explanation the cost and loss function will mean the same thing, i.e. the error of the model.

$$L = -\sum_{i=1}^{C} y_i \log \hat{y}_i \tag{4.1}$$

The derivative of the loss function L is calculated with regards to the model parameters during the backward phase. For the CNN, these parameters are the kernel weights $\boldsymbol{K}$, the biases added to kernels $\boldsymbol{b_k}$, the weights of the fully connected layers $\boldsymbol{W}$ and the bias terms introduced in these layers $\boldsymbol{b_k}$. These parameters will be represented by the symbol θ, thus the loss function in terms of the parameters can be written as $L(\theta)$. The derivative of the loss function is used to update the parameters and this process is called gradient descent.

Gradient descent is an optimisation algorithm that is used to minimise a function iteratively. The algorithm seeks to move in the direction of steepest descent of a function to find a local minimum. This means it is not guaranteed that the algorithm will find the best solution. The direction of steepest descent is in the direction of the negative of the gradient. In this case, the gradient is calculated as $\frac{\partial L}{\partial \theta}$. There are various ways in which the weights can be updated with the derivative, but for now only mini-batch gradient descent will be explained.

In mini-batch gradient descent, the model parameters are updated for every mini-batch of data containing n examples. The algorithm finds how the parameters should change for each of the n examples and then averages the result of each parameter and updates the parameters with these values. If the model had to update the parameters based on the entire batch of data, then there will be redundant computations as it will calculate gradients for similar examples before updating the parameters, [Ruder, 2016]. Mini-batch solves this problem by updating much more frequently with a higher variance than batch gradient descent. By also using mini-batches instead of just one example, the computations are more stable. In many software packages Mini-batch descent and Stochastic gradient descent (SGD) are interchangeable, [Ruder, 2016].

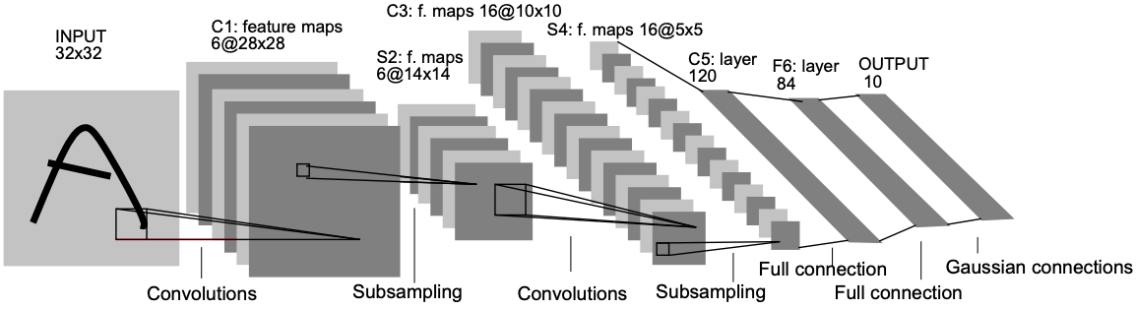
The equation for Mini-batch descent is shown in Equation (4.2). One can see that the new parameters, θ_{new}, are calculated by updating the old parameters, θ_{new}, by

subtracting the gradient calculated on a batch of size n. This is indicated by the loss function being based on the input values $x_{i:i+n}$ and the outputs $y_{i:i+n}$. One will also see the parameters are updated based on a weighted version of the gradients, as they are multiplied by the learning rate α. The learning rate is another hyperparameter the model designer will have to choose. Other versions of gradients descent will also change α over time. By updating the parameters over several epochs, where an epoch is one full loop over the dataset, the model parameters will converge so that a local minimum is found for $L(\theta)$

$$\theta_{new} = \theta_{old} - \alpha \nabla_\theta L(\theta; x_{i:i+n}, y_{i:i+n}) \tag{4.2}$$

4.1.1 Well Known CNN Architectures

In Figure 4.4 one can see the earliest CNN architecture, namely Lenet-5. This architecture was made in 1998 by [LeCun et al., 1998] and this structure is the blueprint for many future architectures. The input is a gray-scale image, with a shape of $32 \times 32 \times 1$. The input image then goes through six kernels of size 5×5, resulting in a feature map of size $28 \times 28 \times 6$. Layer 2 is then a pooling layer where the dimensions are reduced to 14×14. Then again, a convolution layer is applied with 16 kernels of size 5×5. The output is then pooled again before being fed into a convolution layer of 120 kernels with size 5×5.



Source: [LeCun et al., 1998, pp. 7]
Figure 4.4: Architecture of Lenet-5 model developed in 1998.

This is shown as a fully connected layer in the image because the kernel is the same size as the input features, thus they are fully connected, but if the dimensions of the input change it will not be fully connected. This is why it is still referred to as convolution layer. The resulting features map is then flattened and fed into a fully connected network with one layer and an output layer with sizes 84 and 10.

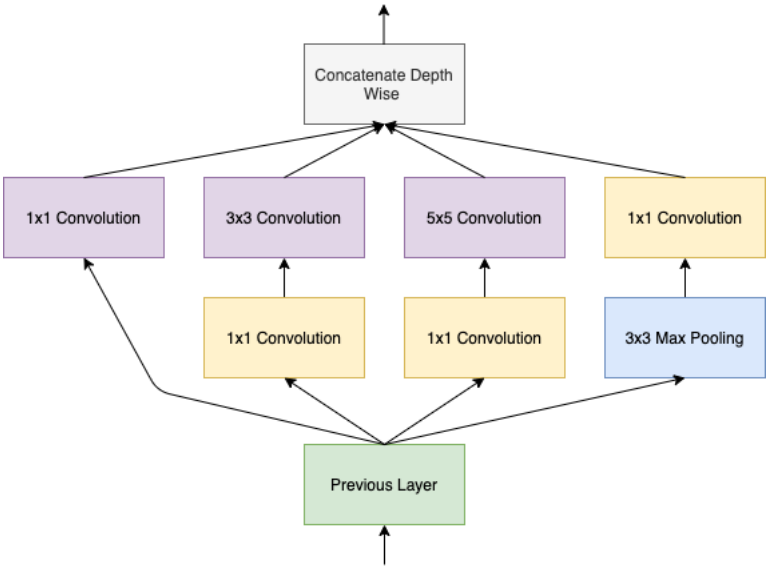
All activation functions are tanh expect for the final layer which has a Softmax activation. The Softmax function is given in Equation (4.3), where z is the output vector of the layer. The function transforms and normalises the vector values such that the output vector consists of values between 0 and 1 and that that the vector values sums to 1. The output of the Softmax function could then be interpreted as the probability of the input belonging to a class.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{4.3}$$

Since the Lenet-5 architecture there has been many model architectures proposed and designed. Two of the most famous architectures are the Inception-V3 and Resnet architectures. There is an Inception-V4 designed as well, but as of the time of writing the TensorFlow package does not support V4 yet, thus the focus will be on V3.

Inception-V3 is an improvement on the Inception-V1 architecture that came out in 2014. Before Inception, models tended to be improved by adding more layers. This bring issues such as the huge number of parameters that have to be trained. Another issue is the location of information in figures can be drastically different in different images. For example, a dog in one image can take up almost the entire image but in another example the dog can only be seen in the corner of the image. This means the number of pixels taken up by the desired information is varying. The choice of kernel size is then a hard choice to make as bigger kernels tend to perform better on information that is found in large amounts of pixels [Szegedy et al., 2014].

In order to solve these issues, [Szegedy et al., 2014] thought of making a sparsely connected CNN architecture. They achieved this by making the architecture wider rather than deeper. By making the architecture more wide (sparse), each layer can consist of different size kernels that manage to pick up different information in the image [Szegedy et al., 2014]. An example of an Inception-V1 layer is shown in Figure 4.5. One will see that there are various kernels of different sizes spread out horizontally rather than vertically. The bigger kernel sizes (3×3 and 5×5) are preceded by 1×1 kernels. This is in order to reduce the number of dimensions before feeding them into the bigger kernels to allow less parameters to be trained. The generated feature maps of the various kernels are then concatenated depth wise, where the feature maps are padded to have the same shape. The concatenated version then gets send forward in the network.



Source: Altered from [Szegedy et al., 2014].
Figure 4.5: Example of a filter bank within the Inception-V1 architecture.

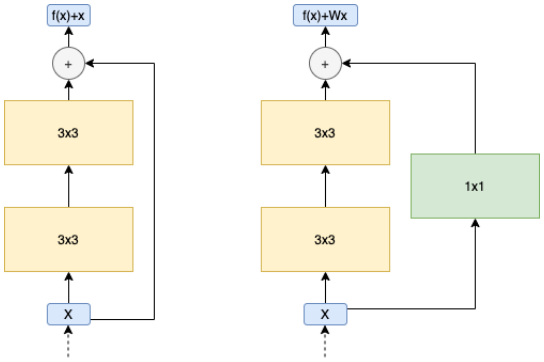
In order to improve upon this original version, [Szegedy et al., 2016] introduced a few new concepts and altered the architecture slightly. The first thing the researchers altered was to reduce the extent to which the dimensions change between convolution operations as they found by doing this that the model performed better. The researchers also aimed to reduce the computation time of the model.

To make the model less computationally expensive, the researchers introduced a method of factorising the 5×5 kernels into two cascading 3×3 kernels. This is done because they found that 5×5 kernels are 2.78 times more computationally expensive than two 3×3 kernels, [Szegedy et al., 2016]. They also introduced another factorising method where they represented a $n \times n$ kernel as two kernels of size $1 \times n$ and $n \times 1$ kernels. This method was only applied to the kernels of size 7×7 in the architecture. The researchers also expanded some filter banks (groups of kernels) by making them wider.

These three techniques are then used in different filter banks throughout the architecture. The model was also trained using the RMSProp optimiser with batch normalization used as regulation technique, [Szegedy et al., 2016]. The final model consisted of 24 million parameters and was 48 layers deep. The model achieved state-of-the-art performance on the Image-net dataset obtaining an 3.58% error rate for the top-5 predictions.

When [He et al., 2016] tried to test the theory that better results can be achieved

by simply making networks deeper, they kept running into the vanishing gradient problem. This is a problem where the gradients that are fed backwards in the network become very small with the more layers they pass through. This results in the parameters not being updated and thus the network not learning. The researchers aimed to solve this problem by introducing a deep residual learning framework. This means that certain layers will provide skip connections to following layers, meaning the input gets added to output at certain layers. Instead of those layers learning the mapping $f(x)$, these layers will now learn the mapping $f(x) + x = h(x)$ [He et al., 2016]. This allows the gradients that are fed back to have "shortcuts" to previous layers and thus help with the vanishing gradient problem. It also means the model in theory only has to learn what to add to x instead of complex mappings, which is the concept that [He et al., 2016] used as an argument for adding the residual block.



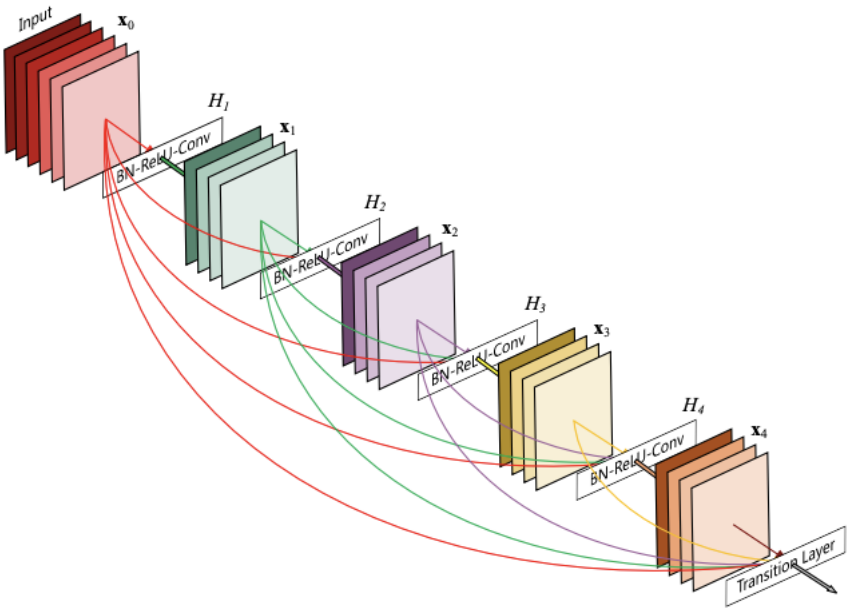
Source: Altered from [He et al., 2016].
Figure 4.6: Example of two different residual blocks used in the ResNet architecture.

In the ResNet architecture, the shortcut connection mostly connects the input and output of two convolutional layers. These convolutional layers all have kernels of size 3×3. The architecture also only uses pooling in the input layer to reduce dimensions, but uses a stride of two instead of one in deeper layers to achieve the same affect. When the dimensions of the input and the output are different in a residual block, a 1×1 convolutional layer is applied to x to make the dimensions of $f(x)$ and x the same. The two versions of the residual blocks can be seen in Figure 4.6. The block on the left shows the normal residual block where the input gets added to output of two 3×3 kernels. The block on the right is an example of how the input's depth dimension gets altered by an 1×1 kernel in order to allow the summation to take place when $f(x)$ alters the dimensions of x. After all these residual layers, the ResNet architecture only appends one fully connected layer at

the end, with the size depending on the number of classes that are predicted.

There are various versions of a ResNet model that were tested, most notably ResNet-32, ResNet-50, ResNet-101 and Resnet-152. The numbers in the name indicate the amount of layers that are present in the network. The ResNet architecture also makes use of batch normalisation to regulate the model. The optimiser used in training is SGD plus momentum (0.9) and weight decay (0.0001). The learning rate starts at 0.1 and gets divided by ten after the validation accuracy plateaus. The batch size used was 256 and the image input dimensions are 224×224, [He et al., 2016].

In 2018 [Huang et al., 2018] released a new architecture, Densenet, again building on previous innovations made. The researchers build upon the work that showed that CNNs can go much deeper by adding in residual connections [He et al., 2016]. Whereas Resnet connected one layer to the next layer, Densenet connects a layer to all of its following layers. The researchers also show that by doing this they reduce the vanishing gradient problem, they increase the amount of information that gets propagated through the network and they reduce the number of parameters in the network substantially [Huang et al., 2018]. In Figure 4.7 one can see an example of the Densenet architecture. [Huang et al., 2018] trained their models with SGD and batch size of 64, applying a decaying learning rate that starts at 0.1.



Source: [Huang et al., 2018, pp. 1]
Figure 4.7: Example of how a Densenet architecture will look.

The Resnet and Densenet architectures explained above as well as the CRNN architecture developed by [Bartz et al., 2017] will be used as feature extractors in the LID systems discussed further in the thesis. In Section 4.2, one can find how these architectures are utilised as feature extractors in the Triplet Entropy Loss training method.

4.2 Triplet Entropy Loss

In multi-class classification networks, the most commonly used technique to train a network is to generate one-hot encoded vectors from the class labels and use Cross Entropy Loss (CEL) to calculate the respective loss value. A one-hot encoded vector is a binary vector representation of the label representing the input data. The vector is created by assigning each unique label an integer and then for the corresponding one-hot vector, the value is 0 everywhere except at the assigned integer index, where it is 1. The formula for CEL is shown in Equation (4.4) where y_i^c is the probability that the ith observation belongs to class c (either 1 or 0), with $\hat{y}_i^c$ being the predicted probability of the ith observation belonging to class c. By optimising the network using Equation (4.4), the weights are optimised in such a way as to improve the estimated probability distribution of the network, such that the correct class has the highest probability output given the input features X.

$$CEL = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_i^c \log \hat{y}_i^c \tag{4.4}$$

The downside with this loss function is that it only penalises the output of the class under consideration as y_i^c is zero for all cases where y_i^c is not the target class. Even if training a model using mini-batch gradient descent, where the weights are optimised based on the average loss of the batch which contains multiple classes, the individual losses used to calculate the average still only considers the correct class prediction and not the overall class interaction for that pass through. Even if the Softmax function is used as a predecessor to the CEL, where the output is calculated by looking at all the class outputs, the final value used in CEL is still just the Softmax output for the given class which is not a reflection of the interactions between the classes for that specific prediction.

For tasks such as language identification where the input data that is present in various classes, such as someone speaking a mix of words from different languages, it will be more optimal to have a loss function that interprets interactions between classes. The loss must optimise the network by learning these interactions between

classes to generalise better to the instances where there is a tiny threshold between the classes.

A loss that loosely fits the above description is the Triplet loss function used in [Bredin, 2017], [Margolis et al., 2018] and [Mingote et al., 2019] for language identification tasks, discussed in Section 2.3. By using the Triplet loss function, the weights are being optimised by comparing different class embeddings with one another and optimising the distance between the embeddings such that different classes are far from one another. The model can then learn characteristics for all the classes and in doing so could be able to better learn the fine connections between the languages *English* and *Zulu* for example. But Triplet loss does not optimise for prediction capabilities directly.

We present the novel training method Triplet Entropy Loss (TEL) that leverages both the strengths of CEL and Triplet loss. [Khosla et al., 2020] implemented something very similar for image classification, as they pre-trained a network using supervised Contrastive loss, whereafter they fine tune the model for classification tasks. The TEL method does not contain a pre-training step, but trains simultaneously with both CEL and Triplet loss, as shown in Figure 4.8. As seen, the final embedding layer feeds into two separate layers where each of these output layers are connected to two different losses.

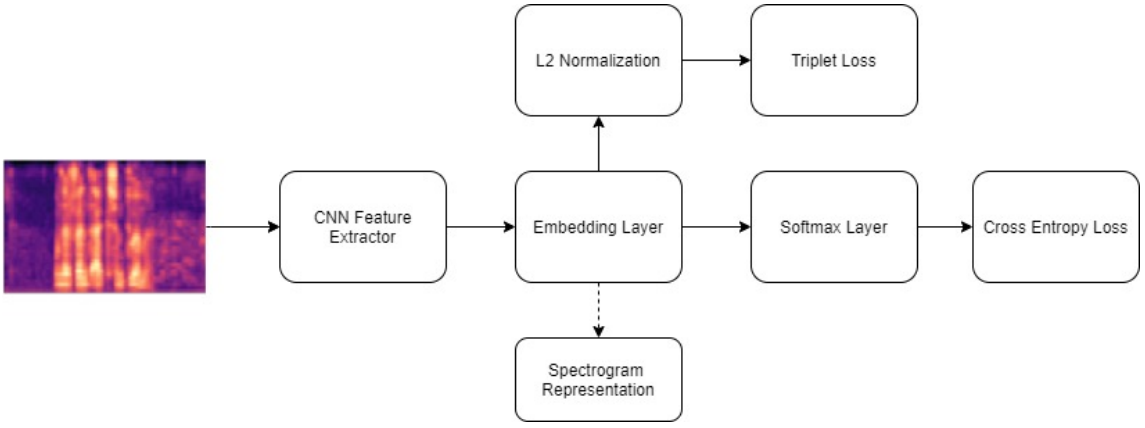


Figure 4.8: Triplet Entropy Loss high level overview

TEL can be represented by Equation (4.5), with σ being the Softmax function and $g()$ the final classification layer. N is the number of examples in the batch being passed through the network. The embeddings generated for the anchor, positive and negative triplets are given by $f(x_i^a)$, $f(x_i^+)$ and $f(x_i^-)$.

$$
\begin{aligned}
TEL &= \sum_{i=1}^{N} CEL_i + TL_i \\
&= \sum_{i=1}^{N} -y_i^a \log \hat{y}_i^a + \left[\left\| f(x_i^a) - f(x_i^+) \right\|_2^2 - \left\| f(x_i^a) - f(x_i^-) \right\|_2^2 + \alpha \right]_+ \\
&= \sum_{i=1}^{N} -y_i^a \log \sigma(g(f(x_i^a)) + \left[\left\| f(x_i^a) - f(x_i^+) \right\|_2^2 - \left\| f(x_i^a) - f(x_i^-) \right\|_2^2 + \alpha \right]_+
\end{aligned}
\tag{4.5}
$$

Looking at Equation (4.5), it can be seen that the model is being optimised to generate embeddings for $f(x_i^a)$ that are strong at predicting the correct class while still ensuring that the interactions between other classes are not ignored. For example, if the weights for the given batch produce high probabilities for the correct class, but do not separate the classes well in the embedding space, the model will still be penalised to further separate classes. In doing so the model will learn what are the features that truly distinguish the classes. This will make the model more effective in new domains because by increasing the distance between classes, it is assumed that less precise predictions made on new domain data will still have room for error and still produce well separated embeddings.

During training of the network, it is very important that triplets are chosen in such a manner that they are "hard" for the network. In other words, if the only triplets being chosen are instances where $\left\| f(x_i^a) - f(x_i^+) \right\| + \alpha < \left\| f(x_i^a) - f(x_i^-) \right\|$, then the network will learn nothing as the contribution to the loss (from the triplets) will be 0. To ensure this does not happen, the triplets generated and fed into the loss will be mined in each batch such that $\left\| f(x_i^a) - f(x_i^-) \right\| < \left\| f(x_i^a) - f(x_i^+) \right\|$. This is known as semi-hard negative online mining and was found to lead to better local minima discovery and help in reducing the chance that the model collapses and predicts all embeddings to be $f(x) = 0$ [Schroff et al., 2015]. In Figure 4.9 one can see the different types of negatives that can be mined for, but as mentioned this research will use semi-hard negative triplets.

Source: https://omoindrot.github.io/triplet-loss

Figure 4.9: Visual explanation of different triplet negative pairs that can be mined for during the training process. For this research semi-hard negatives is used.

It is also assumed that by using TEL the embedding layer is optimised in such a way that the generated embeddings will give an accurate description of the language spoken, especially in cases where it is not someone's first language. For example, speakers that mix their languages frequently, such as *Afrikaans* speakers using a lot of *English* words, should appear closer to one another than speakers that that only use *Afrikaans* words. Another example could be that a famous *Afrikaans* news host's embeddings should not be close to a speaker learning how to speak *Afrikaans*, but both should still form part of the cluster of *Afrikaans* speaker.

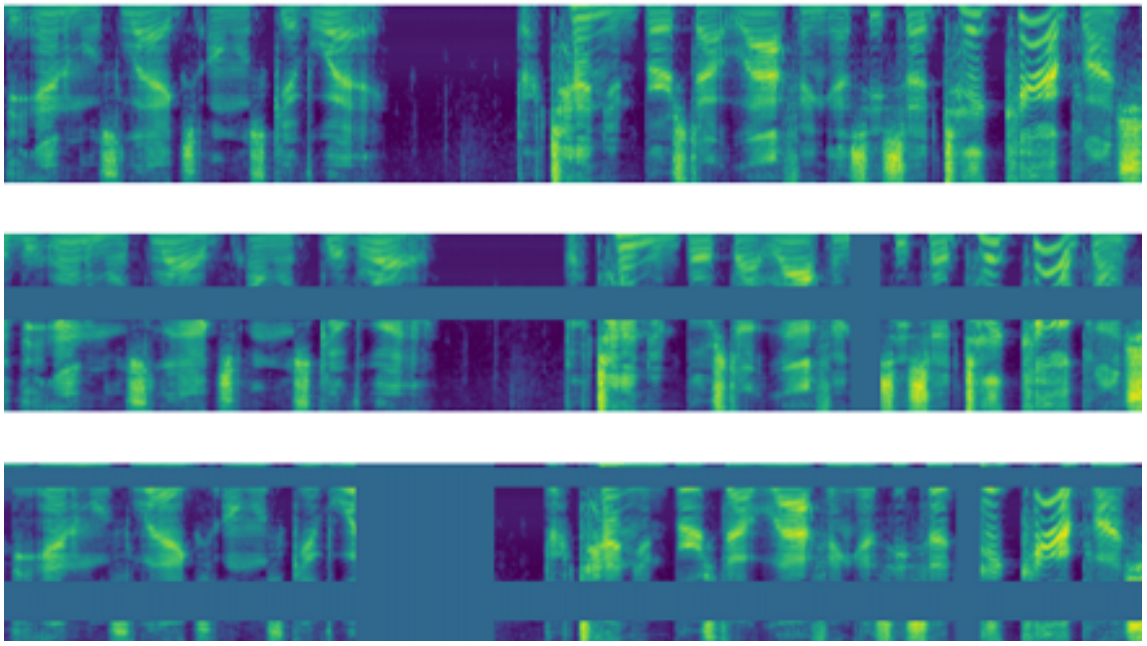
Using TEL, predictions can be made with a model trained on the generated embeddings or by using the same technique such as [Schroff et al., 2015] where the distance between embeddings can be used with a voting system implemented. Future work could look into pre-training a network using TEL and then fine tuning the model for classification tasks. To see the performance of the novel TEL method, please refer to Chapter 5 and Chapter 6.

4.3 Additional Methods

To further improve the generalisation of the LID system, that is improving the performance on new speakers or new datasets, spectral augmentation is investigated as well as the effect pre-training a CNN architecture on Imagenet will have.

4.3.1 Spectral Augmentation

Spectral augmentation comprises masking the spectrogram image horizontally or vertically and sometimes in both directions [Park et al., 2019]. Spectral augmentation is applied directly to the spectrograms during the training phase, as was done by [Park et al., 2019].



Source: [Park et al., 2019, pp. 2]
Figure 4.10: Example of augmentation applied to spectrograms. In the top spectrogram no augmentation is applied, where in the middle one can see a horizontal augmentation and then lastly in the bottom a mixture of augmentations are applied.

An example of this can be found in Figure 4.10. In the top spectrogram no augmentation is applied, where in the middle one can see a horizontal augmentation and then lastly in the bottom a mixture of augmentations are applied. The researchers obtained state-of-the-art performance on the ASR dataset LibriSpeech [Panayotov et al., 2015], using this technique [Park et al., 2019]. The researchers [Korkut et al., 2020, Han et al., 2020] both implemented Spectral Augmentation, with [Korkut et al., 2020] indicating that the accuracy of language identification systems increase when using it.

4.3.2 Pre-Trained CNN Architectures

[Palanisamy et al., 2020] showed that CNN architectures pre-trained on the Imagenet dataset serve as a good baseline for training audio classification models. ImageNet is a database containing images representing all of the nouns in the WordNet database [Deng et al., 2009]. It is one the largest image datasets and contains roughly 3.2 million images and over 1000 classes. In order to test how well this translates to the prediction of South African languages, the experiments performed will also feature the results for when the models are pre-trained. Even though [Palanisamy et al., 2020] experimented with which layers to freeze during the training process, this work will allow all layers in the model to be trainable during the training process. This is done to reduce the amount of experiment runs that need to happen. It is then assumed that this will not affect the results greatly, but is left as future work to investigate further.

To test the TEL method, three different models will be trained with three different methods to compare the results. The three models are the CRNN model used in [Bartz et al., 2017], a Resnet-50 model and a Densenet-121 model. The Resnet-50 and Densenet-121 models will be tested twice, once where there is no pre-trained weights and a second time where the weights are pre-trained on Imagenet. The three different training methods will be using Triplet loss, CEL and then using the TEL method of training. The results of these tests can be found in Chapter 5.

Chapter 5

NCHLT Results

In order to test if the TEL method truly results in better performance, three models will be trained in three different ways. The three models are the CRNN model used in [Bartz et al., 2017], a Resnet-50 model and a Densenet-121 model. All three of the models will be trained using the TEL method as well as CEL and Triplet loss separately. Because of the constraints in compute resources and time, there will be no cross validation performed, but validation of the results are obtained using a hold out set.

The models trained with no spectral augmentation, discussed in Section 4.3, will serve as a baseline for that respective architecture and training method. The success of these experiments will be measured with the accuracy metric and by observing the performance for each language separately by inspecting the confusion matrix. A confusion matrix is a matrix where each column represents the instances in a predicted class the rows represent the instances for the true class. Thus the diagonal of the matrix represents the amount of correct predictions for the class represented by the row and the off diagonal values represent the number of incorrect predictions and what the incorrect prediction was. With the confusion matrix, the confusion that appears between languages can also be observed. Along with this, the embeddings produced will be inspected visually. The performance of the model will be measured with all samples combined and how the model performs for each gender separately. This is to ensure the model is not biased towards any gender. The gender associated with each recording is taken directly from the metadata provided in the NCHLT dataset.

The various models will be trained on a single g4dn.xlarge Amazon Web Service (AWS) EC2 instance. This is not an ideal training environment as the amount of vRAM (16Gb) limits the batch size that can be used. One can also only run a single

experiment at a time and this limits the amount of results that can be gathered as there had to be a trade-off between volume of results and overall cost for the research. The Tensorflow version used is 2.3.0 and the tensorflow-addons package version, used to calculate the Triplet loss, is 0.11.2.

The following section explores how the training methods compare to one another. After that, a thorough analysis into the embeddings generated by the different training methods takes place in order to further understand what the models learn during the training process.

5.1 Comparing the Different Training Methods

To ensure the optimiser does not have a big impact on the results, the models will be trained using the Adam optimiser [Kingma and Ba, 2014] with a learning rate of 0.0001. Due to the computational limits, the optimiser can not be fine tuned as the costs will be too high. It is left as future work to investigate the optimiser choice further. Adam was chosen as optimiser as it requires less hyperparameter tuning than SGD with momentum for example. To reduce overfit within the models, the feed forward layers for all the models and the convolutional layers for the CRNN model contain a regularisation penalty on the kernel weights. This adds an additional penalty to the weights of the model during training, ensuring that the weights of the model do not grow to big. This improves the generalisation of the model as it would not become to reliant on certain weights.

This ensures the weights have smaller values and penalises weights with bigger values, like one would penalise high parameter values in a linear regression model. The L2 penalty is 0.001 for all models. This is the same value as what was used in [Bartz et al., 2017]. All models also use a batch size of 32, as this is the maximum size the available VRAM can handle on the EC2 instances. The largest batch size possible was used as the batch sizes used in the FaceNet paper with Triplet loss was substantially large. The embedding layer produces a 512 dimensional vector in all models. This was the dimensions used in [Lopez et al., 2018] as well.

During training, it was noticed that all methods tended to overfit drastically on the training set. This can be seen on the training graph for Densenet121 architecture in Figure 5.1, where the Triplet loss is shown for the TEL and Triplet trained models. During training, a stopping criteria was implemented where if the validation loss did not decrease for at least five epochs. The Triplet loss is shown so that the models trained using only Triplet loss can be compared to the TEL method models. For more training graphs, please refer to Appendix C.

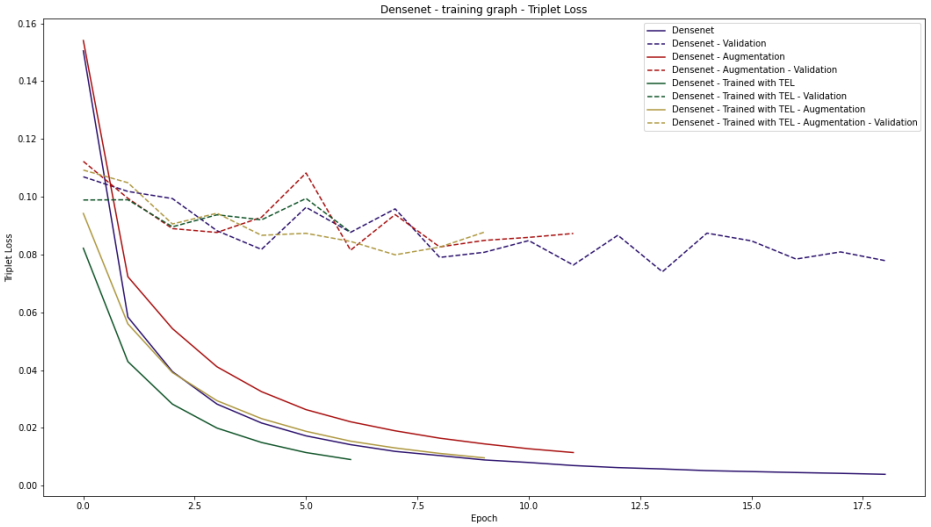


Figure 5.1: Training graph showing the Triplet loss for the DenseNet121 model trained using different methods.

Even though the models do overfit, one can still investigate the accuracy on unseen data to interpret how well the different methods compare to one another. In Table 5.1 one can find the various accuracies produced by the TEL and CEL trained models. The highlighted values show the highest accuracy for that specific model architecture. All the models' best results come from the TEL training methods, except for the Resnet50 model pre-trained on ImageNet data.

Table 5.1: Summary of the experiment results regarding the models trained using only Cross Entropy Loss (CEL) and Triplet Entropy Loss (TEL). The results in the table show the accuracy (%) of the models on unseen NCHLT speaker data.

Model	***CEL Baseline***	***CEL***	***TEL Baseline***	***TEL***
CRNN	71	70	**75**	74
Densenet121 (Imagenet)	78	78	79	**81**
Densenet121	75	74	75	**77**
Resnet50 (Imagenet)	76	**78**	77	**78**
Resnet50	74	72	76	**78**

Table 5.2: Summary of the experiment results with regards to the models trained using only Triplet loss and Triplet Entropy Loss (TEL). The results in the table show the loss value for the models on unseen NCHLT speaker data.

Model	***Triplet Baseline***	***Triplet***	***TEL Baseline***	***TEL***
CRNN	0.220	0.150	**0.095**	0.096
Densenet121 (Imagenet)	**0.068**	0.077	0.076	0.070
Densenet121	**0.074**	0.075	0.088	0.080
Resnet50 (Imagenet)	0.078	**0.075**	0.078	0.076
Resnet50	0.083	0.082	0.083	**0.076**

When looking at Table 5.2, which shows the Triplet loss values, one can see that on average the languages are better separated when the model is trained using only Triplet loss. If one takes the embeddings generated by these models and predict the language in the same way as the FaceNet researchers, namely assigning the embedding the same language as its closest neighbour, then the top accuracy achieved is 78%. This is still lower than the accuracies produced using the TEL method.

Further, the bias of the models are inspected to see how well the models perform on speech data based on the gender of the speaker. These results can be seen in Table 5.3. All of the models perform roughly the same over both genders, with the Resnet50 models seeming to perform a bit better on males compared to females.

Table 5.3: Accuracy (%) performance of the various models based on the gender of the speaker.

Model	***Female***	***Male***
CRNN	76	76
Densenet121 (Imagenet)	80	81
Densenet121	75	76
Resnet50 (Imagenet)	76	80
Resnet50	76	79

The following section takes a deeper look into embeddings generated by the models to determine where these models overfit and what they learn. The only models discussed further in this thesis will be the Densenet121 (pre-trained on Imagenet) and Resnet50 (not pre-trained) models. This is because they are the two best performing models. To see more information regarding the other models, and more info about these two models, please refer to Appendix C.

5.2 Inspecting the Generated Embeddings

To further understand what the models learn during the different training methods, the embeddings generated are inspected visually. This is performed by training a Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) model [McInnes et al., 2020] to project the embeddings down to two dimensions. Other techniques do exist to project the embeddings to fewer dimensions, i.e. dimensionality reduction, in order to inspect them visually, such as T-SNE [Van der Maaten and Hinton, 2008]. UMAP was chosen as the dimensionality reduction algorithm as it preserves more of the global structure [McInnes et al., 2020]. This implies that one can interpret the intra-cluster as well as the inter-cluster relationships in the visualisation. Thus cluster that are close together are more similar than other clusters.

UMAP works by constructing a graph representation of the data points in the high dimensional space and then projecting these points down to the lower dimensions while still trying to maintain the same topological structure. In order to generate the high dimensional graph structure, which is ensured to not have any broken connections, UMAP assigns each data point a variable radius. They base this radius on the distance to the kth nearest neighbour, where k is a hyperparameter in the algorithm. A higher k preserves more of the global structure and a lower k preserves the local structure more. There is no recipe or guidance for choosing k and this can only be found with trail and error. They connect the data points whose radius then overlap with a weighted connection, where this weight is calculated based on the distance between the points. These weights are interpreted to be the probability of the connection existing, thus the weights are scaled to also be in the correct range to be a valid probability value. This constructed graph is then a fuzzy topological structure for the high-dimensional data.

The low dimensional representations are then constructed by minimising the loss between the weights of a connection in the high dimensional space and the weights of those same connections in the low dimensional space, where these low dimensional weights are obtained using the same method as above. Because weights between the connections represent the probability that the connection either exists or does not exist, the loss function will compare vectors of probabilities with one another and thus cross-entropy can be used to minimise the distance. The loss function can be found in Equation (5.1) [McInnes et al., 2020], with C being the set of all connections, $w_h(c)$ being the weight of connection c in the high dimensional space and $w_l(c)$ being the weight in the low dimensional space. In the loss function, the first term provides the attraction force and the second term the repulsive force, much in the same way as how the TEL loss will pull and push representations.

$$L = \sum_{c \in C} w_h(c) \log \frac{w_h(c)}{w_l(c)} + (1 - w_h(c)) \log \frac{1 - w_h(c)}{1 - w_l(c)} \tag{5.1}$$

The optimal k was found to be 15, with a minimum separation distance set at 0.001, using the Cosine distance. The other hyperparameters are the default values set by the software package [McInnes et al., 2018]. The values were chosen in an iterative way until the clusters formed were interpretable visually. The UMAP model is trained using the NHCLT train set embeddings.

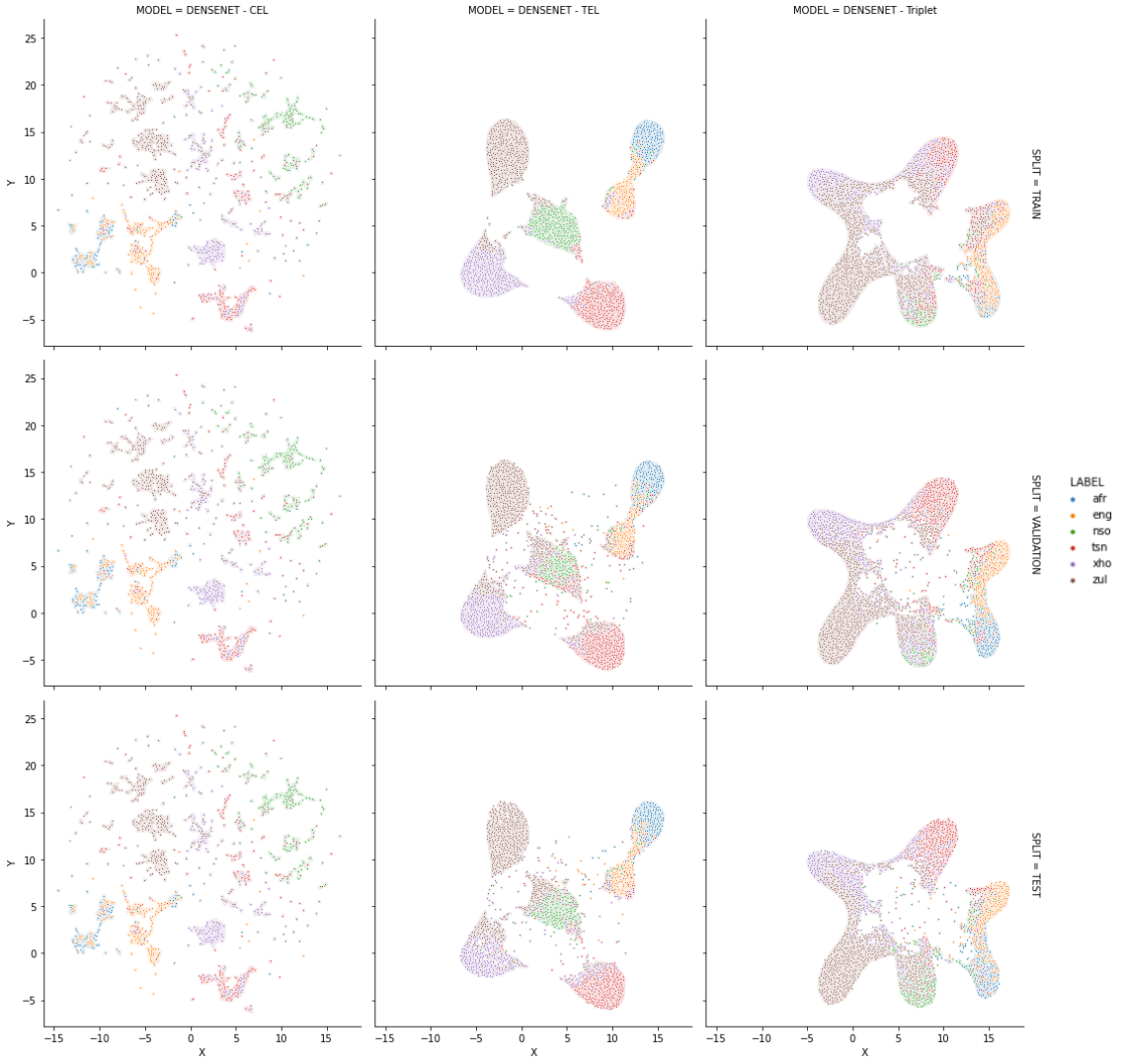


Figure 5.2: Projection of the embeddings generated by the Densenet121 model on the different NCHLT datasets.

In Figure 5.2 one can see the embeddings generated by Densenet121 model, pre-trained on Imagenet, for all of the training methods. One can see that embeddings created using the TEL method are better at separating the languages. This is clear if one looks at the training examples specifically, as the TEL embeddings create a cluster for all languages, but on the Triplet loss side *English* and *Zulu* is being confused with other languages. Looking at the embeddings of the validation and test set embeddings, one can see that both models start confusing the languages, but that overall there are still clear language clusters in the TEL embeddings. The CEL trained model though has no clear clusters forming.

The more interesting result come when one inspects the language families. Looking at the TEL embeddings, one can see that the Germanic languages (*Afrikaans* and *English*) form their own overlapping cluster to the right with the African languages forming their own clusters to the left. Looking at the validation and test embeddings, it can be seen that the confusion occurs more often between the families than with other languages. This confusion appears less in the TEL model than the Triplet loss model, and this leads to the assumption that the TEL learning method is better at understanding the nuances of the languages.

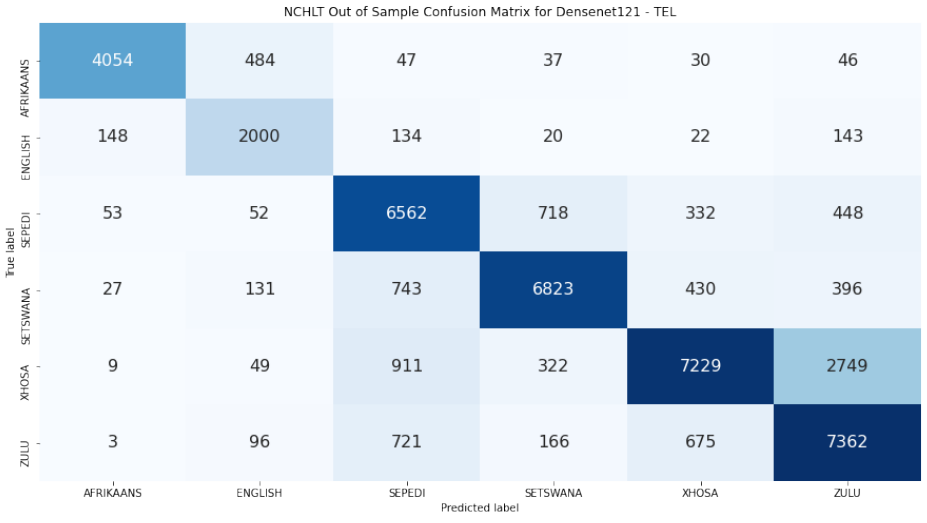


Figure 5.3: Confusion matrix generated by combining the validation and test set predictions for the pre-trained Densenet-121 model fine tuned using the TEL method.

In the confusion matrix for the TEL Densenet121 model found in Figure 5.3, the

confusion can clearly be seen. The Germanic languages are not really confused with other languages but *Xhosa* and *Zulu* have the most confusion between them. Besides *Zulu*, *Sepedi* also gets confused with the other African languages. The confusion could be related to the words present in the prompts given to speakers that share the same words between languages. To visually see the words present in all the NCHLT sets, please refer to Appendix B.

In Figure 5.5, one can see the embeddings generated by the Resnet50 model, which was not pre-trained on Imagenet. In both the TEL and Triplet loss training instances, the embeddings seem to overlap more than in the Densenet121 cases. The TEL embeddings though are stronger grouped together per language than the Triplet loss method as there are more connections between the clusters in the Triplet loss case. The embeddings generated by the CEL method though show no potential for separation. One can also see the same trend in terms of confusion between languages present in the Densenet121 models when looking at the validation and test datasets. The confusion matrix can be found in Figure 5.4. We can see that the Germanic languages are more confused in these results than in Figure 5.3, but the same trends apply to the African languages.

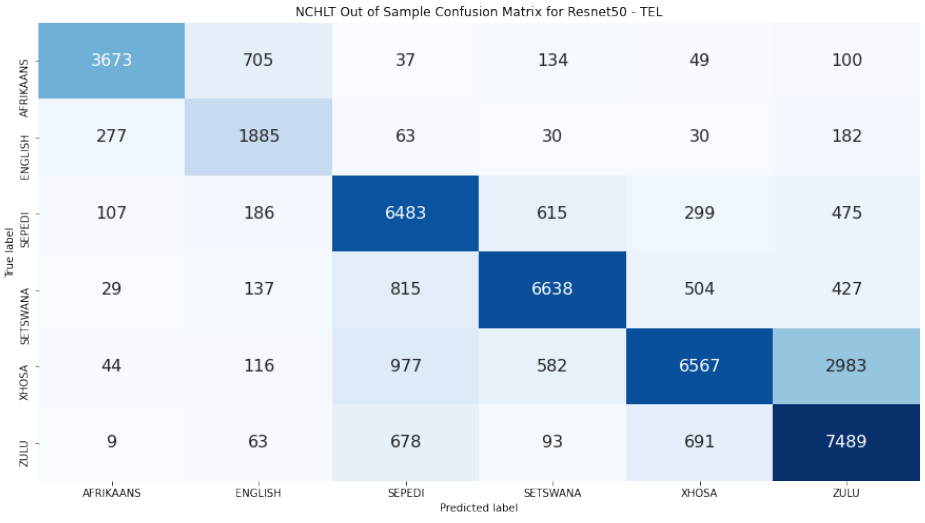


Figure 5.4: Confusion matrix generated by combining the validation and test set predictions for the Resnet50 model trained using the TEL method.

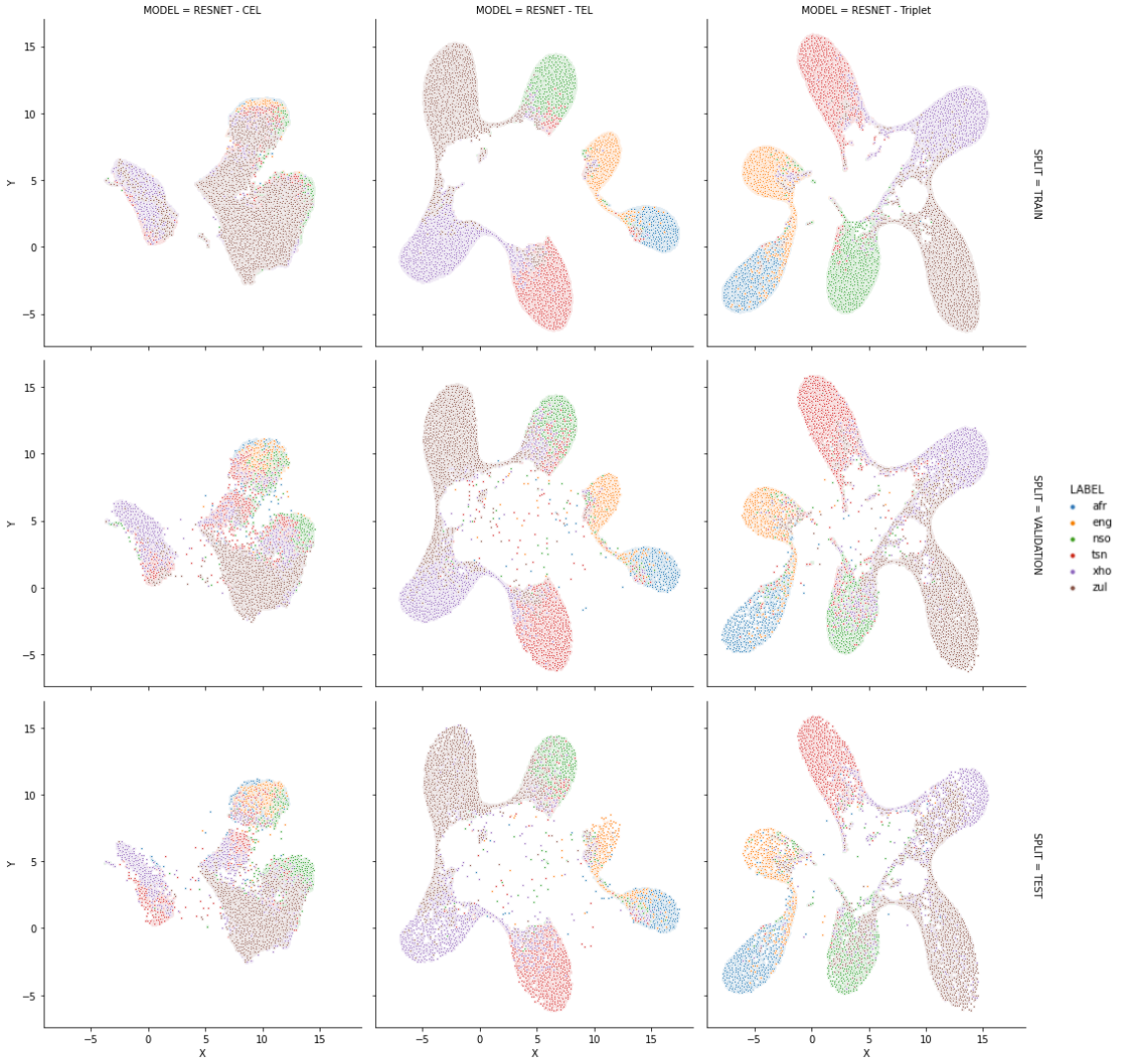


Figure 5.5: Projection of the embeddings generated by the Resnet50 model on the different NCHLT datasets.

5.3 Discussion on Results

Looking at the results in the previous section, one can see that the TEL method does overall perform better on unseen data than the more traditional Triplet loss and CEL methods. This is more apparent when looking at the embeddings in Section 5.2, where the clusters formed by the TEL trained models are more clustered together than the Triplet loss trained models. The TEL trained models also seem to understand the nuances of the languages better.

The results also show that the techniques introduced in [Park et al., 2019] and [Palanisamy et al., 2020] can apply to South African LID systems to improve the overall accuracy. Spectral augmentation though has to be investigated for each use case and each architecture. The reason for this is that the increase in accuracy could perhaps not be worth the extra complexity added to the system. If one looks at the projections for all models in Appendix C.1.2 though, one will see that the models trained with augmentation do tend to create embeddings that separate the languages better than the respective baseline models.

All the models should still be optimised more to reduce the amount of overfitting present. Various optimisations regarding the batch size, image size and optimiser/learning rate could be investigated with larger computational resources.

Even though the results do show that the TEL method has promise, there is still more work to further demonstrate improved generalisation of LID systems. The following section will investigate how the Densenet121 and Resnet50 models perform on out-of-domain data, specifically the Lwazi dataset.

Chapter 6

Performance On the Lwazi Dataset

In this chapter the Densenet121 and Resnet50 models that were trained on the NCHLT speech corpus, discussed in Section 5.1, are tested on out-of-domain samples to test how well the TEL training method generalises to a new domain. As was mentioned in Section 3.2, the recordings for the Lwazi dataset were sampled at $8kHz$ versus the NCHLT dataset which was sampled at $16kHz$. This will undoubtedly influence the results, as will the difference in words spoken between the two datasets.

Table 6.1: Accuracy (%) performance of the Resnet50 and Densenet121 models on the Lwazi dataset.

Model	***TEL Accuracy***	***CEL Accuracy***
Densenet121 (Imagenet)	45	25.8
Resnet50	28	23

In Table 6.1 one can find the performance of models on the new domain. As can clearly be seen, none of the models perform nearly as well as they did in the NCHLT domain. The only model that showed any promise is the Densenet121 model trained using the TEL method as it achieved close to 45% accuracy. In both model cases, one can see that the TEL method does outperform the CEL method, but in the Resnet case it is by an ignorable margin. What the results do show though is that by combining the TEL method with the techniques in [Park et al., 2019] and [Palanisamy et al., 2020] produces a model that is more robust to new domains. This can be further seen in the confusion matrix for the Densenet121 models, found in Figure 6.1. For further results produced by the Resnet50 models, please refer to Appendix C.2.

Figure 6.1: Confusion matrices for the Densenet-121 models trained using the TEL method (top) and CEL (bottom).

Looking at Figure 6.1 one will see the CEL trained Densenet model tends to think that most languages are either *Sepedi* or *Setswana*, with very few samples being predicted to belong either to *Xhosa* or *Zulu*. The TEL trained model shows a

better understanding for the Germanic languages as both *Afrikaans* and *English* have better results. *Xhosa* and *Zulu* are also performing much better, showing a better separation in the African languages.

To investigate further, the embeddings generated are again projected onto a two dimensional plane to inspect it visually, using UMAP. The projected embeddings can be found in Figure 6.2. The models have not generalised well, but the TEL embeddings do have a small subsets which belong to mostly one language or family. This is compared to the Triplet embeddings which have only one cluster dominated by a family.

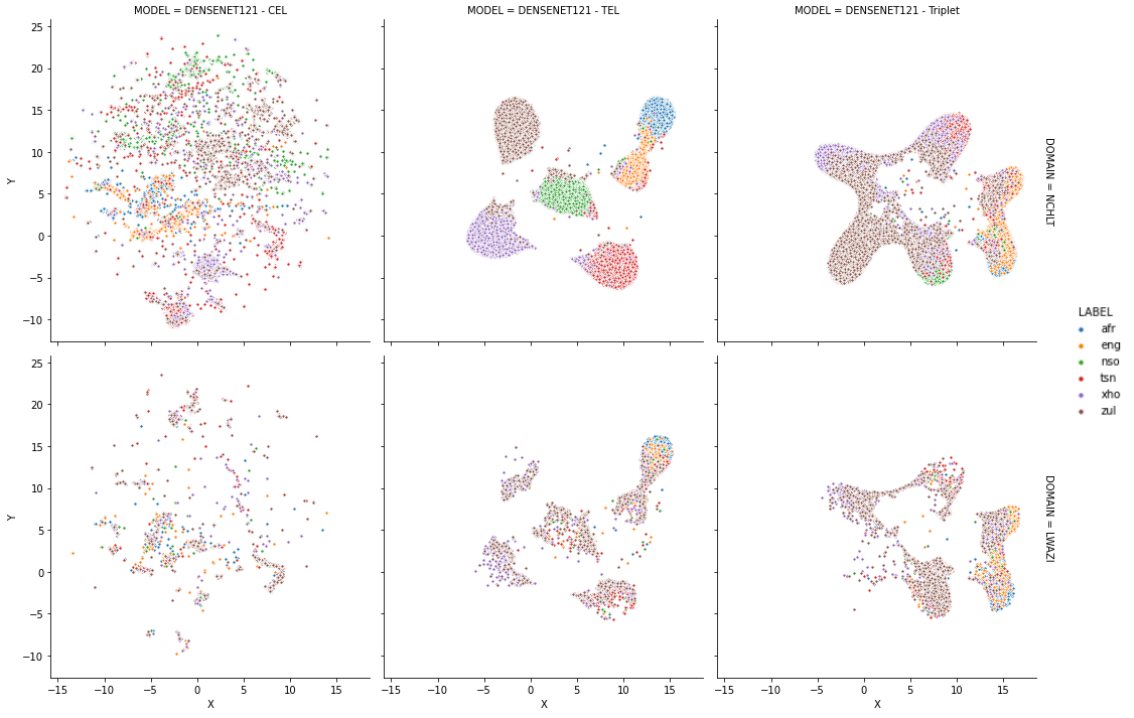


Figure 6.2: Projection of the embeddings generated by the Densenet model on the training NCHLT data (top) as well as the predicted Lwazi embeddings (bottom).

What these results show is that the Densenet121 model (and the other models) might not learn the characteristics of a language, but simply memorise the patterns in the spectrograms representing words. A reason then for the Lwazi results to perform so poorly is due to the fact the words present in the Lwazi corpus are not present in the training data obtained from the NCHLT dataset. This can be seen by looking at the wordclouds formed for the *Zulu* and *Afrikaans* datasets in Figure 6.3 and Figure 6.4. In these figures the size of the word is directly proportional to the

frequency of use in the dataset. One can clearly see the difference in words uttered. For different languages' word clouds, please refer to Appendix B.

Figure 6.3: Word clouds for both the NCHLT Zulu speech (left) and the Lwazi Zulu speech (right).

Figure 6.4: Word clouds for both the NCHLT Afrikaans speech (left) and the Lwazi Afrikaans speech (right).

All the results above show that there is still more work required to train models to perform better on out-of-domain samples and to force models to learn characteristics rather than words. The results show though that the TEL method combined with spectral augmentation and pre-trained models can be a good stepping stone to work from.

Chapter 7

Conclusion

The research presented investigated several methods to improve the generalisation of LID systems to new speakers and to new domains. These methods involve Spectral augmentation and using CNN architectures that are pre-trained on the Imagenet dataset. The primary method investigated was the TEL training method which involves training a network simultaneously using Cross Entropy and Triplet loss. Several tests were run with three different CNN architectures to investigate what the effect all three methods have on generalisation of an LID system. The tests involved training models on six languages from the NCHLT speech corpus and measuring the performance of the models on new speakers from the same domain, and new speakers from a different domain.

It was found that all three of the methods do contribute to improve the accuracy of the models to the new speakers and to new domains. The best two models were found to be a pre-trained Densenet121 model and a Resnet50 model trained from scratch, with both models being trained using the TEL method and Spectral augmentation. Even though both models could achieve 81% and 78% accuracies on new speakers in the NCHLT domain, both models still overfited on the training data, as they achieved close to 100% accuracy on the training speakers. When the models are tested on the Lwazi domain, the Resnet50 model performed poorly and only achieved an accuracy of 28%, whereas the Densenet121 model could achieve an accuracy 45%.

If one looks at the embeddings generated by both of the models for all training methods, the results do indicate that the TEL method has a better understanding of the languages. This is because of the stronger clusters formed in the TEL embeddings and the fact that the confusion happens more often between language families, whereas in the CEL/Triplet case the confusion occurs more often between

all languages.

The results do also show that all the models might still only be memorising words rather than learning language characteristics. This will require more research, as is also recommended in Section 7.1. The TEL training method also significantly reduces the training time required for a model to converge, specifically on the use cases shown in Appendix D, which were done to show the robustness of TEL to different tasks.

7.1 Recommendations for Future Research

The work presented only investigates six of the eleven South African languages. Languages from the Tswa-Ronga and Venda families were also not used in the research. Further work should be done to investigate how well these methods will work in systems where all eleven languages are present.

The methods should also be applied to data from previous papers, specifically the data in [Bartz et al., 2017] and [Revay and Teschke, 2019] as these papers investigate more common languages. The methods mentioned in Section 2.2 should also be thoroughly inspected for out-of-domain performance.

The model architectures and hyperparamaters presented in this thesis are also not optimised yet. Future work, if there are better computation resources and a higher budget available, should further investigate all the parameters and hyperparameters used in this work. Specifically, the following hyperparameters should be investigated, with the use of cross validation, as it is estimated that they will have the biggest impact on results:

- optimiser and learning rate (currently Adam with a learning rate of 0.0001).
- Batch size (currently 32)
- Embedding dimension (currently 512)
- Triplet Loss margin (currently 0.2)

Only the application of the TEL method is presented in this work, showing that the effort required to dive deep into the mathematical foundations of the training method should be undertaken. Future work should look into this.

The code for a very rudimentary chatbot called *Spectobot* can be found on the GitHub page for this project, found in Appendix A. *Spectobot* can, at the moment, prompt a user to send a voice note and then use any LID model provided and predict the language spoken in the voice note where the user can then give feedback. The

results, with the voice note, are then saved. Future work should look at finishing *Spectobot* to further test the out-of-domain performance of TEL trained models and use it to create a South African speech set where the speech does not comprise speakers reading a prompt but rather where they give a more natural response.

Lastly, with the TEL method showing promise for audio classification tasks, it should also be investigated in other areas as well. Areas that will be a good fit for the TEL method are self-supervised tasks. Here Triplet loss could be substituted with Contrastive loss, forming Contrastive Entropy Loss. This is because there are various self-supervised techniques out there already using Contrastive loss instead of Triplet loss and showing good results. The biggest challenges in these tasks will then be to generate labels to be used by CEL.

Appendix A

Project Code

Please refer to `https://github.com/ruanvdmerwe/triplet-entropy-loss` to find all of the code associated with the project.

Appendix B

NCHLT and Lwazi Spoken Words

This section will look at the word clouds produced by transcripts associated with both the NCHLT and Lwazi dataset. A word cloud is a visual representation of the frequency a word appears in a corpus. The bigger the word in the image, the more it occurs within the corpus. The first section will compare the NCHLT training data corpus with the validation and test corpus (combined). The second section will then compare the NCHLT training data corpus with the Lwazi corpus.

B.1 NCHLT vs NCHLT

Figure B.1: Comparison between the spoken words of the NCHLT training and validation Afrikaans subsets.

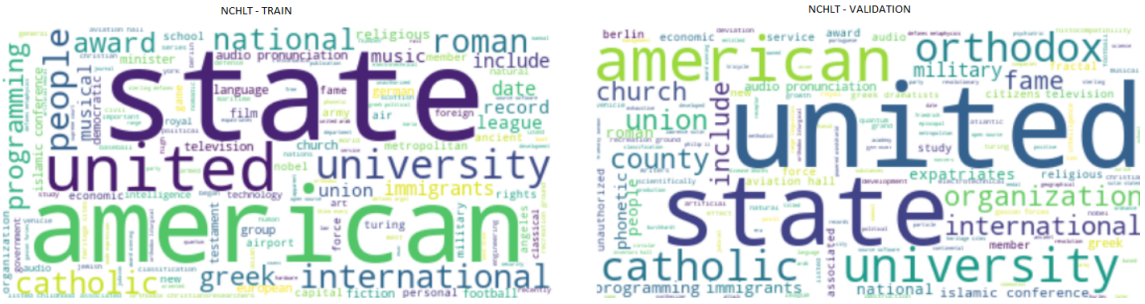


Figure B.2: Comparison between the spoken words of the NCHLT training and validation English subsets.

Figure B.3: Comparison between the spoken words of the NCHLT training and validation Sepedi subsets.

Figure B.4: Comparison between the spoken words of the NCHLT training and validation Setswana subsets.

Figure B.5: Comparison between the spoken words of the NCHLT training and validation Xhosa subsets.

Figure B.6: Comparison between the spoken words of the NCHLT training and validation Zulu subsets.

B.2 NCHLT vs Lwazi

Figure B.7: Comparison between the spoken words of the NCHLT and Lwazi Afrikaans subsets.

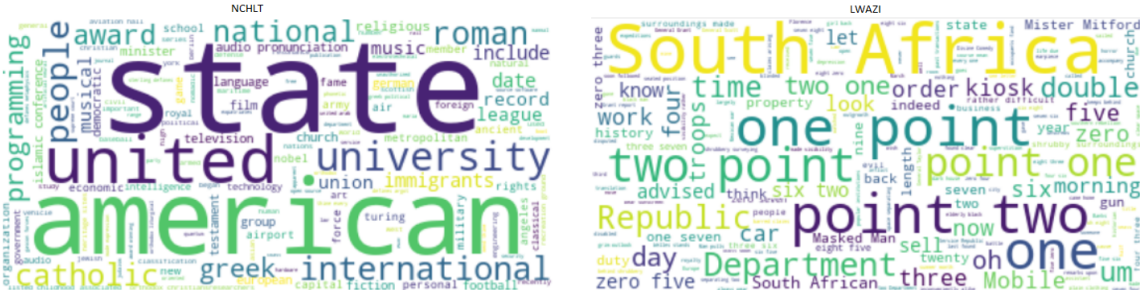


Figure B.8: Comparison between the spoken words of the NCHLT and Lwazi English subsets.

Figure B.9: Comparison between the spoken words of the NCHLT and Lwazi Sepedi subsets.

Figure B.10: Comparison between the spoken words of the NCHLT and Lwazi Setswana subsets.

Figure B.11: Comparison between the spoken words of the NCHLT and Lwazi Xhosa subsets.

Figure B.12: Comparison between the spoken words of the NCHLT and Lwazi Zulu subsets.

Appendix C

More Model Results

This appendix contains additional material relating to the training process of the NCHLT models and the embeddings generated by the embedding layer for all models. The appendix will start out with the training graphs in terms of Triplet loss value, which will then be followed by the graphs in terms of the accuracy. The next section will then showcase the various embeddings generated by the model architectures, where the embeddings generated by the TEL method can be compared to the Triplet loss method.

The final section of the appendix showcases the Resnet performance on the Lwazi domain as the body in thesis focused more closely on the pre-trained Densenet model.

C.1 NCHLT Results

As mentioned above this section will show additional results linked for the training process on the NCHLT dataset. The graphs for the training process are shown in terms of the Triplet loss and the accuracy separately. The accuracy graphs compare the TEL trained models with the CEL trained models and the Triplet loss graphs then compare the TEL trained models with the Triplet loss trained models.

It should be noted that all models had a early stopping criteria where if the validation loss did not decrease for at least five epochs the training will stop.

C.1.1 Training Graphs

Triplet Loss

In Figure C.1 one can find the training graphs for the CRNN model. Looking at the graph, it is obvious to see that the TEL trained methods perform much better than the Triplet trained models. All four models though are overfiting on the training data. In both training methods one can also see that models trained with Spectral augmentation tend to perform worse than the counterpart with no augmentation in the validation results.

In Figure C.2 and Figure C.3 and one can see the performances for the Densenet models. One can see that all the models perform relatively the same, but that the TEL models reach a lower value quicker than the Triplet trained models. There is a trend where augmentation performs slightly better than the models with no augmentation.

The Resnet model training graphs in Figure C.4 and Figure C.5 show the same stories as was present in the Densenet model. Namely all the methods do overfit and perform roughly the same, but that the TEL trained models do tend to train faster than the Triplet models.

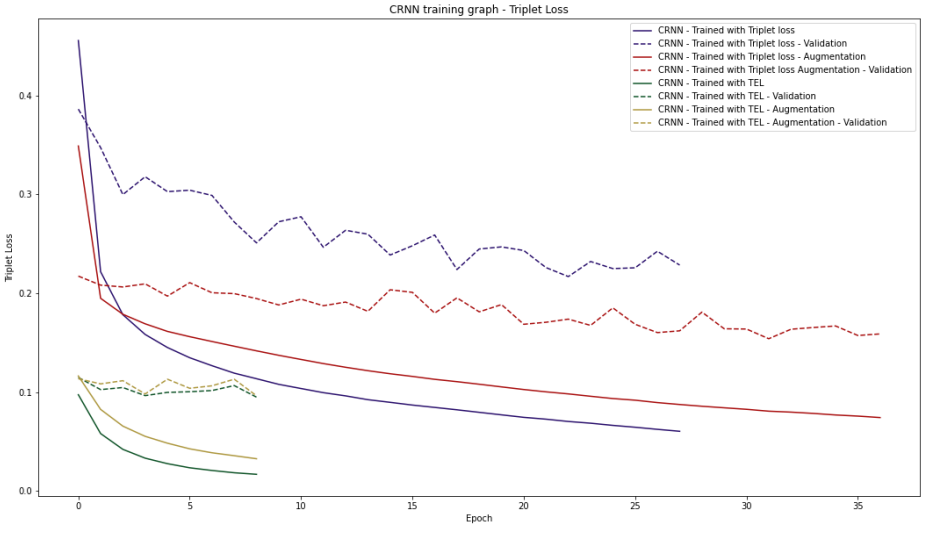


Figure C.1: Training graph showing the Triplet loss for the CRNN model trained using different methods.

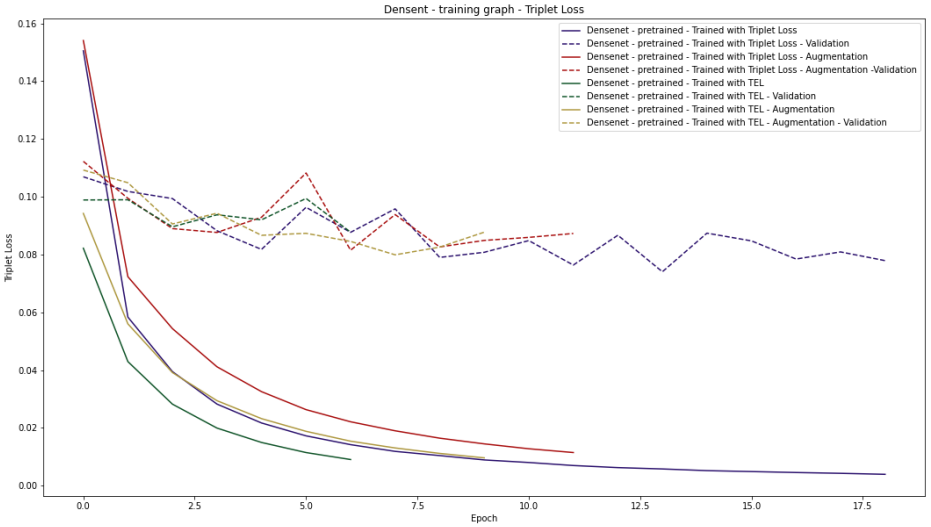


Figure C.2: Training graph showing the Triplet loss for the Densenet121 model trained using different methods.

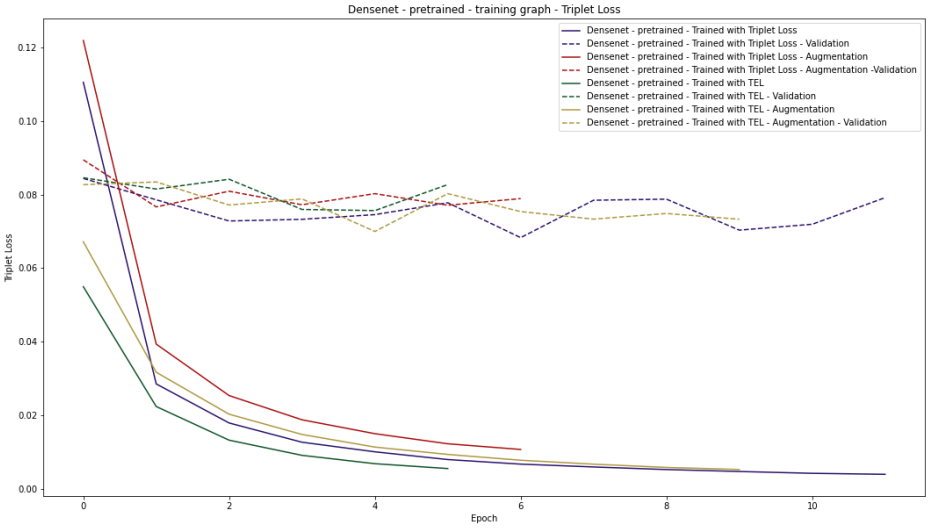


Figure C.3: Training graph showing the Triplet loss for the pre-trained Densenet121 model trained using different methods.

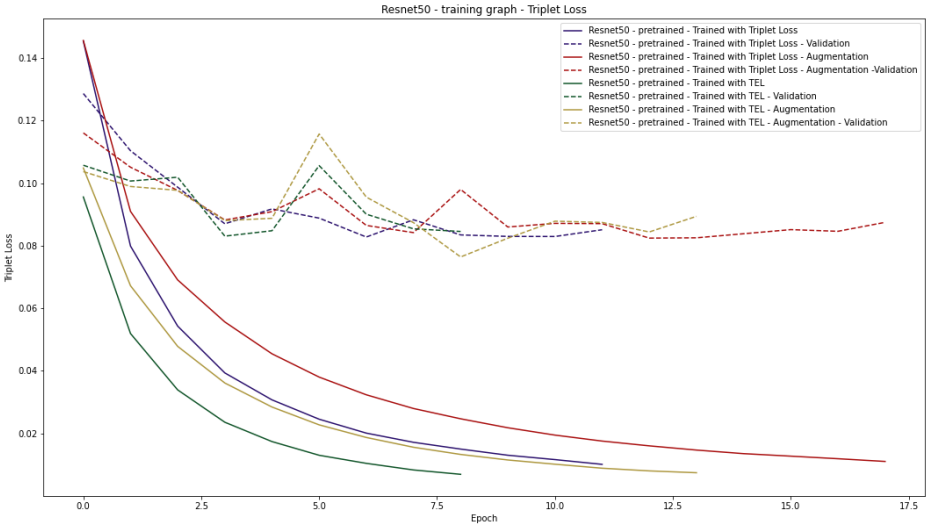


Figure C.4: Training graph showing the Triplet loss for the Resnet50 model trained using different methods.

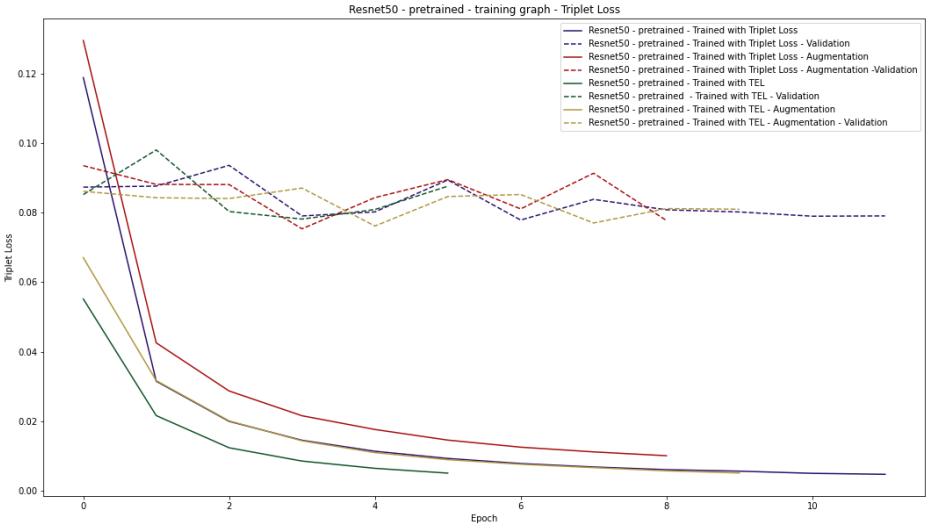


Figure C.5: Training graph showing the Triplet loss for the pre-trained Resnet50 model trained using different methods.

Accuracy

In Figure C.6 one can find the training graphs for the CRNN model. Again the baseline models perform better than the models that contain augmentation. There is also very little difference between the TEL and CEL trained models in terms of performance.

In Figure C.7 and Figure C.7 and one can see the performances for the Densenet models. One can see that all the models perform relatively the same, but that the best model is the TEL trained model with augmentation.

The Resnet model training graphs in Figure C.9 and Figure C.10 show the same stories as was present in the Densenet model. Namely all the methods do overfit and perform roughly the same, but that the TEL trained models do perform better.

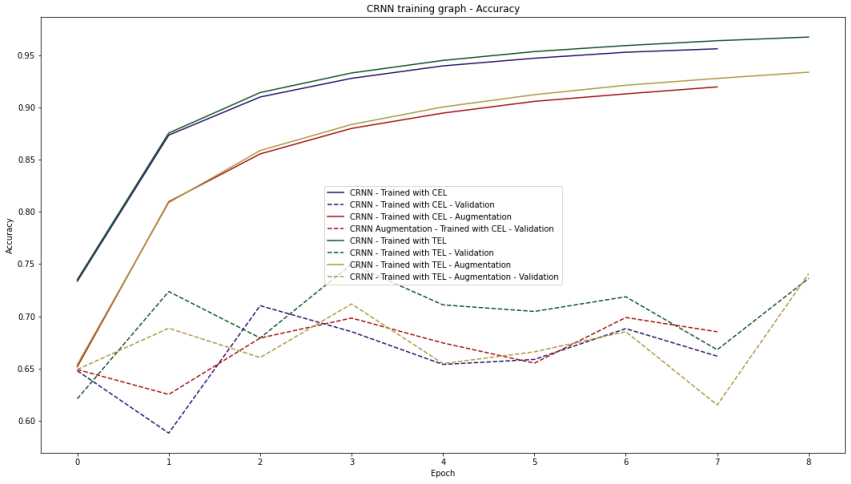


Figure C.6: Training graph showing the accuracy for the CRNN model trained using different methods.

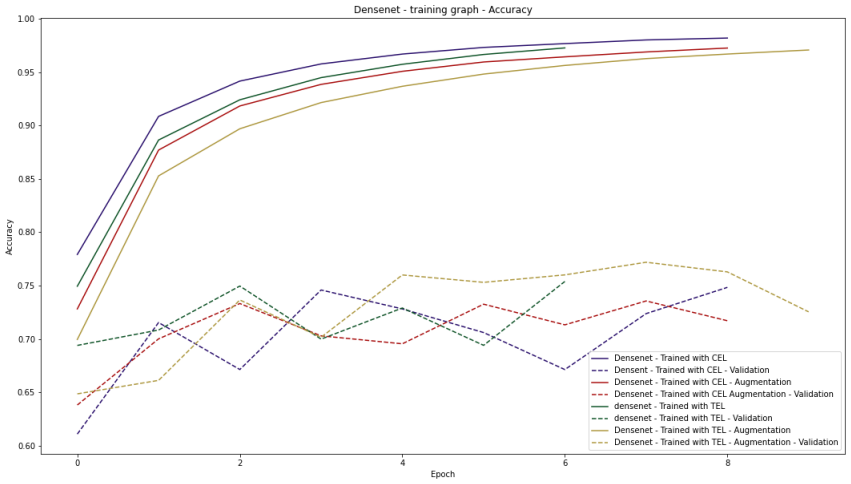


Figure C.7: Training graph showing the accuracy for the Densenet121 model trained using different methods.

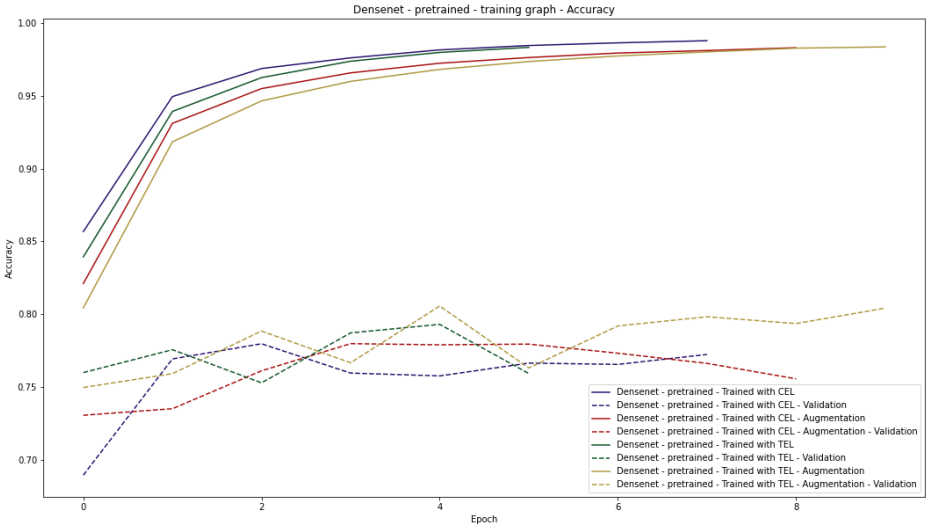


Figure C.8: Training graph showing the accuracy for the pre-trained Densenet121 model trained using different methods.

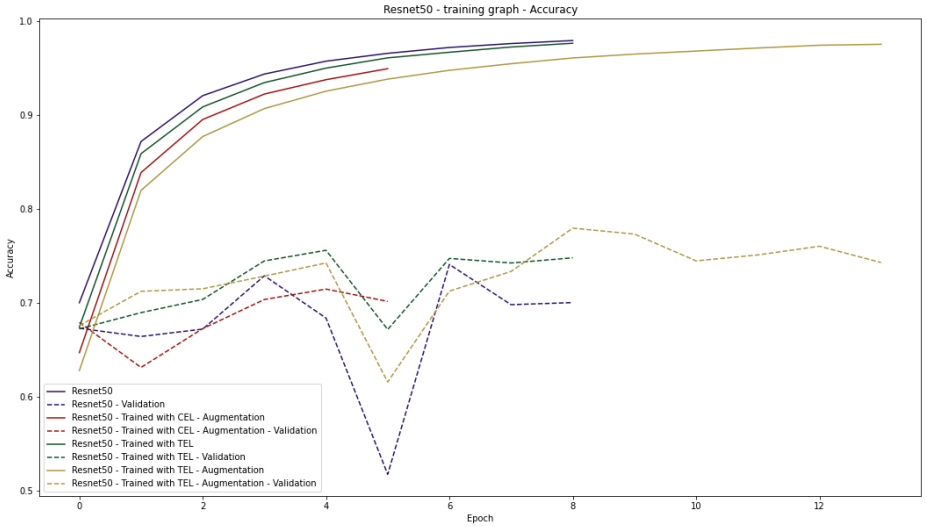


Figure C.9: Training graph showing the accuracy for the Resnet50 model trained using different methods.

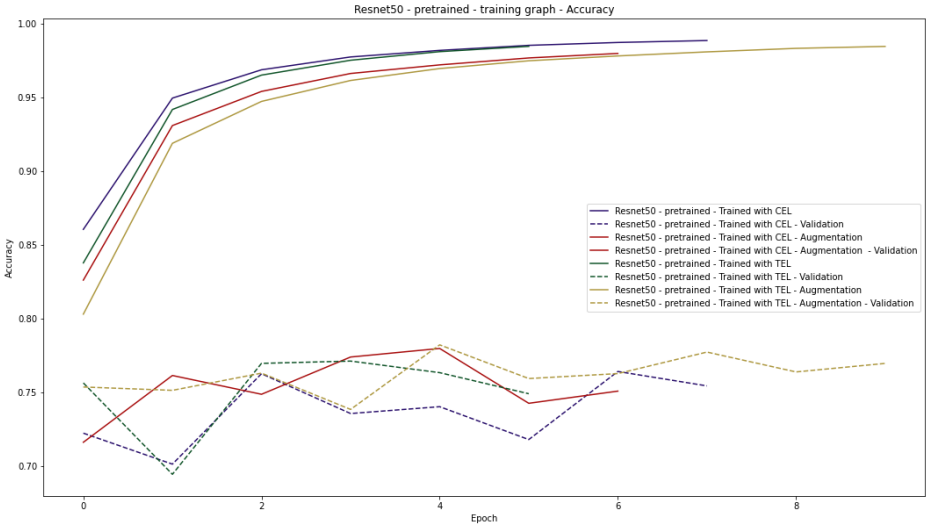


Figure C.10: Training graph showing the accuracy for the pre-trained Resnet50 model trained using different methods.

C.1.2 Embeddings Projected Using UMAP

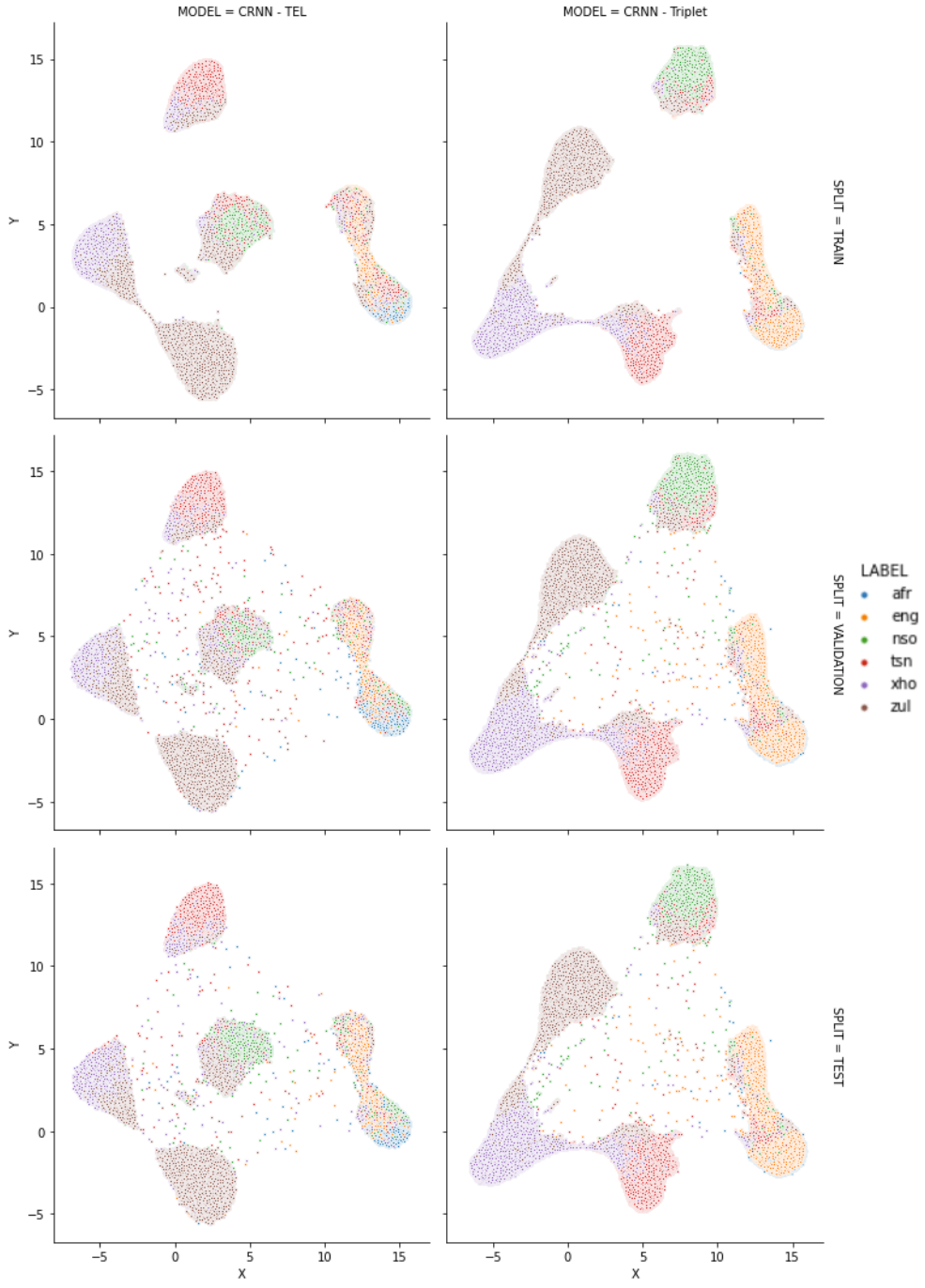


Figure C.11: Embeddings generated by the baseline CRNN model on the NCHLT dataset.

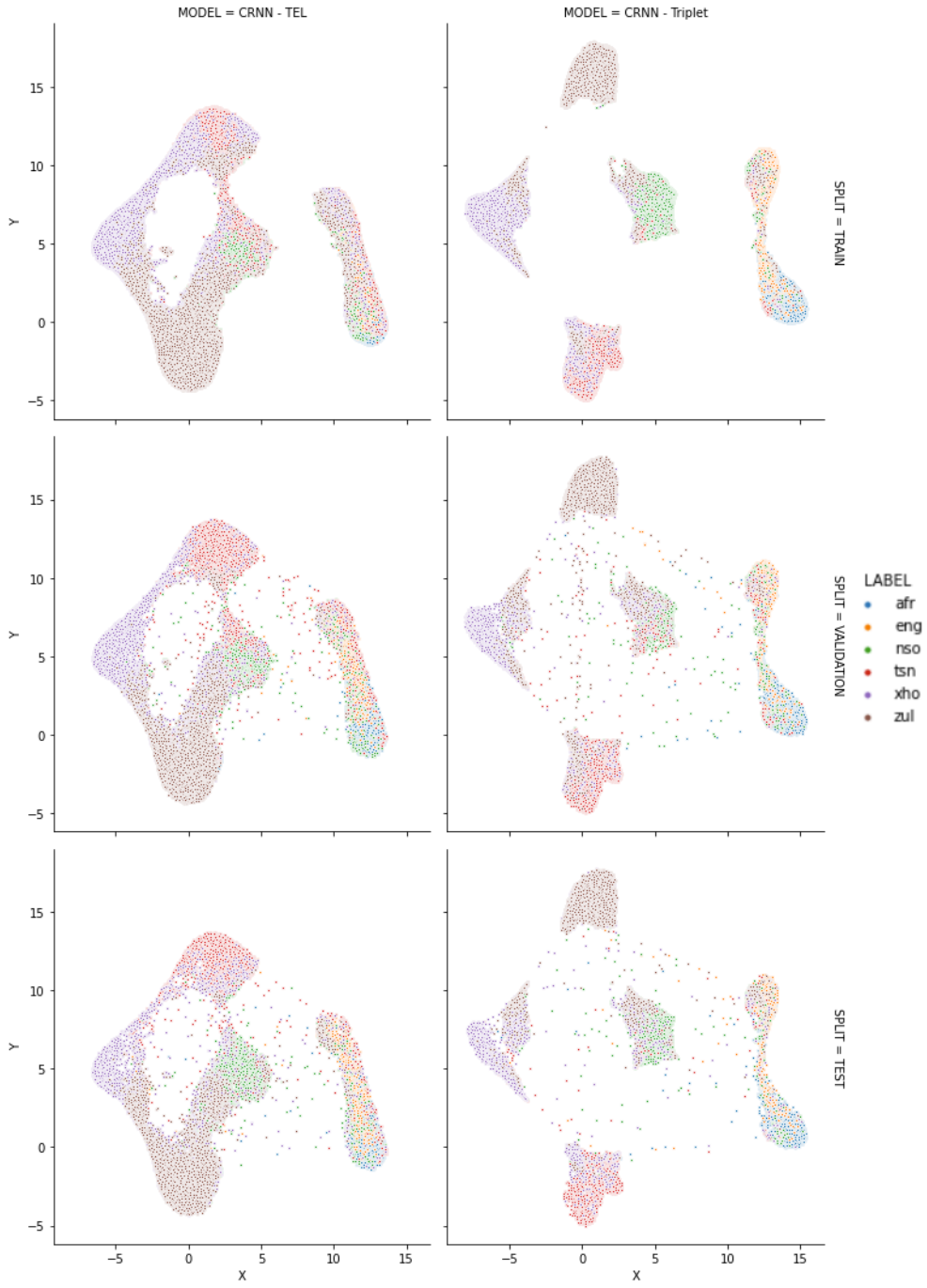


Figure C.12: Embeddings generated by the CRNN model on the NCHLT dataset.

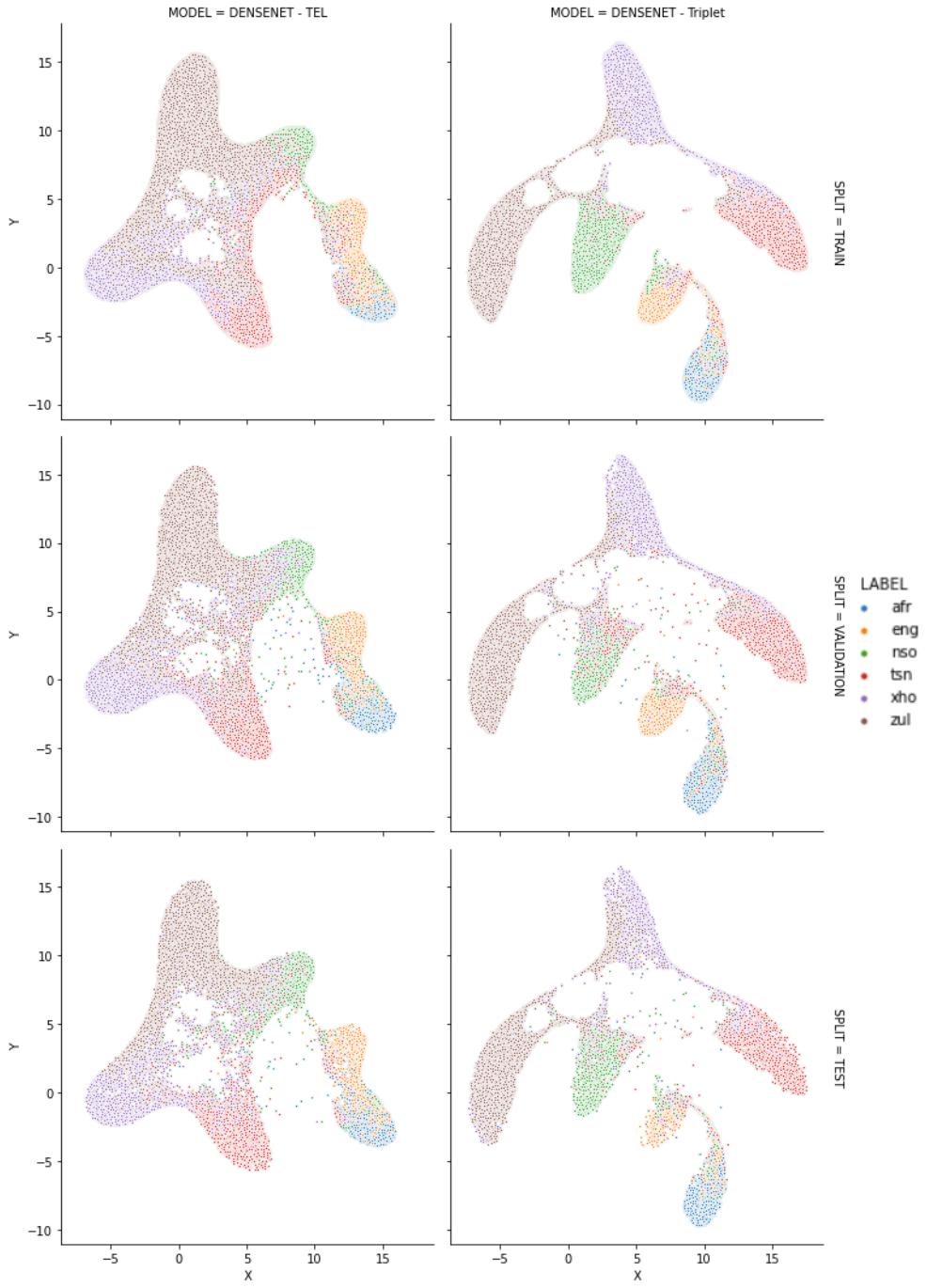


Figure C.13: Embeddings generated by the baseline Densenet121 model on the NCHLT dataset.

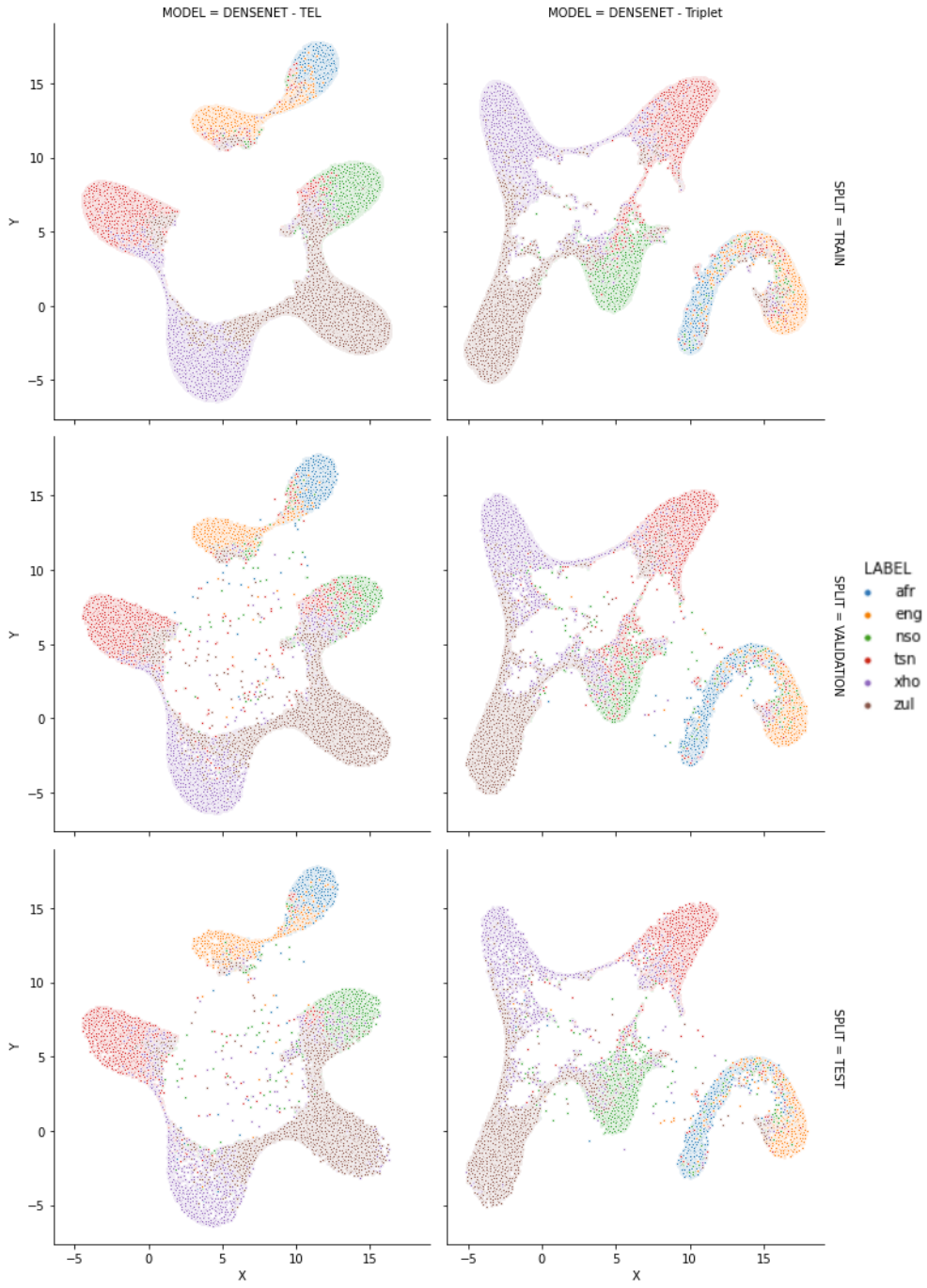


Figure C.14: Embeddings generated by the Densenet121 model on the NCHLT dataset.

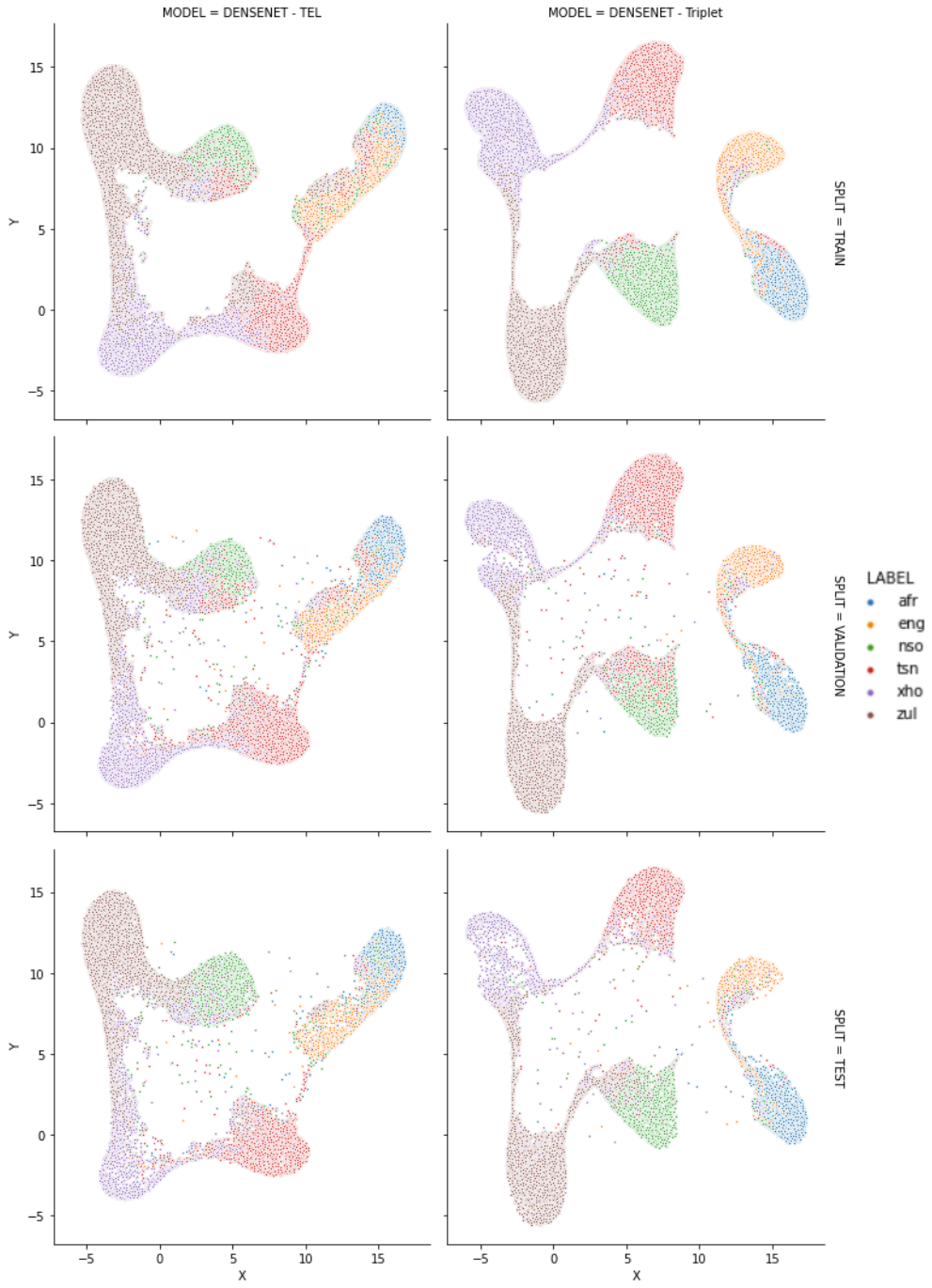


Figure C.15: Embeddings generated by the baseline pre-trained Densenet121 model on the NCHLT dataset.

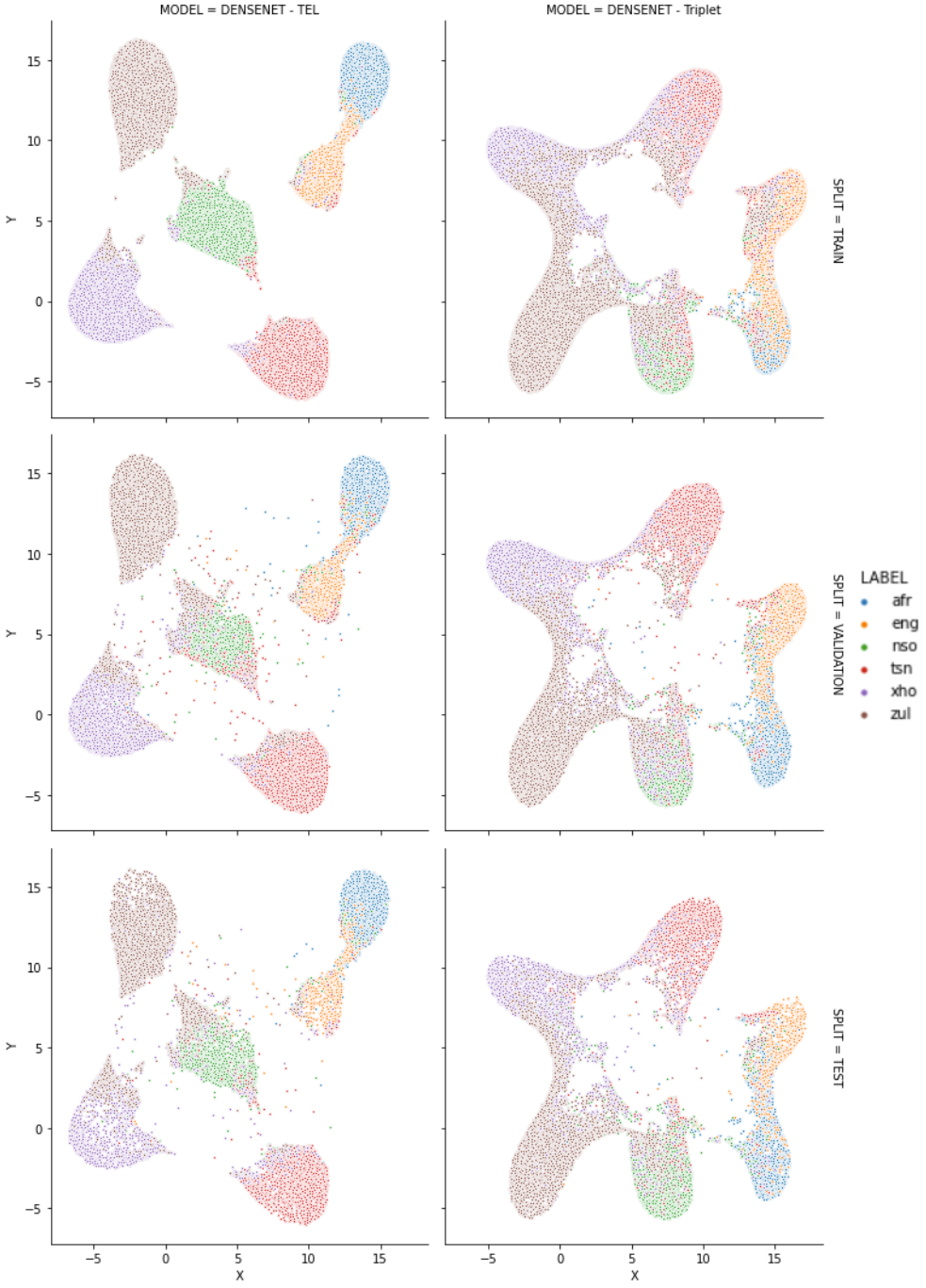


Figure C.16: Embeddings generated by the pre-trained Densenet121 model on the NCHLT dataset.

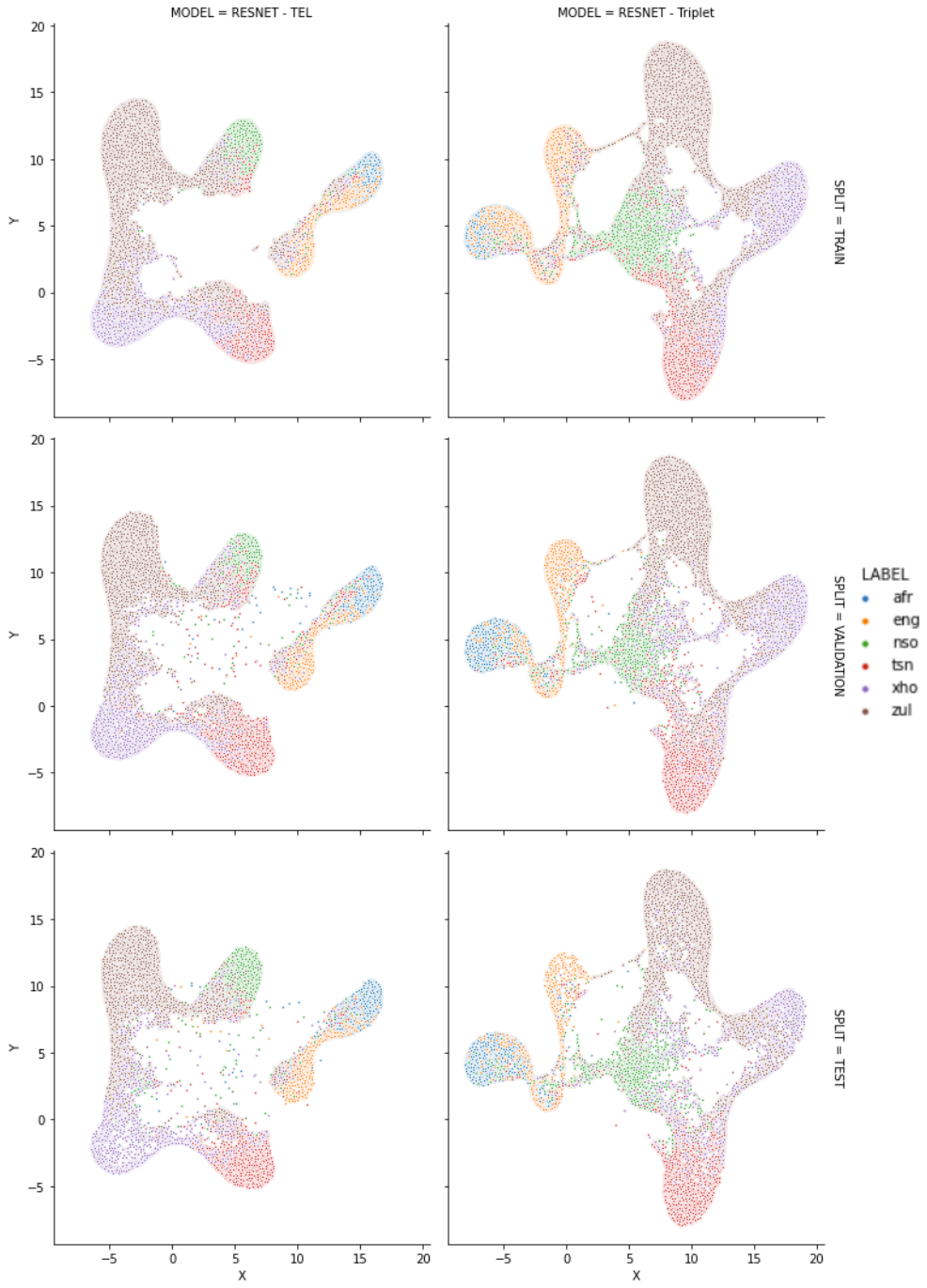


Figure C.17: Embeddings generated by the baseline Resnet50 model on the NCHLT dataset.

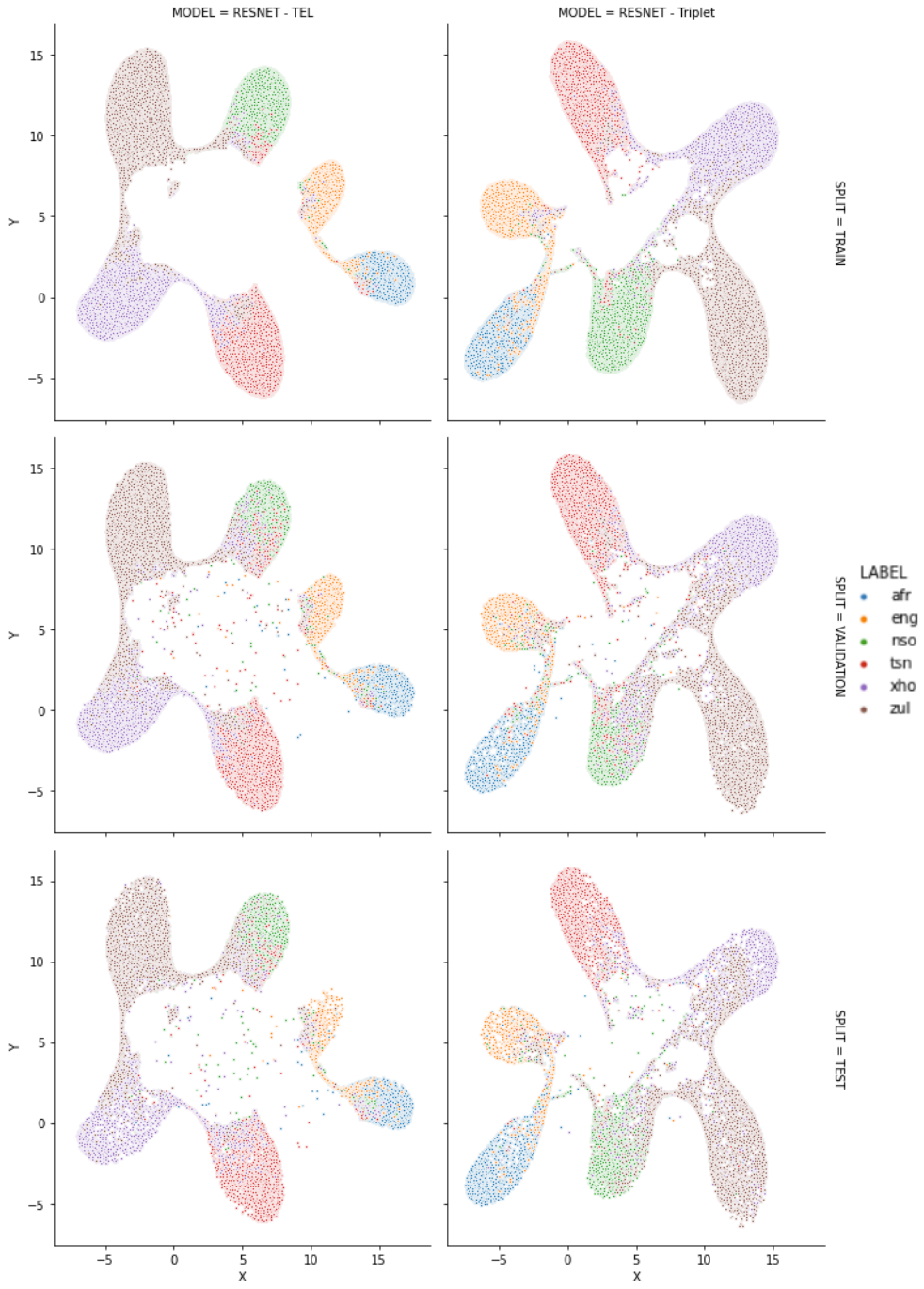


Figure C.18: Embeddings generated by the Resnet50 model on the NCHLT dataset.

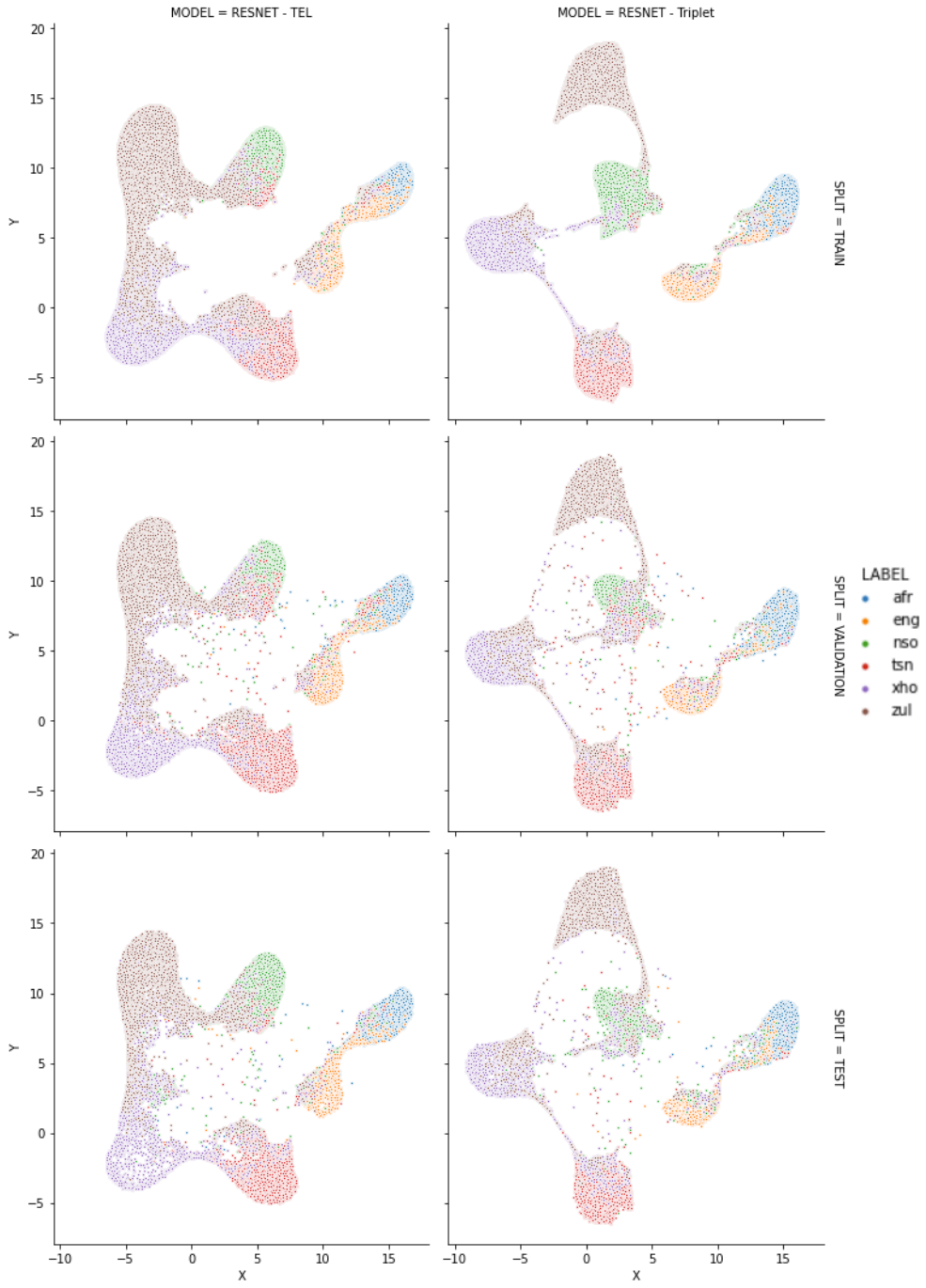


Figure C.19: Embeddings generated by the baseline pre-trained Resnet50 model on the NCHLT dataset.

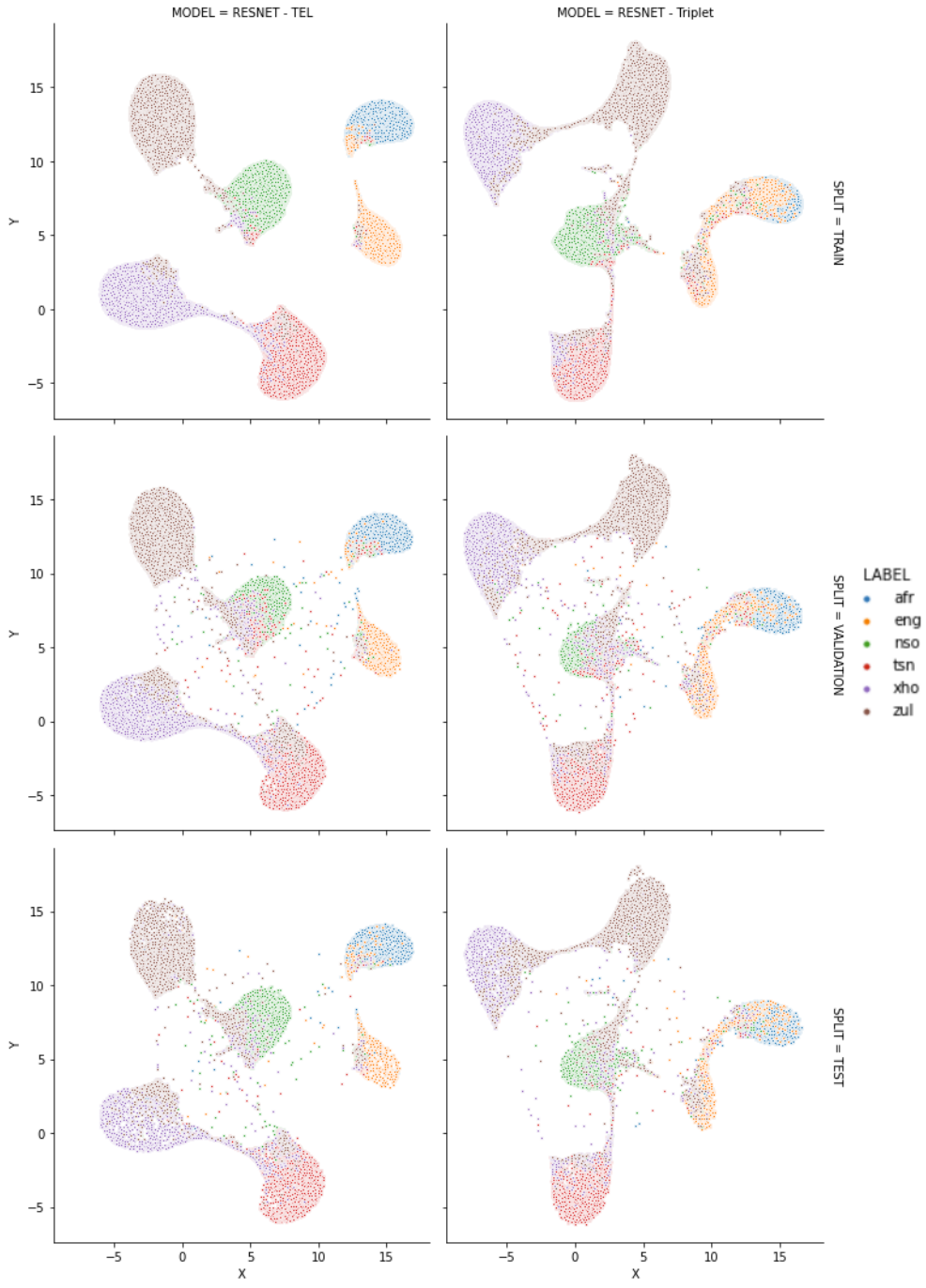


Figure C.20: Embeddings generated by the pre-trained Resnet50 model on the NCHLT dataset.

C.2 Lwazi domain

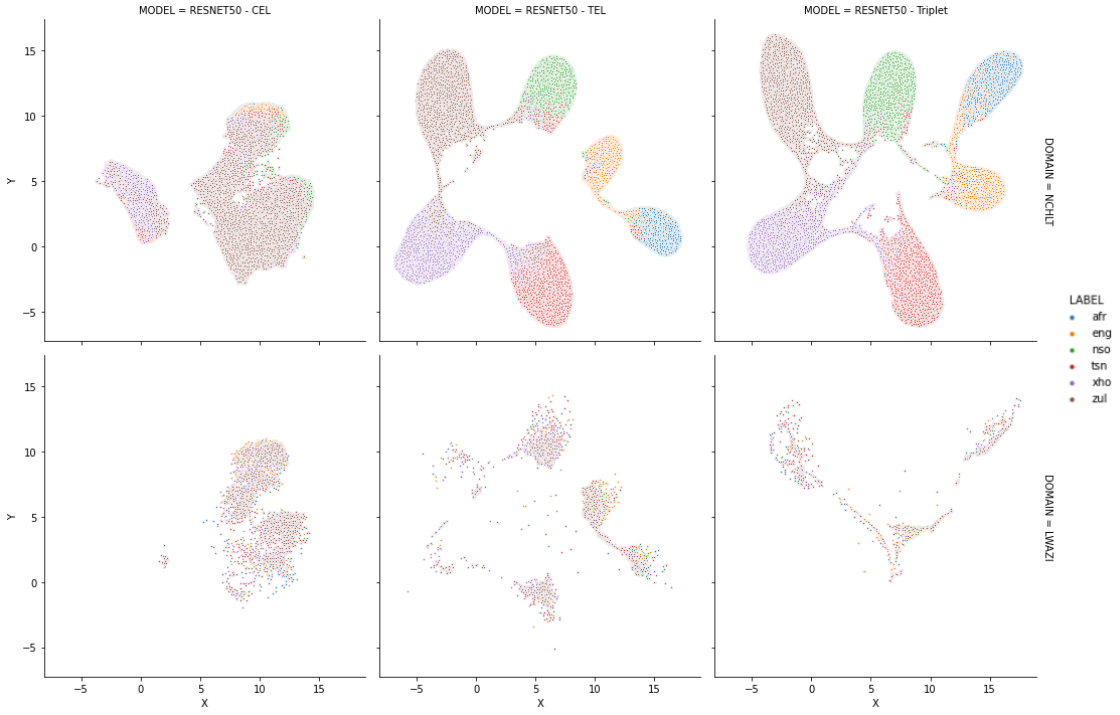


Figure C.21: Projection of the embeddings generated by the Resnet50 model on the NCHLT and Lwazi datasets.

Appendix D

Even More Use Cases

To further test the TEL training method, the Densenet-121 and Resnet50 models will be trained on two different sound classification tasks. The first task is to predict the spoken digit in a recording, using the Free Spoken Digit Dataset (FSDD). The second task will be to predict which category of music a 30 second recording belongs to using the GTZAN dataset.

For both these tasks, the idea is to show how an out of the box model trained using the different methods (TEL, CEL and Triplet loss) performs on common datasets. The purpose of this is to further cement the hypothesis that TEL trained models do generate better embeddings than previous methods, even if the task is not language identification.

D.1 Spoken Digits Classification

The data used in this experiment is open source and can be found at `https://github.com/Jakobovski/free-spoken-digit-dataset/`. The dataset consists of six speakers, where each speaker utters a digit between zero and nine, fifty times each in *English* only. This results in 3000 recordings. This can be imagined as a spoken MNIST dataset.

The data is split into a train and validation set, with the validation set consisting of fifteen utterances for each speaker on each digit, totaling 2700 training samples and 300 validation samples. The models were again all trained using Adam with a learning rate of 0.0001 and a batch size of 64. Each model also had an early stopping criteria where if the validation loss did not improve for two epochs the training stops. In Figure D.1 one can see the training graph in terms of accuracy for the pre-trained Densenet-121 models. As can be seen, both achieve a very high accuracy, but TEL

has a higher accuracy and there is is smaller gap between the train and validation loss.

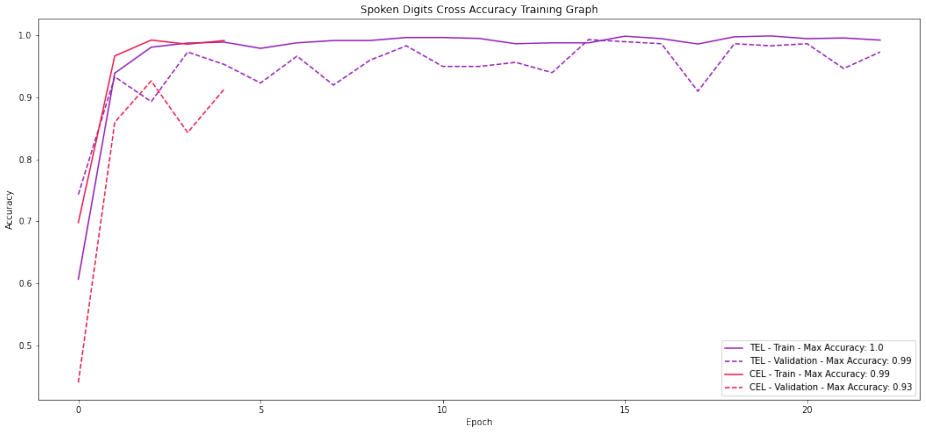


Figure D.1: Training graph of pre-trained Densenet-121 model on the FSDD dataset in terms of accuracy.

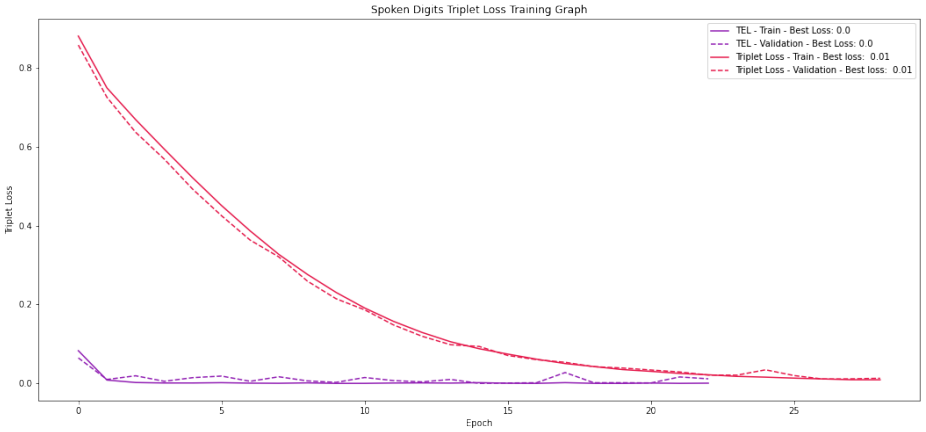


Figure D.2: Training graph of pre-trained Densenet-121 model on the FSDD dataset in terms of Triplet loss.

In Figure D.2 one can see the training graph in terms of the Triplet loss. Here one can see that both models again perform roughly the same, but the speed at which the TEL method reaches its minimum is much faster than the pure Triplet loss

training method. This is the biggest differentiator between the two methods when applied to this dataset.

When looking at the embeddings generated (using UMAP again) in the Figure D.3, one will see there is not much difference here but it is important to note that the TEL method achieved these clusters after just the second epoch.

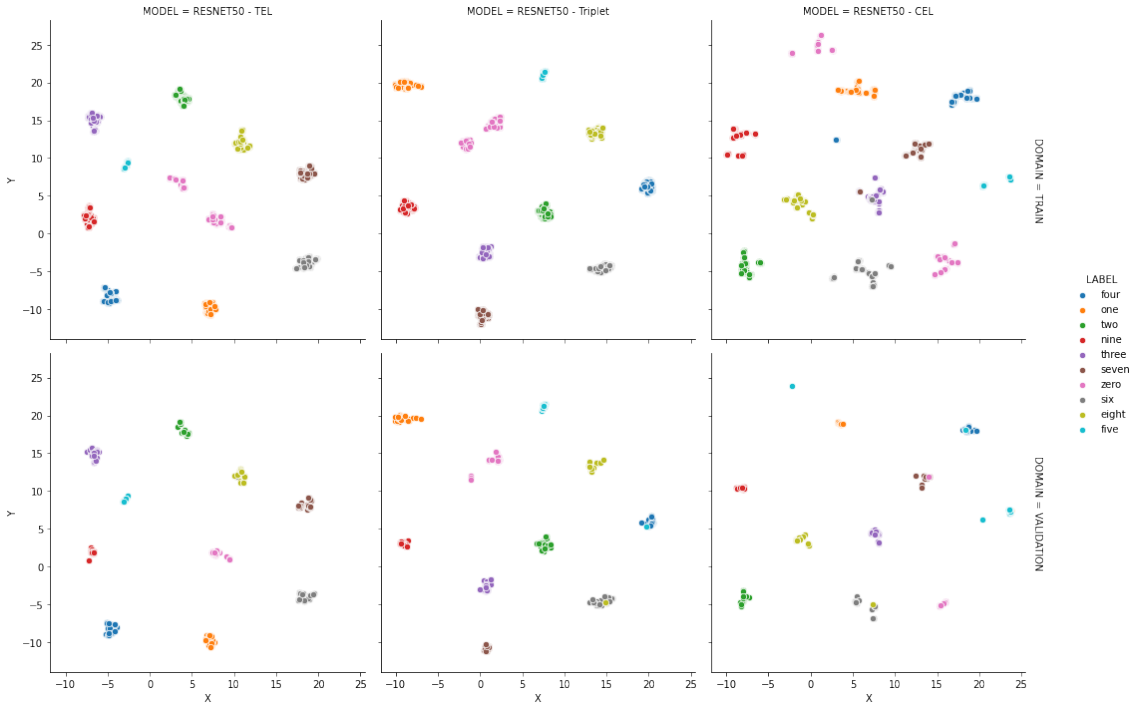


Figure D.3: Projection of the embeddings generated on the FSDD dataset.

D.2 GTZAN - Music Genre Classification

The GTZAN dataset contains 100 recordings for each of the ten distinct classes, namely blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae, and rock. Each of the recordings are 30 seconds long. The version of the dataset used in this project can be found on Kaggle here, where the audio is already converted to spectrograms for ease of use.

The data is split into a train and validation set, with the validation set consisting of 21 recordings for each class. The exact model architecture and training process was followed as was done in the previous section with the FSDD dataset. In Figure D.4 one can see the training graph in terms of accuracy for the pre-trained Densenet-121 models. As can be seen, the validation accuracy for the model trained using CEL

never increases while the validation accuracy for the TEL model reaches close to 60% accuracy.

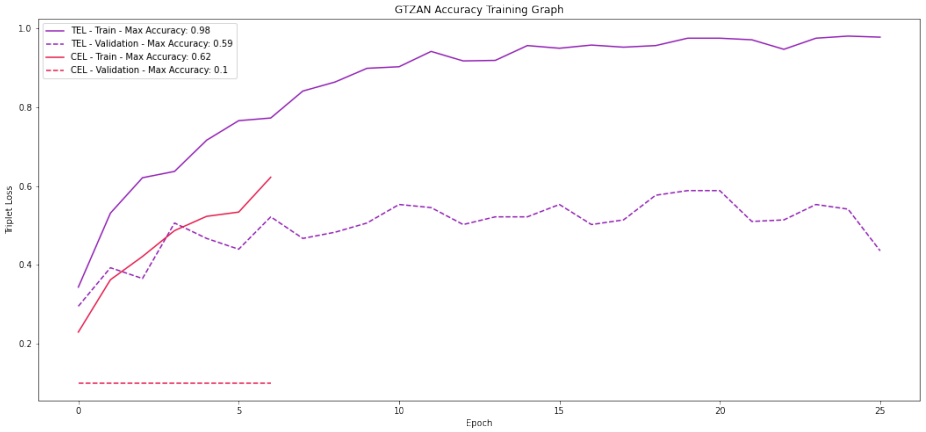


Figure D.4: Training graph of pre-trained Densenet-121 model on the GTZAN dataset in terms of accuracy.

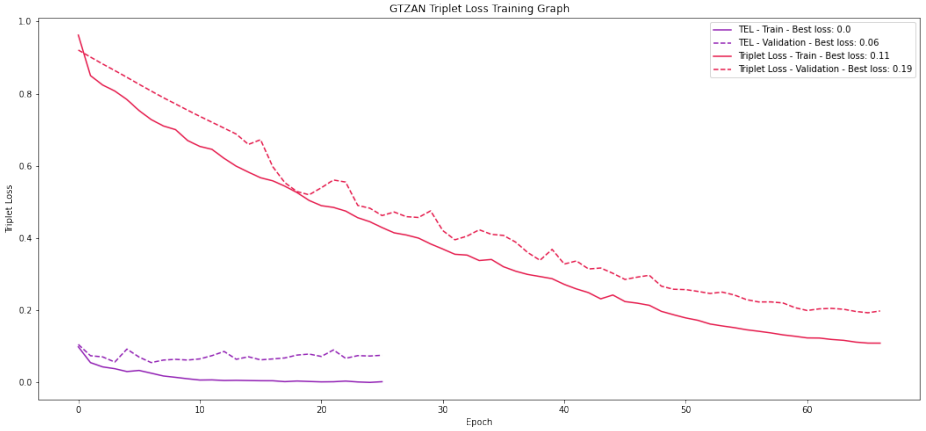


Figure D.5: Training graph of pre-trained Densenet-121 model on the GTZAN dataset in terms of Triplet loss.

In Figure D.5 one can see the training graph in terms of the Triplet loss. One again sees the same trend as in the previous section where the models, trained using TEL and Triplet loss, reach roughly the same performance but with the model trained

using TEL reaching it much quicker. The TEL trained model though is able to seperate the classes based on the Triplet loss value better than the other model.

Bibliography

[Abdullah et al., 2020] Abdullah, B. M., Avgustinova, T., Möbius, B., and Klakow, D. (2020). Cross-domain adaptation of spoken language identification for related languages: The curious case of slavic languages.

[Barnard et al., 2014] Barnard, E., Davel, M. H., van Heerden, C., De Wet, F., and Badenhorst, J. (2014). The nchlt speech corpus of the south african languages. Workshop Spoken Language Technologies for Under-resourced Languages (SLTU).

[Bartz et al., 2017] Bartz, C., Herold, T., Yang, H., and Meinel, C. (2017). Language identification using deep convolutional recurrent neural networks. In *International Conference on Neural Information Processing*, pages 880–889. Springer.

[Blackman and Tukey, 1958] Blackman, R. B. and Tukey, J. W. (1958). The measurement of power spectra from the point of view of communications engineering—part i. *Bell System Technical Journal*, 37(1):185–282.

[Bredin, 2017] Bredin, H. (2017). Tristounet: triplet loss for speaker turn embedding. In *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5430–5434. IEEE.

[Davel et al., 2012] Davel, M. H., Heerden, C. J. v., and Barnard, E. (2012). Validating smartphone-collected speech corpora. In *Spoken Language Technologies for Under-Resourced Languages*.

[De Wet et al., 2017] De Wet, F., Kleynhans, N., Van Compernolle, D., and Sahraeian, R. (2017). Speech recognition for under-resourced languages: Data sharing in hidden markov model systems. *South African Journal of Science*, 113(1-2):1–9.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

[Fergus, 2015] Fergus, R. (2015). Neural networks by rob fergus, machine learning summer school 2015. Technical report. Also available as `http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf`.

[Ganin and Lempitsky, 2014] Ganin, Y. and Lempitsky, V. (2014). Unsupervised domain adaptation by backpropagation.

[Goodfellow et al., 2016] Goodfellow, I. J., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA, USA. `http://www.deeplearningbook.org`.

[Han et al., 2020] Han, W., Zhang, Z., Zhang, Y., Yu, J., Chiu, C.-C., Qin, J., Gulati, A., Pang, R., and Wu, Y. (2020). Contextnet: Improving convolutional neural networks for automatic speech recognition with global context. *arXiv preprint arXiv:2005.03191*.

[Han and Lee, 2016] Han, Y. and Lee, K. (2016). Convolutional neural network with multiple-width frequency-delta data augmentation for acoustic scene classification. *IEEE AASP challenge on detection and classification of acoustic scenes and events*.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Henselmans et al., 2013] Henselmans, D., Niesler, T., and van Leeuwen, D. (2013). Phoneme-and word-based language identification of south african languages using lwazi.

[Huang et al., 2018] Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2018). Densely connected convolutional networks.

[Kecman, 2005] Kecman, V. (2005). Support vector machines–an introduction. In *Support vector machines: theory and applications*, pages 1–47. Springer.

[Khosla et al., 2020] Khosla, P., Teterwak, P., Wang, C., Sarna, A., Tian, Y., Isola, P., Maschinot, A., Liu, C., and Krishnan, D. (2020). Supervised contrastive learning.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kinnunen et al., 2006] Kinnunen, T., Hautamäki, V., and Fränti, P. (2006). On the use of long-term average spectrum in automatic speaker recognition. In *5th Internat. Symposium on Chinese Spoken Language Processing (ISCSLP'06), Singapore*, pages 559–567.

[Korkut et al., 2020] Korkut, C., Haznedaroglu, A., and Arslan, L. (2020). Comparison of deep learning methods for spoken language identification. In *International Conference on Speech and Computer*, pages 223–231. Springer.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Lipton et al., 2015] Lipton, Z. C., Berkowitz, J., and Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*.

[Lopez et al., 2018] Lopez, J., Brummer, N., and Dehak, N. (2018). End-to-end versus embedding neural networks for language recognition in mismatched conditions. pages 112–119.

[Malhotra et al., 2015] Malhotra, P., Vig, L., Shroff, G., and Agarwal, P. (2015). Long short term memory networks for anomaly detection in time series. In *Proceedings*, volume 89, pages 89–94. Presses universitaires de Louvain.

[Margolis et al., 2018] Margolis, B., Ghei, M., and Pardo, B. (2018). Applying triplet loss to siamese-style networks for audio similarity ranking.

[Martin and Greenberg, 2010] Martin, A. F. and Greenberg, C. S. (2010). The 2009 nist language recognition evaluation. In *Odyssey*, volume 30.

[Mateju et al., 2018] Mateju, L., Cerva, P., Zdánskỳ, J., and Safarik, R. (2018). Using deep neural networks for identification of slavic languages from acoustic signal. In *INTERSPEECH*, pages 1803–1807.

[Mbogho and Katz, 2010] Mbogho, A. and Katz, M. (2010). The impact of accents on automatic recognition of south african english speech: a preliminary investigation. In *Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, pages 187–192.

[McFee et al., 2015] McFee, B., Raffel, C., Liang, D., Ellis, D. P., McVicar, M., Battenberg, E., and Nieto, O. (2015). librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8.

[McInnes et al., 2020] McInnes, L., Healy, J., and Melville, J. (2020). Umap: Uniform manifold approximation and projection for dimension reduction.

[McInnes et al., 2018] McInnes, L., Healy, J., Saul, N., and Grossberger, L. (2018). Umap: Uniform manifold approximation and projection. *The Journal of Open Source Software*, 3(29):861.

[Meng et al., 2017] Meng, Z., Chen, Z., Mazalov, V., Li, J., and Gong, Y. (2017). Unsupervised adaptation with domain separation networks for robust speech recognition. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 214–221. IEEE.

[Mingote et al., 2019] Mingote, V., Castan, D., McLaren, M., Nandwana, M. K., Giménez, A. O., Lleida, E., and Miguel, A. (2019). Language recognition using triplet neural networks. In *INTERSPEECH*, pages 4025–4029.

[Mingote et al., 2020] Mingote, V., Miguel, A., Ortega, A., and Lleida, E. (2020). Optimization of the area under the roc curve using neural network supervectors for text-dependent speaker verification. *Computer Speech & Language*, 63:101078.

[Montavon, 2009] Montavon, G. (2009). Deep learning for spoken language identification. In *NIPS Workshop on deep learning for speech recognition and related applications*, pages 1–4. Citeseer.

[Nair and Hinton, 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Icml*.

[Nam et al., 2012] Nam, J., Herrera, J., Slaney, M., and Smith III, J. O. (2012). Learning sparse feature representations for music annotation and retrieval. In *ISMIR*, pages 565–570.

[Nam et al., 2013] Nam, J., Hyung, Z., and Lee, K. (2013). Acoustic scene classification using sparse feature learning and selective max-pooling by event detection. *IEEE AASP Challenge on Detection and Classification of Acoustic Scenes and Events*.

[Palanisamy et al., 2020] Palanisamy, K., Singhania, D., and Yao, A. (2020). Rethinking cnn models for audio classification. *arXiv preprint arXiv:2007.11154*.

[Panayotov et al., 2015] Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. (2015). Librispeech: An asr corpus based on public domain audio books. pages 5206–5210.

[Park et al., 2019] Park, D. S., Chan, W., Zhang, Y., Chiu, C.-C., Zoph, B., Cubuk, E. D., and Le, Q. V. (2019). Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*.

[Pons et al., 2020] Pons, J., Pascual, S., Cengarle, G., and Serrà, J. (2020). Upsampling artifacts in neural audio synthesis. *arXiv preprint arXiv:2010.14356*.

[Rabiner and Juang, 1986] Rabiner, L. and Juang, B. (1986). An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1):4–16.

[Revay and Teschke, 2019] Revay, S. and Teschke, M. (2019). Multiclass language identification using deep learning on spectral images of audio signals.

[Ruder, 2016] Ruder, S. (2016). An overview of gradient descent optimization algorithms.

[Rumelhart et al., 1985] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

[Sarthak et al., 2019] Sarthak, Shukla, S., and Mittal, G. (2019). Spoken language identification using convnets.

[Schroff et al., 2015] Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.

[Shuyo, 2010] Shuyo, N. (2010). Language detection library for java.

[Szegedy et al., 2014] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions.

[Szegedy et al., 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

[Van der Maaten and Hinton, 2008] Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

[Van Heerden et al., 2009] Van Heerden, C., Barnard, E., and Davel, M. (2009). Basic speech recognition for spoken dialogues.

[van Heerden et al., 2016] van Heerden, C., Kleynhans, N., and Davel, M. H. (2016). Improving the lwazi asr baseline.

[Zuo et al., 2015] Zuo, Z., Shuai, B., Wang, G., Liu, X., Wang, X., Wang, B., and Chen, Y. (2015). Convolutional recurrent neural networks: Learning spatial dependencies for image representation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 18–26.