University of New South Wales

COMP9991/COMP9992 - T2 2023 Report

Master of Information Technology

Chromatic Derivatives for Speech Recognition

Louis Milhiet | z5374550

Supervisor: Dr Alan Blair

Assessor: Dr Aleksandar Ignjatovic

August 9, 2023

Contents

List of Figures

List of Tables

1 Introduction

The main objective of this research project is to investigate the use of Chromatic Derivatives (CDs) for speech and phoneme recognition. CDs can be used in speech processing as an alternative to other well-known techniques such as Mel-Frequency Cepstral Coefficients (MFCC) or simply taking the raw signal. We will study CDs as a pre-processing technique used in combination with different types of Deep Neural Networks (DNN). We will study specifically the TIMIT dataset using the Phoneme Error Rate (PER) metric. This research project relies on the use of SpeechBrain which is a new open-source Python library for speech recognition [2]. The code used during the project can be found on GitHub.

2 Problem definition

2.1 Chromatic Derivatives

To better understand the type of prepocessing method we are dealing with, we will briefly introduce some theory behind Chromatic Derivatives (CDs) based on several articles such as [3], [4] and [5].
Let us consider the Legendre polynomials noted as P_n^L

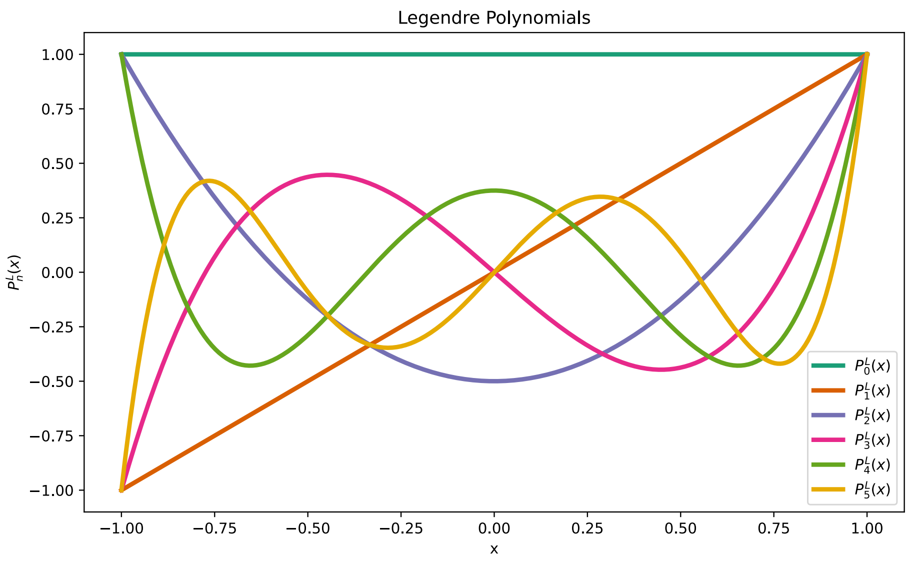


Figure 1: Legendre polynomials $P_n^L(X)$ for $n \in [\![0,5]\!]$ and $X \in [-1,1]$

In figure 1, we can see the first 6 Legendre polynomials. They can be defined using Rodrigues' formula :

$$P_n^L(X) = \frac{1}{2^n n!}\frac{d^n}{dX^n}(X^2-1)^n$$

Legendre polynomials are either even polynomial functions if n is even or odd polynomial functions if n is even (i.e. P_n^L contains only powers of the same parity as n).
Generally, let use consider $(a,b) \in \mathbb{R}$ and E the vector space of continuous functions over (a,b) such that

$$\forall f \in E,\ ||f||_2 = \sqrt{\int_a^b |f(x)|^2 w(x)dx} < +\infty$$

and we define a scalar product over E

$$\forall f \in E,\ \forall g \in E,\ \langle f,g\rangle = \int_a^b f(x)g(x)w(x)dx$$

A chromatic weight $w(x)$ is a non-negative weight that satisfies

$$\int_{-\infty}^{+\infty} e^{\alpha|x|} w(x)dx < +\infty$$

For some $\alpha > 0$.
The Gram-Schmidt process with respect to this scalar product ensures that we can build a sequences of orthogonal polynomials P_n such that $\forall n \in \mathbb{N}, deg(P_n) = n$.
Legendre polynomials are orthogonal over $(-1, 1)$ with respect to the weight $w(x) = 1$ defined over $(-1, 1)$.

$$\forall m \in \mathbb{N}, \forall n \in \mathbb{N}, m \neq n, \quad \left\langle P_m^L, P_n^L \right\rangle = \int_{-1}^{1} P_m^L(x) P_n^L(x) dx = 0$$

With the additional standardization condition $\forall n \in \mathbb{N}, P_n^L(1) = 1$, we have

$$\forall m \in \mathbb{N}, \forall n \in \mathbb{N}, m \neq n, \quad \left\langle P_m^L, P_n^L \right\rangle = \frac{2}{2n+1}\delta(m-n)$$

Using the Gram-Schmidt process, we can find the first few Legendre Polynomials :

- $P_0^L(X) = 1$
- $p_1^L(X) = X, \ P_1^L(X) = X - \frac{\left\langle P_0^L, p_1^L \right\rangle}{||P_0^L||_2^2} P_0^L = X$
- $p_2^L(X) = X^2, \ c_2 \in \mathbb{R}^*, P_2^L(X) = c_2(X^2 - \frac{\left\langle P_0^L, p_2^L \right\rangle}{||P_0^L||_2^2} P_0^L - \frac{\left\langle P_1^L, p_2^L \right\rangle}{||P_1^L||_2^2} P_1^L) = c_2(X^2 - \frac{\frac{2}{3}}{2}) = c_2(X^2 - \frac{1}{3})$ and with the condition $P_2^L(1) = 1$,we find $c_2 = \frac{3}{2}$ and $P_2^L(X) = \frac{3}{2}X^2 - \frac{1}{2}$
- ...

Legendre Polynomials can also be defined by recursion

$$P_0^L(X) = 1$$

$$P_1^L(X) = X$$

$$\forall n \in \mathbb{N}^*, \ (n+1)P_{n+1}^L(X) = (2n+1)XP_n^L(X) - nP_{n-1}^L(X)$$

Let us defined $\tilde{P_n^L}$ the polynomials obtained by normalizing and scaling P_n^L such that s

$$\forall m \in \mathbb{N}, \forall n \in \mathbb{N}, m \neq n, \quad \left\langle \tilde{P_m^L}, \tilde{P_n^L} \right\rangle = \frac{1}{2\pi} \int_{-\pi}^{\pi} \tilde{P_m^L}(\omega) \tilde{P_n^L}(\omega) d\omega = \delta(m-n)$$

If we apply the Gram-Schmidt process, we get

- $\tilde{P_0^L}(\omega) = 1$

- $\tilde{p_1^L}(\omega) = \omega,\ c_1 \in \mathbb{R}^*, \tilde{P_1^L}(\omega) = c_1(\omega - \frac{\left\langle \tilde{P_0^L}, \tilde{p_1^L} \right\rangle}{||\tilde{P_0^L}||_2^2}\tilde{P_0^L}) = c_1\omega$ and $\left\langle \tilde{P_1^L}, \tilde{P_1^L} \right\rangle = \frac{c_1^2\omega^2}{3} = 1$, $\tilde{P_1^L}(\omega) = \frac{\sqrt{3}}{\pi}\omega$
- $\tilde{p_2^L}(\omega) = \omega^2,\ c_2 \in \mathbb{R}^*, \tilde{P_2^L}(\omega) = c_2(\omega^2 - \frac{\left\langle \tilde{P_0^L}, \tilde{p_2^L} \right\rangle}{||\tilde{P_0^L}||_2^2}\tilde{P_0^L} - \frac{\left\langle \tilde{P_1^L}, \tilde{p_2^L} \right\rangle}{||\tilde{P_1^L}||_2^2}\tilde{P_1^L}) = c_2(\omega^2 - \frac{\pi^2}{3})$ and $\left\langle \tilde{P_2^L}, \tilde{P_2^L} \right\rangle = \frac{c_2^2}{2\pi}\frac{8\pi^5}{45} = 1$. Therefore, $c_2 = \frac{3\sqrt{5}}{2\pi^2}$ and $\tilde{P_2^L}(\omega) = \frac{\sqrt{5}(3\omega^2 - \pi^2)}{2\pi^2}$
- ...

Legendre polynomials can be computed with the following recurrence relation

$$\forall n \in \mathbb{N}^*,\ \tilde{P_{n+1}^L}(X) = \frac{X}{\gamma_n}\tilde{P_n^L}(X) - \frac{\gamma_{n-1}}{\gamma_n}\tilde{P_{n-1}^L}(X) \tag{1}$$

with

$$\gamma_n = \frac{\pi(n+1)}{\sqrt{4(n+1)^2 - 1}}$$

We can Chromatic Derivatives (CDs) associated with Legendre polynomials are defined as the following operator polynomials

$$\mathcal{K}^n = (-i)^n \tilde{P_n^L}(i\frac{d}{dt}) \tag{2}$$

which corresponds to

$$\mathcal{K}^0[f] = f$$

$$\mathcal{K}^1[f] = \frac{\sqrt{3}}{\pi}\frac{df}{dt}$$

$$\mathcal{K}^2[f] = \frac{\sqrt{5}(3\frac{d^2f}{dt^2} + \pi^2 f)}{2\pi^2}$$

With the recurrence relation from equation (1), we have

$$\forall n \in \mathbb{N}^*,\ \mathcal{K}^{n+1}[f] = \frac{1}{\gamma_n}\frac{d(\mathcal{K}^n[f])}{dt} + \frac{\gamma_{n-1}}{\gamma_n}\mathcal{K}^{n-1}[f]$$

With equation (2) have

$$\mathcal{K}^n\left[e^{i\omega t}\right] = (-i)^n \tilde{P_n^L}(i\frac{de^{i\omega t}}{dt})$$

$$\mathcal{K}^n\left[e^{i\omega t}\right] = (-i)^n \tilde{P_n^L}(-\omega)e^{i\omega t}$$

We know $\tilde{P_n^L}$ contains only powers of the same parity as n, therefore

$$\mathcal{K}^n\left[e^{i\omega t}\right] = (-i)^n(-1)^n \tilde{P_n^L}(\omega)e^{i\omega t}$$

$$\mathcal{K}^n\left[e^{i\omega t}\right] = i^n \tilde{P_n^L}(\omega)e^{i\omega t} \tag{3}$$

We define $\boldsymbol{BL}(\pi)$ the space of band limited signals of finite energy (i.e. space of continuous L^2 functions with a Fourier transform supported within $[-\pi,\pi]$. For a signal $f \in \boldsymbol{BL}(\pi)$, the inverse Fourier transform formula of the nth derivative of f is given by

$$\frac{d^n f}{dt^n}(t) = \frac{1}{2\pi}\int_{-\pi}^{\pi}(i\omega)^n \widehat{f}(\omega)e^{i\omega t}d\omega$$
$$\frac{d^n f}{dt^n}(t) = \frac{1}{2\pi}\int_{-\pi}^{\pi}\widehat{f}(\omega)\frac{d^n e^{i\omega t}}{dt^n}d\omega \tag{4}$$

Consequently, with equation (3) and equation (4) we have

$$\mathcal{K}^n\left[f\right](t) = \frac{1}{2\pi}\int_{-\pi}^{\pi} i^n \tilde{P}_n^L(\omega)\widehat{f}(\omega)e^{i\omega t}d\omega$$

with $\widehat{f}(\omega)$ the Fourier transform of $f(t)$.
We can also show that

$$\mathcal{K}^n\left[sinc\right](t) = (-1)^n\sqrt{2n+1}j_n(\pi t) \tag{5}$$

where $j_n(x)$ is the spherical Bessel function of the first kind of order n which are shown in figure 2.

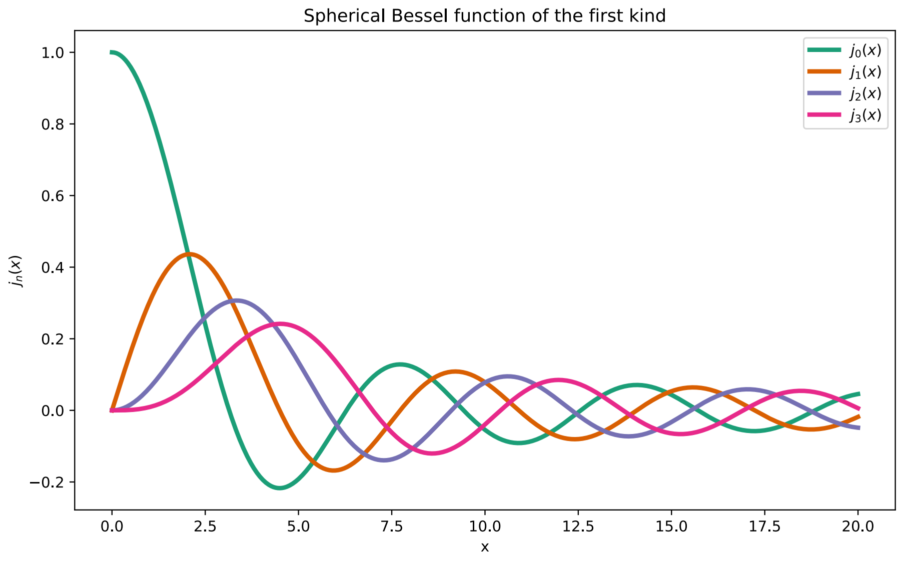


Figure 2: Spherical Bessel functions $j_n(x)$ for $n \in [\![0,3]\!]$ and $x \in (0,20)$

For every function $f \in \boldsymbol{BL}(\pi)$, the Shannon Theorem gives

$$f(t) = \sum_{n=-\infty}^{+\infty} f(n)sinc(t-n) \tag{6}$$

With equation (5) and equation (6), we can show that

$$\mathcal{K}^k\left[f\right](t) = \sum_{n=-\infty}^{+\infty} f(n)\mathcal{K}^k\left[sinc\right](t-n) = \sum_{n=-\infty}^{+\infty} f(n)(-1)^n\sqrt{2n+1}j_n(\pi t)$$

$$\mathcal{K}^k\left[f\right](t) = \left(\mathcal{K}^k\left[sinc\right] * f\right)(t) \tag{7}$$

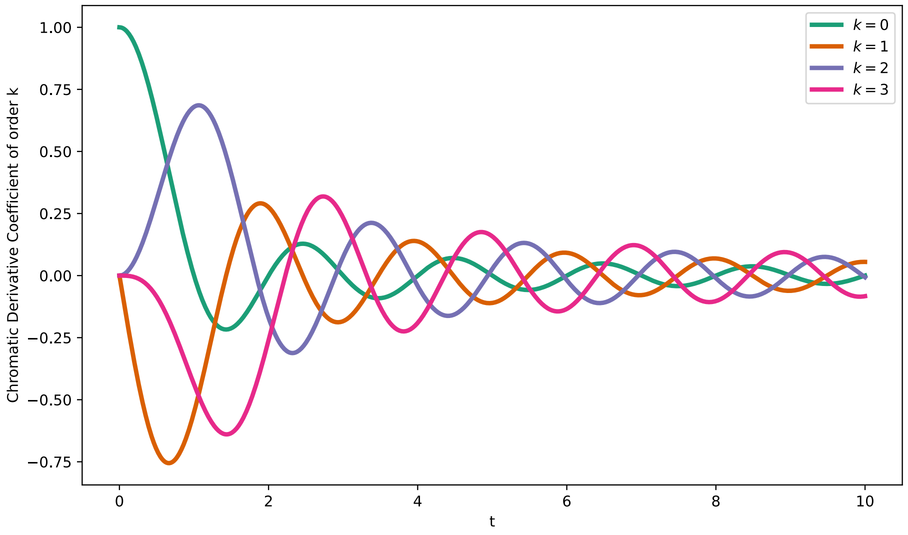


Figure 3: Chromatic Derivatives $(-1)^n\sqrt{2n+1}j_n(\pi t)$ for $t \in (0, 10)$

The values of $(-1)^n\sqrt{2n+1}j_n(\pi t)$ are shown in figure 3.
In practice, we use Remez exchange algorithm to compute the filter coefficients that will be used to compute the convolution shown in equation (7). f would correspond to an audio signal which will be convoluted with the filters shown in figure 4. In this figure, we can see the chromatic derivatives filters of order $k = 1$ to $k = 6$.

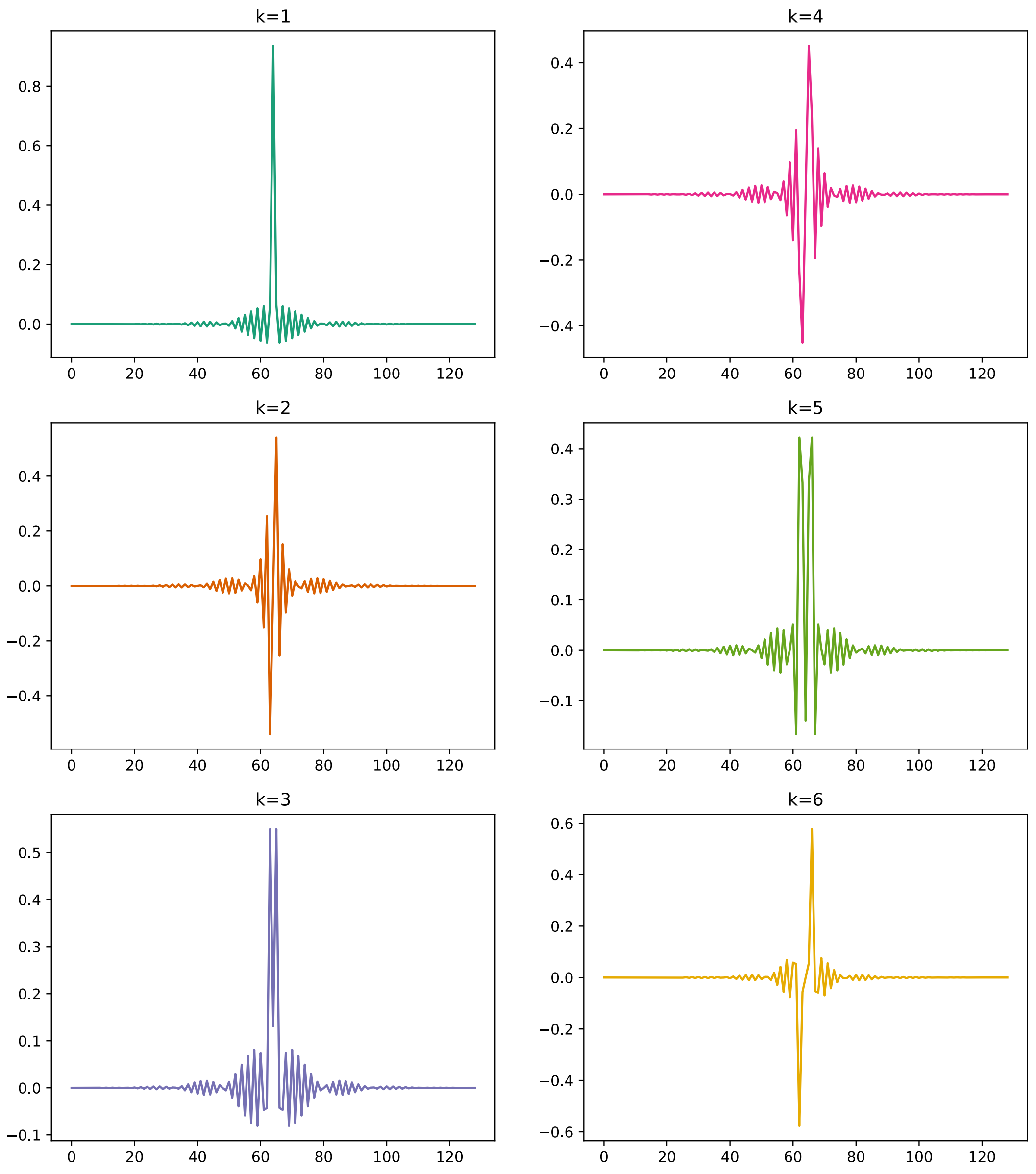


Figure 4: Filter coefficients of the Chromatic Derivatives (corresponding to the Legendre polynomials) computed with the Remez exchange algorithm

We can also consider other orthogonal polynomials sequences such as Chebyshev polynomials of the first kind $\tilde{P_n^C}$ which are defined over $(-1,1)$ with $w(x) = \frac{1}{\sqrt{1-x^2}}$.

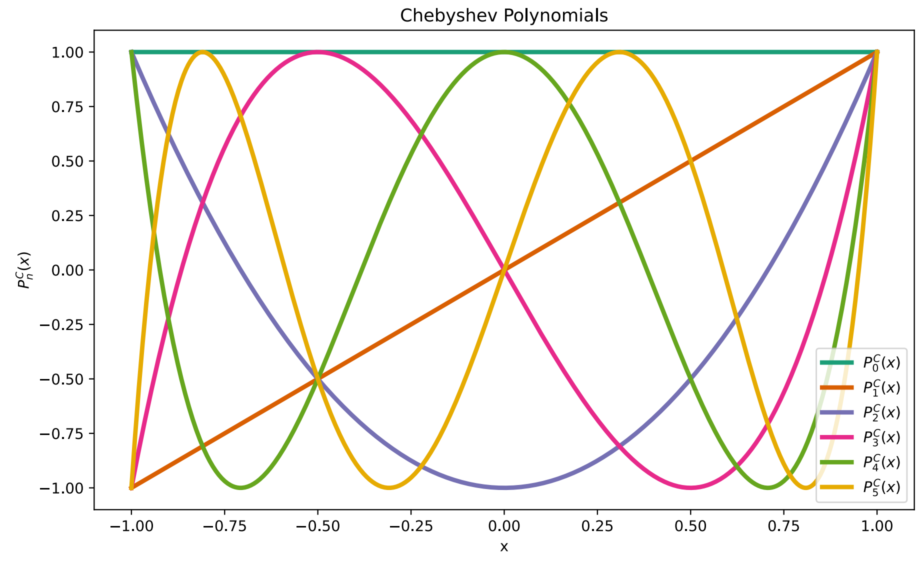


Figure 5: Chebyshev polynomials $P_n^C(X)$ for $n \in [\![0,5]\!]$ and $X \in [-1,1]$

Chebyshev polynomials are defined in figure 5. An alternative is to define CDs with Chebyshev polynomials instead of Legendre polynomials. As mentioned in [4]. Chebyshev Polynomials can also be defined by recursion

$$P_0^C(X) = 1$$

$$P_1^C(X) = X$$

$$\forall n \in \mathbb{N}^*,\ P_{n+1}^C(X) = 2XP_n^C(X) - P_{n-1}^C(X)$$

Similarly to how we defined Legendre polynomials by normalizing and scaling them, we can define defined $\tilde{P_n^C}$

$$\forall m \in \mathbb{N}, \forall n \in \mathbb{N}, m \neq n, \quad \left\langle \tilde{P_m^C}, \tilde{P_n^C} \right\rangle = \frac{1}{\pi} \int_{-\pi}^{\pi} \frac{\tilde{P_m^C}(\omega)\tilde{P_n^C}(\omega)}{\sqrt{\pi^2 - \omega^2}} d\omega = \delta(m-n)$$

which correspond to the following chromatic derivatives

$$\bar{\mathcal{K}}[f] = f$$

$$\bar{\mathcal{K}}^1[f] = \frac{1}{\pi}\frac{df}{dt}$$

$$\forall n \in \mathbb{N}^*,\ \bar{\mathcal{K}}^{n+1}[f] = \frac{2}{\pi}\frac{d(\bar{\mathcal{K}}^n[f])}{dt} + \bar{\mathcal{K}}^{n-1}[f]$$

2.2 Automatic Speech Recognition (ASR) : the TIMIT dataset

Automatic Speech Recognition (ASR) refers to technologies which enable a program to process human speech into a written format. The DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus (TIMIT) is a widely used speech recognition dataset containing recordings of 630 speakers of American English. There are other well-known dataset such as LibriSpeech (corpus of approximately 1000 hours) or CommonVoice (9283 recorded hours). It is important to note that the TIMIT dataset is relatively small (around 4 hours 40 min), which allows models to be trained relatively rapidly. The TIMIT dataset is considered to be a standard benchmark dataset for evaluating speech recognition systems. This dataset includes audio signals of variable duration. The sentences from the TIMIT dataset have been labelled with 61 different phonemes. The metric commonly used for this dataset is the Phoneme Error Rate (PER) which is derived from the Levenstein distance. The PER is defined as

$$PER = \frac{S+D+I}{S+D+C} = \frac{S+D+I}{N}$$

with

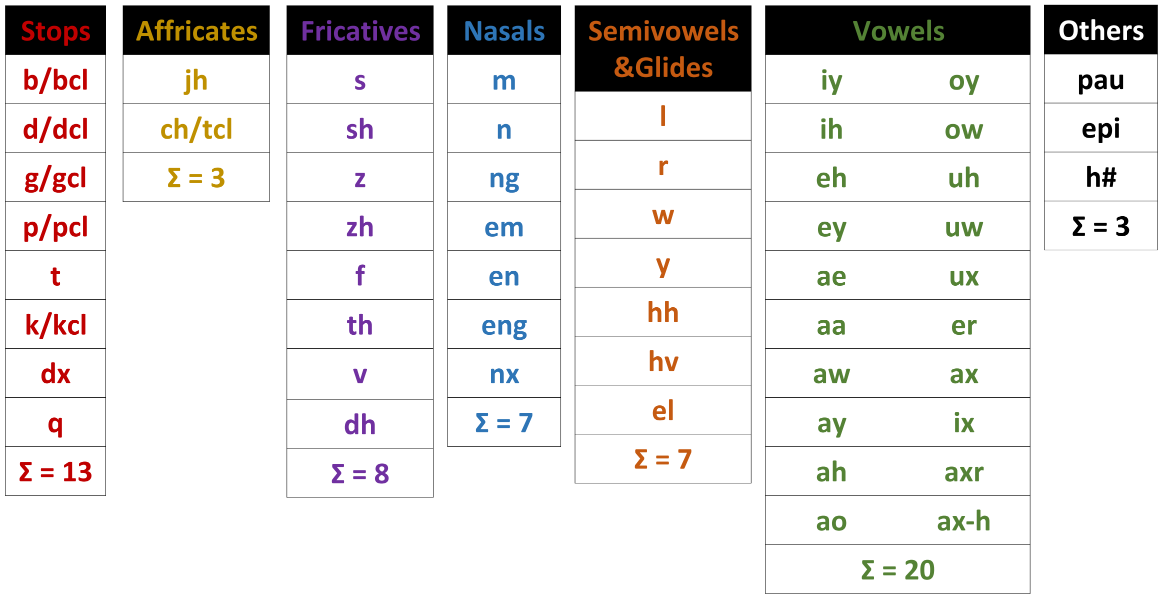
- S : number of substitutions
- D : number of deletions
- I : number of insertions
- C : number of correct phoneme predictions
- N : number of phonemes (ground truth)

The TIMIT dataset as 3 types of sentences as shown in table 1. We only consider the sentences from the SX and SI categories as the SA category only contains 2 different sentences. We would rather have a large variety of sentences than a large variety of speakers as our goals is to predict phonemes. The sentences from the SA category could be more useful if we were trying to predict who is the speaker (gender, age ...). Therefore, we always drop the sentences from the dataset corresponding to the SA category.

Sentence Type	Number of sentences	Number of speakers
Dialect (SA)	2	630
Compact (SX)	450	7
Dialect (SI)	1890	1

Table 1: Repartition of the sentences in the TIMIT dataset

The 61 phonemes can be classified in the categories shown in figure 6.



Stops	Affricates	Fricatives	Nasals	Semivowels &Glides	Vowels		Others
b/bcl	jh	s	m	l	iy	oy	pau
d/dcl	ch/tcl	sh	n	r	ih	ow	epi
g/gcl	Σ = 3	z	ng	w	eh	uh	h#
p/pcl		zh	em	y	ey	uw	Σ = 3
t		f	en	hh	ae	ux	
k/kcl		th	eng	hv	aa	er	
dx		v	nx	el	aw	ax	
q		dh	Σ = 7	Σ = 7	ay	ix	
Σ = 13		Σ = 8			ah	axr	
					ao	ax-h	
					Σ = 20		

Figure 6: Phoneme categories (61 phonemes)

Here are some examples of words with the possible phonetic transcription :

- *ahead* : ***ax hv eh dcl d***
- *azure* : ***ae zh er***
- *bait* : ***bcl b ey tcl t***
- *bite* : ***bcl b ay tcl t***
- *book* : ***bcl b uh kcl k***
- *boot* : ***bcl b uw tcl t***
- *bottle* : ***bcl b aa tcl t el***
- *bottom* : ***b aa tcl t em***
- *bought* : ***bcl b ao tcl t***
- *bout* : ***bcl b aw tcl t***
- *boy* : ***bcl b oy***
- *button* : ***b ah q en***
- *choke* : ***tcl ch ow kcl k***

- *debit :* ***dcl d eh bcl b ix tcl t***
- *fin :* ***f ih n***
- *gay :* ***gcl g ey***
- *hay :* ***hh ey***
- *joke :* ***dcl jh ow kcl k***
- *mom :* ***m aa m***
- *muddy :* ***m ah dx iy***
- *sing :* ***s ih ng***
- *suspect :* ***s ax-h s pcl p eh kcl k tcl t***
- *then :* ***dh e n***
- *thin :* ***th i n***
- *toot :* ***tcl t ux tcl t***
- *van :* ***v ae n***
- *washington :* ***w aa sh eng tcl t ax n***
- *winner :* ***w ih nx axr***
- *zone :* ***z ow n***

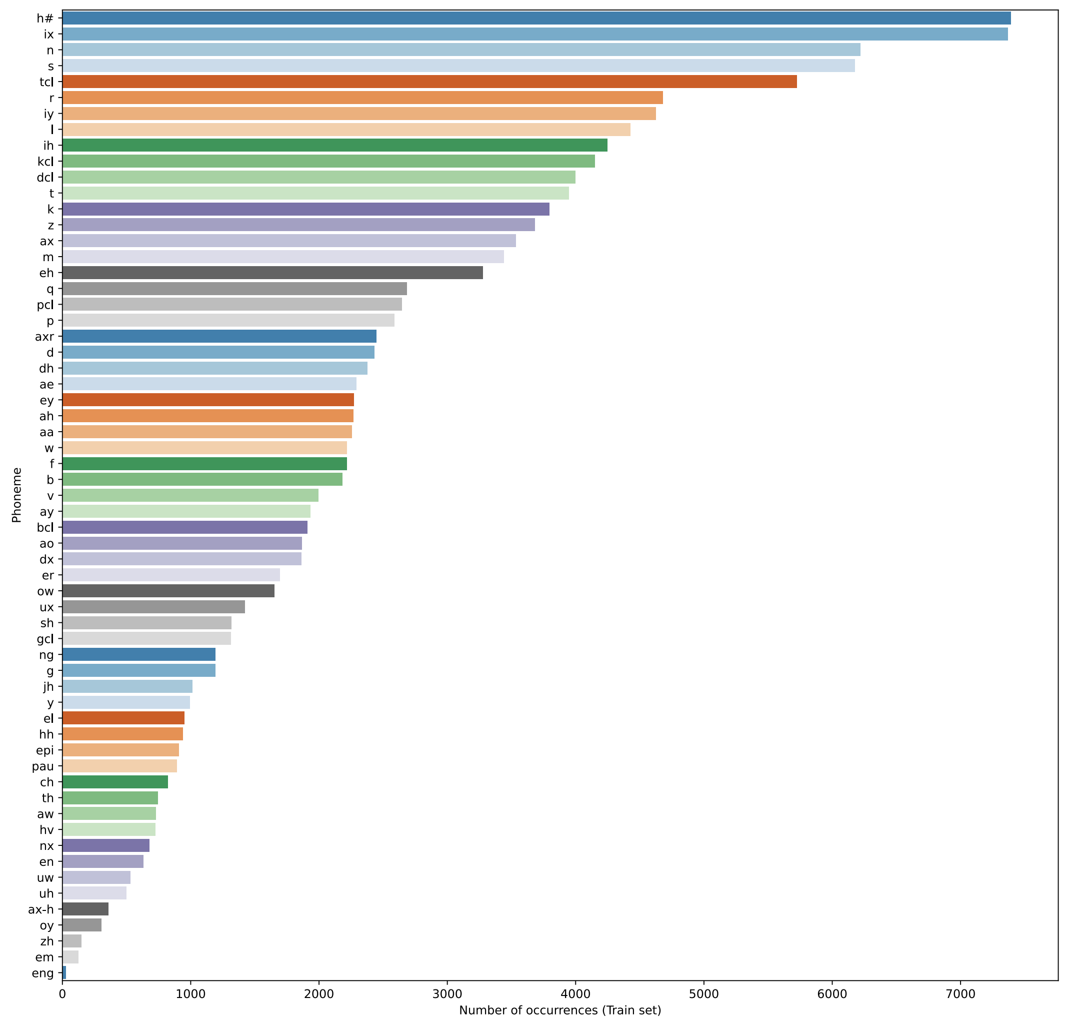
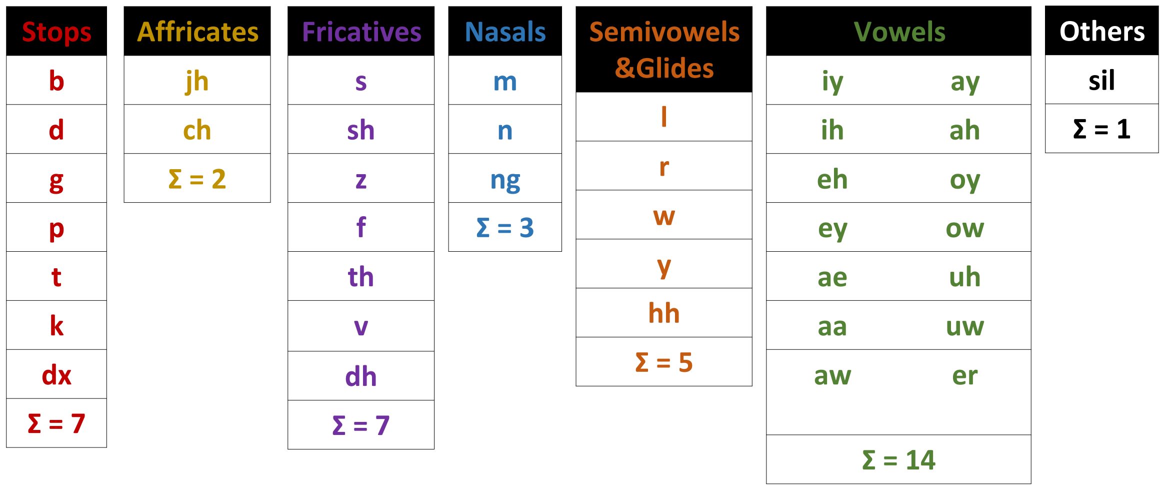


Figure 7: Repartition of phonemes in the TIMIT dataset (61 phonemes)

The repartition of the 61 phonemes in the training set is shown in figure 7. We can see that the phoneme distribution is unbalanced which makes the prediction more challenging.
It seems to be common to combine similar phonemes together (it is done automatically in SpeechBrain). In the project, we mainly work with 39 phonemes instead of 61. The PER corresponding to a model is lower when we consider less phonemes. The 39 phonemes are shown in figure 8 and the number of occurences of each phoneme is shown

in figure 9. As mentioned previously in table 1, we consider sentences from the SI and SX category. In the SX category, we have 450 sentences 7 times (read by 7 different speakers). In figure 7, we consider those sentences 7 times when counting the number of occurrences so it corresponds to the actual train set used to train our models.



Stops	Affricates	Fricatives	Nasals	Semivowels &Glides	Vowels		Others
b	jh	s	m	l	iy	ay	sil
d	ch	sh	n	r	ih	ah	Σ = 1
g	Σ = 2	z	ng	w	eh	oy	
p		f	Σ = 3	y	ey	ow	
t		th		hh	ae	uh	
k		v		Σ = 5	aa	uw	
dx		dh			aw	er	
Σ = 7		Σ = 7			Σ = 14		

Figure 8: Phoneme categories (39 phonemes)

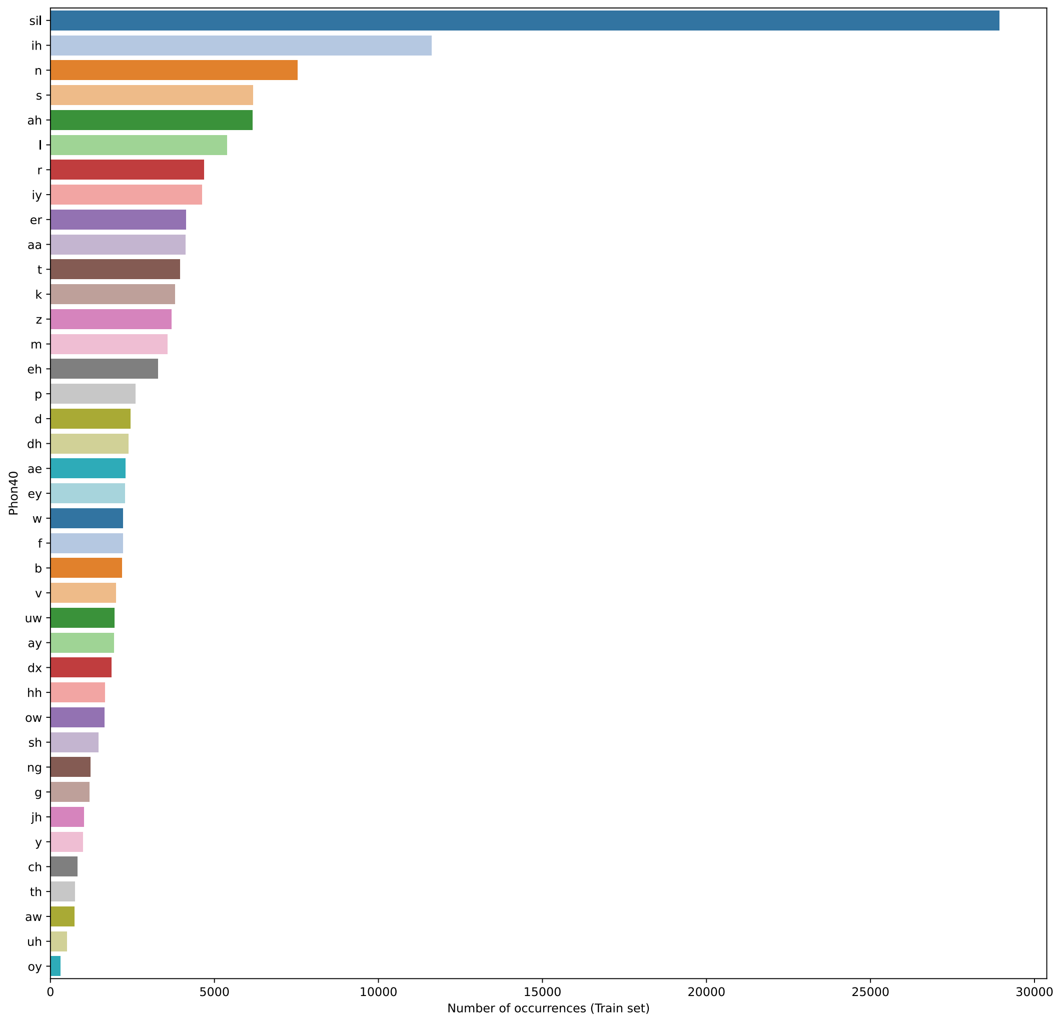


Figure 9: Repartition of phonemes in the TIMIT dataset (39 phonemes)

In figure 8 and figure 9, we consider 39 phonemes where some phonemes from the original 61 were merged together as follows :

- **h#** (begin/end marker, non-speech event), **pau** (pause), **epi** (epenthetic silence), **bcl**, **dcl**, **gcl**, **kcl**, **pcl**, **tcl** → **sil** (silence)

*note : "**cl**" (e.g. in **bcl**) refers to closure intervals as opposed to stop releases (e.g. in **b**)*

- **ao** $\rightarrow$ **aa**
- **ax**, **ax-h** $\rightarrow$ **ah**
- **axr** $\rightarrow$ **er**
- **el** $\rightarrow$ **l**
- **em** $\rightarrow$ **m**
- **en**, **nx** $\rightarrow$ **n**
- **eng** $\rightarrow$ **ng**
- **hv** $\rightarrow$ **hh**
- **ix** $\rightarrow$ **ih**
- **q** $\rightarrow$ $\emptyset$

note : the phoneme ***q*** *is replaced with no phoneme ("empty phoneme")*

- **ux** $\rightarrow$ **uw**
- **zh** $\rightarrow$ **sh**

In figure 9 (compared to figure 7) we can see that the dataset is more unbalanced with 39 phonemes than with 61. In fact, the silence (i.e. **sil**) is the most common phoneme by far with 39 phonemes. It was not the case with 61 phonemes where we had different phonemes depending on the "type of silence" (the most common silence was **h#**).

2.3 Deep Neural Networks (DNN)

2.3.1 Convolutional Neural Networks and correlation matrices

Convolutional Neural Networks (CNN) generally consist of convolutional layers that aim at extracting shift-invariant features from the previous layer. CNN include subsampling or pooling layers which combine the activation functions from multiple units in the previous layer into one unit. They also include fully connected layers that collect spatially diffuse information and an output layer. CNNs perform particularly well on 2-dimensional data such as images. However, our input is an 1-dimensional signal (audio input) as presented in section 2.2.

Using the chromatic filters from figure 4 of order 1 to 48, we obtain a matrix M with N rows. We are then able to compute a correlation matrix R^M which is NxN. The values of the coefficients of R^M are given by

$$R^M_{ij} = \frac{C^M_{ij}}{\sqrt{C^M_{ii} C^M_{jj}}}$$

with C^M the covariance matrix of M.

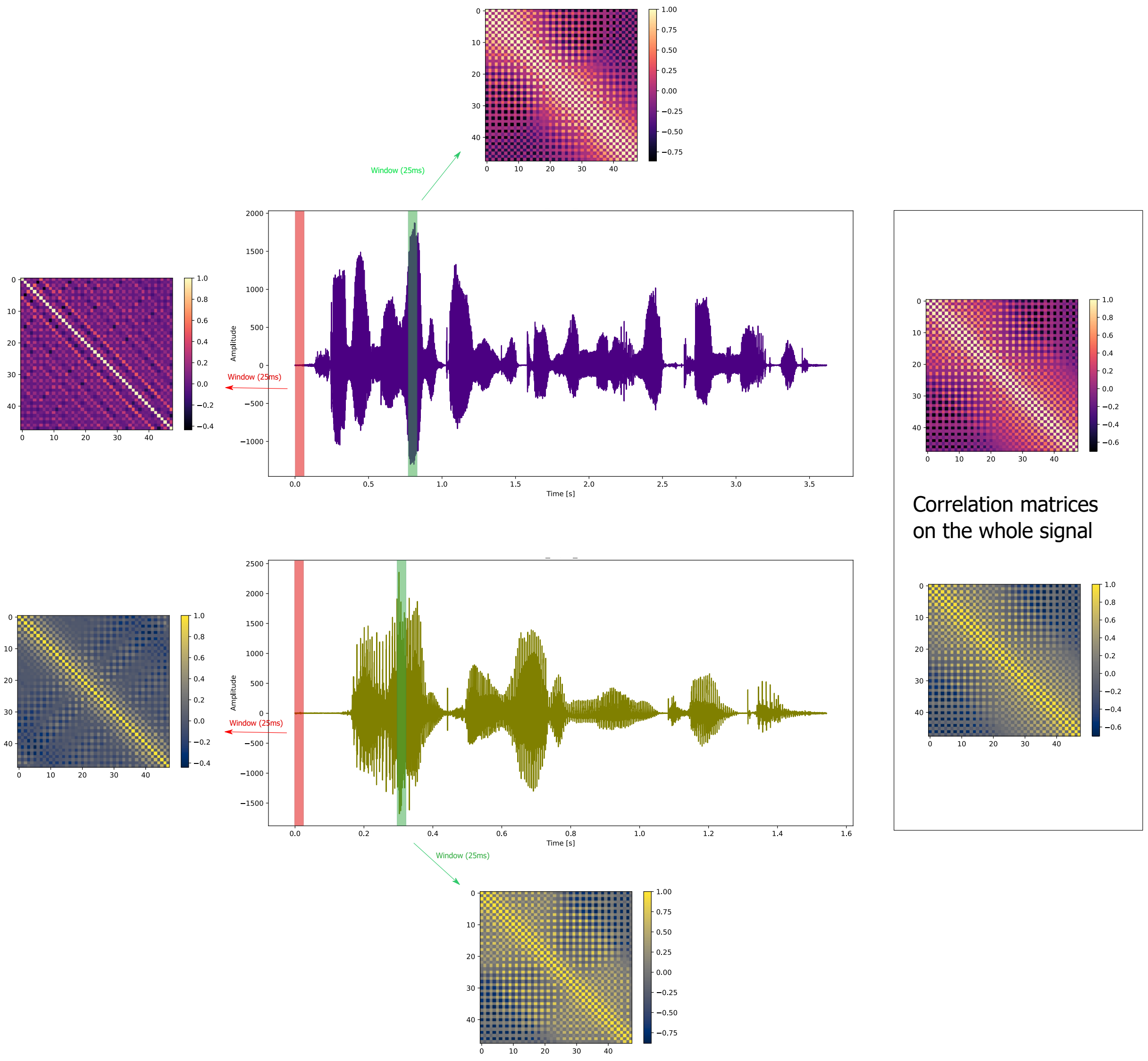


Figure 10: Correlation matrices based on the signal decomposition using 48 chromatic filters

In figure 10, we can see two examples of signals from the TIMIT dataset. The raw signals are shown in the middle and the correlations matrices computed on the whole signal are shown on the right. On the left, we can see the correlation matrices for a 25ms window (red window with $T = 25$ms) at the beginning of the audio signals where the only sound recorded is noise. The signal have a sampling frequency $f_s = 16kHz$. The matrices M corresponding to the window have a shape NxTf_s (with $Tf_s = 0.025 * 16000 = 400$). The green window correspond to a window containing the maximum amplitude of the raw signals and associated correlation matrices are indicated. If we take all the coefficients of each correlation matrix, we get N^2 coefficients (e.g. $48^2 = 2304$ coefficients). However, we can also see in in figure 10 that the correlation matrices are symmetrical. Moreover $R_{ij}^M = 0$ if i and j have a different parity and diagonal coefficient are equal to 1. The number of relevant coefficient is therefore $\frac{N(N-1)}{4}$ with N the number pf CD filter (e.g. with $N = 48$, the number of useful coefficients is $\frac{48(48-1)}{4} = 564$). This expression is valid when N is even (for an odd N, the number of relevant coefficients is $\frac{(N-1)^2}{4}$). Then the correlation matrices are flattened, concatenated together and then fed to the neural network models.

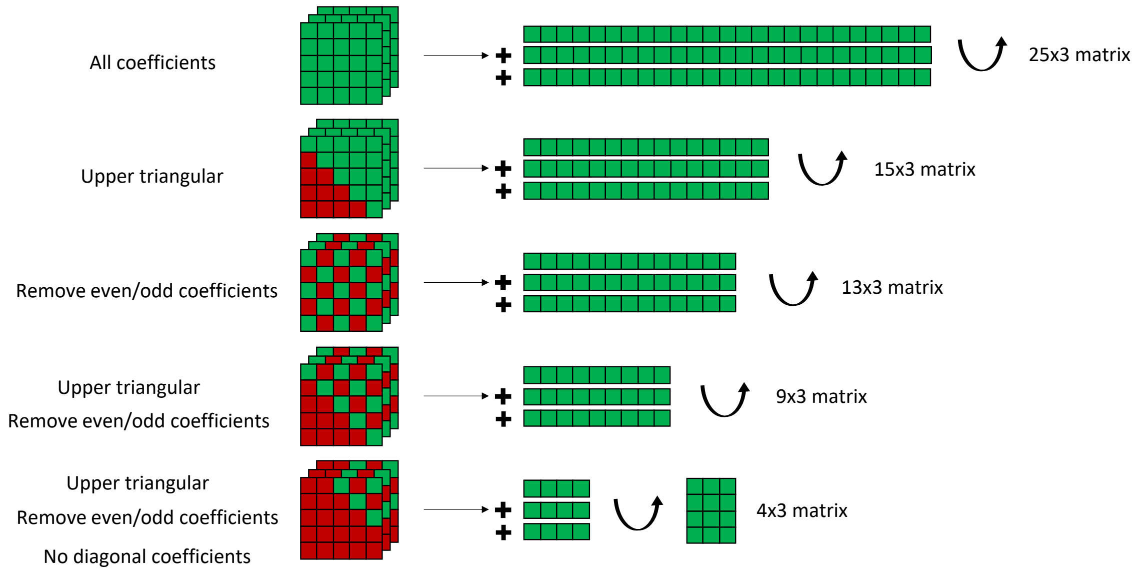


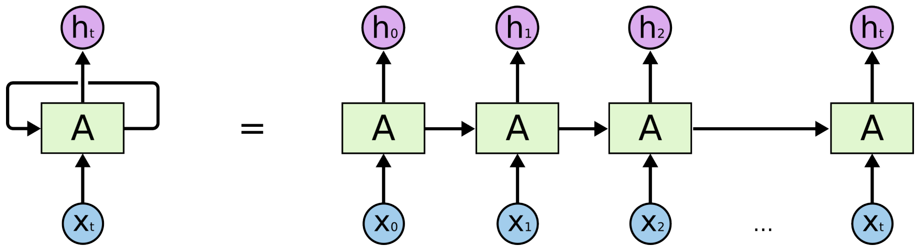
Figure 11: Diffferent possibilities for creating input tensor based on chromatic derivative features

As we can see in figure 11, we explored different possibilities of inputs for the different neural networks tested. In this example (figure 11), we have $N = 5$ and the sliding window produced 3 different correlation matrices. The input of the CNNs (or other neural networks like RNNs for instance) are the matrices on the right side of figure 11 (25x3, 15x3, 13x3, 9x3 or 4x3 depending on the number of coefficients from the correlation matrices we consider). We should only take the odd/odd or even/even coefficients from the upper triangular matrix excluding the diagonal. However, having more features did not affect the results in practice. Sometimes, it even slightly improved results. The main concern related to feature reduction was for large models that have many pareters. The number of parameters increases significantly as the number of used coefficients increases. For instance, only using relevant coefficient and having a smaller value for N allowed us to train the CRDNN model with this approach (CRDNN is a combination of CNN, RNN and DNN layers).
As an important note, this approach is similar to the one used in [1]. In practice, using

correlation matrices and flattening them has given the best results. Each vector (i.e. flattened correlation matrix only using relevant coefficients) However, this approach does not seem ideal. For instance, the use of CNNs (and therefore CRDNNs) does not seem relevant since we flattened matrices. Moreover, when we use the approach using raw signals, it create an input matrix with N rows whereas the correaltion matrix approach creates a $\frac{N(N-1)}{4}$ rows input matrix. The filter bank approach (STFT) which will be discussed in section 2.5.2 is more similar to the raw signal approach in the sense that the number of rows of the input matrix is more similar (filter bank creates input matrices that have *number_ of_ mel_ filters*=40 rows by default in SpeechBrain).
We will both use correlation matrices using the chromatic decomposition of input signals and raw signals themselves as inputs for our models and compare results with filter banks too.

2.3.2 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are designed for tasks where the model output depends on a sequence of inputs (rather than a single input). It is common to use RNNs in speech recognition.



source : [6]

Figure 12: Unrolled RNN scheme

In figure 12, we can see a RNN neuron A with an input x_t and an output h_t. This neuron is equivalent to the feedforward architecture shown on the right. Simple Recurrent Networks (SRN) are one of the first models which uses a context layer. Inputs are fed into the SRN one at a time and at each timestep, the current activations in the hidden layer are copied to the context layer. The new activation functions for the hidden layer are then computed from the current input layer and the context layer. SRNs can also use shortcut connections (e.g. connections from the input to the output or from the output to the hidden layer). SRNs can handle medium-range dependencies but struggle with long-range dependencies. That is where we introduce Long Short Term Memory (LSTM).

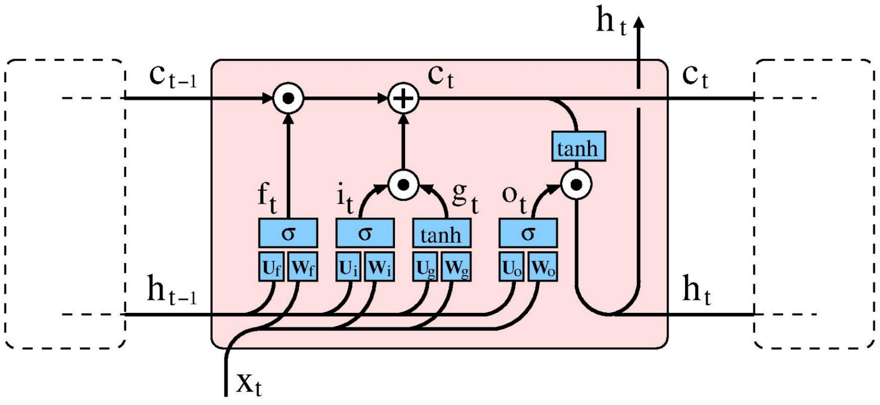


Figure 13: LSTM scheme

A LSTM neuron is shown in figure 13. LSTM also uses a context layer c which is distinct from the hidden layer h. In figure 13, U and W are the weights corresponding to each "gate" (and b is the bias), σ (sigmoid function) and $tanh$ are the activation functions. For each gate, we can write the following equation :

- forget gate f : $f_t = \sigma(U_f h_{t-1} + W_f x_t + b_f)$
- input gate i : $i_t = \sigma(U_i h_{t-1} + W_i x_t + b_i)$
- output gate o : $o_t = \sigma(U_o h_{t-1} + W_o x_t + b_o)$

We also define a candidate activation g also called update values

- $g_t = tanh(U_g h_{t-1} + W_g x_t + b_g)$

The context layer is then equal to

- $c_t = f_t \odot c_{t-1} + i_t \odot g_t$ (with $\odot$ the Hadamard product)

The output is given by

- $h_t = o_t \odot tanh(c_t)$

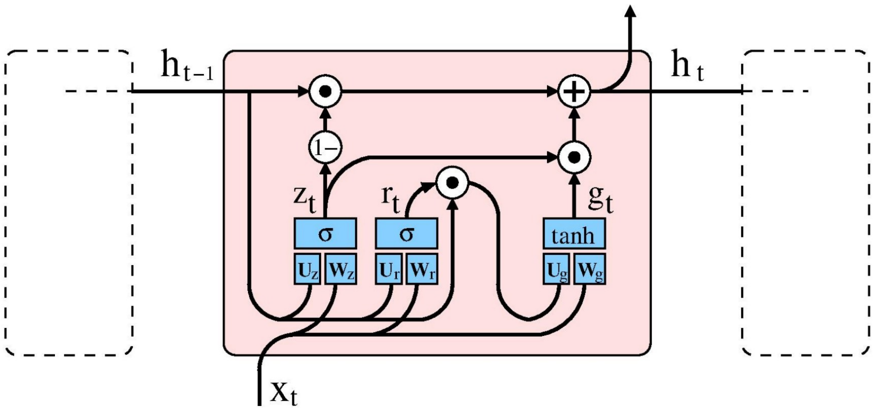


Figure 14: GRU scheme

A Gated Recurrent Unit (GRU) neuron is shown in figure 14. CRU is similar to LSTM but it has 2 gates (instead of 3 for LSTM). The equations are given by

- update gate z : $z_t = \sigma(U_z h_{t-1} + W_z x_t + b_z)$
- reset gate r : $r_t = \sigma(U_r h_{t-1} + W_r x_t + b_r)$
- candidate activation g : $g_t = tanh(U_g(r_t \odot h_{t-1}) + W_g x_t + b_g)$
- output h : $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot g_t$

2.3.3 Connectionist Temporal Classification

As explained in [7] and [8] (original publication), Connectionist Temporal Classification (CTC) is a technique used for mapping a sequence of observations $I = [I_1, I_2 ... I_T]$ (i.e. input) to a sequence of labels of different length $P = [P_1, P_2 ... P_U]$ (i.e. output). Both I and P can vary in length. In our problem, the input I could be a sampled audio signal (we can use the raw signal with a sampling frequency $f_s = 16kHz$) and P is the sequences of phonemes we want to predict. The CTC adds a special token (i.e. phoneme) called “blank”. The CTC algorithm allows us to determine an output distribution over all possible P. Based on this distribution, we can choose the most likely output. The CTC loss function measures the discrepancy between the network's predictions and the ground truth labels. The loss function sums the probability of all possible alignments between the input sequence I and the target sequence P. It provides a way to train the network without requiring precise alignment information. To define the CTC loss, we consider the following conditional probability (i.e. likelihood)

$$p(P|I) = \sum_{A \in \mathcal{A}_{P,I}} \prod_{t=1}^{T} p(a_t|I)$$

with $\mathcal{A}_{P,I}$ the set of all valid alignments and $A = [a_1, a_2 ... a_T]$ a valid alignment in $\mathcal{A}_{P,I}$. The probability $p(a_t|I)$ can be given to us using a RNN. We want to maximize the likelihood so we can define the CTC loss as the negative log-likelihood on our training set $\mathcal{D}$

$$L_{CTC} = -\sum_{P,I \in \mathcal{D}} ln(p(P|I))$$

For a given input, we are trying to find the output sequence P^* (estimate) that maximizes the likelihood

$$P^* = \underset{P}{argmax}\ p(P|I)$$

One trivial approach is to take the most likely output at each time step which gives us the alignment A^* corresponding to the highest probability

$$A^* = \underset{A}{argmax} \prod_{t=1}^{T} p(a_t|I)$$

One assumption of CTC is that every output is conditionally independent of the other outputs given the input. In our case lets us consider the set of 61 phonemes used in the original TIMIT dataset. The phoneme **bcl** (i.e dcl) is always followed with **b** (i.e. **d**). The 2 highest probabilities for the first phoneme prediction correspond to **bcl** and **dcl**. If the first phoneme predicted is **bcl**, then the second phoneme should be **b** with high probability and **d** with a low probability. Similarly, if **dcl** is predicted for the first phoneme, we would like the model to predict **d** with a high probability for the second letter and **b** with a small probability. With this example, we can see that the strong assumption of conditional independence may not be always relevant. CTC algorithm share similarities with Hidden Markov Models (HMM). For instance, computing the sum of probabilities over all alignments can be done efficiently using the forward algorithm also used in HMMs. A major difference between CTC and HMM is that CTC is a discriminative model (i.e. it models $p(P|I)$ directly) whereas HMM is a generative model which uses the Bayes Rule to simplify the problem : $p(P|I) \propto p(I|P)p(P)$. The probability $p(P)$ and the posterior $p(I|P)$ can be determined using a language model and our training dataset. Historically, combinations of neural networks and HMM have been tested on the TIMIT dataset as in [9] but CTC adds improvements to the HMM model.

2.4 State-of-the-art

Model	PER	date	article
wav2vec	8.3	2020	[10]
Li-GRU	14.2	2018	[11][12]
LSTM	14.5	2018	[11]
Bi-RNN + Attention	17.6	2015	[13]

Table 2: TIMIT benchmark

State-of-he-art resuts are shown in table 2 where the test PER, the type of model used and the related publication and date are indicated. A breakthrough has been made in 2020 with the use of models that were pre-trained on a larger set of data. The wav2vec model is a transformer model that can be found on HuggingFace (https://huggingface.co/speechbrain). This model outperforms all the previous models by a large margin. The input given to this particular model is the raw signal, no pre-processing is required. The aim of this project is to study the use of Chromatic Derivatives. Therefore, we will prefer working with RNNs as feature engineering is more relevant in this case.

ANN	MFCC	CD 25ms	CD 50ms	CD 100ms	CD 200ms
MLP	18.2%	31.5%	33.3%	39.4%	52.6%
CNN	18.9%	31.2%	32.1%	39.7%	53%
LiGRU	15.4%	17.7%	18%	22.5%	32.4%
LSTM	15.1%	16.8%	17.8%	22.7%	31.6%

Table 3: PER obtained in [1] with Chromatic Derivatives used for pre-processing on the TIMIT dataset

The results shown in table 3 were obtained in a previous project done in 2022 [1]. The Pytorch-Kaldi library [11] was used to get those results. Pytorch-Kaldi is the previous library made by the authors of the SpeechBrain library [2]. The results shown in table 3 are obtained using MFCC or CD filters. For the CD filters, correlation matrices are computed based on a chosen window of length varying from 25 ms to 200ms. The hop chosen for the window is also indicated in [1] (e.g. 10ms hop for a window of length 25ms). In those results, we can see that CDs used with correlation matrices seem to give the best results when using RNN (LSTM and GRU) compared to CNN or MLP. The results are better when we use a smaller window (which increases the length of the input and therefore the computational time for the model training). For a window of 25ms, the results we obtained are relatively close to the ones obtained using Mel-Frequency Cepstral Coefficients (MFCC) which is one of the best pre-processing technique for this problem.

2.5 SpeechBrain

2.5.1 Description

SpeechBrain [2] is a python library for speech recognition, speaker recognition, speech enhancement, speech separation, language identification or multi-microphone signal processing among other topics. As mentioned earlier, SpeechBrain is a newer version Pytorch-Kaldi [11] which was used in a previous related project [1]. We will the use SpeechBrain for speech recognition on the TIMIT dataset. One of the objectives of this project is to Modify the SpeechBrain library to implement Chromatic Derivatives as an option for pre-processing. Then, we want to compare the results obtained using CDs (with and without using correlation matrices) compared to other pre-processing methods (MFCC, raw signal, filter banks with STFT...). SpeechBrain includes some “recipe” models where some configurations are already defined specifically for the TIMIT dataset.

Model	Test PER
CTC, CRDNN (with LiGRU), STFT filter banks	14.78
seq2seq with CRDNN (with LiGRU), STFT filter banks	14.07
seq2seq with wav2vec + CTC/attention	8.04
Multi-teacher Knowledge Distillation, CTC/attention, STFT filter banks	13.11
Encoder (CRDNN) - Decoder (LiGRU + BS), transducer loss, STFT filter banks	14.12
Encoder (wav2vec) -Decoder (LiGRU + BS), transducer loss	8.91

Table 4: Results obtained in [2] with the “recipe” configuration in SpeechBrain on the TIMIT dataset

The results obtained with the “recipe” configurations from SpeechBrain are shown in table 4. A CRDNN model is used several times. This model is a combination of CNN, RNN and fully-connected networks. The RNN can either be a LiCRU, GRU, LSTM or vanilla RNN. Layer and batch normalization are used for feedforward layers. Time-pooling can be optionally used for downsampling. More precisely all the CRDNN models shown in table 4 have the following configuration : 2 CNNs, 4 LiGRUs and 2 DNN layers.

The Github repository of this project can be found on https://github.com/LouisMlt/COMP9991-2UNSW2023/

2.5.2 Filter banks

The goal of this project is to evaluate the performance of models using chromatic derivatives as a preprocessing method. To do so, we need to compare results with a preprocessing baseline. We chose to use filter banks with STFT as out baseline. In fact, filter banks are a state-of-the-art preprocessing method for RNNs. For a given audio signal, we can compute its spectrogram. We then use triangular filters to define the filter banks features. Filter banks are time-frequency representations of the input signal.

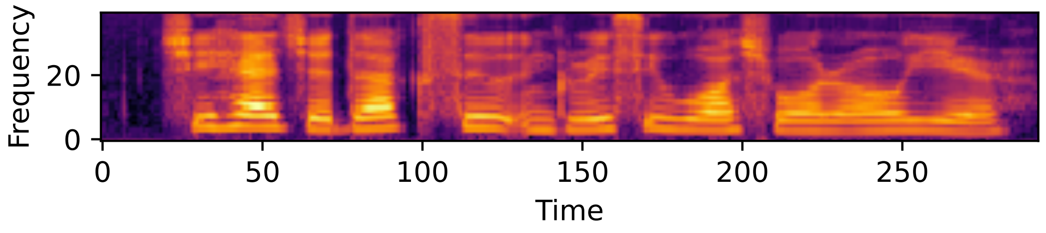


Figure 15: Filter bank representation of an audio signal

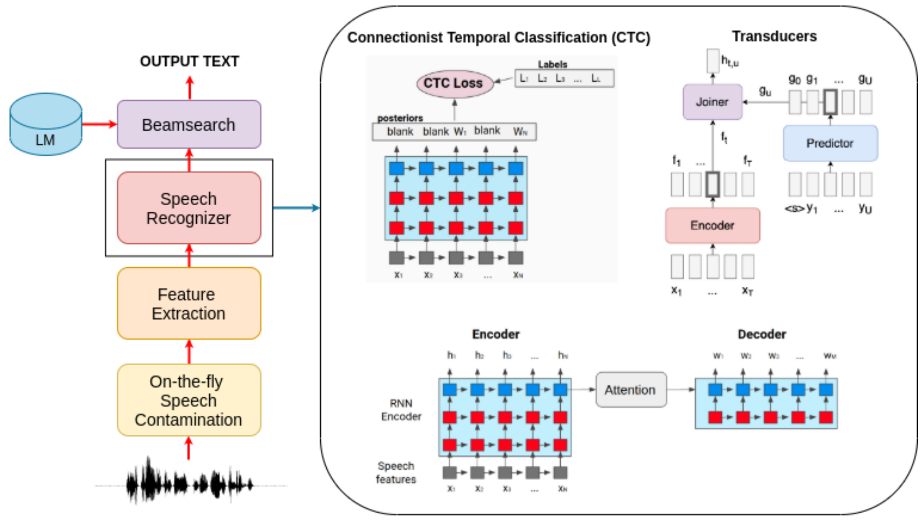
We can see in figure 15 an signal from the TIMIT dataset (*SA1* sentence). The time axis corresponds to each window for the STFT (window is 25 ms and the hop is 10 ms). The frequency axis corresponds to each filter which was computed for the filter bank (40 filters). The filters are designed to be equally-spaced in the mel-frequency domain. Therefore, we have more filters in the lower part of the spectrogram (low frequencies) and less in the higher part (high frequencies). To pass from the frequency domain to the mel domain we can use the following transformation

$$m \approx 2595 log_{10}(1 + \frac{f}{700})$$

with m the mel frequency and f the standard frequency (in Hz). It is important to note that filter banks contain temporal and frequential information about the signal. it is not the case for the chromatic derivative features which have a temporal dimensiona and the dimension corresponding to the chromatic derivative coefficients. We will compare models designed for filter banks by applying them on chromatic derivative features. However, CNNs do not seem appropriate for chromatic derivatives since 2D convolutions will not necessarly make sense (along the "chromatic coefficients" axis).

2.6 Architectures

Different architectures have been tested and are shown in figure 16. The classical CTC architecture includes only one component (e.g. LSTM, GRU or CRDNN) and involves the CTC loss which was discussed in section 2.3.3. The two other architectures are the Encoder-Decoder and the Transducer architectures.



source : SpeechBrain tutorial "Speech Recognition From Scratch"

Figure 16: Different architectures in SpeechBrain for the speech recognizer : CTC, Encoder-Decoder and Transducers

2.6.1 Encoder-Decoder

The Encoder-Decoder architecture consists of two main components:

- The encoder processes the input signal and produces a fixed-length representation (i.e context vector or acoustic features). This representation encodes the essential information from the input signal and is used as the initial state for the decoder.
- The decoder then generates the output sequence of phonemes based on this context vector.

The encoder we used in Speechbrain is the CRDNN model which is a combination on CNN, RNN (LSTMor GRU) and DNN layers. Another option is to use a Transformer which involves attention mechnaisms (e.g. Wav2Vec [14]). The Encoder-Decoder uses the CTC loss.

2.6.2 Transducer

The Transducer architecture is an extension of the CTC approach, enhancing it with the addition of an autoregressive predictor and a join network. This architecture is widely used in speech recognition tasks and has shown improved performance over traditional CTC-based models. The 3 main components of the transducer architecture are :

- The encoder plays a similar role than in for the Encoder-Decoder architecture as described in section 2.6.1.
- The predictor generates a latent representation based on the previously emitted outputs, effectively modeling the context of the previously generated tokens. By incorporating this autoregressive component, the Transducer can better capture long-range dependencies in the output sequence.
- The join network is responsible for combining the information from the encoder and the predictor. It fuses the encoded representations from the encoder with the context from the predictor, creating a joint representation that contains both the input information and the previous output context. The joint representation is then fed into a softmax classifier, which predicts the current phoneme (i.e. token). The softmax classifier computes the probability distribution over the output vocabulary, and the token with the highest probability is selected as the model's prediction for the current time step. During training, CTC loss is used after the classifier.

3 Achievements

The first test was to compare the becnhmark results given in the SpeechBrain documentation for the “recipe” configurations. Those results can vary base on the random weight initialization of the models.

Model	Our test PER	PER from [2]
CTC with CRDNN (with LiGRU), STFT filter banks	14.54	14.78
Encoder (CRDNN with LiGRU) Decoder, STFT filter banks	14.22	14.07
Transducer, STFT filter banks	14.15	14.12
Encoder (Transformer: wav2vec) Decoder + CTC/attention	8.8	8.04

Table 5: Our results compared to the benchmark results from SpeechBrain for 50 epochs on the TIMIT dataset[2]

We can see in table 5 that our results are relatively close to the ones shown in [2] (which are also presented in table 4). We tested the transducer model with a CRDNN as the encoder. It is also possible to use the transducer architecture with a Transformer model for the decoder (a PER of 8.91 was achieved by [2]).
We now compare those results obtained with STFT filter banks (40 filters) with CDs filters. It is important to notice that hyperparameters were optimized for the use of STFT filter bank in pre-processing. Tuning those hyperparameters (e.g. optimizer, learning rate, weight decay ...) when using chromatic derivatives is relatively difficult since it takes a large amount of time to train models. The first important step we made is to implement chromatic derivatives within the pre-processing functions from SpeechBrain. For each input audio signal, we convolve it with the first 48 CD filters (order 1 to 48). The first CD filters were shown in figure 4. This gives us a matrix with 48 rows (and the same number of columns as the input signal) which is the new input of our CRDNN model. This approach is the one using “raw signals” as described in section 2.3.1. We used the *Conv1d* function from pytorch (*torch.nn.Conv1d*) to compute this matrix. One unresolved issue was noticed when specifying the stride parameter s of *Conv1d*. We would like this parameter to be relatively small (1 being the minimum). However, we get an error message when choosing $s = 64$ or less.

Model	pre-processing	test PER
CTC with CRDNN (with LiGRU)	STFT filter banks (40)	14.54
CTC with CRDNN (with LiGRU)	CDs filters (48), stride=96	27.23
CTC with CRDNN (with LiGRU)	CDs filters (48), stride=128	27.03

Table 6: Results with the CTC model with CD filters

As shown in table 6, the results obtained with this method are correct but still far from the ones obtained with STFT filter banks. The best results were obtained using the parameters from the SpeechBrain recipe configuration for the CRDNN :

- *cnn_blocks: 2*
- *cnn_channels: (128, 256)*
- *cnn_kernelsize: (3, 3)*
- *rnn_layers: 4*
- *rnn_neurons: 512*
- *rnn_bidirectional: True*
- *dnn_blocks: 2*
- *dnn_neurons: 512*
- *dropout: 0.15*
- *activation: LeakyReLU*
- *time_pooling : True*
- *optimizer : Adadelta* ($\rho = 0.95$, *learning rate* $\lambda = 1$, $\epsilon = 10^{-8}$)

As we can see, the results obtained with the raw signal approach are not satisfactory compared to the filter banks approach (STFT). An alternative approach was to use correlation matrices after applying the CD filters similarly to what was done in the project [1] and explained in section 2.3.1.

Model	window (ms)	hop (ms)	Library	ref	test PER
GRU, CTC architecture	10	10	SpeechBrain	-	23.12
LSTM	25	10	Pytorch-Kaldi	[1]	16.8

Table 7: Comparison of the results obtained with CD filters and correlation matrices between this work and [1]

The best results for this project were achieved with the correlation matrix approach. The best PER obtained is 23.12 as presented in table 7. A more detailed table of results achieved during this project is shown in table 8.

Index	Model	window (ms)	hop (ms)	Coefficients **(*)**	test PER
1	GRU, CTC architecture	10	10	**A**	23.12
2	CRDNN, CTC architecture	50	25	**B**	24.26
3	LSTM, CTC architecture	25	10	**A**	25.26
4	Encoder (LiGRU) Decoder	50	25	**A**	33.92
5	GRU, CTC architecture	5	3	**A**	76.91

Table 8: Results obtained in this research project using correlation matrices. Window size and hop are in ms. **(*)** For the "Coefficients" column, we assume i is the index for the row and j for the column of a given correlation matrix : **A** means we used all the coefficients from the upper triangular matrix, **B** means we used only coefficients from **A** when i and j have the same parity[1]

As we can see in table 8,results are superior to those obtained with the raw signal approach."However, these results do not match the performance of the filter bank approach or the implementation from [1] using Pytorch-Kaldi.Various implementations for each model have been developed, and these will be discussed next. It is important to point out that at the end of term 1 , the PER obtained for the correlation matrix approach was greater than 70%. There has been significant improvements during term 2.

- Model 1 (i.e. index 1) is the top-performing model trained in this project. It uses the CTC architecture

- Model 2 is a CRDNN model. We first thought that it was not possible to achieve correct results with a CRDNN approach and chromatic derivative based correlation matrices. However, the PER achieved with CRDNN (RNN is a Li-GRU) is relatively similar to the one achieved with GRU (index 1). It was impossible to run the CRDNN model with a window smaller than 50ms because the machine rapidly ran out of memory. That is why the correlation coefficients used (**B** in the table) are limited to only relevant ones to reduce the number of model parameters. We also used an idea similar to logarithmic compression for this approach. We used the diagonal coefficients of the correlation matrix. As explained in section 2.3.1, the correlation matrix coefficients are computed based on the covariance matrix by

$$R_{ij}^M = \frac{C_{ij}^M}{\sqrt{C_{ii}^M C_{jj}^M}}$$

  Rescaling those coefficients was done with

$$\check{R_{ij}^M} = \frac{C_{ij}^M}{\sqrt{C_{ii}^M C_{jj}^M}} log\left(1 + \sqrt{C_{ii}^M C_{jj}^M}\right)$$

- Model 3 is similar to model 1 with a LSTM instead of GRU. As we can see GRU seems to perform slightly better.
- Model 4 uses the encoder decoder architecture. As this architecture is better than the CTC architecture when using filter banks, I was expecting better results for chromatic derivatives too. However, it is not the case. Interestingly, when we look at the phonemes predicted by the model, the length of the sequence of predicted phonemes is too long compared to the true sequence. This phenomenon is even more important when the window size is smaller. I found that a 50ms window was giving me the best results for this architecture.
- Model 5 is the same as model 1 but with smaller window sizes. As we can see the results are not good. The length of the output sequence seems to be related with the size of the hop. If the hop if smaller, we compute more correlation matrix for each signal which implies that the length of the input is larger.

More detailed results can be found on GitHub in *results.ods*.

4 Conclusion and future work to be done

In conclusion, our exploration of using chromatic derivatives (CDs) instead of filter banks with STFT has revealed numerous challenges. Our results closely matched established benchmarks from SpeechBrain, but attempts to replace STFT with CD based approaches showed limitations. This study highlights the potential of the correlation matrices approach while acknowledging the gap between CDs and traditional filter banks. Future work might refine CD preprocessing and tackle computational hurdles. One idea that was not yet tried would be to first extract features from the correlation matrices using CNN before the input is fed to a deep neural network (e.g. CRDNN, GRU...). Another option would be to combine chromatic derivative features with filter bank features. As we saw in section §3, the state-of-the-model for speech recognition are Transformer based models. This is the case for the TIMIT dataset but also other dataset such as the LibriSpeech or CommonVoice datasets. The combination of chromatic derivatives with Transformers could also be studied in a future project. To conclude, even though chromatic derivatives come with complexities, our research opens doors for exploring ways to improve speech processing techniques for speech recognition.

5 Glossary

- *ASR : Automatic Speech Recognition*
- *BS : Beam Search*
- *CD : Chromatic Derivatives*
- *CNN : Convolutional Neural Network*
- *CRDNN : combination of CNN, RNN and fully-connected networks*
- *CTC : Connectionist Temporal Classification*
- *DNN : Deep Neural Network*
- *GPU : Graphics Processing Unit*
- *GRU : Gated Recurrent Unit*
- *LiGRU : Light Gated Recurrent Unit*
- *HMM : Hidden Markov Model*
- *LSTM : Long Short-Term Memory*
- *MFCC : Mel-Frequency Cepstral Coefficients*
- *MLP : MultiLayer Perceptron*
- *PER : Phoneme Error Rate*
- *RNN : Recurrent Neural Network*
- *SRN : Simple Recurrent Network*
- *STFT : Short Term Fourier Transform*

References

[1] Tina Tianle Zhang. Chromatic derivatives for speech recognition, thesis submitted as a requirement for the degree of bachelor of engineering in software engineering at unsw, 2022.

[2] Mirco Ravanelli, Titouan Parcollet, Peter Plantinga, Aku Rouhe, Samuele Cornell, Loren Lugosch, Cem Subakan, Nauman Dawalatabad, Abdelwahab Heba, Jianyuan Zhong, Ju-Chieh Chou, Sung-Lin Yeh, Szu-Wei Fu, Chien-Feng Liao, Elena Rastorgueva, FranÃ§ois Grondin, William Aris, Hwidong Na, Yan Gao, Renato De Mori, and Yoshua Bengio. Speechbrain: A general-purpose speech toolkit, 2021.

[3] Aleksandar Ignjatovic. Chromatic derivatives, chromatic expansions and associated spaces, 2009.

[4] Aleksandar Ignjatovic. Numerical differentiation and signal processing. In *Proc. International Conference on Information, Communications and Signal Processing (ICICS), Singapore*, 2001.

[5] Aleksandar Ignjatovic. Chromatic derivatives and local approximations. *IEEE Transactions on Signal Processing*, 57(8):2998–3007, 2009.

[6] Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[7] Sequence modeling with ctc. https://distill.pub/2017/ctc/.

[8] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376, 2006.

[9] Yoshua Bengio, Renato De Mori, Giovanni Flammia, and Ralf Kompe. Global optimization of a neural network-hidden markov model hybrid. *IEEE transactions on Neural Networks*, 3(2):252–259, 1992.

[10] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460, 2020.

[11] Mirco Ravanelli, Titouan Parcollet, and Yoshua Bengio. The pytorch-kaldi speech recognition toolkit. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6465–6469. IEEE, 2019.

[12] Mirco Ravanelli, Philemon Brakel, Maurizio Omologo, and Yoshua Bengio. Light gated recurrent units for speech recognition. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(2):92–102, 2018.

[13] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. *Advances in neural information processing systems*, 28, 2015.

[14] Qiantong Xu, Alexei Baevski, and Michael Auli. Simple and effective zero-shot cross-lingual phoneme recognition, 2021.