UCLouvain

epl

École polytechnique de Louvain

Comparison between convolutional and recurrent neural networks for keyword recognition applications

Author: **Laura Vandooren**
Supervisors: **David Bol, Christophe De Vleeschouwer**
Readers: **Martin Lefebvre, Laurent Jacques**
Academic year 2019–2020
Master [120] in Electrical Engineering

Abstract

The keyword spotting system (KWS) represents a major part of human-technology interfaces. It requires low-latency and high accuracy response for good user experience. Moreover, KWS typically runs on small micro-controllers and have thus, limited memory and compute capability. To meet these constraints, recurrent neural networks, such as LSTM and Grid LSTM models, as well as a convolutional neural network are trained to determine whether a one-second audio clip contains one of the four predefined words or an unknown word. In that way, we are able to determine the model with the best memory, accuracy and computational complexity trade-off. It comes out that the CNN reaches the best accuracy nearly equal to 93% with 370 millions of operations and only, 500 thousands parameters. While, for resource-constrained problem, the 1-layer LSTM is optimal using the smallest memory space and number of operations but achieves only 86.6% of accuracy. We have also studied the effect of the signal pre-processing on the network performance and we have found that using an optimal window size of 25 ms for the spectrogram computation allows to rise the accuracy by 3%. Finally, the LSTM network is quantized to reduce the model memory size and its computational complexity. The weights and biases can be reduced to 4-bits fixed point numbers if a 3% accuracy drop is tolerated in comparison to the floating point implementation.

Acknowledgements

I would like to thank my two supervisors, David Bol and Christophe De Vleeschouwer, for their availability and encouragement throughout the year. During our regular meetings, they always kept a critical look on my work enabling me to develop this master thesis. They also allowed me to connect to a UCLouvain server, named Cassiopee, to do my simulations.

I am particularly grateful for the assistance given by Martin Lefebvre and Charlotte Frenkel, who always answered very kindly, precisely and rapidly to all my questions. Thanks to their expertise in the field of neural networks, they help me to progress through the different steps of my research with their valuable feedback and advises.

Contents

1 Introduction **3**

2 Background **6**
2.1 Signal processing chain for keyword spotting applications 6
2.2 Feature processing 7
2.2.1 Spectrogram 7
2.2.2 Mel-spectrogram 11
2.3 Convolutional neural networks 12
2.3.1 Fully-connected layers 13
2.3.2 Convolutional layers 14
2.3.3 Maxpooling layers 15
2.4 Long Short-Term Memory 15
2.4.1 Architecture of recurrent neural networks 16
2.4.2 Vanishing gradient problem 18
2.4.3 Long Short-Term Memory cell 19
2.4.4 Improvements to LSTM networks 21
2.5 Training 24
2.5.1 Loss function 24
2.5.2 Parameters update 24
2.5.3 Parameter initialization 27

3 Keyword recognition system **28**
3.1 Dataset 28
3.2 Features processing 29
3.3 Architecture of the neural networks 30
3.3.1 Convolutional neural network 30
3.3.2 Long Short-Term Memory 32
3.3.3 Grid Long Short-Term Memory 34
3.4 Training and weighted cross-entropy loss 36
3.5 Performance computation 38
3.6 Estimation of the computational complexity of the models 39

4 Experimental results **41**
4.1 Study of the LSTM architecture: size of the hidden layer and depth of the network 42
4.2 Comparison between Basic LSTM, Grid LSTM and CNN 44
4.3 Study of the spectrogram shape 47
4.4 Mel-spectrogram feature 52
4.5 Quantization of a LSTM network 53

5 Conclusion **56**

A Histogram of the word duration **58**

B Impact of the weighted cross-entropy **60**

Chapter 1

Introduction

In our society, information and communication technologies become part of the daily routine and enable to solve problems such as user identifications, high data-rate wireless communications or non-invasive low-power medical monitoring. In particular, people are using tablets and smartphones in their everyday life and interacting with those technologies using voice is getting more and more important. For example, Google and Apple devices are already equipped with virtual assistants such as Siri or Alexa. These tasks have been enabled by recent developments in machine and deep learning algorithms which outperformed the humans in a variety of tasks including speech recognition [1]. By analyzing the human voice, the system is able to determine what is said and eventually respond to the user. A voice assistant contains a keyword spotting system (KWS) able to detect and recognize specific keywords, wake-up the device and then, activate the full scale speech recognition. In addition to the voice recognition, generally, they simultaneously performs a face recognition. They are always active and thus, should have a very low-power consumption to maximize the battery life. Furthermore, a KWS system should detect keywords with high accuracy for better user experience and low latency for real-time response. Because KWS is used on mobile devices, the memory footprint and the computational capability are therefore limited. These constraints have defined an interesting area of research for 50 years [2].

Two types of neural network are commonly used in speech recognition: convolutional neural networks (CNN) [3, 4] and recurrent neural networks (RNN) [5–7]. However, comparing those networks is often very difficult because they are trained on different datasets (e.g: "TalkType" dataset in [8], "Alexa" dataset in [9] and Google Speech Commands dataset [10]) or use different input features (Mel-frequency cepstral coefficients (MFCC) [11], spectrogram [3], ...). In [6], a comparison between different neural networks on the Google Speech Commands dataset [12] aims to maximize the accuracy with a limited memory footprint and computational complexity. Apart

from the neural network type, the model performance is also impacted by the input features. In [3], the importance of the hyper parameters for the spectogram computation is highlighted and their impact on the overall accuracy of the network is measured. The network quantization could also help reducing memory footprint and the computational complexity while maintaining its accuracy. A post-training quantization was recently performed on a LSTM network in [13] whose objective is to obtain the minimum bitwidth required to avoid any performance loss compared to the floating-point implementation.

In this thesis, we will train several KWS system on the Google speech commands dataset using a spectrogram as input feature. In particular, they will classify an audio signal in one of the 4 predefined word classes or in the unknown word class. Due to the presence of the complex "unknown" class, a weighted cross-entropy must be used to ensure equivalent performance among the class. Throughout this work, several techniques have been investigated to improve the accuracy or reduce the memory footprint and computational complexity of the models. In the first place, we will compare a CNN and long-short term memories (LSTM) networks in terms of accuracy, memory footprint and computational complexity. The LSTM is a variant of the basic RNN solving its problem of learning information in the long-range. To explore further, an improved version of the LSTM network where the time propagation of the long-term dependencies are reinforced by a propagation through the network depth, called Grid LSTM [14], is added to the comparison. It comes out that the CNN network reaches 93% of accuracy with a reduced number of parameters equal to 500 000 and a middle number of operations of 370 MFLOPs whereas for hard-constrained problems, the 1-layer LSTM should be preferred but reaches only 86.6 % of accuracy. Secondly, we have found that an optimization of the window size can allow an improvement of the performance as high as 3%. Additionally, this optimisation was done for both the CNN and the LSTM networks enabling a fair comparison among both models. Finally, the LSTM network is quantized and the resulting accuracy loss was recorded. In this work, we show that the weights can be quantized up to 4 bits using a post-training quantization method while maintaining the accuracy drop below 3%. The CNN quantification was not performed as this subject has already been widely studied in the literature [15].

To achieve these objectives, my work will be structured as follows. The chapter 2 will present the two main parts of a KWS known as the feature extractor and the neural networks. Both the convolutional and recurrent neural networks will be explained along with the applications of such networks. The chapter will end with some reminders about training algorithms. Then, the chapter 3 explains the design of the KWS implemented for this work including the dataset, the input features,

the architecture of the neural networks, the weighted cross-entropy loss and metrics for the performance and complexity evaluation. Finally, the chapter 4 shows the experimental results previously announced such as the network comparison in terms of accuracy, memory footprint and computational complexity, the study of input features for both CNN and LSTM networks and finally, the LSTM network quantization.

Chapter 2

Background

Chapter 2 explains the main concepts necessary to discuss a keyword spotting system. The overall signal processing chain is presented and then, the speech features and the classifier using neural networks will be studied more in depth. Once the system architecture is established, the model parameters must be optimized according to the dataset we have at our disposal. The final section of this chapter will describe the gradient descent algorithms for the training of the models.

2.1 Signal processing chain for keyword spotting applications

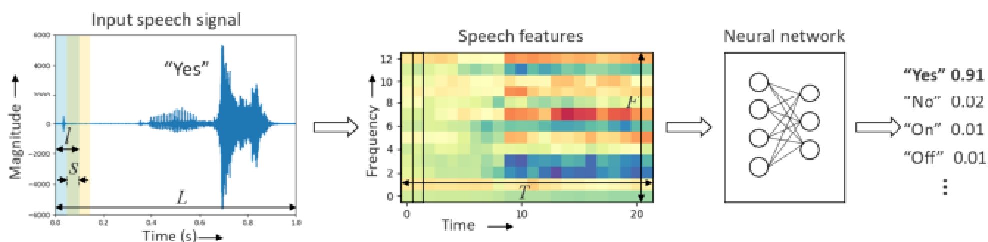


Figure 2.1: Signal processing chain for keyword spotting system [6]

The purpose of a keyword spotting system is to attribute a label to each word utterance. It is made out of a feature extractor and a neural network classifier as shown in Fig 2.1. First, the features of the audio signal are extracted by computing the spectrogram or the mel-frequency cepstral coeffcients (MFCC). Generally speaking, the temporal signal is segmented into T frames and then for each frame, F features are computed. The feature matrix obtained is thus of size F×T. The

spectrogram which is used in [3] is simple to compute but does not mimic the human hearing as well as the MFCC. In contrast, MFCCs are computationally more expensive but are very popular in recent papers [6, 7, 11, 16, 17] because the information is condensed in fewer coefficients.
Then, this feature matrix is fed into the neural network which assigns a label given the probabilities of each word class. Two types of neural networks are commonly used in speech recognition: convolutional neural networks (CNN) and recurrent neural networks (RNN).

2.2 Feature processing

The first step in this signal processing chain is to extract and compress relevant information from the audio recording. In this thesis, we have mainly used the spectrogram as input feature for the neural networks. However, an information extraction similar to the human ear is often used such as the MFCC. As part of this work, we will test the mel-spectrogram which is the first step for the MFCC computation.

2.2.1 Spectrogram

The spectrogram describes the energy content of the signal per frequency band and over time. Figure 2.2 represents a raw temporal signal and its corresponding spectrogram. The vertical axis represents the frequency while the horizontal axis represents the time and the color pattern tells us about the magnitude of the observed frequency for a given frame. The advantage of this representation is that we get the spectral information of the signal while keeping the time localisation. To build the spectrogram, the signal is first broken into frames and then, the Discrete Fourier Transform (DFT) is computed for each frame.

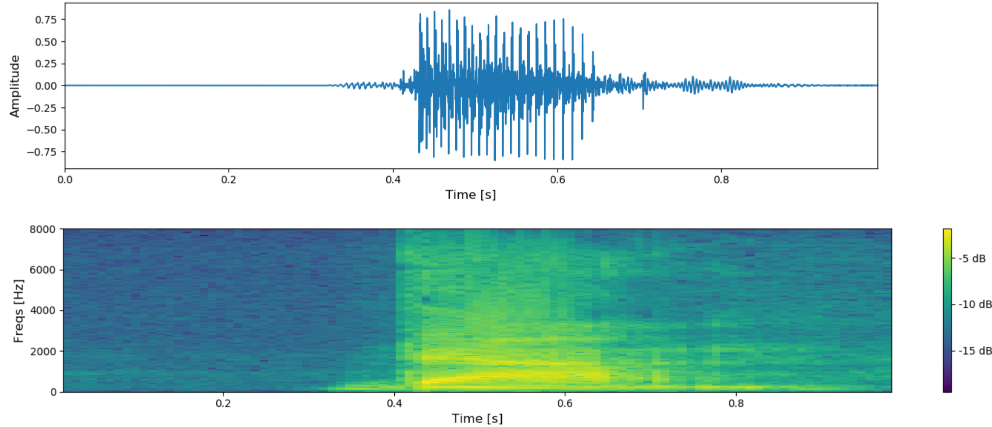


Figure 2.2: Raw temporal signal from the class 'down' and its corresponding spectrogram

Windowing: The signal will be divided into a sequence of frames. For this model to be correct, we assume that the signal is piece-wise stationary (stationary [1] in each frame). The full waveform is multiplied in the time domain by a windowing function. The rectangular window is not often used as it causes discontinuities at the edges and so, introduces broadband noise. Instead, a tapered window is often preferred. Four different window types with their Fourier transform are presented in Fig. 2.3. These windows are generally overlapping to avoid losing information near the edges.

The multiplication by a windowing function in the time domain results in a convolution with its spectrum in the frequency domain. The modifications on the signal spectrum due to the windowing will be smaller if the window spectrum looks like a delta function [18]. If the window spectrum has a too large central peak, the resolution of the windowed signal will be reduced, which means that 2 neighbouring points will be confused easily. A Gibbs phenomenon can also appear when the side lobes are too important, creating energy peaks which were not present in the original signal. By looking at the spectrum of the rectangular window, we can observe that the peak is narrow but the first side lobe is only 13 dB below the main lobe. On the contrary, the Hamming window has side lobes with lower energy but a larger central peak. The choice of the window type is a trade-off between the width of the central peak, the energy of the side lobes and their slope of decreasing energy.

[1]Mathematically, a stationary process is a process whose statistical properties such as the mean, variance and auto-correlation remain constant overtime

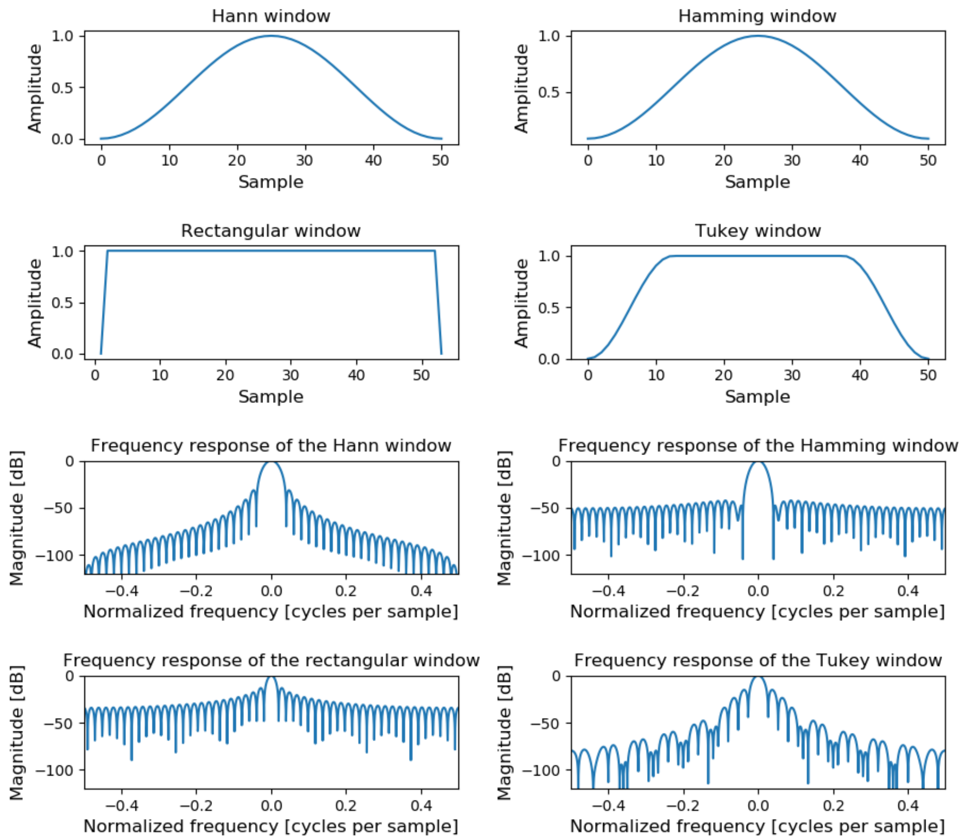


Figure 2.3: Different window types: Hann and Hamming spectrums have both a larger central peak and lower side lobes whereas the Tukey and rectangular spectrums have a narrow central peak but high energy side lobes. Tukey and Hann spectrums have high energy primary side lobes but decreasing rapidly while the Hamming and Rectangular spectrums have low energy primary side lobes but not decreasing very much.

Discrete Fourier transform (DFT): Once the signal has been windowed, the DFT is applied to each frame as described in Fig. 2.4 to extract information in the frequency domain. The DFT of a temporal signal x[n] is computed using:

$$X[k] = \sum_{n=1}^{N} x[n] \mathrm{e}^{-\frac{2\pi j}{N}} \tag{2.1}$$

where X[k] denotes the DFT of the time series x[n] and N represents the number

of samples in each window.

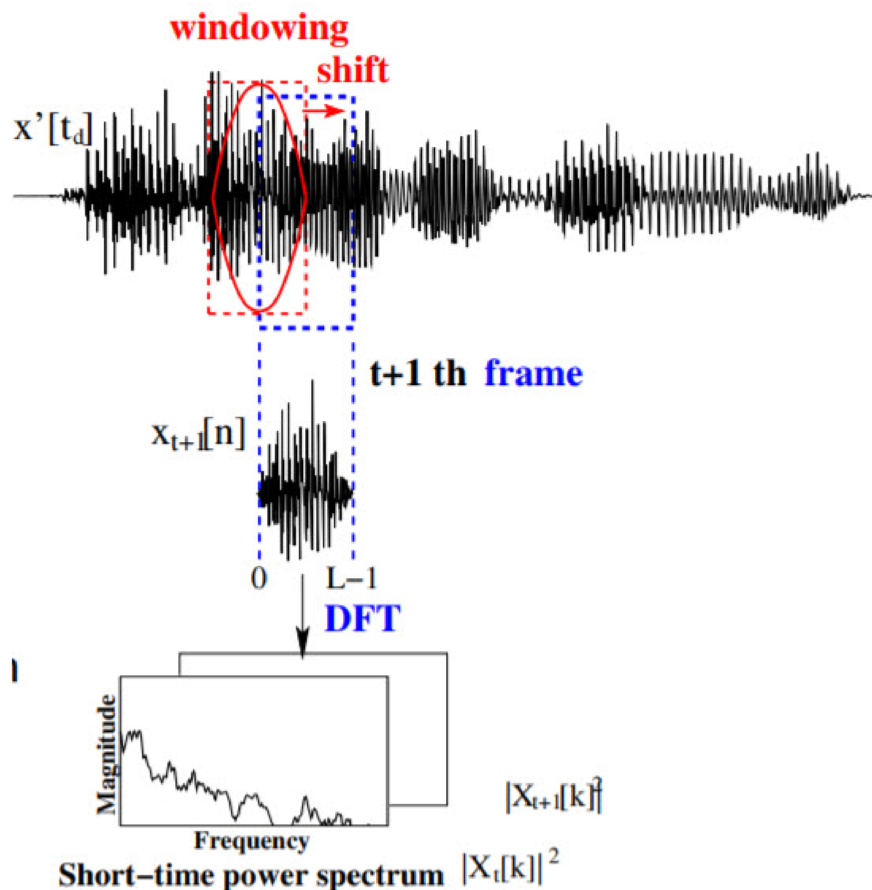


Figure 2.4: Mechanism for computing of a spectrogram [19]: Windowing and DFT computation on each frame

The choice of the window size is governed by a time-frequency trade-off. Indeed, following the Heisenberg uncertainty principle, we cannot know both the time localisation or the frequency of a wave. If we use a short-frame width, we will get a low frequency resolution but a high time resolution and the other way around. In speech recognition task, a usual value for the window size is 25ms with an overlapping of 10 ms [19]. In [20], the window size for the spectrogram computation is studied qualitatively and quantitatively for a speech analysis and an optimal value between 15 and 35 ms comes up.
Finally, to make up the spectrogram, X[k] is not directly used. Instead, it is the

power spectrum of each frame corresponding to the square norm of X[k] which is represented.
When computing a spectrogram, there are many parameters to optimize, among others, the window size, the overlap proportion, the windowing function and the DFT size. In this thesis, we have focused on tuning the window size while keeping the overlapping proportion constant.

2.2.2 Mel-spectrogram

The human auditory system does not perceive the frequency linearly [21]. In 1937, the mel-scale was defined to represent the subjective perceived pitch. It was established using a comparison of sinusoidal tones. A reference frequency is chosen and then, a human listener is asked to adjust the frequency of the other tones to be two times higher or half times lower. Thus, the mel scale is a perceptual scale of pitches judged by listeners to be equal in distance from one another [19]. Wrapping the frequencies following the 'mel' scale as depicted in Fig. 2.5 better mimics the human hearing system.

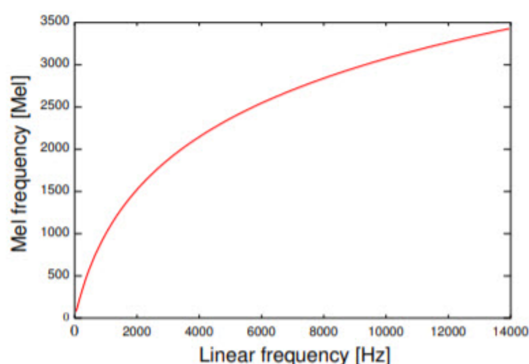


Figure 2.5: Mel scale

To pass from the linear scale to the mel scale, we can use:

$$M(f) = 1127 \ln{(1 + \frac{f}{700})} \tag{2.2}$$

The mel-spectrogram is thus, simply the power spectrum where the frequencies are converted to the mel-scale. An example is presented in Fig. 2.6.

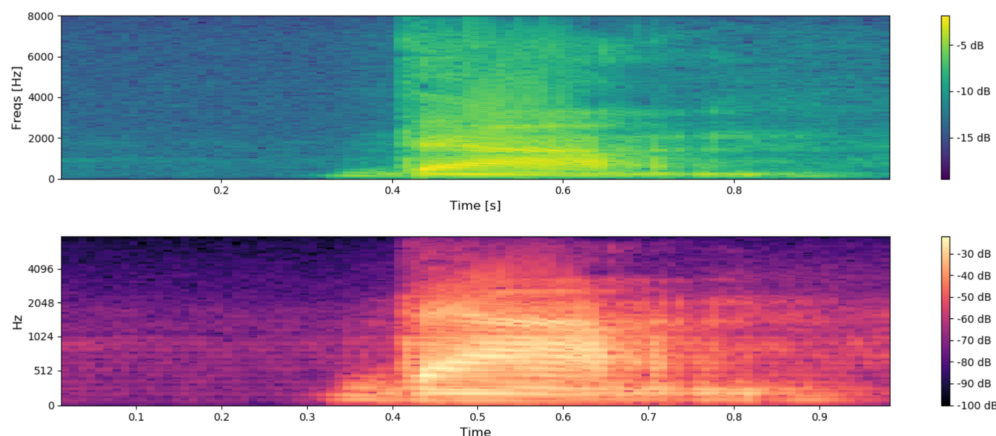


Figure 2.6: Spectrogram and mel-spectrogram corresponding to the signal presented in Fig. 2.2

2.3 Convolutional neural networks

The convolutional neural networks are extensions of the fully-connected neural networks (FC) but applied to image processing. One drawback of the FC is that they do not model the correlations in the input spectrogram. Indeed, to enter a 2D image into a FC network, it is first flattened removing the spatial correlation information. In contrast, the CNN exploits the structure from the input data by treating the input as a 2D image and performs convolution operations over it. An other advantage is the reduced number of parameters of the CNNs in comparison to the FC networks.
The CNNs date back to the the 80's [22–24] but only draw attention in 2012 with a work from Krizhevsky et al. for the ImageNet challenge [25]. Since then, the CNNs were applied in various pattern recognition tasks, including classification of traffic signs [26], house numbers [27, 28] and handwritting digits [29] or pedestrian detection [30], and electron microscopy detection [31]. Later, the CNNs have been successfully applied on speech recognition tasks with the TIMIT database [32–34]. They have shown improvements over conventional FC networks [4, 35]. Two years ago, the TensorFlow Speech Recognition Challenge [36] was organised whereupon several KWS were implemented [3].
A CNN is generally made out of several convolutional layers extracting important features from the image followed by fully-connected layers for classification as depicted in Fig. 2.7. Several types of layer present in a CNN are explained in the following subsections.

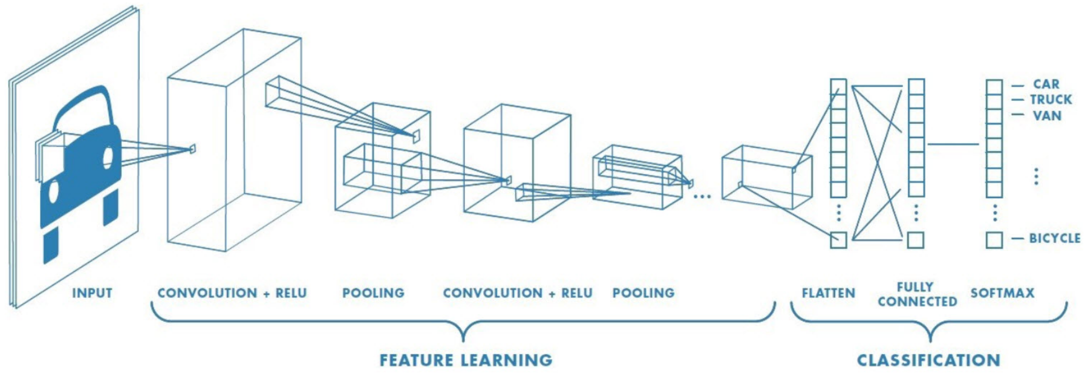


Figure 2.7: CNN architecture for keyword spotting system [37]: Convolutional layers for feature extraction followed by fully-connected layers for classification

2.3.1 Fully-connected layers

The fully-connected layer also called dense layer, is at the same time, the simplest and oldest type of layers. They connect every neuron in one layer to every neuron in the next layer as represented in Fig. 2.8. Inside each neuron, a linear combination of its inputs is computed. Then, a non-linear activation function is applied on it to get the output value. This mechanism is illustrated in Fig. 2.9. In a vector representation, the output value of a dense layer Y is the result of a matrix product of the weight matrix W with the input vector X plus the bias term b on top of which an activation function f is applied:

$$Y = f(W^T X + b) \tag{2.3}$$

where $W_{i,j}$ is the weight applied by neuron Y_i on the neuron X_j of the previous layer. In a FC layer, we can count $n_i \times n_o + n_o$ trainable parameters(n_i and n_o being respectively the number of input neurons and of output neurons).

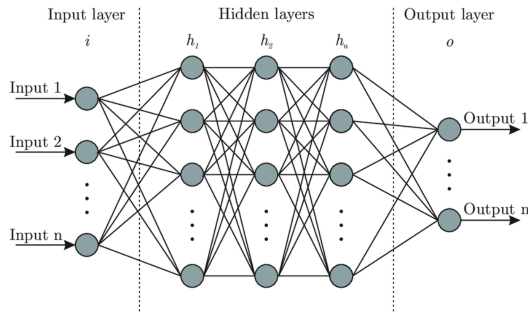


Figure 2.8: Fully-connected layers [38]: all input neurons are connected to all output neurons

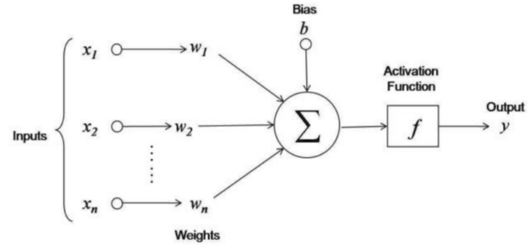


Figure 2.9: Single neuron mechanism [39]: the output value is a linear combination of the input stimuli with a non-linear activation such as the ReLU, hyperbolic tangent or sigmoid function

2.3.2 Convolutional layers

Each convolutional layer is defined by a series of filters and operates by convolving the input image with each filter. Then, a ReLU activation function is applied on each element of the features maps. The ReLU stands for rectifed linear unit and is mathematically defined as $f(x) = \max(0, x)$. The mechanism of a 2D convolution is described in Fig. 2.10. An element-wise product between the filter (K) and the sub-part of the input image (red square) is computed. To browse the whole image, the filter is shifted across the image.

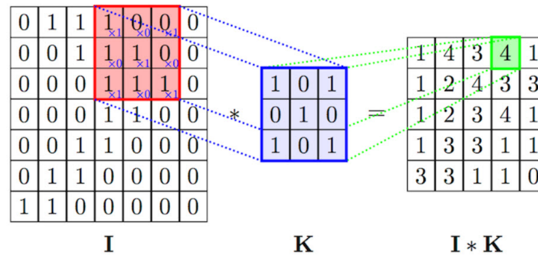


Figure 2.10: 2D convolutions [40]

Mathematically, the jth feature map is obtained by:

$$\text{out}(c_{out,j}) = \text{bias}(c_{out,j}) + \sum_{k=0}^{c_{in}-1} \text{weight}\ (c_{out,j}, k) \star \text{input}(k) \tag{2.4}$$

where $\star$ denotes the 2D cross-correlation. The convolutional layer is determined by 4 parameters:

- **The depth**: the depth represents the number of filters used in the convolutional layer.
- **The stride**: the stride tells by how much the kernel is shifted to compute the convolution of the input image with the kernel. By default, the stride is equal to 1.
- **The padding**: to obtain an output image of the same size as the input image, the input image can be zero-padded, which means that zeros are added on the image edges.
- **The kernel size**: the kernel size defines the size of the convolution kernels, provided as a tuple (k_2, k_1). In the example of Fig. 2.10, the filters are of size (3,3). Generally, kernels are square ($k_1 = k_2$) because input images are also square.

If an input image of depth c_{in} is applied on a convolutional layer with c_{out} filters of size (k_1,k_2), the weights will be of size $c_{in} \times c_{out} \times k_1 \times k_2$. A bias term can also be added which adds c_{out} trainable parameters. The convolutional layers have a much reduced number of trainable parameters in comparison to the fully connected layers but quite a lot of hyperparameters which have to be chosen before training: the depth, the kernel size and the stride for each direction.

2.3.3 Maxpooling layers

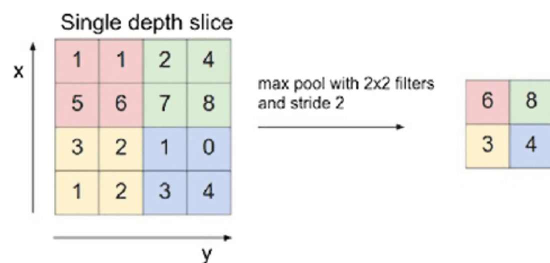


Figure 2.11: Maxpooling mechanism [41]: the maximum value is output from each sub-region

The maxpooling layers perform a downsampling operation enabling a reduction of the number of parameters and reducing overfitting. It divides the input image into a set of non-overlapping rectangles and outputs the maximum of each subregion. Hence, it reduces the size of the input images. In figure 2.11, a filter of size 2x2 with a stride of 2 is used and allows a reduction of the input image of size 4x4 to 2x2.

2.4 Long Short-Term Memory

Unlike fully-connected neural networks or CNN, RNNs contain cyclic connections and process the data sequentially making them suitable to model temporal data

and enable real-time computation. Activations from previous time steps are used as inputs for the current time step. They are stored in the internal state of the network and provide contextual information about the past events. RNNs have been successfully used in many sequence modeling tasks, such as speech recognition [11], language modeling [42, 43] or translation [44, 45]. When they are applied to long sequences, the recurrent neural networks suffer from the same problem as the deep neural networks. During the training, the error gradient is back-propagated through time which cause the gradient to either vanish or explode [46]. In other words, the information from previous time steps rapidly attenuates when it progresses through time. This makes the RNN incapable to store contextual information for a period longer than 5-10 time steps. [47]
Long-short term memory (LSTM) [48] and gated recurrent units (GRU) [45] are two types of recurrent neural network architectures that have been designed to address the vanishing gradient problem of conventional RNNs. Gating functions are added to store information on the long-term. In LSTMs, the gating mechanism controls the access to a memory cell and prevents the network from modifying its content for multiple time steps. In this way, the information and the errors propagate for a much longer time than in traditional RNNs.
In this section, we will also present variants of the standard LSTM model. Bidirectional LSTM networks similar to bidirectional RNNs have been proposed for phonetic labelling of acoustic frames on the TIMIT dataset [49]. They operate on the input sequence in both directions to make a decision for the current input. We will also briefly present the long short-term memory with a projection layer (LSTMP) to reduce the complexity of the network while maintaining its accuracy [11].

2.4.1 Architecture of recurrent neural networks

The RNNs operate sequentially on T time steps where for each time step, we have a feature vector of size F. The RNNs process data as described in Fig. 2.12. The first feature vector $x^{<1>}$ is fed into a neural network layer which outputs a first activation. When processing the second feature vector $x^{<2>}$, the activation value $a^{<1>}$ from time step 1 is also passed on as input for time step 2. And it continues like this until time step T where it takes as input $x^{<T>}$ and the activation value $a^{<T-1>}$ coming from time step T-1 and depending on all the preceding time steps. The activation value is often initialised to 0: $a^{<0>} = 0$. This activation value $a^{<t>}$ is sometimes called hidden layer or state.
As can be seen in Fig. 2.12, in a RNN, weights are reused across all time steps. That is why, in comparison to feedforward networks such FC or CNN, RNN models tend to have fewer parameters.

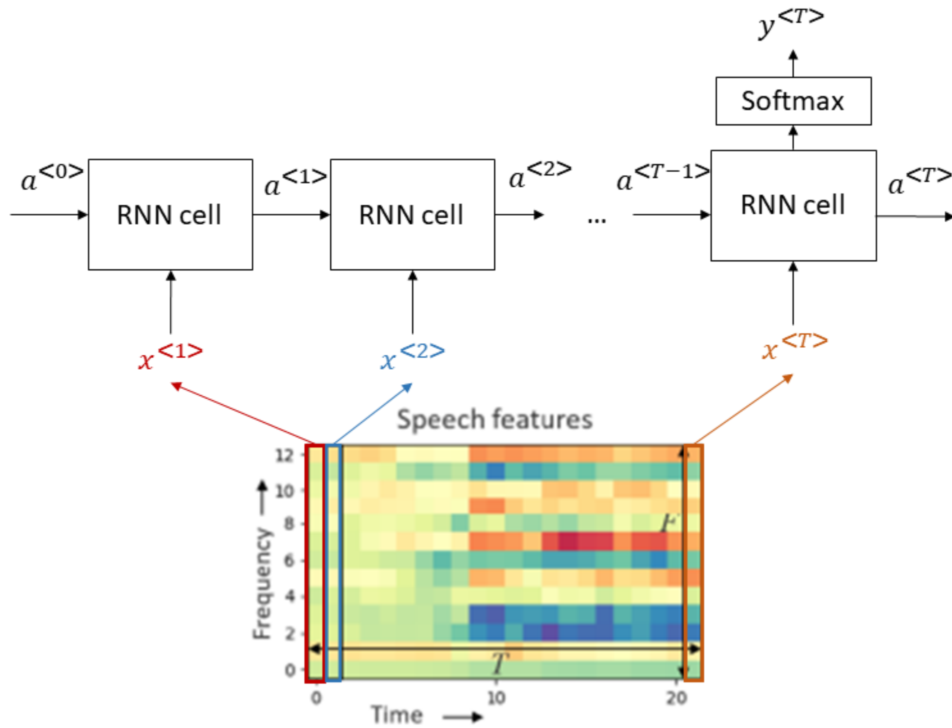


Figure 2.12: The feature vector $x^{<t>}$ of each time step is processed sequentially with an activation value $a^{<t-1>}$ coming from the previous time step and giving contextual information to compute the next activation.

The equation governing the RNN cell is simply:

$$a^{<t>} = \tanh(W_{ax}\, x^{<t>} + W_{aa}\, a^{<t-1>} + b_a) \tag{2.5}$$

where $x^{<t>}$ is the input vector at time t and $a^{<t>}$ is the activation from time t to t+1.

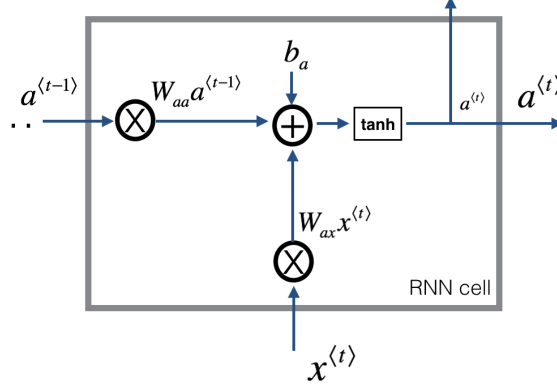


Figure 2.13: Model of the RNN cell: the input value $x^{<t>}$ and the activation $a^{<t-1>}$ coming from the preceding time step are combined [50]

2.4.2 Vanishing gradient problem

Basic RNNs suffer from the vanishing or exploding gradient problem and are unable to capture long term dependencies. During the training, a cost function $\mathcal{J}^{<t>}$ measures the performance of the network on a given task. The vanishing gradient problem comes from the fact that the gradient of the cost function is backpropagated through the neural network to update the weights of each layer, including earlier layers as depicted in Fig. 2.14. However, in the case of a very long sequence or deep network, it can be difficult for the output error to affect earlier layers. Using the chain rule, we will write the gradients expression in order to highlight the gradient exploding or vanishing problem. Consider the gradient of the loss $\mathcal{J}^{<j>}$ on step j with respect to the activation from a previous time step i:

$$\frac{d\mathcal{J}^{<j>}}{da^{<i>}} = \frac{d\mathcal{J}^{<j>}}{da^{<j>}} \prod_{j<t<i} \frac{da^{<t>}}{da^{<t-1>}} \tag{2.6}$$

From 2.5, we can compute the partial derivatives, which can then be inserted into 2.6:

$$\frac{da^{<t>}}{da^{<t-1>}} = \text{diag}\ \left(\tanh'\left(W_{ax}\, x^{<t>} + W_{aa}\, a^{<t-1>} + b_a\right)\right)\, W_{aa} \tag{2.7}$$

$$\frac{d\mathcal{J}^{<j>}}{da^{<i>}} = \frac{d\mathcal{J}^{<j>}}{da^{<j>}} \prod_{j<t<i} \text{diag}\ \left(\tanh'\left(W_{ax}\, x^{<t>} + W_{aa}\, a^{<t-1>} + b_a\right)\right)\, W_{aa} \tag{2.8}$$

By computing the L2 norm:

$$\left\| \frac{d\mathcal{J}^{<j>}}{da^{<i>}} \right\| \leq \left\| \frac{d\mathcal{J}^{<j>}}{da^{<j>}} \right\| \prod_{j<t<i} \left\| \text{diag}\ \left(\tanh'\left(W_{ax}\, x^{<t>} + W_{aa}\, a^{<t-1>} + b_a\right)\right) \right\|\, \|W_{aa}\| \tag{2.9}$$

By relying on singular values, we can say that $\left\|\text{diag}(\tanh'(x))\right\| \leq 1$. It was proven in [47] that if the largest singular value of W_{aa} is smaller than 1, the gradient will shrink towards 0. If it is larger than 1, the gradient will explode. This small development [51] explains why the output $y^{<T>}$ is mainly affected by local values and is only hardly influenced by early inputs in the sequence.

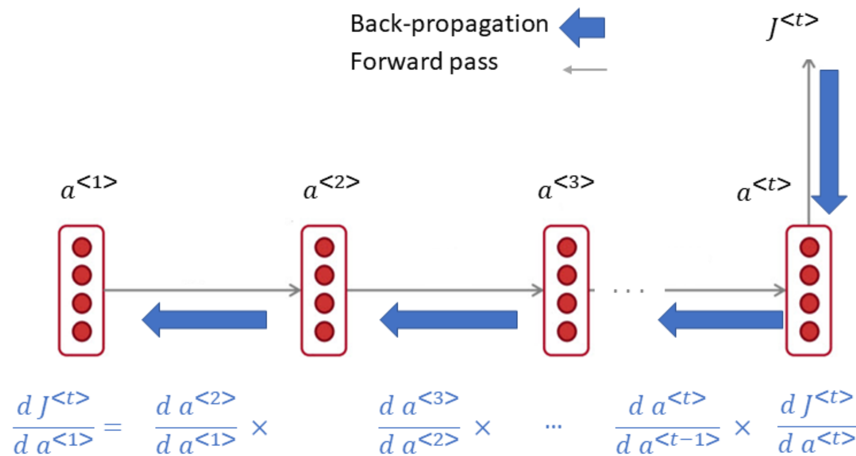


Figure 2.14: By applying the chain rule, the gradient of the loss function can be computed for each hidden layer. If the partial derivatives are small, the gradient gets smaller and smaller when it back-propagates further in time.

2.4.3 Long Short-Term Memory cell

To overcome the problem of vanishing gradient and to learn problems with long range temporal dependencies, LSTM cells contain memory cells ($c^{<t>}$) with self-connections storing and remembering the temporal state of the network. Special multiplicative units called gates are also added to control the flow of information in and out of the cell but also its influence on the activation. At every time steps, the value in the memory cell is either

- updated : Update gate (Γ_u) decides whether a new value should enter the cell
- forget : Forget gate (Γ_f) decides if the value should stay in the cell
- output: Output gate (Γ_o) decides how the cell content impacts the output value

The new value candidate for entering the memory cell ($\tilde{c}^{<t>}$) is computed from the activation ($a^{<t-1>}$) and input value ($x^{<t>}$). The memory cell is updated using a

combination of the old memory cell ($c^{<t-1>}$) state multiplied by the forget gate and the update gate multiplying the new candidate. The equations governing the value of the memory cell are given below as well as the structure of a typical LSTM cell depicted in Fig. 2.15.

$$
\begin{aligned}
\tilde{c}^{<t>} &= \tanh(W_c\, x^{<t>} + U_c\, a^{<t-1>} + b_c) \\
i^{<t>} &= \sigma(W_i\, x^{<t>} + U_i\, a^{<t-1>} + b_u) \\
f^{<t>} &= \sigma(W_f\, x^{<t>} + U_f\, a^{<t-1>} + b_f) \\
o^{<t>} &= \sigma(W_o\, x^{<t>} + U_o\, a^{<t-1>} + b_o) \\
c^{<t>} &= \Gamma_u\, \tilde{c}^{<t>} + \Gamma_f\, c^{<t-1>} \\
a^{<t>} &= \Gamma_o\, \tanh c^{<t>}
\end{aligned}
$$

where $x^{<t>} \in R^F$, $a^{<t>} \in R^D$, $c^{<t>} \in R^D$ and $\Gamma \in R^D$. W ($\in R^{F\times D}$) and U ($\in R^{D\times D}$) are the weights of the LSTM layer and b ($\in R^D$) are the biases. D is the size of the activation and F is the size of the input layer.

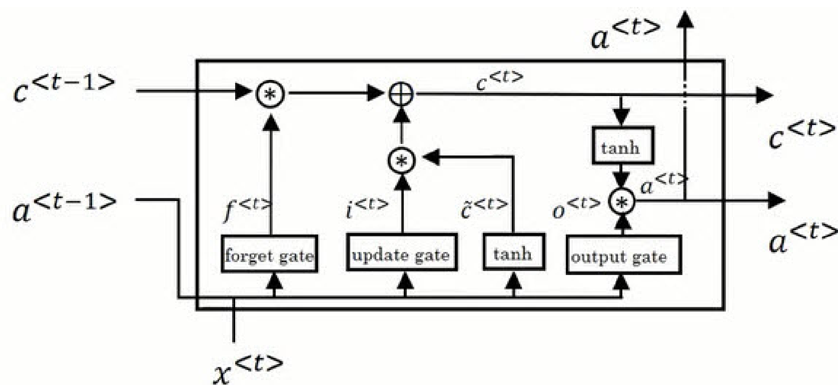


Figure 2.15: Structure of a LSTM cell [52]

The number of parameters in a LSTM layer can be computed as :

$$
\begin{aligned}
\text{Size} &= Size_U + Size_W + Size_b \\
&= 4 \times D \times D + 4 \times D \times F + 2 \times D
\end{aligned}
\tag{2.10}
$$

LSTM models have already proved to be very powerful networks in many sequence modeling tasks such as speech recognition [6, 7, 11, 53] or translation [44].

2.4.4 Improvements to LSTM networks

Bidirectionnal LSTM model

The basic idea of bidirectional recurrent neural nets (BRNNs) [54, 55] is to present each training sequence forwards and backwards using two separate recurrent networks. The output layer is influenced by both nets which means that for every point in a given sequence, the BRNN has information about all points before but also, after it. BRNNs have given improved results in sequence learning tasks, notably protein structure prediction (PSP) [56, 57] and speech processing [58]. Inspired by BRNN, BLSTM were developed and used in keyword spotting tasks [49]. Figure 2.17 illustrates how the forward and reverse subnets combine to classify phonemes. The bidirectional output combines the predictions of the forward and reverse subnets. The global structure of the BLSTM network is presented in Fig. 2.16. However, a problem with this kind of architecture is that the end of the recording must known to start processing the signal preventing real-time computation.

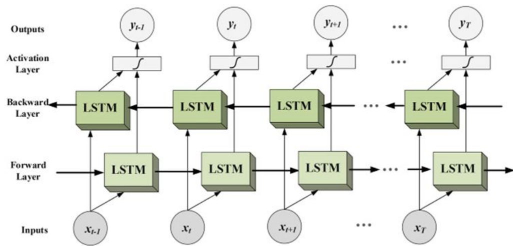


Figure 2.16: General structure of a bidirectional LSTM network presenting the 2 separate nets: 'forward' and 'backward' both connected to the output layer. [59]

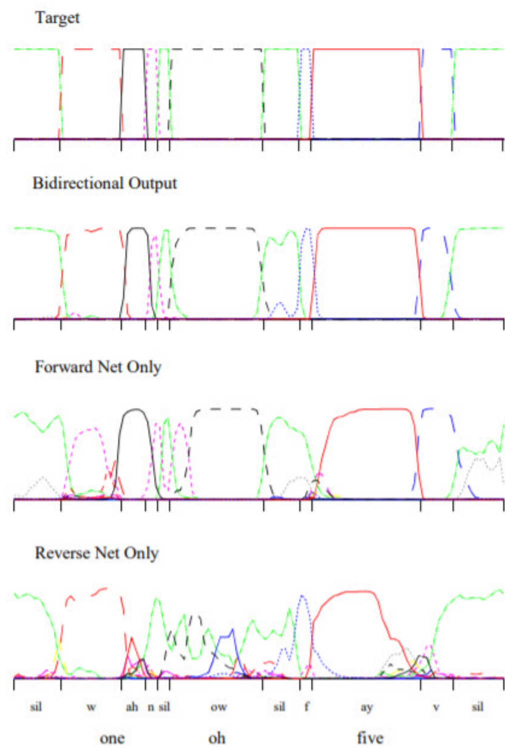


Figure 2.17: A bidirectional LSTM net classifying the utterance "one oh five". The different lines represent the activations(or targets) of different output nodes. [49]

Recently, combinations of deep, bidirectional LSTM (DBLSTM) are often proposed [5, 60]. Multiple LSTM layers are stacked on top of each other combining the effectiveness of several levels of representation already largely used in deep networks with the flexible use of long range context proper to RNNs.
Moreover, to obtain an end-to-end speech recognition, LSTM are nearly always combined with Connectionist Temporal Classification (CTC) method for unsegmented sequence data. [5, 11]

Long short-term memory with projection layer

Long Short-Term Memory Projection (LSTMP) is a variant of LSTM to further optimize speed and performance of LSTM by adding a projection layer. Its central idea is to project the hidden state to a lower-dimensional vector by applying a

projection layer to address the computational complexity of learning LSTM. The cell output units are connected to a recurrent projection layer which connects to the cell input units and gates for recurrence. With the proposed LSTMP architecture, the equations are:

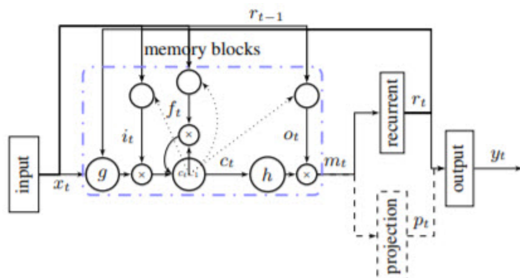


Figure 2.18: LSTMP based RNN architecture with a recurrent projection layer and an optional non recurrent-layer. A single memory block is shown for clarity. [11]

$$
\begin{aligned}
\tilde{c}_t &= \tanh\left(W_{cx}\, x_t + W_{cr}\, r_{t-1} + b_c\right) \\
i_t &= \sigma(W_{ix}\, x_t + W_{ir}\, r_{t-1} + b_i) \\
f_t &= \sigma(W_{fx}\, x_t + W_{rf}\, r_{t-1} + b_f) \\
o_t &= \sigma(W_{ox}\, x_t + W_{or}\, r_{t-1} + b_o) \\
c_t &= i_t\, \tilde{c}_t + f_t\, c^{<t-1>} \\
m_t &= o_t\, \tanh c_t \\
r_t &= W_{rm}\, m_t \\
p_t &= W_{pm}\, m_t \\
y_t &= W_{yr}\, r_t + W_{yp}\, p_t + b_y
\end{aligned}
$$

In these formulas, r_t denotes the projection layer and the other equations are the same as for the LSTM. The activations which are denoted by m_t are not directly passed to the next time step. Instead, the gates (i_t, f_t and o_t) and the new candidate $\tilde{c}_t$ are computed using the projection layer resulting from the matrix multiplication of m_t with the weight matrix W_{rm}. The proposed architecture modifies the standard LSTM to make better use of the model parameters and therefore, improves the computational efficiency, particularly for large networks. Indeed, when the amount of input data is large and that we need to enlarge the size of the memory cell, this can lead to high computational complexity. The number of parameter in a LSTM cell grows in a square pattern with the size of the memory cell. In a LSTMP, the size of the recurrent layer can be tuned to lower this complexity. In [11], the networks were trained on a large vocabulary speech recognition task. Globally, it has been shown that the proposed architecture improves the performances of the LSTM networks significantly over classical LSTM networks. For instance, an LSTMP network achieves a word error rate (WER) of 18% whereas a classical LSTM with approximately the same number of parameters, acheives a WER of 21%.
Later, LSTMP was used in piano continuous note recognition [61] and for indonesian speech recognition [62].

2.5 Training

Once the architecture is established, we must train the network to optimize the network parameters. To do so, we will use training samples for which the label is known. A metric is also needed to estimate the similarity between the obtained output value and the target and so, update the parameter value accordingly. This metric is called the loss.

2.5.1 Loss function

The loss function is the function used to evaluate the output error. In a classification problem, it is quite common to use the cross-entropy loss function. It is defined as:

$$\mathcal{L} = -\sum_{c=1}^{M} y_{o,c} \log p_{o,c} \tag{2.11}$$

where M is the number of classes, $y_{o,c}$ is the true labeling for observation o (equal to 1 if observation of belongs to class c and 0 otherwise) and $p_{o,c}$ denotes the predicted probability that observation o belongs to class c.
Let's take an example. Suppose that we have 4 possible classes ('dog','cat','snake' and 'fish'), the input sample x is from class 'cat' and the network predicts probabilities for each output classes y=[0 0.7 0.2 0.1]. The loss value will be $\mathcal{L} = -\log 0.7$ The cross-entropy increases as the predicted probability diverges from the actual label. That's why, we try to minimize the cross-entropy loss. A max-pooling based loss function for training KWS models with LSTM is also proposed in [9] and achieves better accuracy performances than LSTMs trained with the cross-entropy loss.

2.5.2 Parameters update

The explanation provided in the following section is inspired from [63]. During training, we would like to update the weights in the network according to the cross-entropy loss at the output of the network. To do so, a stochastic gradient descent (SGD) algorithm is used. First, there is a forward pass through the network obtaining the activation for each layer. Then, there is a backward pass where the error gradients are back-propagated through the network. Once this is done, we can update the weights (W) and biases (b) according to this gradient using a SGD:

$$W^{[l]} = W^{[l]} - l_r \frac{d\mathcal{L}}{dW^{[l]}} \tag{2.12}$$

$$b^{[l]} = b^{[l]} - l_r \frac{d\mathcal{L}}{db^{[l]}} \tag{2.13}$$

where l_r is the learning rate and l is the layer index.
The previous equations 2.12 and 2.13 refer to the stochastic gradient descent but, what is the difference with a classical gradient descent? In a classical gradient descent algorithm, the cost function is the sum of the cross-entropy loss on the whole dataset whereas in a stochastic algorithm, the cost function is computed on a single example. In a classical gradient descent algorithm, the weights are updated following:

$$W^{[l]} = W^{[l]} - l_r \frac{d\mathcal{J}}{dW^{[l]}} \text{ where } \mathcal{J} = \frac{1}{m} \sum_{i=0}^{m} \mathcal{L}_i \tag{2.14}$$

where m is the number of samples in the training set. The cost function J gives more accurate gradients whereas $\mathcal{L}_i$ is more noisy. The stochastic gradient descent tries to approximate the classical gradient descent which has the disadvantage to be computationally expensive. The activation has to be computed for the whole dataset in a single forward and backward phase. However, the stochastic gradient descent does not guarantee a good convergence. In practice, the mini-batch gradient descent is a good compromise between the classical gradient descent and the stochastic gradient descent. Batches are sets of samples taken randomly from the training set. In this case, the cost function $\mathcal{J}_{mb}$ is computed:

$$\mathcal{J}_{mb} = \frac{1}{B} \sum_{i=0}^{B} \mathcal{L}_i \tag{2.15}$$

where B is number of samples in a mini-batch.
The choice of the learning rate has also a large impact on the convergence of the model. A too large learning rate can cause the model to fluctuate around the minimum whereas a too small learning rate means a slow convergence. Another challenge that has to be dealt with is the problem of local minima. We need to avoid getting trapped in a local sub-optimal minimum. That is why gradient descents with momentum have been developed to mitigate this problem even if there is still no guarantee of convergence.

Stochastic gradient algorithm with momentum

The stochastic gradient descent with momentum [64] helps accelerate SGD in the relevant direction and dampens oscillations as illustrated in Fig. 2.19.

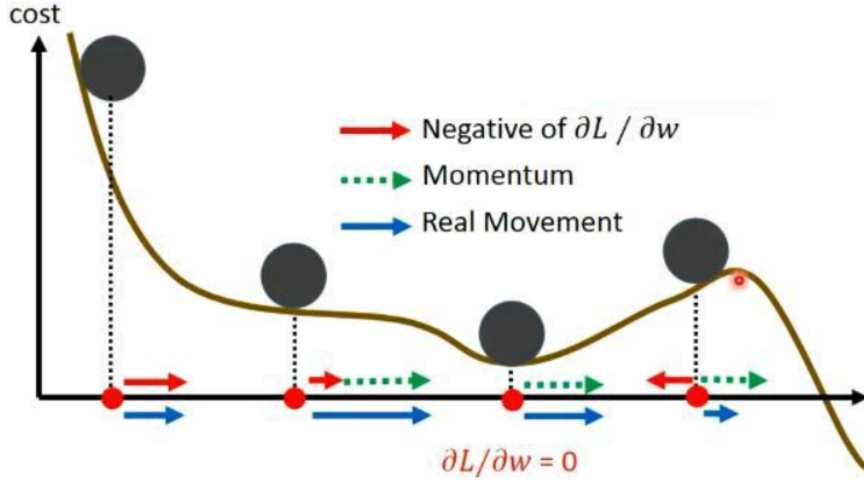


Figure 2.19: Illustration to understand the effect of the momentum and its hope to reach global minima [65]

A velocity vector v_t is defined to keep track of the past update. Now, the weight update depends on the loss function and on the velocity with a proportion defined by μ, the momentum (generally, $\mu = 0.9$).

$$v_t = \mu v_{t-1} + \frac{d\mathcal{L}}{dW^{[l]}} \tag{2.16}$$

$$W^{[l]} = W^{[l]} - l_r v_t \tag{2.17}$$

A better type of momentum is the Nesterov accelerated gradient [66] [67] improving the rate of convergence of the model. The standard momentum method first computes the gradient at the current location and then takes a big jump in the direction of the updated accumulated gradient. On the other hand, the Nesterov first makes a big jump in the direction of the previous accumulated gradient and then, measures the gradient where it ends up and make a correction. The standard momentum and the Nesterov mechanisms are illustrated in Fig. 2.20 and the equations of the Nesterov accelerated gradient are detailed.

$$W_{\text{est}}^{[l]} = W_t^{[l]} - \mu v_{t-1} \tag{2.18}$$

$$v_t = \mu v_{t-1} + \eta \frac{d\mathcal{L}}{dW_{\text{est}}^{[l]}} \tag{2.19}$$

$$W_{t+1}^{[l]} = W_t^{[l]} - v_t \tag{2.20}$$

where $W_{est}^{[l]}$ is the weights values resulting from the first big jump in the direction of the previous accumulated gradient.

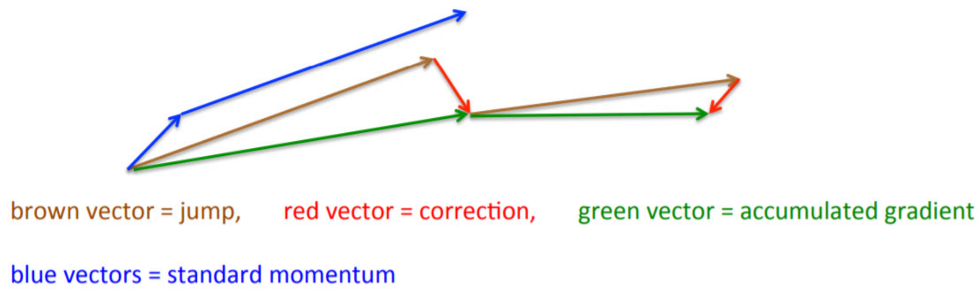


Figure 2.20: Nesterov mechanism [67]

2.5.3 Parameter initialization

As explained in the previous subsection, the learning process is iterative and so, the initialisation is very important for the convergence of the network. The parameters in a neural network are randomly initialised. The parameters cannot be initialised to 0. Otherwise, the activation values in the neural network would all be set to 0 independently of the input values. The parameters cannot be initialised to the same value, neither. In that case, the activation values for a given layer would be equal for all the neurons inside that layer. It means that all neurons within a layer would be updated by the same value.
In practice, the He initialisation [68] strategy is a good weight initialisation when using ReLU activation function:

1. Standard random uniform initialisation: $W^{[l]} \sim \mathcal{U}(0,1)$
2. Multiply according to the number of incoming connections (n): $W^{[l]} = \frac{\sqrt{2}}{\sqrt{n}} W^{[l]}$
3. Biases set to 0: $b^{[l]} = 0$

Once the parameters have been initialised, we can begin the parameters training using a gradient descent algorithm.

Chapter 3

Keyword recognition system

This chapter is dedicated to the explanation of the specific keyword spotting system implemented as part of this work. As illustrated in section 2.1, the system is composed of a feature extractor and classifier implemented using neural networks. Hence, this chapter will detail in its 3 top sections, the dataset, the feature processing and the architecture of the neural networks. Then, the weighted cross-entropy loss used for training will be explained. It is used to ensure a fair learning of each word class. Finally, the metric for the evaluation of the performances will be presented as well as the method estimating the number of operations required for each type of layer.

3.1 Dataset

The Google speech command dataset is an audio dataset of spoken words designed to help train and evaluate keyword spotting systems. The dataset is free and can be uploaded from [12]. It contains a total of 65,000 utterances of 30 different word classes giving around 2000 samples per classes. In other words, we get 2000 recordings of the same word but said by different people, from different countries and with different noise levels. The sampling frequency used is 16000 Hz and each sample is approximately one-second long. As part of this work, to link the word duration with the window size, the distribution of the word duration for each class is estimated using the technique detailed in Appendix A.

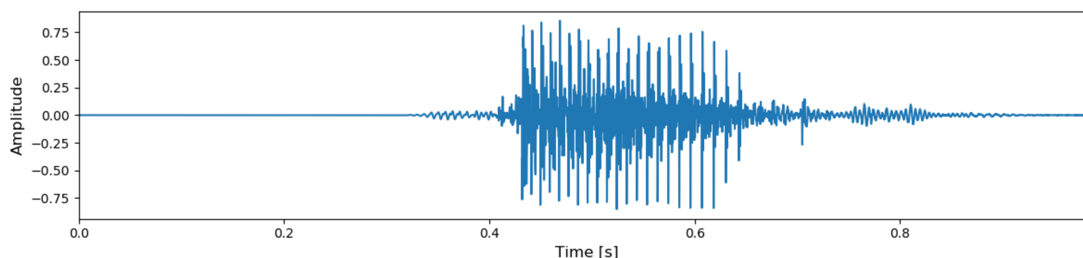


Figure 3.1: Example of the utterance of the word 'down': Words utterance starts around 0.4s and ends around 0.8s.

For this work, only a small part of this dataset is actually used to train and test the networks for complexity issues. The networks are trained to distinguish between 5 classes: "up","down","left","right" and "unknown". In other words, the networks should classify the 4 words correctly and reject any other words in the class "unknown". Each class is composed of approximately 2000 samples making a total of 10000 samples. In the case of the "unknown" class, the 2000 samples come equally from 7 different classes ("dog","bed","house","three","eight","stop","one") to make up a mix. In a real world application, a speech recognition system must be able to deal with unknown words. These classes have been chosen randomly. However, the 'unknown' class contains only short words of 1 or 2 syllables similarly to the 4 other classes.
Two thirds of the dataset is dedicated to training and the remaining third is used for testing. The precise number of samples per classes is summarized in Table 3.1.

	"up"	"left"	"right"	"down"	"unknown"
Training set [# of samples]	1662	1642	1654	1644	1401 (200 from each subclass)
Test set [# of samples]	713	711	713	715	701 (100 from each subclass)

Table 3.1: Dataset composition

3.2 Features processing

First of all, each signal is either extended or cut to contain exactly 16000 samples. These operations are done randomly. The elements are added or deleted partially at the beginning and at the end of the signal in a random proportion.
In the example of figure 3.2, we can observe that signal was extended at the beginning and at the end to last 1 second. Until 0.3s, the speaker remains silent and between 0.3s and 0.9s, the word is said and the frequency content gets interesting.

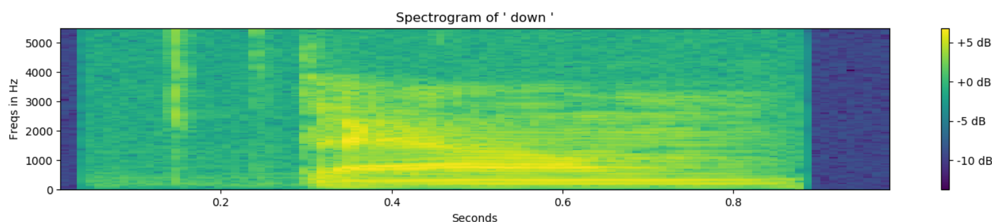


Figure 3.2: Example of a spectrogram where the signal has to be extended at the extremities (blue bands). Word utterance started around 0.3 second. High frequencies have been deleted from the spectrogram

The temporal signal is not fed directly into the neural network. Instead, to extract the relevant information of the signal, it is transformed into the frequency-domain. The spectrogram will be mainly used in this report but the mel-spectrogram will also be shortly studied. The computation of these 2 speech features is described in section 2.2. To compute the spectrogram, each window is made out of 400 samples (25ms) with 260 overlapping samples (15ms). A DFT size of 512 points is used. For each window, we get 256 frequency bins representing frequencies ranging from 0 to 8kHz because our sampling frequency is 16kHz. So, the spectrogram is of size 98x257 (98 frames x 257 frequency bins each). To delete high frequency noise, frequencies higher than 5.5 kHz are deleted from the spectrogram resulting in a spectrogram of size 98x177 (98 time frames with 177 frequency bins). An additional frequency wrapping according to the mel-scale is necessary to obtain the mel-spectrogram.

3.3 Architecture of the neural networks

In this section, the architecture of the neural networks used in this work will be presented: the convolutional neural network (CNN), the long short-term memory (LSTM) and the Grid LSTM.

3.3.1 Convolutional neural network

The structure of the CNN was largely inspired from [10, 69]. In [10], the CNN was used to build an accurate, small-footprint, low-latency keyword recognition system using the Speech Commands Dataset. It is composed of 2 convolutional layers followed by 2 fully-connected layers ending with a softmax layer, as depicted in Fig. 3.3.

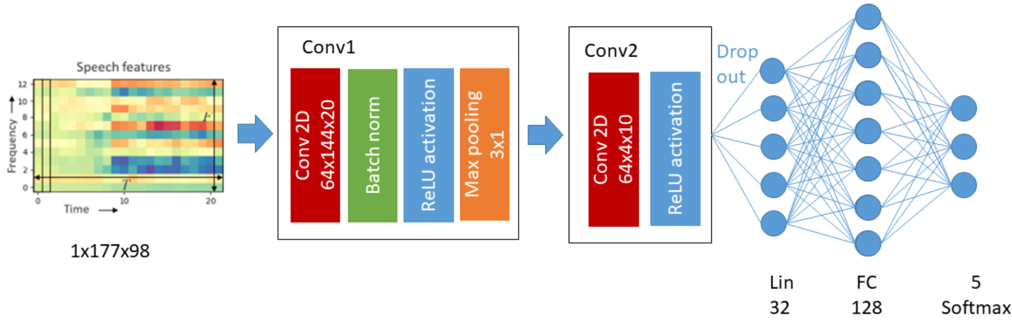


Figure 3.3: Structure of the CNN network: 2 convolutional layers for feature extraction and 3 fully-connected layers for classification

- **Convolutional layers:** The first convolutional layer uses 64 kernels of size 140x20 and a stride of (3,1) which allows the network to pass from a spectrogram of size 177x98 to 64 channels of size 12x79. The second convolutional layer is applied with a kernel of size 4x10 and a stride of (1,1). The output dimension is 64x1x70. In the convolutional layers, a ReLU activation is used. The asymmetric shape of the filters and strides can be justified by the fact that spectrogram dimensions represent different quantities: the time for the x dimension and the frequencies for the y dimension.

- **Max-pooling layer:** The max function is applied on area of size 3 by 1 and with a corresponding stride (3,1). The size along the time dimension is divided by 3 passing from 64 channels of size 12x79 to 64 images of size 4x79.

- **Batch normalization [70]:** It applies a standardization to each layer of the input image which means subtracting the mean and dividing by the batch variance.

$$y = \frac{x - E[x]}{Var[x]} \tag{3.1}$$

  where $E[x]$ denotes the mean and $Var[x]$ represents the variance computed over the mini-batches and per features maps. The batch normalization is a technique to improve speed, performances and stability of artificial neural networks.

- **Linear low-rank [71]:** Linear layers transform the output of the convolutional layer into discrete nodes. The problem when a CNN layer is directly connected to a large fully-connected layer is that the number of parameters required can be very large, especially in comparison to the convolutional layers. In this example, if the convolutional layer was directly connected to a

fully-connected layer with 128 neurons, that would make 70x1x64x128= 573 440 parameters. Instead, this FC layer with a weight matrix A is decomposed as a linear layer with weight matrix B and a FC layer with weight matrix C such that A=B.C. In this example, the linear layer contains 32 output neurons. The number of parameters is thus reduced to 32x70x1x64+128x32=147.456 parameters which means dividing the number of parameters by 4.

- **Fully-connected layers:** The network ends with 3 dense layers: a Linear low-rank layer with 32 neurons to flatten the 2D images, a fully-connected layer (ReLU activation function) with 128 units and finally, a softmax layer with 5 output units.
- **Softmax layer:** The softmax function is applied to the 5 output classes and re-scales them so that the elements lie in the range [0,1] and sum to 1. It is computed for element x_i as:

$$\text{Softmax}(x_i) = \frac{\mathrm{e}^{x_i}}{\sum_{j=1}^{5} \mathrm{e}^{x_j}} \tag{3.2}$$

- **Dropout:** The dropout is a technique to reduce over-fitting. It only keeps a neuron active at random with a certain probability p or set it to 0 otherwise. p is chosen equal to 0.5. The dropout is only active during training. The dropout is applied on the first linear layer as depicted in Fig. 3.3

3.3.2 Long Short-Term Memory

Our input features are spectrograms and have, thus, a temporal dimension. That is why, it could be interesting to exploit it by using a recurrent neural network and more specifically a LSTM network. It is composed of one LSTM layer containing 300 units, followed by a softmax layer for classification. As a reminder, in a LSTM network, each feature vector is processed sequentially along with an activation value coming from preceding time steps.

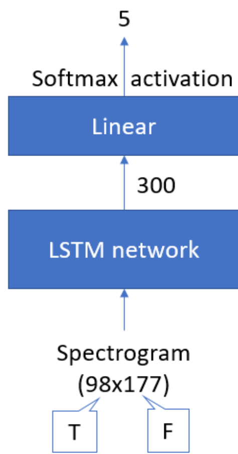


Figure 3.4: LSTM architecture

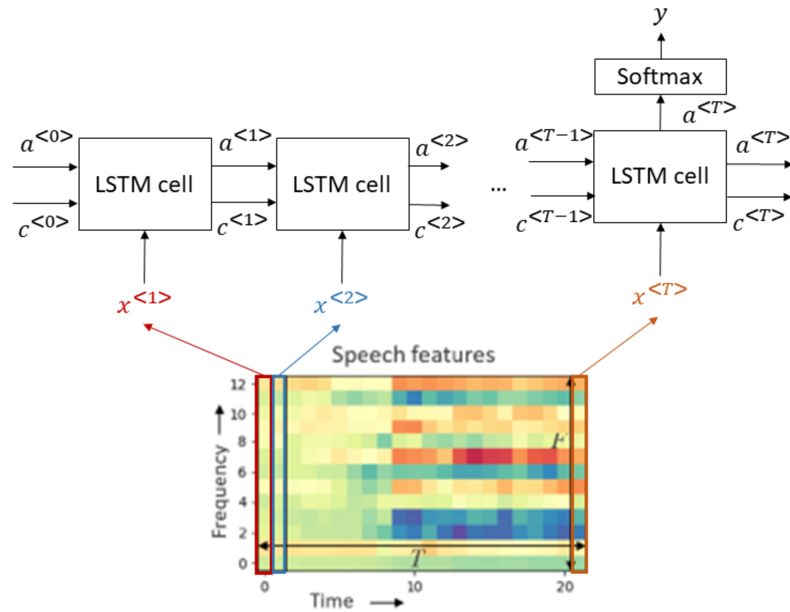


Figure 3.5: 1-layer LSTM architecture

To improve the network performances, two LSTM layers can be stacked on top of each other. They are superimposed so that the hidden states of the first layer are passed as input to the second LSTM layer as represented in Fig. 3.6. Each hidden state and memory cell contain 300 units. Both, the 1-layer and 2-layers LSTM will be explored in this work.

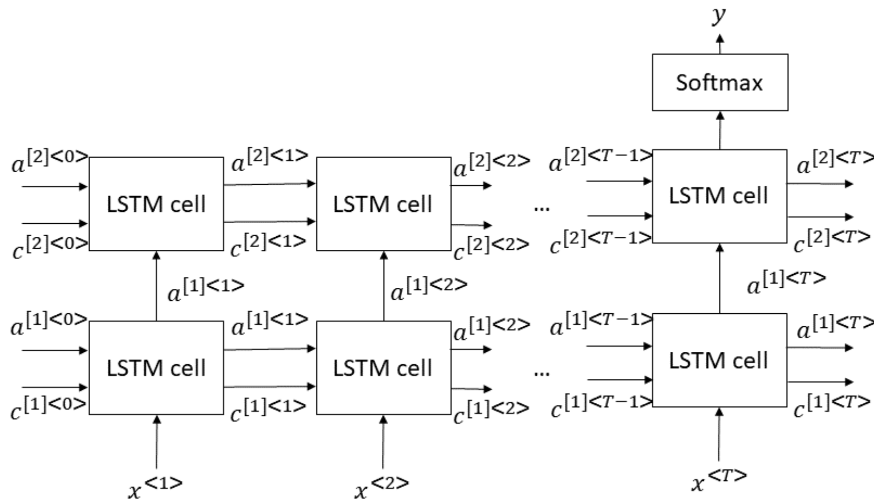


Figure 3.6: 2-layers stacked LSTM: the hidden states of the first layer is passed as input to the second layer.

3.3.3 Grid Long Short-Term Memory

The problem with the 2-layers Stacked LSTM is that the cell value is not passed from one layer to the next layer as mapped in Figs. 3.7 and 3.6. Indeed, it is only the hidden states which are passed as input to the next layer. This means that the cell value cannot affect the output value unless it passes through the hidden layer beforehand. To solve this problem, Grid LSTM [14] networks propagate the memory cell value through the network depth as presented in Fig. 3.8. A 2D Grid LSTM cell is using a LSTM cell to propagate through the time dimension (red arrows) and another through the depth dimension (green arrows) as explained in Fig. 3.9. Considering the two hidden vectors h_1 and h_2 coming respectively from the previous layer and preceding time step as well as the corresponding memory cells m_1 and m_2, the Grid LSTM block computes the output hidden states h'_1, h'_2 and memory cells m'_1,m'_2 by:

$$(h'_1, m'_1) = \text{LSTM Cell}(h_1, h_2, m_1) \tag{3.3}$$

$$(h'_2, m'_2) = \text{LSTM Cell}(h_1, h_2, m_2) \tag{3.4}$$

The notations have been slightly modified in comparison to the previous sections to follow the notation from the paper. The hidden states denoted by $a^{<t>}$ in the previous sections are replaced by h and the memory cell denoted by $c^{<t>}$ in the previous sections are represented by m. Each LSTM cell has distinct weight matrices and applies the standard LSTM mechanism described in section 2.4.3 across the respective dimension. However, sharing the weight matrices can be specified along any dimension in a Grid LSTM. It induces invariance in the computation along that dimension. When the weights are shared along all dimensions including the depth, we call it a Tied Grid LSTM. It helps reducing the number of parameters in the model. The whole network is described in [14] where the model was used for algorithmic tasks such as 15-digit integer addition and sequence memorization on the Wikipedia character prediction benchmark.

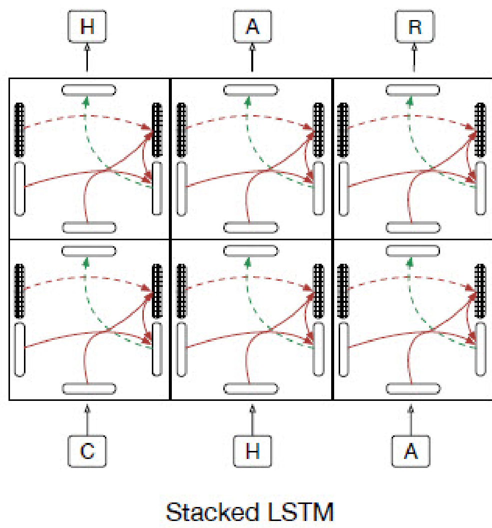


Figure 3.7: Scheme of a 2D Stacked LSTM [14]: The cell value does not propagate through the depth dimension

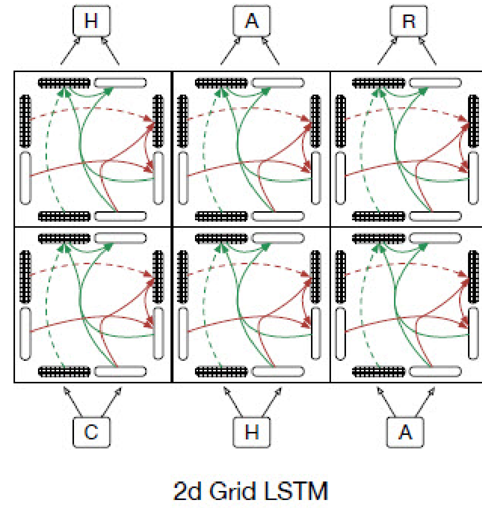


Figure 3.8: Scheme of 2D Grid LSTM [14]: the cell value propagates through the network depth

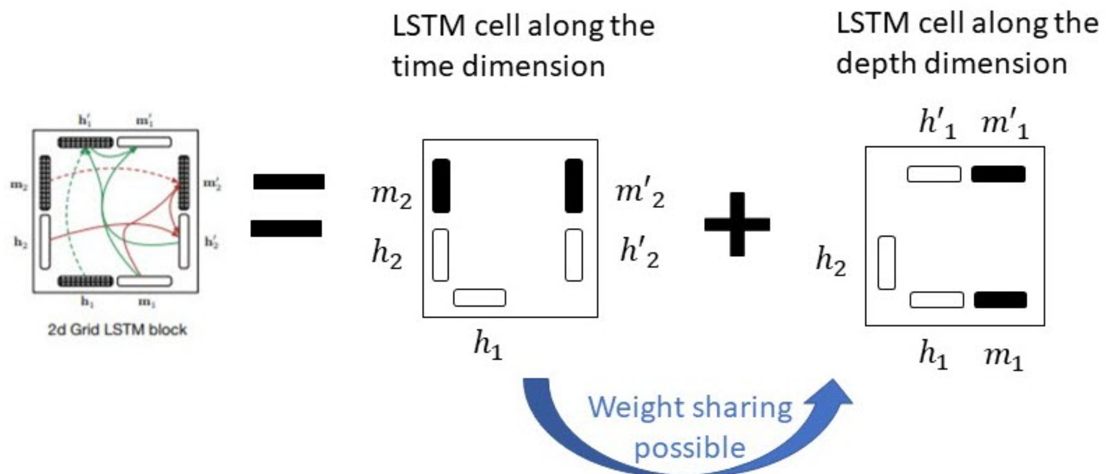


Figure 3.9: Grid LSTM cell contains 2 LSTM cells, one for each dimension: the propagation through the depth dimension is represented in green and the propagation through the time dimension is presented in red.

3.4 Training and weighted cross-entropy loss

Concerning the training, we use the stochastic gradient descent (SGD) algorithm with momentum. For the convolutional neural network, the Nesterov method is used whereas the SGD with momentum is used for LSTM networks and the Grid LSTM ones. For LSTM, the learning rate and the momentum are respectively equal to 0.01 and 0.9 while for CNN, they are set to 0.001 and 0.9 respectively . All the networks are trained on 360 epochs as it ensures their convergence and mini-batches of 100 samples will be used.
The "unknown" label gathers signals from different subclasses which are extremely different making the "unknown" class difficult to learn. Indeed, when we train a network either CNN or LSTM, it does not learn to classify the class "unknown". For instance, the accuracy performance among the different classes can vary largely, reaching 80% for the specific classes ("left","right","up" and "down") and 0% for the unknown class. As explained previously, the large variety of signals labeled under "unknown" makes it difficult to learn in comparison to the other classes. To solve this problem, a weighted cross-entropy can be used. The cross-entropy loss explained in section 2.5.1 is theoretically computed for an observation o coming from class C using:

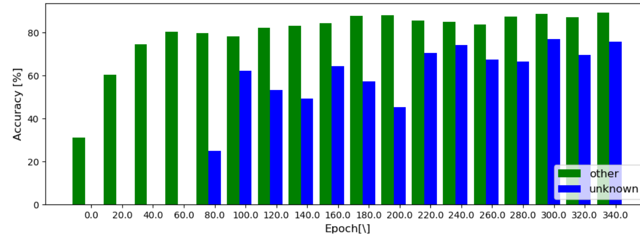
$$\begin{aligned} \mathcal{L}(o, C) &= -\sum_{c=1}^{M} y_{o,c} \log p_{o,c} && (3.5) \\ &= -\log p_{o,C} && (3.6) \end{aligned}$$

If the cross-entropy loss is weighted for each class, the formula becomes:

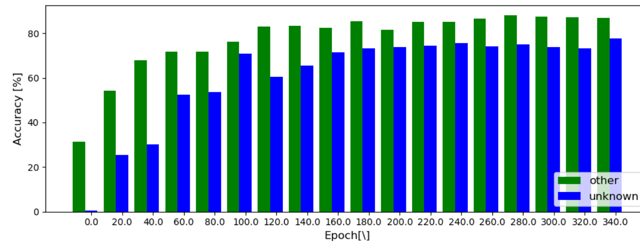
$$\mathcal{L}(y, C) = -w[C] \, \log p_{o,C} \tag{3.7}$$

This weight allows to adjust the importance of the different classes to make sure each class is learned similarly. The weights are updated at each epoch according to the accuracy performances for each class c ($acc[c]$) following an exponential moving average:

$$w_t[c] = \alpha \, (1 - acc_t[c]) + (1 - \alpha) \, w_{t-1}[c] \;\text{ for } c \in [1, 5] \tag{3.8}$$



(a) No weighted cross-entropy: The class 'unknown' is not learned until the 60th epoch.



(b) Weighted cross-entropy: The class 'unknown' is learned from the start thanks the weighted cross-entropy

Figure 3.10: Comparison of the performances for the class 'unknown' and the 4 other classes when the weighted cross-entropy is used and not used on the 1-layer LSTM network. In green, mean accuracy of the 4 classes ('up', 'left', 'right', 'down') and in blue, accuracy of the class 'unknown'.

The results on the 1-layer LSTM obtained with the classical cross-entropy and with the weighted cross-entropy loss are shown in Fig. B.2. We can see that until the 60th epoch, the class 'unknown' is not learned if the weighted cross-entropy is not used. The final performance of the network is not impacted by the use of the weighted cross-entropy. The weighted cross-entropy impacts mostly the first epochs and ensures that the 'unknown' class is learned right from the beginning. If a 1-layer LSTM is trained on 50 epochs without the weighted cross-entropy, that could lead to large class inequalities. The same comparison is done for the CNN network and the conclusions are basically the same as for the LSTM. The results are presented in Appendix B.
The importance of using a weighted cross-entropy is even more present when the sizes of the classes are uneven. If the proportion of each class is different, the system will only learn classes represented with more training samples. For instance, the LSTM was trained using a dataset similar to the one presented in section 3.1. The only difference is that the "unknown" class was enlarged and contains seven times more samples in comparison to the four other predefined words classes. This means that the proportion between these classes and the "unknown" class is

respectively 35% and 65%. In fact, this dataset composition is even more realistic and close to real-life applications where most of the recordings will turn out to be "unknown" words. On this kind of dataset, the weighted cross-entropy is crucial to avoid classifying all the words as "unknown". The results are presented in Appendix B.

The value of the α parameter is very important to avoid large instabilities during the learning. When α is equal to 1, there is in fact no exponential moving average. The weights ($w[c]$) are simply equal to 1-acc[c] which can make the system oscillate as observed in Fig. 3.11a. In Fig. 3.11b and 3.11c, both networks converge with α equal to 0.1 or 0.5 but the learning is much faster and smoother when using $\alpha = 0.1$. However, this faster convergence with $\alpha = 0.1$ comes with the drawback on weight initialisation. Indeed, as the weight update is reduced, more focus should be put on the weights initialisation. If this not done, the learning could be slowed down.

One final point concerns the learning rate. In our case, the learning rate is chosen constant. Reducing the learning rate during training is quite popular in most state-of-the-art articles using adaptive learning algorithms to improve the rate of convergence [10]. However, as we are using a weighted cross-entropy linked to the class accuracy, the effect on the learning algorithm is the same as a learning rate decay. Indeed, the accuracy increases during the learning and therefore, the weights $w[c]$ decrease, reducing the learning step. Adagrad [72] and Adam [73] are also very popular gradient descent algorithms which implement adaptive learning steps.

(a) α=1: No convergence of the network

(b) $\alpha = 0.5$: Large instabilities

(c) $\alpha = 0.1$: Smooth learning

Figure 3.11: Learning using different values for α

3.5 Performance computation

To measure the performances of a classifier, we need to find correct statistical measures. We have decided to use the accuracy as metric for the analysis of our

model. It is computed as :

$$\text{Accuracy} = \frac{\text{Correctly classified examples}}{\text{Total number of examples}} \tag{3.9}$$

For a per class accuracy:

$$\text{Per class accuracy} = \frac{\text{Examples from that class correctly identified}}{\text{Number of examples in that class}} \tag{3.10}$$

This means that the total accuracy is not equal to the average of the 5 class accuracies because the classes do not have exactly the same size. If we want a more detailed analysis than just the mere proportion of correct classifications, we can build the confusion matrix.

		Actual class	
		Cat	Non-cat
Predicted class	Cat	5 True Positives	2 False Positives
	Non-cat	3 False Negatives	3 True Negatives

Figure 3.12: Example of a confusion matrix for a simple binary classifier

This kind of representation allows to visually understand the class errors made by the classifier. One example is presented in Fig. 3.12 for a simple binary classifier classifying objects as cat or as non-cat. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class.

3.6 Estimation of the computational complexity of the models

The purpose of my work is to study the performances of the LSTM and CNN network in relation with their memory footprint and computational complexity. The computational complexity of the networks will be expressed in floating point operations (FLOP). The speed of a network is mainly determined by the number of additions, subtractions, divisions and multiplications required. By estimating the speed of a model with FLOP, we should be aware that we ignore the time required for the data transfer and that all operations are assumed to have the same computational complexity. In reality, the multiplications and the divisions are computationally more expensive.
The operations in neural networks are mainly dot products, such as:

$$y = w[0] * x[0] + w[1] * x[1] + ... + w[n-1] * x[n-1] \tag{3.11}$$

where w and x are vectors and y is a scalar number. In terms of floating point operations, 2n-1 FLOPs are required as there are n-1 additions and n multiplications.

- **Fully-connected layer**: Considering n_i input neurons connected to all n_o output neurons and based on the expression written in section 2.3.1, there is $2 \times n_i \times n_o$ FLOPs. Indeed, a matrix product is simply a set of n_o dot products between the input vector X of size n_i and one column of the weight matrix W. The addition of the bias term adds n_o operations to the $(2n_i-1)n_o$ operations of the matrix product.

- **Convolutional layer:** As described in section 2.3.2, we will consider a convolutional layer with c_{out} filters of size (k_1, k_2). The input and output feature maps are respectively of size $c_{in} \times H_{in} \times W_{in}$ and $c_{out} \times H_{out} \times W_{out}$. There will be $2 \times c_{in} \times k_1 \times k_2 \times c_{out} \times H_{out} \times W_{out}$ FLOPs. As explained in [74], for each pixel of the output feature map $H_{out} \times W_{out}$, a dot product is computed between the filters (k_1, k_2) and the receptive field. This operation is done for the c_{in} input channels and repeated for the c_{out} filters. By using H_{out} and W_{out} in this formula, the stride value is already taken into account.

- **Max pooling layer**: Suppose a filter area of size (k_1, k_2) with a stride (k_1, k_2) over c_{in} feature maps of size (H_{in}, W_{in}), the maxpooling layer will require approximately $H_{in} \times W_{in} \times c_{in}$ FLOPs.

- **LSTM layer**: Assuming that there are D units in the hidden layer and F units in each input vector, there will be approximately $4 \times 2 \times (n_i + n_a) \times n_a$ floating-point operations per LSTM cell, corresponding to 4 FC layers. To get the number of operations in the whole layer, we should multiply by the number of time steps.

When estimating the FLOP of each layer, we have neglected the activation function which is generally negligible in comparison to the matrix multiplication. For example in the fully-connected layer, there will be n_o operations for the ReLU activation in comparison to the $(2n_i - 1) \times n_o$ operations for the matrix product. For other activation functions such as the sigmoid or the hyperbolic tangent, the function application is computationally more expensive but can be reduced by using lookup tables.

Chapter 4

Experimental results

In this chapter, the results obtained with the keyword spotting system described in chapter 2 will be presented and compared in terms of performance, memory and computational complexity. First, the architecture of the LSTM network will be studied to establish if it is better to add an extra layer or to enlarge the hidden state of a 1-layer LSTM. Then, the second section will present a comparison between CNN, LSTM and Grid LSTM models. The effect of the input features will also be investigated. The window size used for the spectrogram computation will be tuned and we will try to use a mel-spectrogram as speech features.
Before going deeper into the results analysis, it seems interesting look at the raw results obtained by simulating the models. As an example, the performances of the 1-layer LSTM model is represented in Fig. 4.1.

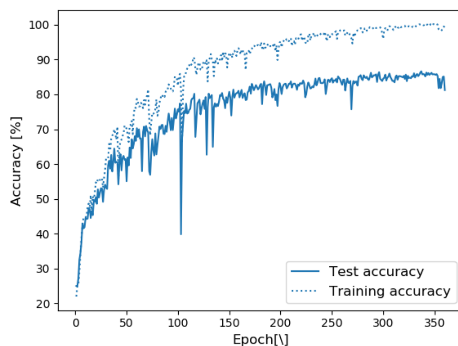


Figure 4.1: Test (solid) and training (dotted) accuracy for a 1-layer LSTM model with 300 neurons in the hidden layer.

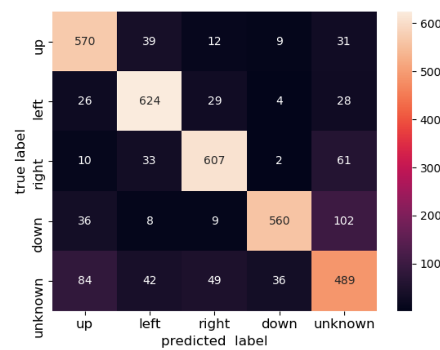
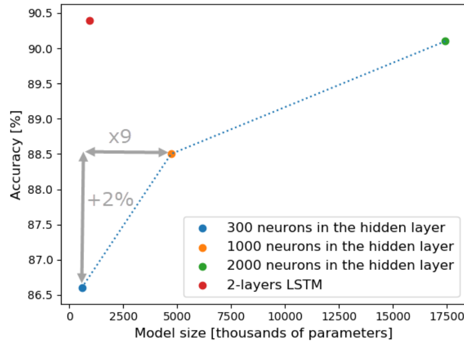


Figure 4.2: The confusion matrix of the 1-layer LSTM network with 300 units in the hidden state.

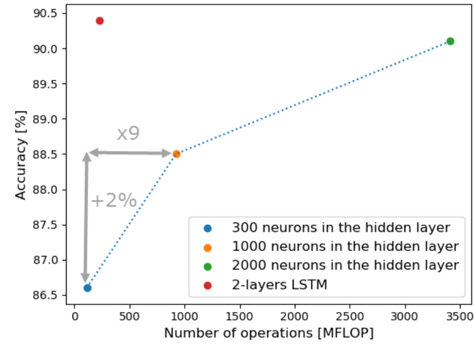
The training and the test accuracy of a 1-layer LSTM with 300 neurons in the hidden layer during the learning process are shown in Fig 4.1. The training accuracy refers to the accuracy obtained on the training samples whereas the test accuracy denotes the performances on the test samples, not used for learning and never seen by the network. During each epoch, the network is first presented with training examples and then, with the test and training examples, to assess its performance. Obviously, the training accuracy is always larger than the test one. In the following sections, the test accuracy will be used for the model comparison to obtain results as in a real-life experience. Figure 4.2 represents its confusion matrix. It enables to determine the most confusing classes. Globally, all classes are well learned. The largest sources of error are words from the class "down" being classified as "unknown". We can see that the class "unknown" is responsible for most of the confusing errors. Indeed, words are wrongly labeled as "unknown" and words from the class "unknown" are also attributed to other classes. We can conclude that the performance would be much better if only predefined word classes are used.

4.1 Study of the LSTM architecture: size of the hidden layer and depth of the network

To improve the LSTM network performances, we analyze if it is better to enlarge the size of the activation and memory cell or to add an extra layer to the network. The architecture of the 1-layer and 2-layers LSTM networks are described in section 3.3.2. Figure 4.3a shows that enlarging the hidden layer indeed increases the network performances. However, the number of parameters and operations in a LSTM network depends on the square of the size of the hidden layer as expressed in Eq.2.11 and in section 3.5. Going from 300 neurons to 1000 neurons allows to increase the accuracy performance of 2 % but at the cost of multiplying by 9 the size and computational complexity of the model. That is why increasing the size of the hidden layer is not efficient in terms of accuracy-memory-operations trade-off.



(a) The number of parameters depends on the square of hidden layer whereas the number of parameters is more or less doubled when adding an extra layer.



(b) The number of operations depends on the square of hidden layer whereas the number of operations is approximately doubled when adding an extra layer.

Figure 4.3: Impact of the size of the hidden layer on the accuracy performance, on the number of operations and parameters

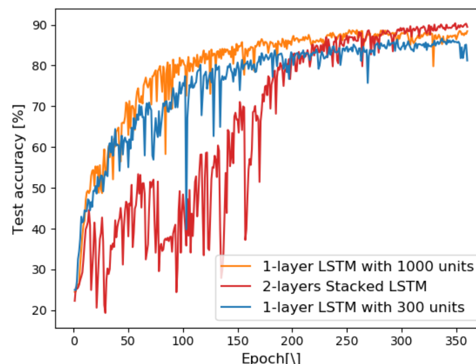


Figure 4.4: The 2-layers network presents large instabilities during the training

Another way to improve the performances of the network could be to stack two LSTM layers one on top of the other. The scheme of the 2-layer LSTM is represented in Fig. 3.6. In comparison to the larger 1-layer LSTM, we can observe in Figs. 4.3a and 4.3b that the 2-layers LSTM network reaches better performances with a much reduced number of parameters and operations. For that reason, it seems better to add an extra layer than to enlarge the hidden layer of the 1-layer LSTM. One drawback of the 2-layers LSTM is the learning instabilities shown in Fig. 4.4. We can see that the network experiences more instabilities and that we have to wait approximately 120 epochs before the network starts learning efficiently.

These learning problems can be due to the vanishing gradient problem along the depth dimension as explained in section 2.4.2. The error gradient tends to vanish when back-propagating through the network depth.
However, these learning difficulties can also be due to non-optimal learning parameters. Indeed, the training process depends on many parameters: the learning rate, the momentum, the learning algorithm,... In this thesis, the learning parameters are chosen to find a convergent network but not optimized. Adding a layer seems to change the behavior of the network during training contrary to enlarging the hidden state of the network. This observation can be used to train larger LSTM network. Indeed, the learning parameters could be optimized on a LSTM network with small hidden layers and once the optimal learning parameters are found, they could be used to train a LSTM network with larger hidden layers.

4.2 Comparison between Basic LSTM, Grid LSTM and CNN

In this section, the different models described in section 3.3 will be compared in terms of accuracy performances, number of parameters and number of operations. Thereafter, this comparison will be put in parallel with the results provided in [6]. The models studied are:

- CNN composed of 2 convolutional layers and 3 dense layers as described in section 3.3.1
- 1-layer LSTM
- Stacked 2-layers LSTM
- Tied 2-layers Grid LSTM
- Tied 4-layers Grid LSTM
- Untied 2-layers Grid LSTM

The ReLU non-linear function is used for the six networks and the activations of all the LSTM models contains 300 units.

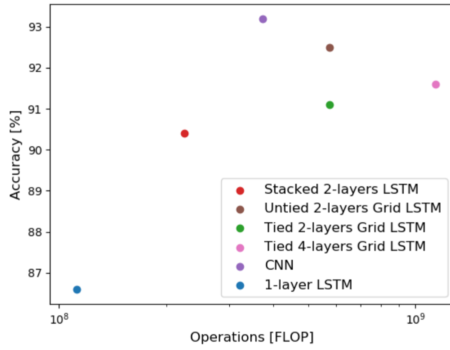


Figure 4.5: Accuracy vs. Computational complexity

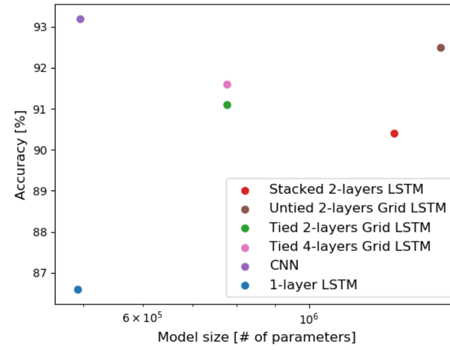


Figure 4.6: Accuracy vs. Memory footprint

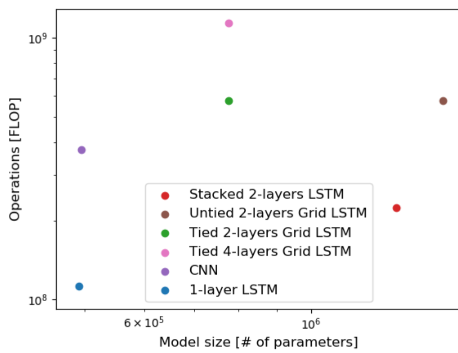


Figure 4.7: Computational complexity vs. Memory footprint

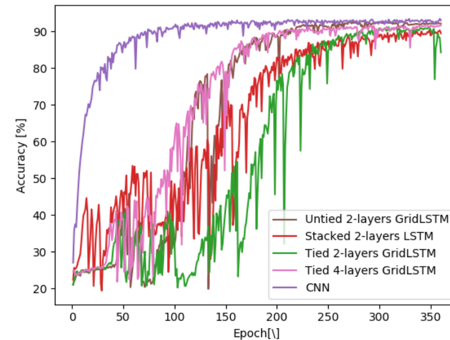


Figure 4.8: Test accuracy for the 5 models implemented: CNN converges much more rapidly

Using Figs. 4.5, 4.6 and 4.7, we will be able to compare the 6 models previously described. The main conclusions from these graphs are explained below:

- The CNN and the Untied 2-layers Grid LSTM reach the best performance with nearly 93% of accuracy . However, the Untied Grid LSTM requires a much larger number of parameters, nearly multiplied by 3 in comparison to the CNN. Therefore, the CNN gives the best accuracy with a small number of parameters and a limited number of operations.

- The Tied Grid LSTM can be an alternative to the Untied version to reduce the number of parameters. However, to reach the same accuracy, the number of layers should be increased and so, the number of operations is also larger. For example, the Tied 4-layers Grid LSTM achieves 91.6% of accuracy with 777

000 parameters and 1.14 GFLOPs, which is 3 times the number of operations required by CNN and there is a factor 2 in comparison to the Untied Grid LSTM.

- The 1-layer LSTM reaches 86.6% of accuracy with the smallest number of operations and parameters, which could be useful for hard-constrained problem regarding the memory and operations.
- The Grid LSTM improve the network performance in comparison to basic LSTM models but at the cost of a higher number of operations.

The evolution of the test accuracy during the learning is shown in Fig.4.8 to compare the rate of convergence. As already mentioned, the information should be taken with care because the training parameters are not optimized. It seems that the CNN converges much more rapidly compared to LSTM and Grid LSTM networks.
As announced in the introduction, a comparison between state-of-the-art models for KWS applications was already proposed in [6] and contrasting those models with the results from this work seems interesting. Table 4.1 compares the different features from both works and it clearly comes out that this master thesis does not achieve state-of-the-art results.

	"Hello Edge: KWS on microcontroller" by Zhang Y., el. 2018 [6]	This work
Dataset	10 classes ("Yes","No", "Left", "Go" "Right","Down", "Up", "On", "Off", "Stop" + "silence") from Google Speech Command dataset	5 classes ("Left","Up", "Right","Down" and "unknown") from Google Speech Command dataset
Training	Adam for 20K iterations	SGD with momentum or Nesterov for 10K iterations
Features Processing	Data augmentation with noise and random time shift MFCC computation	Spectrogram computation Deletion of high frequency noise above 5.5 kHz
Input features Size	40 MFCC x 49 time steps	177 freq. bins x 98 time steps
Best CNN model	497800 parameters 25.3 MFLOP 92.7 %	496256 parameters 372.8 MFLOP 93.2 %
Best basic LSTM model	497 800 parameters 47.9 MFLOP 93.4 %	576 305 parameters 112.2 MFLOP 86.6 %

Table 4.1: Comparison of the results with [6]

As a matter of fact, [6] proposed a CNN reaching 93.4 % of accuracy and with 25.3 MFLOPs whereas, in this work, the CNN model achieves 93.5% of accuracy

with more than 10 times more operations. Regarding the basic LSTM model, this work suggests a LSTM model reaching only 86.6 % with 2 times more operations and a bit more parameters in comparison to [6].
We will known try to spot the differences between the two works responsible of the performance difference. There are many differences between these 2 works such as the training algorithms or the number of classes but to me, the performance gap may come from the training iterations and more importantly, the features processing and the "unknown" word class.
The models are trained for 10K iterations while in [6], they were trained 2 times longer. However, even if any small performance improvements are welcome, it seems from Fig.4.8 that models have converged and that the accuracy has stopped improving.
Secondly, the input features are extremely different. Zhang and el. used the MFCC whereas the simple spectrogram is used as part of this work. Using the MFCC has 2 kinds of advantages in comparison to the simple spectrogram: it is a more complex feature extraction which is supposed to lead to better performance, and a more efficient features compaction reducing the number of parameters and operations in the neural networks classifying those features. However, the MFCC are computationally more expensive than the spectrogram. In section 4.4, the CNN is trained with the mel-spectrogram which is the first step in the MFCC computation, but there is no performance enhancement observed and the learning process is much more unstable as observed in Fig. 4.14.
A final important difference that might be responsible of the accuracy difference observed, is the kind of word classes. As already explained in section 3.4 and the introduction of this chapter, the class "unknown" is much more difficult to learn and thus, can also be responsible for a part of the accuracy drop in comparison with [6]. They uses 10 classes with noise addition whereas we only uses 5 classes but, including the more complex "unknown" class.
To conclude the comparison with this article, we could say that the "unknown" class might be responsible of the accuracy drop. However, it is still interesting to keep improving the input features to reduce memory and computational requirements, which may, at the same time, lead to an accuracy enhancement.

4.3 Study of the spectrogram shape

The impact of the window size on the accuracy performances is studied in this section. By default, a window size of 400 samples corresponding to 25ms is generally recommended as explained in section 4.3. However, we could wonder if this value is suitable for all speech datasets and for all types of networks.

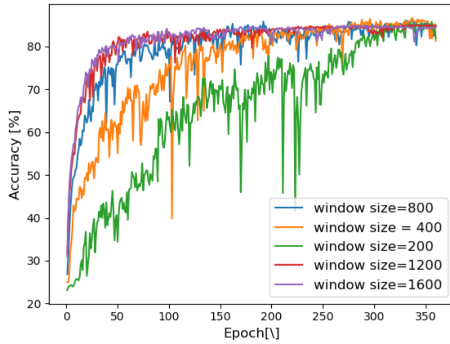
Window size [ms]	12.5	25	50	75	100	125
Spectrogram shape [time steps x input size]	198x89	98x177	48x353	31x529	23x705	18x881

Table 4.2: Shape of the spectrogram for different window size: when the window size is increased, the number of time steps decreases while the input size increases.

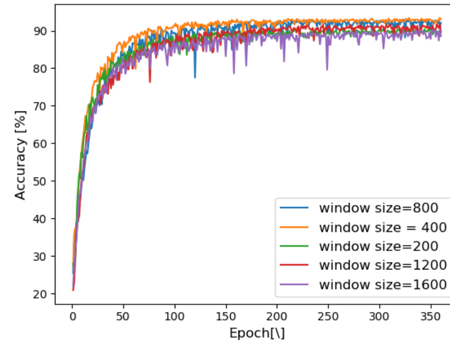
To answer this question, window sizes between 12.5ms and 125ms are tested, while keeping the overlapping proportion constant, to find an optimal value and to compare it between the CNN and LSTM networks. The CNN is described in section 2.7 and we will compare it with a 1-layer LSTM network with 300 units in the hidden state. From Fig. 4.11, it seems that there is indeed an optimal value for the CNN network matching the one for the LSTM network. A 25ms-window gives the best performances for both networks, improving the accuracy as much as 3% in comparison to other window sizes. These results may indicate that there is a link between the word duration and the optimal window size. For that purpose, the distribution of the word duration of each class is represented in Fig. 4.10. The technique used for the computation the histogram is explained in Appendix 4.10 and was applied to the whole dataset, training and test set. The mean and standard deviation for each word class are detailed in Table 4.3. As expected, the word duration for the class "up" is in average much smaller than for the class "down". The standard deviations are quite large in comparison to the mean values meaning that the word duration varies widely within a class. From this histogram, we can conclude that the duration of a word is uncertain and varies between and within a class. Comparing the histogram values with the window size, it seems that the window size must be much smaller than any words from the dataset.

	"Up"	"Left"	"Right"	"Down"
Mean (μ) [ms]	231	333	363	465
Standard deviation (σ) [ms]	187	188	189	182

Table 4.3: Mean and standard deviation of the word duration for each class



(a) 1-layer LSTM network: The window size impacts the rate of convergence



(b) CNN network: The rate of convergence is slightly impacted by the window size

Figure 4.9: Test accuracy for different window sizes with CNN and LSTM networks

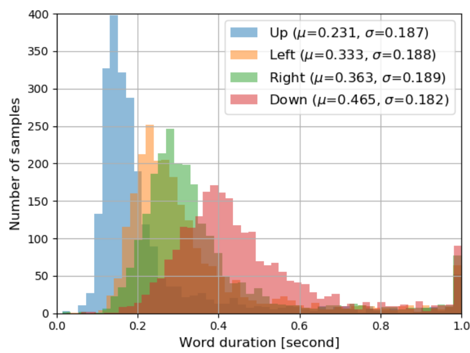


Figure 4.10: Histogram of the word duration for each class. High variance within and between the word classes

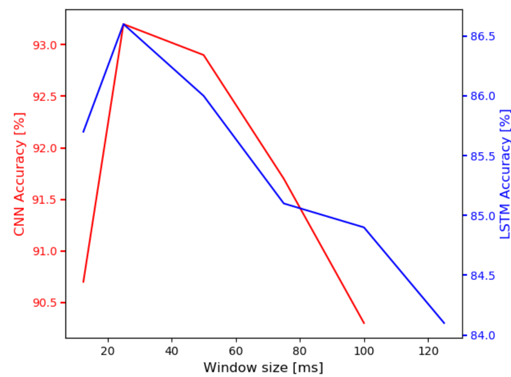
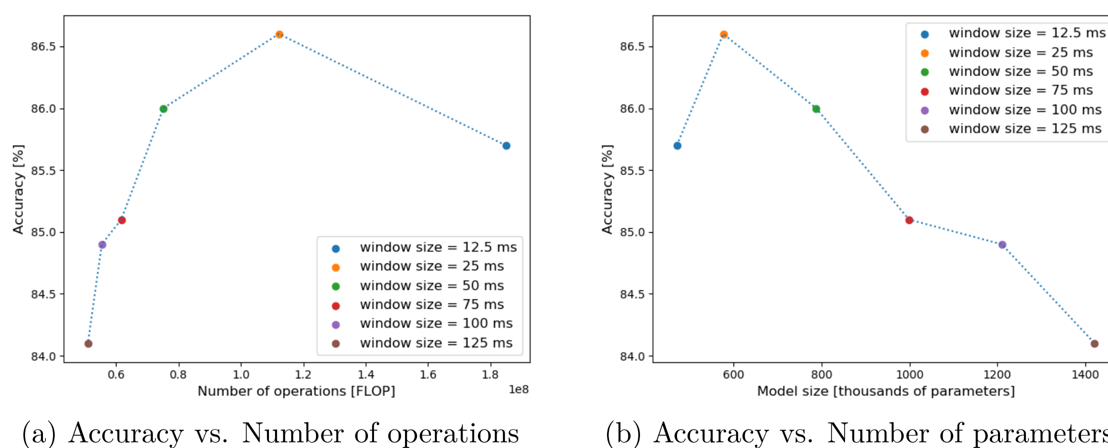


Figure 4.11: Comparison of the CNN and LSTM networks when modifying the window size: the optimal window has the same size for both networks.

Furthermore, modifying the window size also changes the shape of the spectrogram. Therefore, the architectures of the neural networks must also be adapted accordingly and it can affect the network performances. The shapes of the spectrogram for each window size are gathered in Table 4.2. Narrowing the window reduces the input size but increases the number of time steps.
Regarding the 1-layer LSTM model, the number of parameters increases linearly with the input size and does not depend on the number of time steps because the weights are reused across the time steps. The number of parameters is expressed in Eq. 2.11. On the contrary, the number of operations decreases non-linearly with

the window size. The number of operations simultaneously depends on the number of time steps and on the input size as expressed in section 3.5. To compute the model size and the number of operations, the hidden size is fixed to 300 neurons and the input size along with the number of time steps change with the shape of the spectrogram as described in table 4.2. In Figs. 4.12a and 4.12b, a 25ms-window reaches the best performance with a reduced number of parameters but a higher number of operations. From Fig. 4.9a, it seems that changing the window size and therefore, the input size and number of time steps, impacts the rate of convergence of the LSTM network. Using a larger window corresponding to a larger input size with fewer time steps helps accelerating the model convergence. In fact, for the LSTM network, the accuracy drop is mainly due to this difference in the rate of convergence which is not case for the CNN network.



(a) Accuracy vs. Number of operations

(b) Accuracy vs. Number of parameters

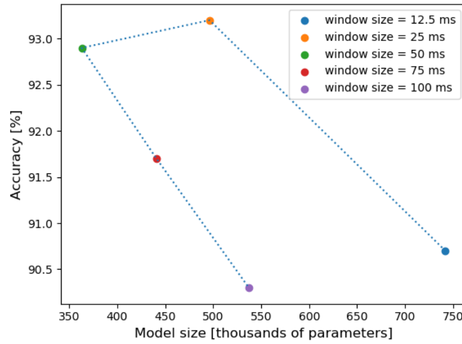
Figure 4.12: Impact of the window size on the accuracy, number of operations and parameters for the 1-layer LSTM network

Concerning the CNN model, the shape of the filters of the convolutional layers are adapted manually so that the different spectrograms fit in the network as detailed in Table 4.4.

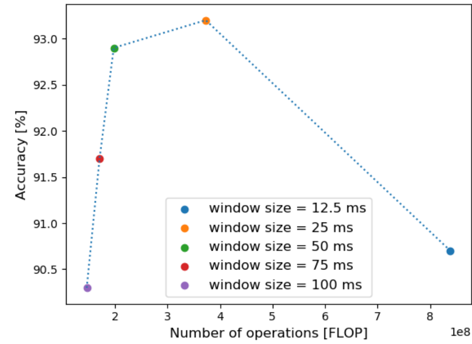
window size $[ms]$	Filter size for Conv1 $[n_{filters}$ x k_1 x $k_2]$	Filter size for Conv2 $[n_{filters}$ x k_1 x $k_2]$
12.5	64 x 56 x 120	64 x 4 x 10
25	64 x 144 x 20	64 x 4 x 10
50	64 x 320 x 10	64 x 4 x 5
75	64 x 496 x 10	64 x 4 x5
100	64 x 672 x 10	64 x 4 x 5

Table 4.4: Hyper-parameters for the convolutional layers for each window size: number of filters, shape of the filters (height and width)

The number of operations in the CNN model decreases with the window size as represented in Fig. 4.13b. In contrast, the relation between the window size and the number of parameters is not straightforward as represented in Fig. 4.13a. From Fig 4.9b, we can see that the rate of convergence of the CNN model is not influenced by the window size unlike the LSTM network. Putting all together, it seems interesting to choose a window size of 50 ms with its corresponding architecture to achieve the best performance, complexity and memory trade-off. In [3], the window size is also tuned with a CNN network and with the same dataset. However, the overlapping part is not adapted according to the window size as we did here. This means that the overlapping part contains a fixed number of samples irrespective of the window size. With a small window size, the windows will overlap significantly creating a large amount of redundant data and with large windows, the overlapping proportion will be small resulting in missing out important data. Under these conditions, Gouda and el. found an optimal window size of 40ms. This explanation relating the optimum window size with the overlapping proportion is not really applicable to this work because it is kept constant in our work and still, performance variations are observed.



(a) Accuracy vs. model size when modifying the window size



(b) Accuracy vs. Complexity when modifying the window size

Figure 4.13: Impact of the window size on the accuracy, number of operations and parameters for the CNN network

4.4 Mel-spectrogram feature

The purpose of this section is to explore the potential benefit of using the mel-spectrogram instead of the spectrogram as input feature. As explained in section 2.2.2, the mel-spectrogram is supposed to perform better because it better mimics the human hearing system. The size of the mel-spectrogram obtained is 120x101 and in the CNN, the convolutional layers have been adapted accordingly. The first layer contains 64 filters of size 95x20 and the second one contains 64 filters of size 4x10. The number of operations required for the mel-spectrogram and for the spectrogram model are respectively of 263.6 MFLOPs and 372.8 MFLOPs. The model with the spectrogram is made out of 496 256 parameters and the mel-spectrogram network uses 439 680 parameters. The mel-spectrogram has also been used with the 1-layer LSTM network but the model never converges. When using the spectrogram, the LSTM network already suffers from larger instabilities in comparison to the CNN as seen in Figs. 4.9. This can explain why using the mel-spectrogram, which induces instabilities, combined with the 1-layer LSTM can lead to a network divergence.

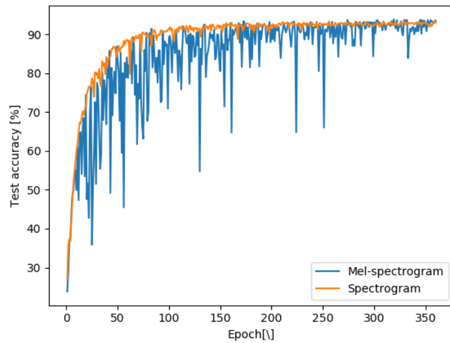


Figure 4.14: The instabilities are much larger during the learning when using the mel-spectrogram.

Figure 4.14 compares the test accuracy during the learning when using the mel-spectrogram and the spectrogram with the CNN network. The instabilities during the learning are much larger when using mel-spectrograms but the obtained accuracies are similar. The maximum accuracy obtained with the spectrogram is 93.6 % and with the mel-spectrogram, the networks reaches 93.9% of accuracy but with a larger variance. Therefore, it does not seem interesting to process the mel-spectrogram as it does not improve the performance.

The origin of the learning instabilities of the CNN and divergence of the 1-layer LSTM network are not clear. To determine if the presence of the class "unknown" can be responsible of the instabilities, the performance for each word class has be studied but not real inequalities have been observed. Besides, we try to obtain a convergent 1-layer LSTM by decreasing the learning rate to 0.005 and 0.001, which was by default set to 0.01 for LSTM networks. The Nesterov training algorithm was also replaced by the Adam or the classical momentum technique. Unfortunately, it was not successful.

4.5 Quantization of a LSTM network

The network quantization is a necessary step to reduce the size of the model and the number of operations to fit in a small microcontroller. In the previous sections, the models are implemented using a floating-point representation. When the weights are stored in a 32-bits floating-point format, it requires an amount of memory 2 times larger in comparison to fixed 16-bits implementation.
As part of this work, a fixed point analysis is proposed for the 1-layer LSTM network with the objective of finding the miminum bitwidth without accuracy loss compared to the floating-point implementation. More precisely, the weights and biases of the networks are quantized in the forward pass but their full resolution values are stored for the backward pass. The quantization is performed by first, bringing down the weights between -1 an 1 and then, quantizing the weights in

this range using:

$$m = 2^{n_b - 1} \tag{4.1}$$
$$w_q = \text{round}(\text{clamp}_{-m,m-1}(w \times m))/Q \tag{4.2}$$

where n_b is the number of bits, w and w_q are respectively the full resolution value and quantized value of the weights. The clamp function brings back the value in the range [-m,m-1]. The relation between the continuous and quantized values are presented in Fig. 4.15.

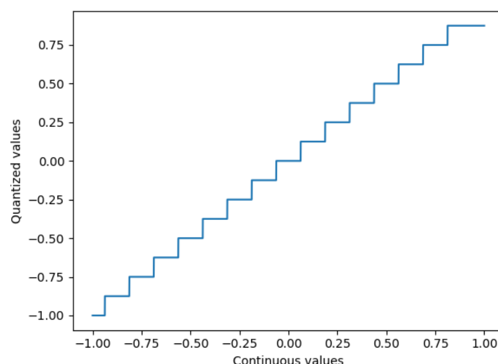
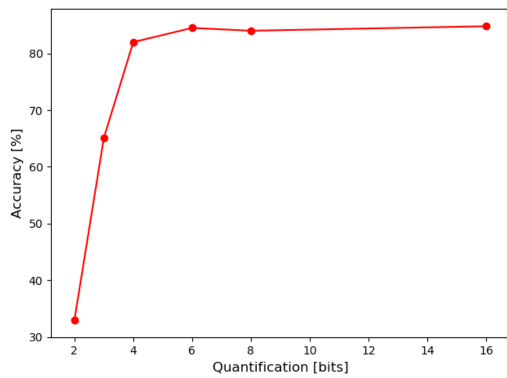
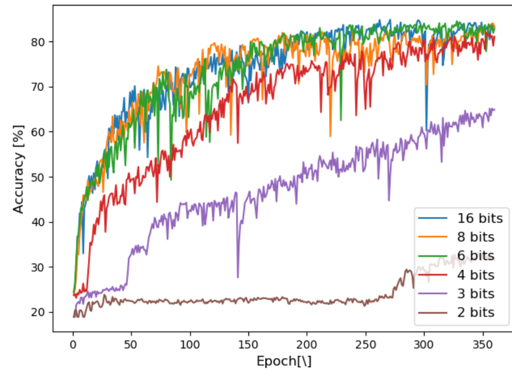


Figure 4.15: Relation between the quantized and continuous values

Concerning the hidden states and memory cells, they are reduced to 16-bits fixed point numbers. The performances for different bitwidths are presented in Figs.4.16a and 4.16b. From these graphs, we can conclude that the minimum bitwidth is equal to 6. In [13], an LSTM network is quantified using a post-training quantification, similar to ours and the minimum bitwidth is found equal to 8. This value is reduced to 4 bits using non-linear quantification. The weights distributions for a 4-bits, 8-bits and in the floating-point representation are shown in Fig. 4.17. We can see that the values of the weights are very small and centered around 0. To make better use of the bits values, we should define the range for the weights value between the weights maximum and minimum values, rather than between -1 and 1. As Figure 4.16b indicates, the rate of convergence is also impacted by the bitwidth. Reducing the number resolution slows down the learning process. The 4-bit quantization shows a slowed down start but after 150 epochs, it catches up the performance of larger bitwidth model. We can thus postulate that the accuracy drop due to the quantization can be compensated by a longer training process.

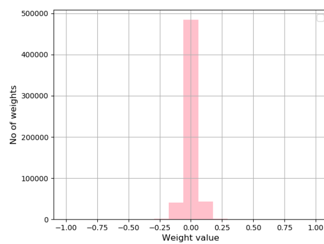


(a) Accuracy vs. bitwidth: The minimum bitwidth required is 6

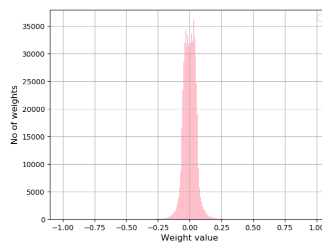


(b) Accuracy evolution during the learning process: the rate of convergence is impacted by the bitwidth.

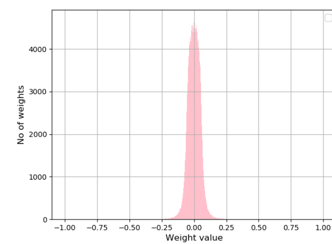
Figure 4.16: Quantified model: effect on the accuracy and rate of convergence



(a) Weights distribution for the 4-bits quantized 1-layer LSTM model



(b) Weights distribution for the 8-bits quantized 1-layer LSTM model



(c) Weights distribution for the floating-point 1-layer LSTM model

Figure 4.17: Comparison of the weights distribution in the floating-point, in the 8-bits and in the 4-bits implementation: the weights values are mainly included between -0.25 and 0.25.

Chapter 5

Conclusion

The main objective of this master thesis was to implement several types of KWS systems in order to find the best memory, accuracy and computational complexity trade-off. In particular, our KWS systems are trained to classified word utterances in one of the four predefined word classes or in the unknown word class. With the development of Internet of Things modules, implementing neural networks which have a low latency and a small memory footprint is getting essential. Therefore, we focus our interest on maximizing the accuracy while minimizing the memory space and computational complexity of the keyword spotting systems. Three different ways have been explored to either reduce the memory and compute requirements or to improve the performance.

First, different models including the LSTM, CNN and Grid LSTM are compared to spot the best model in terms of memory, accuracy and computational complexity. Globally, it comes out from this analysis that the CNN achieves the best accuracy with a reduced memory space and a limited number of operations while the 1-LSTM requires the smallest number of operations and parameters but with only 86.6% of accuracy. Concerning the Grid LSTM model, it reaches accuracy performances as good as the CNN but with a larger number of parameters and operations.

Secondly, the window size for the spectrogram computation is optimized in section 4.3 and it appears that a window size of 25ms is optimal for both the CNN and the 1-layer LSTM network using the Speech Command dataset from Google. As the optimal window matches between both networks, we try to link the window size with the word duration but no evident relation comes out. As expected, enlarging the window decreases the number of time steps and increases the input size, which also impacts the architecture of the neural networks. For that reason, we could prefer a window size of 50ms for the CNN , dividing by 2 the number of operations and parameters and only reducing by 0.5% the accuracy.

Finally, the weights and biases of the 1-layer LSTM are quantified and the activation reduced to 16-bits fixed point numbers. We found a minimum bitwidth for the weights and biases equal to 4, dividing by 8 the memory footprint in comparison to a 32-bits floating point implementation. Of course, the computational complexity is also reduced

Comparing the performances with [6] reveals that several aspects still need to be studied to reach state-of-the-art results. More complex features such as the MFCC should be considered to improve the accuracy on one hand, and reduce the number of operations and number of parameters, on the other hand. The quantification technique could be improved by defining a tighter range for the weights values or even better, by using a non-linear quantification given the normal distribution of the weights. On wider scale, the study of the speech features is very interesting and if a relation with some task characteristics is found, it could be very powerful and adaptable to any sound applications such as bird songs monitoring or sound monitoring for medical devices.

Appendix A

Histogram of the word duration

To compute the distribution of the word duration, it was automatically estimated using the following technique:

1. **Computation of the signal envelope:** The analytic signal of the signal $x(t)$ is obtained using $x_a(t) = x(t) + jx_h(t)$ where $x_h(t)$ is the Hilbert transform of $x(t)$. The amplitude envelope is the norm of $x_a(t)$.

2. **Low-pass filtering:** The signal is then, low-pass filtered using a 1-order Butterworth low-pass filter with a cut-off frequency of 100Hz.

3. **Thresholding:** The portion of the signal is denoted as active when the amplitude is larger than 5% of the maximum value of the signal. This threshold value is chosen so that the sensitivity of the word duration with respect to the threshold value is small.

The different steps are presented in Fig.A.1. Finally, to get the word duration, the sum of the thresholded signal is divided by the sampling frequency which is equal to 16kHz. As additional information, the mean and variance of the word duration is computed for each word class.

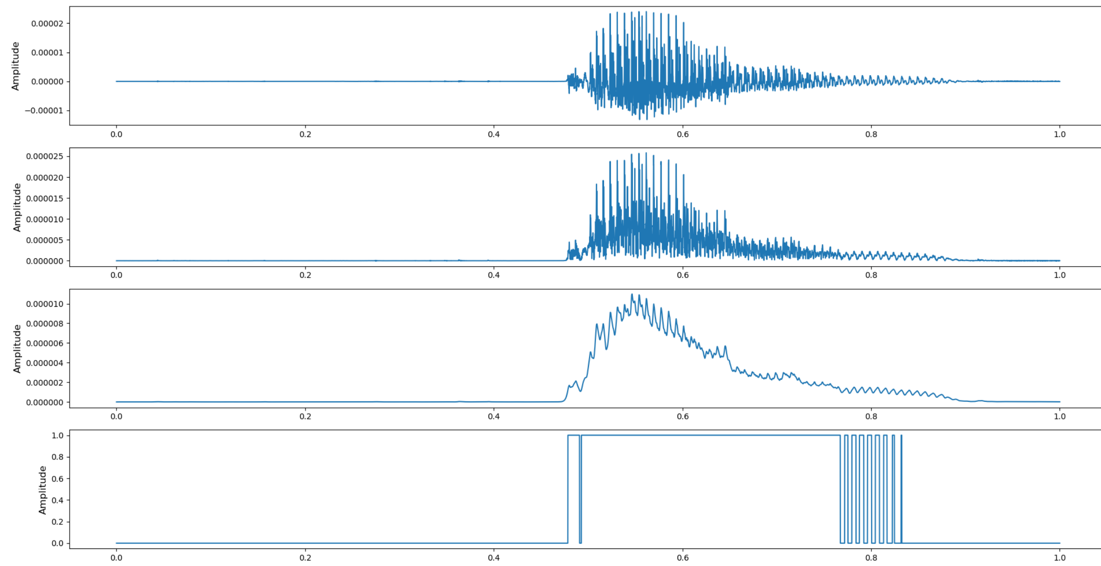
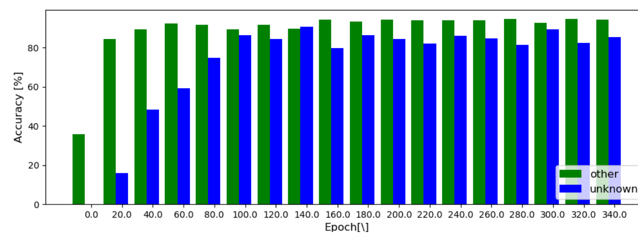


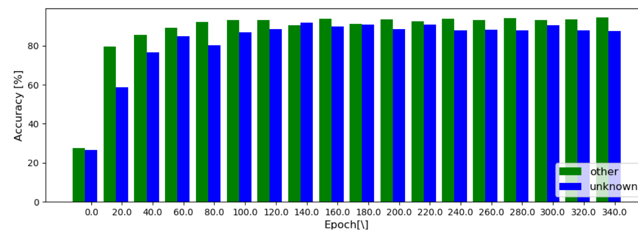
Figure A.1: Steps for the estimation of the word duration with a signal from the class 'down'. (1)Raw signal (2)Amplitude envelope (3)Filtered envelope (4)Thresholded signal(1 when the signal is active and 0 otherwise)

Appendix B

Impact of the weighted cross-entropy

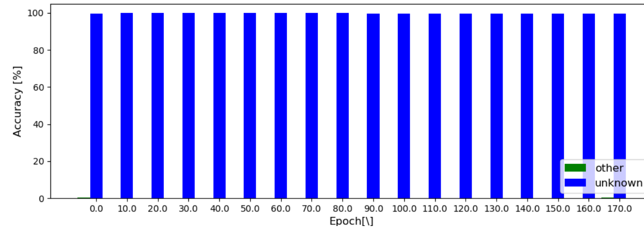


(a) No weighted cross-entropy

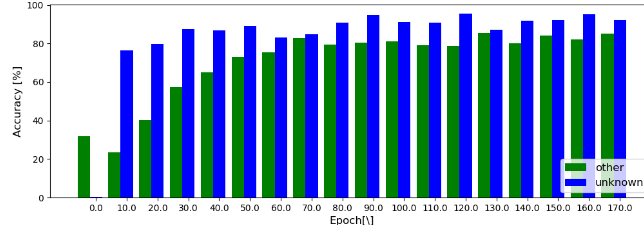


(b) Weighted cross-entropy

Figure B.1: Comparison of the performances for the class 'unknown' and the 4 other classes when the weighted cross-entropy is used and not used on the CNN network. In green, mean accuracy of the 4 classes ('up', 'left', 'right', 'down') and in blue, accuracy of the class 'unknown'.



(a) No weighted cross-entropy



(b) Weighted cross-entropy

Figure B.2: Comparison of the performances for the class 'unknown' and the 4 other classes when the weighted cross-entropy is used and not used on the 1-layer LSTM network with an unbalanced dataset (35% of predefined words and 65% of "unknown" words). In green, mean accuracy of the 4 classes ('up', 'left', 'right', 'down') and in blue, accuracy of the class 'unknown'.

References

[1] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke. The Microsoft 2017 Conversational Speech Recognition System. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 5934–5938, 2018.

[2] C Teacher, H Kellett, and L Focht. Experimental limited vocabulary speech recognizer. *IEEE Transactions on Audio and electroacoustics*, pages 127–130, 1967.

[3] Sanjay Krishna Gouda, Salil Kanetkar, David Harrison, and Manfred K Warmuth. Speech Recognition: Keyword Spotting Through Image Recognition. 2018.

[4] Tara N Sainath, Abdel-rahman Mohamed, Brian Kingsbury, Bhuvana Ramabhadran, I B M T J Watson, and Yorktown Heights. Deep convolutional neural networks for LVCSR. *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 10–14, 2013.

[5] Shilei Zhang, Wen Liu, and Yong Qin. Wake-up-word spotting using end-to-end deep neural network system. *Proceedings - International Conference on Pattern Recognition (ICPR)*, pages 2878–2883, 2016.

[6] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. Hello Edge: Keyword Spotting on Microcontrollers. pages 1–14, 2018.

[7] Jing-yun Zhang, Lu Huang, and Jia-song Sun. Keyword Spotting with Long Short-term Memory Neural Network Architectures. *International Conference on Computer, Electronics and Communication Engineering (CECE)*, pages 58–64, 2017.

[8] Sercan Ö Arık, Markus Kliegl, Rewon Child, Joel Hestness, Andrew Gibiansky, Ryan Prenger, and Adam Coates. Convolutional Recurrent Neural Networks for Small-Footprint Keyword Spotting. pages 5474–5478, 2017.

[9] Ming Sun, Anirudh Raju, George Tucker, Sankaran Panchapagesan, Gengshen Fu, Arindam Mandal, Spyros Matsoukas, Nikko Strom, and Shiv Vitaladevuni. Max-pooling loss training of long short-term memory networks for small-footprint keyword spotting. *2016 IEEE Workshop on Spoken Language Technology, SLT 2016 - Proceedings*, pages 474–480, 2017.

[10] X Li and Z Zhou. Speech Command Recognition with Convolutional Neural Network. *Stanford Journal*, 2016.

[11] Andrew Senior, Hasim Sak, and Francoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. 2014.

[12] Pete Warden. Launching the Speech commands dataset, 2017.

[13] Juan Sebastian P. Giraldo, Steven Lauwereins, Komail Badami, and Marian Verhelst. Vocell: A 65-nm Speech-Triggered Wake-Up SoC for 10-μW Keyword Spotting and Speaker Verification. *IEEE Journal of Solid-State Circuits*, 55(4):868–878, 2020.

[14] Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. Grid long short-term memory. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pages 1–15, 2016.

[15] Bert Moons, Roel Uytterhoeven, Wim Dehaene, and Marian Verhelst. Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable Convolutional Neural Network processor in 28nm FDSOI. *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, 60:246–247, 2017.

[16] Guoguo Chen, Carolina Parada, and Tara N. Sainath. Query-by-example keyword spotting using long short-term memory networks. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 5236–5240, 2015.

[17] Mohit Shah, Jingcheng Wang, David Blaauw, Dennis Sylvester, Hun-Seok Kim, and Chaitali Chakrabarti. A Fixed-Point Neural Network Architecture for Speech Applications on Resource Constrained Hardware. *IEEE*, 2015.

[18] L Vandendorpe. *ELEC 2900 Traitement des signaux.*

[19] Hiroshi Shimodaira and Steve Renals. Speech Signal Analysis. In *Automatic Speech Recognition*. 2019.

[20] James Lyons, Kamil Wojcicki, and Kuldip K. Paliwal. Preference for 20-40 ms window duration in speech analysis. *IEEE Xplore*, 2011.

[21] Pasi Pertila. Mel-frequency cepstral coefficients (MFCCs) and gammatone filter banks. In *SGN[U+2010]14006 Audio and Speech Processing*. 2015.

[22] Kunihiko Fukushima. Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics*, 36:193 202, 1980.

[23] Yann Lecun, B. Boser, J. S. Denker, D. Henderson, and R. E. Howard. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1989.

[24] Yann Lecun, Leon Bottou, Yoshua Bengio, and Patrick Ha. Gradient-Based Learning Applied to Document Recognition. *Proceedings - IEEE*, (November):1–46, 1998.

[25] Alex Krizhevsky and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, pages 1–9, 2012.

[26] Dan Ciresan, Ueli Meier, and Jonathan Masci. Multi-Column Deep Neural Network for Traffic Sign Classification. *Neural Networks*, 2012.

[27] Ian J Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. *arXiv preprint*, pages 1–13, 2014.

[28] Pierre Sermanet, Soumith Chintala, and Yann Lecun. Convolutional Neural Networks Applied to House Numbers Digit Classification. *Proceedings of the 21th International Conference on Pattern recognition (ICPR)*, pages 3288–3291, 2012.

[29] Dan Ciresan, Ueli Meier, and Jurgen Schmidhuber. Multi-column Deep Neural Networks for Image Classification. *Proceedings of the IEEE Conference on Computer Vision and pattern recognition, IEEE*, pages 3642–3649, 2012.

[30] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. Pedestrian Detection with Unsupervised Multi-Stage Feature Learning. *Proceedings of the IEEE Conference on Computer Vision and pattern recognition, IEEE*, pages 3626–3633, 2013.

[31] Dan C. Ciresan, Alessandro Giusti, Luca M. Gambardella, and Jurgen Schmidhuber. Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images. *Advances in Neural Information Processing Systems*, pages 1–9, 2012.

[32] Li Deng, Ossama Abdel-hamid, and Dong Yu. A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion. *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2013.

[33] Honglak Lee, Peter Pham, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, pages 1096–1104, 2009.

[34] Ossama Abdel-hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional Neural Networks for Speech Recognition. *IEEE/ACM TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING*, 22(10):1533–1545, 2014.

[35] Laszlo Toth. Combining time- and frequency-domain convolution in convolutional neural network-based phone recognition. *International Conference on Acoustics, Speech and Signal Processing*, 2015.

[36] Google Brain. TensorFlow Speech Recognition Challenge, 2018.

[37] Sumit Saha. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way, 2018.

[38] Facundo Bre, Juan M. Gimenez, and Víctor D. Fachinotti. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, pages 1429–1441, 2017.

[39] Pathmind. A Beginner's Guide to Neural Networks and Deep Learning.

[40] Ihab S. Mohamed. *Detection and Tracking of Pallets using a Laser Rangefinder and Machine Learning Techniques*. PhD thesis, 2017.

[41] Mc.ai. Deep Learning Cage Match: Max Pooling vs Convolutions.

[42] Tomaš Mikolov. Recurrent neural network based language model. Number January, pages 1045–1048, 2010.

[43] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2016.

[44] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 4:3104–3112, 2014.

[45] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 1724–1734, 2014.

[46] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[47] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficukty of training recurrent neural networks. (2):41–71, 2013.

[48] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[49] L K Halderman and C Chiarello. Framewise phoneme classification with bidirectional {LSTM} and other neural network architectures. *Neural Networks*, 18:602–610, 2005.

[50] Zhang Ruochi. RNN Implementation, 2019.

[51] Abigail See. Lecture 7: Vanishing gradients and fancy RNNs. In *Natural Language Processing with Deep Learning CS224N/Ling284*. 2019.

[52] Andrews Ng. Why sequence models? In *Recurrent Neural Networks*.

[53] Alex Graves, Abdel Rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, (3):6645–6649, 2013.

[54] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on signal processing*, 45:2673–2681, 1997.

[55] Pierre Baldi, Paolo Frasconi, and Gianluca Pollastri. Exploiting the past and the future in proyein secondary structure prediction. *Bioinformatics*, 15:937–946, 1999.

[56] Pierre Baldi, Søren Brunak, Paolo Frasconi, and Gianluca Pollastri. Bidirectional dynamics for protein secondary structure prediction. *Lecture Notes in Computer Science*, pages 80–104, 2001.

[57] Chen Jinmiao and Chaudhari Narendra. Capturing Long-Term Dependencies for Protein Secondary Structure Prediction. *International Symposium on Neural Networks*, 3174, 2004.

[58] Michael Schuster. *On supervised learning from sequential data with applications for speech recognition*. PhD thesis, 1999.

[59] Özal Yildirim. A novel wavelet sequences based on deep bidirectional LSTM network model for ECG signal classification. *Computers in Biology and Medicine*, 2018.

[60] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with deep bidirectional LSTM. pages 273–278, 2013.

[61] Yu Kang Jia, Zhicheng Wu, Yanyan Xu, Dengfeng Ke, and Kaile Su. Long Short-Term Memory Projection Recurrent Neural Network Architectures for Piano's Continuous Note Recognition. *Journal of Robotics*, pages 13–16, 2017.

[62] F. Y. Putri, D. Puji Lestari, and D. H. Widyantoro. Long Short-Term Memory Based Language Model for Indonesian Spontaneous Speech Recognition. *International Conference on Computer, Control, Informatics and its Applications (IC3INA)*, pages 44–48, 2018.

[63] Andrew Ng and Kian Katanforoosh. Deep Learning. In *CS229 Lecture notes*, pages 1–30. 2000.

[64] David Rumelhart, Geoffrey Hinton, and Ronald Williams. Learning representations by back-propagating errors. *Nature*, pages 533–536, 1986.

[65] Yuanrui Dong. Hyper Parameter - Momentum, 2019.

[66] Y. Nesterov. A method for solving the convex programming problem with convergence rate O(1/kˆ2). *CCCP*, 27:543–547, 1983.

[67] Geoffrey Hinton, Srivastava Nitish, and Kevin Swersky. Lecture 6a: Overview of mini[U+2010]batch gradient descent. In *Neural Networks for Machine Learning*. 2017.

[68] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imageNet classification. *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[69] Tara N. Sainath and Carolina Parada. Convolutional recurrent neural networks for small-footprint keyword spotting. *INTERSPEECH*, pages 1606–1610, 2015.

[70] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015*, 1:448–456, 2015.

[71] Tara N. Sainath, Brian Kingsbury, Vikas Sindhwani, and Ebru Arisoy. Low-rank matrix factorization for deep neural network training with dimensional output targets. *IEEE*, pages 6655–6659, 2013.

[72] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[73] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015*, 2017.

[74] Matthijs Hollemans. How fast is my model?, 2018.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**