Final Project: Introduction to Speech Processing

Ilan Benhamou 330139072, Benjamin Maman 3411145811

August 24, 2023

Our code and trained models ara availables at `https://github.com/benm97/SpeechProject`

1 Model: Encoder/Decoder with CTC

1.1 Overview

We implement an architecture inspired by [1]. This model employs the Encoder/Decoder pattern and consists of multiple blocks connected through residual links. Every block has modules that include batch normalization and ReLU layers. The training process utilizes CTC loss.

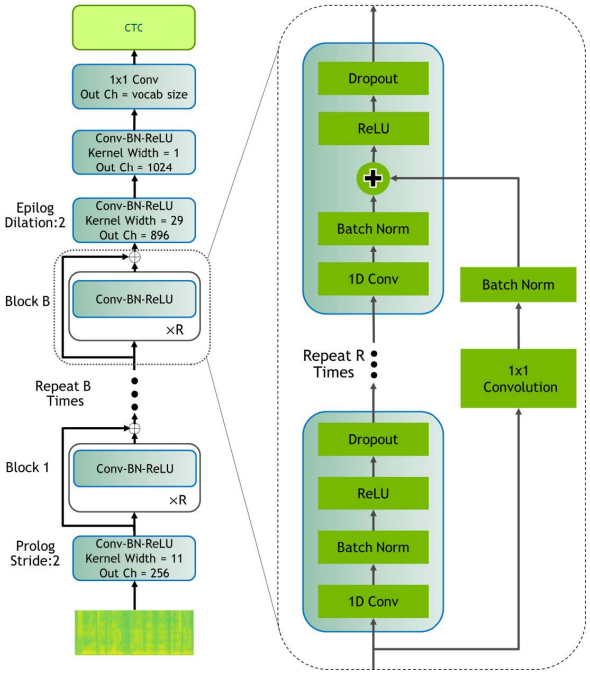


Figure 1: Jasper Architecture

1.2 Description

1.2.1 Architecture:

The encoder is composed of 4 stacked Conv-BN-ReLU blocks with kernel size of, respectively, [1,11,15,19]. Each Conv-BN-ReLU block consist of:

1. a 1D convolutional layer
2. a normalization layer
3. ReLU

4. Dropout

A residual connection links the end to the input. The decoder is a 1D pointwise conv (fully connected).

1.2.2 Optimization

Data augmentation: We check for improvements using data augmentation. We use implementation proposed by [2] that works with 3 different augmentations:

- Time Warping: On the log mel spectrogram, a random point in time is selected and shifted, creating a distortion in the time direction. This simulates variations in speech speed or tempo.
- Frequency Masking: A sequence of mel frequency channels is obscured, simulating the loss of certain frequency information.
- Time Masking: Consecutive time steps in the spectrogram are masked out, representing a loss of a segment of speech.

Decoding: We use greedy decoding, see last section for possible optimization.
Features: We extract 64 MFCCs features. Although we thought, considering the simplicity of the audios, it would help to select a smaller number of features, we didn't see significant improvements.

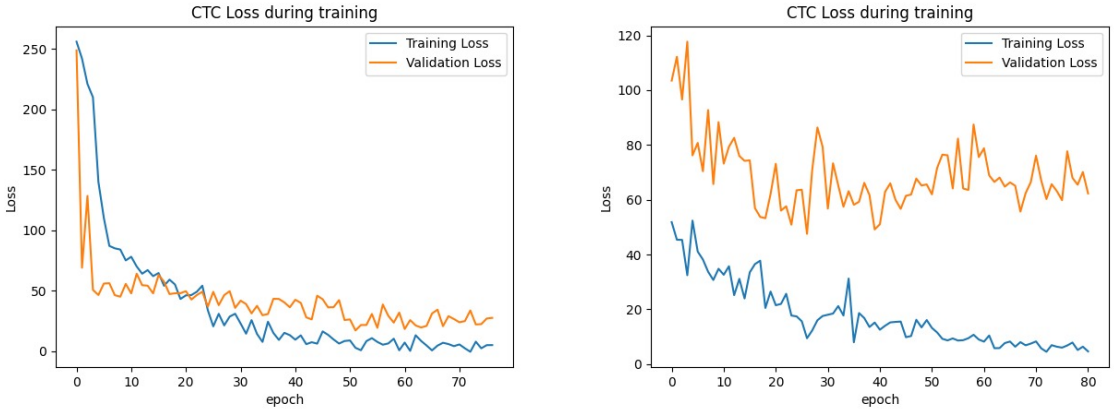
Parameter	Value
N° MFCCs	64
Loss function	CTC Loss
Learning rate scheduler	CosineAnnealing
Start learning rate	0.001
Dropout	0.3
Batch size	32
Epochs	130

Table 1: Parameters of our model

1.3 Results

Model	CER	WER
With data augmentation	0.2380	0.4941
Without data augmentation	0.6514	0.8732

Table 2: Word/Character Error Rate on test set



(a) With data augmentation

(b) Without data augmentation

Figure 2: Loss function during training

1.4 Conclusion

With a CER of **0.24** and a WER of **0.49** this model gave us better result than most of our tries. With his big number of weights (2.4M) compared to the small dataset we can expect for over-fitting. In fact in Figure 2b we can see that when epochs increase, the val loss become much higher than the train loss, and training for more epochs worsen the overfit. The use of data augmentation, that increase artificially the number of samples, give us way better accuracy. Even after 130 epochs, the WER continues to slowly decrease and it seems that training it for a very long time may improve it.

2 Model: Encoder/Decoder with CTC improved (QuartzNET)

2.1 Overview

To improve the first model, we use an architecture proposed by [3]. Like the precedent, this model comprises several blocks linked by residual connections. Each block contains modules with batch normalization, and ReLU layers, and training is done using CTC loss. The main difference that it uses **time-channel separable** 1D convolutions. In Jasper we used a classical 1D Conv Layer which operates on features and time frame at the same time. In comparison, 1D timechannel separable convolutions can be separated into a 1D depthwise convolutional layer with kernel length K that operates on each channel (Mel features) individually but across K time frames and a pointwise convolutional layer that operates on each time frame independently but across all channels.

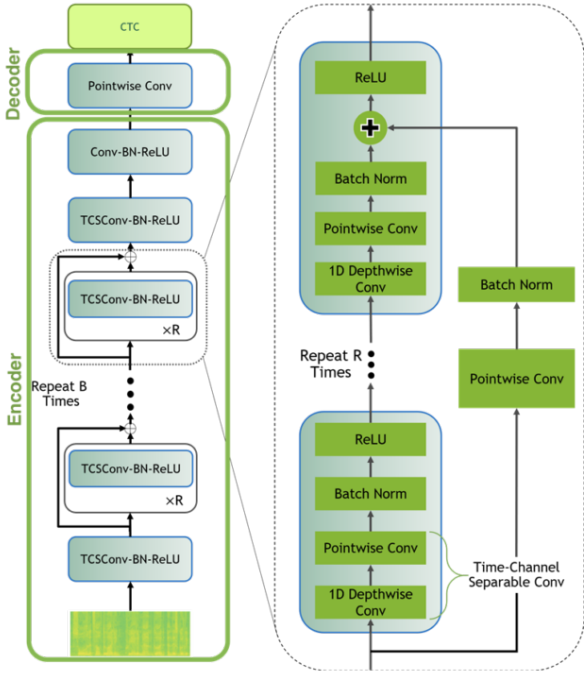


Figure 3: QuartzNET Architecture

2.2 Description

2.2.1 Architecture:

We use an encoder/decoder architecture. The encoder is composed of 6 stacked TSConv-BN-ReLU blocks with kernel size of, respectively, [11,13,15,17,19,21]. Each TSConv-BN-ReLU block consist of:

1. a depthwise convolutional layer
2. a pointwise convolution

3. a normalization layer
4. ReLU

Moreover, a residual connection is sending the output of (3). The decoder is a 1D pointwise conv (fully connected).

2.2.2 Optimization

Data augmentation: We check for improvements using data augmentation, in the same way that in the first model.
Decoding: We use greedy decoding, see last section for possible optimization.
Dropout: We start without dropout. When we try to increase number of epochs, an overfit appears slightly, then we try a 0.3 dropout config to reduce it.
Features: We extract 64 MFCCs features. Although we thought, considering the simplicity of the audios, it would help to select a smaller number of features, we didn't see significant improvements.

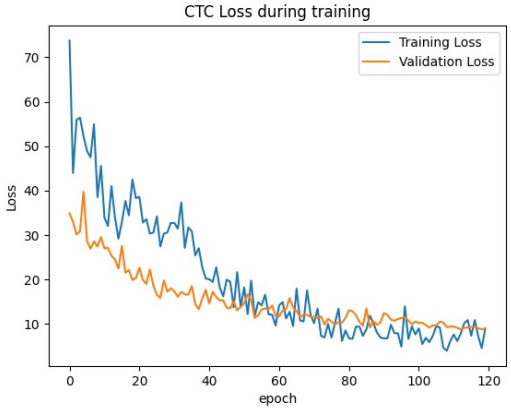
Parameter	Value
N° MFCCs	64
Loss function	CTC Loss
Learning rate scheduler	CosineAnnealing
Start learning rate	0.01
Dropout	0.0/0.3
Batch size	32
Epochs	80

Table 3: Parameters of our QuartzNET implementation

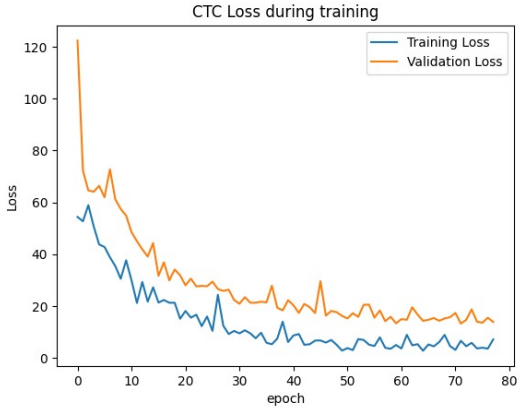
2.3 Results

Model	CER	WER
Without data augmentation (Dropout=0)	0.2503	0.5705
With data augmentation (Dropout=0)	0.1580	0.3557
With data augmentation (Dropout=0.3)	0.0892	0.2328

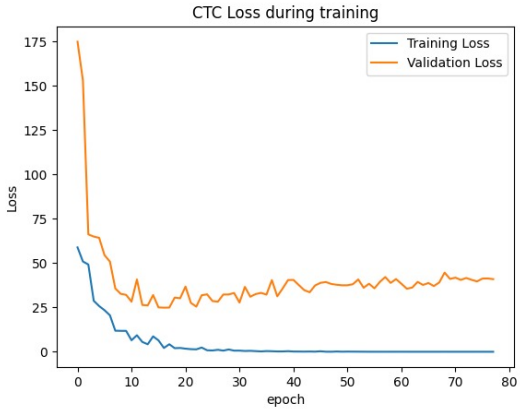
Table 4: Word/Character Error Rate on test set



(a) With data augmentation (Dropout=0.3)



(b) With data augmentation (Dropout=0)



(c) Without data augmentation (Dropout=0)

Figure 4: Loss function during training

2.4 Conclusion

The model give us best accuracy with a CER of **0.09** and a WER of **0.23**. It has 1.2M weights compared to 2.4M of the precedent one, allowing faster convergence, so it gives better accuracy in less steps. Moreover, it seems that less weight allow the model to better generalize unseen data.
It's also clear that the small size of the dataset lead to overfit of our model, as we can see in Figure 4c the *training loss* decrease very fast when the *validation loss* increase, leading to a big gap between them. Data augmentation solve this, *training loss* and *validation loss* stay correlated (Figure 4b), giving better CER/WER.
Finally, adding a 30% dropout allows us to train the model for more epochs with no overfit. As we can see in figure 4a, the val loss and the train loss continue to decrease together, and are closer than in any other configurations, resulting in a better accuracy on test data.

3 Model: LAS

3.1 Overview

LAS (Listen, Attend, and Spell) is an end-to-end model, presented in [4], that maps an audio sequence to a text sequence directly without the need for alignment. It consists of two main parts: an encoder (the "Listen" part) and a decoder with attention (the "Attend and Spell" part). Let's explain each module we use.

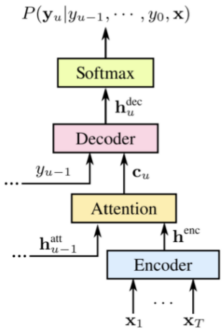


Figure 5: LAS Architecture

3.2 Description

3.2.1 Architecture

- **Encoder:** Processes the input sequence (mfccs from audio) into a higher-level representation using LSTM, a type of RNN that captures long-term sequence dependencies. The outputs serve the attention mechanism.
- **Attention:** Computes a context from encoder outputs using a weighted sum. Attention scores, determined by a feed-forward network with tanh activation, dictate the weights. These scores, after softmax application, yield attention weights, which in turn produce the context vector.
- **Decoder:** Generates the output sequence (e.g., characters) from attention-derived context vectors. Utilizing an LSTMCell, it processes one step at a time, taking the previous output, hidden state, and context as inputs. The LSTMCell's output goes through a connected layer for the current step's output.
- **LAS:** The primary model combining encoder, attention, and decoder. It processes input via the encoder, initializes decoder states to zero, and for each decoder input step, computes the context using attention. The context and current decoder input merge and pass to the decoder. Outputs from all steps combine for the final sequence. In summary, the LAS model processes the input sequence using the encoder, then generates the output sequence using the decoder with the help of the attention mechanism to focus on different parts of the input sequence at each time step.

3.2.2 Optimization

- Data augmentation: We test configuration with augmented data by adding noise, shifting audio and changing pitch.
- Decoding: We use greedy decoding, see last section for possible optimization.
- Features: We extract 39 MFCCs features.

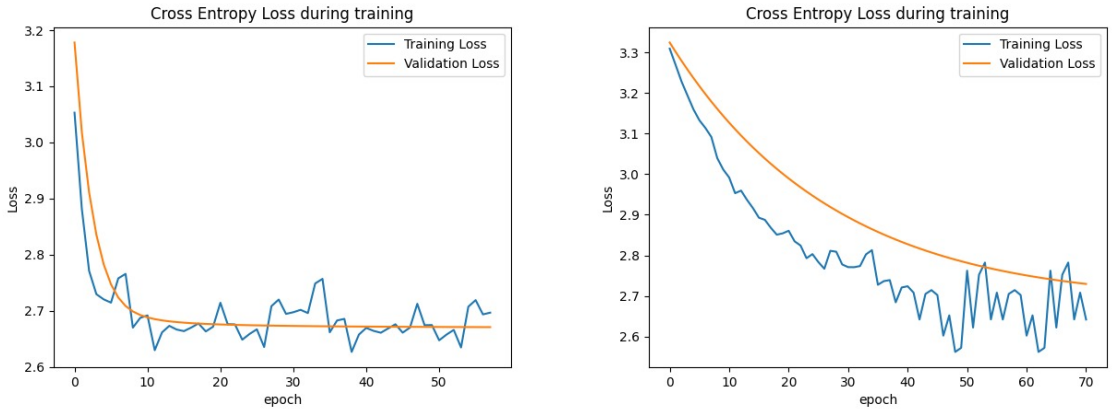
Parameter	Value
N° MFCCs	39
Loss function	Cross entropy
Learning rate scheduler	None
Learning rate	0.002
Dropout	0.0
Batch size	32
Epochs	80

Table 5: Parameters of our LAS implementation

3.3 Results

Model	CER	WER
With data augmentation	0.7012	0.8612
Without data augmentation	0.7993	0.9187

Table 6: Word/Character Error Rate on test set



(a) With data augmentation

(b) Without data augmentation

Figure 6: Loss function during training

3.4 Conclusion

Even if the data augmentation give better correlation between train loss and val loss, it's seems from figure 6b that overfit wasn't an issue. It's not clear why this model gave us such a bad accuracy. We thought that's may be because the decoder predict a char, based on the precedent predicted one. One error can lead to "chain reaction" decoding all the sequence in a bad way.

4 Improved decoding with Beam Search and Language Model

4.1 Description

For now we used Greedy Decoding: At each step, it selects the token with the highest probability as the next character of the sequence. Beam search is a heuristic search algorithm. At each step, instead of choosing just one token, it maintains a set number of the most probable sequences (or "beams"). It then expands each of these sequences with possible next tokens and keeps only the top sequences based on their cumulative scores. We build a simple 3-gram language model using KenLm toolkit to compute sequence probability during the beam search decoding. The LM was trained on our AN4 dataset and we didn't use any other corpus to respect the constraint of the exercise.

4.2 Results

Model	Greedy		Beam search	
	CER	WER	CER	WER
First Model - without augmentation	0.6514	0.8732	0.6231	0.8450
First Model - with augmentation	0.2380	0.4941	0.2178	0.4721
QuartzNET - without augmentation	0.2503	0.5705	0.2432	0.5293
QuartzNET - with augmentation	0.1580	0.3557	0.1312	0.3498
QuartzNET - with augmentation & dropout	0.0892	0.2328	0.0921	0.2407
LAS - without augmentation	0.7993	0.9187	0.7643	0.8704
LAS - with augmentation	0.7012	0.8612	0.6685	0.8355

Table 7: Influence of beam search decoding on Word/Character Error Rate.

4.3 Conclusion

We see that decoding with Beam search instead of greedy enhances accuracy for almost every model. However the improvement is not as significant as expected from the literature. Thus, we didn't expect much of the LM due do the simplicity of our dataset which is composed of words with no apparent link between them.

References

[1] Jasper: An End-to-End Convolutional Neural Acoustic Mode

[2] SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition

[3] QuartzNet: Deep Automatic Speech Recognition with 1D Time-Channel Separable Convolutions

[4] Listen, Attend and Spell