# AI4EU

## WP7.3 RA-7.3.e

**PM6 (baseline)**

**Version 0.1: 2019.06.25**

**Budapest University of Technology and Economics (BME)**

# Contents

# Introduction

One of the novel neural network based TTS methods is Tacotron2 which is quite flexible as an adaptive TTS system. It is the first component of an end-to-end system, but it is tested and published by the authors with 22kHz training dataset. In order to use this system as a TTS component in a real environment, we started to optimize it. Two different goals were set at the beginning: Firstly to reach at least real-time or faster system, and secondly to determine a resource efficient training environment.

# End-to-end Text-to-Speech systems

A state-of-the-art end-to-end Text-to-Speech system may consist of the combination of Tacotron2 and a WaveNet based vocoder. In our experiments we used Tacotron2 and the WaveGlow model which combines the advantages of the Flow and the WaveNet models..

## Tacotron2

Tacotron2 is a fully neural network based solution which is based on a sequence-to-sequence model. There is an encoder part which is responsible for the processing of input text or phoneme sequence. The other component is the decoder with a postnet which generates the Mel spectrum output sequence. The connection between the encoder and the decoder is ensured by the attention mechanism.

The system is described in detail in (Jonathan Shen, 2017): Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, RJ Skerry-Ryan, Rif A. Saurous, Yannis Agiomyrgiannakis, Yonghui Wu: Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions, https://arxiv.org/abs/1712.05884

## WaveGlow

WaveGlow is a model which can generate speech from a Mel spectrum sequence. This model can generate high quality speech, it reaches better MOS (mean opinion score) values than the original WaveNet solution.

The system is described in detail in (Ryan Prenger, 2018): Ryan Prenger, Rafael Valle, Bryan Catanzaro: WaveGlow: A Flow-based Generative Network for Speech Synthesis, https://arxiv.org/abs/1811.00002

# Measurements of Tacotron2 + WaveGlow inference time on different resources.

Deep learning based end-to-end systems require a lot of calculations, the real-time operation is not trivial. The speed depends on the model architecture, the size of the model, the text and the runtime environment. There were three different environments where we synthesized a short text, and calculated the real-time factor.

## Description of systems

There were three systems, a small and a big desktop system and a server one. The oldest GPU is based on the NVIDIA® Maxwell™ architecture, the other one on the Pascal™ architecture and in the server machine the GPU uses the Volta™ architecture. (The current desktop Turing™ architecture is quite similar to the Volta™ architecture in performance).

*Table 1: Small desktop system*

| Name | Small desktop system |
|---|---|
| CPU | Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz |
| System Memory | 8 Gbyte |
| GPU | NVIDIA Titan Xp |
| GPU Memory | 12 Gbyte |
| OS | Ubuntu 16.04.6 LTS |

*Table 2: Big desktop system*

| Name | Big desktop system |
|---|---|
| CPU | Intel(R) Core(TM) i7-6850K CPU @ 3.60GHz |
| System Memory | 128 Gbyte |
| GPU1 | NVIDIA Titan Xp |
| GPU1 Memory | 12 Gbyte |
| GPU2 | NVIDIA Titan X |
| GPU2 Memory | 12 Gbyte |
| GPU3 | NVIDIA Geforce 970 |
| GPU3 Memory | 4 Gbyte |
| OS | Ubuntu 16.04.6 LTS |

*Table 3: Server system*

| Name | server system |
|---|---|
| CPU | Intel(R) Xeon(R) Platinum 8167M CPU @ 2.00GHz |
| System Memory | 128 Gbyte |
| GPU1 | NVIDIA V100 |
| GPU1 Memory | 16 Gbyte |
| OS | Ubuntu 16.04.6 LTS |

## Test set

The test set was a short fable: The North Wind and the Sun. The Hungarian translation was used in the test. The fable contains 9 sentences. The length of the synthesized audio file is 53.34 sec.

## Model parameters

### Tacotron2

During the test the source code from https://github.com/NVIDIA/tacotron2 was used. The main parameters are shown in Table 4.

*Table 4:Tacotron2 default parameters*

| Name of parameter | Value of parameter |
|---|---|
| sampling_rate | 22050 |
| filter_length | 1024 |
| hop_length | 256 |
| win_length | 1024 |
| n_mel_channels | 80 |
| mel_fmin | 0.0 |
| mel_fmax | 8000.0 |

The model was trained with Hungarian sentences, the input was phoneme based.

### WaveGlow

During the test the source from https://github.com/NVIDIA/waveglow was used. The main parameters are shown in Table 5.

*Table 5: WaveGlow default paramters*

| Name of parameter | Value of parameter |
|---|---|
| n_mel_channels | 80 |
| n_flows | 12 |
| n_group | 8 |
| n_early_every | 4 |
| n_early_size | 2 |
| WN_config | |
| n_layers | 8 |
| n_channels | 512 |
| kernel_size | 3 |

The speed of the WaveGlow model is not sensitive for the training, the inference time depends on the model parameters.

## Measurements

The test code was written in python, the pytorch deep learning framework system was used. During the test the text was read from a file, and the synthesized speech was also written to files. The loading of model was not included in the measurement. Because pytorch is asynchronous, the torch.cuda.synchronize() function was used for getting proper timing values.

## Results

The summarized results are shown in Table 6. The first column contains the sample rate (green: small desktop system, orange: big desktop system, pink: server system), the second column shows the system under test. The main parameters of the test system are given above. The CPU and GPU columns show the processor type. The floating-point precisions column is 16 or 32 bits. There are some rows where no GPU was used, in these cases 32 bit floating point precision was applied because the CPU implementation did not support 16 bit alternatives.

The fulltime column shows the generation time of all 9 sentences in seconds. The summary time of the generated speech is the same, all systems generate 53 second speech audio.

The real-time(RT) factor column shows how many seconds of speech is generated during 1 second runtime. The larger number is better, the 1.0 value shows the real-time operation (pink: very low, orange: low, yellow: near rel-time, green: better than real-time RT factor). The synth time of 1 sec column shows the inverted RT factor, how many seconds generation time is required for 1 second speech.

*Table 6: Results of speed test*

| Sample rate | System | CPU | GPU | floating point precision | SW version | Full time | Lenght of wave | RT factor | synth. time of 1s | comment |
|---|---|---|---|---|---|---|---|---|---|---|
| [Hz] | | | | [bits] | | [s] | [s] | | [s] | [*] |
| 22050 | small desktop system | Intel® Core™ i7-2600K CPU @ 3.40GHz | ------ | 32 | Torch 1.0.0 | 1032 | 53.34 | 0.052 | 19.348 | CPU implementation allows only 32 bit floating point precision. 16 bit fp is not implemented. |
| 22050 | small desktop system | Intel® Core™ i7-2600K CPU @ 3.40GHz | Titan Xp | 32 | Torch 1.0.0 | 161 | 53.34 | 0.331 | 3.018 | |
| 22050 | big desktop system | Intel® Core™ i7-6850K CPU @ 3.60GHz | Titan Xp | **16*** | Torch 1.0.1 | 154 | 53.34 | 0.346 | 2.887 | Mixed fp precision, Waveglow 16 bits, Tacotron2 32 bits |
| 22050 | big desktop system | Intel® Core™ i7-6850K CPU @ 3.60GHz | ------ | 32 | Torch 1.0.1 | 336 | 53.34 | 0.159 | 6.299 | |
| 22050 | big desktop system | Intel® Core™ i7-6850K CPU @ 3.60GHz | Titan Xp | 32 | Torch 1.0.1 | 163 | 53.34 | 0.327 | 3.056 | |
| 22050 | big desktop system | Intel® Core™ i7-6850K CPU @ 3.60GHz | Titan X | 32 | Torch 1.0.1 | 281 | 53.34 | 0.190 | 5.268 | |
| 22050 | big desktop system | Intel® Core™ i7-6850K CPU @ 3.60GHz | 970 | 32 | Torch 1.0.1 | 1628* | 53.34 | 0.033 | 30.521 | Estimated time based on the first 4 sentences. The 970 GPU has not enough memory. |
| 8000* | big desktop system | Intel® Core™ i7-6850K CPU @ 3.60GHz | Titan Xp | 32 | Torch 1.0.1 | 60 | 53.34 | 0.889 | 1.125 | Simulated 8kHz. Tacotron2 worked on 22k. The output of Tacotron2 was reduced to 62 channels and the frames were resampled with 0.75 factor. Hop size was 128 instead of the original 256. |
| 8000 | small desktop system | Intel® Core™ i7-2600K CPU @ 3.40GHz | Titan Xp | 32 | Torch 1.0.0 | 59 | 53.34 | 0.904 | 1.106 | |

| Sample rate | System | CPU | GPU | floating point precision | SW version | Full time | Lenght of wave | RT factor | synth. time of 1s | comment |
|---|---|---|---|---|---|---|---|---|---|---|
| [Hz] | | | | [bits] | | [s] | [s] | | [s] | [*] |
| 8000 | server system | Intel® Xeon® Platinum 8167M CPU @ 2.00GHz | V100 | 32 | Torch 1.0.1.post2 | 10 | 53.34 | 5.334 | 0.187 | |
| 8000 | server system | Intel® Xeon® Platinum 8167M CPU @ 2.00GHz | V100 | **16*** | Torch 1.0.1.post2 | 8.6 | 53.34 | 6.202 | 0.161 | Only WaveGlow ran on 16 bits. |
| 22050 | server system | Intel® Xeon® Platinum 8167M CPU @ 2.00GHz | V100 | 32 | Torch 1.0.1.post2 | 16.5 | 53.34 | 3.233 | 0.309 | |
| 22050* | server system | Intel® Xeon® Platinum 8167M CPU @ 2.00GHz | ------ | 32 | Torch 1.0.1.post2 | 189 | 53.34 | 0.282 | 3.543 | CPU load was about 1000% (about 10 cores) |

The fastest environment was the system with V100 GPU. On this GPU the system was faster than real-time both on 22kHz and 8kHz and also on 32 and 16 bits. The older GPUs do not support some matrix operations, which cause the slower than real-time operation. We compared the V100 speed to an NVIDIA RTX 2080 GPU and had similar values. Coloring of RT factor: red: slowest/yellow: realtime/green: fastest

## Increased speed model description

There are several collaborative AI system scenarios where the good quality is required, but the wide spectral range is not necessary, the regular telephony range is enough (300 - 3400Hz). State-of-the-art systems give solutions to 16kHz or 22kHz sample rates. Next to quality the speed of inference time is also important. The real-time speech generation in a multichannel system is currently not cost effective, so increasing the speed of models is required. For a lot of solutions the lower sample rate is acceptable, but the current end-to-end systems do not have this kind of optimized codes. Those systems are hyper-optimized for better quality.

### Description of narrow band audio

The 16kHz and 22kHz audio is well representable with 80 Mel channels with 256 hop size (16ms @ 16000Hz). The 8kHz audio requires less channels. Figure 1 shows the original 80 Mel channels. If we use the same channel number on 8kHz sample rate, the resolution is too detailed (Figure 2), the calculation time would be same for one frame.

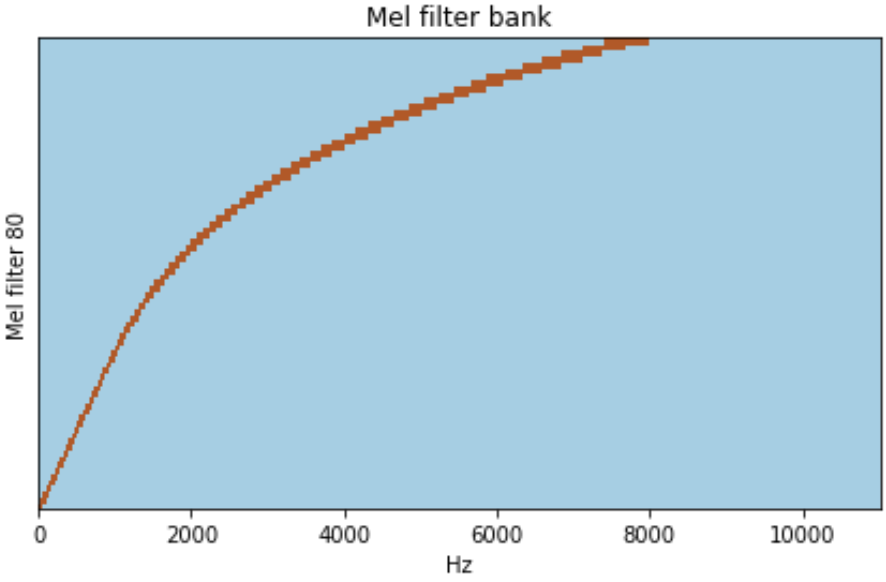


*Figure 1: 80 Mel filters 0-8000Hz*

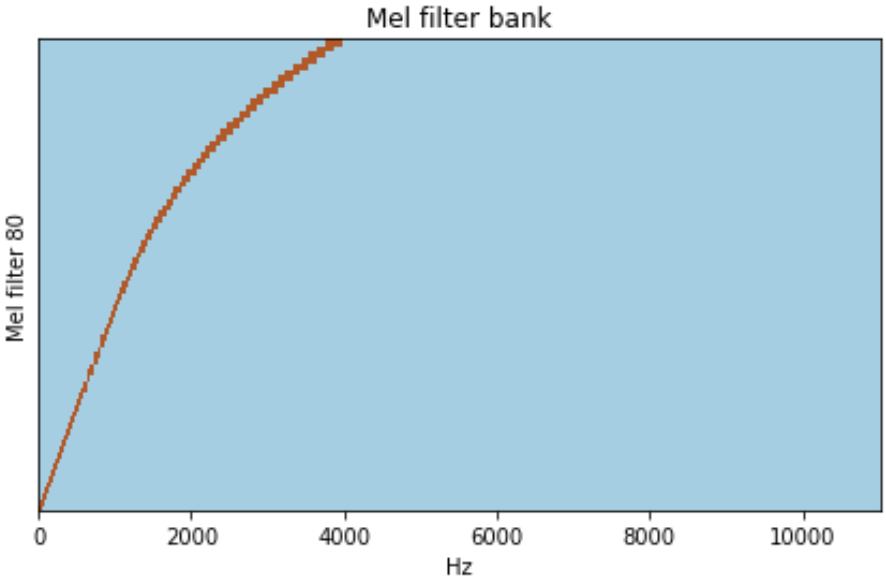


*Figure 2: 80 Mel filters 0-4000Hz*

In order to reach the same quality at 8KHz for the lower band (0-4KHz) than for 16KHz, 62 Mel channels is sufficient (Figure 3).

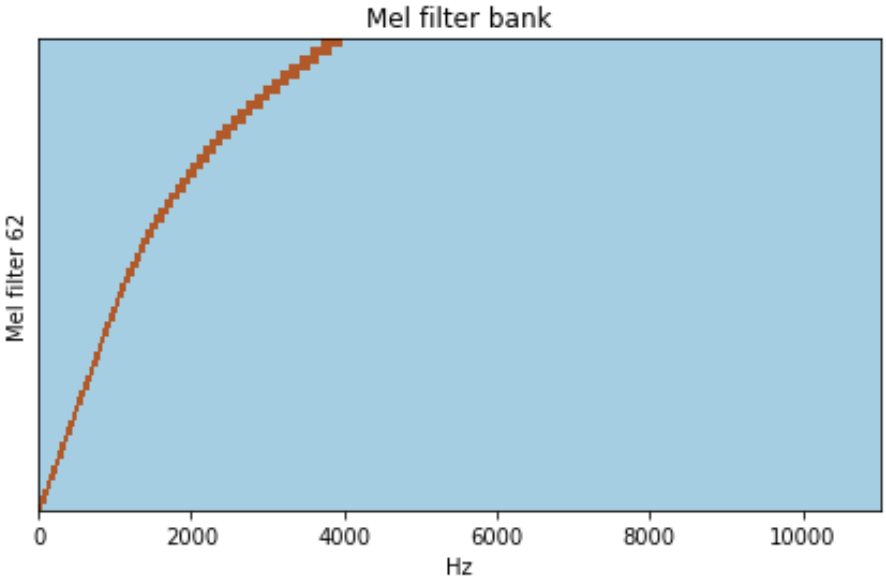


*Figure 3: 62 Mel filters 0-4000Hz*

The 62 channels are almost same than the lower 62 channels of the 80 Mel channel at 16 kHz. The relation of the channels is shown in Figure 4. The light curve shows the 80 Mel channel @ 16kHz, the dark curve shows the 64 Mel channels @ 8kHz.

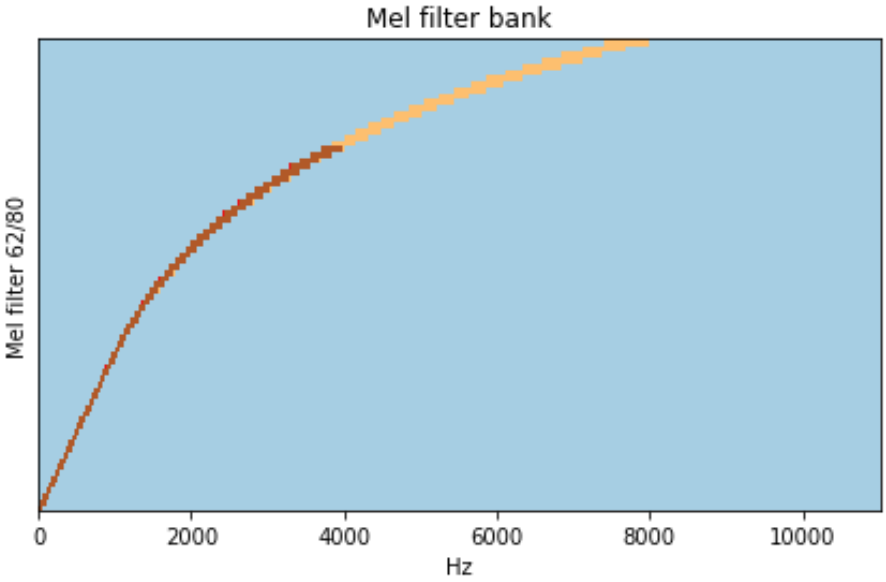


*Figure 4: Compatibility of 62 and 80 Mel filters*

## Modified WaveGlow

The WaveGlow model requires a lot of calculation. Fewer Mel channels give faster speed during training and inference time. The original WaveGlow is not prepared for different number of Mel channels, a small code modification is required. Calling the TacotronSTFT function in class Mel2Samp init function does not contain the n_mel_channels parameter. The modified code is shown at the end of this document.

The other method to increase the speed of WaveGlow inference mode is using fewer WaveNet (WN) layers. The calculation time of WN layers is the largest part of the full calculation time. The original publication contains 8 layers, but for 62 Mel channels WaveGlow with 7 layers also properly works.

The original paper of WaveGlow was published with 512 WN channels. (It is not the same as Mel channels). It is not required on 8 kHz, 256 WN channels is enough for the same quality. The original authors later published a model on 22kHz which use only 256 WN channels, so decreasing of WN channels is widely supported.

# Measurement results of training procedures of increased speed model variants

The decreased sample rate allows the reduction of the Tacotron2 model, too. The method is based on hyper parameter optimization. The source code was the same as the published code (https://github.com/NVIDIA/tacotron2). The stopping criteria is based on validation loss. Next to validation loss the attention model was also observed. Distributed training was used and the typical batch size was 80 per GPU, so there were 640 sentences. A typical training process took about 12-16 hours (on 8x V100).

## Test training environment

The test environment was a server configuration with the following main components:

- GPU: NVIDIA Tesla V100 (16 GB)
- CPU: 2.0 GHz Intel® Xeon® Platinum 8167M
- Number of GPUs: 8
- Number of CPU's cores: 52
- System memory: 768 Gbyte
- OS: Ubuntu 16.04

## Training and validation dataset

The training dataset was LJSpeech 1.1:

> From: https://keithito.com/LJ-Speech-Dataset: *„This is a public domain speech dataset consisting of 13,100 short audio clips of a single speaker reading passages from 7 non-fiction books. A transcription is provided for each clip. Clips vary in length from 1 to 10 seconds and have a total length of approximately 24 hours.*
>
> *The texts were published between 1884 and 1964, and are in the public domain. The audio was recorded in 2016-17 by the LibriVox project and is also in the public domain".*

Source of dataset: https://keithito.com/LJ-Speech-Dataset/

The dataset was cut into three pieces: training, validation and test set. We used the separation as in the tacotron2 source: https://github.com/NVIDIA/tacotron2/tree/master/filelists

## Hyperparameters

Because of big computing requirements we manually changed the hyperparameters. There were 10 different configurations and 12 training procedures. There was a configuration (ljspeech-62-d512) which was examined on three different environments.

The following tables (Table 7 - 16) contain the main parameters of Tacotron2 models. The changed parameters were highlighted with bold.

*Table 7: Tacotron2 parameters (ljspeech-80-d1024)*

| ljspeech-80-d1024 | Name of parameter | Value of parameter |
|---|---|---|
| | sampling_rate | 8000 |
| | filter_length | 512 |
| | hop_length | 128 |
| | win_length | 512 |
| | n_mel_channels | 80 |
| | mel_fmin | 0.0 |
| | mel_fmax | 4000.0 |
| | decoder_rnn_dim | 1024 |
| | attention_rnn_dim | 1024 |
| | encoder_embedding_dim | 512 |
| | symbols_embedding_dim | 512 |

*Table 8: Tacotron2 parameters (ljspeech-62-d1024)*

| ljspeech-62-d1024 | Name of parameter | Value of parameter |
|---|---|---|
| | sampling_rate | 8000 |
| | filter_length | 512 |
| | hop_length | 128 |
| | win_length | 512 |
| | **n_mel_channels** | **62** |
| | mel_fmin | 0.0 |
| | mel_fmax | 4000.0 |
| | decoder_rnn_dim | 1024 |
| | attention_rnn_dim | 1024 |
| | encoder_embedding_dim | 512 |
| | symbols_embedding_dim | 512 |

*Table 9: Tacotron2 parameters (ljspeech-62-d512)*

| ljspeech-62-d512 | Name of parameter | Value of parameter |
|---|---|---|
| | sampling_rate | 8000 |
| | filter_length | 512 |
| | hop_length | 128 |
| | win_length | 512 |
| | **n_mel_channels** | **62** |
| | mel_fmin | 0.0 |
| | mel_fmax | 4000.0 |
| | **decoder_rnn_dim** | **512** |
| | **attention_rnn_dim** | **512** |
| | encoder_embedding_dim | 512 |
| | symbols_embedding_dim | 512 |

*Table 10: Tacotron2 parameters (ljspeech-62-d512-e256)*

| ljspeech-62-d512-e256 | Name of parameter | Value of parameter |
|---|---|---|
| | sampling_rate | 8000 |
| | filter_length | 512 |
| | hop_length | 128 |
| | win_length | 512 |
| | **n_mel_channels** | **62** |
| | mel_fmin | 0.0 |
| | mel_fmax | 4000.0 |
| | **decoder_rnn_dim** | **512** |
| | **attention_rnn_dim** | **512** |
| | **encoder_embedding_dim** | **256** |
| | **symbols_embedding_dim** | **256** |

*Table 11: Tacotron2 parameters (ljspeech-62-d384)*

| ljspeech-62-d384 | Name of parameter | Value of parameter |
|---|---|---|
| | sampling_rate | 8000 |
| | filter_length | 512 |
| | hop_length | 128 |
| | win_length | 512 |
| | **n_mel_channels** | **62** |
| | mel_fmin | 0.0 |
| | mel_fmax | 4000.0 |
| | **decoder_rnn_dim** | **384** |
| | **attention_rnn_dim** | **384** |
| | encoder_embedding_dim | 512 |
| | symbols_embedding_dim | 512 |

*Table 12: Tacotron2 parameters (ljspeech-62-d512-a768)*

| ljspeech-62-d512-a768 | Name of parameter | Value of parameter |
|---|---|---|
| | sampling_rate | 8000 |
| | filter_length | 512 |
| | hop_length | 128 |
| | win_length | 512 |
| | **n_mel_channels** | **62** |
| | mel_fmin | 0.0 |
| | mel_fmax | 4000.0 |
| | **decoder_rnn_dim** | **512** |
| | **attention_rnn_dim** | **768** |
| | encoder_embedding_dim | 512 |
| | symbols_embedding_dim | 512 |

*Table 13: Tacotron2 parameters (ljspeech-62-d640)*

| ljspeech-62-d640 | Name of parameter | Value of parameter |
|---|---|---|
| | sampling_rate | 8000 |
| | filter_length | 512 |
| | hop_length | 128 |
| | win_length | 512 |
| | **n_mel_channels** | **62** |
| | mel_fmin | 0.0 |
| | mel_fmax | 4000.0 |
| | **decoder_rnn_dim** | **640** |
| | **attention_rnn_dim** | **640** |
| | encoder_embedding_dim | 512 |
| | symbols_embedding_dim | 512 |

*Table 14: Tacotron2 parameters (ljspeech-62-d512-e384)*

| ljspeech-62-d512-e384 | Name of parameter | Value of parameter |
|---|---|---|
| | sampling_rate | 8000 |
| | filter_length | 512 |
| | hop_length | 128 |
| | win_length | 512 |
| | **n_mel_channels** | **62** |
| | mel_fmin | 0.0 |
| | mel_fmax | 4000.0 |
| | **decoder_rnn_dim** | **512** |
| | **attention_rnn_dim** | **512** |
| | **encoder_embedding_dim** | **384** |
| | **symbols_embedding_dim** | **384** |

*Table 15: Tacotron2 parameters (ljspeech-62-d512-e640)*

| ljspeech-62-d512-e640 | Name of parameter | Value of parameter |
|---|---|---|
| | sampling_rate | 8000 |
| | filter_length | 512 |
| | hop_length | 128 |
| | win_length | 512 |
| | **n_mel_channels** | **62** |
| | mel_fmin | 0.0 |
| | mel_fmax | 4000.0 |
| | **decoder_rnn_dim** | **512** |
| | **attention_rnn_dim** | **512** |
| | **encoder_embedding_dim** | **640** |
| | **symbols_embedding_dim** | **640** |

*Table 16: Tacotron2 parameters (ljspeech-62-d512-p384)*

| ljspeech-62-d512-p384 | Name of parameter | Value of parameter |
|---|---|---|
| | sampling_rate | 8000 |
| | filter_length | 512 |
| | hop_length | 128 |
| | win_length | 512 |
| | **n_mel_channels** | **62** |
| | mel_fmin | 0.0 |
| | mel_fmax | 4000.0 |
| | **decoder_rnn_dim** | **512** |
| | **attention_rnn_dim** | **512** |
| | encoder_embedding_dim | 512 |
| | symbols_embedding_dim | 512 |
| | **postnet_embedding_dim** | **384** |

# Results

## Validation loss

At the base model we modified only the sample rate and the connected hop, window and filter length. They are not signed by bold, they are the same at all models.

The base model contains 1024 LSTM cells in the decoder but the number of Mel channels were decreased so less cells are enough to model the data. The 384, 512, 640 LSTM cells were investigated, and the 512 LSTM cells was the most successful. The size of the encoder is independent of the decoder's Mel channel number, and the training showed that changing the encoder size cause worse performance. The changing of validation loss is presented in Figure **Hiba! A hivatkozási forrás nem található.**.

-

validation.loss

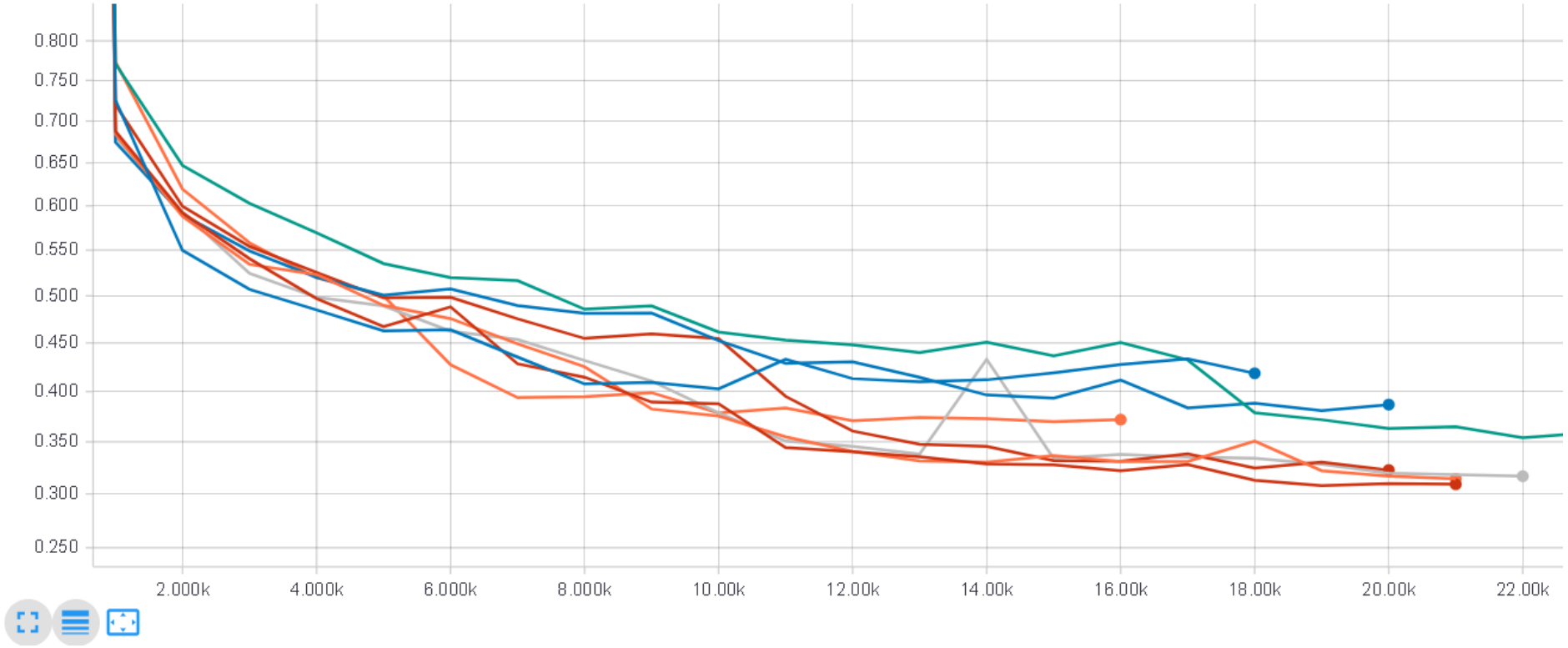


*Figure 5: Validation values of the different models*

## Attention mechanism

Between the encoder and the decoder the attention mechanism supports the connection. The state of the attention mechanism is observable via alignment figures. They show which decoder timesteps use which encoder timesteps. The following figures show the alignments of a training process at three different validations. At the first one (Fig. 6) there is no connection between encoder and decoder, at the second one it started to form (Fig. 7). At the third one (Fig. 8) the connection appears as a diagonal curve.

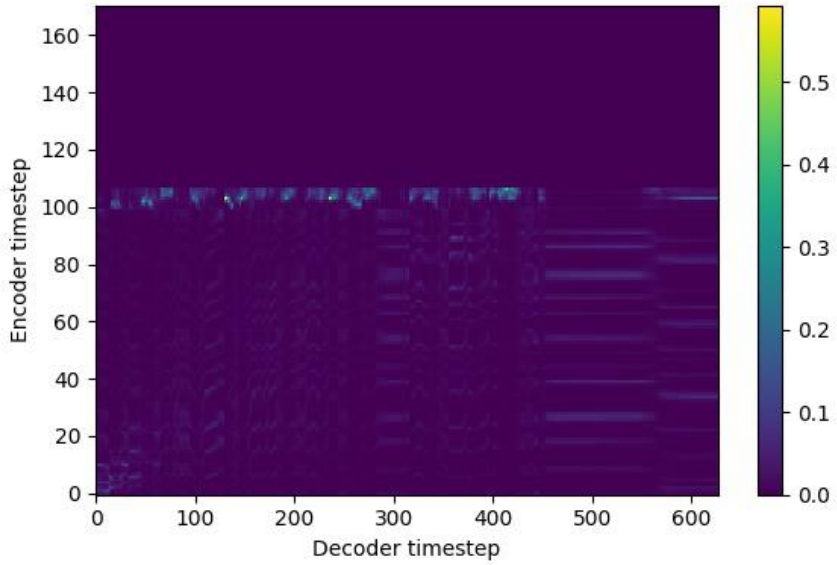


*Figure 6: The attention of ljspeech-62-d512 @8000 iterations*

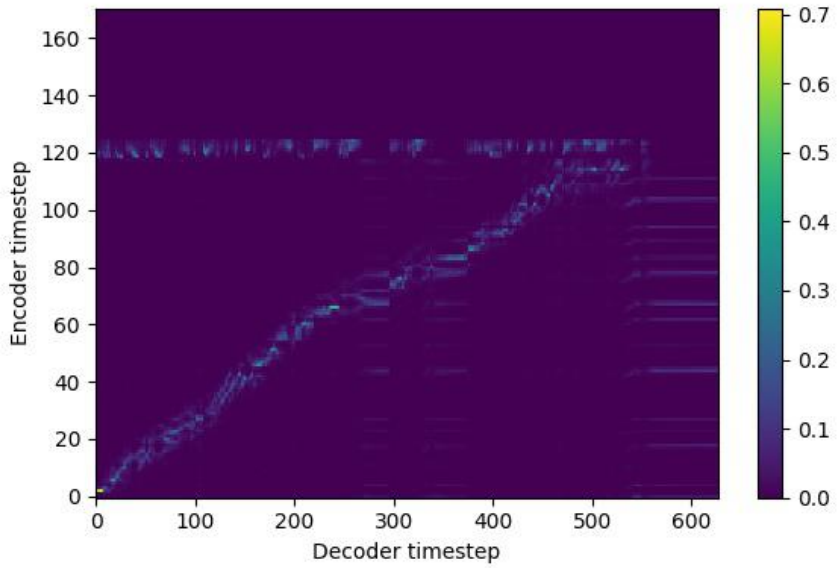


*Figure 7: The attention of ljspeech-62-d512 @9000 iterations*

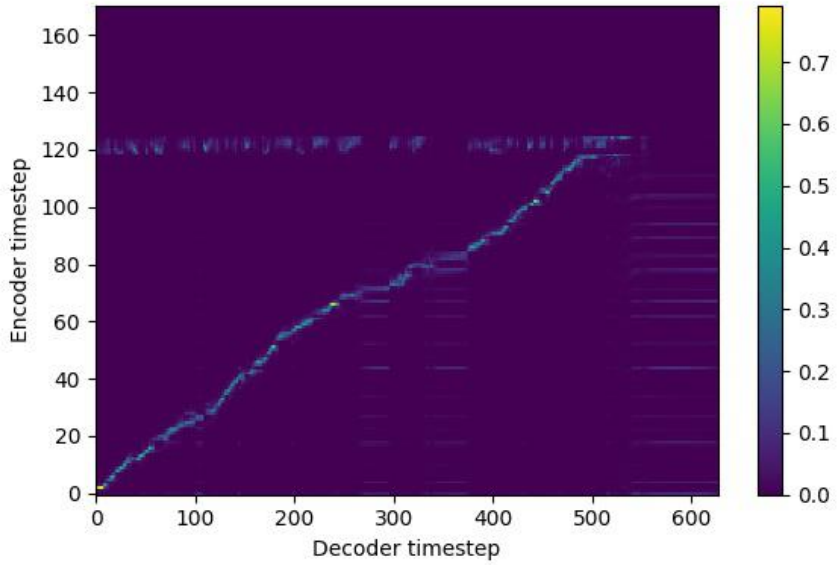


*Figure 8: The attention of ljspeech-62-d512 @10000 iterations*

The table below shows the summarized performance of trainings. The Alignment values show the number of iterations where the diagonal curves started to form on alignment pictures. In comments that training procedures are signed where the form of curve of validation loss suggests the validation loss might not be the best value, a bigger patience of early stopping may allow more training.

*Table 17: The summarized performance of trainings*

| | Alignment | Best validation value | @step | Comment | Number of GPUs | BatchSize |
|---|---|---|---|---|---|---|
| ljspeech-80-d1024 | 6000 | | | | 8 | 80 |
| ljspeech-62-d1024 | 7000 | | | | 8 | 80 |
| ljspeech-62-d512 | 9000 | 0.3078 | 19000 | | 8 | 80 |
| ljspeech-62-d512-e256 | 15000 | | | | 8 | 80 |
| ljspeech-62-d384 | 17000 | 0.3135 | 23000 | may need more training | 8 | 80 |
| ljspeech-62-d512-a768 | 10000 | | | | 8 | 80 |
| ljspeech-62-d640 | 9000 | 0.3168 | 22000 | may need more training | 8 | 80 |
| ljspeech-62-d512-e384 | 8000 | | | | 8 | 80 |
| ljspeech-62-d512-e640 | >20000 | | | no attention curve | 8 | 80 |
| ljspeech-62-d512-p384 | 11000 | | | | 8 | 80 |
| ljspeech-62-d512-1TitanXp | 18000 | 0.3051 | 58000 | | 1 | 64 |
| ljspeech-62-d512-2TitanX | 13000 | 0.3365 | 16000 | may need more training | 2 | 64 |

At some models, where the number of LSTM cells was lower than at the basic model, the memory requirement was smaller, so we could increase the batch size, but we left it on the same value on 8 GPUs systems. It was beneficial because the same iteration steps meant the same epochs.

The speed of learning, the value of validation loss and the alignment values show that the best model is ljspeech-62-d512.

# Proposed model training environment description (GPU based)

## Training on different GPUs

The training of a DNN is a long procedure, so more GPUs generally mean less training time. At Tacotron2 it is true, in our tests the model training was the fastest on a 8 GPUs system and the slowest on a single GPU.

The figure below shows the validation loss of the 8 GPU system. It reached the best value at 19000 iterations. It took 10h50 min time. The batch size of distributed training was 8x80 = 640 sentences

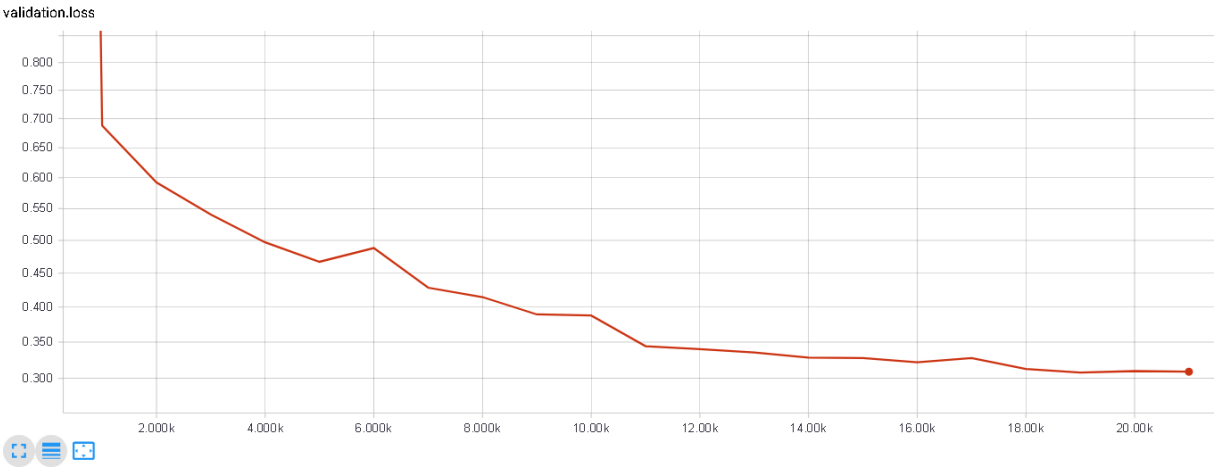


*Figure 9: Validation loss on 8xV100 GPUs*

The question is that smaller systems can be efficient enough? Can they reach same loss value? We ran the same training on a single GPU system. The GPU was an NVidia Titan Xp. Because it contains less memory than the V100 (12 Gbyte instead of 16 Gbyte), the batch size was decreased to 64. It reached the best value after 58000 iterations. The figure below shows the validation loss of training. The alignment became good after 18000 iterations. The best validation loss was 0.3051 which is practically the same as the 8 GPUs system's validation loss (0.3078). It took about 5.4 times more than with 8 GPUs (58h30min). Depending on pricing of GPUs the single GPU environment would be cost effective, if the training time is less important.

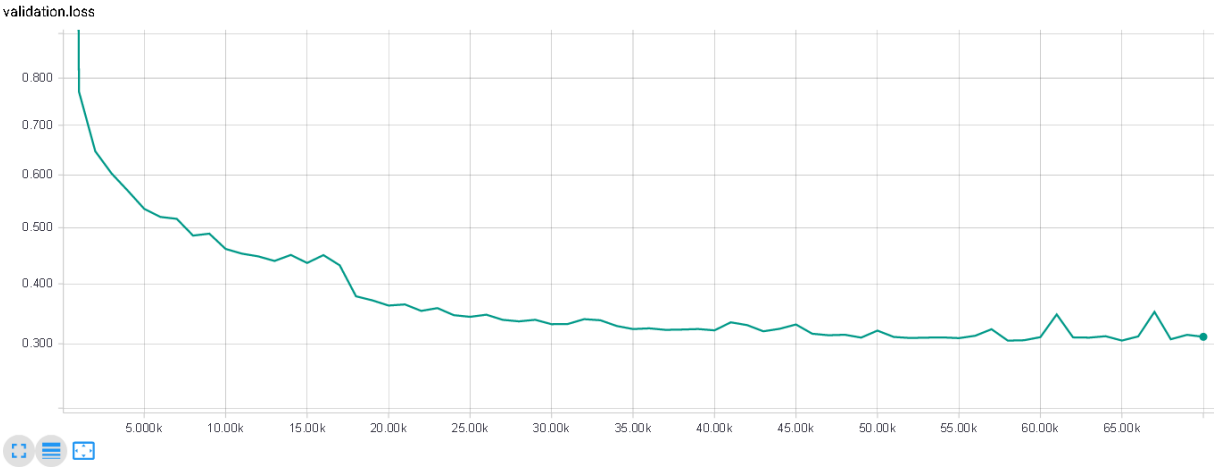


*Figure 10: Validation loss on a single Titan Xp GPU*

The other option is to use more GPUs, but less than 8. The third option was two Nvidia Titan X boards in distributed training. As expected it was faster than one GPU, but slower than 8 GPU training. The batch size was 2x64=128 sentences. It reached the proper alignment after 13000 iterations. In our experiment the best validation loss was only 0.3365, but the tendency was good, so it would reach better values. A technical difficulty caused the end of this training. From tendency and compared with the single GPU training with the same epoch, the two GPU distributed training may provide the same or better results than a single GPU. The 2 GPU training reached the 16000 iterations after 16 hours.

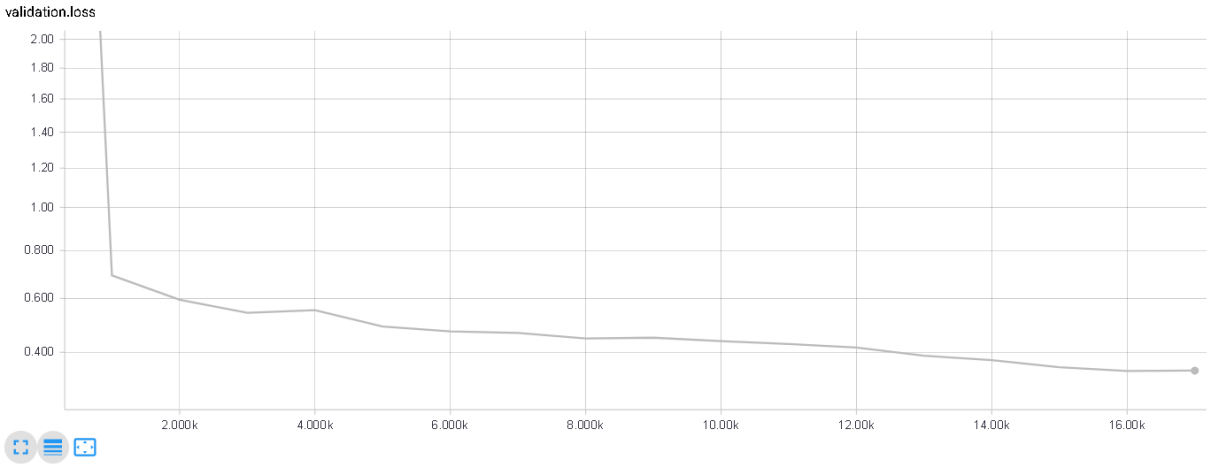


*Figure 11: Validation loss on two Titan X GPUs*

## Parameters of proposed models

### Tacotron2

***Source:***

https://github.com/NVIDIA/tacotron2

***Parameters (hparam.py)***

```
import tensorflow as tf
from text import symbols

def create_hparams(hparams_string=None, verbose=False):
    """Create model hyperparameters. Parse nondefault from given string."""

    hparams = tf.contrib.training.HParams(
        ################################
        # Experiment Parameters        #
        ################################
        epochs=1500,
        iters_per_checkpoint=1000,
        seed=1234,
        dynamic_loss_scaling=True,
        fp16_run=True,
        distributed_run=True,
        dist_backend="nccl",
        dist_url="tcp://localhost:54321",
        cudnn_enabled=True,
        cudnn_benchmark=False,
        ignore_layers=['embedding.weight'],

        ################################
        # Data Parameters              #
        ################################
        load_mel_from_disk=False,
        training_files='filelists/ljs_audio_text_train_filelist.txt',
        validation_files='filelists/ljs_audio_text_val_filelist.txt',
        text_cleaners=['english_cleaners'],

        ################################
        # Audio Parameters             #
        ################################
        max_wav_value=32768.0,
        sampling_rate=8000,
        filter_length=512,
        hop_length=128,
        win_length=512,
        n_mel_channels=62,
        mel_fmin=0.0,
        mel_fmax=4000.0,

        ################################
        # Model Parameters             #
        ################################
        n_symbols=len(symbols),
        symbols_embedding_dim=512,

        # Encoder parameters
        encoder_kernel_size=5,
        encoder_n_convolutions=3,
        encoder_embedding_dim=512,
```

```
        # Decoder parameters
        n_frames_per_step=1,  # currently only 1 is supported
        decoder_rnn_dim=512, # 1024
        prenet_dim=256,
        max_decoder_steps=1000,
        gate_threshold=0.5,
        p_attention_dropout=0.1,
        p_decoder_dropout=0.1,

        # Attention parameters
        attention_rnn_dim=512, #1024
        attention_dim=128,

        # Location Layer parameters
        attention_location_n_filters=32,
        attention_location_kernel_size=31,

        # Mel-post processing network parameters
        postnet_embedding_dim=512,
        postnet_kernel_size=5,
        postnet_n_convolutions=5,

        ################################
        # Optimization Hyperparameters #
        ################################
        use_saved_learning_rate=False,
        learning_rate=1e-3,
        weight_decay=1e-6,
        grad_clip_thresh=1.0,
        batch_size=80,
        mask_padding=True  # set model's padded outputs to padded values
    )

    if hparams_string:
        tf.logging.info('Parsing command line hparams: %s', hparams_string)
        hparams.parse(hparams_string)

    if verbose:
        tf.logging.info('Final parsed hparams: %s', hparams.values())

    return hparams
```

## WaveGlow

### *Source*

https://github.com/NVIDIA/waveglow

### *Parameters (config.json)*

```
{
    "train_config": {
        "fp16_run": true,
        "output_directory": "./out",
        "log_directory": "./log",
        "epochs": 100000,
        "learning_rate": 1e-4,
        "sigma": 1.0,
        "iters_per_checkpoint": 2000,
        "batch_size": 16,
```

```
        "seed": 1234,
        "checkpoint_path": ""
    },
    "data_config": {
        "training_files": "train_files.txt",
        "segment_length": 16768,
        "sampling_rate": 8000,
        "filter_length": 512,
        "hop_length": 128,
        "win_length": 512,
        "mel_fmin": 0.0,
        "mel_fmax": 4000.0
    },
    "dist_config": {
        "dist_backend": "nccl",
        "dist_url": "tcp://localhost:54321"
    },

    "waveglow_config": {
        "n_mel_channels": 62,
        "n_flows": 12,
        "n_group": 8,
        "n_early_every": 4,
        "n_early_size": 2,
        "WN_config": {
            "n_layers": 7,
            "n_channels": 256,
            "kernel_size": 3
        }
    }
}
```

### *Modified function in mel2samp.py*

In order to train WaveGlow with 62 Mel channels you have to modify the source code, because the Mel spectrum calculation does not get this parameter. The default value is 80.

The modified line is emphasized by bold font.

```
...
class Mel2Samp(torch.utils.data.Dataset):
    """
    This is the main class that calculates the spectrogram and returns the
    spectrogram, audio pair.
    """
    def __init__(self, training_files, segment_length, filter_length,
                 hop_length, win_length, sampling_rate, mel_fmin,
mel_fmax):
        self.audio_files = files_to_list(training_files)
        random.seed(1234)
        random.shuffle(self.audio_files)
        self.stft = TacotronSTFT(filter_length=filter_length,
                                 hop_length=hop_length,
                                 win_length=win_length,
                                 sampling_rate=sampling_rate,
                                 n_mel_channels=62,
                                 mel_fmin=mel_fmin, mel_fmax=mel_fmax)
        self.segment_length = segment_length
        self.sampling_rate = sampling_rate
```

# References

Jonathan Shen, R. P.-R. (2017). Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions. Forrás: https://arxiv.org/abs/1712.05884

Ryan Prenger, R. V. (2018). WaveGlow: A Flow-based Generative Network for Speech Synthesis. Forrás: https://arxiv.org/abs/1811.00002