Audio final exercise experiments

Winning final model:

Our final model is the network architecture below with a beam-search and a 4 gram LLM decoder:

CTC Network architecture:
encoder-decoder neural network:

Encoder: 5 residual blocks CNN network
- Conv(1, 16, kernel = 7, stride=1)
- Layernorm
- Gelu
- dropout
- Resblock(16, 16, kernel=3, stride=1)
- Conv(16, 32, kernel = 3, stride=2)
- 4 x Resblock(32, 32, kernel=3, stride=1)
- FC(32*(128//2), rnn_dim)

where a Resblock is:
- residual = x
- 2 x (
  - x = layer_norm(x)
  - x = Gelu(x)
  - x = dropout(x)
  - x = cnn(x)
-)
- x += residual

Decoder:
- 3 layers BidirctionalGRU
- hidden_ dim = 512
- input_size = 512

the encoder is halving the time domain from 751 frames to 375

end2end models:

by observing the data, one thing that comes to mind is how small the data set is. especially for end-to-end solutions with Deep Neural Networks.
The most common solution will be to use a pre-trained Encoder that has a good and useful semantic encoding for our ASR mission.
We decided that we wanted to take this as a challenge and to try and work only with this dataset and to make the best out of it, meaning not using pre-trained audio encoders and decoders.
For that, we decided to try and experiment with augmentations, different network architectures, and different Losses to see what results we could achieve.

For all of our experiments, we are using mel spectrogram representations with a sample rate of 16,000, window length of 25ms hop length of 10 ms,
and 128 mel filter banks. we are padding all of our mel spectrogram representations to be on the same size through all the dataset.
we are using a character level tokenizer with the addition of special tokens:

- '|' silence
- '_' CTC decoder special token
- '<PAD>' padding
- <SOS> start token
- <EOS> end of token

while using the start and end tokens only for the cross entropy loss and not for the CTC Loss.

```
mel_spectrogram = T.MelSpectrogram(
    sample_rate=16000,
    n_fft=int(0.025*16000),     # Corresponding to 25ms
    win_length=int(0.025*16000), # Corresponding to 25ms
    hop_length=int(0.010*16000), # Corresponding to 10ms
    n_mels=128,
    center=True,
    normalized=True,
    power=2.0,
    pad_mode="reflect",
    f_min=150,
    f_max=8000,
    window_fn=torch.hamming_window
)
```

Cross-entropy loss model:

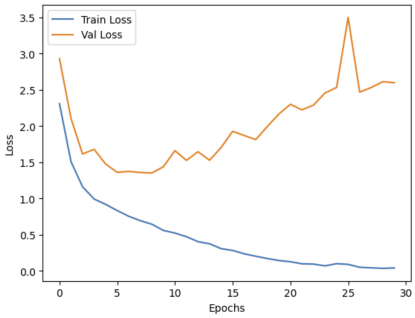
for this experiments, we will use an Encoder-Decoder Neural Network with an Autoregressive LSTM that gets as input the current word and the Mel representation by attention mechanism. our loss is a standart Cross-Entropy loss to have a baseline of how the simplest loss will work on our task.

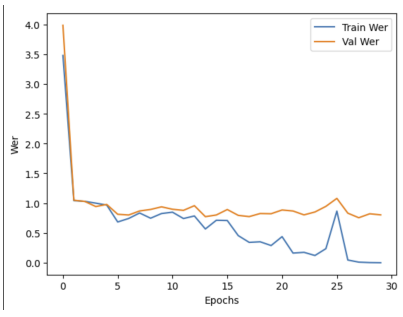
our network architecture is as follows:
CE Network architecture:
Encoder:
Untrained Resnet_18 architecture without the top-3 layers
Decoder:
pos-encoding
one layer 2x Attention-heads
with 1x layers one directional Autoregressive Lstm
2 layers FC classifier

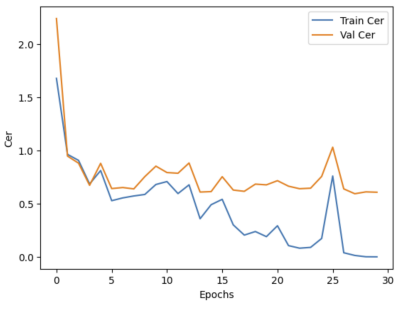
Data:
we are padding every sentence beginning with a start token <SOS> and an end token <EOS> at the end followed by the necessary amount of padding tokens <PAD>.

exp 1:

first we are ex
Network: CE Network
scheduler: None
lr: 0.0005
weight-decay: 1e-5
dropout: 0.1
batch_size: 64
data: non-augmented







* Epochs are in multiples of 10

Test:

```
WER: 0.8124191461836999
CER: 0.6097465886939571
```

Val:

```
WER: 0.8375
CER: 0.6335227272727273
```

Train:

```
WER: 0.029905776321179845
CER: 0.018019137566795077
```

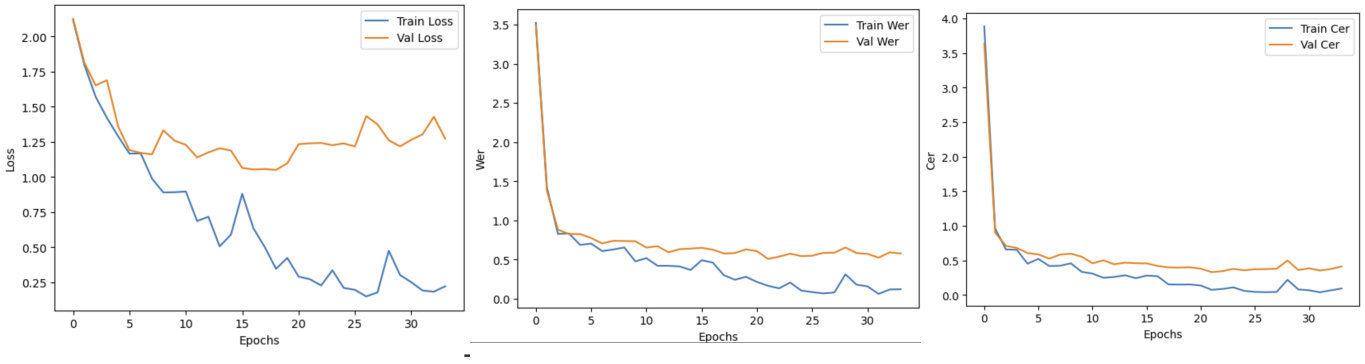
Results:

because of the small dataset the model is overfitted to the data and particularly to the words without considering the mel spectrogram to much, in result, not manage to generalize. we can see it from Val and Train Loss in the Loss graph and from the WER and CER results.

exp 2:

In order to try and force the model to not trust only the previous words that it has seen and by that hopefully relying more on the mel specs we will change the captions that the model sees by masking random tokens from the text.

Network: CE Network
scheduler: None
lr: 0.0005
weight-decay: 1e-5
dropout: 0.1
batch_size: 64
data: Text-masking (masking 65% of the text gave the best results)



* Epochs are in multiples of 10

Test:
```
WER: 0.5795601552393272
CER: 0.3738791423001949
```
Val:
```
WER: 0.5428571428571428
CER: 0.3625
```
Train:
```
WER: 0.04055714870954527
CER: 0.02528892755063999
```

Results:
the results are better but not as good as expected could be that there is a need for better tuning of hyperparameters and neural network architecture.

CTC Loss:

due to the very small dataset, The main problem that we encountered with the standard cross-entropy loss was that the model was overfitting and it was hard for it to generalize so a different, more biased toward the task way is needed.

Here the CTC loss comes for our help.
by using the CTC loss we are biasing the model to count more on the audio wave itself, less on the previous words, and focus on the acoustic features of each frame of the mel spectrogram representation. this is exactly what we want.

for all the following CTC loss experiments we are using the CTC Network a different network than before.
the new network will use the fact that now we can use a bidirectional RNN not like the case with the previous architecture.
also to bias the model to focus more on the current mel spectrogram frame,

we disclude the Attention mechanism.

CTC Network architecture:
encoder-decoder neural network:

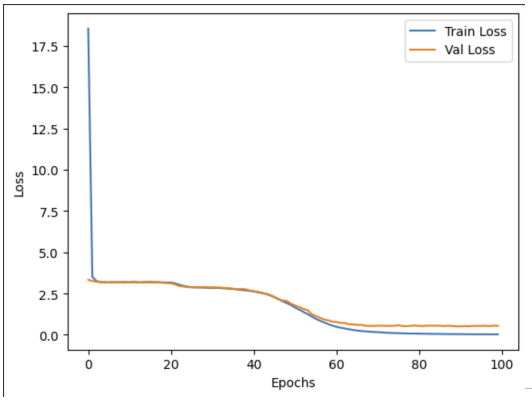
Encoder: 5 residual blocks CNN network
Conv(1, 16, kernel = 7, stride=1)
Layernorm
Gelu
dropout
Resblock(16, 16, kernel=3, stride=1)
Conv(16, 32, kernel = 3, stride=2)
4 x Resblock(32, 32, kernel=3, stride=1)
FC(32*(128//2),rnn_dim)

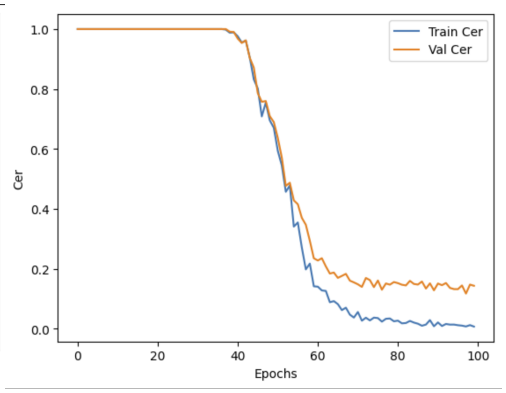
where a Resblock is:
residual = x
2 x (
x = layer_norm(x)
x = Gelu(x)
x = dropout(x)
x = cnn(x)
)
x += residual

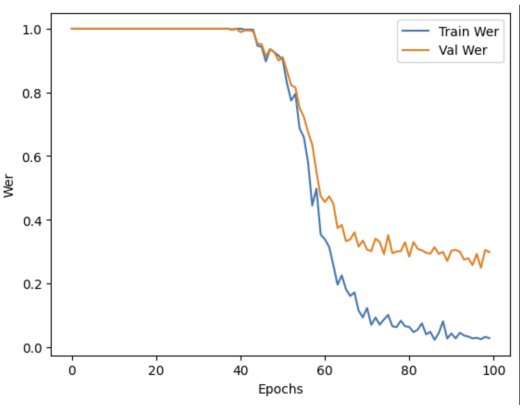
Decoder:
3 layers BidirctionalGRU
hidden_ dim = 512
input_size = 512

exp 1:

Network: CTC Network
scheduler: LinearLR
lr: 0.0005
dropout: 0.1
batch_size: 64
data: non-augmented
CTC decoder: greedy
LLM: None





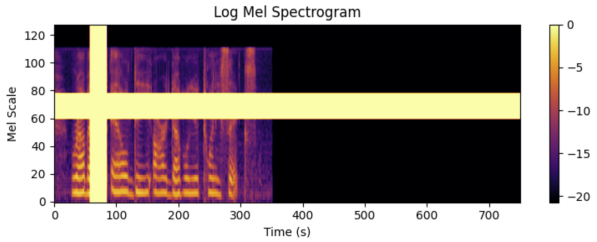


Test results:
`WER: 0.26002587322121606`
`CER: 0.11423001949317739`
Val results:
`WER: 0.2892857142857143`
`CER: 0.13920454545454544`
Train results:
`WER: 0.023555919705038918`
`CER: 0.008512489126382503`

Results:
the result shows that we are doing a lot better than the cross-entropy loss model, but we are still Overfitting on the training data.
this is to be expected due to the size of the training dataset.

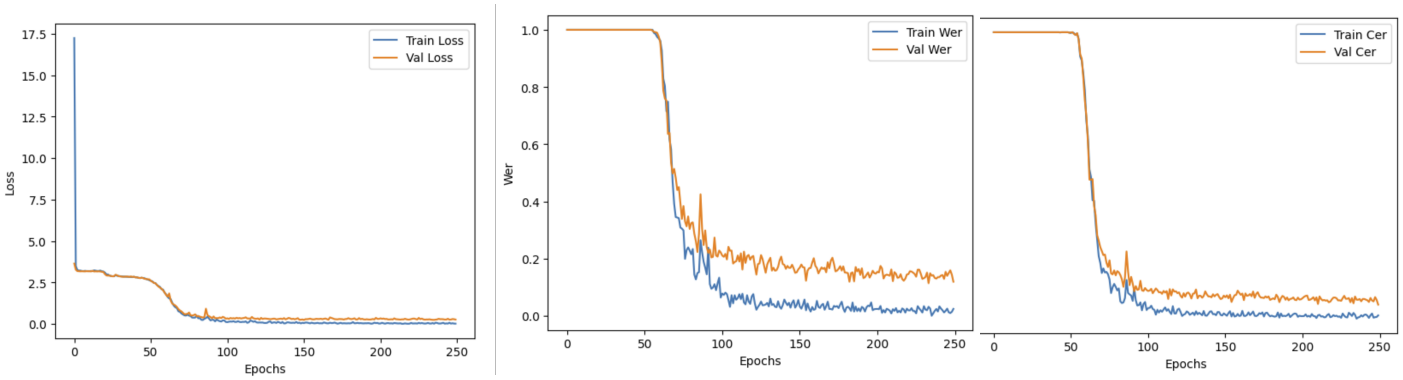
exp 2:

Augmented SPEC-Augmentations (Time and frequency masking)



In this experiment we will try to improve our small training dataset by augmentations, we augment our data by masking frequencies and time from the mel spectrogram itself known as spec augments from the paper *SpecAugment* [Park *et al.*, 2019]
by that we hope to create more variations that will help our model to generalize better.

Network: CTC Network
scheduler: LinearLR
lr: 0.0005
dropout: 0.1
batch_size: 64
data: Spec-Augmentations (Time and frequency masking)
CTC decoder: greedy
LLM: None



Test:
`WER: 0.11772315653298836`
`CER: 0.05146198830409357`
Val:
`WER: 0.1357142857142857`
`CER: 0.0625`
Train:
`WER: 0.00020483408439164277`
`CER: 6.213495712687958e-05`

Results:
The spec masking augmentation helped the model to generalize by creating useful and cheap to compute Variations of the data and by making the mission harder for the model s,t it will be a lot harder to overfit on the data.

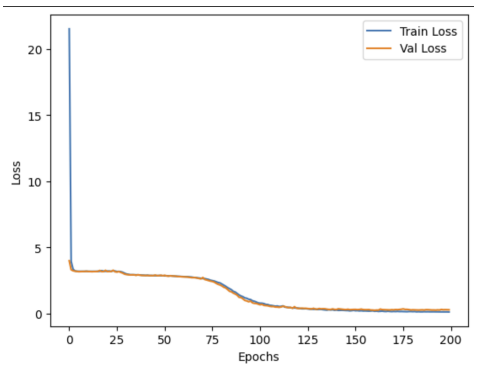
exp 3:

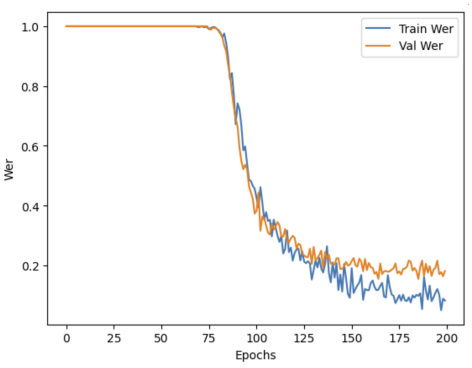
Augmented Audio:
because we are using a very small dataset, we wanted to use augmentations on the audio waves to increase the amount of Variation in the dataset and at the same time try to keep the semantic meaning of the audio
using delay augmentation, for example, won't work because it will change the number of letters in a sentence.
So the basic rule of thumb is that if a human can still understand the sentence then it is a good candidate for an augmentation.
We exclude reverb for example because of the high computing time.

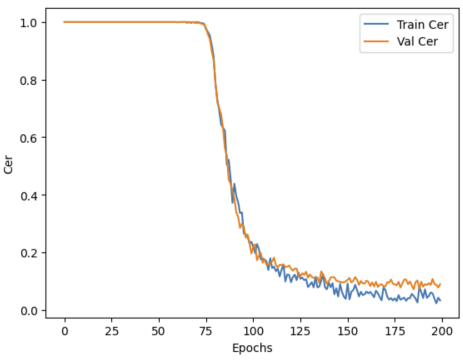
The augmentations we use include the following randomized augmentations:

- cropping
- polarity inversion
- Noise addition
- gain change
- high and low pass filters
- pitch shift
- speed perturbations

Network: CTC Network
scheduler: LinearLR
lr: 0.0005
dropout: 0.1
batch_size: 64
data: Audio augmentations
CTC decoder: greedy
LLM: None







Test:

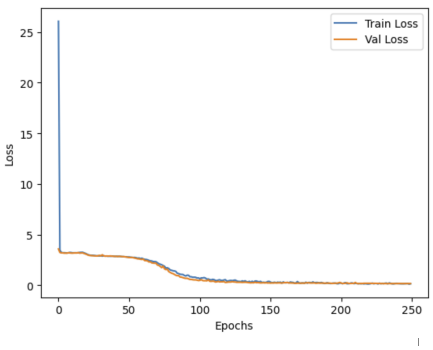
`WER: 0.13583441138421734`
`CER: 0.05808966861598440 6`
Val:
`WER: 0.1732142857142857`
`CER: 0.08125`
Train:
`WER: 0.08500614502253175`
`CER: 0.037405244190381506`

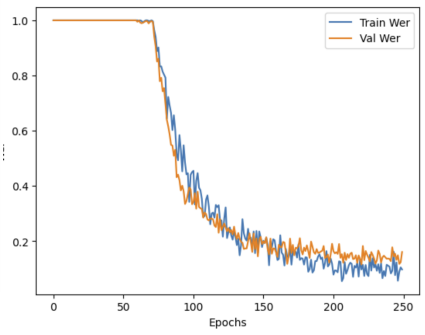
results:
from the results we can see that the results are better than the results with no augmentations. from the graphs we can see that the train and val Loss, WER and CER are closer to each other which means that the augmentations helped the model to generalize as was expected.

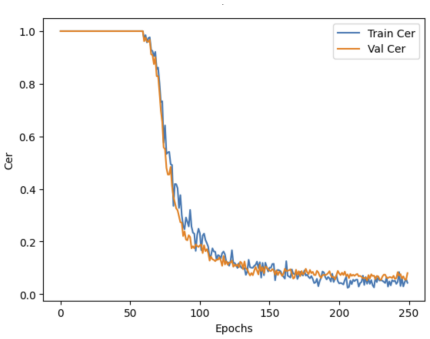
exp 4:

combining the augmentations:
in this experiment, we will check if combining both augmentations will help the model to generalized even more.

Network: CTC Network
scheduler: LinearLR
lr: 0.0005
dropout: 0.1
batch_size: 64
data: Audio + Spec masking augmentations
CTC decoder: greedy
LLM: None







Test:
`WER: 0.10219922380336352`
`CER: 0.04132553606237817`
Val:
`WER: 0.15`
`CER: 0.06818181818181818`
Train:
`WER: 0.04465383039737812`
`CER: 0.01901329688082515`

results:
the results of both augmentations are very similar to the only spec-Augmentations but the main difference is that the Loss, Wer, and Cer of the Val and train data sets are closer during the training, which may suggest that there is more room for improvement with a bigger model or more training.

exp 5:

Adding LLM and beam search:
in this experiment, we are only adding a decoder, so no additional Training.
we are using for the first time a pre-trained model, the KenLM 4-gram language model - `"librispeech-4-gram"` with beam-search.

Network: CTC Network
scheduler: LinearLR
lr: 0.0005
dropout: 0.1
batch_size: 64
data: Audio + Spec masking augmentations
CTC decoder: beam search
LLM: `librispeech-4-gram`

We are using our best current model the model from experiment 4:

results:
the results got even better
our dataset includes many single letters which are not so common as a result the pre-trained LLM only gave good results with a very small weight.

Test Dataset Examples:

Targets	Predictions
six five five eight seven four zero,	six five five eight seven four zero
enter seven,	enter seven
j e f f r e y,	j e f f i e y
four one two two six eight four one four two,	four one two two six eight four one four two
no,	no
enter four five eight two one,	enter four five eight two on
m y e r s,	m y e i s
rubout c b w x v four,	rub out c b w x v four
ten twenty seven sixty two,	ten twee seven sixty two
one fifty,	one fifty
three zero two one,	three zero two one
s p e e r,	s t e e r
yes,	yes
p i t t s b u r g h,	p i t t s b u r g h
four two one eight eight six oh,	four two one eight eight six o
l e x i n g t o n,	l e x i n t o one
rubout u b u t r six,	rub out u b u d i six
w i l l o w b e n d,	w i l l o w b e n

Final Test Results

```
WER: 0.08796895213454076
CER: 0.0354775828460039
```

proposals for improvement:

1. to improve our model furthermore the best thing we can probably do is to train our own LLM on our dataset. our dataset includes many single letters which are not so common as a result the pre-trained LLM only gave good results with a very small weight.
2. play more with the hyper-parameters and architecture.
3. use pre-trained encoders etc.
4. Train a generative model on the dataset in order to have a better encoder so then we can use it for the ASR task.