Neural Network-Based Character-to-Symbol Sequence Translation for Text Compression

Giuseppe Prisco - 1895709

“**Engineering in Computer Science**” Master’s Degree
Deep Learning (2023/2024)

Task Description (1)

The goal of the project was to to develop a neural network model that could convert **human-readable text** into a **machine-readable symbol sequence** that is compatible with unzip softwares.

- “The dogs are cute” ➡ x\x9c\x0b\xc9HUH\xc9O/VH,JUH.-I\x05\x005\x7f\x06\x18
- “The dogs are cute” ➡ eJwLyUhVSMlPL1ZILEpVSC4tSQUANX8GGA==

Task Description (2)

Given the **innovative** nature of this project, I explored various approaches related to:

- **pre-processing** and **post-processing** of input and output data formats

- alternative **architectures** to accomplish the task

Dataset Creation and Pre-processing

Dataset Creation

Download the “**GLUE**” dataset from the Hugging Face Hub and divide it into:

- **training** set
- **validation** set
- **test** set

Pre-process Data

- build pairs of **original texts** and their **compressed version**:
  - just compress with zlib
  - compress with zlib and then encode in base64
- **encode** each string of the pair using T5 or ByT5 tokenizer

Implementation of the Baseline

The tested baseline consist of a simple **Recurrent Neural Network** (RNN) with the following structure:

- **Embedding Layer ➡** the integers representing tokens pass through this layer, which converts each integer into a dense vector (embeddings)
- **Gated Recurrent Unit (GRU) Layer ➡** it process the embeddings, maintaining hidden states, and returns the sequence of hidden states for each step
- **Fully Connected Linear Layer ➡** it projects each hidden state back to the size of the vocabulary, returning the raw predictions of the model

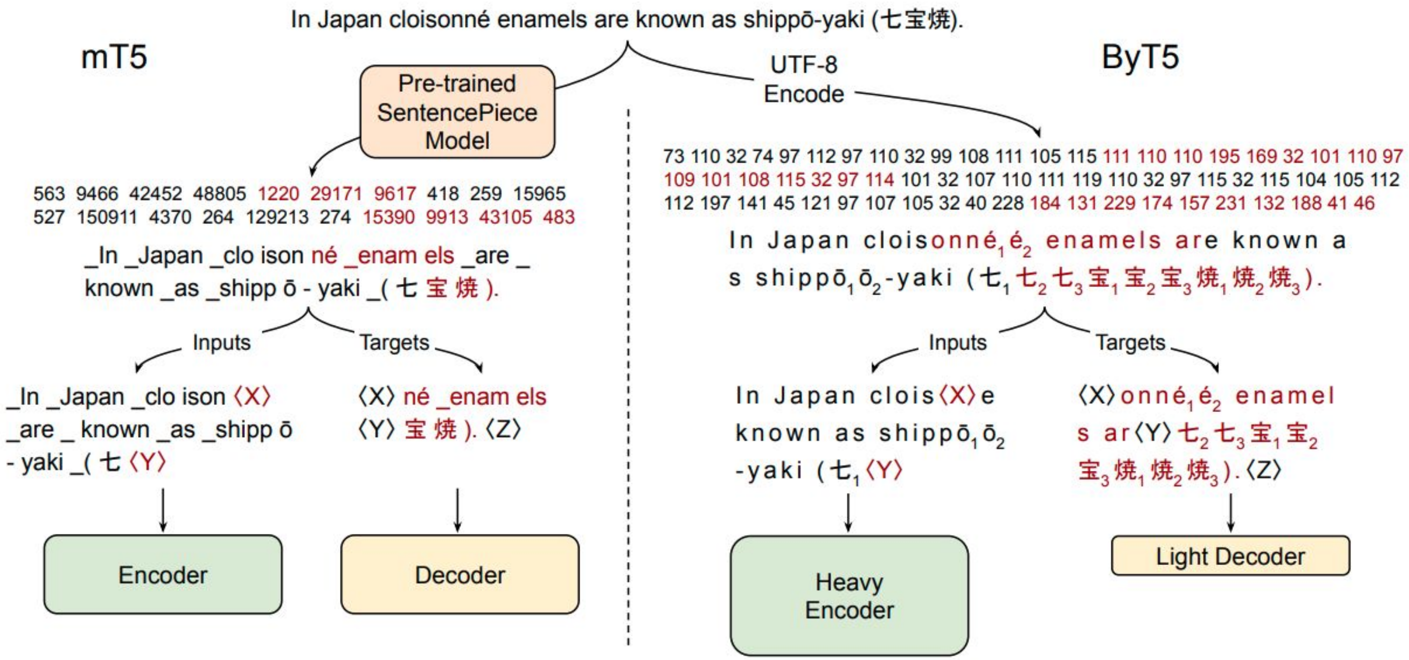
Final Model Architecture (1)

The final system is based on the T5 model from the Hugging Face Transformers library, specifically **T5ForConditionalGeneration** model.

The two tested models differ on the architecture employed ➡ **t5_base** or **byt5_base**:

- **Embedding** layer
- stack of **Encoder** layers
- stack of **Decoder** layers
- **Fully Connected** linear layer

Final Model Architecture (2)



Final Model Architecture (3)

The model is trained with the following:

- **"input_ids"** ➡ they correspond to the tokenized original texts
- **"attention_mask"**➡ it differentiates between tokens that are actual words versus others that just represent the padding
- **"labels"** ➡ they correspond to the tokenized compressed texts

The model outputs a **probability vector**, of size equal to the **"vocab_size"**, for each element of the input sequence and for each sequence in the batch.

Final Model Architecture (4)

T5

- **max sequence length** ➡ 256
- **encoder layers** ➡ 12
- **dropout** ➡ 0.1
- **learning rate** ➡ 0.00001
- **batch size** ➡ 8

ByT5

- **max sequence length** ➡ 64
- **encoder layers** ➡ 14
- **dropout** ➡ 0.1
- **learning rate** ➡ 0.00001
- **batch size** ➡ 8

Implemented Metrics

Levenshtein distance

- It is a **string metric** used to measure the **difference** between two sequences

- It represents the minimum number of **single-character edits** required to change one sequence into another

Unzip metric

- It represents how many predicted sequences can successfully be **decompressed**

- This metric assigns a **score of 1** if the zlib library can **successfully decompress** the model's output sequence, and **0 if it cannot**

Auxiliary Metrics

Predicted Sequence Length

- It measures the length of the sequence **predicted** by the model

True Sequence Length

- It represents the length of the **ground truth** sequence

Min Levenshtein distance

- abs(predicted length – true length)

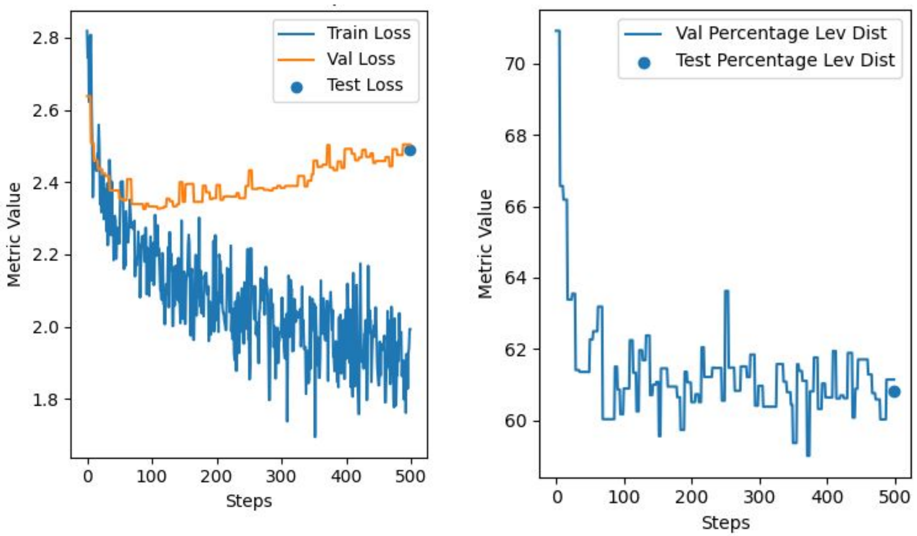
Max Levenshtein distance

- max(predicted length, true length)

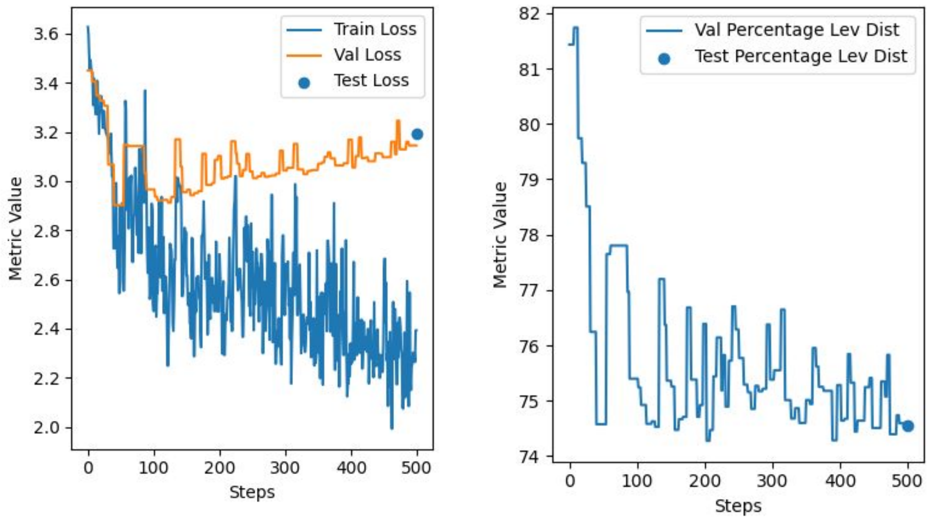
% Levenshtein distance

- ((lev - min) / (max - min)) * 100

MyBaseline - Results

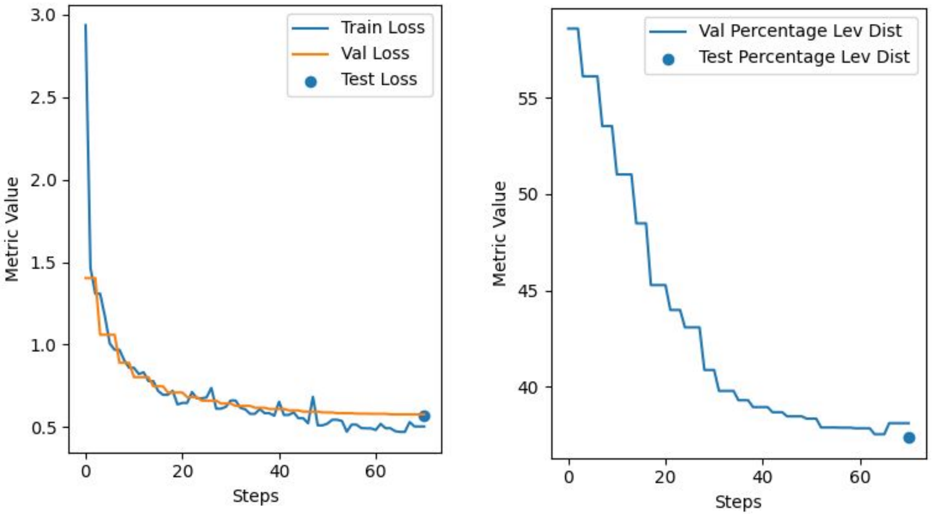


Validation/Test Losses and Percentage Levenshtein Distance, using byt5, no base64 encoding used

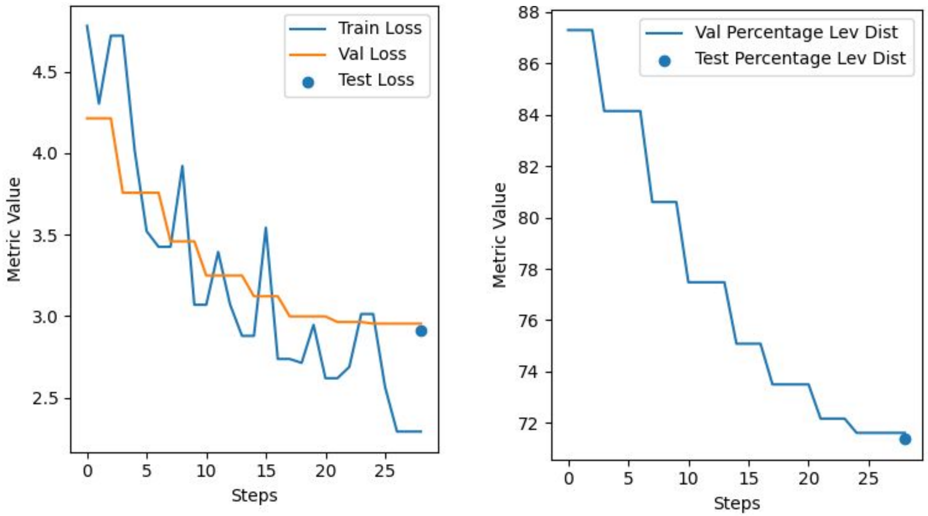


Validation/Test Losses and Percentage Levenshtein Distance, using byt5, base64 encoded

TextCompressionModel - Results



Validation/Test Losses and Percentage Levenshtein Distance, using t5, no base64 encoding used



Validation/Test Losses and Percentage Levenshtein Distance, using byt5, base64 encoded

Summary of Results

In this final table are summarized the **best results** achieved under the runtime constraints imposed by Colab:

Table 1: Models Validation/Test scores for the adopted Metrics

Model Names	Levenshtein Distance (in %)		Unzip Metric	
	No Encoding	Base 64	No Encoding	Base 64
MyBaseline (t5)	68.023	87.781	0	0
MyBaseline (byt5)	60.818	74.563	0	0
TextCompressionModel (t5)	37.347	71.688	0	0
TextCompressionModel (byt5)	37.154	71.382	0	0

As we can see, the best results are obtained when **no encoding** is used, with the "TextCompressionModel" able to achieve better results in comparison to "MyBaseline"

Conclusions

- comparison of “**MyBaseline**” based on GRUs with “**TextCompressionModel**” based on T5
- **great results** achieved for the Percentage Levenshtein Distance metric
- **challenges**: the model would need to achieve a distance of exactly 0% in order to correctly mimic the behaviour of the unzipping tools ➡ unzip metric is always 0
- the model demonstrated its ability to **recognize** the patterns of any compressed string, hinting at future possibilities to successfully perform the given task

In conclusion, the possibility of having a model that could successfully compress and decompress any given text would provide significant efficiency in text compression.

Thanks for the Attention