Deep Comedy

Allegra Adinolfi

Luca Antognetti

Lucia La Forgia

2020-2021

1. INTRODUCTION

The "*Deep Comedy*" is a commissioned exam project consisting of a typical Deep Learning task applied to *Natural Language Processing*, that aims at the generation of new text[1].
In this project, the only allowed learning data, namely the text usable as input of the network, was the "Divine Comedy" by Dante Alighieri, as the name of the project suggests, through which we had to satisfy the task of automatically generating a text with the same shape and features of a classical Divine-Comedy-like *canto* by the usage of *Neural Networks* exclusively, imitating the unique writing's style of the Father of Italian poetry and language.

1.1. Requirements: Linguistic Rules

In order to accomplish our task, we had to firstly identify the main aspects that are indispensable so that a text looks like a Divine Comedy's *canto*. Analyzing the features of the Divine Comedy, we observed that every *canto* had some similar, recurrent features.

These features represented some major sub-goals needed to be satisfied in order to possibly create a Dante-like *canto*:

- respecting the hendecasyllabic structure
- following the *terza rima* scheme (ABA, BCB, CDC…), which consists not only in rhymeness but also in *terzine* partitioning of the text
- having a single verse, at the end of every *canto*, separated from the last *terzina*.

These goals are still macro-tasks depending on some minor aspects that are indispensable in order to possibly accomplish the overall challenge of automatically generating a new *canto*. What these minor aspects consist of, in turn, depend on the type of model it is currently used, in fact, models are also based over what unit of text they learn with.

For example, it could come very natural to think that units of text, both to read and to write it, are *words*, in fact, words-basis has become one of our baseline model's units of elaboration. Quite similarly, it is easy to think of *characters* of a text as another kind of unit of text, so we had a model based on characters learning and generating.

Then, we focused on what we observed discussing over our text data: both our main sub-goals, rhymeness and hendecasyllabicness, are strictly related to the division in syllables of a text. In fact, the definition of the Italian hendecasyllables[2] is:

"*It is the principal metre in Italian poetry. Its defining feature is a constant stress on the tenth syllable, so that the number of syllables in the verse may vary, equaling eleven in the usual case where the final word is stressed on the penultimate syllable.*" [3]

[1] https://www.tensorflow.org/tutorials/text/text_generation
[2] https://it.wikipedia.org/wiki/Endecasillabo
https://www.treccani.it/enciclopedia/endecasillabo
[3] https://en.wikipedia.org/wiki/Hendecasyllable#In_Italian_poetry

To make things clearer, we can say that, in order to be considered a correct Italian hendecasyllable, a verse can have any number of syllable, but the stressed, toned vowel of the last word of the verse must belong to the tenth syllable of the verse.

For this reason, we decided that one further model would be one using syllables as units of text onto which learn and generate text. We understood very soon that the syllabification of the Divine Comedy text, in order to build input data of this last model, was a key aspect.

Syllabification in poetry depends on both grammatical and poetry rules.
Firstly, we used a python module named *Pyphen* to perform grammatical syllables partitioning and we implemented poetry rules by ourselves. In particular, we wanted to highlight some of these rules:

- *Synalepha*[4] is the merging of two words into one. This poetry rule applies when the former of the two words ends with a vowel and the latter starts with a vowel and these two letters form a diphthong. The consequence is that the two words are pronounced as one. We used this poetry rule for all our syllable based models.

- *Dialepha*[5], on the other hand, is the splitting of two words, even if the first ends with a vowel and the second starts with a vowel, because these two vowels generate a *hiatus*. We applied this poetry rule only to the last model.

- *Diaeresis*[6] is a poetry rule that sees two vowels that form a diphthong to have a separate pronunciation, for the sake of meter. Dante applies this rule to those vowels having the umlaut symbol above. The opposite of this rule is *Synaeresis*[7], that is a phonological process of sound change in which two adjacent vowels that create a hiatus within a word are combined into a single syllable as if they were a diphthong. We applied it to the last, toned model, because we noticed that Dante regularly uses this rule in case of words like *'tùo', 'tùa', 'tùe', 'sùo', 'sùa', 'sùe', 'mìo', 'mìa', 'mìe'*.

Further on, we decided to implement by ourselves also grammatical syllabification techniques, in addition to the poetry rules. In particular, we wrote our own definition of hiatus and diphthong in order to examine adjacent vowels properly and we used *regexes* to create tools for the realization of Italian syllabification.

1.2. Deep Learning Techniques and Tools

As mentioned, our task had to be achieved using Deep Learning techniques only. In particular, it was clear we had to include in our network implementation some specific layers such as *Recurrent Neural Networks*. Eventually, we also had to propose some metrics to evaluate the quality of the generated text.

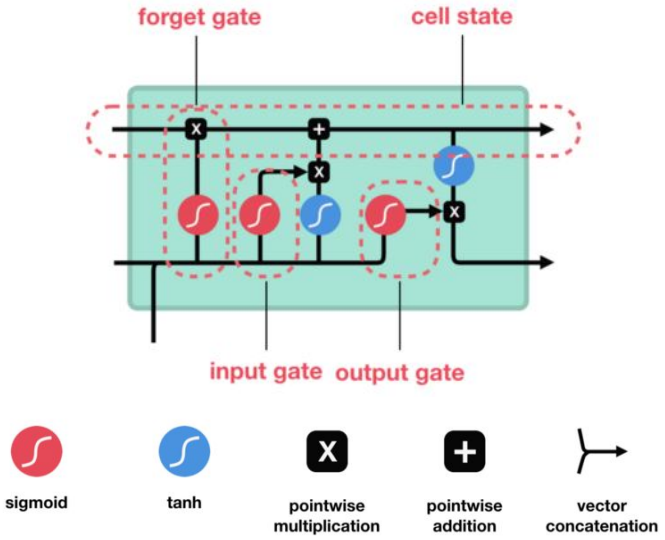
TensorFlow is the software library that we chose for implementing this project.

[4] https://www.treccani.it/enciclopedia/sinalefe_(Enciclopedia-dell'Italiano)/
[5] https://www.treccani.it/enciclopedia/dialefe_%28Enciclopedia-Dantesca%29/
[6] https://www.treccani.it/enciclopedia/sineresi_%28Enciclopedia-Dantesca%29/
[7] https://www.treccani.it/enciclopedia/dieresi

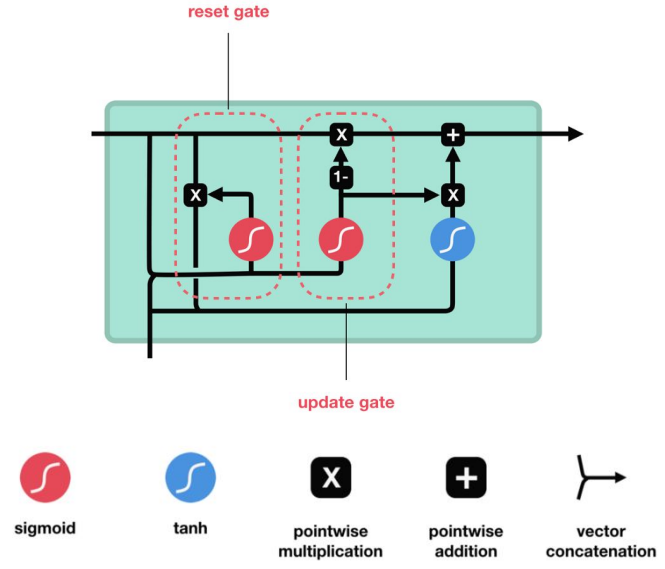
1.2.1. Layers

Typical examples of RNN architectures are the LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) layers. Here is a brief description to these architectures[8]:

- **LSTM**: this network contains several units which have feedback connections and a unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. More specifically, the forget gate decides what information should be thrown away or kept based on the current input and the hidden state of the previous unit. Then the input gate aims to update the cell state, adding the information of the current input to the cell state. At the end, the output gate decides what the next hidden state should be.



- **GRU**: this network is simpler than a LSTM, indeed each unit contains only two gates: the update gate decides what information to throw away and what new information to add while the reset gate is used to decide how much past information to forget. In this case, the hidden state is used to transfer information between units.



[8] https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

1.2.2. Tensorflow Callbacks

TensorFlow offers a complete set of special function and utilities named callbacks[9] that are essential during the training process. These functions provide a strong control tool over many different stages of execution of the training.
In particular, we used:

- ModelCheckpoint: this callback saves the Keras model or model weights at some frequency (*tf.keras.callbacks.ModelCheckpoint*).
- EarlyStopping: this callback stops training's execution when a monitored metric has stopped improving (*tf.keras.callbacks.EarlyStopping*). In particular, we used it depending on loss over validation set.
- ReduceLROnPlateau: this callback reduces learning rate when a metric has stopped improving (*tf.keras.callbacks.ReduceLROnPlateau*). In particular, we used it starting from a maximum learning rate of 0.01 to a minimum of 0.0005 depending on loss over validation set.
- CSVLogger: this callback streams epoch results to a CSV file in order to save the models' features and weights (*tf.keras.callbacks.CSVLogger*).
- TensorBoard: this callback enables visualizations for TensorBoard, the visualization tool provided with TensorFlow (*tf.keras.callbacks.TensorBoard*).

1.2.3. Other aspects

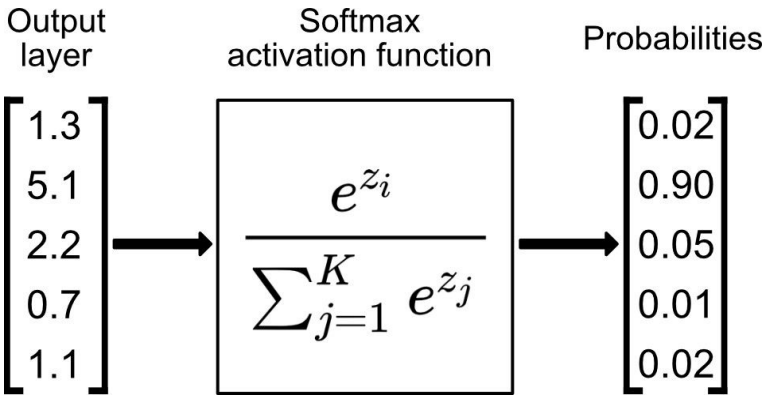
In order to train a Deep Learning model, the typical optimization algorithm used for updating the network's weights is the *Back propagation algorithm*. It works jointly with an iterative optimization algorithm for finding a local minimum of a differentiable function. Among the existing optimization algorithms, we chose the Tensorflow implementation of Adam algorithm (*tf.keras.optimizers.Adam*), that combines the advantages of other optimizers, and then we needed a minimization cost function during the training phase. We chose the *Categorical Crossentropy* as loss function for different reasons: firstly, thinking of text generation's task, this problem can be viewed as a prediction of a class, which is, in this case, the next unit of text to generate, plus the *Categorical Crossentropy* is very frequently applied in literature to this kind of task. A second motivation is given by the fact that the chosen loss function penalizes a lot the model when it makes a prediction mistake while learning, with the consequence of a significative alteration of the network's[10] weights towards improvement.

We wanted to point out that the *Categorical Crossentropy* can be used if the targets which the network should predict are provided in a one-hot-encoded representation. We discarded the idea of one-hot-encoding due to memory occupation and found out the *tf.keras.losses.SparseCategoricalCrossentropy*, having the same advantages mentioned above and where the target of the network can be provided as an integer.

Another important function in a Neural Network layer is the *activation function*, applied as the last mathematical calculation to the layer's output. In particular, since we have already said that the text generation task can be treated as a classification, the *softmax activation function* is one of the most used to face this task because the values returned by the *softmax*

[9] https://blog.paperspace.com/tensorflow-callbacks/
https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/Callback
[10] https://www.youtube.com/watch?v=gIx974WtVb4

function represents a “confidence score” for each class, a value between 0 and 1. The predicted class is, therefore, the item in the list where confidence score is the highest.
Moreover, we wanted to keep our networks’ output as much human understandable as we could and the *softmax* helped us to get this objective, transforming the output of a layer into classes’ propabilities.



2. BASELINE MODELS

Our investigation started with the deployment of three baseline models. These models are implemented with the main difference one to another of using different units of text: by words, by characters and by syllables. These had been our first approaches to the text generation task, starting from the Divine Comedy text.

2.1. Model by word

This implementation aimed to feed the model with text in the format of words from the original Divine Comedy as units of text, in order to train the model through words and their relation, hopefully learning which is the correct association of words to get a correct and well formed verse according to the original text. Here, an important role was played by the data processing approaches applied, since we had to properly prepare the dataset and get the shape we needed for the input sequences of our model.

2.1.1. Data processing

Since the *Model by words* had been one of the very first approaches to the Deep Comedy Project, the preprocessing and the manipulation of the raw data of the Divine Comedy had been important roles.
First in first, it has to be highlighted that, from now on, the following basic data preprocessing approaches will be applied to all our future models.
The basic approach concerns removing all the *cantica*'s titles and the empty lines, and further processing has been added based on the needs of the different strategies.

This model, for instance, required some specific processing in order to correctly isolate each word of the verses and to clean the original data from punctuation and unwanted toned characters. We also decided to add some special tokens to the raw data in order to ease the learning of the structure of the Divine Comedy to the model. This last approach is another common strategy among all the models. These are the applied special tokens:

SPECIAL TOKENS	
'START_OF_CANTO'	*'<start_of_canto>'*
'END_OF_CANTO'	*'<end_of_canto>'*
'START_OF_TERZINA'	*'<start_of_terzina>'*
'END_OF_TERZINA'	*'<end_of_terzina>'*
'END_OF_VERSO'	*'<end_of_verso>'*

Below is an example of the preprocessed text with its special tokens:

```
<start_of_canto> <start_of_terzina> nel mezzo del cammin di nostra vita
<end_of_verso> mi ritrovai per una selva oscura <end_of_verso> chè la diritta via
era smarrita <end_of_verso> <end_of_terzina> <start_of_terzina> ahi quanto a dir
qual era è cosa dura <end_of_verso> esta selva selvaggia e aspra e forte
<end_of_verso> che nel pensier rinova la paura <end_of_verso> <end_of_terzina>
<start_of_terzina> tant' è amara che poco è più morte <end_of_verso> ma per
trattar del ben ch'i' vi trovai <end_of_verso> dirò de l'altre cose ch'i' v'ho
scorte <end_of_verso> <end_of_terzina> <start_of_terzina> io non so ben ridir
com' i' v'intrai <end_of_verso> tant' era pien di sonno a quel punto
<end_of_verso> che la verace via abbandonai <end_of_verso> <end_of_terzina>
```

Each sample of the dataset, built from the previous text, is composed of an input sequence and an output sequence. The former is provided as input to the model, while the latter is shifted of one token in order to teach to the model which is the token that follows the input one. Here is an example:

INPUT SEQUENCE	**OUTPUT SEQUENCE**
<start_of_canto> <start_of_terzina> nel mezzo del cammin di nostra vita <end_of_verso> mi ritrovai per una selva oscura <end_of_verso> chè la diritta via era	<start_of_terzina> nel mezzo del cammin di nostra vita <end_of_verso> mi ritrovai per una selva oscura <end_of_verso> chè la diritta via era **smarrita**

Of course, these words had to be represented as integers to be digested and elaborated by a neural network, so we created three python *dictionaries* from the text we had as input of this network and we saved them in *json* format in order to conveniently represent the data. Since the dictionaries are built over the input text, of course these dictionaries contain every word present in the text of the Divine Comedy, which are 13731 unique words.
Here is a sample of this model's dictionaries:

```
{
   "vocab": [
      '<end_of_canto>', '<end_of_terzina>',
      '<end_of_verso>', '<start_of_canto>',
      '<start_of_terzina>', 'a', "a'",
      'ab', 'abate', 'abbaglia', 'abbandona',
      … ],

   "idx2text": {
      '202': '<end_of_canto>', '203': '<end_of_terzina>',
      '204': '<end_of_verso>', '205': '<start_of_canto>',
      '206': '<start_of_terzina>', '207': 'a', '208': "a'",
      '209': 'ab', '210': 'abate', '211': 'abbaglia', '212': 'abbandona',
      … },

   "text2idx": {
      '<end_of_canto>': 202, '<end_of_terzina>': 203,
      '<end_of_verso>': 204, '<start_of_canto>': 205,
      '<start_of_terzina>': 206, 'a': 207, "a'": 208,
      'ab': 209, 'abate': 210, 'abbaglia': 211, 'abbandona': 212,
      … }
}
```

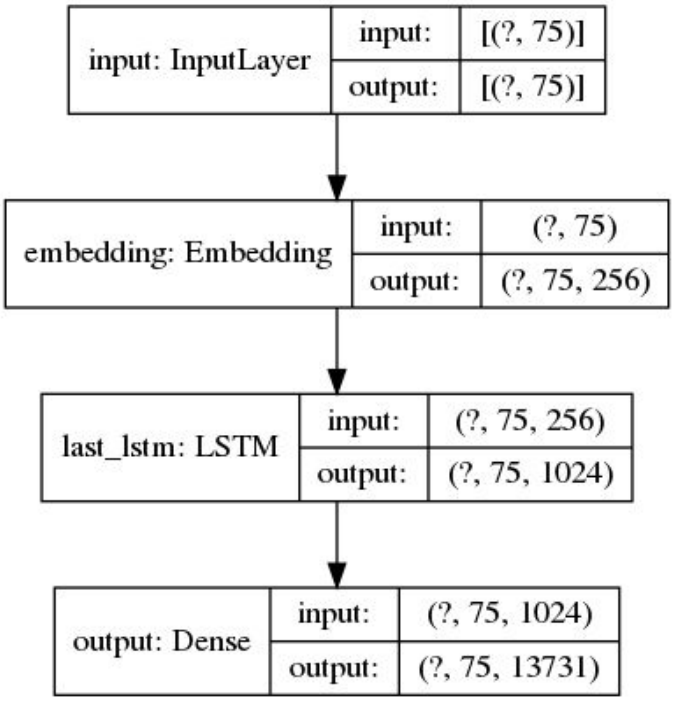
2.1.2. Model architecture

It is important to highlight that we had to embed our words to correctly feed our network.
We developed three different models to process this approach, meaning that we designed our first model with different components and combinations of these in order to assess how these aspects affected the model behaviour and they could possibly improve or worsen our evaluations.
In the input layer, we input a sequence of 75 embedded words, in order to have in output the predicted succession of words following the input sequence. The choice of using 75 units of words and special tokens is that we considered it a valuable number to feed the network and let it learn and output the text and rhyme structure because it includes more or less two *terzine*.

Model 1

- **Input layer** kicks off the elaboration of the input sequence. In this case, input sequences have a length of 75, as previously explained;
- **Embedding layer** learns a high-dimensional representation of each word, to provide a better input sequence to the next layer. The size of the embedding is set to 256;
- **LSTM layer** processes the sequence of words and it learns the succession of them and special tokens cadence to understand and learn how to compose verses and text structure. The layer's parameter is the number of RNN units, set to 1024;
- **Dense layer** is the last and a fully-connected layer aimed to give out the probabilities of next words. The activation function used within this layer is *softmax*. The number of nodes is equal to the vocabulary size, 13731.

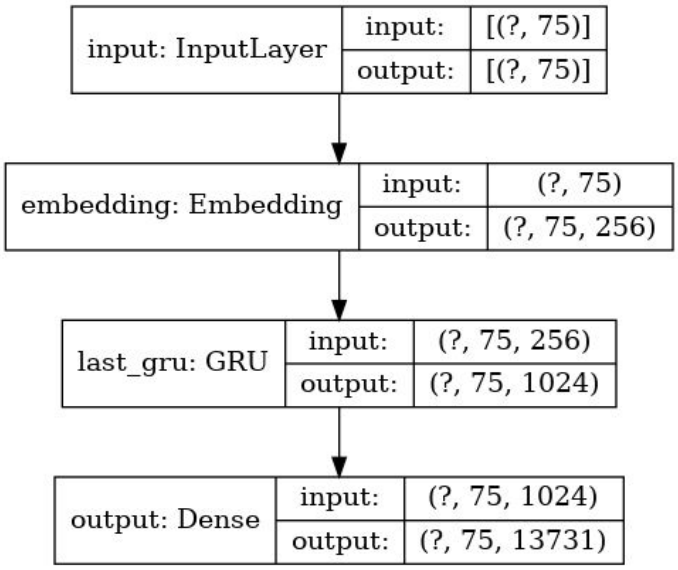
Below a chart showing the model structure and the dimensionality of each layer.



Model 2

- **Input layer** kicks off the elaboration of the input sequence. In this case, input sequences have a length of 75, as previously explained;
- **Embedding layer** learns a high-dimensional representation of each word, to provide a better input sequence to the next layer. The size of the embedding is set to 256;
- **GRU layer** processes the sequence of words and it learns the succession of them and special tokens cadence to understand and learn how to compose verses and text structure. The layer's parameter is the number of RNN units, set to 1024;
- **Dense layer** is the last and a fully-connected layer aimed to give out the probabilities of next words. The activation function used within this layer is *softmax*, as in the previous cases. The number of nodes is equal to the vocabulary size, 13731.

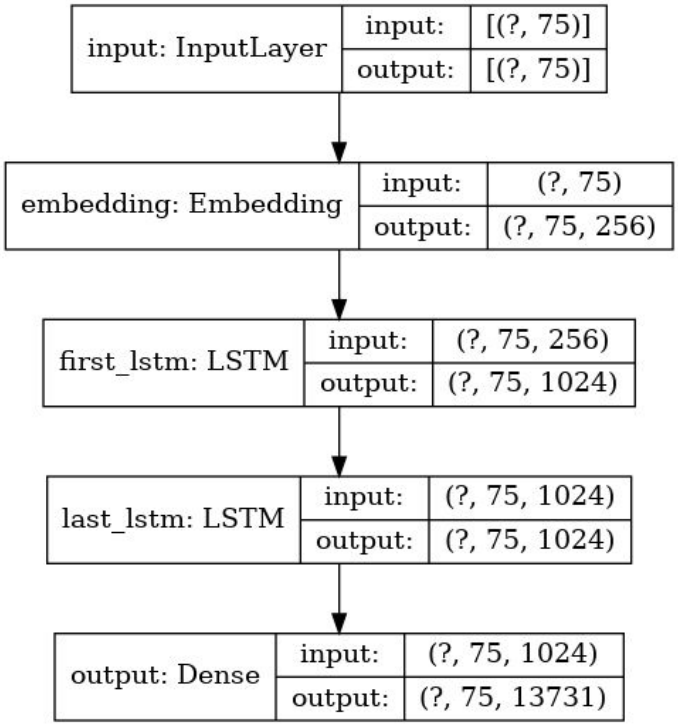
Below is a chart showing the model structure and the dimensionality of each layer:



Model 3

- **Input layer** kicks off the elaboration of the input sequence. In this case, input sequences have a length of 75, as previously explained;
- **Embedding layer** learns a high-dimensional representation of each word, to provide a better input sequence to the next layer. The size of the embedding is set to 256;
- **LSTM layer** processes the sequence of words and it learns the succession of them and special tokens cadence to understand and learn how to compose verses and text structure. The layer's parameter is the number of RNN units, set to 1024;
- **LSTM layer** processes the sequence of words and it learns the succession of them and special tokens cadence to understand and learn how to compose verses and text structure. The layer's parameter is the number of RNN units, set to 1024;
- **Dense layer** is the last and a fully-connected layer aimed to give out the probabilities of next words. The activation function used within this layer is *softmax*, as in the previous cases. The number of nodes is equal to the vocabulary size, 13731.

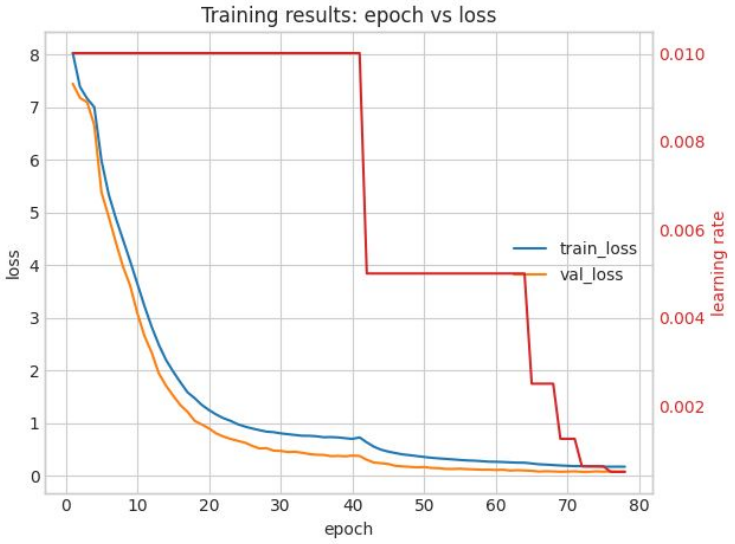
Ahead is a chart showing the model structure and the dimensionality of each layer.

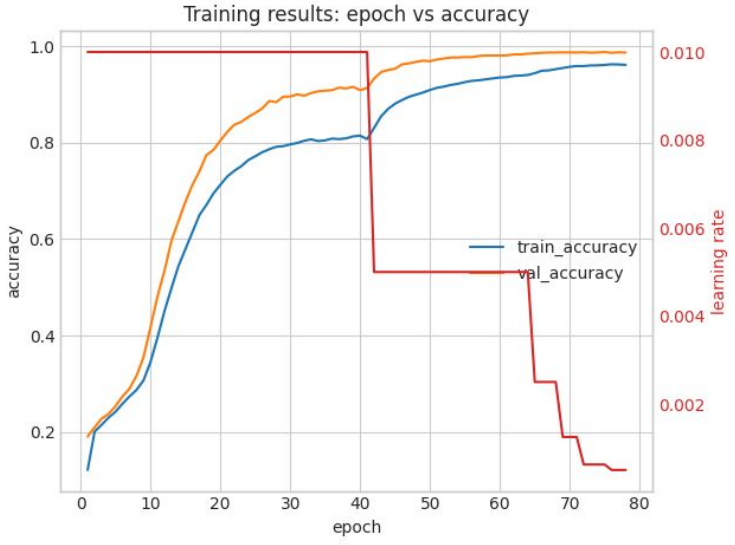


2.1.3. Training and graphics

In the training phase, that is the exact moment in which our network learns what to output, the same parameters were used across the three different implementations of the model.
We had set at most 200 epochs of execution, a batch size of 32 and the *Sparse Categorical Crossentropy* as loss function. As mentioned, we monitored the metrics over the validation set. In particular, we used *ReduceLROnPlateau* callback to refine the learning rate during the training and stopped it when the model did not improve its performance anymore through the usage of *EarlyStopping* callback.

Model 1 (Embedding - LSTM - Dense)

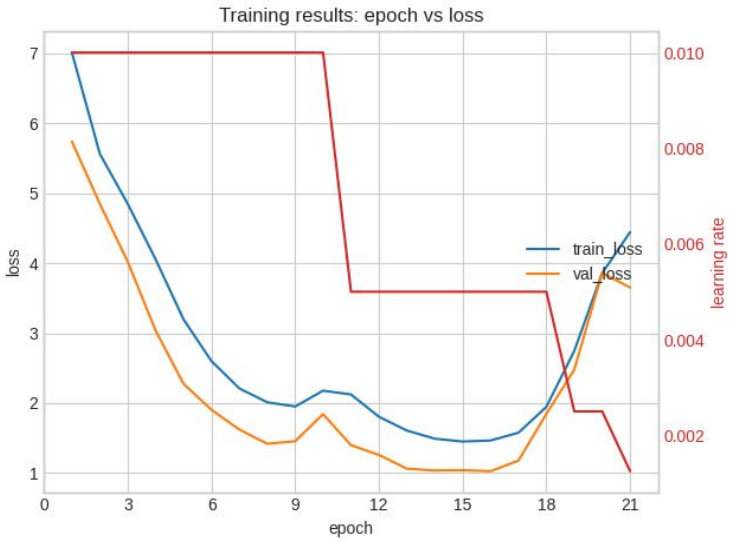


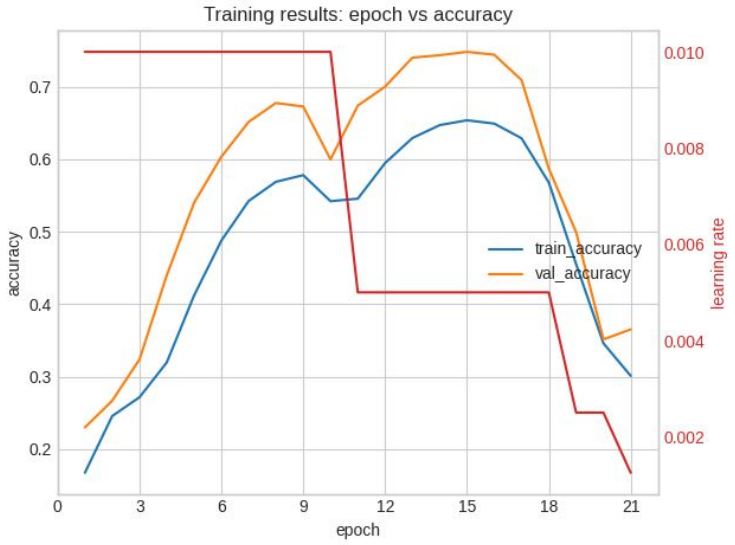


Training stopped after 78 epochs achieving a training accuracy of about 97% and validation accuracy of about 99%.
The accuracy suggests that the model behaves perfectly. However, we found that this metric might be not so representative for our models because this was such a particular task.

We defined our policy to have a look at the final output in order to have an understanding of how the models behave, believing that it would be the best way to evaluate such models. Samples to understand this model's behaviour are presented in the *Text generation* section.

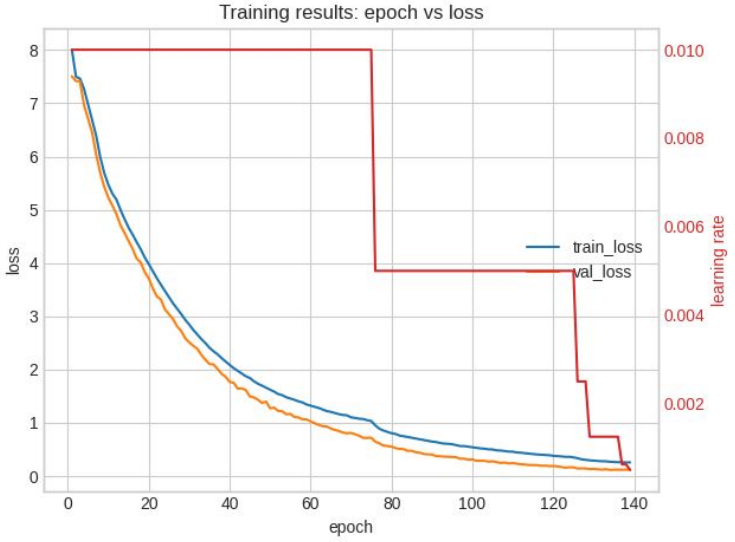
Model 2 (Embedding - GRU - Dense)

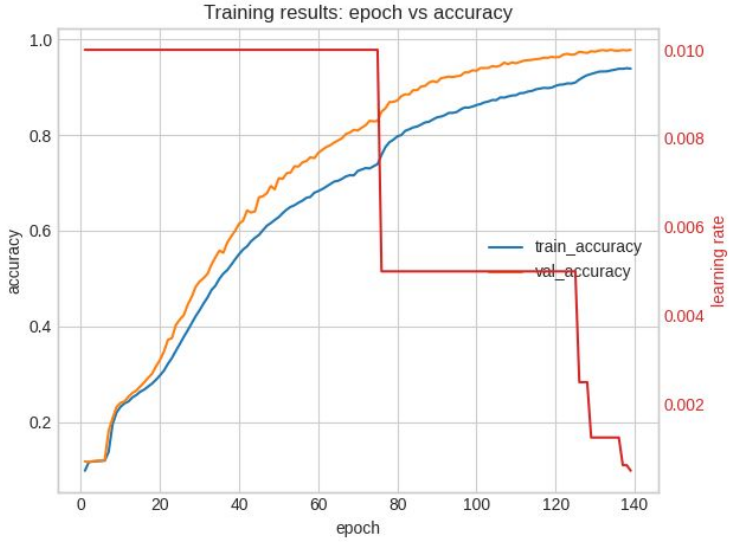




Training stopped after 21 epochs achieving a training accuracy of about 65% and validation accuracy of about 75%.
Even in this case, the accuracy suggests that the model behaves quite well, anyway we found that this metric might be not so representative for our models because this was such a particular task. As said for the previous model, our policy is to have a look at the final output in order to have an understanding of how the models behave. Samples to understand this model's behaviour are presented in the *Text generation* section.

Model 3 (Embedding - LSTM - LSTM - Dense)





Training stopped after 140 epochs achieving a training accuracy around 97% and validation accuracy around 99%.
Even in this case, the accuracy suggests that the model behaves perfectly, however we found that this metric might be not so representative for our models because this was such a particular task. As said for the previous model, our policy is to have a look at the final output in order to have an understanding of how the models behave. Samples to understand this model's behaviour are presented in the *Text generation* section.

2.1.4. Text generation

Finally, to get our generated *canto*, we used our model in a loop in order to get each time a new word to append to our generated *canto*. The input sequences for our model will be picked from the last verses of a random *canto* of the Divine Comedy.
Since we had a sequence-to-sequence model, we initially applied the *argmax* function over the last output element of the sequence to get the most probable word that should follow the current input of the model. However, we immediately noticed that, in the generated text, we

often got a repetition of the same verses, which we were not expecting. In order to avoid this behaviour, instead of always picking the most probable word, as it performs applying the *argmax*, we found that a more suitable approach could be to choose the next word on the basis of the probability distribution (the output of the model) starting from all the possible words in our vocabulary. Following this methodology, we did not select the word that had deterministically the highest score, but instead the choice was realized using *numpy.random.choice*[11] function, which has a double behaviour: it can sample from a uniform distribution where all the elements have the same probability to be chosen or, as we needed, sample from a non-uniform distribution letting some element to be selected more easily than the others. We observed that our model was still able to generate a correct sequence of words and, in addition, it did not get stuck in a loop as the previous situation being now able to keep generating different text.
In the end the main approach to let the model end the generation of text, according to the Divine Comedy, should be to look for the *'<end_of_canto>'* special token and then stop the generation. In addition we also looked for a more suitable approach to get our end of *canto* within a reasonable number of verses whenever it does not meet the *'<end_of_canto>'* , therefore we put a maximum limit of 151 verses where the model should stop anyway the text generation.

Here a sample of the final output of the model by word using a single LSTM layer (Model 1), our showcase choice is based on the best results showed by the previous graphs:

CANTO

```
oppresso di stupore a la mia stella
mi volsi come parvol che ricorre
sempre colà dove più si confida

e quella che gennaio tutto si sverni
li occhi rivolgi al logoro che gira
lo rege etterno con le rote magne

aguzza qui lettor ben si sale
oh felice che tosto la fortuna
che la dolcezza in cui dolor non era

ell' è la prima che dal gran giovanni
de la natura del ciel si fece
tu che la sua natura diede aperta

di tutte queste dote s'avvantaggia
l'umana creatura e s'una manca
di sua nobilità convien che caggia

solo il peccato è quel che la terra
e la donna di lingua che sanno
seder tra filosofica famiglia

tutti si donna che per tu tu credi
e io e guarda non che tu credi
e io a loro e a' miei conti
[...]
```

[11] https://numpy.org/doc/stable/reference/random/generated/numpy.random.choice.html

2.2. Model by character

This implementation aimed to feed the model with text in the format of character from the original Divine Comedy as units of text, in order to let the model be trained and hopefully be able to learn which is the correct succession of letters and get the correct word according to what is present in the original text. Here, an important role was played by the data processing approaches applied, since we had to properly prepare the dataset and get the shape we needed for the input sequences of our model.

2.2.1. Data processing

For the *Model by character* data processing we applied the basic data cleaning approach already explained earlier in the document, adding some differences to correctly tell apart each character and to not remove punctuation and toned characters since here are important for the structure of the output of this model.
Again, we decided to add some special tokens to the raw data in order to let the model learn the structure of the Divine Comedy, in this case, the special tokens were slightly different from the previous model, since we had the need to represent them as single characters:

SPECIAL TOKENS	
'START_OF_CANTO'	'+'
'END_OF_CANTO'	'@'
'START_OF_TERZINA'	'$'
'END_OF_TERZINA'	'#'
'END_OF_VERSO'	'&'

Below is an example of the preprocessed text with special tokens:

```
+$nel mezzo del cammin di nostra vita&mi ritrovai per una selva oscura,&chè la
diritta via era smarrita.&#$ahi quanto a dir qual era è cosa dura&esta selva
selvaggia e aspra e forte&che nel pensier rinova la paura!&#$tant' è amara che
poco è più morte;&ma per trattar del ben ch'i' vi trovai,&dirò de l'altre cose
ch'i' v'ho scorte.&#$io non so ben ridir com' i' v'intrai,&tant' era pien di
sonno a quel punto&che la verace via abbandonai.&#
```

Each sample of the dataset, built from the previous text, is composed of an input sequence and an output sequence. The former is provided as input to the model while the latter is shifted of one token in order to teach to the model which is the token that follows the input one.

INPUT SEQUENCE	**OUTPUT SEQUENCE**
+$nel mezzo del cammin di nostra vit	$nel mezzo del cammin di nostra vit**a**

Of course, these characters had to be represented as integers to be digested and elaborated by a neural network, so we created three python *dictionaries* from the text we had as input of this network and we saved them in *json* format in order to conveniently represent the data. Since the dictionaries are built over the input text, of course these dictionaries contain every character present in the text of the Divine Comedy, which are 44 unique characters.
Here is a sample of this model's dictionaries:

```
{
   "vocab": [
      '+', ',', '-', '.', ':', ';',
      '?', '@', 'a', 'b', 'c', 'd',
      'e', 'f', 'g', 'h', 'i', 'j',
      … ],

   "idx2text": {
      '7': '+', '8': ',', '9': '-', '10': '.', '11': ':', '12': ';',
      '13': '?', '14': '@', '15': 'a', '16': 'b', '17': 'c', '18': 'd',
      '19': 'e', '20': 'f', '21': 'g', '22': 'h', '23': 'i', '24': 'j',
      … },

   "text2idx": {
      '+': 7, ',': 8, '-': 9, '.': 10, ':': 11, ';': 12,
      '?': 13, '@': 14, 'a': 15, 'b': 16, 'c': 17, 'd': 18,
      'e': 19, 'f': 20, 'g': 21, 'h': 22, 'i': 23, 'j': 24,
      … }
}
```

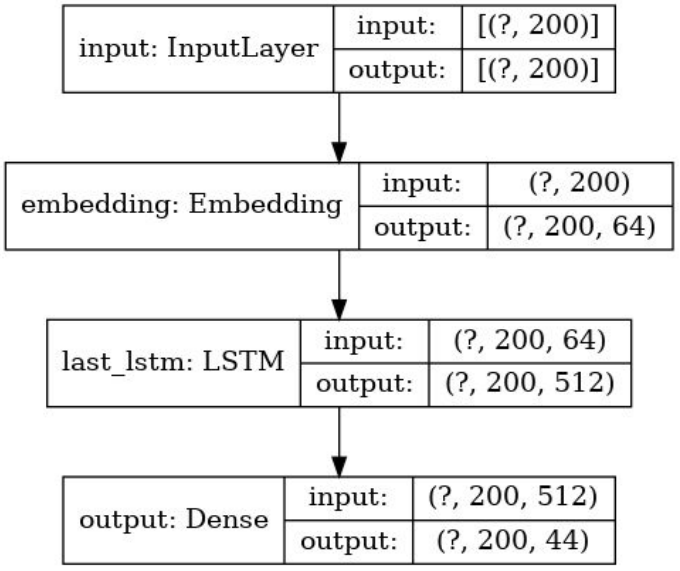
2.2.2. Model architecture

It is important to highlight that we had to embed our characters to correctly feed our network. We developed three different models to process this approach, meaning that we designed our first model with different components and combinations of these in order to assess how these aspects affected the model behaviour and they could possibly improve or worsen our evaluations.
In the input layer, we fed a sequence of 200 embedded characters in order to have in output the predicted single character following the input sequence. The choice of using 200 sequences of characters and special tokens is because we considered it a valuable number, more in the specific two *terzine*, to feed the network and let it learn and output the text and rhyme structure.

Model 1

- **Input layer** kicks off the elaboration of the input sequence. In this case, input sequences have a length of 200, as previously explained;
- **Embedding layer** learns a high-dimensional representation of each single character, to provide a better input sequence to the next layer. The embedding dimension parameter is set to 64;
- **LSTM layer** processes the sequence of characters and it learns the succession of them along with special tokens to understand and learn how to compose words and text structure. The parameter of the layer is the number of RNN units, set to 512;
- **Dense layer** is the last and a fully-connected layer aimed to give out the probabilities of next chars. The activation function used within this layer is *softmax*, as in the previous cases. The number of nodes is equal to the vocabulary size, 44.

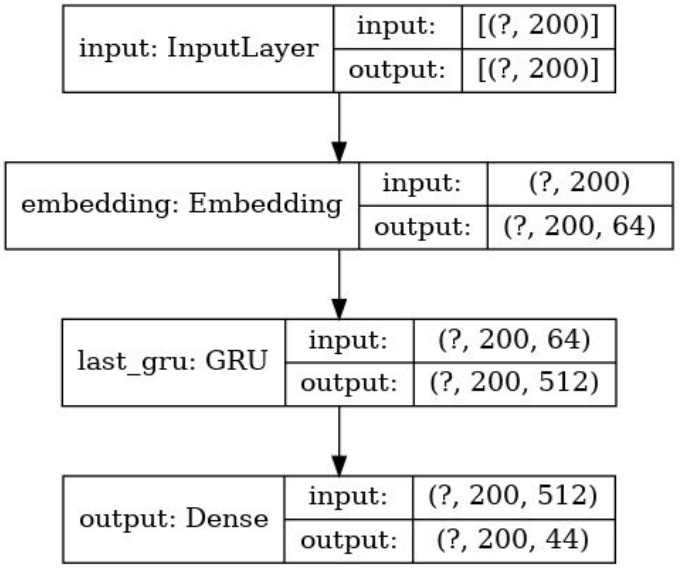
Below a chart showing the model structure and the dimensionality of each layer.



Model 2

- **Input layer** kicks off the elaboration of the input sequence. In this case, input sequences have a length of 200, as previously explained;
- **Embedding layer** learns a high-dimensional representation of each single character, to provide a better input sequence to the next layer. The embedding dimension parameter is set to 64.
- **GRU layer** processes the sequence of characters and it learns the succession of them along with special tokens to understand and learn how to compose words and text structure. The parameter of the layer is the number of RNN units, set to 512.
- **Dense layer** is the last and a fully-connected layer aimed to give out the probabilities of next chars. The activation function used within this layer is *softmax*, as in the previous cases. The number of nodes is equal to the vocabulary size, 44.

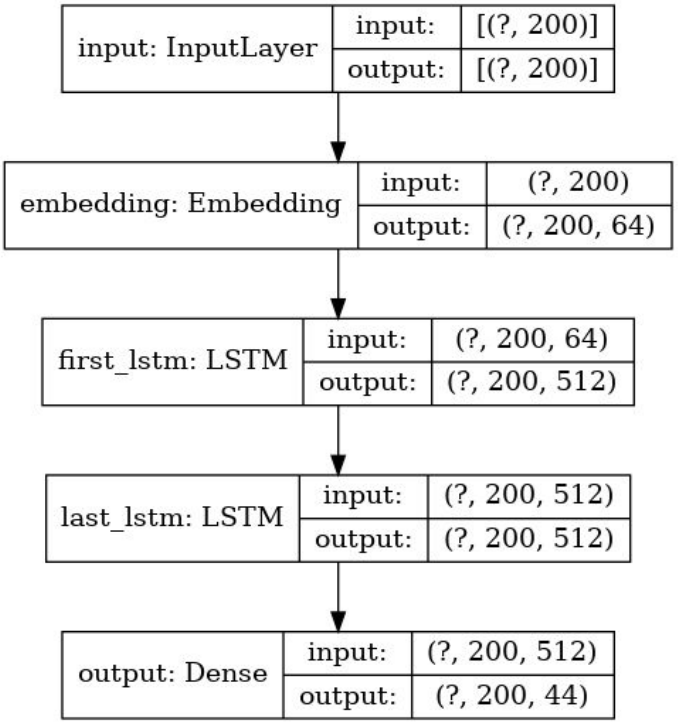
Below a chart showing the model structure and the dimensionality of each layer.



Model 3

- **Input layer** kicks off the elaboration of the input sequence. In this case, input sequences have a length of 200, as previously explained;
- **Embedding layer** learns a high-dimensional representation of each single character, to provide a better input sequence to the next layer. The embedding dimension parameter is set to 64;
- **LSTM layer** processes the sequence of characters and it learns the succession of them along with special tokens to understand and learn how to compose words and text structure. The parameter of the layer is the number of RNN units, set to 512;
- **LSTM layer** with the same purpose of the previous one. The choice of having a second LSTM is to try to get more info about the sequences going deeper with the layers;
- **Dense layer** is the last and a fully-connected layer aimed to give out the probabilities of next chars. The activation function used within this layer is *softmax*, as in the previous cases. The number of nodes is equal to the vocabulary size, 44.

Below a chart showing the model structure and the dimensionality of each layer.



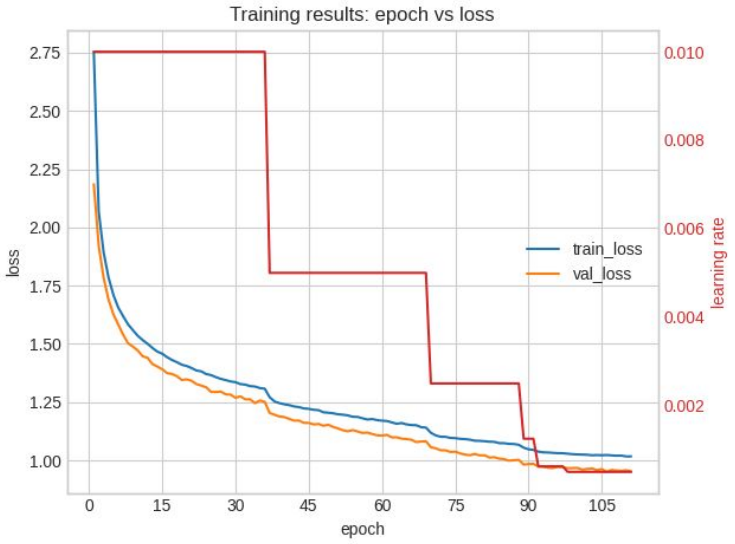
2.2.3. Training and graphics

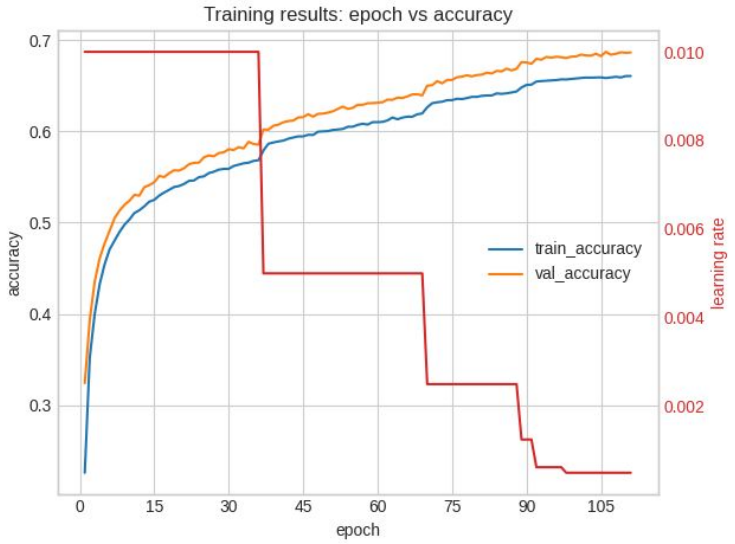
The training phase, that is the exact moment in which our network learns what to output, the same parameters are used across the three different implementations of the model.

We had set at most 200 epochs of execution, a batch size of 32 and the *Sparse Categorical Crossentropy* as loss function. As mentioned, we monitored the metrics over the validation set. In particular, we used *ReduceLROnPlateau* callback to refine the learning rate during

the training and stopped it when the model did not improve its performance anymore through the usage of *EarlyStopping* callback.

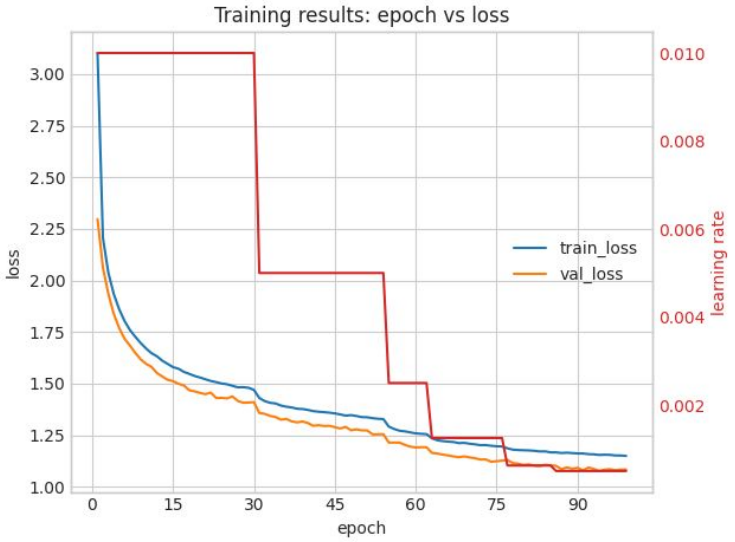
Model 1 (Embedding - LSTM - Dense)

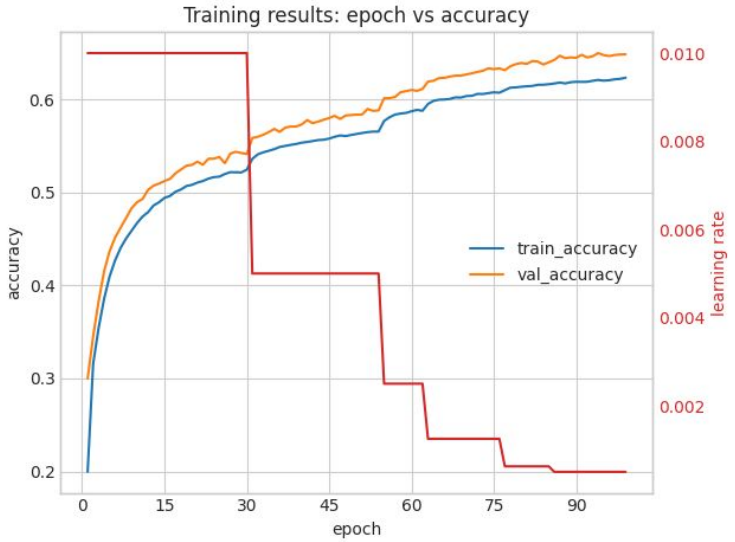




Training stopped after 110 epochs achieving a training accuracy of around 65% and validation accuracy of around 68%.
Even in this case, the accuracy suggests the model behaves quite well, anyway we found that this metric might be not so representative for our models because this was such a particular task. As said for the previous model, our policy is to have a look at the final output in order to have an understanding of how the models behave. Samples to understand this model's behaviour are presented in the *Text generation* section.

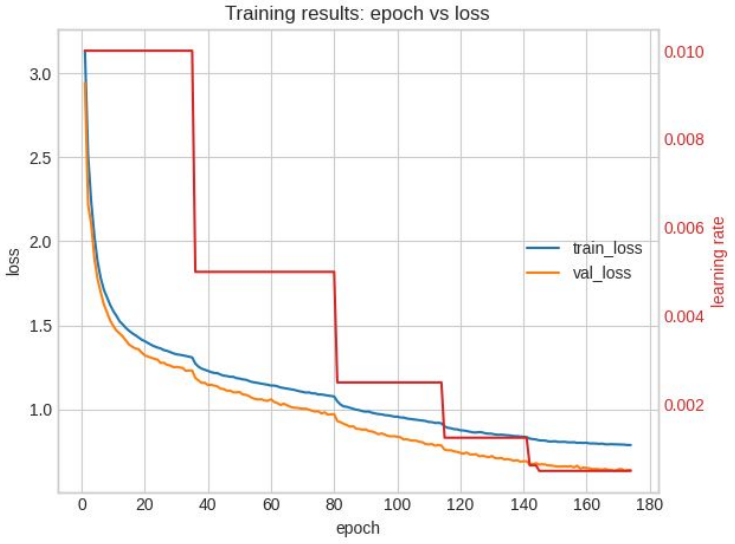
Model 2 (Embedding - GRU - Dense)

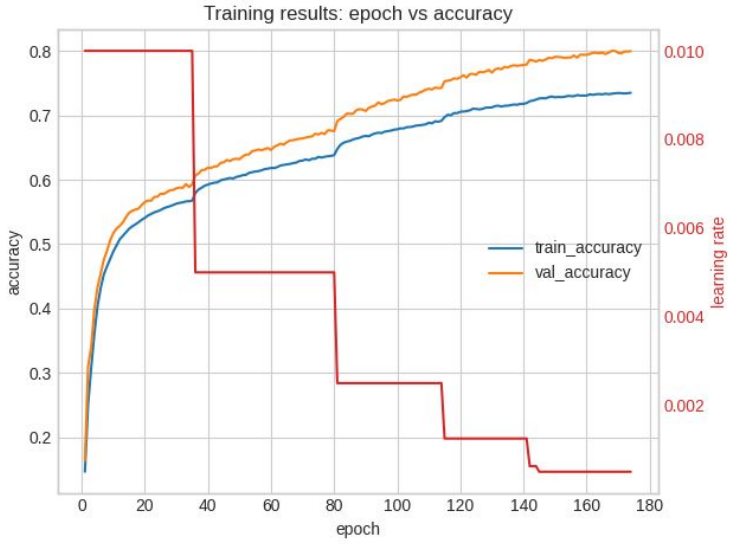




Training stopped after 98 epochs achieving a training accuracy of about 63% and validation accuracy of about 65%.
In this case, the accuracy suggests the model does not behave well as in some of the other experiments. As previously said, this metric might be not so representative for our models, therefore, our policy is to have a look at the final output in order to have an understanding of how the models behave. Samples to understand this model's behaviour are presented in the *Text generation* section.

Model 3 (Embedding - LSTM - LSTM - Dense)





Training stopped after 173 epochs achieving a training accuracy of about 73% and validation accuracy of about 80%.
Even in this case, the accuracy suggests the model behaves quite well, anyway we found that this metric might be not so representative for our models because this was such a particular task. As said for the previous model, our policy is to have a look at the final output in order to have an understanding of how the models behave. Samples to understand this model's behaviour are presented in the *Text generation* section.

2.2.4. Text generation

Finally, to get our generated *canto*, based on the assumption already made on the generating paragraph of the word model, we applied to the last element of the output sequence the *numpy.random.choice* function to sample from a non-uniform distribution letting some element, in this case characters, to be selected more easily than the others.
Again here, the main approach to let the model end the generation of text, according to the Divine Comedy structure should be to look for the *'<end_of_canto>'* special token and then stop the generation, in addition we also looked for a more suitable approach to get our end of *canto* within a reasonable number of generated verses whenever it does not meet the '*<end_of_canto>*', therefore we put a maximum limit of 151 verses where the model should stop anyway the text generation.

This is a sample of the final output of the Model by character:

CANTO

```
in su l'isola di rinfiar la spera
del padre mio, però ch'ella ebbe senno,
ch'alcuna virtù non la verso node.

l'una e l'altra un legno a te presso
attenta,
un fango, e più e 'l mio voler più
merdo;
e di pietro stare assai ci rompesse.

la seguente cosa di tutto 'l giro,
la proggia attento a vicenda guisa!
e se l'animo più sua bella guancia

a chi più d'un giacer de' loco spandi
lo duca mio discese le pecole;
chè questa natura con muse nova,

grazia mi fu' io compagni del mondo",
rispuose a me, ch'era protestiamo
e regna lunga grave in su la piuma.

ma qual sotto colui che contra mia mena
lavar le tue parole, questi fonde
hhe fai? di noi, per quella onde
l'ordigna?

ed è chi avea tanto il suo dir porse
referbate la famiglia, e più fissi;
e allor lo minor mi davansi.

[...]
```

2.3. Model by syllable

This implementation aimed to feed the model with text in the format of syllables from the original Divine Comedy as units of text, in order to let the model be trained and hopefully be able to learn which is the correct succession of syllables and get the correct word according to what is present in the original text. Here, an important role was played by the data processing approaches applied, since we had to properly prepare the dataset and get the shape we needed for the input sequences of our model.

2.3.1. Data processing

For the *Model by syllable*'s data processing, we applied the basic data cleaning approach already explained earlier in the document, adding some differences to correctly syllabify each word and to remove punctuation and toned characters.
The additional process required in this case, which consisted in syllabifying the whole text, was performed through the usage of *Pyphen* python module.

As in the previous models, we added some special tokens to the raw data in order to let the model learn the structure of the Divine Comedy. Here, the special tokens are slightly different from the previous models: we integrated the text with the same special tokens of the *Model by words*, but with the introduction of an additional special token, that helped us with the separation of consecutive syllables of different words, in order to distinguish their origin. Here is a report of the special token of this model:

SPECIAL TOKENS	
'START_OF_CANTO'	*'<start_of_canto>'*
'END_OF_CANTO'	*'<end_of_canto>'*
'START_OF_TERZINA'	*'<start_of_terzina>'*
'END_OF_TERZINA'	*'<end_of_terzina>'*
'END_OF_VERSO'	*'<end_of_verso>'*
'WORD_SEP'	*'<word_sep>'*

Below an example of the preprocessed text with special tokens:

```
<start_of_canto>
<start_of_terzina>
nel <word_sep> mezzo <word_sep> del <word_sep> cammin <word_sep> di <word_sep>
nostra <word_sep> vita <end_of_verso>
mi <word_sep> ritrovai <word_sep> per <word_sep> una <word_sep> selva <word_sep>
oscura <end_of_verso>
ché <word_sep> la <word_sep> diritta <word_sep> via <word_sep> era <word_sep>
smarrita <end_of_verso>
<end_of_terzina>
<start_of_terzina>
ahi <word_sep> quanto <word_sep> a <word_sep> dir <word_sep> qual <word_sep> era
<word_sep> è <word_sep> cosa <word_sep> dura <end_of_verso>
esta <word_sep> selva <word_sep> selvaggia <word_sep> e <word_sep> aspra
<word_sep> e <word_sep> forte <end_of_verso>
che <word_sep> nel <word_sep> pensier <word_sep> rinova <word_sep> la <word_sep>
paura <end_of_verso>
<end_of_terzina>
```

Each sample of the dataset, built from the previous text, is composed of an input sequence and an output sequence. The former is provided as input to the model while the latter is shifted of one token in order to teach to the model which is the token that follows the input one.

INPUT SEQUENCE	**OUTPUT SEQUENCE**
<start_of_canto> <start_of_terzina> nel <word_sep> mezzo <word_sep> del <word_sep> cammin <word_sep> di <word_sep> nostra <word_sep> vita <end_of_verso> mi <word_sep> ritro	<start_of_terzina> nel <word_sep> mezzo <word_sep> del <word_sep> cammin <word_sep> di <word_sep> nostra <word_sep> vita <end_of_verso> mi <word_sep> ritro**vai**

Of course these syllables had to be represented as integers to be digested and elaborated by a neural network, so we created three python *dictionaries* from the text we had as input of this network and we saved them in *json* format in order to conveniently represent the data. Since the dictionaries are built over the input text, of course these dictionaries contain every syllable present in the text of the Divine Comedy, which are 5791 unique syllables.
Here is a sample of this model's dictionaries:

```
{
   "vocab": [
      '<end_of_canto>', '<end_of_terzina>',
      '<end_of_verso>', '<start_of_canto>',
      '<start_of_terzina>', '<word_sep>', 'a',
      "a'", "a<word_sep>'l", 'a<word_sep>a',
      'a<word_sep>ac', 'a<word_sep>al', 'a<word_sep>am',
      'a<word_sep>e', 'a<word_sep>e<word_sep>al',
      … ],
```

```
   "idx2text": {
      '46': '<end_of_canto>', '47': '<end_of_terzina>',
      '48': '<end_of_verso>', '49': '<start_of_canto>',
      '50': '<start_of_terzina>', '51': '<word_sep>', '52': 'a',
      '53': "a'", '54': "a<word_sep>'l", '55': 'a<word_sep>a',
      '56': 'a<word_sep>ac', '57': 'a<word_sep>al', '58': 'a<word_sep>am',
      '59': 'a<word_sep>e', '60': 'a<word_sep>e<word_sep>al',
      … },

   "text2idx": {
      '<end_of_canto>': 46, '<end_of_terzina>': 47,
      '<end_of_verso>': 48, '<start_of_canto>': 49,
      '<start_of_terzina>': 50, '<word_sep>': 51, 'a': 52,
      "a'": 53, "a<word_sep>'l": 54, 'a<word_sep>a': 55,
      'a<word_sep>ac': 56, 'a<word_sep>al': 57, 'a<word_sep>am': 58,
      'a<word_sep>e': 59, 'a<word_sep>e<word_sep>al': 60,
      … }
}
```

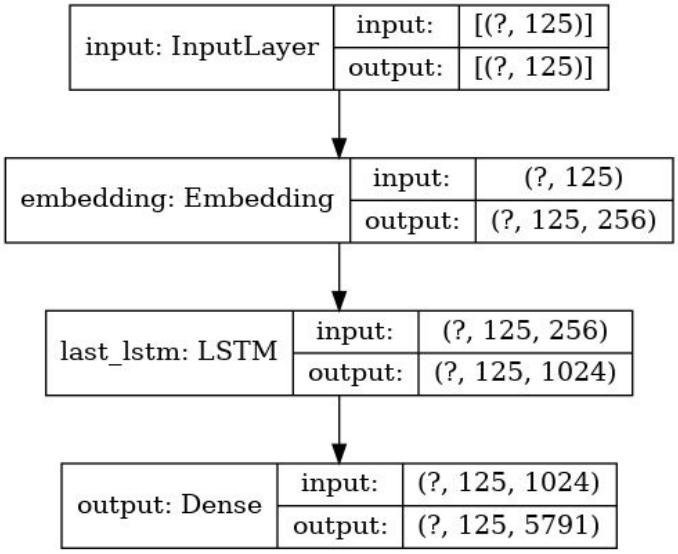
2.3.2. Model architecture

Again, it is important to highlight that we had to embed our syllables to correctly feed our network.
We developed three different models to test this approach, meaning that we modelled our first model in order to assess how the combinations of different components could improve or worsen our evaluation.
In the input layer, we input a sequence of 125 embedded syllables in order to have in output the predicted syllable following the input sequence. The choice of using 125 sequences of syllables and special tokens is because we considered it a valuable number, more in the specific two *terzine*, to feed the network and let it learn and output the text and rhyme structure.

Model 1

- **Input layer** kicks off the elaboration of the input sequence. In this case, input sequences have a length of 125, as previously explained;
- **Embedding layer** learns a high-dimensional representation of each single syllable, to provide a better input sequence to the next layer. The embedding dimension parameter is set to 256;
- **LSTM layer** processes the sequence of syllables and it learns the succession of them and special tokens cadence to understand and learn how to compose words and text structure. The layer's parameter is the number of RNN units, set to 1024;
- **Dense layer** is the last and a fully-connected layer aimed to give out the probabilities of next syllables. The activation function used within this layer is *softmax*, as in the previous cases. The number of nodes is equal to the vocabulary size, 5791.

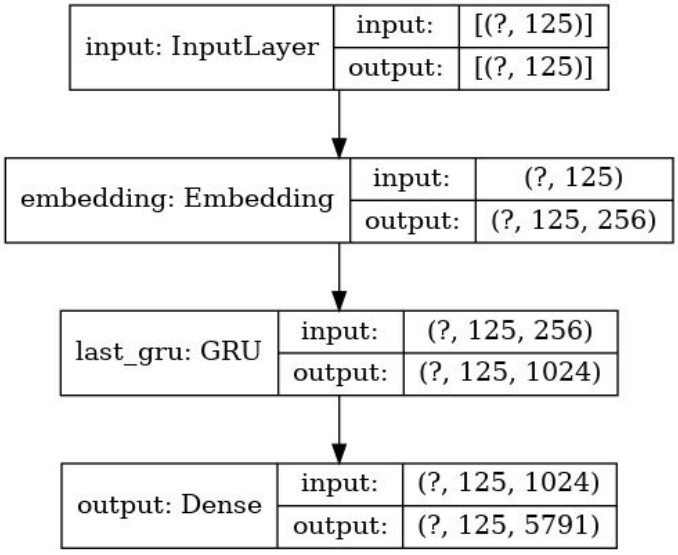
Ahead is a chart showing the model structure and the dimensionality of each layer.



Model 2

- **Input layer** kicks off the elaboration of the input sequence. In this case, input sequences have a length of 125, as previously explained;
- **Embedding layer** learns a high-dimensional representation of each single syllable, to provide a better input sequence to the next layer. The embedding dimension parameter is set to 256;
- **GRU layer** processes the sequence of syllables and it learns the succession of them and special tokens cadence to understand and learn how to compose words and text structure. The layer's parameter is the number of RNN units, set to 1024;
- **Dense layer** is the last and a fully-connected layer aimed to give out the probabilities of next syllables. The activation function used within this layer is *softmax*, as in the previous cases. The number of nodes is equal to the vocabulary size, 5791.

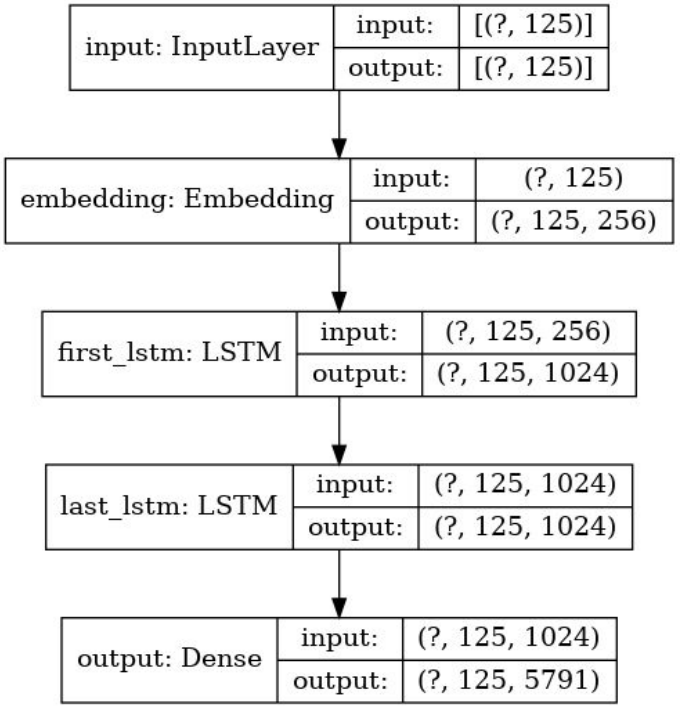
Below a chart showing the model structure and the dimensionality of each layer.



Model 3

- **Input layer** kicks off the elaboration of the input sequence. In this case, input sequences have a length of 125, as previously explained;
- **Embedding layer** learns a high-dimensional representation of each single syllable, to provide a better input sequence to the next layer. The embedding dimension parameter is set to 256;
- **LSTM layer** processes the sequence of syllables and it learns the succession of them and special tokens cadence to understand and learn how to compose words and text structure. The layer's parameter is the number of RNN units, set to 1024;
- **LSTM layer** processes the sequence of syllables and it learns the succession of them and special tokens cadence to understand and learn how to compose words and text structure. The layer's parameter is the number of RNN units, set to 1024;
- **Dense** is the last and a fully-connected layer aimed to give out the probabilities of next syllables. The activation function used within this layer is *softmax*, as in the previous cases. The number of nodes is equal to the vocabulary size, 5791.

Below a chart showing the model structure and the dimensionality of each layer.



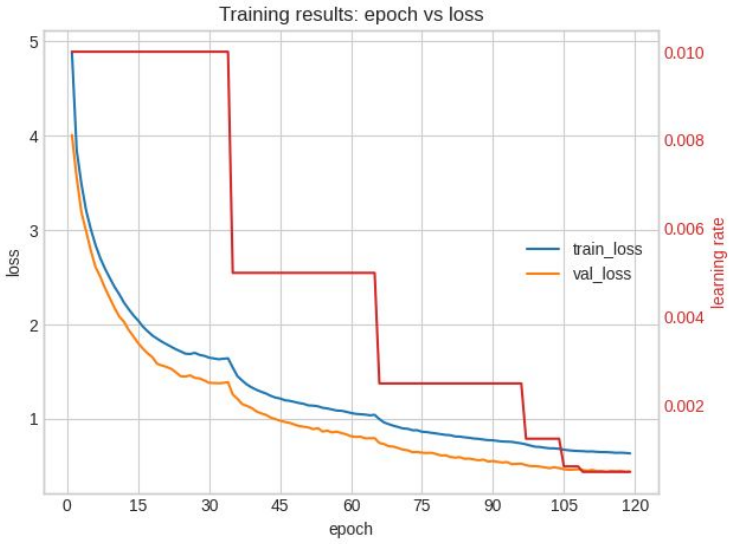
2.3.3. Training and graphics

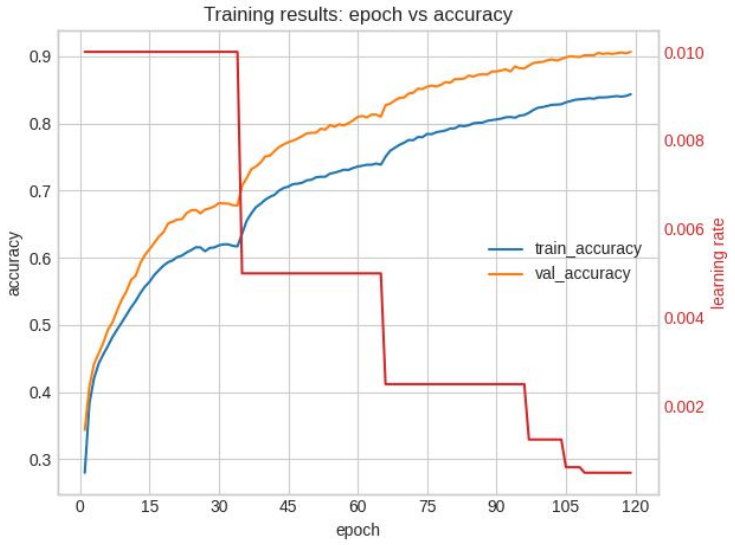
In the training phase, that is the exact moment in which our network learns what to output, the same parameters are used across the three different implementations of the model.

We had set at most 200 epochs of execution, a batch size of 32 and the *Sparse Categorical Crossentropy* as loss function. As mentioned, we monitored the metrics over the validation set. In particular, we used *ReduceLROnPlateau* callback to refine the learning rate during

the training and stopped it when the model did not improve its performance anymore through the usage of *EarlyStopping* callback.

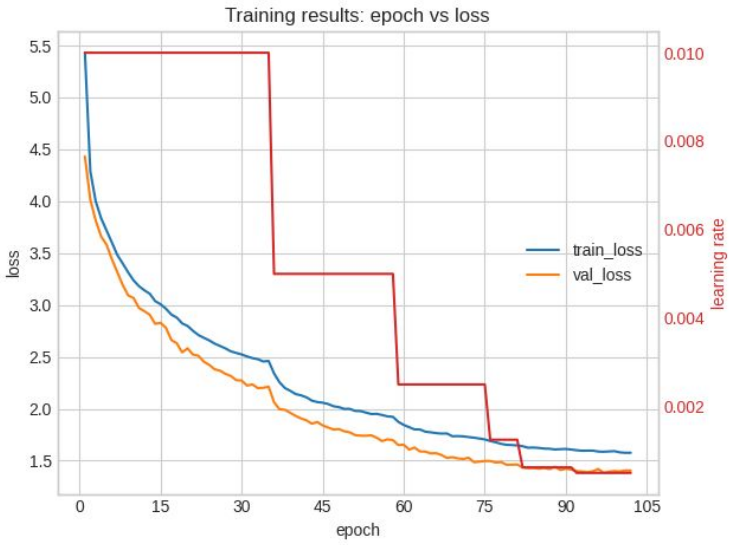
Model 1 (Embedding - LSTM - Dense)

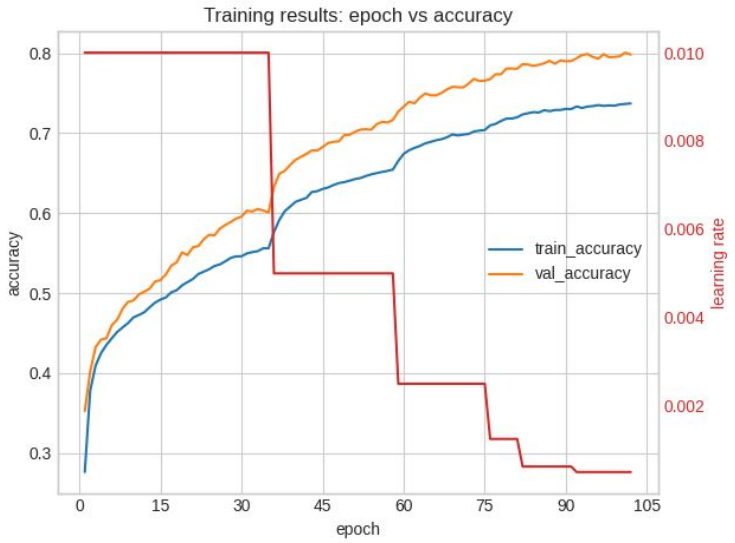




Training stopped after 120 epochs achieving a training accuracy of about 83% and validation accuracy of about 90%.
Even in this case, the accuracy suggests the model behaves very well, anyway we found that this metric might be not so representative for our models because this was such a particular task. As said for the previous model, our policy is to have a look at the final output in order to have an understanding of how the models behave. Samples to understand this model's behaviour are presented in the *Text generation* section.

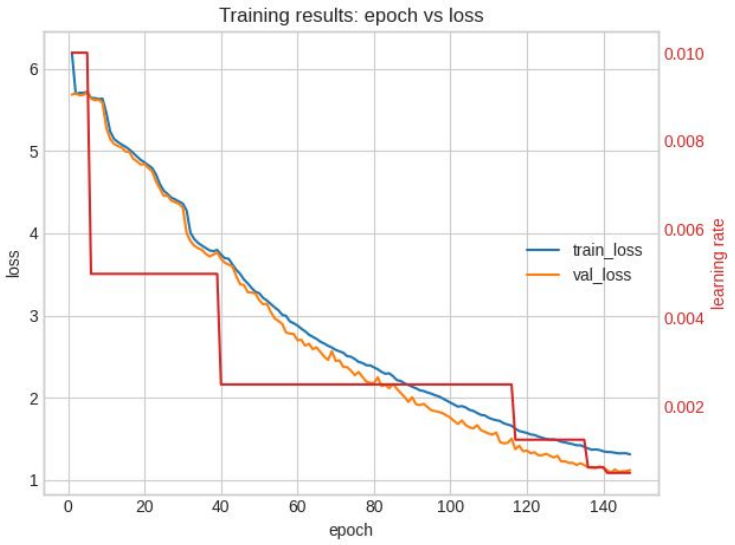
Model 2 (Embedding - GRU - Dense)

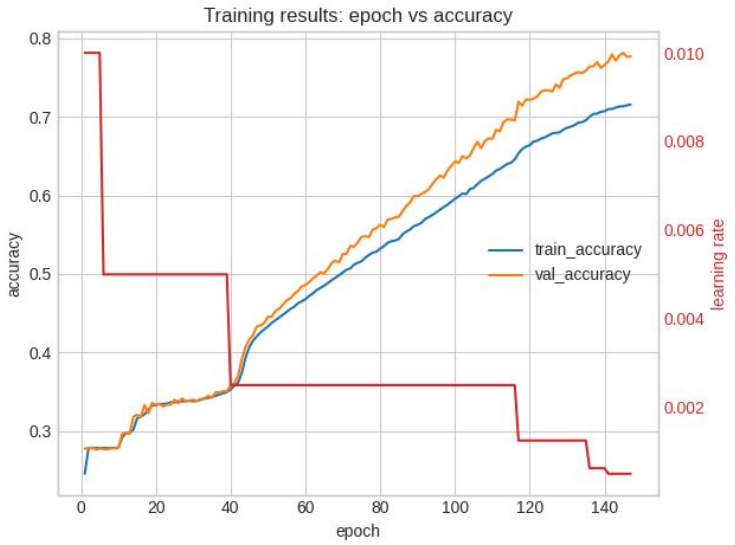




Training stopped after 105 epochs achieving training accuracy of 73% and validation accuracy of 80%.
Even in this case, the accuracy suggests the model behaves quite well, anyway we found that this metric might be not so representative for our models because this was such a particular task. As said for the previous model, our policy is to have a look at the final output in order to have an understanding of how the models behave. Samples to understand this model's behaviour are presented in the *Text generation* section.

Model 3 (Embedding - LSTM - LSTM - Dense)





Training stopped after 140 epochs achieving a training accuracy of about 72% and validation accuracy of about 79%.
Even in this case, the accuracy suggests the model behaves quite well, anyway we found that this metric might be not so representative for our models because this was such a particular task. As said for the previous model, our policy is to have a look at the final output in order to have an understanding of how the models behave. Samples to understand this model's behaviour are presented in the *Text generation* section.

2.3.4. Text generation

Finally to get our generated *canto*, based on the assumption already made for the previous models, we applied to the last element of the output sequence the *numpy.random.choice* function to sample from a non-uniform distribution letting some element, in this case characters, to be selected more easily than the others.

Again here, the main approach to let the model end the generation of text, according to the Divine Comedy, structure should be to look for the '*<end_of_canto>*' special token and then stop the generation, in addition we also looked for a more suitable approach to get our end of *canto* within a reasonable number of verses whenever it does not meet the *'<end_of_canto>'*, therefore we put a maximum limit of 151 verses where the model should stop anyway the text generation.

This is a sample of the final output of the *Model by syllable*:

```
CANTO

intra due figli a riveder le stelle
de l'animo che là sù hanno lasciasti
pur com' om fa d'onor cosa che lece

quella tenebre che la cura cala
disse 'l maestro certo là 've lo fida
e frutto ha in sé che sua qui natura

tutti ne veder ben com' io innalzo
la fiera pessima per simillа
e rotano lì pur altrimentira

con quelle altr' ombre pria disse via
molto
o dignitosa come il volto meco
che dal quarto al quinto argine è
tragetto

ma ficca li occhi a valle ché
s'approccia
la qual lunga scala con di noi
poi li addentò e l'una e l'altra
guancia

lo dosso e 'l gioco chi sì ti fallio
per difender lor ville e lor sentenza
d'i tuoi amori a dio modi che sare

e riguardai figliuolo or di lor vani
dicea fra me medesmo 'al novo cenno
di cui la gente in suo voler ne
'nvoglia

[...]
```

Finally, we were able to state that the generated texts obtained from the baseline models did not satisfy our expectations and therefore we proceeded to further investigation, ideas and implementation looking for more meaningful results.

3. ADVANCED MODELS

Once we went through a first session of text generation by the first baseline models, it was clear that some of the peculiar requirements in order to generate Divine-Comedy-like text were not so easy to obtain for such models. We found out that the most difficult aspect to accomplish by automatic neural networks' output is the rhyming structure, followed by hendecasyllabicness. In order to achieve these required characteristics, we realised a new kind of model was needed.

3.1. Model by reversed syllables

Firstly, we decided to face the issue of non-rhymeness in generated text.
Discussing about which approach could be the best to have in new experiments, we ended up with the idea of emulating the way we, as humans, would face the challenge of rhyming. All of us came up with the same hint: we would surely approach the objective of making up a rhyme in these steps: we would focus on the final letters of the word we need to rhyme with, namely, in natural language, those letters following and including the word's toned vowel, which defines a word pronunciation; then, we would think of the words we know ending up with the very same tone and following letters; eventually, we would choose the preferred alternative among the same-finishing words. For more clearance, we felt like building a rhyme could be seen as identifying the final, meaningful part of the word and, in some way, completing the rest, antecedent part of if afterwards.

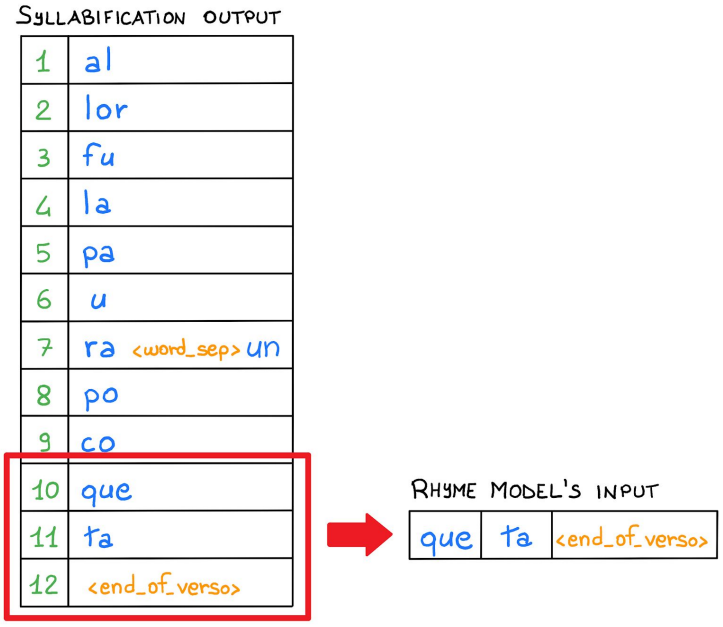
The idea for implementing this intuition by means of Neural Networks translated into two separated models each responsible for one of the two steps:

- one for the identification of the rhyme, which would be carried out in the first place: the Rhyme model;
- one for the completion of the word and, by extension, for our specific context, the verse, subject to the first model: the Verse model.

In addition, we also needed to figure out which one of these models would have to learn the structure of the *canto* (a sequence of *terzine* and a finishing solo verse). As said in the introduction, Divine-Comedy-like rhyming structure follows *terza rima* pattern, therefore, in our domain, building up rhymes comes hand in hand with *terzina* structure of the text: the model will have to learn, for each new line, which is the one to rhyme with. So, in the end, we decided to allocate the task of learning *canto*'s structure to the Rhyme model. This task consists in learning the occurrences of special tokens, as it was for previous models.

Once we made up these decisions, we had to choose, for each of these models, what data type was more suitable as input, hence what the structure of the two models would be. Looking at the previous models' generated text, all them had terrible results in rhymeness, but *Model by syllable* and *Model by character* did a little better concerning verses' lengths, so *Model by word* was not considered as a possibility. Moreover, we felt like syllable-based models would be particularly suitable in order to pursue correct hendecasyllables, since the definition of hendecasyllabicness is strictly dependent with the concept of syllable and toned syllables. For this reason, we decided to use two models by syllable.

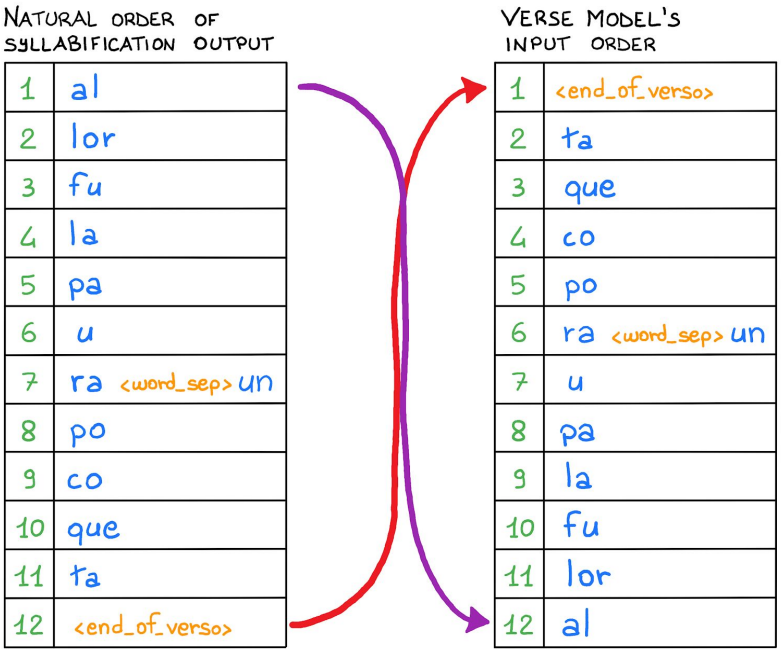
Here an example to better understand what the Rhyme model's input would look like:



Following our reasoning, we hope it comes natural to think that the whole model will firstly generate the next rhyming syllables, which, at the end of the whole process, will appear at the end of the verse. This is carried out by the Rhyme model and it happens before we have the completion of the verse itself. In fact, the Rhyme model's output will be, in some way, part of the Verse model's input, since the Verse model's generation will take place afterwards.
This is where we ran into the actual problem of this procedure…

Our new model is expected to generate separately final rhyming syllables and earlier verses' syllables. In order to preserve the creation of possibly reasonable and meaningful words, we cannot simply and superficially accept concatenating Rhyme model's generated syllables with Verse model's ones and just hope these syllables together will make sensible words.
Indeed, we generally got existing words with all the models so far because they are spawned by coherent scoring and probabilities assigned to units of text among which the models choose the next element on the basis of what we got previously, as input. This points out that we needed to figure out a reliable work-around to be able to use our freshly designed model.

What we thought of was to reverse the Verse model's input. Basically, within every verse, the order of the elements, namely the syllables, both text's ones and special tokens, would be reversed, having syllable '*<end_of_verso>*' at the beginning of every verse and the once-first syllable at the end of it. In other words, the Verse model would learn over verses that go from the end to the beginning and then it will generate verses in the same order.
Here an example to better follow the meaning of what explained:



For Neural Networks, a straight word would not make any difference from a *reversed* one, therefore we had confidence to believe that, training the model over *reversed* text and generating *reversed* words, hence verses, would create verses as reasonable as the straightly trained-and-generated ones, once reordered properly. This work-around over the text input is our solution to the generation of possibly existing words.

3.1.1. Rhyme model

This is the first model to carry out its task, which is learning and, in second place, generating the rhymes pattern and the text structure of the Divine Comedy. The Rhyme model generates only the last syllables of each verse and the structure for separation of *terzine* by predicting special tokens.

3.1.1.1. Data processing

After manipulating the raw text of the Divine Comedy, we performed both cleaning and preprocessing steps in the same way as all the other models, in order to prepare the text to build the training dataset.

We integrated the special tokens as it was for the *Model by syllable* baseline, marking the structure of the text.

Ahead is the report of the integrated special tokens, which are the same introduced in the *Model by syllable* section.

SPECIAL TOKENS	
'START_OF_CANTO'	*'<start_of_canto>'*
'END_OF_CANTO'	*'<end_of_canto>'*
'START_OF_TERZINA'	*'<start_of_terzina>'*
'END_OF_TERZINA'	*'<end_of_terzina>'*
'END_OF_VERSO'	*'<end_of_verso>'*
'WORD_SEP'	*'<word_sep>'*

Here is a sample of the re-elaborated text, after special tokens integration:

```
<start_of_canto>
<start_of_terzina>
nel <word_sep> mezzo <word_sep> del <word_sep> cammin <word_sep> di <word_sep>
nostra <word_sep> vita <end_of_verso>
mi <word_sep> ritrovai <word_sep> per <word_sep> una <word_sep> selva <word_sep>
oscura <end_of_verso>
ché <word_sep> la <word_sep> diritta <word_sep> via <word_sep> era <word_sep>
smarrita <end_of_verso>
<end_of_terzina>
<start_of_terzina>
ahi <word_sep> quanto <word_sep> a <word_sep> dir <word_sep> qual <word_sep> era
<word_sep> è <word_sep> cosa <word_sep> dura <end_of_verso>
esta <word_sep> selva <word_sep> selvaggia <word_sep> e <word_sep> aspra
<word_sep> e <word_sep> forte <end_of_verso>
che <word_sep> nel <word_sep> pensier <word_sep> rinova <word_sep> la <word_sep>
paura <end_of_verso>
<end_of_terzina>
```

For this model, we chose a sequence-to-sequence model, therefore the dataset needs to be a sequence of the specific unitary elements: single syllables. For this reason, verses are split in syllables according to Italian grammar rules implemented in the python module *Pyphen* and our synalepha implementation.

Moreover, the input of this model is formed exclusively by the final syllables of each verse, discarding the others, since it is responsible only for rhymeness and text structure. In particular, for each verse we kept at most the last three syllables; this is to preserve the part of the words needed to learn rhyming. In fact, the majority of words in Italian language is paroxytone or “piana”, that means that the toned vowel which defines the word’s pronounce is in the second last syllable. Saving and using the last three syllables per verse reasonably means keeping either the rhymes (third last and second last saved syllables, if present) and ‘*<end_of_verso>*’ syllable or, for special tokens verses, the only syllable, one among [*‘<start_of_canto>’, ‘<end_of_canto>’, ‘<start_of_terzina>’, ‘<end_of_terzina>’*].

In addition, it is important to highlight that, by being responsible for learning the structure, this model is also supposed to be taught the general length of a *canto*, because it is the only model going through the '*<end_of_canto>*' special token.
In order to enhance the probability of the model to fairly learn to predict the end of a new *canto*, we tried increasing the input sequence's length to include at least two special tokens '*<start_of_canto>*', hence *SEQ_LENGTH* was set at 580 (tokens).

During the model activities, each sample of the dataset consists of input sequence and output sequence: the former is provided to the model, the latter is a one-token shift of the sequence, so that the model learns which token would follow the input sequence.

INPUT SEQUENCE	**OUTPUT SEQUENCE**
<start_of_canto> <start_of_terzina> vi ta <end_of_verso> scu ra <end_of_verso> ri ta <end_of_verso> <end_of_terzina> <start_of_terzina> du ra <end_of_verso> for te <end_of_verso> u	<start_of_terzina> vi ta <end_of_verso> scu ra <end_of_verso> ri ta <end_of_verso> <end_of_terzina> <start_of_terzina> du ra <end_of_verso> for te <end_of_verso> u **ra**

Of course these syllables had to be represented as integers to be digested and elaborated by a neural network, so we created three Rhyme model's python *dictionaries* from the text we had as input of this network and we saved them in *json* format in order to conveniently represent the data. Since the dictionaries are built over the input text, of course only those syllables that appear at the end of verses will be stored in these dictionaries: the unique syllable of those verses that contain only special tokens or the last three syllables of those verses that make up *terzine*. The unique syllables are 1577.

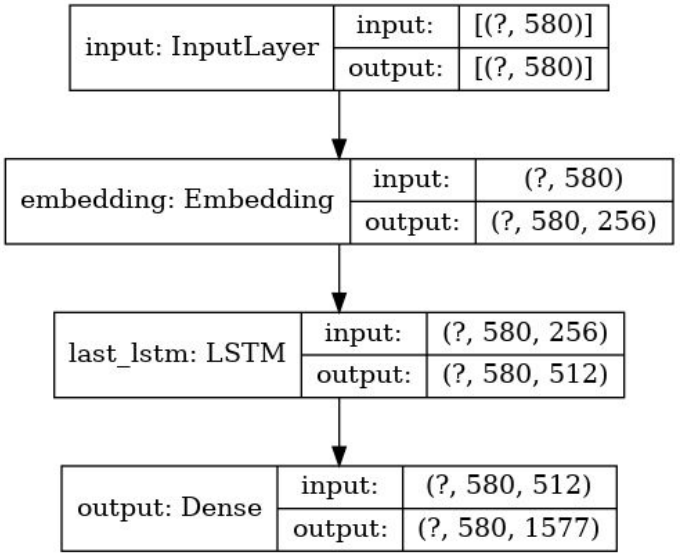
Here is a sample of this model's dictionaries:

```
{
   "vocab": [
      "'bea", "'co", "'deo'", "'din", "'e",
      "'po", "<end_of_canto>", "<end_of_terzina>",
      "<end_of_verso>", "<start_of_canto>",
      "<start_of_terzina>", "<word_sep>", "a",
      … ],
   "idx2text": {
      "0": "'bea", "1": "'co", "2": "'deo'", "3": "'din", "4": "'e",
      "5": "'po", "6": "<end_of_canto>", "7": "<end_of_terzina>",
      "8": "<end_of_verso>", "9": "<start_of_canto>",
      "10": "<start_of_terzina>", "11": "<word_sep>", "12": "a",
      … },
   "text2idx": {
      "'bea": 0, "'co": 1, "'deo'": 2, "'din": 3, "'e": 4,
      "'po": 5, "<end_of_canto>": 6, "<end_of_terzina>": 7,
      "<end_of_verso>": 8, "<start_of_canto>": 9,
      "<start_of_terzina>": 10, "<word_sep>": 11, "a":12,
      … }
}
```

3.1.1.2. Model architecture

The architecture of Rhyme model is composed of the following layer:

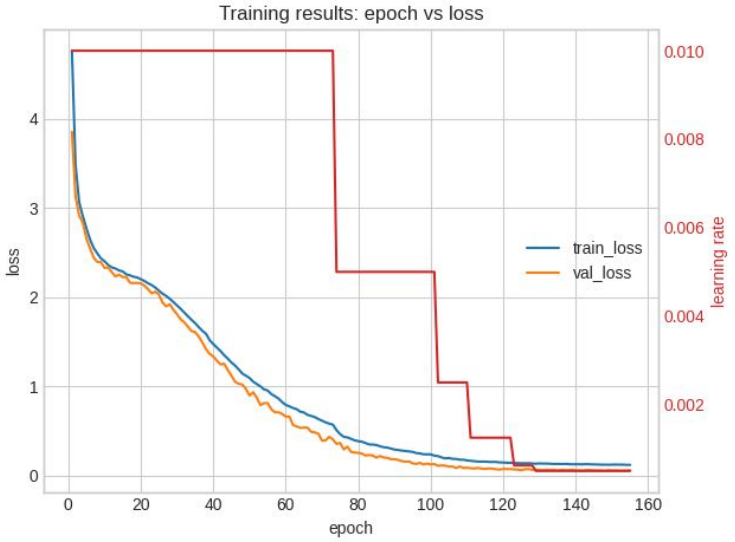
- **Input layer** kicks off the elaboration of the input sequence. In this case, input sequences have a length of 580. We decided for such a size because we wanted these sequences to include at least two *cantos* in order to pursue the text's structure learnment;
- **Embedding layer** learns a high-dimensional representation of each token of the input, to provide a better input sequence to the next layer. We set the embedding dimension to 256;
- **LSTM layer** processes sequences and it learns the succession of syllables and special tokens to understand the pattern of structure and build correct rhymes. The parameter of the layer is the number of RNN units, set to 512 as in the previous models;
- **Dense layer** is a fully-connected layer aimed to get the probability of the next syllables in the sequence. It contains how many output nodes as the number of unique syllables present in the dataset: 1577. The activation function used within this layer is *softmax*, as in the previous cases.

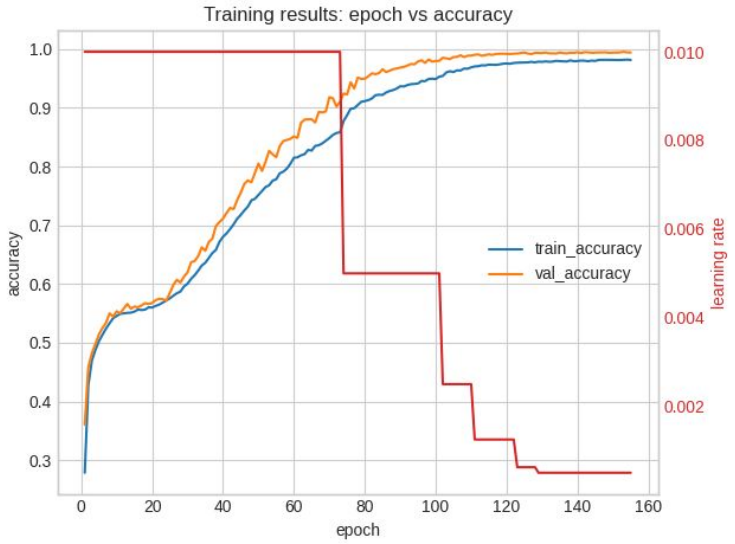


3.1.1.3. Training and graphics

We had set at most 200 epochs of execution, a batch size of 4 and the *Sparse Categorical Crossentropy* as loss function. As mentioned, we monitored the metrics over the validation set. In particular, we used *ReduceLROnPlateau* callback to refine the learning rate during the training and stopped it when the model did not improve its performance anymore through the usage of *EarlyStopping* callback.

The following plots show the History of the training epochs:





The accuracy suggests the model behaves perfectly, but we learnt in the previous models that this metric might be not so representative for our models because of such a particular task.
Having a look to the final output is still the best way to evaluate such models. Samples to understand the model's behaviour are again presented in the *Text generation* section.

3.1.2. Verse model

This model aims at completing correct hendecasyllables, on the basis of the Rhyme model's output, with no concern about generating text structure. As deeply discussed previously, the peculiarity of this model is about the order of the input's syllabic sequences during the training, hence the order of the output generated syllables, which are both in reversed line, elaborated from right to left, from the last syllable of verse to the first one.

3.1.2.1. Data processing

The Verse model is trained on a different dataset of the Rhyme model, but both of them are built from the same text of the Divine Comedy, so, as it has been done for the first model, we had to preprocess the raw text, integrating in the text only some of the special tokens we added for the Rhyme model's input. In particular, for this model, we were interested only in filling in the special tokens: [*'<end_of_verso>', '<word_sep>'*], skipping those that highlight the structure of the text, which is something that had been left up exclusively to the Rhyme mode: [*'<start_of_canto>', '<end_of_canto>', '<start_of_terzina>', '<end_of_terzina>'*].
Here is a sample of preprocessed text:

```
nel <word_sep> mezzo <word_sep> del <word_sep> cammin <word_sep> di <word_sep>
nostra <word_sep> vita <end_of_verso>
mi <word_sep> ritrovai <word_sep> per <word_sep> una <word_sep> selva <word_sep>
oscura <end_of_verso>
ché <word_sep> la <word_sep> diritta <word_sep> via <word_sep> era <word_sep>
smarrita <end_of_verso>
ahi <word_sep> quanto <word_sep> a <word_sep> dir <word_sep> qual <word_sep> era
<word_sep> è <word_sep> cosa <word_sep> dura <end_of_verso>
esta <word_sep> selva <word_sep> selvaggia <word_sep> e <word_sep> aspra
<word_sep> e <word_sep> forte <end_of_verso>
che <word_sep> nel <word_sep> pensier <word_sep> rinova <word_sep> la <word_sep>
paura <end_of_verso>
```

As mentioned, this model has to be trained with end-to-begin verses, so that it will be able to generate verse-completion with possibly existing words for each presented rhyme generated by the Rhyme model. For this reason, the next step of data processing procedure is syllables reversing.
Here is an example of what the input and output would look like after this manipulation:

INPUT SEQUENCE	**OUTPUT SEQUENCE**
<end_of_verso> ta vi <word_sep> stra no <word_sep> di <word_sep> min cam <word_sep> del <word_sep> zo mez <word_sep> nel <end_of_verso> ra scu va<word_sep>o sel <word_sep> na u <word_sep> per <word_sep> vai tro ri <word_sep>	ta vi <word_sep> stra no <word_sep> di <word_sep> min cam <word_sep> del <word_sep> zo mez <word_sep> nel <end_of_verso> ra scu va<word_sep>o sel <word_sep> na u <word_sep> per <word_sep> vai tro ri <word_sep> **mi**

Of course these syllables had to be represented as integers to be digested and elaborated by a neural network, so we created three Verse model's python *dictionaries* from the text we had as input of this network and we saved them in *json* format in order to conveniently

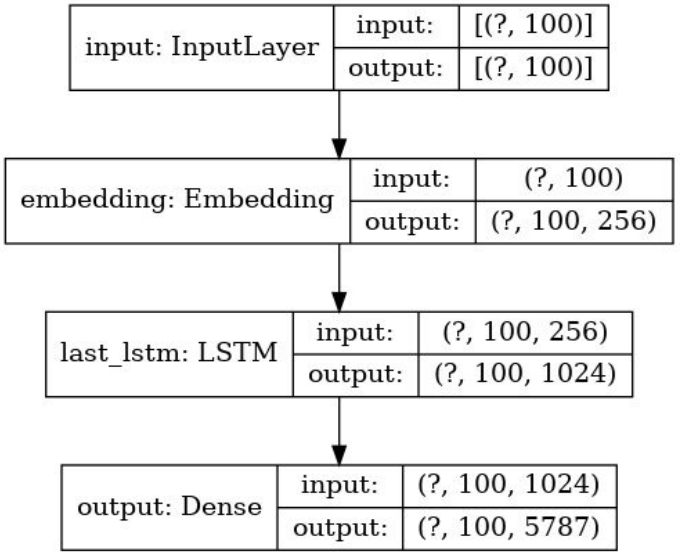
represent the data. Since the dictionaries are built over the input text, of course these dictionaries contain every syllable present in the text of the Divine Comedy, which are 5787 unique syllables.
Here is a sample of this model's dictionaries:

```
{
   "vocab": [
      '<end_of_verso>', '<word_sep>', 'a',
      "a'", "a<word_sep>'l", 'a<word_sep>a',
      'a<word_sep>ac', 'a<word_sep>al',
      'a<word_sep>am', 'a<word_sep>e',
      … ],

   "idx2text": {
      '46': '<end_of_verso>', '47': '<word_sep>', '48': 'a',
      '49': "a'", '50': "a<word_sep>'l", '51': 'a<word_sep>a',
      '52': 'a<word_sep>ac', '53': 'a<word_sep>al',
      '54': 'a<word_sep>am', '55': 'a<word_sep>e',
      … },

   "text2idx": {
      '<end_of_verso>': 46, '<word_sep>': 47, 'a': 48,
      "a'": 49, "a<word_sep>'l": 50, 'a<word_sep>a': 51,
      'a<word_sep>ac': 52, 'a<word_sep>al': 53,
      'a<word_sep>am': 54, 'a<word_sep>e': 55,
      … }
}
```

3.1.2.2. Model architecture

The architecture of Verse model is made up by the following layer:

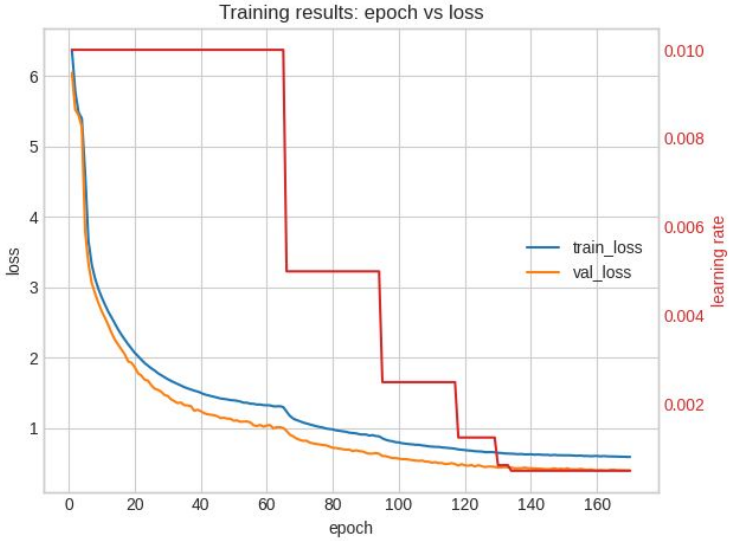
- **Input layer** kicks off the elaboration of the input sequence. In this case, input sequences have a length of 100. We chose a size big enough to include some entire verses, in order to learn verses length, but not as wide as Rhyme model's one because this model is not responsible for learning the text structure;
- **Embedding layer** learns a high-dimensional representation of each token of the input, to provide a better input sequence to the next layer. We set the embedding dimension to 256;
- **LSTM layer** processes sequences and it learns the succession of syllables to build correct words. RNN units are set to 1024 as the previous cases;
- **Dense layer** is a fully-connected layer aimed to get the probability of the next syllables in the sequence. It contains how many output nodes as the number of unique syllables present in the dataset, which, in this case, was 5787. The activation function used within this layer is *softmax*, as in the previous cases.

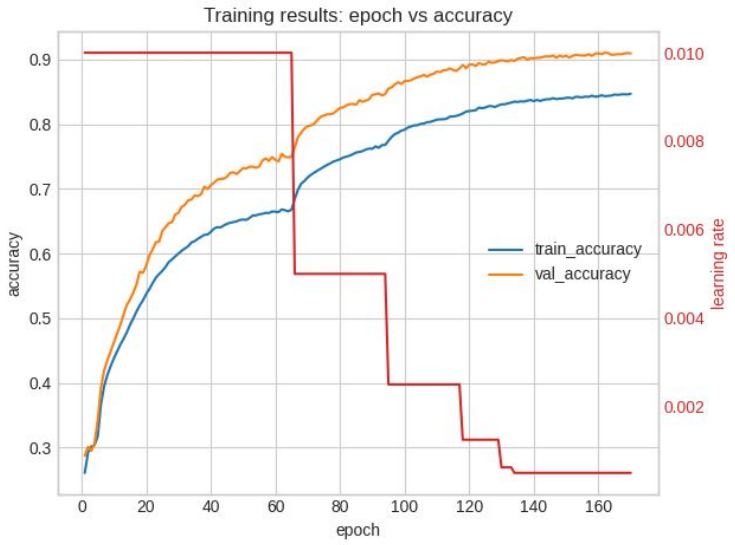


3.1.2.3. Training and graphics

We had set at most 200 epochs of execution, a batch size of 32 and the *Sparse Categorical Crossentropy* as loss function. As mentioned, we monitored the metrics over the validation set. In particular, we used *ReduceLROnPlateau* callback to refine the learning rate during the training and stopped it when the model did not improve its performance anymore through the usage of *EarlyStopping* callback.

The following plots show the History of the training epochs:



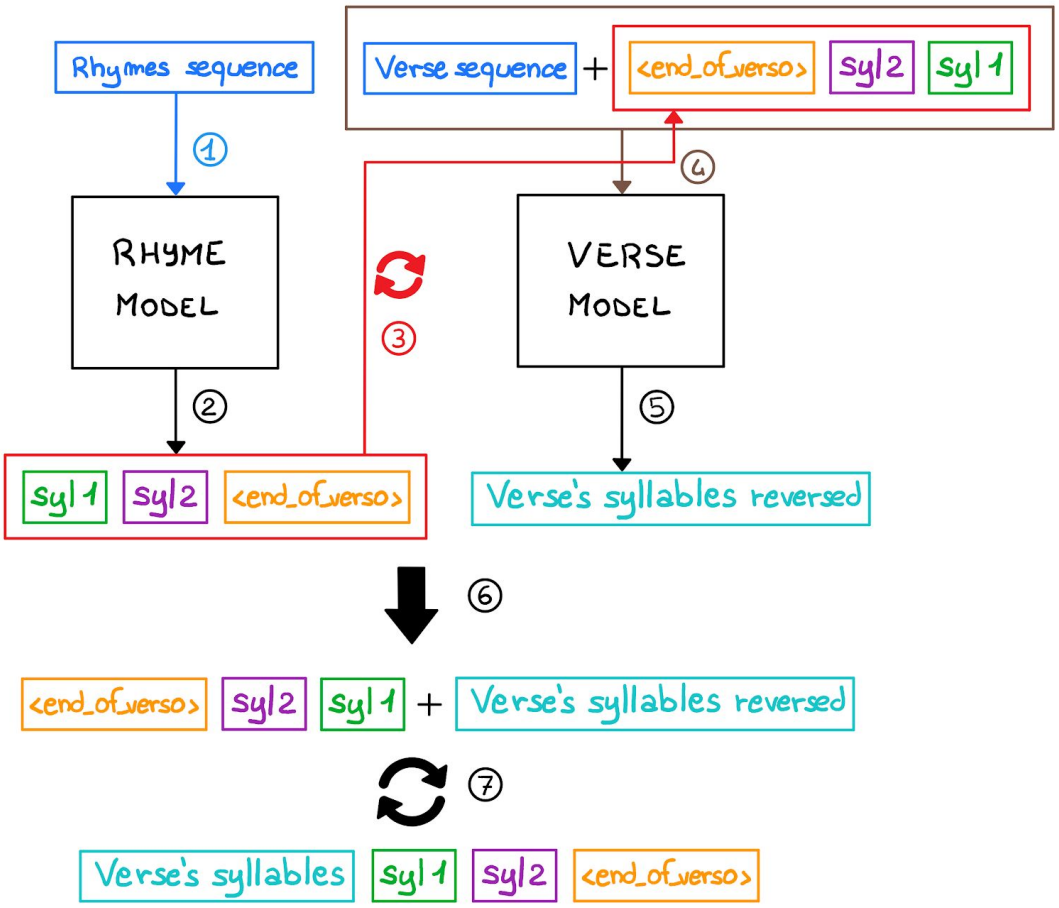


The training stopped after 165 epochs. At that point, the model had reached an accuracy of 90% on validation set in predicting the next syllables in a sequence.
The accuracy suggests the model behaves perfectly, but we learnt in the previous models that this metric might be not so representative for our models because of such a particular task.
Having a look to the final output is still the best way to evaluate such models. Samples to understand the model's behaviour are again presented in the *Text generation* section.

3.1.3. Text generation

Eventually, we got to the generating phase of the process. In order to start the generation of new text, we needed a starting sequence to use as input for each of the two models. For the Rhyme model, the starting sequence must be a sequence of rhyme syllables only, while, for the Verse model, the input consists of verses' syllables. In particular, the starting sequence of the Rhyme model is a sequence of last syllables of verses starting from the final verse of a random *canto* and going backwards for a length of Rhyme model's *SEQ_LENGTH*. After the generation of the first next rhyme, which represents a sort of partial output of the Rhyme model, the syllables of this new rhyme is concatenated to the input syllables of the Verse model. The first input of the Verse model is a sequence of verse syllables starting from the final verse of another random *canto* back to have a length of Verse model's *SEQ_LENGTH*, considering also the fresh new rhyme's syllables. Of course, the Verse model's input is reversed in order. The generation of this partial output of the Verse model stops when the special token '*<end_of_verso>*' shows up. Once the Verse model has finished to produce this partial output, this loop repeats. From the second generation to the end of the complessive generation, which happens when the Rhyme model gives as output the special token '*<end_of_canto>*', the Rhyme model keeps generating the next rhyme syllables and these, reversed, are attached to last output of the Verse model.

Here is a graphic representation of what just explained:



In particular, for what concerns the Rhyme model, we decided to make a changement in the selection of the next syllable generation with respect to previous models. In fact, we thought that the output should have been the one specified by a selection through *argmax*, because one and only one is the actual correct rhyme. For this reason, we believed that, in this case, applying *numpy.random.choice* would not be as well-behaving as in the previous experiments. Instead, in the case of the Verse model's selection, we opted for keeping the *numpy.random.choice* in order to avoid repetitions.

Here is an example of the partial output of the Rhyme model:

```
<start_of_canto>
<start_of_terzina>
sacro <end_of_verso>
punta <end_of_verso>
to<word_sep>acro <end_of_verso>
<end_of_terzina>
<start_of_terzina>
cunta <end_of_verso>
cusa <end_of_verso>
giunta <end_of_verso>
<end_of_terzina>
<start_of_terzina>
```

```
fusa <end_of_verso>
spense <end_of_verso>
schiusa <end_of_verso>
<end_of_terzina>
[...]
```

We understood that we had an extreme improvement in rhymeness.

While this is a sample of the final output of the *Model by reversed syllables*:

```
CANTO

lo collo l'assannò sì come sacro
voce del gran cui l'ombra s'appunta
puote disnebbiar vostro intelletto acro

e come 'l sol fermar dietro a lor cunta
e nel figlio forte a sé stessi s'accusa
io vidi sol che vi parea congiunta

crescerann' ei dopo la rima confusa
sì come la propria madre spense
maggior paura già di fuor dischiusa

dicendo sofferse cose che pense
ne l'una parte l'anime triste
per ch'io dissi quell' anime offense

suffolando e scegliendo fioco miste
alor sicuramente apri' la bocca
dolo suo coro non parver si viste

rivolsi li occhi a quel ch'or si scocca
al punto te che passi vide l'arco
l'un luogo mio che già mai non si tocca

tu troverai non sermone e per lo
'ncarco
la naturalmente per che s'aspiri
e al suo temo che mi trasse al varco

[...]
```

We concluded that we achieved quite correctly the rhymeness goal, but we still had some margin of improvement in hendecasyllabicness.

3.2. Model by toned and reversed syllables

Since the previous model got good results in terms of rhymeness, we now focused on the hendecasyllables. In order to do this, we tried to improve the training data highlighting the stressed vowels into the text to achieve better results in terms of hendecasyllabicness. Indeed, we realized that the *Pyphen* module of python used before was not suitable to syllabify a poetry text, so for this last *Model by toned and reversed syllables* we developed our syllabification module to better syllabify each verse, trying to apply metric figures by need as Dante has done writing the Divine Comedy in order to make the correct split of the hendecasyllables. Indeed, as explained in the *Linguistic Rules* section, syllabification and hendecasyllabicness especially are based over stressed vowels, for this reason we based our split on the fact that the stressed syllables of the last word must be at the tenth syllable of the verse, therefore we needed a way to highlight toned vowels in each words. We forced this behavior when splitting the Divine Comedy to build the training sequences of syllables for the networks, but we completely let the model learn how to compose a correct hendecasyllable: our idea was that the more correct are the training data in terms of syllabification, the easier will be for the model to learn the patterns.
We developed our syllabification rules on the basis of a quick analysis of the Divine Comedy's text, which pointed out some errors when the syllabification was performed by the *Pyphen* module due to some specific metrics figure of the poetry.
Moreover, our syllabification module is suitable also to check and verify, in all the generated *cantos*, if the model produced correct hendecasyllables and to compare all the models in the next *Results* section.

In order to tone all the Divine Comedy, we built the Tone model, which learns how to tone a word. Because the Italian language has not fixed rules to decide the position of the tonic accent within a word, our thoughts went to neural networks and we hoped that, after showing a lot of Italian toned words as examples, the networks would learn some patterns from the sequence of characters which compose the word.
More in detail, the *Model by toned and reversed syllables* relies on three sub-models:

- Tone model
- Rhyme model
- Verse model

The Rhyme model and the Verse model are those developed for text generation and thanks to this separation, we can better face up all the requirements, allowing the Rhyme model to learn the rhyming pattern and the *terzine* structure while the Verse model is focused only on hendecasyllables. We also wanted to allow the generating model to stop by itself when it writes a special token and working in this way, the model composes longer *cantos* than the baseline ones.

3.2.1. Tone model

This model aims at predicting the position of the accent in a word: given the untoned word as input it provides us what is the stressed letter. In this way we are able to transform each word into the correspondent toned version.

3.2.1.1. Data processing

The Tone model was not used for text generation, but only in the preprocessing phase to build the dataset for generation models. So, in order to train this model we used a different dataset and not the Divine Comedy text. We downloaded a batch of approximately 65000 Italian words from *Wikidizionario*[12], all of them with the accent on the correct letter. Since we wanted the accent's position as output of the network when the untoned word is provided to the network, each sample is decomposed as in the following table, where the 'word' is the input and the 'index' is the output:

word	**index**
abate	3
abbandonato	9
abbandono	7
abbarbaglio	7
abbondante	7
abete	3
...	...

At the end, we split the dataset keeping 80% for the training set and 20% for the evaluation set, to measure how good the model predicts the accent of a word.

Of course these words had to be represented as integers to be digested and elaborated by a neural network, and since also this Tone model requires a sequence as input, we considered each word as a sequence of characters and we created three Tone model's python *dictionaries* from the word we had in the dataset and we saved them in *json* format in order to conveniently represent the data. Since the dictionaries are built over all the words, of course these dictionaries contain every character present in them, which are 34 unique characters.

[12] https://it.wiktionary.org/

Here is a sample of this model's dictionaries:

```
{
   "vocab": [
      '#', "'", 'a', 'b', 'c', 'd',
      'e', 'f', 'g', 'h', 'i', 'j',
      'k', 'l', 'm', 'n', 'o',
      … ],

   "idx2text": {
      '0': '#', '1': "'", '2': 'a', '3': 'b', '4': 'c', '5': 'd',
      '6': 'e', '7': 'f', '8': 'g', '9': 'h', '10': 'i', '11': 'j',
      '12': 'k', '13': 'l', '14': 'm', '15': 'n', '16': 'o',
      ...
   },

   "text2idx": {
      '#': 0, "'": 1, 'a': 2, 'b': 3, 'c': 4, 'd': 5,
      'e': 6, 'f': 7, 'g': 8, 'h': 9, 'i': 10, 'j': 11,
      'k': 12, 'l': 13, 'm': 14, 'n': 15, 'o': 16,
      ...
    }
}
```

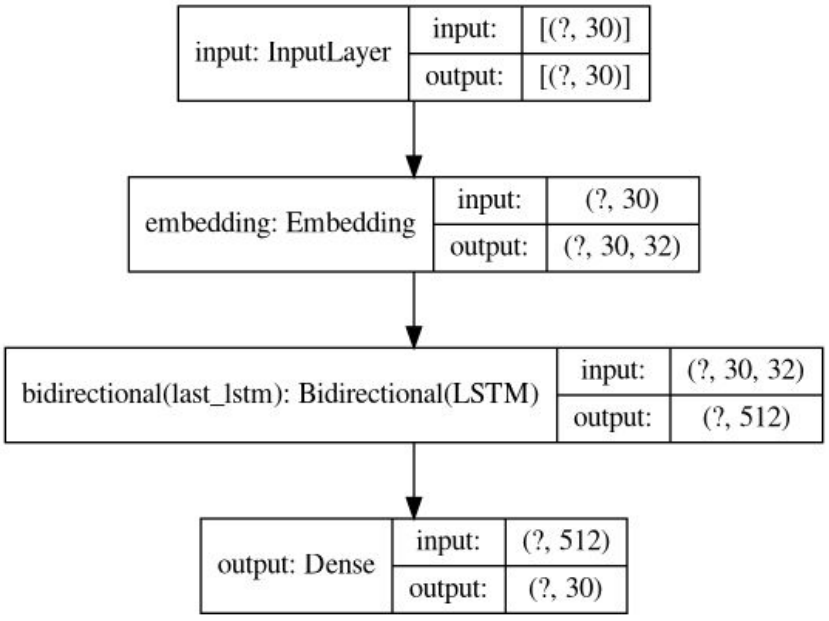
3.2.1.2. Model architecture

Since this problem is a kind of text classification task, we needed to learn an embedding for a word and then predicted the position of the accent as the output.
Analyzing the dataset we decided to fix the input sequence length to 30 characters as the first parameter of the network, so we needed to pad input up to the *MAX_WORD_LENGTH* parameter. Moreover, we kept the embedding dimension quite low and fixed the parameter *EMBEDDING_DIM* to 32 because our inputs are the character of the alphabet with some other symbols, provided to the network as a sequence. For the LSTM layer we set 256 *RNN_UNITS*, which becomes 512 in the layer output because of the Bidirectional.

Keeping in mind that the input of the Tone model is a sequence of characters, which compose the word, the model is built with the following layers:

- **Input**: this layer kicks off the elaboration of the input sequence. In this case, input sequences have a length of 30, as previously explained;
- **Embedding**: this layer learns a high-dimensional representation of each single character, to provide a better input sequence to the next layer;
- **Bidirectional LSTM**: a layer which processes sequences is used to compute a final embedding of the whole word, allowing the network to see the order of letters in the word. Because of the bidirectionality, the sequence is seen also in the inverted order to improve the final classification;
- **Dense**: the last layer is a fully-connected layer aimed to learn the class (which is the position of the accent in that word) from the word embedding and because we have thought this task as a multi-label classification problem we use *softmax* as final activation function.

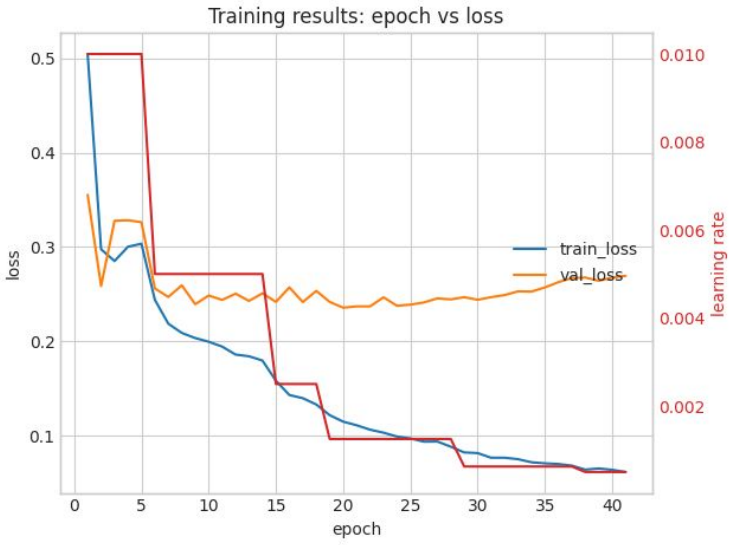
The following Figure shows the summary of the model architecture, with input and output shape depending on the parameters described above.

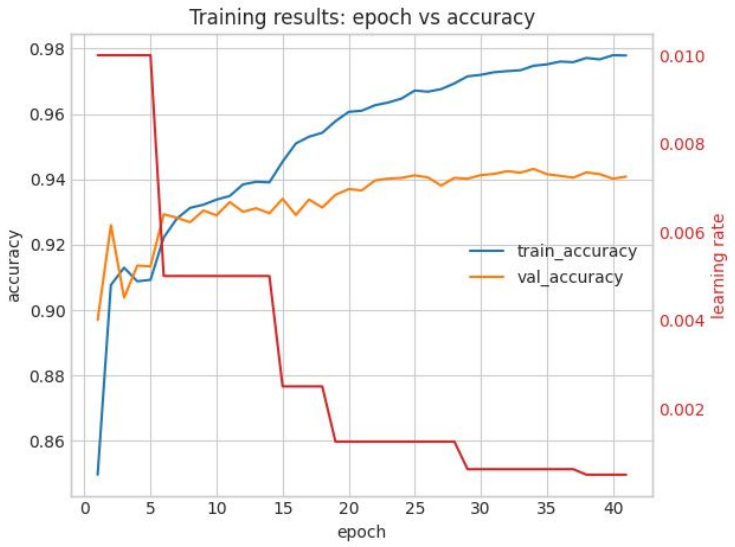


3.2.1.3. Training and graphics

We had set at most 200 epochs of execution, a batch size of 32 and the *Categorical Crossentropy* as loss function. As mentioned, we monitored the metrics over the validation set. In particular, we used *ReduceLROnPlateau* callback to refine the learning rate during the training and stopped it when the model did not improve its performance anymore through the usage of *EarlyStopping* callback.

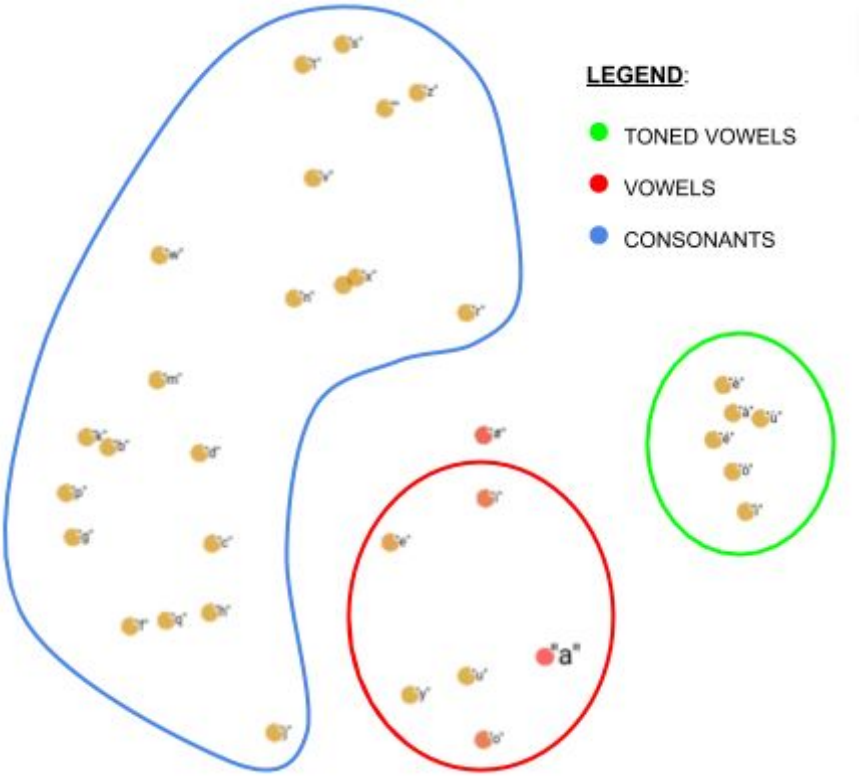
We plotted the progress of training, showing up the loss and accuracy both on the training set and the validation set along with the decreasing learning rate.





The training stopped after 41 epochs and reached an accuracy of nearly 98% on the training set and around 94% on the validation set, which is a good result.

Since in the Italian language the accent falls always on a vowel and the Tone model sees a word character by character, it should learn a clear separation between vowels and consonants. Indeed, plotting the embeddings of each input character in a 2-dimensional space after applying a PCA for dimension reduction, the previous hypothesis is confirmed and some clusters are visible. In particular, the toned vowels (in green), present as the last letter of some Italian words, are very close to each other and far from the others in the plane. A second group is composed of the normal vowels (in red) plus 'y', which can be considered a vowel in the English alphabet, even if is not present in the standard Italian one. All the remaining letters are consonants and are circled in blue.



3.2.2. Rhyme model

The Rhyme model is dedicated to learning the rhymes pattern and the text structure of the Divine Comedy, which consists of split the hendecasyllables in *terzine* and *canto*'s length.
It will generate only the last syllables of each verse and the structure for separation of *terzine*.

3.2.2.1. Data processing

As far as we have manipulated the raw text of the Divine Comedy for all the other models, we needed here the same cleaning and preprocessing steps before preparing the text for building the dataset for training.
First of all, we applied some special tokens to mark the structure of the text and set clearly where each part of text starts and finishes. These are the special tokens which are added to the text:

SPECIAL TOKENS	
'START_OF_CANTO'	*'<start_of_canto>'*
'END_OF_CANTO'	*'<end_of_canto>'*
'START_OF_TERZINA'	*'<start_of_terzina>'*
'END_OF_TERZINA'	*'<end_of_terzina>'*
'END_OF_VERSO'	*'<end_of_verso>'*
'WORD_SEP'	*'<word_sep>'*

As the last step, we remember that in this last model we always work with the toned words, so we use the Tone model described before on each word in order to obtain the Divine Comedy with stressed words. Our preprocessed text appears as below:

```
<start_of_canto>
<start_of_terzina>
nel <word_sep> mèzzo <word_sep> del <word_sep> cammìn <word_sep> di <word_sep>
nòstra <word_sep> vìta <end_of_verso>
mi <word_sep> ritrovài <word_sep> per <word_sep> ùna <word_sep> sèlva <word_sep>
oscùra <end_of_verso>
ché <word_sep> la <word_sep> dirìtta <word_sep> vìa <word_sep> èra <word_sep>
smarrìta <end_of_verso>
<end_of_terzina>
<start_of_terzina>
àhi <word_sep> quànto <word_sep> a <word_sep> dir <word_sep> quàl <word_sep> èra
<word_sep> è <word_sep> còsa <word_sep> dùra <end_of_verso>
èsta <word_sep> sèlva <word_sep> selvàggia <word_sep> e <word_sep> àspra
<word_sep> e <word_sep> fòrte <end_of_verso>
che <word_sep> nel <word_sep> pensièr <word_sep> rinòva <word_sep> la <word_sep>
paùra <end_of_verso>
<end_of_terzina>
```

The Rhyme model is a sequence-to-sequence model, so we wanted to build a dataset of sequences starting from the toned and marked text. Our element of the sequence will be a single syllable, so we split each verse in syllables according to Italian grammar rules. Then, since for this model we are interested only in rhymes and text structure, we kept the last syllables of each verse from the last toned syllables to the end of the verse, discarding all the previous syllables. Indeed, the accents on words also helped us to extract the syllables which compose rhymes in a more correct way following the real definition of a rhyme: we were able to deal with the different kinds of words present in Italian language, such as words which are *tronche, sdrucciole, bisdrucciole* instead of considering all words as *piane*, like in the *Model by reversed syllables*. We wanted to point out that this is only a better way to consider the concept of rhyme, which came for free, in this model, since we had toned the Divine Comedy, but the *Model by reversed syllables* had already reached good results in terms of rhymeness.

We would also like this model to be taught the general length of a *canto*. In order to do this, we tried extending the input sequence's length up to catch at least two special tokens '*<start_of_canto>'* by fixing the *SEQ_LENGTH* equals to 580 (tokens).

Each sample of the dataset is composed of an input sequence and an output sequence. The former is provided as input to the model while the latter is shifted of one token in order to teach to the model which is the token that follows the input one.

INPUT SEQUENCE	**OUTPUT SEQUENCE**
<start_of_canto> <start_of_terzina> vì ta <end_of_verso> scù ra <end_of_verso> rì ta <end_of_verso> <end_of_terzina> <start_of_terzina> dù ra <end_of_verso> fòr te <end_of_verso> ù	<start_of_terzina> vì ta <end_of_verso> scù ra <end_of_verso> rì ta <end_of_verso> <end_of_terzina> <start_of_terzina> dù ra <end_of_verso> fòr te <end_of_verso> ù **ra**

Of course these syllables had to be represented as integers to be digested and elaborated by a neural network, so we created three Rhyme model's python *dictionaries* from the text we had as input of this network and we saved them in *json* format in order to conveniently represent the data. Since the dictionaries are built over the input text, of course only those syllables that appear at the end of verses will be stored in these dictionaries: the unique syllable of those verses that contain only special tokens or the last three syllables of those verses that make up *terzine.* The unique syllables are 1482.
Here is a sample of this model's dictionaries:

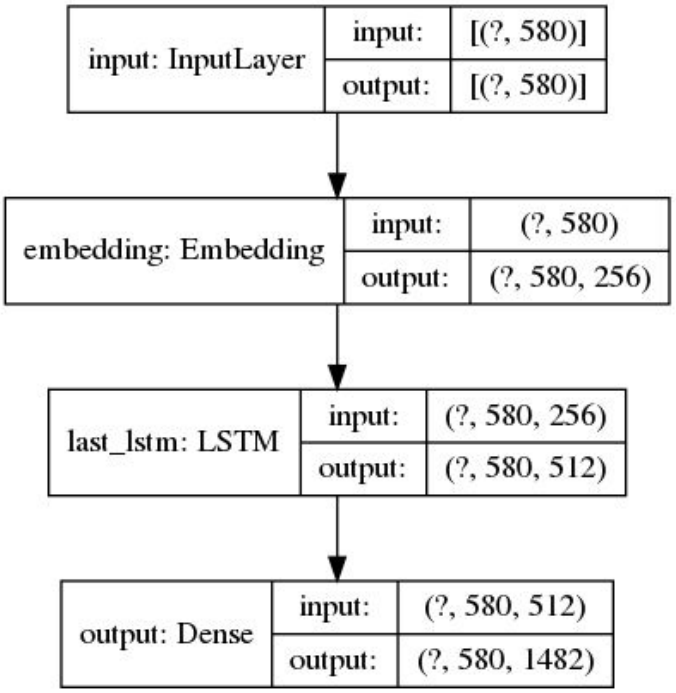
```
{
   "vocab": [
      '<end_of_canto>', '<end_of_terzina>',
      '<end_of_verso>', '<start_of_canto>',
      '<start_of_terzina>', '<word_sep>', 'a',
      "a'", "a<word_sep>'fòr", 'a<word_sep>òm',
      … ],
```

```
   "idx2text": {
      '9': '<end_of_canto>', '10': '<end_of_terzina>',
      '11': '<end_of_verso>', '12': '<start_of_canto>',
      '13': '<start_of_terzina>', '14': '<word_sep>', '15': 'a',
      '16': "a'", '17': "a<word_sep>'fòr", '18': 'a<word_sep>òm',
      … },

   "text2idx": {
      '<end_of_canto>': 9, '<end_of_terzina>': 10,
      '<end_of_verso>': 11, '<start_of_canto>': 12,
      '<start_of_terzina>': 13, '<word_sep>': 14, 'a': 15,
      "a'": 16, "a<word_sep>'fòr": 17, 'a<word_sep>òm': 18,
      … }
}
```

3.2.2.2. Model architecture

The architecture of Rhyme model is composed of the following layer:

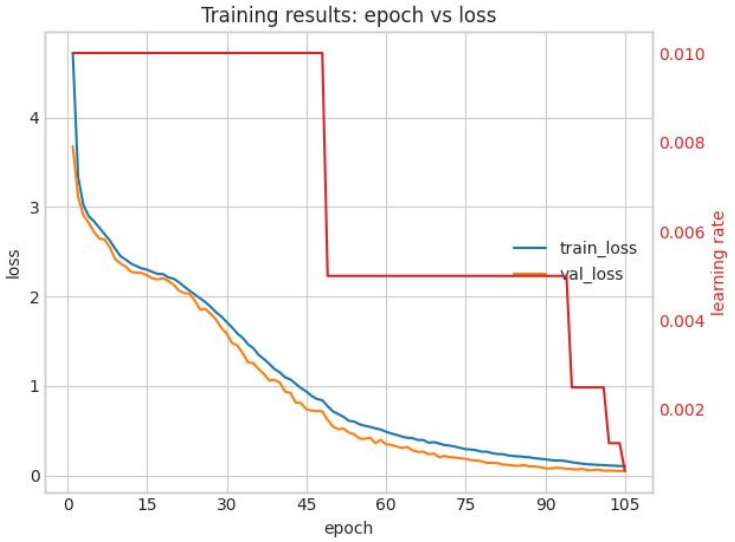
- **Input**: this layer kicks off the elaboration of the input sequence. In this case, input sequences have a length of 580. We decided for such a size because we wanted these sequences to include at least two *cantos* in order to pursue the text's structure learnment;
- **Embedding**: this layer learns a high-dimensional representation of each token of the input, to provide a better input sequence to the next layer. We set the embedding dimension to 256;
- **LSTM**: this layer processes sequences and it learns the succession of syllables and special tokens to understand the pattern of structure and build correct rhymes. The parameter of the layer is the number of RNN units, set to 512;
- **Dense**: the last layer is a fully-connected layer aimed to get the probability of the next syllables in the sequence. It contains how many nodes as the number of unique syllables present in the dataset, which, in this case, was 1482. The activation function used within this layer is *softmax*, as in the previous cases.



3.2.2.3. Training and graphics

We had set at most 200 epochs of execution, a batch size of 4 and the *Sparse Categorical Crossentropy* as loss function. As mentioned, we monitored the metrics over the validation set. In particular, we used *ReduceLROnPlateau* callback to refine the learning rate during the training and stopped it when the model did not improve its performance anymore through the usage of *EarlyStopping* callback.

As usual, we monitored the metrics on a validation set and stopped the training when the model started to perform bad. The following plots show the History of the training epochs:



The accuracy suggests the model behaves perfectly, but we learnt in the previous models that this metric might be not so representative for our models because of such a particular task.
Having a look to the final output is still the best way to evaluate such models. Samples to understand the model's behaviour are again presented in the *Text generation* section.

3.2.3. Verse model

The single aim of the Verse model is to compose correct hendecasyllables without caring about any kind of structure. The peculiarity of the model concerns the order of syllables in the input sequence used to train it, so each verse is written in the inverse order, starting from the end of verse to the beginning, as explained for *Model by reversed syllables*.

3.2.3.1. Data processing

The Verse model was trained on a different dataset of the Rhyme model, but both of them were built from the same text of the Divine Comedy, so we needed to preprocess the raw text as before, replacing words with their toned version and applying the following special tokens:

SPECIAL TOKENS	
'START_OF_CANTO'	*'<start_of_canto>'*
'END_OF_CANTO'	*'<end_of_canto>'*
'START_OF_TERZINA'	*'<start_of_terzina>'*
'END_OF_TERZINA'	*'<end_of_terzina>'*
'END_OF_VERSO'	*'<end_of_verso>'*
'WORD_SEP'	*'<word_sep>'*

Below an example of our preprocessed text:

```
<start_of_canto>
<start_of_terzina>
nel <word_sep> mèzzo <word_sep> del <word_sep> cammìn <word_sep> di <word_sep>
nòstra <word_sep> vìta <end_of_verso>
mi <word_sep> ritrovài <word_sep> per <word_sep> ùna <word_sep> sèlva <word_sep>
oscùra <end_of_verso>
ché <word_sep> la <word_sep> dirìtta <word_sep> vìa <word_sep> èra <word_sep>
smarrìta <end_of_verso>
<end_of_terzina>
<start_of_terzina>
àhi <word_sep> quànto <word_sep> a <word_sep> dir <word_sep> quàl <word_sep> èra
<word_sep> è <word_sep> còsa <word_sep> dùra <end_of_verso>
èsta <word_sep> sèlva <word_sep> selvàggia <word_sep> e <word_sep> àspra
<word_sep> e <word_sep> fòrte <end_of_verso>
che <word_sep> nel <word_sep> pensièr <word_sep> rinòva <word_sep> la <word_sep>
paùra <end_of_verso>
<end_of_terzina>
```

As already mentioned, the Verse model should only learn how to write verses, so we prepared the dataset removing all special tokens which concern the text's structure such as '*<start_of_canto>', '<end_of_canto>', '<start_of_terzina>', '<end_of_terzina>'*, therefore keeping only *'<end_of_verso>'* and *'<word_sep>'*.

As we already explored in the previous *Model by reversed syllables* section, we split verses in syllables, we built up two different data inputs (one with the final syllables exclusively and one with the "reversed" verses). The Verse model, trained over reversed verses in order to preserve meaning in the generated words, works jointly and based on the Rhyme model's progress, in order to achieve rhyming, as previously explained.

So, the dataset contains sequences where verses are in reversed order, like in the following example:

INPUT SEQUENCE	**OUTPUT SEQUENCE**
<end_of_verso> ta vì <word_sep> stra nò <word_sep> di <word_sep> mìn cam <word_sep> del <word_sep> zo mèz <word_sep> nel <end_of_verso> ra scù va<word_sep>o sèl <word_sep> na ù <word_sep> per <word_sep> vài tro ri <word_sep>	ta vì <word_sep> stra nò <word_sep> di <word_sep> mìn cam <word_sep> del <word_sep> zo mèz <word_sep> nel <end_of_verso> ra scù va<word_sep>o sèl <word_sep> na ù <word_sep> per <word_sep> vài tro ri <word_sep> **mi**

Of course these syllables had to be represented as integers to be digested and elaborated by a neural network, so we created three Verse model's python *dictionaries* from the text we had as input of this network and we saved them in *json* format in order to conveniently represent the data. Since the dictionaries are built over the input text, of course these dictionaries contain every syllable present in the text of the Divine Comedy, which are 5859 unique syllables.

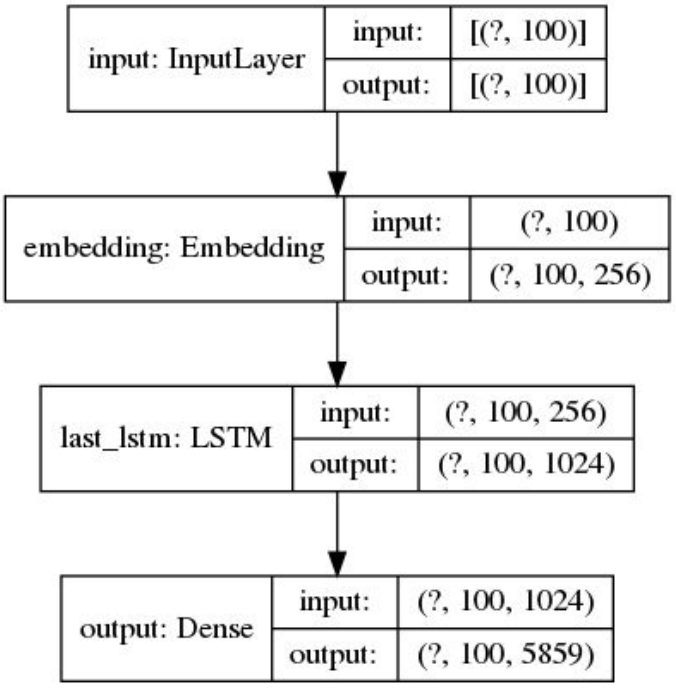
Here is a sample of this model's dictionaries:

```
{
   "vocab": [
      '<end_of_verso>', '<word_sep>', 'a',
      "a'", "a<word_sep>'a", "a<word_sep>'che",
      "a<word_sep>'dìl", "a<word_sep>'fòr",
      "a<word_sep>'l", "a<word_sep>'m",
      … ],

   "idx2text": {
      '66': '<end_of_verso>', '67': '<word_sep>', '68': 'a',
      '69': "a'", '70': "a<word_sep>'a", '71': "a<word_sep>'che",
      '72': "a<word_sep>'dìl", '73': "a<word_sep>'fòr",
      '74': "a<word_sep>'l", '75': "a<word_sep>'m",
      … },

   "text2idx": {
      '<end_of_verso>': 66, '<word_sep>': 67, 'a': 68,
      "a'": 69, "a<word_sep>'a": 70, "a<word_sep>'che": 71,
      "a<word_sep>'dìl": 72, "a<word_sep>'fòr": 73,
      "a<word_sep>'l": 74, "a<word_sep>'m": 75,
      … }
}
```

3.2.3.2. Model architecture

The architecture of Verse model is composed of the following layer:

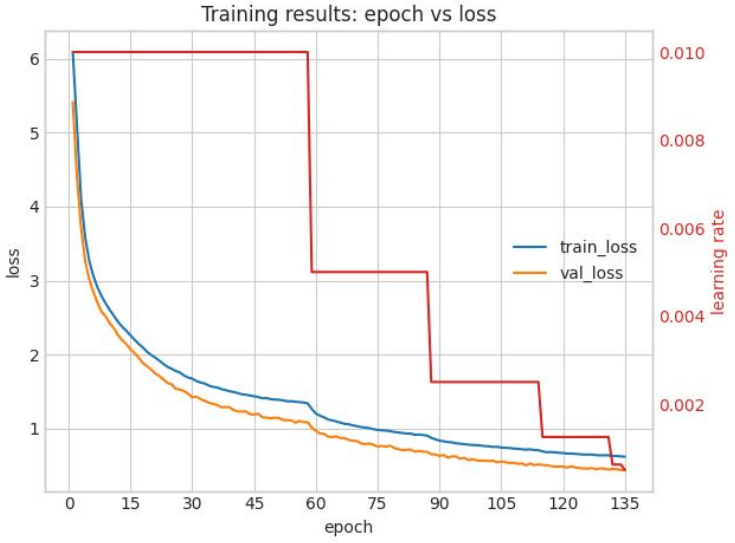
- **Input**: this layer kicks off the elaboration of the input sequence. In this case, input sequences have a length of 100. We chose a size big enough to include some entire verses, in order to learn verses length, but not as wide as Rhyme model's one because this model is not responsible for learning the text structure;
- **Embedding**: this layer learns a high-dimensional representation of each token of the input, to provide a better input sequence to the next layer. We set the embedding dimension to 256;
- **LSTM**: this layer processes sequences and it learns the succession of syllables to build correct words. We fix the sequence's length to 100 and the RNN units is set to 1024;
- **Dense**: the last layer is a fully-connected layer aimed to get the probability of the next syllables in the sequence. It contains how many nodes as the number of unique syllables present in the dataset, which, in this case, was 5859. The activation function used within this layer is *softmax*, as in the previous cases.

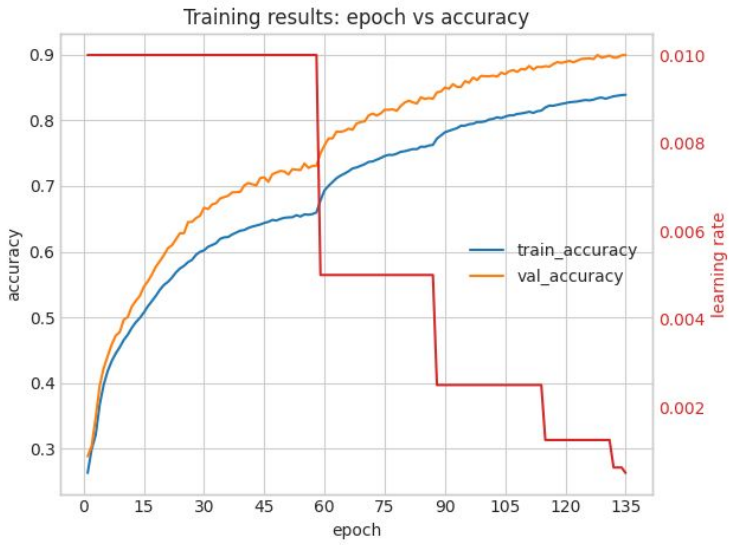


3.2.3.3. Training and graphics

We had set at most 200 epochs of execution, a batch size of 32 and the *Sparse Categorical Crossentropy* as loss function. As mentioned, we monitored the metrics over the validation set. In particular, we used *ReduceLROnPlateau* callback to refine the learning rate during the training and stopped it when the model did not improve its performance anymore through the usage of *EarlyStopping* callback.

As usual, we monitored the metrics on a validation set and stopped the training when the model started to perform badly. The following plots show the History of the training epochs:





The training stopped after 135 epochs when the model reached an accuracy of 90% on validation set in predicting the next syllables in a sequence.

Once again, the accuracy suggests the model behaves perfectly, but we learnt in the previous models that this metric might be not so representative for our models because of such a particular task.
Having a look to the final output is still the best way to evaluate such models. Samples to understand the model's behaviour are again presented in the *Text generation* section.

3.2.4. Text generation

After training all the three sub-model, we got to the generating phase of the process. In order to start the generation of new text we need a starting sequence to pass as input for each of the two models. Since the process of generation is the same as the *Model by reversed syllables*, for the Rhyme model, the starting sequence must be a sequence of rhyme syllables only, while, for the Verse model, the input consists of verses' syllables. In particular, the starting sequence of the Rhyme model is a sequence of last syllables of verses starting from the final verse of a random *canto* and going backwards for a length of Rhyme model's *SEQ_LENGTH*. Then, the generation's phase went on as already described for the Model by reversed syllables, where after got the first next rhyme, which represents a sort of partial output of the Rhyme model, the syllables of this new rhyme is concatenated to the input syllables of the Verse model. The first input of the Verse model is a sequence of verse syllables starting from the final verse of another random *canto* back to have a length of Verse model's *SEQ_LENGTH*, considering also the fresh new rhyme's syllables. Of course, the Verse model's input is reversed in order. The generation of this partial output of the Verse model stops when the special token '<*end_of_verso*>' shows up. Once the Verse model has finished to produce this partial output, this loop repeats. From the second generation to the end of the complessive generation, which happens when the Rhyme model gives as output the special token '<*end_of_canto*>', the Rhyme model keeps generating the next rhyme syllables and these, reversed, are attached to last output of the Verse model.
As discovered in the baseline models, since we had two possible approaches to select the next syllables generation, which are through *argmax* or applying *numpy.random.choice*, we adopted the former solution to the Rhyme model because one and only one is the actual

correct rhyme. Instead, in the case of the Verse model's selection, we opted for keeping the *numpy.random.choice* in order to avoid repetitions.

Here is an example of the partial output of the Rhyme model:

```
<start_of_terzina>
mòrte<end_of_verso
vàrco<end_of_verso
scòrte<end_of_verso
<end_of_terzina>
<start_of_terzina>
màrco<end_of_verso
mài<end_of_verso
l'àrco<end_of_verso
<end_of_terzina>
<start_of_terzina>
vài<end_of_verso
prègo<end_of_verso
rài<end_of_verso
<end_of_terzina>
[...]
```

We wanted to point out that here, differently from the *Model by reversed syllables* and thanks to the tones, we could stop considering always two rhyme syllables plus the special tokens *'<end_of_verso>'*, assumption that we made in the previous model, since most of the Italian words are *piane*. Indeed, looking at the example above, this Rhyme model was also able to generate rhymes composed only by a single syllable, which can be considered as a little improvement in the way we captured the concept of rhyme, as already explained in the previous *Data processing* section of the Rhyme model.

With this advanced model, we also aimed to improve the hendecasyllabicness of the text.
Since we had toned the whole Divine Comedy, preparing the dataset for training the Verse model, we tried to apply the correct definition of the hendecasyllable, explained in the *Linguistic rules* section: the last toned syllable has to be the tenth syllable in each verse. Theoretically, the tones should help in this direction and the Verse model may generate, always in reversed order, verses which follow the hendecasyllable's definition. In particular, looking at the following example of the partial output of the Verse model, the second verse is composed by ten syllables because the last word is *tronca*, so this model could also generate hendecasyllables with a different number of syllables, which are called *tronchi, piani, sdruccioli, etc*.

```
so fès <word_sep> po tròp <word_sep> qui <word_sep> son <word_sep> ma <word_sep>
sto tù <word_sep> me <word_sep> che <end_of_verso>

mài <word_sep> ci tà ri spi <word_sep> te à fï <word_sep> se spès <end_of_verso>

te scòr <word_sep> la vèl no <word_sep> in <word_sep> te gèn <word_sep> di vì
<word_sep> e <end_of_verso>

glio cì <word_sep> be<word_sep>e <word_sep> 'lgàm <word_sep> le <word_sep> fé
<word_sep> mi <word_sep> ti rèn ve re <end_of_verso>

[...]
```

While this is a sample of the final output of the *Model by toned and reversed* syllables:

CANTO

un altro ché non credi tu paventi
licito poeta de la tua fame
dirò de s'appiattò miser li denti

qual era cacciator per tutte brame
raggio diè dito e poi vedea temendo
ingiusto fece di quelli a gran dame

che solo il mio il qual io riprendo
chercuti a la luce foglia sospinto
perché questi ciascun lor maggior
temendo

molto temprando per l'aere dipinto
viso pelato che tagne ed ello
su la fiumana ove 'l mar non distinto

contente furon d'acqua e danïello
a la voce che ciò l'accende d'ira
e io maestro disdegnoso e fello

io anime nomota e tira
corso ch'io ferna a e altra cura
io dico come sotto mi spira

onde nel novo di fresca verdura
poscia ch'io dico non d'altro cagione
io vive e sente e sé stesso misura

[...]

4. RESULTS

4.1. Metrics

Since the *Deep Comedy project* deals with the original text of the Divine Comedy, we had the necessity to introduce some specific metrics to catch in the best possible way all the features of Dante's writing style, and in order to be able to recognize if our generation of text has the Divine Comedy's fingerprint from the structure and rhyme perspective. What we did not consider in our evaluation is the grammatical structure of the sentences and the semantic meaning of the generated words and phrases.

4.1.1. Our metrics

As far as we built the Tone model to get the accent on every word, we developed our metrics with the awareness that we could tone any generated text because, in poetry and mostly in Dante's style, the accents are useful both to split hendecasyllables in the correct way and to identify rhymes.
So, now we are going to explain all metrics which will apply to all *cantos* generated by each of our model, in order to compare them and the original *canto* of the Divine Comedy:

- Number of verses: by counting the number of verses of a *canto* we want to measure the length of generated text. Indeed our advanced model stops by itself the generation when they think the *canto* should finish and we expect that the length is quite close to those one of the Divine Comedy *cantos*, which are composed of a variable number of verses between 115 and 160[13];

- Number of strophes: here we count the groups of verses which are separated by an empty line. This value is very useful together with the next two following metrics to understand completely the correctness of *canto*'s structure;

- Number of well formed *terzine*: because the Divine Comedy is written in *terzine* and our model should learn this schema, in terms of structure it is important that they separate correctly verses. So for a perfect structure, this counter should be as the number of strophes minus 1 (for the last single verse);

- Last single verse: it simply indicates if the *canto* ends with a single verse. We would also add this metrics because looking at Divine Comedy all the *cantos* end with a lonely verse, so also this is going to contribute to a correct structure of the whole *canto*;

- Average syllables per verse: even if the length of verse is not more meaningful when we talk about hendecasyllables, we want to show the average verse's length in syllables of our generated *cantos* along with its variance, to understand also if Dante uses the most common *endecasillabo piano* or some rare forms of hendecasyllables composed by more than twelve syllables;

[13] https://it.wikipedia.org/wiki/Divina_Commedia

- Hendecasyllabicness score: a more accurate consideration we can do on each single verse is to assert that the verse respects the hendecasyllable's rules described in the *Linguistic rules* section, where the last stressed syllable must be always the tenth in the verse. Due to some errors of the Tone model and some exceptions in Italian syllabification rules, this metrics does not perform a perfect score neither on one *canto* of the Divine Comedy, but we will use it thinking that "the higher, the better";

- Rhymeness score: thanks to our Tone model, we are able to check every pair of verse to verify the "chained" rhyming pattern used by Dante (ABA BCB CDC ...) and because we always know the position of tone on a word we can decide with enough confidence if two words rhyme to each other or not. This metric is a score between 0 and 1 but it can be made a bit dirty by some errors in Tone model predictions.

OUR METRICS ON 1 CANTO OF DIVINE COMEDY	
Number of verses	136
Number of strophes	46
Number of well formed terzine	45
Last single verse	True
Average syllables per verse	11.07 ± 0.41
Hendecasyllabicness score	0.9044
Rhymeness score	0.9710

4.2. Computing metrics on generated *cantos*…

In the following tables, we summed up the metric results for every model. It is important to say that the presented results are related to the best generated example for each model.

Model by word

OUR METRICS ON GENERATED CANTO:	
Number of verses	151
Number of strophes	51
Number of well formed terzine	50
Last single verse	True
Average syllables per verse	11.03 ± 0.92
Hendecasyllabicness score	0.6026
Rhymeness score	0.0200

In this example, the model appeared to have learnt the correct structure in terms of *terzine* and single final verse, but it is clear that it did not stop because of the generation of a '<*end_of_canto*>', but due to our forced condition about the maximal length of 151 verses. In general, structure is generated correctly, as a multitude of *terzine*, but in most of the generated text we found one or two *terzine* that are not well-formed. Moreover, the model sometimes achieves in generating the '<*end_of_canto*>' token.

Hendecasyllabicness does not achieve good results, but the mean of verses' length is almost 11, which, as mentioned, it is not a completely reliable information, since the metric works depending on real hendecasyllabicness, but it shows it learnt more or less verses' length. This value is quite indicative for all generations performed by this kind of model.

For what concerns Rhymeness, in this case it is evident that the *Model by word* is not able to learn the rhyme schema. This is confirmed by all other text generations as well.

Model by character

OUR METRICS ON GENERATED CANTO:	
Number of verses	89
Number of strophes	30
Number of well formed terzine	29
Last single verse	False
Average syllables per verse	11.28 ± 0.60
Hendecasyllabicness score	0.6629
Rhymeness score	0.0111

In this example, the model generated the '<*end_of_canto*>' token but it generally did not use it coherently with the average length of *cantos*. Moreover, in the generated texts, there is generally at least one not well-formed *terzina*, plus it is quite common not to have the single final verse.

Hendecasyllabicness does not achieve good results, but the mean of verses' length is just slightly above 11, which, as mentioned, it is not a completely reliable information, since the metric works depending on real hendecasyllabicness, but it shows it learnt more or less verses' length. This value is quite indicative for all generations performed by this kind of model.

For what concerns Rhymeness, in this case it is evident that the *Model by character* is not able to learn the rhyme schema. This is confirmed by all other text generations as well.

Model by syllable

OUR METRICS ON GENERATED CANTO:	
Number of verses	151
Number of strophes	50
Number of well formed terzine	49
Last single verse	False
Average syllables per verse	11.05 ± 0.68
Hendecasyllabicness score	0.7351
Rhymeness score	0.0067

In this example, the model did not stop because of the generation of a '<*end_of_canto*>', but due to our forced condition about the maximal length of 151 verses. In general, structure is generated correctly, as a multitude of *terzine*, but in most of the generated text we found one or two *terzine* that are not well-formed.

We noticed an improvement in hendecasyllabicness, compared to the previous models' results. Since this value is indicative for all generations performed by this kind of model, achieving even higher scores in some cases, we understood that *Model by syllable* would be the best starting point for the *Advanced Models*.

For what concerns Rhymeness, in this case it is evident that the *Model by syllable* is not able to learn the rhyme schema. This is confirmed by all other text generations as well.

Model by reversed syllables

OUR METRICS ON GENERATED CANTO:	
Number of verses	163
Number of strophes	55
Number of well formed terzine	54
Last single verse	True
Average syllables per verse	11.12 ± 0.57
Hendecasyllabicness score	0.7607
Rhymeness score	0.9394

For this model, we decided not to force the end of the *canto*, therefore the model always stopped because of the generation of a '<*end_of_canto*>'. In general, generated texts appeared to be reasonably long, with the tendency to be slightly shorter than the average *canto*. In most cases, the structure is generated correctly, as a multitude of *terzine*, despite this, in some rare instances, we found one or two *terzine* that were not well-formed.

The improvement in hendecasyllabicness persisted.

For what concerns Rhymeness, results showed incredible improvement. This highlights that the strategy of dedicating a network to the rhymeness task revealed succeeding.

Model by toned and reversed syllables

OUR METRICS ON GENERATED CANTO:	
Number of verses	124
Number of strophes	42
Number of well formed terzine	41
Last single verse	True
Average syllables per verse	11.12 ± 0.53
Hendecasyllabicness score	0.7661
Rhymeness score	0.9365

Even in this case, we decided not to force the end of the *canto*, therefore the model always stopped because of the generation of a '<*end_of_canto*>'.

In general, generated texts appeared to be reasonably long, as in this example, but the overall tendency was to be slightly longer than the average *canto*. The generated structure is most frequently completely correct, as a multitude of *terzine* and with the presence of the final single verse.

The improvement in hendecasyllabicness remained the same as the *Model by reversed syllables*. Our hope of furtherly enhancing this score did not totally realize.

Rhymeness score confirmed the success of *Model by reversed syllables*' strategy.

In the following table, there is an example of the best result achieved in hendecasyllabicness, which happened to be of a generated *canto* with an unacceptable length:

OUR METRICS ON GENERATED CANTO:	
Number of verses	301
Number of strophes	101
Number of well formed terzine	100
Last single verse	True
Average syllables per verse	11.10 ± 0.59
Hendecasyllabicness score	0.7940
Rhymeness score	0.9340

Comparison between scores

All in all, the Baseline models' results appeared incredibly worse than the Advanced models' ones, accordingly with the impressions we got reading the generated text in *Text generation* sections.

Between *Model by reversed syllables* and *Model by toned and reversed syllables*, which both got high scores in rhymeness, we think that the latter is our best model because it considers more coherently and correctly the concept of rhyme and it has a proper way of syllabifying verses, which permitted to better relate with hendecasyllabicness.
Moreover *Model by toned reversed syllables* brought a further improvement to the Hendecasyllabicness score, even if slight, and it appeared to be the model to better learn the structure of the *cantos*, generating almost in every case well-formed *terzine* and final single verse.

4.3. Other metrics

OTHER METRICS ON GENERATED CANTO:	
Number of putative terzine	41
Number of well formed terzine	41
Average structuredness	0.9940
Average hendecasyllabicness	0.9246
Average rhymeness	1.0
Plagiarism	0.8502

As suggested, we compared a *canto* obtained by our *Model by toned and reversed syllables*, which we considered as our best model, to the results achieved by our colleagues, using their metrics. In particular, thanks to the Plagiarism score[14], we noticed that sometimes our best model tends to write some verses which are almost the same as those present in the Divine Comedy.

[14] Project of Luca Salvatore Lorello and Giorgio Tsiotas

4.4. Generated text

Here is a *canto* generated by our best model, *Model by toned and reversed syllables*:

CANTO

```
un altro ché non credi tu paventi
licito poeta de la tua fame
dirò de s'appiattò miser li denti

qual era cacciator per tutte brame
raggio diè dito e poi vedea temendo
ingiusto fece di quelli a gran dame

che solo il mio il qual io riprendo
chercuti a la luce foglia sospinto
perché questi ciascun lor maggior temendo

molto temprando per l'aere dipinto
viso pelato che tagne ed ello
su la fiumana ove 'l mar non distinto

contente furon d'acqua e danïello
a la voce che ciò l'accende d'ira
e io maestro disdegnoso e fello

io anime nomota e tira
corso ch'io ferna a e altra cura
io dico come sotto mi spira

onde nel novo di fresca verdura
poscia ch'io dico non d'altro cagione
io vive e sente e sé stesso misura

come 'n peschiera che furon cagione
che due figli a riveder le stelle
così fer molti antichi di guittone

cominciò 'l duca mio che mi favelle
or voi digi la mente ma di pria
regolar da l'altra già m'avea felle

com' io vidi che richesto più s'india
e tante fissi e a mente giovanni
e amor ne farà cosa maria

gravi stava mentre com' aso scanni
e come quei che più innanzi appariro
ambo le ruote e vidi uscirne un anni

così per lo suo corso si fuggiro
che ne l'avea ben letta questa vita
la bella donna dirizzò lo spiro

e quanti verso lei perché sortita
ch'avea certo colore e certo segno
per che l'ha tanta discordia assalita
```

```
e sotto molle e quella del mio ingegno
e non fu da li tuoi occhi riprende
e come abete in alto si fu degno

tempo è in questa gente scoscende
ohi ombre portate se demano
quindi non è se non è chi la 'ntende

subitamente mi prese per mano
sé desideroso va si senta
con esso i piè ma qui pon giù non sano

che tu dei a colui che si argomenta
veggendo atare e spade ma vede
fosse orazïone in prima senta

lo fele e vo di me nel mondo riede
credendo quella quindi esser decisa
vendetta del piacer che 'nvidia le diede

vegnati in voglia del peccatore a guisa
el cominciò il qual prender si puote
e la natura e gorgoglian ne la risa

dunque che faccia il secol per sue ruote
di quel color che dio per la qual forse
giù nel bene operar non ti percuote

che sopra l'amor ch'a noi si torse
faccianli onore ed esser può far lo giove
ti giran sì parlava ed el trascorse

del süo astanza i glorïoso move
chi l'uno e l'altro vidi tal milizia
son li giusti occhi tuoi rivolti altrove

rispuose l'un d'i pastor vostra giustizia
questo fia che par foco l'argomento
a guido e anco ad alcuna nequizia

io fui guido del duro pavimento
onte né degno or m'aiutate
e veliquie nel tuo vero attento

e 'l capo ed eran dinanzi aggroppate
di cui più didorami quel che sforza
de la montagna o che sie fuor passate

nostro intelletto e s'altri non s'ammorza
e io de l'antico e qual io vidi un foco
fossi di là per vïolenza il torza

qui se' sì poscia ch'io volsi per poco
e io che son giaciuto a queste fero
oh secaritate arde in quel loco

e 'l canto di quel del corpo intero
se tu dei a colui che si digrada
```

ne l'altro universo dio severo

non saver tra lita più che strada
e ciò che fo le vostre diece sciolte
primo perché non lagrimai né rada

a molti son molte piante necolte
com' io a la galodomi del casso
la carità del regno e tanta volte

da noi venite e troverete il passo
ma poi diss' io aperta per te stesso
poscia che iunone al pel com' uom lasso

prendendo la fortuna volse in basso
grave a la terra men nota perché forte
esser restate ma studiate il passo

o anima non vi fé sì torte
ubidire a la sua beata vista
che per conoscer le prima corte

si sentir ché l'una de l'altro lista
poscia che le peccata mia offesa
tal vera perfezion quivi s'acquista

de le due come albero a lui tesa
de la sua virtute diece tintinno
e solo a render prima non è intesa

s'ella non vide mai veder lo rinno
e dì nel cielo io me ne melode
quando lo 'mperador che sempre l'inno

rispuos' io lui mi smarri' in una lode
bestemmian quivi per lo fecer vinci
e di loro il mondo esser non ode

forse non impediva tanto quinci
ché quel che non è una silocosa
come dintorno le pecore e vinci

come nel suo figlio ch'ancor troppo osa
di te che non vuol che più belli
ancor nel dolce mondo sanza posa

e drizzò il dito questo giron suggelli
e non ti primo e risalire in suso
quel corno qui per odi muover quelli

né li altri due un serpentello accuso
che l'anima ch'ad ogne passo vero
villan che la ripa ch'era dischiuso

questo radio omai più in quel cero

5. CONCLUSIONS

We started by attempting our goal with simple approaches, in order to find out which were the main challenges that came with the major objective of this work. While doing so, we tried out many architectures and components' combinations in order to cover different possibilities and find out which could give us the best result in this specific context. In particular, we tested three different layers architectures for each of three models identified by different units of text: by words, by characters and by syllables.

Moreover, we furtherly delved into this project's domain, analyzing how to possibly implement the main poetry features of this domain, especially the concept of poetry rhythm, *terza rima* rhyme scheme and hendecasyllabicness. We did so even by following specialized strategies and building and training dedicated networks, in order to represent our context. In particular, we developed two additional models, both involving further specialized networks. These two models had a rhyme dedicated network and a verse completion network, the second also had a first step network responsible for setting the stressed vowels of the text's words with a tone.

In addition, in order to be able to evaluate the output of such particular outputs and context, we took care of designing meaningful metrics, for each of the identified features that the output had to pursue.

According to the scores obtained with these metrics, our best network is the *Model by toned and reversed syllables*.

All in all, we think that specializing networks has been a winning strategy, in particular in such a polyhedral task. This is supported by the results that our best network achieved.