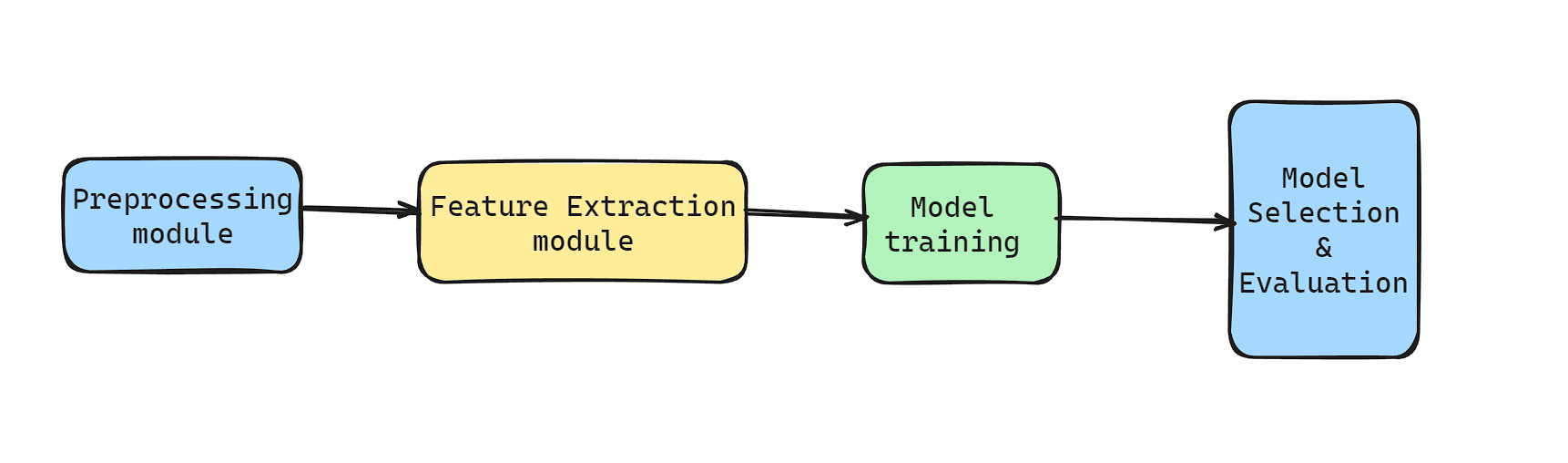
**NLP Project**

**Arabic-text-Diacritization**

|  |  |
| --- | --- |
| **Team Members** | |
| **Name** | **Email** |
| **Abdelaziz Salah Mohammed Abdo** | **Abdelaziz**[**132001@gmail.com**](mailto:132001@gmail.com) |
| **Abdelrahman fathi** | **fathi791112@gmail.com** |
| **Khaled Hesham Elsayed Taha** | **kha.hesham@gmail.com** |
| **Kirollos Samy** | **KirollosSamyHakim@gmail.com** |

1. **Project Pipeline:**

* **Training**:

We first clean the training data, then pass the cleaned data to extract features, then pass the features to train the model, then we tune the hyperparameters on the training set. Finally, we evaluate the model performance on the validation set and select the best model.

* **Inference**:

Like training, we first pass the test data on the preprocessing module, then load the weights of the model to be used for inference. The model is then used to predict the diacritics and finally we evaluate the performance.

1. **Models Arch:**

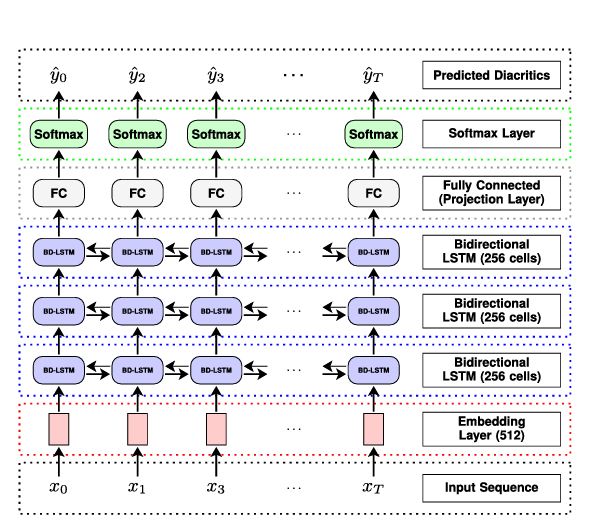
We tried two model architectures; from a paper they called the first the baseline model which is a straightforward approach. The second model is more sophisticated, which they call the CBHG model.

**Baseline Model:**

This is a simple model architecture that works surprisingly well despite its simplicity. It consists of a built-in embedding layer which works on the character level. This layer can be replaced with our other feature extraction methods like mixing word embeddings with character embeddings or can be replaced with a pretrained model for character embeddings.

The core of the model is 3 layers of **Stacked Bidirectional LSTMs**.

The output of the last layer of LSTM is passed to a fully connected layer which acts as a projection layer. Finally, a SoftMax layer is used to compute the probability distribution of the diacritic classes.

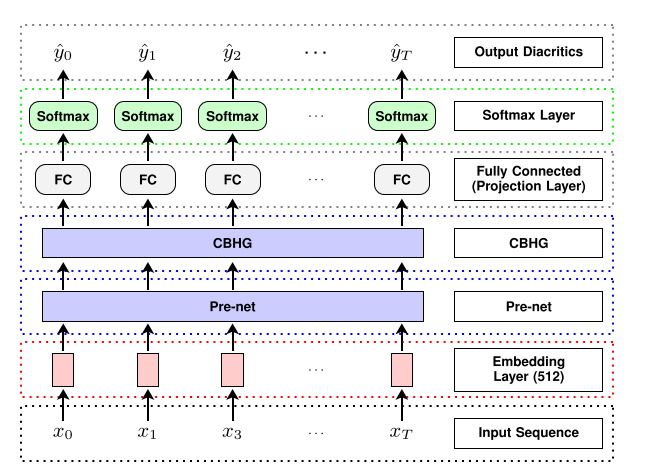


**CBHG Model:**

The CBHG model works as follows: a 512-dimensions embedding layer ﬁrst processes the input sequence. The embedding output is passed to two layers of non-linear trans-formation called pre-net: the ﬁrst consists of 512 units with a RELU activation function and a dropout probability of 0.5, and the second consists of 256 units with the same activation and dropout probability. The pre-net output is then fed to the CBHG module, which outputs the input sequence’s ﬁnal representation. We added a fully connected layer to project down the CBHG module’s output to the number of possible diacritics. Lastly, we used a SoftMax layer to output the probability distribution for each diacritic this model were introduced in an [IEEE paper](https://ieeexplore.ieee.org/document/9274427) this was very powerful in solving seq to seq problems.

here is a detailed explanation about each part in our model:

1. **Embedding Layer**
   1. **Functionality**: Converts discrete input features into continuous vector representations.
   2. **Importance**: Enables the model to learn meaningful continuous representations of input symbols, which is crucial for capturing relationships and patterns in sequential data.
2. **Prenet**
   1. **Functionality:** Applies a series of linear transformations with ReLU activation and dropout to the embedded input.
   2. **Importance:** Helps in feature transformation and non-linearity, allowing the model to focus on relevant information and improving the overall representational power.
3. **Convolutional Bank Highway Network Gated Recurrent Unit)**
   1. **Convolutional Banks:**
      1. **Functionality:**  Extracts features at different context sizes through multiple convolutional filters.
      2. **Importance:** Captures local patterns and varying contextual information in the input sequence.
   2. **Highway Network:**
      1. **Functionality:** Facilitates the flow of information through shortcut connections, allowing the model to selectively pass important features.
      2. **Importance:** Helps in mitigating the vanishing gradient problem, allowing the model to effectively learn long-range dependencies.
   3. **Gated Recurrent Unit (GRU):**
      1. **Functionality:** Captures sequential dependencies in both forward and backward directions.
      2. **Importance:** Captures long-term dependencies and contextual information across the entire input sequence.
4. **Post-CBHG Layers (LSTM Layers with Batch Normalization):**
   1. **Functionality:** Further processes the contextual information captured by the CBHG block.
   2. **Importance:** Adds additional depth to the model, allowing it to refine and extract more complex features from the sequential input.
5. **Projections Layer:**
   1. **Functionality:** Maps the final contextual features to the output space.
   2. **Importance:** Produces the final output sequence, whether it be a sequence of labels or some other relevant information.
6. **SoftMax Layer:**
   1. Applies the SoftMax function to the output vector to select the best class out of 15 different classes to represent our target diacritic for this character



1. **Detailed Description of each phase:**

**2.1- Data preprocessing:**

* The data preprocessing can be divided into two stages:

1. Data Cleaning:

We clean the corpus, removing everything that is not an Arabic letter nor a punctuation, we also retain separators like full stops and spaces to be used during sentence segmentation and word tokenization.

1. Sentence segmentation: This phase involves segmenting the corpus into sentences. It’s challenging in Arabic to identify sentence boundaries.

In addition, relying solely on a period would result in excessively long sentences, as Arabic writings often link sentences with coordinators, using a period only at the end of each paragraph. This makes the maximum sentence size very long which affects the training since all the sentences are padded to the maximum size.

To address this, we first split on periods, if the sentence size is still longer than a certain threshold, we further split on other separators like commas, and semicolons.

**2.2- Feature extraction:**

* **First method:**

We rely on a built-in embedding layer which is trained along with the model to learn the character embeddings. This layer can be thought of as a transformation layer which transforms the encoded character to its embedding.

* **Second method:**

We Tried to use the contextual embeddings using a pre-trained model to generate embeddings for individual characters using the Bidirectional Encoder Representations from Transformers (BERT) or Robust optimized BERT approach (RoBERTa). And this was easily done by just importing the transformers library and calling the model by its name such as **‘bert-base-arabic’** and applying the tokenization then getting the embeddings from the last hidden layer. But when we used the first method, we got the highest accuracy, so we stick with it.

* **Third method**

same as the second method but applying it on the word level. And we also found that the first method provides us with better accuracy, so we stick to it.

**2.3- Model training and Evaluation:**

we tested our model after training on the training dataset and validating with the val.txt we found that baseline model best accuracy was about **95%** which is not bad regards the model size and simple architecture while on training the CBHG model we find accuracy on val.txt nearly **97.5%** which is quite good compared to the state of the art accuracy we believe that CBHG model can be more accurate but it needs further tunning of its hyperparameters here is a table of mostly used trials and approximately the accuracy of each one:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **version** | **Model** | **learning rate** | **epochs** | **batch size** | **accuracy** |
| 1 | baseline | 0.01 | 10 | 512 | 94.5% |
| 2 | baseline | 0.001 | 10 | 512 | 95.5% |
| 3 | CBHG | 0.01 | 10 | 512 | 93.2% |
| 4 | CBHG | 0.01 | 10 | 256 | 96.7 |
| 5 | CBHG | 0.001 | 20 | 256 | 97.3 |
| 6 | CBHG | 0.0009 | 20 | 256 | 97.6 |
| **7** | **CBHG** | **0.001** | **10** | **64** | **97.5** |
| 8 | CBHG | 0.001 | 10 | 32 | 96.4% |

**Used Model in the test set submission on Kaggle:**

The used model on submission is the highlighted one with green above the other models may have similar accuracy or some of them approaches accuracy 99.3% in training but while testing accuracy decreases to 95% which I guess was overfitted so we sticked to version **7** which predicts around **96.2%** of diacritics correctly