CSE484 HW-3

Emre Yılmaz - 1901042606

January 29, 2024

1 Important Note

IMPORTANT NOTE IN TURKISH: Model, çıktı olarak True ya da False değil, verilen cümledeki ”de” veya ”ki” bağlaç mı yoksa ek mi olduğunu vermektedir. Cümleyi yanlış da yazsanız, doğru da yazsanız bu sonuç değişmemektedir. Yani ”Masada et var” cümlesinin çıktısı DE_SUF (Ek) iken, ”Masa da et var” cümlesinin çıktısı da DE_SUF (Ek)'tir. Sonuçlar hem doğru yazılmış, hem de yanlış yazılmış cümlelerle test edilerek bu durum gösterilmiştir. Figure 7'deki sondan ikinci örnekte, sondan üçüncü örnekte, sondan dördüncü örnekte, sondan beşinci örnekte bunu görebilirsiniz.

2 Introduction

The task is to develop a classifier that predicts whether the suffixes 'de' and 'ki' and their derivatives in Turkish should be written separately or together. For this, the Turkish Wikipedia dataset was used. First, the dataset was cleaned, and parts that should not be evaluated were removed. Then, all sentences were extracted from the dataset and filtered so that each sentence contained only one instance of the suffix 'de' or 'ki.' Following this, tokenization was performed, and the necessary JSON file for the model was created. The details will be processed step by step.

Google Colab Pro GPUs are used for training in this homework.

The results in this report and the outputs of ipynb file may be a little bit different because of the only sample sizes.

3 Preprocessing

3.1 Data

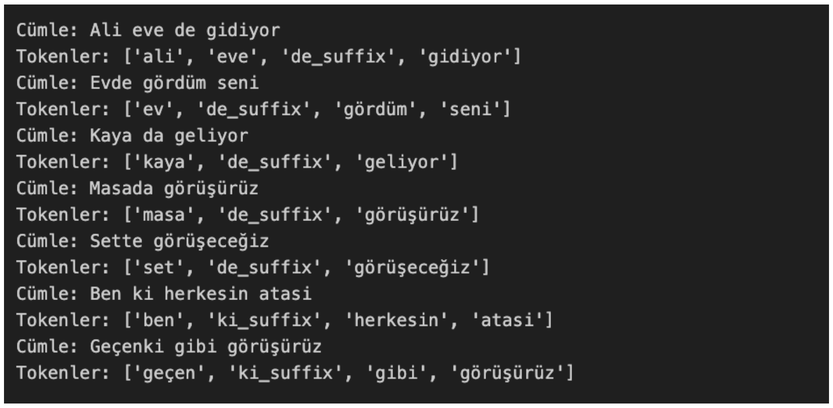
To accomplish this task, the Turkish Wikipedia dataset has been used from kaggle

3.2 Cleaning and Extracting Sentences

Initially, parts containing links were removed from the dataset. Then, sentences that were mixed up were organized, and each line was formatted to be one sentence. After this, a filtering operation was applied. Sentences containing only one instance of the suffix 'ki' or only one 'de' were filtered, and all other sentences were discarded. Both suffixes written separately and together were included in this filtering process.

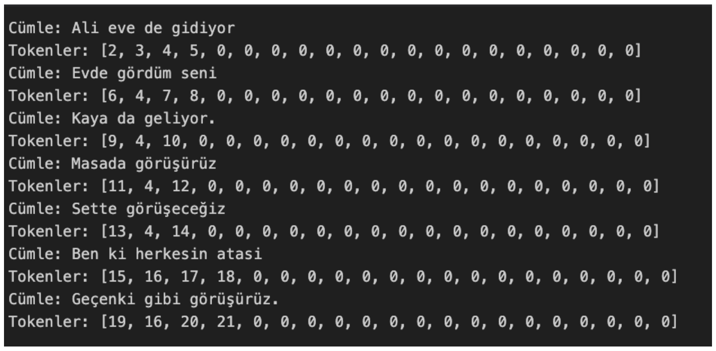
3.3 Tokenization and Generating Dictionary

Subsequently, a tokenization process was applied. During this process, no library was used, as tokenizing every word would have resulted in the loss of 'de' and 'ki' suffixes written together with words. I wrote my own tokenize function. Suffixes 'de' written together and separately were tokenized as 'de_suffix.' Similarly, 'ki' suffixes, whether written together or separately, were tokenized as 'ki_suffix'. You can see an example of this tokenization process in Figure 1.



```
Cümle: Ali eve de gidiyor
Tokenler: ['ali', 'eve', 'de_suffix', 'gidiyor']
Cümle: Evde gördüm seni
Tokenler: ['ev', 'de_suffix', 'gördüm', 'seni']
Cümle: Kaya da geliyor
Tokenler: ['kaya', 'de_suffix', 'geliyor']
Cümle: Masada görüşürüz
Tokenler: ['masa', 'de_suffix', 'görüşürüz']
Cümle: Sette görüşeceğiz
Tokenler: ['set', 'de_suffix', 'görüşeceğiz']
Cümle: Ben ki herkesin atasi
Tokenler: ['ben', 'ki_suffix', 'herkesin', 'atasi']
Cümle: Geçenki gibi görüşürüz
Tokenler: ['geçen', 'ki_suffix', 'gibi', 'görüşürüz']
```

Figure 1: Some Examples of Tokenization Result

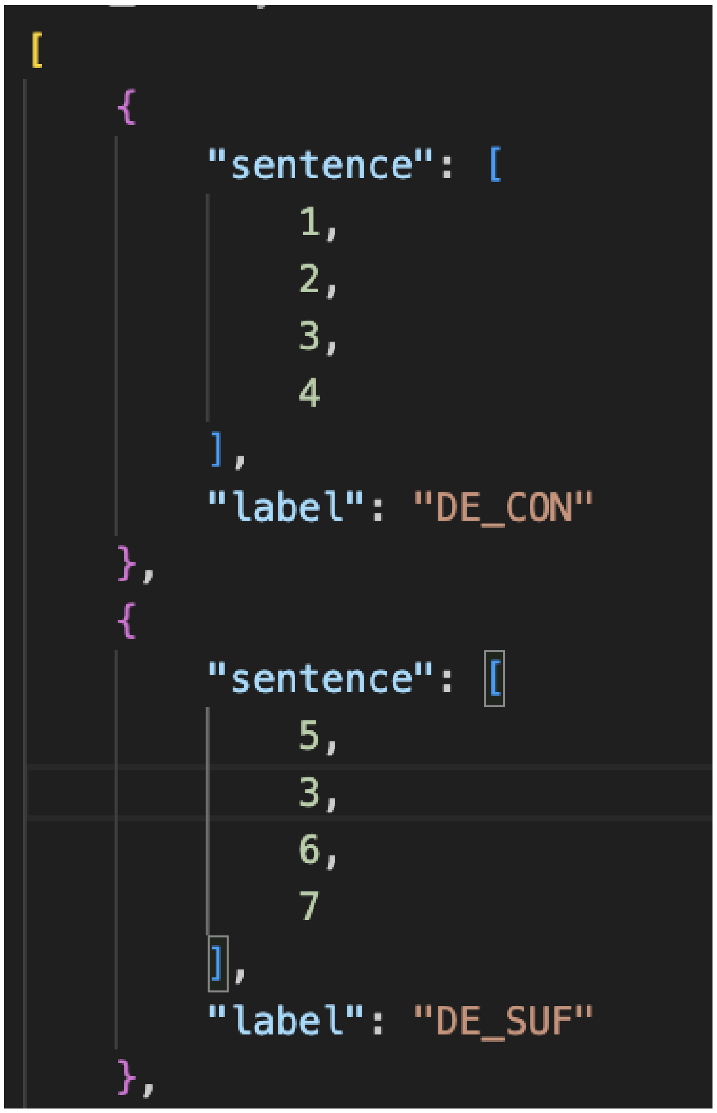


```
Cümle: Ali eve de gidiyor
Tokenler: [2, 3, 4, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Cümle: Evde gördüm seni
Tokenler: [6, 4, 7, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Cümle: Kaya da geliyor.
Tokenler: [9, 4, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Cümle: Masada görüşürüz
Tokenler: [11, 4, 12, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Cümle: Sette görüşeceğiz
Tokenler: [13, 4, 14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Cümle: Ben ki herkesin atasi
Tokenler: [15, 16, 17, 18, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Cümle: Geçenki gibi görüşürüz.
Tokenler: [19, 16, 20, 21, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Figure 2: Some Examples of Numerical Tokens

During this tokenization process, a maximum length was established, and padding was applied to the lists of tokens. This is because for our model to accept the input, each input must be of the same length. You can see some examples of this tokenization process in Figure 2. As you can see, all token lists have the same length. Also, all ”de” and ”ki” suffixes have the same token as you can see.

Following this, all sentences were processed, and a dictionary (vocabulary) was created where each word is assigned a unique numerical value.

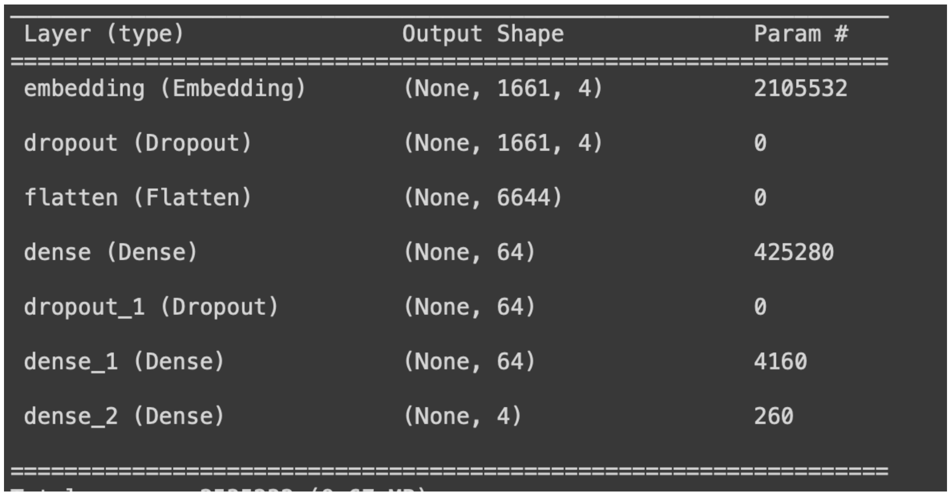


```
[
    {
        "sentence": [
            1,
            2,
            3,
            4
        ],
        "label": "DE_CON"
    },
    {
        "sentence": [
            5,
            3,
            6,
            7
        ],
        "label": "DE_SUF"
    },
```

Figure 3: Example of a Label

3.4 Label and JSON Files

Afterward, all sentences were revisited, and a corresponding list of tokens for each sentence was created. Along with this, each sentence was labeled based on whether the suffix it contained was written together or separately. For example, if the sentence is 'Masada kalem var' (There is a pen on the table), it was labeled as 'DE_SUFFIX.' If the sentence is 'Sen de gel' (You come too), it was labeled as 'DE_CON.' Each sentence, along with its corresponding list of tokens and label, was transferred to a JSON file. Thus, labels for use in the model were obtained. You can see an example of a label in Figure 3.



```
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 1661, 4)           2105532

 dropout (Dropout)           (None, 1661, 4)           0

 flatten (Flatten)           (None, 6644)              0

 dense (Dense)               (None, 64)                425280

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 4)                 260

=================================================================
```

Figure 4: Model Summary

4 Training

You can see the summary of my model in Figure 4 above.

4.1 Embedding Layer

Embedding(input_dim=len(vocab), output_dim=4, input_length=max_token_count): This layer transforms integer arrays, which the model takes as input, into dense vectors. Each word (token) is converted into a vector of a specified size (output_dim). input_dim represents the size of the vocabulary (vocab), and input_length indicates the maximum length of the input sentences. This layer is commonly used for text data and helps capture relationships between words better.

4.2 Dropout Layer

Dropout(0.5): The model contains two Dropout layers. Each is used to prevent overfitting. The Dropout layer "turns off" randomly selected neurons during training (i.e., temporarily deactivates them) to prevent the model from becoming overly dependent on specific neurons.

4.3 Flatten Layer

Flatten(): Converts the multi-dimensional output from the Embedding layer into a flat vector. This makes the output of the Embedding layer compatible with the input format of the Dense layer.

4.4 Dense Layers

Dense(hidden_units, activation='relu'): There are two Dense layers. Each consists of a certain number of neurons (hidden_units) and uses the relu activation function. These layers enhance the model's learning ability and can capture complex relationships.

4.5 Output Layer

Dense(output_dim, activation='softmax'): This is the final layer of the model and uses the softmax activation function. output_dim denotes the number of classes the model is trying to predict. This layer produces a probability distribution for each class, and the class with the highest probability is selected as the model's prediction.

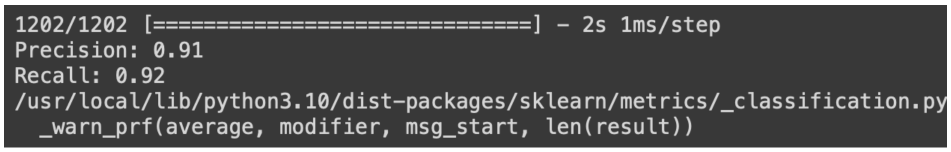
4.6 Model Compilation and Training

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy']): Model is compiled with the adam optimizer, sparse_categorical_crossentropy loss function, and accuracy metric. However, while evaluating my results, I use Precision and Recall values. I am going to explain it in Results and Conclusion chapter.

5 Results and Conclusions

5.1 Precision - Recall

In the success evaluation conducted with the test set, the precision value was 0.91, and the recall value was 0.92 (Figure 5).



```
1202/1202 [==============================] - 2s 1ms/step
Precision: 0.91
Recall: 0.92
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py
  _warn_prf(average, modifier, msg_start, len(result))
```

Figure 5: Precision and Recall Result

Considering that the accuracy value at the end of the model is also around 0.93, we can conclude that our model is quite successful on the Wikipedia dataset. Let's now try it with our own sentences.

5.2 Test

***VERY IMPORTANT NOTE:* Model, çıktı olarak True ya da False değil, verilen cümledeki "de" veya "ki" bağlaç mı yoksa ek mi olduğunu vermektedir. Cümleyi yanlış da yazsanız, doğru da yazsanız bu sonuç değişmemektedir. Yani "Masada et var" cümlesinin çıktısı DE_SUF (Ek) iken, "Masa da et var" cümlesinin çıktısı da DE_SUF (Ek)'tir. Sonuçlar hem doğru yazılmış, hem de yanlış yazılmış cümlelerle test edilerek bu durum gösterilmiştir. Figure 7'deki sondan ikinci örnekte, sondan üçüncü örnekte, sondan dördüncü örnekte, sondan beşinci örnekte bunu görebilirsiniz.**

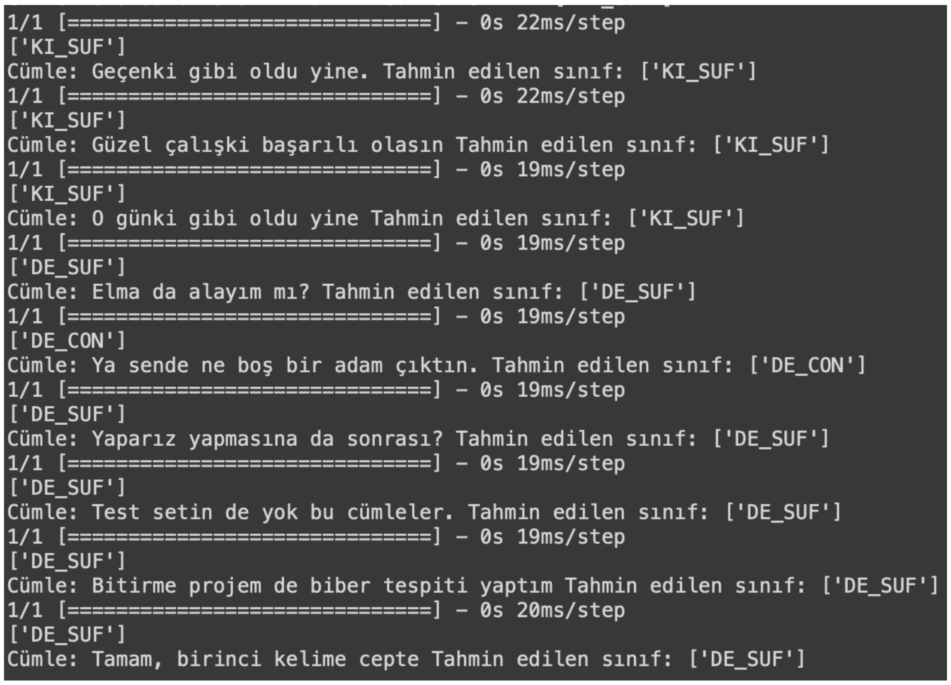
Our model produces four different outputs.

1. DE_CON - Bağlaç durumu. The suffix 'de' written separately.
2. KI_CON - Bağlaç durumu. The suffix 'ki' written separately.
3. DE_SUF - Ek durumu. The suffix 'de' written together.
4. KI_SUF - Ek durumu. The suffix 'ki' written together.

Firstly, it must be said that the model is completely unsuccessful regarding the separation of the suffix 'ki.' It classifies every instance as needing to be written together. Upon investigation, I found that the number of separately written 'ki' words in the Wikipedia dataset is extremely low. Therefore, our model fails in practice in distinguishing the 'ki' suffix. However, the reason for high test and validation results is again due to the limited number of sentences containing 'ki' written together in these datasets.

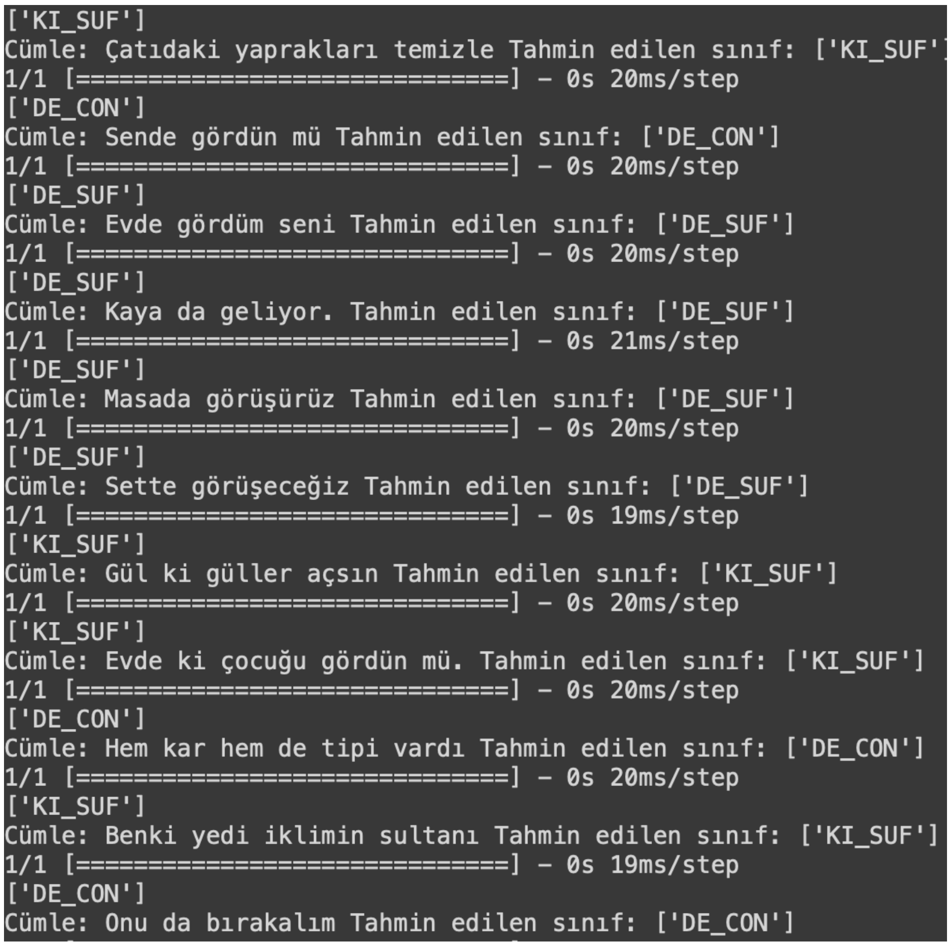
Now, let's examine the 'de' examples in Figure 6-7 one by one:"

- *"Sende gördün mü?"* = Model classified it as DE_CONC (Bağlaç) successfuly. Input is wrong. It must be conjunction (bağlaç).
- *"Evde gördüm seni"* = Model classified it as DE_SUF (Ek) successfuly. Input is correct.
- *"Test setin de yok bu cümleler"* = Model failed. It classified as DE_SUF (Ek). Input was wrong
- *"Kaya da geliyor"* = Model failed. It classified it as DE_CONC (Bağlaç). Input was correct.
- *"Bitirme projem de biber tespiti yaptım"* = Model classified it as DE_CON (Bağlaç) successfuly. Input was wrong.
- *"Masada görüşürüz"* = Model classified it as DE_SUF (Ek) successfuly. Input is correct.
- *"Sette görüşeceğiz"* = Model classified it as DE_SUF (Ek) successfuly. Input is correct.
- *"Hem kar hem de tipi vardı"* = Model classified it as DE_CON (Bağlaç). Input is correct.
- *"Onu da bırakalım"* = Model classified it as DE_CON (Bağlaç) successfuly. Input is correct.
- *"Elma da alayım mı?"* = Model failed. Model classified it as DE_SUF (Ek). Input was correct.
- *"Ya sende ne boş bir adam çıktın"* = Model classified it as DE_CON (Bağlaç) successfuly. Input was wrong
- *"Yaparız yapmasına da sonrası?"* = Model failed. Model classified it as DE_SUF. Input was correct.
- *"Tamam, birinci kelime cepte"* = Model classified it as DE_SUF (Ek) successfuly. Input was correct.



```
1/1 [==============================] - 0s 22ms/step
['KI_SUF']
Cümle: Geçenki gibi oldu yine. Tahmin edilen sınıf: ['KI_SUF']
1/1 [==============================] - 0s 22ms/step
['KI_SUF']
Cümle: Güzel çalışki başarılı olasın Tahmin edilen sınıf: ['KI_SUF']
1/1 [==============================] - 0s 19ms/step
['KI_SUF']
Cümle: O günki gibi oldu yine Tahmin edilen sınıf: ['KI_SUF']
1/1 [==============================] - 0s 19ms/step
['DE_SUF']
Cümle: Elma da alayım mı? Tahmin edilen sınıf: ['DE_SUF']
1/1 [==============================] - 0s 19ms/step
['DE_CON']
Cümle: Ya sende ne boş bir adam çıktın. Tahmin edilen sınıf: ['DE_CON']
1/1 [==============================] - 0s 19ms/step
['DE_SUF']
Cümle: Yaparız yapmasına da sonrası? Tahmin edilen sınıf: ['DE_SUF']
1/1 [==============================] - 0s 19ms/step
['DE_SUF']
Cümle: Test setin de yok bu cümleler. Tahmin edilen sınıf: ['DE_SUF']
1/1 [==============================] - 0s 19ms/step
['DE_SUF']
Cümle: Bitirme projem de biber tespiti yaptım Tahmin edilen sınıf: ['DE_SUF']
1/1 [==============================] - 0s 20ms/step
['DE_SUF']
Cümle: Tamam, birinci kelime cepte Tahmin edilen sınıf: ['DE_SUF']
```

Figure 6: Result Examples



```
['KI_SUF']
Cümle: Çatıdaki yaprakları temizle Tahmin edilen sınıf: ['KI_SUF']
1/1 [==============================] - 0s 20ms/step
['DE_CON']
Cümle: Sende gördün mü Tahmin edilen sınıf: ['DE_CON']
1/1 [==============================] - 0s 20ms/step
['DE_SUF']
Cümle: Evde gördüm seni Tahmin edilen sınıf: ['DE_SUF']
1/1 [==============================] - 0s 20ms/step
['DE_SUF']
Cümle: Kaya da geliyor. Tahmin edilen sınıf: ['DE_SUF']
1/1 [==============================] - 0s 21ms/step
['DE_SUF']
Cümle: Masada görüşürüz Tahmin edilen sınıf: ['DE_SUF']
1/1 [==============================] - 0s 20ms/step
['DE_SUF']
Cümle: Sette görüşeceğiz Tahmin edilen sınıf: ['DE_SUF']
1/1 [==============================] - 0s 19ms/step
['KI_SUF']
Cümle: Gül ki güller açsın Tahmin edilen sınıf: ['KI_SUF']
1/1 [==============================] - 0s 20ms/step
['KI_SUF']
Cümle: Evde ki çocuğu gördün mü. Tahmin edilen sınıf: ['KI_SUF']
1/1 [==============================] - 0s 20ms/step
['DE_CON']
Cümle: Hem kar hem de tipi vardı Tahmin edilen sınıf: ['DE_CON']
1/1 [==============================] - 0s 20ms/step
['KI_SUF']
Cümle: Benki yedi iklimin sultanı Tahmin edilen sınıf: ['KI_SUF']
1/1 [==============================] - 0s 19ms/step
['DE_CON']
Cümle: Onu da bırakalım Tahmin edilen sınıf: ['DE_CON']
```

Figure 7: Result Examples