Turkish "de", "ki" Suffix/Conjunction Classifier CSE484 HW3

Burak Kocausta
1901042605

January 2024

Contents

1 Introduction

Aim is implementing a binary classifier which gaves the result for "ki", "de" if they are conjunction or suffix. Input sentences are given as "not separated". Model finds if it is "separated" or "not separated".

2 Training Data, Pre-processing, Tokenizing

2.1 Dataset

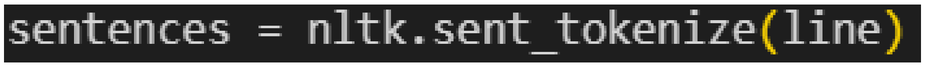
I used 3 different dataset.

1. Turkish Sentences: Consists of separate Turkish Sentence examples. They are useful because sentences are needed. (10.68 MB)
2. Turkish Wikipedia Dump: Large dataset consists of Turkish texts. (452.17 MB)
3. Turkish Corpus: Large dataset which includes other then wikipedia content. (745.21 MB)

   I used all of their content for preprocessing step.

2.2 Pre-processing

At each iteration, all sentences are extracted using ”nltk” library.

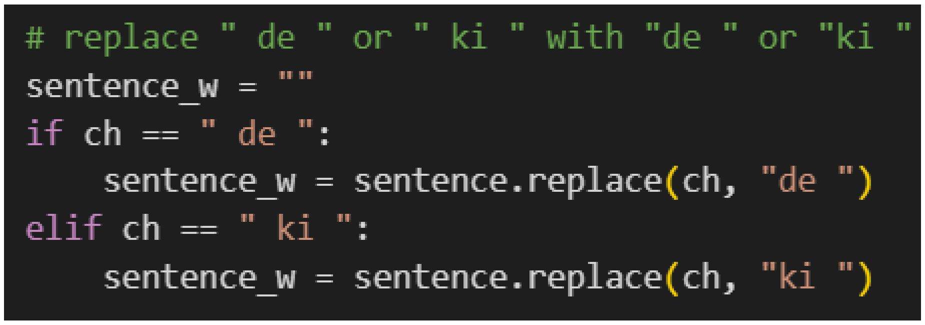


```
sentences = nltk.sent_tokenize(line)
```

Figure 1: Getting Sentences with nltk

Sentence must have punctuation, if it does not include any, they are discarded. Wikidump has unnecessary characters, so they are discarded with only including alphabets. Number of words in a sentences are limited to 15.

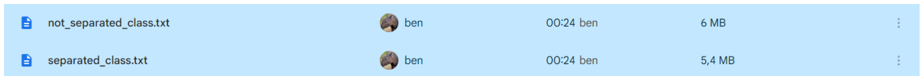
Inputs must be given as ”not separated”, so conjunctions are connected to it. Suffix versions are directly written to the file.



```
# replace " de " or " ki " with "de " or "ki "
sentence_w = ""
if ch == " de ":
    sentence_w = sentence.replace(ch, "de ")
elif ch == " ki ":
    sentence_w = sentence.replace(ch, "ki ")
```

Figure 2: Connecting conjunctions

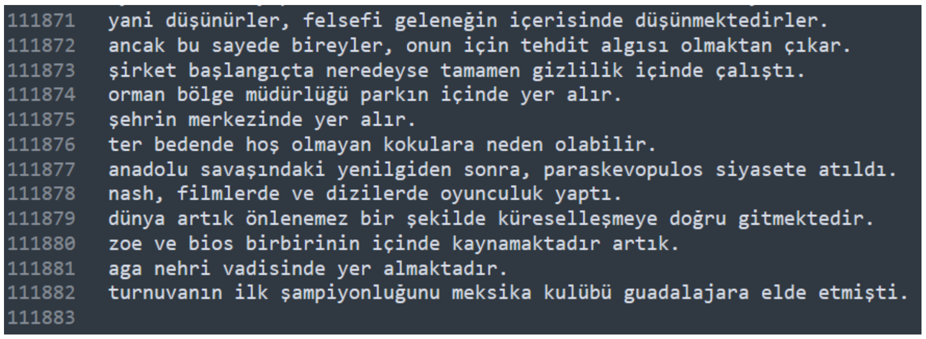
They are saved to 2 different files ”separated_class.txt”, and ”not_separated_class.txt”. Those are the pre-processed sentences.



```
not_separated_class.txt    ben    00:24 ben    6 MB
separated_class.txt        ben    00:24 ben    5,4 MB
```

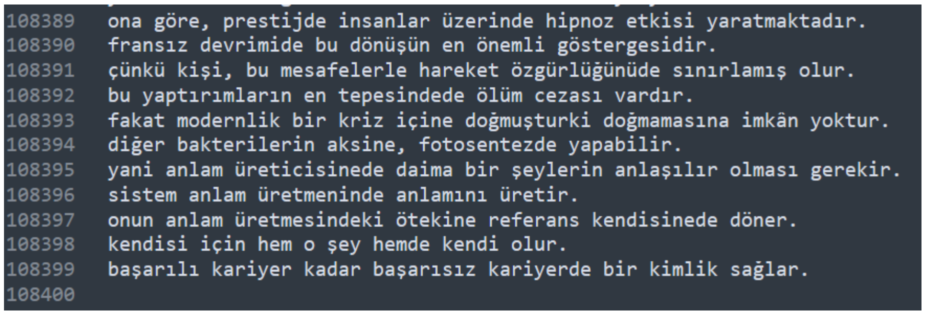
Figure 3: Preprocessed and Separated Sentences

Those sentences will be used for training and testing (80%, 20%).



```
111871   yani düşünürler, felsefi geleneğin içerisinde düşünmektedirler.
111872   ancak bu sayede bireyler, onun için tehdit algısı olmaktan çıkar.
111873   şirket başlangıçta neredeyse tamamen gizlilik içinde çalıştı.
111874   orman bölge müdürlüğü parkın içinde yer alır.
111875   şehrin merkezinde yer alır.
111876   ter bedende hoş olmayan kokulara neden olabilir.
111877   anadolu savaşındaki yenilgiden sonra, paraskevopulos siyasete atıldı.
111878   nash, filmlerde ve dizilerde oyunculuk yaptı.
111879   dünya artık önlenemez bir şekilde küreselleşmeye doğru gitmektedir.
111880   zoe ve bios birbirinin içinde kaynamaktadır artık.
111881   aga nehri vadisinde yer almaktadır.
111882   turnuvanın ilk şampiyonluğunu meksika kulübü guadalajara elde etmişti.
111883
```

Figure 4: Sentences which are labeled as ”Not Separated” and number of sentences for that class



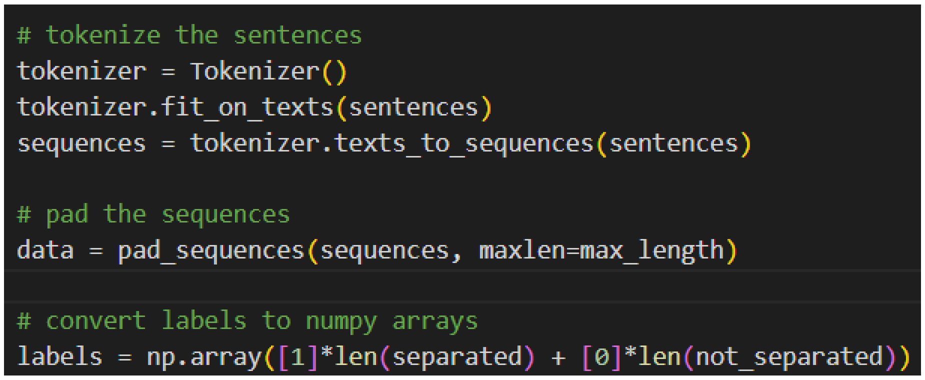
```
108389   ona göre, prestijde insanlar üzerinde hipnoz etkisi yaratmaktadır.
108390   fransız devrimide bu dönüşün en önemli göstergesidir.
108391   çünkü kişi, bu mesafelerle hareket özgürlüğünüde sınırlamış olur.
108392   bu yaptırımların en tepesindede ölüm cezası vardır.
108393   fakat modernlik bir kriz içine doğmuşturki doğmamasına imkân yoktur.
108394   diğer bakterilerin aksine, fotosentezde yapabilir.
108395   yani anlam üreticisinede daima bir şeylerin anlaşılır olması gerekir.
108396   sistem anlam üretmeninde anlamını üretir.
108397   onun anlam üretmesindeki ötekine referans kendisinede döner.
108398   kendisi için hem o şey hemde kendi olur.
108399   başarılı kariyer kadar başarısız kariyerde bir kimlik sağlar.
108400
```

Figure 5: Sentences which are labeled as ”Separated” and number of sentences for that class

2.3 Tokenizing

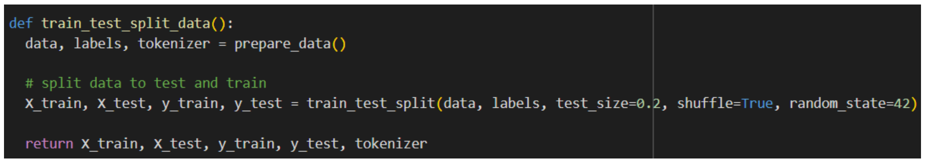
Tokenizer library is used for this. It converts text to sequence of integers. Unique integer is assigned to each word. Max length is given, it means maximum number of words in a sentence. Then each of the sentences are labeled.

After converting text to integer representation, and labeling them, they are separated as test and train data with shuffling. I used library for that process.



```
# tokenize the sentences
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)
sequences = tokenizer.texts_to_sequences(sentences)

# pad the sequences
data = pad_sequences(sequences, maxlen=max_length)

# convert labels to numpy arrays
labels = np.array([1]*len(separated) + [0]*len(not_separated))
```

Figure 6: Texts are converted to sequences, and labeled



```
def train_test_split_data():
  data, labels, tokenizer = prepare_data()

  # split data to test and train
  X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, shuffle=True, random_state=42)

  return X_train, X_test, y_train, y_test, tokenizer
```

Figure 7: Data is splitted to test, and train as (0.8, 0.2)

3 Model Details

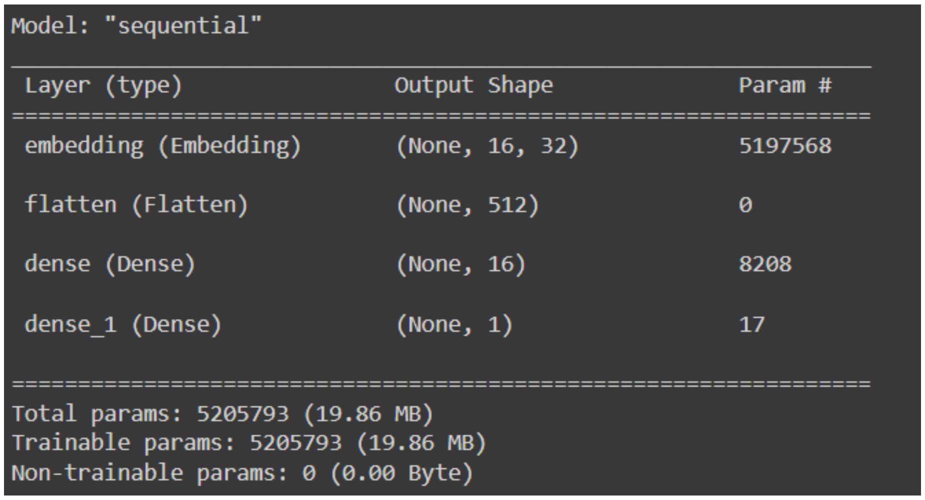
3.1 Layers

I've tried different configurations for the best results, and indicated model is the best design that gave better results between them.

First layer is the embedding layer, it takes output of the tokenizing process which are integer sequences. Then maps each integer to a dense vector.

Second layer is a hidden layer that converts output of the Embedding to 1D vector. It is needed for Dense layers. Vector size is determined as 32.

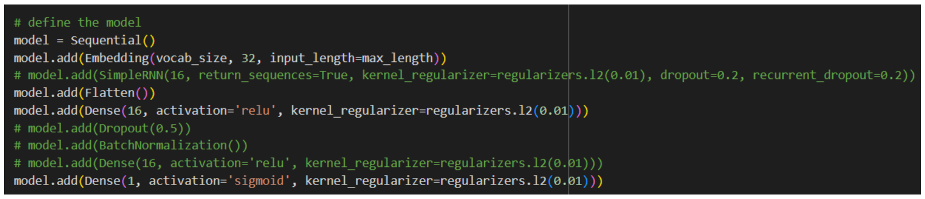
Third layer is a hidden layer with 16 units that is needed for non-linearity with different activation function from the final layer.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 16, 32)            5197568

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 16)                8208

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 5205793 (19.86 MB)
Trainable params: 5205793 (19.86 MB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 8: Layers of the Model

Output layer is a Dense layer with a Sigmoid Function in order to make binary classification.



```
# define the model
model = Sequential()
model.add(Embedding(vocab_size, 32, input_length=max_length))
# model.add(SimpleRNN(16, return_sequences=True, kernel_regularizer=regularizers.l2(0.01), dropout=0.2, recurrent_dropout=0.2))
model.add(Flatten())
model.add(Dense(16, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
# model.add(Dropout(0.5))
# model.add(BatchNormalization())
# model.add(Dense(16, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(1, activation='sigmoid', kernel_regularizer=regularizers.l2(0.01)))
```

Figure 9: Model Definition

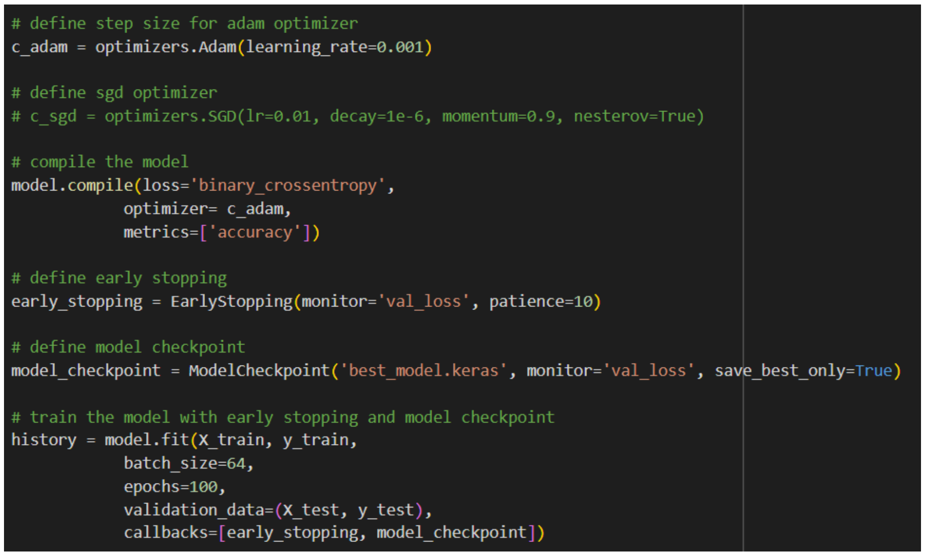
Commented Layers(Figure 9) are different configurations that are tried, best result I get between those is the uncommented configuration. Single hidden layer except flatten is enough for this problem to reach 90% accuracy which will be mentioned later. All values are determined according to their performance (embedding dimension, etc.).

Activation Functions: Sigmoid, and ReLU are used. Sigmoid is needed at the output layer, because aim is making binary classification and it generates result between 0 and 1. ReLU is used in hidden layer, because it gave far more better results than Sigmoid.

Optimizer: Adams Optimizer is used instead of SGD. It is faster, and works better for this model.

Loss Function: Binary Cross Entropy Loss is used, it is well suited for binary classification.

Early Stopping and Epoch: Early Stopping is used because, best value loss value must be saved, and its patience is 10. After some point it will give worse results due to overfit, therefore it must be stopped at some point to prevent **overfitting**. Total epoch limit is 100. Also, batch size is determined as 64.



```
# define step size for adam optimizer
c_adam = optimizers.Adam(learning_rate=0.001)

# define sgd optimizer
# c_sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

# compile the model
model.compile(loss='binary_crossentropy',
            optimizer= c_adam,
            metrics=['accuracy'])

# define early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10)

# define model checkpoint
model_checkpoint = ModelCheckpoint('best_model.keras', monitor='val_loss', save_best_only=True)

# train the model with early stopping and model checkpoint
history = model.fit(X_train, y_train,
            batch_size=64,
            epochs=100,
            validation_data=(X_test, y_test),
            callbacks=[early_stopping, model_checkpoint])
```

Figure 10: Parameters of the Model and Early Stopping

4 Model Performance

4.1 Measurement Method

Loss: It is a value to measure how well model is performing. **Binary Cross Entropy Loss** function is used for measuring this. Aim is minimizing that value for **training loss**. Then **value loss** is measured for every epoch, and model performance is determined according to that. If it is low, result is better. Aim is making value accuracy lower.

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}(y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i))$$

where:

N is the total number of instances in the dataset.
y_i is the true label of the i-th instance (0 or 1).
$\hat{y}_i$ is the predicted probability that the i-th instance belongs to the positive class (class 1).

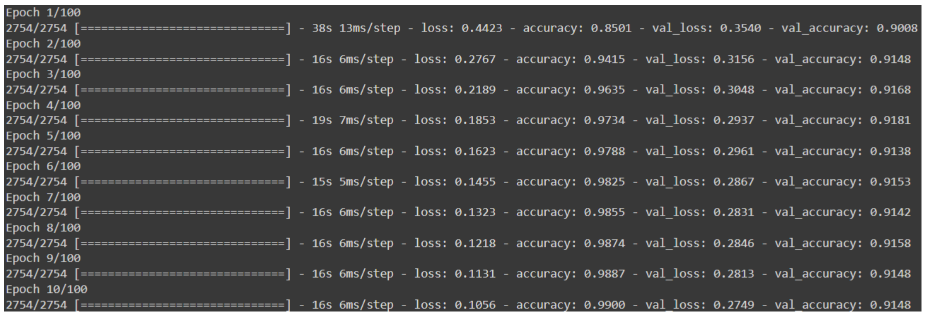
Accuracy: Accuracy is a method for measuring how accurate that data is classified. It makes this calculation based on labeled data, some of them are test and others are training. There are **Training** and **Value** accuracy. Aim is improving the test accuracy.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

4.2 Performance of the Model

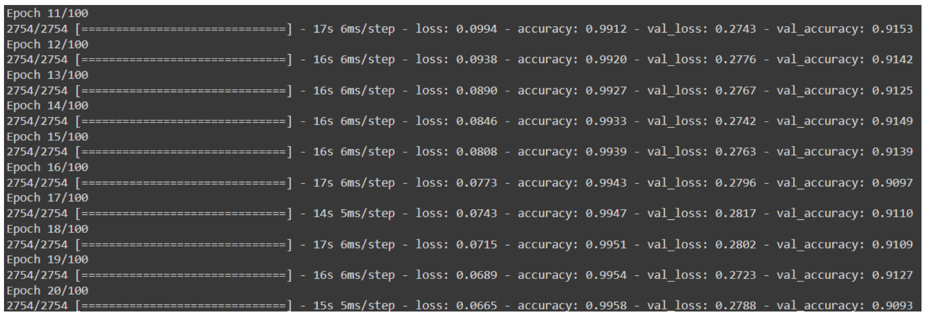
4.2.1 Each Epoch

Epoch is determined as 100 at first, but it uses early stopping with 10 patience. Therefore it can stop early.



```
Epoch 1/100
2754/2754 [==============================] - 38s 13ms/step - loss: 0.4423 - accuracy: 0.8501 - val_loss: 0.3540 - val_accuracy: 0.9008
Epoch 2/100
2754/2754 [==============================] - 16s 6ms/step - loss: 0.2767 - accuracy: 0.9415 - val_loss: 0.3156 - val_accuracy: 0.9148
Epoch 3/100
2754/2754 [==============================] - 16s 6ms/step - loss: 0.2189 - accuracy: 0.9635 - val_loss: 0.3048 - val_accuracy: 0.9168
Epoch 4/100
2754/2754 [==============================] - 19s 7ms/step - loss: 0.1853 - accuracy: 0.9734 - val_loss: 0.2937 - val_accuracy: 0.9181
Epoch 5/100
2754/2754 [==============================] - 16s 6ms/step - loss: 0.1623 - accuracy: 0.9788 - val_loss: 0.2961 - val_accuracy: 0.9138
Epoch 6/100
2754/2754 [==============================] - 15s 5ms/step - loss: 0.1455 - accuracy: 0.9825 - val_loss: 0.2867 - val_accuracy: 0.9153
Epoch 7/100
2754/2754 [==============================] - 16s 6ms/step - loss: 0.1323 - accuracy: 0.9855 - val_loss: 0.2831 - val_accuracy: 0.9142
Epoch 8/100
2754/2754 [==============================] - 16s 6ms/step - loss: 0.1218 - accuracy: 0.9874 - val_loss: 0.2846 - val_accuracy: 0.9158
Epoch 9/100
2754/2754 [==============================] - 16s 6ms/step - loss: 0.1131 - accuracy: 0.9887 - val_loss: 0.2813 - val_accuracy: 0.9148
Epoch 10/100
2754/2754 [==============================] - 16s 6ms/step - loss: 0.1056 - accuracy: 0.9900 - val_loss: 0.2749 - val_accuracy: 0.9148
```

Figure 11: Epoch 1 - 10

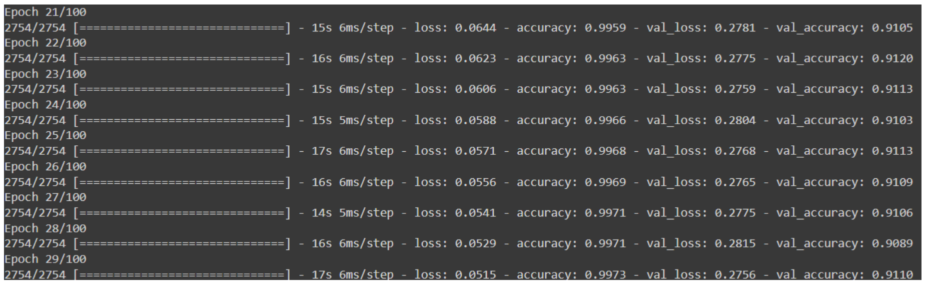


```
Epoch 11/100
2754/2754 [==============================] - 17s 6ms/step - loss: 0.0994 - accuracy: 0.9912 - val_loss: 0.2743 - val_accuracy: 0.9153
Epoch 12/100
2754/2754 [==============================] - 16s 6ms/step - loss: 0.0938 - accuracy: 0.9920 - val_loss: 0.2776 - val_accuracy: 0.9142
Epoch 13/100
2754/2754 [==============================] - 16s 6ms/step - loss: 0.0890 - accuracy: 0.9927 - val_loss: 0.2767 - val_accuracy: 0.9125
Epoch 14/100
2754/2754 [==============================] - 16s 6ms/step - loss: 0.0846 - accuracy: 0.9933 - val_loss: 0.2742 - val_accuracy: 0.9149
Epoch 15/100
2754/2754 [==============================] - 16s 6ms/step - loss: 0.0808 - accuracy: 0.9939 - val_loss: 0.2763 - val_accuracy: 0.9139
Epoch 16/100
2754/2754 [==============================] - 17s 6ms/step - loss: 0.0773 - accuracy: 0.9943 - val_loss: 0.2796 - val_accuracy: 0.9097
Epoch 17/100
2754/2754 [==============================] - 14s 5ms/step - loss: 0.0743 - accuracy: 0.9947 - val_loss: 0.2817 - val_accuracy: 0.9110
Epoch 18/100
2754/2754 [==============================] - 17s 6ms/step - loss: 0.0715 - accuracy: 0.9951 - val_loss: 0.2802 - val_accuracy: 0.9109
Epoch 19/100
2754/2754 [==============================] - 16s 6ms/step - loss: 0.0689 - accuracy: 0.9954 - val_loss: 0.2723 - val_accuracy: 0.9127
Epoch 20/100
2754/2754 [==============================] - 15s 5ms/step - loss: 0.0665 - accuracy: 0.9958 - val_loss: 0.2788 - val_accuracy: 0.9093
```

Figure 12: Epoch 11 - 20

Best loss is found here as 0.2723.(Figure 12)

Goes 9 steps more (Figure 13), but this value couldn't become better so, it is stopped due to early stopping. In order to prevent **overfitting**.



```
Epoch 21/100
2754/2754 [==============================] - 15s 6ms/step - loss: 0.0644 - accuracy: 0.9959 - val_loss: 0.2781 - val_accuracy: 0.9105
Epoch 22/100
2754/2754 [==============================] - 16s 6ms/step - loss: 0.0623 - accuracy: 0.9963 - val_loss: 0.2775 - val_accuracy: 0.9120
Epoch 23/100
2754/2754 [==============================] - 15s 6ms/step - loss: 0.0606 - accuracy: 0.9963 - val_loss: 0.2759 - val_accuracy: 0.9113
Epoch 24/100
2754/2754 [==============================] - 15s 5ms/step - loss: 0.0588 - accuracy: 0.9966 - val_loss: 0.2804 - val_accuracy: 0.9103
Epoch 25/100
2754/2754 [==============================] - 17s 6ms/step - loss: 0.0571 - accuracy: 0.9968 - val_loss: 0.2768 - val_accuracy: 0.9113
Epoch 26/100
2754/2754 [==============================] - 16s 6ms/step - loss: 0.0556 - accuracy: 0.9969 - val_loss: 0.2765 - val_accuracy: 0.9109
Epoch 27/100
2754/2754 [==============================] - 14s 5ms/step - loss: 0.0541 - accuracy: 0.9971 - val_loss: 0.2775 - val_accuracy: 0.9106
Epoch 28/100
2754/2754 [==============================] - 16s 6ms/step - loss: 0.0529 - accuracy: 0.9971 - val_loss: 0.2815 - val_accuracy: 0.9089
Epoch 29/100
2754/2754 [==============================] - 17s 6ms/step - loss: 0.0515 - accuracy: 0.9973 - val_loss: 0.2756 - val_accuracy: 0.9110
```

Figure 13: Epoch 21 - 29

4.2.2 Plots

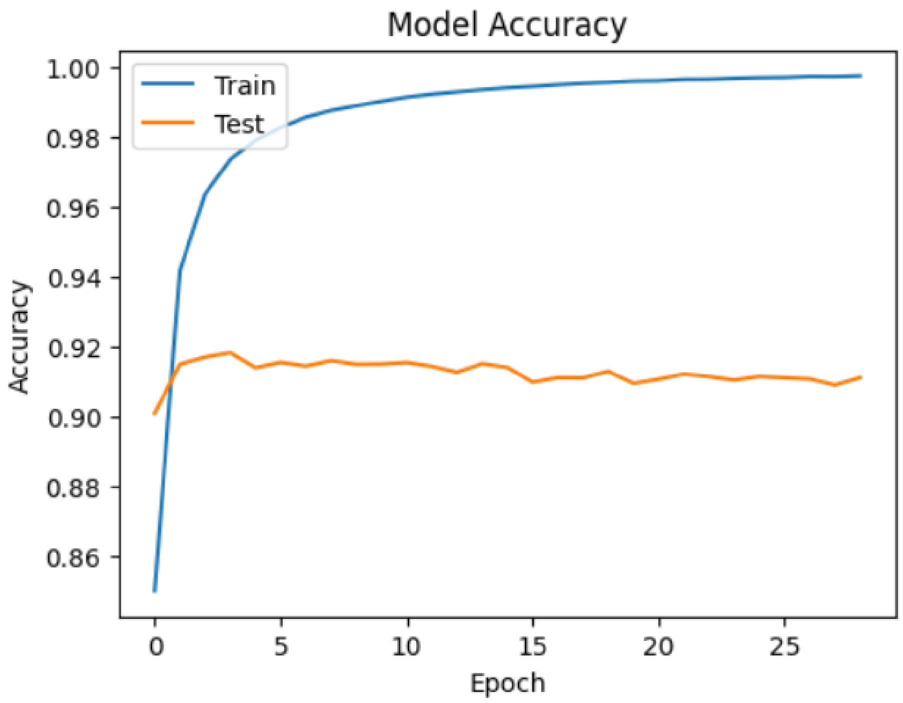


Figure 14: Training-Validation Accuracy

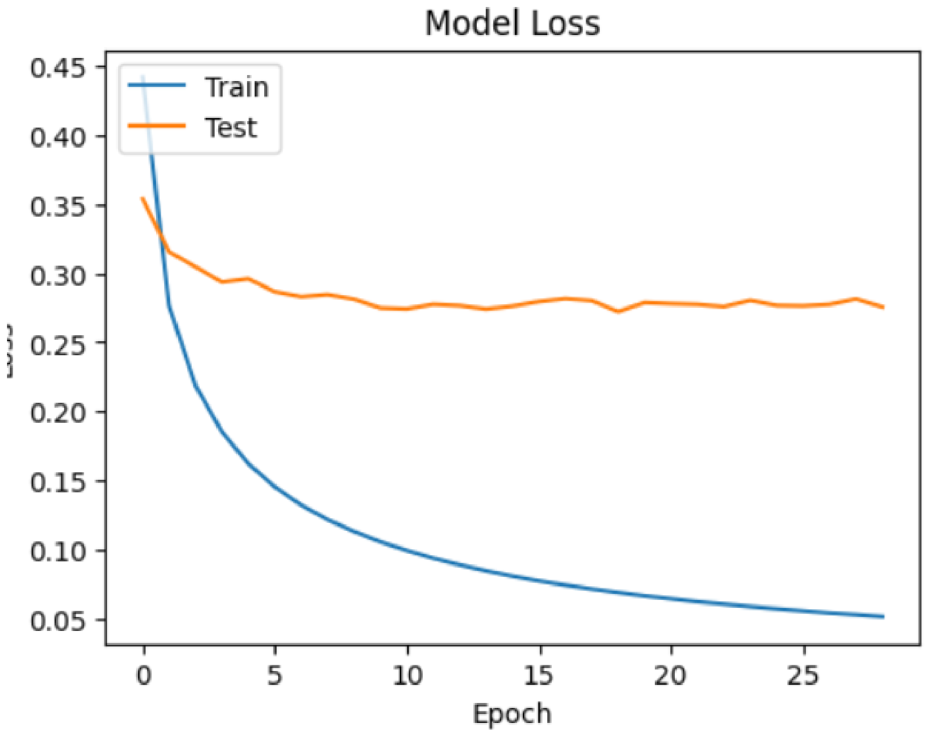
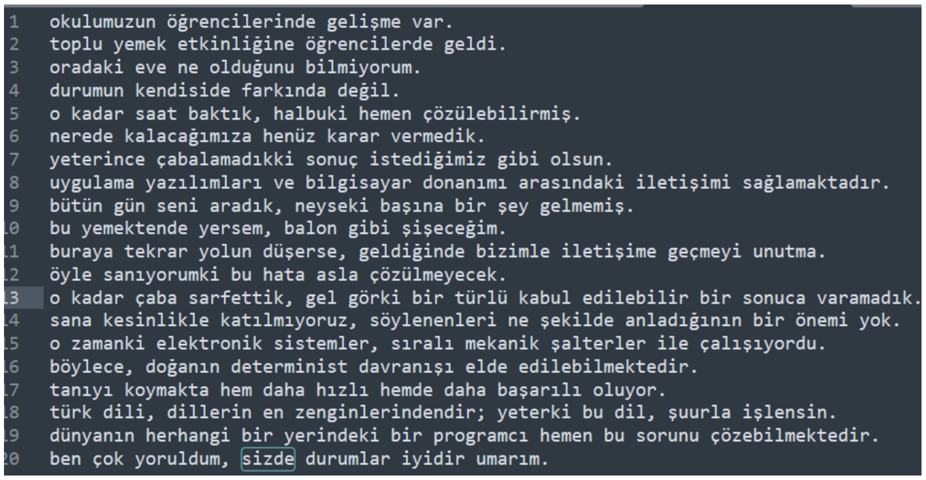


Figure 15: Training-Validation Loss

Accuracy: Validation accuracy is around 90%. Selected **validation accuracy is 0.9127**, training accuracy for that is 0.9954. It is picked according to validation loss. **Loss:** Validation loss is around 0.27-0.3, so the model that gives best validation loss is picked. After some point it can be seen that optimization doesn't affect the test performance. This model can be trained with a few epoch. Selected **validation loss is 0.2723**, training accuracy is 0.0689. Model stopped till there is no improvement on next 10 epoch.

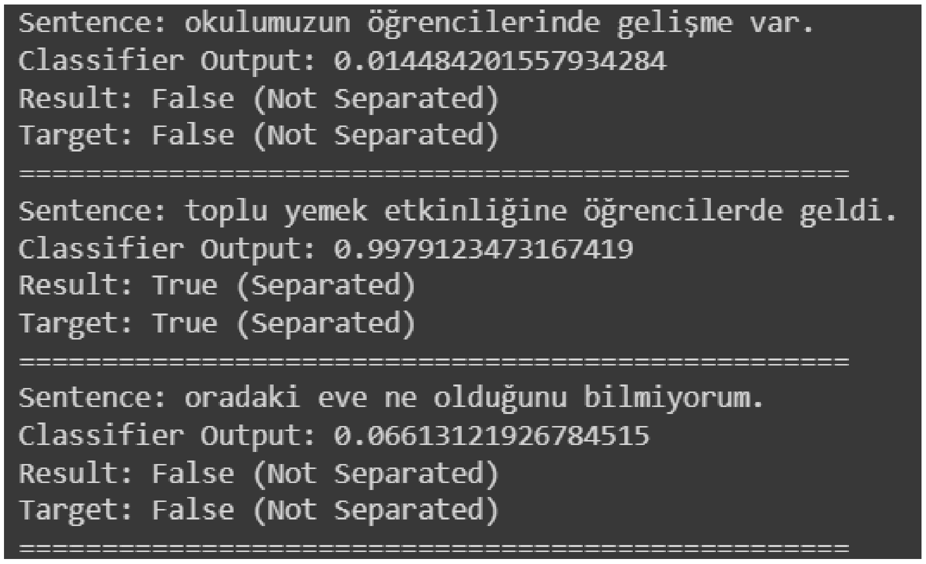
5 Example Sentences

Model is tested with 20 different sentences (Figure 16). Their classification value, and Separated, not separated info is also printed. As described in the homework instructions inputs are given as **not separated**. All of them are made correct classification.



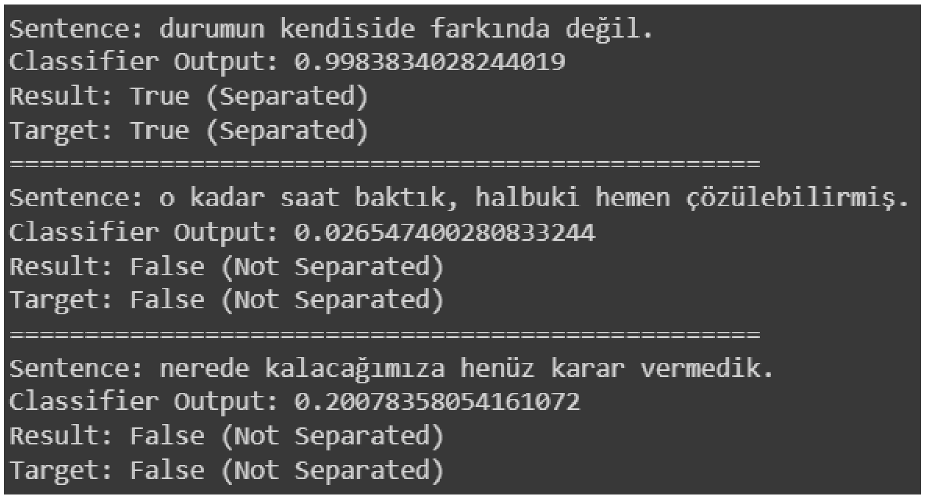
```
okulumuzun öğrencilerinde gelişme var.
toplu yemek etkinliğine öğrencilerde geldi.
oradaki eve ne olduğunu bilmiyorum.
durumun kendiside farkında değil.
o kadar saat baktık, halbuki hemen çözülebilirmiş.
nerede kalacağımıza henüz karar vermedik.
yeterince çabalamadıkki sonuç istediğimiz gibi olsun.
uygulama yazılımları ve bilgisayar donanımı arasındaki iletişimi sağlamaktadır.
bütün gün seni aradık, neyseki başına bir şey gelmemiş.
bu yemektende yersem, balon gibi şişeceğim.
buraya tekrar yolun düşerse, geldiğinde bizimle iletişime geçmeyi unutma.
öyle sanıyorumki bu hata asla çözülmeyecek.
o kadar çaba sarfettik, gel görki bir türlü kabul edilebilir bir sonuca varamadık.
sana kesinlikle katılmıyoruz, söylenenleri ne şekilde anladığının bir önemi yok.
o zamanki elektronik sistemler, sıralı mekanik şalterler ile çalışıyordu.
böylece, doğanın determinist davranışı elde edilebilmektedir.
tanıyı koymakta hem daha hızlı hemde daha başarılı oluyor.
türk dili, dillerin en zenginlerindendir; yeterki bu dil, şuurla işlensin.
dünyanın herhangi bir yerindeki bir programcı hemen bu sorunu çözebilmektedir.
ben çok yoruldum, sizde durumlar iyidir umarım.
```

Figure 16: Test Sentences



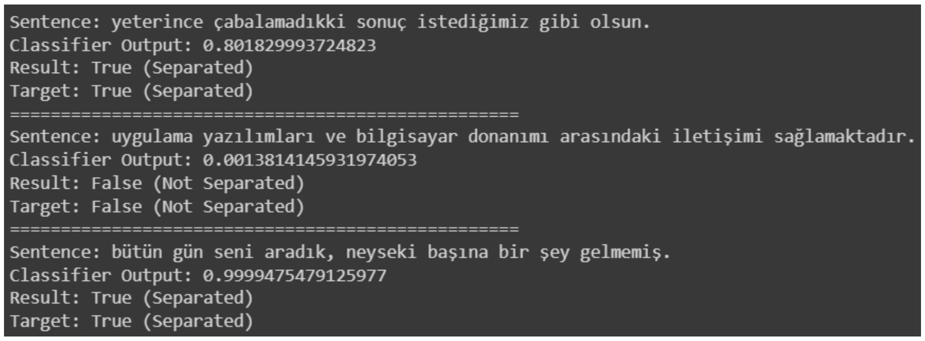
```
Sentence: okulumuzun öğrencilerinde gelişme var.
Classifier Output: 0.014484201557934284
Result: False (Not Separated)
Target: False (Not Separated)
==================================================
Sentence: toplu yemek etkinliğine öğrencilerde geldi.
Classifier Output: 0.9979123473167419
Result: True (Separated)
Target: True (Separated)
==================================================
Sentence: oradaki eve ne olduğunu bilmiyorum.
Classifier Output: 0.06613121926784515
Result: False (Not Separated)
Target: False (Not Separated)
==================================================
```

Figure 17: Result 1-3



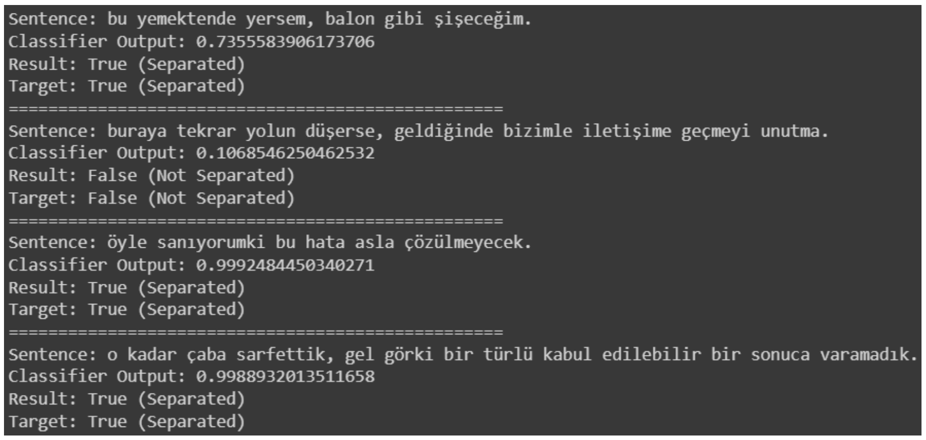
```
Sentence: durumun kendiside farkında değil.
Classifier Output: 0.9983834028244019
Result: True (Separated)
Target: True (Separated)
==================================================
Sentence: o kadar saat baktık, halbuki hemen çözülebilirmiş.
Classifier Output: 0.026547400280833244
Result: False (Not Separated)
Target: False (Not Separated)
==================================================
Sentence: nerede kalacağımıza henüz karar vermedik.
Classifier Output: 0.20078358054161072
Result: False (Not Separated)
Target: False (Not Separated)
```

Figure 18: Result 4-6



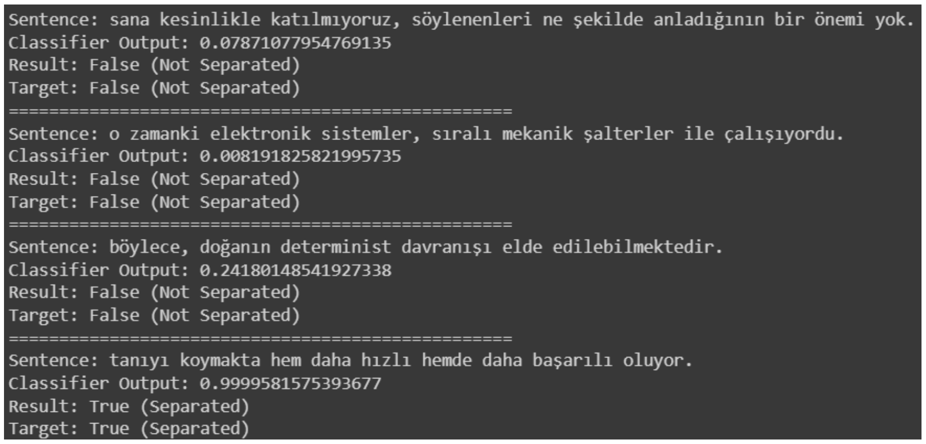
```
Sentence: yeterince çabalamadıkki sonuç istediğimiz gibi olsun.
Classifier Output: 0.801829993724823
Result: True (Separated)
Target: True (Separated)
==================================================
Sentence: uygulama yazılımları ve bilgisayar donanımı arasındaki iletişimi sağlamaktadır.
Classifier Output: 0.0013814145931974053
Result: False (Not Separated)
Target: False (Not Separated)
==================================================
Sentence: bütün gün seni aradık, neyseki başına bir şey gelmemiş.
Classifier Output: 0.9999475479125977
Result: True (Separated)
Target: True (Separated)
```

Figure 19: Result 7-9



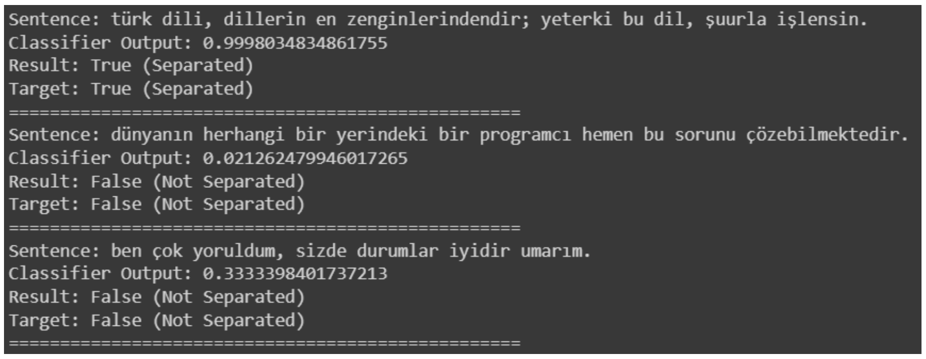
```
Sentence: bu yemektende yersem, balon gibi şişeceğim.
Classifier Output: 0.7355583906173706
Result: True (Separated)
Target: True (Separated)
==================================================
Sentence: buraya tekrar yolun düşerse, geldiğinde bizimle iletişime geçmeyi unutma.
Classifier Output: 0.1068546250462532
Result: False (Not Separated)
Target: False (Not Separated)
==================================================
Sentence: öyle sanıyorumki bu hata asla çözülmeyecek.
Classifier Output: 0.9992484450340271
Result: True (Separated)
Target: True (Separated)
==================================================
Sentence: o kadar çaba sarfettik, gel görki bir türlü kabul edilebilir bir sonuca varamadık.
Classifier Output: 0.9988932013511658
Result: True (Separated)
Target: True (Separated)
```

Figure 20: Result 10-13



```
Sentence: sana kesinlikle katılmıyoruz, söylenenleri ne şekilde anladığının bir önemi yok.
Classifier Output: 0.07871077954769135
Result: False (Not Separated)
Target: False (Not Separated)
==================================================
Sentence: o zamanki elektronik sistemler, sıralı mekanik şalterler ile çalışıyordu.
Classifier Output: 0.008191825821995735
Result: False (Not Separated)
Target: False (Not Separated)
==================================================
Sentence: böylece, doğanın determinist davranışı elde edilebilmektedir.
Classifier Output: 0.24180148541927338
Result: False (Not Separated)
Target: False (Not Separated)
==================================================
Sentence: tanıyı koymakta hem daha hızlı hemde daha başarılı oluyor.
Classifier Output: 0.9999581575393677
Result: True (Separated)
Target: True (Separated)
```

Figure 21: Result 14-17



```
Sentence: türk dili, dillerin en zenginlerindendir; yeterki bu dil, şuurla işlensin.
Classifier Output: 0.9998034834861755
Result: True (Separated)
Target: True (Separated)
==================================================
Sentence: dünyanın herhangi bir yerindeki bir programcı hemen bu sorunu çözebilmektedir.
Classifier Output: 0.021262479946017265
Result: False (Not Separated)
Target: False (Not Separated)
==================================================
Sentence: ben çok yoruldum, sizde durumlar iyidir umarım.
Classifier Output: 0.3333398401737213
Result: False (Not Separated)
Target: False (Not Separated)
==================================================
```

Figure 22: Result 18-20