5. IMPLEMENTASI SISTEM

Pada bab ini akan membahas implementasi dari desain sistem yang telah dibuat pada bab sebelumnya. Implementasi sistem akan menjelaskan tentang perangkat lunak yang akan digunakan seperti *library*, *syntax*, *cleaning & preprocessing data*, *training* dan *testing model* algoritma yang digunakan. Disertakan juga dengan penerapan pembuatan *website* yang digunakan untuk menerima dan memperlihatkan hasil penerjemahan bahasa.

Tabel 5.1

Implementasi Desain ke Dalam Segmen Program

Gambar Bab 3	**Segmentasi Program**	**Keterangan**
Gambar 3.5	4.43	Memasukkan *Model* Kedalam *Interface*
Gambar 3.1	4.1, 4.2, 4.3, 4.4, 4.5, 4.10, 4.11, 4.13, 4.14, 4.16, 4.17, 4.19, 4.22, 4.28, 4.33, 4.34, 4.36, 4.37, 4.39, 4.42	*Model*
Gambar 3.2	4.7, 4.8, 4.9	*Data Cleaning and preprocessing*
Gambar 3.3	4.12	*Training Model Bi-LSTM*
Gambar 3.4	4.15, 4.18, 4.20, 4.38, 4.40, 4.41	*Example Translation*

4.1. Implementasi Perangkat Lunak yang Digunakan

Implementasi *deep learning* pada skripsi ini adalah berupa *website*. *Website* dibuat dengan Bahasa Pemrograman *Python* versi 3.9.13 dengan bantuan Gradio yang berfungsi sebagai *Interface*. Selain pada pembuatan *Interface*, *Python* juga digunakan dalam pengolahan data dan pembuatan derta pelatian model algoritma.

4.2. Model

4.2.1. Model Bi-LSTM dan IndoBERT

4.2.1.1. *Library* yang Digunakan

Pada skrpsi ini, menggunakna *Google Collaboration* dan *Visual Code Studio* sebagai sarana yang digunakan untuk mengedit teks pemrograman *python*, melakukan *training* dan *testing* untuk model algoritma yang digunakan, serta untuk instalasi *library*. Pada Segmen Program ini berfungsi untuk mengintalasi *library* yang digunakan dalam skripsi ini. Berikut beberapa fungsi dari Library yang digunakan:

Segmen Program 5.1

Instalasi Library *Pada* Model

```
from transformers import TFAutoModelForSequenceClassification, AutoTokenizer, TFT5ForConditionalGeneration
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences
import nltk
import pandas as pd
import string
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
```

- *Import* beberapa *library Transformer* seperti *TFAutoModelForSequenceClassification*, *AutoTokenizer*, dan *TFT5ForConditionalGeneration*. *TFAutoModelForSequenceClassification* nantinya akan digunakan untuk mengambil model *Transformer* yang sebelumnya sudah di-*train* untuk tugas klasifikasi urutan. Nantinya ini akan berfungsi untuk mempermudah penggunaan *model Transformer* untuk klasifikasi teks dengan *Tensorflow*. *AutoTokenizer* akan digunakan untuk mengambil *Tokenizer* yang

sesuai dengan *model Transformer* yang dipilih, nantinya *AutoTokenizer* akan memberikan *Tokenizer* yang sesuai dengan *model Tansformer* untuk *preprocessing* teks. *TFT5ForConditionalGeneration* akan digunakan untuk mengambil *model* T5 yang telah di-*Train* sebelumnya untuk tugas generasi teks kondisional.

- *Import library Tensorflow* untuk pembelajaran mesin *Deep Learing*. dimana, merupakan *framework* untuk membangun dan melatih algoritma.
- *Import library sklearn.preprocessing.LabelEncoder* akan digunakan untuk merubah label menjadi angka.
- *Import library Numpy* yang adalah *library* fundamental untuk komputasi ilmiah dengan *python*. Library ini menyediakan berbagai array multidimensi dan berbagai fungsi matematika yang dapat digunakan.
- *Import library* nltk atau *Natural Language Toolkit* yang mana merupakan *library* yang bekerja dengan data teks dan menyediakan alat dan sumber daya untuk pemrosesan Bahasa alami.
- *Import library Pandas* diguankan untuk manipulasi dan analisis data.
- *Import library string* yang mneyediakan konstanta dan kelas untuk *string*.
- *Import sklearn.model_selection.train_test_split* digunakan untuk membagi dataset menjadi subset *test* dan *train*.
- *Import Tokenizer* yang diambil dari *library* keras utnuk tokenizer teks.

4.2.1.2. Memuat *Dataset* dengan Pandas

Segmen Program 5.2

Load kedua *Dataset* dengan menggunakan *Pandas*

```
df_word = pd.read_excel('DATASET TERM.xlsx')
```

Metode *read_excel* dari *Library pandas* digunakan untuk mengambil *file excel* yang akan dimasukkan kedalam *DataFrame* agar *dataset* dapat digunakan untuk *training* dan *testing model*.

4.2.1.3. Menetapkan Kolom *Text* dan *Label*

Segmen Program 5.3

Menetapkan Kolom *Text* dan *Label*

```
texts_words = df_word['BALI']
labels_words = df_word['INDONESIA']
```

Pada skripsi ini label dan text di tetapkan secara manual dimana nantinya code akan membaca kolom dengan tulisan "BALI" pada bari pertama *file Excel* yang menandakan kolom tersebut adalah kolom *text*. Sedangkan kolom yang memiliki tulisan "INDONESIA" akan menandakan bahwa kolom tersebut adalah kolom *Label*.

4.2.1.4. Menghilangkan Tanda Baca dan *Preprocessing* Pada *Dataset*

Segmen Program 5.4

Membuat Fungsi Untuk Menghilangkan Tanda Baca dan *Preprocessing* Pada *Dataset*

```
def remove_punctuation(text):
translator = str.maketrans('', '', string.punctuation)
return text.translate(translator)
def preprocess_text(text):
    return text.str.lower().apply(remove_punctuation)

texts_words = preprocess_text(df_word['BALI'])
labels_words = preprocess_text(df_word['INDONESIA'])
```

Dengan membuat fungsi *remove_punctuation* akan menghapus tanda baca yang ada didalam dataset. Hal ini akan membantu pengolahan *dataset* menjadi lebih akurat. Natinya akan dipanggil pada fungsi *Preprocess_text* yang sudah ditambahkan *function .lower* untuk membuat teks menjadi huruf kecil. Sehingga untuk label dan text hanya perlu memanggil 1 fungsi saja.

4.2.1.5. Tokenizer

Segmen Program 5.5

Menggunakan LabelEncoder() kepada *label*

```
label_encoder_words = LabelEncoder()
labels_encoded_words                                    =
label_encoder_words.fit_transform(labels_words)


vocab_size = 20000
max_length = 128
embedding_dim = 200
```

LabelEncoder() membantu mengubah data menjadi numerik agar dapat digunakan untuk melatih algoritma. Pada bagian ini juga ditentukan besar *vocab_size*, *max_length*, dan juga *embedding_dim*.

Segmen Program 5.6

Memanggil *function* pada l*ibrary NLTK*

```
def tokenize_words(text):
    return nltk.word_tokenize(text)
```

Pada Segmen Prgoram 4.6 diatas menunjukkan pemanggilan function *word_tokenize* yang trdapat pada *library* NLTK.

Segmen Program 5.7

Membuat *Vocabulary*

```
all_words = [word for text in texts_words for word in
tokenize_words(text)]
unique_words = list(set(all_words))
word_index = {word: i+1 for i, word in
enumerate(unique_words)}
```

Pada bagian ini yang pertama kali dilakukan adalah mengumpulaknn seluruh kata yang terdapat pada text dalam dataset. Kemudian, *unique_word* membuat sekumpulan kata unik yang idambil dari *all_words*. Dan, *word_index* digunakan untuk membuat pengelompokkan kata unik menjadi index integer.

Segmen Program 5.8

Mengubah Teks Menjadi *Sequences*

```
def texts_to_sequences(texts, word_index):
    return [[word_index.get(word, 0) for word in
tokenize_words(text)] for text in texts]

def batch_texts_to_sequences(texts, word_index,
max_length):
```

```
    tokenized_texts = texts_to_sequences(texts,
word_index)
    return pad_sequences(tokenized_texts,
maxlen=max_length, padding='post')
```

Pada Segmen Program 4.8 terdapat *function 'text_to_sequences'* untuk mengubah setiap kata dalam *text* menjadi index yang sesuai dengan menggunakan *word_index* yang sudah dibuat. Nantinya apabila kata tidak ditemukan maka akan diberinkan nilai '0'. Terdapat juga *function 'batch_texts_to_sequences'* digunakan untuk menerapkan *'text_to_sequences'* ke setiap teks dan melakukakn *padding* pada setiap urutan untuk memastikan bahwa semua urutan memiliki panjang yang sama.

Segmen Program 5.9

Melakukan *Tokenizer* dan *Padding*

```
tokenized_texts_words =
batch_texts_to_sequences(texts_words, word_index,
max_length)
```

Pada Segmen Program 4.9 dilakukan Tokenizer dan padding terhadap kolom teks dengan memanggil *'batch_texts_to_sequences'*.

4.2.1.6. Split Data

Segmen Program 5.10

Split Data menjadi *Train* dan *Test* Untuk Teks dan Label

```
# Split the word data
train_texts, test_texts, train_labels, test_labels =
train_test_split(tokenized_texts_words,
labels_encoded_words, test_size=0.2, random_state=42)
```

Split Data dilakukan dengan mengambil fungsi dari *library scikit-learn 'train_test_split'*. Dimana fungsi ini berguna untuk membagi *dataset* menjadi data *train* dan *test*. Yang mana dalam Segmen Program 4.8 data dibagi menjadil *train_texts, test_texts, train_labels, test_labels*. Kemudian, *'tokenized_texts_words'* berisi teks yang akan telah diterjemahkan ke dalam bentuk ID-token. Kemudian, *'labels_encoded_words'* dalam hal ini merupakan label yang berhubungan dengan teks dalam *'tokenized_texts_words'*, dapat dilihat pada Segmen Program 4.5. Kemudian,

'test_size' adalah parameter untuk menentukan pembagian data test, dimana pada bagian ini telah diatur sebanyak 0.2. Artinya, 20% dari data merupakan data *test* dan 80% nya adalah data *train*. Dan *'random_state=42'* adalah parameter yang menentukan *random data*. Dengan memberikan nilai yang sama memastikan bahwa *code* dijalankan, pembagian data menjadi subset *train* dan *test* akan memiliki nilai yang konsisten. Hal ini karena mereka menggunakan nilai *random state* yang sama.

4.2.1.7. Implementasi Bi-LSTM

Segmen Program 5.11

Model Bi-LSTM

```
bi_lstm_model = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=vocab_size+1,
output_dim=embedding_dim, input_length=max_length),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128,
return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,
return_sequences=True)),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(vocab_size+1,
activation='softmax')
])
```

Pada model diatas, Lapisan *Embedding* digunakan untuk mengubah ID-token menjadi *vector embedding*. Dimana pada *code* ini jumlah kata dalam vocabulary sebagai dimensi input dan setiap ID-token akan diubah menjadi vektor dengan panjang 128. Kemudian, *layer Bidirectional* digunakan agar model bisa melihat konteks secara 2 arah. Hal ini dikarenakan LSTM yang bersifat *undirectional* hanya meilhat konteks dari kiri kekanan maupun sebaliknya, sehingga dengan menggunakan *layer* ini dapat membantu LSTM lebih mengenali konteks dari berbagai arah. *Dropout* pada lapisan ini digunakna untuk mengurangi *overfitting* pada model dengan mematikan sebagian unit selama *training*. *Layer Global Average Pooling* digunakan untuk menghitung rata-rata dari semua *vector output* LSTM dalam dimensi waktu, yang nantikan akan menghasilkan vector dengan panjang yang tetap untuk setiap sampelnya. Kemudian, *layer Dropout* digunakan

untuk membantu mencegah *overfitting*. Dimana *layer* ini bekerja dengan tingkat *dropout* sebesar 0,5. Yang mana artinya setiap model dilatih maka, setiap unit lapisan ini akan matikan secara acak dengan probabilitas 0,5. Dan layer yang terakhir adalah *layer dense* dimana layer ini berguna untuk melakukan pemetaan dari fitur-fitur yang dihasilkan oleh lapisan sebelumnya kedalam ruang dimensi yang lebih rendah.

Segmen Program 5.12

Compile Model

```
# Compile the model
bi_lstm_model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
```

Pada Segmen Program 4.12, variable *bi_lstm_model* yang berisikan model yang telah dibangun sebelumnya. Pada *'.compile'* digunakan untuk mengonfigurasi model agar siap untuk dilatih. Metode ini meiliki parameter berupa *'loss_function'*, '*optimizer'*, dan juga metrik yang akan digunakan selaha pelatihan. Pada *loss* yang menggunakan *sparse_categorical_crossentropy* akan digunakan untuk masalah klasifikasi yang dimana label telah di *encoded* sebelumnya. Kemudian optimizer akan menentukan algoritma yang digunakan untuk memperbaharui bobot model selama waktu pelatihan. Pada kasus ini *optimizer* menggunakan *Adam*. *Adam* sendiri merupakan salah satu *optimizer* yang terkenal dengan efisiensinya dan kemampuannya menangani *sparse gradient* pada masalah *machine learning*. 'metrics=['accuracy']' adalah metrik yang sering digunakan dalam masalah klasifikasi, yang mana menunjukkan persentase yang benar dari total prediksi yang dibuat model.

Segmen Program 5.13

Train Model Bi-LSTM yang Telah Dibuat

```
num_epochs = 10
early_stopping =
tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=3, restore_best_weights=True)

# Call fit directly outside of any tf.function
bi_lstm_model.fit(train_texts,
```

```
                    train_labels,
                    epochs=num_epochs,
                    validation_data=(test_texts,
test_labels),
                    callbacks=[early_stopping],
                    verbose=2)
```

Pada Segmen Program 4.13 dapat dilihat penentuan jumlah *epoch* yang akna dilakukan. Kemudian membuat *early_stopping* menggunakan *library* dari keras . kemudian dilakukan pemanggilan metode '*.fit*' sebagai langkah awal melatih kode. Beberapa parameter yang dimiliki oleh metode ini adalah *'train_texts'*, *'train_labels'*, '*epoch'*, '*validation_data'*, '*callbacks'*, dan '*verbose'*. Parameter-parameter tersebut yang nantinya akan melakukan proses pelatihan terhadap model yang telah dibuat.

Segmen Program 5.14

Save Model Bi-LSTM

```
bi_lstm_model.save('bi_lstm_model_biLSTM.h5')
```

Pada Segmen Program 4.14 dilakukan *save model* terhadap algoritma yang terlah di latih agar model dapat digunakan dan dimasukkan kedalam *interface*.

4.2.1.8. Membuat *Function batch_predict_translation*

Segmen Program 5.15

Membuat *Function batch_predict_translation*

```
def batch_predict_translation(words, model, word_index,
label_encoder):
tokenized_texts = batch_texts_to_sequences(words,
word_index, max_length)
predictions = model.predict(tokenized_texts, verbose=0)
predicted_label_ids = np.argmax(predictions, axis=1)
predicted_labels =
label_encoder.inverse_transform(predicted_label_ids)
return predicted_labels
```

```
loaded_bi_lstm_model =
tf.keras.models.load_model('bi_lstm_model.h5')
```

Pada Segmen Program 4.15 *function* digunakan untuk memprediksi terjemahan kata-kata individu menggunakan model Bi-LSTM yang telah dilatih sebelumnya. Fungsi ini memiliki parameter '*words*', '*model*', '*word_index*', dan '*label_encoder*'. Langkah pertama adalah membuat tokenisasi dengan menggunakna fungsi '*batch_texts_to_sequences*'. Kemudian teks yang telah di tokenisasi akan di prediksi dengan menggunakan model yang telah disimpan. Kemudian, indeks tertinggi dari setiap *output* model akan diambil. Indeks ini nantinya akna menunjukkan label yang paling memungkinkan untuk setiap kata. kemudian, indeks prediksi akan diubah mnejadi bentuk aslinya dengan menggunakan *label_encoder.inverse_transform*. Kemudian hasil akan di *return*. kemudian *model* akan di-*load* untuk digunakan saat melakukan prediksi.

4.2.1.9. Inisialisasi Model T5

Segmen Program 5.16

Inisialisasi Model T5

```
t5_model =
TFT5ForConditionalGeneration.from_pretrained("t5-small")
tokenizer_t5 = AutoTokenizer.from_pretrained("t5-small")
```

Pada Segmen Program 4.16 dilakukan inisialisasi model T5 dengan menggunakan *TFT5ForConditionalGeneration.from_pretrained("t5-small")*. Dimana model ini memuat model T5 yang sudah dilatih sebelumnya dengan ukuran *'small'* dari repositori *Hugging Face*. Kemudian dilakukan *tokenizer* dengan menggunakan *AutoTokenizer.from_pretrained("t5-small")*.

4.2.1.10. Inisialisasi *Tokenizer* IndoBERT

Segmen Program 5.17

Inisialisasi *Tokenizer* IndoBERT

```
tokenizer_indobert =
AutoTokenizer.from_pretrained("indobenchmark/indobert-
base-p1")
```

Pada Segmen Program 4.17 dilakukan inisialisasi *tokenizer* dengan menggunakan *AutoTokenizer.from_pretrained("indobenchmark/indobert-base-p1"). Tokenizer* akan disesuaikan dengan model IndoBERT varian '*base*' versi p1 dari dataset IndoBERT yang disebut *indobenchmark*.

4.2.1.11. Membuat Fungsi Refine_Sentence

Segmen Program 5.18

Membuat Fungsi *Refine Sentence*

```
def refine_sentence(sentence, model, tokenizer):
inputs = tokenizer(sentence, return_tensors="tf",
padding=True, truncation=True, max_length=128)
outputs = model.generate(inputs['input_ids'])
refined_sentence = tokenizer.decode(outputs[0],
skip_special_tokens=True)
return refined_sentence
```

Fungsi *'refine_sentence'* berfungsi untuk memperbaiki atau menghasilkan ulang sebuah kalimat yang telah dilatih sebelumnya dan tokenizer yang sesuai. Para meter untuk fungsi ini adalah *'sentence'*, *'model'*, *'tokenizer'*. Langkah pertama yang dilakukan adalah kalimat *input* akan diproses oleh *tokenizer* menjadi representasi numerik yang sesuai dengan token-token yang dikenali oleh *model*. Kemudian *'outputs'* berisikan metode *generates* yang digunakan untuk menghasilkan teks baru berdasarkan representasi input numerk. Representasi numerik dari kalimat yang telah diperoleh dari *tokenizer* akan dimasukkan kedalam model untuk menghasilkan kalimat. Kemudian, *refined_sentence* digunakan untuk men-*decode ouput* menjadi teks kalimat yang benar. *Return refined_sentense* akan mengeluarkan hasil akhir.

4.2.1.12. Menerjemahkan Bahasa Bali ke Indonesia

Segmen Program 5.19

Menerima *Input* dan di *Tokenizer*

```
example_text = input("Enter the text to translate: ")

example_words =
tokenizer_indobert.tokenize(example_text.lower())
```

Pada Segmen Program 4.19 menunjukkan bahwa *example_text* menerima *input* yang kemudian akan dimasukan kedalam *example_words* untuk di *tokenize*. *Tokenize* akan dilakukan dengan *tokenizer IndoBERT.* Hal ini bertujuan untuk membagi kalimat menjadi token-token yang dapat dimengerti oleh model.

Segmen Program 5.20

Prediksi dan Penggabungan Kata

```
translations = batch_predict_translation(example_words,
loaded_bi_lstm_model, word_index, label_encoder_words)

translated_sentence = ' '.join(translations)
```

Pada '*Translation*' dilakukan prediksi terjemahan untuk setiap kata dalam teks input. Fungsi *batch_predict_translation* digunakan untuk memprediksi terjemahan setiap kata. Nantinya hasil prediksi akan disimpan kedalam *Translation*. Kemudian, *'translated_sentence'* digunakan untuk menggabungkan kata-kata yang telah diterjemahkan oleh *translation*.

Segmen Program 5.21

Memperbaiki dan Menampilkan Hasil Terjemahan

```
refined_translation =
refine_sentence(translated_sentence, t5_model,
tokenizer_t5)
print("Translation:", refined_translation)
```

Pada Segmen Program 4.21 Kalimat yang telah diterjemahkan akan dimasukkan kedalam model T5 untuk memprebaiki hasil terjemahan. Hasil dari terjemahan kemudian akan *di print*.

4.2.1.13. Evaluasi *BLEU*

Segmen Program 5.22

Evaluasi *BLEU*

```
reference_translation = input("Enter the reference
translation: ")

reference_words =
tokenize_words(reference_translation.lower())

bleu_score = sentence_bleu([reference_words],
translations,
smoothing_function=SmoothingFunction().method1)
```

```
print("BLEU score:", bleu_score)
```

Pada Segmen Program 4.22 Langkah pertama yang dilakukan adalah memasukkan input terjemahan yang benar dari penerjemahan. Kemudian terjemahan yang dimasukkan, akan ditokenisasi dengan fungsi *tonenize_words*. Kemudian, *bleu_score* menghitung hasil terjemahan yang dilakukan oleh *translation* dengan terjemahan referensi.

4.2.2. Model Bi-LSTM

4.2.2.1. Import Library

Segmen Program 5.23

Import Library Yang Digunakan

```
import numpy as np
import pandas as pd
import string
import nltk
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.sequence import
pad_sequences
import tensorflow as tf
from nltk.translate.bleu_score import sentence_bleu,
SmoothingFunction
```

4.2.2.2. Mengatur *Level Logging* Pada *Tensorflow*

Segmen Program 5.24

Mengatur *Level Logging* Pada *Tensorflow*

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
tf.get_logger().setLevel('ERROR')
```

Pada Segmen Program 4.24, pada baris pertama memanggil *import os* yang menyediakan fungsi untuk pengaturan variabel. Kemudian pada bari kedua dilakukan pengaturan pada variabel lingkungan *TF_CPP_MIN_LOG_LEVEL* ke nilai 3. Dnegan mengatur ke nilai 3 maka pesan log hanya menampilkan log tingkat *error*. Kemudian pada bari ketiga dilakukan pengaturan *level logging*

melalui *API Tensirflow*. Dimana pada baris terkahir menggunakan *API Tensorflow* untung mengatur *level* secara langsung. *tf.get_logger()* digunakna untuk mendapatkan *logger default* dari *Tensorflow*. *setLevel('ERROR')* digunakan untuk mengatur *level logging 'error'*. Sehingga dengan mengatur *level logging* bisa mnegurangi *noise* dari *log Tensorflow*.

4.2.2.3. *Load Dataset* Dengan Pandas

Segmen Program 5.25

Load Dataset Term Dengan Pandas

```
df_word = pd.read_excel('DATASET TERM.xlsx')
```

Metode *read_excel* dari *Library pandas* digunakan untuk mengambil *file excel* yang akan dimasukkan kedalam *DataFrame* agar *dataset* dapat digunakan untuk *training* dan *testing model*.

4.2.2.4. Menetapkan Kolom *Text* dan *Label*

Segmen Program 5.26

Menetapkan Kolom *Teks* dan *Label*

```
texts_words = df_word['BALI']
labels_words = df_word['INDONESIA']
```

Pada skripsi ini label dan text di tetapkan secara manual dimana nantinya code akan membaca kolom dengan tulisan "BALI" pada bari pertama *file Excel* yang menandakan kolom tersebut adalah kolom *text*. Sedangkan kolom yang memiliki tulisan "INDONESIA" akan menandakan bahwa kolom tersebut adalah kolom *Label*.

4.2.2.5. Menghilangkan Tanda Baca dan *Preprocessing* Pada *Dataset*

Segmen Program 5.27

Membuat Fungsi Untuk Menghilangkan Tanda Baca dan *Preprocessing* Pada *Dataset*

```
def remove_punctuation(text):
translator = str.maketrans('', '', string.punctuation)
return text.translate(translator)
def preprocess_text(text):
    return text.str.lower().apply(remove_punctuation)

texts_words = preprocess_text(df_word['BALI'])
labels_words = preprocess_text(df_word['INDONESIA'])
```

Dengan membuat fungsi *remove_punctuation* akan menghapus tanda baca yang ada didalam dataset. Hal ini akan membantu pengolahan *dataset* menjadi lebih akurat. Natinya akan

dipanggil pada fungsi *Preprocess_text* yang sudah ditambahkan *function .lower* untuk membuat teks menjadi huruf kecil. Sehingga untuk label dan text hanya perlu memanggil 1 fungsi saja.

4.2.2.6. Tokenizer

Segmen Program 5.28

Menggunakan LabelEncoder() kepada *label*

```
label_encoder_words = LabelEncoder()
labels_encoded_words                                    =
label_encoder_words.fit_transform(labels_words)


vocab_size = 20000
max_length = 128
embedding_dim = 200
```

LabelEncoder() membantu mengubah data menjadi numerik agar dapat digunakan untuk melatih algoritma. Pada bagian ini juga ditentukan besar *vocab_size*, *max_length*, dan juga *embedding_dim*.

Segmen Program 5.29

Memanggil *function* pada l*ibrary NLTK*

```
def tokenize_words(text):
    return nltk.word_tokenize(text)
```

Pada Segmen Prgoram 4.29 diatas menunjukkan pemanggilan function *word_tokenize* yang trdapat pada *library* NLTK.

Segmen Program 5.30

Membuat *Vocabulary*

```
all_words = [word for text in texts_words for word in
tokenize_words(text)]
unique_words = list(set(all_words))
word_index = {word: i+1 for i, word in
enumerate(unique_words)}
```

Pada bagian ini yang pertama kali dilakukan adalah mengumpulaknn seluruh kata yang terdapat pada text dalam dataset. Kemudian, *unique_word* membuat sekumpulan kata unik yang idambil dari *all_words*. Dan, *word_index* digunakan untuk membuat pengelompokkan kata unik menjadi index integer.

Segmen Program 5.31

Mengubah Teks Menjadi *Sequences*

```
def texts_to_sequences(texts, word_index):
    return [[word_index.get(word, 0) for word in
tokenize_words(text)] for text in texts]

def batch_texts_to_sequences(texts, word_index,
max_length):
    tokenized_texts = texts_to_sequences(texts,
word_index)
    return pad_sequences(tokenized_texts,
maxlen=max_length, padding='post')
```

Pada Segmen Program 4.31 terdapat *function 'text_to_sequences'* untuk mengubah setiap kata dalam *text* menjadi index yang sesuai dengan menggunakan *word_index* yang sudah dibuat. Nantinya apabila kata tidak ditemukan maka akan diberinkan nilai '0'. Terdapat juga *function 'batch_texts_to_sequences'* digunakan untuk menerapkan *'text_to_sequences'* ke setiap teks dan melakukakn *padding* pada setiap urutan untuk memastikan bahwa semua urutan memiliki panjang yang sama.

Segmen Program 5.32

Melakukan *Tokenizer* dan *Padding*

```
tokenized_texts_words =
batch_texts_to_sequences(texts_words, word_index,
max_length)
```

Pada Segmen Program 4.32 dilakukan Tokenizer dan padding terhadap kolom teks dengan memanggil *'batch_texts_to_sequences'*.

4.2.2.7. Split Data

Segmen Program 5.33

Split Data menjadi *Train* dan *Test* Untuk Teks dan Label

```
# Split the word data
train_texts, test_texts, train_labels, test_labels =
train_test_split(tokenized_texts_words,
labels_encoded_words, test_size=0.2, random_state=42)
```

Split Data dilakukan dengan mengambil fungsi dari *library scikit-learn 'train_test_split'*. Dimana fungsi ini berguna untuk membagi *dataset* menjadi data *train* dan *test*. Yang mana dalam Segmen Program 4.33 data dibagi menjadil *train_texts, test_texts, train_labels, test_labels*. Kemudian, *'tokenized_texts_words'* berisi teks yang akan telah diterjemahkan ke dalam bentuk ID-token. Kemudian, *'labels_encoded_words'* dalam hal ini merupakan label yang berhubungan dengan teks dalam *'tokenized_texts_words'*, dapat dilihat pada Segmen Program 4.28. Kemudian, *'test_size'* adalah parameter untuk menentukan pembagian data test, dimana pada bagian ini telah diatur sebanyak 0.2. Artinya, 20% dari data merupakan data *test* dan 80% nya adalah data *train*. Dan *'random_state=42'* adalah parameter yang menentukan *random data*. Dengan memberikan nilai yang sama memastikan bahwa *code* dijalankan, pembagian data menjadi subset *train* dan *test* akan memiliki nilai yang konsisten. Hal ini karena mereka menggunakan nilai *random state* yang sama.

4.2.2.8. Implementasi Bi-LSTM

Segmen Program 5.34

Model Bi-LSTM

```
bi_lstm_model = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=vocab_size+1,
output_dim=embedding_dim, input_length=max_length),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128,
return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,
return_sequences=True)),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dropout(0.5),
```

```
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(vocab_size+1,
activation='softmax')
])
```

Pada model diatas, Lapisan *Embedding* digunakan untuk mengubah ID-token menjadi *vector embedding*. Dimana pada *code* ini jumlah kata dalam vocabulary sebagai dimensi input dan setiap ID-token akan diubah menjadi vektor dengan panjang 128. Kemudian, *layer Bidirectional* digunakan agar model bisa melihat konteks secara 2 arah. Hal ini dikarenakan LSTM yang bersifat *undirectional* hanya meilhat konteks dari kiri kekanan maupun sebaliknya, sehingga dengan menggunakan *layer* ini dapat membantu LSTM lebih mengenali konteks dari berbagai arah. *Dropout* pada lapisan ini digunakna untuk mengurangi *overfitting* pada model dengan mematikan sebagian unit selama *training*. *Layer Global Average Pooling* digunakan untuk menghitung rata-rata dari semua *vector output* LSTM dalam dimensi waktu, yang nantikan akan menghasilkan vector dengan panjang yang tetap untuk setiap sampelnya. Kemudian, *layer Dropout* digunakan untuk membantu mencegah *overfitting*. Dimana *layer* ini bekerja dengan tingkat *dropout* sebesar 0,5. Yang mana artinya setiap model dilatih maka, setiap unit lapisan ini akan matikan secara acak dengan probabilitas 0,5. Dan layer yang terakhir adalah *layer dense* dimana layer ini berguna untuk melakukan pemetaan dari fitur-fitur yang dihasilkan oleh lapisan sebelumnya kedalam ruang dimensi yang lebih rendah.

Segmen Program 5.35

Compile Model

```
# Compile the model
bi_lstm_model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
```

Pada Segmen Program 4.35, variable *bi_lstm_model* yang berisikan model yang telah dibangun sebelumnya. Pada *'.compile'* digunakan untuk mengonfigurasi model agar siap untuk dilatih. Metode ini meiliki parameter berupa *'loss_function'*, '*optimizer'*, dan juga metrik yang akan digunakan selaha pelatihan. Pada *loss* yang menggunakan *sparse_categorical_crossentropy* akan digunakan untuk masalah klasifikasi yang dimana label telah di *encoded* sebelumnya. Kemudian optimizer akan menentukan algoritma yang digunakan untuk memperbaharui bobot model selama waktu pelatihan. Pada kasus ini *optimizer* menggunakan *Adam*. *Adam* sendiri merupakan salah

satu *optimizer* yang terkenal dengan efisiensinya dan kemampuannya menangani *sparse gradient* pada masalah *machine learning*. 'metrics=['accuracy']' adalah metrik yang sering digunakan dalam masalah klasifikasi, yang mana menunjukkan persentase yang benar dari total prediksi yang dibuat model.

Segmen Program 5.36

Train Model Bi-LSTM yang Telah Dibuat

```
num_epochs = 10
early_stopping =
tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=3, restore_best_weights=True)

# Call fit directly outside of any tf.function
bi_lstm_model.fit(train_texts,
                  train_labels,
                  epochs=num_epochs,
                  validation_data=(test_texts,
test_labels),
                  callbacks=[early_stopping],
                  verbose=2)
```

Pada Segmen Program 4.36 dapat dilihat penentuan jumlah *epoch* yang akna dilakukan. Kemudian membuat *early_stopping* menggunakan *library* dari keras . kemudian dilakukan pemanggilan metode '*.fit*' sebagai langkah awal melatih kode. Beberapa parameter yang dimiliki oleh metode ini adalah *'train_texts'*, *'train_labels'*, '*epoch'*, '*validation_data'*, '*callbacks'*, dan '*verbose'*. Parameter-parameter tersebut yang nantinya akan melakukan proses pelatihan terhadap model yang telah dibuat.

Segmen Program 5.37

Save Model Bi-LSTM

```
bi_lstm_model.save('bi_lstm_model_biLSTM.h5')
```

Pada Segmen Program 4.37 dilakukan *save model* terhadap algoritma yang terlah di latih agar model dapat digunakan dan dimasukkan kedalam *interface*.

4.2.2.9. Membuat *Function batch_predict_translation*

Segmen Program 5.38

Membuat *function batch_predict_translation*

```
def batch_predict_translation(words, model, word_index,
label_encoder):
tokenized_texts = batch_texts_to_sequences(words,
word_index, max_length)
predictions = model.predict(tokenized_texts, verbose=0)
predicted_label_ids = np.argmax(predictions, axis=1)
predicted_labels =
label_encoder.inverse_transform(predicted_label_ids)
return predicted_labels

loaded_bi_lstm_model =
tf.keras.models.load_model('bi_lstm_model_biLSTM.h5')
```

Pada Segmen Program 4.15 *function* digunakan untuk memprediksi terjemahan kata-kata individu menggunakan model Bi-LSTM yang telah dilatih sebelumnya. Fungsi ini memiliki parameter '*words*', '*model*', '*word_index*', dan '*label_encoder*'. Langkah pertama adalah membuat tokenisasi dengan menggunakna fungsi '*batch_texts_to_sequences*'. Kemudian teks yang telah di tokenisasi akan di prediksi dengan menggunakan model yang telah disimpan. Kemudian, indeks tertinggi dari setiap *output* model akan diambil. Indeks ini nantinya akna menunjukkan label yang paling memungkinkan untuk setiap kata. kemudian, indeks prediksi akan diubah mnejadi bentuk aslinya dengan menggunakan *label_encoder.inverse_transform*. Kemudian hasil akan di *return*. kemudian *model* akan di-*load* untuk digunakan saat melakukan prediksi.

4.2.2.10. Menerjemahkan Bahasa Bali Ke Bahasa Indonesia

Segmen Program 5.39

Menerima *Input* dan di *Tokenizer*

```
example_text = input("Enter the text to translate: ")


example_words = tokenize_words(example_text.lower())
```

Pada Segmen Program 4.39 langkah pertama yang dilakukan adalah membuat tempat untuk menerima *input*. Teks akan diterima di *console* dan disimpan kedalam variable '*example_teks*'. Kemudian, teks akan diubah menjadi huruf kecil dengan '*example_text.lower()*'. Hal ini dilakukan agam menjaga konsistensi pada data. Kemudian teks akan di tokenisasi.

Segmen Program 5.40

Menerjemahkan Bahasa Bali ke Bahasa Indonesia

```
translations = batch_predict_translation(example_words,
loaded_bi_lstm_model, word_index, label_encoder_words)
```

Pada Segmen Program 4.40 dilakukan penerjemahan dengan menggunakan *batch_predict_translation* yang memiliki parameter '*example_word*', '*loaded_bi_lstm_model*', '*word_index*', dan '*label_encoder_words*'. Nantinya, *translation* akan menyimpan hasil dari kata-kata yang telah diterjemahkan oleh *'batch_predict_translation'*.

Segmen Program 5.41

Menggabungkan hasil terjemahan dan di *print*.

```
translated_sentence = ' '.join(translations)
print("Translation:", translated_sentence)
```

Pada Segmen Program 4.41 dilakukan penggabungan kata terhadap kata-kata yang telah diterjemahkan sebelumnya. Kemudian hasil akan di *print* untuk melihat hasil terjemahan.

4.2.2.11. Evaluasi *BLEU*

Segmen Program 5.42

Evaluasi *BLEU*

```
reference_translation = input("Enter the reference
translation: ")

reference_words =
tokenize_words(reference_translation.lower())

bleu_score = sentence_bleu([reference_words],
translations,
smoothing_function=SmoothingFunction().method1)

print("BLEU score:", bleu_score)
```

Pada Segmen Program 4.42 Langkah pertama yang dilakukan adalah memasukkan input terjemahan yang benar dari penerjemahan. Kemudian terjemahan yang dimasukkan, akan

ditokenisasi dengan fungsi *tonenize_words*. Kemudian, *bleu_score* menghitung hasil terjemahan yang dilakukan oleh *translation* dengan terjemahan referensi.

4.2.3. Website

Segmen Program 5.43

Tampilan *Python Code* untuk *Gradio* dan *Model*

```
import gradio as gr
import tensorflow as tf
import numpy as np
import nltk
from nltk.tokenize import word_tokenize
from tensorflow.keras.preprocessing.sequence import
pad_sequences
import pandas as pd
import string
from sklearn.preprocessing import LabelEncoder

# Load CSS
with open("style.css") as f:
    styles = f.read()

# Load the model directly
model_path = 'bi_lstm_model_biLSTM.h5'
loaded_bi_lstm_model =
tf.keras.models.load_model(model_path)

# Download NLTK tokenizer data
nltk.download('punkt')

# Define utility functions
def tokenize_words(text):
    return word_tokenize(text)

def remove_punctuation(text):
    translator = str.maketrans('', '',
string.punctuation)
    return text.translate(translator)

def texts_to_sequences(texts, word_index):
    return [[word_index.get(word, 0) for word in
tokenize_words(text)] for text in texts]

def batch_texts_to_sequences(texts, word_index,
max_length):
    tokenized_texts = texts_to_sequences(texts,
word_index)
    return pad_sequences(tokenized_texts,
maxlen=max_length, padding='post')
```

```
def batch_predict_translation(texts, model,
word_index, label_encoder):
    tokenized_texts = batch_texts_to_sequences(texts,
word_index, max_length)
    predictions = model.predict(tokenized_texts,
verbose=0)
    predicted_label_ids = np.argmax(predictions,
axis=1)
    predicted_labels =
label_encoder.inverse_transform(predicted_label_ids)
    return predicted_labels

# Load the dataset
df_word = pd.read_excel('DATASET TERM.xlsx')

texts_words = df_word['BALI']
labels_words = df_word['INDONESIA']

texts_words =
texts_words.str.lower().apply(remove_punctuation)
labels_words =
labels_words.str.lower().apply(remove_punctuation)

label_encoder_words = LabelEncoder()
label_encoder_words.fit(labels_words)

# Build vocabulary and assign indices
all_words = [word for text in texts_words for word in
tokenize_words(text)]
unique_words = list(set(all_words))
word_index = {word: i+1 for i, word in
enumerate(unique_words)}

vocab_size = 60000
max_length = 128

# Define Gradio interface
def translate_text(input_text):
    input_text =
remove_punctuation(input_text.lower())
    input_words = tokenize_words(input_text)
    translation =
batch_predict_translation(input_words,
loaded_bi_lstm_model, word_index, label_encoder_words)
    return ' '.join(translation)

# Creating Gradio interface with Blocks
with gr.Blocks(css=styles, elem_classes="gradio-
container") as iface:
    gr.Markdown("Penerjemahan Bahasa Bali ke Bahasa
Indonesia", elem_classes="gr-center-markdown")
```

```
        with gr.Row():
            bali_input = gr.Textbox(label="Input Bahasa
Bali", elem_classes="gr-Textbox")
            indo_output = gr.Textbox(label="Output Bahasa
Indonesia", elem_classes="gr-Textbox")
        with gr.Column(elem_classes="gr-centered-
container"):
            translate_button = gr.Button("Translate",
elem_classes="gr-Button")
        translate_button.click(fn=translate_text,
inputs=bali_input, outputs=indo_output)

# Launching the interface
iface.launch(share=True)
```

Pada Segmen 4.43 menampilkan code untuk memanggil model, melakukan *preprocessing* sebelum code di terjemahkan, dan juga melakukan penerjemahan. Terdapat juga kode gradio yang digunakan untuk membuat *website*. Terdapat 2 *textbox* yang dimana nantinya *textbox* ini akan berperan sebagai penerima input dan *output*. Ada juga *button translate* yang akan digunakan bila ingin melakukan penerjemahan. Pada button ini nantinya akan menyambungkan kepada fuction *translate_text* yang nantinya akan menerjemahkan Bahasa Bali ke bahasa Indonesia.