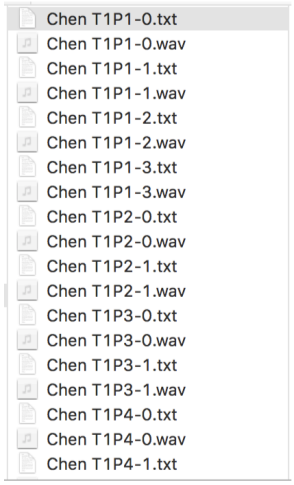
IMPLEMENTASI SISTEM

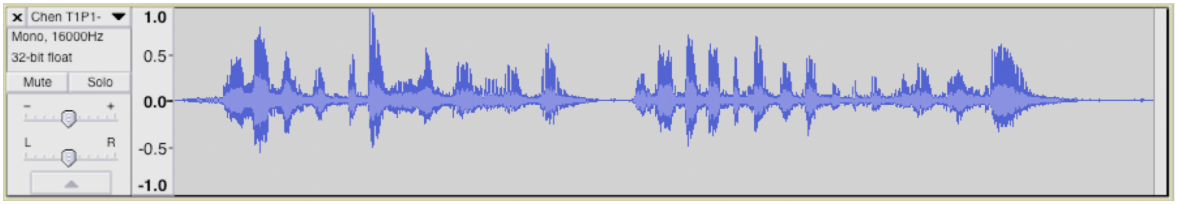
Pada bab ini akan dibahas tentang implementasi sistem sesuai dengan analisa dan desain sistem. Implementasi sistem meliputi Pengolahan Data, Implementasi Model Neural Network, dan Implementasi Language Model Untuk Testing.

4.1. Pengolahan Data

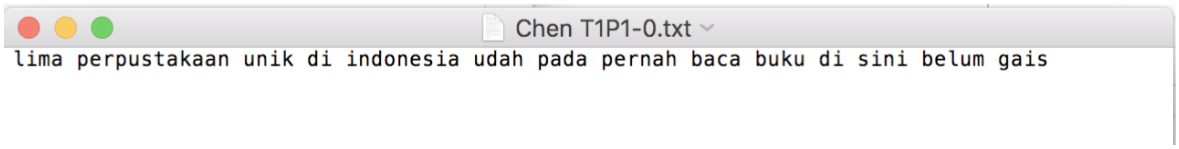
Pada tahap ini data direkam melalui *smartphone* dengan konfigurasi *mono* 44.1kHz yang di-*convert* menjadi 16kHz melalui *software* Audacity. Setiap data rekaman suara perlu dilakukan segmentasi setiap kalimat dalam rentang waktu 3-10 detik dan diberi kode nama_pembicara T#P#-### (T = transkrip, P = paragraf, ### = no urut). Bentuk penamaan nama file dapat dilihat pada Gambar 4.1, representasi data WAV dapat dilihat pada Gambar 4.2, representasi label teks dalam grafem (lihat Gambar 4.3) dan fonem (lihat Gambar 4.4)



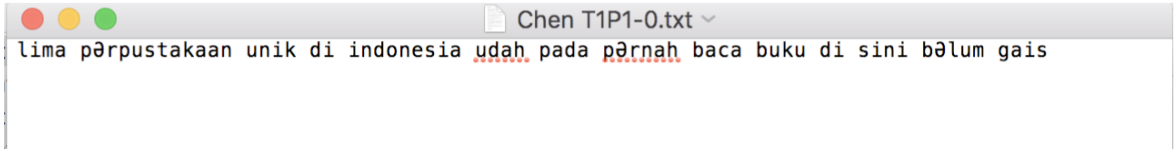
Gambar 4.1. Contoh bentuk penamaan *file* untuk *training dataset*.



Gambar 4.2. Representasi *wav* satu *sample* pada *software Audacity*.



Gambar 4.3. Representasi *label text* dalam bentuk grafem



Gambar 4.4. Representasi *label text* dalam bentuk fonem

4.1.2. Implementasi Prepocesing Data

Proses *preprocesing data* merupakan proses yang dilakukan sebelum menjalankan rangkaian neural network (*training* dan *testing*), proses ini sangat penting karena membantu memudahkan dan mempercepat proses berikutnya. Dataset suara beserta labelnya akan diolah terlebih dahulu menjadi file biner yang pengaksesannya lebih cepat ketimbang RAW data. Pada proses ini dibuat dalam *script python* bernama *preprocesing.py* dengan memberikan argumen sebagai berikut :

1. *Raw directory*, sebagai tempat *directory* / *folder* yang akan dibaca dan berisi file .wav dan .txt.
2. *Target directory*, sebagai tempat tujuan / *target directory* / *folder* yang akan dituju ketika menuliskan hasil outputnya berupa .npy *(numpy file)*.
3. *Feature type*, definisi tipe fitur yang akan di ekstrak, misalnya *mfcc* / *spectrogram*.

Sehingga ketika dijalankan, proses ini akan secara otomatis membaca semua file

Num of context, jumlah n konteks yang akan ditambahkan ke dalam indeks fitur (t-n)-t-(t+n). Pada preprocessing data membutuhkan modules *data_representation.py. Header code* implementasi *preprocessing.py* terdapat pada Segmen 4.1.

Segmen 4.1. *Header code* untuk meng-*handle* argumen pada *script preprocesing.py*

```
import sys
sys.path.append("../")
import os
import numpy as np
import modules.features.data_representation as data_rep
import modules.features.spectrogram as spectrogram
from scipy import signal
from scipy.io import wavfile

if len(sys.argv) < 5:
    print ('this method needs 3 args RAW_DIR FEATURE_DIR
NUM_CONTEXT')
    print ('RAW_DIR ~> directory of audio raw formatted
.WAV')
    print ('FEATURE_DIR ~> directory of target
preprocessing files will be created (MFCC)')
    print ('FEATURE_TYPE ~> mffc / spectrogram')
    print ('NUM_CONTEXT ~> mfcc : number of past and future
context, spectrogram : binsize ')
```

4.1.3. Pembacaan Data

Pada tahap ini argumen pertama diberikan sebagai acuan *root directory* untuk dibaca penggunaan *native library python* bernama os. Berikut contoh *source code* cara pembacaan data dengan sekaligus memisahkan proses pada extension file WAV dan TXT dapat dilihat di Segmen 4.2.

Segmen 4.2. *Source code* dalam membaca *dataset* pada *directory*.

```
feature_dir = os.path.join(feature_dir, preprocess_type)
            if not os.path.exists(feature_dir):
                os.makedirs(feature_dir)
            for root, dirs, files in os.walk(raw_dir,
topdown=False):
                for file in files:
                    name, ext = file.split('.')
                    if ext == 'wav':
                        print("preprocessing " + name)
                        with open(os.path.join(raw_dir,name
+ '.txt'), encoding='utf-8') as filetarget:
                            target = filetarget.read()
                            target =
target.replace("\n","")
                            indices_target =
data_rep.text_to_indices(target)

np.save(os.path.join(feature_dir,'_' +
name),indices_target)
                            filetarget.close()
                        filename = os.path.join(root, file)
```

Segmen 4.2 *Source code* dalam membaca *dataset* pada *directory*. (lanjutan)

```
                        vector_feature =
data_rep.mfcc_num_context(filename, num_context)
                        target_vector_dir =
os.path.join(feature_dir, name)
                        np.save(target_vector_dir,
vector_feature)
                        print (name + " has been saved to "
+ target_vector_dir)
```

4.1.4. Normalisasi Data

Pada tahap ini data dilakukan normalisasi data pada saat dibaca oleh proses sebelumnya dengan memanggil metode / fungsi normalize_to_db dengan 2 argumen:

1. *X* berupa *aray of int*, signal wav dalam tipe data int16
2. Db berupa int, target desibel sebagai acuan normalisasi, default = 0

Proses ini dilakukan dengan module terpisah dengan nama *function* normalize_to_db terdapat pada *data_representation.py* dapat dilihat di Segmen 4.3.

Segmen 4.3. *Source code* implementasi *peak normalization*

```
def normalize_to_db(x, db=0):
    targetdB = 10**(db/20)
    peak = np.abs(x).max()
    return (((targetdB * 32767) / peak) * x).astype(np.int)
```

4.1.5. Fitur Ekstraksi MFCC

Pada tahap ini dilakukan pada saat pembacaan data dengan memanggil metode / fungsi bernama mfcc_num_context dengan 2 argumen :

1. *Filename* berupa *string*, *filepath* lokasi audio file
2. *Numcontext* berupa int, jumlah n konteks

Metode / fungsi ini terdapat pada modul yang sama dengan normalisasi data yaitu *data_representation.py* dengan menggunakan *library numpy*, python_speech_features untuk mendapatkan mfcc, scipy.io untuk pembacaan wav file. *Source code* dapat dilihat pada Segmen 4.4.

Segmen 4.4. *Source code* implementasi *mfcc_num_context* untuk menghasilkan fitur MFCC

```
import numpy as np
from python_speech_features import mfcc
from scipy.io import wavfile
import wave
import random
from numpy import mean, sqrt, square, arrange

def mfcc_num_context(audio_filename, numcontext):
    fs, audio = wavfile.read(audio_filename)
    if len(audio.shape) == 2:
        audio = audio[:,0]
    numcep = 24
    audio = normalize_to_db(audio, 0)
    orig_inputs = mfcc(audio, samplerate=fs,winlen=0.02,
winstep=0.01,nfft=512, numcep=numcep,winfunc=lambda
x:np.hamming((x)))
    orig_inputs = orig_inputs[::2]
    train_inputs = np.array([], np.float32)
    train_inputs.resize((orig_inputs.shape[0], numcep + 2 *
numcep * numcontext))
    empty_mfcc = np.array([])
    empty_mfcc.resize((numcep))
    time_slices = range(train_inputs.shape[0])
    context_past_min = time_slices[0] + numcontext
    context_future_max = time_slices[-1] - numcontext
    for time_slice in time_slices:
        need_empty_past = max(0, (context_past_min -
time_slice))
        empty_source_past = list(empty_mfcc for empty_slots
in range(need_empty_past))
        data_source_past = orig_inputs[max(0, time_slice -
numcontext):time_slice]
        need_empty_future = max(0, (time_slice -
context_future_max))
        empty_source_future = list(empty_mfcc for
empty_slots in range(need_empty_future))
        data_source_future = orig_inputs[time_slice +
1:time_slice + numcontext + 1]
        if need_empty_past:
            past = np.concatenate((empty_source_past,
data_source_past))
        else:
            past = data_source_past
        if need_empty_future:
            future = np.concatenate((data_source_future,
empty_source_future))
        else:
            future = data_source_future
        past = np.reshape(past, numcontext * numcep)
        now = orig_inputs[time_slice]
        future = np.reshape(future, numcontext * numcep)

        train_inputs[time_slice] = np.concatenate((past,
now, future))
    return train_inputs
```

4.1.6. Representasi Label Data

Pada tahap ini dilakukan pada saat pembacaan data dengan memanggil metode / fungsi bernama text_to_indices dengan 1 argumen :

4. *Text* berupa *string*, teks yang ingin dijadikan kedalam bentuk numpy file

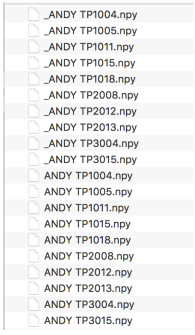
Metode / fungsi ini terdapat pada modul yang sama dengan normalisasi data yaitu *data_representation.py*, *source code* dapat dilihat di Segmen 4.5.

Segmen 4.5. Implementasi *text_to_indices*

```
#charset =
[‘a’,'b','c','d','e','ə','f','g','h','i','j','k','l','m','n','ŋ','ñ','o'
,'p','r','s','S','t','u','w','x',
'y','z',' ',''] #30
charset =
['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o
','p','r','s','t','u','w','y','z',' ',''] #25

def text_to_indices(text):
    text = text.replace(".","")
    text = text.replace(",","")
    text = text.replace("v","f")
    text = text.replace("-"," ")
    text = text.replace("?","")
    return np.array([charset.index(char) for char in text])
```

Dari semua proses yang dilakukan oleh preprocesing.py akan dihasilkan pada *Target Directory* sesuai dengan argumen yang diberikan, setiap data suara WAV akan di ubah menjadi 2 dimensi (waktu, kedalaman fitur) *array numpy (numpy file)* dalam bentuk file biner sesuai dengan nama file asli, sedangkan setiap label TXT akan di ubah sesuai dengan index charset menjadi 1 dimensi (banyak huruf) array numpy *(numpy file)* dalam bentuk file biner sesuai dengan nama file asli yang diawali dengan _. Contoh hasil preprocesing.py dapat dilihat Gambar 4.5



Gambar 4.5. Hasil dari *preprocessing.py*

4.2. Implementasi Model Neural Network

Setelah proses preprocessing data dilakukan akan dihasilkan 2 buad dataset untuk training dan testing. Kedua dataset inilah yang akan dibaca oleh proses *neural network* nantinya sehingga dapat dilakukan perulangan untuk tiap iterasinya. Pada proses ini semua dilakukan pada session di Tensorflow. Proses ini dibuat dalam *script python* yang bernama tensorflow_gpu.py / tensorflow_cpu.py (tergantung *hardware* yang digunakan), sebagai contoh berikut penjelasan mengenai tensorflow_gpu.py. Proses ini membutuhkan argumen sebagai berikut :

1. *Training directory*, sebagai tempat *directory* / *folder train* yang akan dibaca dan berisi file hasil dari preprocessing data.
2. *Testing directory*, sebagai tempat *directory* / *folder dev* yang akan dibaca dan berisi file hasil dari preprocessing data.
3. *Model directory*, sebagai tempat *directory* / *folder* model yang akan digunakan sebagai penyimpanan *checkpoint* setiap iterasi, restorasi *tensorflow mode*l, dan pencatatan hasil *report* dalam bentuk TXT dan CSV.
4. *Epoch,* jumlah epoch sekali proses dijalankan sehingga ketika dijalankan, proses ini akan secara otomatis membaca semua file. Epoch berarti setiap satu semua iterasi batch data dianggap satu epoch.
5. *Input Dimension,* jumlah panjang indeks fitur, contoh jika n = 5, maka (24 * (5+1+5) = 264.
6. Num of Character Recognition, jumlah karakter yang ingin dikenali sesuai dengan charset pada data_representation.py

Berikut adalah contoh header tensorflow_gpu.py, sebagian besar library yang digunakan adalah numpy dan tensorflow, dapat dilihat pada Segmen 4.6.

Segmen 4.6. *Header source code* yang digunakan pada *tensorflow_gpu*

```
import os
import sys
import csv

sys.path.append('../')
from time import gmtime, strftime
import modules.features.data_representation as data_rep
import numpy as np
```

4.2.1. Konfigurasi Awal Persiapan Model Neural Network

Pada tahap ini proses akan menerima argumen yang diperlukan untuk konfigurasi awal dalam membentuk jaringan pada neural network di Tensorflow. Adapun konfigurasi yang diperlukan dapat dilihat pada

Tabel 4.1. Tabel konfigurasi yang diperlukan untuk membentuk jaringan *neural network* pada *tensorflow_gpu.py*

#	Nama konfigurasi	Variable	Kegunaan
1	*Epoch*	epoch	Jumlah *epoch* setiap proses keseluruhan dijalankan
2	*Training Batch*	training_batch	Jumlah *batch* setiap *iteration* dalam satu epoch pada *training* dataset
3	*Testing Batch*	testing_batch	Jumlah *batch* setiap *iteration* dalam satu epoch pada *testing* dataset
4	*Number of Cepstrum*	num_cep	Jumlah dimensi input, pada MFCC jika n konteks = 5 maka 24 * (5+1+5) = 264
5	*Minimal Character Error Rate Difference*	min_char_error_rate_diff	Digunakan sebagai *warning report* jika *testing dataset* tidak mengalami perubahan yang signifikan
6	*Scale*	scale	Digunakan untuk rumus dengan notasi skala pada *Batch Normalization*
7	*Offset*	offset	Digunakan untuk rumus dengan notasi offset pada *Batch Normalization*
8	*Variance Epsilon*	variance_epsilon	Digunakan untuk rumus dengan notasi variance epsilon pada *Batch Normalization*
9	*Mean*	mean	Digunakan perhitungan rata-rata pada inisialisasi *random weight* dengan menggunakan distribusi

			normal
10	*Standard Deviation*	std	Digunakan perhitungan standar deviasi pada inisialisasi *random weight* dengan menggunakan distribusi normal
11	*ReLU Clip*	relu_clip	Sebagai batas atas activation function menggunakan ReLU (Rectified-Linear Unit)
12	*Number of neuron in Hidden Layer #*	n_hidden[1_7]	Jumlah *neuron* pada setiap *hidden layer*
13	*Forget Bias*	forget_bias	Digunakan parameter pada LSTM (Long-Short Term Memory)
14	*Beta 1* dan *Beta 2*	beta_1, beta_2	Digunakan untuk parameter dengan *variable beta 1 dan beta 2* pada *Adam Optimizer*
15	*Epsilon*	epsilon	Digunakan untuk parameter dengan *variable epsilon* pada *Adam Optimizer*
16	*Learning Rate*	learning_rate	Digunakan untuk parameter dengan *learning rate* pada *Adam Optimizer*

Dari konfigurasi di atas akan digunakan sebagai parameter pembentuk neural network dimana akan dibahas pada subbab 4.2.3.

4.2.2. Pembacaan Dataset Training dan Testing

Pada tahap ini akan dilakukan pembacaan dataset training dan testing sesuai dengan argumen yang telah diberikan. Semuanya akan di masukkan kedalam array masing-masing (di-*load* ke dalam RAM). Berikut source code pembacaan dataset training dan testing dapat dilihat pada Segmen 4.7.

Segmen 4.7. *Source code* pembacaan *dataset training* dan *testing* sesuai dengan argumen yang diberikan.

```
    training_dataset = []
    target_training_dataset = []
    testing_dataset = []
    target_testing_dataset = []
    for root, dirs, files in os.walk(training_dir,
topdown=False):
        for file in files:
            if file[0] != '_':
                print(os.path.join(training_dir, file))
```

Segmen 4.7 *Source code* pembacaan *dataset training* dan *testing* sesuai dengan argumen yang diberikan. (lanjutan)

```
target_training_dataset.append(np.load(os.path.join(trainin
g_dir, '_' + file)))
                new_training_set =
np.load(os.path.join(training_dir, file))
                training_dataset.append(new_training_set)
print('Loading testing dataset')
    for root, dirs, files in os.walk(testing_dir,
topdown=False):
        for file in files:
            if file[0] != '_':
                print(os.path.join(testing_dir, file))

target_testing_dataset.append(np.load(os.path.join(testing_
dir, '_' + file)))
                new_testing_set =
np.load(os.path.join(testing_dir, file))
                testing_dataset.append(new_testing_set)
```

4.2.3. Pembentukan Jaringan Neural Network

Pada pembentukan jaringan neural network akan dibagi-bagi menjadi 7 *scope* / lingkup besar, yaitu input, forward-net, biRNN, logits, decoder, loss, accuracy, dan optimizer. Semuanya dihubungkan / *link* melalui *variable-variable* yang berkaitan dan semuanya menggunakan *library* Tensorflow. Berikut pemenggalan setiap scope mulai dari input awal hingga *scope / lingkup* terakhir :

1. Input

Pada *scope* / lingkup ini dibentuk jaringan input yang dapat diberi *type placeholder,* dimana *placeholder* ini nantinya dapat diberikan argumen untuk dataset mana yang akan dimasukan dalam Tensorflow disebut *feed dictionary.* Source code dapat dilihat pada Segmen 4.8.

Segmen 4.8. *Source code* pada *input scope*

```
    with tf.device('/gpu:0'):
        start = timer()
        print('Building the model')
        alpha = tf.Variable(0.001,name='alpha')
        is_training = tf.placeholder(tf.bool,
name='is_training')
        input_batch = tf.placeholder(tf.float32, [None,
None, None], 'input')
        seq_len = tf.placeholder(tf.int32, [None],
name='sequence_length')
```

2. Forward-net

Pada *scope* / lingkup ini dibentuk jaringan *forward neural network* yang terdiri dari 3 *hidden layer*. Setiap pada hidden layer dilakukan juga setelahnya *batch normalization* (akan dijelaskan pada subbab 4.3.4) dan *dropout.* Source code dapat dilihat pada Segmen 4.9.

Segmen 4.9. *Source code* pada *forward-net scope*

```
        with tf.name_scope('forward-net'):
            shape_input_batch = tf.shape(input_batch)
            transpose_input_batch =
tf.transpose(input_batch, [1,0,2]) num_cepstrum]
            reshape_input_batch =
tf.reshape(transpose_input_batch, [-1, num_cep])
         w1 = tf.get_variable('fc1_w',[num_cep,
n_hidden_1],tf.float32,tf.random_normal_initializer(mean,st
d))
            b1 =
tf.get_variable('fc1_b',[n_hidden_1],tf.float32,tf.random_n
ormal_initializer(mean,std))
            h1 =
tf.minimum(tf.nn.relu(tf.add(tf.matmul(reshape_input_batch,
w1), b1)), relu_clip)
            h1_bn = batch_norm(h1, 'fc1_bn',
tf.cast(is_training, tf.bool))
            h1_dropout = tf.nn.dropout(h1_bn,1 - 0.05)

            w2 = tf.get_variable('fc2_w',[n_hidden_1,
n_hidden_2],tf.float32,tf.random_normal_initializer(mean,st
d))
            b2 =
tf.get_variable('fc2_b',[n_hidden_2],tf.float32,tf.random_n
ormal_initializer(mean,std))
            h2 =
tf.minimum(tf.nn.relu(tf.add(tf.matmul(h1_dropout, w2),
b2)), relu_clip)
            h2_bn = batch_norm(h2, 'fc2_bn',
tf.cast(is_training, tf.bool))
            h2_dropout = tf.nn.dropout(h2_bn,1 - 0.05)

            w3 = tf.get_variable('fc3_w',[n_hidden_2,
n_hidden_3],tf.float32,tf.random_normal_initializer(mean,st
d))
            b3 =
tf.get_variable('fc3_b',[n_hidden_3],tf.float32,tf.random_no
rmal_initializer(mean,std))
            h3 =
tf.minimum(tf.nn.relu(tf.add(tf.matmul(h2_dropout, w3),
b3)), relu_clip)
            h3_bn = batch_norm(h3, 'fc3_bn',
tf.cast(is_training, tf.bool))
            h3_dropout = tf.nn.dropout(h3_bn,1 - 0.05)
```

Pada input layer pada *batch data* diperlukan penyesuaian dengan cara kerja Tensorflow, diperlukan setiap indeks i yang sama diparalel terhadap batch j yang sama. Jika tidak dilakukan paralel data seperti ini, akan bermasalah pada penghitungan *Adam Optimizer* nantinya. Misalkan data setiap *batch data* adalah sebagai berikut :

```
[[[ 1,  2,  3],                                  [[[ 1,  2,  3],
  [ 4,  5,  6]],.....diubah menjadi.....          [ 7,  8,  9],
 [[ 7,  8,  9],                                   [[ 4,  5,  6],
  [10, 11, 12]]]                                   [10, 11, 12]]]
```

3. Bi-RNN

Pada *scope* / lingkup ini dibentuk jaringan *bidirectional Neural Network.* Sebelum dilakukan bi-RNN perlu ada penyesuaian cara kerja Tensorflow, bi-RNN hanya menerima bentuk [time x batchsize x neuron] dibandingkan data dimensi sebelumnya yang hanya [time * bathsize (paralel) x num_cep]. Source code dapat dilihat pada Segmen 4.10

Segmen 4.10. *Source code* pada *bi-RNN scope*

```
        with tf.name_scope('biRNN'):
            h3_dropout = tf.reshape(h3_dropout, [-1,
shape_input_batch[0], n_hidden_3])
            forward_cell_1 = BasicLSTMCell(n_hidden_4,
forget_bias=1.0, state_is_tuple=True)
            forward_cell_1 =
DropoutWrapper(forward_cell_1,1.0 - 0.0, 1.0 - 0.0)
            backward_cell_1 = BasicLSTMCell(n_hidden_4,
forget_bias=1.0, state_is_tuple=True)
            backward_cell_1 =
DropoutWrapper(backward_cell_1, 1.0 - 0.0, 1.0 - 0.0)
            outputs, _ =
tf.nn.bidirectional_dynamic_rnn(cell_fw=forward_cell_1,
cell_bw=backward_cell_1, inputs=h3_dropout,time_major=True,
sequence_length=seq_len, dtype=tf.float32)
            outputs = tf.concat(outputs, 2)
            w6 = tf.get_variable('fc6_w',[n_hidden_3,
n_hidden_6],tf.float32,tf.random_normal_initializer(mean,st
d))
            b6 =
tf.get_variable('fc6_b',[n_hidden_6],tf.float32,tf.random_n
ormal_initializer(mean,std))
            h5 = tf.reshape(outputs, [-1, 2 * n_hidden_5])
            h6 = tf.minimum(tf.nn.relu(tf.add(tf.matmul(h5,
w6), b6)), relu_clip)
            h6_bn = batch_norm(h6, 'fc6_bn',
tf.cast(is_training, tf.bool))
```

Pada akhir operasi dilakukan kembali pengubahan dimensi menjadi 2 dimensi untuk kembali di hitung pada hidden layer ke- 6 *(forward neural network).*

4. Logits

Pada *scope* / lingkup ini dibentuk jaringan *logits*. *Logits* adalah hasil dari semua jaringan *neural network* di *hidden layer* sebelum masuk ke dalam *output layer*. *Output layer* yang digunakan adalah *Softmax Operation* dimana sudah termasuk pada bagian metode / fungsi ctc_loss. *Source code* dapat dilihat pada Segmen 4.11.

Segmen 4.11. *Source code* pada *logits scope*

```
        with tf.name_scope('logits'):
            w7 = tf.get_variable('fc7_w',[n_hidden_6,
n_hidden_7],tf.float32,tf.random_normal_initializer(mean,st
d))
            b7 =
tf.get_variable('fc7_b',[n_hidden_7],tf.float32,tf.random_n
ormal_initializer(mean,std))
            h7 = tf.add(tf.matmul(h6_dropout, w7), b7)
            logits = tf.reshape(h7, [-1,
shape_input_batch[0], n_hidden_7])
```

Pada akhir operasi dilakukan kembali pengubahan dimensi menjadi 3 dimensi karena pada ctc_loss diperlukan 3 dimensi [time x batchsize x neuron].

5. Decoder dan Loss

Pada 2 *scope* / lingkup ini dilakukan Softmax Operation menggunakan ctc_loss dan penghitungan hasil prediksi kata-kata yang dinamakan decoder menggunakan ctc_beam_search_decoder. *Source code* dapat dilihat pada

Segmen 4.12. *Source code* pada *decoder* dan *loss scope*

```
        with tf.name_scope('decoder'):
            decode, log_prob =
tf.nn.ctc_beam_search_decoder(inputs=logits,
sequence_length=seq_len, merge_repeated=True)
            targets = tf.sparse_placeholder(tf.int32,
[None, None], name='target')
       with tf.name_scope('loss'):
            ctc_loss = tf.nn.ctc_loss(labels=targets,
                                      inputs=logits,
sequence_length=seq_len)
            avg_loss = tf.reduce_mean(ctc_loss)
            tf.summary.histogram('avg_loss', avg_loss)
```

6. Accuracy

Pada *scope* / lingkup ini dilakukan *Edit Distance (Levenshtein Distance)* menggunakan edit_distance. *Source code* dapat dilihat pada Segmen 4.13

Segmen 4.13. *Source code* pada *accuracy scope*

```
        with tf.name_scope('accuracy'):
            distance = tf.edit_distance(tf.cast(decode[0],
tf.int32), targets)
            ler = tf.reduce_mean(distance,
name='label_error_rate')
```

7. Optimizer

Pada scope / lingkup ini dilakukan Adam Optimizer menggunakan class AdamOptimizer yang tersedia di Tensorflow. Untuk parameter didapat dari konfigurasi pada subbab 4.3.1. *Source code* dapat dilihat Segmen 4.14

Segmen 4.14. *Source code* pada *Adam Optimizer*

```
        with tf.name_scope('optimizer'):
            optimizer =
tf.train.AdamOptimizer(learning_rate=alpha,
                                                beta1=beta1,
                                                beta2=beta2,
epsilon=epsilon)
            optimizer = optimizer.minimize(avg_loss)
```

4.2.4. Menjalankan Neural Network

Sebelum semuanya dijalankan perlu adanya session yang dinisialiasi pertama kali sebelum menjalankan neural network. Pada proses setelah pembentukan neural network akan dijalankan dari setiap iterasi. Di dalam Tensorflow dinamakan *feed dictionary* untuk memasukkan inputan neural network (*placeholder* yang sudah di atur pada subab 4.3.3), *source code* dapat dilihat pada Segmen 4.15.

Segmen 4.15. *Source code* dalam menjalankan *iteration* pada setiap *batch*

```
        for iter in range(epoch):
            report_training = open(os.path.join(report_dir,
'report_training.txt'), 'a')
            reporttrainingcsv =
open(os.path.join(report_dir, 'result_training.csv'), 'a')
            report_testing = open(os.path.join(report_dir,
'report_testing.txt'), 'a')
            reporttestingcsv =
open(os.path.join(report_dir, 'result_testing.csv'), 'a')
            trainingcsvwriter =
csv.writer(reporttrainingcsv)
            testingcsvwriter = csv.writer(reporttestingcsv)
            # ================================TRAINING
```

```
PHASE================================
            print('epoch #' + str(iter + last_epoch))
            report_training.write('epoch #' + str(iter +
last_epoch) + '\n')
            csv_training_values = []
            csv_training_values.append(iter + last_epoch)
            if iter > 0:
                training_old_losses = training_losses
                training_losses = []

            training_shuffled_index =
np.arange(len(training_dataset))
            np.random.shuffle(training_shuffled_index)

            for i in range(int(len(training_dataset) /
int(training_batch))):
                start = timer()
                csv_training_values = []
print('================================TRAINING PHASE
BATCH #' + str(i) + ' EPOCH AT ' + str(
                    iter) +
'================================')
                report_training.write(

'================================TRAINING PHASE BATCH #' +
str(i) + ' EPOCH AT ' + str(
                        iter) +
'================================' + '\n')
                csv_training_values.append(i)
                csv_training_values.append(learning_rate)

                # get batch shuffled index
                batch_i = []
                target = []
                for j in range(training_batch):

batch_i.append(training_dataset[training_shuffled_index[j +
(i * training_batch)]])

target.append(target_training_dataset[training_shuffled_ind
ex[j + (i * training_batch)]])

                batch_i = data_rep.sparse_dataset(batch_i)
                print(batch_i.shape)
                # batch_i =
training_dataset[(i*batch):(i*batch)+batch]
                sequence_length =
np.array([batch_i.shape[1] for _ in range(training_batch)])

                # target =
target_training_dataset[(i*batch):(i*batch)+batch]
                sparse_labels =
data_rep.SimpleSparseTensorFrom(target)
                feed = {
                    input_batch: batch_i,
                    seq_len: sequence_length,
                    targets: sparse_labels,
                    is_training: True,
                    alpha: learning_rate
```

```
                }
                loss, logg, _ = sess.run([avg_loss, decode,
optimizer], feed)
                print('Encoded CTC :')
                report_training.write('Encoded CTC :' +
'\n')
                decode_text =
data_rep.indices_to_text(logg[0][1])
                print(decode_text)
                print('first target : \n' +
data_rep.indices_to_text(target[0]))
                report_training.write(decode_text + '\n')
                report_training.write('first target : ' +
data_rep.indices_to_text(target[0]) + '\n')

csv_training_values.append(data_rep.indices_to_text(target[
0]))

                print('negative log-probability :' +
str(loss))
                report_training.write('negative log-
probability :' + str(loss) + '\n')
                csv_training_values.append(loss)
                csv_training_values.append(decode_text)
csv_training_values.append(data_rep.indices_to_text(target[
0]))
trainingcsvwriter.writerow(csv_training_values)
                training_losses.append(loss)
                elapsed_time = timer() - start
                cycle_batch = int(len(training_dataset) /
int(training_batch))
                remaining_time = (((epoch - iter) *
cycle_batch) - i) * elapsed_time
                print('Elapsed time : ' +
str(elapsed_time))
                report_training.write('Elapsed time: ' +
str(elapsed_time) + '\n')
                print('Remaining time : ' +
str(remaining_time))
                report_training.write('Remaining time: ' +
str(remaining_time) + '\n')
```

4.2.5. Restorasi dan Penyimpanan Hasil Neural Network

Restorasi dan Penyimpanan menggunakan class tf.Saver, dimana Tensorflow memiliki bentuk file model sendiri yang berekstensi .CKPT. Source code dapat dilihat pada Segmen 4.16

Segmen 4.16. *Source code* restorasi tensorflow model

```
print('Restoring ' +
os.path.join(os.path.join(checkpoint_dir,
last_checkpoint)))
            # saving model state
            saver = tf.train.Saver()
            saver.restore(sess,
os.path.join(os.path.join(checkpoint_dir, last_checkpoint),
'tensorflow_1.ckpt'))
            target_checkpoint_dir =
os.path.join(os.path.join(checkpoint_dir, last_checkpoint))
```

Segmen 4.17. source code penyimpanan tensorflow model

```
print('Saving ...')
                now = strftime('%Y-%m-%d-%H-%M-%S',
gmtime())
                saver = tf.train.Saver()
                target_checkpoint_dir =
os.path.join(checkpoint_dir, 'DMC-' + now)
                if not
os.path.exists(target_checkpoint_dir):
                    os.makedirs(target_checkpoint_dir)
                save_path = saver.save(sess,
os.path.join(target_checkpoint_dir, 'tensorflow_1.ckpt'))
                print('Checkpoint has been saved on path :
' + str(save_path))
                report_training.write('Checkpoint has been
saved on path : ' + str(save_path) + '\n')
```

4.3. Pembuatan Language Model

Language model digunakan untuk membenarkan / mengkoreksi kata-kata yang dihasilkan kurang tepat oleh *neural network*. Language model dibentuk dalam *file* JSON karena dibuat *dictionary* / kamus agar memudahkan cepat pengaksesan dengan indeks berupa *string*. Bentuk language model dapat dilihat pada Gambar 4.6.

Gambar 4.6. *sample* daripada *language model* berbentuk JSON

Terdapat dua indeks utama, pertama adalah dictionaries yaitu kumpulan kata-kata yang terdapat pada semua label. Kedua adalah *word gram* kumpulan kata yang tersusun dari n-gram. Membuat *language model* terdapat pada script python bernama create_model_language.py dengan memberikan argumen sebagai berikut :

1. *Dataset directory*, sebagai tempat *directory* / *folder dev* dan *train* yang berisi label TXT.
2. *N Gram*, sebagai jumlah n-gram, biasanya 2

3. *Target JSON*, sebagai filepath beserta nama berekstensi .JSON sebagai hasil dari proses create_model_language.py

Source code untuk membuat *language model* dapat dilihat pada segmen Segmen 4.18

Segmen 4.18. *Header code* pada *create_model_language.py*

```
import sys
sys.path.append("../")
import os
import json
from modules.model.language import word_to_word_gram
```

Pada proses create_model_language.py diperlukan metode / fungsi word_to_word_gram dimana menerima 3 argumen sebagai berikut :

1. *Word Gram* berupa *dict*, tempat penyimpanan hasil *word gram*
2. *Word* berupa *string*, kata baru yang ingin dimasukkan
3. N berupat *int*, jumlah n-gram

Source code implementasi metode word_to_word_gram dapat dilihat segmen Segmen 4.19

Segmen 4.19. *Source code* implementasi metode / fungsi *word_to_word_gram*

```
def add_gram_to_word_gram(word_gram, gram, new_word):
    if gram not in word_gram:
        word_gram[gram] = []
    word_gram[gram].append(new_word)
def n_gram(word,n):
    gram = []
    end = len(word)
    for i in range(end):
        if len(word[i:(i+n)]) == n:
            gram.append(word[i:(i+n)])
            gram.sort()
    return gram
def word_to_word_gram(word_gram, new_word,n):
    for g in n_gram(new_word,n):
        add_gram_to_word_gram(word_gram,g,new_word)
```

Proses create_model_language.py melakukan pembaca pada directory sesuai argumen yang diberikan secara otomatis mengambil semua kata dalam kalimat pada label TXT. Source code implementasi create_model_language.py dapat dilihat pada Segmen 4.20.

Segmen 4.20. source code implementasi create_model_languge.py

```
if (len(sys.argv) < 4):
    print("DATASET_DIR ~> dataset dir containing dev and train")
    print("N_GRAM ~> number of gram")
    print("TARGET_JSON ~> target json file")
else:
    dataset_dir = sys.argv[1]
    n = int(sys.argv[2])
    target_jsonfile = sys.argv[3]
    word_gram = dict()
    word_count = dict()
    dictionaries = []
    for root, dirs, files in os.walk(os.path.join(dataset_dir,'dev'),
topdown=False):
        for file in files:
            if file[-4:] == '.txt':
                with open(os.path.join(root, file), encoding='utf-8')
as filetarget:
                    target = filetarget.read()
                    target = target.replace(',','')
                    target = target.replace('.','')
                    target = target.replace('-',' ')
                    target = target.replace('?',' ')
                    target = target.lower()
                    words = target.split(' ')
                    for word in words:
                        word = word.replace('\n','')
                        if (not word in dictionaries):
                            dictionaries.append(word)
                            word_to_word_gram(word_gram,word,n)
    for root, dirs, files in
os.walk(os.path.join(dataset_dir,'train'), topdown=False):
        for file in files:
            if file[-4:] == '.txt':
                with open(os.path.join(root, file), encoding='utf-8')
as filetarget:
                    target = filetarget.read()
                    target = target.replace(',','')
                    target = target.replace('.','')
                    target = target.replace('-',' ')
                    target = target.replace('?',' ')
                    target = target.lower()
                    words = target.split(' ')
                    for word in words:
                        word = word.replace('\n','')
                        if (not word in dictionaries):
                            dictionaries.append(word)
                            word_to_word_gram(word_gram,word,n)
    dictionaries.sort()
    with open(target_jsonfile, 'w') as outfile:
       json.dump({'dictionaries':dictionaries,'word_gram':word_gram},
outfile)
```

4.3.2. Implementasi Language Model Pada Neural Network

Dalam pengimplementasian server menggunakan *Session* pada Tensorflow untuk menjalankan neural network sesuai dengan *feeding dict* yang diberikan. Komunikasi antara server dan client menggunakan socket (libraryi menggunakan *python native socket programming*). Header code pada implementasi server yang bernama test_server.py dapat dilihat pada Segmen 4.21

Segmen 4.21. *Header code* implementasi *test_server.py*

```
import sys
sys.path.append("../../../")
import os
import json
import numpy as np
import wave
from time import gmtime, strftime
import tensorflow as tf
from tensorflow.python.training import moving_averages
from tensorflow.contrib.rnn import BasicLSTMCell, DropoutWrapper
from timeit import default_timer as timer
import configparser as cp
import server.www.scripts.preprocessing as preproc
import modules.model.language as lang
import socket
from socket import error as SocketError
import socketserver
from binascii import unhexlify
```

Proses pertama kali yang dilakukan oleh test_server.py Adalah menginisialisasi Session pada Tensorflow dengan mengambil konfigurasi sama seperti pada subbab 4.3.1. Restorasi akan dilakukan diawal dengan *format directory* yang sudah didefinisikan manual secara *code*. Source code cara membuka *port* untuk *listen* pada *server port* tertentu pada Segmen 4.22.

Segmen 4.22. Source code membuka port untuk listen

```
serversocket = socket.socket(
    socket.AF_INET, socket.SOCK_STREAM)
host = "127.0.0.1"
port = 14093
serversocket.bind(('', port))
```

Ketika port sudah dibuka, *Session* Tensorflow akan dijalankan dengan merestorasi *tensorflow model* dengan menunggu hingga ada data suara yang masuk. Data yang dibaca sebanyak 1280 *bytes* karena mengikuti konfigurasi *minimum buffer* pada *Android Audio Recorder*. Source code implementasi *test_server.py* dapat dilihat pada Segmen 4.23.

Segmen 4.23. Source code implementasi keseluruhan test_server.py

```
with tf.Session(config=tf.ConfigProto(allow_soft_placement=True,
log_device_placement=False)) as sess:
#restoring model
    serversocket.listen(1)
    print('Listen from server ' + str(host) + ':' + str(port))
    isConnectToClient = False
    while True:
        clientsocket, addr = serversocket.accept()
        isConnectToClient = True

        print("Got a connection from %s" % str(addr))
        now = strftime("%Y-%m-%d-%H-%M-%S", gmtime())
        wavfile = wave.open(os.path.join(wav_data,now+'.wav'), 'w')
        wavfile.setparams((1, 2, 16000, 0, 'NONE', 'not compressed'))
        length = 0
        print("waiting")
        while isConnectToClient:
            try:
                data = clientsocket.recv(1280)
                if (len(data) > 0):
                    if str(data[-4:]) == "b'!EOF'":
                        wavfile.writeframes(data[0:len(data)-4])
                    else:
                        wavfile.writeframes(data)
                    length += len(data)
                    if str(data[-4:]) == "b'!EOF'":
                        wavfile.close()
                        print("saved : "+ now+'.wav')
                        now = strftime("%Y-%m-%d-%H-%M-%S", gmtime())
                        preprocessing_data(sess,
clientsocket,model_lang)
                        wavfile = wave.open(os.path.join(wav_data,
now + '.wav'), 'w')
                        wavfile.setparams((1, 2, 16000, 0, 'NONE',
'not compressed'))
                        length = 0
            except SocketError as e:
                print(e)
                clientsocket.close()
                isConnectToClient = False
```

Pada proses setelah ada data yang masuk akan diproses menjadi data suara WAV yang disimpan sementara pada server. Ketika data terakhir mengandung !EOF berarti data suara audio telah selesai direkam dan harus diproses ke *preprocessing data.* Pada preprocessing data sama dengan subbab 4.2.3. Setelah dihasilkan data numpy file akan dijadikan sebagai *feed dict* pada *neural network.* Lalu akan dilakukan decoder dan koreksi kata dengan *language model. Language model* di-*load* disesuaikan dengan *manual code* untuk pathfile JSON-nya. Source code metode / fungsi preprocessing_data, run_model dan

run_language_model yang terdapat pada proses test_server.py pada Segmen 4.24, Segmen 4.25, dan Segmen 4.26

Segmen 4.24. *Source code* metode / fungsi preprocessing_data

```
def preprocessing_data(sess, client, model_language):
    for root, dirs, files in os.walk(wav_data, topdown=False):
        for file in files:
            fileproperties = file.split('.')
            if (file[0] != '_' and fileproperties[-1] == 'wav'):
                if type == 'mfcc':
                    print("Feature extraction mfcc")
                    preproc.mffc_rep_wav(root,file,num_context)

os.rename(os.path.join(root,file),os.path.join(root,'_'+filepropertie
s[0] + '.wav'))
    run_model(sess,client, model_language)
```

Segmen 4.25. Source code metode / fungsi run_model

```
def run_model(sess,clientsocket,model_language):
    datas = []
    for root, dirs, files in os.walk(preprocessed_data,
topdown=False):
        for file in files:
            if (len(file.split('.')) == 2):
                filename, ext = file.split('.')
                if file[0] != '_' and ext == 'npy':
                    filename = os.path.join(root,file)
                    datas.append(np.load(filename))
                    os.remove(os.path.join(root, file))
        for data in datas:
            batch_i = preproc.data_rep.sparse_dataset([data])
            sequence_length = np.array([batch_i.shape[1] for _ in
range(testing_batch)])
            target = np.array([1])
            sparse_labels =
preproc.data_rep.SimpleSparseTensorFrom([target])
            feed = {
                input_batch: batch_i,
                seq_len: sequence_length,
                targets: sparse_labels,
                is_training: False,
                alpha: 0
            }
    print("Running decoder")
            decoder = sess.run(decode, feed)
            decode_text =
preproc.data_rep.indices_to_text(decoder[0][1]) + "\n"
            trytosend(decode_text.encode(),clientsocket)
            print(decode_text)

run_language_model(sess,clientsocket,model_language,decode_text)
```

Segmen 4.26. Source code metode / fungsi run_language_model

```
def run_language_model(sess,clientsocket, model_language, s):
    sentence = ""
    words = s.split(' ')
    for w in words:
        if len(w) > 1:
            count_words = []
            prob_words = []
            gram = lang.n_gram(w, 2)
            for g in gram:
                if g in model_lang['word_gram']:
                    for prob_word in model_lang['word_gram'][g]:
                        if prob_word in prob_words:
                            index = prob_words.index(prob_word)
                            count_words[index] += 1
                        else:
                            count_words.append(1)
                            prob_words.append(prob_word)
            sparse_prob_word, sparse_input_word =
lang.wordSparseTensor(prob_words, w)
            res = sess.run(sim, feed_dict={
                ref : sparse_prob_word,
                raw : sparse_input_word
            })
            sentence += prob_words[res.argmin()] + " "
    print("language_model")
    print(sentence)
```

4.3.3. Implementasi Client Pada Android

Implementasi pada client akan dilakukan Pembukaan koneksi internet dengan alamat IP dan *port* yang sesuai dengan *server.* Semua operasi pengiriman data dan penerimaan data di-*handle* pada class bernama TCPClient. Di dalam TCPClient terdapat 3 class utama yang digunakan untuk *Connect*, *Send*, dan *Receive.* Pada *Connect* dan *Receive* mengimplementasikan *Runnable* untuk menjalankan *Thread*. S*ource code* implementasi *class ConnectRunnable* dapat dilihat pada Segmen 4.27.

Segmen 4.27. *Source code* implementasi class ConnectRunnable

```
class ConnectRunnable implements Runnable {
    @Override
    public void run() {
        try {
            InetAddress serverAddress =
InetAddress.getByName(serverIP);
            startTime = System.currentTimeMillis();
            connectionSocket = new Socket();
            connectionSocket.connect(new
InetSocketAddress(serverAddress, serverPort), timeout);
```

```
            long time = System.currentTimeMillis() - startTime;
            Log.d(TAG, "Connected! Current duration: " + time + "
ms");
            if (mList != null) mList.OnConnectionSuccess();

        } catch (Exception e) {
            if (mList != null) {
                mList.OnConnectionerror();
            }
        }
        Log.d(TAG, "Connection thread stopped!");
    }
}
```

Class ConnectRunnable yang terdapat pada segmen program 4.28 berguna untuk meng-handle threading pada koneksivitas terhadap server melalui port dan IP tertentu.

Kedua, implementasi class SendService dapat dilihat pada Segmen 4.28.

Segmen 4.28. *Source code* implementasi *class SendService*

```
class SendService {
    byte[] data;
    private OutputStream out;

    public SendService(Socket socket) {
        try {
            this.out = socket.getOutputStream();
        } catch (Exception e) {
        }
    }

    public void Send(byte[] bytes) {
        this.data = bytes;
        SendToServer();
    }

    private void SendToServer() {
        startTime = System.currentTimeMillis();
        try {
            //Send the data
            this.out.write(data, 0, data.length);
            //Flush the stream to be sure all bytes has been written
out
            this.out.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
        this.data = null;
        long time = System.currentTimeMillis() - startTime;
        Log.d(TAG, "Data has been sent! Current duration : " + time +
" ms");
    }
}
```

Class SendService bertujuan untuk mengirim data ke *server*. Dalam pengiriman data tidak perlu hingga threading namun hanya cukup memanggil fungsi pada baris ke 21 pada segmen program 4.29.

Ketiga, implementasi *class ReceiveRunnable* dapat dilihat pada Segmen 4.29.

Segmen 4.29. *Source code* implementasi class ReceiveRunnable

```
class ReceiveRunnable implements Runnable {
    private Socket socket;
    private InputStream input;

    public ReceiveRunnable(Socket socket) {
        this.socket = socket;
        try {
            input = socket.getInputStream();
        } catch (Exception e) {}
    }
    @Override
    public void run() {
        Log.d(TAG,"Receiving started!");
        while (!Thread.currentThread().isInterrupted() &&
isConnected()) {
            if (!receiveThreadRunning)
                receiveThreadRunning = true;
            try {
                byte[] data = new byte[1];
                StringBuilder sb = new StringBuilder();
                InputStream bis = new BufferedInputStream(input);
                int read = 0;
                while ((read = bis.read(data)) != -1) {
                    String newChar = new String(data,0, read, "UTF-
8");
                    if (newChar.equals("\n"))
                        break;
                    sb.append(newChar); //Append the data to the
StringBuilder
                }
                if (mList != null) {
                    mList.OnMessageReceived(sb.toString());
                }
                long time = System.currentTimeMillis() - startTime;
                Log.d(TAG, "Data received! Took: " + time + "ms and
got: " + sb.toString());
                stopThreads();
            } catch (IOException e) {
                Disconnect();
            }
        }
        receiveThreadRunning = false;
        Log.d(TAG, "Receiving stopped");
    }
}
```

Class ReceiveRunnable bertujuan untuk menerima data yang dikirim dari *server*. Dibutuhkan *thread* khusus untuk mengambil data hingga selesai (pembacaan sesuai *buffer bytes*).