SPEECH TO TEXT UNTUK BAHASA INDONESIA

Disusun Oleh:

N a m a : Teguh Puji Laksono

NIM : 14523043

PROGRAM STUDI TEKNIK INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2018

HALAMAN PENGESAHAN DOSEN PEMBIMBING

SPEECH TO TEXT UNTUK BAHASA INDONESIA

TUGAS AKHIR

Disusun Oleh:

N a m a : Teguh Puji Laksono

NIM : 14523043

Yogyakarta, 16 Agustus 2018

Pembimbing,

(Ahmad Fathan Hidayatullah S.T., M.Cs.)

HALAMAN PENGESAHAN DOSEN PENGUJI

SPEECH TO TEXT UNTUK BAHASA INDONESIA

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Teknik Informatika
di Fakultas Teknologi Industri Universitas Islam Indonesia
Yogyakarta, 16 Agustus 2018

Tim Penguji

Ahmad Fathan Hidayatullah S.T., M.Cs.

Anggota 1

Dr. Sri Kusumadewi, S.Si., M.T.

Anggota 2

Taufiq Hidayat, S.T., M.C.S.

Mengetahui,
Ketua Program Studi Teknik Informatika - Program Sarjana
Fakultas Teknologi Industri
Universitas Islam Indonesia

(Dr. Raden Teduh Dirgahayu, S.T., M.Sc.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Teguh Puji Laksono

NIM : 14523043

Tugas akhir dengan judul:

SPEECH TO TEXT UNTUK BAHASA INDONESIA

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila dikemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung resiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 16 Agustus 2018

(Teguh Puji Laksono)

HALAMAN PERSEMBAHAN

Alhamdulilahi Robbil 'Alamin puji syukur atas segala nikmat dan karunia yang Allah SWT yang berikan kepada saya sehingga tugas akhir ini dapat terselesaikan dengan baik. Atas semua dukungan dan bantuan yang telah diberikan, saya persembahkan tugas akhir ini untuk,

Kedua orang tua saya yang tercinta, Bapak Suherno dan Ibu Syamsiah,
Kakak saya Lisa Tanjung Sari
serta Keluarga Besar,

Seluruh teman-teman 2014 yang menjadi teman seperjuangan di bangku perkuliahan,
Dan seluruh teman-teman yang tidak bisa disebutkan satu per satu.

HALAMAN MOTO

" Sungguh, Allah tidak akan mengubah nasib suatu kaum sampai mereka sendiri mengubah dirinya"

(Terjemah QS Ar Ra'd : 11)

"Allah meninggikan orang yang beriman diantara kamu dan orang-orang yang diberi ilmu pengetahuan, beberapa derajat

(Terjemah QS Al-Mujadilah : 11)

" Dan bersama kesukaran pasti ada kemudahan. Karena itu bila selesai suatu tugas, mulailah tugas yang lain dengan sungguh-sungguh. Hanya kepada Tuhanmu hendaknya kau berharap"

(Terjemah QS Asy-Syarh : 6 - 8)

" Apabila telah datang pertolongan Allah dan kemenangan"

(Terjemah QS AN-Nasr : 1)

'Kebahagiaan sejati adalah ketenangan jiwa'

'Keindahan yang sempurna adalah kesabaran'

'Kenikmatan yang hakiki adalah rasa syukur'

(Jalaluddin Rumi)

KATA PENGANTAR

Assalamu'alaikum Warahmatullahi Wabarakatuh

Alhamdulilah, penulis ucapkan puji syukur kepada Allah SWT yang telah memberikan rahmat, hidayah, serta karunia-Nya, sehingga laporan Tugas Akhir dapat penulis selesaikan. Tidak lupa shalawat serta salam kami ucapkan kepada junjungan nabi Allah Nabi Muhammad SAW, beserta para keluarga dan sahabatnya yang telah membawa kita dari zaman kegelapan menuju zaman terang benderang.

Tugas Akhir ini dibuat sebagai salah satu syarat yang harus dipenuhi untuk memperoleh gelar sarjana di Jurusan Teknik Informatika Universitas Islam Indonesia. Adapun Tugas Akhir kami mengenai "*Speech To Text* Untuk Bahasa Indonesia."

Pelaksanaan Tugas Akhir ini merupakan salah satu mata kuliah wajib dari jurusan Teknik Informatika Fakultas Teknologi Industri Universitas Islam Indonesia dan juga merupakan sarana bagi penulis untuk menambah wawasan serta pengalaman dalam menerapkan keilmuan, sesuai dengan yang dibambil di bangku perkuliahan.

Oleh, karena itu, pada kesempatan ini penulis ingin menyampaikan rasa terima kasih kepada:

1. Orang tua dan keluarga penulis atas segala do'a dan dukungan selama penulis melakuakn Tugas Akhir.
2. Bapak Hendrik, S.T., M.Eng., selaku Ketua Jurusan Teknik Informatika Fakultas Teknologi Industri Universitas Islam Indonesia.
3. Bapak Dr. Raden Teduh Dirgahayu, S.T., M.Sc. selaku Ketua Studi Program Sarjana Teknik Informatika Fakultas Teknologi Industri Universitas Islam Indonesia.
4. Bapak Ahmad Fathan Hidayatullah, S.T., M.Cs. selaku Dosen Pembimbing Tugas Akhir di Jurusan Teknik Informatika Fakultas Teknologi Industri Universitas Islam Indonesia.
5. Ibu Chanifah Indah Ratnasari, S.Kom., M.Kom. yang telah memberikan ilmu, nasihat dan ide penelitian kepada penulis.
6. Segenap keluarga besar teman-teman di Fakultas Teknologi Industri terutama dari Jurusan Teknik Informatika Universitas Islam Indonesia yang telah memberikan bantuan dan dukungannya.

7. Bapak Dr. Raden Bagus Fajriya Hakim, S.Si., M.Si. selaku Ketua Jurusan Statistika Fakultas Matematika dan Ilmu Pengetahuan Alam yang telah memberikan izin kepada penulis untuk menggunakan *server* Statistika.
8. Hafizhan Aliady selaku *admin server* Statistika yang sudah banyak membantu penulis dalam memakai *server* Statistika.
9. Semua pihak yang telah banyak membantu penulis dalam pelaksanaan Tugas Akhir yang tidak dapat penulis sebutkan satu persatu.

Penulis menyadari bahwa laporan ini masih belum sempurna, karena keterbatasan kemampuan dan pengalaman di lapangan. Oleh karena itu, penulis mengharapkan kritik dan saran yang membangun demi kesempurnaan Laporan Tugas Akhir ini. Akhir kata, penulis berharap agar laporan ini dapat bermanfaat bagi semua pihak.

Wassalamu'alaikumm Warahmatullahi Wabarakatuh

Yogyakarta, 16 Agustus 2018

(Teguh Puji Laksono)

SARI

Speech to text merupakan suatu teknologi yang dapat mengubah data suara menjadi data *text.* Hal tersebut memungkinkan komputer dapat mengerti bahasa manusia melalui perintah suara. Pada prosesnya, data suara yang digunakan pada *speech to text* diubah terlebih dahulu menjadi data numerik sehingga komputer dapat membacanya. Data numerik tersebut nantinya diproses sehingga komputer dapat menerjemahkannya menjadi kata. Hal tersebut dilakukan menggunakan *deep learning* sebagai metodenya.

Deep learning dipilih karena kemampuan pembelajarannya yang dapat seperti manusia. Hal tersebut menjadikan *deep learning* menjadi metode yang tepat untuk *speech to text. Deep learning* sendiri adalah metode dari jaringan syaraf yang menggunakan *multi layer perseptron* sebagai *layer* pembelajarannya. *Layer* tersebut mempunyai *neuron* yang berjumlah banyak sehingga dapat menghasilkan pembelajaran yang lebih dalam. Pada penelitian ini dilakukan *speech to text* menggunakan *deep learning* pada data suara Bahasa Indonesia.

Hasil pengujian *speech to text* menunjukkan akurasi 65% pada data Bahasa Indonesia dan 57% pada data Bahasa Jawa. Pada data uji 50 kata. Hal tersebut dapat disimpulkan *speech to txt* dengan metode *deep learning* dapat digunakan sebagai solusi dari mengubah rekaman suara menjadi *text.*

Kata kunci: *speech to text, neural network, deep learning, multi layer perseptron*

GLOSARIUM

Error	Kekeliruan, ketidaktepatan atau kesalahan yang dapat disebabkan oleh *software* atau perangkat lunak, *hardware* atau perangkat keras, dan *human error* yang berarti kesalahan dikarenakan pengguna.
Input	Semua data dan perintah yang dimasukkan ke dalam komputer melalui perangkat keras dan perangkat lunak untuk selanjutnya diproses lebih lanjut oleh komputer.
Output	Hasil yang dikeluarkan oleh komputer dari proses komputasi dan ditampilkan pada monitor.
File	Suatu dokumen pada komputer yang berupa *text, audio, video* ataupun *image.*
Debugging	Sebuah metode yang dilakukan oleh para pemrogram dan pengembang perangkat lunak untuk mencari dan mengurangi *bug*, atau kerusakan di dalam sebuah program komputer atau perangkat keras sehingga perangkat tersebut bekerja sesuai dengan harapan.
Bug	Kesalahan yang terjadi pada program komputer.
IDE	Program komputer yang memiliki beberapa fasilitas yang diperlukan dalam pembangunan perangkat lunak.
Artificial Intelligence	Bagian dari ilmu komputer yang mempelajari bagaimana membuat mesin (komputer) dapat melakukan pekerjaan seperti dan sebaik yang dilakukan oleh manusia.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Speech to text adalah suatu metode yang dapat mengubah suara kedalam bentuk tulisan (Herlim, 2002). Hal tersebut memungkinkan komputer dapat mengerti bahasa manusia melalui perintah suara (Khilari & Bhope, 2015). Pada prosesnya, data suara yang digunakan pada *speech to text* diubah terlebih dahulu menjadi data numerik sehingga komputer dapat membacanya. Data numerik tersebut nantinya diproses sehingga komputer dapat menerjemahkannya menjadi kata (Deng et al., 2013).

Pada implementasinya *speech to text* sudah mulai digunakan pada beberapa bidang seperti: penerjemah, rekam medis, pendukung keputusan dan lain-lain (Khilari & Bhope, 2015). Pada bidang-bidang tersebut *speech to text* digunakan sebagai pengganti masukkan berupa *text* langsung, sehingga dapat memudahkan dalam menggunakan aplikasi-aplikasi yang digunakan pada bidang tersebut (Suryadharma et al., 2014). Salah satu dari aplikasi yang memanfaatkan metode *speech to text* adalah penerjemah yang digunakan oleh google. Pada aplikasi tersebut sistem dapat menerima masukkan berupa suara sehingga sistem dapat langsung menerjemahkan kata yang diucapkan oleh seseorang dan mengubahnya menjadi bahasa lain (Abadi et al., 2016).

Penelitian pada *speech to text* sudah banyak dilakukan pada berbagai bahasa seperti: Bahasa Inggris, Bahasa Jepang, Bahasa Arab, Bahasa Indonesia dan lain-lain (Areni & Bustamin, 2017; Chamidy, 2016; Hotta, 2011). Pada penelitian tersebut digunakan jumlah data suara yang berbeda-beda. Hal tersebut dikarenakan ketersediaan data suara yang terbatas (Mon & Tun, 2015). Pada Bahasa Indonesia, data suara yang bersifat terbuka untuk *speech to text* masih sedikit. Sehingga penelitian pada Bahasa Indonesia pada *speech to text* belum banyak dilakukan (Chamidy, 2016).

Pada speech to text terdapat metode yang digunakan seperti: *Gaussian Mixture Models* (GMM), *Deep Neural Network* (DNN), *Recurrent Neural Network* (RNN), *Hidden Markov Model* (HMM), dan lain sebagainya (Khilari & Bhope, 2015; Shatkay, 1999). Metode yang sering digunakan adalah *Hidden Markov Model* (HMM), metode ini dipilih karena pada implementasinya digunakan prinsip probabilitas yaitu mencari kemungkinan kata pada setiap kata pada daftar kata yang dibuat (Wicaksono & Purwarianti, 2010). Kelebihan metode ini adalah komputasi yang dilakukan tidak memakan *resource* yang besar. Tetapi pada metode ini

jika data yang digunakan besar maka hasil yang didapat menjadi kurang tepat karena memungkinkan terjadinya kesamaan atau kemiripan probabilitas antar katanya (Sidiq et al., 2015) Sedangkan pada metode *neural network* dapat mengatasi masalah pada data yang besar. Hal tersebut dikarenakan *neural network* menggunakan prinsip statistik, yaitu dengan mengambil nilai numerik dari data suara dan melakukan perhitungan dengan nilai tersebut. Akan tetapi kekurangan pada metode ini adalah komputasi yang besar (Deng et al., 2013). Namun, metode *neural network* lebih baik untuk hasil yang didapat dari pada HMM, karena pada perkembangan teknologi, data yang harus dianalisis semakin meningkat dan spesifikasi komputer juga mengalami perkembangan. Sehingga masalah komputasi yang besar dapat diatasi (LeCun et al., 2015).

Berdasarkan latar belakang diatas, pada penelitian ini dilakukan *speect to text* untuk Bahasa Indonesia dengan menggunakan pendekatan *neural network* yaitu *deep learning*. *Deep learning* dipilih karena pada *deep learning* pembelajaran yang dilakukan lebih dalam sehingga keluaran yang dihasilkan diharapkan dapat lebih baik (Schmidhuber, 2015).

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, ada beberapa rumusan masalah, yaitu:

a. Bagaimana membangun *Speech to Text* untuk data Bahasa Indonesia dan Bahasa Jawa?
b. Bagaimana mendapatkan *text* dari data suara?

1.3 Batasan Masalah

Dalam penelitian ini dilakukan 2 batasan pekerjaan, yaitu:

a. Data yang dipakai adalah suara dalam Bahasa Indonesia dan Bahasa Jawa.
b. Hasil dari *Speech to Text* bentuk satu kata yang berhubungan dengan kamus medis yang dibuat.

1.4 Tujuan Penelitian

Adapun tujuan dari penelitian ini adalah membangun *Speech to Text* untuk mengambil *text* pada rekaman suara Bahasa Indonesia dan Bahasa Jawa.

1.5 Manfaat Penelitan

Manfaat yang dihasilkan oleh penelitian ini adalah berupa suatu sistem *Speech to Text* untuk pengambilan *text* pada rekaman suara.

1.6 Metode Penelitian

Pada penelitian ini digunakan metode yaitu: *Mel Frequency Cepstral Coefficient* (MFCC), *Deep Learning (Deep Neural Network)* dan *Connectionist Temporal Classification* (CTC). *Mel Frequency Cepstral Coefficient* (MFCC) adalah suatu metode untuk mengekstrak ciri pada sebuah suara sehingga didapatkan sebuah nilai ciri dari sebuah suara tersebut.

Kelebihan dari MFCC adalah pada *mel-frequency* yaitu suatu metode untuk ekraksi ciri dengan mengadaptasi pendengaran manusia dengan memfilter secara linier dibawah 1000Hz dan secara logaritmik pada frekuensi di atas 1000Hz (Sidiq et al., 2015). Sedangakan *Deep Learning* adalah suatu algoritma pembelajaran mesin atau *machine learning* dengan menggunakan suatu jaringan atau pembelajaran secara berulang-ulang dengan memanfaatkan basis pengetahuan yang dibuat oleh MFCC (Wang, 2003).

Connectionist Temporal Classification (CTC) adalah suatu metode pada pemodelan bahasa dalam *neural network* yang digunakan sebagai penentu atau sebagai representasi dari pembelajaran yang dilakukan dalam *neural network* tersebut (Graves, 2012).

1.7 Sistematika Penulisan

Sistematika penulisan penelitian ini disusun untuk memberikan gambaran umum tentang penelitian yang dijalankan. Sistematika penelitian ini adalah sebagai berikut:

BAB I PENDAHULUAN

Pada bagian ini berisi tentang latar belakang permasalahan, identifikasi masalah, menentukan batasan masalah yang akan dibahas, tujuan dan manfaat dari penetilan, asumsi metodologi penelitian, dan sistematika penulisan

BAB II LANDASAN TEORI

Pada bagian ini menjelaskan teori-teori yang digunakan di dalam penelitian. Setiap teori yang digunakan di dalam penelitian akan dijabarkan di bagian ini.

BAB III METODOLOGI PENELITIAN

Pada bagian ini menampilkan metode yang akan digunakan untuk mengumpulkan data penelitian. Penelitian ini akan menggunakan *Mel Frequency Cepstral Coefficient* (MFCC), *Deep Learning (Deep Learning Network)* dan *Connectionist Temporal Classification* (CTC).

BAB IV HASIL DAN PEMBAHASAN

Pada bagian ini menjelaskan bagaimana kata pada Bahasa Indonesia dan Bahasa Jawa didapatkan pada proses *Speech to Text*.

BAB V KESIMPULAN DAN SARAN

Pada bagian ini akan dijabarkan kesimpulan yang didapatkan setelah menyelesaikan penelitian. Selain itu juga diberikan saran yang memberikan poin-poin yang dapat ditingkatkan untuk penelitian selanjutnya.

BAB II
LANDASAN TEORI

2.1 Tinjauan Pustaka

Penelitian yang mengambil tema tentang *speech to text* dengan berbagai bahasa sudah banyak dilakukan sebelumnya. Beberapa penelitian yang terkait dengan *speech to text*, MFCC, *Connectionist Temporal Classification* (CTC) dan *Deep Learning* dibahas lebih lanjut pada bab ini.

Hotta (2011) dalam penelitiannya mengusulkan dan mencoba untuk melakukan *speech to text* terhadap kata homonin pada *speech* Bahasa Jepang. Penelitian ini melakukan pengambilan makna dari kosa kata bahasa Jepang dari kosa kata yang sama seperti: "*hashi*" yang berarti "*chop-sticks*" dan "*hashi*" yang berarti "*bridge*". Pada penelitian ini tidak digunakan *pitch extraction* tetapi menggunakan *accent model* sebagai *phoneme label*. Metode yang digunakan dalam mengambil ekstraksi ciri pada penelitian ini yaitu dengan menggunakan MFCC. Dalam analisisnya penelitian ini menggunakan metode HMM dengan akurasi yang didapatkan 89%.

Areni & Bustamin (2017) dalam penelitiannya mengusulkan dan mencoba untuk melakukan *speech to text* terhadap kata homofon pada *speech* Bahasa Indonesia. Penelitian ini melakukan pengambilan makna dari kosa kata Bahasa Indonesia pada kosa kata yang memiliki suara pengucapan yang sama. Pada penelitian ini digunakan dua jenis data yaitu data *training* dan data validasi. Data yang digunakan yaitu diambil dari 6 narasumber yang terdiri dari 3 laki-laki dan 3 perempuan. Metode yang digunakan dalam mengambil ekstraksi ciri pada penelitian ini yaitu MFCC. Dalam analisisnya penelitian ini menggunakan metode *backpropagation* dengan akurasi 53.33% untuk data laki-laki dan 36.8% untuk data perempuan.

Chamidy (2016) dalam penelitiannya mengusulkan dan mencoba untuk melakukan *speech to text* terhadap penuturan kosa kata bahasa Arab berdasarkan dialek yang berbeda. Penelitian ini melakukan kecocokan suara dengan *text* Bahasa Arab yang terdapat pada Al-Qur'an. Data yang digunakan adalah 3000 contoh suara dengan 150 contoh suara yang diperoleh dari 5 penutur asli Indonesia yang tidak terlalu fasih Bahasa Arab. Metode yang digunakan dalam mengambil ektraksi ciri pada penelitian ini yaitu MFCC. Dalam analisisnya penelitian ini menggunakan metode HMM dengan akurasi 83.1% pada data sampling di frekuensi 8000Hz, 82.3% pada data sampling di frekuensi 22050Hz dan 82.2% pada data sampling di frekuensi 44100Hz.

Deng et al. (2013) dalam penelitiannya mengusulkan dan mencoba untuk melakukan analisis *speec*h pada Microsoft menggunakan *deep learning*. Penelitian ini dilakukan sebagai perbaikan penelitian *speech* sebelumnya. Penelitian *speech* di Microsoft sebelumnya menggunakan metode *Hidden Markov Model with Gaussian Mixture Emissions* (GMM-HMM) kemudian diperbaharui menggunakan metode *Deep Neural Network* (DNN) atau lebih dikenal sebagai *deep learning.* Data yang digunakan menggunakan beberapa data suara dari beberapa bahasa seperti: Bahasa Prancis, Jerman, Spanyol, dan Italia. Masing-masing dataset tersebut mempunyai durasi yaitu: 138 jam, 195 jam, 63 jam, dan 93 jam. Berdasarkan percobaan tersebut *error* yang didapatkan lebih rendah dari sebelumnya. Pada percobaan menggunakan GMM-HMM didapatkan error sebesar 23% sedangkang menggunakan DNN menurun menjadi 13%.

Dahl et al. (2012) dalam penelitiannya mengusulkan dan mencoba untuk melakukan analisis *speech* pada *Large Vocabulary Speech Recognition* (LVSR) menggunakan *Context Dependent-Deep Neural Network-Hidden Markov Model* (CD-DNN-HMM). Penelitian ini dilikukan sebagai pembanding terhadap metode *Deep Neural Network-Hidden Markov Model* (DNN-HMM) yang sudah dilakukan sebelumnya. Pada penelitian ini juga dibandingkan komputasi yang dilakukan pada kedua metode tersebut. Akurasi yang dihasilkan pada percobaan ini meningkat dari 5.8% pada DNN-HMM menjadi 9.2% dan error yang diberikan menurun dari 23.3% menjadi 16.0%.

Graves et al. (2006) dalam penelitiannya mengusulkan dan mencoba untuk melakukan analisis *speech* pada TIMIT *speech corpus* dengan menggunakan metode *hybrid Hidden Markov Model-Recurrent Neural Networks* (HMM-RNN) dengan *Connectionist Temporal Classification* (CTC) pada kasus pelabelan tidak tersegmentasi. Data yang digunakan berasal dari TIMIT dengan data suara Bahasa Inggris. Hasil yang didapatkan dari percobaan tersebut adalah nilai *Label Error Rate* (LER) yaitu: 33.84 ± 0.06 % dari metode HMM-RNN dengan 0.06% adalah standar error dan jika menggunakan CTC menjadi 30.51 ± 0.19 % dengan 0.19% adalah standar *error*. Gambaran perbandingan beberapa penelitian mengenai *Speech to Text* dan *Deep Learning* diperlihatkan pada Tabel 2.1.

Tabel 2.1 Perbandingan Penelitian

No.	Penulis	Objek	Fokus Penelitian	Metode
1.	Hotta (2011)	*Speech* Bahasa Jepang	Kata homonin pada *speech* bahasa Jepang	*Hidden Markov Model (HMM)*
2.	Areni & Bustamin (2017)	*Speech* Bahasa Indonesia	Kata homofon pada *speech* Bahasa Indonesia	Backpropagation
3.	Chamidy (2016)	*Speech* Bahasa Arab	Penuturan kata berdasarkan dialek	*Hidden Markov Model (HMM)*
4.	Deng et al. (2013)	*Speech* Bahasa Prancis, Jerman, Spanyol, dan Italia	Perbandingan akurasi	*Hidden Markov Model with Gaussian Mixture Emissions (GMM-HMM) dan Deep Neural Network (DNN)*
5.	Dahl et al. (2012)	*Large Vocabulary Speech Recognition (LVSR)*	Analisis *speech* pada pidato	*Context Dependent-Deep Neural Network-Hidden Markov Model* (CD-DNN-HMM) dan *Deep Neural Network-Hidden Markov Model* (DNN-HMM)
6.	Graves et al. (2006)	TIMIT *speech corpus*	Perbandingan akurasi	*Hidden Markov Model-Recurrent Neural Networks* (HMM-RNN)

2.2 *Speech to Text*

Speech to Text adalah suatu teknologi yang memungkinkan komputer dapat menerima masukan berupa kata yang diucapkan dan kemudian diterjemahkan menjadi suatu data yang dimengerti oleh komputer (Herlim, 2002). Masukan berupa suara mampu diubah menjadi *teks* yang mampu dibaca. Suara/bahasa yang diterima oleh komputer nantinya dicocokkan dengan suatu pola tertentu sehingga suara/bahasa tersebut dapat dikenali oleh komputer. Tulisan atau *teks* yang dihasilkan selanjutnya dapat diolah oleh komputer untuk digunakan pada kebutuhan lain. Teknologi ini juga memungkinkan komputer dapat berkomunikasi dengan manusia dengan menggunakan bahasa alami (Suryadharma et al., 2014).

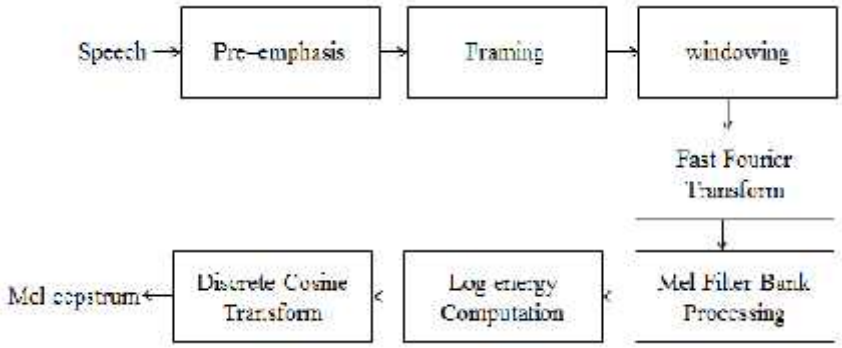
Pada implementasi *Speech to Text* terdapat beberapa disiplin ilmu yang digunakan, diantaranya yaitu: Pemrosesan Sinyal, Aljabar Linier, Probabilitas, *Linguistic*/Ilmu Bahasa, dan Ilmu Komputer. Beberapa disiplin ilmu tersebut digunakan sebagai dasar pembentukan *Speech to Text* karena dalam pengolahan suara yang menjadi masukan adalah suatu sinyal digital sehingga membutuhkan komputasi matematika untuk mengenali pola dari suara tersebut. Secara umum tahapan dalam *Speech to Text* yaitu: masukan dalam bentuk *file* suara atau dari *microphone* akan didigitalisasi oleh komputer sehingga akan menghasilkan suatu pola tertentu dan akan dicocokkan dengan pola yang ada untuk mencari bahasa dari pola tersebut (Suryadharma et al., 2014).

Pada *Speech to Text* terdapat beberapa pendekatan/metode dalam proses analisis pola suara/bahasa ke dalam bentuk tulisan/teks. Beberapa pendekan/metode yang sering digunakan diantaranya yaitu: *Gaussian Mixture Models* (GMM), *Deep Neural Network* (DNN), *Recurrent Neural Network* (RNN), *Hidden Markov Model* (HMM), dan lain sebagainya (Khilari & Bhope, 2015; Shatkay, 1999). Terdapat beberapa penelitian *Speech to Text* yang sudah dilakukan. Arini dan Bustamin melakukan *Speech to Text* untuk menemukan *homophone* pada Bahasa Indonesia (Areni & Bustamin, 2017). Noertjahyana dan Adipranata melakukan pengenalan suara dengan aplikasi SAPI 5.1 dan Delphi 5 (Noertjahyana & Adipranata, 2004). Mon dan Tun melakukan *Speech to Te*xt *convertion* dengan menggunakan *Hidden Markov Model* (HMM) (Mon & Tun, 2015). Wijaya membuat aplikasi pada *Android* yang dapat mengenali suara dalam Bahasa Indonesia dan mengubahnya menjadi tulisan (WIJAYA, SUSANTO, & Galih Salman, 2013).

2.3 *Mel Frequency Cepstral Coefficient* (MFCC)

Mel Frequency Cepstral Coefficient (MFCC) adalah suatu cara dalam mengambil suatu ciri dari sebuah gelombang suara dan mengubahnya menjadi beberapa tipe representatif yang

dapat diproses lebih lanjut (Sidiq et al., 2015). Metode ini mulai diperkenalkan oleh Davis dan Mermelstein di tahun 1980an sebagai koefisiensi yang mempresentasikan suatu gelombang suara (Chamidy, 2016). Pada MFCC untuk menandai proses ekstraksi ciri digunakan pengubahan data suara menjadi data citra berupa spektrum gelombang. Kebanyakan sistem yang memanfaatkan ekstrasi suara menggunakan MFCC sebagai *feature* karena sistem pengenal suara atau *Speech Recognition* menjadi lebih presisi dalam berbagai kondisi (Sidiq et al., 2015). Diagram alur ekstraksi ciri dari MFCC dapat dilihat pada Gambar 2.1.



Gambar 2.1 Ekstraksi Ciri MFCC

2.3.1 *Pre-emphasis*

Pada langkah ini dilakukan filtering terhadap sinyal menggunakan *FIR filter orde* satu untuk meratakan spektral sinyal tersebut. Proses ini mencakup penambahan energi suara pada frekuensi tinggi. Secara matematis, dapat dirumuskan seperti pada persamaan (2.1).

$$S_p(n) = S(n) - 0{,}97_s.(n-1) \qquad (2.1)$$

Di mana $S_p(n)$ adalah sinyal yang ditekan, sedangkan S(n) adalah sinyal terdigitasi. Koefisien dengan nilai 0.97 menunjukkan sinyal yang diekstrak merupakan 97% sinyal aslinya.

2.3.2 *Frame Blocking (Tracking) dan Windowing*

Pada langkah ini sinyal ucapan yang telah teremphasis dibagi menjadi beberapa *frame* dengan masing-masing *frame* memuat N sampel sinyal dan *frame* yang saling yang berdekatan dipisahkan sejauh M sampel. Menyusun sinyal ke dalam bingkai yang lebih pendek. Panjang *frame* yang membagi setiap sampel menjadi beberapa *frame* tersebut berdasarkan waktu yang terletak di antara 20 hingga 40ms. Dengan asumsi bahwa frekuensi suara adalah 18 kHz, maka sampel yang diekstrak adalah 0,025 detik * 18.000 Hz = 450 sampel.

Windowing merupakan proses pembobotan terhadap setiap *frame* yang telah dibentuk pada langkah sebelumnya. Proses ini menggunakan fungsi *Window*. Ada dua fungsi *window*

yang biasa digunakan, yaitu *Rectangular Window* dan *Hamming Window*. *Rectangular Window* yang didefinisikan pada persamaan (2.2).

$$W = \begin{cases} 1 & 0 \le n \le N \\ 0 & L \end{cases} \quad (2.2)$$

Fungsi *window* ini menghasilkan sinyal yang diskontinyu. Salah satu cara untuk menghindari diskontinyu pada ujung *window* adalah dengan meruncingkan sinyal menjadi nol atau dekat dengan nol sehingga dapat mengurangi kesalahan. Fungsi *Hamming* digunakan seperti bentuk jendela dengan mempertimbangkan blok berikutnya dalam rantai pemrosesan ekstraksi fitur dan memadukan semua garis frekuensi terdekat. Fungsi *Hamming Window* di definisikan pada persamaaan (2.3).

$$W_n = \begin{cases} 0{,}54 - 0{,}46 \cdot \cos(2\pi \ /(N-1)) & 0 \le n \le N \\ 0 & L \end{cases} \quad (2.3)$$

Setelah itu, dikalikan dengan hasil persamaan *Hamming Window* dengan sinyal masukan/input yang telah ditetapkan, perhitungan tersebut dijelaskan pada persamaan (2.4).

$$Y(n) = y(n) . w(n) \quad (2.4)$$

N = Banyaknya Sample Tiap Frame

Y(n) = Sinyal Output

y(n) = Sinyal Input

w(n) = Hamming Windows

2.3.3 *Fast Fourier Transform* (FFT)

Pada langkah ini setiap *frame* hasil dari fungsi *window* dikenai proses FFT. Fungsi FFT digunakan untuk mengubah sinyal yang semula merupakan *time domain* menjadi *frequency domain*. Langkah ini mengubah tiap *frame* N sampel dari domain waktu ke dalam domain frekuensi. Dalam pengolahan suara, *Fast Fourier Transform* (FFT) berguna untuk mengubah konvolusi getaran celah suara dan respon gelombang saluran suara dalam domain waktu.

2.3.4 *Mel-scale* dan *Filter Bank*

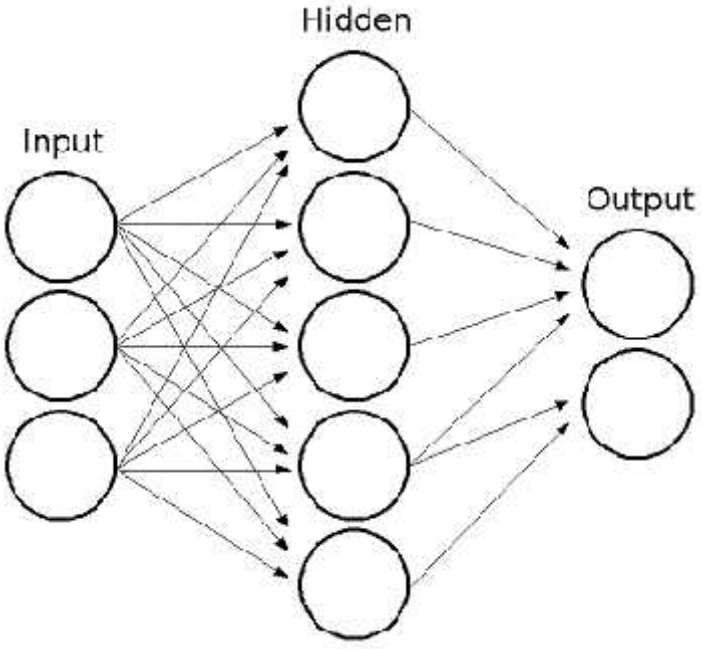
Pada tahap ini dilakukan *wrapping* terhadap spektrum yang dihasilkan dari FFT sehingga dihasilkan *Mel-scale* untuk menyesuaikan resolusi frekuensi terhadap properti pendengaran manusia. Kemudian *Mel-scale* dikelompokkan menjadi sejumlah *critical bank* menggunakan *filter bank*. Jangkauan frekuensi dalam spektrum sangatlah luas dan sinyal suara tidak

mengikuti skala linear. Sehingga setelah spektrum terkomputasi, data dipetakan dalam skala Mel menggunakan filter segitiga yang saling tumpang tindih.

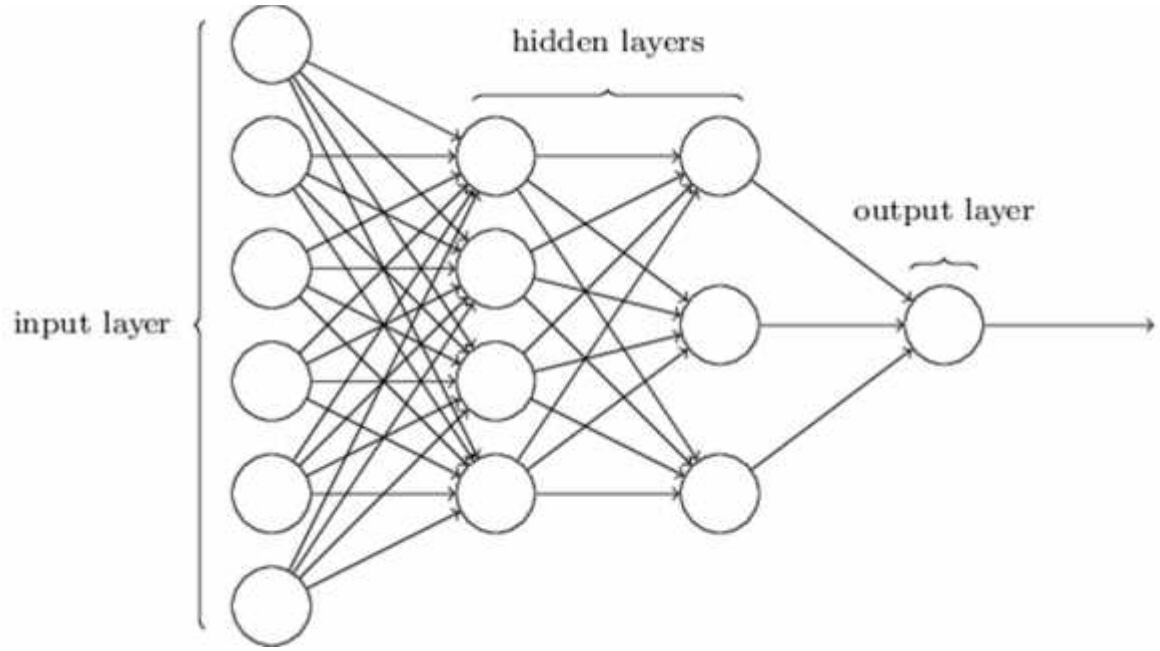
2.4 *Deep Learning (Deep Neural Network)*

Deep Neural Network atau lebih dikenal dengan *Deep Learning* adalah suatu cabang dari *machine learning* yang menggunakan konsep jaringan syaraf tiruan (*Neural Network*) (Schmidhuber, 2015). *Neural Network* adalah suatu model yang terinspirasi dari jaringan syaraf yang bekerja pada otak manusia (Goodfellow et al., 2016). Pada otak manusia informasi sampai ke otak melalui *neuron-neuron* yang saling terhubung (LeCun et al., 2015). Pada *Neural Network* setiap *neuron* menerima *input* kemudian *neuron* tersebut melakukan suatu komputasi dengan bobot dan bias sehingga menghasilkan fungsi aktifasi. Fungsi aktifasi terus diperbaharui dengan *input* yang masuk sehingga menghasilkan suatu nilai *output* (Goodfellow et al., 2016).

Fungsi aktifasi pada *Neural Network* memiliki fungsi menentukan sebuah *neuron* aktif atau tidak berdasarkan bobot dari *input* (Wang, 2003). Fungi aktifasi memiliki banyak jenis, seperti: *linier, sigmoid, tanh, ReLU*, dan lain-lain (Hagan et al., 1996). *Neural Network* memiliki dua arsitektur yaitu: *single layer perseptron* dan *multi layer perseptron*. Arsitektur tersebut dapat dilihat pada Gambar 2.2 dan Gambar 2.3



Gambar 2.2 *Single Layer Perseptron*



Gambar 2.3 *Multi Layer Perseptron*

Pada *single layer perseptron* terdapat satu *hidden layer* yang digunakan sedangkan pada *multi layer perseptron* digunakan lebih dari dua *hidden layer* yang digunakan et al., 2016). Pada *Deep Neural Network* atau *Deep Learning* jumlah *hidden layer* yang digunakan bisa sangat banyak, sehingga hasil yang didapat semakin akurat (He et al., 2016). Salah satu *tools* yang dapat digunakan untuk menjalankan *Deep Learning* adalah *Tensorflow*.

2.5 *Tensorflow*

Tensorflow adalah *tools* yang dikembangkan oleh *Google* untuk melakukan komputasi *Deep Learning* dan kode program yang disediakan bersifat bebas (*Open Source*) (Abadi et al., 2016). Lisensi yang terdapat pada *tensorflow* adalah lisensi *Apache 2.0* sehingga *tensorflow* dapat digunakan oleh siapapun termasuk kompetitor *Google* sendiri. Pada *tensorflow* terdapat banyak fungsi yang dapat digunakan untuk *machine learning* diantara yaitu: *image analytic, speech to text, video analytic* dan lain-lain. *Tensorflow* dapat berjalan pada GPU sebagai komputasi paralel.

2.6 *Connectionist Temporal Classification* (CTC)

Connectionist Temporal Classification (CTC) adalah suatu metode pada pemodelan bahasa dalam *neural network* yang digunakan sebagai penentu atau sebagai representasi dari pembelajaran yang dilakukan dalam *neural network* tersebut (Graves, 2012). Pada CTC diambil nilai dari pembelajaran yang dihasilkan oleh perhitungan *neural network* sehingga menghasilkan suatu nilai yang mempresentasikan kata atau huruf yang terdapat pada data yang

digunakan seperti pada pengenalan tulisan ataupun deteksi suara seperti *speech to text* (Graves et al., 2006).

Nilai CTC pada *neural network* memiliki nilai yang terus berkurang dengan semakin banyaknya *layer* pembelajaran yang digunakan (Huang et al., 2016). Hal ini juga dapat disebut dengan istilah CTC *Loss*. Semakin kecil nilai CTC *Loss* maka akurasi dari pembelajaran *neural network* tersebut menjadi semakin baik. Tetapi jika nilai CTC *Loss* ini terlalu kecil maka dapat menyebabkan *Overfitting* (Hawkins, 2004). *Overfitting* adalah suatu keadaan di mana proses pembelajaran berjalan melebihi data yang diberikan sehingga pembelajaran juga dapat menggunakan *noise* sebagai pembelajarannya. Hal ini berakibat *output* atau keluaran yang dihasilkan menjadi kosong (Cawley & Talbot, 2010). Proses yang dilakukan pada pengambilan CTC *Loss* pada data audio dapat dilihat pada Gambar 2.4.

Gambar 2.4 Proses Pengambilan CTC *Loss*

BAB III
METODOLOGI

3.1 Analisis Kebutuhan

Analisis kebutuhan merupakan tahapan di mana dilakukan pengumpulan data dan informasi untuk mendukung dan menunjang dalam pembuatan sistem sebagai jawaban dari rumusan masalah. Metode yang digunakan dalam analisis kebutuhan adalah dengan studi pustaka. Dalam studi pustaka informasi didapatkan dengan membaca literatur dalam bentuk buku, jurnal ilmiah, penelitian sebelumnya dan juga dapat menggunakan internet. Informasi yang dibutuhkan dalam membangun sistem *speech to text* adalah proses pengambilan data suara, *preprocessing* yaitu berupa ekstraksi ciri, *training, testing* dan lain-lain.

3.1.1 Analisis Fungsional

Analisis fungsional menjelaskan tentang kebutuhan fungsional yang dikembangakan oleh sistem sebagai jawaban dari rumusan masalah. Adapun kebutuhan fungsional sistem ini diantaranya:

a. Sistem dapat melakukan *preprocessing* terhadap data suara dan data label.
b. Sistem dapat melakukan *training* terhadap nilai MFCC dan label.
c. Sistem dapat melakukan *testing* terhadap model yang sudah dibuat.

3.1.2 Analisis *Input*

Data *input* merupakan data yang digunakan sebagai masukan pada sistem *speech to text* yang dibangun. Data yang digunakan antara lain:

a. Data yang digunakan adalah data suara Bahasa Indonesia dan Bahasa Jawa.

3.1.3 Analisis *Output*

Data *output* merupakan data hasil keluaran dari sistem *speech to text* yang dibangun. Adapun data *output* nya adalah:

a. Bentuk *text* dari suara yang digunakan sebagai masukan.

3.1.4 Analisis Kebutuhan Perangkat Lunak

Penelitian ini menggunakan beberapa perangkat lunak sebagai pendukung dalam pengerjaan sistem *speech to text.* Adapun perangkat lunak tersebut adalah:

a. Ubuntu 16.04 LTS.
b. Windows 10 Pro.
c. Voice Recorder Windows.
d. Libav-tools.
e. FFMPEG.
f. Python 2.7.
g. Sugartensor 1.0.0.2.
h. Tensorflow-gpu 1.7.0.

3.1.5 Analisis Kebutuhan Perangkat Keras

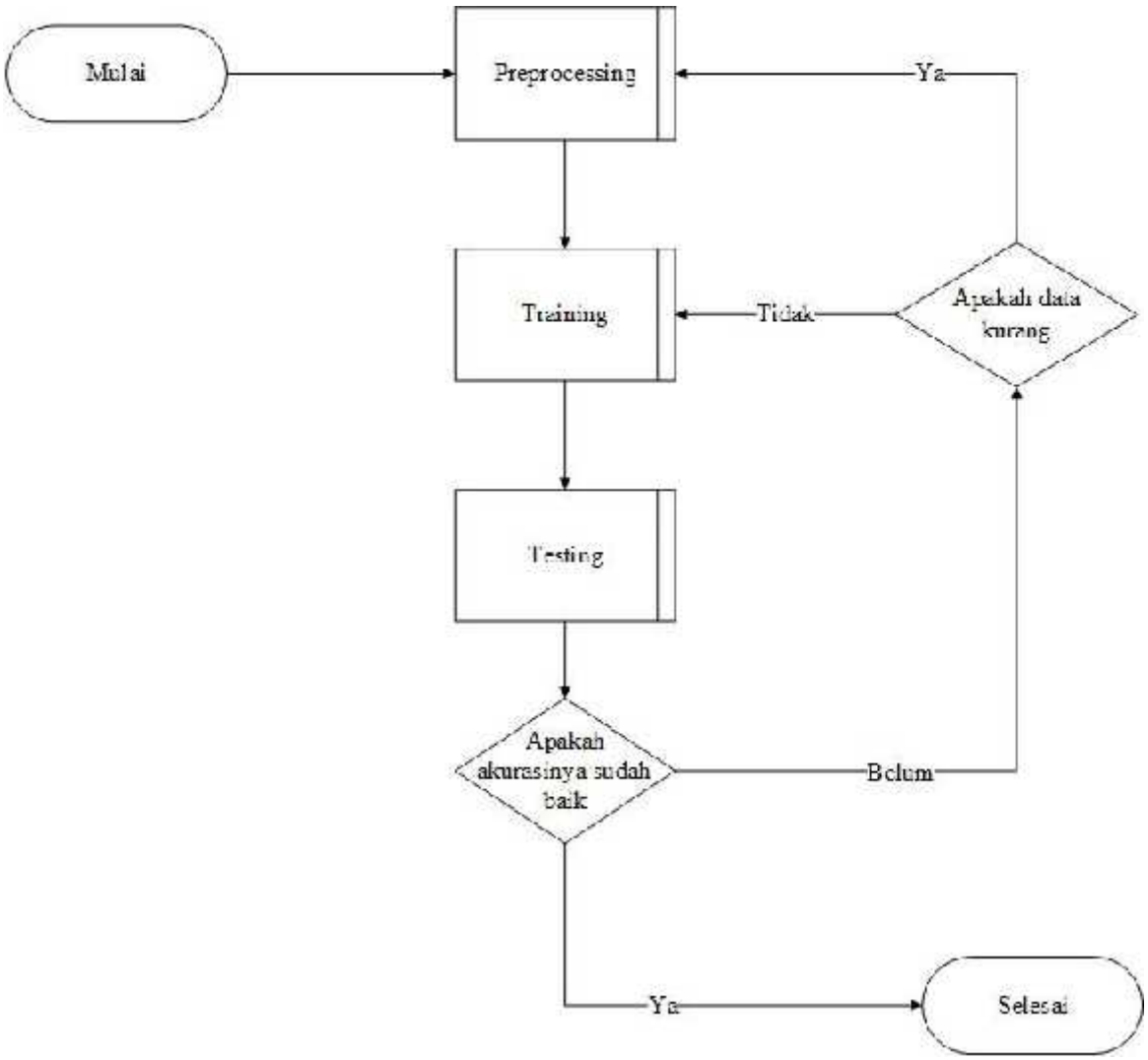
Penelitian ini menggunakan beberapa perangkat keras sebagai pendukung dalam pengerjaan sistem *speech to text.* Adapun perangkat keras tersebut adalah:

a. NVIDIA Tesla P100-PCIE-16GB.
b. Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz.
c. RAM 64 GB.
d. Microphone SF-920 Noise Reduction.
e. Headset Sades SA-901 Wolfang.

3.2 Perancangan

Pada penelitian ini dilakukan beberapa tahapan untuk membangun *speech to text.* Tahapan tersebut dilakukan secara manual dan otomatis melalui sistem. Pada tahap manual dilakukan sebelum membangun *speech to text*. Hal ini dilakukan sebagai persiapan untuk membangun *speech to text*. Tahap tersebut adalah pengambilan data, melabel suara, dan mengganti tipe *file.* Sedangkan untuk tahap utama dalam *speech to text* dilakukan secara otomatis menggunakan bahasa pemrograman.

Speech to text pada penelitian ini memiliki 3 tahapan utama yang dilakukan untuk mendapatkan hasil dalam bentuk data *text* dari masukan data suara. Tahapan tersebut yaitu *preprocessing*, *training* dan *testing*. Pada tahap *preprocessing* dilakukan pemrosesan dari data suara dan label untuk diambil nilai MFCC *feature* dan nilai label. Kemudian pada tahap *training* dilakukan perhitungan dari nilai MFCC *feature* dan nilai label untuk diproses menjadi model dengan metode *deep learning*. Pada tahap terakhir yaitu *testing* dilakukan pengujian pada model untuk diambil akurasi dan kategori dari data uji. Tahapan tersebut dapat dilihat pada Gambar 3.1.



Gambar 3.1 *Flowchart Speech to Text*

3.2.1 Pengambilan Data

Pada tahap pengambilan data dilakukan proses rekaman suara pada beberapa narasumber. Narasumber akan diberikan 50 daftar kata. Daftar kata tersebut berisikan kata dalam Bahasa Indonesia dan beberapa dalam Bahasa Jawa yang sering digunakan. Narasumber diminta membacakan daftar tersebut satu per satu sehingga didapatkan satu kata menjadi satu *file* suara. Proses pengambilan data ini dilakukan menggunakaan *microphone noise reduction* dengan keadaan sekitar yang tenang sehingga suara yang didapatkan tidak terdapat *noise* atau suara yang mengganggu. Hal ini juga dapat dikategorikan sebagai tahap *preprocessing.*

3.2.2 Melabel Suara

Setelah data suara didapatkan maka tahap selanjutnya adalah melabeli *file* suara tersebut dengan kata-kata yang terdapat pada daftar kata. Pada label suara juga terdapat informasi narasumber yang disertakan seperti jenis kelamin. Satu *file* suara yang disapatkan terdapat satu

kata yang harus dilabeli. Setiap *file* suara akan diberi *id* untuk memudahkan mencocokan dengan *file* label dari suara tersebut. Daftar label kata yang digunakan dapat dilihat pada Tabel 3.1.

Tabel 3.2 Daftar Kata

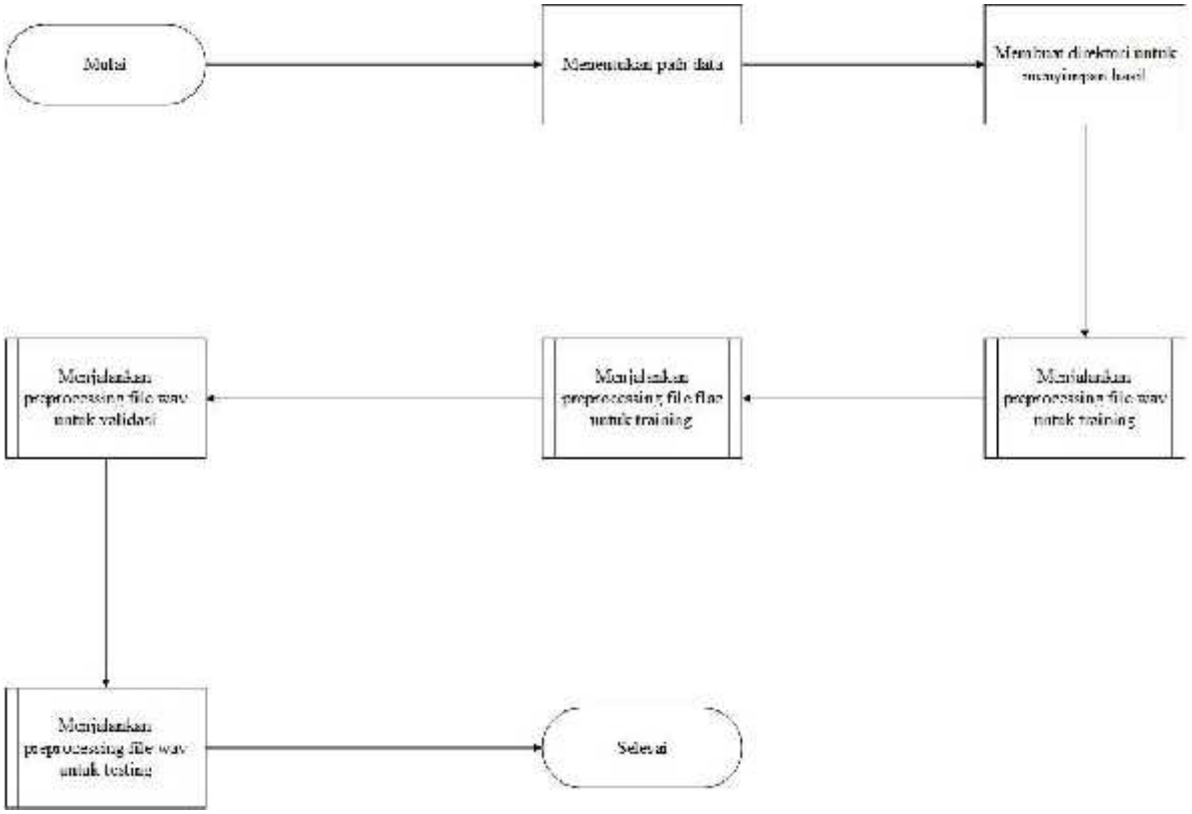
No.	Kata	No.	Kata
1.	Batuk	26.	Ngilu
2.	Bersin	27.	Pegal
3.	Pusing	28.	Berdarah
4.	Demam	29.	Bernanah
5.	Gatal	30.	Memar
6.	Pilek	31.	Mimisan
7.	Mual	32.	Lemas
8.	Muntah	33.	Lesu
9.	Kembung	34.	Sembelit
10.	Perih	35.	Kejang
11.	Kram	36.	Kaku
12.	Sesak	37.	Pingsan
13.	Sesek	38.	Kesemutan
14.	Radang	39.	Flu
15.	Nyeri	40.	Gemetar
16.	Sariawan	41.	Sakit
17.	Panas	42.	Melepuh
18.	Dingin	43.	Bentol
19.	Kedinginan	44.	Kesel
20.	Diare	45.	Watuk
21.	Meler	46.	Enek
22.	Bengkak	47.	Puyeng
23.	Benjol	48.	Meriyang
24.	Luka	49.	Mumet
25.	Alergi	50.	Nggreges

3.2.3 Mengganti Tipe *File*

Setelah *file* suara tersebut dilabeli maka selanjutnya adalah *file* suara tersebut diganti tipe *file*-nya menjadi tipe *file wav* dan *flac*. Hal ini diperlukan untuk tahap selanjutnya yaitu pada tahap *training.* Tipe *file* dari file suara juga diganti untuk menyeragamkan tipe *file* sehingga dapat dengan lebih mudah dalam proses *training*. Perangkat lunak yang digunakan dalam proses ini adalah *libav-tools*. Perangkat lunak ini berjalan pada sistem operasi windows, linux ataupun mac, tetapi dalam penelitian ini digunakan sistem operasi linux.

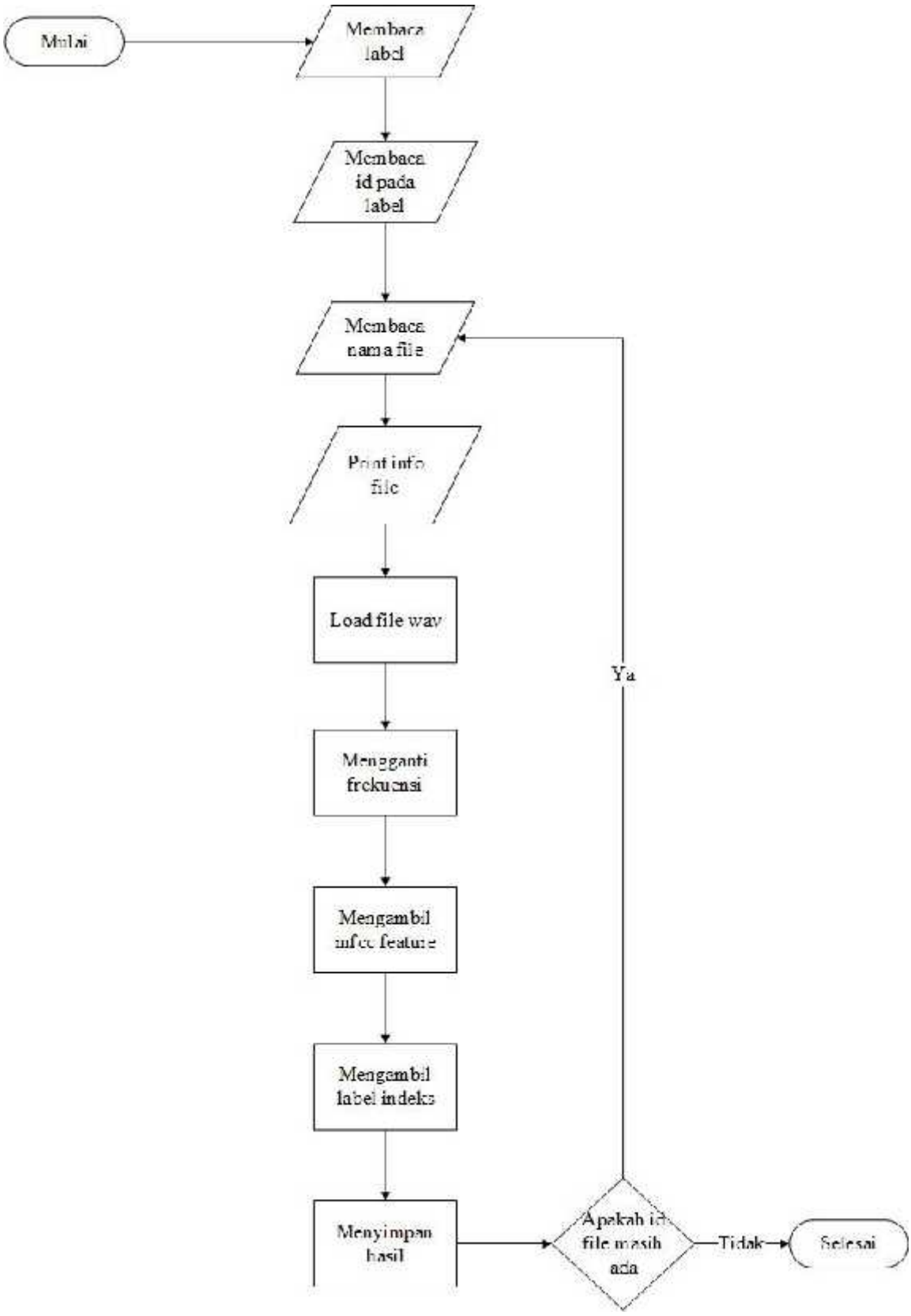
3.2.4 *Preprocessing*

Pada tahap *preprocessing*, *training* dan *testing* digunakan Bahasa pemrograman *python* dengan versi 2.7. *Flowchart* pada tahap *preprocessing* dapat dilihat pada Gambar 3.2.



Gambar 3.2 *Flowchart Preprocessing*

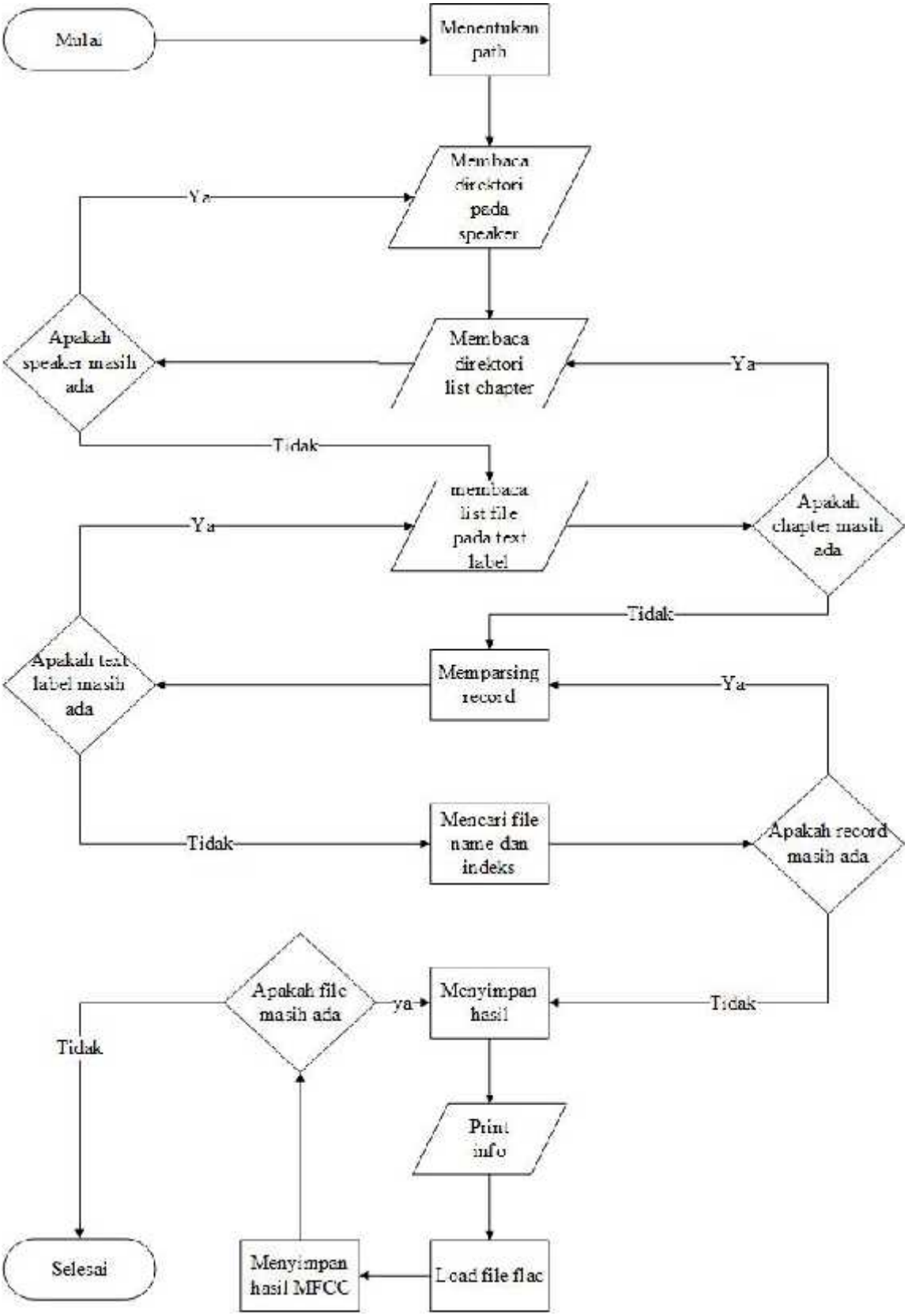
Pada *flowchart* tersebut proses dimulai dengan menentukan *path* data yang nantinya diambil MFCC *feature* nya. Kemudian membuat direktori tempat menyimpan hasil dari MFCC *feature* dari tahap *preprocessing* ini. Selanjutnya dilakukan fungsi untuk mengambil MFCC *feature* pada *file wav* yang nantinya digunakan sebagai data *training.* Pada *file flac* juga dilakukan pengambilan MFCC *feature* untuk digunakan pada tahap *training*, validasi dan juga *testing. Flowchart* tahap *preprocessing* pada *file wav* dapat dilihat pada Gambar 3.3.



Gambar 3.3 *Flowchart Preprocessing Wav*

Pada *flowchart* tersebut dimulai dengan membaca label yang *path* nya sudah ditentukan pada awal *preprocessing*. Kemudian dilakukan pembacaan *id* pada label. Setelah id didapatkan maka *id* tersebut disamakan dengan *id* pada *file* suara. Informasi tersebut ditampilan pada monitor sehingga jika terdapat *error* maka dapat dicari dengan mudah.

Kemudian tahap selanjutnya adalah *load file wav* yang nantinya diambil MFCC *feature* nya dan juga frekuensi pada file tersebut diganti. Setelah file di *load* maka file tersebut dapat diambil MFCC *feature* nya beserta label indeksnya. Kemudian informasi tersebut disimpan untuk tahap selanjutnya. Proses ini dilakukan sampai semua *file* suara selesai diambil MFCC *feature* nya. Sedangakan pada *file flac* dapat dilihat pada Gambar 3.4.



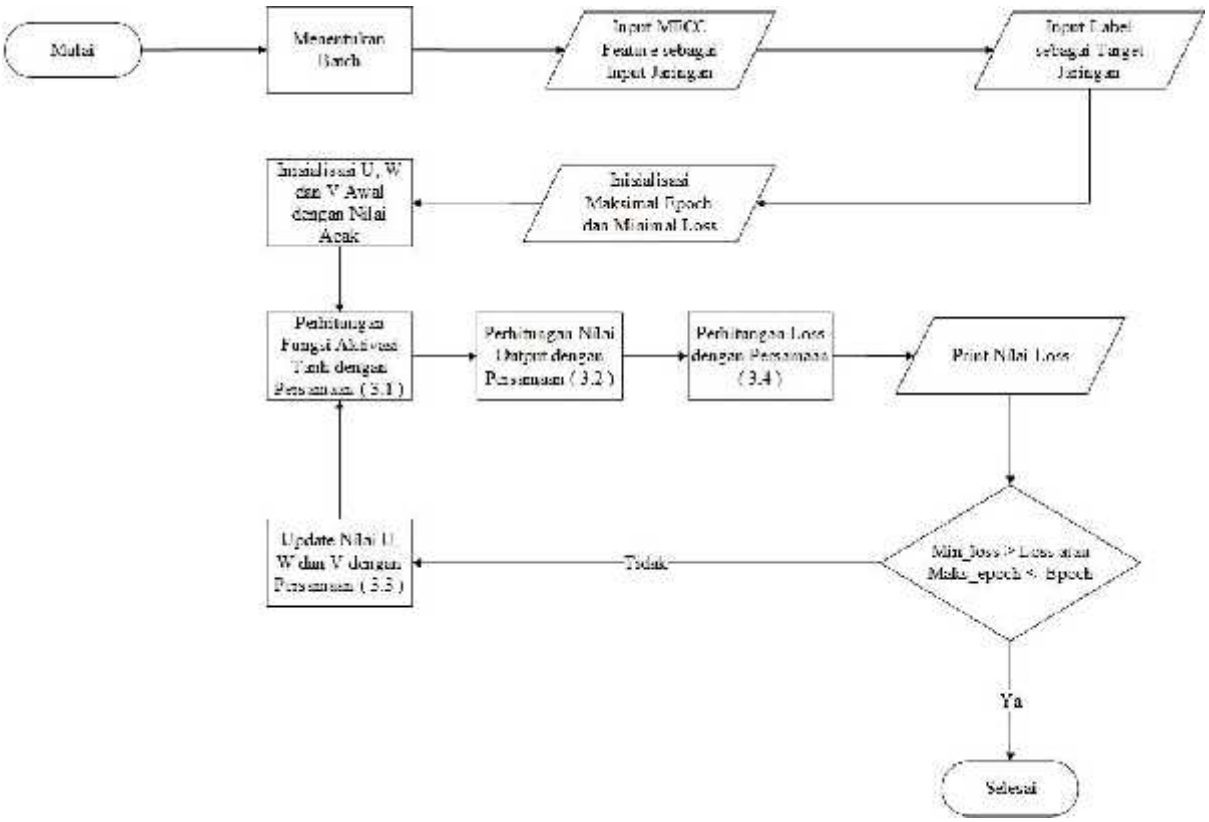
Gambar 3.4 *Flowchart Preprocessing Flac*

Pada fungsi *preprocessing file flac* juga dimulai dengan menentukan *path file* suara yang nantinya diambil MFCC *feature* nya. Kemudian dilakukan pembacaan direktori yang terdapat pada *speaker* pada label *file*. Proses ini terus dilakukan sampai semua *speaker* terbaca. Kemudian dilanjutkan pembacaan *chapter* pada label *file*. Proses ini juga dilakukan sampai semua *chapter* terbaca. Kemudian dilakukan pembacaan *list file* pada label *file*. Proses ini dilakukan samapai *list file* terbaca semua. Jika sudah terbaca semua maka selanjutnya adalah memparsing *record* yang terdapat pada label *file*. Proses ini dilakukan sampai record terparsing semua. Setelah pembacaan *id* pada label tersebut selesai dilakukan maka selanjutnya dicocokan dengan *id* pada *file* suara yang nantinya diambil MFCC *fearure* nya.

Tahap selanjutnya adalah menampilkan info dari proses tersebut sehingga jika terjadi *error* maka dapat dengan mudah dicari. Selanjutnya hasil MFCC *feature* tersebut disimpan. Hal ini dilakukan sampai semua *file* suara diambil MFCC *feature* nya.

3.2.5 *Training*

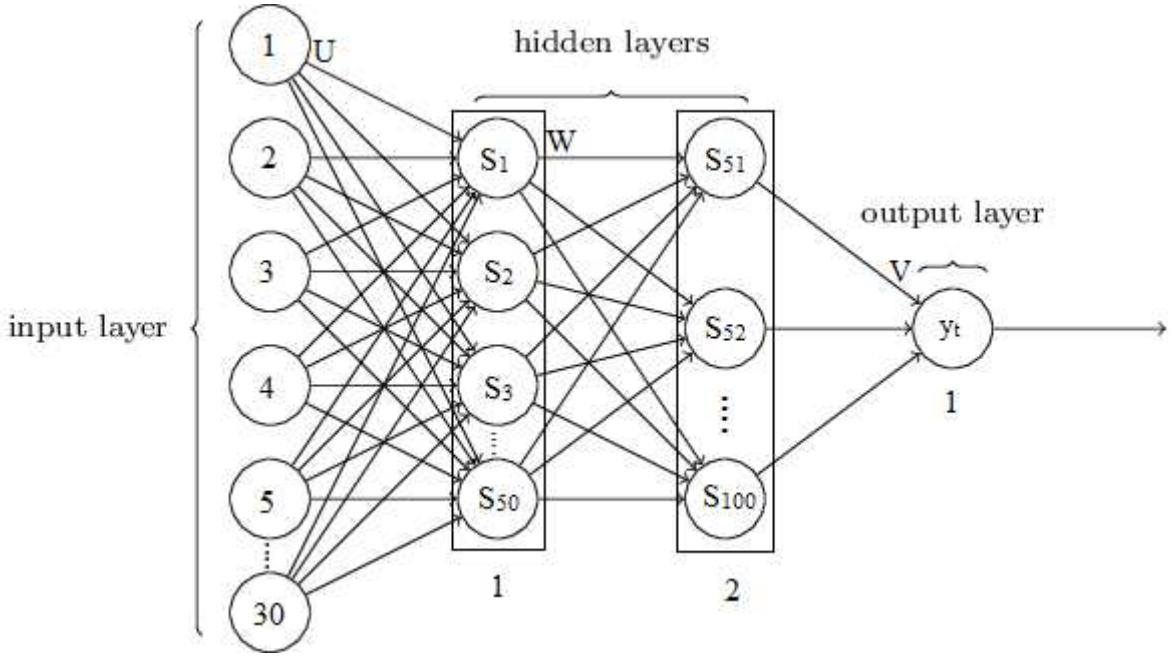
Pada tahap training dilakukan proses *machine learning* dengan metode *deep learning*. Tahap ini dilakukan dengan nilai MFCC *feature* yang sudah diambil dari proses sebelumnya. Pada proses training ini juga dilakukan pengambilan nilai CTC *Loss* yang dapat mempresentasikan akurasi dari *training* yang dilakukan. Untuk *flowchart* tahap *training* dapat dilihat pada Gambar 3.5.



Gambar 3.5 *Flowchart Training*

Pada tahap *training* dimulai dengan menentukan *batch file*, yaitu jumlah file yang masuk ke dalam perhitungan ke *deep learning* sebagai bahan dari pembelajaran, *batch file* ini dapat disebut juga pembagi data, misal terdapat 10 data dan *batch* nya adalah 2 maka data yang masuk ke dalam *deep learning* adalah berjumlah 5 sekaligus.

Selanjutnya adalah memasukan nilai dari MFCC *feature* yang sudah didapatkan pada tahap sebelumnya. Kemudian juga memasukkan label untuk proses pembelajaran. Tahap selanjutnya dalah inisialisasi *learning rate*, maksimal epoch dan minimum loss yang digunakan pada *neural netwok*. Selanjutnya inisialisasi bobot yang dipakai pada proses perhitungan pada neural network. Kemudian dilakukan perhitungan dengan metode *deep learning* dengan nilai *input* MFCC *feature* dan target adalah label. Tahap terakhir dihitung nilai *Loss* dan menampilkannya. Hal tersebut dilakukan sampai nilai *Loss* mencapai minimum atau nilai epoch mencapai maksimun. Pada *deep learning* digunakan *multi layer perseptron* sebagai *layer* pembelajaran. *Multi layer* tersebut menggunakan *hidden layer* dengan jumlah *neuron* yang banyak. Arsitektur jaringan yang digunakan pada tahap ini dapat dilihat pada Gambar 3.6.



Gambar 3.6 Arsitektur Jaringan

Pada jaringan tersebut digunakan data sebanyak *batch file* sebagai *neuron input* yang sudah ditentukan yaitu 30 *neuron*. *Input* tersebut berisi nilai MFCC *feature* dengan tipe data array yang berisi nilai numerik. Sedangkan untuk banyaknya *neuron output* adalah 1 yaitu berisi nilai numerik dalam bentuk array yang mempresentasikan label. Nilai tersebut diproses melalui *hidden layer* dengan banyaknya *hidden layer* yaitu 2 dengan setiap *hidden layer* mempunyai *neuron* sebanyak 50. Perhitungan pada jaringan tersebut adalah dengan menggunakan fungsi aktifasi tanh seperti pada persamaan (3.1).

$$S_t = \tanh(U.x_t + W.S_{t-1}) \quad (3.1)$$

S_t = *Neuron* ke t

U = Bobot *Input*

x_t = *Input Neuron*

W = Bobot Antar *Hidden Layer*

Setelah perhitunagn tersebut nantinya dicari nilai *output* untuk dihitung nilai *loss/error* nya. Perhitungan output tersebut dapat dilihat pada persamaan (3.2).

$$y_t = \text{softmax}(V.S_t) \quad (3.2)$$

y_t = *Output*

V = Bobot *Output*

Nilai U, W, dan V pada persamaan (3.1) dan (3.2) merupakan nilai yang diambil secara acak, tetapi untuk membatasi nilainya maka digunakan rentang seperti pada persamaan (3.3).

$$[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}] \quad (3.3)$$

n = Jumlah Koneksi yang Baru Masuk dari Layer Sebelumnya

Nilai S pada persamaan (3.1) diperbaharui melalui *hidden layer* pada jaringan syaraf. Setelah mendapatkan nilai *output* maka selanjutnya dihitung *loss/error* dengan rumus pada persamaan (3.4).

$$L_t(y, y_t) = -\mathrm{y} \log y_t \quad (3.4)$$

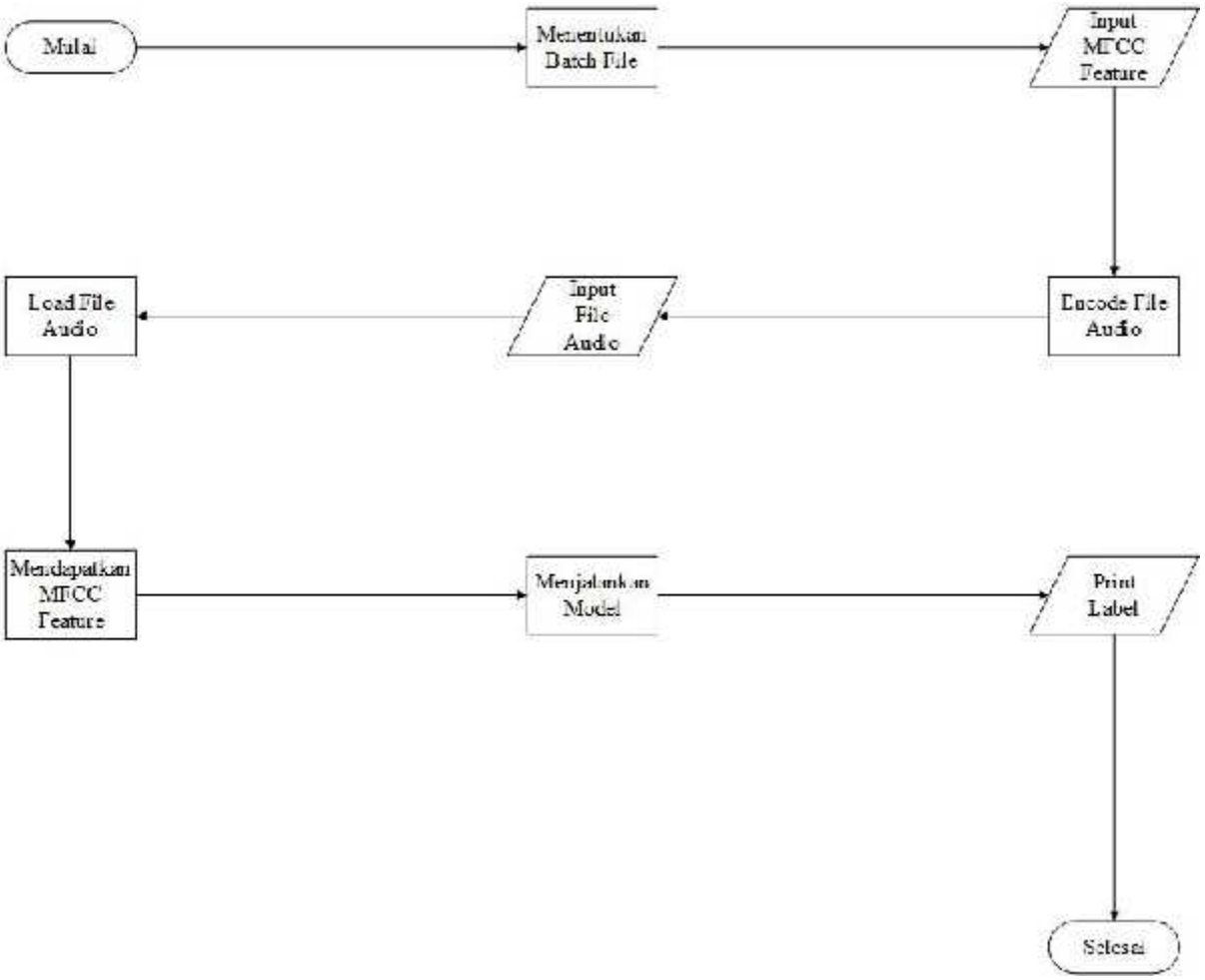
L_t = *Loss/error*

y = Label/Target

y_t = *Output* Jaringan

3.2.6 *Testing*

Pada tahap *testing* dilakukan percobaan dengan data baru yang nantinya dikenali oleh *speech to text* berdasarkan model yang sudah dibuat. Pada tahap ini dapat dilihat kecocokan kata yang dihasilkan oleh *speech to text* dengan suara yang terdapat pada *file* rekaman. *Flowchart* tahap *testing* dapat dilihat pada Gambar 3.7.



Gambar 3.7 *Flowchart Testing*

Pada tahap *testing* dimulai dengan menentukan *batch file* untuk dilakukan proses pengenalan terhadap suara tersebut sama halnya dengan tahap *training*, namun *batch file* untuk *testing* yaitu berjumlah satu karena *file* yang dicoba untuk dikenali adalah satu. Kemudian dilakukan *input* MFCC *feature* dnari basis pengetahuan. Selanjutnya meng-*encode file* suara tersebut. Kemudian memasukkan *file audio* ke dalam model dan juga di *load*. Selanjutnya dilakukan *input* MFCC *feature* dari *file* baru yang nantinya untuk *testing*. Selanjutnya adalah menjalankan model yang sudah dibuat pada tahap t*raining*.

Terakhir adalah menampilkan hasil pembelajaran dari model tersebut. Pada proses *testing* ini dapat dilihat akurasi yang dihasilkan oleh model, jika akurasi masih belum memenuhi maka diulang pada tahap *training* atau jika data yang digunakan belum memenuhi maka diulang ke tahap pengambilan data.

BAB IV
HASIL DAN PEMBAHASAN

4.1 Implementasi

Implementasi adalah tahap selanjutnya dari tahap perancangan yang sudah direncanakan pada tahap sebelumnya. Pada tahap ini dapat menunjukan perancangan yang sudah dibuat sebelumnya dapat berjalan dengan lancar dan sesuai dengan perancangan. Pada implementasi ini dilakukan tahapan-tahapan yang sudah direncanakana sebelumnya, yaitu: pengambilan data, melabel suara, mengganti tipe *file*, mengambil MFCC *feature*, *training*, dan *testing*

4.1.1 Implementasi Pengambilan Data

Pada pengambilan data ini dilakukan pada 10 Narasumber dengan jenis kelamin laki-laki, dengan usia antara 20 tahun sampai 22 tahun. Pada proses ini narasumber diminta membacakan daftar kata yang sudah diberikan satu per satu. Daftar kata tersebut berjumlah 50 kata yang sering digunakan. Daftar kata tersebut sudah dijelaskan pada tahap perancangan. Pengambilan suara tersebut dilakukan menggunakan perekam suara atau *microphone* dan menggunakan perangkat lunak perekam suara yang tersedia pada sistem operasi *windows 10 pro*. Perekaman dilakukan dengan pembacaan kata pada daftar kata satu per satu sehingga mendapatkan 50 data suara pada setiap narasumber. Setiap data suara rata-rata berdurasi sekitar 2 detik. Setelah data suara diambil maka data suara tersebut nantinya diubah namanya berdasarkan *id* yang digunakan dalam proses selanjutnya.

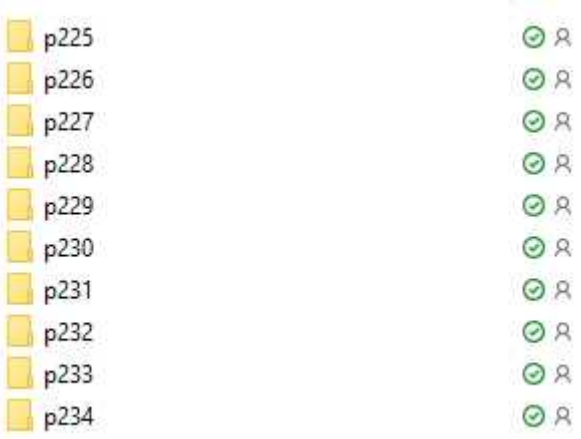
4.1.2 Implementasi Melabel Suara

Pada implementasi melabel suara ini dilakukan 2 tahap, yaitu melabel suara pada data yang akan digunakan pada tipe *file wav* dan melabel suara pada data yang akan digunakan pada tipe *file flac*. Pada format pelabelan suara pada data *wav* digunakan format yaitu satu *file* suara memiliki satau *file* label dalam ekstensi *txt*. Dengan disimpan pada direktori *txt*, sedangkan untuk *file audio*-nya disimpan pada direktori *wav48*. Pada pelabelan ini juga dibuat *file speaker* yang berisikan informasi tentang narasumber seperti *id*, umur, jenis kelamin, dan bahasa yang digunakan. Sedangkan pada data *flac* label disimpan pada satu *file* dengan tipe *file txt* yang berisi daftar kata secara keseluruhan dan diberi nama sesuai dengan *id* suara.

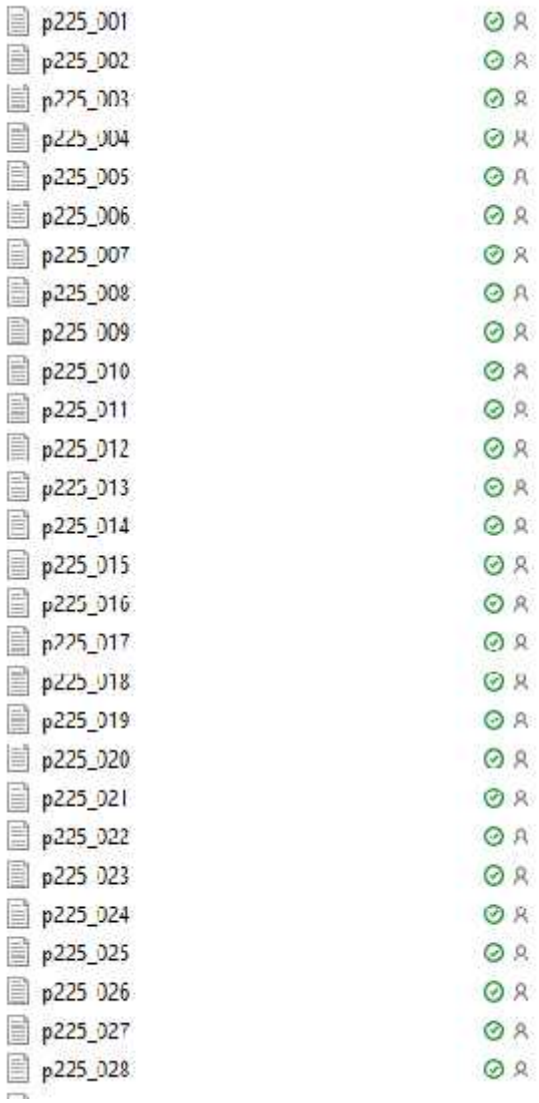
Pada data *flac* label dibagi menjadi tiga bagian yaitu *training, testing*, dan *valid*. Pada setiap bagian tersebut terdapat *file* label dan *file* suara yang ditempatkan pada satu direktori.

Pada *file flac* tidak diberikan informasi mengenai narasumber. Proses ini dilakukan secara manual atau tanpa menggunakan perangkat lunak. Hasil yang dilakukan pada proses ini dapat dilihat pada Gambar 4.1, Gambar 4.2, Gambar 4.3, Gambar 4.4, Gambar 4.5 dan Gambar 4.6.

Gambar 4.1 Direktori Label *Wav* 1



Gambar 4.2 Direktori Label *Wav* 2



Gambar 4.3 Direktori Label *Wav* 3

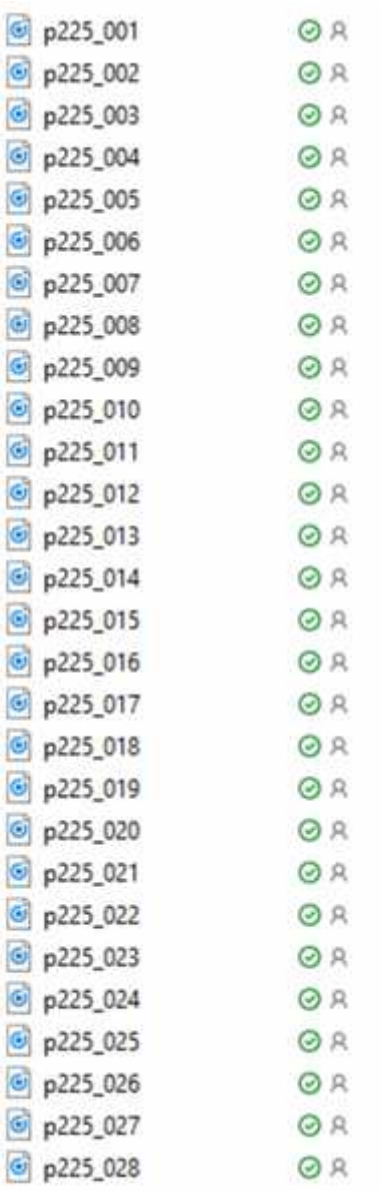
Gambar 4.4 Direktori Label *Flac* 1

Gambar 4.5 Direktori Label *Flac* 2

Gambar 4.6 Direktori Label *Flac* 3

4.1.3 Implementasi Mengganti Tipe *File*

Pada tahap ini dilakukan penggantian atau merubah tipe *file* dari data suara menjadi *file wav* dan *flac*. Hal ini dilakukan untuk memudahkan pada tahap selanjutnya. Seperti pada tahap sebelumnya data suara diubah terlebih dahulu namanya mengikuti dengan nama *file* label yang sudah dibuat sebelumnya. Nama *file* suara tersebut adalah *id* yang digunakan pada tahap selanjutnya. *Id* yang digunakan harus sama dengan *file* label yang berisi kata yang sesuai dengan suara pada *file* tesebut. Hasil dari proses di atas dapat dilihat pada Gambar 4.7 dan Gambar 4.8



Gambar 4.7 Direktori *File Wav*

Gambar 4.8 Direktori *File Flac*

Perintah yang digunakan dalam mengganti tipe *file* dari *file* suara ke *wav* seperti pada Gambar 4.9.

```
# for f in *.m4a; do avconv -i "$f" "${f/%m4a/wav}"; done
```

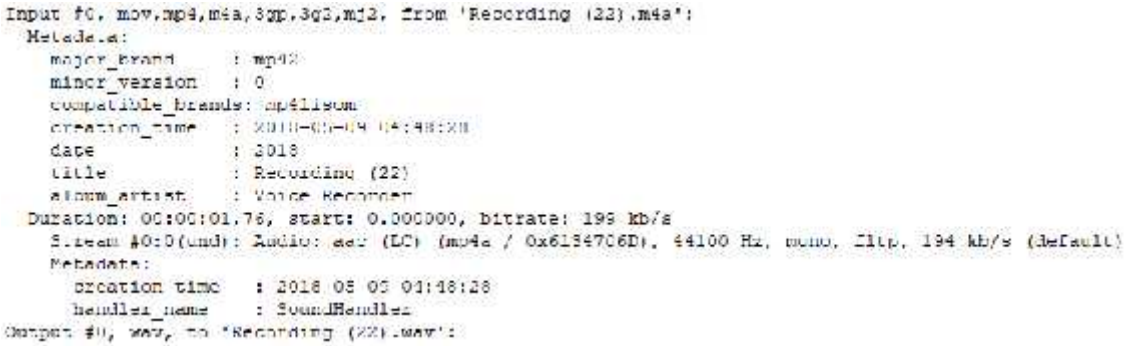
Gambar 4.9 Perintah Mengganti Tipe *File* ke *Wav*

Perintah di atas berfungsi untuk mengganti tipe *file* pada *file* suara yang didapatkan secara *batch* sehingga semua data suara yang terdapat pada satu direktori yang sama akan diganti menjadi tipe *file wav*. Sedangkan untuk mengganti ke tipe *file flac* dapat dilihat pada Gambar 4.10.

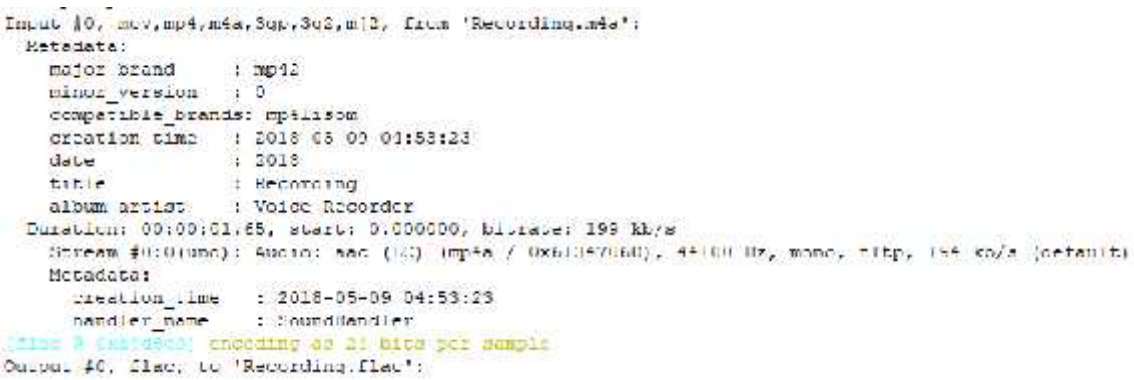
```
# for file in *.m4a; do avconv -i "$file" -f flac "`basename
"$file" .m4a`.flac"; done
```

Gambar 4.10 Perintah Mengganti Tipe *File* ke *Flac*

Perintah di atas juga berfungsi untuk mengganti tipe *file* secara *batch.* Keluaran dari perintah di atas dapat dilihat pada Gambar 4.11 dan Gambar 4.12.



Gambar 4.11 *Output File Wav*



Gambar 4.12 *Output File Flac*

4.1.4 Implementasi *Prepocessing*

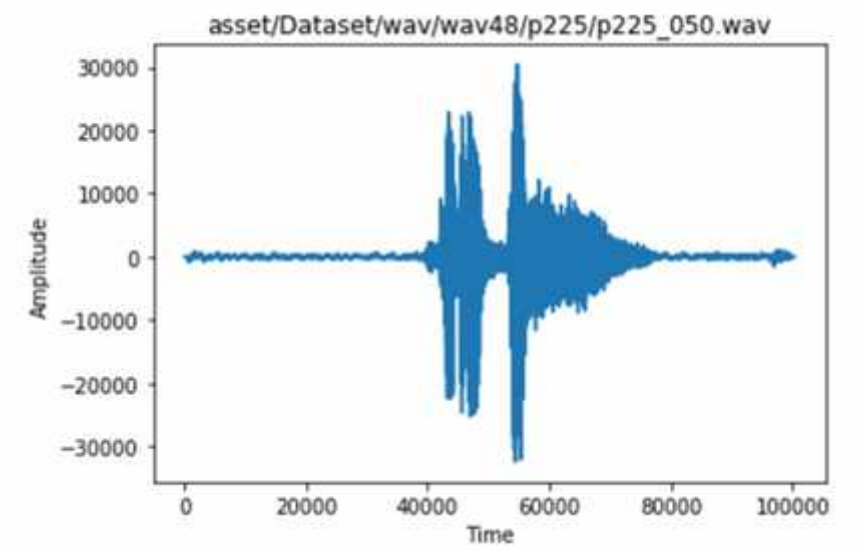
Pada tahap tahap implementasi *prepocessing* ini dilakukan pembuatan kode program dari *flowchart* yang sudah dijelaskan pada tahap sebelumnya. Pembuatan kode program ini dilakukan dengan menggunakan bahasa pemrograman *python* dengan versi 2.7. Pada *preprocessing* ini dilakukan 2 tahap yaitu *preprocessing* terhadap data *wav* data data *flac.* Kedua tahap tersebut dilakukan dengan menggunakan fungsi pada *python.* Pada proses *preprocessing* ini digunakan beberapa *library* yang terdapat pada *python* untuk memudahkan proses *preprocessing. Library* yang dipakai dapat dilihat pada Gambar 4.13.

```
import numpy as np
import pandas as pd
import glob
import csv
import librosa
import scikits.audiolab
import data
import os
import subprocess
```

Gambar 4.13 Library pada Proses Pengambilan MFCC *Feature*

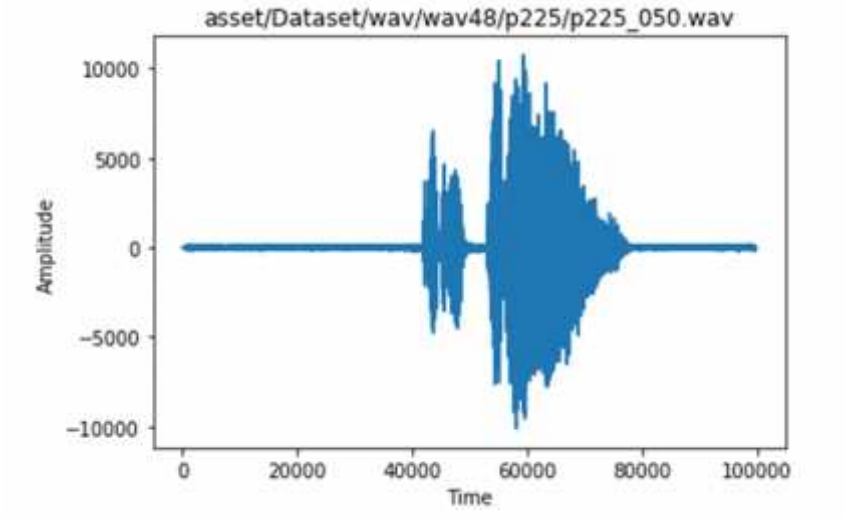
Library di atas berfungsi untuk proses pembacaan *file,* penulisan *file,* menyimpan *file* dan proses perhitungan. Penjelajasan dari *library* tersebut yaitu: *library numpy* yang digunakan sebagai pengambil nilai numerik untuk dijadikan perhitungan selanjutnya. *Library pandas* dan *csv* digunakan untuk membaca *file csv,* menulis *file csv* dan membaca *file csv. Library librosa* dan *scikits.audiolab* digunakan sebagai pembaca *file audio* dan juga sebagai pengambil nilai dari MFCC *feature*. *Library data, os* dan *subprocess* digunakan sebagai pembacaaan dataset, pembacaan *path* pada sistem operasi, mengatur *input/ouput* dari sistem, dan lain-lain. *Library* tersebut digunakan pada proses yang dilakukan pada tahap selanjutnya. Proses selanjutnya yang dilakukan setelah deklarasi *library* tersebut adalah membuat fungsi untuk mendapatkan MFCC *feature* dari *file wav*. Pada pengambilan MFCC feature dilakukan beberapa tahapan seperti yang sudah dijelaskan pada baba sebelumnya. Tahapan tersebut meliputi *pre-emphasis, frame blocking, mel-scale* dan *filter bank.*

Pada proses pengambilan nilai dari MFCC *feature* tahap pertama yang dilakukan adalah pembacaan *file* audio dengan menggunakan *library librosa*. *File* audio tersebut dapat diambil panjang gelombang atau amplitudo dan waktu rambat dari gelombang tersebut. Kedua parameter tersebut bisa disebut juga dengan frekuensi. Hal ini berguna untuk melakukan tahap selanjutnya yaitu *frame blocking.* Visualisasi dari file audio tersebut dapat dilihat pada Gambar 4.14.



Gambar 4.14 Visualisasi *File* Audio

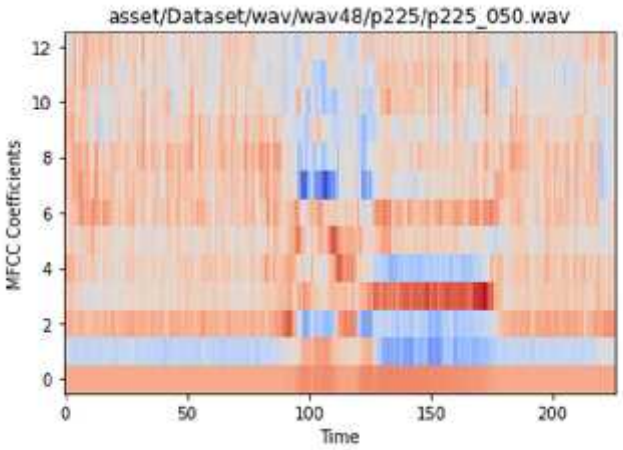
Pada gambar di atas dapat dilihat amplitudo dan waktu rambat dari gelombang tersebut. Pada *file* audio tersebut terdapat amplitudo tertinggi dan amplitudo terendah berdasarkan waktu pengucapan dari narasumber. Pada visualisasi ini sudah dilakukan pemadatan suara untuk dilihat pola suara lebih detail dikarenakan durasi data suara rata-rata adalah 2 detik. Setelah didapatkan hasil tersebut langkah selanjutnya adalah melakukan *pre-emphasis*. Tahap *pre-emphasis* dilakukan untuk menghilangkan sinyal frekuensi rendah yang cenderung tidak berubah nilainya. Hal ini berguna untuk mendapatkan frekuensi yang nilainya mengalami perubahan yang cepat. Hasil dari tahap tersebut dapat dilihat pada Gambar 4.1.



Gambar 4.15 Visualisasi *File* Audio setalah *Pre-emphasis*

Pada gambar di atas dapat dilihat terdapat perubahan dari sebelumnya. Perubahan tersebut dapat dilihat pada amplitudo yang berkurang. Perubahan lainnya terdapat pada

frekuensi awal dan akhir, dapat disimpulkan frekuensi rendah pada awal dan akhir hilang akibat dari proses *pre-emphasis* ini. Tahap selanjutnya adalah pemecahan bagian-bagian dari gelombang suara tersebut menjadi bagian kecil untuk diambil MFCC *feature*-nya. Hal ini dilakukan setiap perubahan 20ms. Hasil MFCC tersebut dapat dilihat pada Gambar 4.16.



Gambar 4.16 Visualisasi MFCC *Feature*

Berdasarkan visualisasi di atas didapatkan beberapa nilai koefisien dari MFCC yaitu dimulai dari 0 sampai 12. Koefisien tersebut didapatkan dari proses *mel-scale* dan *filter bank*. Koefisen ini berbeda-beda pada setiap waktunya, Perubahan ini digambarkan sebagai spektrum warna yang berbeda-beda. Sedangkan nilai-nilai yang dihasilkan pada visualisasi di atas dapat dilihat pada Gambar 4.17.

```
[[  8.96261601  12.77814887  12.74513413 ...  12.9533701   12.77378764
   12.73796292]
 [-12.72860106 -13.91107905 -15.87793958 ...  -9.88454827 -11.49835948
  -10.91706879]
 [  5.33893022  10.77444134  12.70626486 ...  20.81834779  17.60603699
   17.08071152]
 ...
 [  7.55972668   5.21810656   4.34737674 ... -14.86081233  -3.75643238
   -6.3881897 ]
 [ -3.45777826  -0.62304111   3.68930231 ...  -3.52282274  -1.59448202
   -6.86091379]
 [ -2.32670591 -13.16981301   2.05311679 ...   5.21959686  -4.26750084
   -7.83963545]]
```

Gambar 4.17 Nilai MFCC *Feature*

Nilai yang disebutkan di atas adalah nilai koefisien dari MFCC *feature* yang dihasilkan dari *library librosa*. Nilai tersebut selanjutnya disimpan pada *file* numerik dengan menggunakan *library numpy*. Sedangkan untuk nilai dari label yang terdapat pada *file* label

audio tersebut juga diambil sebagai data label untuk mencocokan nilai yang terdapat pada MFCC. Nilai dari label tersebut dapat dilihat pada Gambar 4.18.

```
['p225_001.wav', '3', '2', '21', '22', '12']
['p225_002.wav', '3', '6', '19', '20', '10', '15']
['p225_003.wav', '17', '22', '20', '10', '15', '8']
['p225_004.wav', '5', '6', '14', '2', '14']
['p225_005.wav', '8', '2', '21', '2', '13']
['p225_006.wav', '17', '10', '13', '6', '12']
['p225_007.wav', '14', '22', '2', '13']
['p225_008.wav', '14', '22', '15', '21', '2', '9']
['p225_009.wav', '12', '6', '14', '3', '22', '15', '8']
['p225_010.wav', '17', '6', '19', '10', '9']
['p225_011.wav', '12', '19', '2', '14']
['p225_012.wav', '20', '6', '20', '2', '12']
['p225_013.wav', '20', '6', '20', '6', '12']
['p225_014.wav', '19', '2', '5', '2', '15', '8']
['p225_015.wav', '15', '26', '6', '19', '10']
['p225_016.wav', '20', '2', '19', '10', '2', '24', '2', '15']
['p225_017.wav', '17', '2', '15', '2', '20']
['p225_018.wav', '5', '10', '15', '8', '10', '15']
['p225_019.wav', '12', '6', '5', '10', '15', '8', '10', '15', '2', '15']
```

Gambar 4.18 Nilai Label

Nilai-nilai tersebut mempresentasikan huruf yang terdapat pada file label suara. Huruf-huruf tersebut ini diambil sebagai nilai angka untuk memudahkan proses perhitungan pada tahap selanjutnya. Nilai hasil dari label ini kemudian disimpan pada *file csv*. Data di atas dibuat menjadi 3 direktori yaitu direktori training, direktori valid dan direktori testing. Masing-masing direktori tersebut berisikan *file* label dalam bentuk *file csv* dan *file* numerik dari *library numpy.* Kode program pada *preprocessing* tersebut dapat dilihat pada Lampiran B.

Proses yang pertama kali dilakukan dalam kode program tersebut adalah membuat *writer* pada *csv* yang berfungsi untuk menyimpan hasil dari pengambilan MFCC *feature* terhadap *file wav* ke *file csv*. Kemudian tahap selanjutnya adalah pembacaan label *info* yang terdapat pada *file speaker* yang berisi informasi narasumber khususnya *id*. Dilanjutkan dengan pembacaan *id* yang terdapat pada *file speaker* tersebut. Kemudian membaca nama *file* berdasarkan *id* yang sudah didapatkan pada tahap sebelumnya.

Informasi tersebut ditampilkan ke layar monitor untuk melihat apakah proses tersebut sudah benar. Selanjutnya adalah me-*load* data *file* suara ke dalam program dengan menggunakan *library librosa* dan diubah frekuensinya dari 48000Hz ke 16000Hz. Hal ini dilakukan untuk meringankan proses *preprocessing*. Kemudian proses selanjutnya adalah mengambil MFCC *feature* pada data suara berdasarkan dengan label yang sudah diambil

sebelumnya. Proses terakhir adalah menyimpan hasil MFCC *feature* ke dalam *file* numerik dengan menggunakan *library numpy*. Sedangkan f*ile* label disimpan ke dalam *file csv*. Fungsi pada pengambilan MFCC *feature* data *flac* dapat dilihat pada Lampiran C.

Pada fungsi *flac* di atas program dimulai dengan membuat *csv writer* sama seperti fungsi pada pemgambilan MFCC *feature* pada *file wav.* Yang berbeda pada fungsi ini adalah pembacaan *id,* karena pada data *flac, id* yang digunakan tidak hanya satu tetapi memiliki dua tingkat *id*. Pembacaan *id* dimulai dengan membaca *id* pada file *speaker*. Kemudian dilanjutkan dengan pembacaan *id* berdasarkan *chapter*. Selanjutnya adalah label *text* yang terdapat pada *id* tersebut. Setelah label didapatkan maka selanjutnya label tersebut diparsing untuk mendapatkan kata pada label tersebut. Setelah *id* dan label didapatkan maka tahap selanjutnya adalah mencocokan *id* dan nama file pada data suara *flac* yang nantinya diambil MFCC *feature*-nya.

Tahap terakhir adalah menyimpan keluaran. Proses penyimpanan ini dilakukan dengan me-*load* file *flac* kemudian mengambil MFCC *feature* dari *file* tersebut dan hasil itu disimpan dalam *file* numerik dengan menggunakan *library numpy*. Sedangkan label yang sudah didapatkan disimpan dalam *file csv*. Setelah kedua fungsi di atas sudah dideklarasikan maka selanjutnya memasuki tahap *preprocessing.* Kode program pada tahap *preprocessing* ini dapat dilihat Lampiran D.

Pada kode program di atas dilakukan pembuatan direktori untuk menyimpan hasil dari MFCC *feature* yang sudah didapatkan yaitu direktori mfcc dan direktori meta. Direktori tersebut berguna untuk menyimpan hasil dari MFCC dan juga file numerik dari *library numpy* Kemudian tahap selanjutnya adalah menjalan *preprocessing*. Pada data *wav* dilakukan satu kali *preprocessin*g yaitu *preprocessing* untuk data yang nantinya digunakan untuk *training.* Sedangkan pada data *flac* dilakukan tiga kali *preprocessing* yaitu: pada data *training,* data *testing* dan data *validasi.*

Ketiga proses tersebut menggunakan fungsi yang sama yaitu fungsi *process_flac*, fungsi tersebut berguna untuk mengambil MFCC *feature* pada data tersebut. Program di atas dapat dijalankan langsung dengan menggunakan perintah pada terminal linux dan juga dapat langsung dijalankan untuk proses *preprocessing*, perintah tersebut dan hasil yang didapatkan dapat dilihat pada gambar 4.19 dan Gambar 4.20.

```
root@mipa:/home/teguh/Speech-to-Text# python preprocess.py
/usr/local/lib/python2.7/dist-packages/scikits/audiolab/soundio/play.py:48: UserWarn
n building audiolab
  warnings.warn("Could not import alsa backend; most probably, "
dataset wav preprocessing (0 / 500) - 'asset/Dataset/wav/wav48/p225/p225_001.wav']
dataset wav preprocessing (1 / 500) - 'asset/Dataset/wav/wav48/p225/p225_002.wav']
dataset wav preprocessing (2 / 500) - 'asset/Dataset/wav/wav48/p225/p225_003.wav']
dataset wav preprocessing (3 / 500) - 'asset/Dataset/wav/wav48/p225/p225_004.wav']
dataset wav preprocessing (4 / 500) - 'asset/Dataset/wav/wav48/p225/p225_005.wav']
dataset wav preprocessing (5 / 500) - 'asset/Dataset/wav/wav48/p225/p225_006.wav']
dataset wav preprocessing (6 / 500) - 'asset/Dataset/wav/wav48/p225/p225_007.wav']
dataset wav preprocessing (7 / 500) - 'asset/Dataset/wav/wav48/p225/p225_008.wav']
dataset wav preprocessing (8 / 500) - 'asset/Dataset/wav/wav48/p225/p225_009.wav']
dataset wav preprocessing (9 / 500) - 'asset/Dataset/wav/wav48/p225/p225_010.wav']
dataset wav preprocessing (10 / 500) - 'asset/Dataset/wav/wav48/p225/p225_011.wav']
dataset wav preprocessing (11 / 500) - 'asset/Dataset/wav/wav48/p225/p225_012.wav']
dataset wav preprocessing (12 / 500) - 'asset/Dataset/wav/wav48/p225/p225_013.wav']
dataset wav preprocessing (13 / 500) - 'asset/Dataset/wav/wav48/p225/p225_014.wav']
dataset wav preprocessing (14 / 500) - 'asset/Dataset/wav/wav48/p225/p225_015.wav']
```

Gambar 4.19 *Preprcessing File Wav*

```
dataset flac preprocessing (0 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0001.flac']
dataset flac preprocessing (1 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0002.flac']
dataset flac preprocessing (2 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0003.flac']
dataset flac preprocessing (3 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0004.flac']
dataset flac preprocessing (4 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0005.flac']
dataset flac preprocessing (5 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0006.flac']
dataset flac preprocessing (6 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0007.flac']
dataset flac preprocessing (7 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0008.flac']
dataset flac preprocessing (8 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0009.flac']
dataset flac preprocessing (9 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0010.flac']
dataset flac preprocessing (10 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0011.flac']
dataset flac preprocessing (11 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0012.flac']
dataset flac preprocessing (12 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0013.flac']
dataset flac preprocessing (13 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0014.flac']
dataset flac preprocessing (14 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0015.flac']
dataset flac preprocessing (15 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0016.flac']
dataset flac preprocessing (16 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0017.flac']
dataset flac preprocessing (17 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0018.flac']
dataset flac preprocessing (18 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0019.flac']
dataset flac preprocessing (19 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0020.flac']
dataset flac preprocessing (20 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0021.flac']
dataset flac preprocessing (21 / 500) - 'asset/Dataset/flac/train/4/4/4-4-0022.flac']
```

Gambar 4.20 *Preprocessing File Flac*

4.1.5 Implementasi *Training*

Pada tahap *training* ini dilakukan proses pembelajaran pada data suara yang sudah diolah dari tahap sebelumnya yaitu pada tahap *preprocessing*. Data yang digunakan pada tahap ini berjumlah 500 data suara. Pada tahap ini *speech to text* mulai untuk melakukan pembelajaran dengan menggunakan metode *deep learning* yang sudah dijelaskan pada bab sebelumnya. Untuk proses yang dilakukan pada *training* ini yaitu proses yang sudah dibuat pada *flowchart* perancangan. *Flowchart* tersebut kemudian dibuat dalam bahasa pemrograman *python*. Kode program pada tahap *training* ini dapat dilihat pada Lampiran E.

Kode program tersebut dimulai dengan deklarasi *library* yang dipakai pada tahap *training* ini. *Library* tersebut berfungsi untuk membaca *file*, menyimpan *file* dan melakukan perhitungan. Penjelasan dari *library* tersebut adalah *sugartensor* sebagai *library* utama untuk melakukan perhitungan dan pembuatan model, *library* data dan model digunakan dalam pembacaan data dan peyimpanan model dan *library os* untuk mengatur *input/output* pada sistem.

Tahap pertama dimulai dengan pengaturan *log* yang digunakan untuk *debugging* atau menelusuri kesalahan yang mungkin terjadi pada program di atas. Selanjutnya adalah adalah membuat *batch file* atau pembagi data. *Batch file* ini digunakan sebagai pembagi jumlah data yang masuk pada tahap pembelajaran. Jika *batch* terdapat 500 data dengan menggunakan *batch file* 10 maka data yang dimasukkan ke dalam proses pembelajaran adalah 50 sekaligus. Kemudian setelah *batch file* sudah ditentukan maka *file t*ersebut masuk ke *tensorflow* beserta MFCC *feature* dan juga label pada data suara tersebut. Kemudian dilakukan pembacaan isi dari label yang ssudah dimasukkan sebelumnya.

Tahap selanjutnya adalah pengambilan nilai dari CTC *Loss*. Nilai ini berfungsi untuk melihat apakah pembelajaran pada *speect to text* ini sudah baik atau belum. Semakin kecil nilai CTC *Los*s maka pembelajaran yang diberikan semakin baik, tetapi jika nilai CTC terlalu kecil maka pembelajaran dapat mengalami *Overfitting* atau program mengalami pembelajarn yang berlebih. Hai ini mengakibatkan keluaran dari pembelajaran bernilai kosong ataua tanpa *output.* Hai tersebut dapat dihindari dengan mengganti berapa kali perulangan yang dipakai dalam pembelajaran ini.

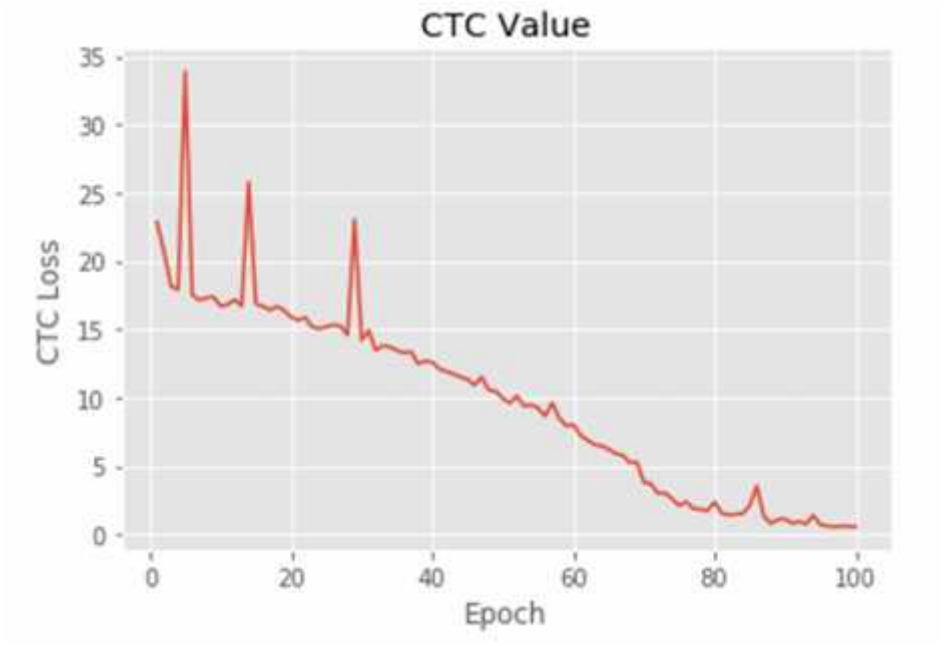
Tahap terakhir pada proses *training* dengan menjalankan *deep learning,* ini adalah tahap pembelajaran. Pada tahap ini data yang sudah masuk mengalami pembelajaran sehingga didapatkan *output* berupa model pembelajaran yang nantinya digunakan pada tahap selanjutnya. *Input* yang digunakan pada tahap ini adalah nilai MFCC *feature* dan nilai label yang sudah didapatkan pada tahap *preprocessing.* Jumlah *input* pada jaringan syaraf ini berjumlah sesuai dengan banyaknya *batch file* yang sudah dideklarasikan. Seperti yang sudah dijelaskan sebelumnya pembelajaran ini dilakukan dengan menentukan perulangan sebagai penentu akurasi dari model yang dihasilnya. Untuk menghindari *Overfitting* perulangan dilakukan dengan melihat nilai dari CTC *Los*s. Jika nilai tersebut mulai turun dan penurunannya sudah sedikit atau tidak terlalu signifikan maka didapatkan perulangan yang sesuai. Pada percobaan ini digunakan rentang perulangan yaitu dari 0 sampai dengan 100. Sehingga didapatkan CTC *Loss* seperti pada Tabel 4.1.

Tabel 4.3 Nilai CTC *Loss*

Epoch	CTC Loss
0	22.830822
1	20.63934
2	18.132956
3	17.898919
4	33.879594
⋮	⋮
95	0.714438
96	0.623234
97	0.565023
98	0.616237
99	0.601662
100	0.567696

Pada tabel di atas dapat dilihat nilai dari CTC *Loss* terus menurun dari 22.830822 menjadi 0.567696. Pada perulangan ke 100 didapatkan nilai yang kurang dari 1. Nilai tersebut dirasa sudah cukup karena pada perulangan 70an nilai CTC *Loss* mengalami penurunan yang tidak signifikan. Untuk melihat proses lebih detail dari tahap *training* ini dapat dilihat Lampiran F.

Nilai-nilai CTC yang didapatkan tersebut adalah hasil dari proses perulangan yang dilakukan. Nilai CTC tersebut mengalami perubahan sesuai dengan model yang dibuat. Perubahan tersebut cenderung semakin kecil, sehingga dapat disimpulkan nilai perulangan/*epoch* yang sesuai adalah 100. Hal tersebut diambil karena perubahan nilai tidak terlalu signifikan lagi. Perubahan tersebut dapat dilihat pada visualisasi pada Gambar 4.21.



Gambar 4.21 Visualisasi CTC *Loss*

Berdasarkan grafik di atas dapat terdapat beberapa nilai yang *outlier* atau nilai yang terlalu besar kemudian kembali mengecil dengan jarang yang jauh. Tetapi semakin banyak perulangan yang dilakukan nilai tersebut mengalami perubahan yang tidak terlalu besar. Nilai CTC *Loss* mengalami perubahan yang tidak terlalu signifikan pada perulangan di atas 90 dan mengalami nilai yang datar pada perulangan ke 100. Hal ini disimpulkan perulangan terbaik yang digunakan adalah perulangan ke 100.

4.1.6 Implementasi *Testing*

Pada tahap terakhir dari proses *speech to text* ini akan dilakukan tahap pengujian atau *testing* terlebih dahulu. Pada tahap ini *speech to text* dilakukan pengujian untuk melihat keluaran yang dihasilkan dari model yang sudah dibuat sebelumnya pada tahap *training.* Proses *testing* menggunakan 50 sampel data baru. Seperti pada *flowchart* yang sudah dibuat pada tahap perancangan pada tahap ini dilakukan pembuatan kode program. Kode program tersebut dapat dilihat pada Lampiran G.

Pada kode program tersebut tahap pertama yang dilakukan adalah deklarasi *library* yang digunakan pada program. *Library* tersebut digunakan sebagai pembacaan *file, load file,* menampilkan *file* dan proses perhitungan. Penjelasan dari *library* tersebut adalah *tensorflow* dan *sugartensor* sebagai *library* utama untuk melakukan perhitungan, *library librosa* dan *numpy* sebagai *library* untuk membaca *file* audio, mendapatkan nilai MFCC dan nilai labelnya,

library model dan data sebagai pembaca *file* data dan model dan *library os*, dan *sys s*ebagai *library* untuk pembacaan parameter dan *input/output* pada sistem.

Tahap pertama adalah mengatur *log level* untuk proses *debugging*, sehingga jika terjadi kesalahan maka dapat ditelusuri lebih mudah. Kemudian seperti tahap *training* dilakukan penentuan *batch file*, tetapi berbeda seperti tahap *training* pada tahap *testing batch file* yang dipakai adalah satu karena jumlah *file* yang masuk untuk dilakukan perhitungan *speech to text* adalah satu per satu *file.*

Tahap selanjutnya adalah memasukan panjang kata pada label. Kemudian juga memasukan MFCC *feature* pada *file dataset* yang digunakan pada tahap *testing* dengan panjang *sequenc*e tanpa *zero padding*. Dilanjutkan dengan meng-*encode file* dan *ctc decoding* hal ini dilakukan untuk mpembacaan pada *tensorflow*. Tahap selanjutnya adalah melakukan perhitungan *speech to text* yaitu proses *recognize*. Pada proses ini dimulai dengan pembacaan *path file* audio. Kemudian me-*load file* audio yang digunakan pada *testing*. Selanjutnya diambil MFCC *feature*-nya seperti pada tahap sebelumnya.

Tahap terakhir adalah proses perhitungan *speech to text* berdasarkan model yang sudah dibuat. Pada proses ini dilakukan pembacaan *file train* hasil dari tahap *training.* Selanjutnya dilakukan perhitungan *speech to text* dan menampilkan label dari hasil perhitungan tersebut. Keluaran pada tahap tersebut dapat dilihat pada Lampiran H.

Keluaran tersebut menunjukan *output* yang dihasilkan oleh *speech to text* dan nilai kemiripan dengan label. Kemiripan tersebut diambil dengan metode *sequence matcher* menggunakan *library difflib* pada *python*. Metode tersebut mengambil kemiripan dengan mencari huruf yang sama dengan label dan membaginya dengan jumlah huruf keseluruhan. Rumus tersebut dapat dilihat pada persamaan (4.1).

$$K \quad i \quad = \frac{Ju \quad h\,H \quad y \quad Si \quad d \quad L}{Ju \quad h\,H \quad K \quad ha} \qquad (4.1)$$

Nilai kemiripan tersebut mempunyai rentang penilaian dari 0 sampai 1. Nilai tersebut menunjukan apabila nilai kemiripan mendekati nilai 1 maka hasil *testing* semakin mirip dengan data label, begitu sebaliknya. Berdasarkan nilai kemiripan tersebut dapat disimpulkan kemiripan rata-rata dari pengujian 50 data *testing* adalah 0.7879207838944682. Nilai kemiripan tersebut dikategorikan menjadi tiga kategori yaitu: tepat, kurang tepat dan tidak tepat. Penentuan dari ketiga kategori tersebut diambil berdasarkan rentang nilai pada nilai kemiripan. Nilai tersebut yaitu: jika nilai kemiripan yang dihasilkan sama dengan 1 maka keluaran tersebut masuk ke dalam kategori tepat, karena hasil dari *speech to text* sudah sama

dengan label. Jika nilai kemiripan diantara 1 dan 0.75 maka keluaran tersebut masuk ke dalam kategori kurang tepat. Sedangkan jika nilai tersebut dibawah 0.75 maka masuk ke dalam kategori tidak tepat.

Pada ketiga kategori tersebut dapat dibedakan berdasarkan keluaran dari *speech to text.* Diantaranya adalah perbedaan jumlah huruf antara label dan keluaran, hilangnya atau bertambahnya jumlah huruf pada keluaran, atau tertukarnya huruf pada keluaran *speech to text*. Hal tersebut disebabkan oleh karena suara pada dataset yang terlalu cepat dalam pengucapannya, adanya kemiripan pengucapan huruf, ada huruf yang tidak terbaca dengan jelas, adanya kedekatan nilai pada label, jumlah huruf yang terdapat pada kata tersebut, dan lain-lain. Dari kategori tersebut dapat dicari akurasi berdasarkan nilai kemiripannya.

Akurasi dicari dengan membagi data menjadi dua yaitu tepat dan tidak tepat. Hal tersebut didapat dengan menggunakan nilai *threshold* atau nilai ambang batas yaitu 0.75. Jika nilai kemiripan di atas nilai *threshold* atau sama dengan nilai *threshold* maka hasil *output* akan dianggap sama dengan label dan masuk ke kategori tepat. Jika tidak memenuhi maka *output* masuk pada kategori tidak tepat. Akurasi tersebut dicari pada Bahasa Indonesia dan Bahasa Jawa. Keterangan kategori dan jumlah dari masing-masing kategori dapat dilihat pada Tabel 4.2.

Tabel 4.4 Jumlah Hasil *Testing* berdasarkan Kategori

Bahasa Indonesia		Bahasa Jawa	
Kategori	Jumlah	Kategori	Jumlah
Tepat	28	Tepat	4
Tidak Tepat	15	Tidak Tepat	3
Total	43	Total	7
Akurasi	65%	Akurasi	57%

Berdasarkan tabel di atas dapat disimpulkan bahwa nilai akurasi pada Bahasa Indonesia adalah 65% Sedangkan untuk Bahasa Jawa adalah 57%. Nilai tersebut didapatkan dengan rumus pada persamaan (4.2). Hal tersebut sudah menunjukan bahwa *speech to text* sudah dapat mengenali 65% data pada Bahasa Indonesia dan 57% data pada Bahasa Jawa.

$$A = \frac{Ju \quad h\,K \qquad S \quad h\,T}{T \quad D} . 100\% \qquad (4.2)$$

Berdasarkan tabel di atas data Bahasa Jawa hanya terdapat 7 data. Hal tersebut penting untuk ditingkatkan pada banyaknya variasi dari dataset khususnya Bahasa Jawa sehingga akurasi dari *speech to text* dapat menjadi lebih baik lagi.

4.2 Analisis Sistem

Pada tahap sistem merupakan tahapan di mana sebuah sistem dinilai kelebihan dan kekurangannya setelah dilakukan tahap pengujian. Setiap kelebihan dan kekurangan yang didapatkan dapat menjadi bahan pertimbangan bagi penulis dan pembaca untuk menilai keberhasilan dari sistem ini. Kekurangan yang didapat dari sistem ini juga diharapkan dapat menjadi pertimbangan untuk mengembangkan sistem yang lebih sempurna dimasa mendatang. Kelebihan dari penelitian yang menjadi nilai tambah dari *speech to text* adalah sebagai berikut.

4.2.1 Kelebihan

a. *Speech to text* dapat menerima masukan pembelajaran dengan data tipe *wav* dan *flac*.
b. *Speech to text* dapat digunakan dalam Bahasa Indonesia dan Bahasa Jawa.
c. *Speech to text* dapat mengenali 50 kata walaupun belum sempurna.

4.2.2 Kekurangan

a. Dataset yang digunakan masih terbatas.
b. Semakin besar dataset maka komputasi yang digunakan juga semakin besar.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan perancangan dan juga pengujian yang sudah dilakukan pada penelitian ini maka, didapatkan kesimpulan bahwa penelitian ini yaitu:

a. Pendekatan *speech to text* dapat digunakan sebagai analisis pada data suara atau rekaman pada kata Bahasa Indonesia dan Bahasa Jawa.
b. *Speech to text* dapat menerima masukan dalam bentuk *file wav* dan *file flac.*
c. *Speech to text* dapat mengenali dalam Bahasa Indonesia dan Bahasa Jawa.
d. Akurasi yang dihasilkan *speech to text* pada data Bahasa Indonesia adalah 65% Sedangkan untuk data Bahasa Jawa adalah 57%.

5.2 Saran

Setelah didapatkan sebelumnya bahwa sistem ini masih memiliki banyak kekurangan seperti yang telah dijelaskan sebelumnya, maka diharapkan pada penelitian selanjutnya dapat menghilangkan atau menutupi kekurangan yang ada pada *speech to text* ini. Bahkan sangat diharapkan penelitian selanjutnya dapat mengembangkan *speech to text* ini lebih luas lagi. Saran untuk pengembangan *speech to text* pada penelitian selanjutnya adalah sebagai berikut

a. Dapat memasukkan kata lebih dari satu atau kalimat pada dataset, sehingga *speech to text* dapat mengenali kata lebih dari dua pada tahap pembelajarannya.
b. Dapat dibuat tampilan yang lebih menarik sehingga lebih mudah dalam proses *speech to text.*

DAFTAR PUSTAKA

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., … Isard, M. (2016). TensorFlow: A System for Large-Scale Machine Learning. In *OSDI* (Vol. 16, pp. 265–283).

Areni, I. S., & Bustamin, A. (2017). Improvement in Speech to Text for Bahasa Indonesia Through Homophone Impairment Training. 電腦學刊, *28*(5), 1–10.

Cawley, G. C., & Talbot, N. L. (2010). On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, *11*(Jul), 2079–2107.

Chamidy, T. (2016). Metode Mel Frequency Cepstral Coeffisients (MFCC) Pada klasifikasi Hidden Markov Model (HMM) Untuk Kata Arabic pada Penutur Indonesia. *Matics*, *8*(1), 36–39.

Dahl, G. E., Yu, D., Deng, L., & Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, *20*(1), 30–42.

Deng, L., Li, J., Huang, J.-T., Yao, K., Yu, D., Seide, F., … Williams, J. D. (2013). Recent advances in deep learning for speech research at Microsoft. In *ICASSP* (Vol. 26, p. 64).

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1). MIT press Cambridge.

Graves, A. (2012). Connectionist Temporal Classification. In *Supervised Sequence Labelling with Recurrent Neural Networks* (pp. 61–93). Springer.

Graves, A., Fernández, S., Gomez, F., & Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning* (pp. 369–376). ACM.

Hagan, M. T., Demuth, H. B., & Beale, M. H. (1996). *Neural network design* (Vol. 20). Pws Pub. Boston.

Hawkins, D. M. (2004). The problem of overfitting. *Journal of Chemical Information and Computer Sciences*, *44*(1), 1–12.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).

Herlim, L. (2002). *Pengenalan kata dengan menggunakan fuzzy logic untuk menggerakkan robot mobil* (PhD Thesis). Petra Christian University.

Hotta, H. (2011). Japanese speaker-Independent homonyms speech recognition. *Procedia-Social and Behavioral Sciences*, *27*, 306–313.

Huang, D.-A., Fei-Fei, L., & Niebles, J. C. (2016). Connectionist temporal modeling for weakly supervised action labeling. In *European Conference on Computer Vision* (pp. 137–153). Springer.

Khilari, M. P., & Bhope, V. P. (2015). A Review On Speech To Text Conversion Methods. *International Journal of Advanced Research in Computer Engineering & Technology*, *4*(7).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436.

Mon, S. M., & Tun, H. M. (2015). Speech-To-Text Conversion (STT) System Using Hidden Markov Model (HMM). *International Journal of Scientific & Technology Research*, *4*(6), 349–352.

Noertjahyana, A., & Adipranata, R. (2004). Implementasi Sistem Pengenalan Suara Menggunakan SAPI 5.1 dan Delphi 5. *Jurnal Informatika*, *4*(2), pp–96.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117.

Shatkay, H. (1999). Learning hidden Markov models with geometrical constraints. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence* (pp. 602–611). Morgan Kaufmann Publishers Inc.

Sidiq, M., Wirayuda, T. A. B., & Sa'adah, S. (2015). Desain Dan Implementasi Voice Command Menggunakan Metode Mfcc Dan Hmms. *eProceedings of Engineering*, *2*(1).

Suryadharma, K., Budiman, G., & Irawan, B. (2014). Perancangan Aplikasi Speech To Text Bahasa Inggris Ke Bahasa Bali Menggunakan Pocketsphinx Berbasis Android. *eProceedings of Engineering*, *1*(1).

Wang, S.-C. (2003). Artificial neural network. In *Interdisciplinary computing in java programming* (pp. 81–100). Springer.

Wicaksono, A. F., & Purwarianti, A. (2010). HMM Based part-of-speech tagger for bahasa indonesia. In *Fourth International MALINDO Workshop, Jakarta*.

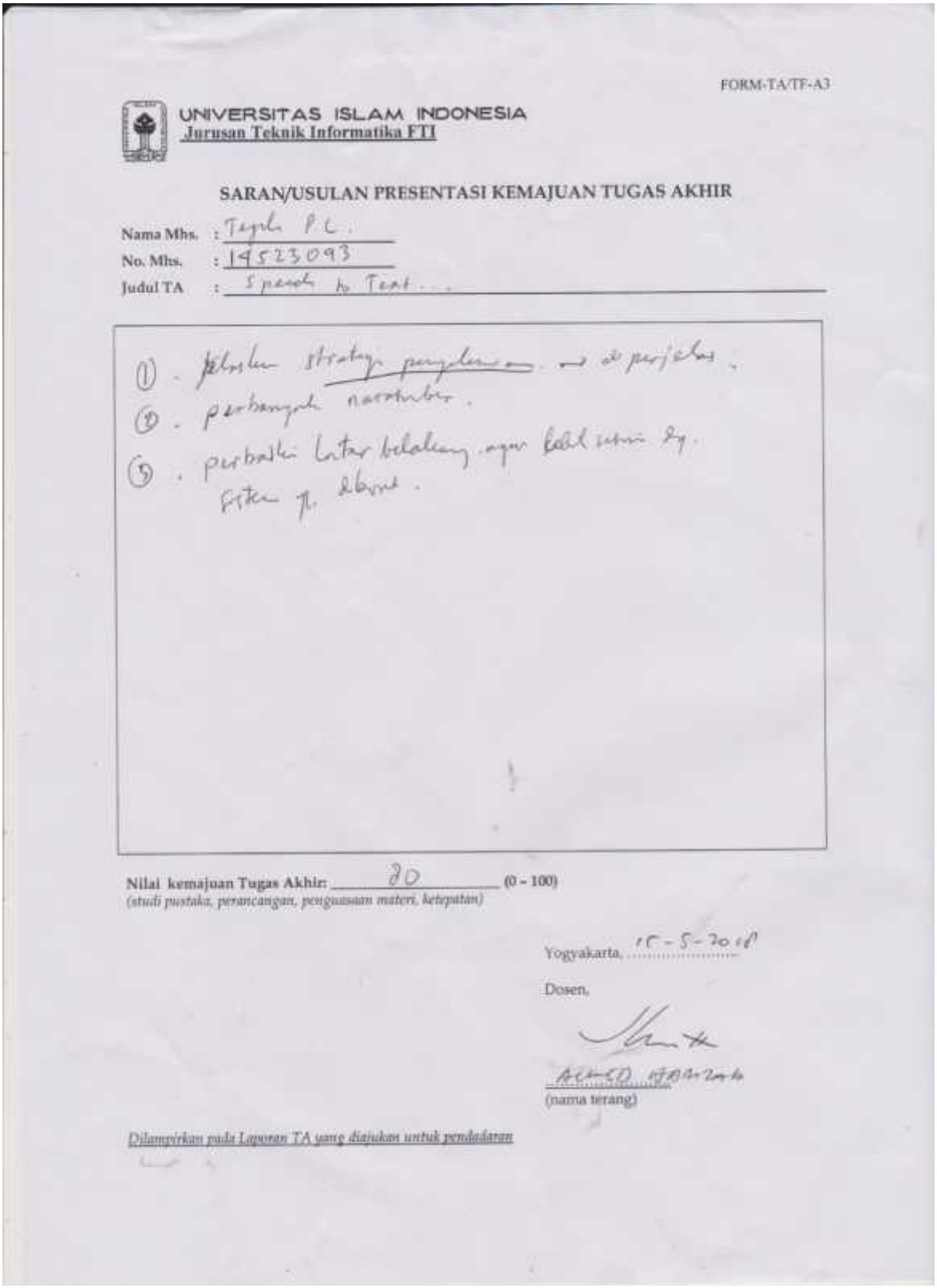
WIJAYA, T., SUSANTO, S., & Galih Salman, S. T. (2013). *Speech Recognition Bahasa Indonesia Untuk Android* (PhD Thesis). BINUS.

Kim & Park. *Speech-to-Text-WaveNet*. 2016. GitHub repository. *github.com/buriburisuri*.

LAMPIRAN

LAMPIRAN A

Lembar Kemajuan Tugas Akhir



FORM-TA/TF-A3

UNIVERSITAS ISLAM INDONESIA
Jurusan Teknik Informatika FTI

SARAN/USULAN PRESENTASI KEMAJUAN TUGAS AKHIR

Nama Mhs. : Teyla P.L.
No. Mhs. : 14523093
Judul TA : Speech to Text ...

① . Jelaskan strategi penyelesaian → di perjelas.
② . perbanyak narasumber.
③ . perbaiki latar belakang agar lebih urut dg. fokus yg. dibuat.

Nilai kemajuan Tugas Akhir: 80 (0 - 100)
(studi pustaka, perancangan, penguasaan materi, ketepatan)

Yogyakarta, 15-5-2018

Dosen,

AHMD 15/5/2018
(nama terang)

Dilampirkan pada Laporan TA yang diajukan untuk pendadaran

LAMPIRAN B

Kode Program *Prepocessing* pada Data *Wav*

```
# path data
_data_path = "asset/Dataset/"

# prosedur process dataset wav
def process_wav(csv_file):
    # membuat csv writer
    writer = csv.writer(csv_file, delimiter=',')
    # membaca label info
    df = pd.read_table(_data_path + 'wav/speaker-info.txt',
usecols=['ID'],
                       index_col=False, delim_whitespace=True)

    # membaca id file
    file_ids = []
    for d in [_data_path + 'wav/txt/p%d/' % uid for uid in
df.ID.values]:
        file_ids.extend([f[-12:-4] for f in sorted(glob.glob(d +
'*.txt'))])

    for i, f in enumerate(file_ids):
        # membaca filename pada data wav
        wave_file = _data_path + 'wav/wav48/%s/' % f[:4] + f +
'.wav'
        fn = wave_file.split('/')[-1]
        target_filename = 'asset/Dataset/preprocess/mfcc/' + fn +
'.npy'
        if os.path.exists( target_filename ):
            continue
        # print info
        print("dataset wav preprocessing (%d / %d) - '%s']" % (i,
len(file_ids), wave_file))
        # load wav file
        wave, sr = librosa.load(wave_file, mono=True, sr=None)
        # re-sample frekuensi ( 48K -> 16K )
        wave = wave[::3]
        # mendapatkan mfcc feature
        mfcc = librosa.feature.mfcc(wave, sr=16000)
        # mendapatkan label index
        label = data.str2index(open(_data_path + 'wav/txt/%s/' %
f[:4] + f + '.txt').read())

        # menyimpan hasil
        if len(label) < mfcc.shape[1]:

        # menyimpan meta info
            writer.writerow([fn] + label
)
            # menyimpan mfcc
            np.save(target_filename, mfcc, allow_pickle=False)
```

LAMPIRAN C

Kode Program *Prepocessing* pada Data *Flac*

```
# prosedur process dataset flac
def process_flac(csv_file, category):
    parent_path = _data_path + 'flac/' + category + '/'
    labels, wave_files = [], []
    # membuat csv writer
    writer = csv.writer(csv_file, delimiter=',')
    # membaca direktori list dari speaker
    speaker_list = glob.glob(parent_path + '*')
    for spk in speaker_list:
        # membaca direktori list dari chapter
        chapter_list = glob.glob(spk + '/*/')
        for chap in chapter_list:
            # membaca list file pada text label
            txt_list = glob.glob(chap + '/*.txt')
            for txt in txt_list:
                with open(txt, 'rt') as f:
                    records = f.readlines()
                    for record in records:
                        # memparsing record
                        field = record.split('-')  # split by '-'
                        speaker = field[0]
                        chapter = field[1]
                        field = field[2].split()  # split
field[2] by ' '
                        utterance = field[0]  # kolom pertama
pada id ucapan

                        # file name flac
                        wave_file = parent_path + '%s/%s/%s-%s-
%s.flac' % \
                                                    (speaker,
chapter,  speaker, chapter, utterance)
                        wave_files.append(wave_file)
                        # label index
                        labels.append(data.str2index('
'.join(field[1:])))  # kolom terakhir menjadi text label
    # menyimpan hasil
    for i, (wave_file, label) in enumerate(zip(wave_files,
labels)):
        fn = wave_file.split('/')[-1]
        target_filename = 'asset/Dataset/preprocess/mfcc/' + fn +
'.npy'
        if os.path.exists( target_filename ):
            continue
        # print info
        print("dataset flac preprocessing (%d / %d) - '%s']" %
(i, len(wave_files), wave_file))
        # load flac file
        wave, sr = librosa.load(wave_file, mono=True, sr=None)
        # mendapatkan mfcc feature
        mfcc = librosa.feature.mfcc(wave, sr=16000)
```

```
        # menyimpan hasil
        if len(label) < mfcc.shape[1]:
            # menyimpan meta info
            writer.writerow([fn] + label)

            # menyimpan mfcc
            np.save(target_filename, mfcc, allow_pickle=False)
```

LAMPIRAN D

Kode Program *Preporcessing*

```
# membuat direktori
if not os.path.exists('asset/Dataset/preprocess'):
    os.makedirs('asset/Dataset/preprocess')
if not os.path.exists('asset/Dataset/preprocess/meta'):
    os.makedirs('asset/Dataset/preprocess/meta')
if not os.path.exists('asset/Dataset/preprocess/mfcc'):
    os.makedirs('asset/Dataset/preprocess/mfcc')

# menjalankan preprocessing

# dataset wav untuk training
csv_f = open('asset/Dataset/preprocess/meta/train.csv', 'w')
process_wav(csv_f)
csv_f.close()

# dataset flac untuk training
csv_f = open('asset/Dataset/preprocess/meta/train.csv', 'a+')
process_flac(csv_f,'train')
csv_f.close()

# dataset flac untuk validasi
csv_f = open('asset/Dataset/preprocess/meta/valid.csv', 'w')
process_flac(csv_f, 'valid')
csv_f.close()

# dataset flac untuk testing
csv_f = open('asset/Dataset/preprocess/meta/test.csv', 'w')
process_flac(csv_f, 'test')
csv_f.close()
```

LAMPIRAN E

Kode Program *Training*

```
import tensorflow as tfs
tfs.logging.set_verbosity(tfs.logging.ERROR)
import os
import sugartensor as tf
from data import SpeechCorpus, voca_size
from model import *
from tqdm import tqdm
tqdm.monitor_interval = 0

# mengatur log level untuk debug
tf.sg_verbosity(10)

# hyper parameters
batch_size = 16    # total batch size

# menginput corpus ke tensorflow
data = SpeechCorpus(batch_size=batch_size * tf.sg_gpus())

# menginput mfcc feature dari file audio
inputs = tf.split(data.mfcc, tf.sg_gpus(), axis=0)

# mengambil label
labels = tf.split(data.label, tf.sg_gpus(), axis=0)

# panjang sequence kecuali zero-padding
seq_len = []
for input_ in inputs:
    seq_len.append(tf.not_equal(input_.sg_sum(axis=2),
0.).sg_int().sg_sum(axis=1))

# pemprosesan parallel untuk mengambil loss
@tf.sg_parallel
def get_loss(opt):
    # encode audio feature
    logit = get_logit(opt.input[opt.gpu_index],
voca_size=voca_size)
    # CTC loss
    return logit.sg_ctc(target=opt.target[opt.gpu_index],
seq_len=opt.seq_len[opt.gpu_index])

# train
tf.sg_train(lr=0.0001, loss=get_loss(input=inputs, target=labels,
seq_len=seq_len),
            ep_size=data.num_batch, max_ep=100)
```

LAMPIRAN F

Proses *Training*

```
I 0704:22:15:57.851:sg_train.py:301]    Epoch[000:gs=62] - loss = 22.830822
I 0704:22:16:12.976:sg_train.py:301]    Epoch[001:gs=124] - loss = 20.639340
I 0704:22:16:27.502:sg_train.py:301]    Epoch[002:gs=186] - loss = 18.132956
I 0704:22:16:42.177:sg_train.py:301]    Epoch[003:gs=248] - loss = 17.898919
I 0704:22:16:56.522:sg_train.py:301]    Epoch[004:gs=310] - loss = 33.879594
I 0704:22:17:11.174:sg_train.py:301]    Epoch[005:gs=372] - loss = 17.499906
I 0704:22:17:25.449:sg_train.py:301]    Epoch[006:gs=434] - loss = 17.145426
I 0704:22:17:39.936:sg_train.py:301]    Epoch[007:gs=496] - loss = 17.275030
I 0704:22:17:54.461:sg_train.py:301]    Epoch[008:gs=558] - loss = 17.372066
I 0704:22:18:08.715:sg_train.py:301]    Epoch[009:gs=620] - loss = 16.694771
I 0704:22:18:23.148:sg_train.py:301]    Epoch[010:gs=682] - loss = 16.799820
I 0704:22:18:37.672:sg_train.py:301]    Epoch[011:gs=744] - loss = 17.152891
I 0704:22:18:51.930:sg_train.py:301]    Epoch[012:gs=806] - loss = 16.696842
I 0704:22:19:06.189:sg_train.py:301]    Epoch[013:gs=868] - loss = 25.753040
I 0704:22:19:20.455:sg_train.py:301]    Epoch[014:gs=930] - loss = 16.856163
I 0704:22:19:34.925:sg_train.py:301]    Epoch[015:gs=992] - loss = 16.673309
I 0704:22:19:49.210:sg_train.py:301]    Epoch[016:gs=1054] - loss = 16.408082
I 0704:22:20:03.664:sg_train.py:301]    Epoch[017:gs=1116] - loss = 16.668346
I 0704:22:20:18.110:sg_train.py:301]    Epoch[018:gs=1178] - loss = 16.376942
I 0704:22:20:32.668:sg_train.py:301]    Epoch[019:gs=1240] - loss = 15.895343
I 0704:22:20:46.926:sg_train.py:301]    Epoch[020:gs=1302] - loss = 15.665079
```

⋮

```
I 0704:22:32:46.279:sg_train.py:301]    Epoch[070:gs=4402] - loss = 3.801103
I 0704:22:33:00.578:sg_train.py:301]    Epoch[071:gs=4464] - loss = 3.704835
I 0704:22:33:14.863:sg_train.py:301]    Epoch[072:gs=4526] - loss = 2.996940
I 0704:22:33:29.385:sg_train.py:301]    Epoch[073:gs=4588] - loss = 3.037602
I 0704:22:33:43.625:sg_train.py:301]    Epoch[074:gs=4650] - loss = 2.606005
I 0704:22:33:57.922:sg_train.py:301]    Epoch[075:gs=4712] - loss = 2.102852
I 0704:22:34:12.130:sg_train.py:301]    Epoch[076:gs=4774] - loss = 2.397319
I 0704:22:34:26.417:sg_train.py:301]    Epoch[077:gs=4836] - loss = 1.890426
I 0704:22:34:40.881:sg_train.py:301]    Epoch[078:gs=4898] - loss = 1.834656
I 0704:22:34:55.100:sg_train.py:301]    Epoch[079:gs=4960] - loss = 1.728576
I 0704:22:35:09.329:sg_train.py:301]    Epoch[080:gs=5022] - loss = 2.381576
I 0704:22:35:23.554:sg_train.py:301]    Epoch[081:gs=5084] - loss = 1.558959
I 0704:22:35:38.998:sg_train.py:301]    Epoch[082:gs=5146] - loss = 1.431296
I 0704:22:35:53.311:sg_train.py:301]    Epoch[083:gs=5208] - loss = 1.496717
I 0704:22:36:07.560:sg_train.py:301]    Epoch[084:gs=5270] - loss = 1.511195
I 0704:22:36:21.867:sg_train.py:301]    Epoch[085:gs=5332] - loss = 2.133372
I 0704:22:36:36.354:sg_train.py:301]    Epoch[086:gs=5394] - loss = 3.537837
I 0704:22:36:50.660:sg_train.py:301]    Epoch[087:gs=5456] - loss = 1.328373
I 0704:22:37:04.894:sg_train.py:301]    Epoch[088:gs=5518] - loss = 0.808426
I 0704:22:37:19.166:sg_train.py:301]    Epoch[089:gs=5580] - loss = 1.099148
I 0704:22:37:33.603:sg_train.py:301]    Epoch[090:gs=5642] - loss = 1.143844
I 0704:22:37:47.907:sg_train.py:301]    Epoch[091:gs=5704] - loss = 0.815924
I 0704:22:38:02.157:sg_train.py:301]    Epoch[092:gs=5766] - loss = 0.955288
I 0704:22:38:16.395:sg_train.py:301]    Epoch[093:gs=5828] - loss = 0.770164
I 0704:22:38:30.860:sg_train.py:301]    Epoch[094:gs=5890] - loss = 1.401977
I 0704:22:38:45.099:sg_train.py:301]    Epoch[095:gs=5952] - loss = 0.714438
I 0704:22:38:59.361:sg_train.py:301]    Epoch[096:gs=6014] - loss = 0.623234
I 0704:22:39:13.590:sg_train.py:301]    Epoch[097:gs=6076] - loss = 0.565023
I 0704:22:39:28.079:sg_train.py:301]    Epoch[098:gs=6138] - loss = 0.616237
I 0704:22:39:42.592:sg_train.py:301]    Epoch[099:gs=6200] - loss = 0.601662
I 0704:22:39:56.882:sg_train.py:301]    Epoch[100:gs=6262] - loss = 0.567696
```

LAMPIRAN G

Kode Program *Testing*

```
import tensorflow as tfs
tfs.logging.set_verbosity(tfs.logging.ERROR)
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
import sugartensor as tf
import numpy as np
import librosa
from model import *
import data
import sys

# mengatur log level untuk debug
tf.sg_verbosity(10)

# hyper parameters
batch_size = 1     # batch size

# inputs
# panjang kata
voca_size = data.voca_size

# menginput mfcc feature pada file audio
x = tf.placeholder(dtype=tf.sg_floatx, shape=(batch_size, None,
20))

# panjang sequence kecuali zero-padding
seq_len = tf.not_equal(x.sg_sum(axis=2),
0.).sg_int().sg_sum(axis=1)

# encode audio feature
logit = get_logit(x, voca_size=voca_size)

# ctc decoding
decoded, _ =
tf.nn.ctc_beam_search_decoder(logit.sg_transpose(perm=[1, 0, 2]),
seq_len, merge_repeated=False)

# to dense tensor
y = tf.sparse_to_dense(decoded[0].indices,
decoded[0].dense_shape, decoded[0].values) + 1

# regcognize audio file

# perintah untuk menginput path file audio
tf.sg_arg_def(file=('', 'speech wave file to recognize.'))

# load audio file
file = sys.argv[1]
wav, sr = librosa.load(file, mono=True, sr=16000)
```

```
# mendapatkan mfcc feature
mfcc = np.transpose(np.expand_dims(librosa.feature.mfcc(wav,
16000), axis=0), [0, 2, 1])

# run network
with tf.Session() as sess:

    # init variables
    tf.sg_init(sess)

    # restore parameters
    saver = tf.train.Saver()
    saver.restore(sess,
tf.train.latest_checkpoint('asset/train'))

    # run session
    label = sess.run(y, feed_dict={x: mfcc})

    # print label
    data.print_index(label)
```

LAMPIRAN H

Tabel Hasil *Testing*

No.	Label	Hasil *Output*	Kemiripan	Kategori	*Output* dengan Kemiripan ≥ 0.75
1.	batuk	keuk	0.444	Tidak Tepat	keuk
2.	bersin	lirsin	0.667	Tidak Tepat	lirsin
3.	pusing	pesing	0.833	Kurang Tepat	pusing
4.	demam	mema	0.667	Tidak Tepat	mema
5.	gatal	gatol	0.8	Kurang Tepat	gatal
6.	pilek	nelel	0.4	Tidak Tepat	nelel
7.	mual	mutal	0.889	Kurang Tepat	mual
8.	muntah	muna	0.8	Kurang Tepat	muntah
9.	kembung	pembang	0.714	Tidak Tepat	pembang
10.	perih	lerih	0.8	Kurang Tepat	perih
11.	kram	kaam	0.75	Kurang Tepat	kram
12.	sesak	kesak	0.8	Kurang Tepat	sesak
13.	sesek	kese	0.667	Tidak Tepat	kese
14.	radang	merdang	0.77	Kurang Tepat	radang
15.	nyeri	nyeri	1.0	Tepat	nyeri
16.	sariawan	sariawan	1.0	Tepat	sariawan
17.	panas	pans	0.889	Kurang Tepat	panas
18.	dingin	dingin	1.0	Tepat	dingin
19.	kedinginan	kedingina	0.948	Kurang Tepat	kedinginan
20.	diare	niare	0.8	Kurang Tepat	diare
21.	meler	mel	0.75	Kurang Tepat	meler
22.	bengkak	benkal	0.77	Kurang Tepat	bengkak
23.	benjol	benjol	1.0	Tepat	benjol
24.	luka	mea	0.286	Tidak Tepat	mea
25.	alergi	alergi	1.0	Tepat	alergi
26.	ngilu	diilu	0.6	Tidak Tepat	diilu
27.	pegal	pe	0.571	Tidak Tepat	pe
28.	berdarah	berdarah	1.0	Tepat	berdarah

29.	bernanah	pernah	0.714	Tidak Tepat	pernah
30.	memar	memar	1.0	Tepat	memar
31.	mimisan	mimisan	1.0	Tepat	mimisan
32.	lemas	lemas	1.0	Tepat	lemas
33.	lesu	lesu	1.0	Tepat	lesu
34.	sembelit	sembelit	1.0	Tepat	sembelit
35.	kejang	bejan	0.727	Tidak Tepat	bejan
36.	kaku	aaku	0.75	Kurang Tepat	kaku
37.	pingsan	kinisan	0.714	Tidak Tepat	kinisan
38.	kesemutan	kesmtan	0.875	Kurang Tepat	kesemutan
39.	flu	dlyu	0.571	Tidak Tepat	dlyu
40.	gemetar	benmetr	0.714	Tidak Tepat	benmetr
41.	sakit	raki	0.667	Tidak Tepat	raki
42.	melepuh	melepuh	1.0	Tepat	melepuh
43.	bentol	kentol	0.833	Kurang Tepat	bentol
44.	kesel	kesl	0.889	Kurang Tepat	kesel
45.	watuk	patuk	0.8	Kurang Tepat	watuk
46.	enek	pemeu	0.444	Tidak Tepat	pemeu
47.	puyeng	peyang	0.667	Tidak Tepat	peyang
48.	meriyang	periyang	0.875	Kurang Tepat	meriyang
49.	mumet	pume	0.667	Tidak Tepat	pume
50.	nggreges	negreges	0.875	Kurang Tepat	nggreges
Rata-rata Kemiripan			0.788		