SPEECH RECOGNITION MENGGUNAKAN ELMAN RECURRENT *NEURAL NETWORK* UNTUK KATA DALAM BAHASA INDONESIA

Wildan Kurniadi[1], Irfan Maliki[2]

[1,2] Teknik Informatika - Universitas Komputer Indonesia
Jalan Dipatiukur No 112-116 Bandung, 40132
E-mail : kurniadiwildan@email.unikom.ac.id [1], irfan.maliki@email.unikom.ac.id [2]

ABSTRAK

Speech Recognition merupakan salah satu cabang ilmu komputer yang masih terus dikaji hingga saat ini. Dalam sebuah *speech recognition* diperlukan sebuah algoritma klasifikasi untuk mengenali makna dari sinyal suara. Salah satu algoritma klasifikasi yang dapat digunakan pada *speech recognition* adalah *Elman Recurrent Neural network* (ERNN). ERNN telah dicoba pada beberapa penelitian mengenai *speech recognition* dalam Bahasa Inggris, Bahasa Arab, dan Bahasa Indonesia dalam kasus kata yang berbeda-beda dan menghasilkan akurasi yang dapat dikatan baik yaitu diantara 87.5% hingga 99.47%.

Pada penelitian ini ERNN digunakan untuk mengenali sepuluh kata verba dengan *rank* paling tinggi menurut *Leipzig Corpora Collection.* Dataset diambil dari 40 pembicara berjenis kelamin laki-laki dimana setiap orang akan mengucapkan 10 kata dan diulang 4 kali sehingga didapatkan 1600 dataset (40 × 10 × 4 = 1600). Dataset yang dimiliki disimpan dalam bentuk file .wav dengan frekuensi sampling 16000 KHz. Data latih yang digunakan adalah 800 sampel data (20 pembicara ×10 kata × 4 repetisi=800) dan data uji adalah 800 sampel data dari 20 orang yang berbeda dengan data latih. Adapun metode ekstraksi ciri yang digunakan penelitian ini adalah MFCC, akurasi terbaik yang dihasilkan oleh penelitian ini sebesar 83.88%..

Kata kunci : *Speech Recognition*, Bahasa Indonesia, *Elman Recurrent Neural network*, MFCC.

1. PENDAHULUAN

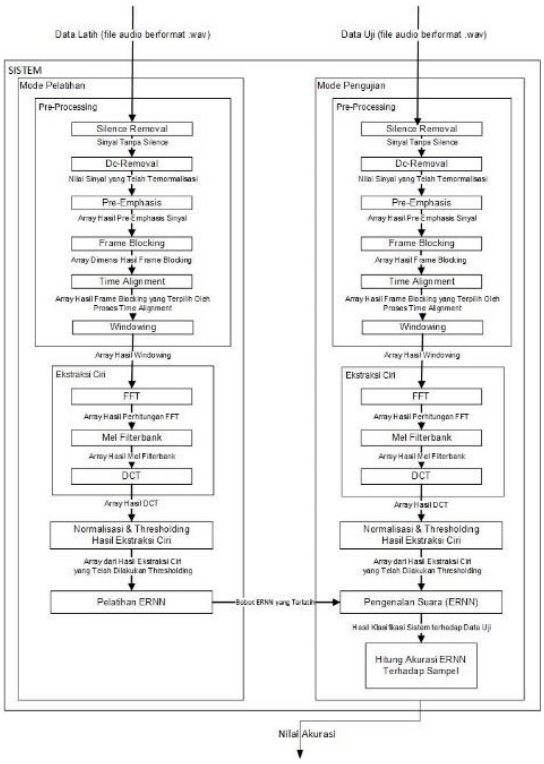
Speech Recognition adalah proses menginterpretasikan ucapan manusia agar dikenali maknanya oleh komputer [1]. Dalam sebuah sistem speech recognition diperlukan sebuah algoritma klasifikasi untuk mengenali makna dari sinyal suara. Terdapat berbagai macam algoritma klasifikasi yang dapat digunakan pada sebuah speech recognition, salah satunya adalah Elman Recurrent *Neural network* (ERNN).

Elman Recurrent *Neural network* (ERNN) dapat bekerja dengan baik dalam speech recognition dikarenakan kemampuannya untuk mengenali pola yang berbeda berdasarkan waktu[2]. Terdapat beberapa penelitian sebelumnya yang menggunakan ERNN sebagai algoritma klasifikasi dalam sistem speech recognition. Diantaranya adalah penelitian yang dilakukan Robert B. Allen mengenai ERNN untuk mengenali ucapan berupa kata dalam Bahasa Inggris yang menghasilkan akurasi mencapai 87.5% [3]. Pada penelitian lainnya yang dilakukan oleh Yousef Alotaibi [4], ERNN menghasilkan akurasi 99.47% dan 96.46% untuk mengenali ucapan berupa angka dalam Bahasa Arab. Dengan bahasa yang sama namun kasus yang berbeda penelitian yang dilakukan oleh M. M. El Choubassi menyatakan bahwa ERNN mampu menghasilkan akurasi 90% pada sistem Automatic Speech Recognition (ASR) yang dapat mengenali 6 kata dalam Bahasa Arab [5]. Pada penelitian yang dilakukan Muhammad Fachrie [6], ERNN mampu menghasilkan akurasi paling tinggi sebesar 99.30% pada kasus pengenalan ucapan digit 0 sampai 9. Dari beberapa penelitian tersebut ERNN terlihat mampu memberikan hasil yang baik ketika digunakan untuk speech recognition dalam beberapa kasus yang berbeda, namun performa dari ERNN untuk kasus speech recognition kata dalam Bahasa Indonesia belum diketahui.

Berdasarkan uraian di atas, maka pada penelitian ini akan menggunakan ERNN sebagai algoritma klasifikasi pada speech recognition dengan kasus kata dalam Bahasa indonesia. Harapan dari penelitian ini adalah untuk mengetahui bagaimana akurasi yang dihasilkan ERNN untuk mengenali ucapan berupa kata verba dalam Bahasa Indonesia.

2. METODE

Sistem *speech recognition* yang akan dibangun terdiri dari beberapa proses utama, yaitu *pre-processing*, Ekstraksi Ciri, Normalisasi dan Thresholding, dan bagian ERNN(Pelatihan ataupun pengujian, tergantung mode yang digunakan). Secara umum gambaran sistem yang dibangun dapat dilihat pada gambar 1.



Gambar 1. Gambaran Umum Sistem

Data masukan (data latih dan data uji) merupakan file audio hasil rekaman pembicara menyebutkan satu kata dalam durasi 1 detik dengan format .wav

Pada mode pelatihan akan dilakukan *preprocessing* terhadap data suara yang telah dipilih dari bagian sebelumnya, dimana dalam *preprocessing* dilakukan *silence removal*, *DC removal*, *pre emphasis*, *frame blocking* dan *windowing*. Setelah melewati tahap *preprocessing* maka akan dilakukan ekstraksi ciri melalui tahap DFT, *Mel Frequency Warping*, dan DCT. Pada hasil keluaran DCT yang berupa *mel frequency cepstrum coefficient* akan dilakukan normalisasi untuk mendapatkan nilai dengan persebaran 0 sampai 1. Setelah dilakukan normalisasi akan dilakukan *thresholding* terhadap nilai hasil normalisasi, supaya nilai dari hasil ekstraksi ciri nantinya akan bernilai 0 atau 1 sehingga dapat dilakukan pelatihan menggunakan ERNN.

Pada mode pengujian akan dilakukan *preprocessing*, *ekstraksi fitur* dan normalisasi dan thresholding hasil ekstraksi ciri kemudian dilakukan pengenalan terhadap hasil ekstraksi ciri dari suara yang dimasukan menggunakan ERNN yang bobotnya telah dilatih pada tahap pelatihan sebelumnya. Hasil dari pengujian ini adalah sistem akan menampilkan nilai akurasi dari proses pengenalan ucapan kata dalam Bahasa Indonesia.

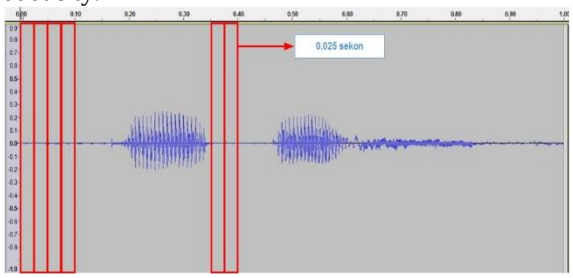
2.1 Pre-processing

Pre-processing pada data masukkan bertujuan untuk menghasilkan nilai sinyal yang baik untuk dilakukan ekstraksi ciri. Proses yang dilakukan pada pre-processing terdiri dari 6 tahap yang terdiri dari *Silence Removal*, DC *Removal*, *Pre Emphasis*, *Frame Blocking*, *Time Alignment*, *Windowing*.

a. Silence Removal

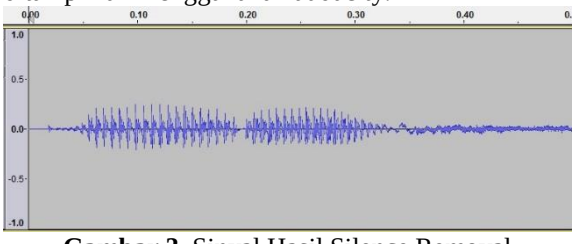
Silence removal merupakan tahap untuk menghilangkan sinyal *silence* (sinyal tanpa suara ucapan). Proses *silence removal* yang dilakukan pada penelitian ini mengadaptasi dari penelitian lainnya[7]. *Silence removal* cukup penting karena akan mempengaruhi jumlah data yang nantinya akan diproses pada tahap selanjutnya. *Silence removal* dilakukan dengan cara manual, yaitu membagi sinyal kedalam beberapa bagian(*frame*) yang memiliki panjang tertentu, kemudian akan membuang sinyal tersebut bila nilai maksimal sampel pada *frame* tersebut bernilai dibawah *threshold* yang telah ditentukan.

Pada gambar 2, dapat terlihat sinyal audio seorang pembicara menyebutkan kata “adalah”, sinyal tersebut dilihat menggunakan aplikasi audacity.



Gambar 2. Pembagian sinyal kedalam *frame-frame*

Sinyal pada gambar 2 akan dibagi kedalam beberapa blok *frame* (misal 0,025 sekon untuk setiap *frame*). Kemudian akan dilakukan pengecekan terhadap setiap *frame*, jika didalam *frame* tersebut ada sinyal yang bernilai di atas *threshold* maka *frame* tersebut akan dianggap bukan *silence frame* dan disimpan kedalam array sinyal yang baru., sebaliknya jika tidak ada sinyal dengan nilai melebihi threshold maka *frame* tertsebut akan dianggap sebagai *silence frame* dan tidak digunakan pada proses berikutnya. Setelah proses pemilihan *silence frame* telah selesai, maka sinyal baru yang didapatkan dari kumpulan *frame* tanpa sinyal *silence* disatukan. Hasil *silence removal* dari sinyal audio pada gambar 2 dapat dilihat pada gambar 3 (gambar ditampilkan menggunakan audacity.



Gambar 3. Sinyal Hasil Silence Removal

b. DC *Removal*

Setelah melakukan *silence removal* maka tahap selanjutnya yang akan dilakukan adalah *DC removal*. Proses *DC removal* dilakukan dengan

menghitung nilai rata-rata dari data sampel suara dan mengurangkan nilai setiap sampel suara dengan nilai rata-rata tersebut[8]. Adapun untuk menghitung hasil dari DC Removal digunakan persamaan (1).

$$dc_i = s_i - \bar{x}\ ,\ 0 \le i \le N\text{-}1 \quad (1)$$

Dimana

dc_i = titik sinyal hasil proses *DC removal*
s_i = titik sinyal ke-i
$\bar{x}$ = nilai rata-rata input titik sinyal.
i = 0,1,2,…, N-1
N = panjang sinyal (banyaknya titik sinyal)

Dimana untuk mendapatkan nilai rata-rata input titik sinyal dgunakan persamaan sebagai berikut.

$$\bar{x} = \frac{s_0+s_1+s_2+\cdots+s_{N-1}}{N-1} \quad (2)$$

Dengan keterangan :

$\bar{x}$ = nilai rata-rata dari titik sinyal input
s_i = input sinyal ke-i
i = 0,1,2,…, N-1

c. Pre Emphasis

Setelah melakukan DC Removal, tahap selanjutnya yang akan dilakukan adalah tahap *pre emphasis*. *Pre-emphasis* merupakan sebuah filter yang digunakan untuk memperbaiki sinyal dari gangguan *noise*[8].

$$p_i = dc_i + pf_i \quad (3)$$

Dengan keterangan :

p_i = titik sinyal baru ke-i
dc_i = sampel sebelum *pre emphasis* ke-i (hasil DC Removal)
pf_i = *pre emphasis filter* pada sampel ke-i
i = 0,1,2,…, N-1
N = panjang sinyal (banyaknya titik sinyal)

Dimana untuk mendapatkan nilai pf_i didapatkan dari persamaan berikut.

$$pf_i = dc_i - (dc_{i-1} \times \alpha) \quad (4)$$

dengan keterangan :

pf_i = sinyal hasil pre-emphasize filter ke-i
dc_i = sinyal sebelum pre-emphasize filter ke-i
dc_{i-1} = sinyal sebelum pre-emphasize filter ke i-1
α = pre-emphasize filter
i = 0,1,2,…, N-1
N = panjang sinyal (banyaknya titik sinyal)

d. Frame Blocking

Proses yang dilakukan pada tahap *frame blocking* adalah membagi titik-titik sinyal kedalam beberapa *frame*, dimana setiap *frame* memiliki ukuran *sample point* yang sama, selain itu setiap *frame* melakukan *overlapping* dengan *frame* lainnya. Jarak *overlapping* antar *frame* yang bisa digunakan berkisar antara 1/2 sampai 1/3 panjang *frame* (*sample point*). Pada penelitian ini *overlapping* yang akan digunakan adalah 1/2 dari panjang *frame* (*sample point*) dan akan disimbolkan dengan huruf M. Adapun untuk mendapatkan nilai SP dilakukan dengan cara mengalikan *sample rate* dengan durasi *frame* dalam satuan detik (s). Pada penelitian ini durasi *frame* yang akan digunakan adalah 0.025 sekon sedangkan nilai *sample rate* adalah 16000. Adapun untuk menghitung jumlah *frame blocking* adalah sebagai berikut.

$$I = ((NPE\text{-}SP)/M)+1) \quad (5)$$

Dengan keterangan :

I = Jumlah *frame*
NPE = Jumlah sampel (banyak sampel yang akan disegmentasi)
SP = *Sample point* dalam tiap *frame* (*Sample rate* × panjang *frame* dalam detik(s))
M = SP/2 (*Overlapping*)

e. Time Alignment

Setelah sinyal ucapan telah dimasukkan ke dalam bentuk *frame*, langkah selanjutnya yang harus dilakukan adalah melakukan *time alignment* (pemilihan *frame* yang akan digunakan). Tujuan dari *time alignment* adalah untuk menenetukan jumlah *frame* yang akan didapatkan koefisien mel frekuensinya sesuai jumlah CF. *Time alignment* akan dilakukan dengan menentukan CF (*frame* yang diinginkan), SF (awal *frame* yang diinginkan), dan EF (akhir *frame* yang diinginkan). Adapun nilai SF didapatkan dengan mengalikan nilai SFD (nilai SF yang diinginkan dalam desimal) dengan total keseluruhan *frame* yang ada, sedangkan EF didapatkan dari nilai EFD (nilai EF yang diinginkan dalam desimal) dengan total keseluruhan *frame* yang ada, dimana nilai SFD merupakan bilangan desimal bernilai 0 sampai 1 dengan ketentuan nilai SFD ditambah EFD akan bernilai 1[7].

Berikut merupakan perhitungan jika dimisalkan CF adalah 5, jumlah *frame* yang dimiliki adalah 35 *frame*, dan SFD adalah 0.06, (sehingga EFD = 0.94)

CF = 5

$$SF = F \times SFD \quad (6)$$

= 35 × 0.06 = 1.4 = 1 (dibulatkan ke bilangan terdekat)

$$EF = F \times EFD \quad (7)$$

= 35 × 0.94 = 33.6 = 34 (dibulatkan ke bilangan terdekat)

f. *Windowing*

Frame yang terpilih pada tahap *time alignment* akan menjadi *frame* ke-0 sampai *frame* ke-CF pada *frame* yang baru. Setelah tahap *time alignment* dilakukan maka akan dilakukan *windowing*. *Windowing* bertujuan untuk mengatasi kebocoran spektral. Adapun perhitungan *windowing* dilakukan terhadap *sample point* pada tiap *frame* dengan mengalikan fungsi *hamming window* dengan nilai *sample point*. Perhitungan tersebut dapat dilihat pada persamaan berikut.

$$w_i = p_i \,.\, fw_i \quad (8)$$

Dengan keterangan :

w_i = nilai titik sinyal hasil *windowing*
p_i = hasil pre emphasis ke-l dari *frame* ke-i
fw_i = fungsi *window* ke-l untuk *frame* ke-i

NPE = banyaknya *sample point* pada *frame*
l = *sample point* ke-l pada *frame* (dimana nilai n = 0,1,…,NPE-1)

Dimana untuk mendapatkan nilai fw_i dapat menggunakan persamaan berikut.

$$fw_i = 0.54 - 0.46\ cos\frac{2\pi i}{NSP-1} \quad (9)$$

Dengan keterangan :

fw_i = fungsi *hamming window* untuk titik sinyal ke-l pada *frame* ke-i
SP = *sample point* (dapat disebut sebagai panjang *frame)*
l = 0,1,...,SP-1

2.2 Ekstraksi Ciri

Sinyal yang telah melewati tahap pre-processing akan diproses untuk didapatkan nilai ekstraksi cirinya. Tujuan dari ekstraksi ciri adalah untuk mendapatkan fitur-fitur penting yang nantinya akan diolah untuk dikenali oleh ERNN. proses yang dilakukan pada tahap ekstraksi ciri terdiri dari DFT, *Mel Frequency Warping*, DCT.

1. DFT

Setelah dilakukan *windowing* terhadap *frame* yang dipilih, maka proses selanjutnya yang akan dilakukan adalah *Discrete Fourier Transform* (DFT). *Discrete Fourier Transform* akan mentransformasikan sinyal diskrit pada domain waktu menjadi sinyal diskrit pada domain frekuensi. Persamaan pada DFT dapat dilihat sebagai berikut[8]

$$ft_k = \sum_{i=0}^{SP-1}\left(w_i\ cos\frac{2\pi ik}{SP}\right) - j\sum_{i=0}^{SP-1}\left(w_i sin\frac{2\pi ik}{SP}\right) \quad (10)$$

Dimana untuk mendapatkan hasil dari DFT digunakan persamaan berikut.

$$|ft_k| = [R^2 + I^2]^{1/2} \quad (11)$$

Hasil dari dft nantinya akan disebut sebagai magnitude dft dan akan dipetakan kedalam skala mel.

2. Mel Frequency Warping

Magnitude yang telah dihasilkan proses DFT akan melalui tahap *filterbank*. Pada tahap ini akan dilakukan filter terhadap suara yang masuk, dimana *filterbank* yang digunakan akan mengikuti skala *mel* (skala pendengaran manusia). Untuk mendapatkan nilai hasil *Mel Frequency Warping*, digunakan persamaan berikut.[xx]

$$MF_m = \sum_{k=0}^{SP-1} ft_k \times H_{mk} \quad (12)$$

Dimana :

MF_m = Hasil Mel Frequency Warping ke-*m*
ft_k = Hasil DFT ke-*k*
H_{mk} = koefisien dari mel *filterbank*
SP = Banyaknya data hasil DFT
m = 0,1,2,..,*Nfbank*
Nfbank = Jumlah *Mel Filterbank*

Dimana koefisien *mel filterbank* didapatkan dengan cara membuat sebuah *triangular filterbank*, menggunakan persamaan *mel scale*[8].

$$mel\ (f) = 2595 \times log10(1 + \frac{f}{700}) \quad (13)$$

dan inverse mel scale

$$mel^{-1}(f) = 700 \times (10^{\frac{f}{2595}} - 1) \quad (14)$$

Kemudian tentukan *filterbank boundary* yang akan digunakan, yaitu sebanyak jumlah segitiga pada *filterbank* yang akan dibuat. Untuk menghitung nilai *filterbank boundary* dapat digunakan persamaan berikut[9].

$$fb_m = \left(\frac{SP}{FS}\right) \times mel^{-1}\left(mel(Nlow) + m\frac{mel(Nhigh)-mel(Nlow)}{Nfbank}\right) \quad (15)$$

Dimana :

fb_m = hasil *boundary point filter*
SP = banyaknya sample point
FS = Frekuensi Sampling

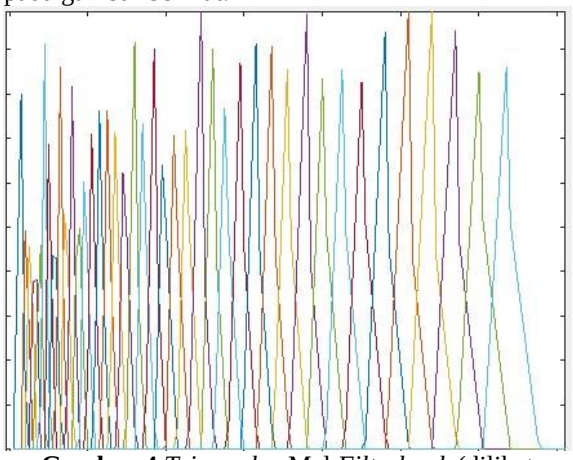
Setelah nilai *filterbank boundary* didapatkan, selanjutnya dengan menggunakan aturan pada persamaan dibawah maka akan dihasilkan *triangular filterbank* sebanyak jumlah *filterbank boundary*.

$$H_{mk} = \begin{cases} 0,\ untuk\ k < fb_m \\ \frac{k - fb_{m-1}}{fb_m - fb_{m-1}},\ untuk\ fb_{m-1} \le k < fb_m \\ \frac{fb_{m+1} - k}{fb_{m+1} - fb_{m-1}},\ untuk\ fb_m \le k < fb_{m+1} \\ k > fb_{m+1} \end{cases} \quad (16)$$

Dimana :

fb_m = hasil *boundary point filter*
H_{mk} = koefisien dari mel *filterbank*

Sehingga jika langkah-langkah pada pembuatan *triangular filterbank* dilakukan maka, akan dihasilkan sebuah *triangular filterbank*, pada penelitian ini *triangular filterbank* yang diguanakan adalah 40, hal ini merujuk pada penelitian sebelumnya [6], [7], dengan demikian *triangular filterbank* pada penelitian ini dapat dilihat seperti pada gambar berikut.



Gambar 4 *Triangular* Mel *Filterbank* (dilihat menggunakan matlab)

3. DCT

DCT merupakan proses terakhir yang dilakukan pada bagian ekstraksi ciri. DCT akan melakukan kompresi terhadap sinyal hasil *Mel Frequency Warping*, DCT dikenal baik untuk melakukan kompresi pada nilai-nilai sinyal.

$$C_{koef} = \sqrt{\frac{2}{Nfbank}}\sum_{m=0}^{Nfbank-1}(\log MF_m)\cos\left[koef\left(\frac{2m-1}{2}\right)\frac{\pi}{Nkoef}\right] \quad (17)$$

2.3 Normalisasi dan Thresholding

Hasil ekstraksi ciri yang telah didapatkan perlu dilakukan normalisasi dan thresholding agar dapat diproses oleh ERNN. hal ini karena *neural network* akan lebih mudah untuk dilatih dengan nilai diskrit daripada kontinu.

1. Normalisasi

Normalisasi adalah penskalaan terhadap nilai-nilai masuk ke dalam suatu *range* tertentu. Hal ini dilakukan agar nilai *input* dan target *output* sesuai dengan range dari fungsi aktivasi yang digunakan dalam jaringan. Normalisasi dilakukan dengan menggunakan persamaan berikut [10].

$$normal_{koef} = \frac{0.8(C_{koef}-a)}{b-a} + 0.1 \quad (18)$$

2. Thresholding

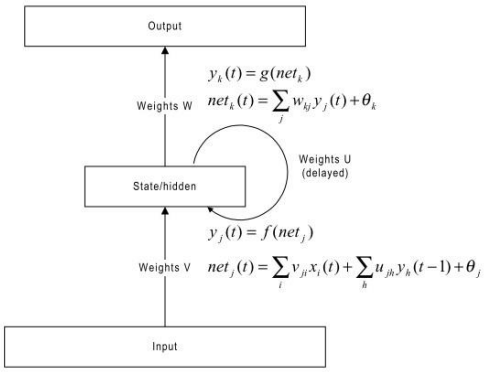
Thresholding dilakukan untuk merubah nilai dari hasil normalisasi menjadi bernilai 0 atau 1. Hal ini dilakukan karena *neural network* lebih mudah dilatih dengan data diskrit dibanding kontinu [11]. Cara kerjanya adalah memberikan nilai *threshold* (ambang batas) bila nilai dari data hasil normalisasi berada di atas nilai *threshold* maka nilai tersebut akan dijadikan 1, sebaliknya, jika nilai hasil normalisasi dibawah *threshold* maka nilai tersebut akan dijadikan 0.

2.4 *Elman Recurrent Neural network* (ERNN)

Pada bagian *Elman Recurrent Neural network* (ERNN) terdapat dua jenis proses yang dapat dilakukan yaitu, pelatihan dan pengujian.

1. Pelatihan

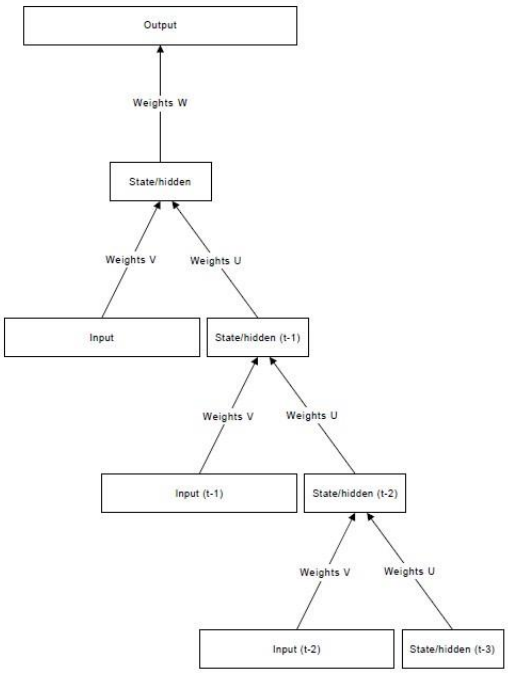
Pelatihan yang digunakan pada penelitian ini merujuk kepada Boden [12]. Secara sederhana sebuah *recurrent neural network* dapat digambarkan seperti pada Gambar 5. Dimana setiap *hidden neuron* / *state neuron* akan digunakan sebagai input pada *timestep* berikutnya (dalam hal ini, *hidden neuron* sebelumnya disebut sebagai *context neuron* dan layernya disebut *context layer*). Fungsi yang digunakan pada elman recurrent *neural network* pun sebenarnya hampir sama dengan *neural network* lainnya yaitu didapatkan dari jumlah perkalian antara nilai input neuron dengan bobotnya, hanya saja pada *elman recurrent neural network* terdapat perbedaan yaitu dengan menambahkan nilai jumlah perkalian antara nilai *hidden neuron* pada *timestep* sebelumnya (*context neuron*) dengan bobot *context*-nya.



Gambar 5 Gambaran Sederhana *ERNN*

Jika dibayangkan, sebenarnya di dalam Gambar 5 terdapat beberapa *input layer* dan *hidden layer*. Jumlah *input layer* dan *hidden layer* yang dimiliki oleh ERNN bergantung pada banyaknya jumlah *timestep* yang dimiliki oleh arsitektur ERNN tersebut. Sehinggga jika *timestep* yang dimiliki oleh ERNN adalah 3 maka pada arsitektur *neural network* yang dibangun akan terdapat 3 *input layer* dan 3 *hidden layer*, hal ini dapat dilihat pada Gambar 6.

Pelatihan pada ERNN dilakukan dengan melakukan *error backpropagtion* terhadap setiap bobot mulai dari bobot W, dan juga bobot V dan U pada setiap *timestep*(mulai dari *timestep* terakhir sampai pertama). Karena *backpropagation* pada *recurrent neural network* dilakukan melewati *timestep* tertentu, maka proses ini disebut dengan *backpropagation through time* (BPTT). Prinsip dasar dari BPTT adalah adanya "*unfolding*" [12](yang terjadi pada gambar 5 adalah hasil *unfolding*).



Gambar 6 ERNN Setelah *Unfolding*

Dengan adanya prinsip *unfolding* berarti setiap bobot *recurrent* (bobot V dan U) perlu disimpan dalam memori agar dapat dilatih (*error* hasil pelatihan akan disimpan di dalam memori).

Perhitungan yang terjadi di dalam ERNN dijelaskan sebagai berikut. Untuk meghitung nilai pada neuron hidden di *timestep* tersebut digunakan persamaan 19.

$$net_j(t) = \sum_i v_{ji} x_i(t) + \sum_h u_{jh} y_h(t-1) + \theta_j \quad (19)$$

Dimana :

$net_j(t)$ = *hidden layer vector* pada *timestep* ke-t

v_{ji} = bobot dari neuron *input* ke-i menuju *neuron* output ke-j

x_i = nilai neuron input ke-i

u_{jh} = bobot dari neuron *context* ke-h menuju *neuron* output ke-j

y_h = nilai neuron *context* ke-h (nilai neuron *hidden* ke-h pada *timestep* sebelumnya)

t = *timestep*

Selanjutnya nilai pada *hidden neuron* perlu diaktivasi menggunakan *activation function* untuk dapat digunakan. Pada penelitian ini *activation function* yang digunakan adalah tanh (*hyperbolic tangent*). Fungsi tanh dapat dilihat pada persamaan 20.

$$net_j(t) = \frac{e^{2 \times net_{j-1}}}{e^{2 \times net_{j+1}}} \quad (20)$$

Adapun turunannya adalah sebagai berikut.

$$\text{dnet}_j = (1 - net_j) \times (1 + net_j) \quad (21)$$

Setelah dilakukan aktivasi terhadap nilai setiap *hidden neuron* pada *timestep* tersebut, jika *timestep* bukan *timestep* terakhir maka hasil aktivasi *hidden neuron* tersebut akan menjadi nilai input pada *timestep* berikutnya (menjadi *context neuron*). Jika *timestep* merupakan *timestep* terakhir maka nilai hidden neuron yang telah diaktivasi akan digunakan untuk mendapatkan nilai *output neuron* pada *output layer*. Pada *output layer* digunakan persamaan berikut untuk medapatkan nilai pada *output neuron*.

$$net_k(t) = \sum_j w_{kj}\, y_j(t) + \theta_k \quad (22)$$

Seperti pada *hidden neuron*, neuron pada *output layer* juga perlu diaktivasi, hanya saja pada *output layer* jenis *activation function* yang digunakan bukan tanh lagi, tetapi softmax. Hal ini karena fungsi aktivasi *softmax* merupakan fungsi aktivasi yang dapat merepresentasikan distribusi kategoris. Adapun funsgi aktivasi *softmax* adalah sebagai berikut.

$$net_k = \frac{e^{net_k}}{\sum_{k=1}^{K} e^{net_k}} \quad (23)$$

Adapun untuk mendapatkan nilai turunan dari *softmax activation function* digunakan persamaan 24.

$$dnet_k = \left(net_k \times (1 - net_k)\right) \times (target_k - net_k) + \theta_k \quad (24)$$

Dengan menggunakan persamaan yang telah disebutkan sebelumnya, maka pelatihan ERNN ERNN menggunakan *Backpropagation Through Time* dijelaskan sebagai berikut.

1. Inisialisasi bobot dengan nilai random untuk setiap bobot antara neuron input-hidden, context-hidden, hidden-output, dan bias-biasnya.
2. Tentukan nilai hyperparameter seperti learning_rate, max_*epoch*.
3. Lakukan langkah no. 4 sampai no. 8 sebanyak max_*epoch* kali.
4. Hitung secara *feedforward*.
   Dalam proses perhitungan *feedforward* gunakan persamaan 20 untuk menghitung nilai pada *hidden neuron*, kemudian gunakan persamaan 21 untuk menghitung hasil aktivasi dari tiap-tiap *hidden neuron*, jika proses perhitungan untuk setiap nilai *hidden neuron* telah dilakukan maka hitung nilai pada setiap *output neuron* dengan menggunakan persamaan 22 kemudian lakukan aktivasi terhadap setiap *output neuron* tersebut dengan menggunakan persamaan 23.
5. Fase Mundur
   Hitung niai error pada setiap *output neuron*, (nilai errror pada *output neuron* didapatkan dari selisih nilai output yang diharapkan dengan output yang dihasilkan pada setiap neuron) Gunakan persamaan 24.
   $\text{Error}_k = dnet_k$
   Hitung nilai error/gradient pada setiap hidden neuron, di setiap *timestep*. Adapun nilai error pada setiap hidden neuron dapat dihitung dengan menggunakan persamaan 25.
   $\text{Error}_j = \text{dnet}_j \times \text{sum}_j \quad (25)$
6. Untuk mendapatkan nilai dari dnet_j digunakan persamaan 21.
   Sedangkan untuk mendapatkan nilai sum digunakan persamaan 26 jika pada *timestep* terakhir.
   $\text{sum}_j = \sum_{k=0}^{k} Error_k \times w_{jk} \quad (26)$
   Atau menggunakan persamaan 27 jika bukan pada *timestep* terakhir.
   $\text{sum}_h = \sum_{j=0}^{j} j \times u_{jh} \quad (27)$
7. Setelah nilai error pada setiap hidden neuron di setiap *timestep* didapatkan maka hitung delta untuk setiap bobot menggunakan persamaan 28
   $\text{delta}_{ij} = \alpha \times \text{dgradient} \times \text{input} \quad (28)$
   dimana :
   α = Learning rate
   dgradient = nilai gradient (error) pada layer ke-j (Error_k atau Error_j)
   input = Nilai input dari neuron ke-i pada layer input
8. Lakukan update terhadap nilai bobot dengan cara nilai delta + nilai bobot asal.
   $v_{ji} = \text{delta}_{ji} + v_{ji} \quad (29)$
   $u_{jh} = \text{delta}_{jh} + u_{jh} \quad (30)$
   $w_{kj} = \text{delta}_{kj} + w_{kj} \quad (31)$

9. Fase pelatihan selesai.

2. Pengujian

Proses yang dilakukan pada tahap pengujian dilakukan dengan cara menghitung nilai input terhadap arsitektur ERNN yang telah dilatih. Yaitu dengan menghitung secara *feedforward*, sebagaimana pada proses no. 4 pada tahap pelatihan. Jika hasil pengenalan sesuai

3. HASIL PENGUJIAN

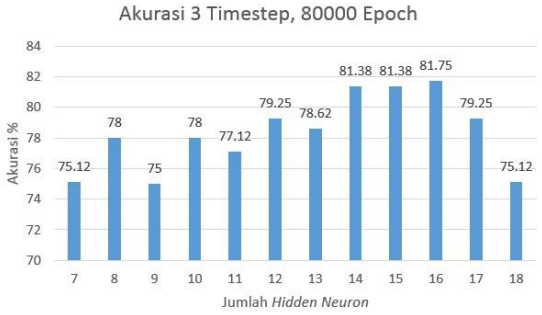
Pada penelitian ini pengujian dilakukan dengan merubah beberapa variabel yaitu pengaruh *timestep*, jumlah *hidden neuron* dalam satu layer, dan banyaknya *epoch* yang digunakan.

3.1 Hasil Pengujian Dengan *Timestep* 3

Pada sub bab ini akan hasil pengujian dari speech recognition yang menggunakan ERNN akan ditampilkan. Adapun banyaknya jumlah *hidden neuron* yang dicoba mulai dari 7 neuron per layer hingga 18 neuron per layer. Dan *epoch* yang digunakan adalah 80000, 100000, 120000, dan 150000.

a. *Epoch* 80000

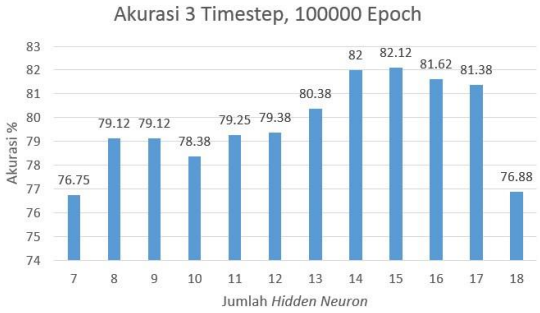
Adapun nilai hasil pengujian dengan *timestep* 3 dan *epoch* 80000 dapat dilihat pada Gambar 7. Pada grafik tersebut terlihat bahwa nilai akurasi tertinggi untuk *timestep* 3 dan epoh 80000 dicapai oleh arsitektur dengan jumlah *hidden neuron* 16 untuk setiap layernya (12-16-10), sedangkan akurasi minimum yang didapatkan pada jumlah *hidden neuron* 9 (12-9-10).



Gambar 7 Grafik Hasil Pengujian *Timestep* 3 *Epoch* 80000

b. *Epoch* 100000

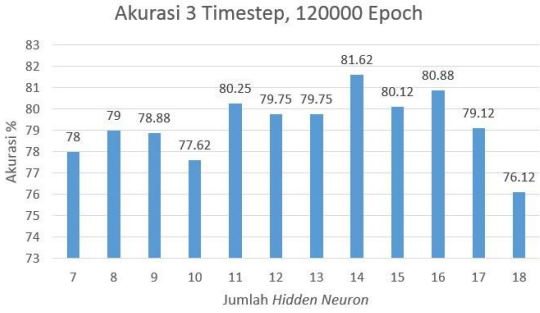
nilai hasil pengujian dengan *timestep* 3 dan *epoch* 100000 dapat dilihat pada Gambar 8. Pada grafik tersebut terlihat bahwa nilai akurasi tertinggi untuk *timestep* 3 dan epoh 100000 dicapai oleh arsitektur dengan jumlah *hidden neuron* 15 untuk setiap layernya (12-15-10), sedangkan akurasi minimum yang didapatkan pada jumlah *hidden neuron* 7 (12-7-10).



Gambar 8 Grafik Hasil Pengujian *Timestep* 3 *Epoch* 100000

c. *Epoch* 120000

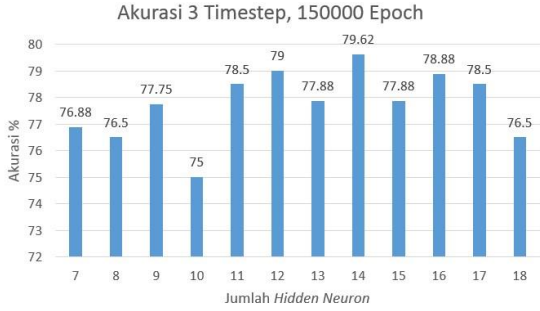
nilai hasil pengujian dengan *timestep* 3 dan *epoch* 120000 dapat dilihat pada Gambar 9. Pada grafik tersebut terlihat bahwa nilai akurasi tertinggi untuk *timestep* 3 dan epoh 120000 dicapai oleh arsitektur dengan jumlah *hidden neuron* 14 untuk setiap layernya (12-14-10), sedangkan akurasi minimum yang didapatkan pada jumlah *hidden neuron* 9 (12-18-10).



Gambar 9 Grafik Hasil Pengujian *Timestep* 3 *Epoch* 120000

d. *Epoch* 150000

nilai hasil pengujian dengan *timestep* 3 dan *epoch* 120000 dapat dilihat pada Gambar 10. Pada grafik tersebut terlihat bahwa nilai akurasi tertinggi untuk *timestep* 3 dan epoh 120000 dicapai oleh arsitektur dengan jumlah *hidden neuron* 14 untuk setiap layernya (12-14-10), sedangkan akurasi minimum yang didapatkan pada jumlah *hidden neuron* 10 (12-10-10).



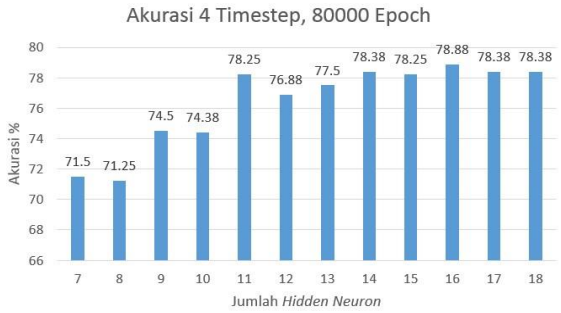
Gambar 10 Grafik Hasil Pengujian *Timestep* 3 *Epoch* 150000

3.2 Hasil Pengujian Dengan *Timestep* 4

Sama seperti pada pengujian dengan *timestep* 4, pengujian pada *timestep* 4 akan menguji bagaimana pengaruh jumlah *hidden neuron* per layer terhadap akurasi yang dihasilkan ERNN.

1. *Epoch* 80000

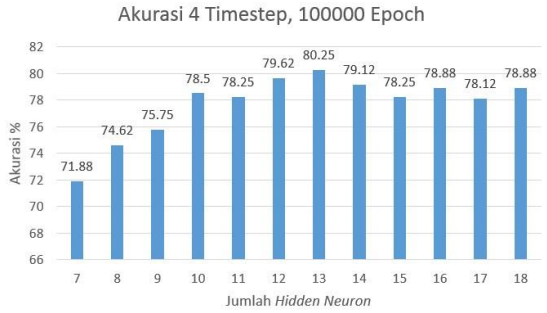
Nilai hasil pengujian dengan *timestep* 4 dan *epoch* 80000 dapat dilihat pada gambar xx. Pada grafik tersebut terlihat bahwa nilai akurasi tertinggi untuk *timestep* 3 dan epoh 80000 dicapai oleh arsitektur dengan jumlah *hidden neuron* 16 untuk setiap layernya (12-16-10), sedangkan akurasi minimum yang didapatkan pada jumlah *hidden neuron* 8 (12-8-10).



Gambar 11 Grafik Hasil Pengujian *Timestep* 4 *Epoch* 80000

2. *Epoch* 100000

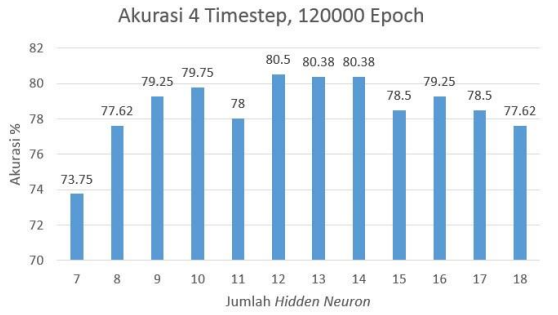
Nilai hasil pengujian dengan *timestep* 4 dan *epoch* 80000 dapat dilihat pada Gambar 12. Pada grafik tersebut terlihat bahwa nilai akurasi tertinggi untuk *timestep* 4 dan epoh 100000 dicapai oleh arsitektur dengan jumlah *hidden neuron* 13 untuk setiap layernya (12-13-10), sedangkan akurasi minimum yang didapatkan pada jumlah *hidden neuron* 7 (12-7-10).



Gambar 12 Grafik Hasil Pengujian *Timestep* 4 *Epoch* 100000

3. *Epoch* 120000

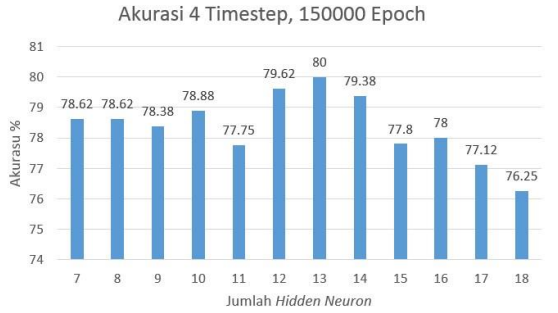
Nilai hasil pengujian dengan *timestep* 4 dan *epoch* 80000 dapat dilihat pada Gambar 13. Pada grafik tersebut terlihat bahwa nilai akurasi tertinggi untuk *timestep* 4 dan epoh 120000 dicapai oleh arsitektur dengan jumlah *hidden neuron* 12 untuk setiap layernya (12-12-10), sedangkan akurasi minimum yang didapatkan pada jumlah *hidden neuron* 7 (12-7-10).



Gambar 13 Grafik Hasil Pengujian *Timestep* 4 *Epoch* 120000

4. *Epoch* 150000

Nilai hasil pengujian dengan *timestep* 4 dan *epoch* 150000 dapat dilihat pada Gambar 14.



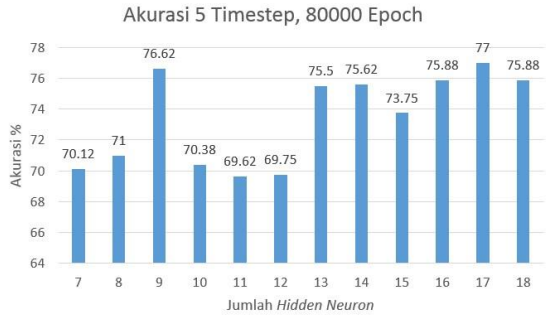
Gambar 14 Grafik Hasil Pengujian *Timestep* 4 *Epoch* 150000

Pada grafik tersebut terlihat bahwa nilai akurasi tertinggi untuk *timestep* 4 dan epoh 150000 dicapai oleh arsitektur dengan jumlah *hidden neuron* 13 untuk setiap layernya (12-13-10), sedangkan akurasi minimum yang didapatkan pada jumlah *hidden neuron* 18 (12-18-10).

3.3 Hasil Pengujian Dengan *Timestep* 5

1. *Epoch* 80000

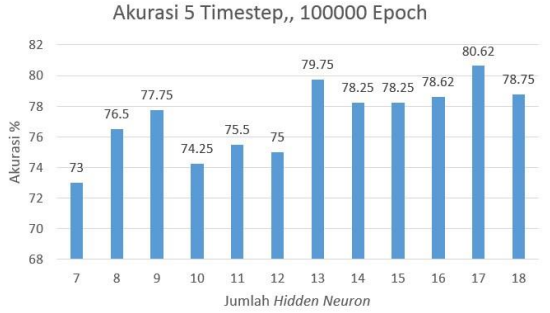
Nilai hasil pengujian dengan *timestep* 5 dan *epoch* 80000 dapat dilihat pada Gambar 15. Pada grafik tersebut terlihat bahwa nilai akurasi tertinggi untuk *timestep* 5 dan epoh 80000 dicapai oleh arsitektur dengan jumlah *hidden neuron* 17 untuk setiap layernya (12-17-10), sedangkan akurasi minimum yang didapatkan pada jumlah *hidden neuron* 11 (12-11-10).



Gambar 15 Grafik Hasil Pengujian *Timestep* 5 *Epoch* 80000

2. *Epoch* 100000

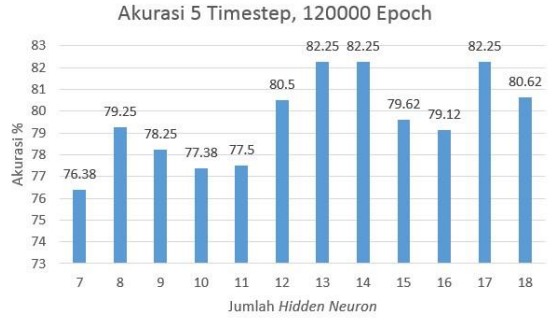
Nilai hasil pengujian dengan *timestep* 5 dan *epoch* 80000 dapat dilihat pada Gambar 16. Pada grafik tersebut terlihat bahwa nilai akurasi tertinggi untuk *timestep* 5 dan epoh 80000 dicapai oleh arsitektur dengan jumlah *hidden neuron* 17 untuk setiap layernya (12-17-10), sedangkan akurasi minimum yang didapatkan pada jumlah *hidden neuron* 7 (12-7-10).



Gambar 16 Grafik Hasil Pengujian *Timestep* 5 *Epoch* 100000

3. *Epoch* 120000

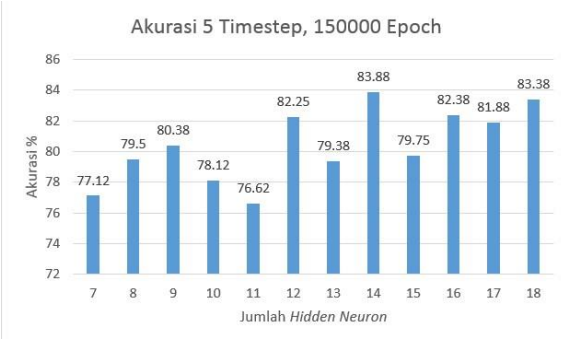
Nilai hasil pengujian dengan *timestep* 4 dan *epoch* 120000 dapat dilihat pada Gambar 17. Pada grafik tersebut terlihat bahwa nilai akurasi tertinggi untuk *timestep* 3 dan epoh 120000 dicapai oleh arsitektur dengan jumlah *hidden neuron* 12,13, dan 17, sedangkan akurasi minimum yang didapatkan pada jumlah *hidden neuron* 7 (12-7-10).



Gambar 17 Grafik Hasil Pengujian *Timestep* 5 *Epoch* 120000

4. *Epoch* 150000

Nilai hasil pengujian dengan *timestep* 4 dan *epoch* 150000 dapat dilihat pada gambar 18. Pada grafik tersebut terlihat bahwa nilai akurasi tertinggi untuk *timestep* 3 dan *epoh* 150000 dicapai oleh arsitektur dengan jumlah *hidden neuron* 14 untuk setiap layernya (12-14-10) dengan akurasi 83.88%, sedangkan akurasi minimum yang didapatkan pada jumlah *hidden neuron* 11 (12-11-10) dengan akurasi 76.62% .



Gambar 18 Grafik Hasil Pengujian *Timestep* 5 *Epoch* 150000

3.4 Hasil Pengujian Terbaik

Pengujian yang menghasilkan akurasi terbaik pada penelitian ini adalah ERNN dengan arsitektur 5 *timestep*, dengan 14 *hidden neuron* yang dilatih sebanyak 150000 *epoch* (12-14-10). Adapun akurasi yang dihasilkan oleh arsitektur tersebut adalah 83.88 %, untuk lebih detailnya hasil pengenalan terbaik tersebut dapat dilihat dalam bentuk *confussion matrix* seperti pada gambar 17.

		Kata Hasil Pengenalan										
	Kelas	Ada	Adalah	Beri	Bisa	Guna	Jadi	Laku	Milik	Rupa	Sebut	Akurasi
Label Kata dari Dataset	Ada	65	9	0	0	1	0	1	0	3	1	81.25%
	Adalah	8	60	0	0	4	0	6	1	1	0	75.00%
	Beri	0	0	63	0	1	5	1	10	0	0	78.75%
	Bisa	0	1	0	78	0	0	0	1	0	0	97.50%
	Guna	6	3	1	1	65	0	0	0	4	0	81.25%
	Jadi	0	0	5	0	0	71	0	4	0	0	88.75%
	Laku	0	0	1	1	0	0	71	0	5	2	88.5%
	Milik	0	0	2	0	2	3	0	72	1	0	90.00%
	Rupa	2	4	0	1	6	0	2	0	65	0	81.25%
	Sebut	0	4	1	0	1	2	8	1	2	61	76.25%
Total Akurasi												83.85%

Gambar 19 Confussion Matrix Hsil Pengenalan Terbaik

3.5 Analisis Hasil Pengujian

Dari hasil pengujian akurasi didapatkan hasil terbaik yaitu 83.88 % yang didapatkan dari pengujian dengan *timestep* 5 dengan 150000 *epoch*, kemudian hasil lainnya yaitu 82.12% jika *timestep* yang digunakan adalah 4 dan *epoch* 120000, begitu juga dengan penggunaan 3 *timestep* yang mampu menghasilkan akurasi terbaik di atas 80% meskipun jumlah *epoch* yang tidak terlalu banyak. Dari pengamatan tersebut terlihat bahwa hasil akurasi tertinggi memang didapatkan oleh *timestep* terbanyak (yaitu *timestep* 5) namun, hal tersebut juga diiringi oleh dibutuhkannya jumlah *epoch* yang

besar agar akurasi sistem dapat optimal. Sebaliknya, jumlah *timestep* paling kecil (*timestep* 3) selalu mampu menghasilkan akurasi terbaik di atas 80% dalam setiap *epoch*, namun akurasi yang mampu dihasilkan oleh *timestep* ini tidak terlalu berkembang meskipun *epoch*nya bertambah.

4. PENUTUP

Setelah melakukan pengujian terhadap ERNN untuk mengenali kata verba dalam Bahasa Indonesia dapat disimpulkan bahwa akurasi yang dihasilkan ERNN cukup baik yaitu mencapai angka 83.88%, hal ini tidak menutup kemungkinan bahwa akurasi bisa lebih baik lagi jika metode ekstraksi ciri lain digunakan, atau dilakukan modifikasi terhadap metode ekstraksi ciri yang digunakan.

Adapun saran dari penelitian ini yaitu untuk mencoba metode ekstraksi ciri yang lain seperti LPC atau RASTA-PLP, bisa juga dengan menambahkan delta dan delta log kedalam MFCC sebagai input pada ERNN (yang pada penelitian ini tidak digunakan), atau dengan penggunaan momentum saat melatih ERNN, ataupun menggunakan kata yang berbeda. Selain itu jenis *recurrent neural network* lain seperti LSTM (Long-Short Term Memory) juga dapat dicoba untuk kasus yang sama.

DAFTAR PUSTAKA

[1] M. Forsberg, "Why is Speech Recognition Difficult?," *Technology*, pp. 1–10, 2003.

[2] J. Heaton, *Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks*, vol. 3. 2015.

[3] R. B. Allen and C. a Kamm, "A Recurrent Neural Network for Word Identification from Continuous Phoneme Strings," Morristown, 1991.

[4] Y. A. Alotaibi, "Spoken Arabic Digit Recognizer Using Recurrent Neural Network," pp. 1–5, 2004.

[5] M. M. El Choubassi, H. E. El Khoury, C. E. J. Alagha, J. A. Skaf, and M. A. Al-Alaoui, "Arabic speech recognition using recurrent neural networks," *Proc. 3rd IEEE Int. Symp. Signal Process. Inf. Technol. (IEEE Cat. No.03EX795)*, pp. 543–547, 2003.

[6] M. Fachrie, "Robust Indonesian Digit Speech Recognition using Elman Recurrent Neural Network," pp. 49–54, 2015.

[7] Y. A. Alotaibi, "A Simple Time Alignment Algorithm for Spoken Arabic Digit Recognition," *JKAU*, vol. 20, pp. 29–43, 2009.

[8] A. D. Andriana, "METODE MEL FREQUENCY CEPSTRUM COEFFICIENTS Jurnal Ilmiah Komputer dan Informatika (KOMPUTA)," *J. Ilm. Komput. dan Inform.*, vol. 2, no. 1, pp. 21–26, 2013.

[9] S. Chen and Y. Luo, "Speaker verification using MFCC and support vector machine," *Proc. Int. MultiConference ...*, vol. I, pp. 18–21, 2009.

[10] Z. Ramadhan and S. N. Endah, "Perintah Suara Berbahasa Indonesia untuk Membuka dan Menutup Aplikasi dalam Sistem Operasi Windows Menggunakan Metode Mel Frequency Cepstrum Coefficient dan Metode Backpropagation," no. Snik, pp. 33–41, 2016.

[11] S. Muis, *Teknik Jaringan Syaraf Tiruan*. Yogyakarta: Penerbit Graha Ilmu, 2006.

[12] M. Boden, "A guide to recurrent neural networks and backpropagation," *Electr. Eng.*, no. 2, pp. 1–10, 2001.