# PETIR
JURNAL
PENGKAJIAN DAN PENERAPAN
TEKNIK INFORMATIKA

VOLUME 8 - NOMOR 1 | MARET 2015 | ISSN 1978-9262





ISSN 1978-9262
771978 926272

| SEKOLAH TINGGI TEKNIK - PLN (STT-PLN) | | | | | |
|---|---|---|---|---|---|
| PETIR | VOL. 8 | NO. 1 | HAL. 1 - 132 | JAKARTA, MARET 2015 | ISSN 1978-9262 |

# IMPLEMENTASI DAN ANALISA JARINGAN SARAF TIRUAN DENGAN FEATURE NORMALIZATION DAN PRINCIPAL COMPONENT ANALYSIS UNTUK DIGIT CLASSIFIER

**Nur Abdul Wahid (a), Sarwo, Adi Wahyu Setiawan**
(a) STMIK Mercusuar - Teknik Informatika

***A b s t r a k***

*Digit classifier yang dikembangkan, digunakan untuk mengklasifikasikan angka dari tulisan tangan. Sudah banyak sekali digit classifier yang dikembangkan dengan berbagai algoritma machine learning, salah satu yang populer dan terus berkembang adalah jaringan saraf tiruan. Dengan motivasi tersebut, penulis juga membangun digit classifier menggunakan jaringan saraf tiruan yang dipadukan dengan teknik optimisasi. Jenis arsitektur jaringan saraf tiruan yang akan digunakan adalah feedforward dan back propagation. Teknik optimisasi feature normalization dan principal component analysis (PCA) juga akan digunakan untuk meningkatkan performa model.*

*Set data yang digunakan untuk melatih model merupakan data Mixed National Institute of Standards and Technology (MNIST). Dengan memadukan jaringan saraf tiruan dan teknik-teknik optimisasi tersebut diharapkan dapat meningkatkan performa dan akurasi untuk mengklasifikasikan angka serta memberikan pemahaman baru bagi penulis.*

***Kata Kunci:*** *machine learning, klasifikasi, pengenal angka, jaringan saraf tiruan, feedforward, mnist, pca, normalisasi fitur.*

## 1. Pendahuluan

### 1.1 Latar Belakang

Klasifikasi merupakan salah satu kategori permasalahan dalam *machine learning*, yang merupakan salah satu teknologi mutakhir dan sangat menarik (Hamilton, 2014) (Kosner, 2013). Permasalahan tersebut dapat dipecahkan dengan membangun model *classifier*, dengan cara memetakan (mengklasifikasikan) data ke dalam satu atau beberapa kelas yang sudah didefinisikan sebelumnya. Teknik atau algoritma *machine learning* yang digunakan juga sangat beragam, seperti algoritma sederhana *logistic regression*, algortima yang sudah tua *neural networks* (jaringan saraf tiruan), *support vector machines* (SVM), *naïve bayes* dan *k-Nearest neighbor*.

*Digit classifier* merupakan *model classifier* yang berfungsi mengklasifikasikan data berupa gambar angka dari tulisan tangan ke dalam satu dari sepuluh kelas angka. *Digit classifier* adalah cikal bakal *handwriting recognition* (pengenal tulisan tangan). Aplikasi sistem serupa saat ini sudah sangat luas, seperti aplikasi pembaca nominal kertas cek di bank, pembaca alamat pada amplop surat di kantor pos, mesin pemindai dokumen, hingga *smartphone* yang sangat dekat dengan setiap individu. Hingga saat ini, telah banyak dikembangkan *digit classifier* dengan berbagai algoritma *machine learning*, salah satu yang populer dan terus berkembang adalah jaringan saraf tiruan.

Jaringan saraf buatan sejatinya merupakan teknologi lama. Sempat ditinggalkan karena proses *training* yang membutuhkan waktu yang sangat lama. Namun saat ini menjadi salah satu teknologi mutakhir dalam *machine learning* (Andrew Ng, 2012) (Russel dan Norvig, 2010). Dalam perkembangan *digit classifier*, jaringan saraf buatan standar yang diaplikasikan menghasilkan model *classifier* dengan akurasi rendah dan membutuhkan waktu *training* yang lama. Sehingga para pengembang melakukan banyak percobaan dengan memodifikasi jaringan saraf tiruan dari sisi arsitektur, seperti memilih jenis arsitektur, jumlah lapisan jaringan saraf buatan dan parameter-parameter yang berpengaruh. Selain itu percobaan juga dilakukan dengan mengkombinasikan jaringan saraf tiruan dengan teknik lain, baik pada tahap pra-proses data maupun tahap *training* (latih). Percobaan-percobaan tersebut ternyata memberikan hasil yang sangat baik, yang sebagian besar telah dibuat jurnal dan terkumpul di halaman web

http://yann.lecun.com/exdb/mnist/ (Sermanet et al, 2012) (Wan et al, 2013) (Hinton et al, 2006).

Dalam pengembangan sistem ini, penulis akan menggunakan jaringan saraf tiruan dengan arsitektur tiga lapisan jaringan, dengan kata lain memiliki satu *hidden layer*. Jenis algoritma *training* yang dipilih adalah *feed forward* dan *back propagation*. Selain itu, akan dilakukan juga kombinasi dengan teknik optimisasi pada tahap pra-proses data, yaitu *feature normalization* dan *principal component analysis* (PCA). Sementara data *training* dan data *test* yang akan digunakan merupakan data yang sangat terkenal dan sama dengan yang telah digunakan para pengembang di atas, yaitu data *Mixed National Institute of Standards and Technology* (MNIST). Objektif dari pengembangan sistem ini adalah memahami efek parameter-parameter jaringan saraf tiruan dan penerapan *feature normalization* serta PCA pada model *classfier* yang berhasil dibangun. Hasil yang diperoleh juga diharapkan dapat bersaing dengan hasil dari para pengembang profesional di dunia.

### 1.2 Rumusan Masalah

Berdasarkan dari uraian latar belakang masalah di atas, rumusan masalah yang diangkat adalah:

1) Bagaimana membangun *digit classifier* menggunakan jaringan saraf tiruan dipadukan dengan *feature normalization* dan PCA.
2) Bagaimana performa dan akurasi jaringan saraf tiruan yang dipadukan dengan *feature normalization* dan PCA.

### 1.3 Batasan Masalah

Untuk hasil penyelesaian masalah yang lebih terarah, maka perlu adanya pembatasan pembahasan sebagai berikut:

1) Pembahasan mencakup pembuatan model *digit classifier* dengan algoritma jaringan saraf buatan.
2) Data latih (*training data*) dan data cek atau data tes (*test data*) adalah gambar hitam putih berupa angka dari tulisan tangan yang diperoleh dari database *Mixed National Institute of Standards and Technology* (MNIST).
3) Data tes dapat pula berupa gambar yang diperoleh dari kamera secara langsung, namun tidak ada pembahasan mencakup pemrosesan gambar menjadi data yang sesuai dengan atribut model.
4) Peningkatan performa dengan teknik optimasi *feature normalization*, *principal component analysis* (PCA) dan memodifikasi beberapa konstanta model.
5) PCA merupakan salah satu teknik optimasi dalam *dimensionality reduction* yang diterapkan dengan menggunakan library dalam Octave, yaitu svd (*singular value decomposition*). Tidak akan dilakukan pembuktian tentang bagaimana kerja svd.
6) Model *classifier* berupa matriks parameter dalam bahasa pemrograman tingkat tinggi dan *open source*, Octave, kompatibel dengan Matlab.
7) Sistem yang dibangun berupa program *digit classifier* yang berdiri sendiri (*stand alone application*).

### 1.4 Tujuan dan Manfaat

Penelitian ini dilakukan dengan tujuan sebagai berikut:

1) Merancang dan membangun sistem digit classifier menggunakan jaringan saraf tiruan dipadukan dengan *feature normalization* dan PCA.
2) Melakukan analisa performa dan akurasi jaringan saraf tiruan yang dipadukan dengan *feature normalization* dan PCA.

### 1.5 Hipotesis

Dari hasil penelitian-penelitian yang terkumpul pada halaman *web database* MNIST http://yann.lecun.com/exdb/mnist/, jaringan saraf tiruan terbukti dapat dikembangkan, dimodifikasi dan dipadukan dengan beberapa teknik atau algoritma *machine learning* yang lain serta memberikan hasil yang baik. Hasil terburuk adalah *classifier* yang kembangkan oleh LeCun et al. pada 1998 dengan *test error rate* 4,7. Sementara hasil yang terbaik adalah *classifier* dengan enam lapisan jaringan oleh Ciresan et al. pada 2010 dengan *test error rate* 0,35.

Pada pengembangan sistem *digit classifier* ini, akan digunakan jaringan saraf tiruan yang dipadukan dengan *feature normalization* dan PCA. Dengan menerapkan teknik optimisasi tersebut, *classifier* yang berhasil dibangun harus memiliki hasil *test error rate* lebih baik dari 4,7 atau akurasi lebih baik dari 96%, dan diharapkan dapat bersaing dengan hasil lainnya.

## 2. Landasan Teori

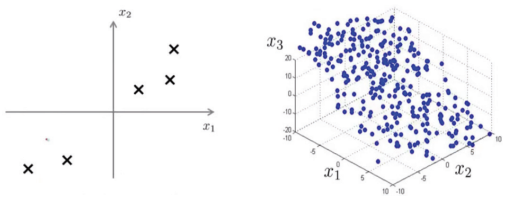
### 2.1 Data

Para peneliti dan pelaksana *machine learning*, Prof. Andrew Ng (2012) salah satunya, telah sepakat bahwa keberhasilan *machine learning* bukan semata-mata karena algoritma yang baik, namun data yang baik. Menurut Conway dan White (2012), akan sangat berguna saat pelaksana *machine learning* membagi data ke dalam dua kategori, yaitu *exploration* dan *confirmation*. Langkah dalam *exploratory* mencakup menyajikan data ke dalam table rangkuman dan melakukan visualisasi terhadap data. Kemudian melakukan analisa dan diharapkan menemukan sebuah pola, dan dapat mengurangi perhatian pada data yang secara kasat mata tidak memiliki pola. Dengan begitu, dipastikan pelaksana *machine learning* akan lebih mudah menentukan perlakukan yang sesuai terhadap data, dari awal hingga akhir. Data semacam ini disebut pula *training data* atau data latih.

Sementara analisa data dalam *confirmatory* mencakup:

1) Pengujian model *machine learning* yang diperoleh dari data *exploration* terhadap data baru, yaitu test data.
2) Menggunakan teori probabilitas untuk menguji, apakah model yang didapat berdasarkan data latih telah diperoleh dan menghasilkan output yang dikehendaki secara baik.

Data dapat diperoleh dari hasil pengamatan maupun percobaan. Representasi data juga dapat dilakukan dalam berbagai cara, seperti tabel, matriks atau histogram. Data yang digunakan dalam *machine learning*, khususnya data latih, umunya merupakan data dengan jumlah atau ukuran yang besar. Dengan alasan tersebut, vektorisasi dalam proses kalkulasi berpengaruh sangat baik karena proses berjalan jauh lebih cepat. Oleh karena itu, banyak data yang diubah ke dalam format numerik, dan dalam representasi matriks.




| ... | ... |
|---|---|
| "1" | 73.847017017515 |
| "0" | 58.9107320370127 |

| Name | Age |
|---|---|
| Drew Conway | 28 |
| John Myles White | 29 |

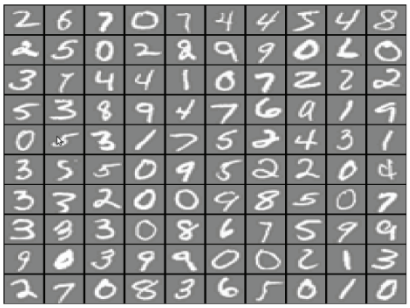
**Gambar 2.1.** Contoh representasi data

#### 2.1.1 *The Mixed National Institute of Standards and Technology (MNIST) Database*

MNIST *database* adalah sebuah *database* yang besar berupa gambar hitam putih tulisan tangan angka dari nol sampai dengan sembilan. Setiap gambarnya berukuran tinggi 28 piksel dan lebar 28 piksel juga, dengan total 784 piksel. Dalam setiap pikselnya dapat dipetakan menjadi suatu nilai, yang mengindikasikan tingkat terang dan gelap. Niai setiap pikselnya berupa integer dari 0 hingga 255, semakin besar nilai semakin gelap piksel dalam gambar (0 berarti putih, sementara 255 berarti hitam).

```
000 001 002 003 ... 026 027
028 029 030 031 ... 054 055
056 057 058 059 ... 082 083
 |   |   |   |  ...  |   |
728 729 730 731 ... 754 755
756 757 758 759 ... 782 783
```

**Gambar 2.2.** Pemetaan piksel berupa nilai acak dari salah satu gambar hitam putih tulisan tangan angka.



**Gambar 2.3.** Contoh seratus gambar hitam putih tulisan tangan, yang diambil secara acak.

Data yang diunduh dari MNIST *database* terdiri atas 60.000 data latih, 60.000 label data latih, 10.000 data tes dan 10.000 label data tes. Masing-masih set data dapat diunduh dari tautan berikut:
http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz;
http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz;

http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz; dan http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz.

## 2.2 *Machine Learning*

*Machine learning* secara aktif digunakan dewasa ini, mungkin lebih dari yang Anda bayangkan. Professor Andrew Ng (2012) menyatakan bahwa tidak ada definisi yang dapat diterima dengan baik oleh para pelaksana *machine learning* tentang apa itu *machine learning*. Beliau memberikan contoh dari orang-orang yang mencoba mendefinisikannya, yaitu:

1) Arthur Samuel (1959). *Machine learning*: bidang studi yang memberikan kemampuan kepada komputer untuk belajar tanpa diprogram secara eksplisit. Dibuktikan dengan pembuatan *game* komputer berupa permainan catur yang awalnya bodoh mejadi pintar dengan melakukan simulasi sebanyak mungkin terhadap permainan tersebut.
2) Tom Mitchell (1998). *Machine learning* diajukan sebagai berikut: program komputer dikatakan belajar dari pengalaman E (*experience*) dengan tujuan untuk melakukan tugas T (*task*) dan pengukuran kinerja P (*performance*), jika kinerja melakukan T dengan diukur menggunakan P meningkat dengan pengalaman E.

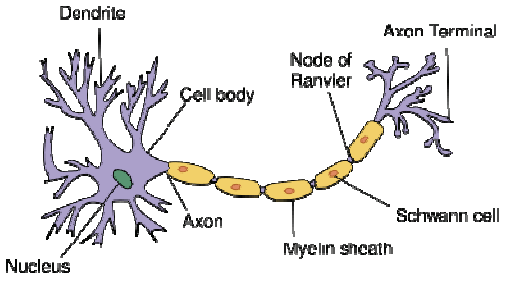

**Gambar 2.5.** Contoh *machine learning* di kehidupan sehari-hari.

*Face recognition, handwriting digit recognition on envelope, spam filtering, and product recommendations from Amazon.com* (Harrington, 2012, p4).

## 2.3 Jaringan saraf tiruan

Jaringan saraf tiruan atau *neural network* sebernarnya merupakan algortima machine learning yang sudah muncul dan banyak digunakan sejak lama pada 1980 dan 1990, dan atas beberapa alasan pernah ditinggalkan pada akhir tahun 90-an untuk beberapa saat. Namun kini menjadi landasan dari algoritma machine learning. Jaringan saraf tiruan sudah terbuki dapat digunakan untuk memecahkan banyak masalah dalam machine learning (Andrew Ng, 2012), salah satu contohnya dapat ditemukan dalam pemecahan masalah digit classifier di tautan http://yann.lecun.com/exdb/mnist/.

Salah satu fakor utama kebangkitan algoritma jaringan saraf buatan sekarang ini adalah kemajuan kemampuan komputer yang mampu melakukan komputasi berat dengan sangat cepat. Karena pada dasarnya jaringan saraf tiruan merupakan algoritma yang agak berat secara komputasional. Gagasan utama jaringan saraf tiruan adalah dengan menirukan otak manusia. Jaringan saraf tiruan dikembangkan dengan mensimulasikan jaringan saraf pada manusia.




**Gambar 2.6.** Satu sel saraf (Wikipedia, 2014)

### 2.3.1 *Cost Function*

*Cost function* untuk perhitungan *error* dilakukan dengan *cost function logistic regression* dengan *generalization*. Untuk masalah *binary classification* dalam jaringan saraf buatan, *cost function* yang digunakan adalah sebagi berikut:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2 \quad (2.4)$$

Dimana:

$J(\theta)$ = *cost function*;
$m$ = jumlah data;
$y$ = label data;
$h_\theta\left(x^{(i)}\right)$ = fungsi hipotesis;

$x$ = data;
$\sum_{j=1}^{n} \theta_j^2$ = *regularization*
(dengan tidak mengikutkan parameter bias $\theta_0$ ).

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(\theta_{ji}^{(l)})^2 \quad (2.5)$$

Dimana:
$J(\theta)$ = *cost function*;
$m$ = jumlah data;
$K$ = jumlah kelas;
$L$ = jumlah *layer*;
$s_l$ = jumlah unit pada *layer l*;
$y$ = label data;
$h_\theta(x^{(i)})$ = fungsi hipotesis;
$x$ = data;
$\lambda$ = konstanta *regularization*;

$\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(\theta_{ji}^{(l)})^2$ = *regularization* (dengan tidak mengikutkan parameter bias $\theta_0$ ).

### 2.3.2 Algoritma *Feed Forward Propagation*

Diberikan sebuah *training data* (x,y):
*Forward propagation*:
$a^{(1)} = x$
$z^{(2)} = \Theta^{(1)} a^{(1)}$
$a^{(2)} = g(z^{(2)})$ tambahkan parameter bias $a_0^{(2)}$
$z^{(3)} = \Theta^{(2)} a^{(2)}$
$a^{(3)} = g(z^{(2)})$ tambahkan parameter bias $a_0^{(3)}$
$z^{(4)} = \Theta^{(3)} a^{(3)}$
$a^{(4)} = h_\Theta(x) = g(z^{(4)})$

### 2.3.3 Algoritma *Back Propagation*

Algoritma memperoleh turunan parsial pada dasarnya adalah melakukan evaluasi error:
$\delta^{(4)} = a^{(4)} - y$
$\delta^{(4)} = (\Theta^{(2)})^T \delta^{(4)} .* g'(z^{(3)})$
$g'(z^{(3)}) = a^{(3)} .* (1 - a^{(3)})$
$\delta^{(3)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$
$g'(z^{(2)}) = a^{(2)} .* (1 - a^{(2)})$

Algoritma *back propagation*:
Training data $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$
Set $\Delta_{ij}^{(l)} = 0$ (untuk semua i,j,l)
For $i = 1$ to $m$
{
Set $a^{(1)} = x^{(i)}$
Lakukan forward propagation untuk menghitung $a^{(l)}$ untuk $l = 2, 3, \ldots, L$
Dengan $y^{(i)}$, hitung $\delta^{(L)} = a^{(L)} - y^{(i)}$
Hitung $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$
$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$
}
$D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$ jika $j \neq 0$
$D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)}$ jika $j = 0$
$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$

### 2.4 *Digit Classifier*

*Classifier* merupakan salah satu kategori permasalahan dalam *machine learning*. Penyelesaian dari masalah klasifikasi adalah dengan membangun suatu model *classifier* dengan salah satu algoritma *machine learning*, algoritma jaringan saraf buatan contohnya. Model *classifier* bekerja dengan cara memetakan (mengklasifikasikan) data ke dalam satu atau beberapa kelas yang sudah didefinisikan sebelumnya. Model *classifier* dibangun dan dilatih dengan data latih. Pengujian performa dan akurasi dilakukan dengan menjalankan model pada data tes. Kemudian diputuskan apakah model butuh diperbaiki berdasarkan hasil tes (LeCun) (Andrew Ng, 2012).

*Digit classifier* adalah model *multiclass classifier*, tepatnya sepuluh kelas, yang dibangun dan dilatih untuk mengklasifikasikan gambar hitam putih dari tulisan tangan yang belum pernah diketahui oleh model ke dalam kelasnya yang benar dan dengan benar. Jadi, *digit classifier* diharapkan mampu mengenali tulisan tangan berupa angka yang baru dengan akurasi di atas 99% (LeCun).

### 2.5 *Feature Normalization*

*Feature normalization* adalah teknik optimisasi yang bertujuan untuk menormalisasikan data dengan cara mengurangi rentang nilai minimum dan maksimum. Pada data asli nilai minimum yang ada adalah nol, sementara nilai maksimum adalah 255.

Teknik ini bekerja dengan mengurangi setiap nilai pada data *training* dengan nilai rata-rata data, sehingga data yang diperoleh akan memiliki rata-rata baru nol. Kemudian setiap nilai pada data dibagi dengan standar deviasi seluruh data. Namun ada alternatif lain sebagai pengganti standar deviasi, yaitu nilai maksimum data. Data yang telah dinormalisasi umumnya memiliki rentang nilai yang kecil, umumnya antara minus satu hingga satu, $\{-1 \leq x \leq 1\}$.

Dengan begitu diharapkan beban kalkulasi yang akan dilakukan pada proses *training* menjadi lebih ringan dan memper-

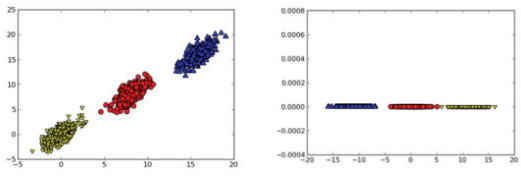
cepat proses *training*. Di samping itu, kecilnya rentang data diharapkan mampu meningkatkan akurasi model karena model lebih merata (Andrew Ng, 2012).

## 2.6 *Dimensionality Reduction*

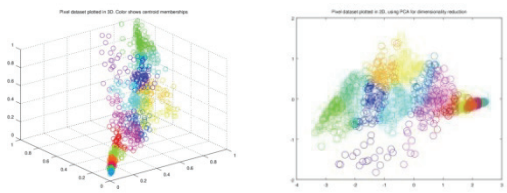
### 2.6.1 *Principal Component Analysis*

Teknik ini dapat diterapkan untuk memproses data pada banyak algoritma *machine learning*. Teknik ini digunakan pada *trainng* data untuk mengurangi jumlah *input*, sehingga dapat mengurangi *noise*, membuat data lebih mudah digunakan, mengurangi beban komputasi dari banyak algoritma *machine learning*, dan mempermudah membaca output (Harrington, 2012).

Dalam PCA, dataset ditransformasikan dari sistem koordinat asli menjadi ke dalam system koordinat baru. Sistem koordinat yang baru dipilih oleh data itu sendiri. Sumbu pertama ditentukan searah dengan data yang memilki varian tertinggi. Sumbu kedua adalah sumbu yang tegak lurus dengan sumbu pertama dan searah dengan data dengan variasi tertinggi. Prosedur tersebut dilakukan berulang-ulang sesuai jumlah input pada data asli. Kemudian akan ditemukan mayoritas variasi pada sedikit sumbu-sumbu pertama. Oleh karena itu, sumbu-sumbu yang terbentuk lebih akhir, dengan kata lain sumbu dengan variasi rendah, dapat diabaikan. (Harrington, 2012).



**Gambar 2.12.** Penerapan PCA untuk merubah sistem koordinat data dari dua-dimensi menjadi satu-dimensi (Harrington, 2012).



**Gambar 2.13.** Penerapan PCA untuk merubah sistem koordinat data dari tiga-dimensi menjadi dua-dimensi.

#### 2.6.1.1 Algoritma PCA

1) *Training data* asli berupa matriks $X \in \mathbb{R}^{m \times n}$

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}$$

Dimana:
$m$ = jumlah data;
$n$= jumlah fitur.

Akan dikompresi dengan PCA menjadi matriks $Z \in \mathbb{R}^{m \times k}$

$$\begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1k} \\ z_{21} & z_{22} & \cdots & z_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ z_{m1} & z_{m2} & \cdots & z_{mk} \end{bmatrix}$$

Dimana:
$m$ = jumlah data;
$k$= jumlah fitur.

2) *Preprocessing data*. Dengan menerapkan *feature scaling* atau *mean normalization* terhadap data.
3) Menghitung *covariance matrix* $\Sigma$, sesuai persamaan

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)})^T (x^{(i)}) \qquad (2.7)$$

Dimana:
$\Sigma \in \mathbb{R}^{n \times n}$ = *covariance matrix*;
$m$ = jumlah data;
$x^{(i)} \in \mathbb{R}^n$ = data ke-i dalam n-*vector*;
$(x^{(i)})^T$ = data ke-i;

4) Menghitung *eigenvectors* dari *covariance matrix*.
Dengan alasan yang sama, Prof. Ng (2012) dan Harrington (2012) tidak membahas pembuktian dan kalkulasi *eigenvectors* secara matematis, yaitu karena kompleks dan meskipun tanpa melakukannya, seseorang tetap dapat membangun *machine learning* yang baik. Di samping itu, banyak bahasa pemrograman yang telah menyediakan *library* untuk menghitung *eigenvectors*, antara lain `svd()` dalam Octave, `svd()` dalam Octave, `eig()` dalam NumPy dan `eigen()` dalam R. Fungsi-fungsi tersebut akan menghasilkan satu atau lebih hasil. Hasil fungsi tersebut yang akan digunakan dalam PCA adalah matriks $U \in \mathbb{R}^{n \times n}$.

5) Menghitung data kompresi $Z \in \mathbb{R}^{m \times k}$.
Mengambil $k$ kolom pertama dari hasil perhitungan *eigenvectors* sebagai basis variasi.
Dalam Octave misalnya, fungsi svd() akan mengembalikan hasil, salah satunya, matriks $U \in \mathbb{R}^{n \times n}$. Dari matriks tersebut diambil $k$ kolom pertama untuk kemudian dijadikan sebagai $Ureduce \in \mathbb{R}^{n \times k}$

$$Z = X.Ureduce \tag{2.8}$$

Dimana:
$Z \in \mathbb{R}^{m \times k}$ = matriks kompresi;
$X \in \mathbb{R}^{m \times n}$ = matriks data asli;
$Ureduce \in \mathbb{R}^{n \times k}$ = matriks k kolom *eigenvector*.

Sehingga didapat,

$$\begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1k} \\ z_{21} & z_{22} & \cdots & z_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ z_{m1} & z_{m2} & \cdots & z_{mk} \end{bmatrix}$$

Dimana:
$m$ = jumlah data;
$k$ = jumlah fitur.

### 2.6.1.2 Rekonstruki data

Data asli tidak dapat dikembalikan menjadi sama persis, namun dapat dilakukan penaksiran data asli dari data kompresi, dengan melakukan perkalian antara data kompresi $Z \in \mathbb{R}^{m \times k}$ dengan data hasil *eigenvectors* $Ureduce \in \mathbb{R}^{n \times k}$.

$$Xapprox = Z.Ureduce^T \tag{2.9}$$

Dimana:
$Z \in \mathbb{R}^{m \times k}$ = matriks kompresi;
$Xapprox \in \mathbb{R}^{m \times n}$ = matriks perkiraan data asli;
$Ureduce \in \mathbb{R}^{n \times k}$ = matriks k kolom *eigenvectors*.

### 2.6.1.3 Memilih *k*

*k* adalah jumlah komponen utama dalam PCA, karena *k* kolom pertama data PCA merupakan data dengan variasi teritinggi. Data tersebut diharapkan masih dapat menyimpan 99% varian data asli, agar output dari *machine learning* tetap terjaga meski jumlah data input dikurangi. Jadi, pemilihan *k* adalah hal penting. Persamaan yang digunakan untuk menentukan *k* yaitu:

$$Average\ squared\ projection\ error: \quad \frac{1}{m}\sum_{i=1}^{m} \left\| x^{(i)} - x_{approx}^{(i)} \right\|^2$$

$$Total\ variation: \quad \frac{1}{m}\sum_{i=1}^{m} \left\| x^{(i)} \right\|^2 \tag{2.11}$$

$$Menjaga\ agar\ varian \geq 99\% \quad => \frac{\frac{1}{m}\sum_{i=1}^{m} \left\| x^{(i)} - x_{approx}^{(i)} \right\|^2}{\frac{1}{m}\sum_{i=1}^{m} \left\| x^{(i)} \right\|^2} \leq 0.01 \tag{2.12}$$

Dimana:
$m$ = jumlah data;
$x^{(i)} \in \mathbb{R}^n$ = data ke-i dalam n-*vector*;
$x^{(i)} \in \mathbb{R}^n$ = data perkiraan ke-i dalam n-*vector*;

Algoritma memilih *k* adalah sebagai berikut:
1) Mencoba PCA dengan $k$ = 1, 2, ...
2) Hitung *Ureduce*, *Z*, *Xapprox*
3) Cek hasil varian menggunakan persamaan (2.x), varian harus terjaga 99%.

## 2.7 Evaluasi Performa

Evaluasi yang akan dilakukan menggunakan parameter F-*Measure* yang terdiri dari perhitungan *precision* dan *recall*. *Recall*, *precision* dan F-*Measure* merupakan metode pengukuran efektifitas yang biasa dilakukan pada proses klasifikasi (Jason, 2004) (Andrew Ng, 2012).

### 2.7.1 F-*Measure*

F-*Measure* didefinisikan sebagai kombinasi dari *recall* dan *precision* dengan bobot yang seimbang. F-*Measure* merupakan unit perhitungan yang digunakan dalam *information retrieval* dan *natural language*. Perhitungan F-*measure* sendiri bergantung pada nilai *precision* dan *recall*.

*Precision* (p) merupakan rasio penempatan dokumen yang benar oleh sistem sesuai kelas tertentu dibagi oleh total jumlah prediksi pada kelas yang benar. *Recall* (r) didefinisikan sebagai rasio penempatan dokumen yang benar oleh sistem sesuai kelas tertentu dibagi oleh total jumlah dokumen yang seharusnya berada pada kelas tertentu tersebut. Perhitungan tersebut dapat dilakukan sesuai persamaan berikut:

$$Precision\ (p) = \frac{TP}{TP + FP} \quad (2.13)$$

$$Recall\ (r) = \frac{TP}{TP + FN} \quad (2.14)$$

$$F - measure = \frac{2 \times p \times r}{p + r} \quad (2.15)$$

## 3. Analisa dan Perancangan

### 3.1 Identifikasi Masalah

1) Bagaimana membangun digit classifier menggunakan jaringan saraf tiruan dipadukan dengan feature normalization dan PCA?
2) Bagaimana performa dan akurasi jaringan normalization dan PCA?

### 3.2 Analisa Kebutuhan dan Deskripsi Sistem

- Ekstraksi data MNIST mentah menjadi data yang siap untuk proses komputasi.
- Menghitung dan menyimpan rata-rata setiap fitur pada data yang diperoleh dari proses *feature normalization*, untuk digunakan pada data tes atau data baru.
- Menghitung dan menyimpan parameter U, S dan k yang diperoleh dari proses PCA, untuk digunakan pada data tes atau data baru.
- Memilih parameter jaringan saraf tiruan berupa banyaknya jumlah iterasi proses training, sementara jumlah *hidden layer* dan jumlah unit pada *hidden layer* dibuat tetap.
- Memilih parameter sigma atau konstanta *regularization* (persamaan 2.5) dari bagian *regularization* jaringan saraf buatan.

Parameter-parameter yang digunakan untuk menghasilkan sistem di atas adalah sebagai berikut:

- `mu` merupakan rata-rata data latih, yang kemudian dijadikan konstanta, dan diterapkan pada data tes dan data baru.
- `U`, `S` dan `k` merupakan parameter PCA, yang dihasilkan dari perhitungan *library* octave `svd()`. `U` digunakan untuk mengurangi dimensi pada data latih, data tes dan data baru sebanyak `k` dimensi. `S` dapat digunakan untuk mengembalikan data hasil kompresi menjadi data yang mendekati data asli.
- `maxIter` merupakan jumlah iterasi proses *training* jaringan saraf tiruan.
- `sigma` merupakan konstanta *regularization* pada bagian *regularization* jaringan saraf tiruan.

### 3.3 Studi Literatur

Sebagian besar literatur yang berupa buku tidak mencantumkan tentang bagaimana melakukan kode, namun berupa teori yang mendalam. Penulis mendapat kesulitan untuk mempelajari teknik melakukan kode jaringan saraf tiruan, feature normalization dan PCA. Oleh karena itu, penulis mengikuti kursus machine learning secara online dengan pengajar utama adalah Professor Andrew Ng. Sementara sebagai literatur tambahan, dan sebagai pembanding, penulis mengunduh banyak jurnal dari situs web http://yann.lecun.com/exdb/ mnist/, yang merupakan sumber data MNIST dan penelitian-penelitian yang telah dilakukan sejak 1998.

### 3.4 Hipotesis

Asumsi sementara penulis mengenai hasil penelitian adalah bahwa *digit classifier* menggunakan jaringan saraf tiruan berhasil dibangun dengan baik namun dengan hasil akurasi dan performa yang buruk. Sementara dengan menerapkan *feature normalization*, penulis tidak yakin dapat memperbaiki performa *digit classifier*, namun akurasi model seharusnya membaik. Sementara dengan PCA, akurasi yang dihasilkan seharusnya tidak lebih baik, namun karena dimensi data yang dikurangi, maka dipastikan proses training *digit classifier* akan berjalan lebih cepat, dengan kata lain dengan performa yang membaik.

Hipotesis netral yang dibuat didasarkan pada hasil-hasil pengembangan *digit classifier* jurnal-jurnal pada halaman *web* http://yann.lecun.com/ exdb/mnist/.

Berdasarkan pengamatan hasil-hasil yang ada, maka *classifier* yang berhasil dibangun harus memiliki hasil *test error rate* lebih baik dari 4,7 atau akurasi lebih baik dari 96%, dan diharapkan dapat bersaing dengan hasil lainnya.

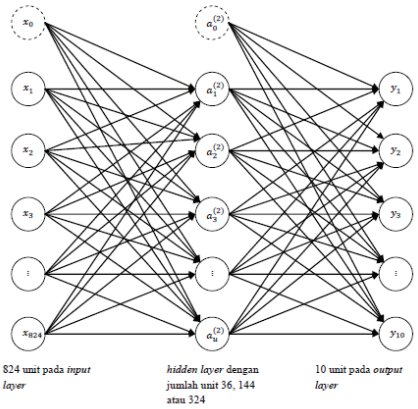
### 3.5 Pengumpulan Data

"Kunci sukses sebuah *classifier* bukan algoritma, melainkan data", kata Prof. Andrew Ng (2012). Data yang dimaksud adalah data *training* (data latih). *Digit classifier* sendiri sudah sejak lama dikembangkan mengguna-

kan berbagai teknik dan metode *machine learning*, seperti pada tautan http://yann. lecun.com/exdb/mnist/. Sehingga, untuk kebutuhan penelitian, pendiri halaman *web* tersebut, LeCun Yann berserta peneliti lainnya menyediakan set data untuk keperluan pengembangan *digit classifier*. Jadi, data latih dan data uji yang baik dapat diperoleh dari halaman *web* tersebut. Sementara hasil-hasil sebelumnya telah dibuat ke dalam jurnal-jurnal dan dapat diunduh pada halaman web tersebut.

### 3.6 Desain Metodologi dan Perancangan Sistem

Dengan penggunaan jumlah unit yang berbeda maka akan lebih mudah untuk melakukan analisa mengenai pengaruh jumlah unit terhadap performa dan akurasi model *classifier*. Berikut visualisasi dari arsitektur jaringan saraf tiruan yang dibangun:




Keterangan:

$x_{1...824}$ = data input

$a^{(2)}_{1...36}$ = *activation unit*

= unit

= bias unit dengan nilai sama dengan 1

Alur proses yang terjadi dalam merancang model classifier jaringan saraf tiruan secara garis besar dibagi menjadi:

1) *Data preprocessing* (pra-proses data)
   *Preprocessing* yang wajib dilakukan pada sistem yang dibangun kali ini adalah *parsing* data, karena data yang didapat tidak dapat langsung digunakan. Teknik optimisasi *feature normalization* dan PCA juga diterapkan pada tahap ini.
2) *Training*
   Secara garis besar, tahap ini menggunakan data *training* (latih) hasil dari tahap *preprocessing* sebagai input dan menghasilkan output berupa konstanta yang merupakan model *digit classifier* dengan sepuluh kelas, masing-masing mereperesentasikan angka dari nol hingga sembilan.
3) Implementasi
   Tahap implementasi sebenarnya merupakan tahap *training* yang dilakukan dengan data penuh. Artinya, jika saat proses *training* membangun program model *digit classifier* menggunakan semua data, yang pada kesempatan ini sangat besar, maka proses *training* akan memakan waktu sangat lama. Sehingga untuk membuat program dengan menerapkan algoritma dan memiliki tujuan konvergen, maka tidak perlu menggunakan semua data, namun cukup sebagian, setidaknya sepuluh persen. Setelah program diyakini memberikan hasil yang konvergen, maka saatnya melakukan implementasi program pada data penuh.
4) Tes
   Tahap ini dilakukan dengan menerapkan konstanta hasil training pada data tes, dengan hasil berupa kelas antara nol hingga sembilan, disebut pula dengan *classifying* (klasifikasi). Data tes ini merupakan data yang sama sekali berbeda dengan data *training*. Sebelum melakukan klasifikasi, data tes harus menjalani proses *preprocessing* terlebih dahulu, namun dengan parameter-parameter yang sama dengan hasil *preprocessing* data *training*, untuk membuat data tes kompatibel dengan model, karena model yang dibangun adalah model yang membawa parameter-parameter data *training* sejak awal.

#### 3.6.1 *Data Preprocessing*

Semua data yang terlibat, khususnya data *training* dan data *test* akan melalui semua proses *data preprocessing*. Namun, untuk kebutuhan perancangan sistem, maka hanya akan digunakan sebagian data *training* dan label *training* sebanyak sepuluh persen dari total data. Sehingga dapat dipastikan proses *training* akan memakan waktu lebih sedikit dan memudahkan merancang sistem yang efektif.

### 3.6.1.1 Data yang digunakan

Perancangan data merupakan bentuk menyiapkan data yang akan digunakan sebagai data *training* dan data testing. Dalam sistem yang dibangun ini, data training dan data testing diperoleh dari *database* MNIST berupa gambar tulisan tangan angka hitam putih yang seluruh gambarnya telah diubah ke dalam format file *binary* yaitu:

| [offset] | [type] | [value] | [description] |
|---|---|---|---|
| 0000 | 32 bit integer | 0x00000803(2051) | magic number |
| 0004 | 32 bit integer | 60000 | number of images |
| 0008 | 32 bit integer | 28 | number of rows |
| 0012 | 32 bit integer | 28 | number of columns |
| 0016 | unsigned byte | ?? | pixel |
| ..... | | | |
| xxxx | unsigned byte | ?? | pixel |
| Keterangan: | | | |
| Nama file: train-images.idx3-ubyte.gz | | | |
| Nilai piksel antara 0 hingga 255, 0 berarti putih, 255 berarti hitam. | | | |

Tabel 3.1 – *Training set data*.

- *Training set label* : train-labels.idx1-ubyte;

| [offset] | [type] | [value] | [description] |
|---|---|---|---|
| 0000 | 32 bit integer | 0x00000801(2049) | magic number (MSB first) |
| 0004 | 32 bit integer | 60000 | number of items |
| 0008 | unsigned byte | ?? | label |
| 0009 | unsigned byte | ?? | label |
| ..... | | | |
| xxxx | unsigned byte | ?? | label |
| Keterangan: | | | |
| Nama file: train-labels.idx1-ubyte.gz | | | |
| Nilai label antara 0 hingga 9. | | | |

Tabel 3.2 – *Training set label*.

- *Test set data*: t10k-images.idx3-ubyte; dan

| [offset] | [type] | [value] | [description] |
|---|---|---|---|
| 0004 | 32 bit integer | 10000 | number of images |
| 0008 | 32 bit integer | 28 | number of rows |
| 0012 | 32 bit integer | 28 | number of columns |
| 0016 | unsigned byte | ?? | pixel |
| 0017 | unsigned byte | ?? | pixel |
| ..... | | | |
| xxxx | unsigned byte | ?? | pixel |
| Keterangan: | | | |
| Nama file: t10k-images.idx3-ubyte.gz | | | |
| Nilai piksel antara 0 hingga 255, 0 berarti putih, 255 berarti hitam. | | | |

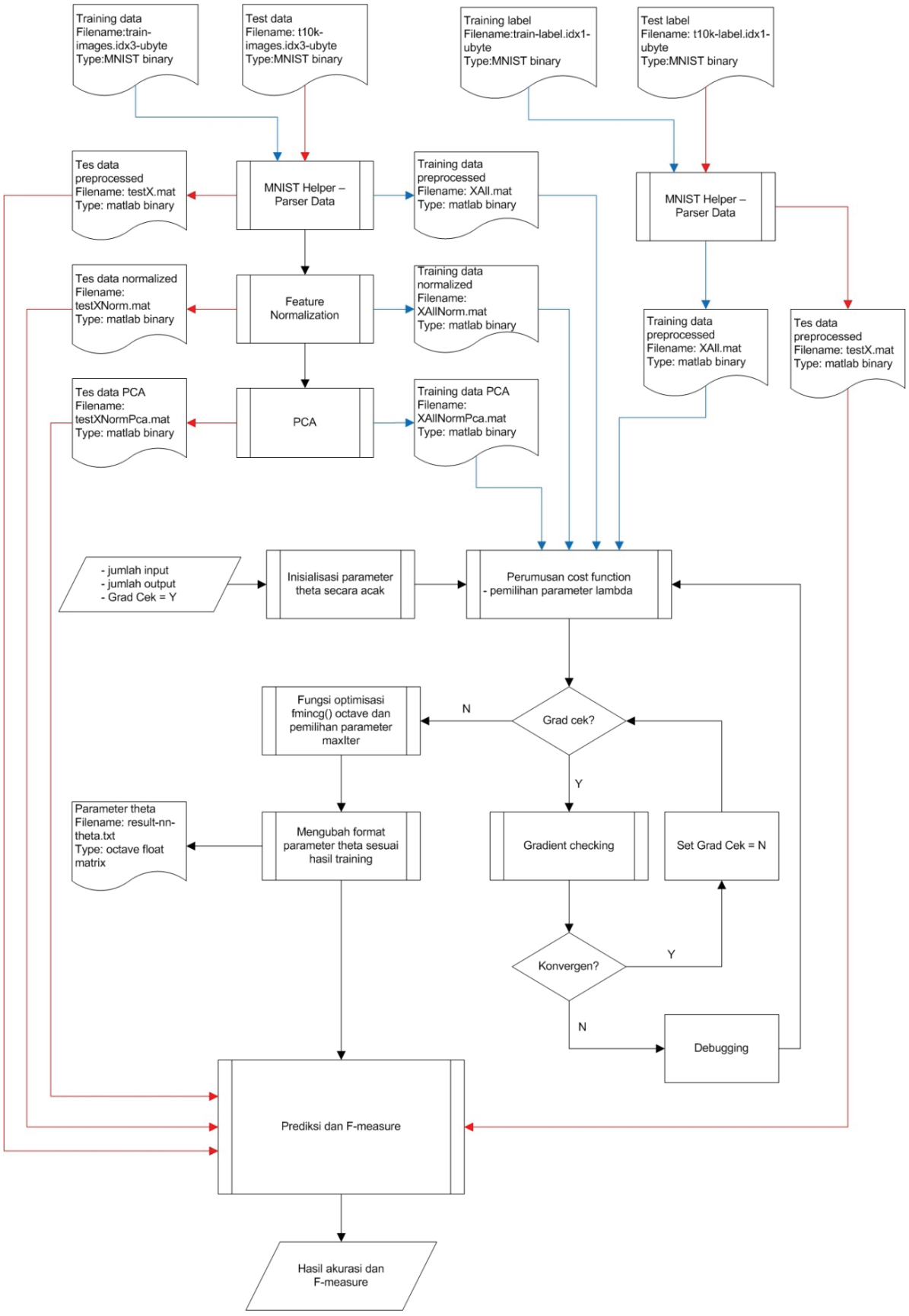
Tabel 3.3 – *Test set data*.

- *Test set label*: t10k-labels.idx1-ubyte.

| [offset] | [type] | [value] | [description] |
|---|---|---|---|
| 0000 | 32 bit integer | 0x00000801(2049) | magic number (MSB first) |
| 0004 | 32 bit integer | 10000 | number of items |
| 0008 | unsigned byte | ?? | label |
| 0009 | unsigned byte | ?? | label |
| ..... | | | |
| xxxx | unsigned byte | ?? | label |
| Keterangan: | | | |
| Nama file: t10k-labels.idx1-ubyte.gz | | | |
| Nilai label antara 0 hingga 9. | | | |

Tabel 3.4 – *Test set label*.

Secara garis besar, alur kerja sistem yang dibangun adalah sebagai berikut:




**Gambar 3.2.** Alur sistem yang dibangun

### 3.6.1.2 Perancangan *parser* data

Penulis membangun dua buah *parser* dalam bahasa pemrograman *Octave* yaitu:

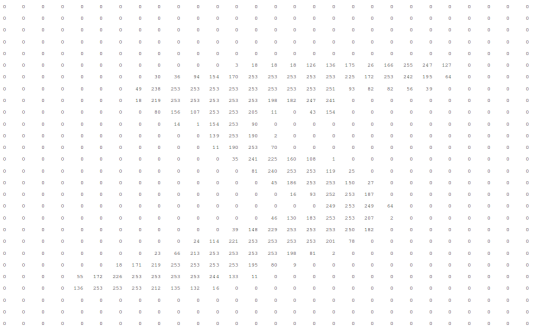
- helperLoadMnistData.m

Dibangun untuk membaca file *binary* MNIST data *training* sesuai Tabel 3.1 dan data *test* sesuai Tabel 3.3 menjadi file *binary* matlab yang siap digunakan pada proses *training*. Data gambar berada pada *offset* 16 sampai dengan *offset* terakhir. Pada data *training*, jumlah data piksel dari *offset* 16 sampai dengan *offset* terakhir adalah 47.040.000. Nilai tersebut diperoleh dari jumlah keseluruhan piksel yang ada pada data *training* tersebut, di mana jumlah gambar sama dengan 60.000, sedangkan setiap gambar memiliki ukuran piksel 28 × 28. Jadi, 47.040.000 = 60.000 × 28 × 28. Data tersebut kemudian akan ditampung sementara ke dalam variabel dengan tipe data matriks satu dimensi atau *vector* 47040000 × 1.

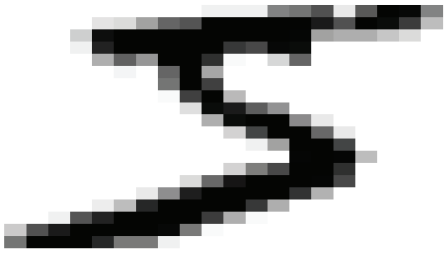
| *gambar ke* — | *data ke* — | |
|---|---|---|
| 1 | 1 | 0 |
| | 2 | 0 |
| | 3 | 0 |
| | ⋮ | ⋮ |
| | 169 | 43 |
| | 170 | 18 |
| | 171 | 0 |
| | ⋮ | ⋮ |
| | 782 | 0 |
| | 783 | 0 |
| | 784 | 0 |
| 2 | 785 | 0 |
| | 786 | 0 |
| | ⋮ | ⋮ |
| | 955 | 255 |
| | 956 | 255 |
| | ⋮ | ⋮ |
| | 1568 | 0 |
| 3 | 1569 | 0 |
| | ⋮ | ⋮ |
| | 3136 | 0 |
| 4 | 3137 | 0 |
| | ⋮ | ⋮ |
| | 3920 | 0 |
| 5 | 3921 | 0 |
| ⋮ | ⋮ | ⋮ |
| 60000 | 47039216 | 0 |
| | ⋮ | ⋮ |
| | 47040000 | 0 |

**Gambar 3.3.** Struktur data gambar *training* MNIST

Jika diambil contoh gambar pertama, atau dengan kata lain data ke-1 sampai dengan data ke-784, maka akan diperoleh vector dengan ukuran 784 × 1. Jika dilakukan matriks transformasi, atau dengan kata lain diambil setiap 28 data dari awal hingga akhir, kemudian disusun, maka akan menjadi matriks dengan ukuran 28 × 28, yang merupakan pemetaan dari piksel pada gambar asli. Berikut adalah visualisasinya:



**Gambar 3.4.** 784 data MNIST pertama dalam matriks 28 × 28.



**Gambar 3.5.** Data pada gambar 3.4 yang diubah menjadi gambar.

Semua data tersebut kemudian menjalani transformasi matriks yang mirip dari gambar 3.3 menjadi data seperti pada gambar 3.4, sebagai berikut.

$$X \in \square^{47040000 \times 1} \rightarrow X \in \square^{28 \times 28 \times 60000} \rightarrow X \in \square^{60000 \times 784}$$

Struktur data test MNIST juga memiliki struktur yang sama dengan data *training*, hanya saja jumlah gambar yang hanya 10.000, sehingga jumlah data sama dengan 7.840.000.

- helperLoadMnistLabel.m

Dibangun untuk membaca file *binary* MNIST label *training* sesuai Tabel 3.2 dan label *test* sesuai Tabel 3.4 menjadi file *binary* matlab, serta mengambil 6000 label *training*

sebagai *sample* sesuai dengan data *training sample* sehingga dihasilkan file ySample.mat.

Berbeda dengan data gambar, data label, yang ada pada *offset* 8 sampai dengan *offset* terakhir, hanya berupa bilangan bulat antara 1 sampai dengan 10, yang merepresentasikan kelas data gambar.

| *label ke* — | *data ke* — | |
|---|---|---|
| 1 | 1 | 5 |
| 2 | 2 | 0 |
| 3 | 3 | 4 |
| ⋮ | ⋮ | ⋮ |
| 60000 | 60000 | 8 |

**Gambar 3.6.** Struktur data label *training* MNIST

### 3.6.1.3 Perancangan feature normalization

Tujuan dibangunnya *feature normalization* featureNormalizeDm adalah menormalisasikan data *training*, data *training sample* dan data *test*, sehingga diharapkan dapat memperbaiki akurasi dan performa model yang akan dibangun. Data asli memiliki rentang yang tinggi, yaitu 255, diperoleh dari nilai terkecil 0 dan tertinggi 255. Dengan teknik ini, rentang data tersebut akan dibuat menjadi minimal, karena operasi utama pada teknik ini adalah melakukan pengurangan data asli dengan dengan nilai rata-ratanya. Dalam sistem yang dibangun ini, data asli dibagi dengan nilai tertinggi data dengan tujuan agar rentang dapat minimal.

### 3.6.1.4 Perancangan PCA

Data yang digunakan telah terlebih dahulu diterapkan *feature normalization*.

Hal penting perlu diperhatikan adalah matriks *eigenvector* U dan S serta konstanta k yang harus ikut disimpan bersama matiks hasil PCA Z. Matriks *eigenvectors* ini nantinya akan diterapkan pada data test dan data baru lainnya saat mencari PCA dari data tersebut. Jadi, model *digit classifier* yang dilatih dengan data *training* hasil PCA hanya kompatibel pada data yang telah diterapkan *preprocessing* dengan parameter-parameter yang sama dengan data *training*, termasuk U hasil PCA dan konstanta k.

## 3.6.2 *Training*

### 3.6.2.1 Inisialisasi parameter theta

Parameter theta yang dibutuhkan adalah theta1 untuk menghitung setiap nilai *activation* pada *hidden layer* dan theta2 untuk menghitung setiap nilai output. Dengan tanpa melupakan *bias unit*, maka theta1 berupa matriks $36 \times 825$ dan theta2 berupa matriks $10 \times 37$ ($\theta^{(1)} \in \square^{36 \times 825}$ dan $\theta^{(2)} \in \square^{10 \times 37}$). Dalam jaringan saraf tiruan, inisialisasi parameter theta tidak boleh nol, karena jika parameter theta diterapkan pada persamaan sigmoid (persamaan 2.1 dan 2.2) maka saat dilakukan *feed forward*, nilai hasil pada *activation unit* selalu bernilai nol.

### 3.6.2.2 Perumusan *cost function*

Tahap perancangan ini bertujuan untuk membuat program yang menghitung *cost function* sesuai dengan persamaan 2.5. Input program tersebut adalah (1) parameter theta, (2) data *training* X, (3) label *training* y, dan (4) konstanta *regularization* lambda. Output program dapat dikatakan sebagai nilai *error*, sehingga semakin kecil nilai *error* tersebut, seharusnya dapat menghasilkan model yang baik. Pada program tersebut juga diterapkan algoritma *feed forward* dan *back propagation*.

Program yang dihasilkan akan menghitung nilai error dengan persamaan (2.5) berikut:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\theta(x^{(i)})\right)_k + (1-y_k^{(i)}) \log\left(1-\left(h_\theta(x^{(i)})\right)_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\left(\theta_{ji}^{(l)}\right)^2$$

Sebelum dimasukkan ke dalam persamaan, label training y dimodifikasi. Label nol (0) perlu diubah menjadi sepuluh (10), sedangkan lainnya dibiarkan sama.

Hasil dari program tersebut adalah theta1_grad dan theta2_grad yang dipengaruhi oleh konstanta *regularization* lambda. Jadi, dalam implementasi nanti, akan diberikan beberapa nilai lambda untuk menganalisa pengaruh lambda terhadap akurasi model.

### 3.6.2.3 Melakukan *gradient checking*

Untuk menghitung *gradient cost function* secara manual dapat dilakukan dengan persamaan berikut:

$$f_i \approx \frac{J\left(\theta^{(i+1)}\right) - J\left(\theta^{(i-1)}\right)}{2\epsilon} \quad (3.1)$$

Di mana:
$f_i$ = taksiran *gradient* elemen ke- $i$
$J\left(\theta^{(i+1)}\right)$ = *cost function* pada elemen $i+1$
$J\left(\theta^{(i-1)}\right)$ = *cost function* pada elemen $i-1$
$\epsilon = 10^{-4}$

Setelah menjalankan program tersebut, maka akan ditampilkan selisih *gradient* hasil program *cost function* dan *gradient* hasil perhitungan manual. Jika program yang dibangun adalah benar dan konvergen, maka nilai selisih lebih kecil atau sama dengan 1e-9.

#### 3.6.2.4 Proses *training* dengan fmincg() Octave

Bahasa pemgrograman Octave menyediakan fitur *advanced optimization* untuk *cost function* fmincg(). Fungsi tersebut dijamin dapat melakukan optimisasi *cost function* selama program *cost function* yang dibangun memberikan nilai *gradient* yang benar, dan hal tersebut dapat dipastikan dengan metode *gradient checking*. Jika pada program *cost function*, konstanta lambda merupakan objek analisa *training*, maka pada bagian ini objek analisa ada pada jumlah iterasi maxIter. Parameter maxIter akan diberikan kepada fungsi fmincg()sebagai parameter *option*.

Fungsi fmincg() mengharuskan fungsi optimisasi, dalam sistem yang dibangun ini adalah fungsi *cost function*, mampu menghasilkan nilai turunan parsial. Jika *gradient checking* yang telah dilakukan sebelumnya menunjukkan bahwa algoritma *feed forward* dan *back propagation* adalah konvergen, maka fmincg() dipastikan dapat mengoptimalkan fungsi sesuai jumlah iterasi yang diberikan.

#### 3.6.2.5 *Debugging*

Jika hasil *training* tidak menunjukkan anomali, maka dapat diasumsikan bahwa program yang dibangun dapat menghasilkan model *digit classifier* yang baik. Namun, ada kalanya hal tersebut tidak tercapai. Kesalahan dapat saja terjadi di bagian *training* atau bahkan pada bagian *data preprocessing*. Untuk mempermudah mencari kesalahan program tentu saja *debugging* adalah cara paling ampuh.
Berikut ini dapat dilakukan untuk *debug* program:

- Membiarkan program mengeluarkan stream output perintah.
  Dalam Octave hal ini dapat dilakukan dengan tidak memberikan ";" pada akhir perintah. Contoh:
  A2 = sigmoid(Z2); ➔
  A2 = sigmoid(Z2)

- Mencetak ukuran matriks memberikan info yang sangat berguna, sehingga sering kali memberikan dampak yang siginifkan dalam debugging.
  Contoh:
  size(A2)
- Menghentikan program sementara.
  Dalam Octave perintah pause; dapat melakukannya.
- Melakukan *plot* dan visualisasi.
  Dalam Octave hal ini dapat dilakukan dalam berbagai cara, seperti perintah plot() atau hist().

### 3.6.3 Perancangan Komponen Klasifikasi

#### 3.6.3.1 Menentukan hasil prediksi

Basis perhitungan untuk menentukan klasifikasi data yang dites adalah dengan persamaan sigmoid (persamaan (2.2)). Dalam bahasa pemrograman Octave, klasifikasi tersebut dilakukan dengan cara:

(1) Menghitung sigmoid h1 dan h2

```
h1 = sigmoid([ones(m, 1) X] * theta1');
h2 = sigmoid([ones(m, 1) h1] * theta2');
```

Perkalian antara $\square \in \square^{6000 \times 825}$ (nilai 825 adalah jumlah *feature* 824 ditambah dengan *bias*) dengan $\square^{(1)} \in \square^{825 \times 36}$ (*invers* $\square^{(1)} \in \square^{36 \times 825}$) adalah $\square_1 \in \square^{6000 \times 36}$.
Perkalian antara $\square_1 \in \square^{6000 \times 37}$ (nilai 37 adalah jumlah *feature* 36 ditambah dengan *bias*) dengan $\square^{(2)} \in \square^{37 \times 10}$ (*invers* $\square^{(2)} \in \square^{10 \times 37}$) adalah $\square_2 \in \square^{6000 \times 10}$.

(2) Yang artinya adalah h2 merupakan matriks dengan jumlah baris yang sama dengan *training label* ySample. Untuk menentukan apakah data X yang dites

termasuk ke dalam kelas 1, 2 atau 10 (10 artinya 0, yang telah diubah guna memudahkan proses *training*) dengan mengambil nilai terbesar dari sepuluh kolom di setiap barisnya.

#### 3.6.3.2 Menghitung akurasi langsung

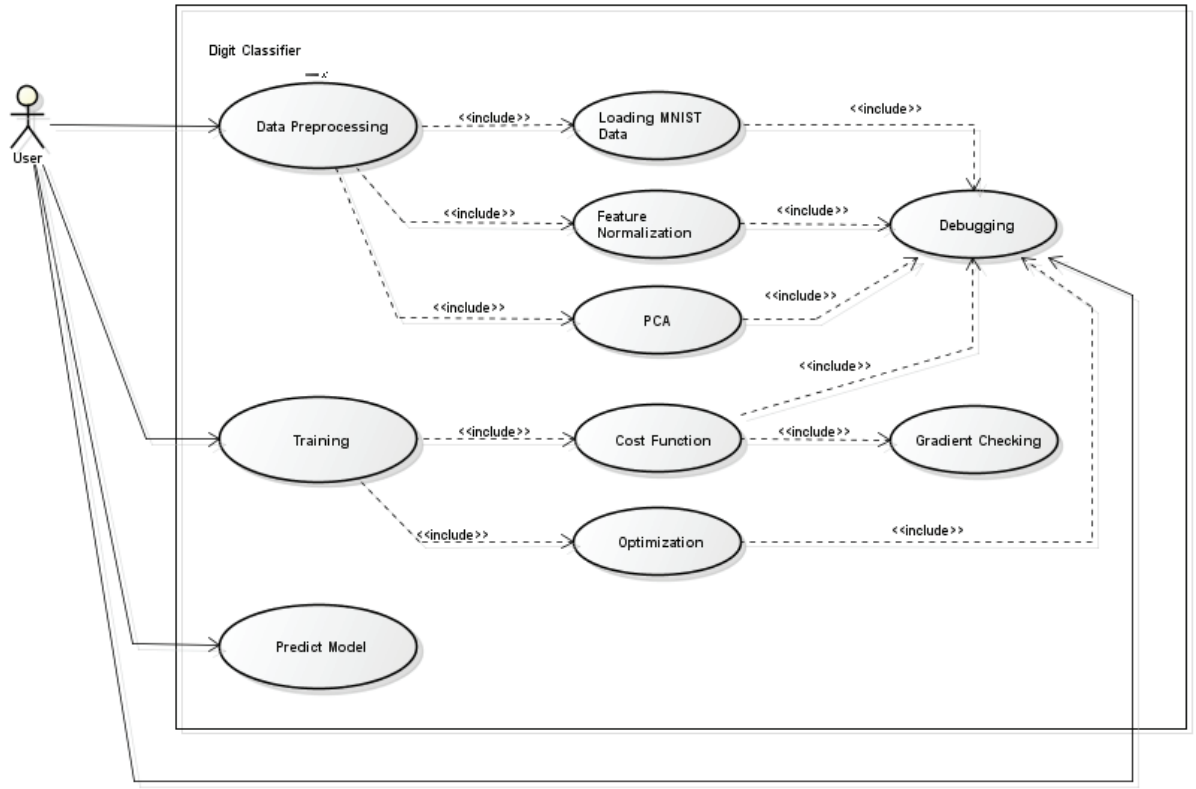
Untuk mengetahui akurasi pada tahap ini, dilakukan dengan sangat mudah, karena sudah *vector* p diperoleh dari hasil perhitungan di atas dan label *training* y.

```
acc = mean(double(p == y))*100;
```

#### 3.6.3.3 Menghitung F-*Measure*

F-*measure* yang didapat dari perhitungan *precision* (p) dan *recall* (r) sesuai dengan persamaan (2.13), persamaan (2.14) dan persamaan (2.15). Nilai F-*measure* dihasilkan berjumlah sama dengan jumlah kelas, yang artinya mewakili nilai per kelas. Sehingga hasil klasifikasi model untuk setiap kelas dapat diketahui. Dari hasil F-measure per kelas tersebut memungkinkan untuk melakukan analisa tentang sebaik apa classifier pada setiap kelasnya. F-measure global dapat diperoleh dengan cara menghitung rata-rata dari sepuluh F-measure yang didapat sebelumnya.

### 3.7 Diagram *Use Case*




Gambar 3.7 – Diagram *use case digit classifier*

### 3.8 Pengujian dan Evaluasi

Secara garis besar, objek yang akan dievaluasi adalah pengaruh parameter-parameter dalam *digit classifier*, baik parameter jaringan saraf tiruan, *feature normalization* dan PCA terhadap performa dan akurasi *digit classifier*.

## 4. Implementasi dan Evaluasi

### 4.1 Implementasi

#### 4.1.1 Lingkungan Implementasi

1) Spesifikasi perangkat keras
   - Intel® Core™ i5-2430M, 4 CPUs @ 2.40GHz
   - RAM 8GB DDR3
   - 500GB 720rpm *harddisk*

2) Spesifikasi perangkat lunak
   - Sistem Operasi Ubuntu precise 12.04 LTS 64-bit
   - GNU Octave, version 3.6.1
   - GNU bash, version 4.2.25(1) release (x86_64-pc-linux-gnu)
   - gedit, version 3.4.1
   - GNOME Terminal, version 3.4.11

### 4.1.2 Implementasi *Prepocessing*

Semua data gambar baik *training*, *sample training* maupun *test* data akan melalui tahap *preprocessing* sesuai dengan program yang telah dirancang pada Bab Perancangan. Hasil dari tahap *preprocessing* ini adalah *file binary*.

#### 4.1.2.1 Hasil *parsing* data MNIST

Hasil *preprocessing* tanpa teknik optimisasi. Program yang digunakan pada kategori *preprocessing* ini hanya *parser* helperLoadMnistData.m, yang dikode ke dalam Octave sesuai dengan struktur pada tabel 3.1 dan tabel 3.3, sebagai berikut:

- Memproses data sesuai struktur

```
magicNumber =
fread(mnist, 1, 'int32', 0, 'ieee-be');
numImages =
fread(mnist, 1, 'int32', 0, 'ieee-be');
numRows =
fread(mnist, 1, 'int32', 0, 'ieee-be');
numCols =
fread(mnist, 1, 'int32', 0, 'ieee-be');
result =
fread(mnist, inf, 'uchar', 'ieee-be');
```

- Melakukan transformasi matriks

```
result =
reshape(result,numRows,numCols,numImages);
result =
permute(result, [2 1 3]);
result =
reshape(result,numImages,numRows*numCols);
```

Hasilnya adalah:

- Data *training sample*: XSample.mat
- Data *training*: XAll.mat
- Data *test*: testX.mat

#### 4.1.2.2 Implementasi *feature normalization*

- Mencari nilai terbesar X
- Membagi semua X dengan nilai terbesar X
- Menghitung rata-rata X
- Melakukan operasi pengurangan X dengan nilai rata-ratanya

```
maxVal = max(X(:));
XNorm = bsxfun(@rdivide, X, maxVal);
mu = mean(XNorm);
XNorm = bsxfun(@minus, XNorm, mu);
```

Algortima tersebut diterapkan pada data hasil *parsing* XSample.matdan XAll.mat.

Objek yang disimpan dalam file tersebut adalah matriks data gambar X, nilai tertinggi data *training* maxVal dalam tipe *float* dan nilai rata-rata data training mu dalam tipe *float*. Cara berbeda dilakukan untuk menghasilkan data *test* dengan *feature normalization*. Pada data *test*, nilai maxVal dan mu tidak dihitung dari data *test*, melainkan menggunakan nilai hasil perhitungan pada data *training*, dengan kata lain menggunakan nilai pada file *normalization* yang telah disimpan sebelumnya.

#### 4.1.2.3 Implementasi PCA

Data yang digunakan sebagai input adalah hasil *feature normalization*. Sehingga program yang digunakan adalah helperLoadMnistData.m, featureNormalize.m dan applyPca.m. Berikut adalah kode program dalam Octave sesuai dengan algoritma PCA:

XNorm: data training hasil feature normalization

- Menghitung covariance matrix sigma (persamaan 2.7)
- Menghitung eigenvectors dalam matrix hasil svd()

```
sigma = (1/m) * XNorm' * XNorm;
[U S V] = svd(sigma);
```

- Menentukan nilai k: jumlah komponen utama. Dengan cara melakukan iterasi sejumlah k, sedemikian hingga 99% variasi dapat dijaga (persamaan 2.12).

```
SAll = sum(S(:));
Sk = 0;
k = 1;
while Sk/SAll < 0.99
        k++
        Sk = sum(S(1:k,:)(:));
        Sk/SAll
end;
```

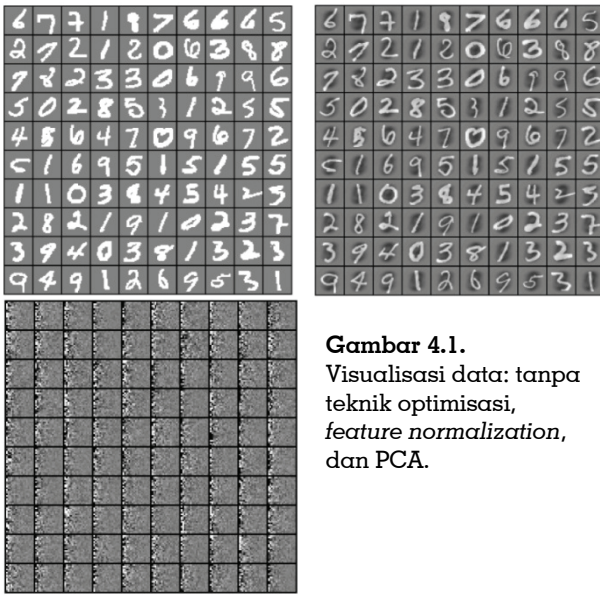
- Menghitung data kompresi hasil PCA Z

```
Z = X * U(:, 1:k);
```

Data hasil PCA ini, Z, berbentuk matriks dengan ukuran 6000 × k, artinya memiliki fitur sebanyak k. Pada data *training sample*, nilai k yang didapat adalah 824. Sementara pada seluruh data training, nilai nilai k yang didapat adalah 861. Pada nilai k tersebut, varian yang dapat dijaga, dihitung sesuai dengan persamaan 2.12 adalah di atas 99%.

Objek yang disimpan dalam file tersebut adalah matriks data gambar X, matriks *eigenvectors* U, matriks *eigenvalues* S dan jumlah komponen utama k dalam tipe *integer*. Seperti pada *feature normalization*, parameter-parameter tersebut disimpan untuk digunakan pada data *test*.

#### 4.1.2.4 Perbandingan hasil *preprocessing* data gambar



**Gambar 4.1.** Visualisasi data: tanpa teknik optimisasi, *feature normalization*, dan PCA.

#### 4.1.2.5 *Parsing* label gambar

Sementara label data, baik *training* maupun *test*, memiliki struktur yang sederhana seperti pada tabel 3.2 dan tabel 3.4, sehingga *preprocessing* yang dilakukan cukup tahap parsing saja.

Jadi, program yang digunakan hanya helperLoadMnistLabel.m, yang dikode sebagi berikut:

### 4.1.3 *Training*

#### 4.1.3.1 Inisialisasi theta

- Menentukan nilai epsilonInit yang mendekati nol sebagai basis perhitungan theta
- Secara acak menghitung theta yang akan menghasilkan matriks berukuran out × in

```
epsilonInit = sqrt(6)/sqrt(in+out)
P = rand(out, 1 + in) * 2 * epsilonInit -
  epsilonInit;
```

Basis perhitungan untuk inisialisasi parameter theta adalah konstanta epsilonInit dengan jumlah unit sebagai parameternya. Arsitektur jaringan saraf tiruan yang dibangun ada tiga, sehingga hasil perhitungan epsilonInit dan theta adalah:

(1) 36 *hidden units*
- epsilonInit matriks 36 × 785 theta1 = 0.085540
- epsilonInit matriks PCA sample 36 × 325 theta1 = 0.12910
- epsilonInit matriks PCA 36 × 362 theta1 = 0.12294
- epsilonInit matriks 10 × 37 theta2 = 0.36116

(2) 144 *hidden units*
- epsilonInit matriks PCA 144 × 362 theta1 = 0.10900
- epsilonInit matriks 10 × 145 theta2 = 0.19739

(3) 324 *hidden units*
- epsilonInit matriks PCA 324 × 362 theta1 = 0.093590
- epsilonInit matriks 10 × 325 theta2 = 0.13403

#### 4.1.3.2 Membangun *cost function* berserta turunan parsial dengan algoritma *feed forward* dan *back propagation*

Tahap ini adalah tahap utama dalam *training* jaringan saraf buatan. Program yang dibangun akan menggunakan data dan parameter yang telah didapat, yaitu:
- Data *training* X dan labelnya y yang telah disimpan dalam file *binary* pada saat *preprocessing*.
- Parameter theta1 dan theta2.
- Jumlah *input unit*, *hidden unit* dan *output unit*.

Persamaan utama yaitu persamaan (2.5),

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\theta(x^{(i)})\right)_k + (1-y_k^{(i)}) \log\left(1-\left(h_\theta(x^{(i)})\right)_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\left(\theta_{ji}^{(l)}\right)^2$$

bersama dengan *feed forward* dan *back propagation*, diterjemahkan ke dalam bahasa

pemrograman Octave sesuai dengan algoritma berikut:

- Mengubah label 0 pada y menjadi 10
  y(find(y==0)) = 10;
- Feed forward, menghitung nilai aktivasi
  A1 = X;
  A1 = [ones(m,1) A1];
  Z2 = A1*Theta1';
  A2 = sigmoid(Z2);
  m2 = size(A2,1);
  A2 = [ones(m2,1) A2];
  Z3 = A2*Theta2';
  A3 = sigmoid(Z3);

- Cost function
  J = (1/m) * (-yAll1' * log(hx) - (1-yAll1)' * log(1-hx)) + ((0.5*lambda/m) * (nnParams' * nnParams));
- Back propagation yang menghasilkan turunan parsial cost function

```
for i=1:m
        a1 = A1(i,:);
        z2 = Z2(i,:);
        a2 = A2(i,:);
        z3 = Z3(i,:);
        a3 = A3(i,:);
        delta3 = (a3-yAll(i,:));
        delta2 = delta3 * Theta2 .* (a2 .* (1-a2));
        delta2 = delta2(2:end);
        Theta1_grad = Theta1_grad + delta2' * a1;
        Theta2_grad = Theta2_grad + delta3' * a2;
end;
```

Fungsi eksternal sigmoid() berasal dari persamaan sigmoid (persamaan 2.2)

$$\square_{\square}(\square) = \frac{1}{1+\square^{-\square_{\square}}}$$

yang dikode ke dalam Octave menjadi:

```
g = 1.0 ./ (1.0 + exp(-z));
```

#### 4.1.3.3 Implementasi *gradient checking*

Program *cost function* yang menerapkan fungsi optimisasi dengan algoritma *feed forward* dan *back propagation* akan dicek apakah dapat konvergen atau tidak, dengan membandingkan perhitungan turunan parsial dari program tersebut dengan perhitungan *numerical gradient*. Dikode sebagai berikut:

- Menghitung numerical gradient dengan fungsi cost function J
  loss1 = J(theta - perturb);
  loss2 = J(theta + perturb);
  numgrad(p) = (loss2 - loss1) / (2*e);
- Menghitung selisih turunan parsial
  diff = norm(numgrad-grad)/norm(numgrad+grad);

Hasil perbandingan lima nilai terakhir adalah sebagi berikut:

**Tabel 4.1.** Perbandingan *numerical gradient* dan *gradient* hasil program cost function

| numerical grad | grad ) |
|---|---|
| 5.3697E-02 | 5.3697E-02 |
| 4.7146E-02 | 4.7146E-02 |
| 1.4957E-01 | 1.4957E-01 |
| 5.3154E-02 | 5.3154E-02 |
| 4.6560E-02 | 4.6560E-02 |

Selisih nilai *gradient* yang dihasilkan adalah 2.09443e-11. Dari hasil pada tabel 4.1 di atas, dan karena hasil selisih *gradient* adalah tidak lebih besar dari 1e-1, maka program *cost function* dapat konvergen.

#### 4.1.3.4 *Training* dengan fmincg()

Algoritma *training* menggunakan fmincg() dikode sebagai berikut:

- Mendefinisikan costFunction dengan memanggil program costFunction
- Set opsi MaxIter dari fmincg()
  costFunction = @(params) nnCostFunction(params, numInputs, numHiddens, numLabels, X, y, lambda);

```
options = optimset('MaxIter', maxIter);
[nnParams, cost] = fmincg(costFunction, initParams, options);
```

Saat program berjalan, secara *default*, fmincg() akan menampilkan nilai ] yang merupakan hasil perhitungan costFunction atau nilai *error*. Jika program yang dibangun benar, maka nilai *error* tersebut akan berkurang pada setiap iterasinya. Parameter nnParams adalah gabungan parameter theta1 dan theta2 yang telah dioptimalkan sedemikian hingga cost mendekati nol, atau setidaknya paling minimal. Setelah program training berhenti, parameter nnParams yang dilatih akan dipecah kembali sesuai ukuran semula. Hasil akhir *training* adalah matriks theta1 dan theta2 yang kemudian disimpan ke dalam file berformat txt sebagai plain text matriks.

Dengan tujuan analisa, maka *training* dilakukan berulang kali, dengan cara

memodifikasi parameter-parameter dan arsitektur dengan nilai-nilai berikut:

- Jumlah hidden unit (numHiddens): 36, 144 dan 324.
- Jumlah iterasi (maxIter): 20, 50, 100, 150, 300 dan 500.
- Parameter regularization (lambda): 0, 1, 3, 5 dan 8.

### 4.1.4 Hasil Klasifikasi

Untuk menentukan kelas data yang diklasifikasi menggunakan parameter theta1 dan theta2 merupakan kelas angka nol, satu atau sembilan, penulis membangun program untuk melakukan prediksi.
Berikut adalah program predict.m:

- Menghitung sigmoid hidden (h1)
- Menghitung sigmoid output label (h2)
- Menentukan kelas berdasarkan hasil h2

```
h1 = sigmoid([ones(m, 1) X] * Theta1');
h2 = sigmoid([ones(m, 1) h1] * Theta2');
[dummy, p] = max(h2, [], 2);
```

Gambar berikut adalah contoh tiga baris pertama matriks h2

```
1.85e-7  1.63e-5  7.44e-3  1.44e-5  3.76e-5
1.17e-9 9.99e-1 6.50e-6 1.17e-4 2.26e-4

2.92e-4  9.78e-1  9.86e-3  8.97e-9  4.78e-4
1.09e-4 6.12e-8 4.20e-4 8.05e-9 5.40e-3

2.31e-3  5.06e-4  2.91e-5  1.05e-3  1.10e-3
3.29e-4 1.04e-3 2.41e-4 3.12e-7 9.96e-1
```

**Gambar 4.2.** Tiga *record* pertama hasil klasifikasi.

Jadi, kelas untuk data pertama adalah 7, kelas untuk data kedua adalah 2, dan kelas untuk data ketiga adalah 10 (atau 0).

Dalam Octave, prediksi tersebut dapat dilakukan sekaligus dengan perintah:

```
[dummy, p] = max(h2, [], 2);
```

Hasil yang digunakan adalah p yang merupakan *vector* dengan ukuran sama dengan y.

### 4.1.5 Menghitung Akurasi Model *Classifier*

#### 4.1.5.1 Menghitung akurasi langsung

- Nilai nol pada label y harus diubah menjadi 10

```
y(find(y==0)) = 10;
pred = predict(theta1, theta2, X);
acc = mean(double(pred == y)) * 100;
```

#### 4.1.5.2 Menghitung F-*measure*

for i = 1:10

- Menghitung true positive
  `tp = sum((pred(i) == 1) & (y(i) == 1));`
- Menghitung false positive
  `fp = sum((pred(i) == 1) & (y(i) == 0));`
- Menghitung false negative
  `fn = sum((pred(i) == 0) & (y(i) == 1));`
- Menghitung precision
  `prec = tp/(tp+fp);`
- Menghitung recall
  `reca = tp/(tp+fn);`
- Menghitung F-measure untuk kelas i
  `F1(i) = 2 * prec * reca / (prec+reca);`

Contoh hasil F-*measure* per kelas:

**Tabel 4.2** – Tabel F-*measure* per kelas

| F-*measure* |
|---|
| 0.949230 |
| 0.838280 |
| 0.832100 |
| 0.859050 |
| 0.773230 |
| 0.909980 |
| 0.873260 |
| 0.812440 |
| 0.797200 |
| 0.937290 |

Hasil F-*measure* total adalah nilai rata-rata dari F-*measure* per kelas. Rata-rata F-*measure* untuk contoh di atas adalah 0.85821.

## 4.2 Pengujian dan Evaluasi

### 4.2.1 Pengujian

#### 4.2.1.1 Dokumen uji

Dokumen yang digunakan adalah dokumen dengan format atau bentuk yang sama dengan dokumen latih, dan dokumen uji tersebut diperoleh dari *database* MNIST yang diunduh bersamaan dengan dokumen latih.

Pada dasarnya dokumen tersebut harus menjalani *preprocessing* yang sama dengan data latih untuk dapat digunakan sebagi input model yang terlah berhasil dilatih pada tahap implementasi. Tidak hanya tahap *preprocessing* yang sama, namun juga parameter-parameternya. Hal tersebut dilakukan dengan menggunakan parameter-parameter yang

telah disimpan sebelumnya pada tahap implementasi *preprocessing* terhadap data latih. Parameter-parameter yang dimaksud adalah:

- Parameter maxIter ∈ bilangan bulat pada *feature normalization*;
- Parameter mu ∈ bilangan bulat pada *feature normalization*;
- Parameter U ∈ matriks pada PCA;
- Parameter S ∈ matriks pada PCA;
- Parameter k ∈ bilangan bulat pada PCA.

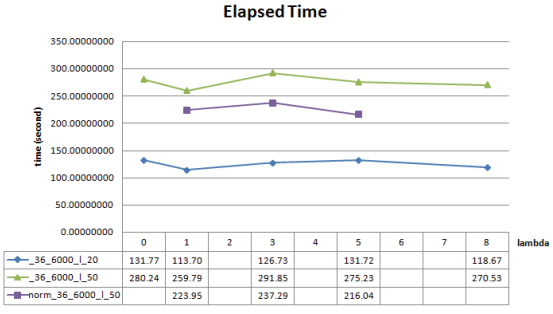
#### 4.2.1.2 Implementasi klasifikasi data uji dan hasil pengujian

Untuk melakukan klasifikasi, pada tahap sebelumnya, penulis telah merancang dan melakukan implementasi kode untuk itu, yaitu predict.m. Program tersebut dirancang dan dibangun untuk dapat menerima input berupa matriks. Di samping itu, dokumen uji kini telah menjadi data yang siap untuk digunakan sebagai input model *classifier*, yaitu berupa matriks. Dengan itu, maka klasifikasi data uji dapat dilakukan dengan mudah. Hasil klasifikasi tersebut adalah basis perhitungan akurasi model dan perhitungan *f-measure*, yang cara perhitunggan sama dengan perhitungan terhadap data latih.

### 4.2.2 Strategi Pengujian

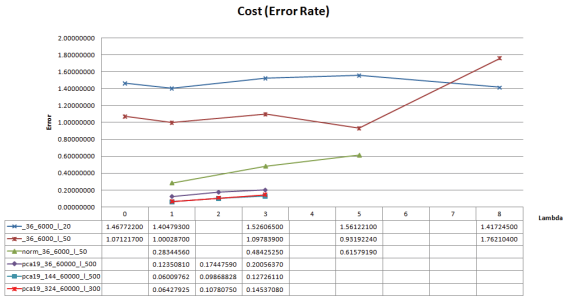
Strategi pengujian penting dilakukan untuk meningkatkan efisiensi analisa dan evaluasi. Proses training memakan waktu yang tidak sama pada setiap perubahan parameter. Untuk menentukan parameter-parameter yang optimal dalam *training* dan pengujian, digunakan data *training sample* sebanyak 6000 data untuk menentukan parameter optimal tersebut. Dengan kata lain, pada tahap ini dilakukan eliminasi parameter-parameter yang tidak optimal. Strategi yang dimaksud adalah:

#### 4.2.2.1 Observasi parameter lambda pada performa dan akurasi



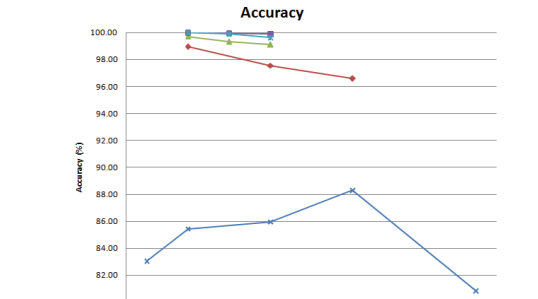

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| _36_6000_l_20 | 131.77 | 113.70 | | 126.73 | | 131.72 | | | 118.67 |
| _36_6000_l_50 | 280.24 | 259.79 | | 291.85 | | 275.23 | | | 270.53 |
| norm_36_6000_l_50 | | 223.95 | | 237.29 | | 216.04 | | | |

**Gambar 4.3.** Grafik perbandingan parameter lambda terhadap waktu *training*



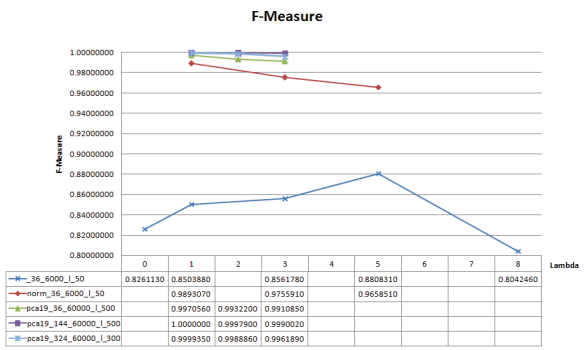

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| _36_6000_l_20 | 1.46772200 | 1.40479300 | | 1.52606500 | | 1.56122100 | | | 1.41724500 |
| _36_6000_l_50 | 1.07121700 | 1.00028700 | | 1.09783900 | | 0.93192240 | | | 1.76210400 |
| norm_36_6000_l_50 | | 0.28344560 | | 0.48425250 | | 0.61579190 | | | |
| pca19_36_60000_l_500 | | 0.12350810 | 0.17447590 | 0.20056370 | | | | | |
| pca19_144_60000_l_500 | | 0.06009762 | 0.09868828 | 0.12726110 | | | | | |
| pca19_324_60000_l_300 | | 0.06427925 | 0.10780750 | 0.14537080 | | | | | |

**Gambar 4.4.** Grafik perbandingan parameter lambda terhadap *error rate*




| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| _36_6000_l_50 | 83.0500 | 85.4500 | | 85.9666 | | 88.3000 | | | 80.8666 |
| norm_36_6000_l_50 | | 98.9333 | | 97.5666 | | 96.6000 | | | |
| pca19_36_60000_l_500 | | 99.7050 | 99.3216 | 99.1100 | | | | | |
| pca19_144_60000_l_500 | | 100.000 | 99.9783 | 99.8983 | | | | | |
| pca19_324_60000_l_300 | | 99.9933 | 99.8866 | 99.6166 | | | | | |

**Gambar 4.5.** Grafik perbandingan parameter lambda terhadap akurasi

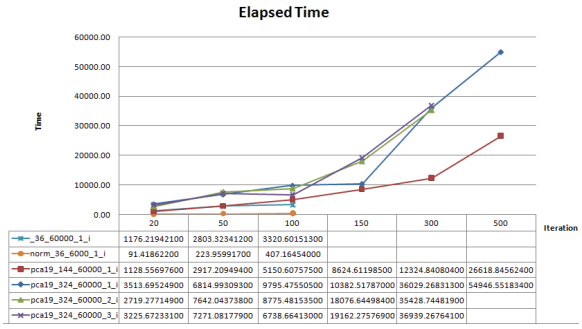



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| _36_6000_l_50 | 0.8261130 | 0.8503880 | | 0.8561780 | | 0.8808310 | | | 0.8042460 |
| norm_36_6000_l_50 | | 0.9893070 | | 0.9755910 | | 0.9658510 | | | |
| pca19_36_60000_l_500 | | 0.9970560 | 0.9932200 | 0.9910850 | | | | | |
| pca19_144_60000_l_500 | | 1.0000000 | 0.9997900 | 0.9990020 | | | | | |
| pca19_324_60000_l_300 | | 0.9999350 | 0.9988860 | 0.9961890 | | | | | |

**Gambar 4.6.** Grafik perbandingan parameter lambda terhadap *f-measure*

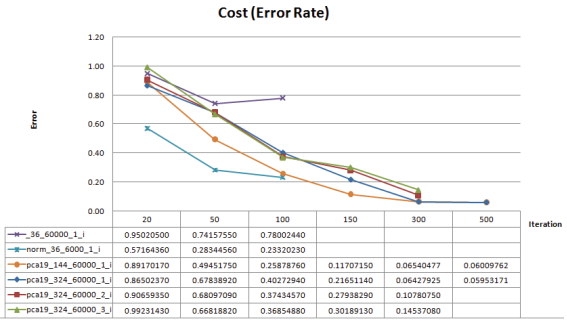
Berdasarkan grafik-grafik di atas, dapat disimpulkan bahwa parameter lambda tidak memiliki pengaruh berarti pada performa *training* dan semakin besar nilai lambda semakin kecil nilai akurasi *training*. Sementara parameter yang optimal untuk pengujian adalah parameter dengan nilai: 1, 2 dan 3.

#### 4.2.2.2 Observasi parameter maxIter pada performa dan akurasi



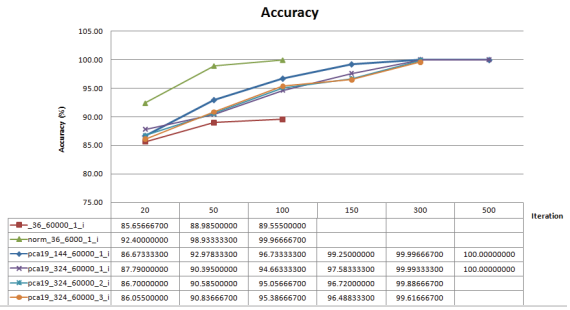

| | 20 | 50 | 100 | 150 | 300 | 500 |
|---|---|---|---|---|---|---|
| _36_60000_1_i | 1176.21942100 | 2803.32341200 | 3320.60151300 | | | |
| norm_36_6000_1_i | 91.41862200 | 223.95991700 | 407.16454000 | | | |
| pca19_144_60000_1_i | 1128.55697600 | 2917.20949400 | 5150.60757500 | 8624.61198500 | 12324.84080400 | 26618.84562400 |
| pca19_324_60000_1_i | 3513.69524900 | 6814.99309300 | 9795.47550500 | 10382.51787000 | 36029.26831300 | 54946.55183400 |
| pca19_324_60000_2_i | 2719.27714900 | 7642.04373800 | 8775.48153500 | 18076.64498400 | 35428.74481900 | |
| pca19_324_60000_3_i | 3225.67233100 | 7271.08177900 | 6738.66413000 | 19162.27576900 | 36939.26764100 | |

**Gambar 4.7.** Grafik perbandingan parameter maxIter terhadap waktu *training*



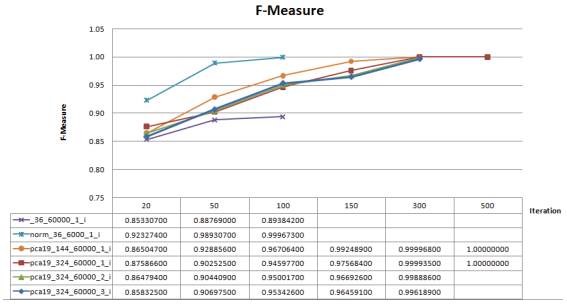

| | 20 | 50 | 100 | 150 | 300 | 500 |
|---|---|---|---|---|---|---|
| _36_60000_1_i | 0.95020500 | 0.74157550 | 0.78002440 | | | |
| norm_36_6000_1_i | 0.57164360 | 0.28344560 | 0.23320230 | | | |
| pca19_144_60000_1_i | 0.89170170 | 0.49451750 | 0.25878760 | 0.11707150 | 0.06540477 | 0.06009762 |
| pca19_324_60000_1_i | 0.86502370 | 0.67838920 | 0.40272940 | 0.21651140 | 0.06427925 | 0.05953171 |
| pca19_324_60000_2_i | 0.90659350 | 0.68097090 | 0.37434570 | 0.27938290 | 0.10780750 | |
| pca19_324_60000_3_i | 0.99231430 | 0.66818820 | 0.36854880 | 0.30189130 | 0.14537080 | |

**Gambar 4.8.** Grafik perbandingan parameter maxIter terhadap *error rate*




| | 20 | 50 | 100 | 150 | 300 | 500 |
|---|---|---|---|---|---|---|
| _36_60000_1_i | 85.65666700 | 88.98500000 | 89.55500000 | | | |
| norm_36_6000_1_i | 92.40000000 | 98.93333300 | 99.96666700 | | | |
| pca19_144_60000_1_i | 86.67333300 | 92.97833300 | 96.73333300 | 99.25000000 | 99.99666700 | 100.00000000 |
| pca19_324_60000_1_i | 87.79000000 | 90.39500000 | 94.66333300 | 97.58333300 | 99.99333300 | 100.00000000 |
| pca19_324_60000_2_i | 86.70000000 | 90.58500000 | 95.05666700 | 96.72000000 | 99.88666700 | |
| pca19_324_60000_3_i | 86.05500000 | 90.83666700 | 95.38666700 | 96.48833300 | 99.61666700 | |

**Gambar 4.9.** Grafik perbandingan parameter maxIter terhadap akurasi

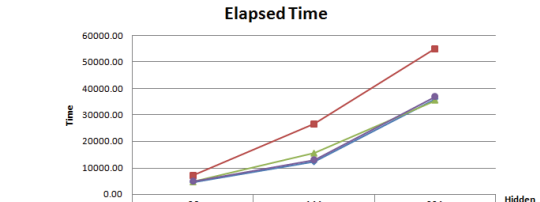



| | 20 | 50 | 100 | 150 | 300 | 500 |
|---|---|---|---|---|---|---|
| _36_60000_1_i | 0.85330700 | 0.88769000 | 0.89384200 | | | |
| norm_36_6000_1_i | 0.92327400 | 0.98930700 | 0.99967300 | | | |
| pca19_144_60000_1_i | 0.86504700 | 0.92885600 | 0.96706400 | 0.99248900 | 0.99996800 | 1.00000000 |
| pca19_324_60000_1_i | 0.87586600 | 0.90252500 | 0.94597700 | 0.97568400 | 0.99993500 | 1.00000000 |
| pca19_324_60000_2_i | 0.86479400 | 0.90440900 | 0.95001700 | 0.96692600 | 0.99888600 | |
| pca19_324_60000_3_i | 0.85832500 | 0.90697500 | 0.95342600 | 0.96459100 | 0.99618900 | |

**Gambar 4.10.** Grafik perbandingan parameter maxIter terhadap *f-measure*

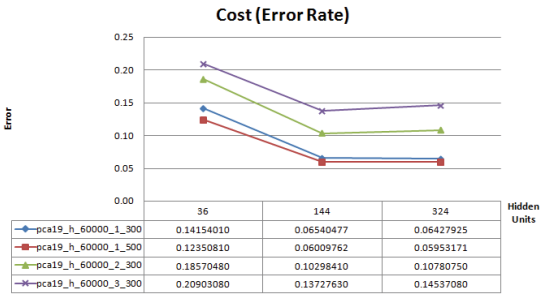
Berdasarkan grafik-grafik di atas, dapat disimpulkan bahwa parameter maxIter atau jumlah iterasi sangat berpengaruh baik terhadap performa maupun akurasi. Semakin banyak iterasi, semakin baik akurasi pada *training*, namun semakin lama proses *training* berlangsung. Sementara pengujian akan dilakukan dengan jumlah iterasi sama dengan *training*, yaitu 20, 50, 100, 150, 300 dan 500.

### 4.2.2.3 Observasi jumlah *hidden units* pada performa dan akurasi



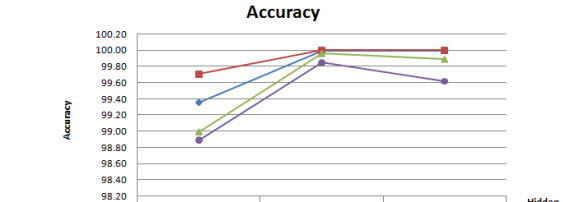

| | 36 | 144 | 324 |
|---|---|---|---|
| pca19_h_60000_1_300 | 4703.85389000 | 12324.84080400 | 36029.26831300 |
| pca19_h_60000_1_500 | 7171.60937000 | 26618.84562400 | 54946.55183400 |
| pca19_h_60000_2_300 | 4760.17859700 | 15538.45838400 | 35428.74481900 |
| pca19_h_60000_3_300 | 4901.48749000 | 12945.98099100 | 36939.26764100 |

**Gambar 4.11.** Grafik perbandingan jumlah *hidden units* terhadap waktu *training*



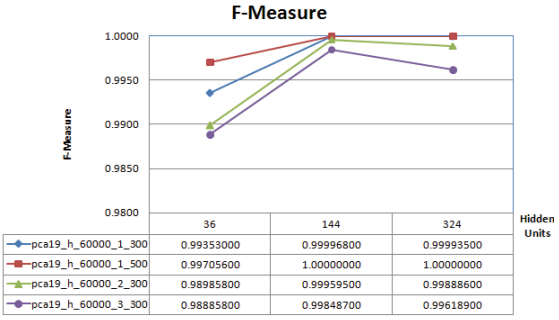

| | 36 | 144 | 324 |
|---|---|---|---|
| pca19_h_60000_1_300 | 0.14154010 | 0.06540477 | 0.06427925 |
| pca19_h_60000_1_500 | 0.12350810 | 0.06009762 | 0.05953171 |
| pca19_h_60000_2_300 | 0.18570480 | 0.10298410 | 0.10780750 |
| pca19_h_60000_3_300 | 0.20903080 | 0.13727630 | 0.14537080 |

**Gambar 4.12.** Grafik perbandingan jumlah *hidden units* terhadap *error rate*




| | 36 | 144 | 324 |
|---|---|---|---|
| pca19_h_60000_1_300 | 99.35333300 | 99.99666700 | 99.99333300 |
| pca19_h_60000_1_500 | 99.70500000 | 100.00000000 | 100.00000000 |
| pca19_h_60000_2_300 | 98.98833300 | 99.95833300 | 99.88666700 |
| pca19_h_60000_3_300 | 98.88833300 | 99.84666700 | 99.61666700 |

**Gambar 4.13.** Grafik perbandingan jumlah *hidden units* terhadap akurasi

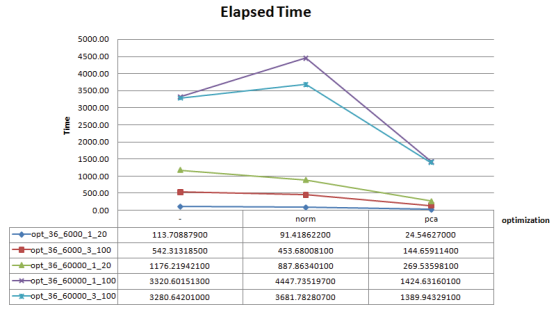



| | 36 | 144 | 324 |
|---|---|---|---|
| pca19_h_60000_1_300 | 0.99353000 | 0.99996800 | 0.99993500 |
| pca19_h_60000_1_500 | 0.99705600 | 1.00000000 | 1.00000000 |
| pca19_h_60000_2_300 | 0.98985800 | 0.99959500 | 0.99888600 |
| pca19_h_60000_3_300 | 0.98885800 | 0.99848700 | 0.99618900 |

**Gambar 4.14.** Grafik perbandingan jumlah *hidden units* terhadap *f-measure*

Berdasarkan grafik-grafik di atas, dapat disimpulkan bahwa semakin banyak jumlah *hidden units*, semakin baik akurasi pada *training*, namun performa *training* menurun drastis. Sementara pengujian akan dilakukan dengan jumlah *hidden units* sama dengan *training*, yaitu 36, 144 dan 324.

### 4.2.2.4 Observasi teknik optimisasi pada performa dan akurasi

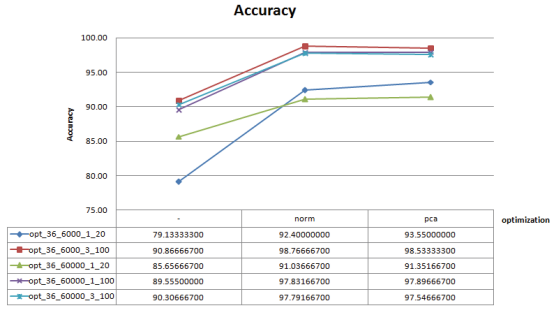



| | - | norm | pca |
|---|---|---|---|
| opt_36_6000_1_20 | 113.70887900 | 91.41862200 | 24.54627000 |
| opt_36_6000_3_100 | 542.31318500 | 453.68008100 | 144.65911400 |
| opt_36_60000_1_20 | 1176.21942100 | 887.86340100 | 269.53598100 |
| opt_36_60000_1_100 | 3320.60151300 | 4447.73519700 | 1424.63160100 |
| opt_36_60000_3_100 | 3280.64201000 | 3681.78280700 | 1389.94329100 |

**Gambar 4.15.** Grafik perbandingan teknik optimisasi terhadap waktu *training*

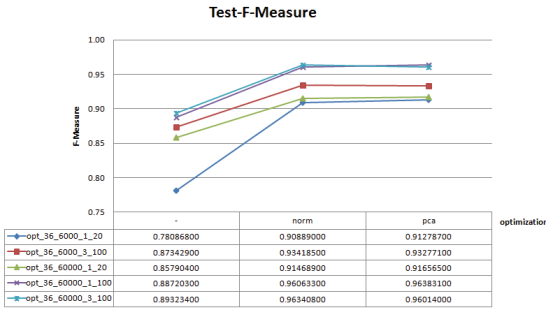

| | - | norm | pca |
|---|---|---|---|
| opt_36_6000_1_20 | 1.40479300 | 0.57164360 | 0.52290380 |
| opt_36_6000_3_100 | 0.82447090 | 0.44999290 | 0.45493280 |
| opt_36_60000_1_20 | 0.95020500 | 0.61026870 | 0.56792590 |
| opt_36_60000_1_100 | 0.78002440 | 0.20800200 | 0.20567310 |
| opt_36_60000_3_100 | 0.66138590 | 0.24918520 | 0.26140590 |

**Gambar 4.16.** Grafik perbandingan teknik optimisasi terhadap *error rate*




| | - | norm | pca |
|---|---|---|---|
| opt_36_6000_1_20 | 79.13333300 | 92.40000000 | 93.55000000 |
| opt_36_6000_3_100 | 90.86666700 | 98.76666700 | 98.53333300 |
| opt_36_60000_1_20 | 85.65666700 | 91.03666700 | 91.35166700 |
| opt_36_60000_1_100 | 89.55500000 | 97.83166700 | 97.89666700 |
| opt_36_60000_3_100 | 90.30666700 | 97.79166700 | 97.54666700 |

**Gambar 4.17.** Grafik perbandingan teknik optimisasi terhadap akurasi



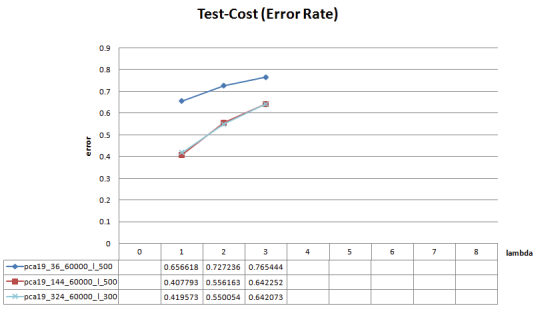

| | - | norm | pca |
|---|---|---|---|
| opt_36_6000_1_20 | 0.78086800 | 0.90889000 | 0.91278700 |
| opt_36_6000_3_100 | 0.87342900 | 0.93418500 | 0.93277100 |
| opt_36_60000_1_20 | 0.85790400 | 0.91468900 | 0.91656500 |
| opt_36_60000_1_100 | 0.88720300 | 0.96063300 | 0.96383100 |
| opt_36_60000_3_100 | 0.89323400 | 0.96340800 | 0.96014000 |

**Gambar 4.18.** Grafik perbandingan teknik optimisasi terhadap *f-measure*

Berdasarkan grafik-grafik di atas, dapat disimpulkan bahwa penerapan teknik optimisasi memberikan dampak yang sangat baik pada performa atau akurasi. *Feature normalization* meningkatkan akurasi *training* dengan signifikan, namun memberikan peningkatan yang kecil terhadap waktu *training*. Sementara PCA tidak meningkatkan akurasi secara signifikan, namun proses *training* berlangsung jauh lebih cepat.

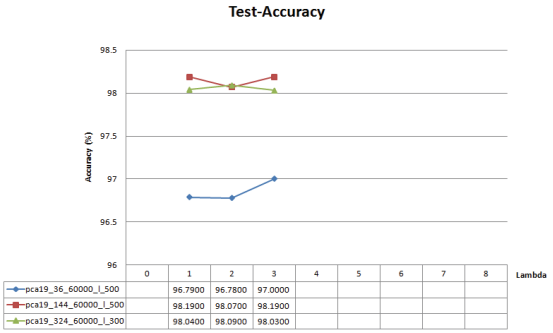
### 4.2.3 Analisa dan Evaluasi Pengujian

### 4.2.3.1 Evaluasi parameter lambda

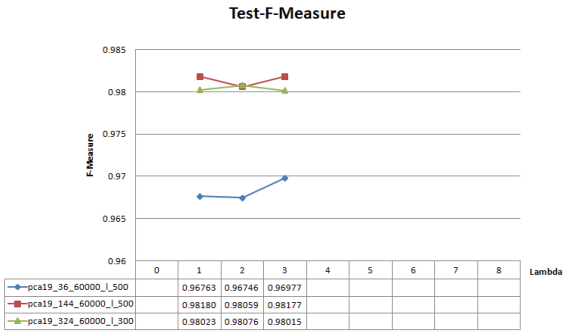



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| pca19_36_60000_l_500 | | 0.656618 | 0.727236 | 0.765444 | | | | | |
| pca19_144_60000_l_500 | | 0.407793 | 0.556163 | 0.642252 | | | | | |
| pca19_324_60000_l_300 | | 0.419573 | 0.550054 | 0.642073 | | | | | |

**Gambar 4.19.** Grafik perbandingan parameter lambda terhadap *error rate*



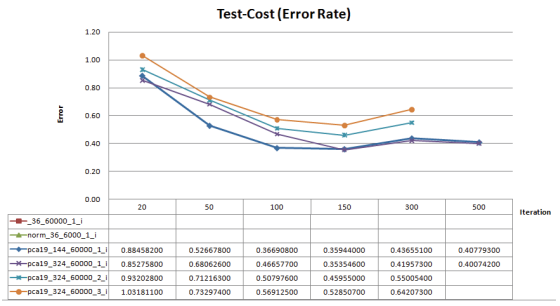

**Gambar 4.20.** Grafik perbandingan parameter lambda terhadap akurasi



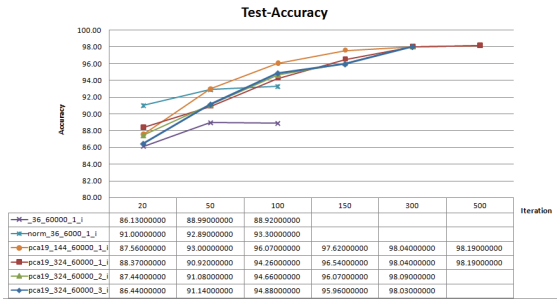

**Gambar 4.21.** Grafik perbandingan parameter lambda terhadap *f-measure*

Berdasarkan hasil pengujian, dapat disimpulkan bahwa nilai lambda yang optimal adalah 1.

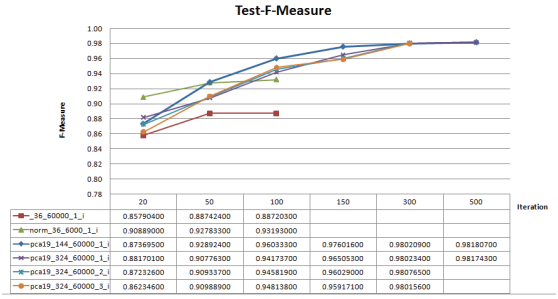
### 4.2.3.2 Evaluasi parameter maxIter




**Gambar 4.22.** Grafik perbandingan parameter maxIter terhadap *error rate*



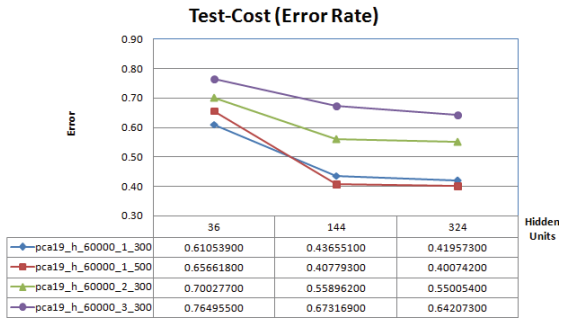

**Gambar 4.23.** Grafik perbandingan parameter maxIter terhadap akurasi



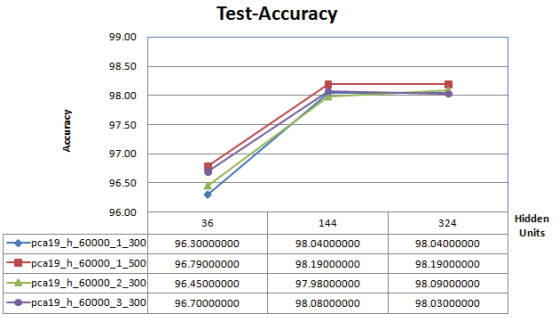

**Gambar 4.24.** Grafik perbandingan parameter maxIter terhadap *f-measure*

Berdasarkan hasil pengujian, dapat disimpulkan bahwa parameter maxIter atau jumlah iterasi sangat berpengaruh baik terhadap performa. Semakin banyak iterasi, semakin baik akurasi pada training.

### 4.2.3.3 Evaluasi jumlah *hidden units*

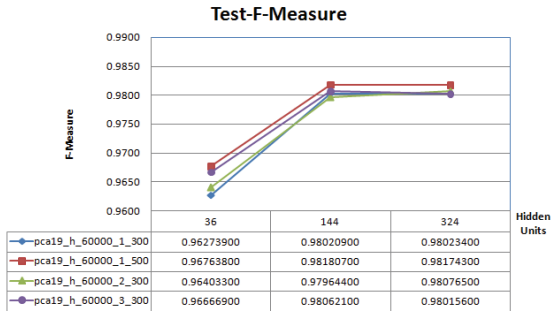



**Gambar 4.25.** Grafik perbandingan jumlah hidden units terhadap *error rate*




| | 36 | 144 | 324 |
|---|---|---|---|
| pca19_h_60000_1_300 | 96.30000000 | 98.04000000 | 98.04000000 |
| pca19_h_60000_1_500 | 96.79000000 | 98.19000000 | 98.19000000 |
| pca19_h_60000_2_300 | 96.45000000 | 97.98000000 | 98.09000000 |
| pca19_h_60000_3_300 | 96.70000000 | 98.08000000 | 98.03000000 |

**Gambar 4.26.** Grafik perbandingan jumlah *hidden units* terhadap akurasi

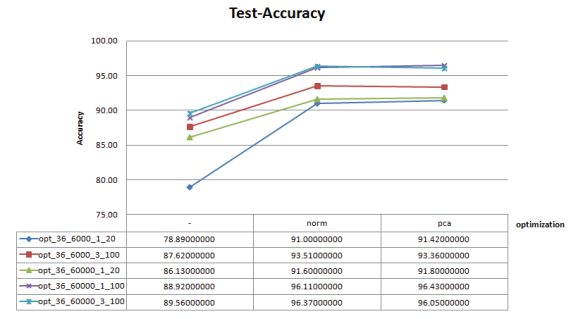



| | 36 | 144 | 324 |
|---|---|---|---|
| pca19_h_60000_1_300 | 0.96273900 | 0.98020900 | 0.98023400 |
| pca19_h_60000_1_500 | 0.96763800 | 0.98180700 | 0.98174300 |
| pca19_h_60000_2_300 | 0.96403300 | 0.97964400 | 0.98076500 |
| pca19_h_60000_3_300 | 0.96666900 | 0.98062100 | 0.98015600 |

**Gambar 4.27.** Grafik perbandingan jumlah *hidden units* terhadap *f-measure*

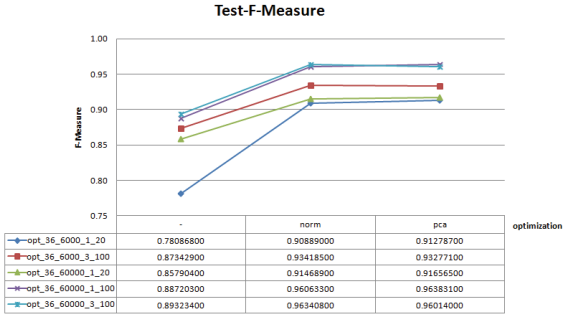
Berdasarkan hasil pengujian, dapat disimpulkan bahwa semakin banyak jumlah *hidden units*, semakin baik akurasi pada *training*.

### 4.2.3.4 Evaluasi penerapan teknik optimisasi




| | - | norm | pca |
|---|---|---|---|
| opt_36_6000_1_20 | 78.89000000 | 91.00000000 | 91.42000000 |
| opt_36_6000_3_100 | 87.62000000 | 93.51000000 | 93.36000000 |
| opt_36_60000_1_20 | 86.13000000 | 91.60000000 | 91.80000000 |
| opt_36_60000_1_100 | 88.92000000 | 96.11000000 | 96.43000000 |
| opt_36_60000_3_100 | 89.56000000 | 96.37000000 | 96.05000000 |

**Gambar 4.28.** Grafik perbandingan teknik optimisasi terhadap akurasi




| | - | norm | pca |
|---|---|---|---|
| opt_36_6000_1_20 | 0.78086800 | 0.90889000 | 0.91278700 |
| opt_36_6000_3_100 | 0.87342900 | 0.93418500 | 0.93277100 |
| opt_36_60000_1_20 | 0.85790400 | 0.91468900 | 0.91656500 |
| opt_36_60000_1_100 | 0.88720300 | 0.96063300 | 0.96383100 |
| opt_36_60000_3_100 | 0.89323400 | 0.96340800 | 0.96014000 |

**Gambar 4.29.** Grafik perbandingan teknik optimisasi terhadap *f-measure*

Berdasarkan hasil observasi training dan pengujian, dapat disimpulkan bahwa penerapan teknik optimisasi memberikan dampak yang sangat baik pada performa atau akurasi. *Feature normalization* meningkatkan akurasi *training* dengan signifikan, namun memberikan peningkatan yang kecil terhadap waktu *training*. Sementara PCA tidak meningkatkan akurasi secara signifikan, namun proses *training* berlangsung jauh lebih cepat.

## 5. Penutup

### 5.1 Simpulan

1) *Feature normalization* bekerja dengan meminimalkan rentang nilai data *training*. *Feature normalization* berhasil dengan signifikan meningkatkan akurasi model. Namun *feature normalization* tidak memberikan peningkatan berarti pada performa, dengan kata lain proses *training* tetap membutuhkan waktu yang lama.
2) PCA bekerja dengan mengurangi dimensi data *training* sehingga membuat volume data lebih kecil atau terkompresi. PCA berhasil dengan signifikan meningkatkan performa *training* model, dengan kata lain proses *training* berlangsung jauh lebih cepat. Namun PCA tidak memberikan peningkatan berarti pada akurasi model.
3) Parameter-parameter digunakan pada proses *training*, dan pengaruhnya terhadap akurasi model dan performa *training* adalah sebagai berikut:
   a) Jumlah *hidden units* menentukan jenis arsitektur jaringan saraf buatan. Semakin banyak jumlah *hidden units*, semakin baik akurasi model, namun semakin lama waktu yang dibutuhkan pada proses *training*.

b) Parameter maxIter atau jumlah iterasi menentukan banyak iterasi pada proses optimisasi fungsi *cost function*. Semakin banyak jumlah iterasi, semakin baik akurasi model, namun semakin lama waktu yang dibutuhkan pada proses *training*.

c) Parameter lambda digunakan pada bagian pada fungsi *cost function* yaitu *regularization*, sehingga disebut pula sebagai *regularization parameter*. Lambda berupa bilangan riil, sehingga tidak mempengaruhi kecepatan *training*. Akurasi model saat *training* dan saat pengujian akan berbeda untuk nilai lambda yang sama. Semakin besar nilai lambda, akurasi pada proses *training* akan menurun, namun meningkat pada proses pengujian. Nilai lambda yang menghasilkan rata-rata akurasi training dan pengujian terbaik adalah nilai lambda optimal.

### 5.2 Saran

1) Menerapkan teknik-teknik *debugging* sejak awal mengembangkan program, jangan menunggu munculnya *bug*.
2) Akurasi model dapat terus menerus ditingkatkan dengan beberapa cara, yaitu:
   a) Menambah jumlah *hidden layer*.
   b) Menambah jumlah *hidden units*.
   c) Menambah jumlah iterasi.
3) Untuk menerapkan cara-cara meningkatkan performa model seperti di atas harus diimbangi dengan perangkat lunak dan perangkat keras yang jauh lebih baik dibandingkan dengan perangkat yang digunakan penulis dalam penelitian ini.

## DAFTAR PUSTAKA


1. Conway, Drew., White, J. M. (2012). *Machine Learning for Hacker*. O'Reilly, California.
2. D. M. R. Jason, (2004), *Derivation of F-Measure*, MIT Press.
3. Hamilton, Laura, (2013), *How Google Uses Machine Learning to Detect Spam Blogs (Maybe)*, http://www.lauradhamilton.com/how-google-uses-machine-learning-to-detect-spam-blogs-maybe.
4. Hamilton, Laura, (2014), *Six Novel Machine Learning Applications*, http://www.forbes.com/sites/85broads/2014/01/06/six-novel-machine-learning-applications/.
5. Harrington, Peter. (2012). *Machine Learning in Action*. Manning, New York.
6. Hinton, G. E., Osindero, S., & The, Y., (2006), *A Fast Learning Algorithm for Deep Belief Nets*, http://www.cs.toronto.edu/~hinton/absps/fastnc.pdf.
7. Jason, D. M. R. (2004). *Derivation of F-Measure*. MIT Press.
8. Kosner, Anthony Wing, (2013), *Why Is Machine Learning (CS 229) The Most Popular Course At Stanford?*, http://www.forbes.com/sites/anthonykosner/2013/12/29/why-is-machine-learning-cs-229-the-most-popular-course-at-stanford/.
9. LeCun, Y., Cortes, C., & Burges, C. J. C., *THE MNIST DATABASE*, http://yann.lecun.com/exdb/mnist/.
10. Ng, Andrew, 2014, *Course: Stanford Machine Learning*, https://class.coursera.org/ml-005.
11. Russell, Stuart., Norvig, Peter. (2010). *Artificial Intelligence, A Modern Approach (Third Edition)*. Prentice Hall, New Jersey.
12. Sermanet, P., Chintala, S., & LeCun, Y., (2012), *Convolutional Neural Networks Applied to House Numbers Digit Classification*, http://arxiv.org/abs/1204.3968v1.
13. Suyanto. (2007). *Artificial Intelligence (Searching, Reasoning, Planning dan Learning)*. Informatika, Bandung.
14. Wan, Li., Zeiler, M., & Sixin, Z., & LeCun, Y., (2013), *Regularization of Neural Networks using DropConnect*, http://www.cs.nyu.edu/~wanli/dropc.