TUGAS AKHIR – KI141502

IMPLEMENTASI ALGORITMA DETEKSI *SPAM* YANG TERSISIPI INFORMASI CITRA DENGAN METODE SVM DAN *RANDOM FOREST*

AGUS TRI WIBOWO
NRP 5110 100 156

Dosen Pembimbing I
Ahmad Saikhu, S.Si., M.T.

Dosen Pembimbing II
Rully Soelaiman, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2016

(Halaman ini sengaja dikosongkan)

FINAL PROJECT – KI141502

IMPLEMENTATION OF IMAGE CONTAINED SPAM DETECTION ALGORITHM USING SVM AND RANDOM FOREST

AGUS TRI WIBOWO
NRP 5110 100 156

Supervisor I
Ahmad Saikhu, S.Si., M.T.

Supervisor II
Rully Soelaiman, S.Kom., M.Kom.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
SEPULUH NOPEMBER INSTITUTE OF TECHNOLOGY
SURABAYA 2016

(Halaman ini sengaja dikosongkan)

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul ***"IMPLEMENTASI ALGORITMA DETEKSI SPAM YANG TERSISIPI INFORMASI CITRA DENGAN METODE SVM DAN RANDOM FOREST".***

Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis. Dengan pengerjaan Tugas Akhir ini, penulis bisa belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Teknik Informatika ITS. Dengan Tugas Akhir ini penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari.

Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan terima kasih kepada:

1. Bapak, ibu dan kakak yang telah memberikan dukungan moral dan material serta do'a yang tak terhingga untuk penulis. Serta selalu memberikan semangat dan motivasi pada penulis dalam mengerjakan Tugas Akhir ini.
2. Bapak Ahmad Saikhu, S.Si., M.T. selaku pembimbing I yang telah membantu dan membimbing penulis dalam menyelesaikan Tugas Akhir ini dengan sabar.
3. Bapak Rully Soelaiman, S.Kom., M.Kom., selaku pembimbing II yang telah memberikan motivasi, nasehat, bimbingan dan bantuan yang banyak kepada penulis dalam mengerjakan Tugas Akhir ini.

4. Ibu Dr. Eng. Nanik Suciati, S.Kom., M.Kom. selaku dosen wali penulis, segenap dosen Teknik Informatika yang telah memberikan ilmunya.
5. Pak Yudi dan segenap staf Tata Usaha yang telah memberikan segala bantuan dan kemudahan kepada penulis selama menjalani kuliah di Teknik Informatika ITS.
6. Bapak, ibu, kakak yang selalu memberikan do'a, motivasi dan nasehat kepada penulis.
7. Kawan-kawan angkatan 2010 yang selalu menjaga kebersamaan, kakak-kakak angkatan 2007, 2008, dan 2009, serta adik-adik angkatan 2011 dan 2012 yang membuat penulis untuk selalu belajar.
8. Serta semua pihak yang yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depan.

Surabaya, Januari 2016

LEMBAR PENGESAHAN

IMPLEMENTASI ALGORITMA DETEKSI *SPAM* YANG TERSISIPI INFORMASI CITRA DENGAN METODE SVM DAN *RANDOM FOREST*

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Cerdas dan Visi
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh
AGUS TRI WIBOWO
NRP. 5110 100 156

Disetujui oleh Dosen Pembimbing Tugas Akhir:

1. Ahmad Saikhu, S.Si., M.T. .....
   NIP:197107182006041001 (Pembimbing 1)

2. Rully Soelaiman, S.Kom., M.Kom. .....
   NIP:197002131994021001 (Pembimbing 2)

KEMENTERIAN RISET, TEKNOLOGI DAN PENDIDIKAN TINGGI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
FAKULTAS TEKNOLOGI INFORMASI
JURUSAN TEKNIK INFORMATIKA

SURABAYA
JANUARI 2016

(Halaman ini sengaja dikosongkan)

IMPLEMENTASI ALGORITMA DETEKSI *SPAM* YANG TERSISIPI INFORMASI CITRA DENGAN METODE SVM DAN *RANDOM FOREST*

Nama : **Agus Tri Wibowo**
NRP : **5110100156**
Jurusan : **Teknik Informatika – FTIf ITS**
Dosen Pembimbing I : **Ahmad Saikhu, S.Si., M.T**
Dosen Pembimbing II : **Rully Soelaiman, S.Kom., M.Kom.**

Abstrak

Email spam dapat didefinisikan sebagai *email* sampah yang kedatangannya tidak dikehendaki oleh penerima *email* tersebut. Perkembangan metode deteksi *email spam* mengikuti perkembangan metode dari para *spammer*. Mulai deteksi *email spam* berdasarkan teks dan atribut dari *email* hingga *citra* yang terkandung pada email tersebut.

Pada Tugas Akhir dilakukan implementasi deteksi *spam* menggunakan *random forest* dan SVM berdasarkan fitur tekstur pada citra yang disisipkan pada *email*. Untuk mengklasifikasikan citra yang *spam* dengan yang bukan *spam* diperlukan ekstraksi fitur teksur yang berjumlah total 302 fitur. Karena banyaknya dimensi fitur yang perlu untuk diproses, maka metode PCA digunakan untuk mereduksi dimensi fitur menjadi jauh lebih kecil.

Dari hasil uji kinerja didapatkan rata-rata nilai akurasi, *precision* dan *recall* 98.64%, 99.02% dan 98.29% untuk *random forest* serta 97.84%, 98.26% dan 97.43% untuk SVM tanpa melakukan PCA pada *dataset Image Spam Hunter* (ISH). Jika digunakan PCA nilai-nilai tersebut menjadi 97.28%, 97.69% dan 96.94% untuk *random forest* serta 97.22%, 98.04% dan 96.53%

untuk SVM. Jika ditinjau dari waktu komputasi, penggunaan PCA menyebabkan waktu komputasi menjadi 3.10 kali lebih cepat untuk SVM dan 1.82 kali lebih cepat untuk *random forest*. Sedangkan jika digunakan untuk mendeteksi *dataset* baru, rata-rata nilai akurasi, *precision* dan *recall* untuk SVM didapatkan 81.01%, 84.81% dan 78.82% lebih tinggi dibandingkan *random forest* dengan rata-rata72.47%, 67.09% dan 75.18%.

Kata kunci: ***spam, email,* tekstur, *random forest, support vector machine.***

IMPLEMENTATION OF IMAGE CONTAINED SPAM DETECTION ALGORITHM USING SVM AND RANDOM FOREST

Name	**: Agus Tri Wibowo**
NRP	**: 5110100156**
Department	**: Informatics Engineering, FTIf, ITS**
Supervisor I	**: Ahmad Saikhu, S.Si., M.T**
Supervisor II	**: Rully Soelaiman, S.Kom., M.Kom.**

Abstract

Spam email can be defined as trash email which was not desired by the said email receiver. Development of spam detection follows the development of spammer's method to spread spam. From spam detection using text body and attribute in the email to image contained within email.

This Undergraduate Thesis was proposed to implement spam detection using random forest and SVM by using texture features of image contained within spam.To classify spam images, 302 texture features must be extracted from the said images. To reduce this huge dimension PCA was used.

From performance test, it was found that the proposed method resulted average accuracy, precision dan recall of 98.64%, 99.02% and 98.29% for random forest also 97.84%, 98.26% and 97.43% for SVM using Image Spam Hunter (ISH) dataset. If PCA were used those values would become 97.28%, 97.69% and 96.94% for random forest also 97.22%, 98.04% and 96.53% for SVM. By using PCA computation time would become 3.10 times faster for SVM dan 1.82 times faster for random forest. When new dataset were used, the proposed method resulted average accuracy, precision and recall of 81.01%,

84.81% and 78.82% for SVM and 72.47%, 67.09% dan 75.18% for random forest.

Keywords: spam, email, texture, random forest, support vector machine.

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Email spam merupakan *email* yang tidak diinginkan yang masuk pada *inbox email*. *Email* ini biasanya berisi pesan atau citra yang berupa iklan produk atau bahkan tipuan. *Email spam* masih merupakan masalah yang masih melandai dunia *internet* sampai saat ini. *Email spam* biasanya dikirimkan secara massal oleh *botnet* yang dikendalikan *spammer*. Menurut studi yang telah dilakukan oleh The Radicati Group, Inc., pada tahun 2013 *email spam* telah mencakup 84% dari keseluruhan *traffic email* per harinya [1]. Tentu saja hal ini menimbulkan banyak kerugian baik itu bagi pengguna *email* maupun penyedia layanan *email*. *Email spam* merugikan bagi pengguna *email* karena *email spam* telah membuang-buang waktu pengguna *email* ketika mengecek *email* mereka, bahkan tidak sedikit pula yang menjadi korban penipuan karena *email spam* yang diterimanya. Sedangkan bagi penyedia layanan *email, email spam* telah memakan sumber daya *bandwidth* yang terbatas dan media penyimpanan *email* yang telah disediakan.

Sudah begitu banyak juga metode yang diajukan dari berbagi riset yang dilakukan oleh kalangan akademisi dan perusahaan untuk memfilter *email spam.* Salah satu metode yang populer adalah *Bayesian Filtering* yang telah digunakan oleh *SpamAssassin* sebuah perangkat lunak untuk *email spam filtering* [2].

Seiring dengan berkembangnya teknologi *email spam filtering*, para *spammer* juga mengembangkan metode yang digunakan untuk menyebarkan *email spam* mereka salah satunya yaitu dengan menyamarkan pesan yang biasanya menggunakan teks biasa dengan menggunakan medium teks yang ada pada citra sehingga hal ini sangat menyulitkan perangkat *spam filtering* yang telah dijalankan oleh penyedia layanan *email.* Untuk mengatasi hal tersebut telah digunakan *Optical Character*

Recognition (OCR) untuk mengenali teks yang ditanamkan pada citra, akan tetapi *spammer* kemudian menggunakan CAPTCHA (*Completely Automated Public Turing Test to Tell Computer and Human Apart*), dengan ini *spammer* bisa men-*distort*, menambahkan latar yang berwarna-warni atau ber-*noise* sehingga hanya manusia saja yang bisa membaca teks pada citra [3] [4]. Oleh karena itu dikembangkan teknik *email spam filtering* yang berdasarkan fitur *low-level* yang terkandung di dalam citra pada *email spam*.

Pada Tugas Akhir ini, citra pada *email* akan diklasifikasikan berdasarkan fitur tekstur yang terkandung pada citra itu dengan menggunakan metode klasifikasi *Random Forest* sebagai citra *spam* atau *ham* (citra bukan *spam*), kemudian akan digunakan juga metode klasifikasi SVM (*Support Vector Machine*) sebagai pembanding.

1.2 Rumusan Permasalahan

Rumusan masalah yang dapat diangkat dalam Tugas Akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana cara menerapkan metode klasifikasi SVM dan *Random Forest* pada deteksi citra *spam* dengan melakukan PCA pada *dataset*?
2. Bagaimanakah pengaruh penggunaan *Principal Component Analysis* pada kinerja klasifikasi?
3. Bagaimanakah perbandingan kinerja SVM dan *Random Forest* pada deteksi citra *spam*?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Ekstraksi fitur tekstur akan menggunakan perangkat lunak MaZda [5].
2. Implementasi menggunakan bahasa pemrograman Python dengan pustaka SciPy [6], NumPy [7], scikit-image [8], scikit-learn [9], serta Pandas [10].

3. Implementasi menggunakan metode klasifikasi *Random Forest* dan SVM sebagai pembanding.
4. *Dataset* yang digunakan adalah *Image Spam Hunter dataset* [11].

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Mengetahui penerapan klasifikasi dengan metode *Random Forest* dan SVM untuk mendeteksi citra *spam* dengan melakukan PCA pada *dataset*.
2. Mengetahui pengaruh *Principal Component Analysis* pada klasifikasi citra *spam* dengan menggunakan *Random Forest* dan SVM.
3. Mengevaluasi kinerja *Random Forest* dibandingkan dengan SVM dengan melakukan uji coba.

1.5 Manfaat

Tugas akhir ini dikerjakan dengan harapan mendapatkan metode yang cepat dan efisien untuk mendeteksi citra *spam*.

1.6 Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir.
   Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir. Proposal Tugas Akhir yang diajukan memiliki gagasan untuk mengimplementasikan algoritma deteksi *spam* yang tersisipi informasi citra dengan menggunakan SVM dan *Random Forest*.

2. Studi literatur
   Pada tahap ini dilakukan pencarian, pengumpulan, pembelajaran dan pemahaman informasi dan literatur yang diperlukan untuk mengiimplementasikan metode deteksi *spam* yang tersisipi informasi citra. Dasar informasi yang diperlukan pada pembuatan implementasi ini di antaranya

mengenai fitur-fitur tekstur yang bisa diekstrak dari citra, cara menggunakan perangkat lunak MaZda dan cara penggunaan pustaka Sckit-learn. Informasi dan literatur didapatkan dari buku dan sumber-sumber informasi lain yang berhubungan.

3. Perancangan perangkat lunak
   Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep teknologi dari perangkat lunak yang ada. Tahap ini mendefinisikan alur dari implementasi. Langkah-langkah yang dikerjakan juga didefinisikan pada tahap ini. Pada tahapan ini dibuat *prototype* sistem, yang merupakan rancangan dasar dari sistem yang akan dibuat. Serta dilakukan desain suatu sistem dan desain proses-proses yang ada.

4. Implementasi perangkat lunak
   Implementasi merupakan tahap membangun rancangan sistem yang telah dibuat. Pada tahapan ini merealisasikan apa yang terdapat pada tahapan sebelumnya, sehingga menjadi sebuah sistem yang sesuai dengan apa yang telah direncanakan.

5. Pengujian dan evaluasi
   Pada tahapan ini dilakukan uji coba terhadap perangkat lunak yang telah dibuat. Pengujian dan evaluasi akan dilakukan dengan melihat kesesuaian dengan perencanaan. Tahap ini dimaksudkan juga untuk mengevaluasi jalannya sistem, mencari masalah yang mungkin timbul dan mengadakan perbaikan jika terdapat kesalahan.

6. Penyusunan buku Tugas Akhir.
   Pada tahapan ini disusun buku yang memuat dokumentasi mengenai pembuatan serta hasil dari implementasi perangkat lunak yang telah dibuat.

1.7 Sistematika Penulisan

Buku Tugas Akhir ini disusun dengan sistematika penulisan sebagai berikut:

BAB I. PENDAHULUAN

Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

BAB II. DASAR TEORI

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini.

BAB III. PERANCANGAN PERANGKAT LUNAK

Bab ini berisi tentang desain sistem yang disajikan dalam bentuk diagram alir dan *pseudocode*.

BAB IV. IMPLEMENTASI

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa kode yang digunakan untuk proses implementasi.

BAB V. UJI COBA DAN EVALUASI

Bab ini menjelaskan kemampuan perangkat lunak dengan melakukan pengujian kebenaran dan pengujian kinerja dari sistem yang telah dibuat.

BAB VI. KESIMPULAN DAN SARAN

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

(Halaman ini sengaja dikosongkan)

BAB II
DASAR TEORI

Bab ini berisi penjelasan teori-teori yang berkaitan dengan metode yang diajukan pada pengimplementasian perangkat lunak. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap sistem yang dibuat dan berguna sebagai penunjang dalam pengembangan perangkat lunak.

2.1 Fitur Tekstur Citra

Walaupun tidak ada definisi yang pasti untuk tekstur citra, hal ini dapat dikenal dan dipercaya sebagai sumber yang kaya akan informasi dari citra itu sendiri. Pada umumnya, tekstur merupakan pola visual yang kompleks yang terdiri dari berbagai subpola seperti ukuran, kecerahan, warna, kemiringan, dan lain-lain [12]. Pada Tugas Akhir, pola tekstur citra yang digunakan oleh penulis untuk mendeteksi citra *spam* adalah *histogram*, *gradient*, *run length matrix*, *co-ocurence matrix*, *autoregressive model*, dan *wavelet*. Masing-masing pola tekstur tersebut bila dihitung akan didapatkan beberapa fitur yang merepresentasikan tekstur dari citra.

Persamaan untuk menghitung fitur tekstur yang ditulis pada buku ini diasumsikan bahwa intensitas pada citra bernilai dari 1 sampai N_g, dimana $N_g = 2^k$, dan k adalah nilai *bit* tiap pixel pada citra.

2.1.1 *Histogram*

Histogram merupakan representasi grafik dari distribusi intensitas warna dari citra. Pada persamaan di bawah nilai *histogram* untuk intensitas i direpresentasikan sebagai p(i). Nilai *histogram* yang digunakan pada persamaan ini perlu untuk dinormalkan terlebih dahulu yaitu dengan membagi nilai *histogram* dengan banyak total *pixel* pada citra. Fitur tekstur yang dihitung berdasarkan *histogram* adalah *mean*, *variance*, *skewness*, dan *kurtosis*.

1. *Mean*

Fitur *mean* merupakan rata-rata intensitas dari citra. Fitur ini dihitung menggunakan Persamaan 2.1 [13].

$$\mu = \sum_{i=1}^{N_g} i\, p(i) \tag{2.2}$$

2. *Variance*

Fitur *variance* menggambarkan variasi dari nilai intensitas pada citra berdasarkan pada fitur *mean*. Untuk menghitung fitur ini digunakan Persamaan 2.2 [13].

$$\sigma^2 = \sum_{i=1}^{N_g} (i-\mu)^2 p(i) \tag{2.2}$$

3. *Skewness*

Fitur *skewness* menjadi indikasi kesimetrian *histogram*. Jika bernilai nol maka *histogram* simetris terhadap *mean* sedangkan jika positif bercondong di atas *mean* dan jika bernilai negatif *histogram* bercondong di bawah *mean*. Fitur ini dihitung menggunakan Persamaan 2.3 [13].

$$\mu_3 = \sigma^{-3} \sum_{i=1}^{N_g} (i-\mu)^3 p(i) \tag{2.3}$$

4. *Kurtosis*

Fitur *kurtosis* digunakan untuk mengukur kedataran dari *histogram*. Fitur ini dihitung menggunakan Persamaan 2.4 [13].

$$\mu_4 = \sigma^{-4} \sum_{i=1}^{N_g} (i - \mu)^4 p(i) - 3 \quad (2.4)$$

5. *Percentile*

Percentile merupakan nilai x yang didapatkan dari histogram yang memenuhi syarat Persamaan 2.5.

$$Percentile \leq \sum_{i=1}^{x} p(i) \quad (2.5)$$

2.1.2 *Gradient*

Gradient merupakan pola tekstur yang menggambarkan arah perubahan intensitas warna pada citra. Untuk menghitung fitur *gradient* pada *pixel* x(i,j) yang diilustrasikan pada Gambar 2.1 digunakan Persamaan 2.6 [13]. Persamaan ini akan meghasilkan nilai dari *gradient* dari 3x3 *pixel* yang berpusat di x(i,j). Nilai hasil dari persamaan ini bisa disebut juga sebagai *absolute gradient value* dari x(i,j) (ABSV(i,j)). Sedangkan fitur tekstur yang bisa didapatkan dari *gradient* adalah *mean absolute gradient, variance of absolute gradient, skewness of absolute gradient, kurtosis of absolute gradient*.

A	*B*	*C*	*D*	*E*
F	*G*	*H*	*I*	*J*
K	*L*	$x(i,j)$	*N*	*O*
P	*Q*	*R*	*S*	*T*
U	*V*	*W*	*Y*	*Z*

Gambar 2.1 *Pixel* x(i,j) pada citra

$$ABSV(i,j) = \sqrt{(R-H)^2 + (N-L)^2} \tag{2.6}$$

Berikut merupakan persamaan-persamaan yang digunakan untuk menghitung fitur tekstur yang berdasarkan dari *gradient* dengan M merupakan banyaknya *pixel* pada citra.

1. *Mean of Absolute Gradient*
   Untuk menghitung fitur ini digunakan Persamaan 2.7 [13].

$$GrMean = \frac{1}{M}\sum_{i,j} ABSV(i,j) \tag{2.7}$$

2. *Variance of Absolute Gradient*
   Untuk menghitung fitur ini digunakan Persamaan 2.8 [13].

$$GrVariance = \frac{1}{M}\sum_{i,j} (ABSV(i,j) - GrMean)^2 \tag{2.8}$$

3. *Skewness of Absolute Gradient*
   Untuk menghitung fitur ini digunakan Persamaan 2.9 [13].

$$GrSkewness = \frac{1}{\left(\sqrt{GrVariance}\right)^3}\frac{1}{M}\sum_{i,j}(ABSV(i,j) - GrMean)^3 \tag{2.9}$$

4. *Kurtosis of Absolute Gradient*
   Untuk menghitung fitur ini digunakan Persamaan 2.10 [13].

$$GrSkewness = \frac{1}{\left(\sqrt{GrVariance}\right)^4} \frac{1}{M} \sum_{i,j} (ABSV(i,j) - GrMean)^4 - 3 \quad (2.10)$$

2.1.3 *Run Length Matrix*

Run length matrix merupakan matriks yang berisi *gray level run* dari sebuah citra. Sedangkan *grey level run* itu sendiri merupakan set dari sekumpulan *pixel* yang berderetan secara linear dan mempunyai intensitas warna yang sama [14]. Selain berisi banyaknya *gray level run*, *run length matrix* juga berisi arah penghitungan dari banyaknya *gray level run*, yaitu: horisontal , vertikal, diagonal 45 derajat dan diagonal 135 derajat. Perputaran sudut untuk arah penghitungan berlawanan dengan arah jarum jam. Misalkan pada sebuah *run length matrix* dari sebuah citra, elemen p(i,j) menunjukkan banyaknya *grey level run* yang sepanjang j *pixel*. Sebagai contoh, misalkan sebuah citra berukuran 4x4 mempunyai intensitas warna berkisar 0 sampai 3 seperti yang diilustrasikan pada Gambar 2.2, maka *run length matrix* untuk citra tersebut seperti yang telah tercantum pada Tabel 2.1, Tabel 2.2, Tabel 2.3 serta Tabel 2.4. Masing-masing table tersebut secara berurutan merupakan *run length matrix* untuk arah horisontal, diagonal 45^o, vertikal, dan diagonal 135^o.

0	1	2	3
0	2	3	3
2	1	1	1
3	0	3	0

Gambar 2.2 Citra yang berintensitas 0-3

Tabel 2.1 *Run length matrix* arah horisontal untuk Gambar 2.2

0^o		panjang			
		1	2	3	4
Intensitas	0	4	0	0	0
	1	1	0	1	0
	2	3	0	0	0
	3	3	1	0	0

Tabel 2.2 *Run length matrix* arah diagonal 45^o untuk Gambar 2.2

45^o		panjang			
		1	2	3	4
Intensitas	0	4	0	0	0
	1	4	0	0	0
	2	0	0	1	0
	3	3	1	0	0

Tabel 2.3 *Run length matrix* arah vertikal untuk Gambar 2.2

90^o		panjang			
		1	2	3	4
intensitas	0	2	1	0	0
	1	4	0	0	0
	2	3	0	0	0
	3	3	1	0	0

Tabel 2.4 *Run length matrix* arah diagonal 135^o untuk Gambar 2.2

135^o		panjang			
		1	2	3	4
intensitas	0	4	0	0	0
	1	4	0	0	0
	2	3	0	0	0
	3	5	0	0	0

Fitur-fitur tekstur yang bisa dihitung berdasarkan *run length matrix* adalah *short run emphasis inverse moments*, *long run emphasis moments*, *grey level nonuniformity*, *run length nonuniformity* serta *fraction of image in runs*. Untuk menghitung fitur-fitur tersebut perlu dihitung terlebih dahulu konstanta C dengan menggunkan Persamaan 2.11.

$$C = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j) \tag{2.11}$$

1. *Short Run Emphasis Inverse Moments*

Untuk menghitung fitur ini digunakan Persamaan 2.12 [13] [14].

$$ShrtREmph = \left(\sum_{i=1}^{N_g} \sum_{j=1}^{N_r} \frac{p(i,j)}{j^2} \right) / C \tag{2.12}$$

2. *Long Run Emphasis Moments*

Untuk menghitung fitur ini digunakan Persamaan 2.13 [13] [14].

$$LngREmph = \left(\sum_{i=1}^{N_g} \sum_{j=1}^{N_r} j^2 p(i,j) \right) / C \tag{2.13}$$

3. *Grey Level Nonuniformity*

Untuk menghitung fitur ini digunakan Persamaan 2.14 [13] [14].

$$GLevNonUni = \left\{ \sum_{i=1}^{N_g} \left(\sum_{j=1}^{N_r} p(i,j) \right)^2 \right\} / C \tag{2.14}$$

4. *Run Length Nonuniformity*

Untuk menghitung fitur ini digunakan Persamaan 2.15 [13] [14].

$$RLNonUni = \left\{ \sum_{i=1}^{N_r} \left(\sum_{j=1}^{N_g} p(i,j) \right)^2 \right\} / C \tag{2.15}$$

5. *Fraction of Image in Runs*

Untuk menghitung fitur ini digunakan Persamaan 2.16 [13] [14].

$$Fraction = \left(\sum_{i=1}^{N_g} \sum_{j=1}^{N_r} p(i,j) \right) / \left(\sum_{i=1}^{N_g} \sum_{j=1}^{N_r} jp(i,j) \right) \tag{2.16}$$

2.1.4 *Co-occurrence Matrix*

Co-occurrence matrix yang dinotasikan sebagai P(i, j, d, θ) merupakan matriks yang berisi banyaknya kejadian dimana dua *pixel* yang bertentanggaan yang keduanya mempunyai intensitas masing-masing i dan j yang berjarak d dan keduanya berada dalam satu deret yang berarah θ [15]. Misalkan P(2, 3, 2, 0^o) menunjukkan banyakanya kejadian dua *pixel* yang masing-masing berintensitas 2 dan 3 yang berjarak d dan berada dalam satu deret yang berarah horisontal. Jika ρ((k,l), (m,n)) merupakan jarak antara dua *pixel* yang berada pada posisi (k,l) dan (m,n) maka untuk menghitung jarak antara dua titik tersebut digunakan Persamaan 2.17.

$$\rho\big((k,l),(m,n)\big) = \max\{|k-m|,|l-n|\} \tag{2.17}$$

Sebagai contoh, *co-occurrence matrix* dengan parameter d=1 untuk citra pada Gambar 2.3 dapat dilihat pada Tabel 2.5, Tabel 2.6, Tabel 2.7 dan Tabel 2.8. Masing-masing tabel secara berurutan merupakan *co-occurrence matrix* dengan parameter θ 0^o, 45^o, 90^o serta 135^o. Contoh penghitungan untuk P(0, 0, 1, 0^o) yang terdapat pada Tabel 2.5 didapatkan nilai 4. Hal ini menandakan bahwa *co-occcurrence* untuk intensitas i=0 dan j=0 dengan jarak d=1 dan pada deret horisontal θ=0^o ditemukan 4 kali pada citra. Sedangkan untuk set dari P(0, 0, 1, 0^o) adalah {(0,0), (0,1)}, {(0,1), (0,0)}, {(1,0), (1,1)} dan {(1,1), (1,0)}; masing-masing menunjukkan posisi *co-occurrence* dari 2 *pixel* pada Gambar 2.3.

0	0	1	1
0	0	1	1
0	2	2	2
3	2	3	3

Gambar 2.3 Citra yang berintensitas 0-3

Tabel 2.5 *Co-occurrence matrix* P(i, j, d, θ) untuk d=1 dan θ=0°

0°		intensitas			
		0	1	2	3
intensitas	0	4	2	1	0
	1	2	4	0	0
	2	1	0	6	1
	3	0	0	1	2

Tabel 2.6 *Co-occurrence matrix* P(i, j, d, θ) untuk d=1 dan θ=45°

45°		intensitas			
		0	1	2	3
intensitas	0	4	1	0	0
	1	1	2	2	0
	2	0	2	4	1
	3	0	0	1	0

Tabel 2.7 *Co-occurrence matrix* P(i, j, d, θ) untuk d=1 dan θ=90°

90°		intensitas			
		0	1	2	3
intensitas	0	6	0	2	0
	1	0	4	2	0
	2	2	2	2	2
	3	0	0	2	0

Tabel 2.8 *Co-occurrence matrix* P(i, j, d, θ) untuk d=1 dan θ=135°

135°		intensitas			
		0	1	2	3
intensitas	0	2	1	3	0
	1	1	2	1	0
	2	3	1	0	2
	3	0	0	2	0

Co-occurrence matrix di atas masih belum dinormalisasi untuk menggunakan matriks diatas sebagai dasar penghitungan fitur tekstur perlu dinormalisasi dengan membagi tiap-tiap nilai pada matriks dengan konstanta R yang bisa didapatkan dengan Persamaan 2.18. Pada persamaan tersebut N_x dan N_y merupakan ukuran *pixel* kolom dan baris dari citra.

$$R = \begin{cases} 2N_y(N_x - 1), & \text{jika } \theta = 0^o \\ 2(N_y - 1)(N_x - 1), & \text{jika } \theta = 45^o \\ 2N_x(N_y - 1), & \text{jika } \theta = 90^o \\ 2(N_x - 1)(N_y - 1), & \text{jika } \theta = 130^o \end{cases} \quad (2.18)$$

Fitur tekstur yang didapatkan dari *co-occurrence matrix* adalah *angular second moment*, *contrast*, *correlation*, *sum of squeares*, *inverse difference moment*, *sum average*, *sum variance*, *sum entroy*, *difference variance* serta *difference entropy*.

Untuk persamaan yang digunakan untuk menghitung nilai fitur-fitur tersebut perlu diketahui persamaan-persamaan berikut [13] [15]:

$$p_x(i) = \sum_{j=1}^{N_g} p(i,j) \tag{2.19}$$

$$p_y(j) = \sum_{i=1}^{N_g} p(i,j) \tag{2.20}$$

$$p_{x+y}(k) = \sum_{i=1}^{N_g} \sum_{\substack{j=1 \\ i+j=k}}^{N_g} p(i,j)\,, \quad k = 2,3,4,\dots,2N_g \tag{2.21}$$

$$p_{x-y}(k) = \sum_{i=1}^{N_g} \sum_{\substack{j=1 \\ |i-j|=k}}^{N_g} p(i,j)\,, \quad k = 0,1,2,\dots,N_g - 1 \tag{2.22}$$

$$\mu_x = \sum_{i=1}^{N_g} i\, p_x(i) \tag{2.23}$$

$$\mu_y = \sum_{i=1}^{N_g} i\, p_y(i) \tag{2.24}$$

$$\sigma_x = \sqrt{\sum_{i=1}^{N_g} (i - \mu_x)^2 p_x(i)} \tag{2.25}$$

$$\sigma_y = \sqrt{\sum_{i=1}^{N_g} (i - \mu_y)^2 p_y(i)} \tag{2.26}$$

Jika p(i,j) merupakan nilai *co-occurrence matrix* untuk *pixel* berintensitas i dan j yang telah dinormalisasi, maka persamaan-persamaan yang digunakan untuk menghitung fitur tekstur yang berdasarkan *co-occurrence matrix* untuk citra yang berintensitas 1 sampai N_g adalah sebagai berikut:

1. *Angular Second Moment*

Untuk menghitung fitur ini digunakan Persamaan 2.27 [13] [15].

$$AngScMom = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j)^2 \tag{2.27}$$

2. *Contrast*

Untuk menghitung fitur ini digunakan Persamaan 2.28 [13] [15].

$$Contrast = \sum_{n=0}^{N_g-} n^2 \left\{ \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j) \right\} \quad (2.28)$$

3. *Correlation*

Untuk menghitung fitur ini digunakan Persamaan 2.29 [13] [15].

$$Correlat = \frac{\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} i\, j\, p(i,j) - \mu_x \mu_y}{\sigma_x \sigma_y} \quad (2.29)$$

4. *Sum of Squares*

Untuk menghitung fitur ini digunakan Persamaan 2.30 [13] [15].

$$SumOfSqs = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} (i - \mu_x)^2\, p(i,j) \quad (2.30)$$

5. *Inverse Difference Moment*

Untuk menghitung fitur ini digunakan Persamaan 2.31 [13] [15].

$$InvDfMom = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} \frac{1}{1 + (i-j)^2}\, p(i,j) \quad (2.31)$$

6. *Sum Average*

Untuk menghitung fitur ini digunakan Persamaan 2.32 [13] [15].

$$SumAverg = \sum_{i=2}^{2N_g} ip_{x+y}(i) \tag{2.32}$$

7. *Sum Variance*

Untuk menghitung fitur ini digunakan Persamaan 2.33 [13] [15].

$$SumVarnc = \sum_{i=2}^{2N_g} (i - SumAverg)^2 \, p_{x+y}(i) \tag{2.33}$$

8. *Sum Entropy*

Untuk menghitung fitur ini digunakan Persamaan 2.34 [13] [15].

$$SumEntrp = -\sum_{i=2}^{2N_g} p_{x+y}(i) \log\{p_{x+y}(i)\} \tag{2.34}$$

9. *Entropy*

Untuk menghitung fitur ini digunakan Persamaan 2.35 [13] [15].

$$Entropy = -\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j) \log p(i,j) \tag{2.35}$$

10. *Difference Variance*

Untuk menghitung fitur ini digunakan Persamaan 2.36 [13] [15].

$$DifVarnc = \sum_{i=0}^{N_g-1} (i - \mu_{x-y})^2 \, p_{x-y}(i) \tag{2.36}$$

11. *Difference Entropy*

Untuk menghitung fitur ini digunakan Persamaan 2.37 [13] [15].

$$DiffEntrp = -\sum_{i=0}^{N_g-1} p_{x-y}(i) \log\{p_{x-y}(i)\} \tag{2.37}$$

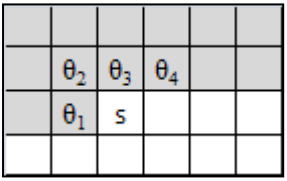
2.1.5 *Autoregressive Model*

Autoregressive model mengasumsikan bahwa terdapat interaksi antar *pixel* pada citra dimana intensitas dari suatu *pixel* merupakan jumlah dari *pixel* tetangga yang masing-masing mempunyai boot tertentu [13] [16]. Dengan asumsi bahwa citra f merupakan *zero-mean random field* atau citra yang seluruh intensitas dari *pixel* berata-rata nol, maka intensitas dari suatu pixel f_s dari citra tersebut dapat didefinisikan dengan menggunakan Persamaan 2.38.

$$f_s = \sum_{r \in N_s} \theta_r f_r + e_s \tag{2.38}$$

Notasi f_s adalah intensitas dari *pixel* pada citra dengan posisis s, e_s melambangkan *independent and identically*

distributed (i.i.d) *noise*, N_s adalah set *pixel* tetangga dari *pixel* berposisi s, serta θ_r merupakan parameter dari vektor model, jika $\{\theta_s\}$ merupakan set dari parameter vektor model dari *pixel* s maka set ini juga valid untuk seluruh *pixel* pada citra. Pada Gambar 2.4 dapat dilihat pada area yang di arsir sebelah kiri dan atas dari *pixel* berposisi s merupakan *pixel-pixel* tetanggak yang bisa menjadi kandidat untuk menghitung intensitas dari *pixel* s.



Gambar 2.4 *Pixel* tentangga dari *pixel* s

Untuk *Autoregressive model* dari Gambar 2.4, parameter model terdiri dari standar deviasi σ dari *noise* e_s dan vektor dari model parameter $\theta=[\theta_1, \theta_2, \theta_3, \theta_4]$. Parameter-parameter ini dapat diestimasi menggunakan persamaan-persamaan berikut [13] [15]:

$$\hat{\theta} = \left(\sum_s w_s {w_s}^T\right)^{-1} \left(\sum_s w_s f_s\right) \tag{2.39}$$

$$\sigma^2 = \frac{1}{N^2}\sum_s \left(f_s - \hat{\theta} w_s\right) \tag{2.40}$$

dimana w_s = col[f_i, i∈N_s] merupakan matriks kolom yang berisi intensitas *pixel* tentangga dari *pixel* s dan diasumsikan bahwa citra berukuran NxN.

Fitur tekstur yang didapatkan berdasar *autoregressive model* adalah 4 nilai nilai yang ada di dalam matriks $\hat{\theta}$ dan nilai dari σ.

2.1.6 *Wavelet*

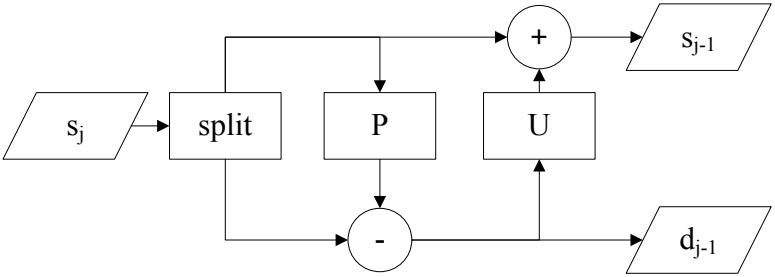
Fitur tekstur yang didapatkan berdasarkan *wavelet* tak terlepas dari *wavelet transform*. *Wavelet transform* merupakan salah satu metode pemrosesan citra berbasis *transform*. Berbeda dengan *Fourier transform* yang menggunakan fungsi berbasis sinus, *wavelet transform* berbasiskan *wave* kecil disebut *wavelet*, yang mempunyai frekuensi yang bervariasi dan dalam durasi yang terbatas. Hal ini mebuat *wavelet* seperti skor musik untuk citra, yang menjabarkan tidak hanya nada (frekuensi) untuk dimainkan tetapi juga kapan untuk memainkan nada itu. Sedangkan *fourier transform* hanya menyediakan informasi nada (frekuensi), informasi temporal (kapan memainkan nada itu) hilang ketika proses tranformasi [17].

2.1.6.1 *Discrete Wavelet Transform 1 Dimension*

Untuk mendapatkan hasil *Discrete Wavelete Transform* (DWT) digunakan metode *lifting*. Sedangkan *lifting* secara umum merupakan transformasi yang melibatkan *mean* dan *difference*. Dengan asumsi bahwa terdapat korelasi antara 2 *sample* yang beruntut, kemudian kita hitung *difference* dengan menggunakan *sample* pertama sebagai *prediction* dari *sample* kedua.

Kita juga menghitung *mean* dari 2 *sample*. Poin pertama, hal ini bisa dianggap sebagai proses untuk menyimpan karakteristik dari sinyal asli atau ekstraksi fitur penting dari sinyal. Untuk poin yang kedua berdasarkan bahwa *mean* dari 2 *sample* sinyal bisa merepresentasikan seluruh sinyal dengan hanya sepanjang separuh *sample* dari sinyal asli. Operasi ini disebut *update*.

Pada awalnya, misalkan sebuah sinyal sebagai *finite sequence* s_j dengan panjang 2^j. Kemudian ditransformasikan menjadi 2 *sequence* dengan masing-masing s_{j-1} dan d_{j-1} sepanjang 2^{j-1}. Perlu diperhatikan bahwa indeks dari *sequence* dimulai dari 0.



Gambar 2.5 Proses *prediction* dan *update*

Berikut penjelasan proses dari diagram pada Gambar 2.5:

- *Split*
  Tiap *sample* dikategorikan menjadi 2 berdasarkan indeks, yaitu genap dan ganjil.
- *Prediction*
  Jika sinyal mempunyai suatu struktur, dapat diperkirakan adanya korelasi antara *sample* dengan *sample* terdekatnya. Misalnya, *sample* pada indeks 2n, kita dapat memprediksi pada *sample* 2n+1 bernilai sama. Kemudian nilai pada 2n+1 diganti dengan nilai koreksi pada prediksi, yaitu *difference* atau selisih dari nilai pemrediksi dan yang diprediksi.

$$d_{j-1}[n] = s_j[2n+1] - s_j[2n] \tag{2.41}$$

- *Update*
  Setelah memprediksi *sample* ganjil, kemudian kita *update sample* genap dengan rata-rata 2 *sample* sebelumnya.

$$s_{j-1}[n] = s_j[2n] + d_{j-1}[n]/2 \tag{2.42}$$

Algoritma yang telah dijelaskan sebelumnya merupakan *one step lifting*. Berikut merupakan persamaan *lifting* untuk Haar *transform* [18]:

$$d_{j-1}^{(1)}[n] = s_j[2n+1] - s_j[2n] \tag{2.43}$$

$$s_{j-1}^{(1)}[n] = s_j[2n] + \frac{1}{2} d_{j-1}^{(1)}[n] \tag{2.44}$$

$$s_{j-1}[n] = \sqrt{2}\, s_{j-1}^{(1)}[n] \tag{2.45}$$

$$d_{j-1}[n] = \frac{1}{\sqrt{2}}\, d_{j-1}^{(1)}[n] \tag{2.46}$$

2.1.6.2 *Discrete Wavelet Transform 2 Dimension*

Untuk sinyal 2 dimensi, misalkan citra, tidak diperbolehkan untuk langsung menggunakan DWT 1 dimensi dengan menggabungkan tiap baris pada sinyal tersebut sebagai sinyal 1 dimensi. Hal ini dikarenakan adanya korelasi antar baris pada sinyal 2 dimensi.

Untuk menggunakan DWT 1 dimensi pada sinyal 2 dimensi diperlukan cara khusus. Berikut langkah-langkah yang digunakan:

1. Representasikan sinyal 2 dimensi sebagai matriks 2 dimensi dengan ukuran 2n x 2n.
2. Lakukan DWT 1 dimensi pada tiap-tiap baris matriks sehingga dapat dihasilkan 2 matriks yaitu matriks L untuk meyimpan *mean* dan matriks H untuk menyimpan *difference* dengan masing masing berukuran 2n x n.
3. Lakukan DWT 1 dimensi pada tiap-tiap kolom matriks L sehingga dapat dihasilkan 2 matriks yaitu matriks LL untuk meyimpan *mean* dan matriks LH untuk menyimpan *difference* dengan masing masing berukuran n x n.
4. Lakukan DWT 1 dimensi pada tiap-tiap kolom matriks H

sehingga dapat dihasilkan 2 matriks yaitu matriks HL untuk meyimpan *mean* dan matriks HH untuk menyimpan *difference* dengan masing masing berukuran n x n.

5. Hasil dari DWT 2 dimensi merupakan matriks LL, LH, HL, HH

Jika langkah-langkah di atas dilakukan satu kali maka akan didapatkan 4 citra hasil dekomposisi 1 *level*. Untuk mendapatkan hasil dekomposisi 2 *level*, maka langkah-langkah di atas dilakukan 2 kali iterasi dengan digunakan matriks LL sebagai inputan untuk iterasi kedua dan seterusnya.

2.1.6.3 *Wavelet Energy*

Wavelet energy merupakan hasil yang digunakan sebagai fitur tekstur yang digunakan untuk klasifikasi. Untuk mendapatakan fitur ini, masing-masing citra hasil dekomposisi dihitung nilai *wavelet energy* menggunakan Persamaan 2.47 [13].

$$E = \frac{\sum_{x,y}\left(d_{x,y}\right)^2}{n} \tag{2.47}$$

Notasi $d_{x,y}$ pada persamaan di atas merupakan nilai intensitas citra hasil dekomposisi pada posisi x dan y pada citra,. Sedangkan n adalah jumlah *pixel* total pada citra d.

2.2 *Principal Component Analysis*

Principal Component Analysis (PCA) merupakan proses untuk mereduksi dimensi dari *dataset* tanpa menghilangkan banyak informasi dari *dataset* itu sendiri dengan cara mengidentifikasi pola dari data. Dengan mereduksi dimensi dari *dataset* diharapkan untuk mengurangi *cost* dari komputasi dan eror dari estimasi parameter. Berikut merupakan langkah-langkah secara umum untuk melakukan PCA pada *dataset*:

1. Ambil *dataset* yang terdiri dari *sample* berdimensi *d* tanpa label kelas.

2. Hitung matriks *convariant* dari semua *dataset*.
3. Hitung vektor *eigen* dan nilai *eigen* yang terkait, kemudian masukkan dua nilai tersebut dalam *list eigen*.
4. Urutkan *list eigen* berdasarkan nilai *eigen* dari besar ke kecil, kemudian ambil sebanyak *k* vektor *eigen* untuk dijadikan matriks W berukuan *dxk*.
5. Kalikan matriks *dataset* berukuran *nxd* dengan matriks *eigen* berukuran *dxk*. Hasil dari perkalian tersebut merupakan matriks *dataset* baru berukuran *nxk*.

2.3 Metode-Metode Klasifikasi

2.3.1 *Random Forest*

Random forest merupakan metode *bagging* yaitu metode yang membangkitkan sejumlah *tree* dari data *sample* dimana pembuatan satu *tree* pada saat *training* tidak bergantung pada *tree* sebelumnya kemudian keputusan diambil berdasarkan *voting* terbanyak [19].

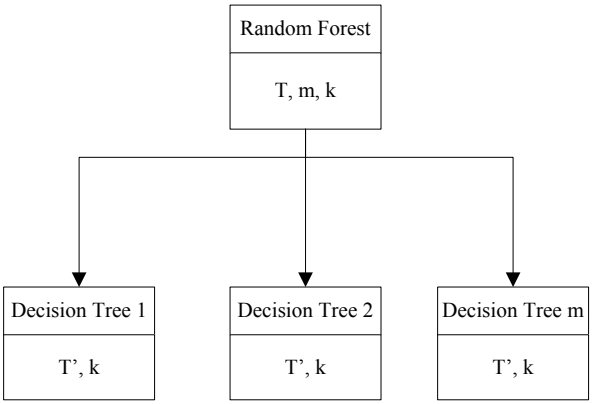
Dua konsep yang menjadi dasar dari *random forest* adalah membangun *ensemble* dari *tree* via *bagging* dengan *replacement* dan penyeleksian fitur secara acak untuk tiap *tree* yang dibangun. Hal yang pertama berarti tiap *sample* yang diambil dari *dataset* untuk *training tree* bisa dipakai lagi untuk *training tree* yang lain, sedangkan hal yang kedua berarti bahwa fitur yang digunakan pada saat *training* untuk tiap *tree* merupakan *subset* dari fitur yang dimiliki oleh *dataset* [20].

Klasifikasi berbasis *ensemble* hanya akan mempunyai performa yang maksimal jika antar *basic learner* mempunyai korelasi yang rendah. Untuk mengatasi hal ini, sebuah *ensemble* harus membangun *basic learner* yang lemah, karena *learner* yang kuat kemungkinan besar akan mempunyai korelasi yang tinggi dan biasanya juga menyebabkan *overfit*. Sedangkan *random forest* meminimalkan korelasi serta mempertahankan kekuatan

klasifikasi dengan cara melakukukan pengacakan pada proses *training*, yaitu dengan memilih sejumlah fitur secara acak dari semua fitur yang ada pada setiap melakukan *training tree*, kemudian menggunakannya menggunakan fitur-fitur yang terpilih untuk mendapatkan percabangan *tree* yang optimal. Berbeda dengan proses *training tree* pada *decision tree* biasa, proses *training tree* yang menjadi bagian dari *random forest* tidak menggunakan proses *pruning* akan tetapi precabangan akan terus dilakukan sampai ukuran batas *leaf* tercapai.

Random forest mempunyai dua parameter utama, yaitu: m jumlah *tree* yang akan dipakai dan *k* yaitu maksimal banyaknya fitur yang dipertimbangkan ketikan proses percabangan. Semakin banyak nilai m maka semakin bagus hasil klasifikasi, sedangkan untuk nilai k direkomendasikan sebesar akar kuadrat atau logaritma dari jumlah total fitur [21].

Pada Gambar 2.6, proses *training* untuk *random forest* menggunakan *dataset* T dengan sejumlah m *tree* sebagai *basic leaner* dan k fitur yang dipilih secara acak dari total fitur yang ada untuk percabangan pada setiap *tree*. Proses *training* pada setiap *tree* menggunakan *dataset* T' yang merupakan hasil dari *bootstrap* dari *dataset* yang dijadikan parameter untuk *random forest*. *Bootstrap* merupakan proses memilih *sample* dari *dataset* yang akan digunakan proses *training tree*, perlu diperhatikan bahwa dalam metode *ensemble*, *bootstrap* merupakan proses *sampling with replacement* sehingga, *sample* yang diambil untuk proses *training tree* yang satu masih bisa dipakai lagi untuk proses *training tree* yang lainnya.



Gambar 2.6 Ilustrasi *Random Forest*

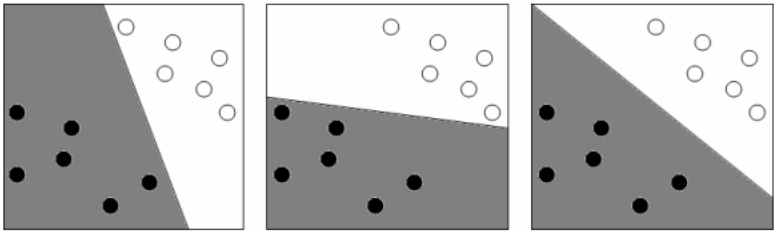
2.3.2 *Support Vector Machine*

Support Vector Machine (SVM) merupakan salah satu *supervised learning* dimana tiap label untuk setiap data masukan diprediksi berdasarkan data contoh yang sudah diberi label. Label ini menunjukkan tiap data itu termasuk dalam suatu kelas. SVM biasanya digunakan untuk klasifikasi biner. Cara kerja metode ini adalah dengan mencari *hyperplane* yang digunakan untuk memisahkan *d*-dimensional data menjadi 2 kelas [22]. Akan tetapi, di dalam praktek dalam dunia nyata, data sering kali tidak bisa dipisahkan secara linear. Sehingga diperkenalkan SVM dengan *"kernel induced feature space"* yang mana data yang akan diklasifikasikan akan dikonversi ke ruang dimensi yang lebih tinggi dimana data dapat dipisahkan [23].

SVM mengklasifikasikan *dataset* menjadi 2 kelas dengan cara membuat *hyperplane* (dalam contoh pada Gambar 2.7 berupa garis karena fitur *dataset* masih 2 dimensi) yang memisahkan *dataset* menjadi 2 area yang berbeda yaitu kelas +1 dan kelas -1.

Dari Gambar 2.7 terlihat berbagai cara untuk memisahkan *dataset*, akan tetapi untuk mendapatkan hasil yang optimal perlu untuk membuat garis yang berjarak paling jauh dengan 2 kelas

yang dipisahkan, terlihat pada gambar paling kanan merupakan yang paling optimal dari yang lain karena *hyperplane* memisahkan 2 kelas dengan menjaga jarak terjauh dari 2 kelas yang terpisakan.



Gambar 2.7 Ilustrasi 3 cara pemisahan *dataset*

Hyperplane pada SVM dibuat dengan membangun *decision function* baik itu yang linear maupun non-linear yang dapat memisahkan *dataset* pada 2 area yang berbeda. Fungsi matematika dasar pada linear SVM adalah sebagai berikut:

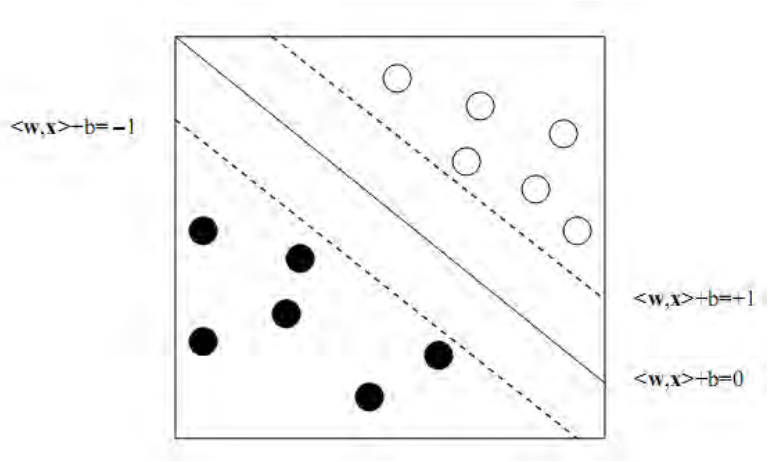
$$g(x) = sign(f(x)) \tag{2.48}$$

$$f(x) = \langle w, x \rangle + b \tag{2.49}$$

Fungsi g(x) merupakan fungsi yang menghasilkan label prediksi untuk input vektor x yaitu +1 atau -1. Sedangkan f(x) merupakan *decision function* yang digunakan untuk klasifikasi dengan parameter w yang merupakan vektor bobot dan b merupakan besaran skalar. Nilai dari <w, x> didapatkan dengan Persamaan 2.50, dimana d merupakan dimensi fitur dari input vektor x.

$$\langle w, x \rangle = \sum_{i=1}^{d} w_i x_i \tag{2.50}$$

Untuk mendapatkan *decision function* yang optimal memisahkan *dataset* diperlukan untuk melakukan optimasi dan pastinya untuk melakukan optimasi diperlukan batasan-batasan tertentu.



Gambar 2.8 Pemisahan *dataset* secara linear

Dari Gambar 2.8, dapat dilihat bahwa garis lurus yang merupakan <w,x>+b=0 adalah garis pemisah menjadi dua kelas (+1 untuk lingkaran putih dan -1 untuk lingkaran hitam) dari hal ini dapat disimpulkan untuk memisahkan 2 kelas diperlukan untuk memenuhi batasan berikut:

$$\begin{aligned} \langle w, x_i \rangle + b > 0, &\quad \text{untuk } y_i = 1 \\ \langle w, x_i \rangle + b < 0, &\quad \text{untuk } y_i = -1 \end{aligned} \tag{2.51}$$

Jika nilai kelas masing-masing batasan dikalikan dengan dengan ruas kiri tiap batasan maka 2 batasan di atas kemudian dapat diringkas menjadi satu batasan berikut:

$$(\langle w, x_i \rangle + b) y_i > 0, \quad \text{untuk } i = 1 \ldots n \tag{2.52}$$

Untuk membuat garis <w,x>+b=0 dapat memisahkan 2 kelas dengan jarak sejauh mungkin dari masing-masing kelas oleh karena itu dibuatlah garis putus-putus yaitu garis <w,x>+b=+1 dan <w,x>+b=-1 yang merupakan garis yang berada di antara garis <w,x>+b=0 dengan *dataset* dan sejajar dengan garis <w,x>+b=0. Dua garis tersebut digunakan untuk membantu menemukan margin maksimal antara *dataset* dan garis pemisah dengan cara menjauhkan garis-garis tersebut secara paralel dari garis pemisah. Dari hal ini didapatkan batasan selanjutnya yaitu:

$$y_i(\langle w, x_i\rangle + b) > 1, \quad \text{untuk } i = 1 \dots n \tag{2.53}$$

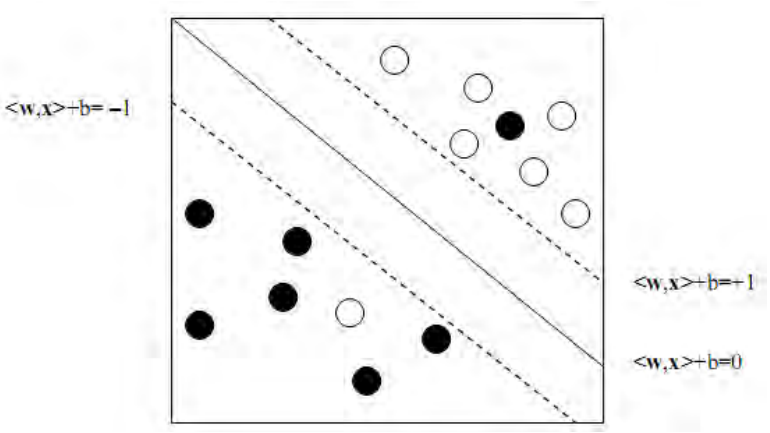
Untuk memaksimalkan jarak antara garis putus-putus seperti pada ilustrasi Gambar 2.8 maka digunakan Persamaan 2.54.

$$\begin{aligned} &\min_{w,b} \frac{1}{2} w.w \\ &\text{sehingga } y_i(\langle w, x_i\rangle + b) \geq 1 \\ &\text{untuk } i = 1 \dots m \end{aligned} \tag{2.54}$$

Persamaan 2.54 bisa untuk untuk memisahakan *dataset* yang bisa dipisahkan secara linear, jika *dataset* tidak bisa dipisahkan secara linear seperti pada Gambar 2.9 maka persamaan tersebut perlu untuk digeneralisasi sehingga memperbolehkan beberapa data untuk melanggar garis batas permisah dua kelas.

Setelah dilakukan generalisasi pada Persamaan 2.54 didapatkan persamaan berikut:

$$\begin{aligned} &\min_{w,b,\xi} \frac{1}{2} w.w + C \sum_{i=1}^{m} \xi_i \\ &\text{sehingga } y_i(\langle w, x_i\rangle + b) + \xi_i \geq 1 \\ &\text{untuk } i = 1 \dots m \end{aligned} \tag{2.55}$$



Gambar 2.9 Pemisahan *dataset* yang tidak bisa dipisahakan secara linear

Dari Persamaan 2.55 terdapat parameter C dan ξ_i. Untuk meminimalisasi *objective function* pada Persamaan 2.55 perlu untuk menjaga nilai ξ_i selalu kecil. Dengan adanya perkalian konstanta C pada penjumlahan ξ_i, bisa dikatakan semakin besar C maka *objective function* fokus untuk menjaga nilai ξ_i kecil tanpa memperhatikan jarak garis pemisah.

2.4 N-*Fold Cross Validation*

Untuk menguji kinerja klasifikasi penulis menggunakan 10-*fold cross validation*. Berikut langkah-langkah untuk melakukan *n-fold* cross *validation* [22]:

a. Bagi *dataset* menjadi *n* partisi yang berukuran sama, kemudian berikan nomor untuk tiap partisi 1 sampai *n*.
b. Lakukan *training* pada model klasifikasi menggunakan partisi 2 sampai *n*.
c. Lakukan *testing* model klasifikasi menggunakan partisi 1.

Ulangi lagi langkah a sampai c sehingga setiap partisi telah digunakan untuk *testing* dengan selalu menggunakan partisi yang lain untuk *training*.

(Halaman ini sengaja dikosongkan)

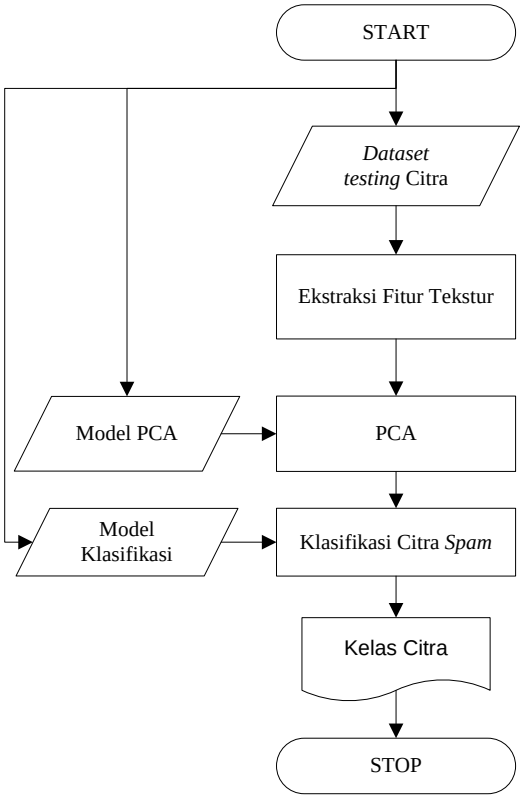
BAB III
PERANCANGAN PERANGKAT LUNAK

Pada bab ini akan dijelaskan perancangan perangkat lunak deteksi citra *spam*. Proses secara keseluruhan akan dijelaskan menggunakan diagram alir, kemudian untuk penjelasan yang lebih detail akan ditampilkan dalam bentuk *pseudocode*.

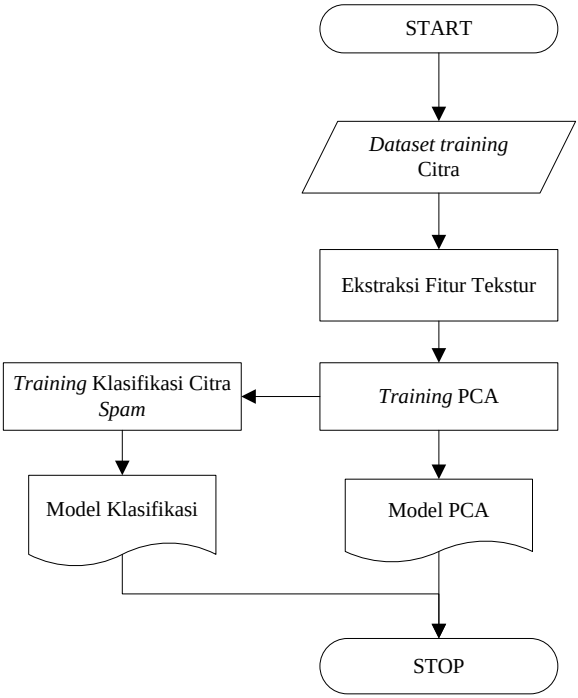
3.1 Perancangan Proses Secara Umum

Di dalam perancangan perangkat lunak terdapat 3 proses utama yaitu proses *training* model dan proses klasifikasi citra *spam* menggunakan model hasil *training*. Untuk proses ekstraksi fitur terdapat perancangan ekstraksi fitur *wavelet energy*, sedangkan untuk proses klasifikasi terdapat perancangan PCA dan metode klasifikasi *random forest*.

Untuk medeteksi citra *spam*, citra masukan akan masuk ke proses ekstraksi fitur tekstur. Pada proses ini fitur tekstur yang terkandung pada citra akan dihitung menggunakan perangkat lunak MaZda dan program implementasi untuk ekstraksi fitur *wavelete* energy. Perangkat lunak MaZda ini digunakan untuk ekstraksi fitur *histogram, gradient, run length matrix, co-occurrence matrix*, dan *autoregressive model*. Hasil dari ekstraksi fitur tekstur kemudian dijadikan masukan untuk proses *Principal Component Analysis* (PCA). Proses PCA ini digunakan untuk mereduksi dimensi fitur tekstur dari hasil ekstraksi fitur. Dengan melakukan PCA diharapakan agar *cost* untuk komputasi pada proses klasifikasi citra *spam* menjadi lebih kecil. Setelah fitur pada data masukan tereduksi pada proses PCA, citra kemudian akan dikasifikasikan sebagai citra *spam* atau citra *ham* pada proses klasifikasi citra *spam*. Proses deteksi citra *spam* secara umum dapat dilihat pada Gambar 3.1. Pada proses tersebut model PCA dan model klasifikasi didapatkan dari proses training yang dapat dilihat padaGambar 3.2.



Gambar 3.1 Proses deteksi citra *spam* secara umum



Gambar 3.2 Proses untuk *training* model PCA dan klasifikasi

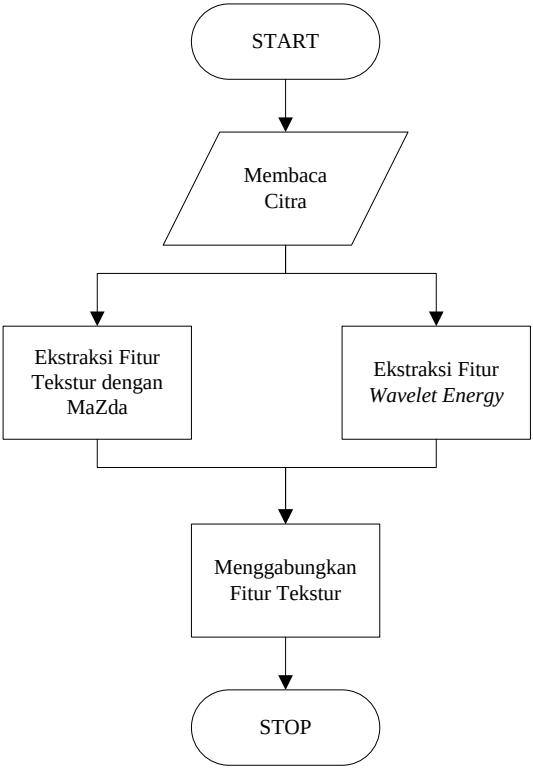
3.2 Perancangan Ekstraksi Fitur Tekstur

Proses ekstraksi fitur tekstur dari citra *dataset* dibagi menjadi 2 bagian. Bagian yang pertama, perangkat lunak MaZda digunakan untuk ektraksi fitur tekstur yang berdasarkan *histogram*, *gradient*, *run length matrix*, *co-occurrence matrix*, dan *autoregressive model*. Sedangkan untuk ekstraksi fitur tekstur yang berdasarkan *wavelet* dilakukan dengan menggunakan implementasi sendiri.

Seperti yang telah diilustrasikan pada Gambar 3., untuk setiap citra akan mempunyai fitur hasil ekstraksi menggunakan MaZda dan fitur hasil ekstraksi fitur *wavelet energy*. Setelah didapatkan hasil ekstraksi fitur tekstur yang berupa 2 berkas teks *report* yang berisi daftar nilai dari penghitungan fitur tekstur

untuk masing-masing citra maka daftar fitur tekstur dari 2 berkas tersebut akan digabung dan dimasukkan dalam *dataset*.

Pada subbab-subbab selanjutnya akan dijelaskan *pseudocode* untuk ekstraksi fitur tekstur dari citra *dataset*. Pada Tabel 3.1 dan Tabel 3.2 berisikan daftar variabel yang digunakan sedangkan pada Tabel 3.3 dan Tabel 3.4 berisi fungsi-fungsi yang digunakan pada *pseudocode*.



Gambar 3.3 Diagram alir ekstraksi fitur tekstur

Tabel 3.1 Daftar variabel yang digunakan dalam *pseudocode* ekstraksi fitur tekstur (Bagian Pertama)

No.	Nama Variabel	Tipe	Penjelasan
1	`data`	double	Sinyal input
2	`s`	double	*Sequence* sinyal *approximation*
3	`d`	double	*Sequence* sinyal detail
4	`n`	int	Panjang *sequence* dari data
5	`zeropadding`	boolean	Jika bernilai True maka akan dilakukan *zeropadding*
6	`LL`	double	Citra *approximation*
7	`HL`	double	Citra detail *horizontal*
8	`LH`	double	Citra detail *vertical*
9	`HH`	double	Citra detail *diagonal*
10	`row`	double	*Array* 1 dimensi
11	`L`	double	Variabel penyimpanan sinyal *approximation* hasil DWT per baris
12	`H`	double	Variabel penyimpanan sinyal detail hasil DWT per baris
13	`sum`	double	Untuk menyimpan nilai *wavelet energy*
14	`inp`	string	*Path* letak berkas citra
15	`level`	int	Maksimum level dekomposisi yang akan dilakukan
16	`wavenergy`	double	Array 2 dimensi berisi nilai *wavelet energy* untuk tiap level dekomposisi
17	`imgdir`	string	Direktori tempat citra *dataset*

Tabel 3.2 Daftar variabel yang digunakan dalam *pseudocode* ekstraksi fitur tekstur (Bagian Kedua)

No.	Nama Variabel	Tipe	Penjelasan
18	`reportdir`	string	Direktori tempat berkas *report*
19	`imgname`	string	Nama berkas citra
20	`rppath`	string	*Path* letak berkas *report*
21	`wlen`	double	Array berisi nilai hasil penghitungan *wavelet energy*
22	`output_file`	string	*Path* output berkas CSV
23	`report`	string	*Path* output berkas report MaZda
24	`value_list`	string	*Array* berisi nilai hasil *parsing report*
25	`wl_rpname`	string	*Path* output berkas *report wavelet energy*

Tabel 3.3 Daftar fungsi yang digunakan dalam *pseudocode* ekstraksi fitur tekstur (Bagian Pertama)

No.	Nama Fungsi	Penjelasan
1	`length`	Fungsi untuk mendapatkan panjang *array*
2	`dwt`	Fungsi untuk DWT 1 dimensi
3	`transpose`	Fungsi untuk *transpose array* 2 dimensi
4	`append`	Fungsi untuk menambahkan nilai pada akhir *list*
5	`read`	Fungsi untuk membaca citra
6	`zeros`	Fungsi untuk menginisiasi *array* yang berisi nilai nol
7	`height`	Fungsi untuk mendapatkan tinggi dari *array* 2 dimensi
8	`width`	Fungsi untuk mendapatkan lebar dari *array* 2 dimensi

Tabel 3.4 Daftar fungsi yang digunakan dalam *pseudocode* ekstraksi fitur tekstur (Bagian Kedua)

No.	Nama Fungsi	Penjelasan
9	`dwt2d`	Fungsi untuk DWT 2 dimensi
10	`energy`	Fungsi untuk mengitung *energy* dari sinyal 2 dimensi
11	`write`	Fungsi untuk menulis nilai variabel pada berkas teks
12	`write_csv`	Fungsi untuk menulis nilai variabel pada berkas CSV

3.2.1 Ekstraksi Fitur Menggunakan MaZda

Perangkat lunak MaZda merupakan perangkat lunak yang biasa digunakan sebagai alat untuk analisis citra medis berdasarkan fitur tekstur. Perangkat lunak ini dapat untuk menganalisis fitur tekstur dari area bagian dari citra yang biasa disebut ROI (*Region of Interest*). ROI ini biasanya ditentukan oleh pengguna MaZda dengan memberi warna area tertentu pada citra yang akan dianalisis dan untuk tiap citra bisa didefinisikan 16 ROI.

Pada Tugas Akhir ini digunakan satu ROI yang mencakup seluruh bagian citra, sehingga hasil komputasi fitur tekstur dari MaZda merupakan fitur tekstur mencakup semua area dari satu citra. Dari proses ini kemudian akan didapatkan 282 fitur tekstur yang terdaftar pada Tabel A.1 sampai Tabel A.12. Hasil ekstraksi fitur dari tiap citra akan disimpan dalam bentuk berkas teks berekstensi par.

3.2.2 Ekstraksi Fitur *Wavelet Energy*

Untuk ekstraksi fitur *wavelet energy* dibagi menjadi 3 bagian, yaitu: DWT 1 dimensi, DWT 2 dimensi, penghitungan *energy* dari sinyal 2 dimensi dan penghitungan *wavelet energy* dari berkas citra.

3.2.2.1 DWT 1 Dimensi

Untuk melakukan DWT pada sebuah *sequence* sinyal perlu untuk memastikan bahwa panjang sinyal merupakan kelipatan dari 2. Apabila panjang sinyal bukan kelipan 2 maka perlu dilakukan *padding* pada sinyal sehingga panjang sinyal menjadi kelipan dari 2. Pada Tugas Akhir ini metode *padding* yang digunakan adalah *zero padding*, yaitu dengan menambahkan sinyal 0 pada akhir *sequence* atau mengangap bahwa sinyal 0 ditambahkan pada *sequence* sinyal. Penjelasan dalam bentuk *pseudocode* untuk melakukan DWT 1 dimensi dapat dilihat pada Gambar 3.4.

Masukan	Sequence sinyal (data)
Keluaran	Sequence sinyal aproximation (s)
	Sequence sinyal detail (d)

```
 1  inisialisasi n = length(data)/2
 2  inisialisasi zeropadding = False
 3  if length(data)%2 > 0
 4  zeropadding <- True
 5    n <- n +1
 6  end if
 7  s <- zeros(n)
 8  d <- zeros(n)
 9  for i=0 to n-1
10    if zeropadding == True and i == n-1
11      d[i] <- 0 - data[2*i]
12    else
13      d[i] = data[2*i+1] - data[2*i]
14    end if
15    s[i] <- data[2*i] + d[i]/2
16
17    d[i] <- d[i] / sqrt(2)
18    s[i] <- s[i] * sqrt(2)
19  end for
```

Gambar 3.4 *Pseudocode* untuk DWT 1 dimensi

3.2.2.2 DWT 2 Dimensi

Dengan menggunakan langkah-langkah yang telah dijelaskan pada Bab 2, *pseudocode* untuk melakukan DWT 2 dimensi dapat dilihat pada Gambar 3.5 dan Gambar 3.6.

Masukan	Matriks 2 dimensi represntasi dari citra dalam grayscale (data)
Keluaran	Aproximation dari citra(LL)
	Detail horizontal citra (HL)
	Detail vertical citra (LH)
	Detail diagonal citra (HH)

```
 1  inisialisasi L = array kosong
 2  inisialisasi H = array kosong
 3
 4  for each row in data
 5    s, d <- dwt(row)
 6    append(L, s)
 7    append(H, d)
 8  end for
 9
10  L <- transpose(L)
11  H <- transpose(H)
12
13  inisialisasi LL = array kosong
14  inisialisasi HL = array kosong
15  for each row in L
16    s, d <- dwt(row)
17    append(LL, s)
18    append(HL, d)
19  end for
20
21  inisialisasi LH = array kosong
22  inisialisasi HH = array kosong
```

Gambar 3.5 *Pseudocode* untuk melakukan DWT 2 dimensi (Bagian Pertama)

```
23  for each row in H
24    s, d <- dwt(row)
25    append(LH, s)
26    append(HH, d)
27  end for
28
29  LL <- transpose(LL)
30  HL <- transpose(HL)
31  LH <- transpose(LH)
32  HH <- transpose(HH)
```

Gambar 3.6 *Pseudocode* untuk melakukan DWT 2 dimensi (Bagian Kedua)

3.2.2.3 Penghitungan *Energy*

Penghitungan *energy* dilakukan pada sinyal matriks 2 dimensi yang berupa citra *grayscale*. Untuk penjelasan dalam bentuk *pseudocode* dapat dilihat pada Gambar 3.7.

Masukan	Matriks 2 dimensi represntasi dari citra dalam grayscale (data)
Keluaran	Keluaran Wavelet Energy (sum)

```
1  inisialisasi sum = 0
2  for i=0 to length(data)
3    inisialisasi row = data[i]
4    for j=0 to length(row)
5      sum <- sum + (data[i][j] * data[i][j]) /
   size(data)
6    end for
7  end for
```

Gambar 3.7 *Pseudocode* untuk penghitungan *energy* dari matriks 2 dimensi representasi dari citra

3.2.2.4 Penghitungan *Wavelet Energy* dari Citra

Fitur *wavelet energy* didapatkan dari penghitungan *energy* dari setiap hasil DWT 2 dimensi dari citra asli. Untuk satu level dekomposisi didapatkan 4 matriks. Dalama Tugas Akhir ini fitur *wavelet energy* dihitung dari 5 level dekomposisi, sehingga akan didapatkan 20 fitur *wavelet energy*. Untuk penjelasan dalam bentuk *pseudocode* dapat dilihat pada Gambar 3.8. Sedangkan untuk menulis fitur *wavelet energy* pada berkas *report* dijelaskan dalam bentuk *pseudocode* pada Gambar 3.9.

Masukan	Path dari citra (inp)
	Level dekomposisi pada citra (level)
Keluaran	Array Matriks 2 dimensi (wavenergy)

```
 1  inisialisasi im = read(inp,
    mode="grayscale")
 2  inisialisasi wavenergy = zeros(5,4)
 3  for i=0 to level
 4    if height(inp) < 2 or width(inp) <2
 5      break
 6    end if
 7
 8    inisialisasi LL, HL, LH, HH = dwt2d(im)
 9    wavenergy[i][0] <- energy(LL)
10    wavenergy[i][1] <- energy(HL)
11    wavenergy[i][2] <- energy(LH)
12    wavenergy[i][3] <- energy(HH)
13  end for
```

Gambar 3.8 *Pseudocode* untuk menghitung *wavelet energy* dari citra

Masukan	Direktori letak berkas citra dataset(imgdir)
	Direktori letak berkas report(reportdir)
Keluaran	

```
1  for each imgname in list_files(imgdir)
2    inisialisasi imgpath = imgdir + imgname
3    inisialisasi rppath = reportdir + imgname
4    inisialisasi wlen =
   wavedec_energy(imgpath)
5    write wlen to rppath
6  end for
```

Gambar 3.9 *Pseudocode* untuk menulis berkas *report* ang berisi fitur *wavelet energy*

3.2.3 Penggabungan Hasil Ekstraksi Fitur MaZda dengan Fitur *Wavelet Energy*

Hasil ekstraksi fitur tekstur dengan MaZda didapatkan berkas *report* berekstensi par, sedangkan hasil ekstraksi *wavelet energy* berupa berkas *report* dengan nama yang sama tanpa ekstensi par. Untuk penjelasan dalam *pseudocode* dapat dilihat pada Gambar 3.10.

Masukan	Direktori letak berkas report(reportdir)
	Berkas dataset berupa CSV (output_file)
Keluaran	

```
1  for each report in list_files(reportdir)
2    inisialisasi value_list =
   parse_report(report)
3    inisialisasi wl_rpname = hapus string .par
   dari nama report
4    append(value_list,
   parse_report(wl_rpname))
5    write_csv value_list to output_file
6  end for
```

Gambar 3.10 *Pseudocode* untuk membaca fitur tekstur dari berkas *report* kemudian menulisnya dalam berkas CSV

3.3 Perancangan Deteksi Citra *Spam*

Proses uji deteksi citra *spam* menggunakan *dataset* hasil dari proses ektraksi fitur tekstur. Kemudian dilakukan PCA pada *dataset* untuk mengurangi dimensi fitur dari *dataset*. Penggunaan PCA diharapakan dapat mempercepat proses klasifikasi citra tanpa mengurangi akurasi dari klasifikasi secara signifikan. *Dataset* hasil dari PCA ini kemudian dijadikan sebagai data masukan untuk klasifikasi; dalam Tugas Akhir ini metode klasifikasi yang digunakan adalah *Random Forest* dan SVM.

Pada subbab-subbab selanjutnya akan dijelaskan *pseudocode* untuk ekstraksi fitur tekstur dari citra *dataset*. Pada Tabel 3.5, Tabel 3.6 dan Tabel 3.7 berisikan daftar variabel yang digunakan sedangkan pada Tabel 3.8 dan Tabel 3.9 berisi fungsi-fungsi yang digunakan pada *pseudocode*.

Tabel 3.5 Daftar variabel yang digunakan dalam *pseudocode* deteksi citra *spam* (Bagian Pertama)

No.	Nama Variabel	Tipe	Penjelasan
1	`dataset`	double	*Dataset* masukan
2	`data`	double	*Dataset* tanpa kelas
3	`c`	double	Data kelas dari *sample*
4	`data_train`	double	*Dataset training*
5	`data_test`	double	*Dataset testing*
6	`c_train`	double	Data kelas untuk *training*
7	`c_test`	double	Data kelas untuk *testing*
8	`clf`		Model dari metode klasifikasi
9	`c_pred`	double	Data kelas hasil prediksi
10	`data_pca`	double	*Dataset* hasil PCA

Tabel 3.6 Daftar variabel yang digunakan dalam *pseudocode* deteksi citra *spam* (Bagian Kedua)

No.	Nama Variabel	Tipe	Penjelasan
11	`n_components`	int	Banyaknya vektor *eigen* yang diambil
12	`cover_cumul`	double	Batas nilai kumulatif maksimal dari nilai *eigen* dari vektor *eigen* yang diambil
13	`mat_transform`	double	Matriks tranformasi yang terdiri dari vektor *eigen*
14	`mat_cov`	double	Matriks *covariant* dari data
15	`eig_val`	double	Nilai *eigen*
16	`eig_vec`	double	Vektor *eigen*
17	`eig_valsum`	double	Jumlah total nilai *eigen* dari semua vektor *eigen* yang ada
18	`eig_pair`	double	*Array* berisi pasangan nilai *eigen* dan vektor *eigen*
19	`cumulative`	double	Jumlah dari nilai *eigen*
20	`n_tree`	int	Banyaknya *tree* yang dipakai sebagai *basic learner*
21	`X`	double	Data masukan fitur dari *sample*
22	`y`	double	Data kelas dari *sample*
23	`Xt`	double	Data hasil *bootstrap*
24	`yt`	double	Data kelas hasil *bootstap*
25	`max_feature`	int	Jumlah fitur maksimal yang dipertimbangkan pada saat percabangan pada *decision tree*

Tabel 3.7 Daftar variabel yang digunakan dalam *pseudocode* deteksi citra *spam* (Bagian Ketiga)

No.	Nama Variabel	Tipe	Penjelasan
26	`tree`		Model hasil *training decision tree*
27	`c1`	int	Banyaknya kelas 0 dari hasil prediksi
28	`c2`	int	Banyaknya kelas 1 dari hasil prediksi

Tabel 3.8 Daftar fungsi yang digunakan dalam *pseudocode* deteksi citra *spam* (Bagian Pertama)

No.	Nama Fungsi	Penjelasan
1	`read_csv`	Fungsi untuk membaca berkas CSV
2	`split_class`	Fungsi untuk memisahkan daftar kelas dari *dataset*
3	`split_cv`	Fungsi untuk memisahkan data *training* dan data *testing* untuk n-*fold cross validation*
4	`train_rf`	Fungsi untuk mendapatkan model hasil *training Random Forest*
5	`predict_rf`	Fungsi untuk memprediksi kelas dari *sample* menggunakan *Random Forest*
6	`get_accuracy`	Fungsi untuk menghitung akurasi klasifikasi
7	`train_svm`	Fungsi untuk mendapatkan model hasil *training* SVM
8	`predict_svm`	Fungsi untuk memprediksi kelas dari *sample* menggunakan SVM
9	`PCA`	Fungsi untuk melakukan PCA pada *dataset*
10	`covariant`	Fungsi untuk menghitung matriks *covariant* dari *dataset*

Tabel 3.9 Daftar fungsi yang digunakan dalam *pseudocode* deteksi citra *spam* (Bagian Kedua)

No.	Nama Fungsi	Penjelasan
11	`eigen`	Fungsi untuk menghitunga nilai *eigen* dan vektor *eigen* dari matriks *covariant*
12	`sum`	Fungsi untuk menghitung jumlah total nilai dari *array*
13	`pair_each_row`	Fungsi untuk memasangkan nilai dari tiap elemen dari 2 array
14	`reverse_sort`	Fungsi untuk mengurutkan *array* dari besar ke kecil
15	`length`	Fungsi untuk menghitung panjang *array*
16	`append`	Fungsi untuk menambahkan elemen pada akhir *array*
17	`transpose`	Fungsi *transpose* untuk matriks 2 dimensi
18	`dot`	Fungsi untuk menghitung nilai *dot product* dari 2 matriks

3.3.1 Program Deteksi Citra *Spam*

Proses deteksi citra *spam* pada Tugas Akhir ini terbagi menjadi dua tahap. Tahap pertama adalah tahap untuk melakukan *training* pada model menggunkan data *training*. Pada proses *training* ini akan didapatkan dua model, yaitu: model untuk PCA dan model untuk klasifikasi. Model klasifikasi yang digunakan ada dua metode klasifikasi yaitu metode klasifikasi SVM dan metode klasifikasi *random forest*. Sedangkan proses yang kedua adalah proses prediksi kelas dari dari data masukan. Pada proses ini data masukan akan diprediksi apakah citra masukan merupakan citra *spam* atau citra *ham* (bukan *spam*) menggunakan model PCA dan model klasifikasi dari hasil proses *training*. Untuk proses *training* dijelaskan lebih detail pada *pseudocode* Gambar 3.11, sedangkan untuk proses prediksi dijelaskan lebih detail pada *pseudocode* pada Gambar 3.12.

Masukan	Dataset input (data)
Keluaran	Model PCA (dataset_pca)
	Model Random Forest (clf_rf)
	Model Random SVM (clf_svm)

```
1  inisialisasi data, c = split_class(data)
2  inisialisasi clf_rf
3  inisialisasi clf_svm
4
5  pca_model <- pca_fit(data)
6  data <- pca_transform(pca_model, data)
7  clf_rf = train_rf(data, c)
8  clf_svm = train_svm(data, c)
```

Gambar 3.11 Proses *training* model PCA dan model klasifikasi

Masukan	Dataset input (data)
	Model PCA (dataset_pca)
	Model Random Forest (clf_rf)
	Model Random SVM (clf_svm)
Keluaran	Model PCA (dataset_pca)

```
1  inisialisasi data, c = split_class(data)
2
3  data <- pca_transform(pca_model, data)
4  c_pred = predict_rf(clf_rf, data)
5  print(get_accuracy(c_pred, c))
6  c_pred = predict_svm(clf_svm, data)
7  print(get_accuracy(c_pred, c))
```

Gambar 3.12 Proses prediksi kelas *dataset* menggunakan model PCA dan model klasifikasi

3.3.2 10-*Fold Cross Validation*

Pada program 10-*fold cross validation* dilakukan uji coba proses deteksi citra *spam*. Program ini akan memanggil fungsi-fungsi untuk membaca *dataset* dari berkas CSV kemudian memecah *dataset* menjadi 2 data, yaitu: *dataset* yang hanya berisi fitur tekstur dan *dataset* yang hanya berisi kelas dari tiap *sample*. Proses uji coba pada program utama ini dilakukan dalam 2 cara, yaitu: dengan tanpa melakukan PCA pada *dataset* dan dengan

melakukan PCA pada *dataset*. Jika dilakukan PCA pada *dataset* maka akan diambil vektor *eigen* sejumlah n_components. Kemudian *dataset* dijadikan masukan untuk 10-*fold cross validation*. Proses 10-*fold cross* validation ini dilakukan dengan membagi *dataset* menjadi 10 partisi kemudian menggunakan salah satu partisi sebagai data *testing* dan keseluruhan partisi yang lain sebagai data *training*. Untuk penjelasan dalam bentuk *pseudocode* dapat dilihat pada Gambar 3.13.

Masukan	
Keluaran	

```
inisialisasi dataset =
read_csv("dataset_ish_std.csv")
inisialisasi data, c = split_class(dataset)
inisialisasi data_train, data_test
inisialisasi c_train, c_test
inisialisasi clf
inisialisasi c_pred

print("Tanpa PCA")
for partition=0 to 9
  data_train, data_test <- split_cv(data,
partition)
  c_train, c_test <- split_cv(c, partition)

  clf <- train_rf(n_tree, data_train,
c_train)
  c_pred <- predict_rf(clf, data_test)
  print("Akurasi Random Forest partisi :" +
partition)
  print(get_accuracy(c_pred, c_test))

  clf = train_svm(data_train, c_train)
  c_pred = predict_svm(clf, data_test)
  print("Akurasi SVM partisi :" + partition)
  print(get_accuracy(c_pred, c_test))
end for

```

```
24  inisialisasi n_components = 10
25  inisialisasi data_pca = PCA(data,
26  n_components)
27  print("Dengan PCA")
28  for partition=0 to 9
      data_train, data_test <-
29  split_cv(data_pca, partition)
30    c_train, c_test <- split_cv(c, partition)
31
      clf <- train_rf(n_tree, data_train,
32  c_train)
33    c_pred <- predict_rf(clf, data_test)
      print("Akurasi Random Forest partisi :" +
34  partition)
35    print(get_accuracy(c_pred, c_test))
36
37    clf = train_svm(data_train, c_train)
38    c_pred = predict_svm(clf, data_test)
39    print("Akurasi SVM partisi :" + partition)
40    print(get_accuracy(c_pred, c_test))
    end for
```

Gambar 3.13 *Pseudocode* untuk 10-*fold cross validation*

3.3.3 PCA

Proses PCA pada dilakukan pada *dataset* yang tidak berisi kelas untuk tiap *sample*. Seperti yang telah dijelaskan pada Bab 2, penggunaan PCA bertujuan untuk mengurangi dimensi fitur dari *dataset*. Banyaknya fitur pada hasil PCA bergantung pada banyaknya vektor *eigen* yang diambil dalam proses PCA. Pengambilan vektor *eigen* ini bisa berdasarkan kumulatif dari nilai *eigen* atau mendefinisikan banyaknya vektor *eigen* yang diambil secara langsung. Untuk penjelasan dalam bentuk *pseudocode* dapat dilihat pada Gambar 3.14.

Masukan	Dataset input (data)
	Maksimal banyaknya eigen vektor yang diambil (n_components)
	Cakupan nilai kumulatif eigen (cover_cumul)
Keluaran	Dataset hasil PCA (dataset_pca)

```
inisialisasi data_pca

inisialisasi mat_cov = covariant(data)
inisialisasi eig_val, eig_vec =
eigen(mat_cov)
inisialisasi eig_valsum = sum(eig_val)

eig_pair <- pair_each_row(eig_val, eig_vec)

reverse_sort(eig_pair)

inisialisasi cumulative=0.0
for i=0 to length(eig_pair)-1
  cumulative <- cumulative + eig_pair[i][0]
  append(mat_transform, eig_pair[i][1])
  if i+1 >= n_components and n_components>0
    break
  else if cumulative >= cover_cumul
    break
  end if
end for

data_pca = dot(data,
transpose(mat_transform))
```

Gambar 3.14 *Pseudocode* untuk proses PCA pada *dataset*

3.3.4 Metode Klasifikasi *Random Forest*

Metode klasifikasi *Random Forest* dibagi menjadi 2 tahap, yaitu *training* dan prediksi kelas dari *sample* pada *dataset*. Proses *training* akan didapatkan sebuah model klasifikasi yang kemudian akan digunakan untuk prediksi kelas untuk tiap *sample* pada

dataset untuk *testing*. Untuk proses *training* dijelaskan dalam *pseudocode* pada Gambar 3.15, sedangkan proses prediksi ditunjukkan dalam *pseudocode* pada Gambar 3.16 dan Gambar 3.17.

Masukan	Banyaknya tree yang dipakai sebagai basic learner (n_tree)
	Dataset berisi fitur untuk tiap sample (X)
	Dataset berisi kelas untuk tiap sample (y)
Keluaran	Model klasifikasi (clf)

```
 1  inisialisasi clf
 2  inisialisasi Xt, yt
 3  inisialisasi tree
 4
 5  inisialisasi max_feature = sqrt(length(X[0])
 6
 7  for i=0 to 9
 8    Xt, yt = bootstrap(X, y)
 9    tree <- train_DT(Xt, yt, max_feature)
10    append(clf, tree)
11  end for
```

Gambar 3.15 *Pseudocode* untuk proses *training* pada *Random Forest*

Masukan	Model klasifikasi (clf)
	Dataset berisi fitur untuk tiap sample (X)
Keluaran	Data berisi kelas prediksi untuk tiap sample (y)

```
 1  inisialisasi y
 2  inisialisasi c1
 3  inisialisasi c2
 4
 5  for i=0 to length(X)-1
```

Gambar 3.16 *Pseudocode* untuk proses prediksi kelas pada *Random Forest* (Bagian Pertama)

```
  c1 <- 0
  c2 <- 0
  for each tree in clf
    if predict_DT(tree, X[i], i) == 0
      c1++
    else
      c2++
    end if
  end for
  if c1 > c2
    append(y, 0)
  else
    append(y, 1)
  end if
end for
```

Gambar 3.17 *Pseudocode* untuk proses prediksi kelas pada *Random Forest* (Bagian Kedua)

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Sebelum penjelasan implementasi akan ditunjukkan terlebih dahulu lingkungan untuk melakukan implementasi.

4.1 Lingkungan Implementasi

Lingkungan implementasi yang digunakan untuk mengimplementasikan program utama utama dan program ekstraksi fitur *wavelet energy* adalah program Anaconda 2.4.1 yang merupakan distribusi Python 2.7.11 bersama pustaka SciPy pada sistem operasi Linux Mint Debian Edition 2. Sedangkan untuk ekstraksi fitur teksur selain *wavelet energy* menggunakan perangkat lunak MaZda pada sistem operasi Windows XP Professional Service Pack 3.

4.2 Implementasi

Pada subbab ini akan dijelaskan implementasi dari subbab-subbab pada bab sebelumnya yaitu perancangan perangkat lunak. Pada subbab ini akan dijelaskan mengenai implementasi dari deteksi citra dari *email spam* berdasarkan fitur tekstur mulai dari implementasi dari proses ekstraksi fitur tekstur dari citra dan implementasi untuk klasifikasi citra *spam*.

4.2.1 Implementasi Ekstraksi Fitur Tekstur

Implementasi ekstraksi fitur tekstur dari citra dalam Tugas Akhir ini dibagi menjadi 3 bagian, yaitu: implementasi ekstraksi fitur tekstur dengan perangkat lunak MaZda, implementasi ekstraksi fitur *wavelet energy* serta implementasi untuk

penggabungan berkas laporan hasil dari 2 implementasi sebelumnya.

4.2.1.1 Implementasi Ekstraksi Fitur Menggunakan MaZda

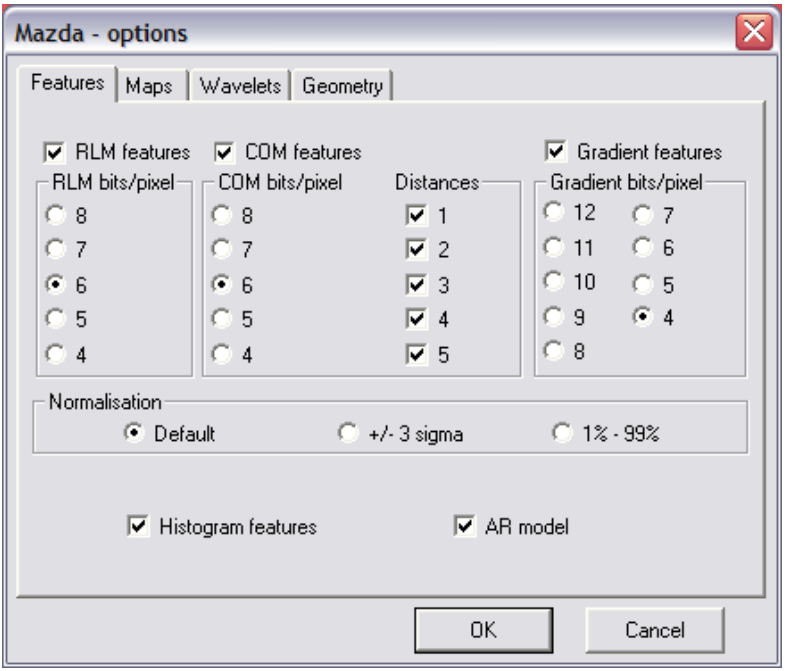
Untuk implementasi ekstraksi fitur menggunakan MaZda tidak menggunakan bahasa pemrograman tertentu akan tetapi menggunakan *macro* yang khusus untuk MaZda. Contoh *macro* yang digunakan untuk ekstraksi fitur tekstur dari citra dapat dilihat pada Kode Sumber 4.1 dan Kode Sumber 4.2. Pada *macro* baris 1 bertujuan untuk memerintahkan kepada MaZda untuk memuat pilihan untuk ekstraksi fitur tekstur pada MaZda yang terletak pada berkas options.ini. Berkas options.ini dihasilkan dari GUI MaZda yang ditampilkan pada Gambar 4.1. Pada baris 3 bertujuan untuk memindahkan tempat MaZda bekerja kemudian pada baris 4 bertujuan untuk melakukan iterasi pada berkas gambar berformat bmp. Pada baris 5 sampai 6 memerintahkan MaZda untuk memuat citra *dataset* kemudian memuat ROI. ROI ini merupakan citra dengan warna monoton untuk menandakan area pada citra *dataset* yang akan dianalisis, dalam hal ini ROI mencakup seluruh area pada citra *dataset*. Pada baris 7 sampai 8 bertujuan untuk memerintahkan pada MaZda untuk memulai proses penghitungan fitur tekstur pada citra kemudian menyimpan berkas laporan pada alamat berkas yang telah ditentukan.

```
1  LoadOptions options.ini
2
3  Chdir .\test
4  For %file *.bmp
5      LoadImage %file
6      ColorChannel Y
7      LoadROI ..\roi.bmp
```

Kode Sumber 4.1 Contoh *macro* untuk ekstraksi fitur tekstur menggunakan MaZda (Bagian Pertama)

```
    RunAnalysis
    SaveReport ..\test_report\spam_%file.par
    CloseReport
End
Chdir ..
```

Kode Sumber 4.2 Contoh *macro* untuk ekstraksi fitur tekstur menggunakan MaZda (Bagian Kedua)



Gambar 4.1 Pilihan untuk mengaktifkan penghitungan fitur tekstur pada perangkat lunak MaZda

4.2.1.1 Implementasi Ekstraksi Fitur *Wavelet Energy*

Implementasi proses ekstraksi fitur *wavelet energy* dibagai menjadi 4 bagian, yaitu: DWT 1 dimensi, DWT 2 dimensi,

penghitungan *wavelet energy* dari satu citra dan ekstraksi fitur *wavelet energy* dari citra *dataset* secara keseluruhan.

4.2.1.1.1 Implementasi DWT 1 Dimensi

Untuk implementasi DWT 1 dimensi dapat dilihat pada Kode Sumber 4.3. Pada implementasi ini jika panjang sinyal masukan tidak mencukupi untuk melakukan DWT 1 dimensi, maka akan dilakuakan *zero padding* pada citra dengan menganggap bahwa terdapat sinyal bernilai 0 pada akhir deret sinyal.

```
 1  def dwt(data):
 2      n = len(data)/2
 3      zeropadding = len(data)%2 > 0
 4      if zeropadding:
 5          n = n + 1
 6      s = np.zeros([n], dtype=float)
 7      d = np.zeros([n], dtype=float)
 8
 9      for i in range(n):
10          if zeropadding and i == n-1:
11              d[i] = 0 - data[2*i]
12          else:
13              d[i] = data[2*i+1] - data[2*i]
14          s[i] = data[2*i] + d[i]/2.
15
16          d[i] = d[i] / sqrt(2.)
17          s[i] = s[i] * sqrt(2.)
18
19      return s, d
```

Kode Sumber 4.3 Proses penghitungan DWT 1 dimensi dari citra

4.2.1.1.2 Implementasi DWT 2 Dimensi

Implementasi untuk DWT 2 dimensi dapat dilihat pada Kode Sumber 4.4 dan Kode Sumber 4.5. DWT 2 dimensi dilakukan dengan melakukan DWT 1 dimensi pada tiap baris pada sinyal 2 dimensi. Kemudian melakukan DWT 1 dimensi pada tiap kolom pada hasil DWT sebelumnya.

```
 1  def dwt2d(data):
 2      L, H = [], []
 3      for row in data:
 4          s, d = dwt(row)
 5          L.append(s)
 6          H.append(d)
 7
 8      L = np.transpose(L)
 9      H = np.transpose(H)
10
11      LL, HL = [], []
12      for row in L:
13          s, d = dwt(row)
14          LL.append(s)
15          HL.append(d)
16
17      LH, HH = [], []
18      for row in H:
19          s, d = dwt(row)
20          LH.append(s)
21          HH.append(d)
22
```

Kode Sumber 4.4 Proses penghitungan DWT 2 dimensi (Bagian Pertama)

```
23      LL = np.transpose(LL) # approximation
24      HL = np.transpose(HL) # horizontal
25      LH = np.transpose(LH) # vertical
26      HH = np.transpose(HH) # diagonal
27
28      return LL, HL, LH, HH
```

Kode Sumber 4.5 Proses penghitungan DWT 2 dimensi (Bagian Kedua)

4.2.1.1.3 Implementasi Penghitungan *Wavelet Energy*

Implementasi untuk penghitungan *wavelet energy* pada citra dibagi menjadi 2 bagian, yaitu: implementasi penghitungan *energy* dari sinyal 2 dimensi dan implementasi penghitungan *energy* pada hasil DWT 2 dimensi. Implementasi penghitungan *energy* pada sinyal 2 dimensi dapat dilihat pada Kode Sumber 4.6. Sedangkan implementasi dari penghitungan *energy* pada hasil DWT 2 dimensi dapat dilihat pada Kode Sumber 4.7.

```
1   def energy(data):
2       sum_ = 0.
3       for i in range(len(data)):
4           row = data[i]
5           for j in range(len(row)):
6               sum_ = sum_ + (data[i][j] *
    data[i][j]) / data.size
7       return sum_
```

Kode Sumber 4.6 Proses penghitungan *energy* dari sinyal 2 dimensi

```
def wavdec_energy(inp, level=5):
    if type(inp) == np.ndarray:
        im = inp
    else:
        im = ndimage.imread(inp, mode="L")
    wavenergy = np.zeros([5,4])
    for i in range(level):
        if im.shape[0] < 2 and im.shape[1] < 2:
            break;
        LL, HL, LH, HH = dwt2d(im)
        wavenergy[i][0] = energy(LL)
        wavenergy[i][1] = energy(HL)
        wavenergy[i][2] = energy(LH)
        wavenergy[i][3] = energy(HH)
        im = LL
    return wavenergy
```

Kode Sumber 4.7 Proses penghitungan *energy* pada hasil DWT 2 dimensi

4.2.1.1.4 Implementasi Ekstraksi Fitur *Wavelet Energy* dari citra *dataset*

Implementasi untuk ekstraksi fitur *wavelet energy* dari citra *dataset* terbagi menjadi beberapa bagian, yaitu: bagian pertama berupa fungsi yang memanggil fungsi penghitungan *wavelet energy* kemudian menulisnya pada berkas *report* dan bagian kedua berupa fungsi utama yang menerima masukan dari pengguna kemudian melakukan *directory traversal* pada direktori tempat citra *dataset*. Untuk implementasi bagian pertama dapat dilihat pada Kode Sumber 4.8. Sedangkan implementasi untuk bagian kedua dapat dilihat pada Kode Sumber 4.9 dan Kode Sumber 4.10. Pada *directory traversal* pada Kode Sumber 4.10

baris 28, tiap berkas yang ditemukan akan dipanggil fungsi visit yang didefinisikan pada Kode Sumber 4.8.

```
 1  def wl_comp_and_save(imgpath, rppath):
 2      print "INFO: computing", imgpath
 3      wlen = wl.wavdec_energy(imgpath)
 4      fd = open(rppath, "a")
 5      for row in wlen:
 6          for i in row:
 7              fd.write(str(i) +"\n")
 8      fd.close()
 9
10  def visit(dummy, dirname, files):
11      if (len(files)) > 0:
12          for i in files:
13              # report file path
14              rppath = os.path.join(reportdir,
    cattype + "_" + i)
15              # image file path
16              imgpath = os.path.join(dirname,
    i)
17              wl_comp_and_save(imgpath,
    rppath)
```

Kode Sumber 4.8 Proses penghitungan *wavelet energy* kemudian menulisanya pada berkas *report*

```
1  def print_usage_then_exit():
2      print "Cara penggunaan:"
3      print sys.argv[0], "<type :spam/ham>
   <image folder> <report folder>"
4      sys.exit(1)
5
6  def main(argv):
7      global cattype
8      global imgdir
9      global reportdir
10
11     if len(argv) != 3 :
12         print_usage_then_exit()
13
14     cattype = argv[0]
15     imgdir = argv[1]
16     reportdir = argv[2]
17
18     if (not os.path.isdir(imgdir)) or (not
   os.path.isdir(reportdir)):
19         print "Error: Dir is not valid"
20         print_usage_then_exit()
21
22     try:
23         reponse = raw_input("Tekan [Enter]
   Untuk Mulai. Jika tidak, tekan [Ctrl]-[C] ")
24     except KeyboardInterrupt:
25         sys.exit(1)
```

Kode Sumber 4.9 Proses menerima masukan dari pengguna kemudian melakukan *directory traversal* pada direktori tempat citra *dataset* (Bagian Pertama)

```
26
27      print "\nProgram akan memproses file
    report secara rekursif"
28      os.path.walk(imgdir,    visit, "")
29
30  if __name__ == "__main__":
31      main(sys.argv[1:])
```

Kode Sumber 4.10 Proses menerima masukan dari pengguna kemudian melakukan *directory traversal* pada direktori tempat citra *dataset* (Bagian Kedua)

4.2.1.2 Penggabungan Hasil Ekstraksi Fitur MaZda dengan Fitur *Wavelet Energy*

Proses penggabungan hasil ekstraksi dari perangkat lunak MaZda dengan hasil ekstraksi fitur *wavelet energy* akan dihasilkan berkas CSV yang kemudian akan dijadikan masukan untuk proses klasifikasi citra *spam*. Implementasi untuk proses ini terbagi menjadi 3 bagian, yaitu: bagian pertaman untuk menulis baris pada berkas CSV, bagian kedua untuk membaca berkas *report* kemudian menulis hasil bacaan ke berkas CSV dan bagian ketiga untuk menerima masukan dari pengguna kemudian melakukan *directory traversal* pada direktori tempat berkas *report*. Implementasi untuk bagian pertama dapat dilihat pada Kode Sumber 4.11. Implementasi untuk bagian kedua dapat dilihat pada Kode Sumber 4.12 dan Kode Sumber 4.13. Implementasi untuk bagian ketiga dapat dilihat pada Kode Sumber 4.14 sampai Kode Sumber 4.17. Program untuk bagian ketiga akan menerima masukan dari pengguna kemudian akan membaca berkas feature_list_282.txt yang berisi daftar hasil fitur tekstur dari MaZda yang akan dimasukkan pada berkas CSV.

```
1  ## mode w: overwrite
2  ## mode a: append
3  def write_csv_line(data, mode, last_col):
4      feature_total_write = feature_total + 20
5      print "INFO:Menulis CSV header di",
   output_file
6      line = ""
7      cnt = 0
8      for i in data:
9          line += str(i).replace(",","|")
10         cnt += 1
11         line += ","
12         if cnt == feature_total_write:
13             break
14     line += last_col
15     line += "\n"
16     fd = open(output_file, mode)
17     fd.write(line)
18     fd.close()
19     print "INFO:Tertulis", cnt, "atribut"
20     if cnt != feature_total_write:
21         print "ERR :Jumlah fitur yang
   terbaca tidak konsisten. Seharusnya
   ada",feature_total,\
22             "fitur, tetapi
   terbaca:",cnt,"fitur"
23         sys.exit(1)
```

Kode Sumber 4.11 Proses untuk menulis baris pada berkas CSV

```
 1  def parse_report(report):
 2      fd = open(report, "r")
 3      begin_parse = False
 4      cnt = 0
 5      value_list = []
 6      for i in fd:
 7          if begin_parse and len(i)>0:
 8              val = i.split("\t", 2)
 9              if feature_list[cnt] == val[0]:
10                  value_list.append(val[1])
11                  cnt += 1
12              else:
13                  print "ERR :Urutan atribut
    tidak konsisten"
14                  sys.exit(1)
15          elif i.find("-- FEATURES --") != -1:
16              begin_parse = True
17
18          if cnt == feature_total:
19              break
20      spam_class = "0"
21      if
    str(os.path.basename(report)).find("spam_")
    == 0:
22          spam_class = "1"
23      wl_rpname =
    str(report).replace(".par","")
24      with open(wl_rpname) as fd_wl:
```

Kode Sumber 4.12 Proses untuk membaca berkas *report* kemudian menulis hasil bacaan ke berkas CSV (Bagian Pertama)

```
25          for i in fd_wl:
26              i = i.rstrip("\n")
27              value_list.append(i)
28      write_csv_line(value_list, "a",
    spam_class)
29
30  def visit(dummy, dirname, files):
31      if (len(files)) > 0:
32          for i in files:
33              if str(i).endswith(".par"):
34                  print "Parsing:",
    os.path.join(dirname,i)
35
    parse_report(os.path.join(dirname,i))
```

Kode Sumber 4.13 Proses untuk membaca berkas *report* kemudian menulis hasil bacaan ke berkas CSV (Bagian Kedua)

```
 1  def print_usage_then_exit():
 2      print "Cara penggunaan:"
 3      print sys.argv[0], "<report folder>
    <output.csv>"
 4      sys.exit(1)
 5
 6  def main(argv):
 7      global output_file
 8      global feature_list
 9      global feature_total
10
11      if len(argv) != 2:
```

Kode Sumber 4.14 Proses untuk menerima masukan dari pengguna kemudian melakukan *directory traversal* pada direktori tempat berkas *report* (Bagian Pertama)

```
12          print "Parameter kurang"
13          print_usage_then_exit()
14      report_folder = argv[0]
15      output_file = argv[1]
16
17      ## cek folder input dan file output
18      if os.path.isdir(report_folder) ==
False:
19          print "ERR :Folder report tidak
valid"
20          print_usage_then_exit()
21      if os.path.isfile(output_file):
22          print "WARN:Sudah ada file yang yang
bernama", output_file
23          try:
24              reponse = raw_input("Tekan
[Enter] jika file akan ditimpa. Jika tidak,
tekan [Ctrl]-[C] ")
25          except KeyboardInterrupt:
26              sys.exit(1)
27
28      ## cek dan baca file feature_list.txt
29      ftlist_file = "feature_list_282.txt"
30      if
os.path.isfile(os.path.join(os.path.curdir,
ftlist_file)):
31          ## baca file kemudian masukkan dict
feature_list
32          fd = open(ftlist_file, "r")
33          feature_list = []
```

Kode Sumber 4.15 Proses untuk menerima masukan dari pengguna kemudian melakukan *directory traversal* pada direktori tempat berkas *report* (Bagian Kedua)

```
34          for val in fd:
35              feature_list.append(val.strip())
36          feature_total = len(feature_list)
37
38          # tambah fitur wavelet energy
39          feature_list_wlen = []
40          for i in range(1,6):
41
    feature_list_wlen.append("WlEnLL" + str(i))
42
    feature_list_wlen.append("WlEnHL" + str(i))
43
    feature_list_wlen.append("WlEnLH" + str(i))
44
    feature_list_wlen.append("WlEnHH" + str(i))
45
46          feature_list_final = feature_list +
    feature_list_wlen
47
48          write_csv_line( feature_list_final ,
    "w", "SPAM")
49      else:
50          print "ERR :File feature_list.txt
    tidak ditemukan!!!"
51          sys.exit(1)
52
53      print "\nProgram akan memproses file
    report secara rekursif (report harus
    berekstensi *.par)"
```

Kode Sumber 4.16 Proses untuk menerima masukan dari pengguna kemudian melakukan *directory traversal* pada direktori tempat berkas *report* (Bagian Ketiga)

```
54      os.path.walk(report_folder,    visit,
        "")
55
56
57  if __name__ == "__main__":
58      main(sys.argv[1:])
```

Kode Sumber 4.17 Proses untuk menerima masukan dari pengguna kemudian melakukan *directory traversal* pada direktori tempat berkas *report* (Bagian Keempat)

4.2.2 Implementasi Proses Deteksi Citra *Spam*

Prose deteksi citra *spam* dilakukan dengan menerima masukan berupa berkas CSV berisi hasil ekstraks fitur tekstur dari subbab sebelumnya. Implementasi proses deteksi citra *spam* ini terbagi menjadi 3 bagian utama, yaitu: implementasi proses PCA, implementasi metode klasifikasi *random forest* dan implementasi program utama yang akan melakukan uji coba deteksi citra *spam* menggunakan metode *random forest* dan SVM baik itu dengan melakukan PCA pada *dataset* maupun tidak.

4.2.2.1 Implementasi PCA

Implementasi PCA dapat dilihat pada Kode Sumber 4.18. Implementasi ini menerima satu parameter wajib dan 2 parameter opsional. Parameter data merupakan parameter wajib yang harus ada ketika memanggil fungsi PCA. Variabel data ini berisi *dataset* yang akan diproses menggunakan PCA. Sedangkan parameter opsional yang didefinisikan adalah cover_cumul yang merupakan nilai maksimal dari kumulatif nilai *eigen* dari *eigen* vektor yang akan diambil serta variabel n_components yang merupakan banyaknya vektor *eigen* yang akan diambil.. Jika n_components berisi nilai *integer* maka parameter ini lebih diutamakan dibandingkan variable cover_cumul.

```
1  import numpy as np
2  import pandas as pd
3
4  def pca(data, cover_cumul=0.95,
   n_components=None):
5      mat_cov = data.cov()
6      eig_val, eig_vec =
   np.linalg.eigh(mat_cov)
7      eig_valsum = np.sum(eig_val)
8
9      eig_pair = [
   (np.abs(eig_val[i])/eig_valsum,
   eig_vec[:,i]) for i in range(len(eig_val)) ]
10     eig_pair.sort(reverse=True)
11
12     cumulative = 0.0
13     mat_transform = []
14     for i in range(len(eig_pair)):
15         cumulative = cumulative +
   eig_pair[i][0]
16         mat_transform.append(eig_pair[i][1])
17         if not n_components is None and
   n_components == len(mat_transform):
18             break
19         elif cumulative > cover_cumul and
   n_components is None:
20             break
21     mat_transform = np.array(mat_transform)
22     data_pca = data.dot(mat_transform.T)
23
24     return data_pca
```

Kode Sumber 4.18 Implementasi fungsi PCA

4.2.2.2 Implementasi *Random Forest*

Implementasi metode klasifikasi *random forest* didefinisikan sebagai kelas yang di dalamnya berisi *method* sebagai *constructor* untuk model dari *random forest*, *method* fit untuk *training* model dari *random forest* serta *method* predict untuk memprediksi kelas dari *sample* yang dijadikan masukan. *Constructor* menerima 2 parameter, yaitu n_tree untuk mendefinisikan banyaknya *tree* sebagai *basic leaner*, serta max_feature untuk mendefinisikan maksimal banyaknya fitur dipilih secara acak yang akan dipertimbangkan ketika melakukan percabangan pada *tree*. Pada implementasi ini *decision tree* sebagai *basic learner* untuk *random forest* menggunakan implementasi dari pustaka Scikit-learn. Implementasi dari *random forest* ini dapat dilihat pada Kode Sumber 4.19 dan Kode Sumber 4.20.

```
1  from sklearn.tree import
   DecisionTreeClassifier
2  import numpy as np
3  from scipy import stats
4
5  class RandomForest:
6      def __init__(self, n_tree=10,
   max_feature=None):
7          self.n_tree = n_tree
8          self.max_feature = max_feature
9          self.n_feature = 0
10         self.tree_list = []
11
12     def fit(self, X, y):
13         self.n_feature = X.shape[1]
14         if self.max_feature == None:
```

Kode Sumber 4.19 Implementasi metode klasifikasi *random forest* (Bagian Pertama)

```
15              self.max_feature =
    np.int(np.sqrt(self.n_feature))
16
17          rnd = np.random.RandomState()
18          idx = np.arange(X.shape[0])
19          # ambil 2/3 sampling training
20          n_train = X.shape[0]*2/3
21
22          for _ in range(self.n_tree):
23              # bootstraping
24              choice = np.random.choice(idx,
    size=n_train, replace=False)
25              Xt = X.take(choice)
26              yt = y.take(choice)
27              tree =
    DecisionTreeClassifier(max_features=self.max
    _feature, random_state=rnd)
28
    self.tree_list.append(tree.fit(Xt,yt))
29
30      def predict(self, X):
31          res = []
32          for tree in self.tree_list:
33              res.append(tree.predict(X))
34          res = np.array(res)
35
36          # ambil vote
37          cls, cnt = stats.mode(res)
38
39          return cls.reshape((cls.shape[1],))
```

Kode Sumber 4.20 Implementasi metode klasifikasi *random forest* (Bagian Kedua)

4.2.2.3 Implementasi *Confusion Matrix*

Implementasi dari *confusion matrix* ini bertujuan untuk mempermudah dalam penghitungan kinerja klasifikasi dengan menghitung *confusion matrix* serta akurasi. Implementasi ini dapat dilihat pada Kode Sumber 4.21 sampai dengan Kode Sumber 4.23.

```
# orig : array kelas asli
# pred : array kelas hasil klasifikasi
# c_num : array representasi klass [positif, negatif]
class CM:
    def __init__(self, orig, pred, c_num):
        assert len(orig) == len(pred), "Panjang array tidak sama"
        self.orig = orig
        self.pred = pred
        self.c_num = c_num
        self.TP = .0
        self.TN = .0
        self.FP = .0
        self.FN = .0

        for i in range(len(orig)):
            if orig[i] == pred[i]:
                # True
                if orig[i] == c_num[0]:
                    # Positif
                    self.TP += 1.
```

Kode Sumber 4.21 Implementasi *confusion matrix* (Bagian Pertama)

```
21                  elif orig[i] == c_num[1]:
22                      # Negatif
23                      self.TN += 1.
24              else:
25                  # False
26                  if orig[i] == c_num[0]:
27                      # Positif
28                      self.FP += 1.
29                  elif orig[i] == c_num[1]:
30                      # Negatif
31                      self.FN += 1.
32
33          assert (self.TP + self.TN + self.FP
    + self.FN) == len(orig), \
34              "Ada yang salah pada perhitungan
    Confusion Matrix"
35
36      def accuracy(self):
37          return
    (self.TP+self.TN)/(len(self.orig))
38
39      def precision(self):
40          return (self.TP)/(self.TP+self.FP)
41
42      def recall(self):
43          return (self.TP)/(self.TP+self.FN)
44
45      def f_measure(self):
```

Kode Sumber 4.22 Implementasi *confusion matrix* (Bagian Kedua)

```
46          return (2. * self.precision() *
    self.recall()) / (self.precision() + \
47              self.recall())
48
49      def __str__(self):
50          return "TP = %.2f\nTN = %.2f\nFP =
    %.2f\nFN = %.2f" % (
51              self.TP, self.TN, self.FP,
    self.FN)
```

Kode Sumber 4.23 Implementasi *confusion matrix* (Bagian Ketiga)

4.2.2.4 Impementasi 10-*Fold Cross Validation*

Implementasi program utama terdiri dari 4 proses utama, yaitu: proses persiapan *dataset*, proses klasifikasi menggunakan *random forest*, proses klasifikasi menggunakan SVM dan proses 10-*fold cross validation*. Secara berurutan proses-proses tersebut dapat dilihat pada Kode Sumber 4.24 sampai dengan Kode Sumber 4.30. Untuk proses klasifikasi dengan metode SVM, digunakan implementasi dari pustaka Scikit-learn dengan melakukan standarisasi pada fitur *dataset* untuk memaksimalkan kinerja SVM pada proses klasifikasi. Proses standarisasi fitur *dataset* ini dilakukan dengan menggunakan pustaka Scikit-learn, hasil dari proses standarisasi fitur *dataset* akan menyebabkan rata-rata fitur antar *sample* menjadi nol dan standar deviasi fitur antar *sample* menjadi satu.

```
####
# Baca dataset_ish.csv dengan `pandas`
def read_dataset():
    global dataset

    dataset = pd.read_csv("dataset_ish_std.csv")

def shuffle_dataset():
    global dataset

    dataset = dataset.reindex(index=np.random.permutation(dataset.index))

###
# Pisahkan atribut dengan kelas
def do_extract_class_list():
    global data
    global c

    data = dataset[dataset.columns[:302]]
    c = dataset['SPAM']

###
# PCA
def do_pca():
    global dataset_train, dataset_test
```

Kode Sumber 4.24 Implementasi proses persiapan *dataset* (Bagian Pertama)

```
26      pca_model = PCA.fit(dataset_train,
    n_components=10)
27      dataset_train = PCA.transform(pca_model,
    dataset_train)
28      dataset_test = PCA.transform(pca_model,
    dataset_test)
29
30  ###
31  # Pemisahan dataset untuk cross validation
32  def do_split_dataset_select(fold=3, part=2):
33      global dataset_train
34      global c_train
35      global dataset_test
36      global c_test
37
38      i_st = part*data.shape[0]/fold
39      i_ed = (part+1)*data.shape[0]/fold
40      dataset_train =
    data[:i_st].append(data[i_ed:])
41      c_train = c[:i_st].append(c[i_ed:])
42      dataset_test = data[i_st:i_ed]
43      c_test = c[i_st:i_ed]
```

Kode Sumber 4.25 Implementasi proses persiapan *dataset* (Bagian Kedua)

```
def do_rf():
    clf = RF.RandomForest(n_tree=20)
    clf.fit(dataset_train, c_train)
    res = clf.predict(dataset_test)

    cm = util.CM(c_test.values, res, [1, 0])

    print "\nRandom Forest"
    print cm
    print "Akurasi %.4f%%" %
(cm.accuracy()*100.)
```

Kode Sumber 4.26 Proses klasifikasi menggunakan *random forest*

```
def do_svm():
    global clf

    ss =
prep.StandardScaler().fit(dataset_train)
    dataset_train_ =
ss.transform(dataset_train)
    dataset_test_ =
ss.transform(dataset_test)

    clf = svm.SVC(kernel="rbf",
cache_size=500)
    clf.fit(dataset_train_, c_train)
    res = clf.predict(dataset_test_)

    cm = util.CM(c_test.values, res, [1, 0])
```

Kode Sumber 4.27 Proses klasifikasi menggunakan SVM (Bagian Pertama)

```
13
14      print "\nSVM"
15      print cm
16      print "Akurasi %.4f%%" %
(cm.accuracy()*100.)
```

Kode Sumber 4.28 Proses klasifikasi menggunakan SVM (Bagian Kedua)

```
1   def cross_validate():
2       global cm_list_rf, cm_list_svm
3
4       fold = 10
5       read_dataset()
6       shuffle_dataset()
7
8       ## Cross validation tanpa PCA
9       print "\n== Cross validation tanpa PCA
==="
10      do_extract_class_list()
11      for i in xrange(fold):
12          print "\nITERASI =",i
13          do_split_dataset_select(fold=fold,
part=i)
14          # Klasifikasi
15          do_rf()
16          do_svm()
17
18      ## Cross validation dengan PCA
19      print "\n== Cross validation dengan PCA
==="
```

Kode Sumber 4.29 Proses 10-*fold cross validation* (Bagian Pertama)

```
20      do_extract_class_list()
21      for i in xrange(fold):
22          print "\nITERASI =",i
23          do_split_dataset_select(fold=fold,
part=i)
24          # PCA
25          do_pca()
26          # Klasifikasi
27          do_rf()
28          do_svm()
29
30  cross_validate()
```

Kode Sumber 4.30 Proses 10-*fold cross validation* (Bagian Kedua)

(Halaman ini sengaja dikosongkan)

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan uji coba yang dilakukan pada aplikasi yang telah dikerjakan serta analisis dari uji coba yang telah dilakukan. Pembahasan pengujian meliputi lingkungan uji coba, skenario uji coba yang meliputi uji kinerja klasifikasi tanpa melakukan PCA pada *dataset* dan uji kinerja klasfikasi dengan melakukan PCA pada *dataset* serta analisis setiap pengujian.

5.1 Lingkungan Uji Coba

Lingkungan uji coba menjelaskan lingkungan yang digunakan untuk menguji implementasi deteksi citra *spam* pada Tugas Akhir. Lingkungan Uji Coba meliputi perangkat keras dan perangkat lunak yang dijelaskan sebagai berikut:

1. Perangkat keras
   a. Prosesor : Intel® Core™i3 CPU @ 2.10GHz x 2
   b. Memori : 5.8GB
   c. Tipe system : 64 bit
2. Perangkat lunak
   a. Sistem operasi: Linux Mint Debian Edition 2
   b. Perangkat pengembang : Anaconda 2.4.1 dan Python 2.7.11

5.2 Data Uji Coba

Uji coba kinerja deteksi citra *spam* dilakukan menggunakan citra *dataset* dari *Image Spam Hunter* (ISH) [11] dengan citra yang dipakai sebanyak 1620 citra yang terdiri dari 810 citra *spam* dan 810 citra *ham*, serta *dataset* citra baru sebanyak 316 citra yang terdiri dari 158 citra *spam* dan 158 citra *ham*. Pada *dataset* dilakukan ekstraksi fitur tesktur dengan hasil berkas CSV yang kemudian menjadi data masukan untuk proses klasifikasi citra *spam*. Fitur hasil ekstraksi berjumlah total 302 fitur yang terdiri dari 282 fitur dari MaZda dan 20 fitur *wavelet energy*. Contoh

citra yang diambil dari *dataset* dapat dilihat pada Gambar 5.1 dan Gambar 5.2. Sedangkan contoh hasil esktraksi fitur dari 10 citra yang terdiri dari 5 citra *spam* dan 5 citra *ham* dapat dilihat pada Tabel 5.1. Pada tabel tersebut 5 kolom pertama merupakan contoh fitur hasil ekstraksi dan kolom terakhir merupakan kelas dari citra. Jika kelas bernilai 0 maka citra merupakan bukan *spam* sedangkan jika kelas bernilai 1 maka citra merupakan *spam*.

Gambar 5.1 Berkas 0uS3tts9xP.bmp sebagai contoh citra *spam* dari *dataset*

Gambar 5.2 Berkas zzz_10963_03507d6116_m.bmp sebagai contoh citra *ham* dari *dataset*

Tabel 5.1 Contoh fitur hasil ekstraksi fitur tekstur dari *dataset*

Mean	Perc.99%	GrMean	Teta1	WlEnLL1	SPAM
112.35	251.00	3.16	0.41	150149.85	0
92.00	256.00	1.65	0.83	105957.15	0
169.19	244.00	4.92	0.35	226644.31	0
119.58	224.00	1.54	0.77	161277.21	0
130.49	255.00	1.08	0.73	146041.49	0
233.00	255.00	1.67	0.62	263973.38	1
241.29	255.00	0.99	0.58	267267.45	1
142.78	256.00	1.71	0.88	165216.48	1
23.07	256.00	2.93	0.22	35681.77	1
173.08	230.00	2.46	0.02	226535.23	1

Pada Tabel 5.1, *Mean* merupakan salah satu fitur *histogram*, *Perc.99%* merupakan fitur *percentile* 99%, *GrMean* merupakan salah satu fitur *gradient*, Teta1 merupakan salah satu fitur dari *autoregressive model*, *WlEnLL1* merupakan fitur *wavelet energy* dari citra *approximation* dari hasil *Discrete Wavelet Transform* (DWT) 2 dimensi.

5.3 Skenario Uji Coba

Parameter yang digunakan untuk metode-metode tersebut dijelaskan sebagai berikut:

1. PCA
   - n_components : 10
2. *Random forest*
   - n_tree : 20
   - max_feature : $\sqrt{302}$
3. SVM
   - kernel: *radial basis function* (RBF)
   - C : 1.0
   - gamma : 1/302

Sedangkan skenario uji coba yang digunakan adalah sebagai berikut:

1. Uji coba *random forest* dan SVM tanpa melakukan PCA pada *dataset*.
2. Uji coba *random forest* dan SVM dengan melakukan PCA pada *dataset*.
3. Uji coba *random forest* dan SVM dengan melakukan PCA pada dataset baru.

Uji coba skenario 1 dan 2 dilakukan dengan menggunakan 10-*fold cross validation* pada metode klasfikasi *random forest* dan SVM dengan melakukan PCA pada *dataset* maupun tidak.

5.4 Hasil Uji Coba

Hasil uji coba berupa daftar nilai akurasi dari hasil klasifikasi dari 10 partisi dalam 10-*fold cross validation*.

5.4.1 Uji Coba Tanpa Melakukan PCA Pada *Dataset*

1. Hasil uji coba metode klasifikasi *random forest* dan SVM tanpa melakukan PCA pada *dataset* terlebih dahulu.

Sesuai pada Tabel 5.2 dan Tabel 5.3 kinerja akurasi dan *precison* dan *recall random forest* lebih tinggi atau sama jika dibandingkan dengan SVM pada sebagian besar partisi. Sedangkan jika dilihat dari Tabel 5.4, rata-rata kinerja akurasi, *precision* dan *recall* dari *random forest* selalu lebih tinggi dibanding dengan SVM.

Tabel 5.2 Hasil kinerja pada tiap partisi untuk *random forest* dan SVM tanpa PCA (Bagian Pertama)

Partisi	Akurasi (%)		Precision (%)		Recall (%)	
	RF	SVM	RF	SVM	RF	SVM
1	98.77	96.30	100.00	96.05	97.44	96.05
2	96.30	97.53	96.34	100.00	96.34	95.35
3	99.38	98.77	100.00	98.84	98.85	98.84
4	99.38	98.15	100.00	98.67	98.68	97.37

Tabel 5.3 Hasil kinerja pada tiap partisi untuk *random forest* dan SVM tanpa PCA (Bagian Kedua)

Partisi	Akurasi (%)		Precision (%)		Recall (%)	
	RF	SVM	RF	SVM	RF	SVM
5	98.77	98.15	98.63	97.26	98.63	98.61
6	99.38	98.77	100.00	98.82	98.82	98.82
7	98.77	97.53	98.72	100.00	98.72	95.12
8	98.77	97.53	97.56	96.34	100.00	98.75
9	98.77	98.15	100.00	98.75	97.56	97.53
10	98.15	97.53	98.92	97.85	97.87	97.85

Tabel 5.4 Statistik kinerja pada *cross validation* untuk *random forest* dan SVM tanpa PCA

Statistik	Akurasi (%)		Precision (%)		Recall (%)	
	RF	SVM	RF	SVM	RF	SVM
Minimum	96.30	96.30	96.34	96.05	96.34	95.12
Maksimum	99.38	98.77	100.00	100.00	100.00	98.84
Rata-rata	98.64	97.84	99.02	98.26	98.29	97.43

2. Hasil uji coba waktu komputasi metode klasifikasi *random forest* dan SVM tanpa melakukan PCA pada *dataset* terlebih dahulu.

Jika dibandingkan berdasarkan waktu komputasi SVM jauh lebih unggul dibandingkan dengan *random forest*. Hal ini dapat dibuktikan pada Tabel 5.5 dan Tabel 5.6. Pada tabel tersebut waktu SVM selalu lebih cepat dibandingkan *random forest* dan rata-rata SVM juga lebih rendah.

Tabel 5.5 Hasil waktu komputasi pada tiap partisi untuk *random forest* dan SVM tanpa PCA

Partisi	Waktu komputasi (milidetik)	
	RF	**SVM**
1	325.2	169.3
2	295.6	176.1
3	309.3	170.4
4	299.8	164.5
5	302.5	167.9
6	319.2	173.3
7	296.5	167.9
8	306.5	166.3
9	320.0	166.3
10	311.0	166.1

Tabel 5.6 Statistik waktu komputasi pada *cross validation* untuk *random forest* dan SVM tanpa PCA

Statistik	Waktu komputasi (milidetik)	
	RF	**SVM**
Minimum	295.6	164.5
Maksimum	325.2	176.1
Rata-rata	308.6	168.8

5.4.2 Uji Coba Dengan Melakukan PCA Pada *Dataset*

1. Hasil uji coba metode klasifikasi *random forest* dan SVM dengan melakukan PCA pada *dataset* terlebih dahulu.

Dari Tabel 5.7 dan Tabel 5.8 terlihat bahwa untuk akurasi dan *recall* dari *random forest* mempunyai rata-rata lebih tinggi dibandingkan SVM. Sedangkan untuk *precision* SVM berata-rata lebih tinggi dibandingkan dengan *random forest.*

Tabel 5.7 Hasil kinerja pada tiap partisi untuk *random forest* dan SVM dengan PCA

Partisi	Akurasi (%)		Precision (%)		Recall (%)	
	RF	SVM	RF	SVM	RF	SVM
1	97.53	96.30	100.00	100.00	95.00	92.68
2	95.68	95.06	97.56	96.34	94.12	94.05
3	97.53	98.77	97.67	98.84	97.67	98.84
4	97.53	97.53	97.33	97.33	97.33	97.33
5	96.91	98.15	97.26	97.26	95.95	98.61
6	98.15	98.15	96.47	97.65	100.00	98.81
7	96.30	96.30	97.44	98.72	95.00	93.90
8	97.53	98.77	96.34	98.78	98.75	98.78
9	99.38	98.15	100.00	98.75	98.77	97.53
10	96.30	95.06	96.77	96.77	96.77	94.74

Tabel 5.8 Statistik kinerja pada *cross validation* untuk *random forest* dan SVM dengan PCA

Statistik	Akurasi (%)		Precision (%)		Recall (%)	
	RF	SVM	RF	SVM	RF	SVM
Minimum	95.68	95.06	96.34	96.34	94.12	92.68
Maksimum	99.38	98.77	100.00	100.00	100.00	98.84
Rata-rata	97.28	97.22	97.69	98.04	96.94	96.53

2. Hasil uji coba waktu komputasi metode klasifikasi *random forest* dan SVM dengan melakukan PCA pada *dataset* terlebih dahulu.

Dari Tabel 5.9 dan Tabel 5.10, SVM selalu lebih unggul dibandingkan dengan *random forest* jika dilihat berdasarkan waktu komputasi.

Tabel 5.9 Hasil kinerja pada tiap partisi untuk *random forest* dan SVM dengan PCA

Partisi	Waktu komputasi (milidetik)	
	RF	**SVM**
1	149.3	48.5
2	151.1	48.2
3	148.7	49.0
4	150.3	46.0
5	145.2	48.4
6	147.8	47.3
7	149.3	47.9
8	145.0	48.1
9	150.3	46.5
10	151.5	49.1

Tabel 5.10 Statistik waktu komputasi pada *cross validation* untuk *random forest* dan SVM dengan PCA

Statistik	Waktu komputasi (milidetik)	
	RF	**SVM**
Minimum	145.0	46.0
Maksimum	151.5	49.1
Rata-rata	148.9	47.9

5.4.3 Hasil uji coba *random forest* dan SVM dengan melakukan PCA pada dataset baru.

Jika dilakukan uji coba pada *dataset* baru, akurasi, *precision* dan *recall* dari SVM lebih tinggi dibandingkan dengan *random forest*.

Tabel 5.11 Hasil kinerja *random forest* dan SVM dengan PCA pada data baru

Akurasi (%)		Precision (%)		Recall (%)	
RF	**SVM**	**RF**	**SVM**	**RF**	**SVM**
72.47	81.01	67.09	84.81	75.18	78.82

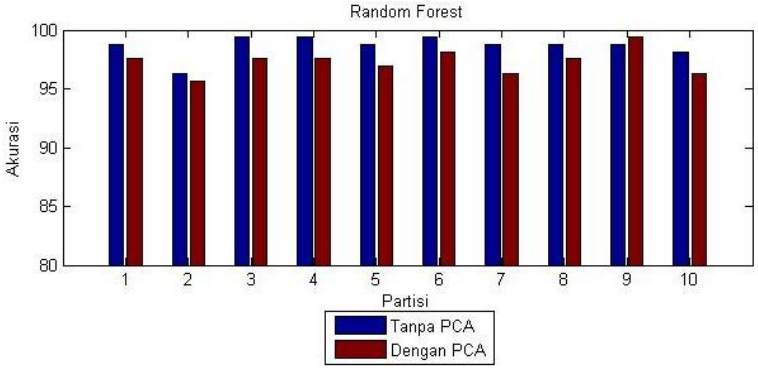
5.5 Evaluasi

Pada subbab ini akan dijelaskan perbandingan kinerja antar metode klasifikasi menggunakan grafik dan tabel statistik.

5.5.1 Grafik Perbandingan Kinerja Antar Metode

1. Grafik perbandingan kinerja *random forest* dengan melakukan PCA dan tanpa melakukan PCA.

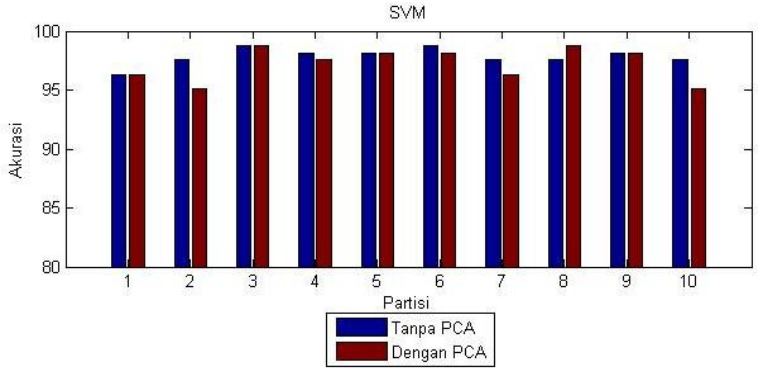
Dengan melakukan PCA, nilai akurasi dari *random forest* lebih rendah pada semua kasus dibandingkan dengan yang tanpa melakukan PCA. Penjelasan lebih jauh dapat dilihat pada Gambar 5.3.



Gambar 5.3 Hasil metode klasifikasi *Random Forest* tanpa melakukan PCA dan yang melakukan P
CA

2. Grafik perbandingan kinerja SVM dengan melakukan PCA dan tanpa melakukan PCA.

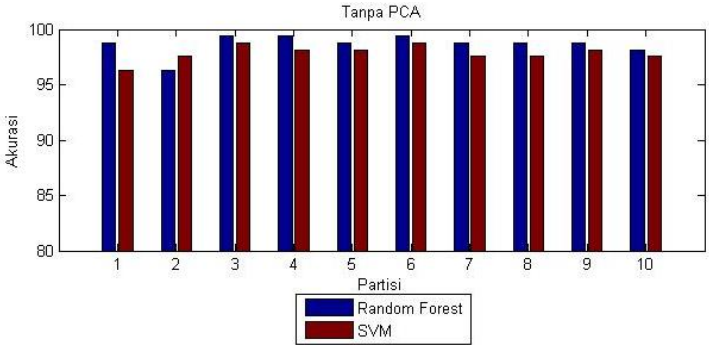
Dalam beberapa kasus, penggunaan PCA bisa menghasilkan nilai akurasi yang lebih tinggi untuk SVM. Untuk penjelasan yang lebih detail dapat dilihat pada Gambar 5.4.



Gambar 5.4 Hasil metode klasifikasi SVM tanpa melakukan PCA dan yang melakukan PCA

3. Grafik perbandingan kinerja *random forest* dengan SVM tanpa melakukan PCA.

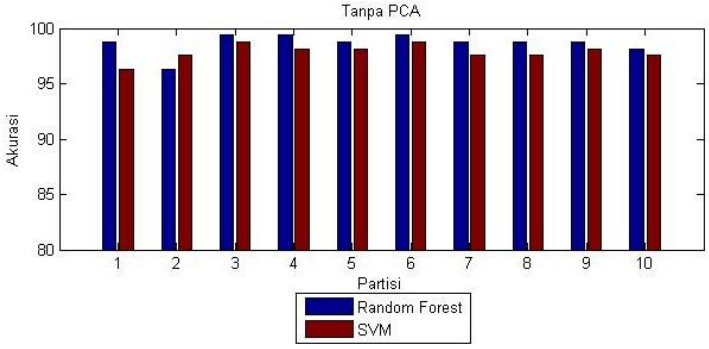
Nilai akurasi *random forest* lebih tinggi dibandingkan dengan SVM pada semua kasus ketika tidak melakukan PCA. Untuk penjelasan lebih lanjut dapat dilihat pada Gambar 5.5.



Gambar 5.5 Hasil metode klasifikasi *Random Forest* dan SVM tanpa melakukan PCA

4. Grafik perbandingan kinerja *random forest* dengan SVM dengan melakukan PCA.

Ketika digunakan PCA, pada beberapa kasus nilai akurasi SVM lebih tinggi dibandingkan *random forest*. Untuk penjelasan yang lebih detail dapat dilihat pada Gambar 5.6.



Gambar 5.6 Hasil metode klasifikasi *Random Forest* dan SVM dengan melakukan PCA

5.5.2 Tabel Statistik Perbandingan Kinerja Antar Metode

Rata-rata nilai akurasi, *precision*, *recall* dari *random forest* lebih tinggi dibandingkan rata-rata nilai dari SVM jika tanpa menggunakan PCA. Ketika PCA digunakan rata-rata *precision* untuk SVM lebih tinggi dibandingkan *random forest*. Kemudian penggunaan PCA pada *dataset* menyebabkan rata-rata akurasi turun 1.35% untuk *random forest* dan 0.62% untuk SVM. Serta rata-rata nilai *precision* turun sebesar 1.32% untuk *random forest* dan naik 0.21% untuk SVM. Kemudian rata-rata nilai *recall* turun 1.35% untuk *random forest* dan 0.90% untuk SVM. Untuk penjelasan yang lebih detail dapat dilihat pada statistik di Tabel 5.12, Tabel 5.13 dan Tabel 5.14.

Sedangkan rata-rata waktu komputasi dari SVM lebih rendah dibandingkan dengan rata-rata dari *random forest* baik itu menggunakan PCA maupun tanpa PCA. Jika digunakan PCA maka waktu komputasi menjadi 3.10 kali lebih cepat untuk SVM dan 1.82 kali lebih cepat untuk *random forest*. Untuk penjelasan lebih detail dapat dililihat pada Tabel 5.15.

Tabel 5.12 Perbandingan kinerja akurasi antar metode

Statistik	**Akurasi (%)**			
	Tanpa PCA		**Dengan PCA**	
	RF	**SVM**	**RF**	**SVM**
Minimum	96.30	96.30	95.68	95.06
Maksimum	99.38	98.77	99.38	98.77
Rata-rata	98.64	97.84	97.28	97.22

Tabel 5.13 Perbandingan kinerja *precision* antar metode

Statistik	Precision (%)			
	Tanpa PCA		Dengan PCA	
	RF	SVM	RF	SVM
Minimum	96.34	96.05	96.34	96.34
Maksimum	100.00	100.00	100.00	100.00
Rata-rata	99.02	98.26	97.69	98.04

Tabel 5.14 Perbandingan kinerja *recall* antar metode

Statistik	Recall (%)			
	Tanpa PCA		Dengan PCA	
	RF	SVM	RF	SVM
Minimum	96.34	95.12	94.12	92.68
Maksimum	100.00	98.84	100.00	98.84
Rata-rata	98.29	97.43	96.94	96.53

Tabel 5.15 Perbandingan waktu komputasi antar metode

Statistik	Waktu komputasi (milidetik)			
	Tanpa PCA		Dengan PCA	
	RF	SVM	RF	SVM
Minimum	295.6	164.5	145.0	46.0
Maksimum	325.2	176.1	151.5	49.1
Rata-rata	308.6	168.8	148.9	47.9

5.5.3 Hasil Evaluasi Perbandingan Kinerja Antar Metode

Berikut merupakan hasil evaluasi dari dari perbandingan antar metode:

1. PCA lebih berpengaruh pada hasil klasifikasi *random forest* dibandingkan pada SVM. Hal ini dapat dibuktikan pada Tabel 5.12 sampai Tabel 5.14. Selisih rata-rata akurasi, *precision*,

recall dari penggunaan PCA dan tanpa penggunaan PCA untuk *random forest* lebih tinggi dibandingkan dengan SVM yaitu 1.35%, 1.32% dan 1.35% untuk *random forest* dibandingkan dengan 0.62%, -0.21% dan 0.90% untuk SVM.

2. Klasifikasi dengan menggunakan *random forest* mempunyai rata-rata akurasi, dan *recall* yang lebih tinggi dibandingkan dengan SVM baik itu dengan menggunakan PCA pada *dataset* maupun tidak. Sedangkan rata-rata nilai *precision* dari SVM lebih tinggi dibandingkan dengan *random forest*. Poin ini dapat dibuktikan pada Tabel 5.12 sampai Tabel 5.14.
3. Rata-rata waktu komputasi SVM lebih kecil dibandingkan *random forest* baik itu menggunakan PCA maupun tidak. Hal ini dapat dilihat pada Tabel 5.15. Sedangkan penggunaan PCA menyebabkan waktu komputasi menjadi 3.10 kali lebih cepat untuk SVM dan 1.82 kali lebih cepat untuk *random forest*.
4. Sedangkan jika digunakan untuk mendeteksi citra *spam* pada *dataset* baru SVM lebih unggul dari pada *random forest* dengan rata-rata akurasi, *precision* dan *recall* bernilai 81,01%, 84,81% dan 78,82% dibandingkan *random forest* dengan nilai rata-rata 72,47%, 67,09% dan 75,18%.

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan perangkat lunak lebih lanjut.

6.1 Kesimpulan

Dari hasil pengerjaan Tugas Akhir ini penulis berkesimpulan sebagai berikut:

1. Fitur tekstur bisa digunakan untuk deteksi citra *spam* dengan rata-rata nilai akurasi, *precision* dan *recall* 98.64%, 99.02% dan 98.29% untuk *random forest* serta 97.84%, 98.26% dan 97.43% untuk SVM tanpa melakukan PCA pada *dataset* ISH.
2. Metode PCA dapat digunakan untuk mengurangi waktu komputasi dengan menyebabkan waktu komputasi menjadi 3.10 kali lebih cepat untuk SVM dan 1.82 kali lebih cepat untuk *random forest* dengan menggunakan *dataset* ISH.
3. Penggunaan PCA pada *dataset* menyebabkan rata-rata nilai akurasi dan *recall* menjadi turun 1.35% dan 1.35% untuk *random forest* dan 0.62% dan 0.90% untuk SVM. Sedangkan rata-rata nilai *precision* dari *random forest* turun 1.32% dan untuk SVM naik 0.21%.
4. Jika digunakan untuk mendeteksi citra baru SVM menghasilkan rata-rata akurasi, *precision* dan *recall* bernilai 81.01%, 84.81% dan 78.82% lebih tinggi dibandingkan *random forest* dengan nilai rata-rata 72.47%, 67.09% dan 75.18%.

6.2 Saran

Saran untuk pengembangan deteksi citra *spam* adalah perlu untuk mencari alternatif selain perangkat lunak MaZda untuk ekstraksi fitur tekstur. Selain karena tidak menyediakan pustaka

untuk memanfaatkannya pada pengembangan perangkat lunak yang memerlukan analisis fitur tekstur, MaZda juga merupakan perangkat lunak yang kode sumbernya tertutup sehingga tidak dimungkinkan untuk memberikan kontribusi pengembangan perangkat lunak tersebut tanpa seizing pemilik perangkat lunak tersebut.

LAMPIRAN A

Tabel A.1 Daftar nama fitur tekstur hasil ekstraksi oleh MaZda (Bagian Pertama)

No.	Fitur yang dihitung	Fitur Dasar	Keterangan Tambahan
1	_Area	*Histogram*	
2	Mean		
3	Variance		
4	Skewness		
5	Kurtosis		
6	Perc.01%		
7	Perc.10%		
8	Perc.50%		
9	Perc.90%		
10	Perc.99%		
11	_Area_S(1,0)	*Co-occurrence Matrix*	d=1 dan $\theta=90^o$
12	S(1,0)AngScMom		
13	S(1,0)Contrast		
14	S(1,0)Correlat		
15	S(1,0)SumOfSqs		
16	S(1,0)InvDfMom		
17	S(1,0)SumAverg		
18	S(1,0)SumVarnc		
19	S(1,0)SumEntrp		
20	S(1,0)Entropy		
21	S(1,0)DifVarnc		
22	S(1,0)DifEntrp		
23	_Area_S(0,1)		d=1 dan $\theta=0^o$
24	S(0,1)AngScMom		

Tabel A.2 Daftar nama fitur tekstur hasil ekstraksi oleh MaZda (Bagian Kedua)

No.	Fitur yang dihitung	Fitur Dasar	Keterangan Tambahan
25	S(0,1)Contrast	*Co-occurrence Matrix*	
26	S(0,1)Correlat		
27	S(0,1)SumOfSqs		
28	S(0,1)InvDfMom		
29	S(0,1)SumAverg		
30	S(0,1)SumVarnc		
31	S(0,1)SumEntrp		
32	S(0,1)Entropy		
33	S(0,1)DifVarnc		
34	S(0,1)DifEntrp		
35	_Area_S(1,1)		d=1 dan $\theta=135^o$
36	S(1,1)AngScMom		
37	S(1,1)Contrast		
38	S(1,1)Correlat		
39	S(1,1)SumOfSqs		
40	S(1,1)InvDfMom		
41	S(1,1)SumAverg		
42	S(1,1)SumVarnc		
43	S(1,1)SumEntrp		
44	S(1,1)Entropy		
45	S(1,1)DifVarnc		
46	S(1,1)DifEntrp		
47	_Area_S(1,-1)		d=1 dan $\theta=45^o$
48	S(1,-1)AngScMom		
49	S(1,-1)Contrast		

Tabel A.3 Daftar nama fitur tekstur hasil ekstraksi oleh MaZda (Bagian Ketiga)

No.	Fitur yang dihitung	Fitur Dasar	Keterangan Tambahan
50	S(1,-1)Correlat	*Co-occurrence Matrix*	d=1 dan θ=45°
51	S(1,-1)SumOfSqs		
52	S(1,-1)InvDfMom		
53	S(1,-1)SumAverg		
54	S(1,-1)SumVarnc		
55	S(1,-1)SumEntrp		
56	S(1,-1)Entropy		
57	S(1,-1)DifVarnc		
58	S(1,-1)DifEntrp		
59	_Area_S(2,0)		d=2 dan θ=90°
60	S(2,0)AngScMom		
61	S(2,0)Contrast		
62	S(2,0)Correlat		
63	S(2,0)SumOfSqs		
64	S(2,0)InvDfMom		
65	S(2,0)SumAverg		
66	S(2,0)SumVarnc		
67	S(2,0)SumEntrp		
68	S(2,0)Entropy		
69	S(2,0)DifVarnc		
70	S(2,0)DifEntrp		
71	_Area_S(0,2)		d=2 dan θ=0°
72	S(0,2)AngScMom		
73	S(0,2)Contrast		
74	S(0,2)Correlat		
75	S(0,2)SumOfSqs		
76	S(0,2)InvDfMom		
77	S(0,2)SumAverg		

Tabel A.4 Daftar nama fitur tekstur hasil ekstraksi oleh MaZda (Bagian Keempat)

No.	Fitur yang dihitung	Fitur Dasar	Keterangan Tambahan
78	S(0,2)SumVarnc	*Co-occurrence Matrix*	d=2 dan θ=0°
79	S(0,2)SumEntrp		
80	S(0,2)Entropy		
81	S(0,2)DifVarnc		
82	S(0,2)DifEntrp		
83	_Area_S(2,2)		d=2 dan θ=135°
84	S(2,2)AngScMom		
85	S(2,2)Contrast		
86	S(2,2)Correlat		
87	S(2,2)SumOfSqs		
88	S(2,2)InvDfMom		
89	S(2,2)SumAverg		
90	S(2,2)SumVarnc		
91	S(2,2)SumEntrp		
92	S(2,2)Entropy		
93	S(2,2)DifVarnc		
94	S(2,2)DifEntrp		
95	_Area_S(2,-2)		d=2 dan θ=45°
96	S(2,-2)AngScMom		
97	S(2,-2)Contrast		
98	S(2,-2)Correlat		
99	S(2,-2)SumOfSqs		
100	S(2,-2)InvDfMom		
101	S(2,-2)SumAverg		
102	S(2,-2)SumVarnc		
103	S(2,-2)SumEntrp		

Tabel A.5 Daftar nama fitur tekstur hasil ekstraksi oleh MaZda (Bagian Kelima)

No.	Fitur yang dihitung	Fitur Dasar	Keterangan Tambahan
104	S(2,-2)Entropy	*Co-occurrence Matrix*	d=2 dan θ=45^o
105	S(2,-2)DifVarnc		
106	S(2,-2)DifEntrp		
107	_Area_S(3,0)		d=3 dan θ=90^o
108	S(3,0)AngScMom		
109	S(3,0)Contrast		
110	S(3,0)Correlat		
111	S(3,0)SumOfSqs		
112	S(3,0)InvDfMom		
113	S(3,0)SumAverg		
114	S(3,0)SumVarnc		
115	S(3,0)SumEntrp		
116	S(3,0)Entropy		
117	S(3,0)DifVarnc		
118	S(3,0)DifEntrp		
119	_Area_S(0,3)		d=3 dan θ=0^o
120	S(0,3)AngScMom		
121	S(0,3)Contrast		
122	S(0,3)Correlat		
123	S(0,3)SumOfSqs		
124	S(0,3)InvDfMom		
125	S(0,3)SumAverg		
126	S(0,3)SumVarnc		
127	S(0,3)SumEntrp		
128	S(0,3)Entropy		

Tabel A.6 Daftar nama fitur tekstur hasil ekstraksi oleh MaZda (Bagian Keenam)

No.	Fitur yang dihitung	Fitur Dasar	Keterangan Tambahan
129	S(0,3)DifVarnc	*Co-occurrence Matrix*	d=3 dan θ=0°
130	S(0,3)DifEntrp		
131	_Area_S(3,3)		d=3 dan θ=135°
132	S(3,3)AngScMom		
133	S(3,3)Contrast		
134	S(3,3)Correlat		
135	S(3,3)SumOfSqs		
136	S(3,3)InvDfMom		
137	S(3,3)SumAverg		
138	S(3,3)SumVarnc		
139	S(3,3)SumEntrp		
140	S(3,3)Entropy		
141	S(3,3)DifVarnc		
142	S(3,3)DifEntrp		
143	_Area_S(3,-3)		d=3 dan θ=45°
144	S(3,-3)AngScMom		
145	S(3,-3)Contrast		
146	S(3,-3)Correlat		
147	S(3,-3)SumOfSqs		
148	S(3,-3)InvDfMom		
149	S(3,-3)SumAverg		
150	S(3,-3)SumVarnc		
151	S(3,-3)SumEntrp		
152	S(3,-3)Entropy		
153	S(3,-3)DifVarnc		

Tabel A.7 Daftar nama fitur tekstur hasil ekstraksi oleh MaZda (Bagian Ketujuh)

No.	Fitur yang dihitung	Fitur Dasar	Keterangan Tambahan
154	S(3,-3)DifEntrp	*Co-occurrence Matrix*	d=3 dan θ=45^o
155	_Area_S(4,0)		d=4 dan θ=90^o
156	S(4,0)AngScMom		
157	S(4,0)Contrast		
158	S(4,0)Correlat		
159	S(4,0)SumOfSqs		
160	S(4,0)InvDfMom		
161	S(4,0)SumAverg		
162	S(4,0)SumVarnc		
163	S(4,0)SumEntrp		
164	S(4,0)Entropy		
165	S(4,0)DifVarnc		
166	S(4,0)DifEntrp		
167	_Area_S(0,4)		d=4 dan θ=0^o
168	S(0,4)AngScMom		
169	S(0,4)Contrast		
170	S(0,4)Correlat		
171	S(0,4)SumOfSqs		
172	S(0,4)InvDfMom		
173	S(0,4)SumAverg		
174	S(0,4)SumVarnc		
175	S(0,4)SumEntrp		
176	S(0,4)Entropy		
177	S(0,4)DifVarnc		
178	S(0,4)DifEntrp		

Tabel A.8 Daftar nama fitur tekstur hasil ekstraksi oleh MaZda (Bagian Kedelapan)

No.	Fitur yang dihitung	Fitur Dasar	Keterangan Tambahan
179	_Area_S(4,4)	*Co-occurrence Matrix*	d=4 dan θ=135^o
180	S(4,4)AngScMom		
181	S(4,4)Contrast		
182	S(4,4)Correlat		
183	S(4,4)SumOfSqs		
184	S(4,4)InvDfMom		
185	S(4,4)SumAverg		
186	S(4,4)SumVarnc		
187	S(4,4)SumEntrp		
188	S(4,4)Entropy		
189	S(4,4)DifVarnc		
190	S(4,4)DifEntrp		
191	_Area_S(4,-4)		d=4 dan θ=45^o
192	S(4,-4)AngScMom		
193	S(4,-4)Contrast		
194	S(4,-4)Correlat		
195	S(4,-4)SumOfSqs		
196	S(4,-4)InvDfMom		
197	S(4,-4)SumAverg		
198	S(4,-4)SumVarnc		
199	S(4,-4)SumEntrp		
200	S(4,-4)Entropy		
201	S(4,-4)DifVarnc		
202	S(4,-4)DifEntrp		
203	_Area_S(5,0)		d=5 dan θ=90^o

Tabel A.9 Daftar nama fitur tekstur hasil ekstraksi oleh MaZda (Bagian Kesembilan)

No.	Fitur yang dihitung	Fitur Dasar	Keterangan Tambahan
204	S(5,0)AngScMom	*Co-occurrence Matrix*	d=5 dan θ=90^o
205	S(5,0)Contrast		
206	S(5,0)Correlat		
207	S(5,0)SumOfSqs		
208	S(5,0)InvDfMom		
209	S(5,0)SumAverg		
210	S(5,0)SumVarnc		
211	S(5,0)SumEntrp		
212	S(5,0)Entropy		
213	S(5,0)DifVarnc		
214	S(5,0)DifEntrp		
215	_Area_S(0,5)		d=5 dan θ=0^o
216	S(0,5)AngScMom		
217	S(0,5)Contrast		
218	S(0,5)Correlat		
219	S(0,5)SumOfSqs		
220	S(0,5)InvDfMom		
221	S(0,5)SumAverg		
222	S(0,5)SumVarnc		
223	S(0,5)SumEntrp		
224	S(0,5)Entropy		
225	S(0,5)DifVarnc		
226	S(0,5)DifEntrp		
227	_Area_S(5,5)		d=5 dan θ=135^o
228	S(5,5)AngScMom		

Tabel A.10 Daftar nama fitur tekstur hasil ekstraksi oleh MaZda (Bagian Kesepuluh)

No.	Fitur yang dihitung	Fitur Dasar	Keterangan Tambahan
229	S(5,5)Contrast	*Co-occurrence Matrix*	d=5 dan θ=135^o
230	S(5,5)Correlat		
231	S(5,5)SumOfSqs		
232	S(5,5)InvDfMom		
233	S(5,5)SumAverg		
234	S(5,5)SumVarnc		
235	S(5,5)SumEntrp		
236	S(5,5)Entropy		
237	S(5,5)DifVarnc		
238	S(5,5)DifEntrp		
239	_Area_S(5,-5)		d=5 dan θ=45^o
240	S(5,-5)AngScMom		
241	S(5,-5)Contrast		
242	S(5,-5)Correlat		
243	S(5,-5)SumOfSqs		
244	S(5,-5)InvDfMom		
245	S(5,-5)SumAverg		
246	S(5,-5)SumVarnc		
247	S(5,-5)SumEntrp		
248	S(5,-5)Entropy		
249	S(5,-5)DifVarnc		
250	S(5,-5)DifEntrp		
251	Horzl_RLNonUni	*Run Length Matrix*	θ=0^o
252	Horzl_GLevNonU		
253	Horzl_LngREmph		

Tabel A.11 Daftar nama fitur tekstur hasil ekstraksi oleh MaZda (Bagian Kesebelas)

No.	Fitur yang dihitung	Fitur Dasar	Keterangan Tambahan
254	Horzl_ShrtREmp	*Run Length Matrix*	$\theta=0^o$
255	Horzl_Fraction		
256	Vertl_RLNonUni		$\theta=90^o$
257	Vertl_GLevNonU		
258	Vertl_LngREmph		
259	Vertl_ShrtREmp		
260	Vertl_Fraction		
261	45dgr_RLNonUni		$\theta=45^o$
262	45dgr_GLevNonU		
263	45dgr_LngREmph		
264	45dgr_ShrtREmp		
265	45dgr_Fraction		
266	135dr_RLNonUni		$\theta=135^o$
267	135dr_GLevNonU		
268	135dr_LngREmph		
269	135dr_ShrtREmp		
270	135dr_Fraction		
271	_AreaGr	*Gradient*	
272	GrMean		
273	GrVariance		
274	GrSkewness		
275	GrKurtosis		
276	GrNonZeros		

Tabel A.12 Daftar nama fitur tekstur hasil ekstraksi oleh MaZda (Bagian Keduabelas)

No.	Fitur yang dihitung	Fitur Dasar	Keterangan Tambahan
277	_AreaARM	*Autoregressive Model*	
278	Teta1		
279	Teta2		
280	Teta3		
281	Teta4		
282	Sigma		

DAFTAR PUSTAKA

[1] THE RADICATI GROUP, INC., "Email Statistics Report, 2009-2013," THE RADICATI GROUP, INC., Palo Alto, 2013.

[2] Apache Software Foundation, "What is SpamAssassin?," 13 Februari 2009. [Online]. Available: http://wiki.apache.org/spamassassin/SpamAssassin. [Diakses 18 Februari 2013].

[3] N. Aye dan M. W. Win, "Identification of Image Spam by Using Histogram," International Journal of Science and Research (IJSR), vol. 2, no. 11, p. 310, 2013.

[4] B. Al-Duwairi, I. Khater dan O. Al-Jarrah, "Detection Image Spam Using Image Texture Features," International Journal for Information Security Research (IJISR), vol. 2, no. 3/4, p. 344, 2012.

[5] P. Szczypinski, M. Strzelecki, A. Materka dan A. Klepaczko, "MaZda-A software package for image texture analysis," Computer Methods and Programs in Biomedicine, vol. 94(1), pp. 66-76, 2009.

[6] E. Jones, T. Oliphant dan P. Peterson, "SciPy: Open Source Scientific Tools for Python," 2001--. [Online]. Available: http://www.scipy.org/. [Diakses 25 12 2015].

[7] S. v. d. Walt, S. C. Colbert dan G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation," Computing in Science & Engineering, vol. 13, no. 2, pp. 22-30, 2011.

[8] J. L. S. J. N.-I. F. B. J. D. W. N. Y. E. G. T. Y. Stéfan van der Walt, "scikit-image: Image processing in Python," PeerJ 2:e453, 2014.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V.

Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot dan E. Duchesnay, "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.

[10] W. McKinney, "Data Structures for Statistical Computing in Python," Proceedings of the 9th Python in Science Conference, pp. 51-56, 2010.

[11] Y. Gao, M. Yang dan X. Zhao, "Image Spam Hunter," Acoustics, Speech and Signal Processing ICASSP 2008, pp. 1765, 1768, 2008.

[12] A. Materka dan M. Strzelecki, "Texture Analysis Methods – A Review," dalam COST B11 report, Brussels, 1998.

[13] A. Materka, MaZda User's Manual, 1999-2006.

[14] M. M. Galloway, "Texture Analysis Using Gray Level Run Lengths," COMPUTER GRAPHICS AND IMAGE PROCESSING, vol. 4, pp. 172-179, 1975.

[15] R. M. Haralick, K. Shanmugam dan I. Dinstein, "Textural Features for Image Classification," IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, Vol. %1 dari %2SMC-3, no. 6, pp. 610-621, 1973.

[16] Y. Hu dan T. Dennis, "Texture image segmentation by context enhanced clustering," Image Signal Process, vol. 141, no. 6, pp. 413-421, 1994.

[17] R. C. Gonzales dan R. E. Woods, Digital Image Processing 2nd Edition, New Jesey: Prentice Hall, 2002.

[18] A. Jensen dan A. Cour-Harbo, Ripples in Mathematics The Discrete Wavelete Transform, Springer Berlin Heidelberg, 2001.

[19] A. Law dan M. Wiener, "Classification and Regression by randomForest," Desember 2002. [Online]. Available: ftp://131.252.97.79/Transfer/Treg/WFRE_Articles/Liaw_02_Classification%20and%20regression%20by%20randomForest.pdf. [Diakses 25 Februari 2014].

[20] V. Sazona, "Implementation and Evaluation of a Random Forest Machine Learning Algorithm," [Online]. Available: https://studentnet.cs.manchester.ac.uk/pgt/COMP61011/goodProjects/Sazonau.pdf. [Diakses 15 Desember 2015].

[21] L. Breiman, "Random Forest," Machine Learning, vol. 45, pp. 5-32, 2001.

[22] B. C. Lovell dan C. J. Walder, "Support Vector Machines for Business Applications," [Online]. Available: http://www.researchgate.net/profile/Brian_Lovell2/publication/37617731_Support_Vector_Machines_for_Business_Applications/file/9fcfd50741a046d340.pdf. [Diakses 26 Februari 2014].

[23] D. Boswell, "Introduction to Support Vector Machines," 6 Agustus 2002. [Online]. Available: http://www.work.caltech.edu/~boswell/IntroToSVM.pdf. [Diakses 25 Februari 2014].

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Agus Tri Wibowo, lahir di Klaten, 7 Agustus 1992. Penulis menempuh pendidikan mulai dari SD Negeri Baturan I (1998-2004), SMP Negeri Prambanan I (2004-2007), SMA Negeri I Klaten (2007-2010) dan S1 Teknik Informatika ITS.

Penulis dapat dihubungi melalui *email*: agus.tri.wb@gmail.com

(Halaman ini sengaja dikosongkan)