

**KLASIFIKASI IKAN MENGGUNAKAN
*VIOLA-JONES OBJECT DETECTION FRAMEWORK***

Proposal Skripsi

**Disusun untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Komputer**



Oleh:

Nehemiah Austen Pison

1313619021

**PROGRAM STUDI ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN
ALAM UNIVERSITAS NEGERI JAKARTA**

LEMBAR PENGESAHAN

Dengan ini saya mahasiswa Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Jakarta.

Nama : Nehemiah Austen Pison

No. Registrasi : 1313619012

Program Studi : Ilmu Komputer

Judul : Klasifikasi Ikan Dengan Menggunakan Viola-Jones
Object Detection Framework

Menyatakan bahwa proposal skripsi ini telah siap diajukan untuk seminar pra skripsi.

Menyetujui,

Dosen Pembimbing I

Dosen Pembimbing II



Muhammad Eka Suryana, M.Kom

NIP. 19851223 201212 1 002

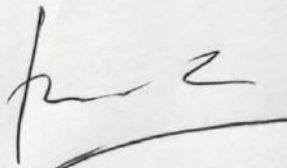


Med Irzal, M.Kom.

NIP. 19770615 200312 1 001

Mengetahui,

Koordinator Program Studi Ilmu Komputer



Ria Arafiah, M.Si.

NIP. 19751121 200501 2 004

Kata Pengantar

Puji Syukur penulis panjatkan kepada kehadiran Tuhan Yang Maha Esa, karena atas rahmat dan karunia-nya sehingga penulis dapat menyelesaikan proposal skripsi ini dengan baik. Adapun jenis penelitian dengan judul *Klasifikasi Ikan Dengan Menggunakan Viola-Jones Object Detection Framework*.

Dalam menyelesaikan proposal ini, penulis selalu mendapat bantuan dan dorongan dari orang sekitar. Oleh maka dari itu, penulis ingin menyampaikan terima kasih kepada:

1. Para petinggi di lingkungan FMIPA Universitas Negeri Jakarta
2. Ibu Ria Arafiah, M.Si. selaku Koordinator Program Studi Ilmu Komputer.
3. Bapak Muhammad Eka Suryana, M.Kom selaku Dosen Pembimbing I yang telah membimbing dalam pengerjaan proposal skripsi ini.
4. Med Irzal, M.Kom selaku Dosen Pembimbing II yang telah membimbing dalam pengerjaan proposal skripsi ini.
5. Orangtua penulis yang telah memberi dukungan selama pengerjaan proposal skripsi ini.
6. Teman-teman yang telah memberikan dukungan dan bantuan dalam pengerjaan proposal skripsi ini.

Dalam penulisan proposal skripsi ini penulis menyadari keterbatasan ilmu pengetahuan dan kemampuan penulis yang menyebabkan proposal ini jauh dari sempurna, baik dari segi penulisan, penyajian materi, dan juga bahasa. Oleh karena itu penulis meminta kritik dan saran yang dapat dijadikan sebagai pembelajaran serta dapat membangun penulis agar lebih baik lagi kedepannya.

Akhir kata, penulis berharap proposal skripsi ini dapat bermanfaat bagi semua pihak. Semoga Tuhan Yang Maha Esa senantiasa membalas kebaikan semua pihak yang telah membantu penulis dalam menyelesaikan proposal ini.

Jakarta, 2 Juni 2023

Penulis

DAFTAR ISI

LEMBAR PENGESAHAN	i
Kata Pengantar	ii
DAFTAR ISI.....	iii
DAFTAR GAMBAR	v
BAB 1 PENDAHULUAN	1
1.1 Latar Belakang Masalah.....	1
1.2 Rumusan Masalah	4
1.3 Pembatasan Masalah	4
1.4 Tujuan Penelitian.....	5
1.5 Manfaat Penelitian.....	5
BAB 2 KAJIAN PUSATAKA.....	6
2.1 Pengertian Deteksi Objek.....	6
2.2 Integral Image	7
2.3 Boosting	9
2.4 Adaboost	9
2.4.1 Adaboost.M1	10
2.4.2 Adaboost.M2.....	12
2.4.3 Weak Learner	14
2.5 Haar Like Features	19
2.5.1 Feature Pool	19
2.6 Attentional Cascade	22
BAB 3 DESAIN MODEL.....	24

3.1 Flowchart Viola-Jones Object Detection	24
3.2 Desain Sistem.....	24
3.3 Training Strong Classifier	25
3.3.1 Input Dataset Pelatihan Yang Sudah Dianotasi	25
3.3.2 Menentukan Nilai Range Optimum Untuk Setiap Fitur	26
3.3.3 Boosting	27
3.3.4 Pembuatan Strong Classifier dan Attentional Casacade	28
3.3.5 Input Gambar, Grayscale dan Penghitungan Integral Image	29
3.3.6 Klasifikasi Ikan Pada Setiap Sub-Window	30
DAFTAR PUSTAKA	33

DAFTAR GAMBAR

Gambar 2.1: Sebuah nilai Integral Image dan area yang diwakilinya	7
Gambar 2.2: Jumlah dari intensitas cahaya pada persegi.....	8
Gambar 2.3: Pseudocode Adaboost.M.1	10
Gambar 2.4: Pseudocode Adaboost.M.2.....	12
Gambar 2.5: Decision Tree simpel untuk $F=G=1$ atau $J=K=1$	17
Gambar 2.6: Contoh persegi tegak dan miring 45 derajat	19
Gambar 2.7: Contoh-contoh Haarfeatures	21
Gambar 3.1: Diagram alir untuk algoritma pelatihan klasifikasi objek Adaboost	24
Gambar 3.2: Diagram alir untuk algoritma pendeteksian objek Adaboost.....	24
Gambar 3.3 Gambar asli - Grayscale – 72 x 41 pixel	26
Gambar 3.4: Haar-Like Features yang akan digunakan.....	27
Gambar 3.5: Workflow dari Attentional Cascade.....	29
Gambar 3.6 tiga fitur pertama percobaan Weber	31
Gambar 3.7 ukuran Sub-Window yang digunakan Viola Jones	32
Gambar 3.8 anotasi bounding-box dari deteksi muka Weber	32

BAB 1

PENDAHULUAN

1.1 Latar Belakang Masalah

Ikan merupakan salah satu kekayaan hayati yang dimiliki oleh bangsa Indonesia dalam jumlah yang sangat besar. Ikan telah menjadi sumber makanan bagi bangsa Indonesia dari zaman dahulu, dibuktikan dengan banyaknya resep masakan ikan yang ada di kuliner tradisional rakyat Indonesia. Selain itu banyak orang yang memelihara ikan tidak untuk dimakan melainkan untuk hiasan, seperti didalam kolam maupun akuarium.

Ikan bisa didapat melalui 2 sumber, dengan menangkap ikan dari habitat aslinya dan melalui budidaya. Budidaya ikan memerlukan banyak infrastruktur pendukung yang tidaklah murah seperti lahan, tambak atau kolam yang memadai, dan pakan dalam jumlah yang cukup besar. Walaupun mahal, budidaya ikan dapat menghasilkan keuntungan yang besar. Ikan budidaya contohnya bisa dijual dalam keadaan hidup diwilayah-wilayah yang jauh dari habitat asli ikan, hal yang sulit dilakukan dengan ikan hasil tangkapan liar. Budidaya ikan juga dapat mengurangi permintaan besar untuk ikan, yang secara tidak langsung berkontribusi terhadap berkurangnya *overfishing* atau penangkapan ikan berlebih yang berdampak pada berkurangnya populasi ikan dalam jumlah yang besar di habitat aslinya. Permintaan untuk ikan ini sangat besar di Indonesia. Kementerian Kelautan dan Perikanan Indonesia mencatat bahwa produksi ikan budidaya di Indonesia mencapai 14 juta ton pada tahun 2021 dengan nilai sebesar Rp196.000.000.000.000,00 trilliun rupiah (KKP, 2021).

Budidaya ikan di Indonesia memiliki problem yang lumayan besar yaitu diperlukannya usaha besar untuk menghitung dan mengawasi jumlah ikan yang dibudidaya. Dalam penghitungan bibit contohnya, dalam penghitungan bibit lele para pedagang masih menghitung ikan dengan cara manual (Al-Amri, 2020).

Ikan dipindahkan satu persatu atau ditimbang sesuai berat untuk mendapatkan jumlah ikan. Cara-cara ini sangat tidak efisien atau kurang akurat. Dalam metode penghitungan, ikan yang dihitung satu per satu dengan tangan atau dengan bantuan sendok atau centong, yang memungkinkan penghitung untuk mengambil ikan dengan jumlah tertentu. Metode ini bisa memakan waktu yang cukup lama bilamana ikan yang dihitung ada dalam jumlah besar, maka dari itu metode ini biasanya hanya digunakan dalam menghitung ikan dalam jumlah yang sedikit. Tingkat akurasi yang tinggi menjadi keuntungan utama dari metode penghitungan.

Sementara itu metode penimbangan hanya menghasilkan jumlah perkiraan yang tidak selalu akurat. Dalam metode ini ikan dimasukkan kedalam suatu wadah dan lalu beratnya dihitung. Berat hasil penimbangan lalu bisa dijadikan acuan kira-kira jumlah ikan yang terdapat didalam wadah. Walaupun demikian, metode ini cepat dan cukup efisien dalam menghitung ikan dalam jumlah yang sangat besar.

Problem perhitungan ikan ini akan sangat terasa pada industri budidaya ikan yang sangat mementingkan kepadatan populasi dalam tempat budidaya ikan. Populasi ikan yang berlebihan dapat memperlambat pertumbuhan ikan (Diansari et al, 2013), tapi disisi lain populasi ikan yang terlalu kecil akan mengurangi efisiensi lahan yang dimiliki peternak ikan. Dalam mengatasi problem Al-Amri (2020) menciptakan sebuah sistem penghitungan menggunakan sensor *proximity*. Hasil uji coba mendapat hasil yang baik dengan persentase error sebesar 4,07% dengan penghitungan memakan waktu 228 detik per 1000 ikan. Jauh lebih cepat dibanding kecepatan hitung manual yang memakan waktu 20 menit per 1000 ikan. Cara lain dipakai oleh Rusydi (2019) untuk mendeteksi ikan. Rusydi menciptakan sebuah alat penghitungan dengan katup otomatis yang akan terbuka bilamana jumlah ikan yang diinginkan telah tercapai. Alat tersebut mendeteksi ikan menggunakan konsep *through beam* dimana ikan akan terdeteksi ketika melewati pipa oleh inframerah dan photodiode. Alat tersebut dapat mendeteksi ikan dengan kecepatan 58ms per ikan dengan tingkat akurasi 100%.

Penggunaan alat deteksi fisik seperti yang digunakan Al-Amri (2020) maupun Rusydi (2019) memiliki beberapa kekurangan seperti ukuran ikan

bergantung kepada ukuran alat yang dipergunakan. Alat Rusyidi (2019) sangat bergantung dengan kelandaian dan kecepatan lewat ikan yang melewati pipa sensor, mengganti ukuran ikan yang akan dideteksi mengharuskan tes dilakukan ulang untuk mendapatkan pengaturan alat yang paling optimal (dalam tes, Rusyidi (2019) menemukan bahwa kelandaian pipa 30° memberikan hasil paling akurat dalam mendeteksi ikan). Sementara pada alat Al-Amri (2020), hanya lubang keluar ikan yang perlu dimodifikasi untuk mengamodasi ikan yang lebih besar. Metode-metode tersebut sangatlah tidak fleksibel dalam industri peternakan ikan yang tidak hanya menternakan satu jenis ikan saja.

Deteksi Objek Cepat (*Rapid Object Detection*) adalah sebuah algoritma yang diciptakan untuk pendeteksian muka (Viola et al, 2001). Viola (2001) menjelaskan kalau algoritma deteksi yang diciptakannya dapat mendeteksi muka dari gambar berukuran 384 x 288 pixel dari kamera berkecepatan 15 *frame* per detik. Deteksi objek dapat digunakan untuk berbagai hal seperti anotasi gambar, penghitungan mobil, deteksi muka, rekognisi muka, pelacakan gambar dan lain-lain. Setiap algoritma deteksi objek bekerja dengan cara yang berbeda-beda namun dengan konsep yang kurang lebih sama. Setiap kelas objek pasti memiliki fitur yang dapat menunjukkan jati diri objek tersebut, misalnya objek bola sepak pastilah bulat dan umumnya memiliki dua warna yaitu hitam dan putih dengan pola yang spesifik. Muka manusia memiliki mata, hidung dan mulut yang dapat dibedakan dengan makhluk lainnya misalnya dengan kucing. Metode-metode deteksi objek umumnya menggunakan pendekatan *neural network* dan *non-neural network* untuk mendefinisikan fitur dari kelas objek yang berusaha dideteksi.

Adaboost (Freund et al, 1995) adalah sebuah pendekatan *non-neural network* yang sering digunakan untuk mendefinisikan fitur dari objek yang ingin dideteksi (Weber, 2005). Adaboost telah digunakan untuk deteksi berbagai objek seperti deteksi plat nomor kendaraan bermotor (Ho et al, 2009), deteksi muka (Viola et al, 2001), deteksi pesawat terbang (Weber, 2005) dan lain-lain. Adaboost mencari fitur sebuah kelas objek dengan menggunakan sekumpulan *weak learner* untuk membuat sebuah *strong learner*. Kumpulan *weak learner* tersebut nantinya akan dinilai sesuai dengan akurasi mereka, dimana *weak learner* yang secara

konsisten benar menebak fitur sebuah objek akan memiliki nilai lebih dalam keputusan klasifikasi akhir. Weber (1995) menjelaskan bahwa metode Adaboost dapat digambarkan selayaknya seorang pejudi pacuan kuda yang kalah terus menerus. Pejudi tersebut lalu memutuskan untuk menjadikan teman-teman penjudinya menjadi acuan untuk taruhan berikutnya. Adaboost bekerja layaknya penjudi tersebut, membuat asumsi dari berbagai pendapat lemah. Tentu saja bilamana sang penjudi melihat bahwa seorang temannya bertaruh dengan baik berulang-ulang kali, maka dia akan memandang tinggi pendapatnya diatas teman-teman lainnya yang tidak menang sebanyak orang tersebut. Adaboost juga bekerja mirip dengan situasi tersebut.

Berdasarkan latar yang telah dijelaskan, penulis mengusulkan untuk mendeteksi jenis ikan dengan menggunakan metode *Viola-Jones Object Detection Framework*. Adaboost dalam *Viola-Jones Object Detection Framework* berguna untuk mengkonstruksi sebuah classifier objek ikan yang nantinya dapat mendeteksi ikan dalam *input* gambar maupun video. Hasil yang diharapkan adalah sistem mampu mengklasifikasi ikan dari gambar maupun video secara akurat.

1.2 Rumusan Masalah

Dari uraian permasalahan diatas, perumusan masalah dalam penelitian ini adalah “Bagaimana cara mengklasifikasi ikan menggunakan metode *Viola-Jones Object Detection Framework*?”.

1.3 Pembatasan Masalah

Batasan masalah pada penelitian ini adalah:

1. Klasifikasi ikan menggunakan *Viola-Jones Object Detection Framework* yang didapat dari penelitian Viola-Jones (Viola et al, 2004)
2. Program didesain untuk mengklasifikasi ikan saja
3. Klasifikasi dilakukan dengan gambar tampak samping ikan saja
4. Klasifikasi harus bisa melakukan deteksi kelas ikan lebih dari satu (*multi-class classification*)

1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah Menciptakan metode yang mampu mempermudah klasifikasi ikan dengan menggunakan *Viola-Jones Object Detection Framework*.

1.5 Manfaat Penelitian

1. Bagi Penulis

Memperoleh gelar sarjana dalam bidang Ilmu Komputer, serta menambahkan pengalaman dalam pembuatan sebuah program komputer dengan aplikasi dunia nyata. Menambah pengetahuan penulis tentang deteksi objek, metode *Adaboost* dan *Harr-Like Features*.

2. Bagi Program Studi Ilmu Komputer

- Mahasiswa

Diharapkan penelitian ini dapat digunakan sebagai penunjang referensi, khususnya pustaka tentang klasifikasi object dengan *Viola-Jones Object Detection Framework*.

- Bagi Peneliti Selanjutnya

Diharapkan penelitian ini dapat digunakan sebagai dasar atau kajian awal bagi peneliti lain yang ingin meneliti permasalahan yang sama.

BAB 2

KAJIAN PUSTAKA

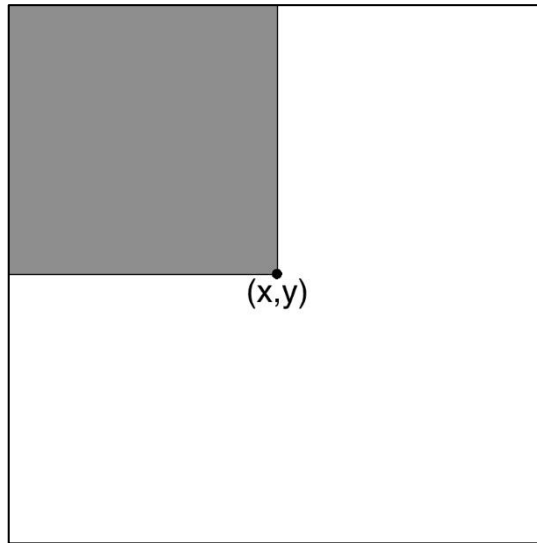
2.1 Pengertian Deteksi Objek

Deteksi Objek berhubungan dengan pendeteksian objek dari satu atau lebih *class* didalam sebuah gambar. Pada dasarnya tujuan utama dari deteksi objek adalah untuk mendeteksi objek dari suatu *class* yang sudah diketahui seperti manusia, mobil, maupun ikan. Umumnya hanya ada sedikit objek yang ada didalam suatu gambar, namun ada banyak sekali lokasi, posisi maupun ukuran dari suatu objek yang terlihat dalam sebuah gambar (Amit et al. 2020).

Setiap deteksi objek akan ditunjukkan oleh suatu bentuk informasi. Informasi ini dapat berupa informasi lokasi objek, lokasi dan ukuran, sebuah *bounding box*, atau *segmentation mask*. Dalam beberapa situasi, informasi yang diberikan akan lebih mendetail dengan paramater transformasi linear dan non-linear. Contohnya, pendeteksian wajah akan menunjukan lokasi dari mata, hidung dan mulut selain *bounding box* yang mengitari seluruh wajah. Lokasi objek juga dapat didefinisikan dengan sebuah transformator yang nantinya akan menghitung jarak objek relatif dengan kamera (Amit et al. 2020).

Tampilan sebuah objek dari suatu *class* pasti sangatlah berbeda satu sama lain. Salah satu perbedaan bisa datang dari pemrosesan gambar. Pergantian penerangan, perubahan lokasi kamera maupun artifak digitalisasi menghasilkan varias yang sangat berbeda pada gambar, bahkan dalam situasi statik dimana posisi semua objek yang akan diambil gambarnya tidak bergerak. Hal kedua yang dapat menjadi sumber variasi objek dalam *class* adalah penampilan objek yang varatif di sebuah *class* yang sama. Contohnya, Manusia memiliki muka yang berbeda-beda dan memakai baju yang berbeda-beda satu sama lain, sebuah angka atau huruf memiliki beberapa variasi penulisan yang berbeda seperti angka tujuh yang bisa ditulis dengan atau tanpa garis horizontal yang memotong ditengah, dan lain-lain (Amit et al. 2020).

2.2 Integral Image



Gambar 2.1: Sebuah nilai Integral Image dan area yang diwakilinya

Fitur-fitur kotak dapat dikomputasi secara cepat menggunakan representasi tidak langsung dari gambar, hal ini diberi nama *integral image*. *Integral image* pada lokasi x, y berisikan penjumlahan di atas dan di kiri dari x, y . *inclusive*:

$$ii(x, y) = \sum_{x^1 \leq x, y^1 \leq y} i(x^1, y^1)$$

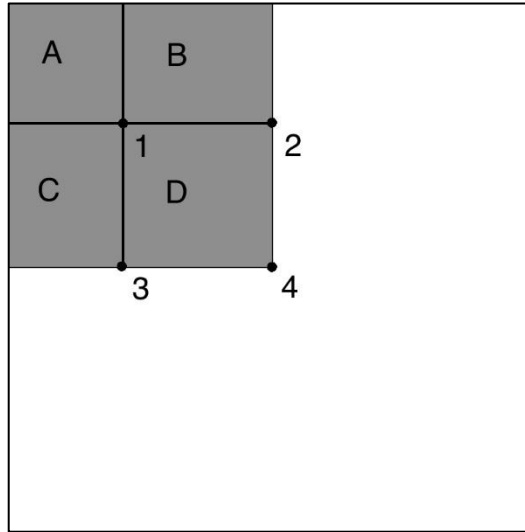
Dimana $ii(x, y)$ adalah *integral image* dan $i(x, y)$ adalah gambar aslinya (2,1).

Perhitungan dapat diulang menggunakan:

$$ii(x, y) = s(x, y) + ii(x - 1, y) + ii(x, y - 1) - ii(x - 1, y - 1)$$

(Viola et al, 2001).

Menggunakan *integral image*, semua jumlah nilai pada persegi dapat dihitung didalam 4 referensi *array* (2,2). Jelas perbedaan diantara kedua jumlah nilai persegi persegi dapat dikomputerasi dengan delapan referensi. Semenjak dua fitur persegi yang didefinisikan diatas melibatkan jumlah persegi disekitarnya, mereka dapat dikomputasi kedalam enam referensi *array*, delapan bilamana terdapat tiga fitur persegi, dan sembilan untuk empat fitur persegi (Viola et al, 2001).



Gambar 2.2: Jumlah dari intensitas cahaya pada persegi D dapat dihitung dengan 4 referensi Array. Nilai dari 1 adalah jumlah intensitas cahaya pada persegi A, Nilai dari 2 adalah persegi A + B, nilai dari 3 adalah A + C dan nilai 4 adalah A+B+C+D.

Salah satu motivasi alternatif dari *integral image* datang dari *boxlets* buatan Simard, et al (1998). Para penulisnya menunjukkan bahwa didalam kasus operasi linear (eg. $f \cdot g$), operasi linear appaun yang dapat dibalik dapat diaplikasikan ke f atau g bilamana kebalikannya dimasukan ke dalam hasil. Sebagai contoh didalam konvolusi, bilamana opeasi turunan operator diaplikasikan kedalam gambar dan *kernel*, maka hasilnya pasti akan dimasukan dua kali:

$$f * g = \iint (f^1 * g^1)$$

Penulis-penulisnya lalu menunjukkan bahwa konvolusi dapat dipercepat bila turunan dari f dan g sporadis (atau dapat dibuat demikian). *Insight* yang mirip adalah operasi linera yang dapat dibalik dapat diaplikasikan kedalam f bilamana *inverse* nya diaplikasikan kedalam g :

$$(f^{11}) * (\iint g) = f * g.$$

(Viola et al, 2001)

Dilihat didalam *framework* ini, komputasi dari jumlah persegi dapat diekspresikan sebagai produk titik, $i \cdot r$, dimana I adalah gambar dan r adalah gambar mobil kotak (dengan nilai 1 didalam *rectangle of interest* dan 0 diluar). Operasi ini dapat ditulis ulang sebagai:

$$i \cdot r = \left(\iint i \right) \cdot r^{11}.$$

Integral image adalah sebuah integral ganda sebuah gambar (yang pertama sesuai baris, yang kedua sesuai kolom). Turunan kedua dari persegi tersebut (yang pertama di baris dan di kolom) menghasilkan empat fungsi *delta* pada sudut persegi. Evaluasi dari produk titik kedua didapatkan dari empat akses *array*. (Viola et al, 2001)

Kecepatan yang didapat dari penghitungan *Integral Image* ini dapat dijustifikasi bila kita melihat bahwa perhitungan manual akan jauh memakan perhitungan yang jauh lebih banyak. Sebagai contoh, misalkan kita sedang mencari intensitas cahaya pada ukuran area 10x10 piksel. Cara manual mengharuskan kita menghitung sampai 100 kali untuk mendapat jumlah intensitas cahaya pada area tersebut, belum lagi proses ini harus diulang terus-menerus dengan ukuran dan lokasi yang berbeda. Dilain sisi, perhitungan menggunakan *Integral Image* hanya perlu mereferensi tabel yang sudah dibuat sebelum semua usaha klasifikasi, dalam hal ini kita hanya perlu melakukan perhitungan 4 kali untuk menghitung intensitas cahaya dalam area tersebut.

2.3 Boosting

Boosting adalah sebuah metode umum yang digunakan untuk meningkatkan algoritma pembelajaran apapun. Dalam teori, *boosting* dapat digunakan untuk mengurangi error dari algoritma pembelajaran “lemah” secara signifikan yang secara terus menerus menghasilkan *classifier* yang hanya perlu sedikit lebih baik dari menebak secara acak. *Boosting* bekerja dengan berulang kali menjalankan algoritma pembelajaran “lemah” dengan *dataset* latihan, lalu menggabungkan *classifier-classifier* menjadi satu *classifier* komposit (Freund et al, 1996).

2.4 Adaboost

Di bagian ini akan dibahas algoritma yang dikenal dengan Adaboost. Penulis akan membahas dua versi dari algoritma yang akan kita panggil

Adaboost.M1 dan Adaboost.M2. Keduanya setara dalam problem klasifikasi biner dan hanya berbeda di penanganan problem dengan kelas lebih dari dua.

2.4.1 Adaboost.M1

Algorithm AdaBoost.M1

Input: sequence of m examples $\langle (x_1, y_1), \dots, (x_m, y_m) \rangle$ with labels $y_i \in Y = \{1, \dots, k\}$
weak learning algorithm **WeakLearn**
integer T specifying number of iterations

Initialize $D_1(i) = 1/m$ for all i .

Do for $t = 1, 2, \dots, T$

1. Call **WeakLearn**, providing it with the distribution D_t .
2. Get back a hypothesis $h_t : X \rightarrow Y$.
3. Calculate the error of h_t : $\epsilon_t = \sum_{i: h_t(x_i) \neq y_i} D_t(i)$. If $\epsilon_t > 1/2$, then set $T = t - 1$ and abort loop.
4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.
5. Update distribution D_t : $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$
where Z_t is a normalization constant (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis: $h_{\text{fin}}(x) = \arg \max_{y \in Y} \sum_{t: h_t(x) = y} \log \frac{1}{\beta_t}$.

Gambar 2.3: Pseudocode Adaboost.M.1

Kami akan mulai dari versi Adaboost yang paling simpel, yaitu AdaBoost.M1. Adaboost mengambil input set pelatihan m contoh: $S = ((x_1, y_1), \dots, (x_m, y_m))$ dimana x_1 adalah sebuah contoh yang ditarik dari X dan diwakilkan oleh beberapa cara (umumnya sebuah vektor yang terdiri dari beberapa nilai atribut) dan $y_i \in Y$ adalah label kelas yang diasosiasikan dengan x_i disini kita selalu akan berasumsi bahwa himpunan kemungkinan label dari Y selalu dari kardinalitas terbatas k .

Selain itu, Adaboost memiliki akses ke algoritma pembelajaran lain yang tidak dispesifikasi, yang umumnya dipanggil *Weak Learn*. Algoritma *Boosting* memanggil *Weak Learn* secara beruntun dalam beberapa seri ronde. Pada ronde ke- t , *Booster* memberikan ke *Weaklearn* distribusi D_t pada set pelatihan S . Setelah itu, *WeakLearn* akan mengkomputasi sebuah *classifier* atau hipotesis $h_1 : X \rightarrow Y$ yang seharusnya me-misklasifikasi pecahan yang tidak terlalu penting di contoh latihan, relatif ke D_t . Adalah tugas dari *Weak Learner* untuk menemukan hipotesis h_t yang meminimalisir *error* pelatihan $\epsilon_t = PR_{i \sim D_t}[h_t(X_i) \neq y_i]$. *Error* ini dapat diukur

dengan distribusi D_t yang dihasilkan oleh *Weak Learner*. Proses ini berlanjut sebanyak T ronde, dan pada akhirnya *Booster* akan menggabungkan seluruh hipotesis lemah h_1, \dots, h_T kedalam sebuah hipotesis akhir h_{fin} . Namun D_t dikomputasi di setiap ronde tanpa dispesifikasi, dan bagaimana h_{fin} dikomputasi. Kedua Adaboost menjawab permasalahan ini dengan cara berbeda. Adaboost.M1 menggunakan peraturan simpel pada gambar Figure 1. Distribusi awal D_1 sama pada S maka $D_1(i) = 1/m$ untuk semua i . Untuk menghitung distribusi D_{t+1} dari D_t dan hipotesis lemah terakhir h_t , kita akan mengalikan bobot dari contoh i dengan beberapa angka $\beta_t \in [0,1]$ bila h_t mengklasifikasi x_i secara benar, selain itu bobotnya tidak akan berubah. Bobotnya ilau akan di renormalisasi dengan cara dibagi dengan normalisasi konstanta Z_t . Secara efektif, contoh “mudah” yang secara benar diklasifikasi oleh banyak *Weak Learner* sebelumnya, mendapat bobot rendah, dan contoh “sulit” yang umumnya salah diklasifikasi, akan mendapat bobot tinggi. Oleh karena itu Adaboost lebih memfokuskan ke bobot dari contoh yang nampaknya sulit diklasifikasi oleh *WeakLearn*.

Angka β_t di (figure) sebagai fungsi dari ϵ_t . Hipotesis akhir h_{fin} adalah voting berbobot (atau sebuah ambang linear berbobot) dari hipotesis lemah. Setiap contoh x , h_{fin} mengeluarkan label y yang memaksimalkan jumlah dari hipotesis lemah yang memprediksi y . Bobot dari hipotesis h_t didefinisikan sebagai $\ln(1/\beta_t)$ supaya bobot yang lebih tinggi diberikan kepada hipotesis dengan error yang rendah.

Kelemahan utama dari AdaBoost.M1 adalah ia tidak bisa menangani hipotesis lemah dengan error melebihi $1/2$. Ekspektasi error dari sebuah hipotesis yang menebak secara acak sebuah label adalah $1 - 1/k$, dimana k adalah jumlah label yang mungkin. Maka dari itu keperluan AdaBoost.M1 untuk $k = 2$ adalah prediksi yang sedikit lebih baik dari menebak secara acak. Namun, ketika $k > 2$, keperluan AdaBoost.M1 jauh lebih kuat dari itu, dan mungkin sulit dipenuhi. (Freund et al, 1996)

2.4.2 Adaboost.M2

Algorithm AdaBoost.M2

Input: sequence of m examples $\langle (x_1, y_1), \dots, (x_m, y_m) \rangle$ with labels $y_i \in Y = \{1, \dots, k\}$
 weak learning algorithm **WeakLearn**
 integer T specifying number of iterations
 Let $B = \{(i, y) : i \in \{1, \dots, m\}, y \neq y_i\}$
Initialize $D_1(i, y) = 1/|B|$ for $(i, y) \in B$.
Do for $t = 1, 2, \dots, T$

1. Call **WeakLearn**, providing it with mislabel distribution D_t .
2. Get back a hypothesis $h_t : X \times Y \rightarrow [0, 1]$.
3. Calculate the pseudo-loss of h_t : $\epsilon_t = \frac{1}{2} \sum_{(i, y) \in B} D_t(i, y) (1 - h_t(x_i, y_i) + h_t(x_i, y))$.
4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.
5. Update D_t : $D_{t+1}(i, y) = \frac{D_t(i, y)}{Z_t} \cdot \beta_t^{(1/2)(1 + h_t(x_i, y_i) - h_t(x_i, y))}$
 where Z_t is a normalization constant (chosen so that D_{t+1} will be a distribution).

Output the hypothesis: $h_{fin}(x) = \arg \max_{y \in Y} \sum_{t=1}^T \left(\log \frac{1}{\beta_t} \right) h_t(x, y)$.

Gambar 2.4: Pseudocode Adaboost.M.2

Versi kedua dari AdaBoost mencoba untuk menyelesaikan masalah ini dengan mengekstensi komunikasi diantara algoritma *boosting* dan *weak learner*. Pertama, kita membiarkan *weak learner* menghasilkan hipotesis yang lebih ekspresif yang menghasilkan vektor $[0,1]^k$, daripada sebuah label Y . Secara intuitif, komponen y -ke dari vektor ini mewakili sebuah "derajat kepercayaan" dimana label yang benar adalah y . Komponen-komponen dengan nilai dekat ke 1 atau 0 sesuai dengan label-label yang dianggap mungkin dan tidak mungkin, secara berurutan.

Karena kita memberi algoritma *weak learner* lebih banyak kekuatan ekspresif, kita juga menaruh persyaratan yang lebih kompleks untuk performa hipotesa lemah. Melainkan menggunakan prediksi error yang biasa, kita meminta hipotesis lemah untuk bekerja baik dengan pengukur kesalahan yang lebih canggih yang kami sebut *pseudo-loss*. Tidak seperti error biasa yang dikomputasi dari distribusi contoh, *pseudo-loss* dikomputasi dari distribusi pasangan contoh-contoh dan label tidak benar. Dengan memanipulasi distribusi ini, algoritma *boosting* dapat memfokuskan *weak learner* tidak hanya kepada contoh yang sulit diklasifikasi, namun lebih tepatnya, ke label tidak benar yang sulit untuk didiskriminasi. Kita akan melihat bahwa algoritma AdaBoost.M2, yang berbasis ide-ide ini, mencapai *boosting* bila setiap hipotesis lemah memiliki *pseudo-loss* yang sedikit lebih baik dari menebak acak.

Secara formal, salah label berpasangan (i, y) dimana i adalah indeks dari contoh latihan dan y adalah label tidak benar yang dikaitkan dengan contoh i . Bilamana B menjadi set dari seluruh salah label:

$$B = \{(i, y) : i \in \{1, \dots, m\}, y \neq y_i\}.$$

Distribusi salah label adalah sebuah distribusi yang didefinisikan set B dari semua salah label.

Secara intuitif, kita merepresentasi seluruh salah label (i, y) sebagai representasi pertanyaan: "Apakah kita memprediksi bahwa label yang diasosikan dengan contoh x_i adalah y_i (label yang benar) atau y (salah satu label yang salah)?" Dengan interpretasi ini, bobot $D_i(i, y)$ yang didesignasi ke salah label ini mewakili pentingnya membedakan label salah y pada contoh x_i .

Sebuah hipotesis lemah h_t lalu di interpretasi dengan cara ini. Bila $h_t(x_i, y_i) = 1$ dan $h_t(x_i, y) = 0$, lalu h_t telah (dengan ebnar) memprediksi label dari x_i adalah y_i , bukan y (Jika h_t menganggap y_i sebagai "mungkin" dan y sebagai "tidak mungkin"). Demikian pula jika $h_t(x_i, y_i) = h_t(x_i, y)$, maka prediski h_t dapat dianggap sebagai tebakan acak. Interpretasi ini menunjukkan kita untuk mendefinisikan *pseudo-loss* dari hipotesis h_t dengan mempertimbangkan label salah distribusi D_t sesuai formula.

$$\epsilon_t = \frac{1}{2} \sum_{(i, y) \in B} D_t(i, y) (1 - h_t(x_i, y_i) + h_t(x_i, y)).$$

Seharusnya sudah jelas, bahwa *pseudo-loss* diminimalisir ketika label-label yang benar y_i diberikan nilai dekat dengan 1 dan label yang tidak benar $y \neq y_i$ diberikan nilai dekat dengan 0. Selebih itu, ingat bahwa *pseudo-loss* $\frac{1}{2}$ dengan mudah dapat dicapai hipotesis dengan nilai konstan h_t , dan terlebih lagi bahwa hipotesis h_t dengan *pseudo-loss* lebih besar dari $\frac{1}{2}$ dapat ditukar dengan hipotesis $1 - h_t$ yang *pseudo-loss* nya lebih kecil dari $\frac{1}{2}$.

Tujuan dari *weak learner* adalh untuk menemukan hipotesis lemah h_t dengan *pseudo-loss* kecil. Maka dari itu algoritma pembelajaran standar memerlukan modifikasi untuk dapat dipakai disituasi ini, walau modifikasi ini umumnya mudah. Setelah emndapatkan h_t , distribusi labael salah di *update*

menggunakan peraturan yang mirip digunakan di AdaBoost.M1. Hipotesis h_{fin} mengeluarkan untuk setiap contoh x , label y yang memaksimalkan bobot rata-rata dari hipotesis lemah $h_t(x, y)$. (Freund et al, 1996)

2.4.3 Weak Learner

Sesuai dengan introduksi, kita menggunakan tiga algoritma *weak learning* didalam eksperimen ini. Didalam semua kasus, contohnya dideskripsikan oleh sebuah vektor nilai yang berhubungan dengan sebuah set tetap fitur atau atribut. Nilai-nilai ini dapat berupa diskrit atau kontinyu. Sebagian contoh dapat memiliki nilai yang hilang. Ketiga *weak learner* mebuat hipotesis yang mengklasifikasi contoh dengan berulang kali mengetes nilai dari atribut yang dipilih. (Freund et al, 1996)

2.4.3.1 FindAttrTest

Weak Learner yang pertama dan plaing simpel yang akan kita panggil FindAttrTest mencari sebuah nilai atribut dengan *error* minimum (atau *pseudo-loss*) pada set latihan. Lebih tepatnya, FindAttrTest mengkomputasi sebuah *classifier* yang didefinisikan oleh sebuah atribut a , sebuah nilai v dan tiga prediksi p_0 , p_1 dan $p_?$. *Classifier* ini mengklasifikasi sebuah contoh baru x dengan cara sebagai berikut: Bila nilai dari atribut a hilang dari x , maka prediksi $p_?$. Bila atribut a adalah kontinyu dan nilainya pada x kurang lebih v , maka prediksi p_0 ; Selain itu, prediksi p_1 . Jika menggunakan *error* biasa (AdaBoost.M1), prediksi ini p_0 , p_1 dan $p_?$ hanya berupa klasifikasi simpel. Untuk *pseudo-loss*, “prediksinya” akan berupa vektor didalam $[0,1]^k$ (dimana k adalah jumlah dari kelas).

Algoritma FindAttrTest mencari sampai habis *classifier* dari bentuk yang diberikan diatas dengan *error* atau *pseudo-loss* minimum dengan mempertimbangkan distribusi yang diberikan oleh *booster*. Dengan kata lain, dengan mempertimbangkan semua nilai yang mungkin dari a , v , p_0 , p_1 dan $p_?$. Pencarian ini dapat dilakukan didalam waktu linear didalam ukuran dari set latihan (untuk setiap ronde *boosting*). (Freund et al, 1996)

2.4.3.2 FindDecRule

Weak Learner kedua melakukan pencarian *decision rule* yang lebih canggih yang mengetes gabungan dari tes nilai atribut. Algoritma ini disebut FindDecRule.

Pertama, algoritma ini memerlukan set pelatihan yang tidak berbobot, jadi kita menggunakan versi *resampling* dari *boosting*. Set pelatihan yang diberikan secara acak dibagi kedalam sebuah set bertumbuh menggunakan 70% dari data dan sebuah set *pruning* dengan sisa 30% nya. Didalam fase pertama, set bertumbuh digunakan untuk menumbuhkan sebuah daftar tes nilai atribut. Setiap tes membandingkan sebuah atribut a ke sebuah nilai v , mirip dengan tes yang digunakan di FindAttrTest. Kita menggunakan fungsi potensial berbasis entropi untuk memandu pertumbuhan dari daftar tes. Tes pada awalnya kosong, dan satu tes ditambahkan satu per satu, setiap waktu memilih tes yang akan menyebabkan penurunan potensial tertinggi. Setelah tes dipilih, hanya satu *branch* yang diekspansi, yaitu *branch* dengan sisa potensi tertinggi. Daftar terus dikembangkan dengan cara ini sampai tidak ada tes lagi yang tersisa yang akan lebih jauh mengurangi potensial.

Di fase kedua, daftar lalu di *prune* dengan memilih *prefix* dari daftar dengan *error* minimum (atau *pseudo-loss*) pada set *pruning*. (Freund et al, 1996)

2.4.3.3 C4.5 Decision Tree

Decision Tree menghasilkan sebuah *classifier* didalam bentuk sebuah *decision tree*, sebuah struktur yang berbentuk

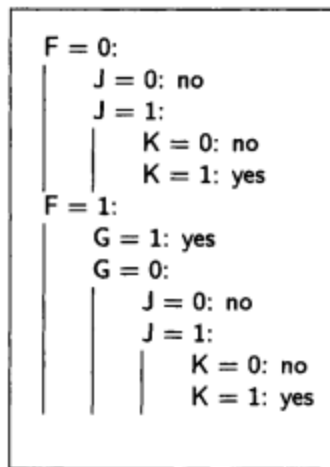
- Sebuah daun, mengindikasikan sebuah kelas, atau
- Sebuah *decision node* yang menspesifikasi sebagian tes untuk dikerjakan atas sebuah nilai atribut, dengan satu *branch* dan *subtree* untuk setiap hasil dari tes.

Sebuah *decision tree* dapat digunakan untuk mengkasifikasi sebuah kasus dengan memulai dari akar pohon dan bergerak sampai sebuah daun ditemukan. Pada setiap *node* yang bukan merupakan daun, hasil dari tes kasus dideterminasi dan perhatian berubah ke akar dari *subtree* sesuai dengan hasil tersebut. Ketika proses pada

akhirnya (dan dengan pasti) menuju ke sebuah daun, kelas dari kasus diprediksi sesuai yang ada di daun. (Quinlan, 1993)

Tulang punggung dari metode Hunt untuk membuat sebuah *decision tree* dari sebuah set T kasus data sangatlah mudah. Kita buat agar kelas didenotasi $\{C_1, C_2, \dots, C_k\}$. Ada tiga kemungkinan:

- T mengandung satu atau lebih kasus, semua dimiliki sebuah kelas C_j :
Decision tree untuk T adalah sebuah daun yang mengidentifikasi kelas C_j .
- T tidak mengandung kasus:
Decision tree lagi adalah sebuah daun, namun kelas yang diasosiasikan dengan daun harus ditentukan informasi selain dari T . Contohnya, daunnya dapat dipilih sesuai dengan sebagian pengetahuan latar dari bidang yang diketahui, seperti mayoritas kelas sepenuhnya. C4.5 memakai kelas paling sering dari *parent node* ini.
- T mengandung kasus yang merupakan bagian dari berbagai kelas:
Dalam situasi ini, idenya adalah untuk menyaring T menjadi himpunan bagian yang, atau terlihat, menuju sebuah koleksi kasus kelas tunggal. Sebuah tes T_{10} dipilih, berdasarkan atribut tunggal, yang memiliki satu atau lebih hasil yang saling eksklusif $\{O_1, O_2, \dots, O_n\}$. T dipartisi ke himpunan bawah T_1, T_2, \dots, T_n , dimana T_1 mengandung semua kasus didalam T yang memiliki hasil O_1 dari tes yang dipilih. *Decision tree* dari T terdiri dari sebuah *decision node* yang mengidentifikasi tes, dan satu *branch* untuk setiap hasil yang mungkin. Mesin pembuat pohon diaplikasikan secara rekursif untuk setiap himpunan turunan dari kasus pelatihan, supaya *branch* ke- i menuju ke *decision tree* yang dibangun dari subjek dari T_i dari kasus latihan. (Quinlan, 1993)



Gambar 2.5: Decision Tree simpel untuk $F=G=1$ atau $J=K=1$

Pada setiap *decision tree*, kondisi yang harus dipenuhi saat sebuah kasus klasifikasi oleh sebuah daun dapat ditemukan dengan menelusuri semua hasil tes yang dilewati dari akar. Pada (gambar 5-2) daun “yes” paling paling dalam diasosiasikan dengan $F=1$, $G=-$, $J=1$ dan $K=1$; kasus apapun yang memenuhi persyaratan ini akan dipetakan sebagai benar. Maka kita dapat menulis, “Bila $F=1$, $G=0$, $J=1$ dan $K=1$ maka kelasnya adalah benar,” maka tiba-tiba peraturan memiliki bentuk produksi *ubiquitous*. Faktanya, bila jalan ke setiap daun diubah ke sebuah peraturan produksi dalam bentuk ini, hasil koleksi peraturannya akan mengklasifikasi kasus sama persis seperti yang *tree* lakukan. Sebagai konsekuensi dari dasar pohon mereka, bagian “if” pada peraturan akan eksklusif dan mendalam, urutan dari peraturan tidaklah penting.

Menulis ulang *tree* menjadi sebuah koleksi peraturan, untuk setiap daun pada pohon, tidak akan menghasilkan apapun yang lebih simpel dari sebuah *tree*, semenjak akan ada sebuah peraturan untuk setiap daun. Namun, sebelum setiap peraturan akan ada kondisi-kondisi yang tidak relevan. Pada peraturan diatas konklusinya tidak ditentukan oleh nilai dari F dan G . Peraturan dapat digeneralisir dengan menghapus kondisi *superflous* tanpa mengurangi akurasi, meninggalkan peraturan:

If $J=1$

$K=1$

then class yes

Untuk menentukan kapan sebuah peraturan harus dihapus kita akan menggunakan hukum R.

If A then class C

dan hukum R- yang lebih umum

if A- then class C

Dimana A- didapatkan dengan menghapus sebuah kondisi X dari kondisi A. bukti dari pentingnya kondisi X harus ditemukan didalam pelatihan kasus yang digunakan untuk membuat *decision tree*. Setiap kasus yang memenuhi pendahuluan A- yang lebih pendek, adalah bagian atau bukan dari kelas C, dan memenuhi atau tidak kondisi X. Jumlah dari kasus pada setiap empat grup ini dapat diorganisir menjadi sebuah tabel kontigensi 2x2:

	Class C	Other Classes
Satisfies condition X	Y_1	E_1
Does not satisfy condition X	Y_2	E_2

Tabel 2.1: tabel kontigensi

Semenjak kita mencari kasus yang hanya memenuhi A-, kasus yang memenuhi kasus X juga akan tercakup oleh peraturan R awal. Ada kasus seperti $Y_1 + E_1$, E_1 dari mereka salah dikalsifikasi oleh R karena mereka bagian dari kelas lain selain C. Demikian juga kasus-kasus yang memenuhi A- tapi tidak memenuhi X akan dicakup oleh peraturan R- tapi tidak oleh peraturan awal. Ada $Y_2 + E_2$ dari mereka dengan error-error E_2 . Semenjak R- juga mencakup semua kasus yang memenuhi R, total dari kasus yang dicakup oleh R- adalah $Y_1 + Y_2 + E_1 + E_2$.

Tentu saja lebih dari satu peraturan mungkin harus dihapus ketika sebuah peraturan digeneralisir. Melaikan dengan melihat semua kemungkinan himpunan bagian yang dapat dihapus, sistemnya melakukan eliminasi langsung secara rakus: Bila satu atau lebih bisa dihapus seperti sebelumnya, kondisi tersebut dihilangkan yang menghasilkan nilai *error rate* pesimistik paling rendah dari peraturan yang

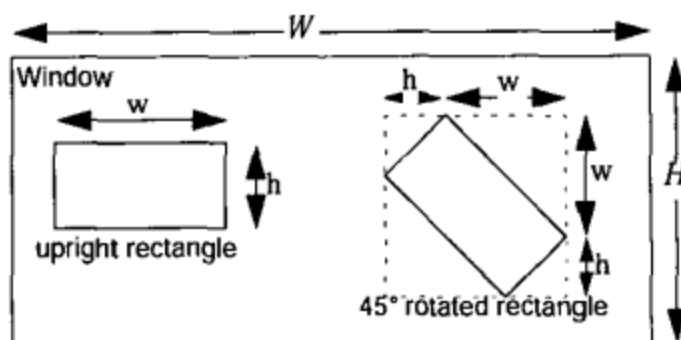
digeneralisir. Layaknya seperti semua pencarian yang rakus, tidak ada jaminan bahwa dengan meminimalisir *error rate* pesimistik pada setiap langkah akan mengarah ke *global minimum*. Sistemnya mungkin dapat ditingkatkan dengan menjalankan sebuah pencarian mengeluruh ketika angka dari kondisi kecil., dan dengan menggunakan pendekatan seperti *simulated annealing* ketika dia tidak. Namun, pencarian rakus seperti bekerja cukup baik dalam praktek, dan secara relatif cepat, sebuah pertimbangan yang penting karena generalisasi ini dilakukan berulang kali.

2.5 Haar Like Features

Fungsi utama dari menggunakan fitur daripada nilai *raw pixel* sebagai input ke sebuah algoritma pembelajaran adalah untuk mengurangi/menambahkan variabel *in-class/out-class* dibandingkan dengan menggunakan data masukan mentah, yang membuat klasifikasi lebih mudah. Fitur biasanya meng-*encode* pengetahuan tentang domain, yang sulit dipelajari dari set data masuk mentah yang terbatas.

Kompleksitas dari evaluasi fitur juga merupakan aspek yang sangat penting karena hampir semua algoritma pendeteksi objek memasukan sebuah *fixed-size window* pada semua skala gambar masuk.

2.5.1 Feature Pool



Gambar 2.6: Contoh persegi tegak dan miring 45 derajat

Marilah berasumsi bahwa unit dasar untuk pengetesan keberadaan suatu objek adalah sebuah *window* $W \times H$ piksel. Asumsi juga bahwa kita memiliki

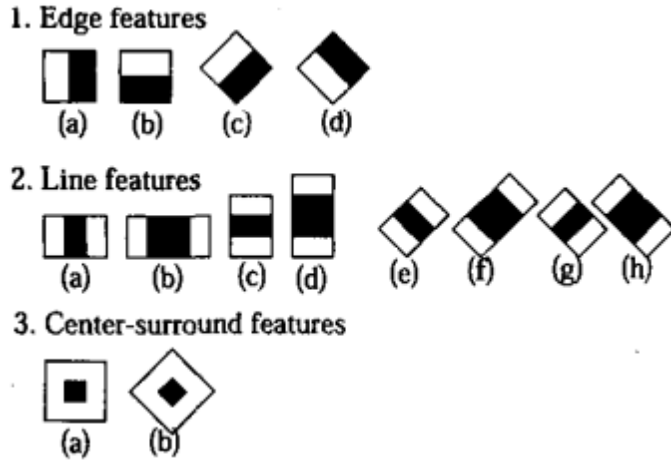
sebuah skema komputasi yang sangat cepat menghitung jumlah dari piksel dari persegi yang tegap berdiri dan diputar 45 derajat didalam *Window*. Sebuah persegi dispesifikasi oleh *tuple* $r = (x, y, w, h, \alpha)$ dengan $0 \leq x, x + w \leq W$, $0 \leq y, y + h \leq H$, $x, y \geq 0$, $w, h > 0$, dan $\alpha \in \{0^\circ, 45^\circ\}$ dan jumlah pikselnya didenotasi oleh $RecSum(r)$. Dua contoh dari persegi seperti itu ditunjukkan di (figur 1).

Set fitur mentah kita adalah set dari seluruh fitur mungkin dari bentuk dimana bobot $\omega_l \in K$, persegi r_l , dan N secara arbriter dipilih. Set fitur mentah ini (hampir) tidak terbatas besar. Untuk alasan praktikalitas, kita reduksi demikian:

1. Hanya kombinasi berbobot jumlah piksel dari 2 persegi yang diperhitungkan ($N=2$)
2. Bobot memiliki tanda yang berlawanan, dan digunakan untuk mengkompensasi perbedaan dari ukuran area diantara dua persegi. Maka, untuk persegi yang tidak bertindih kita memiliki $-w_0 \cdot Area(r_0) = w_1 \cdot Area(r_1)$.
3. Fitur ini meniru fitur haar-like dan fitur awal dari jalur visual manusia seperti *center-surround* dan *directional response*.

Kedua restriksi ini membawa kita ke 14 fitur prototipe yang ditunjukkan (fitur 2):

- Fitur empat pojok
- Fitur delapan garis, dan
- Fitur dua *center-surround*



Gambar 2.7: Contoh-contoh Haarfeatures

Prototipe-prototipe ini diskala secara mandiri pada direksi vertikal dan horizontal untuk menghasilkan fitur yang kaya dan lebih lengkap. Harus diingat bahwa fitur garis dapat dihitung dengan dua persegi saja. Maka dapat diasumsikan bahwa persegi pertama r_0 mencakup persegi hitam dan putih, dan persegi kedua r_1 mewakili area hitam. Sebagai contoh, fitur garis (2a) dengan total tinggi 2 dan lebar 6 pada pojok kiri atas (5,3) dapat dituliskan sebagai.

$$feature_l = -1.RecSum(5,3,6,2,0^\circ) + 3.RecSum(7,3,2,2,0^\circ).$$

Hanya fitur (1a), (1b), (2a), (2c), dan (4a) dari (Figur 2) telah dipakai oleh [3,4,5]. Dalam eksperimen kita, fitur-fitur tambahan secara signifikan memperkuat kekuatan ekspresional dari sistem pembelajaran dan akibatnya mengimprovisasi performa dari sistem deteksi objek. Fitur (4a) tidak dipakai karena diperkirakan oleh fitur (2g) dan (2e).

Jumlah fitur yang berasal dari setiap prototipe cukup besar dan berbeda dari prototipe ke prototipe, dan bisa dikalkulasi sebagai berikut. Biarkan $X = [W/w]$ dan $Y = [H/h]$ menjadi skala maksimum faktor pada direksi x dan y . Sebuah fitur tegap dari ukuran $w \times h$ lalu menghasilkan

$$XY \cdot \left(W + 1 - w \frac{X + 1}{2} \right) \cdot \left(H + 1 - h \frac{Y + 1}{2} \right)$$

fitur untuk sebuah gambar dengan ukuran $W \times H$, sementara fitur yang diputar 45° menghasilkan

$$XY \cdot \left(W + 1 - z \frac{X + 1}{2}\right) \cdot \left(H + 1 - z \frac{Y + 1}{2}\right) \text{ dengan } z = w + h$$

2.6 Attentional Cascade

Attentional Cascade adalah sebuah *Cascade* dari banyak *classifier* yang dibuat untuk meningkatkan performa deteksi dengan secara radikal mengurangi waktu komputasi. Intinya *classifier* kecil yang telah di *boost* dapat dibuat lebih kecil dan efisien, yang dapat menolak banyak *sub-window* negatif selagi mendeteksi sebagian besar dari *sub-window* positif. *Classifier* yang lebih simpel digunakan untuk menolak mayoritas *sub-window* sebelum *classifier* yang lebih kompleks dipanggil untuk menurunkan tingkat *false positives*.

Tahapan pada *cascade* dibangun dengan melatih *classifier* menggunakan *Adaboost*. Dimulai dengan *strong classifier* 2 fitur, sebuah filter yang efektif dapat didapatkan dengan menyesuaikan nilai ambang *strong classifier* untuk meminimalisir *false negatives*. Nilai ambang awal *Adaboost*, $\frac{1}{2} \sum_{t=1}^T \alpha_t$, didesain untuk menghasilkan nilai *error* rendah pada data latihan. Nilai ambang yang lebih rendah menghasilkan jumlah deteksi yang lebih tinggi dan *false positives* yang lebih tinggi. Didasarkan dari performa yang dihitung menggunakan sebuah set latihan validasi, *classifier* dua fitur dapat disesuaikan untuk mendeteksi 100% dari objek dengan tingkat *false positives* 50%.

Performa deteksi dari *classifier* dua fitur tersebut jauh dari hasil yang dapat diterima. Namun *classifier* tersebut dapat secara signifikan mengurangi jumlah *sub-window* yang perlu diproses lebih lanjut dengan sedikit operasi:

1. Evaluasi fitur persegi (memerlukan antara 6 sampai 9 referensi *array* per fitur)
2. Mengkomputasi *weak classifier* untuk setiap fitur (memerlukan satu operasi ambang batas per fitur)
3. Mengkombinasi beberapa *weak classifier* (memerlukan satu operasi per fitur, sebuah penambahan, dan sebuah operasi ambang batas)

Sebuah *classifier* dua fitur memerlukan kira-kira 60 milidetik instruksi *croprocessor*. Sulit dibayangkan kalau filter simpel apapun dapat mencapai tingkat

penolakan yang lebih tinggi. Untuk perbandingan, memindai sebuah templat gambar memerlukan paling tidak 20 kali lebih banyak operasi per *sub window*.

Bentuk kurang lebih dari proses deteksi seperti sebuah *degenerate decision tree*, yang kita sebut sebuah *cascade*. Sebuah hasil tes positif dari *classifier* pertama memicu deteksi pada *classifier* kedua yang juga disesuaikan untuk memiliki tingkat deteksi yang tinggi. Hasil tes positif dari *classifier* kedua akan memicu *classifier* ketiga dan seterusnya. Hasil tes negatif pada tahap manapun akan langsung berujung ke ditolaknya *sub-window*.

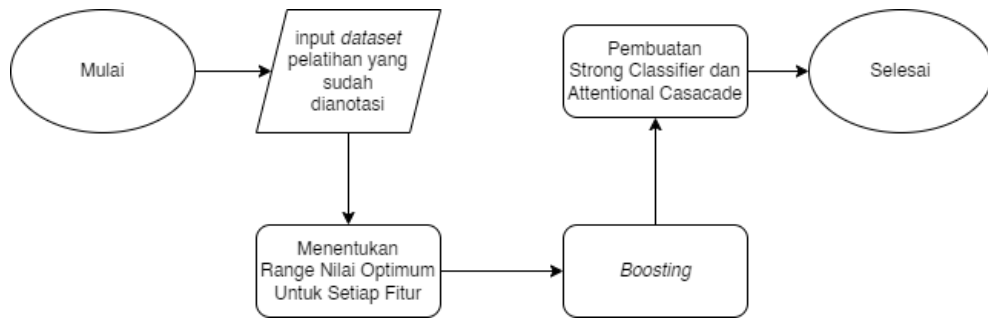
Struktur dari *cascade* merefleksikan fakta bahwa pada gambar apapun mayoritas *sub-window* pasti negatif. Oleh karena itu, *cascade* berusaha untuk sebanyaknya menolak *sub-window* negatif pada tahapan seawal mungkin. Sementara hasil positif akan memicu evaluasi pada setiap *classifier* dalam *cascade*, hal ini sangatlah langka.

Mirip seperti sebuah *decision tree*, *classifier* sebelumnya dilatih menggunakan contoh-contoh yang melewati tahapan-tahapan sebelumnya. Hasilnya, *classifier* kedua akan menerima tugas yang lebih sulit dari yang pertama. Contoh yang melewati tahapan pertama lebih “sulit” dari contoh biasanya. Contoh yang lebih sulit yang dihadapi oleh *classifier* yang lebih dalam mendorong seluruh lengkungan *receiver operating characteristic* (ROC) kebawah. Pada setiap rata-rata deteksi, *classifier* yang dalam secara bersamaan memiliki tingkat *false positive* yang lebih tinggi.

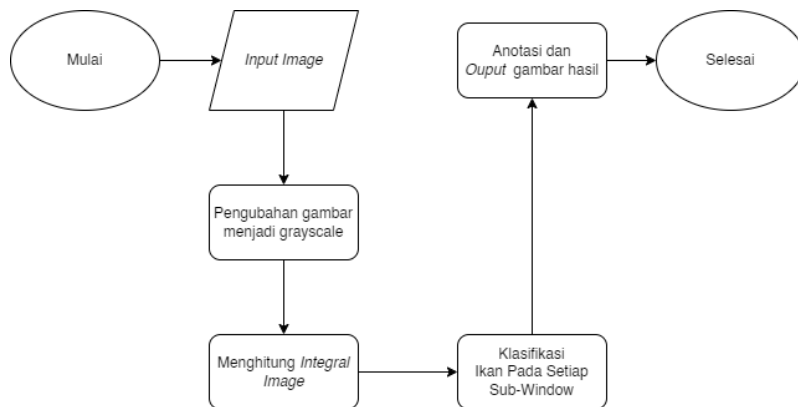
BAB 3

DESAIN MODEL

3.1 Flowchart Viola-Jones Object Detection



Gambar 3.1: Diagram alir untuk algoritma pelatihan klasifikasi objek Adaboost



Gambar 3.2: Diagram alir untuk algoritma pendeteksian objek Adaboost

3.2 Desain Sistem

Dalam proses pembuatan Object Detection ada 2 tahapan besar yaitu Training dan deteksi objek. Training adalah proses dimana Adaboost menentukan bobot dari setiap *weaklearn* dalam menentukan klasifikasi objek pada gambar dengan menggunakan Dataset yang sudah dianotasi. Pada tahapan ini Adaboost akan melakukan perhitungan bobot *weaklearn* satu-persatu dengan memperhitungkan tingkat akurasi *weaklearn* yang diperhitungkan sebelumnya, menggunakan gambar yang sulit diklasifikasi sebagai acuan. Setelah itu Adaboost akan membuat sebuah *strong classifier* untuk nantinya digunakan dalam deteksi objek.

Pada deteksi Objek, strong classifier akan digunakan untuk melakukan pendeteksian objek. Pertama input akan diubah menjadi grayscale untuk mempermudah perhitungan fitur nanti. Kemudian *integral image* akan dihitung untuk menghasilkan *sub-window* pada gambar yang lalu akan dicek menggunakan *strong classifier*. *Sub-window* yang ditemukan memiliki objek yang dicari akan dibuatkan *bounding box* pada kelilingnya. Setelah semua *sub-window* diperiksa, semua bounding box yang bertindihan akan digabungkan sebelum algoritma mengembalikan gambar hasil deteksi.

3.3 Training Strong Classifier

Pada Training ada 3 atau lebih tahapan yang perlu dijalankan untuk menghasilkan sebuah *strong classifier* pendeteksian objek, yaitu penginputan dataset pelatihan yang sudah dianotasi, penentuan nilai ambang optimum atau bobot untuk setiap fitur, dan pemilihan fitur oleh Adaboost untuk membuat *strong classifier*.

3.3.1 Input Dataset Pelatihan Yang Sudah Dianotasi

Dataset yang akan dipakai diambil dari **(Belum ditentukan antara membuat sendiri atau mencari di Internet)**. Dataset pelatihan positif adalah kumpulan gambar ikan yang dianotasi menggunakan *Bounding Box Annotation* yang lalu akan dirubah menjadi ukuran 72x41 pixel, dan warnanya menjadi *greyscale*, contoh positif akan dianotasikan sebagai “1”. Selain itu juga akan dipilih Dataset pelatihan negatif atau kumpulan gambar yang tidak terdapat ikan dengan jumlah yang sama dan perlakuan yang sama, contoh negatif akan dianotasikan sebagai “0”. Jumlah contoh positif dan negatif disamakan agar bobot sampel awal adalah 1. Dataset lalu dibagi menjadi 3 kelompok, yaitu untuk latihan, validasi, dan tes.

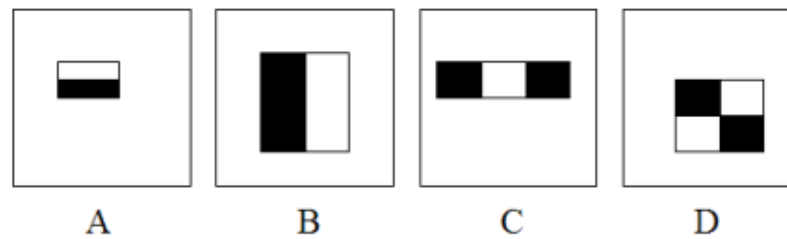


Gambar 3.3 Gambar asli - Grayscale – 72 x 41 pixel

3.3.2 Menentukan Range Nilai Optimum Untuk Setiap Fitur

Adaboost lalu akan melatih *weak learner* untuk mencari nilai ambang atau *threshold* optimalnya. Hal ini dilakukan dengan menjalankan *weak learner* dengan *dataset* pelatihan menggunakan sebuah rentang nilai untuk mencari nilai yang memberikan tingkat kesalahan klasifikasi terendah. Rentang nilai yang dites adalah nilai diantara -1 sampai 1 dengan nilai inkremental 1/255 untuk fitur 2 persegi dan -2 sampai 2 dengan nilai inkremental yang sama untuk fitur 4 persegi.

Setiap *weak learner* adalah sebuah *decision tree* dengan beberapa kelas sebagai daun, salah satu daun pasti adalah kelas “Tidak ada”. Yang menentukan iya atau tidak adalah hasil perhitungan sebuah Haar-Like Features yang hanyalah sebuah pengurangan dari intensitas piksel didalam wilayah putih oleh intensitas piksel didalam wilayah hitam, contoh Haar-Like Features dapat dilihat pada gambar 3.4.



Gambar 3.4: Haar-Like Features yang akan digunakan

Algoritma Adaboost lalu akan mencari nilai ambang terbaik untuk *weak learner* tersebut dengan mencari range yang paling optimal untuk setiap kelas.

3.3.3 Boosting

Setelah menentukan nilai ambang terbaik untuk setiap fitur, Adaboost lalu memilih akan memberikan bobot voting untuk setiap *weak learner* dengan melakukan *boosting* dimulai dari *weak learner* terbaik. Pertama Adaboost memilih *weak learner* terbaik dan menjalankannya untuk mengetes kelompok *dataset* latihan. Semua contoh pada dataset tahap ini awalnya akan dianggap memiliki bobot nilai yang sama dalam pemberian nilai bobot voting, akan tetapi pada akhir pemberian nilai bobot voting sebuah *weak learner* Adaboost akan mengingat contoh mana saja yang salah diklasifikasi oleh *weak learner* tersebut. Algoritma Adaboost lalu merubah bobot contoh latihan yang sulit diklasifikasi oleh *weak learner* pertama, hal ini dengan tujuan agar *weak learner* berikutnya yang berhasil mengklasifikasi contoh “sulit” tersebut akan mendapat bobot yang lebih tinggi. Adaboost akan menjalankan pemberian nilai bobot voting yang sama ke semua *weak learner* dengan bobot contoh yang sudah dimodifikasi oleh hasil tes *weak learner* sebelumnya.

Pada akhir dari setiap sesi *boosting*, Adaboost akan melakukan *cross-validation* menggunakan contoh dari kelompok *dataset* validasi untuk menentukan apakah *boosting* akan diulang atau dihentikan karena sudah mencapai hasil paling optimal. Dalam *cross-validation*, *Strong Classifier* hasil dari sesi itu akan dites performa akurasi dan presisinya. Bila ditemukan bahwa presisi hasil *boosting* mulai

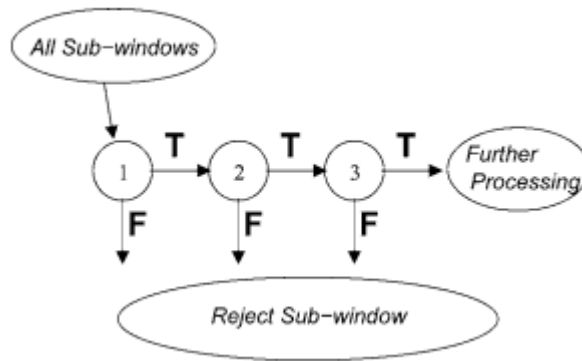
menurun dari hasil *boosting* sebelumnya maka iterasi *boosting* akan dihentikan. Bila tidak, maka iterasi *Boosting* akan diulang lagi.

Selain untuk menentukan bobot voting dari setiap *weak learner*, *Boosting* juga bertujuan untuk mengurutkan *weak learner* dari yang paling akurat ke yang tidak. Hal ini akan sangat berpengaruh dalam pembentukan *Attentional Cascade* di tahapan berikutnya.

3.3.4 Pembuatan Strong Classifier dan Attentional Cascade

Setelah tahap *boosting* berakhir, Adaboost akan membuat *Final Strong Classifier* dan *Attentional Cascade* dengan memilih *weak classifier* terbaik dan membuang *weak classifier* yang kurang berguna dengan melihat bobot individu setiap *weak classifier*. Bila Adaboost merasa sebuah *weak learner* kurang diskriminatif dalam mengklasifikasi maka fitur tersebut tidak akan digunakan dalam pembuatan *Strong Classifier*.

Adaboost pun akan mulai membuat tahapan *Attentional Cascade* dari *weak learner* yang tersisa. Adaboost akan menambahkan *weak learner* satu persatu dengan mempertimbangkan jumlah *false positives* yang dihasilkan dalam pengujian menggunakan kelompok *dataset* tes. Bila tingkat *false positives* sudah mencapai nilai ambang yang ditentukan pada sebuah tahapan *cascade*, maka Adaboost akan lanjut membuat tahapan berikutnya. Proses ini berlanjut hingga akurasi deteksi dengan tingkat *false positives* yang diinginkan dicapai, dalam hal ini sedekat mungkin dengan tingkat akurasi 100%, atau bila Adaboost tidak dapat menambahkan tahapan lagi kedalam *Cascade* karena keterbatasan jumlah *weak learner* yang tersedia.



Gambar 3.5: Workflow dari Attentional Cascade

Strong Classifier yang telah melewati tahapan inilah yang akan menjadi *Final Strong Classifier* yang nantinya akan dipakai pada tahapan deteksi yang sebenarnya.

3.3.5 Input Gambar, Grayscale dan Penghitungan Integral Image

Untuk klasifikasi sebenarnya, gambar *input* akan diproses terlebih dahulu. Gambar awalnya akan melalui proses *pre-processing* dan dirubah kedalam warna *grayscale* untuk mempermudah penghitungan. Setelah tu sebuah matriks sebesar resolusi gambar akan dibuat untuk penghitungan *Integral Image* yang nantinya akan mempercepat proses penghitungan fitur.

Untuk pembuatan *Integral Image* pertama perlu dicari nilai intensitas cahaya baris pertama dan kolom pertama dari gambar. Nilai pada matriks *Integral Image* adalah median dari nilai RGB pada piksel tersebut, namun dikarenakan warna gambar sudah dirubah menjadi *grayscale* maka nilai pada setiap piksel hanya akan berupa bilangan bulat dikisaran 0 sampai 255. Nilai pada kolom pertama hanyalah penjumlahan nilai piksel dari piksel (0, 0) sampai ke piksel (0, j), sementara nilai pada baris pertama juga hanya penjumlahan nilai piksel dari (0, 0) ke (i, 0). Nilai-nilai piksel lainnya lalu dapat dihitung menggunakan rumus: *integral image* (i, j)

$$\begin{aligned}
 &= \text{integral image } (i - 1, j) + \text{integral image } (i, j \\
 &\quad - 1) - \text{integral image } (i - 1, j - 1) + \text{nilai dari piksel } (i, j)
 \end{aligned}$$

Atau jumlah semua nilai piksel dari (0, 0) sampai ke (i, j). Pembuatan *Integral Image* selengkapnya dapat dilihat pada sub-bab 2.2 *Integral Image*. Pembuatan *Integral Image* nantinya akan mempercepat proses penghitungan fitur pada setiap *sub-window*.

3.3.6 Klasifikasi Ikan Pada Setiap Sub-Window

Sebuah *Sub-Window* sebesar 72x41 piksel akan dibuat untuk proses klasifikasi. Ukuran minimum dari *Sub-Window* mengikuti ukuran gambar pada proses pelatihan. Pendeteksian dilakukan dengan menggunakan *Strong Classifier* yang dibuat pada sub-bab 3.3.4. *Sub-window* akan terus bergerak ke kanan sampai *Sub-window* tidak dapat dibuat lagi. Bila demikian proses akan mulai lagi dari kiri dengan *Sub-Window* diturunkan satu piksel kebawah. Proses ini dinamakan *Sliding-Window*.

Bila *Strong Classifier* sudah selesai mendeteksi menggunakan *Sub-Window* dengan ukuran 72x41 piksel, maka *Strong Classifier* akan diperbesar dengan faktor 1.25 menjadi 90x51 piksel dan proses dimulai lagi dari kiri atas. Selain skala *Sub-Window*, nilai ambang dari tiap *weak learner*, ukuran fitur dan lokasi fitur juga akan disesuaikan dengan ukuran *sub-window*. Proses *rescale* ini akan dilakukan sampai *sub-window* tidak dapat diperbesar lagi.

Setelah semua *Sub-Window* dicek, semua *Sub-Window* yang memiliki objek ikan yang dicari akan dianotasi dengan menggambarkan *bounding box* pada gambar hasil. Untuk kelas ikan apa yang terdeteksi akan ditentukan dari hasil voting seluruh weaklearn, bobot votingnya adalah nilai yang sudah ditentukan pada fase boosting.

3.3.6.1 Contoh Pendeteksian Sub-Window

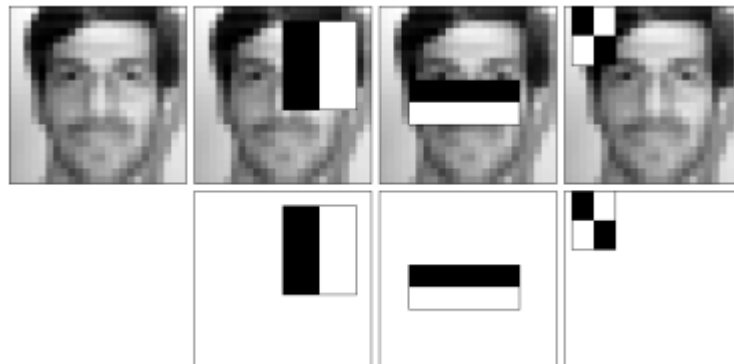
Weber (2005) melakukan percobaan pendeteksian muka dengan Adaboost dengan mengikuti jurnal Viola Jones dari 2004. Pada *strong classifier* miliknya, Weber menjelaskan bahwa Adaboost telah memilih sebuah fitur untuk tahap pertama, lima fitur pada tahap kedua, sepuluh fitur pada tahap ketiga, lima puluh fitur pada tahap empat, dua ratus fitur pada tahap lima, seribu fitur pada tahap enam

dan pada tahap tujuh, atau tahap terakhir, *Adaboost* memilih dua ribu fitur. Konfigurasi *cascade* Weber bisa dilihat pada tabel 3.1.

	Filter 1	Filter 2	Filter 3	Filter 4	Filter 5	Filter 6	Filter 7
Number of features	1	5	10	50	200	1000	2000
Threshold increase	0	0.01	0.01	0.02	0.02	0.03	0.03
λ	0.55	0.75	0.75	0.75	0.8	0.8	0.75

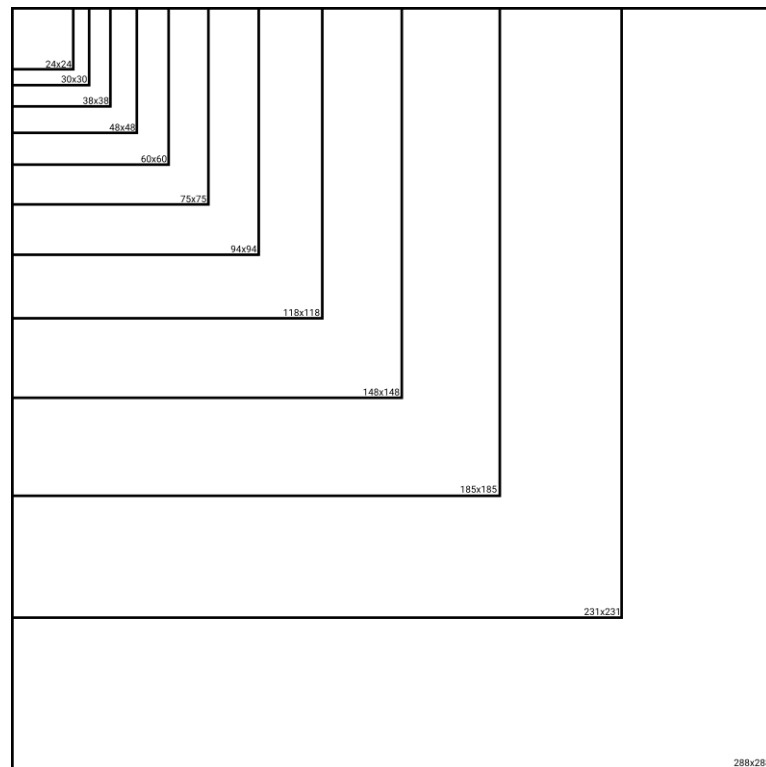
Tabel 3.1 konfigurasi cascade Weber (2004)

Untuk fitur yang dipilih, Weber menunjukkan tiga fitur pertama yang dipilih oleh *Adaboost* setelah proses *Boosting*. Fitur pertama mendeteksi fakta bahwa rambut seseorang biasanya lebih gelap dari muka, fitur kedua mendeteksi fakta bahwa intensitas cahaya disekitar mata jauh lebih gelap dibanding pipi bagian atas, dan fitur yang ketiga mendasarkan diri ke fakta bahwa warna latar biasanya lebih terang dari warna rambut. Fitur-fitur yang dipilih pada percobaan Weber bisa dilihat pada gambar 3.6.



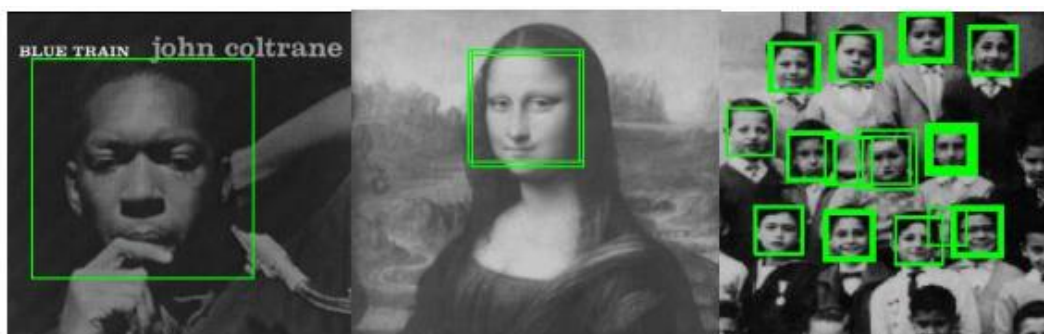
Gambar 3.6 tiga fitur pertama, kedua dan ketiga yang dipilih *Adaboost* pada percobaan Weber

Dalam percobaan Weber, dia menggunakan perbesaran skala *sub-window* seperti yang digunakan Viola Jones yaitu dengan memfaktorkan *Sub-Window*, fitur dan ambang nilai setiap fitur dengan 1.25 sampai *Sub-Window* lebih besar dari ukuran gambar *input*. Gambar 3.7 menunjukkan gambaran skala *Sub-Window* yang dipakai oleh model *Adaboost* milik Viola Jones dengan input gambar 384x288 dimulai dari *Sub-Window* berukuran 24x24 piksel.



Gambar 3.7 ukuran Sub-Window yang digunakan Viola Jones

Pada model percobaan Weber, ketika *strong classifier* menyatakan bahwa sebuah *sub-window* memiliki muka, maka program akan menggambar *bounding box* hijau pada *output image* yang menandakan keberadaan muka. Hasil akhir dari deteksi Weber bisa dilihat pada gambar 3.8.



Gambar 3.8 anotasi bounding-box dari deteksi muka Weber

DAFTAR PUSTAKA

- Al-Amri, C. F. 2020. Rancang Bangun Fish Counter untuk Menghitung Bibit Ikan Lele. Skripsi. Universitas Islam Indonesia, Yogyakarta.
- Diansari, V. R, Arini, E, Elfitasari, T. 2013. Pengaruh Kepadatan Yang Berbeda Terhadap Kelulushidupan Dan Pertumbuhan Ikan Nila (*Oreochromis niloticus*) Pada Sistem Resirkulasi Dengan Filter Zeolit. Jurnal. Universitas Diponogoro, Semarang.
- Rudydi, M. I. 2019. Perancangan Mesin Penghitung Benih Ikan Otomatis untuk Membantu Kinerja Peternak Ikan. Artikel Ilmiah. Universitas Andalas, Padang.
- Viola, P, Jones, M. 2001. Robust Real-time Object Detection. Vancouver
- Freud, Y, Schapire, R. E. 1995. A Desicion.Theoretic Generalization of On-Line Learning and an Application to Boosting. New York
- Weber, G. 2006. Generic Object Detection using AdaBoost. University of California, Santa Cruz
- Ho, W. T, Lim, H. W, and Tay, Y, H. 2009. Two-stage License Plate Detection using Gentle Adaboost and SIFT-SVM. Universiti Tunku Abdul Rahman, Selangor.
- Guo, L. Ge, P,S. Zhang, M,H. Li, L,H. Zhao,Y,B. 2012. Pedestrian detection for intelligent transportation systems combining AdaBoost algorithm and support vector machine. Dalian Nationalities University, Dalian.
- Amit, Y. Felzenszwalb, P. Girshick, R. 2020. Object Detection.
- Simard, P.Y. Bottou, L. Haffner, P. and Cun, Y,L. Boxlets: a fast convolution algorithm for signal processing and neural networks.

Freund, Y. Schapire, R.E. 1996. Experiments with a New Boosting Algorithm.
AT&T Research, Murray Hill.

Quinlan, J.R. 1993. C4.5: Programs For Machine Learning. Morgan Kaufmann,
San Mateo.