KLASIFIKASI IKAN MENGGUNAKAN VIOLA-JONES OBJECT DETECTION FRAMEWORK

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

2023

LEMBAR PERSETUJUAN

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

Nama	: Nehemiah Austen Pison
No. Registrasi	: 1313619021
Program Studi	: Ilmu Komputer
Judul	: Klasifikasi Ikan Dengan Menggunakan Viola-Jones Object Detection Framework

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

Menyetujui,

Dosen Pembimbing I	Dosen Pembimbing II
Muhammad Eka Suryana, M.Kom	**Med Irzal, M.Kom**
NIP. 19770615 200312 1 001	NIP. 19851223 201212 1 002

Mengetahui,
Koordinator Program Studi Ilmu Komputer

Dr. Ria Arafiyah, M.Si
NIP. 19751121 200501 2 004

KATA PENGANTAR

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

Dalam menyelesaikan proposal ini, penulis selalu mendapat bantuan dari orang di sekitar penulis baik dalam bentuk bimbingan dalam mengerjakan proposal ini maupun dorongan semangat dalam pengerjaan. Oleh maka dari itu, penulis ingin menyampaikan terima kasih yang sebesar-besarnya kepada:

1. Yth. Para petinggi di lingkingkungan FMIPA Universitas Negeri Jakarta
2. Yth. Ibu Dr. Ria Arafiyah, M.Si selaku Koordinator Program Studi Ilmu Komputer.
3. Yth. Bapak Muhammad Eka Suryana,M.Kom selaku Dosen Pembimbing I yang telah membimbing dalam pengerjaan proposal skripsi ini
4. Yth. Bapak 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 djadikan sebagai pembelajaran serta dapat membangun penulis agar lebih baik lagi dalam mengerjakan tugas-tugas dan permasalahan yang ada kedepannya.

Akhir kata, penulis berharap proposal skripsi ini dapat bermanfaat bagi semua pihak baik itu bagi FMIPA Universitas Negeri Jakarta, teman-teman dari progran studi Ilmu Komputer Universitas Negeri Jakarta dan para pembaca sekalian.

Semoga Tuhan Yang Maha Esa senantiasa membalas kebaikan semua pihak yang telah membantu penulis dalam menyelesaikan proposal ini.

Jakarta, 2 Juni 2023

Nehemiah Austen Pison

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I

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 melaikan untuk hiasan, seperti di dalam kolam maupun akuarium.

Ikan bisa didapatkan melalui dua sumber, dengan menangkap ikan dari habitat aslinya dan dengan melalukan proses pembudidayaan. Pembudidayaan ikan memerlukan banyak infrastruktur pendukung yang tidaklah murah seperti lahan, konstruksi tambak atau kolam yang memadai, dan pakan dalam jumlah yang cukup besar. Walaupun cukup mahal, budidaya ikan dapat menghasilkan keuntungan yang besar. Ikan budidaya contohnya bisa dijual dalam keadan hidup atau segar ke wilayah-wilayah yang jauh dari habitat asli ikan, hal yang sulit dilakukan dengan ikan hasil tangkapan liar. Budidaya ikan juga dapat mengurangi permintaan pasar untuk ikan secara besar, 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 sangat besar di Indonesia, hal ini menyebabkan besarnya nilai pasar dari industri ini. Kementrian Kelautan dan Perikanan Indonesia mencatat bahwa produksi ikan budidaya di Indonesia mencapai 14 juta ton pada tahun 2021 dengan nilai sebesar Rp. 196.000.000.000.000,- trilliun rupiah (KKP, 2021).

Budidaya ikan di Indonesia memiliki problem yang lumayan besar yaitu diperlukannya usaha yang besar untuk menghitung dan mengawasi jumlah ikan yang dibudidayakan. Dalam penghitungan bibit pada proses jual beli contohnya, dalam menghitungan bibit lele para pedagang masih menghitung ikan dengan cara manual (Al-Amri 2020). Bibit ikan dipindahkan satu persatu atau ditimbang sesuai berat untuk mendapatkan jumlah ikan. Cara-cara ini anatara sangat tidak efisien atau kurang akurat. Dalam metode penghitungan contohnya, ikan dihitung satu per satu dengan tangan atau dengan bantuan sendok atau centong, yang memungkinkan

penghitung untuk mengambil ikan dengan jumlah tertentu. Metode ini bisa memakai 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 bibit ikan dimasukan ke dalam suatu wadah dan lalu beratnya dihitung menggunakan timbangan. Berat hasil penimbangan lalu bisa dijadikan acuan kira-kira jumlah ikan yang terdapat di dalam wadah. Metode ini cepat dan cukup efisien dalam menghitung ikan dalam jumlah yang sangat besar. Namun jumlah bibit ikan tidaklah akurat dan hanya berupa perkiraan belaka.

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 di sisi 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* di mana ikan akan terdeteksi ketika melewati pipa oleh inframerah dan photodioda. Alat tersebut dapat mendeteksi ikan dengan kecepatan 58 ms 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 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), 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 menunjukan 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 mahluk 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, di mana 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 di atas 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 di atas, perumusan masalah dalam penelitian ini adalah '**Bagaimana cara mengklasifikasi ikan menggunakan metode *Viola-Jones Object Detection Framework*?**'.

1.3 Batasan 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 tiga kelas genus ikan, Abudefduf, Amphiprion, Chaetodon, dan satu kelas negatif.

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. Dan menambahkan 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 II

KAJIAN PUSTAKA

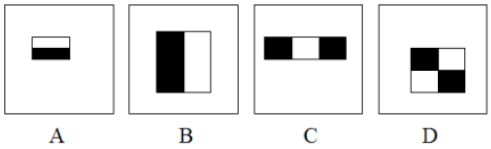
2.1 Pengertian Klasifikasi Objek

Fungsi dari klasifikasi objek adalah untuk memberikan deskripsi label ke sebuah segmentasi objek. Bila dilihat dari representasi fitur objek, ini dapat dicapai dengan melihat tanda-tanda keberadaan fitur yang mengindikasikan kelas dari objek. Hal ini umumnya dicapai dengan menentukan batasan diantara kelas-kelas yang direpresentasikan didalam *training set* yang sudah diberi label. Pelatihan dilakukan dengan secara data yang sudah dilabeli secara manual atau data yang belum dilabeli. Pelatihan otomatis umumnya menekankan distribusi dari setiap kelas, dan melabeli setiap contoh dengan sesuai, Namun, isu yang umum didalam kedua kasus adalah pilihan dari fitur-fitur yang akan digunakan untuk klasifikasi objek (Renno et al, 2007).

2.2 *Viola Jones Object Detection Framework*

Paul Viola dan Michael, J, Jones mempublikasikan sebuah makalah ilmiah dengan judul "*Robust Real-Time Face Detection*". Makalah tersebut mendeskripsikan sebuah framework pendeteksian wajah yang dapat memproses gambar secara cepat dengan tingkat akurasi yang tinggi. Ada tiga kontribusi penting dari makalah tersebut: Pertama adalah pengenalan sebuah representasi gambar baru yang dipanggil *Integral Image* yang memungkinkan penghitungan fitur yang dipakai oleh detektor dilakukan dengan cepat. Yang kedua adalah sebuah classifier yang efisien dan sederhana, yang dibuat menggunakan algoritma pembelajaran *Adaboost* (Freund et al, 1995) untuk memilih sejumlah fitur-fitur kritis dari fitur-fitur potensial. Yang ketiga adalah sebuah metode untuk menggabungkan fitur-fitur tersebut dalam sebuah bentuk *cascade* yang memungkinakan algoritma untuk memfokuskan deteksi di area-area yang memiliki potensial saja.

2.2.1 *Features*

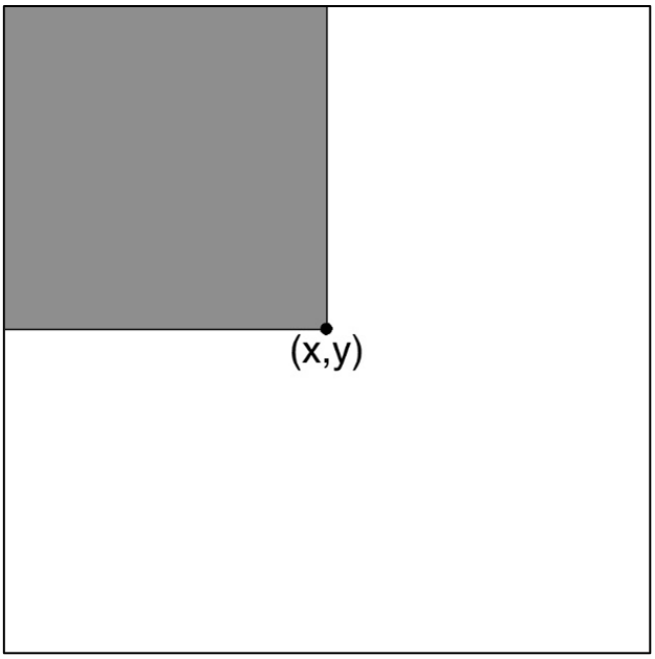


Gambar 2.1: Beberapa *Haar-like features* yang digunakan framework Viola-Jones.

Ada banyak alasan dimana penggunaan fitur lebih baik daripada nilai piksel secara langsung. Alasan paling umum adalah fitur dapat berfungsi untuk meng-*encode ad-hoc domain knowledge* yang sulit dipelajari dengan jumlah data pelatihan yang terbatas. Untuk sistem *framework Viola-Jones* ada alasan besar lainnya untuk penggunaan fitur yaitu sistem berbasis fitur beroperasi lebih cepat daripada sistem yang berbasis nilai piksel.

Fitur sederhana yang digunakan mirip dengan *Haar Basis Function* yang digunakan Papageorgiou et al. (1998). Lebih tepatnya, tiga jenis fitur. Nilai dari sebuah *fitur dua persegi* adalah perbedaaan diantara jumlah nilai piksel didalam dua area persegi. Area-area tersebut memiliki ukuran dan bentuk yang sama, dan juga bersebelahan secara horizontal dan vertikal. Subuah *fitur tiga persegi* menghitung jumlah piksel dua area persegi di bagian luar dikurangi dengan jumlah nilai piksel persegi yang ada ditengah keduanya. Yang terakhir *fitur empat persegi* menghitung perbedaan nilai dari dua pasang persegi diagonal.

2.2.2 *Integral Image*



Gambar 2.2: Sebuah nilai *Integral Image* (x, y) dan area yang diwakilinyak

Fitur-fitur persegi dapat dihitung 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 dan nilai x, y itu sendiri:

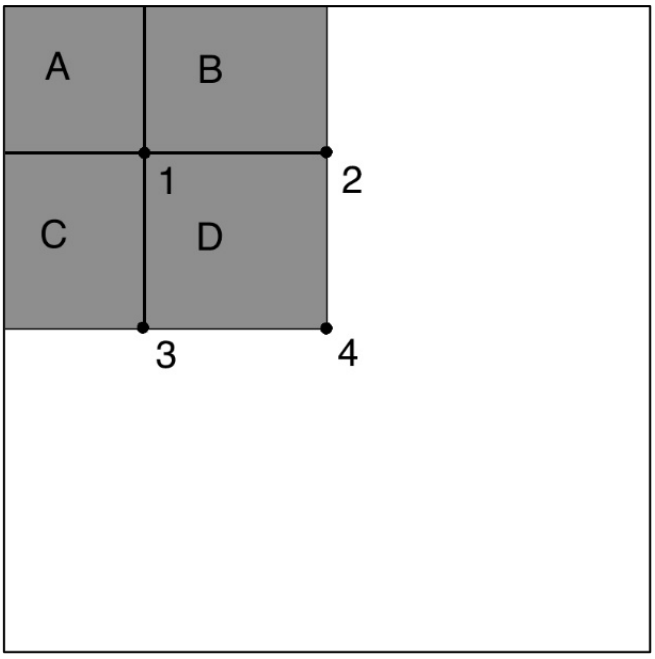
$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i\left(x', y'\right), \tag{2.1}$$

Dimana $ii(x, y)$ adalah *integral image* dan $i(x, y)$ adalah nilai piksel dari gambar aslinya. Menggunakan kedua pengulangan berikut ini:

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{2.2}$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \tag{2.3}$$

(Dimana *s(x, y)* adalah nilai kumulatif dari baris, *s(x, -1) = 0*, dan *ii(-1, y) = 0*) *Integral Image* dari sebuah gambar dapat dihitung dalam sekali jalan.



Gambar 2.3: 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.

Menggunakan *integral image*, semua jumlah nilai pada persegi dapat dihitung didalam 4 referensi *array*. Jelas perbedaan diantara kedua jumlah nilai-nilai persegi dapat dihitung dengan delapan referensi. Karena peresegi dua fitur yang didefinisikan diatas melibatkan juga nilai persegi disebelahnya, mereka dapat dikomputasi dengan enam referensi *array*, delapan referensi bilamana ia adalah persegi tiga fitur, dan sembilan referensi untuk persegi empat fitur.

Kecepatan yang didapat dari penghitungan *Integral Image* ini dapat dijustifikasi bila kita membandingkannya dengan perhitungan manual. Sebagai contoh, misalkan kita sedang mencari jumlah total 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 untuk 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 empat kali untuk menghitung intensitas cahaya dalam area tersebut.

2.2.3 *Adaboost*

Algorithm 1 Algoritma Adaboost

1: Diberikan contoh gambar $(x_1, y_1), ..., (x_n, y_n)$ dimana $y_i = 0$ untuk contoh negatif dan $y_i = 1$ untuk contoh positif
2: Inisialisasi bobot $w_{1,i} = \frac{1}{2_m}, \frac{1}{2l}$ untuk $y_i = 0$ dan $y_i = 1$, dimana m adalah angka negatif dan l adalah angka positif
3: Untuk $t = 1, ..., T$:

1. Normalisasi bobot, $w_{t,1} \leftarrow \frac{w_{t,1}}{\sum_{j=1}^{n} w_{t,j}}$
2. Pilih $weakclassifier$ terbaik dengan melihat error yang telah diberi bobot

$$\in_t = min_{f,p,\theta} \sum_i w_i |h(x_i, f, p, \theta) - y_i|. \tag{2.4}$$

3. Definisikan $h_t(x) = h(x, f_t, p_t, \theta_t)$ dimana f_t, p_t, dan θ_t adalah *minimizer* dari $\in_t$.
4. Perbarui bobot:

$$w_t + 1, i = w_t, i\beta t1 - e_i \tag{2.5}$$

dimana $e_i = 0$ bila contoh x_i diklasifikasi secara benar, selainnya $e_i = 1$, dan $\beta_t = \frac{\in_t}{1-\in_t}$

4: *strong classifier* akhirnya adalah:

$$C(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{selain itu} \end{cases} \tag{2.6}$$

dimana α_t = log $\frac{1}{\beta_t}$

Dalam *framework Viola-Jones* sebuah varian dari *Adaboost* digunakan untuk memilih fitur dan juga untuk melatih *classifier*. Didalam bentuk aslinya, algoritma pembelajaran *Adaboost* digunakan untuk mem-*boost* performa klasifikasi dari algoritma pembelajaran sederhana. *Adaboost* melakukan ini dengan menggabungkan sekumpulan *classifier* lemah untuk membuat sebuah *classifier* kuat. Didalam istilah *Boosting*, *classifier* lemah disebut juga dengan *weak learner*. Sebagai contoh, algoritma pembelajaran *perceptron* mencari dari sekelompok *perceptron* dan mengambil *perceptron* dengan tingkat kesalahan klasifikasi terendah. Algoritma pembelajaran disebut lemah karena kita tidak berharap *classifier* terbaik untuk mengklasifikasi data dengan benar. Nyatanya, *perceptron* terbaik mungkin

hanya memiliki tingkat akurasi 51%. Agar *weak learner* dapat di-*boost*, ia akan dipanggil untuk menyelesaikan sederet problem pembelajaran. Setelah satu ronde pembelajaran selesai, contoh pembelajarannya akan dibobot ulang untuk menekankan problem yang salah diklasifikasi oleh *weak learner* sebelumnya. Bentuk *final strong classifier* adalah *perceptron*, sebuah kombinasi *weak learner* berbobot yang diikuti oleh *threshold*.

Dalam *Framework Viola Jones*, *Adaboost* yang digunakan hanya bekerja untuk klasifikasi biner saja. Oleh karena itu perlu adanya modifikasi *Adaboost* yang dapat melakukan klasifikasi *Multi-class*. Yang pertama adalah anotasi contoh gambar $(x_1, y_1), ..., (x_n, y_n)$ dimana $y_i \in Y = 1, ..., k$. Maka dari itu distribusi (D) nilai bobot (w) akan berubah sesuai dengan jumlah contoh (N): $w_i^1 = D(i)$ for $i = 1, ..., N$.

2.2.4 *Weaklearn*

Algorithm 2 Metode Pembuatan *Decision Tree*

1: Anotasi semua dataset sesuai kelasnya. *Decision Tree* lalu akan dimulai dari akar
2: Pilih atribut terbaik untuk melakukan *split* dengan melakukan *Information Gain* (IG):

$$IG(S, A) = Entropi(S) - sum_v \frac{S_v}{S} x Entropi(S_v) \tag{2.7}$$

dimana:

- S: kumpulan sampel pada *node* sekarang
- A: atribut yang digunakan untuk *split*
- v: Sebuah nilai atribut A yang membagi S menjadi turunan S_v
- Entropi(S) = $-sum_c p_c log_2(p_c)$: Ukuran kemurnian pada kumpulan contoh dengan target label. Dimana p_c adalah proporsi contoh S yang ada pada kelas c.

3: *split* pohon ke *node* turunan sesuai atribut yang dipilih
4: tentukan apabila seluruh contoh sudah jatuh ke kelas yang benar, bila tidak maka ulangi langkah 2 dan 3. Hal ini dilakukan dengan memvalidasi pohon yang sudah dibuat

Framework Viola Jones menggunakan sebuah *weaklearn* yang bernama *Decision Stump*, atau sebuah *Decision Tree* yang hanya memiliki dua daun kelas saja. *Decision Tree* sendiri mampu digunakan untuk permasalahan *multi-class*.

Decision Tree menghasilkan sebuah classifier didalam bentuk sebuah pohon pilihan, sebuah struktur yang berbentuk:

- Sebuah daun, mengindikasi 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.

2.2.5 *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 mayoritas *sub-window* negatif dan mendeteksi sebagian besar dari *sub-window* positif. *Classifier* yang lebih sederhana digunakan untuk menolak mayoritas *sub-window* sebelum *classifier* yang lebih kompleks dipanggil untuk menurunkan tingkat *false positives*.

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 dari setiap *classifier* dalam *cascade*, hal ini sangatlah langka.

Layaknya sebuah *Decision Tree*, sebuah *classifier* dilatih menggunakan contoh-contoh yang telah berhasil melewati tahap sebelumnya. Oleh karenanya, *classifier* pada tahap kedua menghadapi tantangan yang jauh lebih sulit daripada yang pertama. Contoh yang semakin sulit yang dihadapi *classifier* di tahap-tahap yang semakin jauh menekan seluruh *receiver operating characteristic* (ROC) kebawah.

Algorithm 3 Algoritma Pelatihan Untuk Pembuatan *Cascaded Detector*

1: Pengguna memilih nilai dari *f*, nilai maksimum *false positive* pada setiap tahap yang dapat diterima, dan *d*, nilai minimum *detection rate* per tahap yang dapat diterima
2: Pengguna memilih target keseluruhan *false positive rate*, F_{target}
3: *P* = kumpulan contoh positif
4: *N* = kumpulan contoh negatif
5: $F_0 = 1.0; D_0 - 1.0$
6: $i = 0$
7: $whileF_i > F_{target}$
$-i \leftarrow i + 1$
$-n_i = 0; F_i = F_i{-_1}$
-while $F_i > fxF_i{-_1}$

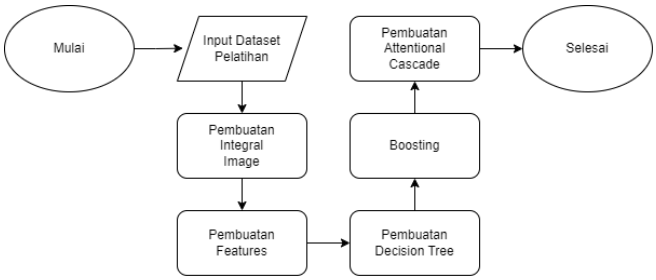
- $n_i \leftarrow n_i + 1$
- Gunakan *P* dan *N* untuk melatih sebuah *classifier* dengan fitur n_i menggunakan *Adaboost*
- Evaluasi *cascade classifier* dengan *set* validasi untuk menentukan F_i dan D_i.
- kurangi *threshold* untuk *classifier* ke-i sampai *cascade classifier* memiliki tingkat deteksi paling tidak $dxD_i{-_1}$ (hal ini juga akan mempengaruhi F_i)

$-N \leftarrow$ Ø
-if $F_i > F_target$ evaluasi *cascade detector* menggunakan *set* negatif dan masukan semua deteksi gagal ke *set* N

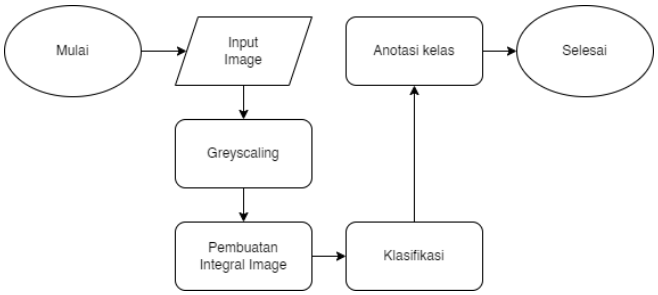
BAB III
METODOLOGI PENELITIAN

3.1 Tahapan Penelitian

Gambar *flowchart* berikut mengilustrasikan proses pelatihan dari dataset dan juga proses penggunaan yang sesungguhnya.



Gambar 3.1: Diagram alir untuk algoritma pelatihan klasifikasi objek



Gambar 3.2: Diagram alir untuk algoritma pendeteksian objek

3.2 Desain Sistem

Dalam proses pembuatan *classifier* perlu dilewati tahapan *training*. Tujuan *training* adalah untuk menciptakan suatu *strong classifier* yang nantinya dapat digunakan untuk melakukan klasifikasi yang sebenarnya. Pertama, gambar yang akan menjadi *set* latihan dianotasi sesuai kelasnya masing-masing, *set* latihan ini juga berisikan gambar-gambar yang tidak memiliki kelas yang benar, atau *false example*. Setelah itu gambar melalui proses *pre-processing* dan disesuaikan untuk mengoptimalkan proses pelatihan. Setiap gambar latihan nantinya akan diubah menjadi *integral image* untuk dapat diproses.

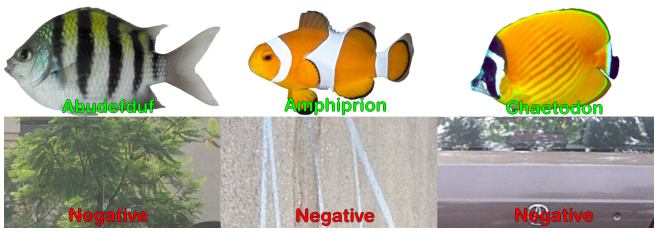
Pertama algoritma akan mengkonstruksi sebuah *decision tree* untuk setiap *weaklearn* dimana *branch* untuk setiap kelas akan ditentukan. Lalu *Adaboost* akan menentukan bobot dari setiap *weaklearn* dalam menentukan hasil akhir klasifikasi gambar. Terakhir, *Adaboost* akan membuat sebuah *strong classifier* dengan membuat sebuah *Attentional Cascade*

Dengan *strong classifier*, klasifikasi objek yang sesungguhnya bisa dilakukan. Pertama gambar yang akan dideteksi akan melalui *pre-processing*. Kemudian *integral image* akan dihitung untuk menghasilkan *sub-window* pada gambar. setiap sub-window 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 gambar hasil dikembalikan ke pengguna.

3.3 Training *Strong Classifier*

Pada *Training* ada tiga tahapan yang perlu dijalankan untuk menghasilkan sebuah *Strong Classifier*, yaitu penginputan *dataset* pelatihan yang sudah dianotasi, pembuatan *decision tree* untuk setiap fitur, *Boosting* dan pemilihan fitur oleh Adaboost untuk membuat *attentional cascade*.

3.3.1 *Input Dataset* Pelatihan



Gambar 3.3: Contoh gambar Abudefduf, Amphiprion, Chaetodon dan contoh gambar-gambar negatif

Dataset yang akan dipakai diambil dari fishR (Dapat dilihat di https://github.com/mekas/fishR) yang berisikan gambar ikan genus Abudefduf, Amphiprion dan Chaetodon. Selain itu *dataset* juga akan ditambahkan dari hasil penelitian lapangan secukupnya, dengan mempertimbangkan waktu komputasi pada tahap pelatihan. Untuk contoh pelatihan kelas-kelas ikan adalah gambar berukuran

72x41 piksel, dengan warna grayscale. Selain itu juga akan dipilih contoh pelatihan negatif atau kumpulan gambar yang tidak terdapat kelas ikan dari http://www.vision.caltech.edu/datasets/ dengan jumlah yang sama, resolusi sama dan perlakuan yang sama. Anotasi dilakukan dengan rumus $(x_1, y_1), ..., (x_n, y_n)$ dimana $y_i \in Y = 1, ..., k$, k disini adalah kelas-kelas yang ada dimana $y_i = 0$ direservasi untuk kelas negatif dan angka-angka setelahnya untuk kelas-kelas lainnya. *Dataset* ini lalu akan dibagi menjadi tiga yaitu *training dataset*, *validation dataset*, dan *test dataset* dengan jumlah yang sama. Nilai bobot pada tiap *dataset* akan di inisialisasi dengan rumus:

$$w_i^1 = D(i) \text{ for } i = 1, ..., N \tag{3.1}$$

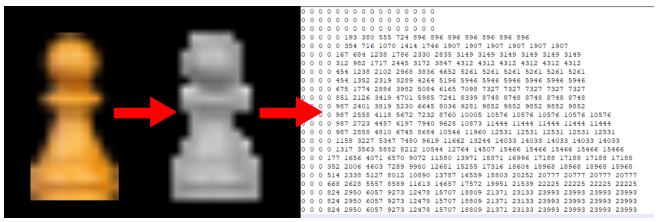
dimana w adalah bobot, D Distribusi dan N jumlah contoh pelatihan.

3.3.2 Pembuatan *Integral Image*

Untuk pembuatan *Integral Image*, pertama perlu dicari nilai piksel 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 *greyscale* 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:

$$\begin{aligned}\text{integral image } (i, j) = \text{integral image } (i-1, j) + \text{integral image } (i, j-1) - \\ \text{integral image } (i-1, j-1) + \text{nilai dari piksel } (i, j)\end{aligned} \tag{3.2}$$

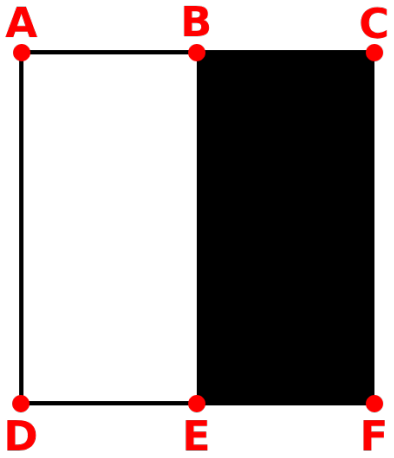
Atau jumlah semua nilai piksel dari $(0, 0)$ sampai ke (i, j). Pembuatan *integral image* nantinya akan digunakan oleh fitur sebagai *input*.



Gambar 3.4: Gambaran *integral image* dari gambar sebuah bidak catur.

3.3.3 Pembuatan *Features*

Fitur-fitur yang akan digunakan dalam klasifikasi dibuat berdasarkan *Haar-like Features* dan berisikan informasi lokasinya pada *sub-window* dan titik-titik *integral image*. Misalnya sebuah fitur dua persegi dengan persegi putih di sebelah kiri, fitur tersebut memerlukan nilai *integral image* dari pojok kiri atas persegi putih, pojok kiri bawah persegi putih, pojok kanan atas persegi putih, pojok kanan bawah persegi putih, , pojok kanan atas persegi hitam, dan pojok kanan bawah persegi hitam.



Gambar 3.5: Nilai yang diperlukan oleh fitur dua persegi, direpresentasi sebagai A, B, C, dan D

Keenam nilai tersebut dapat digunakan untuk mencari nilai fitur dengan rumus:

$$\begin{aligned}&\text{integral image E} - \text{integral image B} - \text{integral image D} + \text{integral image A}\\ &- \text{integral image F} - \text{integral image C} - \text{integral image E} + \text{integral image B}\end{aligned} \tag{3.3}$$

Pada tahap ini *integral image* yang dipilih akan dibuat untuk semua posibilitas lokasi yang ada dan ukuran yang ada. Hal ini dilakukan dengan membuat

fitur dimulai dari kiri atas gambar dengan ukuran 20 x 20 piksel untuk fitur dua persegi, dan 20 x 30 piksel untuk fitur tiga persegi, sampai pojok kanan bawah. Hal ini juga dilakukan untuk ukuran-ukuran yang lebih besar sampai fitur lebih besar dari *sub-window* yang digunakan dan tidak bisa muat lagi didalan *sub-window*. Ukuran paling kecil dari fitur akan disesuaikan pada saat penelitian bila ternyata fitur terlalu banyak dan memperlambat proses pelatihan.

3.3.4 Pembuatan *Decision Tree*

Setiap *weaklearner* adalah sebuah *decision tree* yang mengambil nilai dari *Haar-like features* sebagai variabel klasifikasi. *Haar-like features* dapat mengembalikan sebuah nilai perbandingan intensitas cahaya dari suatu area pada gambar dan mendeteksi keberadaan suatu *fitur* seperti perbedaan warna, garis, maupun perbedaan kontras pada gambar.

3.3.5 *Boosting*

Algoritma *Adaboost* akan mem-*boosting weaklearner* untuk menciptakan sebuah *strong classifier*. *Boosting* dilakukan dari *weaklearner* yang memiliki tingkat akurasi paling tinggi, dan dengan menggunakan *dataset* latihan.

Adaboost akan menjalankan *weaklearner* terbaik untuk mengklasifikasi seluruh contoh *dataset* latihan dan mencatat contoh yang gagal diklasifikasi oleh *weaklearner* tersebut. Nilai dari *weaklearner* tersebut dihitung dengan:

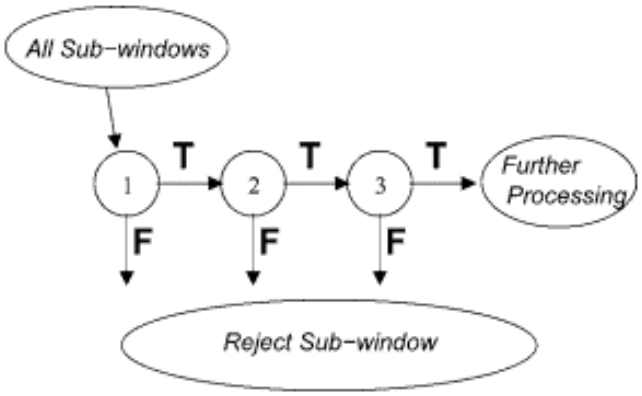
$$sum(N)(i = 1)w_i^t.1(h(x_i) \neq y_i) \tag{3.4}$$

dimana N adalah jumlah total contoh latihan, D_t adalah distribusi bobot pada iterasi t, w_i^t adalah nilai bobot contoh ke-i pada iterasi t, y_i adalah label yang benar untuk contoh latihan ke-i, dan $h(x_i)$ adalah hasil dari *weaklearn*. Hasilnya juga akan dicatat untuk urutan *Boosting* ronde berikutnya dan juga sebagai nilai voting pada klasifikasi yang sebenarnya. Nantinya bobot dari seluruh *dataset* latihan akan dihitung ulang dengan menaikan bobot dari contoh yang gagal diklasifikasi oleh *weaklearner* sebelumnya. Proses ini dilakukan untuk setiap *weaklearner*.

Ketika sebuah ronde *Boosting* sudah selesai dan seluruh *weaklearner* sudah diurutkan sesuai dengan nilai yang didapat dari ronde tersebut. *Adaboost* akan melakukan validasi tingkat akurasi *strong classifier* yang sudah dibuat menggunakan

dataset validasi. hal ini dilakukan dengan membandingkan tingkat presisi *strong classifer* yang dihasilkan ronde ini dengan *strong classifer* yang dihasilkan ronde sebelumnya. Bisa tingkat presisi ternyata semakin berkurang, maka *Boosting* akan dihentikan dan *strong classifier* menjadi *final strong classifier*.

3.3.6 Pembuatan *Attentional Cascade*



Gambar 3.6: *Workflow* dari *Attentional Cascade*

Sebelum pembuatan *attentional cascade*, *weaklearner* yang kurang diskriminatif akan dibuang. *Attentional cascade* akan dibuat dari *weaklearner* yang tersisa secara bertahap. Pertama, target *false positive* pada setiap *cascade* harus ditentukan oleh pengguna. *Framework VIola-Jones* menggunakan target *false positive* 50% untuk *cascade* pertama, dan *false positive* 80% untuk semua *cascade* setelahnya. Target *false posiive* ini akan disesuaikan saat penelitian lapangan, bila target akurasi tidak berhasil dicapai dengan target *false positive* sebelumnya. Pada setiap fase *cascade*, *weaklearner* akan ditambahkan satu-persatu. Setiap sebuah *weaklearner* ditambahkan, tes akan dilakukan dengan *dataset* tes. *Weaklearner* akan terus ditambahkan hingga *false positive rate* yang ditentukan untuk fase itu dicapai. Fase akan terus bertambah hingga akurasi sempurna dicapai, atau hingga *weaklearner* sudah habis.

3.4 Skenario Eksperimen dan Validasi

Tahapan ini adalah penggunakaan *classifier* yang sebenarnya dengan tujuan memvalidasi akurasi dari *classifier* tersebut. Gambar ikan yang akan dipakai dalam

proses validasi akan melalui beberapa langkah dalam tahap ini, yaitu: *Pre-processing* dalam bentuk *grayscaling*, Penghitungan *Integral Image*, dan deteksi yang sesungguhnya menggunakan *strong classsifier* yang telah dibuat dengan metode Sliding Window.

3.4.1 *Pre-processing* dan Penghitungan *Integral Image*

Untuk klasifikasi sebenarnya, gambar *input* akan diproses terlebih dahulu. Gambar awalnya akan melakui proses *pre-processing* dan dirubah kedalam warna *grayscale* untuk mempermudah penghitungan dengan bantuan *library OpenCV*. Setelah tu sebuah matriks sebesar resolusi gambar akan dibuat untuk penghitungan *Integral Image* yang nantinya akan mempercepat proses penghitungan fitur. Pembuatan *integral image* pada tahap ini sama persis dengan pembuatan pada tahap pelatihan.

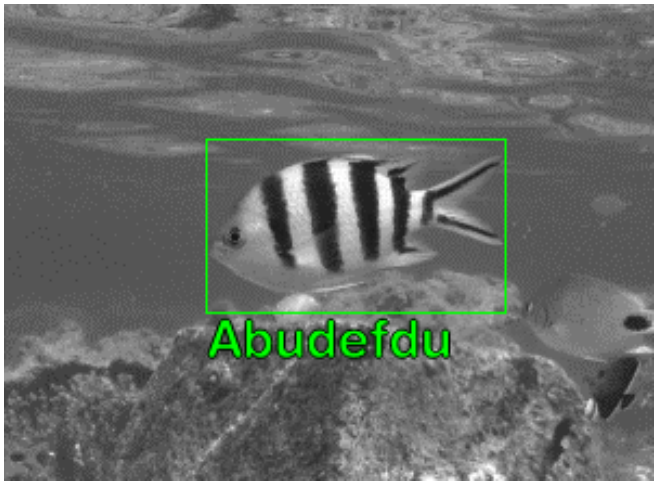
3.4.2 Klasifikasi

Pada tahap ini, gambar yang sudah berubah dalam bentuk *integral image* akan diklasifikasi menggunakan *strong classifier*. *Strong classifier* akan berjalan dari pojok kiri atas gambar, atau piksel pertama, mencoba untuk mengklasifikasi sebuah *sub-window* berukuran 72x41 piksel. Bila posisi tersebut sudah selesai diklasifikasi, terlepas hasilnya, *sub-window* akan bergerak ke kanan sebanyak satu piksel dan mengklasifikasi area baru tersebut. Hal ini akan terus berlanjut hingga *sub-window* bisa dibuat lagi ke kanan. Bilama demikian *sub-window* akan kembali ke ka kanan, namun kali ini diturunkan sebanyak satu piksel. Pergerakan *sliding window* ini bertujuan agar seluruh bagian dari gambar dapat terklasifikasi.

Gambar 3.7: Gambaran ukuran *sub-window* 72x41 piksel pada gambar 300x220

Jika *sub-window* dengan ukuran 72x42 piksel sudah mengklasifikasi seluruh bagian gambar hingga pojok kanan bawah, maka *sub-window* akan diperbesar dengan faktor 1.25 dan proses dimulai lagi dari pojok kiri atas. Selain itu ukuran dan lokasi dari *feature* akan disesuaikan untuk mangakomodasi *sub-window* yang sudah diperbesar. Hal ini dilakukan untuk mengklasifikasi objek yang memiliki ukuran berbeda didalam gambar.

3.4.3 Anotasi



Gambar 3.8: Gambaran *sub-window* yang berhasil mengklasifikasi ikan didalam gambar

Setelah semua *sub-window* sudah diklasifikasi dengan menggunakan *strong classifier*. Algoritma akan menggambarkan di lokasi *sub-window* yang positif terklasifikasi (Memiliki salah satu kelas ikan yang telah dilatih ke *strong classifier*), *bounding box* untuk menganotasi kelasnya. Pengguna lalu dapat menentukan dari hasil anotasi bilamana target akurasi sudah tercapai.

DAFTAR PUSTAKA

Al-Amri, C. F. (2020). “Rancangan Bangun Fish Counter Untuk Menghitung Bibit Ikan Lele”. In.