PROTOTYPE SYSTEM PENDETEKSI SPESIES IKAN MENGGUNAKAN VIOLA-JONES FEATURE EXTRACTION DAN BOOSTING BERBASIS DECISION TREE

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	: Prototype System Pendeteksi Spesies Ikan Menggunakan Viola-Jones Feature Extraction dan Boosting Berbasis Decision Tree

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. 19851223 201212 1 002	NIP. 19770615 200312 1 001

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 skripsi ini dengan baik. Adapaun jenis penelitian dengan judul *Prototype System Pendeteksi Spesies Ikan Menggunakan Viola-Jones Feature Extraction Dan Boosting Berbasis Decision Tree*.

Dalam menyelesaikan skripsi 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 lingkungan 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 skripsi ini
4. Yth. Bapak Med Irzal, M.Kom selaku Dosen Pembimbing II yang telah membimbing dalam pengerjaan skripsi ini.
5. Orangtua penulis yang telah memberi dukungan selama pengerjaan skripsi ini.
6. Teman-teman yang telah memberikan dukungan dan bantuan dalam pengerjaan skripsi ini.

Dalam penulisan skripsi ini penulis menyadari keterbatasan ilmu pengetahuan dan kemampuan penulis yang menyebabkan skripsi 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 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 membalaskan kebaikan semua pihak yang telah membantu penulis dalam menyelesaikan skripsi ini.

Jakarta, 12 Januari 2024

Nehemiah Austen Pison

ABSTRAK

Nehemiah Austen Pison, *Protptype System* Pendeteksi Spesies Ikan Menggunakan *Viola-Jones Feature Extraction* dan *Boosting* Berbasis *Decision Tree*. Skripsi, Program Studi Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Jakarta. Januari 2024.

Sektor perikanan di Indonesia merupakan pasar yang sangat besar dengan nilai mencapai Rp 2.400 triliun pada tahun 2022. Meskipun begitu, potensi perikanan di Indonesia belum sepenuhnya terealisasi. Dalam industri perikanan ada problematika dalam penghitungan dan klasifikasi ikan, ikan masih dihitung secara manual satu-persatu atau dengan menghitungnya dengan wadah. Metode-metode lain yang menggunakan mesin juga punya hambatannya, seperti spesifiknya ukuran ikan yang bisa dideteksi, Dalam penelitian ini kami mengusulkan metode klasifikasi ikan menggunakan metode *Viola-Jones Feature Extraction* dan *Boosting* Berbasis *Decision Tree* untuk menyelesaikan masalah perhitungan maupun klasifikasi ikan untuk semua ukuran. Dengan menggunakan klasifikasi komputer, pendeteksian tidak terhambat oleh ukuran, maupun bentuk ikan yang berbeda-beda. Metode ini juga memiliki potensi klasifikasi jenis ikan. Langkah pertama adalah dengan melatih *weak classifier* untuk mengklasifikasi beberapa fitur beberapa jenis ikan, diikuti dengan *Boosting*, dan pembuatan sebuah *cascade* untuk mempercepat proses klasifikasi. Klasifikasi ini diharapkan dapat membantu dalam proses penghitungan ikan, serta klasifikasi ikan bagi para petani ikan maupun umum.

Kata kunci: klasifikasi, Viola-Jones, Boosting, Decision Tree, Ikan.

ABSTRACT

Nehemiah Austen Pison, Prototype of Fish Species Detection System Using Viola-Jones Feature Extraction and Decision Tree Based Boosting, Computer Science Program, Faculty of Mathematics and Natural Sciences, State University of Jakarta. January 2024.

The fisheries sector in Indonesia represents a vast market with a value reaching IDR 2,400 trillion in 2022. However, the full potential of fisheries in Indonesia has not been fully realized. In the fisheries industry, there is an issue with fish counting and classification, where fish are still manually counted one by one or by using containers. Other methods that employ machines also have their limitations, such as the specificity of the fish sizes that can be detected. In this research, we propose a method for fish classification using the Viola-Jones Feature Extraction and Decision Tree-based Boosting methods to address the challenges of fish counting and classification for all sizes. By employing computerized classification, detection is not hindered by the size or different shapes of fish. This method also has the potential for classifying fish species. The first step involves training weak classifiers to classify various features of different fish species, followed by Boosting and the creation of a cascade to expedite the classification process. This classification is expected to assist in the fish counting process and the classification of fish for both fish farmers and the general public.

Keywords: Classification, Viola-Jones, Boosting, Decision Tree, Fish.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang Masalah

Sektor ikan di Indonesia adalah sebuah pasar yang sangat besar, nilainya mencapai Rp 2.400.000.000.000.000 rupiah pada tahun 2022. Meskipun pasar yang besar, potensi perikanan di Indonesia belum maksmimal, tutur Menteri Kelautan dan Perikanan RI Wahyu Sakti Trenggono. Namun produktivitas perikanan di Indonesia masih kalah dari Vietnam. Ia menegaskan bahwa potensial perikanan tersebut harus ditangkap dengan baik terutama di sektor budidaya yang saat ini berjalan dalam sekala kecil, berbeda dengan negara tetangga seperti Vietnam yang mengusai eksport ke berbagai negara (Sunartono 2023).

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. kedua cara tersebut antara sangat tidak efisien atau kurang akurat. Dalam metode penghitungan, 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 memakan waktu yang cukup lama bila ikan yang dihitung berjumlah besar, karena itu metode ini biasanya hanya digunakan dalam menghitung ikan dalam jumlah yang sedikit. Sementara itu metode penimbangan hanya menghasilkan jumlah perkiraan yang tidak selalu akurat, namun cepat. Dalam metode ini bibit ikan dimasukan ke dalam suatu wadah yang lalu ditimbang. 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 dkk., 2013), tetapi 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 Rusydi 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 menemukan bahwa kelandaian pipa 30° memberikan hasil paling akurat dalam mendeteksi ikan). Sementara pada alat Al-Amri, 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 dkk., 2004. Viola 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 dkk., 1996) adalah sebuah pendekatan *non-neural network* yang sering digunakan untuk mendefinisikan fitur dari objek yang ingin dideteksi (Weber 2006). Adaboost telah digunakan untuk deteksi berbagai objek seperti deteksi plat nomor kendaraan bermotor (Ho dkk., 2009), deteksi muka (Viola dkk., 2004), deteksi pesawat terbang (Freund dkk., 1996) 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.

Berdasarkan latar yang telah dijelaskan, penulis mengusulkan untuk mendeteksi jenis ikan dengan menggunakan metode *Viola-Jones Feature Extraction dan Boosting Berbasis Decision Tree*. *Viola-Jones Feature Extraction dan Boosting Berbasis Decision Tree* 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 secara akurat.

1.2 Rumusan Masalah

Dari uraian permasalahan di atas, perumusan masalah dalam penelitian ini adalah '**Bagaimana caranya mengklasifikasi ikan menggunakan metode *Viola-Jones Feature Extraction dan Boosting Berbasis Decision Tree*?**'.

1.3 Batasan Masalah

Batasan masalah pada penelitian ini adalah:

1. Klasifikasi ikan menggunakan *Viola-Jones Feature Extraction dan Boosting Berbasis Decision Tree*.
2. Klasifikasi harus bisa melakukan klasifikasi tiga kelas spesies ikan, Abudefduf, Amphiprion, Chaetodon, dan satu kelas negatif.
3. Klasifikasi dilakukan dengan gambar tampak samping ikan saja dengan ikan menghadap ke kiri.

1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah menmbuat program yang mampu mengklasifikasi ikan dengan menggunakan *Viola-Jones Feature Extraction dan Boosting Berbasis Decision Tree*. Dengan tujuan dimasa depan integrasi hasil klasifikasi dengan penghitungan ikan, agar penghitungan dapat berjalan secara otomatis.

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 klasifikasi objek, metode *Boosting* 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 Feature Extraction dan Boosting Berbasis Decision Tree*.

- Bagi Peneliti Selanjutnya

  Diharapkan penelitian ini dapat digunakan sebagai acuan dasar atau kajian awal bagi peneliti yang ingin meneliti permasalahan yang sama, terutama yang berkaitan dengan pengembangan aplikasi klasifikasi ikan.

3. Bagi industri

   Dengan harapan agar di masa depan metode ini dapat di-integrasi untuk penghitungan ikan.

4. Komunitas peneliti

   Diharapkan penelitian ini dapat menjadi laporan referensi penggunaan metode yang digunakan dalam klasifikasi ikan.

BAB II
KAJIAN PUSTAKA

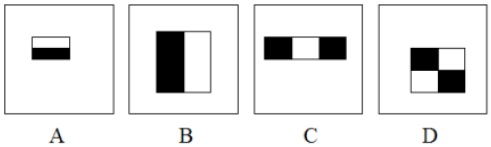
2.1 Pengertian Klasifikasi Objek

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 *threshold* diantara kelas-kelas yang direpresentasikan oleh *training set* yang sudah dilabeli. Pelatihan otomatis umumnya mementingkan distribusi dari setiap kelas (distribusi harus setara), dan melabeli setiap contoh dengan benar, Namun, isu yang umum didalam cara ini pemilihan fitur-fitur yang akan digunakan untuk klasifikasi objek kadang kurang sesuai dan sering kali bergantung terhadap keberuntungan untuk mendapat hasil sangat akurat (Renno dkk., 2007).

2.2 *Viola Jones Object Detection Framework*

Paul Viola dan Michael, J, Jones (ibid.) 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 dkk., 1996) 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 dkk., 1998. Lebih tepatnya ada tiga jenis fitur: dua persegi, tiga persegi dan empat persegi diagonal. Nilai dari sebuah *fitur dua persegi* adalah perbedaaan diantara jumlah nilai piksel didalam dua atau lebih 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 *Adaboost*

Algorithm 1 Algoritma Adaboost

1: Diberikan contoh gambar $(x_1, y_1), \ldots, (x_n, y_n)$ dimana $y_i = 0$ untuk contoh negatif dan $y_i = 1, \ldots, N$ untuk contoh kelas-kelas positif
2: Inisialisasi bobot $w_{i,1} = \frac{1}{\text{total contoh}}$
3: Untuk $t = 1, \ldots, T$:

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

$$\in_t = \min_{f,p,\theta} \sum_i w_i |h(x_i, f, p, \theta) - y_i|. \tag{2.1}$$

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:

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i} \tag{2.2}$$

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:

$$C(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{selain itu} \end{cases} \tag{2.3}$$

dimana $\alpha_t = \log \frac{1}{\beta_t}$

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, setiap contoh pembelajarannya akan dibobot ulang untuk menekankan problem yang salah diklasifikasi oleh *weak learner* sebelumnya. Bentuk *final strong classifier* adalah sebuah kombinasi *weak learner* berbobot.

2.2.3 *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):

$$IG(S, A) = Entropi(S) - sum_v \frac{S_v}{S} x Entropi(S_v) \tag{2.4}$$

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((information gain) = 0) atau tidak bila tinggi maksimum sudah dicapai, bila tidak maka ulangi langkah 2 dan 3.

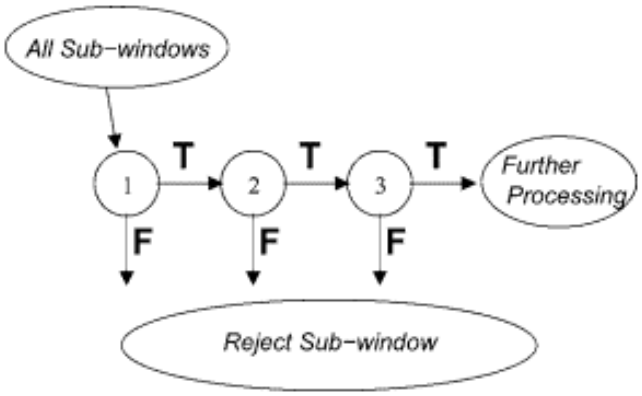
Framework Viola Jones menggunakan sebuah *weaklearn* yang bernama *Decision Stump*, atau sebuah *Decision Tree* yang hanya memiliki dua daun kelas saja. *Decision Tree* sendiri mampu digunakan untuk permasalahan *multi-class*.

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 tes berlanjut ke *node* berikutnya sesuai dengan hasil tes tersebut. Ketika proses pada akhirnya (dan dengan pasti) menuju ke sebuah daun, kelas dari kasus diprediksi sesuai label yang ada pada daun.

2.2.4 *Attentional Cascade*



Gambar 2.2: *Workflow* dari *Attentional Cascade*

Attentional Cascade adalah sebuah *cascade* dari banyak *classifier* yang dibuat untuk meningkatkan performa deteksi dengan secara radikal mengurangi jumlah komputasi. Intinya *weak classifier* 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.

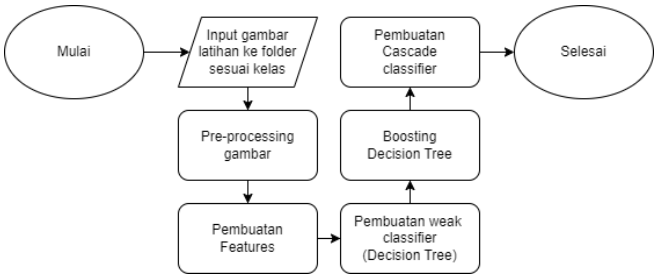
Layaknya sebuah *Decision Tree*, sebuah *classifier* dilatih menggunakan contoh-contoh yang telah berhasil melewati tahap sebelumnya. Oleh karenanya, *classifier* pada tahap kedua menghadapi tantangan yang jauh lebih sulit daripada yang pertama.

Cascade dimulai dengan membuat sebuah *stage* awal, dimana *weak classifier* ditambahkan secara perlahan hingga *detection rate* yang dituju sampai. *Detection rate* ini harus disesuaikan oleh pengguna sesuai dengan keperluannya, tujuannya tetap untuk mengurangi waktu komputasi tapi juga mencoba agar tidak terlalu banyak kasus *false positive* yang dapat lewat.

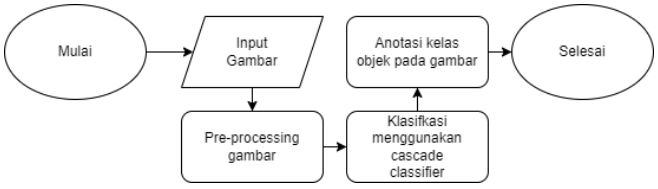
BAB III
METODOLOGI PENELITIAN

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 contoh pelatihan dianotasi sesuai kelasnya masing-masing dengan memberikan label yang sesuai dengan kelas mereka masing-masing, contoh latihan ini berisikan gambar-gambar yang tidak memiliki kelas yang benar, atau *false example* dan juga gambar-gambar yang memiliki kelas yang benar, atau *positive example*. Setelah itu gambar melalui proses *pre-processing* dan disesuaikan untuk mengoptimalkan proses pelatihan.

Pertama sebuah set *features* akan dibuat dengan cara mencoba semua kemugnkinan yang ada dengan bentuk fitur yang dimiliki. Set ini akan berisikan informasi fitur-fitur yang nantinya akan dipakai untuk mendapatkan nilai fitur yang sesungguhnya. Set gambar latihan lalu akan dibaca menggunakan semua fitur ini dan hasilnya akan dicatat dalam tabel csv. Algoritma lalu akan mengkonstruksi

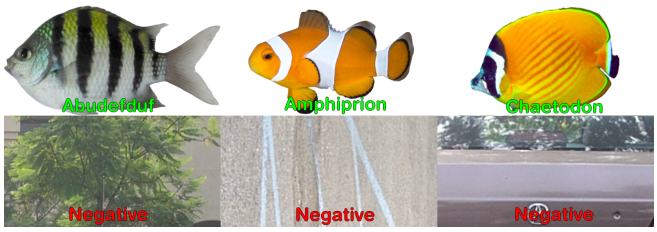
sebuah *decision tree* untuk setiap *feature* untuk menentukan nilai *feature threshold* setiap kelas. *Decision tree* lalu akan di-*boosting* untuk menentukan nilai bobot votingnya pada *strong clossifier*. Akhrinya dari sekumpulan *decision tree* ini dibuatlah sebuah *final strong classifier* yang berbentuk *cascade*.

Dengan *final strong classifier*, barulah klasifikasi objek yang sesungguhnya dapat dilakukan. Pertama gambar yang akan dideteksi akan melalui *pre-processing*. setiap sub-window akan dicek menggunakan *strong classifier* untuk menentukan kelasnya. Hal ini dilakukan pada tiga area: area kiri untuk mengklasifikasi mulut dari ikan, area tengah untuk mengklasifikasi sirip dari ikan, dan terakhir area kanan untuk mengklasifikasi bentuk ekor dari ikan. Hasil ketiga *sub-window* ini nantinya juga akan ber-*voting* dimana jika ada dua atau lebih *sub window* berhasil mengklasifikasikan kelas yang sama, maka kelas itu dipilih sebagai kelas dari objek pada gambar. Kelas dari objek lalu akan dituliskan di pojok kiri atas gambar.

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 *features*, pembuatan *decision tree* untuk setiap *features*, *Boosting* dan pemilihan fitur untuk pemebuatan *attentional cascade*.

3.3.1 *Input Dataset* Pelatihan

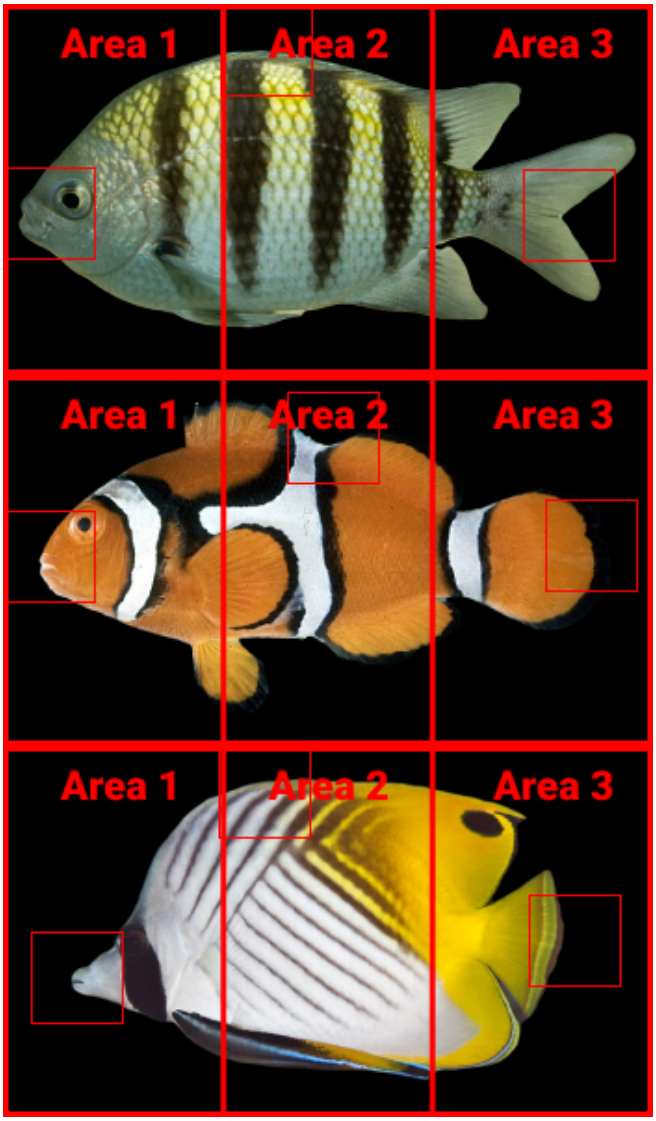


Gambar 3.3: Contoh gambar Abudefduf, Amphiprion, Chaetodon dan contoh gambar-gambar negatif

Dataset yang akan dipakai diambil dari FishBase (Dapat dilihat di https://fishbase.mnhn.fr/) yang berisikan berbagai gambar ikan termasuk gambar dari spesies Abudefduf, gambar dari spesies Amphiprion dan gambar dari spesies Chaetodon. Ketiga spesies ikan ini dipilih karena bentuknya yang berbeda satu-sama lain. Untuk contoh pelatihan gambar dibuat berukuran 350x200 piksel, dengan warna *greyscale*. 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 menyimpan label dalam sebuah *array*, label diambil dari sumber folder gambar. Kelas 0 direservasi untuk kelas negatif, sementara kelas 1, 2 dan 3

direservasi untuk Abudefduf, Amphiprion dan Chaetodon. *Dataset* ini lalu dibagi menjadi tiga yaitu *training dataset*, *testing dataset*, dan *validation dataset* dengan jumlah yang sama.

Selain itu untuk setiap kelas sudah dibuatkan *sub-window* spesifik untuk diklasifikasi. *Sub-window* ini secara spesifik mengklasifikasi tiga bagian ikan yaitu, mulut, sirip dan ekor. klasifikasi spesifik ini nantinya akan bekerja-sama dengan *sliding window* dalam mengklasfikasi gambar yang ada, dengan mencari mulut, sirip dan ekor ikan didalam gambar.



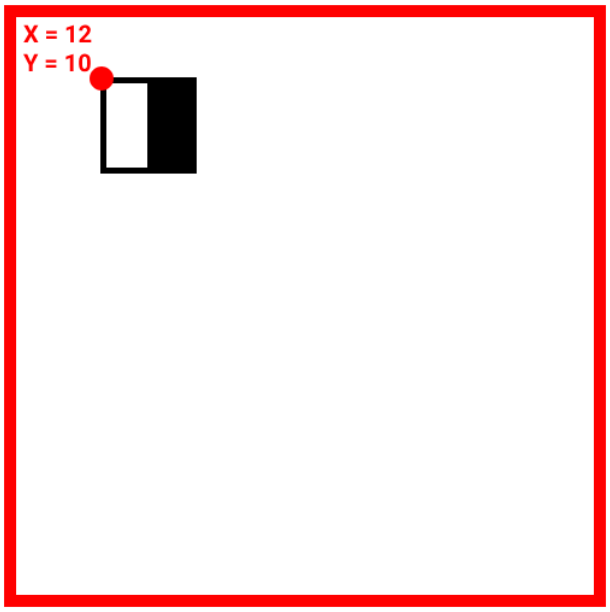
Gambar 3.4: *sub-window* setiap kelas ikan yang akan dipelajari oleh *classifier*

Algorithm 3 offset *sub-window* untuk training setiap kelas

1: class_Window_offset_1 ← [▷ For searching mouth features (0, 0), (88, 0), (73, 0), (100, 15)]
2: class_Window_offset_2 ← [▷ For searching fin features (0, 0), (0, 116), (9, 153), (0, 116)]
3: class_Window_offset_3 ← [▷ For searching tail features (0, 0), (89, 280), (47, 237), (80, 277)]

3.3.2 Pembuatan *Haar like Features*

Fitur-fitur yang akan digunakan dalam klasifikasi dibuat berdasarkan *Haar-like Features* dan berisikan informasi penting yang dapat digunakan untuk mencari nilai sebuah fitur. Sebuah fitur berisikan tipe fiturnya, lokasi fitur tersebut didalam *sub-window*, dan ukuran dari fitur tersebut. Misalnya ada sebuah fitur, ia bertipe dua persegi menghadap ke kiri, lokasi x-nya adalah 12 piksel, dan lokasi y-nya adalah 10 piksel, dia memiliki ukuran 8 x 8 piksel.



Gambar 3.5: Sebuah fitur dua persegi menghadap ke kiri, lokasi x = 12 piksel, lokasi y = 10 piksel, dengan ukuran 8 x 8 piksel.

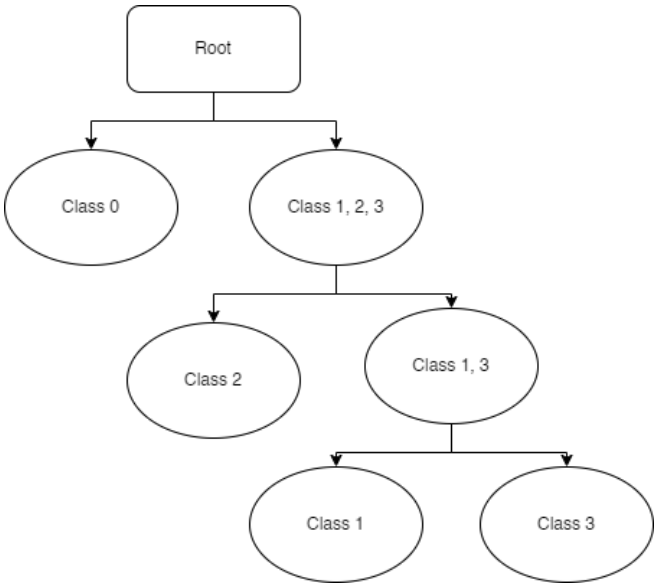
Dengan informasi yang ada didalam sebuah fitur tersebut, nilai fitur dapat dicari dengan mudah menggunakan rumus:

$$\sum \text{nilai piksel area putih} - \sum \text{nilai piksel area hitam} \tag{3.1}$$

Pada tahap ini fitur 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 2 x 2 piksel untuk fitur dua persegi, empat persegi dan diagonal. Dan fitur 1 x 3 piksel atau 3 x 1 piksel untuk fitur tiga persegi. Hal ini juga dilakukan sampai ukuran fitur lebih besar dari *sub-window* dan tidak bisa muat lagi.

Beberapa jenis *feature* yang digunakan adalah 2 persegi horizontal, 2 persegi vertikal, 4 persegi diagonal, 2 segitiga hadap kiri, 2 segitiga hadap kanan, 3 persegi horizontal, dan 3 persegi vertikal.

3.3.3 Pembuatan *Decision Tree*



Gambar 3.6: Contoh sebuah *decision tree* dengan kelas 0, 1, 2 dan 3

Setiap *weak learner* adalah sebuah *decision tree* yang akan mengambil nilai dengan fitur untuk digunakan sebagai variabel klasifikasi. fitur dapat digunakan untuk mencari sebuah nilai perbandingan intensitas cahaya dari dua area pada gambar dan mendeteksi keberadaan suatu fitur seperti perbedaan warna, garis, maupun perbedaan kontras pada gambar.

Decision tree pertama akan dibuat dari *root* atau akar, yang lalu akan bercabang hingga batas maksimum telah dicapai. *threshold* yang digunakan dalam pembuatan *decision tree* adalah salah satu nilai fitur yang dibaca dari gambar. Dengan cara ini, *decision tree* tidak perlu mencoba semua nilai yang mungkin, dan dengan demikian mempercepat proses pembuatan *decision tree*. Batas maksimum tinggi *decision tree* yang dipilih adalah tiga tingkat, hal ini dikarenakan waktu komputasi yang memakan waktu bila *decision tree* memiliki lebih dari 3 tingkat. Di

lain sisi, menggunakan tingkat kurang dari tiga tidak mungkin memenuhi persyaratan klasifikasi empat kelas.

3.3.4 *Boosting*

Boosting ditunjukan untuk memberikan nilai *voting* untuk setiap *weak classifier* yang nantinya akan digunakan dalam klasifikasi akhir. Sebelum *boosting* dimulai, *weak classifier* yang kurang diskriminatif akan dibuang. Hal ini dilakukan dengan membandingkan hasil prediksi *weak classifier* dengan label yang sebenarnya, dimana *weak classifier* yang gagal memprediksi 25% dari set tes akan dibuang. Ini dilakukan untuk mengurangi jumlah *weak classifier* yang akan dipakai berikutnya.

Pada tahap ini *boosting* akan dijalankan dari *decision tree* yang paling akurat ke yang paling tidak akurat. Penentuan akurasi ini dilakukan dengan membandingkan label contoh validasi dengan hasil prediksi setiap *weak classifier*. *Weak classifier* yang tidak akurat akan mendapat suara *voting* yang lemah. Rumus untuk mencari bobot voting α dari sebuah *weak classifier* adalah sebagai berikut:

$$\begin{aligned} \epsilon &= \frac{\sum_i \text{image weights}_i \times \text{indikator}_i}{\sum_i \text{image weights}_i} \\ \alpha &= 0.5 \times \log\left(\frac{1-\epsilon}{\epsilon + 1e - 10}\right) \end{aligned} \tag{3.2}$$

weak classifier terbaik akan mulai dan mengklasifikasi seluruh contoh *dataset* validasi dan mencatat contoh yang gagal diklasifikasi oleh *weak classifier* tersebut. Lalu bobot dari contoh tersebut akan dinaikan, dengan maksud agar bobot *voting* fitur yang paling akurat akan lebih tinggi daripada fitur-fitur yang hanya mendekati menebak. Rumus menaikan bobot:

$$\text{image weights} \times = \exp(\alpha \times \text{indikator}) \tag{3.3}$$

$$\text{image weights} \div = \sum \text{image weights} \tag{3.4}$$

weak classifier berikutnya lalu akan melakukan prediksi dengan set yang sama, namun dengan bobot gambar yang sudah berubah karena *weak classifier*. Sementara bobot *voting weak classifier* sebelumnya akan disimpan ke array untuk digunakan nanti.

Ketika semua *weak learner* sudah dicari bobot *voting*-nya, akan dibandingkan akurasi *strong cloassifier* yang dibuat dengan iterasi sebelumnya. Bila didapat bahwa ada penurunan akurasi, maupun tidak ada perubahan, maka iterasi Boosting akan disudahi dan *strong cloassifier* pada iterasi ini menjadi *final strong classifier*.

3.3.5 Pembuatan *Attentional Cascade*

Karena bobot *voting* dan urutan *weak classifier* sudah ditentukan pada tahap *Boosting*, pada tahap ini hanya perlu membagi *weak classifier* menjadi beberapa *stage* yang nantinya bisa dipanggil secara terpisah. Sebuah *stage* berlaku layaknya sebuah *strong cloassifier* kecil yang ditargetkan hanya memiliki tingkat akurasi paling tidak 50% saja. Hal ini dilakukan agar pendeteksian seluruh *sub-window* dapat dilakukan tanpa harus memanggil keseluruhan dari *strong classifier*. Metode konstruksi sebuah *stage cascade* adalah sebagai berikut:

Algorithm 4 Cascade Train Stage

```
1:  function TRAIN_STAGE
2:      detection_rate ← 0
3:      while detection_rate < 0.5 do
4:          if len(features) == 0 then
5:              break
6:          end if
7:          self.features.append(features)
8:          detection_rate ← accuracy_score()
9:      end while
10: end function
```

Nantinya dengan memanggil keseluruhan *cascade* untuk melakukan klasifikasi, *stage* dengan satu persatu mengklasifikasi *sub-window* yang sedah dicek. Bila sebuah *stage* mem-*voting* sebuah *sub-window* sebagai kelas negatif (Dalam hal ini voting seluruh *weak classifier* didalam *cascade* mengembalikan kelas 0 atau negatif) maka *stage* akan menghentikan klasifikasi untuk *sub-window* tersebut dan melanjutkan ke *sub-window* berikutnya. Sebaliknya bila *stage* mengembalikan kelas selain 0 (1, 2, maupun 3) maka klasifikasi akan dilanjutkan untuk *sub-window* tersebut sampai salah satu *stage* lainnya mem-*voting* kelas negatif atau semua *stage* habis. Dalam situasi habis, *voting* kelas akan mengambil suara dari semua *weak classifer* dari semua *stage*.

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*, dan deteksi yang sesungguhnya menggunakan *cascade* yang telah dibuat dengan metode Sliding Window. Terlebih, klasifikasi akan dilakukan di tiga area berbeda pada gambar: area kiri untuk mendeteksi mulut, area tengah untuk mendeteksi sirip, dan area kanan untuk mendeteksi ekor. Pada akhirnya ketiga *cascade* dari ketiga area tersebut akan

mem-*voting* kelas dari objek pada gambar. Bila didapat bahwa ketiga *cascade* tersebut mem-*voting* tiga kelas yang berbeda, maka kelas yang dipilih adalah 0 atau negatif.

3.4.1 *Pre-processing*

Untuk klasifikasi sebenarnya, gambar yang akan digunakan akan diproses terlebih dahulu. Gambar awalnya akan melakui proses *pre-processing* dan dirubah ke dalam warna *grayscale*. Selain itu resolusi gambar akan dirubah menjadi 350 x 200 piksel untuk mempercepat proses klasifikasi.

3.4.2 Klasifikasi

Pada tahap ini *sliding window* akan mulai bergerak dari pojok kiri atas untuk memulai klasifikasi pada area kiri menggunakan *cascade* yang sudah dilatih untuk mengklasifikasi mulut ikan. Hal ini dilakukan hingga seluruh *sub-window* sudah diklasifikasi dengan hasil negatif, atau bila salah satu *sub-window* mengklasfikasi salah satu kelas positif. Setelah itu maka klasifikasi pada area tengah atau sirip akan dimulai, dan setelahnya baru area kanan atau ekor.

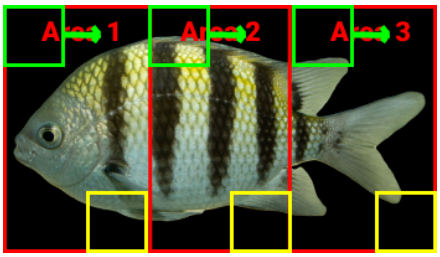
cascade bekerja agak berbeda pada klasifikasi sebenarnya. Melainkan daripada menhitung nilai intensitas cahaya dan fitur pada semua piksel pada gambar, *cascade* hanya menghitung nilai fitur yang berhubungan dengan *weak classifier* dalam *stage cascade* sedang melakukan klasifikasi.

Algorithm 5 Final Cascade Classification

```
1:  function FINALCASCADECLASSIFICATION(image, x_offset, y_offset)
2:      scoreboard ← array([0, 0, 0, 0])
3:      for i in range(len(self.stages)) do
4:          stage_scoreboard ← self.stages[i].stage_prediction()
5:          if stage_scoreboard.argmax() == 0 then break
6:          else
7:              scoreboard ← scoreboard + stage_scoreboard
8:          end if
9:      end for
10:     return scoreboard.argmax()
11: end function
```

Algorithm 6 Stage Prediction

function STAGEPREDICTION(image, x_{offset}, y_{offset}, scoreboard)
    for i **in** range(len(self.features)) **do**
        feature_type, x, y, width, height $\leftarrow$ self.features$[i]$
        $x \leftarrow x + x_{\text{offset}}$
        $y \leftarrow y + y_{\text{offset}}$
        updated_feature $\leftarrow$ (feature_type, x, y, width, height)
        data_features $\leftarrow$ compute_feature_with_matrix(image, 0, updated_feature)
        prediction $\leftarrow$ self.trees$[i]$.predict(data_features)
        scoreboard[prediction] $\leftarrow$ scoreboard[prediction] $+ 1 \times$ self.alpha_list$[i]$
    end for
    return scoreboard
end function



Gambar 3.7: titik awal *sliding window* (kotak hijau) dan titik akhir (kotak kuning)

3.4.3 Anotasi

Gambar 3.8: Anotasi gambar ikan yang sudah diklasifikasi

Setelah ketiga *window* sudah dideteksi oleh *cascade classifer* menggunakan *sliding window*, masing-masing hasilnya akan dipakai untuk menentukan kelas dari objek pada gambar. Setelahnya nama spesies ikan akan ditulis pada kiri atas gambar pada gambar hasil.

BAB IV
HASIL DAN PEMBAHASAN

4.1 *Training Strong Classifier*

4.1.1 *Input* Gambar dan *labeling*

Langkah pertama dalam melatih *classifier* adalah dengan memasukan *dataset* untuk latihan berserta label-labelnya. Gambar pertama dimasukan ke dalam folder sesuai dengan kelasnya. Dalam situasi ini ada empat folder yaitu, untuk kelas satu: abudefduf, untuk kelas dua: amphiprion, untuk kelas tiga: chaetodon dan terakhir untuk kelas nol: negative_examples. Gambar-gambar tersebut lalu akan dibaca menggunakan *library* CV2 yang bertugas juga untuk mengubah gambar menjadi *greyscale*. Berikut adalah *source code* untuk membaca set latihan:

```
def load_images(directory):
  images=[]
  labels=[]
  for filename in os.listdir(directory):
    if filename.endswith(".png"):
      image_path = os.path.join(directory, filename)
      image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
      image = cv2.resize(image, (350, 200))
      images.append(image)
      labels.append(get_label(directory))
  return np.array(images), np.array(labels)
```

Gambar 4.1: *source code: read* gambar, labelisasi, pengubahan ke *greyscale*, dan memastikan ukuran gambar 350 x 200 piksel

```
def get_label(directory):
  # add more to add more class
  if directory == "fish_dataset\\abudefduf": return 1
  if directory == "fish_dataset\\amphiprion": return 2
  if directory == "fish_dataset\\chaetodon": return 3
  else: return 0
```

Gambar 4.2: *source code:* labelisasi gambar sesuai dengan foldernya

Untuk memanggil kedua fungsi load_images dan get_label diperlukan sebuah fungsi lainnya yang berfungsi juga untuk menggabungkan semua gambar dan label menjadi dua buah *array* images dan labels. Kedua *array* ini nantinya akan menjadi *dataset* utama dalam proses pelatihan. Berikut *source code*-nya:

```
def combine_dataset():
  # load datasets from directories
  # add class to get_label first or the class will be considered a negative example
  abudefduf_images, abudefduf_labels = load_images("fish_dataset\\abudefduf")
  amphiprion_images, amphiprion_labels = load_images("fish_dataset\\amphiprion")
  chaetodon_images, chaetodon_labels = load_images("fish_dataset\\chaetodon")
  negatives_images, negatives_labels = load_images("fish_dataset\\negative_examples")

  # combining into a single dataset
  images = np.concatenate((abudefduf_images, amphiprion_images, chaetodon_images, negatives_images), axis =
0)
  labels = np.concatenate((abudefduf_labels, amphiprion_labels, chaetodon_labels, negatives_labels), axis =
0)

  return images, labels
```

Gambar 4.3: *source code: load* gambar-gambar dari folder yang bersangkutan dan menggabungkannya

4.1.2 *Generate Haar-like Features*

Untuk melakukan *generate feature* sebuah fungsi bernama generate_features dipanggil dengan parameter lebar dan tinggi dari *sub-window* yang akan digunakan. Fungsi ini akan menghasilkan kurang lebih sekitar 520000 fitur berbeda untuk *sub-window* berukuran 50 x 50 piksel. Berikut *source code*-nya:

```
def generate_features(image_height, image_width):
  features = []
  features_list = ["Two Horizontal", "Two Vertical", "Four Diagonal", "Right Triangular", "Left Triangular"
, "Three Horizontal", "Three Vertical"]
  for i in features_list:
    match i:
      case "Two Horizontal" | "Two Vertical" | "Four Diagonal" | "Right Triangular" | "Left Triangular":
        feature_height = 4
        feature_width = 4
      case "Three Horizontal":
        feature_height = 4
        feature_width = 6
      case "Three Vertical":
        feature_height = 6
        feature_width = 4
    for w in range (feature_width, image_width+1, feature_width):
      for h in range (feature_height, image_height+1, feature_height):
        for x in range (0, image_width - w):
          for y in range (0, image_height - h):
            feature = (i, x, y, w, h)
            features.append(feature)
    return features
```

Gambar 4.4: *source code: feature* memiliki semua informasi yang diperlukan untuk melakukan perhitungan. Tipe-tipe fitur akan menentukan rumus perhitungan fitur tersebut

4.1.3 *Calculating all features of all images*

Untuk mempermudah proses pembuatan *Decision tree* nantinya, semua fitur pada semua *sub-window*, pada semua gambar akan dihitung dan dimasukan kedalam sebuah dokumen .CSV mengunakan *library* pandas. Fungsi yang dipakai untuk memulai proses ini adalah fungsi write_csv pada Utilities.py. Berikut *source code*-nya:

```
def write_csv(images, labels, features, csv_name):
  print("starting write_csv")
  for window_num in range(3):
    temp_window_values = np.zeros((len(images), len(features)), dtype=object)
    image_ids = np.arange(len(images))

    for i in range(len(images)):
      new_data = Dataset(images[i], labels[i], features)
      if window_num == 0:
        temp_window_values[i] = new_data.window_1_features
      elif window_num == 1:
        temp_window_values[i] = new_data.window_2_features
      elif window_num == 2:
        temp_window_values[i] = new_data.window_3_features

    window_feature = {'image_ids': image_ids}
    for i in range(len(features)):
      column_name = f'win_{window_num + 1}_feature_{i}'
      window_feature[column_name] = temp_window_values[:, i]

    directory = f"Data/{csv_name}_window_{window_num}.csv"

    window_feature = pd.DataFrame(window_feature)
    window_feature.to_csv(directory, index=False)
  print("csv write complete!")
```

Gambar 4.5: *source code:* mengambil semua gambar, label, fitur dan mengkalkulasi semua fitur untuk ketiga *sub-window*

Fungsi ini menghasilkan tiga buah dokumen .csv untuk setiap *sub-window* dengan baris mengikuti jumlah gambar didalam *dataset* dan kolom mengikuti jumlah fitur yang ada. Maka dari itu untuk *dataset* 80 gambar akan dihasilkan sebuah tabel .csv dengan bentuk 80 x 520.000.

Fungsi ini berjalan cukup lama dengan jumlah gambar 80 buah. Penulis menghitung rata-rata waktu yang diperlukan bagi fungsi untuk membuat sebuah dokumen .csv adalah 45 menit minimum. Dan proses ini dilakukan sampai tiga kali agar dapat menciptakan tiga dokumen .csv untuk ketiga *sub-window*. Dokumen .csv ini nantinya akan diperlukan untuk proses pembuatan *decision tree*.

Dataset() adalah sebuah *class* yang digunakan awalnya untuk menyimpan seluruh fitur pada gambar tersebut. Nilai-nilai ini yang lalu akan diambil dan dimasukan kedalam dokumen .csv. berikut bentuk *class* Dataset berserta fungsi bawaannya Find_Feature_Value:

```
class Dataset:
  def __init__(self, image, label, feature_list):
    self.image = image
    self.label = label
    self.window_1_features = self.Find_Feature_Value(image, feature_list, self.class_Window_offset_1[label
][0], self.class_Window_offset_1[label][1])
    self.window_2_features = self.Find_Feature_Value(image, feature_list, self.class_Window_offset_2[label
][0], self.class_Window_offset_2[label][1])
    self.window_3_features = self.Find_Feature_Value(image, feature_list, self.class_Window_offset_3[label
][0], self.class_Window_offset_3[label][1])

  def Find_Feature_Value(self, image, feature_list, x_offset, y_offset):
    features = np.zeros(len(feature_list), dtype=object)
    for i in range(len(feature_list)):
      feature_type, x, y, width, height = feature_list[i]
      x += x_offset
      y += y_offset
      updated_feature = (feature_type, x, y, width, height)
      data_features = compute_feature_with_matrix(image, 0, updated_feature)
      features[i] = data_features
    return features
```

Gambar 4.6: *source code: class Dataset*

Dataset untuk melakukan perhitungan fitur memerlukan *offset* piksel yang penulis sebutkan pada Bab 3. *Offset* digunakan karena masing-masing kelas ikan semuanya memiliki mulut, sirip dan ekor yang berlokasi berbeda satu dengan yang lainnya. Contohnya: Mulut dari spesies ikan Chaetodon agak lebih kebawah dibandingkan kedua kelas ikan lainnya. Selain itu juga untuk setiap bagian ikan yang akan diklasifikasi akan dicari lokasi yang paling terlihat unik daripada kelas-kelas lainnya. Contohnya pada kelas spesies ikan Abudefduf diambil bagian ekor yang memberntuk huruf "V" sementara pada kelas spesies Amphiprion bagian kelas yang dipelajari adalah bagian melengkung bagian atas diantara ekor dan badan. Berikut adalah *offset* yang penulis gunakan untuk pelatihan:

```
class_Window_offset_1 = [
  # order according to label's order in LoadImages
  # for searching mouth features
  (0, 0),
  (0, 88),
  (0, 73),
  (15, 100)
]

class_Window_offset_2 = [
  # order according to label's order in LoadImages
  # for searching fin features
  (0, 0),
  (116, 0),
  (153, 9),
  (116, 0)

]

class_Window_offset_3 = [
  # order according to label's order in LoadImages
  # for searching tail feature
  (0, 0),
  (280, 89),
  (237, 47),
  (277, 80)
]
```

Gambar 4.7: *source code: offset* ini di-inisalisasi untuk setiap kelas Dataset sehingga bisa diakses langsung oleh fungsi Find_Feature_Value

Untuk perhitungan fitur yang sebenernya digunakan fungsi compute_feature_with_matrix. Pertama data fitur diubah dengan menambah lokasi x dan y dari fitur dengan x_offset, y_offset lalu fitur akan mengembalikan sebuah nilai float dari hasil perhitungan tersebut. Berikut *source code* dari compute_feature_with_matrix:

```
def compute_feature_with_matrix(image, feature):
  feature_type, x, y, width, height = feature
  # +1 due to slicing paramter = start at:stop before
  match feature_type:
    case "Two Horizontal":
      white = np.sum(image[y:y + height + 1, x:x + int(width/2) + 1])
      black = np.sum(image[y:y + height + 1, x + int(width/2):x + width + 1])
    case "Two Vertical":
      white = np.sum(image[y:y + int(height/2) + 1, x:x + width+1])
      black = np.sum(image[y + int(height/2):y + height + 1, x:x + width+1])
    case "Three Horizontal":
      white = np.sum(image[y: y + height + 1, x:x + int(width/3) + 1]) + np.sum(image[y: y + height + 1, x
+ int(width*2/3):x + width + 1])
      black = np.sum(image[y: y + height + 1, x + int(width/3):x + int(width*2/3) + 1])
    case "Three Vertical":
      white = np.sum(image[y:y + int(height/3) + 1, x:x + width + 1]) + np.sum(image[y + int(height*2/3):y
+ height + 1, x: x + width + 1])
      black = np.sum(image[y + int(height/3):y + int(height*2/3) + 1, x:x + width + 1])
    case "Four Diagonal":
      white = np.sum(image[y:y + int(height/2) + 1, x + int(width/2): x + width + 1]) + np.sum(image[y +
int(height/2):y + height + 1, x: x + int(width/2) + 1])
      black = np.sum(image[y:y + int(height/2) + 1, x:x + int(width/2) + 1]) + np.sum(image[y + int(height
/2): y + height + 1, x + int(width/2):x + width + 1])
    case "Right Triangular":
      matrix = image[y:y + height + 1, x:x + width + 1]
      white = np.sum(np.tril(matrix))
      black = np.sum(np.triu(matrix))
    case "Left Triangular":
      matrix = np.rot90(image[y:y + height + 1, x:x + width + 1], k=3)
      white = np.sum(np.tril(matrix))
      black = np.sum(np.triu(matrix))
  return int(white) - int(black)
```

Gambar 4.8: *source code: offset* ini di-inisalisasi untuk setiap kelas Dataset sehingga bisa diakses langsung oleh fungsi Find_Feature_Value

4.1.4 *Create Decision Tree for each Feature*

Setelah semua nilai fitur sudah dihitung dan dimasukan kedalam dokumen .csv, selanjutnya bisa dimulai proses pembuatan *decision tree* atau *weak classifier*. Proses ini dimulai dengan pertama membagi contoh menjadi tiga kelompok, yaitu data *training*, data *testing* dan data *validation*. Hal ini dilakukan dengan menggunakan fungsi split_data dibantu dengan fungsi train_test_split dari *library* sklearn:

```
def split_data(features, csv_name, labels):
  data = DecisionTree.get_data(features, csv_name)
  labels_df = pd.DataFrame({'Label' : labels})
  data = pd.concat([data, labels_df], axis=1)

  X = data.iloc[:, :-1].values
  Y = data.iloc[:, -1].values.reshape(-1, 1)

  X_temp, X_train, Y_temp, Y_train = train_test_split(X, Y, test_size=0.3, random_state=42)
  X_valid, X_test, Y_valid, Y_test = train_test_split(X_temp, Y_temp, test_size=0.5, random_state=42)

  print(type(X_train))
  splits = [X_train, Y_train, X_test, Y_test, X_valid, Y_valid]
  return splits
```

Gambar 4.9: *source code: offset* ini di-inisalisasi untuk setiap kelas Dataset sehingga bisa diakses langsung oleh fungsi Find_Feature_Value

Label baru ditambahkan sekarang agar tidak menggangu proses penulisan kolom .csv yang dinamis yang bergantung pada jumlah fitur yang ada. Hasilnya adalah sebuah *object* bernama splits yang memiliki *dataframe*: X_train, Y_train, X_test, Y_test, X_valid, Y_valid. *Dataframe* dengan data awalan X berisikan nilai-nilai fitur yang sudah dikalkulasi di tahap sebelumnya. Sementara data awalan Y berisikan label untuk data X.

split_data mengambil data dari .csv menggunakan fungsi bernama get_data. Fungsi ini hanya bertugas untuk membaca .csv saja dengan bantuan fungsi read_csv:

```
def get_data(features, csv_name):
  col_names = ['image_ids']
  for i in range(len(features)):
    temp_column_name = f'win_1_feature_{i}'
    col_names.append(temp_column_name)
  return Utilities.read_csv(csv_name, col_names)

def read_csv(csv_name, col_names):
  # used to read all column
  directory = "Data/" + csv_name + ".csv"
  data = pd.read_csv(directory, skiprows=1, header=None, names = col_names)
  return data
```

Gambar 4.10: *source code:* get data dan readcsv yang digunakan oleh split data

setelah data di-*split*, barulah *decision tree* bisa deibuat dengan menggunakan fungsi build_all_tree:

```
def build_all_tree(splits, features):
  classifiers = [None] * len(features)
  classifiers_accuracy = [0] * len(features)
  X_train, Y_train, X_test, Y_test, X_valid, Y_valid = splits
  minimum_splits = 3
  maximum_depth = 3
  for i in range(len(features)):
    if i % 1000 == 0: print (f'starting tree {i}')
    classifier = DecisionTreeClassifier(minimum_splits, maximum_depth)
    classifier.fit(temp_X_train, Y_train)
    # classifier.print_tree()

    classifiers[i] = classifier
    Y_pred = classifier.predict(X_test)
    classifiers_accuracy[i] = accuracy_score(Y_test, Y_pred)
  return classifiers, classifiers_accuracy
```

Gambar 4.11: *source code:* untuk membuat semua decision tree untuk setiap fitur

Pada tahap ini build_all_tree mengisolasi kolom nilai fitur yang berhubungan dengan menyimpannya pada temp_X_train, yang nantinya akan digunakan saat pembuatan *decision tree*. *Decision tree* di-inisialisais menggunakan fungsi DecisionTreeClassifier dan disimpan menjadi classifier. Classifier lalu dilatih menggunakan fungsi fit. Hasil dari pelatihan ini lalu langsung dites menggunakan fungsi accuracy_score dari *library* sklearn. classifiers_accuracy ini nantinya akan dipakai dalam proses *Boosting*. Untuk keseluruhan *class* DecisionTreeClassifier dan fungsi-fungsinya bisa dilihat berkut ini:

```
class Node():
  def __init__(self, feature_index=None, threshold=None, left=None, right=None, info_gain=None, value=
None):
    self.feature_index = feature_index
    self.threshold = threshold
    self.left = left
    self.right = right
    self.info_gain = info_gain

    self.value = value
```

Gambar 4.12: *source code: class* node digunakan untuk menyimpan informasi cabang dan *threshold* pada *node decision tree*

```
class DecisionTreeClassifier():
  def __init__(self, minimum_splits = 2, maximum_depth = 2):
    self.root = None

    # stopping condition
    self.minimum_splits = minimum_splits
    self.maximum_depth = maximum_depth
```

Gambar 4.13: *source code: class* digunakan untuk menyimpan tinggi maksimal dan minimal *split* pada *decision tree*. Semua data lainnya disimpan pada node

```
    def build_tree(self, training_dataset, current_depth = 0):

      X, Y = training_dataset[:,:-1], training_dataset[:,-1]
      num_samples, num_features = np.shape(X)

      # split until conditons are met
      if num_samples >= self.min_samples_split and current_depth <= self.max_depth:
        # find best split
        best_split = self.get_best_split(dataset, num_samples, num_features)
        # check if information gain is positive
        if best_split["info_gain"]>0:
          left_subtree = self.build_tree(best_split["dataset_left"], current_depth+1)
          right_subtree = self.build_tree(best_split["dataset_right"], current_depth+1)
          return Node(best_split["feature_index"], best_split["threshold"], left_subtree, right_subtree,
best_split["info_gain"])

      leaf_value = self.calculate_leaf_value(Y)
      return Node(value=leaf_value)
```

Gambar 4.14: *source code:* fungsi utama dari *class* DecisionTreeClassifier

Fungsi build_tree adalah fungsi utama untuk pelatihan *decision tree* yang berjalan secara rekursif sampai sebuah daun sudah didapat, atau kedalaman maksimum sudah dicapai. Sebuah daun sudah didapat bilamana *info gain* dari fungsi get_best_split 0, atau *node* sudah tidak perlu dipecah lagi.

```
    def get_best_split(self, dataset, num_samples, num_features):
      # dictionary to save data
      best_split = {
      "info_gain": -float("inf")  # Initialize info_gain to a very small value
      }
      max_info_gain = -float("inf")

      for feature_index in range(num_features):
        feature_values = dataset[:, feature_index]
        possible_thresholds = np.unique(feature_values)

        for threshold in possible_thresholds:
          # get curent split
          dataset_left, dataset_right = self.split(dataset, feature_index, threshold)
          # check if child not null
          if len(dataset_left) > 0 and len(dataset_right) > 0:
            y, left_y, rigth_y = dataset[:,-1], dataset_left[:,-1], dataset_right[:, -1]
            # compute information gain
            current_info_gain = self.information_gain(y, left_y, rigth_y, "gini")
            # update best split if needed
            if current_info_gain > max_info_gain:
              best_split["feature_index"] = feature_index
              best_split["threshold"] = threshold
              best_split["dataset_left"] = dataset_left
              best_split["dataset_right"] = dataset_right
              best_split["info_gain"] = current_info_gain
              max_info_gain = current_info_gain

      return best_split
```

Gambar 4.15: *source code:* fungsi get_best_split

Fungsi get_best_split bertugas untuk mencari *threshold* paling sesuai untuk memecah cabang suatu *node* dengan mengetes satu-persatu atribut nilai atribut data latihan. Atribut disini adalah nilai *feature* yang sedang dilatih dari semua gambar dari set *train*. Untuk mencari *info gain*, *gini purity* akan dicari menggunakan fungsi information_gain.

```
def split(self, dataset, feature_index, threshold):
  # fuction to split data
  dataset_left = np.array([row for row in dataset if row[feature_index] <= threshold])
  dataset_rigth = np.array([row for row in dataset if row[feature_index] > threshold])
  return dataset_left, dataset_rigth
```

Gambar 4.16: *source code:* fungsi split hanya bertugas membagi berdasarkan *threshold* yang sudah ditemukan

```
def information_gain(self, parent, left_child, right_child, mode="entropy"):
  weight_left = len(left_child) / len(parent)
  weight_rigth = len(right_child) / len(parent)
  if mode == "gini":
    gain = self.gini_index(parent) - (weight_left * self.gini_index(left_child) + weight_rigth * self.
gini_index(right_child))
  else:
    gain = self.entropy(parent) - (weight_left * self.entropy(left_child) + weight_rigth * self.entropy
(right_child))
  return gain
```

Gambar 4.17: *source code:* information_gain mencari data dengan menghitung *gini purity*

```
def entropy(self, y):
  # fuction to count entropy
  class_labels = np.unique(y)
  entropy = 0
  for cls in class_labels:
    p_cls = len(y[y == cls]) / len(y)
    entropy += -p_cls * np.log2(p_cls)
  return entropy
```

Gambar 4.18: *source code:* mencari data dengan menghitung *entrophy*

```
def gini_index(self, y):
  # function to count gini index (lebih cepet aja karna gak pake log)
  class_labels = np.unique(y)
  gini = 0
  for cls in class_labels:
    p_cls = len(y[y == cls]) / len(y)
    gini += p_cls**2
  return 1 - gini
```

Gambar 4.19: *source code:* mencari data dengan menghitung *gini*

```
def calculate_leaf_value(self, Y):

  Y = list(Y)
  return max(Y, key = Y.count)
```

Gambar 4.20: *source code:* fungsi untuk mencari mayoritas kelas pada *leaf node*

```
def fit(self, X, Y):
  # fuction to train tree
  dataset = np.concatenate((X, Y), axis = 1)
  self.root = self.build_tree(dataset)
```

Gambar 4.21: *source code:* fungsi fit adalah fungsi yang dipanggil untuk mulai membangun *decision tree* setelah dibuat

Terakhir, fungsi Predict digunakan untuk klasifikasi yang sebenarnya. Dalam fugnsi ini fit mengambil X atau *dataset* X_test untuk mengetes akurasi dari *decision tree* tersebut yang lalu akan dikomparasi dengan Y_test menggunakan fungsi sklearn accurac_score. Pencarian accuracy_score setiap *decision tree* disini dilakukan untuk mempercepat proses *boosting* di tahap berikutnya karena *decision tree* dapat langsung diurutkan dari yang terkuat ke yang terlemah.

```
def predict(self, X):
  # fuction to predict new dataset
  predictions = [self.make_prediction(x, self.root) for x in X]
  return predictions

def make_prediction(self, x, tree):
  # fuction to detect single datapoint
  if tree.value!=None: return tree.value
  feature_val = x[tree.feature_index]
  if feature_val <= tree.threshold: return self.make_prediction(x, tree.left)
  else: return self.make_prediction(x, tree.right)
```

Gambar 4.22: *source code:* predict digunakan untuk melakukan prodeiksi dengan *decision tree* yang sudah dibuat

Setelah semua *decision tree* dan akurasinya sudah dicari dan disimpan kedalam *array* classifiers dan classifiers_accuracy. Keduanya akan disimpan kedalam dokumen pickle untuk direferensi kedepannya. Penyimpanan kedalam dokumen pickle ini bertujuan agar proses pelatihan tidak perlu diulangi berulang kali bila ada masalah di tahapan berikutnya. Berikut *source code* penyimpanan *decision tree* kedalam pickle:

```
class PickleTree:
  def __init__(self, features, trees, accuracies):
    self.feature_num = np.arange(len(features))
    self.trees = trees
    self.accuracies = accuracies

def dump_to_pickle(file_name, object):
  directory = "Data/" + file_name + ".pickle"
  with open(directory, 'wb') as file:
    pickle.dump(object, file)
```

Gambar 4.23: *source code:* penyimpanan *decision tree* kedalam pickle

Untuk menyimpan *decision tree* kedalam pickle, pertama semua *decision tree* dan akurasinya disimpan kedalam *class* bernama PickleTree yang lalu akan di *dump* menggunakan fungsi dunp_to_pickle kedalam *directory* yang sudah ditentukan.

4.1.5 *Boosting*

Setelah dokumen pickle dari semua *decision tree* atau *weak classifier* dibuat. Semua *weak classifier* akan di-*boosting* untuk memberikan bobot *voting* untuk semuanya. Hal ini dilakukan dengan mengetes *weak classifier* secara berurutan dari yang terkuat ke yang terlemah. Contoh-contoh latihan yang sulit untuk diklasifikasi *weak learner* sebelumnya akan diberikan nilai lebih bila berhasil diklasifikasi *weak learner* berikutnya. Proses ini kita mulai dengan memanggil fungsi training_strong_classifier():

```
def training_strong_classifier(features, trees, splits, accuracy, pickle_name):
  X_train, Y_train, X_test, Y_test, X_valid, Y_valid = splits
  image_weights = Boosting.initialize_weight(Y_test)
  orderlist = np.arange(len(accuracy))
  orderlist = Boosting.get_initial_sorted_accuracy(accuracy, orderlist)

  initial_accuracy = float('-inf')
  current_accuracy = 0
  iteration = 0
  limit = 100 #change according to needs

  # start boosting loop. Will stop when accuracy fell or iteration hit limit
  while True:
    alpha_list = Boosting.start_boosting(trees, X_test, Y_test, image_weights, orderlist)
    validation_prediction = Boosting.strong_prediction(trees, orderlist, X_valid, alpha_list)

    initial_accuracy = current_accuracy
    print(f'current initial accuracy: {initial_accuracy}')
    current_accuracy = accuracy_score(Y_valid, validation_prediction)
    print(f'current after boosting accuracy: {current_accuracy}')

    # check whether accruacy deteriorate or limit hit
    if current_accuracy <= initial_accuracy or iteration >= limit:
      print('final accuracy deteriorate, rolling back to last iteration...')
      alpha_list = last_iteration_alpha_list
      orderlist = last_iteration_orderlist
      break

    print('starting over. Saving alpha...')
    alpha_list, orderlist = Boosting.get_sorted_accuracy(alpha_list, orderlist)
    last_iteration_alpha_list = alpha_list
    last_iteration_orderlist = orderlist
    iteration += 1


  # saving trees, related features and its order in pickle
  final_trees = np.empty(len(orderlist), dtype=object)
  final_features = np.empty(len(orderlist), dtype=object)
  for i in range(len(orderlist)):
    final_trees[i] = trees[orderlist[i]]
    final_features[i] = features[orderlist[i]]

  pickle_this = PickleTreeFinal(final_features, final_trees, alpha_list)
  Utilities.dump_to_pickle(f'{pickle_name}', pickle_this)
```

Gambar 4.24: *source code:* training_strong_classifier

Pertama dalam fungsi ini harus dicari bobot nilai dari setiap contoh latihan, bobot nilai ini berbeda dari bobot *voting weak learner*. Fungsi bobot nilai adalah memberikan nilai bila *weak learner* berhasil mengklasifikasi sebuah contoh latihan dengan benar, oleh karena itu jumlah nilai total dari bobot latihan atau image_weights haruslah kurang lebih satu. image_weights di-inisialisasi menggunakan fungsi initialize_weights() berikut:

```
def initialize_weight(test_images):
  image_weights = np.ones(len(test_images)) / len(test_images)
  return image_weights
```

Gambar 4.25: *source code:* pencarian bobot gambar

Bisa dilihat proses penghitungan bobot dari image_weights hanyalah pembagian satu dengan jumlah total contoh latihan (disini len(test_images)). Untuk proses pelatihan menggunakan 80 contoh gambar latihan, fungsi split() telah mengalokasikan 28 contoh untuk digunakan dalam tahap *Boosting*, yang disimpan

dalam X_valid dan Y_valid. Berikutnya fungsi get_initial_sorted_accuracy() dipanggil untuk mengurutkan *weak classifier* berdasarkan akurasi yang sudah didapat ditahap sebelumnya:

```
    def get_initial_sorted_accuracy(accuracy, orderlist):
      accuracy_threshold = 0.4
      accuracy, orderlist = zip(*sorted(zip(accuracy, orderlist), reverse = True))
      orderlist = [classifier for accuracy, classifier in zip(accuracy, orderlist) if accuracy >=
accuracy_threshold]
      return orderlist
```

Gambar 4.26: *source code:* pengurutan *weak classifier* berdasarkan akurasi

Pada tahap *sorting* ini juga dilakukan elimanasi *weak classifier* yang terlalu lemah. Awalnya penulis mengeliminasi *weak classifier* yang memiliki nilai akurasi dibawah 50% namun karena takut jumlah *weak classifier* terlalu sedikit, maka penulis menurunkan *threshold* menjadi 40%. Eliminasi ini secara signifikan mengurangi jumlah *classifier* yang awalnya berjumlah sekitar 520.000 menjadi: 6742 *weak classifier* pada *classifier* jendela kiri, 8231 *weak classifier* pada *classifier* jendela tengah, dan 10588 *weak classifier* pada *classifier* jendela kanan. Eliminasi yang besar ini mengimplikasikan bahwa mayoritas *weak classifier* yang dibuat dengan mencoba semua probabilitas yang ada memiliki akurasi dibawah 40% dan mungkin hanya akan berkontribusi saja kepada klasifikasi akhir. Selanjutnya proses *Boosting dilanjutkan* dengan mencari alpha_list atau bobot voting setiap *weak classifier* menggunakan fungsi start_boosting(). Berikut adalah *source code*-nya:

```
    def start_boosting(trees, X_test, Y_test, image_weights, orderlist):
      print('Boosting...')
      alpha_list = np.zeros(len(orderlist))
      for i in range(len(orderlist)):
        # make prediction with i-th tree
        treeN = orderlist[i]
        prediction  = trees[treeN].predict(X_test)

        # calculate error of the tree
        indicator = np.array(np.array(prediction).astype(int) != Y_test.flatten(), dtype = float)
        epsilon = np.sum(image_weights * indicator) / np.sum(image_weights)

        # calculate the weight of the tree
        alpha = 0.5 * np.log((1 - epsilon) / (epsilon + 1e-10)) + np.log(4 - 1) #1e-10 const added to prevent
 div by 0. 4 is number of class
        if alpha < 1e-10: alpha = 1e-10 #1e-10 const added to prevent alpha getting too small in np.exp(alpha
 * indicator) later
        alpha_list[i] = alpha

        # update the weight for the samples so the sum of image_weight will be close to 1 for the next
iteration
        image_weights *= np.exp(alpha * indicator)
        image_weights /= np.sum(image_weights)

      return alpha_list
```

Gambar 4.27: *source code:* pencarian nilai bobot voting menggunakan fungsi start_boosting()

Disini hasil klasifikasi yang dilakukan oleh *weak classifier* akan dibandingkan dengan label aslinya yang tersimpan di Y_valid dan disimpan pada array *indidcator* dalam nilai 0 bila klasifikasi dilakukan secara benar, dan 1 bila klasifikasi dilakukan secara salah. Kemudian *alpha* atau bobot voting sang *weak classifier* akan dihitung. Para perhitungan ini, *epsilon* akan ditambahkan dengan 1e-10 untuk mencegah pembagian dengan angka 0 bilamana *weak classifier* benar mengklasifikasi semua contoh dan menghasilnya *indicator* yang hanya berisi angka 0 saja. np.log(4 - 1) disini digunakan agar *alpha* tidak negatif, 4 pada formula ini adalah jumlah kelas yang sedang diklasifikasi yaitu kelas negatuf, Abudefduf, Amphiprion, dan Chaetodon. Berikutnya *alpha* dicek supaya tidak lebih kecil daripada 1e-10 agar tidak menyebabkan normalisasi bobot gambar yang salah di bagian berikutnya. terakhir image_weight diupdate, dimana gambar yang salah diklasifikasi akan dinaikan nilainya, baru setelahnya nilai dinormalisasi lagi agar kurang lebih berjumlah 1. Setelah bobot *voting* sudah dicari, seluruh *weak learner* pada tahap ini akan dites layaknya klasifikasi yang sebeneranya, dimana nilai *voting* setiap *weak classifier* akan diperhitungkan untuk memilih hasil klasifikasi. Klasifikasi pada tahap ini dilakukan oleh fungsi strong_prediciton:

```
def strong_prediction(trees, orderlist, X_valid, alpha_list):
  predictions = [0] * len(X_valid)
  scoreboard = [[0, 0, 0, 0] for _ in range(len(X_valid))]
  for i in range(len(orderlist)):
    tree_index = orderlist[i]
    prediction = trees[tree_index].predict(X_valid)

    # add score to scoreboard according to results and alpha value of tree
    for j in range(len(prediction)):
      weak_learner_prediction = int(prediction[j])
      scoreboard[j][weak_learner_prediction] += 1 * alpha_list[i]

  # return score to the main scoreboard
  for k in range(len(prediction)):
      # print(f'scoreboard {k}: {scoreboard[k]}')
      predictions[k] = scoreboard[k].index(max(scoreboard[k]))
  return predictions
```

Gambar 4.28: *source code:* klasifikasi yang dilakukan setelah setiap iterasi *boosting*

Saat klasifikasi akan dibuatkan sebuah *scoreboard* untuk mencatat total bobot voting suatu kelas. Contohnya suatu *weak classifier* dengan bobot *voting* 0.67 mengklasifkasi suatu fitur sebagai kelas 1 atau Abudefduf. maka *scoreboard* akan berubah manjadi [0, 0.67, 0, 0]. Lalu misalnya *weak classifier* lain dengan bobot *voting* 0.2 memilih kelas 3 atau Amphiprion, maka *scoreboard* akan menjadi [0, 0.67, 0.2, 0]. Klasifikasi akan diakhiri ketika semua *weak learner* sudah dipakai. Setelah itu kelas dengan nilai *voting* paling tinggi akan dipilih sebagai hasil dari klasifikasi. Yang lalu akan dibandingkan dengan Y_valid untuk dicari akurasinya.

Proses iterasi *boositng* ini akan diulang terus menerus hingga tingkat akurasi klasfikasi menggunakan *weak classifier* berbobot mulai mengalami penurunan. Dalam situasi ini nilai bobot voting dan urutan voting pada iterasi sebelumnya akan diambil dan disimpan kedalam dokumen pickle, kali ini dalam *class PickleTreeFinal*

dengan fungsi dump_to_pickle sebelumnya:

```
class PickleTreeFinal:
  def __init__(self, features, trees, alpha_list):
    self.features = features
    self.trees = trees
    self.alpha_list = alpha_list
```

Gambar 4.29: *source code:* bentuk *class PickleTreeFinal*

Perbedaan pickle ini dengan pickle yang menyimpan seluruh fungsi pada tahap sebelumnya adalah pada pickle ini juga disimpan info *feature* juga sesuai dengan urutan dari *weak classifier* yang berhubungan. Hal ini dilakukan agar pada tahapan berikutnya, *cascade*, juga akan digunakan untuk klasifikasi sebenarnya, sehingga memerlukan info *features* untuk dapat langsung membaca nilai fitur langsung dari gambar.

4.1.6 *Training Cascade*

Pelatihan pickle dimulai dengan pertama membuat *class* Cascade yang nantinya akan diisi dengan *stage* yang berisikan *weak classsifier* dengan bobot voting mereka. Berikut bentuk *class* Cascade:

```
class Cascade:

  def __init__(self):
    self.stages = []
```

Gambar 4.30: *source code:* bentuk *class Cascade*

Selanjutnya *stages* akan diisi dengan fungsi fill_cascade() dengan *source code* sebagai berikut:

```
def fill_cascade(self, features, trees, alpha_list, splits):
  print(f'starting to fill cascade...')
  X_train, Y_train, X_test, Y_test, X_valid, Y_valid = splits
  used_features = 0
  print(f'number of used_features: {used_features}')
  while True:
    if used_features >= len(features): break
    new_cascade = CascadeStage()
    new_cascade.train_stage(features, trees, alpha_list, X_valid, Y_valid, used_features)
    used_features += len(new_cascade.trees) #check the total number of features used
    self.stages.append(new_cascade)
    print(f'finished filling stage: {len(self.stages)}')
  print(f'cascade is finished!')
```

Gambar 4.31: *source code:* fungsi untuk mengisi *stages* pada *Cascade*

Fungsi fill_cascade disini mengambil *features, trees* dan *alpha_list* yang sudah diurutkan dan di-*boosting* pada tahap sebelumya. Lalu sebuah *class* baru dibuat untuk menyimpan ketiganya pada sebuah *stage*, *class* itu adalah CascadeStage dengan *source code* sebagai berikut:

```
class CascadeStage:
  def __init__(self):
    self.features = []
    self.trees = []
    self.alpha_list = []
```

Gambar 4.32: *source code: class* CascadeStage

yang lalu diisi dengan menggunakan fungsi train_stage() sebagai berikut:

```
def train_stage(self, features, trees, alpha_list, X_valid, Y_valid, used_features):
  detection_rate = 0
  while detection_rate < 0.5:
    if used_features >= len(features): break
    # append weak classifier into stage one by one
    self.features.append(features[used_features])
    self.trees.append(trees[used_features])
    self.alpha_list.append(alpha_list[used_features])

    orderlist = np.arange(len(self.trees))
    validation_prediction = Boosting.strong_prediction(self.trees, orderlist, X_valid, self.alpha_list)
    detection_rate = accuracy_score(Y_valid, validation_prediction)
    used_features += 1
  print(f'features used in this stage: {used_features}')
```

Gambar 4.33: *source code:* pelatihan *stage* dalam *Cascade*

Pada fungsi ini *weak classifier* ditambah satu persatu berserta bobot dan *feature*-nya, lalu *stage* akan dites menggunakan fungsi strong_prediction yang dipakai juga di tahap *boosting*. Penggunaan fungsi ini dapat dilakukan karena struktur dan cara klasifikasi dari *stage* mirip dengan sederet *weak classifier* pada tahap sebelumnya. Iterasi ini lalu diteruskan hingga akurasi *stage* mencapai 50% atau sampai *weak classifier* habis. Setelah itu *stage* yang sudah dibuat akan di-append() kedalam *array stages* dan proses yang sama diulangi hingga semua *weak classifer* sudah habis terpakai. Terakhir *Cascade* lalu disimpan kedalam pickle untuk digunakan dalam klasifikasi yang sebenarnya. Berikut fungsi yang digunakan untuk menyimpan *cascade* kedalam pickle:

```
def save_to_pickle(self, pickle_name):
  print(f'saving to Pickle...')
  Utilities.dump_to_pickle(f'{pickle_name}', self)
  print(f'complete!')
```

Gambar 4.34: *source code:* fungsi penyimpanan *Cascade* ke pickle dengan menggunakan fungsi dump_to_pickle lagi

4.2 Validasi

Untuk proses validasi atau penggunaan, penulis telah membuat sebuah *file* python baru untuk mengklasifikasi menggunakan *cascade* yang telah dibuat. Untuk gambar yang akan diklasifikasi harus memiliki ukuran 350 x 200 piksel, bertipekan *Portable Network Graphics* atau PNG, dan dengan latar belakang sudah dihilangkan. Gambar-gambar yang akan diklasifikasi harus dimasukan kedalam *folder* bernama *classification_target*, dan pengguna juga membuat satu *folder* lain bernama *classification_results* untuk hasil klasifikasi. Berikut *source code* dari predict.py:

```
import numpy as np
  import os
  import cv2
  from Cascade import *
  from Utilities import *

  # load cascades for each window
  window_cascade = [None, None, None]
  window_prediction = np.zeros(3)
  window_cascade[0] = Utilities.read_from_pickle('window_0_cascade') #for left side/mouth detection
  window_cascade[1] = Utilities.read_from_pickle('window_1_cascade') #for mid side/fin detection
  window_cascade[2] = Utilities.read_from_pickle('window_2_cascade') #for right side/tail detection

  directory = "classification_target"

  for filename in os.listdir(directory):
    if filename.endswith(".png"):
      image_path = os.path.join(directory, filename)
      image_name = filename

      #load target image for classification
      image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
      image_unedited = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
      image = cv2.resize(image, (350, 200))
      image_width = 350
      image_height = 200

      # scan for the whole image using sliding windows
      for i in range(len(window_cascade)):
        # offset for different part all 3 window
        match i:
          case 0: left_window_width = 0
          case 1: left_window_width = int(image_width / 3)
          case 2: left_window_width = int(image_width / 3 * 2)
        for y in range(0, image_height - 50 + 1):
          for x in range(0, int(image_width / 3) - 50 +1):
            prediction = window_cascade[0].final_cascade_classification(image, x + left_window_width, y)
            if prediction != 0: break

          if prediction != 0: break
        # print(f'classification result for window {i}: {prediction}')
        window_prediction[i] = prediction

      # count majority vote and predict class
      print(f'result of {image_name} classification: {window_prediction}')
      unique_elements, counts = np.unique(window_prediction, return_counts=True)
      max_count_index = np.argmax(counts)

      if counts[max_count_index] > len(window_prediction) // 2:
        image_class = unique_elements[max_count_index]
      else:
        image_class = 0

      match image_class:
        case 0: image_class = 'None'
        case 1: image_class = 'Abudefduf'
        case 2: image_class = 'Amphiprion'
        case 3: image_class = 'Chaetodon'

      position = (10, 30)
      font = cv2.FONT_HERSHEY_SIMPLEX
      font_scale = 1
      font_thickness = 2
      font_color = (0, 0, 255)

      output_image_path = os.path.join('classification_results\\', os.path.splitext(filename)[0] + '.jpg'
)
      cv2.putText(image_unedited, image_class, position, font, font_scale, font_color, font_thickness)
      cv2.imwrite(output_image_path, image_unedited)
      print('anotated image completed!')
```

Gambar 4.35: *source code:* predict.py untuk melakukan klasifikasi sebenarnya

Source code selengkapnya termasuk dengan *source code training* bisa dilihat di (https://github.com/EzraelVio/Fish-Viola-Jones) dibawah lisense *GNU General Public License v3.0.* *Prototype System* Pendeteksi Spesies Ikan Menggunakan *Viola-Jones Featues Extraction* dan *Boosting* berbasis *Decision Tree*. Rincian yang

lebih lengkap soal hasil klasifikasi bisa diliat di lampiran.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil implementasi yang telah dilakukan penulis serta pengujian fitur klasifikasi yang telah penulis rancang, maka didapatkan kesimpulan sebagai berikut:

1. Dibuatnya *classifier* spesies ikan berbasis Viola-Jones Feature Extraction dan Boosting Berbasis Decision Tree. Adapun perancangan *classifier* ini dikerjakan dalam waktu kurang lebih satu tahun.

2. Berdasarkan hasil pengetesan akurasi yang didapat dari *classifier* jendela 1 adalah: x %, *classifier* jendela 2 adalah: x % dan *classifier* jendela 3 adalah: x %.

3. Dari hasil percobaan mengklasifikasi 100 gambar dengan persebaran kelas: ... *classifier* akhir berhasil mengklasifikasi x gambar.

5.2 Saran

Klasifikasi ikan dengan menggunakan metode *framework Viola-Jones* dengan *decision tree boosting*membuka peluang untuk melanjutkan penelitian dalam klasifikasi ikan menggunakan metode tersebut. Ada beberapa perubahan yang mungkin bisa dilakukan untuk memutakhirkan metode ini, salah satunya adalah penambahan *object extraction* untuk mempermudah *pre-processing*. Mungkin juga memodifikasi model agar dapat bekerja dengan gambar yang tidak selalu berukuran 350 x 200 piksel. Dan terakhir, mengimplementasikan *multi processing* untuk mempercepat proses pelatihan yang cukup lama, yang dapat kedepannya memungkinkan pengguna untuk melatih *classifier* dengan lebih banyak data latihan dengan waktu yang jauh lebih singkat.

DAFTAR PUSTAKA

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

Diansari, R., E. Arini, and T. Elfitasari (2013). "Pengaruh Kepadatan Yang Berbeda Terhadap Kelulushidupan dan Pertumbuhan Ikan Nila (Oreochromis niloticus) Pada Sistem Resirkulasi Dengan Filter Zeolit".

Freund, Y. and R. E. Schapire (1996). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting".

Ho, W. T., H. W. Lim, and Y. H. Tay (2009). "Two-stage License Plate Detection using Gentle Adaboost and SIFT-SVM".

Papageorgiou, C. P., M. Oren, and T. Poggio (1998). "A General Framework for Object Detection".

Renno, J.-P., D. Makris, and G. A. Jones (2007). "Object Classification in Visual Surveillance Using Adaboost".

Rusydi, M. I. (2019). "Perancangan Mesin Penghitung Benih Ikan Otomatis untuk Membantu Kinerja Peternak Ikan".

Sunartono (2023). *Fantastis! Menteri KKP Sebut Nilai Pasar Ikan Capai Rp2.400 Triliun*. URL: https://news.harianjogja.com/read/2023/02/28/500/1127655/fantastis-menteri-kkp-sebut-nilai-pasar-ikan-capai-rp2400-triliun (visited on 08/08/2023).

Viola, P. and M. J. Jones (2004). "Robust Real-Time Face Detection".

Weber, B. (2006). "Generic Object Detection using AdaBoost".