

**PROTOTYPE SYSTEM PENDETEKSI SPESIES IKAN  
MENGUNAKAN 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**

NIP. 19770615 200312 1 001

**Med Irzal, M.Kom**

NIP. 19851223 201212 1 002

Mengetahui,

Koordinator Program Studi Ilmu Komputer

**Dr. Ria Arafiah, M.Si**

NIP. 19751121 200501 2 004

## KATA PENGANTAR

Puji dan Syukur penulis panjatkan kepada kehadiran 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 lingkungan FMIPA Universitas Negeri Jakarta
2. Yth. Ibu Dr. Ria Arafiah, 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 dijadikan 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 program 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

<b>DAFTAR ISI</b>	<b>vi</b>
<b>DAFTAR GAMBAR</b>	<b>viii</b>
<b>DAFTAR TABEL</b>	<b>ix</b>
<b>I PENDAHULUAN</b>	<b>1</b>
1.1 Latar Belakang Masalah . . . . .	1
1.2 Rumusan Masalah . . . . .	3
1.3 Batasan Masalah . . . . .	3
1.4 Tujuan Penelitian . . . . .	3
1.5 Manfaat Penelitian . . . . .	4
<b>II KAJIAN PUSTAKA</b>	<b>5</b>
2.1 Pengertian Klasifikasi Objek . . . . .	5
2.2 <i>Viola Jones Object Detection Framework</i> . . . . .	5
2.2.1 <i>Features</i> . . . . .	6
2.2.2 <i>Adaboost</i> . . . . .	6
2.2.3 <i>Weaklearn</i> . . . . .	8
2.2.4 <i>Attentional Cascade</i> . . . . .	9
<b>III METODOLOGI PENELITIAN</b>	<b>11</b>
3.1 Tahapan Penelitian . . . . .	11
3.2 Desain Sistem . . . . .	11
3.3 <i>Training Strong Classifier</i> . . . . .	12
3.3.1 <i>Input Dataset</i> Pelatihan . . . . .	13
3.3.2 Pembuatan <i>Haar like Features</i> . . . . .	15
3.3.3 Pembuatan <i>Decision Tree</i> . . . . .	16
3.3.4 <i>Boosting</i> . . . . .	17
3.3.5 Pembuatan <i>Attentional Cascade</i> . . . . .	18
3.4 Skenario Eksperimen dan Validasi . . . . .	19
3.4.1 <i>Pre-processing</i> . . . . .	19
3.4.2 Klasifikasi . . . . .	19
3.4.3 Anotasi . . . . .	21

<b>IV HASIL DAN PEMBAHASAN</b>	<b>22</b>
4.1 <i>Training Strong Classifier</i> . . . . .	22
4.1.1 <i>Input Gambar dan labeling</i> . . . . .	22
4.1.2 <i>Generate Haar-like Features</i> . . . . .	23
4.1.3 <i>Calculating all features of all images</i> . . . . .	24
4.1.4 <i>Create Decision Tree for each Feature</i> . . . . .	27
<b>DAFTAR PUSTAKA</b>	<b>34</b>

## DAFTAR GAMBAR

Gambar 2.1	Beberapa <i>Haar-like features</i> yang digunakan framework Viola-Jones. . . . .	6
Gambar 2.2	<i>Workflow</i> dari <i>Attentional Cascade</i> . . . . .	9
Gambar 3.1	Diagram alir untuk algoritma pelatihan klasifikasi objek . . .	11
Gambar 3.2	Diagram alir untuk algoritma pendeteksian objek . . . . .	11
Gambar 3.3	Contoh gambar Abudehduf, Amphiprion, Chaetodon dan contoh gambar-gambar negatif . . . . .	13
Gambar 3.4	<i>sub-window</i> setiap kelas ikan yang akan dipelajari oleh <i>classifier</i> . . . . .	14
Gambar 3.5	Sebuah fitur dua persegi menghadap ke kiri, lokasi $x = 12$ piksel, lokasi $y = 10$ piksel, dengan ukuran $8 \times 8$ piksel. . . .	15
Gambar 3.6	Contoh sebuah <i>decision tree</i> dengan kelas 0, 1, 2 dan 3 . . . .	16
Gambar 3.7	titik awal <i>sliding window</i> (kotak hijau) dan titik akhir (kotak kuning) . . . . .	21
Gambar 3.8	Gambaran <i>sub-window</i> yang berhasil mengklasifikasi ikan didalam gambar . . . . .	21
Gambar 4.1	<i>source code: read</i> gambar, labelisasi, pengubahan ke <i>greyscale</i> , dan memastikan ukuran gambar $350 \times 200$ piksel .	22
Gambar 4.2	<i>source code: labelisasi</i> gambar sesuai dengan foldernya . . .	22
Gambar 4.3	<i>source code: load</i> gambar-gambar dari folder yang bersangkutan dan menggabungkannya . . . . .	23
Gambar 4.4	<i>source code: feature</i> memiliki semua informasi yang diperlukan untuk melakukan perhitungan. Tipe-tipe fitur akan menentukan rumus perhitungan fitur tersebut . . . . .	23
Gambar 4.5	<i>source code: mengambil</i> semua gambar, label, fitur dan mengkalkulasi semua fitur untuk ketiga <i>sub-window</i> . . . . .	24
Gambar 4.6	<i>source code: class Dataset</i> . . . . .	25
Gambar 4.7	<i>source code: offset</i> ini di-inisialisasi untuk setiap kelas Dataset sehingga bisa diakses langsung oleh fungsi <i>Find_Feature_Value</i> . . . . .	26

Gambar 4.8	<i>source code: offset</i> ini di-inisialisasi untuk setiap kelas Dataset sehingga bisa diakses langsung oleh fungsi <i>Find_Feature_Value</i> . . . . .	27
Gambar 4.9	<i>source code: offset</i> ini di-inisialisasi untuk setiap kelas Dataset sehingga bisa diakses langsung oleh fungsi <i>Find_Feature_Value</i> . . . . .	28
Gambar 4.10	<i>source code: get data dan readcsv</i> yang digunakan oleh <i>split data</i> . . . . .	28
Gambar 4.11	<i>source code: untuk membuat semua decision tree</i> untuk setiap fitur . . . . .	29
Gambar 4.12	<i>source code: class node</i> digunakan untuk menyimpan informasi cabang dan <i>threshold</i> pada <i>node decision tree</i> . . .	29
Gambar 4.13	<i>source code: class</i> digunakan untuk menyimpan tinggi maksimal dan minimal <i>split</i> pada <i>decision tree</i> . Semua data lainnya disimpan pada node . . . . .	30
Gambar 4.14	<i>source code: fungsi utama</i> dari <i>class DecisionTreeClassifier</i> .	30
Gambar 4.15	<i>source code: fungsi get_best_split</i> . . . . .	31
Gambar 4.16	<i>source code: fungsi split</i> hanya bertugas membagi berdasarkan <i>threshold</i> yang sudah ditemukan . . . . .	31
Gambar 4.17	<i>source code: information_gain</i> mencari data dengan menghitung <i>gini purity</i> . . . . .	32
Gambar 4.18	<i>source code: mencari data</i> dengan menghitung <i>entropy</i> . . .	32
Gambar 4.19	<i>source code: mencari data</i> dengan menghitung <i>gini</i> . . . . .	32
Gambar 4.20	<i>source code: fungsi untuk mencari mayoritas kelas</i> pada <i>leaf node</i> . . . . .	32
Gambar 4.21	<i>source code: fungsi fit</i> adalah fungsi yang dipanggil untuk mulai membangun <i>decision tree</i> setelah dibuat . . . . .	33
Gambar 4.22	<i>source code: predict</i> digunakan untuk melakukan prediksi dengan <i>decision tree</i> yang sudah dibuat . . . . .	33



## **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 maksimal, 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 skala kecil, berbeda dengan negara tetangga seperti Vietnam yang menguasai ekspor 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 penghitungan 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 photodiode. 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, Rusydi 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 menunjukkan jati diri objek tersebut, misalnya objek bola sepak pastilah bulat dan umumnya memiliki dua warna yaitu hitam dan putih dengan pola yang spesifik. Muka manusia memiliki mata, hidung dan mulut yang dapat dibedakan dengan makhluk lainnya misalnya dengan kucing. Metode-metode deteksi objek umumnya menggunakan pendekatan neural network dan *non-neural network* untuk mendefinisikan fitur dari kelas objek yang berusaha dideteksi.

Adaboost (Freund 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 Object Detection Framework*. *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 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 Object Detection Framework*?'**.

## 1.3 Batasan Masalah

Batasan masalah pada penelitian ini adalah:

1. Klasifikasi ikan menggunakan *Viola-Jones Object Detection Framework*.
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 membuat program yang mampu mengklasifikasi 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**

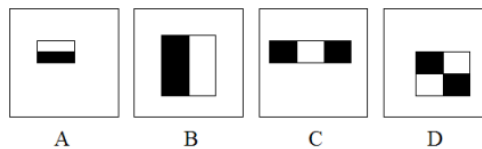
#### **2.1 Pengertian Klasifikasi Objek**

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 *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 memungkinkan 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 perbedaan 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. Sebuah *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

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

---

**Algorithm 1** Algoritma Adaboost
 

---

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

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

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

3. Definisikan  $h_t(x) = h(x, f_t, p_t, \theta_t)$  dimana  $f_t, p_t$ , dan  $\theta_t$  adalah *minimizer* dari  $\epsilon_t$ .

4. Perbarui bobot:

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

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

- 4: *strong classifier* akhirnya adalah:

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

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

---



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):

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

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*.

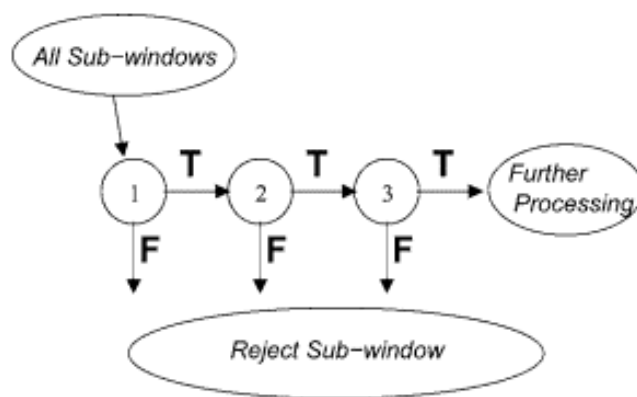
*Decision Tree* menghasilkan sebuah classifier didalam bentuk sebuah pohon pilihan, sebuah struktur yang berbentuk:

- 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.

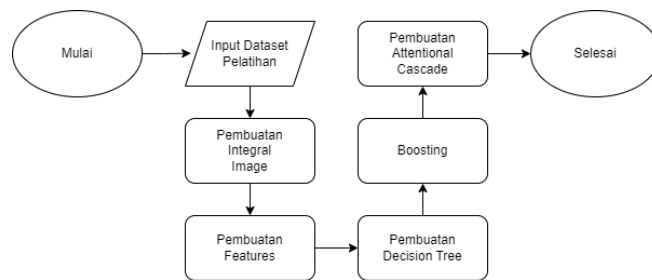
*Cascade* dimulai dengan membuat sebuah *stage* awal, dimana *weak classifier* ditambahkan secara perlahan hingga *detection rate* yang dituju sampai. *Detection rate* ini harus disesuaikan oleh pengguna sesuai dengan keperluannya, tujuannya tetap untuk mengurangi waktu komputasi tapi juga mencoba agar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

## BAB III

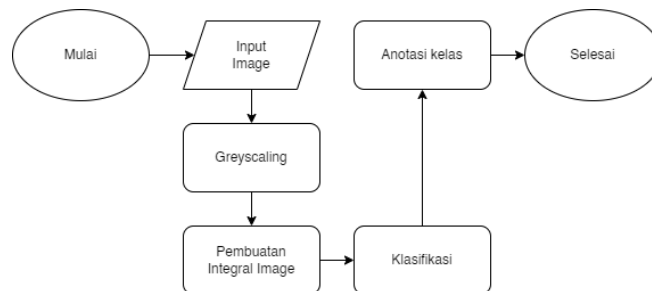
### METODOLOGI PENELITIAN

#### 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.

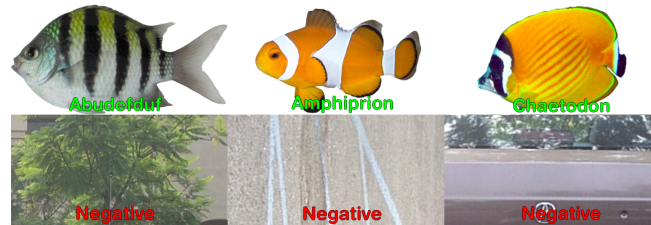
Pertama sebuah set *features* akan dibuat dengan cara mencoba semua kemungkinan yang ada dengan bentuk fitur yang dimiliki. Set ini akan berisikan informasi fitur-fitur yang nantinya akan dipakai untuk mendapatkan nilai fitur yang sesungguhnya. Set gambar latihan lalu akan dibaca menggunakan semua fitur 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 classifier*. Akhirnya 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 pembuatan *attentional cascade*.

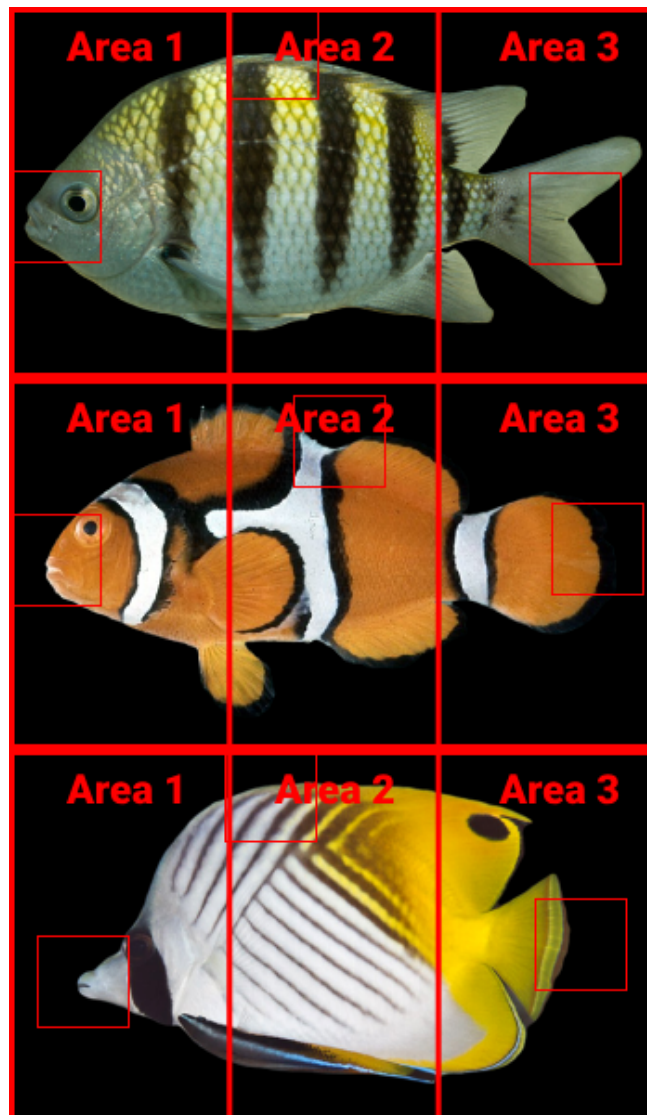
### 3.3.1 *Input Dataset Pelatihan*



**Gambar 3.3:** Contoh gambar Abudedefduf, 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 Abudedefduf, 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 Abudedefduf, 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 mengklasifikasi 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*

---

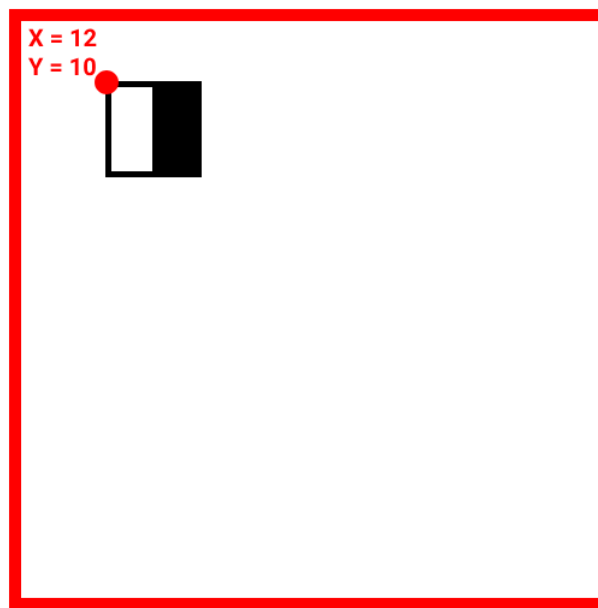
**Algorithm 3** offset *sub-window* untuk training setiap kelas

---

- 1: class\_Window\_offset\_1  $\leftarrow$  [  $\triangleright$  For searching mouth features (0, 0), (88, 0), (73, 0), (100, 15) ]
  - 2: class\_Window\_offset\_2  $\leftarrow$  [  $\triangleright$  For searching fin features (0, 0), (0, 116), (9, 153), (0, 116) ]
  - 3: class\_Window\_offset\_3  $\leftarrow$  [  $\triangleright$  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  $8 \times 8$  piksel.

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

$$\sum \text{nilai piksel area putih} - \sum \text{nilai piksel area hitam} \quad (3.1)$$

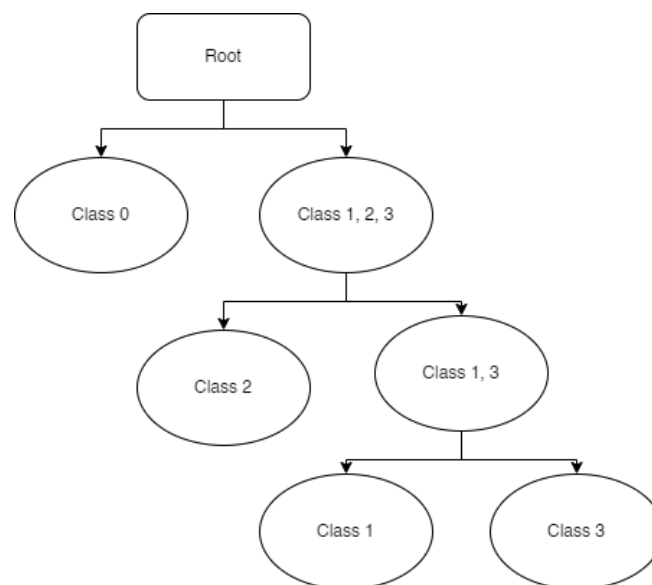
Pada tahap ini fitur yang dipilih akan dibuat untuk semua kemungkinan lokasi yang ada dan ukuran yang ada. Hal ini dilakukan dengan membuat fitur dimulai dari kiri atas gambar dengan ukuran  $2 \times 2$  piksel untuk fitur dua persegi, empat persegi dan diagonal. Dan fitur  $1 \times 3$  piksel atau  $3 \times 1$  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*  $\alpha$  dari sebuah *weak classifier* adalah sebagai berikut:

$$\epsilon = \frac{\sum_i \text{image weights}_i \times \text{indikator}_i}{\sum_i \text{image weights}_i} \quad (3.2)$$

$$\alpha = 0.5 \times \log \left( \frac{1 - \epsilon}{\epsilon + 1e - 10} \right)$$

*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 menaikkan bobot:

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

$$\text{image weights} \div = \sum \text{image weights} \quad (3.4)$$

*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 classifier* yang dibuat dengan iterasi sebelumnya. Bila didapat bahwa ada penurunan akurasi, maupun tidak ada perubahan, maka iterasi Boosting akan disudahi dan *strong classifier* 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 classifier* 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  $\leftarrow$  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  $\leftarrow$  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 classifier* dari semua *stage*.

### 3.4 Skenario Eksperimen dan Validasi

Tahapan ini adalah penggunaan *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 *grayscale*, 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 melalui proses *pre-processing* dan diubah ke dalam warna *grayscale*. Selain itu resolusi gambar akan diubah 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* mengklasifikasi 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 menghitung 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,  $x_{\text{offset}}$ ,  $y_{\text{offset}}$ )
2:   scoreboard  $\leftarrow$  array([0, 0, 0, 0])
3:   for  $i$  in range(len(self.stages)) do
4:     stage_scoreboard  $\leftarrow$  self.stages[ $i$ ].stage_prediction()
5:     if stage_scoreboard.argmax() == 0 then break
6:     else
7:       scoreboard  $\leftarrow$  scoreboard + stage_scoreboard
8:     end if
9:   end for
10:  return scoreboard.argmax()
11: end function

```

---



---

**Algorithm 6** Stage Prediction
 

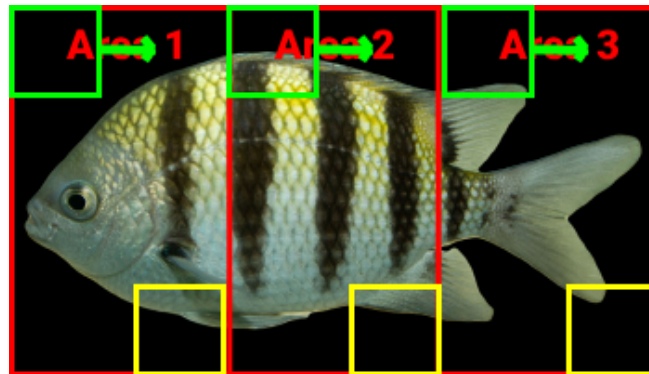
---

```

1: function STAGEPREDICTION(image,  $x_{\text{offset}}$ ,  $y_{\text{offset}}$ , scoreboard)
2:   for  $i$  in range(len(self.features)) do
3:     feature_type,  $x$ ,  $y$ , width, height  $\leftarrow$  self.features[ $i$ ]
4:      $x \leftarrow x + x_{\text{offset}}$ 
5:      $y \leftarrow y + y_{\text{offset}}$ 
6:     updated_feature  $\leftarrow$  (feature_type,  $x$ ,  $y$ , width, height)
7:     data_features  $\leftarrow$  compute_feature_with_matrix(image, 0, updated_feature)
8:     prediction  $\leftarrow$  self.trees[ $i$ ].predict(data_features)
9:     scoreboard[prediction]  $\leftarrow$  scoreboard[prediction] +  $1 \times$  self.alpha_list[ $i$ ]
10:  end for
11:  return scoreboard
12: end function

```

---



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

### 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.

## 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
    abudefdud_images, abudefdud_labels = load_images("fish_dataset\\abudefdud")
    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((abudefdud_images, amphiprion_images, chaetodon_images, negatives_images), axis =
0)
    labels = np.concatenate((abudefdud_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 dimasukkan kedalam sebuah dokumen .CSV menggunakan *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 untuk ketika kelas ikan, semuanya memiliki mulut, sirip dan ekor yang sama satu dengan yang lainnya. Contohnya: Mulut dari spesies ikan *Chaetodon* agak lebih kebawah dibandingkan kedua kelas ikan lainnya. 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),
    (88, 0),
    (73, 0),
    (100, 15)
]

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

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

```

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

Untuk perhitungan fitur yang sebenarnya 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-inisialisasi 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 dimasukkan 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-inisialisasi untuk setiap kelas Dataset sehingga bisa diakses langsung oleh fungsi *Find\_Feature\_Value*

Label baru ditambahkan sekarang agar tidak mengganggu 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 dibuat 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
    for i in range(len(features)):
        if i % 1000 == 0: print(f'starting tree {i}')
        temp_X_train = X_train[:, i].reshape(-1, 1)
        # print(temp_X_train)
        # print(Y_train)
        classifier = DecisionTreeClassifier(min_samples_split=3, max_depth=3)
        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* diinisialisasi 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 berikut 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, min_samples_split=2, max_depth=2):
        self.root = None
        # stopping condition
        self.min_samples_split = min_samples_split
        self.max_depth = max_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, dataset, current_depth = 0):
    X, Y = dataset[:, :-1], 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 current 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, right_y = dataset[:, -1], dataset_left[:, -1], dataset_right[:, -1]
                # compute information gain
                current_info_gain = self.information_gain(y, left_y, right_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_right = np.array([row for row in dataset if row[feature_index] > threshold])
    return dataset_left, dataset_right

```

**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

```
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

## 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".