Perancangan dan Implementasi *Cloud Computing* untuk Deteksi Kesegaran Ikan Menggunakan Model Deep Learning YOLOv8 Pada Aplikasi FishQ

1st Rifqi Fadhilah Firdaus
Fakultas Teknik Elektro
Universitas Telkom
Bandung, Indonesia
rifqifadhilahfirdaus@student.telkomuniversity.ac.id

2nd Ledya Novamizanti
Fakultas Teknik Elektro
Universitas Telkom
Bandung, Indonesia
ledyaldn@telkomuniversity.ac.id

3rd Suryo Adhi Wibowo
Fakultas Teknik Elektro
Universitas Telkom
Bandung, Indonesia
suryoadhiwibowo@telkomuniversity.ac.id

***Abstrak* — Sebagai solusi dari permasalah sortasi ikan, dilakukan pengembangan dan pengimplementasian aplikasi FishQ yang menggunakan teknologi *cloud computing* dan model *deep learning* YOLOv8 untuk mendeteksi kesegaran ikan cakalang. FishQ dirancang untuk meningkatkan efisiensi dan akurasi dalam proses sortasi ikan yang selama ini dilakukan secara manual dan rentan terhadap kesalahan. Pengujian dilakukan pada 30 sampel ikan cakalang dalam kondisi beku dan tidak beku, dengan kategori segar, tidak segar, dan *multiple*. Hasil pengujian menunjukkan bahwa sistem mampu mendeteksi kesegaran ikan dengan akurasi tinggi. Analisis hasil pengujian menunjukkan bahwa sistem *cloud computing* yang dirancang mampu mendeteksi kesegaran ikan dengan efisien dan akurat, terutama lebih cepat pada ikan dalam kondisi beku. Secara keseluruhan, aplikasi FishQ diharapkan dapat membantu perusahaan perikanan dalam meningkatkan efisiensi dan akurasi proses sortasi ikan, sehingga dapat meningkatkan kualitas produk perikanan yang dijual.**

***Kata kunci*— FishQ, *cloud computing*, YOLOv8, deteksi kesegaran ikan, deep learning, sortasi ikan.**

I. PENDAHULUAN

Industri perikanan di Indonesia memiliki peran penting dalam perekonomian nasional. Namun, proses sortasi ikan yang dilakukan secara manual sering kali tidak efisien dan rentan terhadap kesalahan. Hal ini dapat menyebabkan penurunan kualitas produk dan kerugian ekonomi karena ikan yang tidak segar bisa lolos dari proses sortasi [1, 2, 3]. Untuk mengatasi masalah ini, teknologi *cloud computing* dan model *deep learning* seperti YOLOv8 dapat digunakan. YOLOv8 adalah model deteksi objek yang cepat dan akurat, yang dapat membantu dalam mengidentifikasi kesegaran ikan secara otomatis. Dengan memanfaatkan *cloud computing*, proses deteksi dapat dilakukan dengan cepat dan efisien tanpa memerlukan perangkat keras yang mahal.

Penelitian ini bertujuan untuk mengembangkan aplikasi FishQ yang menggunakan teknologi *cloud computing* dan model YOLOv8 untuk mendeteksi kesegaran ikan cakalang. Aplikasi ini diharapkan dapat membantu perusahaan perikanan, seperti Aruna, dalam meningkatkan efisiensi dan akurasi proses sortasi ikan. Dataset yang digunakan mencakup citra ikan cakalang dalam kondisi beku dan tidak beku, yang diperoleh dari berbagai sumber. Implementasi *cloud computing* dalam aplikasi ini menggunakan layanan Google Cloud Platform (GCP) yang menyediakan infrastruktur yang fleksibel dan skalabel untuk menjalankan model *deep learning*. Komponen *cloud computing* memanfaatkan FastAPI, API Endpoint, Docker, dan Cloud Run untuk memproses citra yang diunggah dan menjalankan model *deep learning*.

Dengan adanya aplikasi FishQ, diharapkan proses penilaian kesegaran ikan dapat dilakukan dengan lebih cepat dan akurat, sehingga meningkatkan kualitas produk perikanan dan mengurangi ketergantungan pada penilaian manual [4]. Aplikasi ini tidak hanya meningkatkan kualitas produk ikan yang dijual tetapi juga mengurangi ketergantungan pada sumber daya manusia dalam proses sortasi ikan secara manual.

II. KAJIAN TEORI

Dalam pengembangan aplikasi FishQ untuk deteksi kesegaran ikan menggunakan model *deep learning* YOLOv8, berbagai teknologi modern digunakan untuk memastikan sistem bekerja dengan efisien dan andal. Bagian ini akan membahas teori-teori yang berkaitan dengan variabel-variabel penelitian, termasuk FastAPI, API Endpoint, Docker, Google Cloud Platform, Google Cloud Run, Google Cloud Run, JSON *Response* dan Postman.

A. FastAPI

FastAPI adalah *framework* Python yang dirancang untuk membangun API dengan performa tinggi. Dalam konteks *cloud computing*, FastAPI menawarkan pengembangan API yang cepat dan efisien, serta skalabilitas aplikasi yang tinggi [5]. Dengan dukungan untuk operasi asinkron dan kecepatan yang bersaing dengan NodeJS dan Starlette, FastAPI menjadi pilihan ideal untuk pengembangan aplikasi berbasis *cloud* yang membutuhkan waktu respons cepat, pengelolaan sumber daya yang efisien, dan kemampuan menangani permintaan dalam jumlah besar.

B. API *Endpoint*

Dalam implementasi aplikasi FishQ, API Endpoint adalah *Uniform Resource Locator* (URL) yang berfungsi sebagai penghubung antara pengguna dan layanan *cloud*. API *Endpoint* ini berperan sebagai titik masuk untuk

menerima citra yang diunggah oleh pengguna dan memulai proses pengolahan data [6]. Ketika pengguna mengunggah citra ikan melalui aplikasi, API Endpoint menerima citra tersebut, memprosesnya dengan model *deep learning* yang berjalan di *cloud*, dan mengembalikan hasil deteksi dalam format JSON. Proses ini memungkinkan aplikasi FishQ untuk memberikan hasil deteksi kesegaran ikan secara cepat dan efisien [7, 8].

C. Docker

Docker adalah platform yang memungkinkan pengembang untuk mengemas aplikasi dan dependensinya ke dalam satu paket yang disebut *container*. Docker akan membuat aplikasi FastAPI FishQ yang dijalankan di Cloud Run dari Google Cloud Platform. Prosesnya dimulai dengan membuat Dockerfile yang mendeklarasikan *environment* dan instruksi aplikasi FastAPI FishQ [9, 10]. Setelah itu, image Docker tersebut di-*build* dan di-*push* ke Docker Hub. Terakhir, *image* ini digunakan oleh Cloud Run untuk *hosting* aplikasi secara otomatis dengan skala sesuai kebutuhan untuk memastikan aplikasi FastAPI FishQ dapat diakses oleh pengguna.

D. Google Cloud Platform

Google Cloud Platform (GCP) adalah layanan *cloud computing* yang menyediakan infrastruktur dan alat untuk mengembangkan, menguji, dan menjalankan aplikasi di *cloud* [11]. Dalam pengembangan aplikasi FishQ, GCP digunakan untuk mengimplementasikan model *deep learning* YOLOv8 guna mendeteksi kesegaran ikan. Dengan menggunakan GCP, semua data pengguna, model *machine learning*, dan API dapat terhubung langsung dengan *cloud* [12]. Hal ini memudahkan integrasi dengan aplikasi *mobile* Android, sehingga proses deteksi kesegaran ikan dapat dilakukan dengan cepat dan efisien [13].

E. Google Cloud Run

Cloud Run adalah layanan *serverless* dari Google Cloud Platform yang memungkinkan pemrosesan citra secara efisien [14]. Dalam konteks aplikasi FishQ, *container* yang dijalankan di Cloud Run berisi model *deep learning* yang bertugas menganalisis citra untuk mendeteksi kesegaran ikan cakalang, baik dalam kondisi beku maupun tidak beku. Pengguna dapat mengunggah citra ikan melalui aplikasi, yang kemudian dikirim ke Cloud Run untuk diproses. Model *deep learning* di dalam *container* akan menganalisis citra tersebut dan mengembalikan hasil deteksi dalam format JSON [15]. Hasil ini kemudian digunakan oleh aplikasi untuk menentukan label kesegaran ikan, seperti "fresh" atau "No.-fresh", sesuai dengan kebutuhan pengguna.

F. JSON *Response*

JSON *response* adalah format data yang dikirim oleh *cloud* ke pengguna aplikasi dalam bentuk JSON. Dalam konteks aplikasi FishQ, JSON *response* digunakan untuk mengirimkan hasil deteksi objek dari *cloud* ke aplikasi pengguna [16, 17]. Variabel "response_data" yang digunakan dalam JSON *response* ini mencakup beberapa parameter penting seperti "total", "fresh", "No._fresh", dan "image_data". Parameter total menunjukkan jumlah total objek yang terdeteksi, "fresh" menunjukkan jumlah objek yang segar, "No._fresh" menunjukkan jumlah objek yang tidak segar, dan "image_data" berisi data citra dalam format Base64.

G. Postman

Postman adalah alat yang sangat berguna untuk menguji API. Dengan Postman, pengembang dapat mengirim permintaan HTTP ke server dan menerima respons untuk memastikan API berfungsi dengan benar [18]. Dalam aplikasi FishQ, Postman digunakan untuk menguji endpoint API yang memproses citra ikan yang diunggah oleh pengguna. Endpoint ini menerima permintaan POST yang berisi citra ikan, memprosesnya menggunakan model YOLOv8 di *cloud*, dan mengembalikan hasil deteksi dalam format JSON [19]. JSON *response* ini mencakup informasi seperti jumlah total ikan yang terdeteksi ("total"), jumlah ikan segar ("fresh"), jumlah ikan tidak segar ("No._fresh"), dan data citra hasil deteksi ("image_data"). Dengan menggunakan Postman, pengembang dapat memverifikasi respons yang diterima dan mencatat waktu respons untuk mengevaluasi kinerja API, memastikan bahwa aplikasi FishQ dapat memberikan hasil deteksi yang cepat dan akurat.

III. PERANCANGAN SISTEM

Perancangan sistem menjelaskan gambaran rancangan dalam pembuatan sistem. Dalam perancangan sistem ini terdapat beberapa perancangan yang akan dijelaskan. Perancangan berupa Perancangan Penggunaan *Cloud Computing* (Google Cloud Platform), Perancangan API *Endpoint* untuk Menerima dan Memproses Citra Ikan, Penggunaan Docker untuk Mengemas Aplikasi, Cloud Run untuk Menjalankan *Container* di Lingkungan *Serverless*, JSON *Response* untuk Mengirim Hasil Deteksi ke Aplikasi dan Pengujian Menggunakan Postman dan Cloud Run.

A. Penggunaan *Cloud Computing* (Google Cloud Platform)

Cloud Run dipilih untuk aplikasi FishQ karena menawarkan berbagai keuntungan yang mendukung efisiensi dan keandalan sistem. Cloud Run adalah layanan *serverless* dari Google Cloud Platform (GCP) yang memungkinkan aplikasi berjalan dalam *container* tanpa perlu mengelola server. Dengan menggunakan Cloud Run, aplikasi FishQ dapat memproses citra ikan dengan cepat dan efisien, memastikan hasil deteksi yang akurat dalam waktu singkat, dan memberikan pengalaman pengguna yang responsif dan memuaskan. Untuk alur kerja dari *cloud computing* dapat dilihat pada Gambar 1.

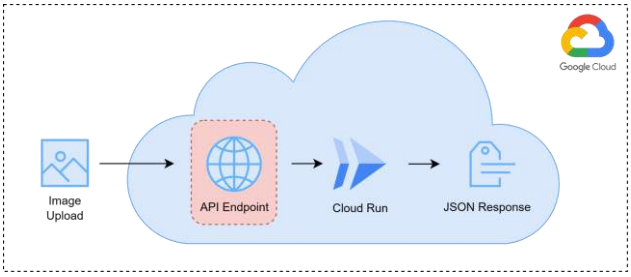


Fig. 1.
Proses pada *Cloud Computing*

Alur kerja dari *cloud computing* dimulai ketika pengguna mengunggah citra melalui kamera atau galeri, yang kemudian diunggah ke sistem yang terhubung dengan layanan *cloud*. Setelah citra berhasil diunggah, langkah selanjutnya adalah pengiriman citra ke API *Endpoint* yang berperan sebagai gerbang penerima citra dan mempersiapkannya untuk diproses lebih lanjut. Pemrosesan citra dilakukan oleh Cloud Run, yang menjalankan *container* berisi model *deep learning* untuk mendeteksi dan menganalisis citra ikan, baik dalam kondisi beku maupun tidak beku, sesuai pilihan pengguna di aplikasi. Setelah model *deep learning* selesai menganalisis citra, hasilnya dikirim kembali ke aplikasi pengguna menggunakan metode POST, sehingga pengguna dapat langsung melihat hasil analisis citra yang telah diproses oleh model *deep learning*.

B. Perancangan API *Endpoint* untuk Menerima dan Memproses Citra Ikan

Dalam konteks aplikasi FishQ, FastAPI digunakan untuk membangun API Endpoint yang berfungsi sebagai gerbang penerima citra ikan yang diunggah oleh pengguna. API *Endpoint* ini memanfaatkan kemampuan asinkron FastAPI untuk menangani permintaan dalam jumlah besar dengan waktu respons yang cepat. FastAPI juga mendukung dokumentasi otomatis menggunakan OpenAPI dan JSON *Schema*, yang memudahkan pengembang dalam mengelola dan menguji API. Berikut merupakan fungsi dari API *Endpoint*.

```
@app.post("/img_object_detection")
def img_object_detection(mode: str, file: bytes = File(...)):
    input_image = get_image_from_bytes(file)
    predict = detect_sample_model(input_image, mode)
    final_image = add_bboxs_on_img(image = input_image,
    predict =predict)

    total = len(predict['name'])
    fresh = len(predict[predict['name'] == 'fresh'])
    No._fresh = len(predict[predict['name'] == 'No.-fresh'])
    image_bytes = get_bytes_from_image(final_image)
    bytes = image_bytes.getvalue()
    image_result = {'image_data':
    base64.b64encode(bytes).decode('utf-8')}

    response_data = {
        'total':total,
        'fresh':fresh,
        'No._fresh':No._fresh,
        **image_result
    }
    return response_data
```

Fungsi diatas merupakan bagian dari API *Endpoint*. Metode POST pada URL "/img_object_detection" menerima parameter "mode" dan *file* citra dalam bentuk *bytes* untuk memproses *input* data dari aplikasi. Fungsi ini kemudian melakukan deteksi untuk membedakan objek "fresh" dan "No.-fresh", memberikan label pada citra, menghitung jumlah objek, dan mengembalikan hasil deteksi dalam format JSON.

C. Penggunaan Docker untuk Mengemas Aplikasi

Docker adalah platform yang memungkinkan pengembang untuk mengemas aplikasi beserta semua dependensinya ke dalam sebuah *container*. *Container* ini berfungsi sebagai unit yang dapat dijalankan di berbagai lingkungan tanpa perlu khawatir tentang perbedaan konfigurasi sistem. Dalam pengembangan aplikasi FishQ, Docker digunakan untuk membuat *container* yang berisi aplikasi FastAPI dan model *deep learning* YOLOv8. Gambar 2 merupakan *repository* pada Docker Hub yang digunakan yaitu "abangal3/fishq" dengan versi *No.* yang dipakai yaitu "1.1" dan dikategorikan sebagai tipe "Image".

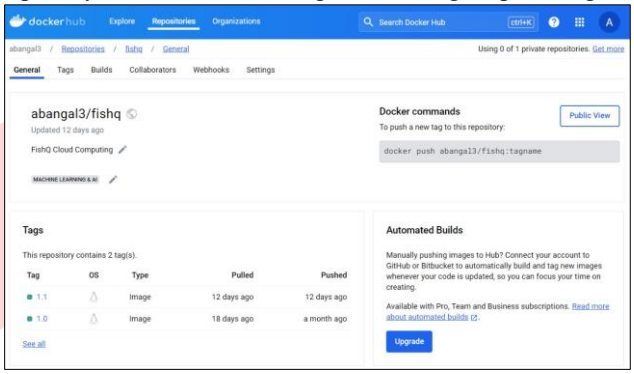


Fig. 2.
Repository pada Docker Hub

Pada proses pembuatan dan pengelolaan *container* pada Docker ada beberapa tahapan diantaranya:

1. Membuat Dockerfile : Dockerfile dibuat untuk mendefinisikan langkah-langkah pembuatan image Docker, seperti instalasi dependensi dan konfigurasi *server*. Berikut merupakan kode dari Dockerfile. Dockerfile dibawah dirancang untuk memastikan bahwa semua yang diperlukan untuk menjalankan aplikasi telah terpasang dan aplikasi dijalankan menggunakan server Uvicorn.

```
FROM python:3.10-slim

WORKDIR /app
COPY . /app

RUN apt update && \
    apt install -y htop libgl1-mesa-glx libglib2.0-0

COPY requirements.txt /tmp/requirements.txt
RUN pip install --no-cache-dir -r /tmp/requirements.txt

CMD uvicorn main:app --port=8000 --host=0.0.0.0
```

2. Membangun *Image* Docker: Perintah "docker build -t fishq:1.1" . digunakan untuk membangun *image* Docker dari Dockerfile. *Image* ini kemudian di-*push* ke Docker Hub agar dapat diakses oleh layanan *cloud*.

3. Mengelola *Container* dengan Docker *Compose*: Docker *Compose* digunakan untuk mendefinisikan dan mengelola layanan *multi-container*. Berikut adalah kode dari "docker-compose.yml" yang digunakan.

```
Version: '3'
```

```
services:
  web:
    build: .
    restart: "always"
    volumes:
    - ./:/app
    working_dir: /app
    ports:
      - "8000:8000"
    command: uvicorn main:app --reload --host 0.0.0.0 -
-port 8000
```

4) *Deploy* ke Cloud Run: *Image* Docker yang telah di-*build* dan di-*push* ke Docker Hub digunakan oleh Cloud Run untuk *hosting* aplikasi secara otomatis dengan skala sesuai kebutuhan. Cloud Run menjalankan *container* berisi model *deep learning* dan API Endpoint, memastikan aplikasi dapat diakses oleh pengguna dengan performa yang optimal. Dengan menggunakan Docker, aplikasi FishQ dapat dikemas dan dikelola dengan efisien, memastikan portabilitas, konsistensi, dan skalabilitas yang tinggi.

D. Cloud Run untuk Menjalankan *Container* di Lingkungan *Serverless*

Cloud Run adalah layanan *serverless* dari GCP yang memungkinkan pengembang untuk menjalankan *container* secara otomatis tanpa perlu mengelola server. Cloud Run menjalankan *container* yang dikemas dengan aplikasi dan dependensinya, sehingga aplikasi dapat berjalan dengan konsisten di berbagai lingkungan. Layanan ini menerima permintaan HTTP dan memprosesnya menggunakan *container* yang telah di-*deploy*. Cloud Run secara otomatis menangani penskalaan aplikasi berdasarkan lalu lintas yang diterima, sehingga aplikasi dapat menangani beban kerja yang bervariasi dengan efisien.

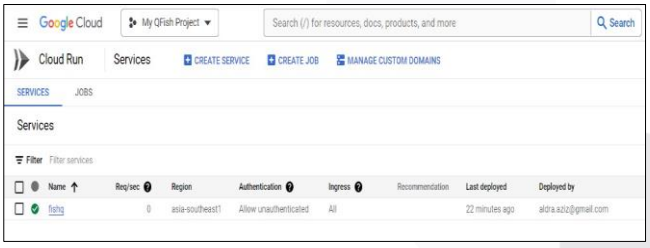


Fig. 3. *Service* yang dibuat pada Cloud Run

Pada Gambar 3 merupakan servis yang dibuat pada Cloud Run dengan nama "fishq". Konfigurasi pada servis ini memakai URL *container* yang didapatkan dari Docker yaitu "abangal3/fishq:1.1" dengan *port* 8000, dan memiliki spesifikasi 2 vCPU (*virtual* CPU) dan 2 GiB (Gibibyte) *Memory*. Rincian spesifikasi juga dapat dilihat pada Tabel I berikut.

TABLE I. SPESIFIKASI *DEPLOY SERVICE* CLOUD RUN

Spesifikasi	Rincian
vCPU	2
Memory Limit (GiB)	2
Protokol	tcp:8000
Image URL	mirror.gcr.io/abangal3/fishq@sha256...

Dalam aplikasi FishQ, Cloud Run digunakan untuk menjalankan *container* yang berisi model *deep learning* YOLOv8 dan API *Endpoint* yang dibangun menggunakan FastAPI. Berikut adalah langkah-langkah implementasinya:

1) Membuat Docker *Container*: Aplikasi FastAPI dan model YOLOv8 dikemas ke dalam *container* menggunakan Docker. Dockerfile digunakan untuk mendefinisikan lingkungan dan dependensi yang diperlukan oleh aplikasi.

2) Men-*deploy Container* ke Cloud Run: *Image* Docker yang telah dibuat di-*push* ke Docker Hub atau Google *Container Registry*. Selanjutnya, *image* ini digunakan oleh Cloud Run untuk menjalankan *container* secara *serverless*.

3) Konfigurasi Cloud Run: Layanan Cloud Run dikonfigurasi untuk menerima lalu lintas dari semua sumber (*Ingress: All*) dan memungkinkan akses tanpa autentikasi (*Authentication: Allow unauthenticated*). Ini memastikan bahwa layanan dapat diakses oleh pengguna aplikasi tanpa hambatan.

4) Pengelolaan dan *Monitoring*: Cloud Run menyediakan fitur *logging* dan *monitoring* yang memungkinkan pengembang untuk memantau performa aplikasi dan mendeteksi masalah secara *real-time*. *Log* hasil *running* pada Cloud Run mencakup informasi koneksi API dari pengguna, jumlah objek ikan yang terdeteksi, dan waktu proses citra ikan.

Dengan menggunakan Cloud Run, aplikasi FishQ dapat memanfaatkan infrastruktur *serverless* yang efisien dan andal, memastikan bahwa proses deteksi kesegaran ikan dapat dilakukan dengan cepat dan akurat. Implementasi ini juga memungkinkan aplikasi untuk diskalakan secara otomatis sesuai dengan kebutuhan, sehingga memberikan pengalaman pengguna yang optimal.

E. JSON *Response* untuk Mengirimkan Hasil Deteksi ke Aplikasi

JavaScript Object Notation (JSON) adalah format data yang digunakan untuk mengirim hasil deteksi dari *cloud* ke aplikasi FishQ. JSON *Response* ini berfungsi untuk mengirimkan informasi hasil deteksi objek dari model *deep learning* yang berjalan di *cloud* kembali ke aplikasi pengguna. Format JSON *Response* mencakup berbagai parameter penting yang memberikan informasi detail mengenai hasil deteksi, seperti jumlah total objek yang terdeteksi, jumlah objek yang segar, jumlah objek yang tidak segar, dan data citra hasil deteksi dalam format Base64. Struktur JSON *Response* yang digunakan dalam aplikasi FishQ terdiri dari beberapa parameter utama, yaitu:

1) "total": Menunjukkan jumlah total objek ikan yang terdeteksi.
2) "fresh": Menunjukkan jumlah ikan yang terdeteksi sebagai segar.
3) "No._fresh": Menunjukkan jumlah ikan yang terdeteksi sebagai tidak segar.
4) "image_data": Berisi data citra hasil deteksi dalam format Base64.

Berikut adalah contoh struktur JSON *Response* yang digunakan:

```
response_data={
    ‘total’:total,
    ‘fresh’:fresh,
    ‘nofresh’:nofresh,
    **image_result
}
return response_data
```

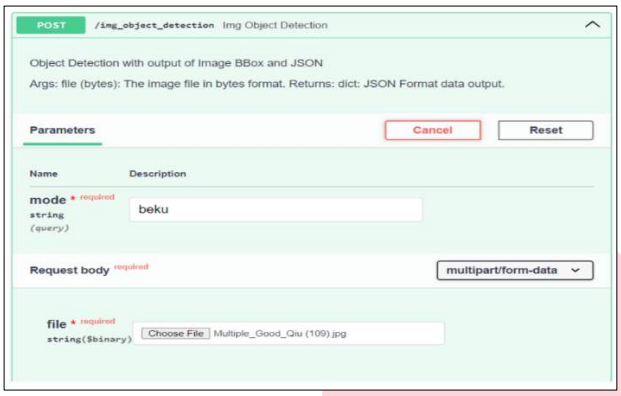


Fig. 4.
JSON API *Request*

Pada Gambar 4, fitur *request* API menunjukkan adanya *query* mode deteksi yang dapat menerima parameter "beku" atau "tidak_beku". Selain itu, pengguna dapat mengunggah *file* citra dalam *request body*, yang kemudian akan diproses untuk deteksi kesegaran ikan.

Fig. 5.
JSON API *Response*

Data JSON *Response* pada Gambar 5 menampilkan hasil deteksi dengan jumlah objek ikan dan data gambar dalam format Base64. Respons server menunjukkan kode status 200, yang berarti permintaan berhasil diproses.

Dengan menggunakan JSON *Response*, aplikasi FishQ dapat mengirim dan menerima data hasil deteksi secara efisien, memastikan bahwa pengguna mendapatkan informasi yang akurat dan *real-time* mengenai kesegaran ikan yang mereka unggah.

F. Pengujian Menggunakan Postman

Postman adalah alat yang digunakan untuk menguji endpoint API dengan mengirimkan permintaan HTTP dan menerima respons. Dalam pengujian aplikasi FishQ, dilakukan pada masing-masing kategori (segar, tidak segar dan *multiple*) dengan 5 citra ikan yang berbeda pada tiap kategori. Postman digunakan untuk mengirim permintaan POST ke *endpoint* “/img_object_detection” dengan menyertakan citra ikan dalam bentuk *file*. Pengujian ini dilakukan untuk memastikan bahwa API dapat menerima dan memproses citra dengan benar, serta mengembalikan hasil deteksi dalam format JSON. Berikut adalah langkah-langkah pengujian menggunakan Postman:

a) Mengirim Permintaan POST : Menggunakan Postman, permintaan POST dikirim ke *endpoint* “/img_object_detection” dengan parameter mode (beku atau tidak beku) dan *file* citra dalam bentuk *bytes*.

b) Memeriksa Respons: Respons dari server yang berisi jumlah total ikan yang terdeteksi (“total”), jumlah ikan segar (“fresh”) dan jumlah ikan tidak segar (“no-fresh”).

1. Pengujian Ikan Cakalang Beku

Pengujian dilakukan pada 30 sampel ikan cakalang dalam kondisi beku yang terdiri dari 10 sampel ikan cakalang segar, 10 sampel ikan cakalang tidak segar, dan 10 sampel ikan cakalang campuran (*multiple*).

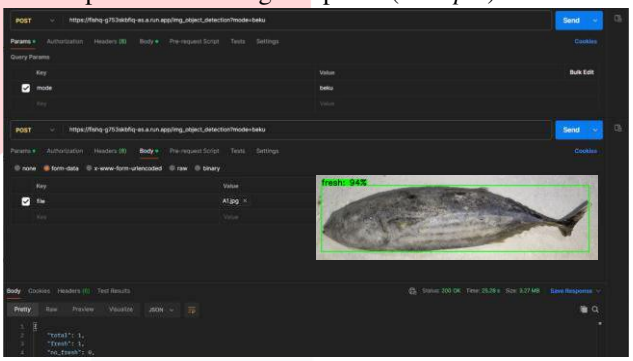


Fig. 6.
Pengujian Ikan Cakalang Beku menggunakan aplikasi Postman

Pada Gambar 6 memperlihatkan metode POST untuk pengujian ikan cakalang beku dengan citra ikan yang sudah tersedia. Untuk IP yang digunakan yaitu “https://fishq-g753skbfiq-as.a.run.app/img_object_detection?Mode=beku”.

a) Ikan Cakalang Segar

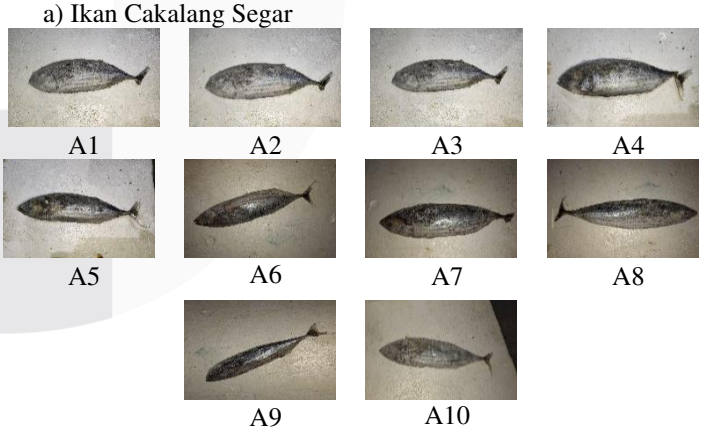


Fig. 7.
Sampel Pengujian Ikan Cakalang Beku Segar

Pada Gambar 7 terlihat 10 sampel ikan cakalang dalam kondisi beku dan segar.

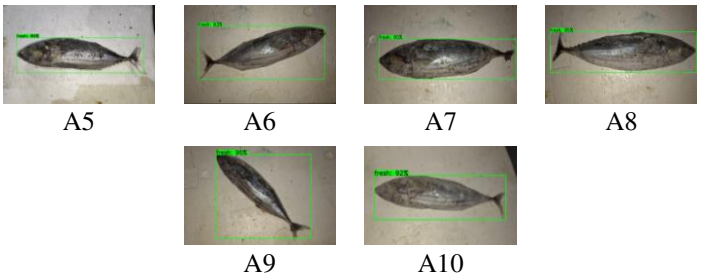


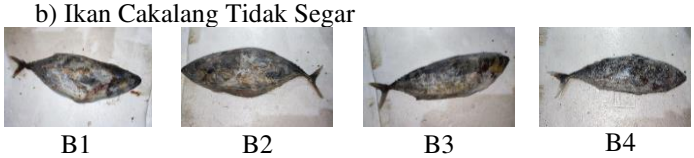
Fig. 8.
Hasil Pengujian Ikan Cakalang Beku Segar

Gambar 8 menunjukkan hasil pengujian pada ikan cakalang dalam kondisi beku dan segar yang telah dikonversi dari format Base64 menjadi format JPEG. Hasil deteksi menunjukkan bahwa semua sampel berhasil diidentifikasi sebagai ikan yang segar. Proses deteksi ini memakan waktu rata-rata 3,71 detik per sampel, seperti yang ditunjukkan pada Tabel II.

TABLE II.
HASIL PENGUJIAN *CLOUD* UNTUK IKAN CAKALANG BEKU DAN SEGAR

Citra Ikan	Hasil Deteksi	Waktu Pemrosesan (detik)
A1	"total": 1, "fresh": 1, "No._fresh": 0	3,63
A2	"total": 1, "fresh": 1, "No._fresh": 0	3,66
A3	"total": 1, "fresh": 1, "No._fresh": 0	3,58
A4	"total": 1, "fresh": 1, "No._fresh": 0	3,90
A5	"total": 1, "fresh": 1, "No._fresh": 0	4,09
A6	"total": 1, "fresh": 1, "No._fresh": 0	3,73
A7	"total": 1, "fresh": 1, "No._fresh": 0	3,61
A8	"total": 1, "fresh": 1, "No._fresh": 0	3,57
A9	"total": 1, "fresh": 1, "No._fresh": 0	3,67
A10	"total": 1, "fresh": 1, "No._fresh": 0	3,61
Rata-rata Waktu Pemrosesan		**3,71**

b) Ikan Cakalang Tidak Segar



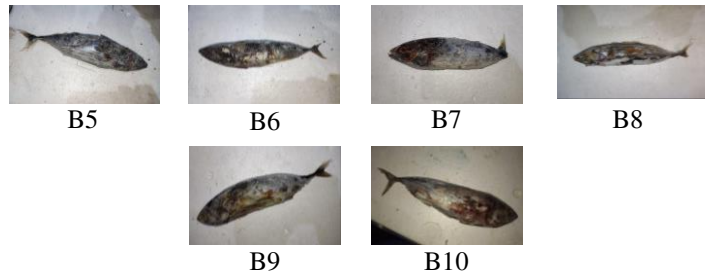


Fig. 9. Sampel Pengujian Ikan Cakalang Beku Tidak Segar

Pada Gambar 9 terlihat 10 sampel ikan cakalang dalam kondisi beku dan tidak segar. Sampel ini akan digunakan untuk pengujian *cloud computing*.

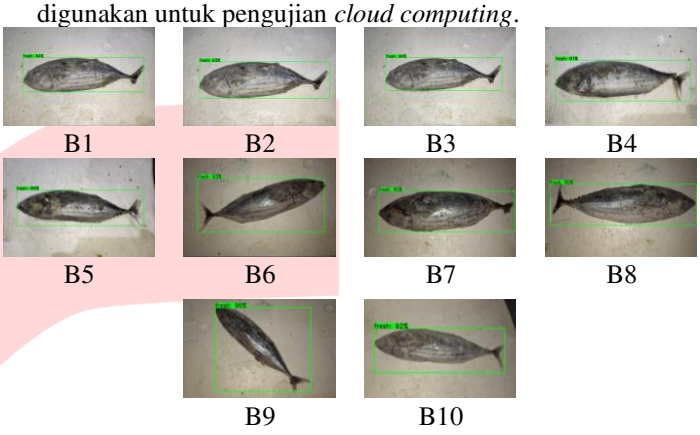


Fig. 10.
Hasil Pengujian Ikan Cakalang Beku Tidak Segar

Gambar 5.10 menunjukkan hasil pengujian pada ikan cakalang dalam kondisi beku dan tidak segar yang telah dikonversi dari format Base64 menjadi format JPEG. Proses deteksi ini memakan waktu rata-rata 3,12 detik per sampel, seperti yang ditunjukkan pada Tabel III.

TABLE III.
HASIL PENGUJIAN *CLOUD* UNTUK IKAN CAKALANG BEKU DAN TIDAK SEGAR

Citra Ikan	Hasil Deteksi	Waktu Pemrosesan (detik)
B1	"total": 1, "fresh": 0, "No._fresh": 1	3,32
B2	"total": 1, "fresh": 0, "No._fresh": 1	3,10
B3	"total": 1, "fresh": 0, "No._fresh": 1	3,17
B4	"total": 1, "fresh": 0, "No._fresh": 1	3,18
B5	"total": 1, "fresh": 0, "No._fresh": 1	3,11
B6	"total": 1, "fresh": 0, "No._fresh": 1	3,13
B7	"total": 1, "fresh": 0, "No._fresh": 1	3,08
B8	"total": 1, "fresh": 0,	3,03

	"No._fresh": 1	
B9	"total": 1, "fresh": 0, "No._fresh": 1	3,06
B10	"total": 1, "fresh": 0, "No._fresh": 1	3,06
Rata-rata Waktu Pemrosesan		**3,12**

c. Ikan Cakalang *Multiple*

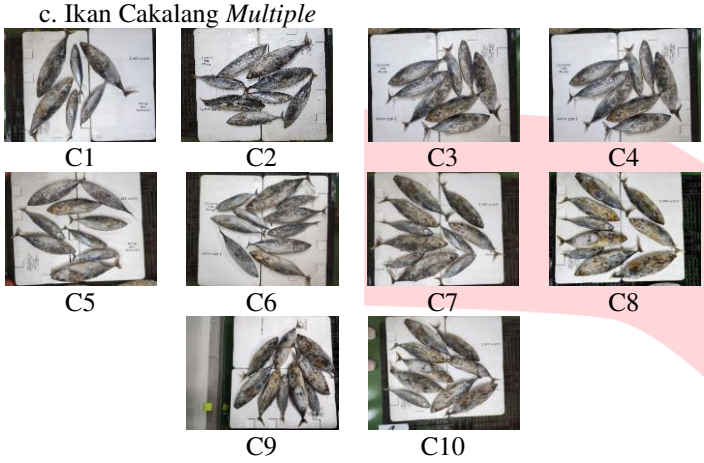


Fig. 11.
Sampel Pengujian Ikan Cakalang Beku *Multiple*

Pada Gambar 11 terlihat sampel ikan cakalang dalam kondisi beku dan memiliki dua kesegaran yang berbeda dalam satu sampel yaitu segar dan tidak segar yang akan digunakan dalam proses pengujian berbasis *cloud computing*.

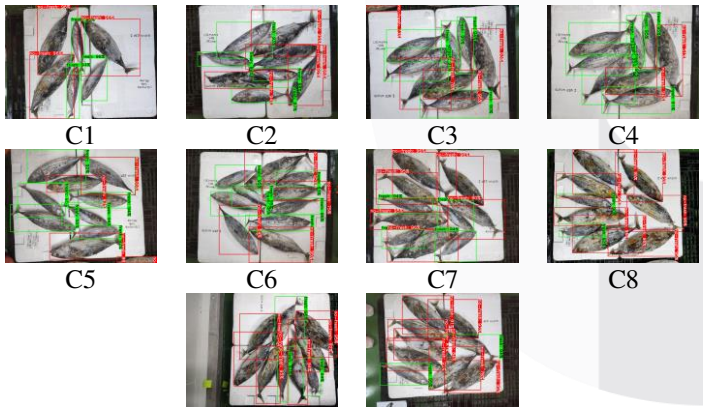


Fig. 12. Hasil Pengujian Ikan Cakalang Beku Multiple

Gambar 12 menunjukkan hasil pengujian ikan cakalang beku dalam bentuk *multiple* yang telah dikonversi dari format Base64 menjadi format JPEG. Proses deteksi ini memakan waktu rata-rata 4,27 detik per sampel, seperti yang ditunjukkan pada Tabel IV.

TABLE IV.
HASIL PENGUJIAN *CLOUD* UNTUK IKAN CAKALANG BEKU DAN BENTUK *MULTIPLE*

Citra Ikan	Hasil Deteksi	Waktu Pemrosesan (detik)
C1	"total": 6, "fresh": 3, "No._fresh": 3	4,22
C2	"total": 8, "fresh": 4, "No._fresh": 4	3,24
C3	"total": 9, "fresh": 5, "No._fresh": 4	4,71
C4	"total": 8, "fresh": 6, "No._fresh": 2	4,58
C5	"total": 11, "fresh": 7, "No._fresh": 4	4,19
C6	"total": 10, "fresh": 6, "No._fresh": 4	4,13
C7	"total": 10, "fresh": 4, "No._fresh": 6	3,65
C8	"total": 9, "fresh": 2, "No._fresh": 7	6,32
C9	"total": 9, "fresh": 3, "No._fresh": 6	3,86
C10	"total": 9, "fresh": 2, "No._fresh": 7	3,85
Rata-rata Waktu Pemrosesan		**4,27**

2. Pengujian Ikan Cakalang Tidak Beku

Pengujian dilakukan pada 30 sampel ikan cakalang dalam kondisi tidak beku yang terdiri dari 10 sampel ikan cakalang segar, 10 sampel ikan cakalang tidak segar, dan 10 sampel ikan cakalang campuran (*multiple*).

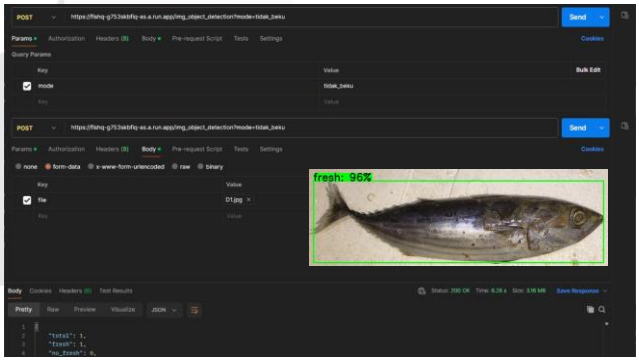


Fig. 13.
Pengujian Ikan Cakalang Tidak Beku menggunakan aplikasi Postman

Gambar 13 memperlihatkan metode POST untuk pengujian ikan cakalang tidak beku dengan citra ikan yang sudah tersedia. Untuk IP yang digunakan yaitu "https://fishqg753skbfiq-as.a.run.app/img_object_detection?Mode=tidak_beku".

a) Ikan Cakalang Segar

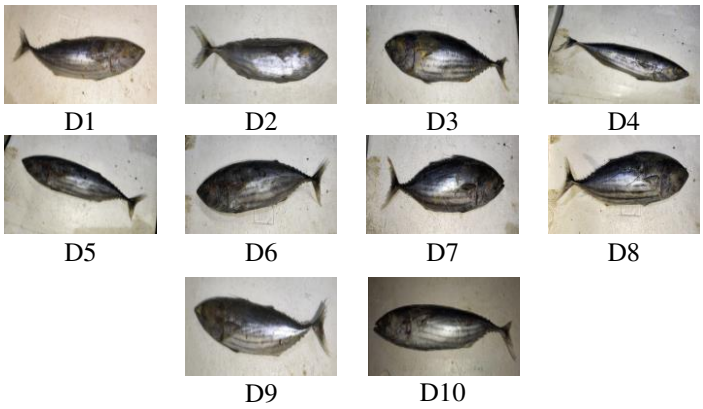


Fig. 14. Sampel Pengujian Ikan Cakalang Tidak Beku Segar

Pada Gambar 5.17 terlihat 10 sampel ikan cakalang dalam kondisi tidak beku dan segar. Sampel ini akan digunakan untuk pengujian *cloud computing*.

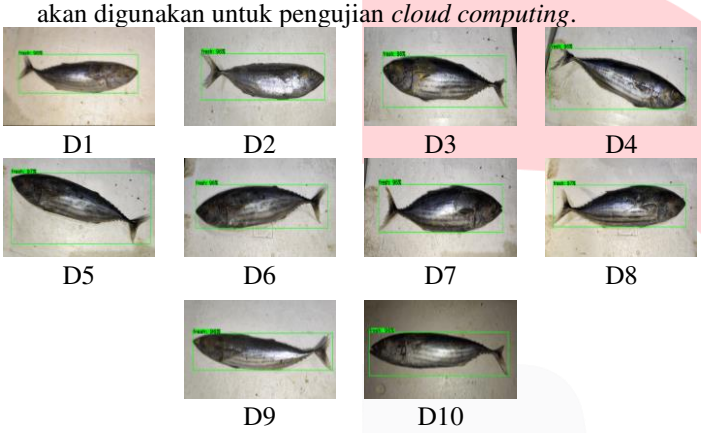


Fig. 15.
Hasil Pengujian Ikan Cakalang Tidak Beku Segar

Gambar 15 menunjukkan hasil pengujian pada ikan cakalang dalam kondisi beku dan segar yang telah dikonversi dari format Base64 menjadi format JPEG. Proses deteksi ini memakan waktu rata-rata 5,64 detik per sampel, seperti yang ditunjukkan pada Tabel V.

TABLE V.
HASIL PENGUJIAN *CLOUD* UNTUK IKAN CAKALANG TIDAK BEKU DAN SEGAR

Citra Ikan	Hasil Deteksi	Waktu Pemrosesan (detik)
D1	"total": 1, "fresh": 1, "No._fresh": 0	5,21
D2	"total": 1, "fresh": 1, "No._fresh": 0	5,36
D3	"total": 1, "fresh": 1, "No._fresh": 0	6,32
D4	"total": 1, "fresh": 1, "No._fresh": 0	5,95
D5	"total": 1, "fresh": 1, "No._fresh": 0	5,94
D6	"total": 1, "fresh": 1, "No._fresh": 0	5,82
D7	"total": 1, "fresh": 1,	5,77
	"No._fresh": 0	
D8	"total": 1, "fresh": 1, "No._fresh": 0	5,52
D9	"total": 1, "fresh": 1, "No._fresh": 0	5,07
D10	"total": 1, "fresh": 1, "No._fresh": 0	5,50
Rata-rata Waktu Pemrosesan		**5,64**

b. Ikan Cakalang Tidak Segar

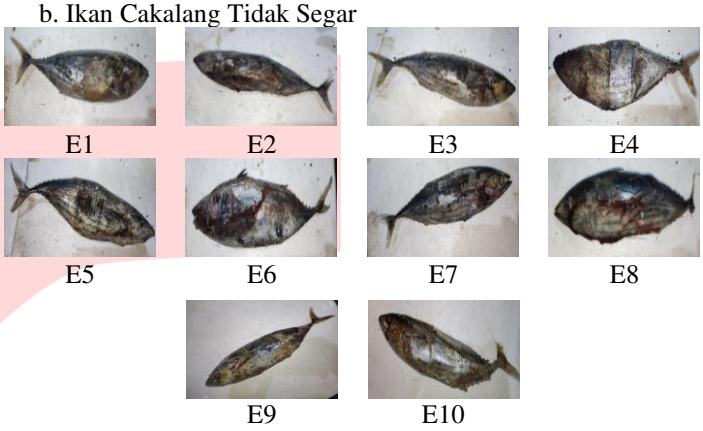


Fig. 16.
Sampel Pengujian Ikan Cakalang Tidak Beku Tidak Segar

Pada Gambar 16 terlihat 10 sampel ikan cakalang dalam kondisi tidak beku dan tidak segar. Sampel ini akan digunakan untuk pengujian *cloud computing*.

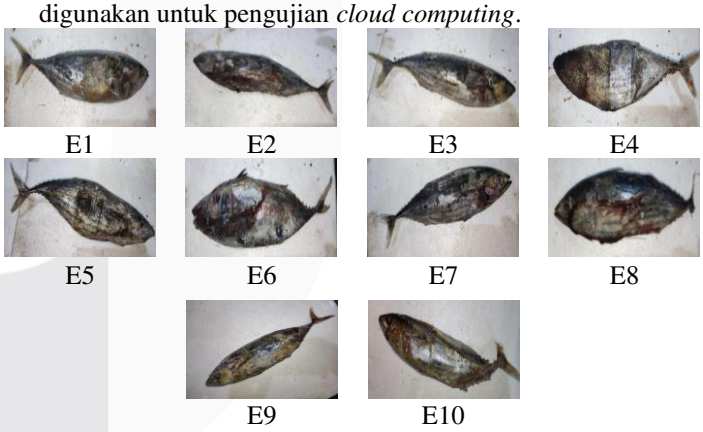


Fig. 17.
Hasil Pengujian Ikan Cakalang Tidak Beku Tidak Segar

Gambar 13 menunjukkan hasil pengujian pada ikan cakalang dalam kondisi beku dan segar yang telah dikonversi dari format Base64 menjadi format JPEG. Proses deteksi ini memakan waktu rata-rata 4,97 detik per sampel, seperti yang ditunjukkan pada Tabel VI.

TABLE VI.
HASIL PENGUJIAN *CLOUD* UNTUK IKAN CAKALANG TIDAK BEKU DAN TIDAK SEGAR

Citra Ikan	Hasil Deteksi	Waktu Pemrosesan (detik)

E1	"total": 1, "fresh": 0, "No._fresh": 1	5,02
E2	"total": 1, "fresh": 0, "No._fresh": 1	4,97
E3	"total": 1, "fresh": 0, "No._fresh": 1	4,82
E4	"total": 1, "fresh": 0, "No._fresh": 1	5,19
E5	"total": 1, "fresh": 0, "No._fresh": 1	4,96
E6	"total": 1, "fresh": 0, "No._fresh": 1	5,06
E7	"total": 1, "fresh": 0, "No._fresh": 1	4,90
E8	"total": 1, "fresh": 0, "No._fresh": 1	5,15
E9	"total": 1, "fresh": 0, "No._fresh": 1	5,04
E10	"total": 1, "fresh": 0, "No._fresh": 1	4,63
Rata-rata Waktu Pemrosesan		**4,97**

c. Ikan Cakalang *Multiple*

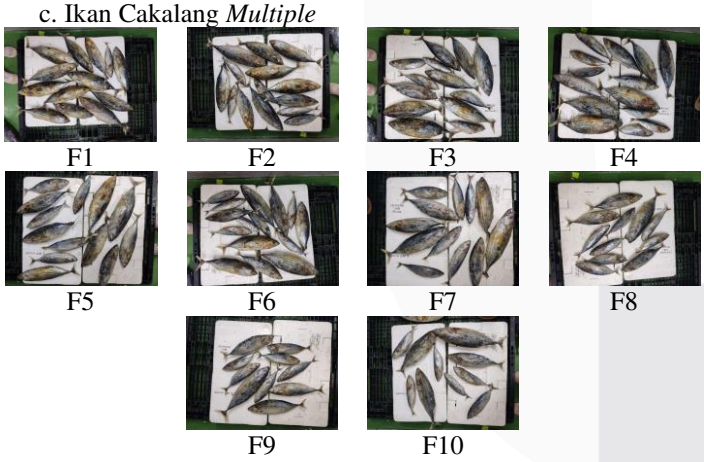


Fig. 18.
Sampel Pengujian Ikan Cakalang Tidak Beku *Multiple*

Pada Gambar 18 terlihat sampel ikan cakalang dalam kondisi tidak beku dan memiliki dua kesegaran yang berbeda dalam satu sampel yaitu segar dan tidak segar yang akan digunakan dalam proses pengujian berbasis *cloud computing*.

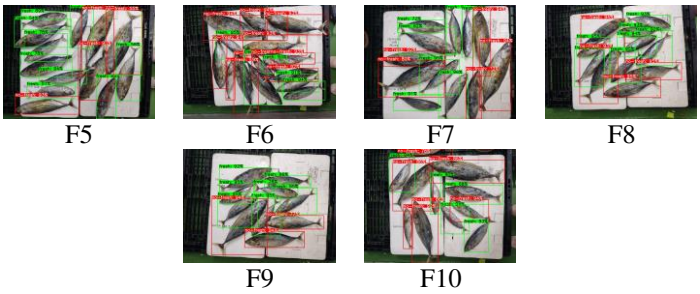


Fig. 19.
Hasil Pengujian Ikan Cakalang Tidak Beku *Multiple*

Gambar 19 menunjukkan hasil pengujian ikan cakalang tidak beku dalam bentuk multiple yang telah dikonversi dari format Base64 menjadi format JPEG. Proses deteksi ini memakan waktu rata-rata 4,04 detik per sampel, seperti yang ditunjukkan pada Tabel VII.

TABLE VII.
HASIL PENGUJIAN *CLOUD* UNTUK IKAN CAKALANG TIDAK BEKU BENTUK *MULTIPLE*

Citra Ikan	Hasil Deteksi	Waktu Pemrosesan (detik)
F1	"total": 13, "fresh": 1, "No._fresh": 12	3,79
F2	"total": 14, "fresh": 4, "No._fresh": 10	4,26
F3	"total": 16, "fresh": 6, "No._fresh": 10	3,99
F4	"total": 14, "fresh": 8, "No._fresh": 6	3,60
F5	"total": 14, "fresh": 10, "No._fresh": 4	3,96
F6	"total": 15, "fresh": 5, "No._fresh": 10	4,40
F7	"total": 12, "fresh": 7, "No._fresh": 5	4,10
F8	"total": 11, "fresh": 7, "No._fresh": 4	3,70
F9	"total": 10, "fresh": 7, "No._fresh": 3	4,36
F10	"total": 11, "fresh": 5, "No._fresh": 6	4,28
Rata-rata Waktu Pemrosesan		**4,04**

IV. HASIL DAN PEMBAHASAN

Pengujian *cloud* dilakukan dengan mengirimkan 10 citra ikan cakalang berbeda ke *endpoint* "/img_object_detection" menggunakan aplikasi Postman dengan metode POST. Pada Tabel VIII. dapat diambil rata-rata dari keseluruhan waktu pemrosesan citra ikan beku dalam berbagai kondisi ikan yaitu 3,70 detik per sampel.

TABLE VIII.
HASIL PENGUJIAN *CLOUD* UNTUK IKAN BEKU

Rata-Rata Waktu Pemrosesan Citra Ikan Beku		
Ikan Segar	**Ikan Tidak Segar**	**Ikan *Multiple***
3,71 detik/sampel	3,12 detik/sampel	4,27 detik/sampel

Selain itu, dilihat pada Tabel IX. dapat diambil rata-rata dari keseluruhan waktu pemrosesan citra ikan beku dalam berbagai kondisi ikan yaitu 4,88 detik per sampel.

TABLE IX.
HASIL PENGUJIAN *CLOUD* UNTUK IKAN TIDAK BEKU

Rata-Rata Waktu Pemrosesan Citra Ikan Tidak Beku		
Ikan Segar	**Ikan Tidak Segar**	**Ikan *Multiple***
5,64 detik/sampel	4,97 detik/sampel	4,04 detik/sampel

Waktu pemrosesan untuk ikan dalam kondisi beku tergolong lebih cepat dibandingkan dengan ikan dalam kondisi tidak beku, menunjukkan bahwa sistem lebih efisien dalam mendeteksi kesegaran ikan dalam kondisi beku. Hal tersebut dapat disebabkan oleh karakteristik visual ikan dalam kondisi tidak beku yang lebih beragam. Analisis hasil pada pengujian ini menunjukkan bahwa sistem *cloud computing* yang dirancang untuk mendeteksi kesegaran ikan cakalang memiliki performa yang baik dan efisien dalam berbagai kondisi, baik itu ikan segar, tidak segar, atau *multiple*.

V. KESIMPULAN

Penelitian ini berhasil merancang dan mengimplementasikan aplikasi FishQ yang menggunakan teknologi *cloud computing* dan model *deep learning* YOLOv8 untuk mendeteksi kesegaran ikan cakalang. Berdasarkan hasil pengujian pada 30 sampel ikan cakalang dalam kondisi beku dan tidak beku, sistem menunjukkan performa yang baik dengan akurasi tinggi dalam mendeteksi kesegaran ikan. Hasil pengujian menunjukkan bahwa sistem *cloud computing* yang dirancang mampu mendeteksi kesegaran ikan dengan efisien dan akurat, terutama lebih cepat pada ikan dalam kondisi beku. Secara keseluruhan, aplikasi FishQ diharapkan dapat membantu perusahaan perikanan dalam meningkatkan efisiensi dan akurasi proses sortasi ikan, sehingga dapat meningkatkan kualitas produk perikanan yang dijual.

REFERENSI

[1] Pusat Hidro-Oseanografi TNI Angkatan Laut, “Data Kelautan Yang Menjadi Rujukan Nasional Diluncurkan,”2018.[Online].Available:https://sidakokhl.kkp.go.id/sidako/data-kelautan. Accessed: Oct. 14, 2023.

[2] Badan Pusat Statistik, “Ekspor Ikan Segar/Dingin Hasil Tangkap menurut Negara Tujuan Utama, 2012-2021,” Feb.2019.[Online].Available:https://www.bps.go.id/id/statistics-table/1/MjAyNCMx/ekspor-ikan-segar-dingin-hasil-tangkap-menurut-negara-tujuan-utama--2012-2022.html. Accessed: Oct. 14, 2022.

[3] T. Rudi Hartanto, S. Suharno, and B. Burhanuddin, “Daya Saing Ekspor Ikan Tuna-Cakalang-Tongkol Indonesia di Pasar Amerika Serikat,” *Jurnal Pengolahan Hasil Perikanan Indonesia*, vol. 24, no. 2, pp. 227–235, Aug. 2021, doi: 10.17844/jphpi.v24i2.36075.

[4] W. Widhoroso, “Aruna Dorong Peningkatan Kesejahteraan Nelayan,” Media Indonesia, Jun. 6, 2023. [Online].Available:https://mediaindonesia.com/humaniora/587196/aruna-dorong-peningkatan-kesejahteraan-nelayan. Accessed: Nov. 20, 2023.

[5] R. Yusuf Azhari, “Web Service Framework : Flask Dan Fastapi,” *Technology and Informatics Insight Journal*, vol. 1, no. 1, pp. 58–65, Feb. 2022, doi: 10.32639/tiij.v1i1.54.

[6] Zeng, R., Jha, A., Kumar, A., & Luo, T. (2023). Cloud-Based Deep Learning: End-To-End Full-Stack Handwritten Digit Recognition. *arXiv preprint arXiv:2304.13506*.

[7] A. Jain, "Using Google Cloud Machine Learning APIs Programmatically in Python - Part 1," Towards Data Science, 31-Jul-2020. [Online]. Available: https://towardsdatascience.com/using-google-cloud-machine-learning-apis-programmatically-in-python-part-1-430f608af6a5. Accessed: Jul. 6, 2024.

[8] Google Cloud, "Solusi Praktis: Pemrosesan gambar AI/ML pada Cloud Functions," Cloud Architecture Center, 2024. [Online]. Available: https://cloud.google.com/architecture/ai-ml/image-processing-cloud-functions?hl=id. Accessed: Jul. 6, 2024.

[9] Google Cloud, "Panduan memulai: Men-deploy aplikasi dalam container ke Cloud Run," Dokumentasi Cloud Build, 2024. [Online]. Available: https://cloud.Google .com/build/docs/deploy-containerized-application-cloud -run?hl=id. Accessed: Jul. 6, 2024.

[10] Google Cloud, "Men-deploy ke Cloud Run menggunakan Cloud Build," Dokumentasi Cloud Build, 2024. [Online]. Available: https://cloud.google.com /b uild/docs/deploying-builds/deploy-cloud-run?hl=id. Accessed: Jul. 6, 2024.

[11] Google Cloud, "Google Cloud: Layanan Cloud Computing," Google Cloud, 2024. [Online]. Available: https://cloud.google.com/?hl=id. Accessed: Jul. 6, 2024.

[12] S. Chikuse, "How to Detect Objects in Images Using the YOLOv8 Neural Network," freeCodeCamp, 2023. [Online]. Available: https://www.freecodecamp.org/news/how-to-detect-objects-in-images-using-yolov8/. Accessed: Jul. 6, 2024.

[13] Google Cloud, "Integrasi dengan Google Cloud | Cloud Storage for Firebase," Google Cloud, 2024. [Online]. Available: https://firebase.google.com/docs/stora ge/gcp -integra tion?hl=id. Accessed: Jul. 6, 2024.
[14] Google Cloud, "Apa itu Cloud Run," Google Cloud, 2024.[Online].Available: https://cloud.google.com/run/ docs/overview/what-is-cloud-run?hl=id. Accessed: Jul. 6, 2024.
[15] Google Cloud, "Deep Learning Containers," Google Cloud, 2024. [Online]. Available: https://cloud.google.com/deep-learning-containers. Accessed: Jul. 6, 2024.
[16] Amazon Web Services (AWS), "Object Detection Request and Response Formats," AWS Documentation, 2024.[Online].Available: https://docs.aws.amazon.com/ sagemaker/latest/dg/object-detection-in-formats.html. Accessed: Jul. 6, 2024.
[17] Landing AI, "JSON Prediction Responses from LandingLens," Landing AI Documentation, 2024. [Online].Available: https://support.landing.ai/docs/json-responses. Accessed: Jul. 6, 2024.
[18] Aris Setiyadi, Ema Utami, "Analisa Kemampuan Algoritma YOLOv8 dalam Deteksi Objek Manusia," Jurnal Sains Komputer & Informatika (J-SAKTI), vol. 7, no. 2, pp. 891-901, Sep. 2023. [Online]. Available: https://tunasbangsa.ac.id/ejurnal/index.php/jsakti/article/download/694/669. Accessed: Jul. 6, 2024.
[19] Imam Maulana, "Analisis Penggunaan Model YOLOv8 (You Only Look Once) terhadap Deteksi Citra Senjata Berbahaya," Jurnal Teknik Informatika, STMIK IKMI, 2023.[Online].Available: https://ejournal.itn.ac.id/index .php/jati/article/view/8271. Accessed: Jul. 6, 2024.
[20] H. M. Lathifah, L. Novamizanti, & S. Rizal, "Fast and accurate fish classification from underwater video using you only look once," in IOP Conference Series: Materials Science and Engineering, vol. 982, no. 1, pp. 012003). IOP Publishing, December 2020.
[21] F. Akhyar, L. Novamizanti, T. Putra, E. N. Furqon, M. C. Chang, & C. Y. Lin, "Lightning YOLOv4 for a surface defect detection system for sawn lumber," in 2022 IEEE 5th International Conference on Multimedia Information Processing and Retrieval (MIPR) (pp. 184-189). IEEE, August 2022.
[22] F. Akhyar, L. Novamizanti, & T. Riantiarni, "Sistem inspeksi cacat pada permukaan kayu menggunakan model deteksi obyek YOLOv5," ELKOMIKA: Jurnal Teknik Energi Elektrik, Teknik Telekomunikasi, & Teknik Elektronika, vol. 10, no.4, pp. 990, 2022.
[23] A. K. Aziz, M. D. Maulana, R. F. Adawiyah, R. F. Firdaus, L. Novamizanti, & F. Ramdhon, "Comparative analysis of YOLOv8 models in skipjack fish quality assessment system," in 2023 3rd International Conference on Intelligent Cybernetics Technology & Applications (ICICyTA) (pp. 237-242). IEEE, December 2023.
[24] S. Azizah, Wahidin, M. Padang, L. Novamizanti, S Saidah. "Identifying the Ripeness and Quality Level of Strawberries Based on YOLOv7-EfficientNet." The 5th International Conference on Data Science and Its Applications (ICoDSA). IEEE, July 2024.
[25] B. A. Wicaksono, L. Novamizanti, & N. Ibrahim, "Tea leaf maturity levels based on ycbcr color space and clustering centroid," In Journal of Physics: Conference Series (Vol. 1367, No. 1, pp. 012028). IOP Publishing, November 2019.