

Aplikasi *Warning Alert* Pendeteksi Kelelahan Ekspresi Wajah Pada Pengemudi Secara *Real-Time* Menggunakan Metode *You Only Look Once* Berbasis Website

Hafidh Ahmad Fauzan¹, Ari Kurniawan²

^{1,3} *Manajemen Informatika, Fakultas Vokasi, Universitas Negeri Surabaya
Jl. Lidah Wetan, Lidah Wetan, Kec. Lakarsantri, Surabaya, Jawa Timur 60213*

¹hafidh.19027@mhs.unesa.ac.id

²arikurniawan@unesa.ac.id

Abstrak— Kelelahan dapat terjadi dimana saja, mulai dari lingkungan pendidikan, lingkungan pekerjaan, lalu lintas, dan masih banyak lainnya. Kelelahan terjadi ketika tubuh mengalami kehabisan energi karena aktivitas yang telah atau sedang dilakukan. Gejala yang dialami oleh orang yang sedang merasakan lelah adalah menurunnya daya kerja serta ketahanan tubuh yang mengakibatkan timbulnya rasa kantuk. Kelelahan dapat menyebabkan dampak buruk pada aktivitas yang dikerjakan, seperti hilangnya fokus atau konsentrasi, dan juga kewaspadaan akan sesuatu. *You Only Look Once* adalah salah satu metode baru tentang Kecerdasan Buatan dan *Machine Learning*. Metode ini merupakan jaringan arsitektur untuk digunakan pada pengembangan fitur atau teknologi deteksi objek dan memiliki julukan “*fastest deep learning object detector*” dengan didominasi pada kecepatan deteksi dan keakuratan akurasi. Untuk mengurangi dampak negatif yang disebabkan oleh kelelahan, peneliti menggunakan teknologi Darknet untuk membuat model deteksi kelelahan berdasarkan citra ekspresi wajah dengan tingkat keakuratan akurasi yang tinggi, dimana ketika model mendeteksi adanya kelelahan pada wajah, maka muncul peringatan dan sirine, sehingga orang secara sadar dapat beristirahat maupun berhenti dari aktivitas yang dilakukan. Produk model deteksi kelelahan yang dibuat oleh peneliti nantinya diuji dan diimplementasikan pada aplikasi berbasis website.

Kata Kunci— *You Only Look Once*, Kecerdasan Buatan, Model Deteksi Kelelahan, Darknet, Deteksi Objek.

I. PENDAHULUAN

Kelelahan dapat terjadi dimana saja, mulai dari lingkungan pendidikan, lingkungan pekerjaan, lalu lintas, dan masih banyak lainnya. Kelelahan terjadi ketika tubuh mengalami kehabisan energi karena aktivitas yang telah atau sedang dilakukan. Gejala yang dialami oleh orang yang sedang merasakan lelah adalah menurunnya daya kerja serta ketahanan tubuh yang mengakibatkan timbulnya rasa kantuk.

Banyak sekali penyebab yang dapat ditimbulkan dari kelelahan. Pada lingkungan pendidikan, mahasiswa kurang dapat fokus pada pembelajaran yang sedang berlangsung dikarenakan kelelahan. Pada lingkungan pekerjaan, misalnya tambang, pekerja dapat mengalami kecelakaan kerja dikarenakan kelelahan yang menyebabkan pekerja tersebut kurang berhati-hati. Pada lalu lintas, menurut data Kepolisian di Indonesia, 61% kecelakaan disebabkan oleh *human error*.

Salah satu faktor utama penyebab terjadinya *human error* dalam berkendara adalah kelelahan. Keadaan lelah saat mengemudi adalah aspek paling berbahaya penyebab kecelakaan di lalu lintas [1]. Berdasarkan perkiraan yang disampaikan oleh Organisasi Kesehatan Dunia (WHO), setiap tahunnya terdapat sekitar 1,3 juta jiwa orang merengas nyawa dan sekitar 20 sampai 50 juta jiwa orang mendapati luka akibat dari kecelakaan pada lalu lintas.

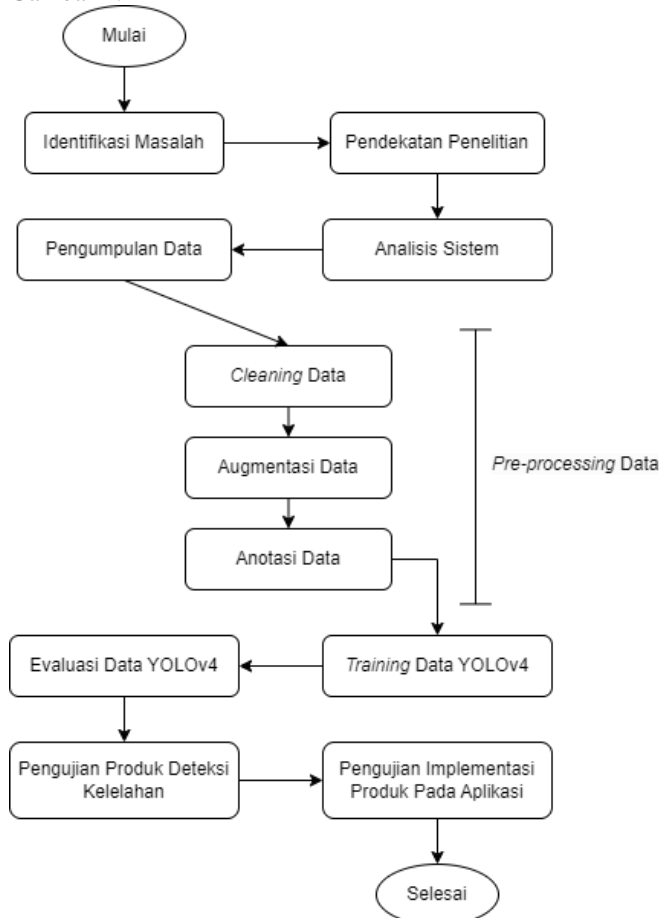
Metode *You Only Look Once* (YOLO) merupakan metode mutakhir deteksi objek dengan tingkat akurasi tinggi yang dapat membantu merealisasikan produk yang dibuat peneliti buat. YOLO adalah pengembangan dari metode Convolutional Neural Network (CNN) menjadi sebuah algoritma baru dan berfokus pada deteksi objek dengan menggunakan ulang atau kembali hasil pengklasifikasi untuk menjalankan fitur deteksi. Arsitektur terpadu YOLO sangat cepat dan mampu memproses gambar secara *real-time* dengan 45 fps (*frame per seconds*). Algoritma jaringan metode YOLO dengan versi lebih kecil dinamakan Fast YOLO, di mana algoritma jaringan ini mampu memproses gambar secara *realtime* hingga 155 fps [2]. Sedangkan versi terbaru dari YOLO dinamakan YOLOv4 yang dikembangkan oleh Bochkovskiy dkk. Pada hasil perbandingan deteksi objek YOLOv4 dengan versi sebelumnya (YOLOv3), Bochkovskiy dkk meningkatkan nilai AP (*Average Precision*) dan fps dari YOLOv3 sebesar 10% dan 12%. YOLO mendeteksi objek secara *real-time* menggunakan *Graphic Processing Unit* (GPU) konvensional, yang memungkinkan penggunaan massalnya dengan nominal harga yang tetap dapat dijangkau. YOLO muncul karena sebelumnya jaringan syaraf modern yang paling akurat tidak beroperasi secara *real-time* dan membutuhkan banyak GPU untuk pelatihan dengan ukuran *mini-batch* yang besar. Untuk mengatasi masalah tersebut, dibangunlah CNN yang beroperasi secara *real-time* pada GPU konvensional dan untuk itu pelatihan hanya memerlukan satu GPU konvensional [3].

Teori dan algoritma untuk *Deep Learning* (Pembelajaran Mendalam) sudah pernah dilakukan dalam mendeteksi kelelahan pengemudi. Oleh karenanya, peneliti mengusulkan dan ingin mengembangkan model pendeteksi kelelahan berdasarkan citra ekspresi wajah pada pengemudi berdasarkan visi komputer yang sudah dilakukan dan ditentukan kelasnya dari pengemudi. Kelas tersebut antara lain: “fokus” dan

“lelah”. Informasi citra ekspresi wajah pengemudi dimasukkan ke jaringan saraf untuk ekstraksi fitur. Selanjutnya informasi fitur yang sudah diekstraksi, diklasifikasikan melalui lapisan *Softmax* untuk penentuan kelas yang didapat berdasarkan kondisi citra ekspresi wajah pengemudi. Pada kelas lelah, citra mendeteksi ekspresi wajah yang menguap, ekspresi wajah dengan mata tertutup (tidur). Pada kelas fokus, citra mendeteksi ekspresi wajah dengan mata terbuka yang menandakan pengemudi sedang terjaga. Produk model deteksi yang dibuat oleh peneliti nantinya diuji dan diimplementasikan pada aplikasi berbasis website.

II. METODE PENELITIAN

Pada bab ini, menjelaskan tentang alur tahapan yang dilakukan peneliti selama proses penelitian berlangsung. Adapun alur tahapan yang dilakukan dapat dilihat pada Gambar 1:



Gbr. 1. Alur Penelitian

A. Identifikasi Masalah

Identifikasi masalah yang digunakan peneliti sebagai sebab utama penelitian dilakukan adalah maraknya berita dari sumber-sumber aktual terkait padatnya lalu lintas yang menyebabkan tingginya angka kecelakaan, di mana faktor tertinggi kecelakaan lalu lintas adalah *human error* (kesalahan manusia) seperti: menyetir dalam keadaan lelah. Hal ini yang mendasari dilakukannya proses pendekatan penelitian yang berhubungan dengan permasalahan dan sumber-sumber terkait,

serta beberapa tinjauan pustaka yang nantinya digunakan sebagai acuan untuk memulai penelitian. Pencarian terkait literatur dilakukan oleh peneliti menggunakan berbagai sumber, seperti: jurnal penelitian terdahulu yang relevan (5 tahun terakhir), buku dan informasi pada internet.

B. Pendekatan Penelitian

Berdasarkan temuan masalah terkait tingginya kecelakaan lalu lintas yang diakibatkan oleh *human error* (kesalahan manusia), peneliti membuat produk pendeteksi kelelahan berdasarkan ekspresi wajah secara *real-time* dengan tingkat keakuratan akurasi yang tinggi. Oleh karenanya, peneliti memutuskan untuk menggunakan metode YOLO sebagai landasan dalam melakukan penelitian untuk dapat memproses gambar secara *real-time* dengan 45 fps sampai 155 fps dan memiliki tingkat keakuratan akurasi yang tinggi.

C. Analisis Sistem

Analisis sistem sangat diperlukan untuk mengetahui kebutuhan dan konfigurasi yang diperlukan untuk membuat produk penelitian berdasarkan permasalahan dan metode yang digunakan. Berikut analisis sistem yang sekiranya diperlukan.

1) Kebutuhan Spesifikasi Komputer Minimal

- CPU Intel Core i5 generasi ke 8 atau AMD Ryzen 3000 series
- RAM 8 GB
- Storage 1 GB
- OS (Windows / Linux / MacOS)
- VGA Nvidia GTX Series (opsional)

2) Aplikasi dan Modul

- Visual Studio Code
- Python
- Anaconda
- Label-studio
- Google Colab
- OpenCV
- Vite
- NodeJS

3) Konfigurasi Hyperparameters YOLOv4

TABEL I
KONFIGURASI HYPERPARAMETER YOLOV4 [3]

Parameter	Value
<i>Classes</i>	sesuai variable untuk kelas klasifikasi
<i>Network Size</i>	416 x 416
<i>Batch</i>	32 / 64
<i>Subdivision</i>	16
<i>Max Batches</i>	(total kelas) x 2000
<i>Steps</i>	(80% <i>Max Batches</i>), (90% <i>Max Batches</i>)
<i>Filters</i>	(total kelas + 5) x 3
<i>Channel</i>	3
<i>Learning Rate</i>	0.001
<i>Momentum</i>	0.949
<i>Convolutional Layer</i>	110
<i>Route Layer</i>	21
<i>Shortcut Layer</i>	23
<i>Downsample Layer</i>	5

<i>Upsample Layer</i>	2
<i>YOLO Layer</i>	3
<i>Stride in Convolutional Layer</i>	1 dan 2

4) Input dan Output

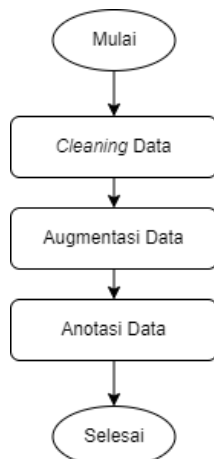
- *Input*: Citra ekspresi wajah pengemudi.
- *Output*: Klasifikasi pengemudi “lelah” atau “fokus”, apabila lelah maka secara otomatis mengeluarkan *warning alert* dan juga pesan melalui Bot DrowsinessDetection_Hafidh pada Telegram.

D. Pengumpulan Data

Ada 2 tahap pengumpulan data yang dilakukan, yaitu pengumpulan data pribadi milik peneliti dan pengumpulan dataset dari Kaggle. Pada tahap pengumpulan data pribadi, hal pertama yang dilakukan peneliti adalah membuat video berdurasi 30 detik dengan 60 fps berdasarkan kedua kelas yang sudah ditentukan sebelumnya, yaitu kelas fokus dan kelas lelah. Dengan kode program yang sudah dibuat oleh peneliti, video diekstrak tiap framenya dan dilakukan pengambilan 250 data saja tiap kelasnya menggunakan metode *random sampling* guna menghindari bias data.

Sedangkan untuk pengumpulan data pada Kaggle, peneliti menggunakan dataset dengan kata kunci “*Drowsiness*”. Dikarenakan metode YOLO adalah metode yang terbilang baru, maka beberapa website penyedia dataset tidak memiliki dataset untuk model arsitektur YOLO dan hanya menyediakan data mentah saja. Hal ini dapat dibuktikan dari penelitian sebelumnya berjudul “*Real-time detection method of driver fatigue state based on deep learning of face video*” oleh Zhe Cui, dkk (Desember 2020). Setelah Cui dkk melakukan pencarian data untuk dataset model, maka dilanjutkan dengan pelabelan secara manual dengan *bounding box* pada bagian mata dan mulut.

E. Pre-processing Data

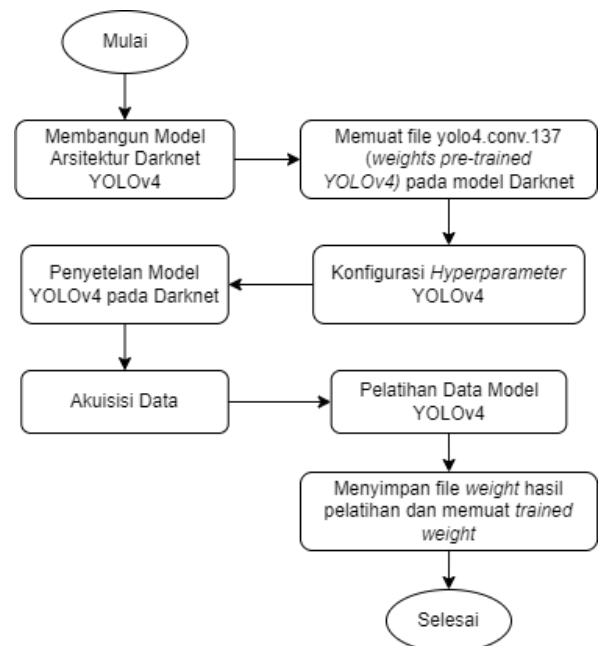


Gbr. 2. Alur Pre-processing Data

Gambar 2 menjelaskan terkait proses *preprocessing* data setelah selesai melakukan tahapan pengumpulan data. Pertama, data dilakukan pembersihan dari *outlier* (pencilan data) terlebih dahulu, seperti: citra ekspresi wajah tidak terlihat dengan jelas, citra tidak sesuai dengan data yang dibutuhkan untuk penelitian, citra gambar terlalu gelap, *noise* dan lainnya.

Proses ini memakan waktu yang cukup lama karena data bersifat deskriptif dan nonnumerik, peneliti menggunakan metode *Check and Save* untuk memeriksa citra gambar satu per satu. Untuk dataset pribadi, dilakukan augmentasi *geometric distortion* berupa flip horizontal untuk memperbanyak jumlah dataset, sedangkan pada dataset dari Kaggle tidak diperlukan kembali proses augmentasi, karena kebanyakan citra gambar dari dataset tersebut sudah dilakukan augmentasi *photometric distortion* (mengubah kecerahan, kontras, ataupun saturasi) dan augmentasi *geometric distortion* (flip horizontal). Ketika data sudah bersih, maka tahapan dapat dilanjutkan untuk proses anotasi data sesuai dengan format model pelatihan YOLOv4 menggunakan Label-Studio.

F. Pelatihan Data



Gbr. 3. Alur Pelatihan Data

Gambar 3 menjelaskan terkait proses pelatihan data yang dilakukan menggunakan Google Colab. Proses diawali dengan membangun model arsitektur Darknet YOLOv4 yang disimpan pada Google Drive yang sudah dihubungkan sebelumnya, hal ini untuk mengantisipasi kehilangan hasil data setelah pelatihan ketika terjadi *error*. Setelah model berhasil dibangun, dilanjutkan dengan memuat *weights pre-trained* YOLOv4 berupa file *yolo4.conv.137* sebagai media *transfer learning* (penggunaan lapisan yang sudah dilatih sebelumnya untuk membangun jaringan berbeda yang mungkin mempunyai kesamaan pada lapisan pelatihan pertama). Sebelum dilakukan penyetelan model YOLOv4 pada Darknet, peneliti melakukan konfigurasi *hyperparameters* YOLOv4 sesuai dengan aturan yang sudah dijelaskan sebelumnya. Peneliti menggunakan model Darknet-53 untuk membangun arsitektur YOLOv4 pada Darknet karena YOLOv4 menggunakan koneksi CSP dengan Darknet-53 sebagai ekstraksi fitur dan *backbone*. Model CSPDarknet-53 memiliki tingkat keakuratan akurasi yang tinggi.

TABEL II
ARSITEKTUR YOLOv4 DARKNET-53 [3]

Type		Filters	Size	Output
Convolutional		32	3 x 3	256 x 256
Convolutional		64	3 x 3 / 2	128 x 128
1x	Convolutional	32	1 x 1	
	Convolutional	64	3 x 3	
	Residual			128 x 128
Convolutional		128	3 x 3 / 2	64 x 64
2x	Convolutional	64	1 x 1	
	Convolutional	128	3 x 3	
	Residual			64 x 64
Convolutional		256	3 x 3 / 2	32 x 32
8x	Convolutional	128	1 x 1	
	Convolutional	256	3 x 3	
	Residual			32 x 32
Convolutional		512	3 x 3 / 2	16 x 16
8x	Convolutional	256	1 x 1	
	Convolutional	512	3 x 3	
	Residual			16 x 16
Convolutional		1024	3 x 3 / 2	8 x 8
4x	Convolutional	512	1 x 1	
	Convolutional	1024	3 x 3	
	Residual			8 x 8
Average Pool		Global		
Connected Softmax		1000		

Pada file konfigurasi YOLOv4, blok "[net]" berisi informasi terkait parameter jaringan seperti "batch", "subdivision", "width", "height", dan "channels". Setelah blok "[net]", terdapat beberapa blok "[convolutional]" yang menggambarkan lapisan konvolusi dalam arsitektur Darknet-53, dimana setiap bagiannya berisi informasi terkait parameter lapisan konvolusi seperti "size", "stride", "pad", "filters", dan "activation". Blok "[convolutional]" juga bisa diikuti oleh blok "[maxpool]" yang berisi informasi terkait lapisan *MaxPooling* dalam arsitektur. Jaringan pada arsitektur Darknet-53 juga memiliki beberapa blok lain seperti blok "[route]" yang berfungsi menyediakan koneksi dengan lapisan sebelumnya, blok "[shortcut]" yang berfungsi menerapkan jalan pintas atau melewati koneksi sebelumnya, dan blok "[upsample]" untuk menjalankan operasi *upsampling*. Pada bagian akhir, terdapat blok "[yolo]" yang menggambarkan hasil dari *output* layer YOLO dengan membawa parameter seperti "classes", "anchors", dan "mask" (Bochkovskiy dkk., n.d.). Operasi *upsampling* adalah operasi untuk meningkatkan dimensi (resolusi) fitur yang diketahui dari lapisan konvolusi

sebelumnya sehingga fitur yang dihasilkan lebih detail dan akurat untuk mendeteksi objek yang lebih kecil atau memiliki detail yang halus. Oleh karena itu, model yang dilatih menggunakan YOLO mampu mendeteksi objek dengan ukuran yang beragam dalam gambar.

Pada proses akuisisi, peneliti menggunakan perbandingan 80:20 di mana 80% untuk data latih dan 20% untuk data uji. Hal ini sesuai dengan implementasi konsep "Train-Test-Split" pada aturan umum *Machine Learning*. Selanjutnya, data dimasukkan pada model arsitektur Darknet YOLOv4 untuk digunakan sebagai pelatihan komputer dalam mendeteksi kelelahan berdasarkan ekspresi wajah. Hasil dari pelatihan data disimpan dalam bentuk file *weight*, proses penyimpanan file dimulai ketika data sudah mencapai iterasi ke-1000 dan selanjutnya selalu disimpan setiap kelipatan 100 iterasi sebagai file *weight* terakhir sebelum pelatihan data berakhir (semua data telah dilakukan pelatihan atau pelatihan berhenti karena terjadinya hambatan, seperti koneksi lemah, GPU tidak aktif dan lainnya).

G. Evaluasi Data

Proses evaluasi data menghasilkan nilai evaluasi dari file *weight* terakhir disimpan ketika proses pelatihan data berakhir, berupa nilai mAP, *Precision*, *Recall* dan *F1-score*. Selain itu, jumlah objek data uji yang terdeteksi beserta persentasenya, jumlah nilai *True Positive* dan *False Positive*, serta jumlah *unique truth count* juga dihitung. Hasil dari evaluasi yang didapatkan menentukan seberapa akurat produk model deteksi kelelahan yang dibuat nantinya. Semakin tinggi nilai mAP yang didapatkan, semakin tinggi pula tingkat keakuratan akurasi model deteksi.

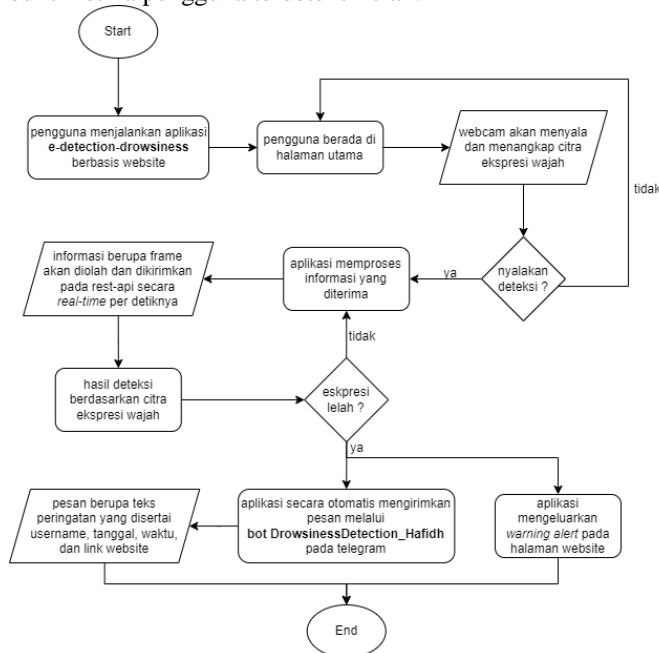
H. Pengujian Produk Deteksi Kelelahan

Sebelum produk model deteksi kelelahan dibuat API dan diimplementasikan pada aplikasi, peneliti melakukan pengujian terlebih dahulu menggunakan kode program yang dibuat menggunakan Python untuk menguji seberapa akurat dan tepat deteksi kelelahan yang sudah dilatih. Pengujian dilakukan menggunakan media foto, video dan juga webcam secara *real-time*.

I. Pengujian Produk Pada Aplikasi Website

Untuk alur kerjanya adalah, pengguna menjalankan aplikasi "E-Detection-Drowsiness" berbasis website, pengguna diarahkan ke halaman utama dan secara otomatis meminta persetujuan pengguna untuk mengaktifkan webcam. Hal ini berguna untuk aplikasi mendapatkan citra ekspresi wajah dari pengguna melalui video yang ditangkap pada webcam. Lalu pengguna dapat menekan tombol "Run Detection" untuk mulai melakukan deteksi. Aplikasi memproses data secara terus-menerus dan mengirim citra ekspresi wajah pada API produk model deteksi kelelahan yang sudah dibuat per detik. Ketika citra ekspresi wajah pengguna dirasa menunjukkan kelas klasifikasi lelah seperti mata tertutup atau menguap, maka aplikasi website secara otomatis mengeluarkan *warning alert* pada tampilan antarmuka website pengguna, serta mengirimkan pesan peringatan melalui "Bot DrowsinessDetection_Hafidh" pada Telegram. Di mana isi

dari pesan Telegram memberikan informasi tentang *username* Telegram, tanggal dan waktu ketika pengguna terdeteksi sedang lelah. Pesan juga dilengkapi dengan tautan yang mengarah ke website di mana berisi tentang informasi tangkapan gambar disertai dengan tanggal dan waktu sebagai bukti ketika pengguna terdeteksi lelah.

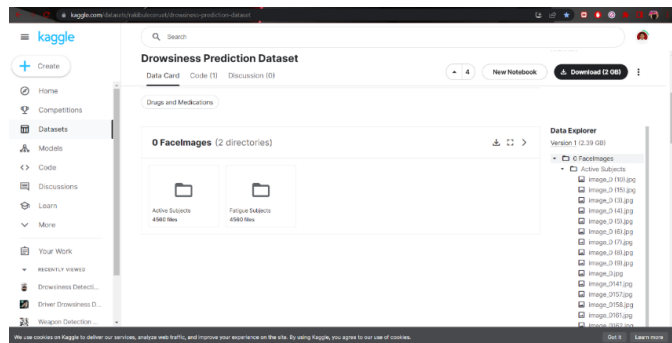


Gbr. 4. Alur Kerja Aplikasi Pendeteksi Kelelahan

III. HASIL DAN PEMBAHASAN

Pada bab ini, peneliti membahas hasil pengumpulan data, hasil *pre-processing* data, hasil pelatihan data, hasil evaluasi data, hasil pengujian produk deteksi kelelahan dan hasil pengujian implementasi produk pada aplikasi website.

A. Pengumpulan Data

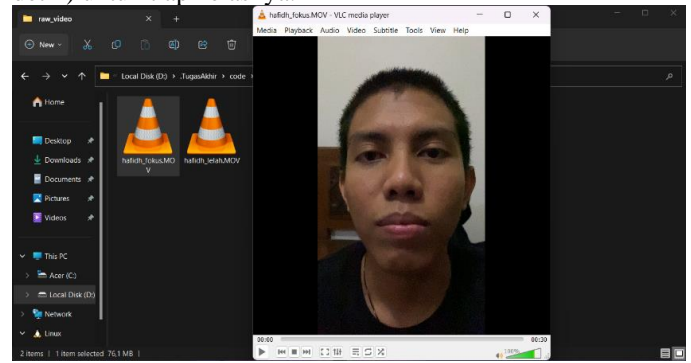


Gbr. 5. Dataset Drowsiness Prediction dari Kaggle

Gambar 5 menjelaskan dataset mentah yang disediakan oleh **RAKIBUL.ECE.RUET** pada Kaggle berjumlah 4560 file tiap kelasnya, yaitu *active* dan *fatigue*. Alasan peneliti memilih dataset ini karena dataset yang disediakan sesuai dengan judul peneliti, di mana ekspresi wajah atau keseluruhan wajah tergambar dengan jelas.

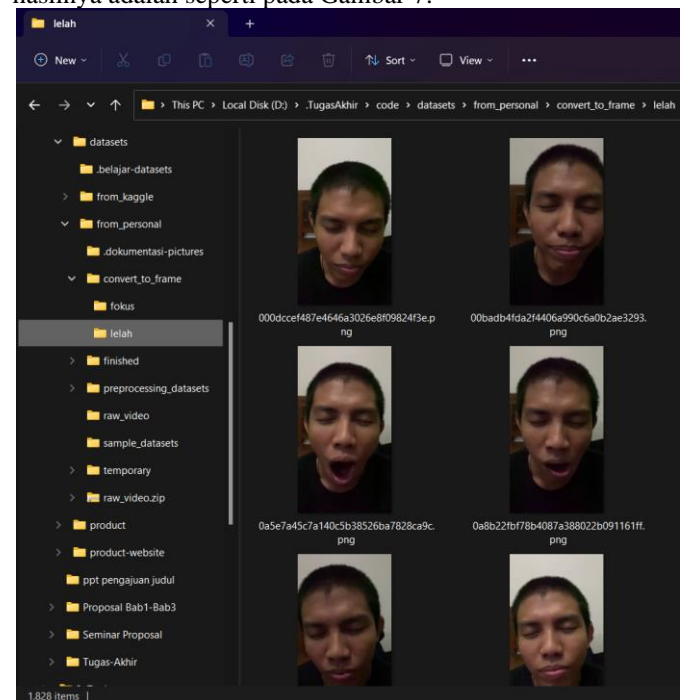
Peneliti membuat dataset pribadi seperti pada Gambar 6 dengan video berdurasi 30 detik tiap kelasnya dengan frame 60 fps. Video tersebut nantinya diekstrak tiap frame nya

sehingga menghasilkan kurang lebih 1800 frame (60 fps x 30 detik) untuk tiap kelasnya.



Gbr. 6. Video Pribadi Peneliti Tiap Kelasnya

Untuk proses ekstraksi video ke frame, peneliti menggunakan kode program yang sudah dibuat dan untuk hasilnya adalah seperti pada Gambar 7.



Gbr. 7. Hasil Ekstraksi Frame Pribadi Peneliti

Selanjutnya, dilakukan pengambilan 250 data acak tiap kelasnya pada dataset pribadi peneliti dan pengambilan 2000 data acak tiap kelasnya pada dataset dari Kaggle menggunakan metode *random sampling*.

B. Pre-processing Data

Pada proses *pre-processing* data, peneliti membagi tahapan menjadi 3 bagian, yaitu: proses *cleaning* data, proses augmentasi data dan proses anotasi data.

1) Cleaning Data

Proses *cleaning* memakan waktu yang cukup lama karena data yang dibersihkan bersifat *non-numeric*. Dataset yang disiapkan oleh peneliti berupa citra gambar, sehingga untuk membersihkan data dari pencilan adalah dengan melakukan metode *Check and Save*, yaitu memeriksa gambar satu per satu apakah citra gambar sesuai dengan data yang dibutuhkan (ekspresi citra wajah).

Pada tahap ini, peneliti membulatkan data untuk mempermudah tahapan akuisisi data nantinya dan mendapatkan hasil 3400 data mentah tanpa *outlier*.

2) Augmentasi Data

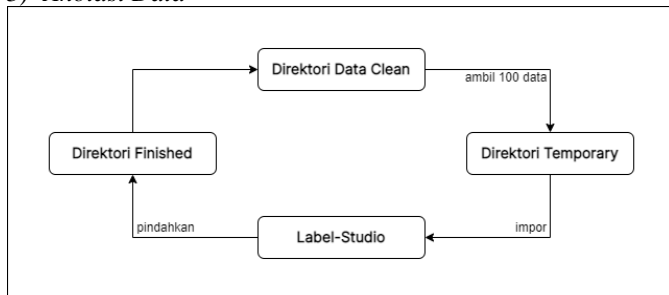
Proses ini dilakukan hanya pada data pribadi peneliti, karena pada dataset dari Kaggle, kebanyakan sudah dilakukan augmentasi *photometric distortion* (mengubah kecerahan, kontras, ataupun saturasi) dan augmentasi *geometric distortion* (flip horizontal). Untuk proses augmentasi, peneliti menggunakan kode program yang sudah dibuat menggunakan modul CV2 untuk melakukan flip dan menyimpannya hasilnya pada *path* yang sudah ditentukan menggunakan *Computer Vision*.



Gbr. 8. Hasil dari Proses Augmentasi (Flip Horizontal)

Gambar 8 adalah hasil dari proses Augmentasi. Setelah proses augmentasi dilakukan, data pribadi peneliti yang semula berjumlah 200 naik 2 kali lipat menjadi 400 data tiap kelasnya. Pada tahap augmentasi, peneliti telah memperoleh 3800 data bersih yang siap dan layak untuk dilakukan anotasi.

3) Anotasi Data

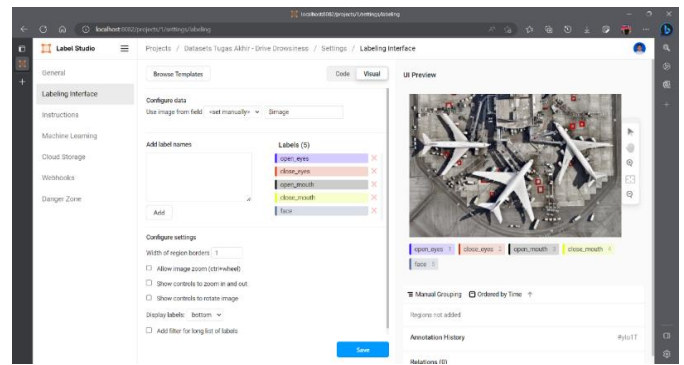


Gbr. 9. Siklus Proses Anotasi

Proses anotasi dilakukan menggunakan tools Label-Studio.

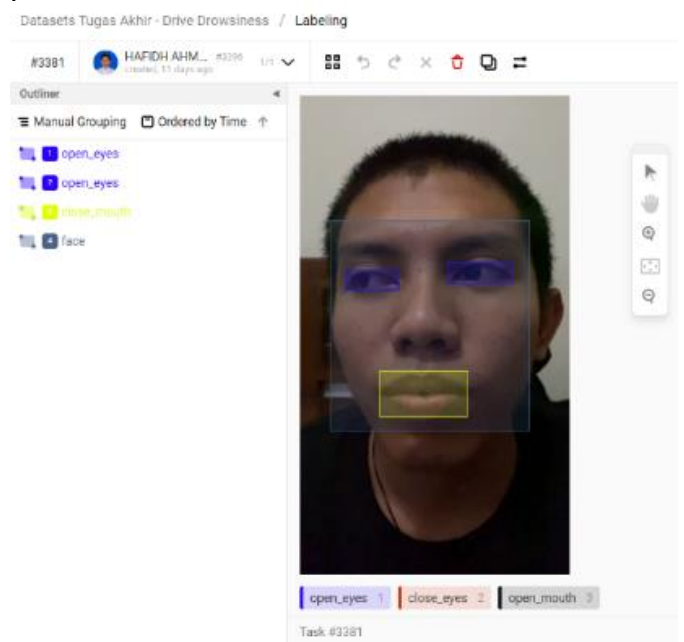
Karena Label-Studio hanya dapat menampung 100 data saja disetiap proses impornya, maka peneliti membuat sebuah siklus yang dikerjakan secara otomatis menggunakan kode program seperti Gambar 9. Dimulai dengan mengambil 100 data untuk dipindahkan ke folder Temporary. Impor dan anotasi data pada Label-Studio. Lalu, pindahkan data pada folder Finished ketika selesai dan lakukan kembali seperti siklus yang sudah digambarkan hingga semua data sudah dianotasi.

Sebelum memulai proses anotasi, perlu dilakukan konfigurasi proyek pada Label-Studio, yaitu: (1) mata terbuka, (2) mata tertutup, (3) mulut terbuka, (4) mulut tertutup dan (5) wajah seperti pada Gambar 10.



Gbr. 10. Konfigurasi Proyek Baru Label-Studio

Setelah konfigurasi proyek selesai dilakukan, maka penelitian dilanjutkan dengan anotasi ekspresi wajah satu persatu.



Gbr. 11. Anotasi Citra Label-Studio

Gambar 11 adalah proses anotasi yang dilakukan dengan memberikan label menggunakan *bounding box* pada ekspresi wajah. Jika dataset yang dibutuhkan sudah dilakukan anotasi seluruhnya, maka hasil anotasi dapat diekspor sesuai dengan format model YOLOv4. Hasil ekspor data dari Label-Studio berupa file .zip yang berisi citra dataset (foto data yang dianotasi), hasil anotasi (berupa file .txt), classess.txt (label yang sudah dikonfigurasi), serta tambahan file berupa notes.json.

C. Pelatihan Data

Dimulai dari proses pelatihan data hingga proses evaluasi data, peneliti menggunakan model arsitektur Darknet YOLOv4 sehingga seluruh prosesnya berada pada lingkungan Google Colab untuk dapat memanfaatkan GPU gratis dan Google Drive sebagai tempat penyimpanannya. Peneliti membagi tahapan menjadi 3 bagian, yaitu: konfigurasi model arsitektur Darknet YOLOv4, akuisisi data dan pelatihan data pada model YOLOv4.

1) Konfigurasi Model Arsitektur Darknet YOLOv4

```

1 # install arsitektur yolov4 from original github
2 !git clone https://github.com/AlexeyAB/darknet
3
4 # install yolov4.conv.137
5 !wget
6 https://github.com/AlexeyAB/darknet/releases/download/
7 darknet_yolo_v3_optimal/yolov4.conv.137 -P
8 "$path_darknet"

```

Gbr. 12. Membangun Arsitektur Darknet YOLOv4

Gambar 12 menjelaskan sebuah logika agar model tidak dibangun ulang apabila sebelumnya model sudah dibangun dan disimpan pada Google Drive. Pada saat model sudah dibangun, langkah selanjutnya adalah memuat *weights pre-trained* YOLOv4 yang disimpan di Darknet agar proses pelatihan data dapat dilakukan, pada kode program di atas peneliti memuat file *yolo4.conv.137* yang diambil menggunakan kode perintah “!wget” pada *original* Github. Setelah model Darknet dan *weight pre-trained* berhasil dibangun, selanjutnya adalah menjalankan GPU milik Google Colab dan menjalankan kode berikut yang berfungsi untuk mengoptimalkan penggunaan GPU seperti Gambar 13.

```

1 !sed -i 's/OPENCV=0/OPENCV=1/' Makefile
2 !sed -i 's/GPU=0/GPU=1/' Makefile
3 !sed -i 's/CUDNN=0/CUDNN=1/' Makefile
4 !sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
5 !make

```

Gbr. 13. Menjalankan GPU Secara Optimal

Sebelumnya peneliti sudah mengunggah folder *config_yolov4* pada Google Drive, sehingga ketika memasuki proses pelatihan data tinggal memanggilnya saja. Folder *config_yolov4* berisi file konfigurasi objek yang dilatih, seperti:

– **Obj.data**

File *obj.data* memuat *path* tempat menyimpan konfigurasi dan data, serta total kelas yang dilatih. Berikut merupakan isi dari file *obj.data*:

```

1 classes = 5
2 train = /content/gdrive/MyDrive/train.txt
3 valid = /content/gdrive/MyDrive/test.txt
4 names = /content/gdrive/MyDrive/obj.names
5 backup = /content/gdrive/MyDrive/model_yolov4

```

– **Obj.names**

File *obj.names* memuat label anotasi yang dilatih (sesuaikan dengan file hasil dari Label-Studio). Berikut merupakan isi dari file *obj.names*:

```

1 close_eyes
2 close_mouth
3 face
4 open_eyes
5 open_mouth

```

– **Yolo4-custom**

File *yolov4-custom* memuat konfigurasi *hyperparameters* YOLOv4, adapun konfigurasi yang diubah oleh peneliti adalah:

TABEL III
KONFIGURASI HYPERPARAMETER YOLOV4 DETEKSI KELELAHAN

Parameter	Value
<i>Classes</i>	5
<i>Network Size</i>	416 x 416
<i>Batch</i>	64
<i>Subdivision</i>	16

<i>Max Batches</i>	10000
<i>Steps</i>	8000, 9000
<i>Filters</i>	30
<i>Channel</i>	3
<i>Learning Rate</i>	0.001

2) **Akuisisi Data**

Sebelum masuk ke proses akuisisi, hal yang disiapkan adalah membuat folder tempat penyimpanan data latih (*path* : *darknet/data/train*) dan data uji (*path* : *darknet/data/test*) dalam model arsitektur Darknet. Dilanjutkan dengan melakukan unzip file hasil ekspor dataset pada penyimpanan Google Drive. Pada proses akuisisi, peneliti menggunakan aturan umum *Machine Learning*, yaitu konsep “*Train-Test-Split*” dengan menggunakan pembagian 80:20 (80% data latih, 20% data uji) seperti Gambar 14.

```

1 # split data train & test
2 train_composition = int((len(list_img_datasets) *
3 0.8))
4 test_composition = len(list_img_datasets) -
5 train_composition

```

Gbr. 14. Akuisisi Dataset

Ketika proses akuisisi data selesai, maka proses selanjutnya adalah melakukan konfigurasi *path* pada tiap frame yang ada pada folder train dan folder test supaya model dapat membaca letak data latih dan data uji yang sudah dibuat menggunakan kode program berikut:

```

1 for file_name in os.listdir(path_train_in_data):
2     if file_name.split(".")[1].lower() in ["jpg",
3 "png", "jpeg"] :
4         image_files.append(path_train_in_data + "/" +
5 file_name)
6 with open(os.path.join(path_data_in_darknet,
7 "train.txt"), "w") as outfile:
8     for frame in image_files:
9         outfile.write(frame)
10        outfile.write("\n")
11    outfile.close()

```

Gbr. 15. Konfigurasi File Train.txt

```

1 for file_name in os.listdir(path_test_in_data):
2     if file_name.split(".")[1].lower() in ["jpg",
3 "png", "jpeg"] :
4         image_files.append(path_test_in_data + "/" +
5 file_name)
6 with open(os.path.join(path_data_in_darknet,
7 "test.txt"), "w") as outfile:
8     for frame in image_files:
9         outfile.write(frame)
10        outfile.write("\n")
11    outfile.close()

```

Gbr. 15. Konfigurasi File Test.txt

Gambar 14 dan Gambar 15, peneliti membuat kode yang sifatnya *reusable* sehingga dapat digunakan kembali apabila proses gagal atau ingin melakukan pelatihan data lainnya.

3) **Pelatihan Data pada Model YOLOv4**

Setelah model Darknet dan konfigurasi YOLOv4 sudah dibuat, selanjutnya proses pelatihan data dapat dilakukan. Karena pada pelatihan model YOLO memerlukan GPU, maka proses dilakukan menggunakan Google Colab. Berikut adalah kode program yang dijalankan untuk melatih data pada model arsitektur Darknet YOLOv4 seperti Gambar 16.

1	!./darknet detector train \
2	"../config_yolov4/obj.data" \
3	"../config_yolov4/yolov4-custom.cfg" \
4	./yolov4.conv.137 \
5	-dont_show

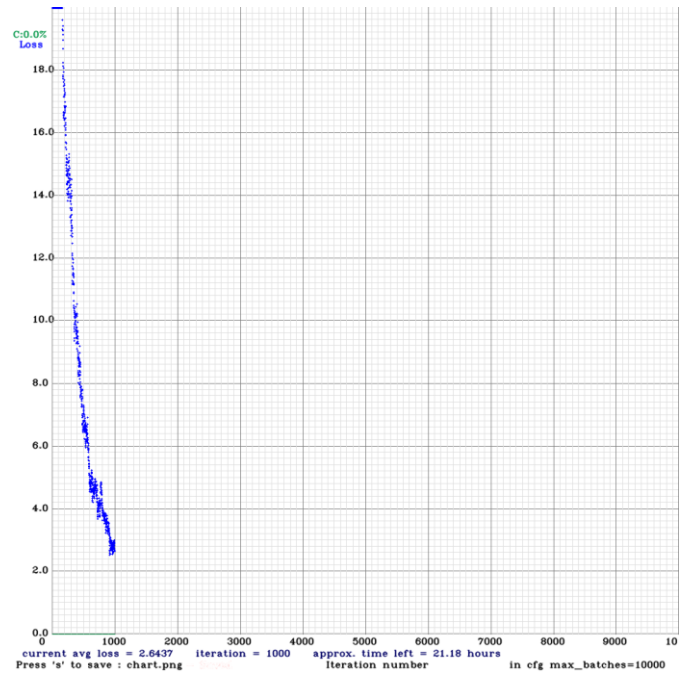
Gbr. 16. Menjalankan Pelatihan Arsitektur Darknet YOLOv4

Proses pelatihan data memerlukan waktu yang cukup lama (lebih dari 12 jam) karena peneliti memberikan nilai 0.001 pada konfigurasi *learning rate*. Selama proses pelatihan, bukan tidak mungkin terdapat hambatan yang mengakibatkan pelatihan data berakhir, seperti sistem mendeteksi tidak ada aktivitas, koneksi internet yang tidak stabil bahkan mati, GPU gratis yang tersedia sudah habis. Oleh karenanya, *mouse* atau *keyboard* sesekali ditekan agar perangkat mendeteksi *user* tidak melakukan AFK (*Away From Keyboard*) dan memaksa program untuk berhenti. Data hasil deteksi beserta bobot nilai mAP tertinggi dari setiap kelipatan 100 iterasi tetap tersimpan pada *path backup* yang sudah dikonfigurasi pada file *obj.data* dengan nama file *yolov4-custom_last.weight*. Pada saat melanjutkan pelatihan akibat terjadinya hambatan, target pelatihan tidak lagi menggunakan *pre-trained weight* milik file *yolov4.conv.137*, tetapi menggunakan hasil dari data yang sudah disimpan pada *backup* sebelum kendala terjadi.

D. Evaluasi Data

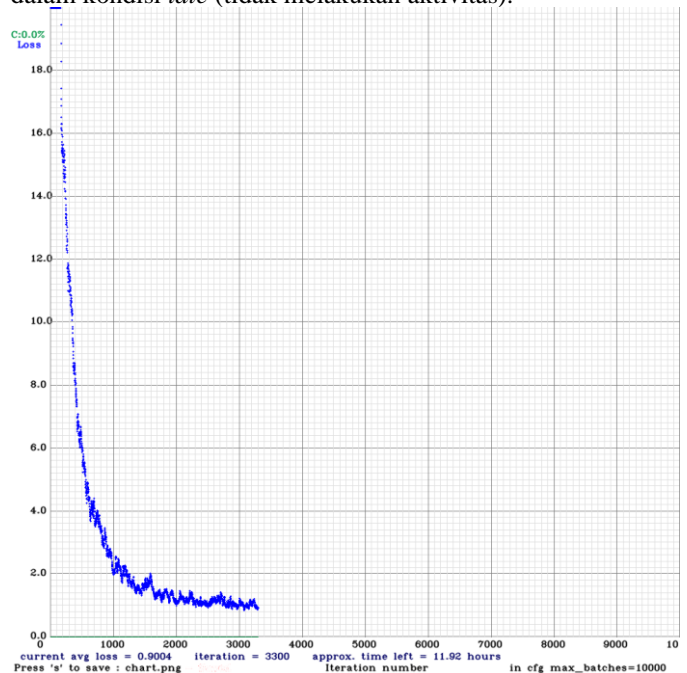
Proses evaluasi data berlangsung bersamaan dengan proses pelatihan data. Hasil dari evaluasi akurasi terbaik dan akurasi terakhir ditampilkan secara singkat setiap 112 iterasi selama proses pelatihan berjalan. Oleh karena itu, nilai detektor mAP dan grafik proses pelatihan dibuat setelah proses berhenti atau berakhir berupa grafik yang tersimpan secara otomatis pada model arsitektur Darknet YOLOv4.

Pada saat proses pelatihan, peneliti mengalami beberapa kendala, seperti pada percobaan pertama, model Darknet tidak dimasukkan ke dalam Google Drive, sehingga ketika GPU sudah habis dipakai, *runtime* secara otomatis kembali ke tampilan awal (tampilan kosong atau data hilang) dan file pada Google Colab runtuh, hal ini mengakibatkan nilai detektor mAP dan grafik proses pelatihan pada Darknet tidak tersimpan. Lalu pada percobaan kedua, peneliti mengalami kendala yang sama tetapi sudah membangun model Darknet pada Google Drive, sehingga ketika GPU sudah habis terpakai atau kendala lain yang menyebabkan harus memulai ulang *runtime* nilai detektor mAP dan grafik proses sudah tersimpan. Berikut hasil evaluasi dari grafik yang sudah tersimpan:



Gbr. 17. Grafik Nilai mAP iterasi ke-0 sampai ke-1000

Pada Gambar 17 menjelaskan hasil evaluasi pelatihan data dari 1000 iterasi, GPU yang digunakan mati karena pada saat percobaan Google Colab mendeteksi bahwa *user* sedang dalam kondisi *idle* (tidak melakukan aktivitas).



Gbr. 18. Grafik Nilai mAP iterasi ke-0 sampai ke-3300

Pada Gambar 18 menjelaskan hasil evaluasi pelatihan data dari 3300 iterasi, GPU yang digunakan sudah habis ketika data mencapai 3344 iterasi, sehingga data yang disimpan hanya sampai iterasi 3300 saja.

Selanjutnya, untuk menampilkan nilai mAP, peneliti menggunakan kode perintah yang sudah disediakan pada model arsitektur Darknet YOLOv4 seperti berikut:


```

1 !./darknet detector map \
2  "../config_yolov4/obj.data" \
3  "../config_yolov4/yolov4-custom.cfg" \
4  "../model_yolov4/2023-04-09/yolov4-custom_last.weights"

```

Gbr. 19. Menjalankan Evaluasi Arsitektur Darknet YOLOv4

Gambar 19 adalah kode program untuk menampilkan nilai evaluasi dari file *weight* yang disimpan ketika proses pelatihan data berakhir, yaitu nilai mAP, *Precision*, *Recall*, *F1-Score*. Berikut adalah hasil dari kode program yang sudah dijalankan:

```

calculation mAP (mean average precision)...
Detection layer: 139 - type = 28
Detection layer: 150 - type = 28
Detection layer: 161 - type = 28
4
detections count = 42, unique_truth_count = 16
class_id = 0, name = close_eyes, ap = 100.00% (TP = 4, FP = 0)
class_id = 1, name = close_mouth, ap = 100.00% (TP = 2, FP = 1)
class_id = 2, name = face, ap = 100.00% (TP = 4, FP = 0)
class_id = 3, name = open_eyes, ap = 100.00% (TP = 4, FP = 0)
class_id = 4, name = open_mouth, ap = 100.00% (TP = 2, FP = 0)

for conf_thresh = 0.25, precision = 0.94, recall = 1.00, F1-score = 0.97
for conf_thresh = 0.25, TP = 16, FP = 1, FN = 0, average IoU = 74.46 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 1.000000, or 100.00 %
Total Detection Time: 5 Seconds

```

Gbr. 20. Hasil Evaluasi 20 Data (1000 iterasi)

Gambar 20 menunjukkan nilai evaluasi akhir dari pelatihan 20 dataset dengan file *weight* 1000 iterasi (data yang disimpan ketika proses pelatihan data berakhir). Hasil evaluasi data yang didapat berupa nilai akhir mAP sebesar 1.000000 atau 100.00%, nilai akhir *Precision* sebesar 0.94, nilai akhir *Recall* sebesar 1.00 dan nilai akhir *F1-Score* sebesar 0.97 yang sudah dilatih oleh model menggunakan nilai ambang batas IOU sebesar 50% atau 0.5. Karena nilai akurasi YOLOv4 dihitung menggunakan mAP, maka akurasi yang didapat dari hasil pelatihan 20 data dengan nilai *weight* yang tersimpan pada 1000 iterasi adalah 100.00%.

```

calculation mAP (mean average precision)...
Detection layer: 139 - type = 28
Detection layer: 150 - type = 28
Detection layer: 161 - type = 28
240
detections count = 1410, unique_truth_count = 958
class_id = 0, name = close_eyes, ap = 94.20% (TP = 32, FP = 0)
class_id = 1, name = close_mouth, ap = 98.80% (TP = 212, FP = 6)
class_id = 2, name = face, ap = 100.00% (TP = 240, FP = 0)
class_id = 3, name = open_eyes, ap = 99.76% (TP = 444, FP = 6)
class_id = 4, name = open_mouth, ap = 92.37% (TP = 21, FP = 2)

for conf_thresh = 0.25, precision = 0.99, recall = 0.99, F1-score = 0.99
for conf_thresh = 0.25, TP = 949, FP = 14, FN = 9, average IoU = 81.05 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.970277, or 97.03 %
Total Detection Time: 11 Seconds

```

Gbr. 21. Hasil Evaluasi 3800 Data (3300 iterasi)

Gambar 21 menunjukkan nilai evaluasi akhir dari pelatihan 3800 dataset dengan file *weight* 3300 iterasi (data yang disimpan ketika proses pelatihan data berakhir). Hasil evaluasi data yang didapat berupa nilai akhir mAP sebesar 0.970277 atau 97.03%, nilai akhir *Precision*, nilai akhir *Recall* dan nilai akhir *F1-Score* sebesar 0.99 yang sudah dilatih oleh model menggunakan nilai ambang batas IOU sebesar 50% atau 0.5. Karena nilai akurasi YOLOv4 dihitung menggunakan mAP seperti yang sudah dijelaskan pada dasar teori sebelumnya, maka akurasi yang didapat dari hasil pelatihan 3800 data dengan nilai *weight* yang tersimpan pada 3300 iterasi adalah **97.03%**.

Dapat disimpulkan bahwa model *Deep Learning* berbasis YOLO memiliki kemampuan generalisasi yang sangat tinggi sehingga dengan sedikit data saja model dapat mendapatkan nilai akurasi yang baik, tidak seperti model *Deep Learning* lainnya yang membutuhkan data yang sangat banyak untuk menghasilkan akurasi yang tinggi [4]. Hal ini juga seperti penelitian yang dilakukan oleh Hidayatullah (2021) dimana model pendeteksinya mencapai nilai akurasi 64,45 mAP, walaupun jumlah data latih yang digunakan hanya 5 gambar saja.

E. Pengujian Produk Deteksi Kelelahan

Setelah hasil evaluasi berupa nilai mAP (tingkat akurasi deteksi) diperoleh, peneliti menguji produk model deteksi kelelahan menggunakan media foto, video dan webcam secara *real-time* untuk mengetahui ketepatan serta keakuratan hasil deteksi pada model menggunakan kode program yang sudah dibuat.

```

Nama file : yolo_opencv.py
1 def predict(self, img):
2     image = img
3
4     # CONVERT IMAGE TO BLOB
5     blob = cv2.dnn.blobFromImage(image, 1 / 255,
6     (self.size_model, self.size_model), (0, 0, 0), 1,
7     crop=False)
8
9     # RUN INFERENCE ON THE YOLO MODEL
10    self.net.setInput(blob)
11    layer_names = self.net.getLayerNames()
12    output_names = [layer_names[i[0] - 1] for i in
13    self.net.getUnconnectedOutLayers()]
14    outputs = self.net.forward(output_names)
15
16    # GET IMAGE & RESULT PREDICTIONS
17    final_img, results =
18    self.__postprocess_result(outputs, image)
19
20    return final_img, results

```

Gbr. 22. Logika Produk Model Deteksi Kelelahan

Sebelum melakukan prediksi, objek yang dideteksi dirubah ke dalam bentuk BLOB (*Binary Large Object*) terlebih dahulu, selanjutnya nilai BLOB dijalankan pada model deteksi kelelahan. Pengujian model deteksi kelelahan dilakukan menggunakan media foto, video dan juga webcam secara *real-time* seperti berikut:

1) Pengujian Menggunakan Gambar

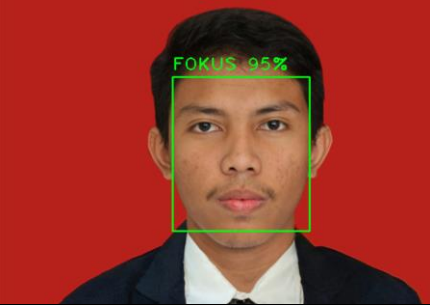
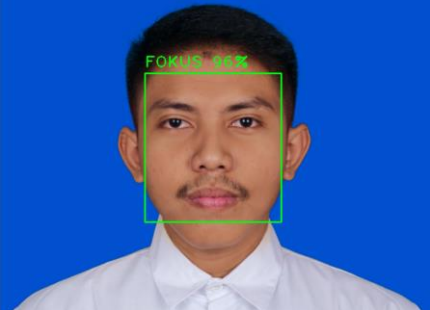

Proses yang berlangsung adalah :

1. Gambar dibaca menggunakan fungsi "cv2.imread()" milik modul *Open Computer Vision*.
2. Untuk mempercepat proses deteksi, dilakukan manipulasi dimensi gambar menggunakan fungsi "cv2.resize()".
3. Gambar dimasukkan pada model deteksi "yolo.predict()".
4. Hasil dari deteksi disesuaikan dengan argumen yang sudah dimasukkan, seperti tampilkan hasil deteksi pada *window* saja atau simpan hasil deteksi pada *path* yang ditentukan saja atau tampilkan hasil deteksi pada

window dan simpan hasil deteksi pada *path* yang ditentukan.

Untuk menampilkan hasil deteksi pada *window* menggunakan fungsi dari “cv2.imshow()” dan untuk menyimpan hasil deteksi menggunakan fungsi dari “cv2.imwrite()”. Peneliti menggunakan gambar pribadi sebagai objek yang dideteksi. Berikut adalah tabel hasil keakuratan deteksi yang dihasilkan:

TABEL IV
HASIL DETEKSI KELELAHAN MENGGUNAKAN GAMBAR

Gambar	Hasil
	<ul style="list-style-type: none"> Kelas : Fokus Confidence : 95%
	<ul style="list-style-type: none"> Kelas : Fokus Confidence : 96%
	<ul style="list-style-type: none"> Kelas : Lelah Confidence : 75%

2) Pengujian Menggunakan Video

Proses yang berlangsung adalah :

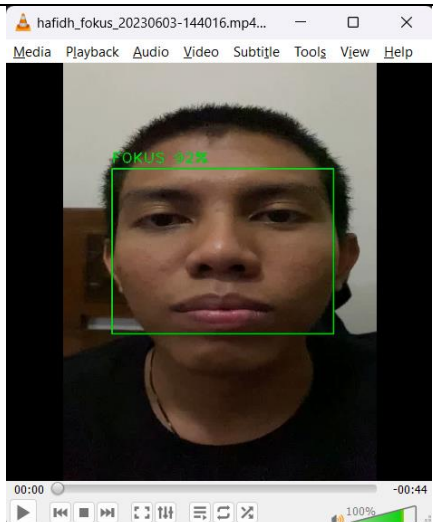
1. Video dibuka menggunakan fungsi “cv2.VideoCapture()” milik modul *Open Computer Vision*.
2. Sesuai dengan argumen yang sudah dimasukkan, deteksi dengan video hanya dapat menampilkan hasil deteksi pada *window* saja atau menampilkan hasil

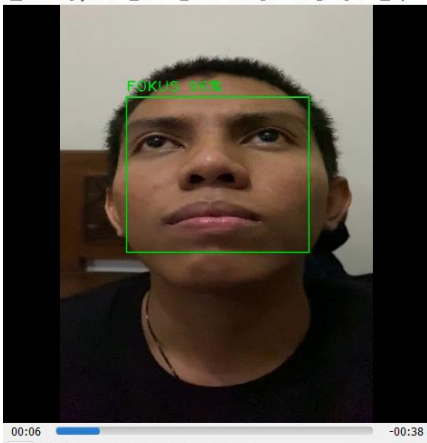

deteksi pada *window* dan menyimpannya pada *path* yang ditentukan. Sedangkan untuk langsung menyimpan saja tidak dapat dilakukan, karena pada proses deteksi perlu untuk membaca tiap frame yang ditampilkan sebelum disimpan pada *path* yang ditentukan.

3. Video dibaca menggunakan fungsi “video.read()” milik modul *Open Computer Vision* untuk diambil frame tiap detiknya.
4. Buat pengkondisian apakah video sedang dibaca atau tidak, apabila iya cek apakah variabel *ret* bernilai *true* (*ret* adalah variabel untuk mengecek apakah video berhasil dibaca), jika bernilai *true*, manipulasi dimensi frame agar proses deteksi berjalan cepat dan masukkan pada model deteksi “yolo.predict()”.

Untuk menampilkan hasil deteksi pada *window* menggunakan fungsi dari “cv2.imshow()” selama video berhasil dibaca, sedangkan untuk menyimpan hasil deteksi, hal pertama yang dilakukan adalah menentukan aturan kode pengkodean terlebih dahulu untuk membaca dan menyimpan video pada format yang sudah diatur sesuai dengan kebutuhan dan perangkat yang digunakan. Peneliti menggunakan format “XVID” pada fungsi “cv2.VideoWriter_fourcc()” karena menggunakan *os windows*. Kesalahan dalam pemilihan kode mengakibatkan video tidak dapat dibaca dan ukuran file yang tidak normal (terlalu besar). Selanjutnya video ditulis ulang sesuai dengan format pengkodean menggunakan fungsi “cv2.VideoWriter()” milik modul *Open Computer Vision* dan diisi dengan hasil deteksi menggunakan fungsi “out.write()” (*out* adalah variabel yang berisi fungsi “cv2.VideoWriter()”) selama video berhasil dibaca. Peneliti menggunakan video pribadi sebagai objek yang dideteksi. Berikut adalah tabel hasil keakuratan deteksi yang dihasilkan:

TABEL V
HASIL DETEKSI KELELAHAN MENGGUNAKAN VIDEO

Gambar	Hasil
	<ul style="list-style-type: none"> Kelas : Fokus Confidence : 92%

	<ul style="list-style-type: none"> • Kelas : Fokus • Confidence : 96%
	<ul style="list-style-type: none"> • Kelas : Lelah • Confidence : 87%

3) Pengujian Menggunakan Webcam

Proses yang berlangsung adalah :

1. Webcam dibuka menggunakan fungsi "cv2.VideoCapture()" milik modul *Open Computer Vision*.
2. Untuk mempercepat proses deteksi, dilakukan manipulasi dimensi webcam menggunakan fungsi "video.set()".
3. Sesuai dengan argumen yang sudah dimasukkan, deteksi dengan webcam secara *real-time* hanya dapat menampilkan hasil deteksi pada *window* saja atau menampilkan hasil deteksi pada *window* dan menyimpannya pada *path* yang ditentukan. Sedangkan untuk langsung menyimpan saja tidak dapat dilakukan, karena pada proses deteksi perlu untuk membaca tiap frame yang yang ditampilkan sebelum disimpan pada *path* yang ditentukan.
4. Webcam dibaca menggunakan fungsi "video.read()" milik modul *Open Computer Vision* untuk diambil frame tiap detiknya.

5. Buat pengkondisian apakah webcam sedang dibaca, apabila iya cek apakah variabel *ret* bernilai *true* (*ret* adalah variabel untuk mengecek apakah webcam berhasil dibaca), jika bernilai *true*, manipulasi dimensi frame agar proses deteksi berjalan cepat dan masukkan pada model deteksi "yolo.predict()".

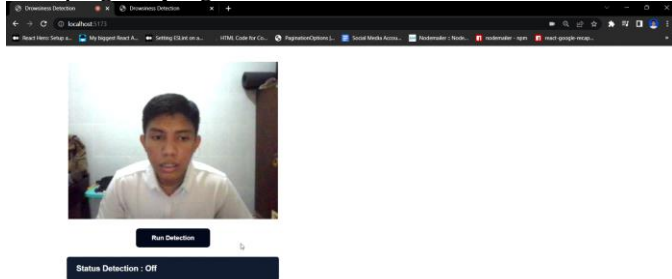
Cara yang dilakukan untuk menampilkan hasil deteksi sama seperti pada pengujian menggunakan video dan untuk subjek yang dideteksi adalah peneliti sendiri. Berikut adalah tabel hasil keakuratan deteksi yang dihasilkan:

TABEL VI
HASIL DETEKSI KELELAHAN MENGGUNAKAN WEBCAM

Gambar	Hasil
	<ul style="list-style-type: none"> • Kelas : Fokus • Confidence : 76%
	<ul style="list-style-type: none"> • Kelas : Lelah • Confidence : 95%
	<ul style="list-style-type: none"> • Kelas : Lelah • Confidence : 79%

F. Pengujian Produk Pada Aplikasi Website

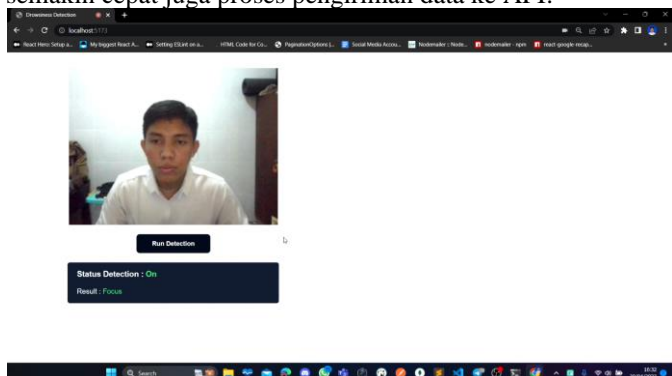
Setelah produk model deteksi kelelahan selesai diuji dan menghasilkan tingkat akurasi yang baik dan tepat. Langkah selanjutnya adalah mengimplementasikan produk model deteksi kelelahan pada website dan memberikan fitur *warning alert* berupa suara sirine peringatan dan notifikasi pada Telegram ketika objek menghasilkan nilai lelah menggunakan kode program yang sudah dibuat.



Gbr. 23. Halaman Muka Website Sebelum Deteksi Dijalankan

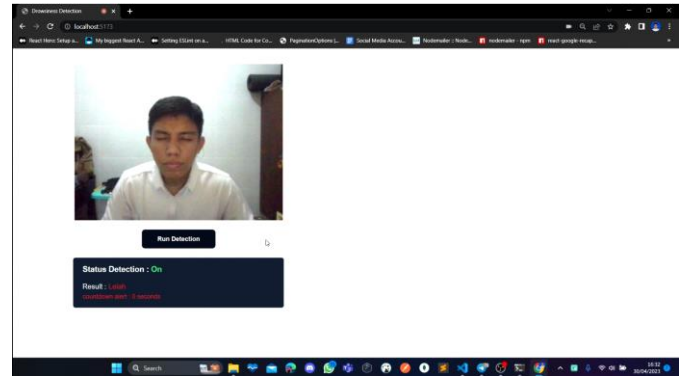
Gambar 23 menjelaskan tentang tampilan awal yang dibuat berupa fitur untuk webcam, tombol untuk menjalankan deteksi dan juga status. Alur proses yang terjadi ketika tombol “Run Detection” ditekan adalah frame tiap detik yang ditangkap melalui webcam dikirimkan pada API produk model deteksi kelelahan, API diakses (*request*) tiap detik karena bersifat *real-time*. Hasil dari deteksi dikirimkan kembali dan ditampilkan pada status.

Pada implementasi website, peneliti tidak menampilkan nilai *confidence* dan *bounding box* dari hasil deteksi yang didapatkan. Karena untuk menampilkan *bounding box* dan nilai *confidence*, frame perlu ditulis ulang seperti halnya dalam pengujian model melalui webcam secara *real-time* sebelumnya, ketika frame ditulis ulang, tampilan subjek yang dideteksi pada layar *window* atau webcam terjadi *lag*, hal ini menghambat jalannya *request* data ke API tiap detiknya dan mempengaruhi performa website karena menyebabkan webcam yang ditampilkan *lagging* (frame putus-putus). Hal lain yang mempengaruhi kecepatan proses deteksi adalah koneksi internet, semakin kuat koneksi internet yang tersedia, semakin cepat juga proses pengiriman data ke API.



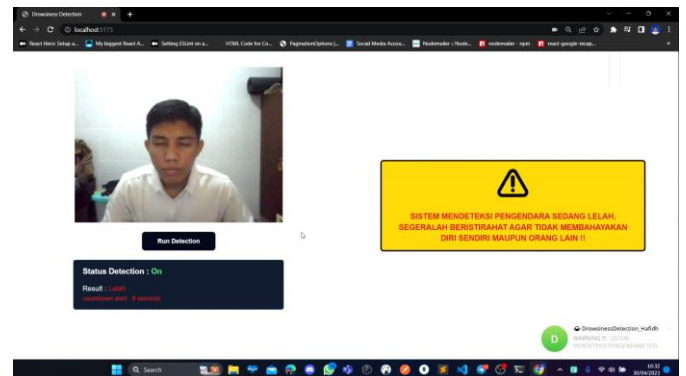
Gbr. 24. Deteksi Berjalan (Status: Fokus)

Gambar 24 menampilkan ketika subjek yang dideteksi dalam kondisi fokus, maka status yang ditampilkan berwarna hijau dengan nilai hasil deteksi berupa fokus. Seperti yang sudah dijelaskan sebelumnya, ciri-ciri dari kelas fokus adalah ketika kedua mata terbuka atau tampak segar.



Gbr. 25. Deteksi Berjalan (Status: Lelah)

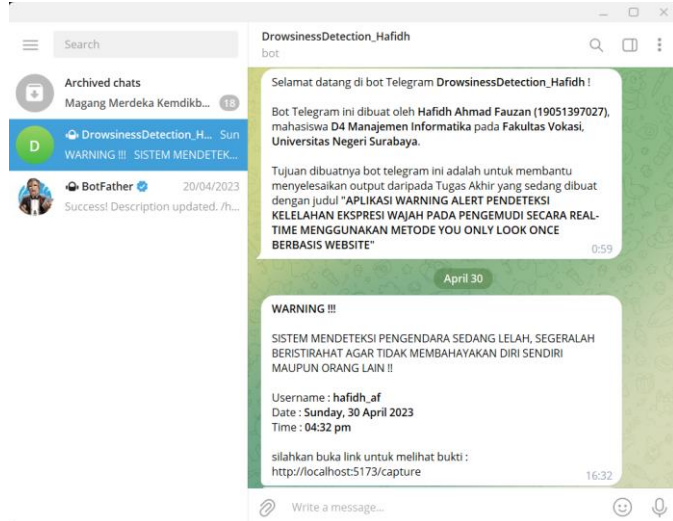
Pada Gambar 25 menampilkan ketika subjek yang dideteksi dalam kondisi lelah, maka status yang ditampilkan berwarna merah dengan nilai hasil deteksi berupa lelah, serta terdapat tambahan status dengan nama “countdown”. *Countdown* berfungsi untuk mengaktifkan peringatan (*warning alert*) ketika kondisi lelah dalam jangka waktu yang ditentukan sudah terpenuhi. Peneliti memberikan nilai *countdown* selama 8 detik sebelum peringatan dan sirine aktif, nilai *countdown* tidak memiliki aturan dan bisa diganti berapapun waktu yang diinginkan, sedangkan alasan peneliti memberikan waktu 8 detik adalah karena faktor *device* yang digunakan tidak terlalu tinggi, dan internet yang kurang stabil. Seperti yang sudah dijelaskan sebelumnya ciri dari kelas lelah adalah ketika sedang menguap, ketika salah satu mata tertutup atau kedua mata tertutup (tidur) atau mata tidak terbuka secara sempurna (tidak tampak segar).



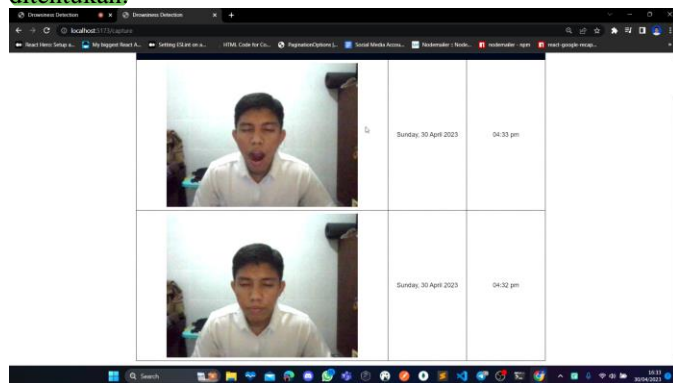
Gbr. 26. Peringatan Aktif (Kondisi Lelah Lebih Dari 8 Detik)

Pada Gambar 26, peringatan (*warning alert*) aktif karena kondisi *countdown* yang ditentukan sudah terpenuhi, di sini peneliti memberikan waktu *countdown* selama 8 detik, hal ini bertujuan agar peringatan tidak aktif ketika objek yang dideteksi sedang berkedip. Hal tersebut merupakan efek daripada kecepatan yang dihasilkan ketika sedang mendeteksi sesuatu secara *real-time* (60 fps). Ketika objek yang dideteksi sedang berkedip, sesaat model deteksi menghasilkan kondisi lelah karena pada saat itu kedua mata sedang tertutup.

Peringatan yang dimunculkan berupa teks peringatan pada tampilan website seperti gambar di atas dan suara sirine peringatan yang menyala selama kondisi lelah sudah berlangsung lebih dari sama dengan 8 detik, suara sirine peringatan berhenti ketika subjek sudah kembali terdeteksi fokus, serta notifikasi yang dikirimkan oleh “Bot DrowsinessDetection_Hafidh” pada Telegram seperti Gambar 27 di bawah.



Gbr. 27. Peringatan Aktif (Kondisi Lelah Lebih Dari 8 Detik)
Notifikasi yang dikirimkan pada Telegram melalui “Bot DrowsinessDetection_Hafidh” berupa teks peringatan yang disertai *username*, tanggal dan waktu ketika subjek terdeteksi dalam kondisi lelah selama jangka waktu yang sudah ditentukan.



Gbr. 28. Halaman Capture

Pada Gambar 28 menjelaskan tentang halaman Capture dimana data yang tersimpan akan secara otomatis diperbarui dan ditampilkan paling awal ketika subjek terdeteksi sedang lelah. Halaman Capture berfungsi sebagai bukti dan penguat

notifikasi yang dikirimkan pada Telegram, karena di sini tersimpan bukti kapan subjek terdeteksi sedang lelah.

IV. KESIMPULAN DAN SARAN

Kesimpulan yang didapatkan setelah semua proses yang dikerjakan adalah YOLO memiliki kemampuan generalisasi yang sangat tinggi sehingga dengan sedikit data saja model dapat mendapatkan nilai akurasi yang baik, model mampu menghasilkan nilai akhir mAP sebesar 1.000000 atau 100.00%, nilai akhir *Precision* sebesar 0.94, nilai akhir *Recall* sebesar 1.00 dan nilai akhir *F1-Score* sebesar 0.97 hanya dengan 20 data saja. Berdasarkan penelitian oleh Bochkovskiy (pengembang metode YOLOv4), faktor yang mempengaruhi akurasi adalah penggunaan metode BoF (Bag of Freebies) dan BoS (Bag of Specials) yang ada pada konfigurasi YOLOv4. Produk yang dibuat menggunakan model deteksi kelelahan YOLOv4, dapat digunakan secara baik dalam mendeteksi objek melalui gambar, video maupun webcam secara *real-time*. Terakhir, peringatan yang diberikan pada aplikasi website ketika subjek terdeteksi lelah sangat jelas, seperti teks peringatan pada tampilan website, suara sirine peringatan, notifikasi peringatan pada Telegram, serta bukti berupa tangkapan gambar ketika subjek sedang lelah yang dilengkapi dengan tanggal dan waktu.

Berdasarkan kesimpulan, dapat diambil beberapa saran seperti diharapkan ke depannya, banyak Anotator yang menyediakan *resource* dataset dengan banyak format sekaligus, pelatihan data model YOLOv4 alangkah baiknya untuk menggunakan Google Colab yang berlangganan (versi premium) karena batasan penggunaan GPU gratis pada Google Colab yang disediakan pada masa sekarang hanya selama 6 jam. Terakhir, untuk mendapatkan tingkat akurasi yang tinggi pada model YOLOv4, baiknya menggunakan nilai 0.001 pada *learning rate* meskipun proses pelatihan data akan berjalan lama.

REFERENSI

- [1] Sinha, A., Aneesh, R. P., & Gopal, S. K. (2021). Drowsiness Detection System using Deep Learning. Proceedings of 2021 IEEE 7th International Conference on Bio Signals, Images and Instrumentation, ICBSII 2021. <https://doi.org/10.1109/ICBSII51839.2021.9445132>
- [2] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (n.d.). You Only Look Once: Unified, Real-Time Object Detection. <https://pjreddie.com/yolo>
- [3] Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. <http://arxiv.org/abs/2004.10934>
- [4] Hidayatullah, P. (2021). Pendeteksian, Pelacakan, Dan Klasifikasi Sperma Untuk Pengukuran Motilitas Sperma Sapi. Jurnal Teknik Elektro dan Informatika (STEI) Institut Teknologi Bandung