UNIVERSITAS MUHAMMADIYAH JAKARTA

TUGAS AKHIR

DETEKSI DAN IDENTIFIKASI RAMBU-RAMBU LALU LINTAS BERBASIS ALGORITMA *YOU ONLY LOOK ONCE* (YOLO)

Diajukan Untuk Memenuhi Sebagian Persyaratan
Dalam Menempuh Ujian Sarjana Teknik Elektro FT-UMJ

Oleh:

BAROKAH ASMARAHMAN TAKAROB
2018420035

FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH JAKARTA
2022

LEMBAR PERSETUJUAN

Tugas Akhir dengan Judul:

DETEKSI DAN IDENTIFIKASI RAMBU-RAMBU LALU LINTAS BERBASIS ALGORITMA *YOU ONLY LOOK ONCE* (YOLO)

Dibuat untuk melengkapi kurikulum pada Jurusan Elektro Fakultas Teknik Universitas Muhammadiyah Jakarta dan disetujui untuk diajukan dalam Sidang Sarjana strata satu (S1) Program Studi Teknik Elektro.

Jakarta, 25 Juli 2022

Pembimbing

Ir. Husnibes Muchtar, M.T.

LEMBAR PENGESAHAN

Tugas Akhir dengan Judul:

DETEKSI DAN IDENTIFIKASI RAMBU-RAMBU LALU LINTAS BERBASIS ALGORITMA *YOU ONLY LOOK ONCE* (YOLO)

Dibuat untuk melengkapi kurikulum pada Jurusan Elektro Fakultas Teknik Universitas Muhammadiyah Jakarta dan disetujui untuk diajukan dalam Sidang Sarjana strata satu (S1) Program Studi Teknik Elektro.

Jakarta, 08 Agustus 2022

Kaprodi Teknik Elektro	Pembimbing
Ir. Husnibes Muchtar, M.T.	**Ir. Husnibes Muchtar, M.T.**
NIDN : 0303016301	**NIDN : 0303016301**

PERNYATAAN KEASLIAN KARYA TULIS

Dengan ini saya menyatakan bahwa, sejauh yang saya ketahui Karya Tulis ini bukan merupakan tiruan atau salinan atau duplikat dari Karya Tulis yang sudah dipublikasikan dan atau pernah dipakai untuk mendapatkan gelar kesarjanaan di lingkungan Universitas Muhammadiyah Jakarta maupun di Perguruan Tinggi atau Instansi manapun, kecuali pada bagian-bagian dimana sumber informasinya dicantumkan sebagaimana mestinya.

Jakarta, 25 Juli 2022

Penulis

(BAROKAH ASMARAHMAN TAKAROB)

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada:

Ir. Husnibes Muchtar, M.T.

Selaku pembimbing yang telah bersedia meluangkan waktunya untuk memberikan bimbingan, petunjuk dan saran-saran serta kemudahan lainnya, sehingga Tugas Akhir ini memberikan mafaat bagi penulis dan pembacanya.

Penulis

ABSTRAK

DETEKSI DAN IDENTIFIKASI RAMBU-RAMBU LALU LINTAS BERBASIS ALGORITMA *YOU ONLY LOOK ONCE* (YOLO)

(Barokah Asmarahman Takarob, 2018420035, 2022, 46 Halaman)

Salah satu masalah yang sulit dan menantang dari manajemen perkotaan khususnya di negara berkembang adalah masalah kecelakaan lalu lintas. Dalam pengolahan citra digital, penerapan *artificial intelligence* seperti *deep learning* belakangan ini meraih kesuskesan dalam pengolahan citra dan video skala besar sehingga menjadikan *deep learning* populer untuk digunakan dan dirancang, salah satunya dalam pengembangan fitur *autonomous vehicle*. Dalam penelitian ini akan disimulasikan identifikasi objek menggunakan algoritma YOLO v4 *(You Only Look Once)* yang merupakan pengembangan dari metode CNN *(Convolutional Neural Network)* dan dapat dijalankan secara *realtime* untuk mendeteksi objek yang akan dilalui oleh *autonomous vehicle.* Penelitian ini menggunakan 3 kelas yaitu rambu larangan parkir, larangan berhenti, dan larangan masuk dengan 329 dataset gambar. Hasil penelitian membuktikan algoritma *You Only Look Once v4* (YOLO v4) mampu mendeteksi objek dengan baik berdasarkan *output* dari hasil *testing* data dengan menghasilkan nilai *mAP (mean Average Precision)* sebesar 100 %, *accuracy* 95,80 % dan 53,2 *FPS (Frame Per Second)*.

Kata kunci: Deteksi Objek, *You Only Look Once v4*, OpenCV, Darknet

ABSTRACT

DETECTION AND IDENTIFICATION OF TRAFFIC SIGNS BASED ON YOU ONLY LOOK ONCE (YOLO) ALGORITHM

(Barokah Asmarahman Takarob, 2018420035, 2022, 46 Pages)

One of the difficult and challenging problems of urban management, especially in developing countries, is the problem of traffic accidents. In digital image processing, the application of artificial intelligence such as deep learning has recently achieved success in large-scale image and video processing, making deep learning popular for use and design, one of which is the development of autonomous vehicle features. In this study, object identification will be simulated using the YOLO v4 (You Only Look Once) algorithm which is a development of the CNN (Convolutional Neural Network) method and can be run in real time to detect objects that will be passed by an autonomous vehicle. This study uses 3 classes, namely no parking signs, no stop signs, and no entry restrictions with 329 image datasets. The results of the study prove that the You Only Look Once v4 (YOLO v4) algorithm is able to detect objects properly based on the output of the data testing results by producing an mAP (mean Average Precision) value of 100%, 95.80% accuracy and 53.2 FPS (Frame Per second).

Keywords: Object Detection, You Only Look Once v4, OpenCV, Darknet

KATA PENGANTAR

Assalamu'alaikum Warahmatullahi Wabarakatu

Segala puji dan syukur atas kehadirat Allah Subhanahu Wa Ta'ala yang selalu mencurahkan segala rahmat dan karunia-Nya tanpa batas, serta selalu teriring shalawat dan salam kepada junjungan dan suri tauladan kita semua yaitu Nabi Muhammad Sallallahu 'Alaihi Wa Sallam yang telah membawa kita dari zaman kebodohan menuju zaman yang penuh dengan ilmu pengetahuan sehingga dapat menyelesaikan Tugas Akhir dengan judul "Deteksi Dan Identifikasi Rambu-Rambu Lalu Lintas Berbasis Algoritma *You Only Look Once* (YOLO)".

Pembuatan Tugas Akhir ini disusun untuk memenuhi syarat mendapatkan gelar Sarjana Teknik pada Jurusan Teknik Elektro Fakultas Teknik Universitas Muhammadiyah Jakarta. Dalam kesempatan ini mengucapkan terima kasih kepada:

1. Dr. Ma'mun Murod, S. Sos., M.Si. selaku Rektor Universitas Muhammadiyah Jakarta.
2. Irfan Purnawan, S.T., M.Chem.Eng. selaku Dekan Fakultas Teknik Universitas Muhammadiyah Jakarta.
3. Ir. Husnibes Muchtar, M.T. selaku Kepala Program Studi Teknik Elektro Fakultas Teknik Universitas Muhammadiyah Jakarta sekaligus Pembimbing Utama Tugas Akhir ini yang telah memberikan bimbingan, ide, dan ilmu selama penelitian.
4. Haris Isyanto, S.T., M.T. dan Riza Samsinar, S.T., M.Kom. selaku Dewan Penguji Sidang Tugas Akhir ini yang telah memberikan saran dan masukan.
5. Riza Samsinar, S.T., M.Kom. selaku Koordinator Tugas Akhir sekaligus Pembimbing Akademik yang telah memberikan arahan serta bimbingan selama masa perkuliahan.
6. Segenap Dosen Program Studi Teknik Elektro Fakultas Teknik Universitas Muhammadiyah Jakarta yang selalu memberikan ilmu pengetahuan selama masa perkuliahan.

7. Kedua orang tua yaitu Bapak Budiyono dan Ibu Nurie Turinah Astuti untuk segala do'a, bimbingan dan motivasi yang selalu diberikan tanpa henti.
8. Saudara dan kerabat, serta keluarga yang selalu memberikan semangat, motivasi, dan dukungan.
9. Dan pihak-pihak yang telah membantu dalam penyusunan tugas akhir ini, yang tidak dapat disebutkan satu persatu.

Penelitian ini masih jauh dari kata sempurna, oleh karena itu diharapkan kritik dan saran yang membangun dari segenap pihak. Semoga penelitian ini dapat menambah dan memperkaya lembar khazanah pengetahuan bagi para pembaca sekalian. Sebelum dan sesudahnya diucapkan terima kasih.

Wassalamu'alaikum Warahmatullahi Wabarakatu

Jakarta, 04 Agustus 2022

Barokah Asmarahman Takarob

DAFTAR ISI

LEMBAR PERSETUJUAN i
LEMBAR PENGESAHAN ii
PERNYATAAN KEASLIAN KARYA TULIS iii
UCAPAN TERIMA KASIH iv
ABSTRAK v
ABSTRACT vi
KATA PENGANTAR vii
DAFTAR ISI ix
DAFTAR TABEL xi
DAFTAR GAMBAR xii
DAFTAR LAMPIRAN xv
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Batasan Masalah 2
1.4 Tujuan Penelitian 3
1.5 Kerangka Pemikiran 3
1.6 Sistematika Penulisan 4
BAB II TINJAUAN PUSTAKA 5
2.1 Landasan Teori 5
2.1.1 Lalu Lintas 5
2.1.2 Deteksi Objek 5
2.1.3 Kecerdasan Buatan *(Artificial Intelligence)* 6
2.1.4 *Deep Learning* 7
2.1.5 *You Only Look Once* (YOLO) 7
2.1.6 OpenCV 10
2.1.7 *Confusion Matrix* 11
2.1.8 *Recall* 11
2.1.9 *Precision* 12

2.1.10 *F1-Score* 13
2.1.11 *Accuracy* 13
2.1.12 *Mean Average Precision* (mAP) 13
2.1.13 *Intersection over Union* (IoU) 13
2.1.14 Python 14
2.1.15 *LabelImg* 14
2.1.16 Darknet 15
2.1.17 *Google Colab* 15
2.2 Penelitian Terkait 16
BAB III METODOLOGI PENELITIAN 17
3.1 Langkah Penelitian 17
3.2 Uraian Penelitian 17
3.2.1 Studi Literatur 17
3.2.2 Perencanaan 17
3.2.3 Pengumpulan Data 18
3.2.4 Implementasi 18
3.2.5 Pengujian 18
3.3 Flowchart 19
3.4 *Activity Plan* 20
BAB IV HASIL DAN PEMBAHASAN 21
4.1 Implementasi 21
4.1.1 Anotasi Objek 21
4.1.2 Instalasi 22
4.1.3 Training Data 31
4.2 Pengujian 42
4.2.1 Pengujian Dan Hasil Deteksi 42
BAB V KESIMPULAN DAN SARAN 46
5.1 Kesimpulan 46
5.2 Saran 46
DAFTAR PUSTAKA
LAMPIRAN

DAFTAR TABEL

Tabel 2.1 Penelitian Terkait 16
Tabel 3.1 Spesifikasi *Software* dan *Hardware* 18
Tabel 3.2 *Activity Plan* 20
Tabel 4.1 Konfigurasi *Darknet* 25
Tabel 4.2 Konfigurasi pada *weights* YOLO v4 26
Tabel 4.3 Data hasil *training* 33
Tabel 4.4 Nilai *mAP* berdasarkan jumlah iterasi 39

DAFTAR GAMBAR

Gambar 2.1 Langkah Pendeteksian Objek 6
Gambar 2.2 Arsitektur YOLO v1 8
Gambar 2.3 Arsitektur YOLO v3 8
Gambar 2.4 Sistem Deteksi YOLO 9
Gambar 2.5 Arsitektur YOLO v4 9
Gambar 2.6 *Recall* 12
Gambar 2.7 *Precision* 12
Gambar 2.8 *Intersection over Union* (IoU) 14
Gambar 2.9 *LabelImg* 15
Gambar 3.1 Diagram Blok Langkah Penelitian 17
Gambar 3.2 *Flowchart* Sistem 19
Gambar 4.1 Anotasi label dengan *LabelImg* 21
Gambar 4.2 Hasil anotasi objek dengan *LabelImg* 22
Gambar 4.3 Folder *obj* yang sudah dijadikan zip 22
Gambar 4.4 File *obj.data* 23
Gambar 4.5 Isi file *obj.data* 23
Gambar 4.6 File *obj.names* 24
Gambar 4.7 Isi file *obj.names* 24
Gambar 4.8 Program pada file *process.py* 24
Gambar 4.9 Perubahan nilai *width, height, max_batches, dan steps* 26
Gambar 4.10 Perubahan parameter *filters* dan *classes* 26
Gambar 4.11 *Google Colab* 27
Gambar 4.12 Pengaturan untuk mengaktifkan virtual GPU 27
Gambar 4.13 Program untuk *mount* file dari *Google Drive* 27
Gambar 4.14 Output program hasil *mount* file dari *Google Drive* 28
Gambar 4.15 Program untuk mengunduh *framework darknet* 28
Gambar 4.16 Program untuk mengaktivasi OpenCV dan GPU 28
Gambar 4.17 *Output* program hasil aktivasi *OpenCV* dan *GPU* 28
Gambar 4.18 Program untuk menyalin file *yolov4-custom.cfg* ke folder *cfg* 29

Gambar 4.19 *Output* program file *yolov4-custom.cfg* yang telah disalin 29
Gambar 4.20 Program untuk proses ekstraksi file *obj.zip* 29
Gambar 4.21 Hasil file *obj.zip* yang diekstraksi 29
Gambar 4.22 Program untuk memindahkan file *yolov4-custom.cfg* ke folder *cfg* 30
Gambar 4.23 Hasil pemindahan file *yolov4-custom.cfg* ke folder *cfg* 30
Gambar 4.24 Program untuk menyalin file *obj.names* dan *obj.data* 30
Gambar 4.25 *Output* hasil file *obj.names* dan *obj.data* yang di salin 30
Gambar 4.26 Program untuk menyalin *script process.py* 30
Gambar 4.27 *Output* program hasil file *script process.py* yang disalin..... 30
Gambar 4.28 Program untuk mengunduh file *weght* model YOLO v4 31
Gambar 4.29 *Output* hasil *download* file *weights* 31
Gambar 4.30 Program untuk training dataset dengan *yolov4-custom.cfg*..... 31
Gambar 4.31 *Output* hasil *training* dengan *yolov4-custom.cfg* 32
Gambar 4.32 File *yolov4-custom_last.weights* 32
Gambar 4.33 Program untuk melanjutkan *training* yang berhenti 32
Gambar 4.34 Hasil akhir proses *training* 33
Gambar 4.35 Grafik *mean Average Precision* dan *loss* hasil *training*..... 35
Gambar 4.36 Program untuk cek *mAP* pada iterasi 1000 35
Gambar 4.37 *Output* hasil cek *mAP* pada iterasi 1000..... 36
Gambar 4.38 Program untuk cek *mAP* pada iterasi 2000 36
Gambar 4.39 *Output* hasil cek *mAP* pada iterasi 2000..... 36
Gambar 4.40 Program untuk cek *mAP* pada iterasi 3000 36
Gambar 4.41 *Output* hasil cek *mAP* pada iterasi 3000..... 37
Gambar 4.42 Program untuk cek *mAP* pada iterasi 4000 37
Gambar 4.43 *Output* hasil cek *mAP* pada iterasi 4000..... 37
Gambar 4.44 Program untuk cek *mAP* pada iterasi 5000 37
Gambar 4.45 *Output* hasil cek *mAP* pada iterasi 5000..... 38
Gambar 4.46 Program untuk cek *mAP* pada iterasi 6000 38
Gambar 4.47 *Output* hasil cek *mAP* pada iterasi 6000..... 38
Gambar 4.48 Grafik *Precision* 39
Gambar 4.49 Grafik *Recall* 40
Gambar 4.50 Grafik *F1-Score* 40

Gambar 4.51 Grafik *Intersection over Union* 41
Gambar 4.52 Grafik *mean Average Precision* 41
Gambar 4.53 Grafik *Accuracy* 42
Gambar 4.54 *Output* pengujian dengan gambar 43
Gambar 4.55 *Output* pengujian dengan video..... 43
Gambar 4.56 *Output* deteksi dengan *video* (a)..... 44
Gambar 4.57 *Output* deteksi dengan *video* (b)..... 44
Gambar 4.58 *Output* deteksi dengan *video* (c)..... 45

DAFTAR LAMPIRAN

Lampiran 1 : *Script process.py*

Lampiran 2 : *Script yolov4-custom.cfg*

BAB I

PENDAHULUAN

1.1 Latar Belakang

Salah satu masalah yang sulit dan menantang dari manajemen perkotaan khususnya di negara berkembang adalah masalah kecelakaan lalu lintas. Tingginya angka kecelakaan lalu lintas seringkali menarik perhatian masyarakat karena masalah ini berdampak menimbulkan kerugian finansial bahkan merenggut korban jiwa. Menurut POLRI, pada tahun 2019 terjadi 107.500 kasus kecelakaan lalu lintas. Jumlah ini meningkat tiga kali lipat dibandingkan tahun 2018 yaitu sebesar 103.672 kasus [1].

Menurut Dirkomsel Korlantas POLRI (2018), kecelakaan lalu lintas yang disebabkan oleh pelanggaran lalu lintas seperti melanggar rambu lalu lintas dan tidak memakai helm menjadi penyumbang sebagian besar kematian akibat kecelakaan lalu lintas yang tercatat di Ditlantas Polda Metro Jaya [2]. Bahkan menurut data WHO, hingga 2,4 juta jiwa meninggal setiap tahun akibat kecelakaan lalu lintas. Kecelakaan lalu lintas ini menjadi penyebab kematian ketiga setelah HIV/AIDS dan TBC [3].

Beberapa penyebab utama kecelakaan adalah faktor kesalahan manusia *(human error)* disebabkan kelalaian pengemudi, keterampilan pengemudi, etika lalu lintas dan komunikasi di jalan. Masalah ini menarik perhatian bagi para ahli untuk mencari jalan keluar dalam meningkatkan keselamatan berlalu lintas . Dalam pengolahan citra digital, penerapan *artificial intelligence* seperti *deep learning* belakangan ini meraih kesuskesan dalam pengolahan citra dan video skala besar sehingga menjadikan *deep learning* populer untuk digunakan dan dirancang [4], salah satunya dalam pengembangan fitur *autonomous vehicle*.

Dengan *autonomous vehicle* ini diharapkan dapat meminimalisir angka kecelakaan lalu lintas yang disebabkan kelalaian manusia. Karena *autonomous vehicle* ini dapat bergerak tanpa kendali dari pengemudi namun perlu dilengkapi dengan berbagai sensor salah satunya menggunakan kamera yang berperan memberikan input visual untuk mendeteksi objek di sekitar *autonomous vehicle*.

Dalam penelitian ini akan direalisasikan simulasi untuk mengidentifikasi objek menggunakan algoritma YOLO v4 *(You Only Look Once v4)* yang menjadi pengembangan dari metode CNN *(Convolutional Neural Network)* dan dapat dijalankan secara *realtime* untuk mendeteksi objek yang akan dilalui oleh *autonomous vehicle*. Metode ini berfungsi untuk mendeteksi objek secara *realtime* dimana hasil keluaran dari deteksi tersebut berupa parameter objek di sekitar *autonomous vehicle*. Hasil deteksi tersebut akan dikirimkan secara serial menuju mikrokontroler yang akan diproses menjadi sinyal kendali.

Penelitian ini diharapkan dapat memberikan kontribusi untuk membangun model *deep learning* dengan menggunakan *dataset* rambu lalu lintas yang berlaku di Indonesia yang selanjutnya dikumpulkan dan dianalisa menggunakan algoritma *You Only Look Once v4*.

1.2 Rumusan Masalah

Adapun perumusan masalah dalam penelitian ini berdasarkan latar belakang di atas adalah sebagai berikut:

a. Bagaimana implementasi metode *You Only Look Once v4* (YOLO v4) dalam mengidentifikasi rambu-rambu lalu lintas secara *realtime*?
b. Bagaimana hasil pengujian deteksi dan identifikasi rambu lalu lintas secara *realtime* menggunakan *You Only Look Once v4* (YOLO v4)?

1.3 Batasan Masalah

Dalam penelitian ini terdapat batasan masalah diantaranya:

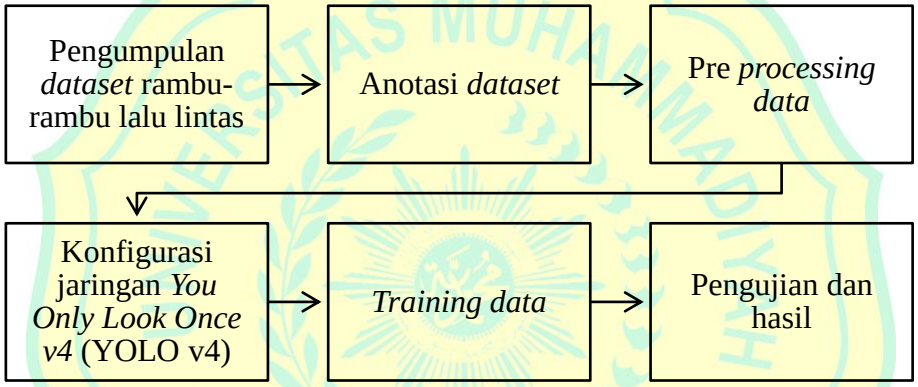
a. Objek yang diidentifikasi serta dataset yang dikumpulkan dan diberikan label berupa beberapa gambar rambu-rambu lalu lintas yang berlaku di Indonesia.
b. Dataset yang menjadi input citra berupa gambar dan video yang diambil dari aplikasi *Google Maps* (*street view)* dan *YouTube*.
c. Proses pengolahan citra dengan menggunakan fitur *Graphics Processing Unit* (GPU) dari *Google Colab*.

1.4 Tujuan Penelitian

Berdasarkan rumusan masalah di atas, maka tujuan dari penelitian ini adalah untuk membangun model *deep learning* menggunakan program *You Only Look Once v4* (YOLO v4) untuk membaca dan mengidentifikasi rambu-rambu lalu lintas yang berlaku di Indonesia. Penelitian ini juga bertujuan untuk menunjukkan kinerja metode YOLO v4 dalam identifikasi objek.

1.5 Kerangka Pemikiran

Berikut adalah kerangka pemikiran yang digunakan dalam penelitian ini:



1.6 Sistematika Penulisan

Berikut adalah sistematika penulisan yang terdapat dalam penelitian ini:

BAB I PENDAHULUAN

Pada bab ini membahas tentang latar belakang penelitian, rumusan masalah, batasan masalah, tujuan penelitian, dan kerangka pemikiran yang akan dilakukan dalam penelitian ini.

BAB II TINJAUAN PUSTAKA

Pada bab ini terdapat teori-teori dan acuan yang digunakan sebagai landasan dalam mendukung penelitian ini.

BAB III METODOLOGI PENELITIAN

Pada bab ini berisi perihal susunan rencana kerja dan tahapan yang dilakukan dalam penelitian ini.

BAB IV HASIL DAN PEMBAHASAN

Dalam bab ini akan menjelaskan hasil dari setiap tahapan dalam penelitian ini seperti pengumpulan dan pengolahan data serta simulasi.

BAB IV PENUTUP

Pada bab ini menjelaskan berupa kesimpulan dan saran dari hasil penelitian sekaligus evaluasi untuk penelitian selanjutnya.

BAB II

TINJAUAN PUSTAKA

2.1 Landasan Teori

2.1.1 Lalu Lintas

Undang-Undang Republik Indonesia Nomor 22 Tahun 2009 tentang Lalu Lintas dan Angkutan Jalan menyatakan lalu lintas adalah sebagai pergerakan kendaraan dan orang dalam ruang lalu lintas jalan, sedangkan ruang lalu lintas jalan adalah sarana dan prasarana yang dibutuhkan untuk pergerakan orang, kendaraan, dan/atau barang berupa jalan serta fasilitas pendukung lainnya [5].

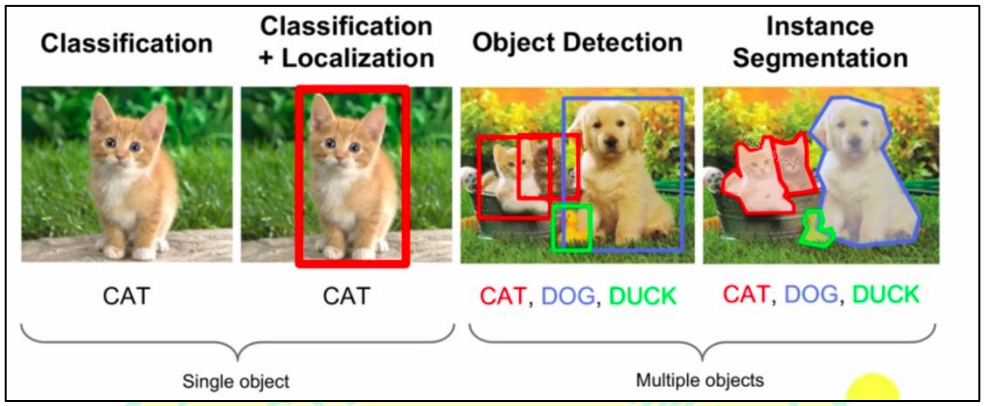
Peraturan Menteri Perhubungan Republik Indonesia Nomor 13 Tahun 2014 BAB I Pasal 1 menjelaskan bahwa rambu lalu lintas adalah bagian dari perlengkapan jalan yang berupa lambang, huruf, angka, kalimat dan/atau kombinasi yang berfungsi sebagai peringatan, larangan, perintah, dan petunjuk bagi pengguna jalan. Rambu lalu lintas berdasarkan jenisnya terdiri dari 4 jenis rambu yaitu [6]:

a. Rambu Peringatan
b. Rambu Larangan
c. Rambu Perintah
d. Rambu Petunjuk

2.1.2 Deteksi Objek

Deteksi objek digunakan untuk mendefinisikan presensi suatu objek, dan parameter serta posisi pada gambar. Pengenalan objek menentukan setiap kelas objek yang ada dalam data *training* yang dimulai dengan pengenalan objek. Terdapat dua jenis deteksi objek yaitu *soft detection* dan *hard dtection*. *Soft detection* hanya mendefinisikan presensi suatu objek, sedangkan *hard detection* mampu mendefinisikan presensi objek serta posisinya dalam citra. Deteksi objek menelusuri seluruh elemen dari gambar untuk mengidentifikasi sisi yang memiliki geometri sesuai dengan objek yang diinginkan dalam *database*.

Deteksi ini terjadi apabila terdapat korelasi yang cukup tinggi antara *templat*e dengan wilayah yang diukur pada gambar dengan cara memindai seluruh posisi, skala, dan rotasi pada *templat*e disetiap gambar. Baru-baru ini, pengenalan objek berbasis citra telah terbukti akurat dan presisi terhadap data *training* [7].



Gambar 2.1 Langkah Pendeteksian Objek
(Sumber: Jalled, 2016 [7])

Proses pendeteksian objek dengan gambar ataupun video memiliki kesamaan dimana sebuah video terdiri dari beberapa susunan gambar atau *frame*. Proses identifikasi objek dalam video dilakukan menggunakan *Graphics Processing Unit* (GPU) dengan membagi video menjadi beberapa *frame* kemudian diproses dan dirangkai kembali menjadi video yang utuh [8].

2.1.3 Kecerdasan Buatan *(Artificial Intelligence)*

Sistem cerdas adalah mesin yang dapat berpikir secerdas manusia. Sistem ini merupakan sistem dalam bidang ilmu komputer yang bertujuan untuk menciptakan mesin yang meniru kerja dan kecerdasan manusia. Sistem cerdas dilengkapi sistem deteksi untuk mengidentifikasi pengguna komputer, *monitoring* dan memantau aktivitas atau objek tertentu. Pengembangan sistem cerdas dituntut oleh para peneliti atau perusahaan manufaktur karena dapat mempermudah pekerjaan manusia.

Ada dua jenis teknik sistem cerdas yaitu *unsupervised learning* dan *supervised learning*. *Unsupervised learning* adalah metode yang mampu menangkap karakteristik masukan untuk proses pembelajaran, generalisasi dan pemahaman tanpa bimbingan ahli atau informasi yang relevan. Contoh metode yang termasuk dalam kategori ini adalah K-Measure. Sementara *supervised learning* adalah teknik yang membutuhkan kemampuan seorang *data scientist* atau *programmer* untuk mengekstrak fitur penting dari gambar.

2.1.4 *Deep Learning*

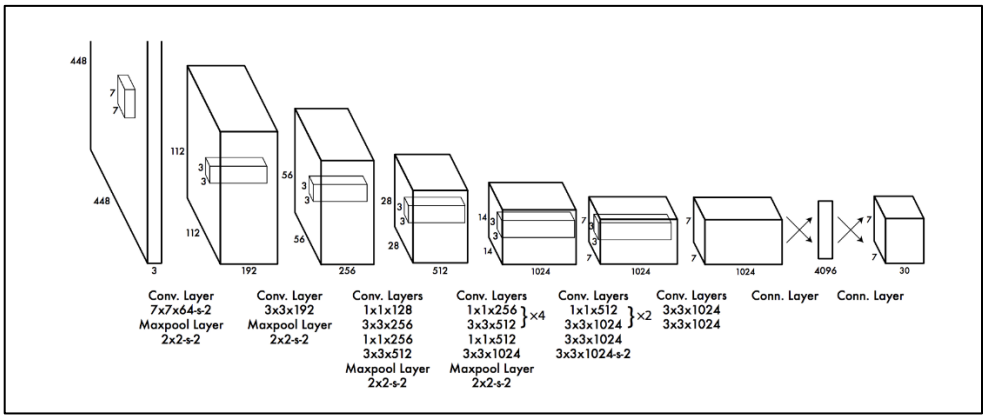
Deep Learning merupakan konsep *machine learning* yang menggunakan beberapa lapisan informasi non linear untuk mengidentifikasi fitur, pola, dan klasifikasi [4]. *Deep learning* menurut Goodfellow menjadi sebuah metode yang memanfaatkan konsep hierarki dalam memecahkan masalah pada sistem *machine learning*. Rancangan ini dilakukan dengan menggabungkan konsep yang lebih sederhana untuk mempelajari konsep yang kompleks [9].

Keutamaan dari *deep learning* adalah memiliki serangkaian transformasi (*hidden layers*) yang mampu mengubah data *non linearly separable* menjadi data *linearly separable* dengan menggunakan *back propagation* pada tahap proses *training*. *Deep learning* juga mampu mensimulasikan interaksi non linier antar fitur dan mencari *decision boundary* yang berbentuk non linier [1].

2.1.5 *You Only Look Once* (YOLO)

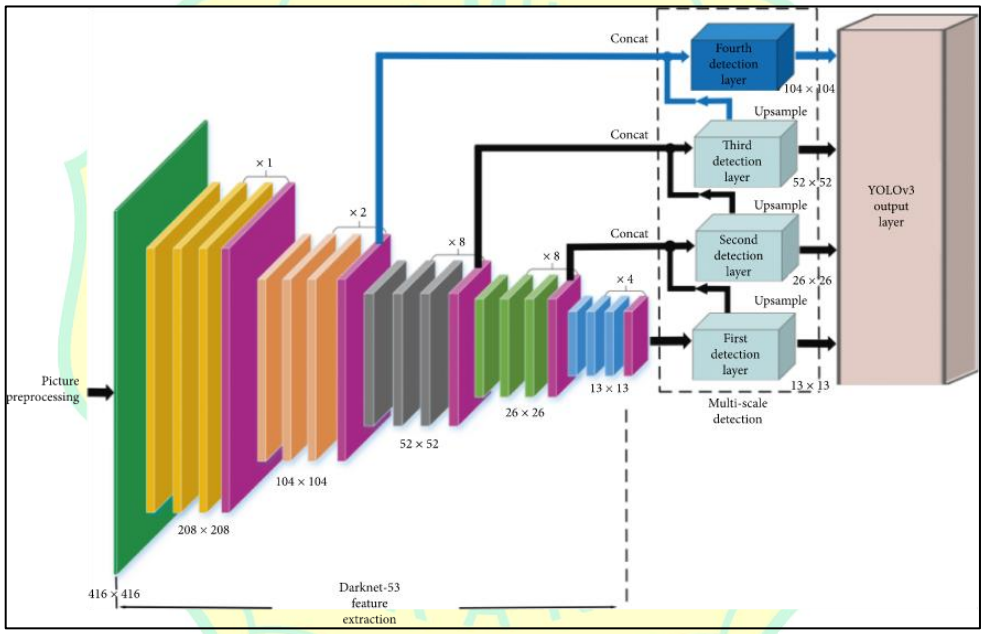
YOLO *(You Only Look Once)* mampu mendeteksi objek secara *real time* karena merupakan algoritma dengan arsitektur CNN *(Convolutional Neural Network)*. Joseph Redmon dan Ali Farhadi mengembangkan algoritma YOLO pada tahun 2015 karena terinspirasi dari model GoogleNet dalam sistem klasifikasi citra [10].

Arsitektur YOLO v1 terdiri dari 24 *convolutional layer* dengan 2 *fully connected layer* [11]. Cara kerja YOLO adalah membagi gambar menjadi *grid* S x S, dan ketika pusat objek sesuai dengan *grid cell* maka *grid cell* tersebut akan mengidentifikasi objek. Gambar berikut merupakan arsitektur dasar YOLO v1.



Gambar 2.2 Arsitektur YOLO v1

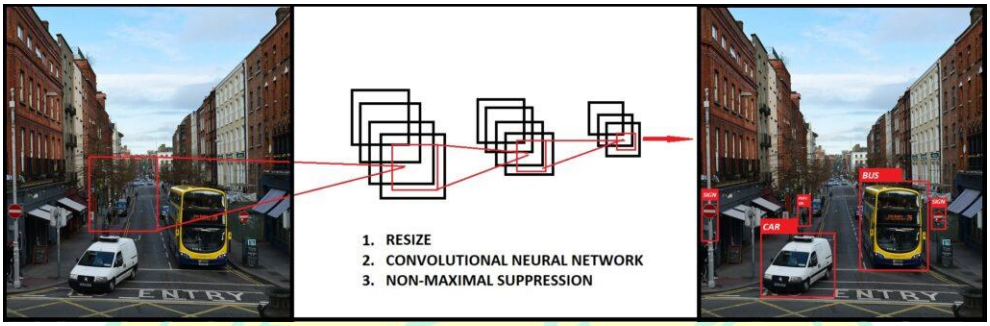
(Sumber: Redmon, 2016 [10])



Gambar 2.3 Arsitektur YOLO v3

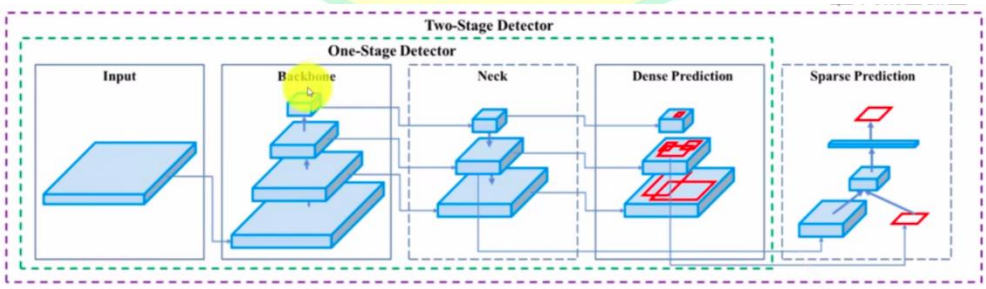
(Sumber: Baojun Zhang, 2021 [12])

Penelitian ini menggunakan YOLO v4 yang dirilis oleh 3 *developer* baru yaitu Alexey Bochkovsky, Chien-Yao Wang, dan Hong-Yuan Mark Liao pada 24 April 2020. YOLO v4 memiliki keunggulan dalam kecepatan dan kinerja yang jauh lebih cepat. Pengembangan YOLO versi 4 dihentikan oleh Joseph Redmon dan Ali Farhadi karena kekhawatiran terhadap masalah privasi dan penggunaan militer sehingga pengembangan selanjutnya dilanjutkan oleh 3 *developer* yang disebutkan di atas.



Gambar 2.4 Sistem Deteksi YOLO
(Sumber: Redmon, 2016 [10])

Arsitektur jaringan YOLO v4 berbeda dari versi sebelumnya dimana pada YOLO v4 terdiri dari 3 elemen yaitu *Backbone*: CSPDarknet53, *Neck*: SPP dan *Head*: YOLO v3 [13]. YOLO v4 memiliki keunggulan kecepatan karena merupakan bagian dari *one stage detector*, sebab pada algoritma *two stage detector* sebelum objek terdeteksi terdapat tahap tambahan yaitu *sparse prediction* sehingga membutuhkan waktu tambahan dalam mendeteksi objek [14].



Gambar 2.5 Arsitektur YOLO v4
(Sumber: Bochkovskiy, 2020 [14])

Arsitektur YOLO terdiri 2 bagian utama yaitu *Backbone* yang berfungsi meningkatkan akurasi sebelum dilakukan pendeteksian dan *Head* yang menjadi bagian utama dalam melakukan pendeteksian. Pendeteksian diawali dengan memposisikan gambar atau video di area *input* kemudian diarahkan ke bagian *Backbone*. Sedangkan *Neck* menggabungkan *feature maps* dari *stage* yang lain setelah *Backbone*.

Cara kerja YOLO v4 adalah dengan menangkap *input* berupa gambar kemudian di-*resize* menjadi ukuran 416 x 416, kemudian hasil *resize* tersebut dianotasikan sebagai matriks untuk proses pada *convolutional layer* dan *pooling layer*. *Convolutional layer* disini menggambarkan *layer* dalam susunan matriks yang berisi angka antara 0.00 sampai 1.00 yang akan dilakukan rekapitulasi matriks dengan filter 3 x 3 untuk menciptakan nilai-nilai yang baru.

Proses ini berlangsung hingga 160 kali dan setiap 7 layer akan dilakukan *down sampling* dimana skala matriks akan di*resize* kembali hingga matriks mengecil menjadi skala 13 x 13. Matriks 13 x 13 tersebut kemudian akan dibagi menjadi 169 *grid cell*. *Cell* yang telah terbagi akan dideteksi untuk menghasilkan *bounding box* dan prediksi kelas. Hasil *bounding box* tersebut akan dikualifikasi dengan *threshold* yang ditentukan.

Output deteksi dengan YOLO akan menghasilkan *bounding box* yang berisi identitas nama kelas dan nilai *confidence*. *Bounding box* berisi variabel *x, y, w* dan *h*, serta *confidence.* Variabel *x* dan *y* mendefinisikan koordinat pusat dari *bounding box*, variabel *w* dan *h* sebagai parameter piksel dari suatu gambar, sedangkan *confidence* menjadi nilai kelas gambar yang terdeteksi antara *predicted box* dengan *ground truth* [10].

2.1.6 OpenCV

OpenCV (*Open Source Computer Vision*) merupakan salah satu *library* perangkat lunak *computer vision* dan *machine learning* yang bersifat *open source.* OpenCV dibentuk sebagai infrastruktur yang umum untuk aplikasi *computer vision* dalam meningkatkan persepsi mesin.

OpenCV berlisensi BSD (*Berkeley Software Distribution*) sehingga memudahkan bagi pelaku bisnis untuk memanfaatkan dan mengubah kode. OpenCV mempunyai lebih dari 2500 algoritma optimal yang terdiri dari sekumpulan algoritma *computer vision* dan *machine learning* bertipe klasik dan modern. Algoritma ini dapat digunakan untuk mendeteksi wajah, mengidentifikasi objek, mengklasifikasikan tindakan manusia dalam video, melacak gerakan kamera, melacak objek yang bergerak, dan lain-lain [16].

2.1.7 *Confusion Matrix*

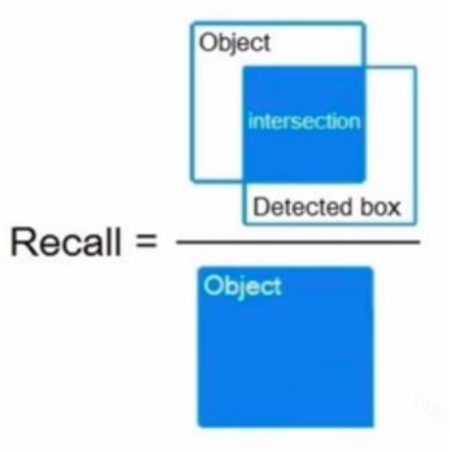
Confusion matrix merupakan rasio hasil prediksi dalam masalah klasifikasi objek. Jumlah penilaian benar atau salah akan dihitung kemudian dibagi dengan setiap kelas. *Confusion matrix* memberikan informasi tidak hanya tentang kesalahan pengklasifikasi, tetapi juga tentang jenis kesalahan [1].
Ketentuan dalam *Confusion matrix* terdiri dari:

a. (P) *Positive* : aktual positif
b. (N) *Negative* : aktual negatif
c. (TP) *True Positive* : aktual positif, prediksi positif
d. (TN) *True Negative* : aktual negatif, prediksi negatif
e. (FP) *False Positive* : aktual negatif, prediksi positif
f. (FN) *False Negative* : aktual positif, prediksi negatif

2.1.8 *Recall*

Recall merupakan rasio komparasi antara total prediksi positif yang dikategorikan benar dibagi dengan total prediksi positif dan prediksi negatif [1].

$$Recall = \frac{TP}{TP + FN} \tag{2.1}$$



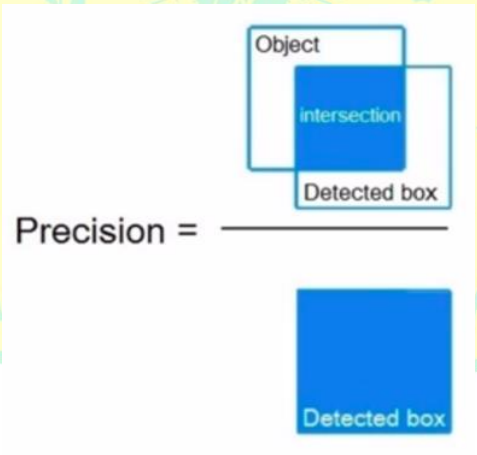
Gambar 2.6 *Recall*

(Sumber: Umar, 2020 [1])

2.1.9 *Precision*

Nilai presisi adalah rasio komparasi antara hasil total prediksi positif yang dikelompokan benar dibagi dengan total prediksi positif [1].

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$



Gambar 2.7 *Precision*

(Sumber: Umar, 2020 [1])

2.1.10 *F1-Score*

F1-Score digunakan untuk menghitung gabungan dari *precision* dan *recall* [1].

$$F1 - Score = \frac{2 \times Recall \times Precision}{Recall + Precision} \quad (2.3)$$

2.1.11 *Accuracy*

Accuracy adalah rasio komparasi antara nilai prediksi dengan nilai aktual.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.4)$$

2.1.12 *Mean Average Precision* (mAP)

Nilai *mAP* adalah matriks yang mengevaluasi hasil pada model deteksi objek dan merupakan nilai rata-rata dari *average precision* (AP) untuk mengukur seberapa presisi performansi dari *weights file* hasil *training data*.

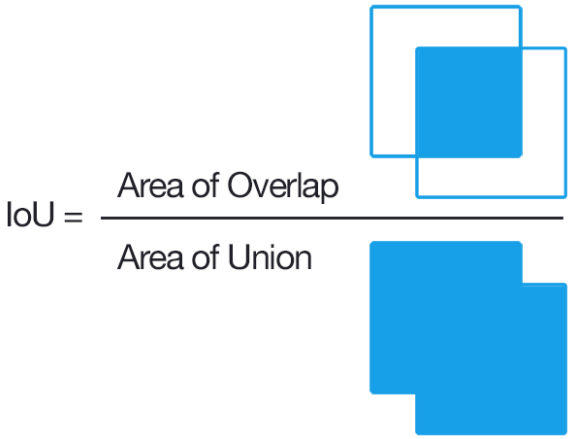
$$AP = \sum_{k=0}^{k=n-1} [Recall\ (k) - Recall\ (k+1)] \times Precision\ (k) \quad (2.5)$$

2.1.13 *Intersection over Union* (IoU)

IoU mendefinisikan rasio komparasi antara hasil total prediksi positif dengan total prediksi *bounding box* yang mengevaluasi keakuratan detektor objek [1] dengan syarat berikut.

a. Terdapat nilai *True Positive* pada hasil *training*
b. Terdapat prediksi *bounding box* pada dataset objek

$$IoU = \frac{TP}{TP + FP + FN} \quad (2.6)$$



Gambar 2.8 *Intersection over Union* (IoU)

(Sumber: Umar, 2020 [1])

2.1.14 Python

Awalnya bahasa pemrograman ini merupakan bahasa program tingkat tinggi pada sistem operasi terdistribusi Amoeba yang dibuat oleh Guido van Rossum dari Amsterdam, Belanda. Saat ini bahasa pemrograman Python umum digunakan oleh kalangan *engineer* seluruh dunia dalam pembuatan perangkat lunak, bahkan bahasa program ini digunakan oleh perusahaan pembuat perangkat lunak komersial seperti *Google, NASA, Instagram, Youtube*, dan lainnya. Python sebagai bahasa pemrograman interpretatif banyak digunakan untuk membuat berbagai macam program, seperti *data science, machine learning, IoT*, dan sebagainya.

2.1.15 *LabelImg*

LabelImg merupakan aplikasi untuk pembuatan label pada dataset gambar berupa *bounding box* yang nantinya akan digunakan untuk input citra dalam *training* dan *testing* dataset. *Biunding box* tersebut disesuaikan dengan area pada objek yang akan dideteksi.

Gambar 2.9 *LabelImg*

2.1.16 Darknet

Darknet merupakan sebuah *framework neural network* yang bersifat *open source* dan disusun dengan bahasa C, CUDA *(Compute Unified Device Architecture)*, dan CUDNN *(CUDA Deep Neural Network)* untuk mendukung dan mengoptimalkan *deep neural network* dalam komputasi CPU dan GPU saat memproses dataset pada arsitektur jaringan YOLO v4. Darknet pada prinsipnya sama dengan *OpenCV*, baik *OpenCV* dan *Darknet* bisa dikombinasikan bersamaan untuk pemrosesan dataset saat *training* dan *testing*.

2.1.17 *Google Colab*

Google Colab merupakan *platform* dari *Google* yang menyediakan fitur pemrograman atau *coding data*. Aplikasi ini *compatible* terhadap berbagai macam bahasa program sesuai dengan kebutuhan pengguna. Di dalamnya juga terdapat sebuah fitur untuk memperoleh GPU yang secara khusus diperlukan dalam pendeteksian objek citra. Fitur tersebut bisa digunakan secara gratis namun dengan batasan tertentu.

2.2 Penelitian Terkait

Berikut adalah rangkuman dari beberapa penelitian yang pernah dilakukan sebelumnya:

Tabel 2.1 Penelitian Terkait

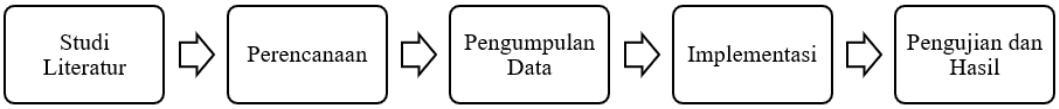
No	Penelitian	Metode	Uraian Singkat
1	Deteksi Kendaraan Secara *Real Time* Menggunakan Metode YOLO Berbasis Android (Hutauruk et al., 2020)	YOLO	Penelitian ini mendeteksi kendaraan yang memasuki jalur tidak untuk kendaraan tersebut dengan 200 dataset, 4 kelas, 10 *batch*, dan 200 *epoch*. *Bounding box* mampu mendeteksi dan objek secara tepat.
2	Sistem Pengenalan Plat Nomor Otomatis Menggunakan YOLO v3 (Syuhaila, 2020)	YOLO	Penelitian ini mengenali plat kendaraan secara otomatis dengan algoritma YOLO menggunakan *framework* Keras. Hasil OCR dengan *software* Tesseract menunjukkan bahwa sistem telah berhasil membaca semua karakter yang dikenali pada plat nomor dengan format alfanumerik 6-7 karakter.
3	Deteksi Jenis Mobil Menggunakan Metode YOLO Dan *Faster* RCNN (Shianto et al., 2019)	YOLO dan Faster R-CNN	Penelitian ini menggunakan metode Faster R-CNN dan YOLO. Tujuan penggunaan kedua metode dalam arsitektur ini adalah untuk meningkatkan akurasi.

Berdasarkan beberapa penelitian sebelumnya yang menggunakan beberapa metode dan kumpulan data, terlihat jelas bahwa penelitian pendeteksian dengan algoritma *You Only Look Once* (YOLO) mampu memberikan tingkat ketelitian yang tinggi. Oleh sebab itu pada riset ini akan diaplikasikan simulasi sistem deteksi dan identifikasi rambu lalu lintas dengan menggunakan algoritma *You Only Look Once v4* (YOLO v4).

BAB III

METODOLOGI PENELITIAN

3.1 Langkah Penelitian



Gambar 3.1 Diagram Blok Langkah Penelitian

3.2 Uraian Penelitian

3.2.1 Studi Literatur

Studi literatur dilakukan dengan melakukan pencarian literatur terkait deteksi dan klasifikasi objek memakai algoritma YOLO dengan *input* gambar atau video. Studi literatur dilakukan untuk menemukan masalah dan informasi yang relevan untuk referensi dalam melakukan penelitian. Ada beberapa kriteria untuk memilih literatur untuk penelitian ini diantaranya:

a. Publikasi ilmiah berbahasa Indonesia dan bahasa Inggris.
b. Membahas tentang deteksi dan klasifikasi objek.
c. Menggunakan kaidah berdasarkan gambar dan video.

3.2.2 Perencanaan

Langkah berikutnya adalah merencanakan pelaksanaan kegiatan penelitian ilmiah. Perencanaan meliputi literatur yang akan dijadikan acuan untuk penelitian. Pilihan *software* dan *hardware* yang akan digunakan dapat dilihat pada Tabel 3.1.

Tabel 3.1 Spesifikasi *Software* dan *Hardware*

Software	OS Windows 10 64 bit
	Python
	OpenCV
	Google Drive
	Google Colab
	Darknet
Hardware	RAM 8 GB
	Intel® Core™ i5-8365U

3.2.3 Pengumpulan Data

Dalam eksperimen ini, dataset yang dipersiapkan berbentuk foto rambu lalu lintas yang diperoleh melalui *Google Maps* dan video dari *YouTube*. Dataset yang telah dikumpulkan akan diproses *training* dan *validation* atau *testing* dengan rasio pembagian sebesar 90 : 10.

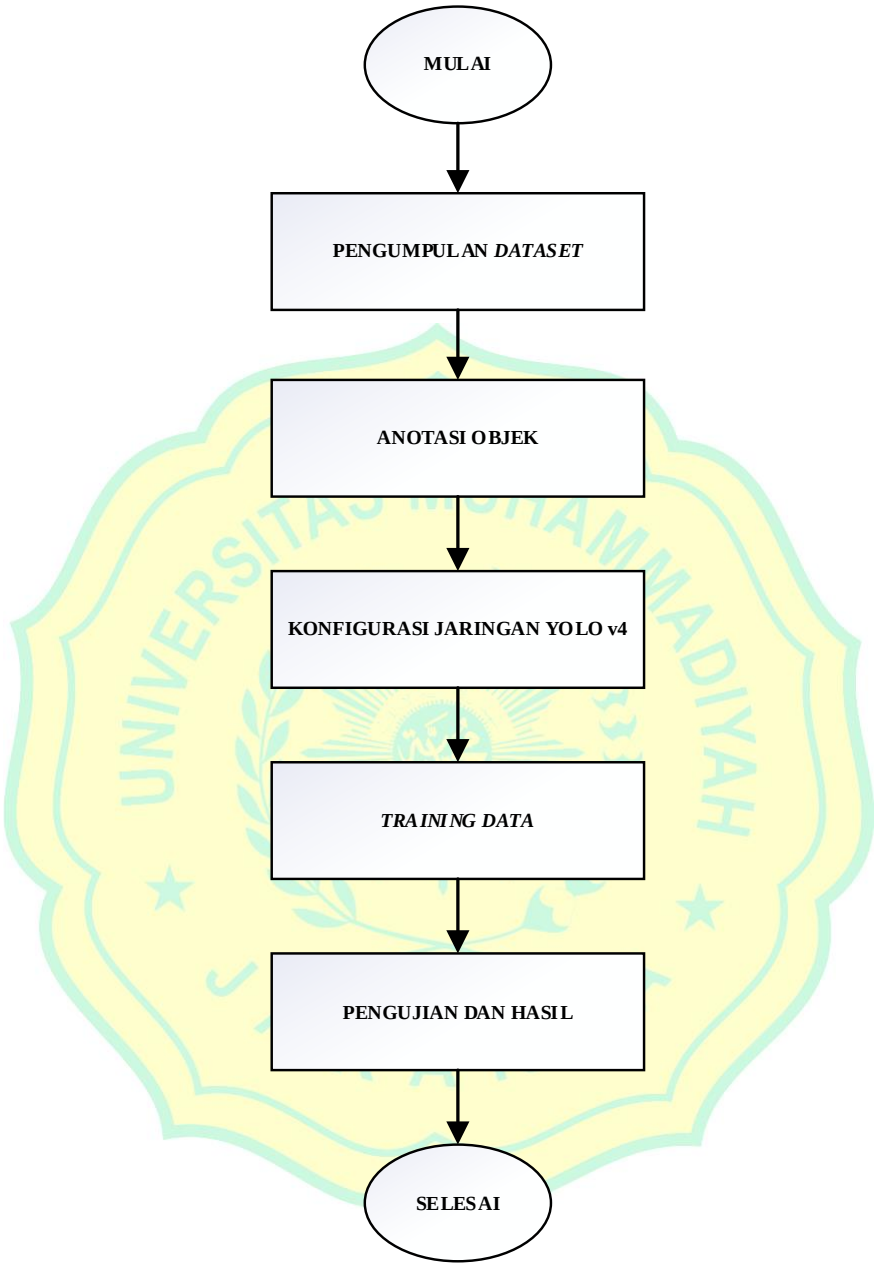
3.2.4 Implementasi

Implementasi dimulai dengan anotasi dataset yang dilakukan untuk menetapkan *marking* pada gambar dengan menentukan *bounding box* disertai nama kelas pada seluruh objek. Berikutnya dilakukan proses *training* untuk mengarahkan komputer dengan mengolah data supaya tersusun suatu logika sebagai evaluasi dalam mencapai sebuah prediksi. Setelah itu adalah mengaplikasikan algoritma *You Only Look Once v4* (YOLO v4) dalam proses deteksi dan klasifikasi objek.

3.2.5 Pengujian

Tahap ini dilakukan untuk menunjukkan hasil deteksi dan identifikasi rambu-rambu berdasarkan implementasi sebelumnya.

3.3 Flowchart



Gambar 3.2 *Flowchart* Sistem

3.4 *Activity Plan*

Tabel 3.2 *Activity Plan*

Kegiatan	Minggu									
	1	2	3	4	5	6	7	8	9	10
Persiapan Penelitian										
Studi Literatur										
Perencanaan										
Pengumpulan Data										
Implementasi										
Pengujian dan Hasil										
Penyusunan Laporan										

BAB IV

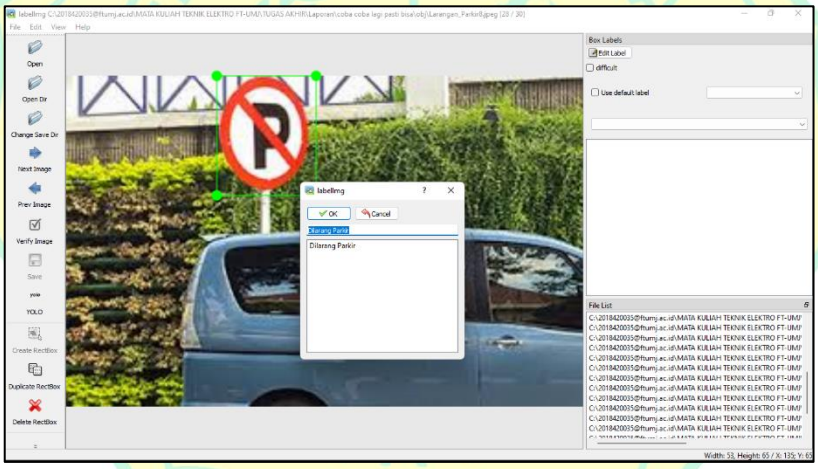
HASIL DAN PEMBAHASAN

4.1 Implementasi

4.1.1 Anotasi Objek

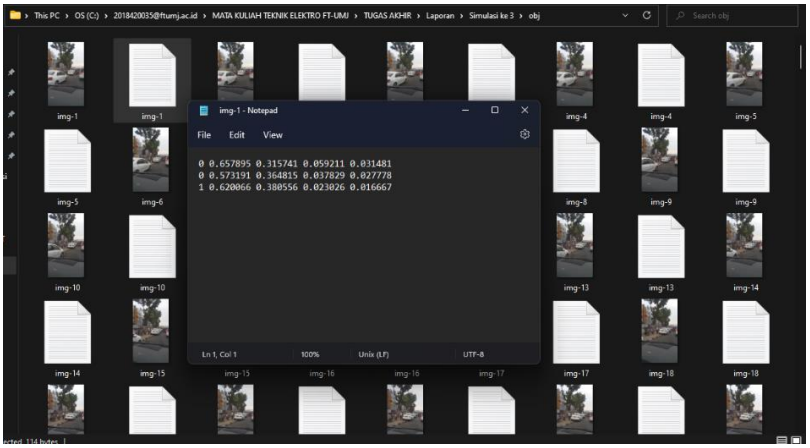
Anotasi objek merupakan pemberian *marking* yang dilakukan dengan menentukan *bounding box* yang berisi nama kelas sesuai masing-masing objek tersebut seperti pada gambar 4.1. Anotasi objek dibagi menjadi 3 kelas rambu, yaitu:

a. Dilarang Parkir
b. Dilarang Berhenti, dan
c. Dilarang Masuk



Gambar 4.1 Anotasi label dengan *LabelImg*

Anotasi objek dilakukan menggunakan *LabelImg* dengan t*ype* YOLO. Hasil anotasi tersebut menghasilkan file data berisi informasi *bounding box* dalam format *.txt* Dalam *.txt* file tersebut terdiri baris [*class*] [*x*] [*y*] [*w*] [*h*]. Pada [*class*] mendefinisikan kelas objek, [*x*] dan [*y*] menjadi koordinat pusat dari *bounding box*, sedangkan *width* [*w*] dan *height* [*h*] adalah *float value* pada dimensi gambar. Hasil dari *.txt* file dapat dilihat pada Gambar 4.2.

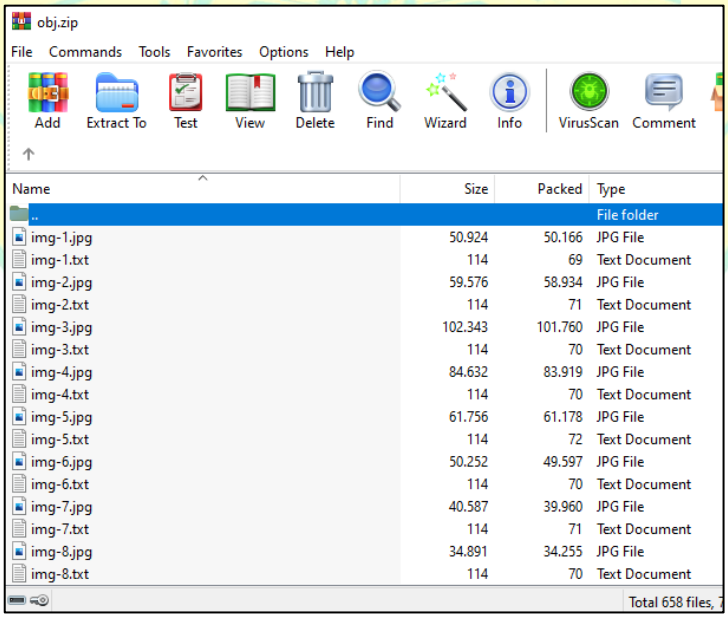


Gambar 4.2 Hasil anotasi objek dengan *LabelImg*

4.1.2 Instalasi

a. Membuat file *obj.zip*

Dataset gambar yang sudah diberi label dengan *LabelImg* selanjutnya dikumpulkan dalam satu folder kemudian dijadikan zip seperti pada Gambar 4.3 berikut.

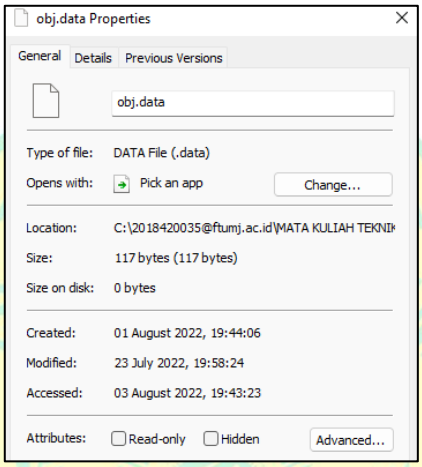


Gambar 4.3 Folder *obj* yang sudah dijadikan zip

b. Membuat file *obj.data*

File *obj.data* ini sebagai acuan dalam proses *training*, *testing*, dan penyimpanan hasil *training data.*

File ini berisi <*classes*> yang menunjukkan jumlah kelas yaitu sebanyak 3, <*train*> yang artinya data untuk *training* dengan format *.txt*, <*valid*> yang merupakan data untuk *testing* dengan format *.txt*, <*names*> yang menunjukkan jenis kelas, dan <*backup*> yang artinya lokasi file untuk penyimpanan hasil *training* dengan beberapa iterasi dalam format *.weights*. File dan isi *obj.data* ditunjukan seperti pada Gambar 4.4 dan 4.5.



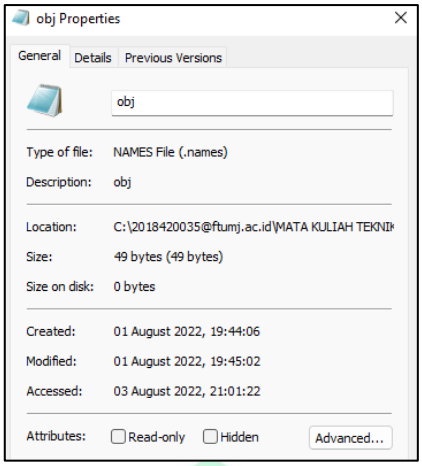
Gambar 4.4 File *obj.data*

```
classes = 3
train   = data/train.txt
valid   = data/test.txt
names   = data/obj.names
backup  = /mydrive/TugasAkhirYolov4/training
```

Gambar 4.5 Isi file *obj.data*

c. Membuat file *obj.names*

File *obj.names* berisi data nama jenis-jenis kelas pada objek yang terdiri dari rambu “Dilarang Parkir”, “Dilarang Berhenti”, dan “Dilarang Masuk” seperti pada Gambar 4.6 dan 4.7.



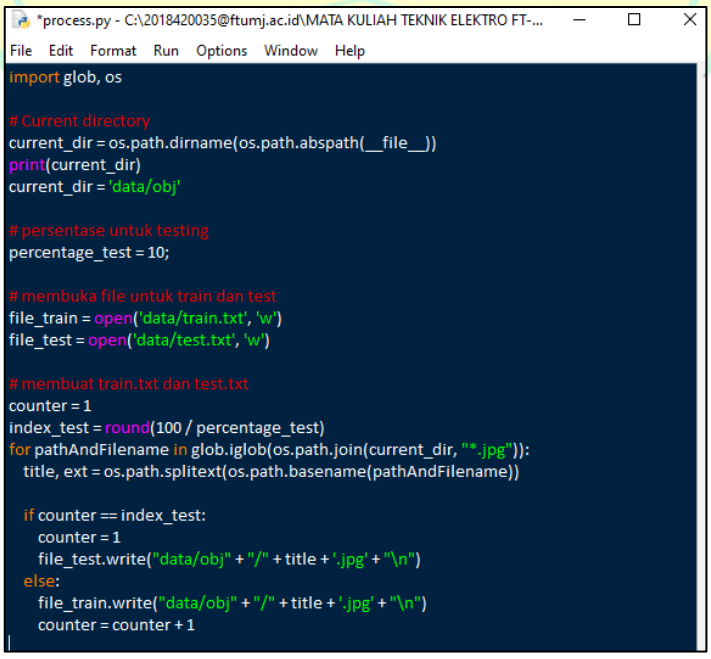
Gambar 4.6 File *obj.names*

```
Dilarang Parkir
Dilarang Berhenti
Dilarang Masuk
```

Gambar 4.7 Isi file *obj.names*

d. Membuat file *process.py*

File ini berisi *script* untuk menentukan jumlah data untuk *training* dan *testing*. Nilai perbandingannya adalah 90 % untuk *training* dan 10 % untuk *testing*.



Gambar 4.8 Program pada file *process.py*

e. Mengubah parameter file *yolov4-custom.cfg*

Proses training menggunakan *load model* Darknet dan *load weight* YOLO v4 dengan konfigurasi seperti pada Tabel 4.1. Sebelum *network weight* diperharui, jumlah gambar yang akan diproses ditentukan dengan nilai *batch.* Sebagian kecil *batch size* pada GPU akan diproses dengan *subdivision*. *Max_batch* merupakan batas maksimal iterasi pada *training*. Proses *training* akan otomatis berhenti ketika iterasi sudah mencapai nilai *max_batches* yang ditentukan.

Tabel 4.1 Konfigurasi *Darknet*

Jenis konfigurasi	Keterangan
Load weights	YOLO v4
Load model	Darknet
OPENCV	1
GPU	1
CUDNN	1
CUDNN_HALF	1
LIBSO	1

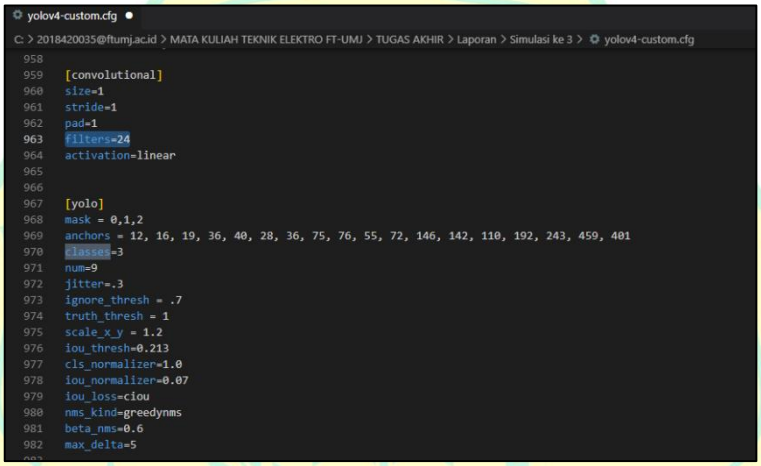
$$\text{max\_}batches = jumlah\ kelas \times 2000 \quad (4.1)$$

$$steps = (80\%\ \text{max\_}batches), (90\%\ \text{max\_}batches) \quad (4.2)$$

$$filters = (jumlah\ kelas + 5) \times 3 \quad (4.3)$$

Pada file *yolov4-custom.cfg* dilakukan perubahan beberapa parameter. Jumlah kelas pada penelitian ini sebanyak 3 kelas, nilai maka *max_batches* sejumlah 6000 iterasi. Untuk *steps* sebesar 4800,5400 dan untuk *filters* sebesar 24. Besar nilai *width* dan *height* diubah menjadi ukuran 416 × 416 untuk proses *resize* gambar.

Gambar 4.9 Perubahan nilai *width, height, max_batches, dan steps*



Gambar 4.10 Perubahan parameter *filters* dan *classes*

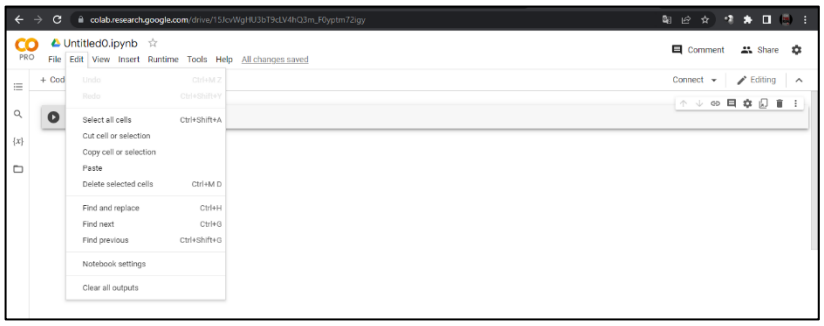
Tabel 4.2 Konfigurasi pada *weights* YOLO v4

Jenis konfigurasi	Keterangan
Width	416
Height	416
Max_batches	6000
Steps	4800,5400

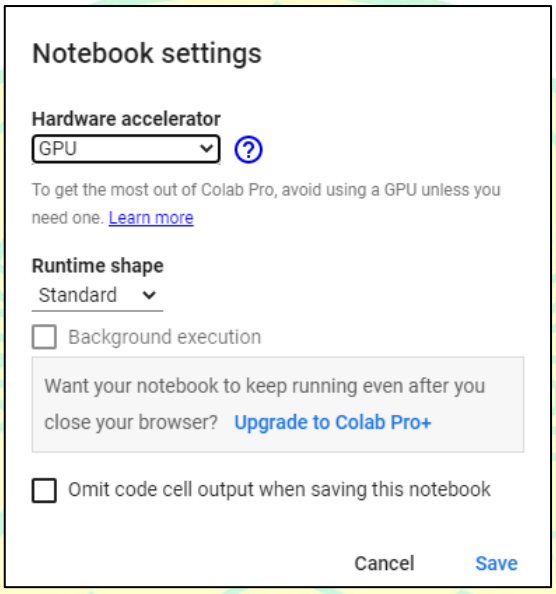
f. *Google Colab* dan mengaktifkan fitur *GPU*

Pada *notebook* di *Google Colab* terdapat virtual GPU yang digunakan untuk memproses dataset, dan cara mengaktifkannya adalah pada menu “Edit” pilih “Notebook setting” seperti pada Gambar 4.11.

Kemudian pada keterangan “Hardware accelerator” pilih “GPU” selanjutnya “Save” seperti pada Gambar 4.12.



Gambar 4.11 *Google Colab*



Gambar 4.12 Pengaturan untuk mengaktifkan virtual GPU

g. Mount file dari *Google Drive*

Tahap ini adalah proses untuk menyalin data di *Google Drive* sebagai tempat penyimpanan.

```
%cd ..
from google.colab import drive
drive.mount('/content/gdrive')
!ln -s /content/gdrive/My\ Drive/ /mydrive
%cd /mydrive/TugasAkhirYolov4
```

Gambar 4.13 Program untuk *mount* file dari *Google Drive*

```
/
Mounted at /content/gdrive
```

Gambar 4.14 Output program hasil *mount* file dari *Google Drive*

h. Mengunduh *framework darknet*

Pada bagian ini adalah proses untuk mengunduh *framework darknet* yang selanjutnya akan tersimpan di folder pada *notebook* di *Google Colab.*

```
!git clone https://github.com/AlexeyAB/darknet
```

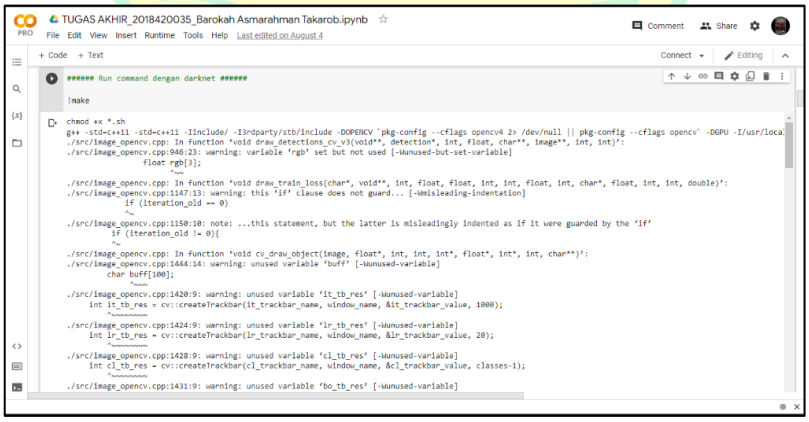
Gambar 4.15 Program untuk mengunduh *framework darknet*

i. Aktivasi OpenCV dan GPU

Pada bagian ini adalah proses untuk mengaktifkan *library* OpenCV dan GPU.

```
%cd darknet/
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed-i's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
!sed -i 's/LIBSO=0/LIBSO=1/' Makefile
!make
```

Gambar 4.16 Program untuk mengaktivasi OpenCV dan GPU



Gambar 4.17 *Output* program hasil aktivasi *OpenCV* dan *GPU*

j. Menyalin file ke *directory* Darknet

Pada tahap ini adalah proses menyalin file *yolov4-custom.cfg* ke folder *cfg* sebagai konfigurasi.

```
%cd data/
!find -maxdepth 1 -type f -exec rm -rf {} \;
%cd ..
%rm -rf cfg/
%mkdir cfg
```

Gambar 4.18 Program untuk menyalin file *yolov4-custom.cfg* ke folder *cfg*

```
/content/gdrive/MyDrive/TugasAkhirYolov4/darknet/data
/content/gdrive/MyDrive/TugasAkhirYolov4/darknet
```

Gambar 4.19 *Output* program file *yolov4-custom.cfg* yang telah disalin

k. Ekstraksi dataset

Pada bagian ini adalah proses mengekstraksi file *obj.zip* ke dalam folder *data* pada folder TugasAkhirYolov4.

```
!unzip /mydrive/TugasAkhirYolov4/obj.zip -d data/
```

Gambar 4.20 Program untuk proses ekstraksi file *obj.zip*

```
Archive:  /mydrive/TugasAkhirYolov4/obj.zip
  inflating: data/obj/img-1.jpg
  inflating: data/obj/img-1.txt
  inflating: data/obj/img-10.jpg
  inflating: data/obj/img-10.txt
  inflating: data/obj/img-100.jpg
  inflating: data/obj/img-100.txt
  inflating: data/obj/img-101.jpg
  inflating: data/obj/img-101.txt
  inflating: data/obj/img-102.jpg
  inflating: data/obj/img-102.txt
  inflating: data/obj/img-103.jpg
  inflating: data/obj/img-103.txt
  inflating: data/obj/img-104.jpg
  inflating: data/obj/img-104.txt
  inflating: data/obj/img-105.jpg
  inflating: data/obj/img-105.txt
  inflating: data/obj/img-106.jpg
  inflating: data/obj/img-106.txt
  inflating: data/obj/img-107.jpg
  inflating: data/obj/img-107.txt
  inflating: data/obj/img-108.jpg
  inflating: data/obj/img-108.txt
  inflating: data/obj/img-109.jpg
  inflating: data/obj/img-109.txt
```

Gambar 4.21 Hasil file *obj.zip* yang diekstraksi

l. Memindahkan file konfigurasi jaringan YOLO v4

Pada tahap ini adalah memindahkan file *yolov4-custom.cfg* ke dalam folder *cfg*.

```
!cp /mydrive/TugasAkhirYolov4/yolov4-custom.cfg cfg
!ls cfg/
```

Gambar 4.22 Program untuk memindahkan file *yolov4-custom.cfg* ke folder *cfg*

```
yolov4-custom.cfg
```

Gambar 4.23 Hasil pemindahan file *yolov4-custom.cfg* ke folder *cfg*

m. Menyalin file *obj.names* dan *obj.data* ke folder *data*

Tahap ini adalah menyalin file *obj.names* dan *obj.data* ke dalam folder *data.*

```
!cp /mydrive/TugasAkhirYolov4/obj.names data
!cp /mydrive/TugasAkhirYolov4/obj.data  data
!ls data/
```

Gambar 4.24 Program untuk menyalin file *obj.names* dan *obj.data*

```
labels  obj  obj.data  obj.names
```

Gambar 4.25 *Output* hasil file *obj.names* dan *obj.data* yang di salin

n. Menyalin *script process.py* untuk membuat file *training* dan *testing*

```
!cp /mydrive/TugasAkhirYolov4/process.py .
!python process.py
!ls data/
```

Gambar 4.26 Program untuk menyalin *script process.py*

```
/content/gdrive/My Drive/TugasAkhirYolov4/darknet
labels  obj  obj.data  obj.names  test.txt  train.txt
```

Gambar 4.27 *Output* program hasil file *script process.py* yang disalin

o. Download file *weight* model YOLO v4

```
!wget
https://github.com/AlexeyAB/darknet/releases/
download/darknet_yolo_v3_optimal/
yolov4.conv.137
```

Gambar 4.28 Program untuk mengunduh file *weght* model YOLO v4

```
--2022-07-24 03:52:01--  https://github.com/AlexeyAB/darknet/releases/download/darknet
Resolving github.com (github.com)... 52.192.72.89
Connecting to github.com (github.com)|52.192.72.89|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be
--2022-07-24 03:52:02--  https://objects.githubusercontent.com/github-production-relea
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.108
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.10
HTTP request sent, awaiting response... 200 OK
Length: 170038676 (162M) [application/octet-stream]
Saving to: 'yolov4.conv.137.2'

yolov4.conv.137.2   100%[===================>] 162.16M  3.88MB/s    in 32s

2022-07-24 03:52:34 (5.10 MB/s) - 'yolov4.conv.137.2' saved [170038676/170038676]
```

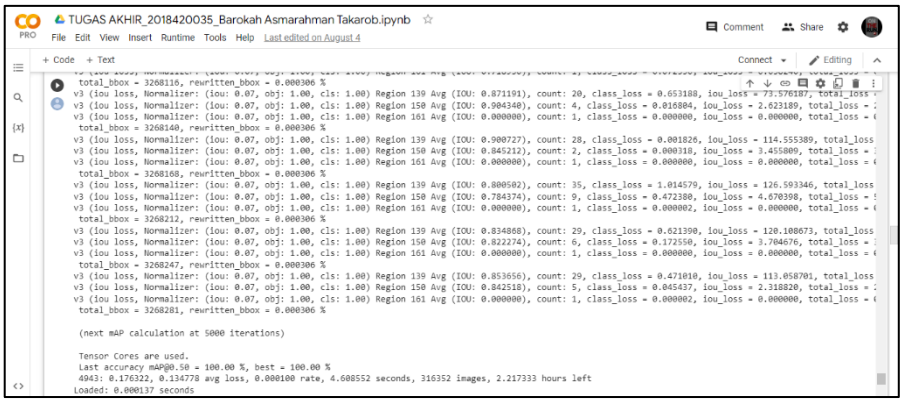
Gambar 4.29 *Output* hasil *download* file *weights*

4.1.3 Training Data

Training dataset dilakukan dengan durasi ± 12 jam tergantung dari jumlah dataset. Tujuan dari proses ini adalah melatih komputer dengan memproses gambar yang telah dianotasi untuk membentuk format dari setiap kelas yang nantinya digunakan komputer sebagai pertimbangan dalam pengambilan keputusan. Pada bagian ini memanfaatkan model konfigurasi yang sudah dikustomisasi sebelumnya yaitu *yolov4-custom.cfg.*

```
!./darknet detector train data/obj.data cfg/
yolov4-custom.cfg yolov4.conv.137 -dont_show -map
```

Gambar 4.30 Program untuk training dataset dengan *yolov4-custom.cfg*



Gambar 4.31 *Output* hasil *training* dengan *yolov4-custom.cfg*

Pada tahan ini terdapat masalah *disconnect* pada koneksi jaringan internet sehingga proses *training* terhenti saat iterasi < 5000 yaitu pada iterasi ke 4943 dan menghasilkan file *yolov4-custom_last.weights* pada folder *training* di *Google Drive* seperti yang ditunjukkan pada Gambar 4.32.

Gambar 4.32 File *yolov4-custom_last.weights*

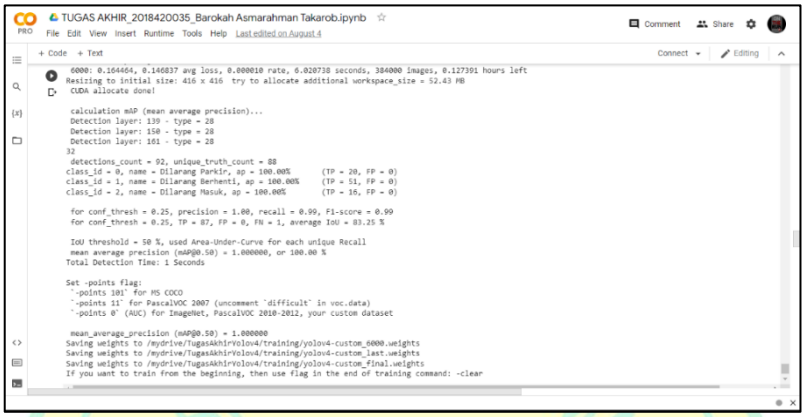
a. *Restart training* dengan file *weights* terakhir

Proses training yang berhenti sebelumnya bisa dilanjutkan kembali tanpa harus memulai lagi dari awal dengan menggunakan file *yolov4-custom_last.weights* yang merupakan hasil *bakcup* saat terjadi suatu masalah seperti *disconnect*.

```
!./darknet detector train data/obj.data cfg/
yolov4-custom.cfg /mydrive/TugasAkhirYolov4/
training/yolov4-custom_last.weights
-dont_show -map
```

Gambar 4.33 Program untuk melanjutkan *training* yang berhenti

Proses *training* dilanjutkan kembali dan apabila proses *training* berhenti kembali seperti sebelumnya maka bisa dilanjutkan kembali dengan program seperti pada Gambar 4.33. Dan hasil akhir proses *training* selesai pada iterasi ke 6000 seperti ditunjukkan pada Gambar 4.34 dan Tabel 4.3.



Gambar 4.34 Hasil akhir proses *training*

Tabel 4.3 Data hasil *training*

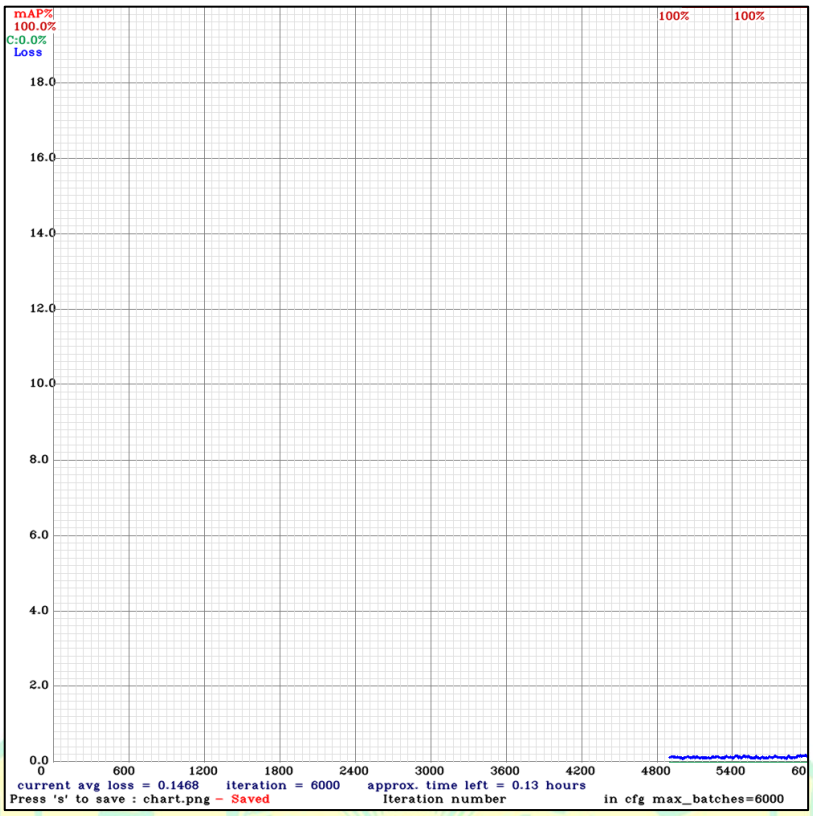
Load model		***yolov4-custom***
Dilarang Parkir	***AP***	100 %
	TP	20
	FP	0
Dilarang Berhenti	***AP***	100 %
	TP	51
	FP	0
Dilarang Masuk	***AP***	100 %
	TP	16
	FP	0
Precision		1,00
Recall		0,99
F1-score		0,99
FN		1
IoU		83,25 %
mAP@0,5		100 %
Detection time		1 detik
Accuracy		95,80 %

Berdasarkan data hasil *training* pada Tabel 4.3, untuk kelas rambu "Dilarang Parkir" menyatakan jumlah *True Positive* 20, *False Positive* 0 dan *AP* 100 %. Pada kelas rambu "Dilarang Berhenti" menyatakan jumlah *True Positive* 51, *False Positive* 0 dan *AP* 100 %. Kelas rambu "Dilarang Masuk" menyatakan jumlah *True Positive* 16, *False Positive* 0 dan *AP* 100%. Hal ini membuktikan sistem mampu mengidentifikasi objek dengan presisi sesuai data *training*. *Yolov4-custom* memerlukan waktu proses 1 detik untuk mengidentifikasi 3 kelas rambu dengan rata-rata *accuracy* 95,80%.

Kemudian, diperoleh nilai *precision* 100 % dalam mengidentifikasi objek. Sementara *recall* bernilai 99 % dalam mengukur kemampuan model untuk menemukan seluruh objek. Pada simulasi dilakukan kalkulasi terhadap nilai *precision* dan *recall* untuk menghasilkan keseimbangan nilai tersebut maka diperoleh nilai *F1-Score* sebesar 99 %.

b. Cek performansi YOLO v4 dengan *mean Average Precision* (mAP)

Berdasarkan hasil *training data* maka diperoleh beberapa jumlah iterasi yang di dalamnya terdapat ketentuan nilai dari *mean Average Precision (mAP)* pada masing-masing iterasi tersebut. Dalam hal ini jumlah iterasi adalah mulai dari 1000 sampai dengan 6000 iterasi. Namun karena sebelumnya terjadi masalah *disconnect* saat iterasi ke 4943 maka nilai *mAP* yang ditampilkan dalam grafik berikut adalah mulai dari iterasi ke 4943 sampai 6000.



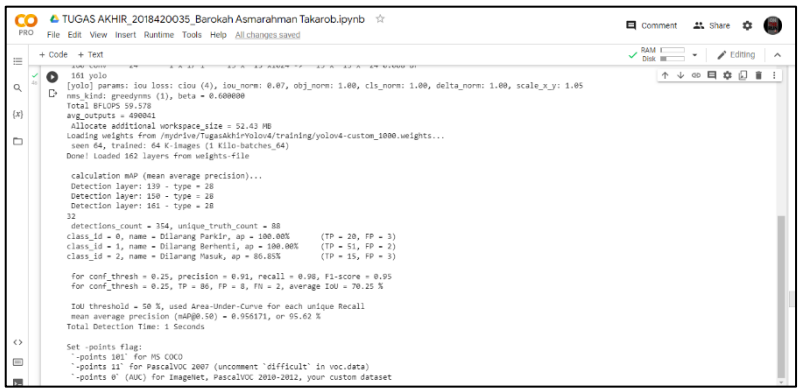
Gambar 4.35 Grafik *mean Average Precision* dan *loss* hasil *training*

Untuk grafik *mAP* dari iterasi 0 sampai 4942 tidak dapat ditampilkan karena disebabkan masalah *disconnect* sebelumnya, namun hasil *training* selalu di-*record* setiap 1000 iterasi. Dan untuk mengetahui nilai *mAP* mulai dari iterasi 1000 sampai 6000 bisa diperoleh berdasarkan data hasil *training* yang disimpan pada *Google Drive* dalam folder *training.*

c. *mean Average Precision* pada iterasi 1000

```
!./darknet detector map data/obj.data cfg/
yolov4-custom.cfg /mydrive/TugasAkhirYolov4/
training/yolov4-custom_1000.weights -points 0
```

Gambar 4.36 Program untuk cek *mAP* pada iterasi 1000

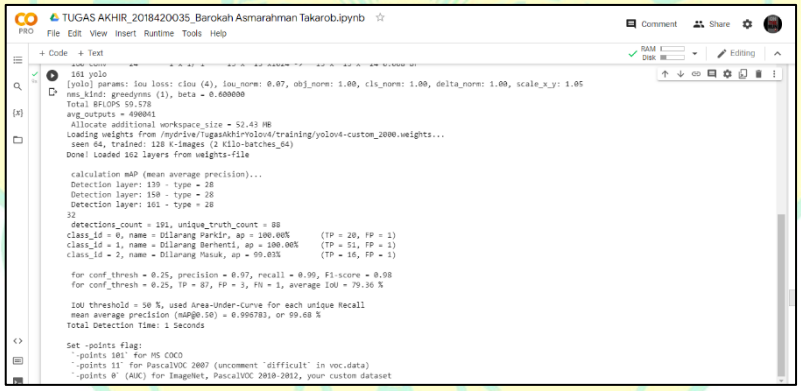


Gambar 4.37 *Output* hasil cek *mAP* pada iterasi 1000

d. *mean Average Precision* pada iterasi 2000

```
!./darknet detector map data/obj.data cfg/
yolov4-custom.cfg /mydrive/TugasAkhirYolov4/
training/yolov4-custom_2000.weights -points 0
```

Gambar 4.38 Program untuk cek *mAP* pada iterasi 2000

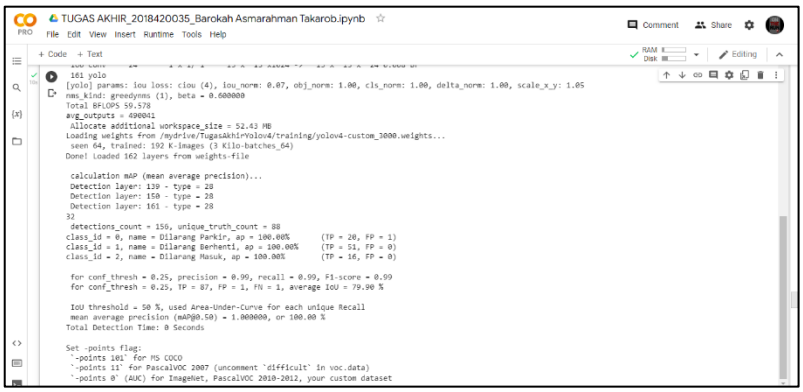


Gambar 4.39 *Output* hasil cek *mAP* pada iterasi 2000

e. *mean Average Precision* pada iterasi 3000

```
!./darknet detector map data/obj.data cfg/
yolov4-custom.cfg /mydrive/TugasAkhirYolov4/
training/yolov4-custom_3000.weights -points 0
```

Gambar 4.40 Program untuk cek *mAP* pada iterasi 3000

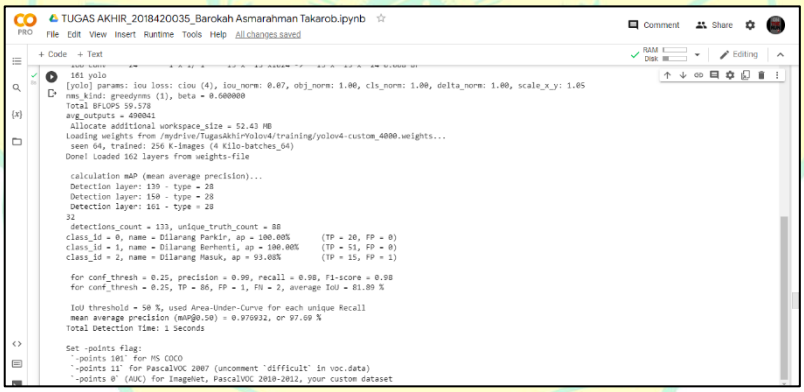


Gambar 4.41 *Output* hasil cek *mAP* pada iterasi 3000

f. *mean Average Precision* pada iterasi 4000

```
!./darknet detector map data/obj.data cfg/
yolov4-custom.cfg /mydrive/TugasAkhirYolov4/
training/yolov4-custom_4000.weights -points 0
```

Gambar 4.42 Program untuk cek *mAP* pada iterasi 4000

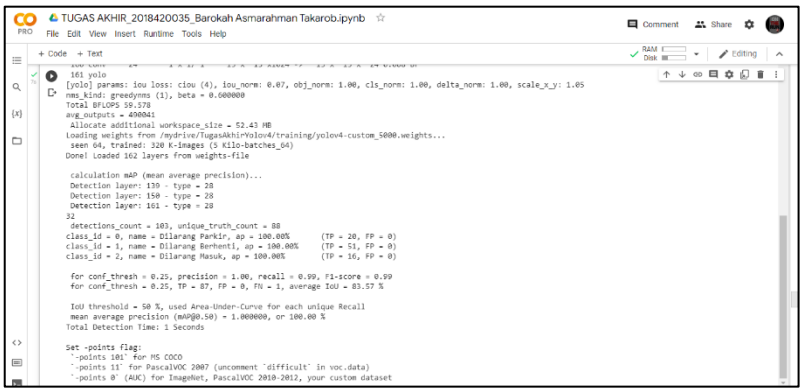


Gambar 4.43 *Output* hasil cek *mAP* pada iterasi 4000

g. *Mean Average Precison* pada iterasi 5000

```
!./darknet detector map data/obj.data cfg/
yolov4-custom.cfg /mydrive/TugasAkhirYolov4/
training/yolov4-custom_5000.weights -points 0
```

Gambar 4.44 Program untuk cek *mAP* pada iterasi 5000

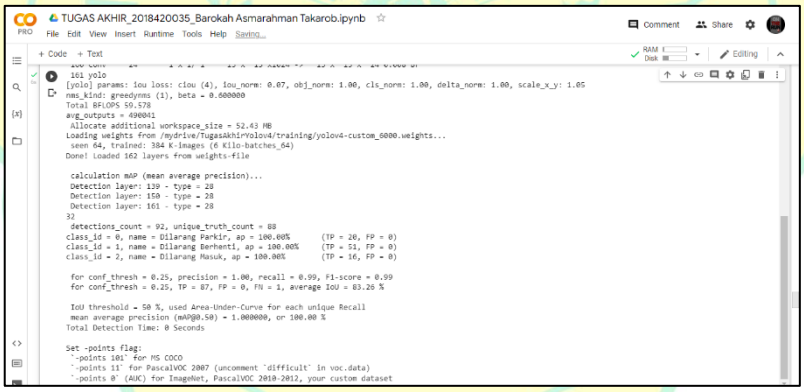


Gambar 4.45 *Output* hasil cek *mAP* pada iterasi 5000

h. *mean Average Precision* pada iterasi 6000

```
!./darknet detector map data/obj.data cfg/
yolov4-custom.cfg /mydrive/TugasAkhirYolov4/
training/yolov4-custom_6000.weights -points 0
```

Gambar 4.46 Program untuk cek *mAP* pada iterasi 6000

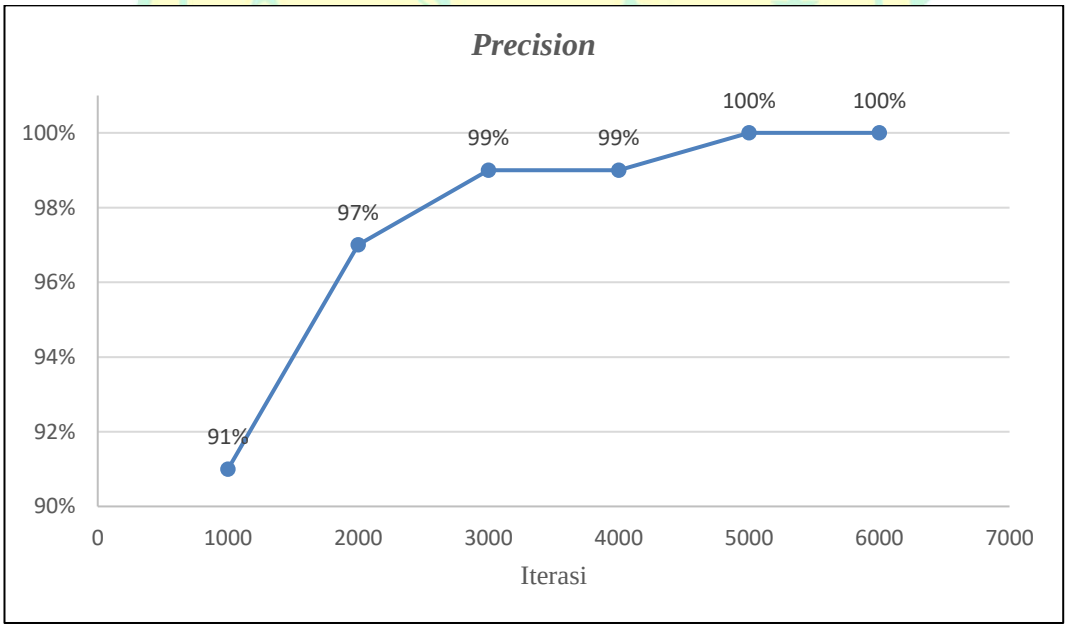


Gambar 4.47 *Output* hasil cek *mAP* pada iterasi 6000

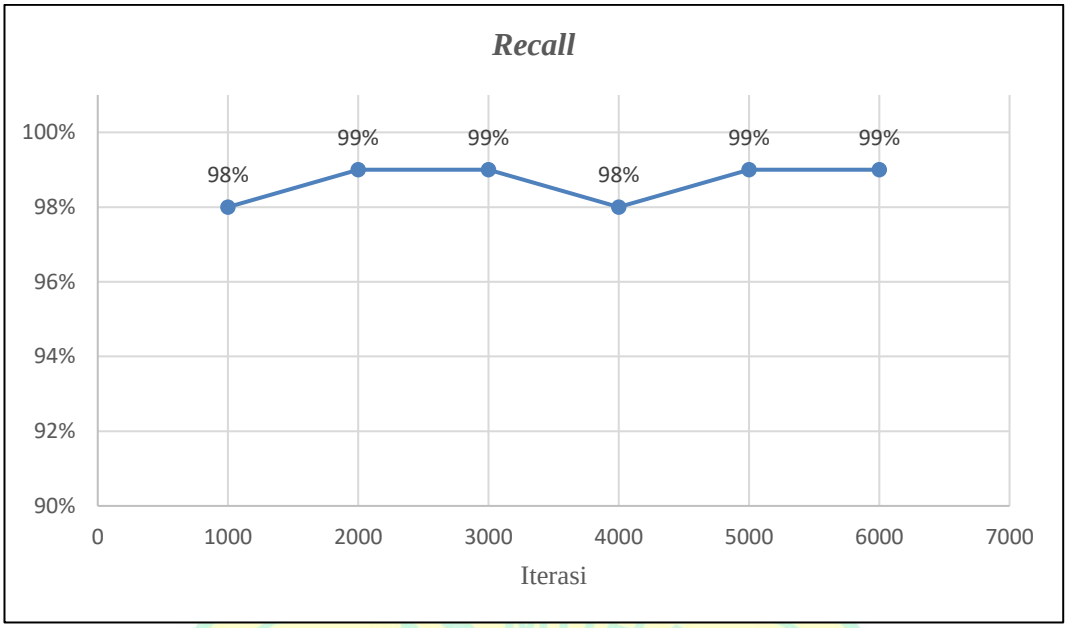
Berdasarkan data dari *output* program pada setiap iterasi mulai dari 1000 sampai dengan 6000 iterasi dapat dilihat rinciannya pada Tabel 4.4 dan gambar berikut.

Tabel 4.4 Nilai *mAP* berdasarkan jumlah iterasi

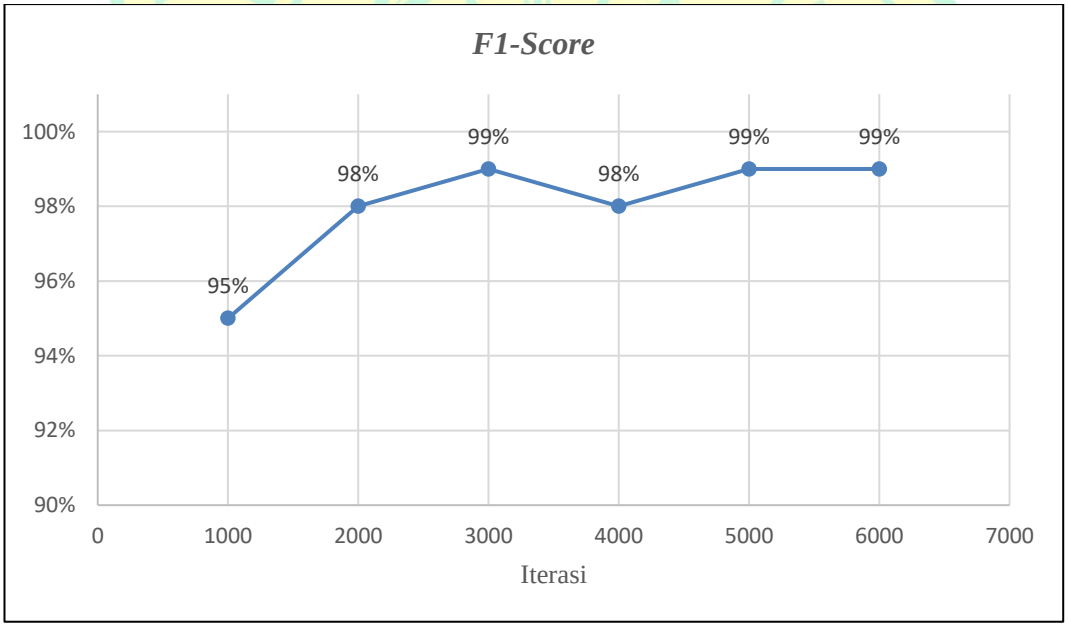
Jenis Kelas		**Jumlah Iterasi**					
		1000	**2000**	**3000**	**4000**	**5000**	**6000**
Dilarang Parkir	***AP***	100 %	100 %	100 %	100 %	100 %	100 %
	TP	20	20	20	20	20	20
	FP	3	1	1	0	0	0
Dilarang Berhenti	***AP***	100 %	100 %	100 %	100 %	100 %	100 %
	TP	51	51	51	51	51	51
	FP	2	1	0	0	0	0
Dilarang Masuk	***AP***	86,85 %	99.03 %	100 %	93,08 %	100 %	100 %
	TP	15	16	16	15	16	16
	FP	3	1	0	1	0	0
Precision		0,91	0,97	0,99	0,99	1,00	1,00
Recall		0,98	0,99	0,99	0,98	0,99	0,99
F1-score		0,95	0,98	0,99	0,98	0,99	0,99
FN		2	1	1	2	1	1
IoU		70,25 %	79,36 %	79,90 %	81,89 %	83,57 %	83,26 %
mAP@0,5		95,62 %	99,68 %	100 %	97,69 %	100 %	100 %
Detection time		20 detik	1 detik	1 detik	1 detik	1 detik	1 detik
Accuracy		82,57 %	92,00 %	94,36 %	90,15 %	95,80 %	95,80 %



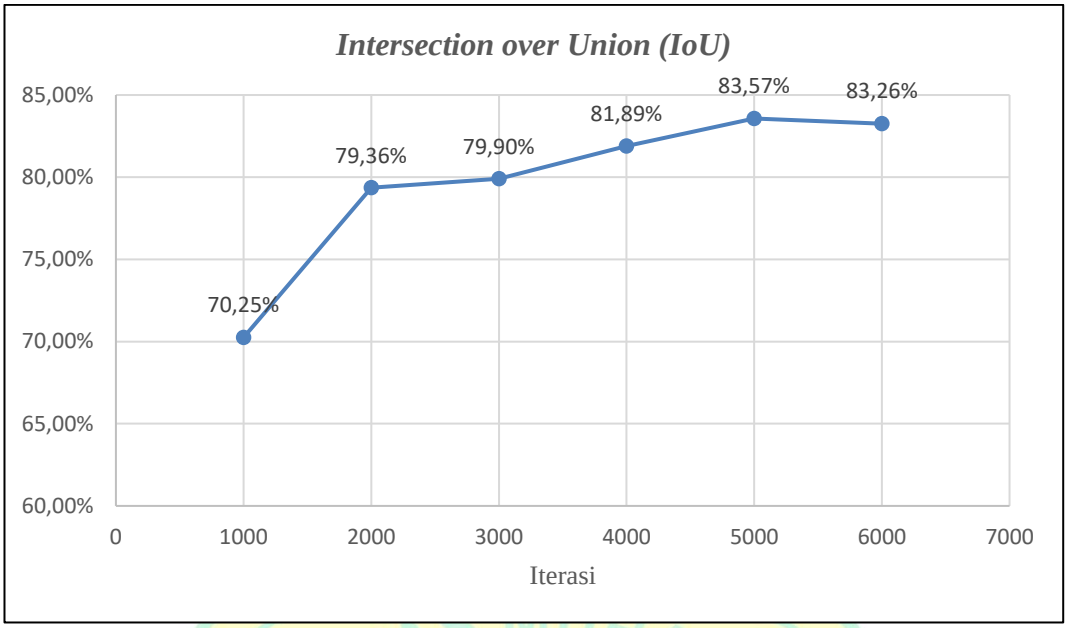
Gambar 4.48 Grafik *Precision*



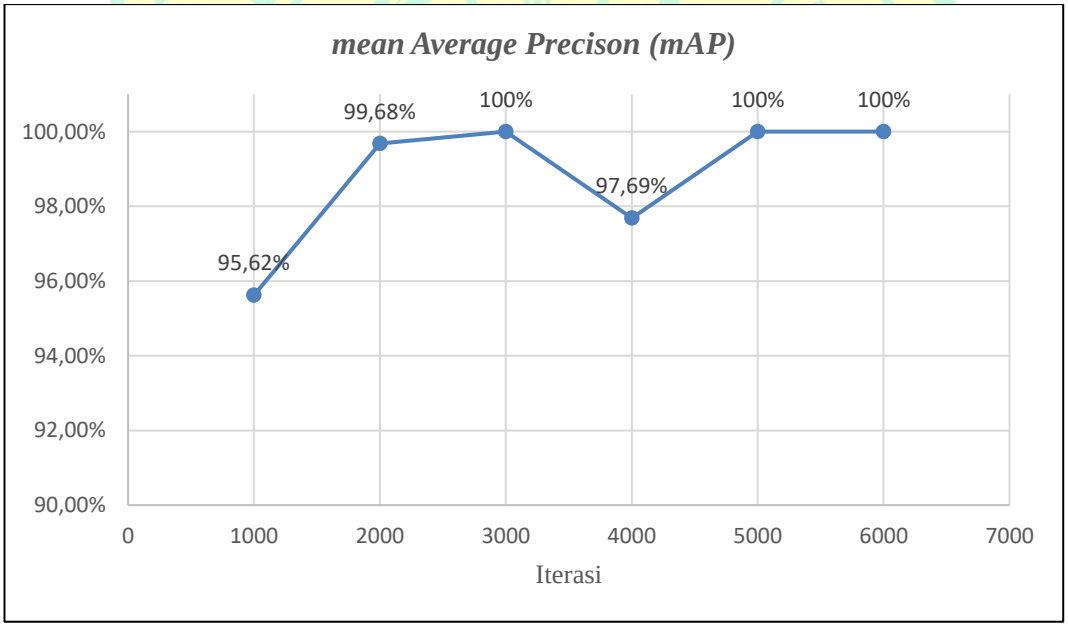
Gambar 4.49 Grafik *Recall*



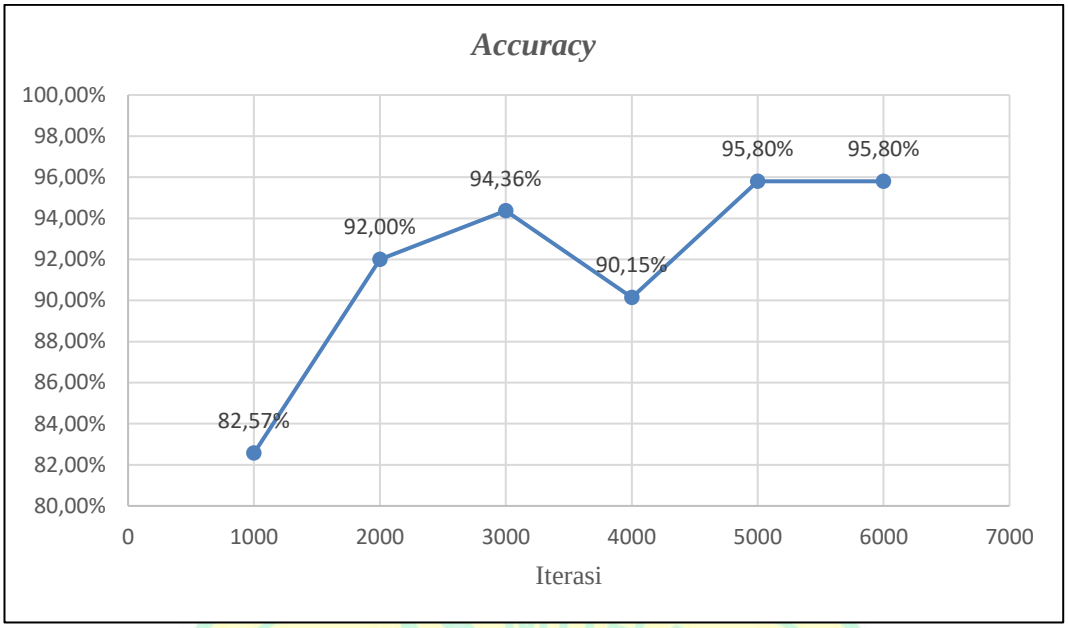
Gambar 4.50 Grafik *F1-Score*



Gambar 4.51 Grafik *Intersection over Union*



Gambar 4.52 Grafik *mean Average Precision*



Gambar 4.53 Grafik *Accuracy*

4.2 Pengujian

4.2.1 Pengujian Dan Hasil Deteksi

Tahap ini akan dilakukan pengujian untuk mengetahui akurasi dari model *weight yolov4-custom*. Pengujian performa *yolov4-custom* menggunakan 3 kelas rambu diantaranya "Dilarang Parkir", "Dilarang Berhenti", dan "Dilarang Masuk" dengan total data 329 gambar dimana 296 gambar untuk *training* dan 33 gambar untuk *testing*.

Data validasi diproses menjadi *ground truth box* dan *output* dari *predicted box* menghasilkan *confusion matrix*. *Confusion matrix* tersebut akan dikalkulasi untuk mendapatkan nilai *Precision*, *Recall*, *AP*, *F1-Score*, *IoU*, *mAP* dan *Accuracy*.

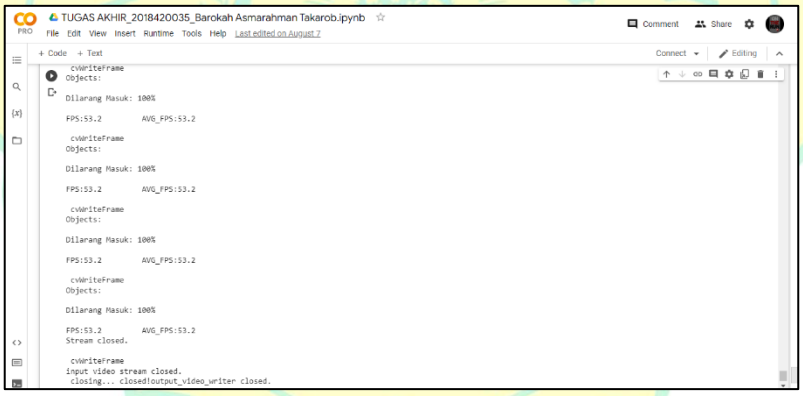
a. Dengan gambar

Pada tahap pengujian dataset dengan menggunakan salah satu gambar, diperoleh *output* hasil deteksi rambu "Dilarang Berhenti" dengan nilai *confidence* sebesar 98 % dan *predicted time* 20,381 *milli seconds* seperti pada gambar berikut.

Gambar 4.54 *Output* pengujian dengan gambar

b. Dengan *video*

Adapun saat pengujian dataset dengan menggunakan salah satu *video*, diperoleh *output* hasil deteksi dengan 3 kelas rambu yaitu "Dilarang Parkir", "Dilarang Berhenti", dan "Dilarang Masuk" dengan nilai *confidence* sebesar 100 % dan rata-rata 53,2 *Frame Per Second* seperti pada gambar berikut.



Gambar 4.55 *Output* pengujian dengan video

Gambar 4.56 *Output* deteksi dengan *video* (a)

Gambar 4.57 *Output* deteksi dengan *video* (b)

Gambar 4.58 *Output* deteksi dengan *video* (c)

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan penelitian yang telah dilaksanakan maka diperoleh kesimpulan sebagai berikut:

a. Hasil simulasi beberapa rambu lalu lintas yaitu “Dilarang Parkir”, “Dilarang Berhenti”, dan “Dilarang Masuk” menggunakan algoritma YOLO v4 terbukti bekerja dengan optimal.
b. Algoritma YOLO v4 memperoleh kecepatan pendeteksian objek dengan nilai *confidence* sebesar 100 % dan kecepatan rata-rata 53,2 *frame per second*.
c. Hasil *training* setiap kelas memperlihatkan jumlah *true positive* lebih besar dibandingkan jumlah *false positive* dengan nilai *mAP* 100 % dan *accuracy* 95,80 % pada iterasi terakhir, hal ini membuktikan sistem dapat mengidentifikasi objek dengan optimal.
d. Tahap *training* membutuhkan waktu yang cukup lama dibandingkan pada tahap *testing*.

5.2 Saran

Berikut adalah beberapa saran untuk penelitian berikutnya berdasarkan simulasi yang sudah dilakukan:

a. Menambahkan jumlah dataset dan anotasi pada setiap jenis kelas untuk meningkatkan tingkat mAP saat *training.*
b. Perlu dikembangkan sebuah metode khusus agar mampu mengatasi durasi proses *training* yang cukup lama.
c. Meningkatkan kecepatan pendeteksian objek dalam pengembangan *autonomous vehicle* sebagai upaya mengurangi angka kecelakaan lalu lintas.

DAFTAR PUSTAKA

[1] Umar, Y., Hanafi, Mardi, S., Nugroho, Susiki, & Rachmadi, R. F., "Deteksi Penggunaan Helm Pada Pengendara," 2020.

[2] A. Aszhari, "Liputan6," Juli 2020. [Online]. Available: https://www.liputan6.com/otomotif/read/3657076/tekan-angkakecelakaan-jangan-pernah-melanggar-lalu-lintas.

[3] WHO, "Global Status Report on Road," World Health Organ, 2018.

[4] Bahri, Saeful; Riza, Samsinar; Panggalih Sako, Denta, "Pengenalan Ekspresi Wajah untuk Identifikasi Psikologis Pengguna dengan Neural Network dan Transformasi Ten Crops," *RESISTOR (Elektronika Kendali Telekomunikasi Tenaga Listrik Komputer) Vol. 5 No. 1*, 2022.

[5] LLAJ , Undang-Undang Republik Indonesia Nomor 22 Tahun 2009, Jakarta, 2009.

[6] Menteri Perhubungan Republik Indonesia, Peraturan Menteri Perhubungan Republik Indonesia Nomor PM 13 Tahun 2014 Tentang Rambu Lalu Lintas, Jakarta, 2014.

[7] Jalled, F., & Voronkov, I., "Object Detection using Image Processing," pp. 1-6, 2016.

[8] A. Salim, "Estimasi Kecepatan Kendaraan Melalui Video Pengawas Lalu Lintas," pp. 1-100, 2020.

[9] S. Dewi, "Deep Learning Object Detection Pada Video Menggunakan Tensorflow Dan Convolutional Neural Network," pp. 1-60, 2016.

[10] Redmon, J., Divvala, S., Girshick, R., Farhadi, A., "You Only Look Once: Unified, Real-Time Object Detection," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779-788, 2016.

[11] Putra, M. H., Yussof, Z. M., Lim, K. C., Salim, S. I., "Convolutional Neural Network for Person and Car Detection using YOLO Framework," *Journal of Telecommunication, Electronic, and Computer Engineering*, pp. 67-71, 2018.

[12] Baojun Zhang, Guili Wang , Huilan Wang, Chenchen Xu, Yu Li, dan Lin Xu, "Detecting Small Chinese Traffic Signs via Improved," *Hindawi Mathematical Problems in Engineering*, p. 10, 2021.

[13] H. M. Alfarisi, "Yolov4: Teknologi Terbaru dalam Perkembangan Algoritma Object Detection," 2020.

[14] Bochkovskiy, A., Wang, C. Y., Liao, H. Y. M., "YOLOv4: Optimal Speed and Accuracy of Object Detection," *arXiv:2004.1093v1*, 2020.

[15] Gautama, T. K., Hendrik, A., & Hendaya, R., "Pengenalan Objek pada Computer Vision dengan Pencocokan Fitur Menggunakan Algoritma SIFT Studi Kasus: Deteksi Penyakit Kulit Sederhana," *Jurnal Teknik Informatika Dan Sistem Informasi*, pp. 437-450, 2016.

[16] Muchtar, Husnibes; Rizky, Apriyadi, "Implementasi Pengenalan Wajah Pada Sistem Penguncian Rumah dengan Metode Template Matching Menggunakan Open Source Computer Vision Library (Opencv)," *RESISTOR (elektRonika kEndali telekomunikaSI tenaga liSTrik kOmputeR) Vol. 2 No. 1*, 2019.

[17] R. Nurhawanti, "Sistem Pendeteksi Sepeda Motor Pelanggar Marka Jalan Menggunakan Metode Convolutional Neural Networks (CNNs)," 2019.

[18] A. Ramadhan, "Polri Sebut Jumlah Kecelakaan Lalu Lintas Meningkat Pada 2019," Jakarta, 2019.

[19] Redmon, J., & Farhadi, A., "YOLOv3: An Incremental Improvement," 2018. [Online]. Available: http://arxiv.org/abs/1804.02767.

[20] Wisna H, J. S., Matulatan, T., dan Hayaty, N., "Deteksi Kendaraan Secara Real Time Menggunakan Metode YOLO Berbasis Android," *Jurnal Sustainable: Jurnal Hasil Penelitian dan Industri Terapan*, pp. 8-14, 2020.

[21] Zulkhaidi, T. C. A., Maria, E., dan Yulianto, "Pengenalan Pola Bentuk Wajah dengan OpenCV," *Jurti*, p. 182, 2019.

LAMPIRAN

Lampiran 1 *Script process.py*

```
import glob, os

# Current directory
current_dir = os.path.dirname(os.path.abspath(__file__))

print(current_dir)

current_dir = 'data/obj'

# Persentase untuk testing
percentage_test = 10;

# File yang dijadikan train.txt dan test.txt
file_train = open('data/train.txt', 'w')
file_test = open('data/test.txt', 'w')

# Membuat train.txt dan test.txt
counter = 1
index_test = round(100 / percentage_test)
for pathAndFilename in glob.iglob(os.path.join(current_dir, "*.jpg")):
    title, ext = os.path.splitext(os.path.basename(pathAndFilename))

    if counter == index_test:
        counter = 1
        file_test.write("data/obj" + "/" + title + '.jpg' + "\n")
    else:
        file_train.write("data/obj" + "/" + title + '.jpg' + "\n")
        counter = counter + 1
```

Lampiran 2 *Script yolov4-custom.cfg*

```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=16
width=416
height=416
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 6000
policy=steps
steps=4800,5400
scales=.1,.1

#cutmix=1
mosaic=1

#:104x104 54:52x52 85:26x26 104:13x13 for 416

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=mish
```

```
# Downsample

[convolutional]
batch_normalize=1
filters=64
size=3
stride=2
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=mish

[route]
layers = -2

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=32
size=1
stride=1
pad=1
activation=mish
```

```
[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=mish

[route]
layers = -1,-7

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=mish

# Downsample

[convolutional]
batch_normalize=1
filters=128
size=3
stride=2
```

```
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=mish

[route]
layers = -2

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=mish
```

```
[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=mish

[route]
layers = -1,-10

[convolutional]
batch_normalize=1
filters=128
```

```
size=1
stride=1
pad=1
activation=mish

# Downsample

[convolutional]
batch_normalize=1
filters=256
size=3
stride=2
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=mish

[route]
layers = -2

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=128
size=1
```

```
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
```

```
filters=128
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear
```

```
[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=mish

[shortcut]
```

```
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=mish
```

```
[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=mish

[route]
layers = -1,-28

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=mish

# Downsample

[convolutional]
batch_normalize=1
filters=512
size=3
stride=2
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=256
size=1
```

```
stride=1
pad=1
activation=mish

[route]
layers = -2

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear



[convolutional]
batch_normalize=1
```

```
filters=256
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear
```

```
[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=mish
```

```
[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=256
```

```
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=mish

[route]
```

```
layers = -1,-28

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=mish

# Downsample

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=2
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=mish

[route]
layers = -2

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=mish
```

```
[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=mish

[shortcut]
```

```
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=mish
```

```
[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=mish

[route]
layers = -1,-16

[convolutional]
batch_normalize=1
filters=1024
size=1
stride=1
pad=1
activation=mish
stopbackward=800

##########################

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
```

```
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

### SPP ###
[maxpool]
stride=1
size=5

[route]
layers=-2

[maxpool]
stride=1
size=9

[route]
layers=-4

[maxpool]
stride=1
size=13

[route]
layers=-1,-3,-5,-6
### End SPP ###

[convolutional]
batch_normalize=1
```

```
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[upsample]
stride=2

[route]
layers = 85

[convolutional]
batch_normalize=1
```

```
filters=256
size=1
stride=1
pad=1
activation=leaky

[route]
layers = -1, -3

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
```

```
filters=512
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[upsample]
stride=2

[route]
layers = 54

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[route]
layers = -1, -3

[convolutional]
batch_normalize=1
```

```
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky
```

```
##########################

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=24
activation=linear


[yolo]
mask = 0,1,2
anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110,
192, 243, 459, 401
classes=3
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
scale_x_y = 1.2
iou_thresh=0.213
cls_normalizer=1.0
iou_normalizer=0.07
iou_loss=ciou
nms_kind=greedynms
beta_nms=0.6
max_delta=5


[route]
```

```
layers = -4

[convolutional]
batch_normalize=1
size=3
stride=2
pad=1
filters=256
activation=leaky

[route]
layers = -1, -16

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
```

```
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=24
activation=linear


[yolo]
mask = 3,4,5
anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110,
192, 243, 459, 401
classes=3
num=9
jitter=.3
```

```
ignore_thresh = .7
truth_thresh = 1
scale_x_y = 1.1
iou_thresh=0.213
cls_normalizer=1.0
iou_normalizer=0.07
iou_loss=ciou
nms_kind=greedynms
beta_nms=0.6
max_delta=5


[route]
layers = -4

[convolutional]
batch_normalize=1
size=3
stride=2
pad=1
filters=512
activation=leaky

[route]
layers = -1, -37

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
```

```
pad=1
filters=1024
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
size=1
```

```
stride=1
pad=1
filters=24
activation=linear



[yolo]
mask = 6,7,8
anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110,
192, 243, 459, 401
classes=3
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
scale_x_y = 1.05
iou_thresh=0.213
cls_normalizer=1.0
iou_normalizer=0.07
iou_loss=ciou
nms_kind=greedynms
beta_nms=0.6
max_delta=5
```

LEMBAR BIMBINGAN TUGAS AKHIR

NAMA MAHASISWA	Barokah Asmarahman Takarob
NIM	2018420035
AKADEMIK	Genap
DOSEN PEMBIMBING	Ir. Husnibes Muchtar, M.T.

NO	TANGGAL	MATERI BIMBINGAN	TANDA TANGAN PEMBIMBING
1	15 April 2022	Tata cara penyusunan laporan Tugas Akhir	
2	24 April 2022	Menentukan latar belakang dan tujuan penelitian	
3	30 April 2022	Menentukan kerangka pemikiran	
4	3 Mei 2022	Cara meninjau teori sebagai dasar acuan	
5	10 Mei 2022	Cara mengutip tulisan dan sumber bacaan	
6	14 Mei 2022	Menentukan bahan-bahan untuk BAB III	
7	15 Juli 2022	Penyusunan BAB IV	
8	17 Juli 2022	Mengatasi masalah pada program	
9	19 Juli 2022	Mengatasi masalah pada program	
10	24 Juli 2022	Menentukan kesimpulan dan saran	

Dosen Pembimbing menyatakan Penulisan Tugas Akhir Sudah Selesai pada Tanggal 25 Juli 2022

Menyetujui untuk disidangkan

Ir. Husnibes Muchtar, M.T.