



TUGAS AKHIR - EC184801

***KNOWLEDGE DISTILLATION PADA SISTEM
PENDETEKSI OBJEK YOLO UNTUK ROBOT
SERVICE***

Billy
NRP 07211540000015

Dosen Pembimbing
Muhtadin, S.T., M.T.
Dr. Eko Mulyanto Yuniarno, S.T., M.T.

DEPATERMEN TEKNIK KOMPUTER
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2019



ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - EC184801

***KNOWLEDGE DISTILLATION PADA SISTEM
PENDETEKSI OBJEK YOLO UNTUK ROBOT
SERVICE***

Billy
NRP 0721154000015

Dosen Pembimbing
Muhtadin, S.T., M.T.
Dr. Eko Mulyanto Yuniarno, S.T., M.T.

DEPARTEMEN TEKNIK KOMPUTER
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2019

[Halaman ini sengaja dikosongkan].



FINAL PROJECT - EC184801

KNOWLEDGE DISTILLATION ON YOLO OBJECT DETECTION SYSTEM FOR ROBOT SERVICE

Billy
NRP 07211540000015

Advisor
Muhtadin, ST., M.T.
Dr. Eko Mulyanto Yuniarno, ST., M.T.

Departement of Computer Engineering
Faculty of Electrical Technology
Sepuluh Nopember Institute of Technology
Surabaya 2019

[Halaman ini sengaja dikosongkan].

PERNYATAAN KEASLIAN TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "***Knowledge Distillation Pada Sistem Pendeteksi Objek Yolo Untuk Robot Service***" adalah benar-benar hasil karya intelektual sendiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya orang lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka.

Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai dengan peraturan yang berlaku.

Surabaya, Juli 2019



Billy

NRP. 07211540000015

LEMBAR PENGESAHAN

Knowledge Distillation pada Sistem Pendeteksi Objek Yolo untuk Robot Service

Tugas Akhir ini disusun untuk memenuhi salah satu syarat memperoleh gelar Sarjana Teknik di Institut Teknologi Sepuluh Nopember Surabaya

Oleh : Billy (NRP: 07211540000015)

Tanggal Ujian : 19 Juni 2019

Periode Wisuda : September 2019

Disetujui oleh:

Muhtadin, S.T., M.T.
NIP: 198106092009121003

(Pembimbing I)

Dr. Eko Mulyanto Yuniarno, S.T., M.T.
NIP: 196806011995121009

(Pembimbing II)

Reza Fuad Rachmadi, S.T., M.T.
NIP. 198504032012121001

(Penguji I)

Ahmad Zaini, S.T., M.T.
NIP: 197504192002121003

(Penguji II)

Eko Pramunanto, S.T., M.T.
NIP: 196612031994121001

(Penguji III)

Mengetahui
Kepala Departemen Teknik Komputer

Dr. I Ketut Eddy Purnama, S.T., M.T.
NIP. 196907301995121001



ABSTRAK

Nama Mahasiswa : Billy
Judul Tugas Akhir : *Knowledge Distillation* pada Sistem Pendeteksi Objek YOLO untuk *Robot Service*
Pembimbing : 1. Muhtadin, ST., MT
2. Dr. Eko Mulyanto Yuniarno, ST., MT.

Deep learning sudah memiliki akurasi yang bagus, namun memerlukan komputasi yang besar. Akibatnya untuk saat diterapkan di perangkat yang memiliki kemampuan komputasi dan memori yang terbatas seperti robot pada umumnya, waktu komputasi model menjadi lambat atau tidak bisa dijalankan. Agar performa model *deep learning* lebih optimal pada perangkat dengan komputasi minimal, diperlukan model yang memiliki ukuran kecil namun akurasi tetap sama dengan model ukuran besar. Sehingga pada tugas akhir ini akan dilakukan uji coba beberapa metode untuk meningkatkan performa model *deep learning* untuk deteksi objek agar dapat dijalankan pada perangkat seperti *robot service* dengan komputasi minimal. Setelah diuji coba pada sistem deteksi objek YOLO[1], metode *knowledge distillation* dapat digunakan untuk meningkatkan akurasi dan metode *batchnorm fusion* dapat digunakan untuk mempercepat komputasi. Dari hasil pengujian menggunakan dataset VOC (*Visual Object Classes*) pada arsitektur YOLO (*You Only Look Once*) dengan *feature extractor* Mobilenet[2], metode *knowledge distillation* dapat meningkatkan akurasi sebesar sebesar 9.4% dari 0.3850 mAP menjadi 0.4215 mAP dan *batchnorm fusion* mempercepat waktu komputasi hingga 100.7% dari 8.3 FPS menjadi 16.66 FPS pada laptop dengan CPU i7. Metode *knowledge distillation* dapat meningkatkan akurasi model, dengan memperkecil ukuran model dan metode *batchnorm fusion* dapat mempercepat komputasi sehingga model dapat digunakan untuk berbagai aplikasi seperti sistem pencari objek pada *robot service* dengan komputasi minimal.

Kata Kunci: *Deep Learning, Object Detection, YOLO, Mobilenet, Knowledge Distillation, Batchnorm Fusion*

Halaman ini sengaja dikosongkan

ABSTRACT

Name : Billy
Title : *Knowledge Distillation on YOLO Object Detection System for Robot Service*
Advisors : 1. Muhtadin, ST., MT
2. Dr. Eko Mulyanto Yuniarno, ST., MT.

Deep learning already has good accuracy, but requires great computing. As a result, when applied to devices that have limited computing and memory capabilities such as robots in general, the computation time of the model becomes slow or not applicable. In order to optimize deep learning model, we need a model with smaller size but with the accuracy of a large model. This final project will be focusing on testing several methods to improve the performance of the deep learning model for object detection to make it capable of running on devices such as robot service with minimal computing. After being tested on the YOLO object detection system [1], knowledge distillation method can be used to increase accuracy and batchnorm fusion method can be used to increase computation speed. From the test results using VOC dataset on YOLO architecture with Mobilenet feature extractor [2], knowledge distillation method can increase accuracy by 9.4% from 0.3850 mAP to 0.4215 mAP and batchnorm fusion can speed up the computation time to 100.7% from 8.3 FPS to 16.66 FPS on laptop with CPU i7. The Knowledge Distillation method can increase model's accuracy, reducing model's size and batchnorm fusion method can speed up computing so that the model can be used for various applications such as object detection system on robot service with minimal computing.

Keywords: Deep Learning, Object Detection, YOLO, Mobilenet, Knowledge Distillation, Batchnorm Fusion

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji dan syukur kepada Tuhan yang Maha Esa atas berkat-Nya selama ini, penulis dapat menyelesaikan penelitian ini dengan judul ***Knowledge Distillation pada Sistem Pendeteksi Objek YOLO Untuk Robot Service*** .

Penelitian ini disusun dalam rangka pemenuhan bidang riset di Departemen Teknik Komputer, serta digunakan sebagai persyaratan menyelesaikan pendidikan S1. Penelitian ini dapat terselesaikan tidak lepas dari bantuan berbagai pihak. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Semua keluarga, mama dan saudara yang telah memberikan dukungan dalam penyelesaian buku penelitian ini.
2. Bapak Kepala Departemen Teknik Komputer ITS Dr. I Ketut Eddy Purnama, S.T., M.T. yang melancarkan jalannya proses penelitian selama ini.
3. Bapak Vice President Alfred Boediman, Bapak Director of Software Risman Adnan, Bapak Director of Service Innovation Andy W. Djiwandono dan Ibu Josephine Kusnadi di Samsung Research and Development Indonesia yang memberikan nasihat mengenai arah penelitian.
4. Bapak Muhtadin, ST., MT dan Bapak Dr. Eko Mulyanto Yuniarno, ST., MT. sebagai dosen pembimbing yang membimbing dan memberikan arahan agar penelitian berjalan dengan baik
5. Bapak Engineer Head of Part AI Junaidillah Fadil, Mas Muchlisin Adi Saputra ,Mas Shah Dehan Lazuardi dan Bapak Yudo Ekanata dari divisi AI SRIN yang membimbing dan memberikan arahan jika ada kesulitan dalam pelaksanaan penelitian
6. Bapak-ibu dosen pengajar Departemen Teknik Komputer ITS, atas pengajaran, bimbingan, serta perhatian yang diberikan kepada penulis selama ini.
7. Seluruh teman-teman Lab B401, Lab B201 dan Teknik Komputer yang saling membantu dalam melaksanakan penelitian ini

Tidak ada manusia yang lepas dari kekurangan, untuk itu penulis memohon segenap kritik dan saran yang membangun. Semoga

penelitian ini dapat memberikan manfaat bagi kita semua. Amin.

Surabaya, Juni 2019

Billy

DAFTAR ISI

Abstrak	i
Abstract	iii
KATA PENGANTAR	v
DAFTAR ISI	vii
DAFTAR GAMBAR	xi
DAFTAR TABEL	xiii
NOMENKLATUR	xv
1 PENDAHULUAN	1
1.1 Latar belakang	1
1.2 Rumusan Masalah	2
1.3 Tujuan	3
1.4 Batasan masalah	3
1.5 Sistematika Penulisan	3
2 TINJAUAN PUSTAKA	5
2.1 Deep Learning	5
2.2 CNN (Convolutional Neural Network)	6
2.2.1 Proses <i>Convolution</i>	7
2.2.2 Jenis-jenis <i>Convolution</i>	8
2.3 YOLO	10
2.3.1 <i>Output</i>	11
2.3.2 <i>Network Architecture</i>	15
2.3.3 <i>Loss</i>	20
2.4 Mobilenet	20
2.4.1 <i>Depthwise Separable Convolution</i>	21
2.4.2 <i>Computational Cost</i>	24
2.4.3 <i>Network Architecture</i>	26
2.5 <i>IoU (Intersection over Union)</i>	28
2.6 <i>mAP (mean Average Precision)</i>	28

2.7	<i>Knowledge Distillation</i>	30
2.8	Batch Normalization Fusion	34
2.9	Odometry	35
2.10	<i>Makeblock</i>	37
3	DESAIN DAN IMPLEMENTASI SISTEM	39
3.1	Desain Sistem	39
3.2	Work Flow	39
3.3	Proses <i>Training Data</i>	40
3.3.1	Dataset	40
3.3.2	Pengaturan <i>Training</i>	43
3.3.3	Penentuan nilai <i>Anchor</i>	44
3.3.4	Perhitungan <i>error (loss)</i>	46
3.4	Penerapan Optimalisasi Model	48
3.4.1	Desain Model	48
3.4.2	Penerapan <i>Knowledge Distillation</i> pada Object Detector	50
3.4.3	Penerapan <i>Batchnorm Fusion</i>	54
3.5	Sistem Deteksi Objek	57
3.6	Sistem Kontrol Robot	59
3.6.1	Pencarian Objek	59
3.6.2	<i>Multiprocessing</i>	60
3.6.3	Penerapan <i>Odometry</i>	61
3.6.4	Penerapan Komunikasi Serial	64
4	PENGUJIAN DAN ANALISA	67
4.1	Pengujian di Berbagai Jenis Model	67
4.2	Pengujian Dataset Objek untuk Manula	68
4.3	Pengujian di Berbagai Prosesor	70
4.4	Pengujian dengan Berbagai Ukuran <i>Input</i>	70
4.5	Pengujian Perbedaan Jumlah Skala	72
4.6	Pengujian <i>Knowledge Distillation</i> pada Deteksi Objek	75
4.7	Pengujian Pengaruh <i>Batchnorm Fusion</i>	78
4.8	Pengujian Deteksi <i>Real-Time</i> pada Robot	79
5	PENUTUP	83
5.1	Kesimpulan	83
5.2	Saran	84

DAFTAR PUSTAKA	87
LAMPIRAN	91
Biografi Penulis	93

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

2.1	Proses langkah - langkah <i>Convolution</i>	7
2.2	Fungsi Aktivasi ReLU	9
2.3	Proses <i>Max Pooling</i>	9
2.4	Proses deteksi dari <i>input</i> hingga <i>output</i> model YOLO v3	13
2.5	penempatan x,y,w,h di <i>Bounding Box</i>	14
2.6	Tiap Convolutional Layer terdiri dari <i>Convolutional</i> , <i>Batch Normalization</i> dan <i>ReLU</i>	15
2.7	Arsitektur Darknet-53	16
2.8	satu <i>Residual Layer YOLOV3</i>	17
2.9	Arsitektur deteksi objek YOLO dengan <i>Feature Extractor</i> Darknet 53	18
2.10	Arsitektur deteksi objek Tiny YOLO dengan <i>Feature Extractor</i> Darknet Reference	19
2.11	Alur <i>Depthwise Separable Filter</i>	21
2.12	<i>Standard Convolution</i>	22
2.13	<i>Separable Depthwise Convolution</i>	23
2.14	<i>Standard Convolution</i> dan <i>Depthwise Separable Convolution</i>	26
2.15	<i>MobileNet Architecture</i>	27
2.16	IoU <i>Intersecton over Union</i>	28
2.17	Perhitungan akurasi dengan mAP	30
2.18	<i>Student-Teacher Distillation</i>	31
2.19	Perbandingan distribusi nilai probabilitas tiap temperatur	33
2.20	Angka 9 memiliki kemiripan dengan angka 7 dan 4 dibandingkan dengan angka lain	33
2.21	Pengukuran posisi robot dengan <i>Odometry</i>	36
3.1	Gambaran Umum Sistem	39
3.2	Blok diagram training <i>neural network</i>	40
3.3	Contoh dataset dan label	42
3.4	Ukuran gambar berubah secara acak untuk deteksi berbagai skala	43
3.5	Penentuan kelompok dengan <i>k-means</i>	45

3.6	Perhitungan loss	46
3.7	Arsitektur deteksi objek YOLO modifikasi dengan <i>Feature Extractor</i> Mobilenet	49
3.8	Tiap <i>box</i> hanya untuk deteksi satu objek	50
3.9	Pendeteksian dengan ukuran box sesuai	51
3.10	Filter <i>Confidence</i> yang lebih rendah dari <i>threshold</i>	51
3.11	Filter box atau <i>anchor</i> yang mendeteksi objek yang sama	52
3.12	Diagram Alir <i>Batchnorm Fusion</i>	55
3.13	Diagram alir proses deteksi objek	57
3.14	Filter <i>Confidence</i>	58
3.15	Filter NMS (<i>Non-Max Suppression</i>)	59
3.16	Sekuensial Diagram Pencarian Objek	60
3.17	Blok Diagram Interaksi Antar Sistem	61
3.18	Ukuran Roda	62
3.19	Menentukan jumlah tick berdasarkan jarak	62
3.20	Alir Diagram Proses Perpindahan Robot	63
3.21	Kontrol Arah Pergerakan Robot	64
3.22	Serial Komunikasi antara Raspberry dan Auriga	65
4.1	Deteksi dengan model yang memiliki 3 skala	73
4.2	Deteksi dengan model yang memiliki 2 skala	73
4.3	Perbandingan jarak deteksi objek dengan menggunakan 3 skala pada robot	74
4.4	Perbandingan jarak deteksi objek dengan menggunakan 2 skala pada robot	74
4.5	Perbandingan performa model Mobilenet YOLO(21MB) sesudah dan sebelum distilasi	77
4.6	Percobaan pencarian objek sebelum <i>Batchnorm Fusion</i>	81
4.7	Percobaan pencarian objek setelah <i>Batchnorm Fusion</i>	81
1	Deteksi berbagai macam objek	91
2	Datasheet Makeblock Auriga	92
3	Robotn Pendeteksi Objek Makeblock Auriga	92

DAFTAR TABEL

3.1	Dataset <i>custom</i> untuk barang milik manula	41
3.2	Format pengiriman data dari Raspberry ke Arduino	65
3.3	Format pengiriman data dari Arduino ke Raspberry	66
4.1	Hasil perbandingan struktur model	68
4.2	Hasil perbandingan performa model	68
4.3	Hasil performa model dataset objek untuk manula (pretrain Imagenet)	69
4.4	Hasil performa model dataset objek untuk manula (pretrain VOC)	69
4.5	Perbandingan Spesifikasi	70
4.6	Hasil pengukuran FPS di berbagai prosesor	70
4.7	Hasil perbandingan berbagai ukuran <i>Input</i>	71
4.8	Hasil perbandingan jumlah deteksi objek dengan skala yang berbeda	72
4.9	Teacher dan Student untuk percobaan distilasi pada objek deteksi (Dataset VOC)	75
4.10	Hasil perbandingan distilasi pada objek deteksi (Dataset VOC)	75
4.11	Teacher dan Student untuk percobaan distilasi pada objek deteksi (Dataset Manula) (pretrain Imagenet)	76
4.12	Hasil perbandingan distilasi pada objek deteksi (Dataset Manula) (pretrain Imagenet)	76
4.13	Teacher dan Student untuk percobaan distilasi pada objek deteksi (Dataset Manula) (pretrain VOC)	77
4.14	Hasil perbandingan distilasi pada objek deteksi (Dataset Manula) (pretrain VOC)	77
4.15	Hasil perbandingan model sebelum dan sesudah Batchnorm Fusion	78
4.16	Hasil perbandingan performa sebelum dan sesudah Batchnorm Fusion	79
4.17	Perbandingan Spesifikasi	80
4.18	Hasil perbandingan FPS untuk performa robot ukuran gambar 224 model mobilenet YOLO(21MB)	80

Halaman ini sengaja dikosongkan

NOMENKLATUR

q_i	: Nilai logits setelah diolah
z_i	: Nilai mentahan output model (logits)
S	: Ukuran <i>Feature Map</i>
B	: Jumlah <i>Bounding Box</i>
C	: Jumlah <i>Class</i>
t_x	: Nilai x mentahan output model
t_y	: Nilai y mentahan output model
c_x	: Jarak <i>box</i> x dari pojok kiri atas
c_y	: Jarak <i>box</i> y dari pojok kiri atas
b_x	: Nilai x output model setelah diproses
b_y	: Nilai y output model setelah diproses
e_w^t	: Nilai w mentahan output model
e_h^t	: Nilai h mentahan output model
p_w	: Ukuran anchor w
p_h	: Ukuran anchor h
b_w	: Nilai w output model setelah diproses
b_h	: Nilai h output model setelah diproses
t_o	: Nilai probabilitas mentahan output model
x_i	: Nilai x mentahan output
\hat{x}_i	: Nilai x label
y_i	: Nilai y mentahan output
\hat{y}_i	: Nilai y label
C_i	: Nilai <i>Confidence</i> mentahan output
\hat{C}_i	: Nilai <i>Confidence</i> label
$p_i(c)$: Nilai probabilitas <i>Class</i> mentahan output
$p_i(\hat{c})$: Nilai probabilitas <i>Class</i> label

D_K	: Ukuran input <i>feature map</i>
D_F	: Ukuran output <i>feature map</i>
M	: Ukuran input <i>channel (depth)</i>
N	: Ukuran output <i>channel (depth)</i>
T	: Nilai <i>hyperparameter</i> temperatur untuk distilasi
α	: <i>hyperparameter</i> alpha untuk distilasi
q_s^T	: Nilai logits <i>student</i> setelah diolah dengan nilai temperatur
q_T^T	: Nilai logits <i>teacher</i> setelah diolah dengan nilai temperatur
q_s	: Nilai logits <i>student</i> setelah diolah
\mathcal{Y}_{true}	: Nilai label
Y	: Nilai <i>output convolutional layer</i>
W	: Nilai <i>Weight</i>
X	: Nilai <i>input convolutional layer</i>
B	: Nilai <i>Bias</i>
γ	: Parameter yang bisa dipelajari
β	: Parameter yang bisa dipelajari
$RunMean$: Nilai rata-rata X
$RunStd$: (Standard Deviation) Jarak penyebaran data relatif pada rata-rata
eps	: nilai konstan
$\Delta Tick$: Selisih <i>Tick</i>
$Tick'$: Total jumlah nilai <i>encoder</i> sebelumnya
$Tick$: Total jumlah nilai <i>encoder</i> sekarang
D_R	: Jarak roda kiri dari titik awal
D_L	: Jarak roda kanan dari titik awal
$FullTick$: Total jumlah <i>encoder</i> roda satu putaran
D_c	: Jarak robot dari titik awal
L	: Jarak roda kanan dan roda kiri
θ	: sudut awal perpindahan robot
θ'	: sudut baru perpindahan robot
X_r	: posisi awal koordinat X robot
Y_r	: posisi awal koordinat Y robot
X'_r	: posisi baru koordinat X robot
Y'_r	: posisi baru koordinat Y robot

BAB 1

PENDAHULUAN

Penelitian ini di latar belakang oleh berbagai kondisi yang menjadi acuan. Selain itu juga terdapat beberapa permasalahan yang akan dijawab sebagai luaran dari penelitian.

1.1 Latar belakang

Perkembangan *deep learning* mempermudah menyelesaikan dari permasalahan-permasalahan yang berhubungan dengan visual seperti klasifikasi objek dan deteksi objek. Ada berbagai macam arsitektur *deep learning* seperti resnet [3], xception[4], mobilenet [2], squeezeNet [5] untuk klasifikasi objek dan YOLO (*You Only Look Once*) [6] , SSD (*Single Shot Detector*) [7] untuk deteksi objek.

Robot service sedang dikembangkan untuk mengatasi berbagai macam aspek yang berhubungan dengan aktivitas kegiatan sehari-hari untuk menciptakan layanan yang efektif [8]. Salah satu aspek yang dapat dikembangkan adalah sistem pencarian objek dengan *robot service*. Sistem ini dapat digunakan manula untuk membantu mencari barang. Faktor yang jadi pertimbangan adalah mengingat penduduk lansia di Indonesia meningkat sekitar dua kali lipat(1971-2017), yakni menjadi 8,97 persen (23,4 juta)[9] dan seiring bertambahnya usia, otak akan mengalami gangguan daya ingat[10].

Untuk dapat mengembangkan objek deteksi pada *robot service* diperlukan jenis model yang sesuai. Model *deep learning* seperti resnet [3], xception[4] untuk klasifikasi dan YOLO [6], SSD [7] untuk deteksi adalah hasil perkembangan dengan fokus untuk meningkatkan akurasi. Untuk mendapatkan akurasi yang optimal diperlukan parameter yang banyak juga, akibatnya ukuran model dengan akurasi yang bagus cenderung berukuran besar. Model berukuran besar akan memerlukan komputasi dan memori yang besar juga sehingga saat diterapkan di perangkat yang memiliki kemampuan komputasi dan memori yang terbatas seperti robot, waktu komputasi model menjadi lambat atau tidak bisa dijalankan. Sedangkan model seperti mobilenet [2], squeezeNet [5] yang didesain untuk dijalankan di perangkat dengan komputasi dan memori yang terbatas akurasinya lebih rendah dibandingkan model yang beru-

ukuran besar.

Untuk mengatasi permasalahan tersebut diperlukan modifikasi pada model sistem deteksi objek yang sudah ada. Sistem deteksi objek yang terbaru saat ini adalah versi terakhir YOLO yaitu YOLO V3 [1]. YOLO V3 adalah pengembangan dari YOLO dengan akurasi yang telah ditingkatkan. Lalu untuk meningkatkan performa model ukuran kecil telah dikembangkan berbagai metode untuk memperkecil ukuran dan meningkatkan performa model. Metode - metode yang dimaksud adalah *pruning* dan *sharing*, *low-rank factorization*, *quantization*, *knowledge distillation* [11].

Kekurangan dari YOLO adalah ukurannya masih terlalu besar untuk dapat diterapkan di *robot service*. YOLO sendiri memiliki model berukuran kecil yang disebut Tiny YOLO, namun akurasi-nya masih rendah. Untuk itu bisa model YOLO perlu dimodifikasi dengan menggunakan *feature extractor* yang sesuai seperti *mobilenet*, untuk mendapatkan akurasi yang hampir sama dengan aslinya namun ukurannya lebih kecil. Saat ini metode untuk memperkecil ukuran dan meningkatkan performa model masih difokuskan untuk permasalahan klasifikasi saja sedangkan untuk masalah deteksi objek masih jarang dilakukan. Klasifikasi dan deteksi memiliki cara kerja yang berbeda sehingga metode-metode yang sudah ada saat ini perlu dimodifikas terlebih dahulu agar dapat diterapkan untuk deteksi objek.

Sehingga pada tugas akhir ini akan dilakukan uji coba beberapa metode untuk meningkatkan performa model *deep learning* untuk deteksi objek agar dapat dijalankan pada perangkat seperti *robot service* dengan komputasi minimal. Modifikasi yang akan diterapkan adalah metode *knowledge distillation* [12] untuk meningkatkan akurasi dan *batchnorm fusion* [13] untuk mempercepat komputasi.

1.2 Rumusan Masalah

1. Perlunya deteksi objek secara real-time untuk robot service untuk membantu pendeteksian objek.
2. Robot service memiliki memori dan kemampuan komputasi yang terbatas seperti *single board computer*.
3. Model deep learning dengan akurasi optimal tidak dapat dijalankan pada robot service karena memiliki memori dan komputasi yang terlalu besar melebihi kapasitas RAM yang dimi-

liki robot service.

4. Model deep learning dengan ukuran kecil dapat dijalankan pada robot service pendeteksi objek seperti *single board computer* namun akurasi rendahnya dibandingkan dengan model deep learning dengan ukuran besar.

1.3 Tujuan

1. Membuat sistem pendeteksi objek secara real-time di Robot Service yang bergerak.
2. Membuat sistem Robot Service yang dapat digunakan untuk membantu mencari barang.
3. Meningkatkan akurasi pengenalan objek model Deep Learning yang dapat dijalankan di Robot Service.
4. Meningkatkan kecepatan komputasi model Deep Learning untuk membuat respon Robot Service yang tepat waktu.

1.4 Batasan masalah

Untuk memfokuskan permasalahan yang diangkat maka dilakukan pembatasan masalah. Batasan-batasan masalah tersebut diantaranya adalah:

1. Objek yang dikenali hanya barang yang bisa terlihat jelas oleh kamera dan tidak terhalang apapun.
2. Kelas Objek yang terdaftar hanya dapat mengenali objek secara umum dan tidak dapat membedakan detail objek di kelas yang sama
3. Pengujian difokuskan untuk mengamati performa CPU untuk proses pendeteksian (*inference*)
4. Rute pencarian Robot sudah ditentukan sebelumnya dan hanya berpindah saat mendeteksi barang lalu kembali ke rute awal.
5. Tidak ada halangan pada rute yang dilalui Robot dan hanya beroperasi pada permukaan datar.

1.5 Sistematika Penulisan

Laporan penelitian Tugas akhir ini tersusun dalam sistematika dan terstruktur sehingga mudah dipahami dan dipelajari oleh pembaca maupun seseorang yang ingin melanjutkan penelitian ini. Alur sistematika penulisan laporan penelitian ini yaitu :

1. BAB I Pendahuluan
Bab ini berisi uraian tentang latar belakang permasalahan, penegasan dan alasan pemilihan judul, sistematika laporan, tujuan dan metodologi penelitian.
2. BAB II Tinjauan Pustaka
Pada bab ini berisi tentang uraian secara sistematis teori-teori yang berhubungan dengan permasalahan yang dibahas pada penelitian ini. Teori-teori ini digunakan sebagai dasar dalam penelitian, yaitu *convolutional neural network*, model arsitektur deteksi objek, *non-maximum suppression*, perhitungan mAP, *knowledge distillation*, *batchNorm fusion*, *odometry* dan teori-teori penunjang lainnya.
3. BAB III DESAIN DAN IMPLEMENTASI SISTEM Bab ini berisi tentang penjelasan-penjelasan terkait desain sistem yang akan dirancang mulai dari alur proses kerja hingga interaksi antar sistem dan penjelasan-penjelasan terkait eksperimen yang akan dilakukan dan langkah-langkah percobaan dimulai dari *training data*, pendeteksian objek (*inference*), optimalisasi performa model, hingga pergerakan robot. Desain akan dijelaskan menggunakan berbagai macam diagram seperti *work flow diagram*, *activity diagram*, *process diagram* dan *sequence diagram*.
4. BAB IV Pengujian dan Analisa
Bab ini menjelaskan tentang pengujian eksperimen yang dilakukan terhadap hasil performa model dengan parameter yang berbeda-beda, hasil uji coba distilasi, hasil uji coba *batchnorm fusion*, dan deteksi objek pada robot. Semua uji coba juga akan dijelaskan analisisnya.
5. BAB V Penutup
Bab ini merupakan penutup yang berisi kesimpulan yang diambil dari penelitian dan pengujian yang telah dilakukan. Saran dan kritik yang membangun untuk mengembangkan lebih lanjut juga dituliskan pada bab ini.

BAB 2

TINJAUAN PUSTAKA

Demi mendukung penelitian ini, dibutuhkan beberapa teori penunjang sebagai bahan acuan dan referensi. Dengan demikian penelitian ini menjadi lebih terarah.

2.1 Deep Learning

Deep Learning adalah sekumpulan teknik yang diterapkan di *neural network*. Sedangkan *neural network* sendiri adalah paradigma pemrograman yang membuat komputer belajar dari data observasi [14].

Masalah utama dari *machine learning* pada umumnya adalah banyaknya faktor yang harus diolah untuk pengambilan keputusan dan solusi yang sama tidak bisa diterapkan ke kasus yang berbeda. Adanya *deep learning* dapat menyelesaikan permasalahan tersebut dengan membuat algoritma yang dapat menentukan sendiri fitur yang harus diambil dengan mempelajari data yang sudah ada [15].

Neural network dapat digunakan sebagai solusi untuk masalah di bidang *image recognition*, *speech recognition* dan *natural language processing*. Permasalahan umum yang dapat dipecahkan oleh Deep Learning adalah *classification* dan *regression*. *Classification* adalah memprediksi permasalahan dimana outputnya adalah bilangan diskrit sedangkan *regression* adalah memprediksi permasalahan dimana outputnya adalah bilangan kontinu.

Model *deep learning* utama dalam *Neural Network* adalah MLP (*Multilayer Perceptron*) atau bisa juga disebut *feedforward neural networks*. MLP adalah sekumpulan layer yang dimana tiap layer memiliki fungsi untuk membantu algoritma menentukan output yang sesuai [15]. Namun MLP tidak bisa langsung diterapkan begitu saja tiap fungsi pada layer perlu disesuaikan dulu tiap variabelnya agar dapat berjalan dengan baik. Untuk itu perlu dilakukan proses *learning* terlebih dahulu dengan menggunakan algoritma yang disebut *backpropagation*.

Backpropagation pertama kali diperkenalkan pada tahun 1970, namun baru terlihat manfaatnya pada tahun 1986 dari paper yang

dimiliki David Rumelhart, Geoffrey Hinton, and Ronald Williams [14]. Kecepatan algoritma *backpropagation* adalah alasan mengapa *neural network* dapat diterapkan untuk memecahkan masalah saat ini.

Berikut Algoritma *backpropagation*:

1. Input : Masukkan nilai data yang ingin dipelajari.
2. Feedforward : Hitung nilai semua fungsi untuk mendapatkan output
3. Output error: Hitung selisih error hasil nilai output dan nilai target
4. Backpropagate: Ubah nilai variabel tiap fungsi dengan menyesuaikan dengan nilai error yang sudah diperoleh
5. Output : Nilai Output baru setelah backpropagate
6. Ulangi Proses hingga hasil Output sudah mendekati target

Proses *training* pada *neural network* bukan proses yang mudah. Proses *training* dilakukan dengan tujuan meminimalisir *error*. Fungsi yang digunakan untuk meminimalisir error disebut dengan *loss function*. Salah satu *loss function* yang sering diterapkan adalah *cross-entropy* seperti persamaan (??). Saat berusaha meminimalisir error dapat mengakibatkan model memperoleh error yang sedikit pada *training data* , namun mendapat error yang tinggi pada *test data*. Permasalahan ini disebut dengan istilah *overfitting*. Banyak strategi yang telah dibuat untuk mengatasi permasalahan ini, strategi ini disebut dengan istilah *Regularization* [15].

Regularization yang sering digunakan saat ini ada L1, L2 dan dropout. L1 (*list absolute deviations*) dan L2 (*least squares error*) meminimalisir *overfitting* dengan menambahkan nilai absolut pada *loss function*. *Dropout* berbeda dengan yang lain tidak memodifikasi *loss function* namun memodifikasi jaringan dengan cara menonaktifkan layer secara *random*, tujuannya agar memaksa *network* untuk mencari pola baru.

2.2 CNN (Convolutional Neural Network)

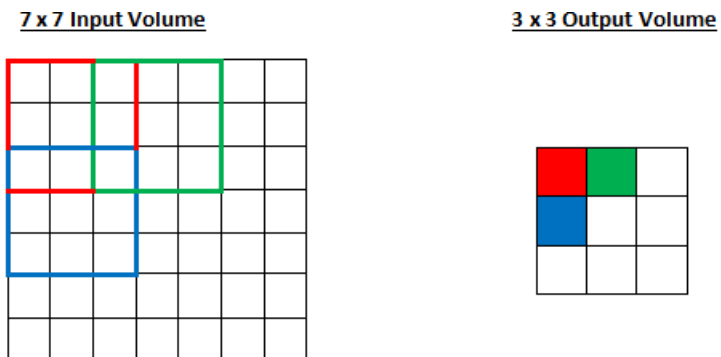
CNN (*Convolutional neural network*) adalah *multi layer neural network* yang di desain untuk mengenali pola visual langsung dari piksel gambar dengan pengolahan minimal. Sama seperti *neural network* lain, CNN dilatih dengan menggunakan algoritma *backp-*

ropagation , dimana yang membedakan adalah arsitekturnya [16]. CNN terinspirasi oleh penelitian yang dilakukan oleh D. H. Hubel and T. N. Wiesel. Mereka mengajukan penjelasan dimana mamalia melihat sekitar mereka menggunakan lapisan arsitektur di otak. Teori ini adalah teori yang menginspirasi insinyur untuk membuat pola yang serupa dalam Visi Komputer [17].

2.2.1 Proses *Convolution*

Saat proses konvolusi ada beberapa parameter yang harus ditentukan terlebih dahulu yaitu *filter*, *stride* dan *padding*. *Filter* adalah matrix yang digunakan untuk mengambil fitur-fitur dari *matrix pixel*. Semakin banyak *filter* maka akan semakin banyak informasi yang diperoleh. *Stride* adalah ukuran langkah pergeseran *filter* untuk melakukan konvolusi. Sedangkan *padding* adalah border yang berisi nilai tambahan yang mengelilingi *matrix* gambar. Salah satu contoh padding yang sering digunakan adalah *zero padding* dimana border *matrix* diberi nilai tambahan nilai nol. Tujuan penambahan *padding* adalah untuk konvolusi yang ingin menyimpan ukuran volume matrix awal, karena tanpa padding ukuran *output matrix* akan mengecil.

Gambar 2.1 menunjukkan proses konvolusi. Untuk menentukan ukuran volume output hasil konvolusi dapat menggunakan persamaan (2.1):



Gambar 2.1: Proses langkah - langkah *Convolution*

$$O = \frac{w - k + 2p}{s} + 1 \quad (2.1)$$

Dengan,

O = Ukuran volume output matrix

w = Ukuran volume input matrix

k = Ukuran filter matrix

p = Ukuran padding

s = nilai stride (langkah pergeseran filter)

2.2.2 Jenis-jenis *Convolution*

Pada umumnya *convolutional neural network* terdiri dari dua tahap yaitu *feature learning* dan *classification*. *feature learning* memiliki beberapa tingkat arsitektur yang terdiri dari beberapa layer. Tiap tingkat umumnya tersusun dari *convolutional layer*, *non-linearity layer* dan *feature pooling layer* [16] sedangkan *classification* terdiri dari *flatten*, *fully connected layer* dan *activation layer*.

Convolutional layer mengambil fitur penting dari gambar dan membuat ukuran menjadi lebih kecil. *convolution* dilakukan dengan cara menggeser *filter* ke tiap piksel gambar untuk mendapatkan *feature map*.

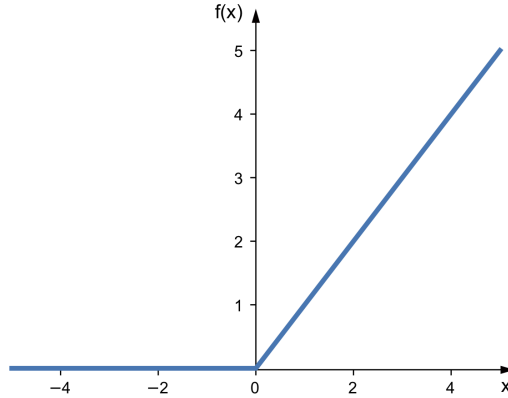
ReLU (*Non-linearity Layer / rectified linear unit*) adalah *activation function* yang mengubah bilangan negatif setelah *convolution* menjadi 0. Tujuannya untuk mencegah error saat operasi matematika setelah operasi *convolution* dan menyesuaikan ke data asli yang hanya dapat bernilai dari 0 sampai 255. ReLU dapat diperoleh dengan menggunakan persamaan (2.2). Fungsi ReLU dalam bentuk grafik dapat dilihat pada gambar 2.2.

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2.2)$$

Dengan,

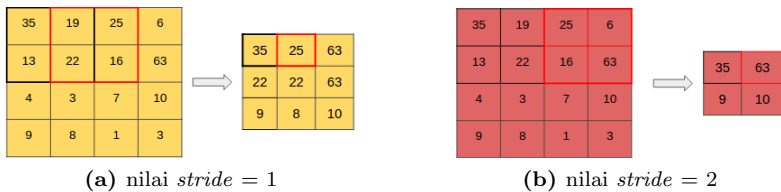
$f(x)$ = nilai setelah aktivasi ReLU

$x = \text{nilai awal}$



Gambar 2.2: Fungsi Aktivasi ReLU

Feature pooling layer digunakan untuk mengurangi dimensi (panjang x lebar) gambar yang terlalu besar. *Pooling* tidak mempengaruhi kedalaman (RGB) dari dimensi gambar. Ada beberapa macam pooling salah satu contoh adalah *max pooling* dan *average pooling*. Cara kerja *max pooling* adalah mengambil pixel yang memiliki nilai terbesar di tiap filter, Sedangkan *average pooling* mengambil nilai rata-rata dari filter. Tujuan dari *pooling layer* adalah agar data yang tersimpan memiliki nilai fitur yang paling dominan diantara kelompok matriks dan memperkecil ukuran matriks. Visualisasi *max pooling* adalah seperti pada gambar 2.3



Gambar 2.3: Proses *Max Pooling*

Setelah proses pengambilan feature , selanjutnya tahap *classification*. Tahap pertama adalah *flatten*, pada proses ini hasil keluaran *feature map* akan disederhanakan menjadi satu dimensi. Setelah itu *fully connected layer* akan memproses *feature* untuk mendapatkan hasil klasifikasi yang diinginkan. Nilai yang keluar dari *fully connected layer* masih berupa nilai mentah (*logits*) yang perlu diproses terlebih dahulu. Untuk mendapatkan nilai distribusi dari output mentah tadi dapat dilakukan dengan menggunakan *activation layer*.

Activation layer digunakan untuk menormalisasi hasil output untuk mendapatkan nilai probabilitas. Jenis *activation layer* yang sering digunakan adalah *softmax* dan *sigmoid*. *softmax* mengubah nilai logits dan mengubahnya menjadi nilai diantara 0 sampai 1 dan jika dijumlah total semua probabilitasnya nilainya adalah 1. Sedangkan *sigmoid* mengubah nilai logits dan mengubahnya menjadi nilai diantara 0 sampai 1 namun jumlah dari semua probabilitas tidak harus bernilai 1. *softmax* cocok digunakan untuk probabilitas dengan banyak kelas sedangkan *sigmoid* cocok untuk probabilitas dengan dua kelas saja. Fungsi *softmax* dapat diperoleh menggunakan persamaan (2.3) dan *sigmoid* dapat diperoleh dengan persamaan (2.4).

$$q_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \quad (2.3)$$

$$q_i = \frac{1}{1 + \exp(-z_i)} \quad (2.4)$$

Dengan,

q_i = Nilai logits setelah diolah

z_i = Nilai mentahan output model (logits)

2.3 YOLO

YOLO(*You Only Look Once*) merupakan pendekatan baru dari sistem deteksi objek. *YOLO* melakukan deteksi objek dengan

memanfaatkan *regression* untuk memisahkan lokasi *bounding box* dan menentukan *class probabilities*. Satu *neural network* digunakan memprediksi *bounding box* dan *class probabilities* langsung dari gambar hanya dengan sekali evaluasi[6].

Saat pertama kali diperkenalkan metode ini berbeda dari teknik *object detection* yang sudah pernah diterapkan sebelumnya, metode seperti DPM (*Deformable Part Model*) dan R-CNN (*Regions with CNN*) menerapkan *classifiers* yang difungsikan sebagai *object detection*. Metode yang digunakan YOLO membuat pendeteksian objek menjadi lebih cepat karena menggabungkan *classification* dan *regression*. Metode yang hampir sama adalah SSD (*Single Shot Detector*) yang menerapkan pendeteksian pada beberapa skala sehingga akurasi lebih baik daripada YOLO. Namun Metode YOLO juga mengalami perkembangan ke YOLO v2 yang mengalami peningkatan performa hingga YOLO v3 yang menerapkan metode-metode objek deteksi yang sudah berhasil sebelumnya.

2.3.1 Output

YOLO menerapkan *object detection* dalam satu *neural network*, Cara kerjanya adalah membagi gambar jadi $S \times S$ *grid*. Tiap *grid cell* akan mendeteksi objek yang ada di wilayahnya [6]. Bentuk *output YOLO* mengalami perubahan dari tiap versinya dengan tujuan membuat akurasi *YOLO* jadi lebih optimal. Versi awal *YOLO* tiap *grid* mendeteksi 2 *bounding box*, YOLO v2 mendeteksi 5 *bounding box* seperti persamaan (2.5). Untuk pengembangan terakhir YOLO v3 memiliki 3 output dengan skala yang berbeda-beda untuk mendeteksi objek dalam berbagai ukuran. Tiap skala memiliki 3 *bounding box* [1]. *YOLOV3* memiliki output yang agak berbeda dari versi sebelumnya seperti pada persamaan (2.6).

$$YOLO\ v2\ box\ shape = (S \times S \times (B \times 5 + C)) \quad (2.5)$$

Contoh : 2 bounding box dan 20 kelas objek
 $(7 \times 7 \times (2 \times 5 + 20))$
 $(7 \times 7 \times 30)$

$$YOLO\ v3\ box\ shape = (S \times S \times B \times (5 + C)) \quad (2.6)$$

Contoh : 3 bounding box dan 80 kelas objek di 3 skala

$$(13 \times 13 \times 3 \times 85)$$

$$(26 \times 26 \times 3 \times 85)$$

$$(52 \times 52 \times 3 \times 85)$$

Dengan,

S = Ukuran *Feature Map*

B = Jumlah *Bounding Box*

C = Jumlah *Class*

5 = $x, y, w, h, confidence$

Tiap grid terdiri dari 5 prediksi yaitu: x, y, w, h dan *confidence* ditambah dengan C (jumlah *class probabilities*). nilai (x,y) koordinat mempresentasikan titik pusat objek berdasarkan *grid cell*. nilai (w,h) adalah panjang dan lebar yang relatif pada ukuran gambar. Terakhir nilai *confidence* yang memprediksi IOU (*Intersection of Union*) antara kotak prediksi dan kotak yang asli. Untuk mendeteksi ada atau tidaknya objek pada grid cell nilai *confidence* di kalikan dengan probabilitas objek seperti pada persamaan (2.7). Jika tidak ada objek maka nilai *confidence* akan mendekati 0 [6]. Gambar 2.4 menunjukkan proses deteksi dari *input* hingga *output* arsitektur YOLO v3.

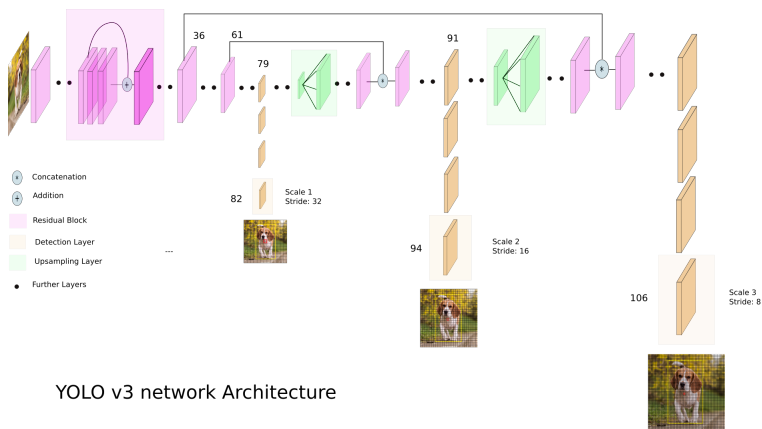
$$confidence = Pr(Object) * IOU_{truth}^{pred} \quad (2.7)$$

Dengan,

confidence = nilai tingkat ada atau tidaknya objek

$Pr(Object)$ = nilai hasil prediksi

IOU_{truth}^{pred} = nilai perbandingan posisi kotak asli dan prediksi



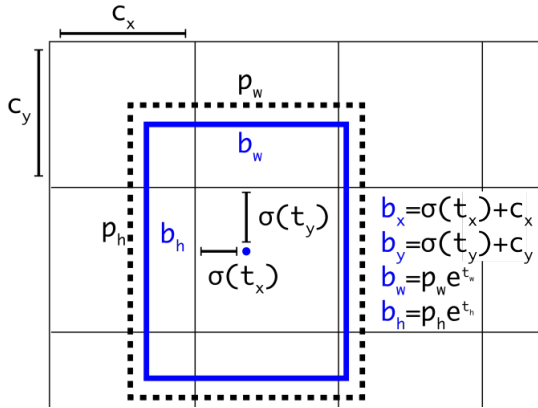
Gambar 2.4: Proses deteksi dari *input* hingga *output* model YOLO v3

Hasil prediksi yang diperoleh hanya berdasarkan informasi spasial dari *bounding box* yang mendeteksi, untuk mendapatkan informasi spasial dari keseluruhan gambar maka nilai dari *output layer* harus diolah dengan persamaan (2.8). Nilai x, y, w, h yang diperoleh kemudian akan diterapkan seperti pada gambar 2.5. Sedangkan nilai *class* probabilitas dan *confidence* digunakan untuk menentukan jenis objek. nilai (cx, cy) berasal dari jarak grid dari pojok kiri atas gambar. Sedangkan nilai p_w, p_h ukuran *bounding box* yang sudah ditentukan sebelumnya. Ukuran *bounding box* ditentukan dengan memisah ukuran label kotak menjadi beberapa kelompok. Dikarenakan ada 3 skala dan tiap skala ada 3 anchor, jadi jumlahnya 9 *anchor*. Pengelompokan diatur dengan algoritma k-means. Saat menggunakan dataset COCO (*Common Object in Context*) dataset 9 ukuran yang digunakan adalah: (10×13) , (16×30) , (33×23) , (30×61) , (62×45) , (59×119) , (116×90) , (156×198) , (373×326) [1].

$$\begin{aligned}
 b_x &= \sigma(t_x) + c_x \\
 b_y &= \sigma(t_y) + c_y \\
 b_w &= p_w e_w^t \\
 b_h &= p_h e_h^t
 \end{aligned} \tag{2.8}$$

Dengan,

- b_x = Nilai x output model setelah diproses
- b_y = Nilai y output model setelah diproses
- b_w = Nilai w output model setelah diproses
- b_h = Nilai h output model setelah diproses
- t_x = Nilai x mentahan output model
- t_y = Nilai y mentahan output model
- c_x = Jarak *box* x dari pojok kiri atas
- c_y = Jarak *box* y dari pojok kiri atas
- e_w^t = Nilai w mentahan output model
- e_h^t = Nilai h mentahan output model
- p_w = Ukuran anchor w
- p_h = Ukuran anchor h



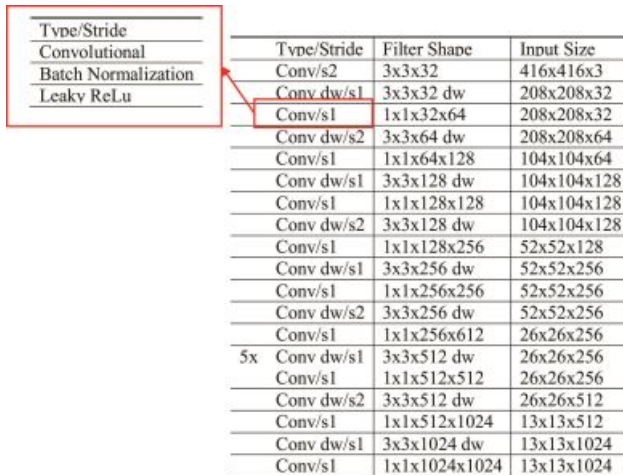
Gambar 2.5: penempatan x,y,w,h di *Bounding Box*

Setelah *output* diproses dan difilter sesuai dengan nilai *confidence* maka akan didapatkan banyak sekali *bounding box* yang terdapat objek. Namun ada kemungkinan satu objek yang sama dideteksi oleh beberapa *bounding box*. Untuk itu kotak dengan prediksi yang sama perlu disingkirkan dengan menggunakan NMS (*Non-Max Suppression*). Cara kerja NMS adalah pertama memilih objek dengan *confidence score* paling tinggi, lalu menyingkirkan semua kotak lain yang memiliki *IoU* (*Intersection Over Union*) dibawah

0.5 dari kotak yang terpilih.

2.3.2 Network Architecture

Objek Deteksi terdiri dari komponen utama *feature extractor* dan *object classifier* [18]. *Feature extractor* digunakan untuk memproses parameter untuk mendapatkan fitur sedangkan *object classifier* untuk memproses nilai menjadi output. Arsitektur awal YOLO terinspirasi dari GoogleNet sebagai *feature extractor* [6], YOLOV2 menggunakan arsitektur disebut *Darknet-19* [19]. Kemudian Pada YOLOV3 arsitektur *feature extractor* yang digunakan dikembangkan lagi menjadi *Darknet-53*. Sistem Deteksi YOLO sendiri memiliki 2 jenis yaitu YOLO dan Tiny YOLO.



Type/Stride	Filter Shape	Input Size
Conv/s2	3x3x32	416x416x3
Conv dw/s1	3x3x32 dw	208x208x32
Conv/s1	1x1x32x64	208x208x32
Conv dw/s2	3x3x64 dw	208x208x64
Conv/s1	1x1x64x128	104x104x64
Conv dw/s1	3x3x128 dw	104x104x128
Conv/s1	1x1x128x128	104x104x128
Conv dw/s2	3x3x128 dw	104x104x128
Conv/s1	1x1x128x256	52x52x128
Conv dw/s1	3x3x256 dw	52x52x256
Conv/s1	1x1x256x256	52x52x256
Conv dw/s2	3x3x256 dw	52x52x256
Conv/s1	1x1x256x612	26x26x256
5x Conv dw/s1	3x3x512 dw	26x26x256
Conv/s1	1x1x512x512	26x26x256
Conv dw/s2	3x3x512 dw	26x26x512
Conv/s1	1x1x512x1024	13x13x512
Conv dw/s1	3x3x1024 dw	13x13x1024
Conv/s1	1x1x1024x1024	13x13x1024

Gambar 2.6: Tiap Convolutional Layer terdiri dari *Convolutional*, *Batch Normalization* dan *ReLU*

Gambar 2.6 menunjukkan inti dari tiap *convolutional layer*. Arsitektur dasar dari sistem deteksi objek adalah *Convolutional Layer*. Tiap Convolutional layer terdiri dari *Convolutional layer*, *Batch Normalization* dan *Leaky ReLU*. Susunan ini hampir digunakan di semua arsitektur baru yang sering digunakan saat ini. *Batch Normalization* digunakan untuk normalisasi hasil konvolusi dan *ReLU* menghilangkan bilangan negatif. *Feature Extractor* kemudian akan

memiliki beberapa *layer* yang akan dibuat bercabang untuk disatukan dengan *Object Classifier* untuk mendapatkan ukuran dimensi *layer* dan parameter yang sesuai. Untuk menyatukan layer digunakan *Upsampling layer* untuk memperbesar ukuran *feature map* lalu *Contentenate layer* untuk menyatukannya.

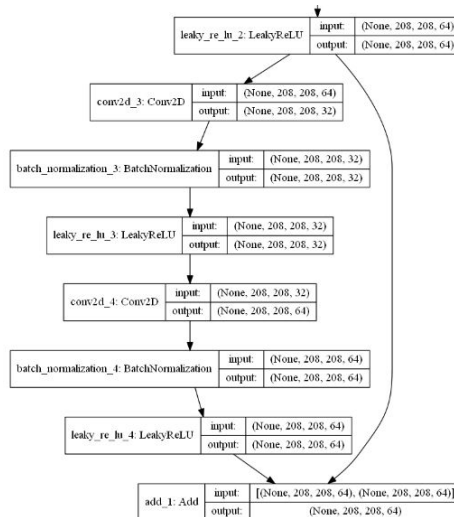
YOLO menggunakan *feature extractor* yang disebut *Darknet 53*. Arsitektur ini memiliki jumlah layer yang banyak dan difokuskan untuk mendapatkan akurasi yang lebih tinggi dari model pendahulunya *Darknet 19*. Susunan utama dari arsitektur ini terdiri dari susunan *convolutional layer* yang disusun secara *residual layer*. Susunan ini bekerja dengan memperkecil ukuran *feature map* dan memperbesar parameter. Untuk lebih jelasnya arsitektur darknet bisa dilihat pada gambar 2.7.

	Type	Filters	Size	Output
	Convolutional	32	3×3	256×256
	Convolutional	64	$3 \times 3 / 2$	128×128
1x	Convolutional	32	1×1	
	Convolutional	64	3×3	
	Residual			128×128
2x	Convolutional	128	$3 \times 3 / 2$	64×64
	Convolutional	64	1×1	
	Convolutional	128	3×3	
	Residual			64×64
8x	Convolutional	256	$3 \times 3 / 2$	32×32
	Convolutional	128	1×1	
	Convolutional	256	3×3	
	Residual			32×32
8x	Convolutional	512	$3 \times 3 / 2$	16×16
	Convolutional	256	1×1	
	Convolutional	512	3×3	
	Residual			16×16
4x	Convolutional	1024	$3 \times 3 / 2$	8×8
	Convolutional	512	1×1	
	Convolutional	1024	3×3	
	Residual			8×8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Gambar 2.7: Arsitektur Darknet-53

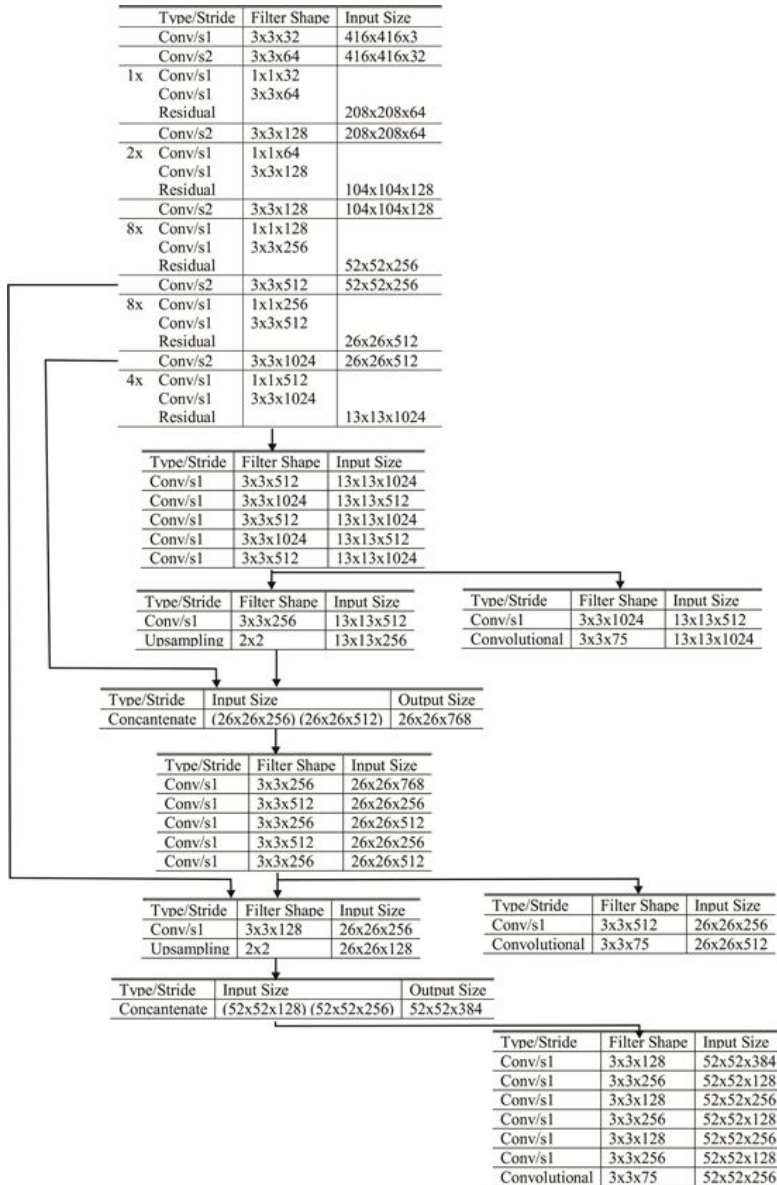
Meskipun arsitektur *Darknet 53* memiliki jumlah layer yang banyak untuk dapat menyimpan fitur yang banyak, Arsitektur ini juga memiliki kelemahan. Karena proses konvolusi digunakan untuk mengurangi data agar proses klasifikasi lebih ringan, maka nilai

parameter akan terus mengecil. Hal ini dapat membuat fitur yang ingin diperoleh hilang karena nilainya sudah terlalu kecil atau mendekati nol. Untuk itu, arsitektur disusun secara *residual layer* untuk mengatasi *vanishing gradient* agar informasi data yang ingin diperoleh tidak hilang [3]. *Residual layer* membuat percabangan pada layer, *input* pertama mentransfer nilai ke tahapan *layer* sedangkan *input* kedua langsung ke *layer* akhir tiap tahap. Dengan metode ini meskipun jika konvolusi membuat fitur yang diperoleh hilang, nilai tidak akan hilang karena masih ada nilai yang teruskan dari *input* kedua. Susunan *Residual layer* seperti gambar 2.8.



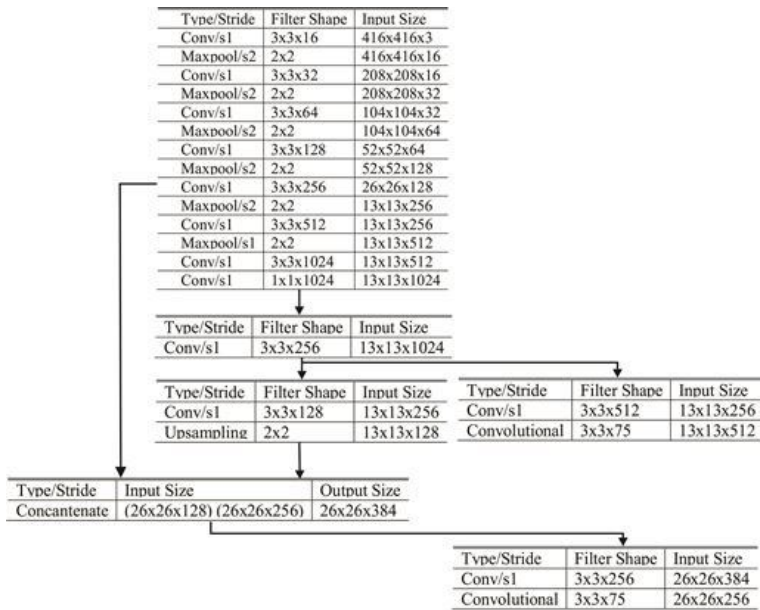
Gambar 2.8: satu *Residual Layer* YOLOV3

Desain Arsitektur YOLO pada gambar 2.9 terdiri dari satu input dan tiga output. Tiap output dari model digunakan untuk mendeteksi objek diskala yang berbeda. Output pertama untuk mendeteksi ukuran besar dengan *feature map* dibagi menjadi 13×13 kotak memiliki 1024 parameter, kedua untuk mendeteksi ukuran sedang dengan *feature map* dibagi menjadi 26×26 kotak memiliki 512 parameter dan ketiga untuk mendeteksi ukuran besar dengan *feature map* dibagi menjadi 52×52 kotak memiliki 256 parameter.



Gambar 2.9: Arsitektur deteksi objek YOLO dengan *Feature Extractor* Darknet 53

Tiny YOLO menggunakan *feature extractor* yang disebut *Darknet Reference*. Arsitektur ini merupakan *feature extractor* yang mirip dengan *Darknet 19* [19] namun dibuat lebih kecil untuk proses komputasi yang lebih cepat. Berbeda dengan *Darknet 53*, arsitektur ini menggunakan *maxpool layer* untuk melakukan *downsampling* (memperkecil ukuran *feature map*). Cara kerjanya adalah mengurangi parameter dengan mengambil nilai yang terbesar di tiap filter. Untuk lebih jelasnya prosesnya sama seperti di gambar 2.3.



Gambar 2.10: Arsitektur deteksi objek Tiny YOLO dengan *Feature Extractor* Darknet Reference

Desain arsitektur Tiny YOLO pada gambar 2.10 terdiri dari satu input dan 2 output. Tiap output dari model digunakan untuk mendeteksi objek diskala yang berbeda. Output pertama untuk mendeteksi ukuran besar dengan *feature map* dibagi menjadi 13×13 kotak memiliki 512 parameter, kedua untuk mendeteksi ukuran

sedang dengan *feature map* dibagi menjadi 26×26 kotak memiliki 256 parameter. Tiny YOLO tidak memiliki output untuk mendeteksi objek berukuran kecil dan jumlah parameternya juga lebih sedikit.

2.3.3 Loss

YOLO Loss Function menggunakan *sum-squared error* [6]. Kalkulasi dilakukan tiap $x, y, w, h, confidence$ dan *probability*. Perhitungan loss dilakukan secara terpisah untuk xy , wh , *confidence* dan *class* lalu dijumlahkan. Fungsi *loss* memiliki fokus untuk menghitung *classification loss* jika objek terdeteksi, *confidence score loss* jika ada objek yang tidak terdeteksi maupun *background* yang dianggap sebagai objek dan *localization error* untuk posisi dari objek yang terdeteksi. Tiap skala memiliki nilai loss sendiri-sendiri lalu dijumlahkan. Berikut persamaan *YOLO Loss Function* (2.9) untuk lebih jelasnya bisa dilihat di *paper* aslinya [6].

$$\begin{aligned} \mathcal{L}_{Yolo} = & f_{loc}(x_{pred}, x_{true}, y_{pred}, y_{true}, w_{pred}, w_{true}, h_{pred}, h_{true}) \\ & + f_{conf}(p_{pred}, p_{true}) + f_{cls}(p_{pred}, p_{true}) \end{aligned} \quad (2.9)$$

Dengan,

\mathcal{L}_{Yolo} = nilai loss YOLO

$x_{pred}, y_{pred}, w_{pred}, h_{pred}, p_{pred}$ = Nilai prediksi

$x_{true}, y_{true}, w_{true}, h_{true}, p_{true}$ = Nilai label asli

$f_{loc}()$ = fungsi *localization error*

$f_{conf}()$ = fungsi *confidence score loss*

$f_{cls}()$ = fungsi *classification loss*

Loss function hanya memberikan penalti pada *classification error* jika ada objek di grid cell. *error* juga ditentukan dari nilai IOU dan *confidence score* yang sudah diatur sebelum *training*.

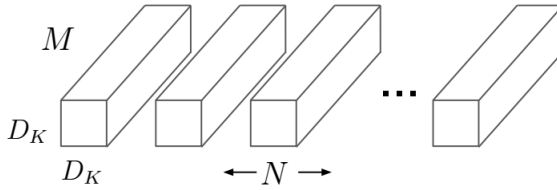
2.4 Mobilenet

Mobilenet adalah *class efficient model* untuk Aplikasi *mobile* dan *embedded vision* [2]. Mobilenet dibuat berdasarkan *streamlined architecture* yang menggunakan *depthwise separable convolu-*

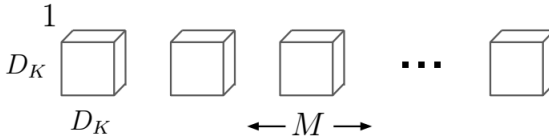
tion untuk membuat *neural network* yang ringan.

2.4.1 Depthwise Separable Convolution

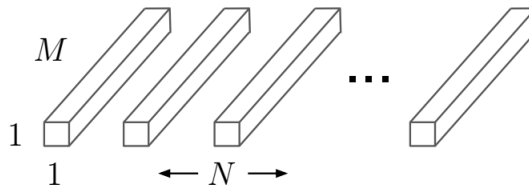
Layer convolution yang mendasari *MobileNet* adalah *depthwise separable filters*. *Depthwise separable convolutions* adalah *convolution* yang memfaktorisasikan *standard convolution* menjadi 2 operasi yaitu *depthwise convolution* dan *1x1 convolution pointwise convolution*. *Standar convolution* melakukan filter dan menggabungkan input menjadi set output baru dalam satu langkah.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



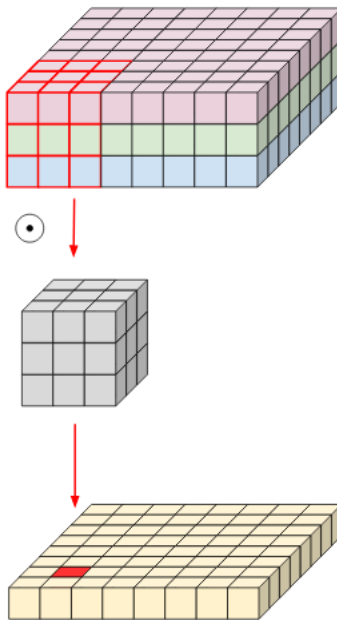
(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Gambar 2.11: Alur *Depthwise Separable Filter*

Seperti pada gambar 2.11 [2]. *Depthwise separable convolution*

memisahnya menjadi dua operasi *layers*, *layer* pertama (*depthwise*) melakukan filtering dan *layer* kedua (*pointwise*) bertugas untuk menyatukan layernya. Faktorisasi ini mempunyai efek mengurangi komputasi dan ukuran model. Gambar berikut menunjukkan bagaimana cara *standard convolution*, difaktorisasikan menjadi *depthwise convolution* dan 1×1 *pointwise convolution*.

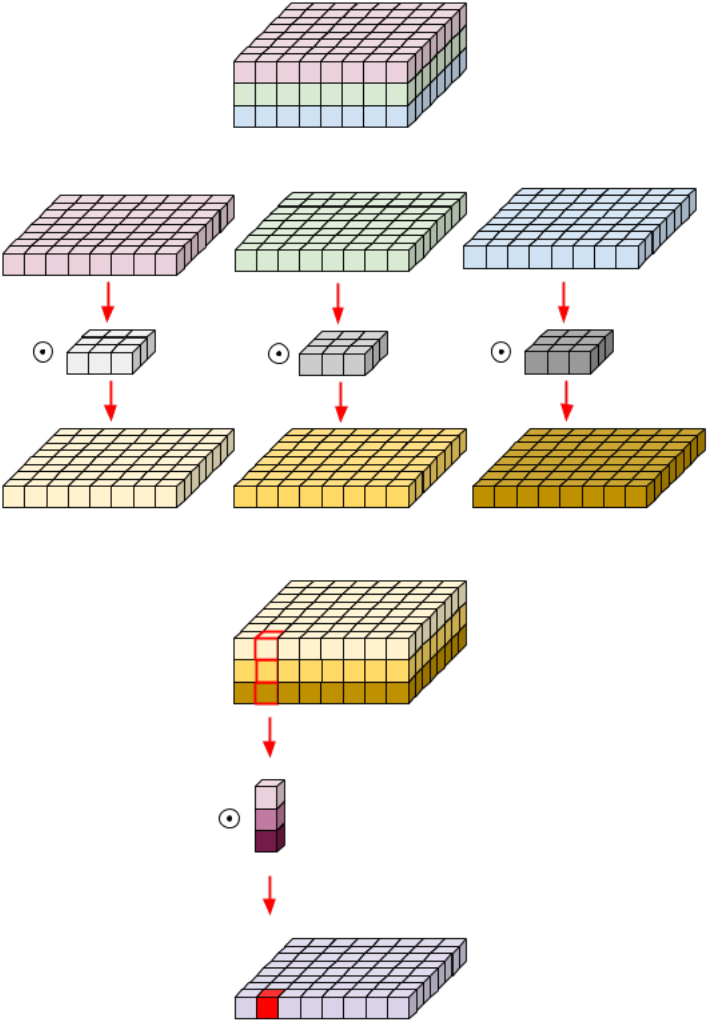
Standard convolution melakukan filter di tiap *channel* dan menyatukannya menjadi satu dimensi. Ilustrasi proses seperti pada gambar 2.12.



Gambar 2.12: *Standard Convolution*

Sedangkan *depthwise separable convolution* memisah prosesnya menjadi dua tahap. Pada tahap *depthwise*, *channel* dipisah dan dilakukan konvolusi sendiri-sendiri. Setelah itu tiap *channel* ditumpuk kembali. Setelah itu saat *pointwise* ketiga *channel* dikonvolusikan menjadi satu dimensi sama seperti output konvolusi biasa. Ilustrasi

proses seperti pada gambar 2.13.



Gambar 2.13: *Separable Depthwise Convolution*

2.4.2 Computational Cost

Setiap konvolusi memiliki input *feature map* $D_K \times D_K \times M$ dan output $D_F \times D_F \times N$ seperti pada persamaan (2.10). Cara mengolah *feature map matrix* mempengaruhi kecepatan dari sebuah arsitektur selain faktor yang lain seperti jumlah layer dan memori. Komputasi dari *Depthwise Separable Convolution* dapat berjalan lebih cepat dari *Standard Convolution*. Hal ini dapat dibuktikan dari membandingkan ukuran komputasinya.

$$\begin{aligned} \text{Input shape} &= D_K \times D_K \times M \\ \text{Output shape} &= D_F \times D_F \times N \end{aligned} \tag{2.10}$$

Dengan :

D_K = ukuran input *feature map*

D_F = ukuran output *feature map*

M = ukuran input *channel (depth)*

N = ukuran output *channel (depth)*

Standard convolution melibatkan semua parameter untuk melakukan operasi. Untuk menghitung ukuran komputasinya, bisa dilakukan dengan mengkalikan semua parameter yang digunakan. Dengan kesimpulan tadi biaya komputasi *standard convolution* dapat dihitung dengan rumus (2.11).

$$\text{standard cost} = D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \tag{2.11}$$

Sedangkan *depthwise separable convolution* memiliki dua operasi yaitu *depthwise* dan *pointwise* sehingga rumus komputasinya pun terpisah. Rumus komputasi hampir sama dengan *standard convolution* namun karena *depthwise* tidak menyatukan *depth* menjadi satu dimensi, maka nilai N tidak diperhitungkan. Jadi, biaya komputasi *depthwise convolution* dapat dihitung dengan rumus (2.12):

$$\text{depthwise cost} = D_K \cdot D_K \cdot M \cdot D_F \cdot D_F \tag{2.12}$$

Bisa *depthwise convolution* lebih efisien dibandingkan *standard*

convolution. Namun konvolusinya hanya mengfilter *input channel* dan tidak menyatukannya menjadi fitur baru. Jadi layer tambahan yang mengkomputasikan kombinasi linear dari keluaran *depthwise convolution* 1×1 *convolution* dibutuhkan untuk membuat fitur baru. *Pointwise convolution* menggunakan 1×1 *convolution* untuk menggabungkan *depth* menjadi satu. *pointwise* mengolah hasil keluaran *depthwise* sehingga tidak perlu melibatkan D_K . Sehingga biaya komputasi *pointwise convolution* dapat dihitung dengan rumus (2.13).

$$\textit{pointwise cost} = M.N.D_F.D_F \quad (2.13)$$

Jika disatukan total dari semua konvolusi, maka biaya komputasi *depthwise separable convolution* akan memiliki rumus (2.14).

$$\textit{depthwise separable cost} = D_K.D_K.M.D_F.D_F + M.N.D_F.D_F \quad (2.14)$$

Dengan membandingkan kedua *convolution* maka pengurangan ukuran yang diperoleh adalah 2.15:

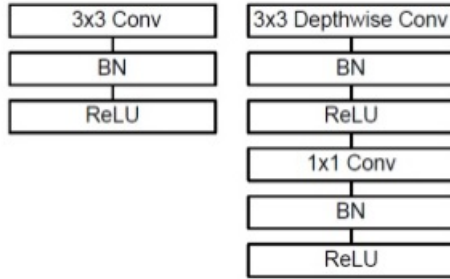
$$\begin{aligned} \textit{reduction cost} &= \frac{D_K.D_K.M.D_F.D_F + M.N.D_F.D_F}{D_K.D_K.M.N.D_F.D_F} \\ &= \frac{1}{N} + \frac{1}{D_k^2} \end{aligned} \quad (2.15)$$

Perbandingan tersebut dapat dibuktikan dengan contoh (2.16):

$$\begin{aligned}
\frac{\text{Input shape}}{\text{Output shape}} &= \frac{D_K \times D_K \times M}{D_F \times D_F \times N} \\
\frac{D_K \times D_K \times M}{D_F \times D_F \times N} &= \frac{52 \times 52 \times 256}{26 \times 26 \times 512} \\
\text{standard cost} &= 3 \times 3 \times 64 \times 112 \times 112 \times 128 \\
&= 924.844.032 \tag{2.16} \\
\text{depthwise separable cost} &= 3 \times 3 \times 64 \times 112 \times 112 + \\
&64 \times 128 \times 112 \times 112 \\
&= 7.225.344 + 102.760.448 \\
&= 109,985,792
\end{aligned}$$

Setelah dibandingkan , *MobileNet* menggunakan 3 X 3 *depthwise separable convolution* membuat komputasi 8 sampai 9 kali lebih sedikit dari *standard convolution* dan hanya diikuti sedikit pengurangan akurasi.

2.4.3 Network Architecture



Gambar 2.14: *Standard Convolution* dan *Depthwise Separable Convolution*

Struktur *MobileNet* terdiri dari *depthwise separable convolutions* , namun untuk *layer* pertama konvolusi penuh. Pola *layer* bisa dilihat pada gambar 2.14. Jika dibandingkan dengan *standard convolution* bisa dilihat jika *standard convolution* hanya terdiri dari

1 *convolutional layer* lalu diikuti dengan *batch normalization layer* dan *ReLU*. Sedangkan *depthwise Separable Convolution* terdiri dari dua *Convolution layer* yaitu *Convolution layer* yang melakukan *Depthwise convolution* dan *Convolution layer* yang melakukan *pointwise convolution*. Kedua layer tersebut kemudian diikuti oleh *Batch Normalization layer* dan *ReLU* sama seperti *Standard convolution*.

Desain arsitektur dibuat dengan mempertimbangkan perbandingan antara kecepatan dan akurasi. Untuk arsitektur Mobilenet secara keseluruhan menggunakan *Depthwise Separable Convolution* kecuali pada layer pertama yang menggunakan *standard convolution*. Hal ini dikarenakan untuk *layer* pertama komputasinya masih kecil, sehingga tidak perlu dirubah. Selanjutnya semua *convolution layer* diikuti oleh *batch normalization layer* dan *ReLU* kecuali layer terakhir sebelum *softmax* untuk klasifikasi. *MobileNet* terdiri dari 28 *layer*. Arsitektur bisa dilihat pada gambar 2.15

Table 1. MobileNet Body Architecture

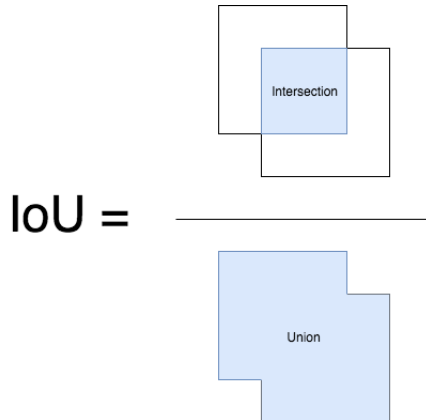
Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5×	Conv dw / s1	$3 \times 3 \times 512$ dw
	Conv / s1	$1 \times 1 \times 512 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Gambar 2.15: MobileNet Architecture

2.5 IoU (*Intersection over Union*)

Intersection over Union adalah evaluasi *metric* untuk mengukur keakuratan pendeteksi objek. IoU mengukur besaran *overlap* dari dua kotak yaitu kotak label asli dan kotak hasil prediksi. Semakin tinggi hasil IoU maka hasil prediksi akan semakin bagus. Penerapan IoU dapat dihitung dengan rumus (2.17) dan diilustrasikan dengan gambar 2.16.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} \quad (2.17)$$



Gambar 2.16: IoU *Intersecton over Union*

2.6 mAP (*mean Average Precision*)

Untuk mengukur keakuratan sebuah objek deteksi dapat digunakan *metric* yang disebut mAP (*mean Average Precision*). mAP adalah hasil pengukuran AP (*Average Precision*) tiap kategori objek yang dirata-rata. Untuk mendapatkan AP bisa diperoleh dengan menghitung nilai *precision* dari tiap *recall*. *Precision* adalah pengukuran keakuratan deteksi sedangkan *Recall* mengukur seberapa banyaknya positif yang ditemukan. *Precision* (2.18) dan *Recall*

(2.19) dapat dijelaskan dengan persamaan dibawah ini.

$$Precision = \frac{TP}{TP + FP} \quad (2.18)$$

$$Recall = \frac{TP}{TP + FN} \quad (2.19)$$

Dengan,

TP = *True Positive*

TN = *True Negative*

FP = *False Positive*

FN = *False Negative*

Setelah memperoleh nilai *Precision* dan *Recall* , maka AP (2.20) dapat dihitung dengan persamaan berikut.

$$AP = \sum (r_{n+1} - r_n) p_{interp}(r_{n+1}) \quad (2.20)$$

$$p_{interp}(r_{n+1}) = \max_{\tilde{r} \geq r} p(\tilde{r})$$

Dengan,

AP = *Average Precision*

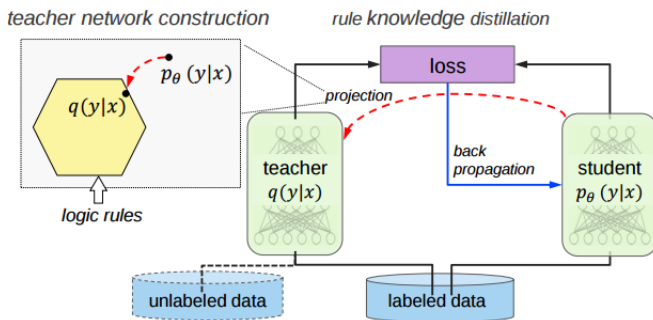
r = *Recall*

p = *Precision*

p_{interp} = *Maximum Precision* \tilde{r} = nilai *Recall* terakhir

Setelah mendapatkan AP tiap objek , maka total akan diperoleh mAP. Proses penghitungan mAP secara keseluruhan adalah seperti diagram pada gambar 2.17.

transfer ilmu dari model besar ke model kecil yang bisa digunakan untuk perangkat ringan [12]. Tujuan dilakukannya *training* adalah untuk mendapatkan data nilai probabilitas yang benar dari banyak *class*, namun efek samping *training* menggunakan *teacher* adalah adanya probabilitas jawaban yang salah meskipun probabilitasnya kecil. Nilai probabilitas dari jawaban yang salah dapat memberitahukan bagaimana model melakukan perataan probabilitas. contohnya gambar BMW dapat memiliki kemungkinan untuk dianggap sebagai truk, namun masih lebih baik daripada dianggap sebagai wortel.



Gambar 2.18: *Student-Teacher Distillation*

Tujuan dilakukan *training* adalah agar model dapat mengenali objek dengan seakurat mungkin. Meskipun begitu model cenderung dilatih untuk mengoptimalkan performa pada data *training* saja, padahal model yang baik seharusnya tetap dapat berfungsi dengan baik pada data yang belum pernah dilihat sebelumnya. Jelas akan lebih baik untuk melatih model agar dapat beradaptasi dengan data baru, namun informasi tentang cara beradaptasi dengan data baru biasanya sulit diperoleh. Salah informasi data yang dapat dicoba adalah dengan menggunakan nilai keluaran dari model dengan akurasi yang tinggi. Nilai keluaran dari sebuah model dapat memiliki informasi-informasi yang tidak dapat diperoleh dari label *training* biasa. Sehingga saat melakukan distilasi kita bisa melatih model kecil untuk beradaptasi dengan cara yang sama seperti model yang

besar.

Proses distilasi dilakukan dengan menggunakan nilai keluaran probabilitas (*logits*) dari Model *teacher* sebagai *soft targets* untuk digunakan pada training model kecil. Logits dari teacher diolah dengan *softmax* untuk dijadikan *soft target*. Persamaan *softmax* untuk distilasi didefinisikan pada persamaan (2.21)

$$q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \quad (2.21)$$

Dengan,

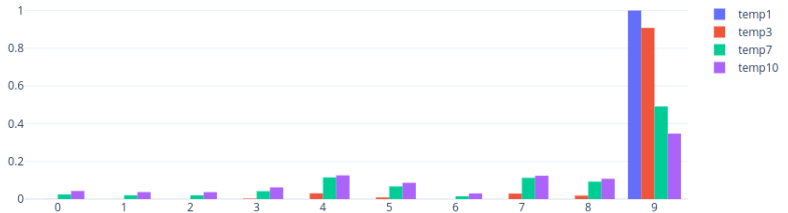
q_i = Nilai logits setelah diolah

z_i = Nilai mentahan output model (logits)

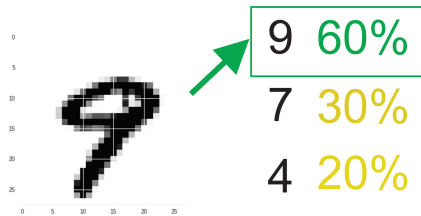
T = Nilai *hyperparameter* temperatur untuk distilasi

Perbedaan *softmax* pada distilasi dan *softmax* biasa nilai T yaitu temperatur. Temperatur digunakan untuk menghasilkan probabilitas yang lebih merata antar kelas. Semakin besar temperatur maka nilai probabilitas yang salah akan semakin terlihat. Temperatur yang besar berguna untuk menunjukkan informasi yang diperoleh oleh model *teacher*. Namun model kecil masih belum mampu menangkap semua informasi dari model besar, jadi temperatur kecil yang lebih cocok digunakan untuk training model kecil (2.22). Contoh perbandingan distribusi nilai probabilitas tiap temperatur untuk angka 9 bisa dilihat pada gambar (2.19).

Jika gambar (2.19) diamati, maka bisa dilihat bahwa nilai klasifikasi angka 9 memiliki penyebaran probabilitas di bilangan lain juga. Saat diamati lebih lanjut, angka 4 dan 7 memiliki nilai yang lebih tinggi dibandingkan angka lain. Jika dilihat pada gambar 2.20, angka sembilan pada dataset yang digunakan, memang memiliki kemiripan dengan bilangan 4 dan 7. Informasi ini membuktikan bahwa tiap dataset memiliki nilai distribusi yang unik. Data ini tidak dapat diperoleh dengan *training* biasa sehingga diperlukan *training* dengan cara distilasi.



Gambar 2.19: Perbandingan distribusi nilai probabilitas tiap temperatur



Gambar 2.20: Angka 9 memiliki kemiripan dengan angka 7 dan 4 dibandingkan dengan angka lain

Alasan mengapa digunakannya temperatur untuk meningkatkan distribusi logits yang salah adalah untuk menyediakan informasi kemiripan diantara output kategori. Dengan begitu *student* juga dapat mempelajari kemiripan diantara kelas dan perbedaan nilai probabilitas kelas yang sama. Selain itu *soft targets* juga dapat bertindak sebagai *regularizers* yang terarah untuk *student* [12]. Persamaan untuk mengatur *loss* dari distilasi adalah sebagai berikut (2.22).

$$\mathcal{L}_{KD} = \alpha T^2 * \mathcal{H}(q_s^T, q_T^T) + (1 - \alpha) * \mathcal{H}(q_s, \mathcal{Y}_{true}) \quad (2.22)$$

Dengan,

\mathcal{L}_{KD} = *knowledge distillation loss*

α = hyperparameter alpha untuk distilasi

q_s^T = Nilai logits *student* setelah diolah dengan nilai temperatur

q_T^T = Nilai logits *teacher* setelah diolah dengan nilai temperatur

q_s = Nilai logits *student* setelah diolah

\mathcal{Y}_{true} = Nilai label

q_s^T dan q_T^T adalah nilai soft target dari student dan *teacher* menggunakan nilai temperatur(T) dan α adalah *hyperparameter* untuk mengatur perataan antara *student-teacher loss* dan loss dari label asli.

2.8 Batch Normalization Fusion

Batch Normalization (BN) adalah salah satu blok utama dari *convolutional neural network* modern. *Batch normalization* adalah teknik untuk meningkatkan kecepatan, performa dan stabilitas dari CNN. Cara kerja *batch normalization* adalah melakukan normalisasi menggunakan nilai rata-rata dan varian dari matriks input. CNN yang modern memiliki banyak sekali BN karena di tiap *convolutional layer* selalu didampingi dengan BN [20]. Semakin banyak layer yang dipakai maka waktu komputasi akan semakin lama karena perpindahan memori data antara layer membutuhkan waktu yang lama. Cara yang efektif untuk mengurangi durasi perpindahan memori adalah dengan menyatukan *batch normalization layer* dengan *convolutional layer* utama. Peningkatan kecepatan bisa mencapai hampir dua kali lipat tergantung seberapa banyak *batch normalization layer* yang disatukan.

Cara menyatukan *batch normalization* dengan *convolution* adalah menggabungkan parameter-parameter di *batch normalization* ke persamaan *convolution* , sehingga *convolution* memiliki persamaan *weight* dan *bias* baru yang sudah terdapat parameter *batch normalization*.

Persamaan *Convolution Layer* (2.23) :

$$Conv : Y = W * X + B \quad (2.23)$$

Persamaan *BatchNormalization Layer* (2.24) :

$$\text{BatchNorm} : Y = \gamma * \frac{X - \text{RunMean}}{\sqrt{\text{RunStd} + \text{eps}}} + \beta \quad (2.24)$$

Persamaan weight dan bias baru setelah di gabung dengan *batch normalization* (2.25) :

$$\begin{aligned} \text{BatchNorm} : Y &= \gamma * \frac{(W * X + B) - \text{RunMean}}{\sqrt{\text{RunStd} + \text{eps}}} + \beta \\ &= \frac{\gamma * W}{\sqrt{\text{RunStd} + \text{eps}}} X + \\ &\quad \left(\text{Beta} + \frac{\gamma * (B - \text{RunMean})}{\sqrt{\text{RunStd} + \text{eps}}} \right) \end{aligned} \quad (2.25)$$

$$\begin{aligned} W' &= \frac{\gamma * W}{\sqrt{\text{RunStd} + \text{eps}}} \\ B' &= \text{Beta} + \frac{\gamma * (B - \text{RunMean})}{\sqrt{\text{RunStd} + \text{eps}}} \end{aligned}$$

Dengan,

Y : Nilai *output convolutional layer*

W : Nilai *Weight*

X : Nilai *input convolutional layer*

B : Nilai *Bias*

γ = gamma *learnable parameter*

β = beta *learnable parameter*

RunMean = Nilai rata-rata X

RunStd = (Standard Deviation) Jarak penyebaran data relatif pada rata-rata

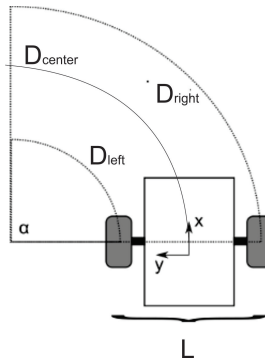
2.9 Odometry

Odometry adalah penggunaan sensor untuk mendeteksi posisi robot. Data *odometry* bisa diperoleh dari perputaran kedua roda pada robot pada sisi kanan dan sisi kiri. Jika ke dua roda berotasi ke depan bersamaan , maka robot akan maju. Jika roda kanan ber-

gerak lebih cepat dari roda kiri maka robot akan bergerak kekiri, begitu juga sebaliknya. Informasi dari rotasi roda tersebut memberikan dua nilai, nilai rotasi belokan robot dan jarak perpindahan robot. Dengan kedua nilai tersebut kita bisa mendapatkan posisi robot (x,y,θ) [21].

Untuk menentukan jumlah rotasi roda bisa dihitung dengan menggunakan *encoder*. *Encoder* adalah sensor untuk menentukan pergerakan dari rotasi roda [22]. Cara kerjanya adalah dengan memberikan penanda (*tick*) di tiap bagian roda, saat berputar akan ada sensor yang mendeteksi tanda tersebut. Jumlah tiap kali tanda terdeteksi dapat digunakan untuk mengukur berapa kali roda telah berotasi.

Ilustrasi pergerakan robot menggunakan odometry 2.21



Gambar 2.21: Pengukuran posisi robot dengan *Odometry*

Untuk mengukur jarak berdasarkan jumlah *tick* dapat digunakan persamaan (2.26)

$$\Delta Tick = Tick' - Tick$$

$$D_R = D_L = 2\pi r \frac{\Delta Tick}{FullTick} \quad (2.26)$$

$$D_c = \frac{D_R + D_L}{2}$$

Setelah mendapatkan nilai Jarak (D), maka posisi dapat diukur menggunakan persamaan

$$\begin{aligned}\theta' &= \theta + \frac{D_R - D_L}{L} \\ X'_r &= X_r + D_C * \text{Cos}\theta \\ Y'_r &= Y_r + D_C * \text{Sin}\theta\end{aligned}\tag{2.27}$$

Dengan,

$\Delta Tick$ = Selisih *Tick*

$Tick'$ = Total jumlah nilai *encoder* sebelumnya

$Tick$ = Total jumlah nilai *encoder* sekarang

D_R = Jarak roda kiri dari titik awal

D_L = Jarak roda kanan dari titik awal

$FullTick$ = Total jumlah *encoder* roda satu putaran

D_c = Jarak robot dari titik awal

L = Jarak roda kanan dan roda kiri

θ = sudut awal perpindahan robot

θ' = sudut baru perpindahan robot

X_r = posisi awal koordinat X robot

Y_r = posisi awal koordinat Y robot

X'_r = posisi baru koordinat X robot

Y'_r = posisi baru koordinat Y robot

2.10 *Makeblock*

Makeblock adalah perangkat untuk belajar robotik dari china dengan mainboard berbasis Arduino. *Mainboard* dari Robot yang digunakan adalah Auriga yang merupakan versi lanjutan dari Orion. Basis desain sirkuit Auriga berasal dari Arduino Mega yang dimodifikasi. Pada Auriga sudah dilengkapi berbagai macam sensor seperti temperatur, sensor suara, *gyroscope*, *buzzer* dan bluetooth. *Makeblock* dapat diprogram menggunakan Arduino IDE.

Halaman ini sengaja dikosongkan

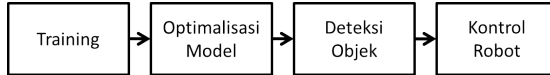
BAB 3

DESAIN DAN IMPLEMENTASI SISTEM

Penelitian ini dilaksanakan sesuai dengan desain sistem berikut dengan implementasinya. Desain sistem merupakan konsep dari pembuatan dan perancangan infrastruktur dan kemudian diwujudkan dalam bentuk blok-blok alur yang harus dikerjakan.

3.1 Desain Sistem

Tujuan dilaksanakannya penelitian ini adalah untuk meningkatkan performa arsitektur *neural network* pendeteksi objek yang ringan dan cepat, namun masih memiliki akurasi yang bisa diandalkan di perangkat dengan komputasi kecil. Proses *Training* akan dilakukan di komputer dan hasil modelnya akan diterapkan pada raspberry yang juga mengontrol gerakan robot. Proses Dasar Sistem ditunjukkan pada gambar 3.1.



Gambar 3.1: Gambaran Umum Sistem

Sesuai dengan diagram 3.1 proses yang pertama yang dilakukan adalah proses training. Tujuan dilaksanakan proses training adalah untuk dapat menerapkan beberapa metode untuk optimalisasi model yang harus dilakukan saat training. Untuk Optimalisasi model yang dilakukan adalah memodifikasi model yang sudah ada agar dapat menjadi lebih kecil, lalu meningkatkan akurasi dan mempercepat komputasi. Setelah itu harus dibuat Sistem deteksi objek yang dapat diterapkan pada robot. Setelah sistem deteksi objek selesai, maka robot juga harus di program untuk mengikuti sistem deteksi objek.

3.2 Work Flow

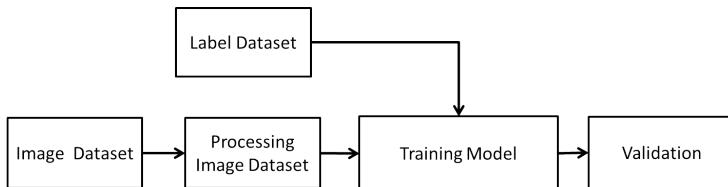
Implementasi akan menerapkan semua teori-teori dan metode-metode yang ada di tinjauan pustaka dan mewujudkan aplikasi dari

desain sistem. Pengerjaan akan dibagi menjadi beberapa tahapan ,yaitu:

1. Proses *training data*
2. Penerapan metode untuk optimalisasi model
3. Sistem deteksi objek
4. Sistem kontrol robot

3.3 Proses *Training Data*

Sebelum *training* gambar harus diproses terlebih dahulu, tujuannya untuk memaksimalkan fitur yang dapat dipelajari oleh model. Setelah itu gambar yang telah diolah akan dimasukkan ke model lalu dicek output prediksinya. Hasil prediksi akan dibandingkan dengan label asli untuk mendapatkan nilai *loss / error* dan validasi untuk melihat performa model dengan beberapa data lain. nilai *loss* kemudian digunakan untuk meningkatkan akurasi model. Proses *training* akan dilakukan untuk mendapatkan model dengan akurasi optimal yang tidak *overfitting*. Gambar 3.2 menunjukkan proses training neural network untuk pendeteksi objek.



Gambar 3.2: Blok diagram training *neural network*

3.3.1 Dataset

Dataset yang digunakan adalah dataset Pascal VOC 2012 [23] dan dataset sendiri yang terdiri dari objek-objek yang sering dicari manula. Dataset VOC memiliki 20 *class* objek barang yang sering dijumpai di kehidupan sehari-hari. Dataset dipisah menjadi 2 bagian yaitu *training* sebanyak 5717 gambar dan *validation* sebanyak 5823 gambar. *Batch Size* yang digunakan sebesar 24. Tiap gambar dapat memiliki lebih dari satu objek sehingga jika di jumlah maka untuk *training* ada 13609 objek dan untuk *validation* ada 13841 objek.

Dataset khusus yang digunakan terdiri dari objek - objek yang sering dicari manula seperti sepatu, kunci, dompet, kacamata dan *hand phone*. Dataset terdiri dari 1222 gambar yang berisi campuran semua objek yang akan dideteksi. Saat *training* dataset akan dipisah menjadi 1029 gambar untuk *training* dan 193 gambar untuk *validation*. Nama objek dan jumlah gambar dapat dilihat pada tabel 3.1.

Tabel 3.1: Dataset *custom* untuk barang milik manula

Nama Class	Jumlah Gambar
Kunci	214
Sepatu/Sandal	286
Kacamata	241
Dompet	244
Smartphone	237
Total	1222

Tiap objek memiliki label yang berisi informasi kelas objek dan posisi. Informasi label disimpan dalam bentuk xml. Dalam tiap gambar dapat terdiri dari beberapa objek yang sama maupun berbeda. Berikut contoh format anotasi 3.1 dalam bentuk format dataset VOC dan gambar 3.3 yang berisi ilustrasi jika informasi xml ditampilkan dengan bentuk gambar.

```

<annotation>
<folder>JPEGImages</folder>
<filename>kwp_00000035.jpg</filename>
<path>D:\python\google_image_dataset\selected_images\mix\
    JPEGImages\kwp_00000035.jpg</path>
<source>
<database>Unknown</database>
</source>
<size>
<width>240</width>
<height>160</height>
<depth>3</depth>
</size>
<segmented>0</segmented>
<object>
<name>wallet</name>
<pose>Unspecified</pose>

```

```

<truncated>0</truncated>
<bndbox>
<xmin>12</xmin><ymin>71</ymin>
<xmax>80</xmax><ymin>148</ymax>
</bndbox>
</object>
<object>
<name>phone</name>
<pose>Unspecified</pose>
<truncated>0</truncated>
<bndbox>
<xmin>93</xmin>
<ymin>61</ymin>
<xmax>137</xmax>
<ymin>151</ymax>
</bndbox>
</object>
<object>
<name>key</name>
<pose>Unspecified</pose>
<truncated>0</truncated>
<bndbox>
<xmin>202</xmin>
<ymin>90</ymin>
<xmax>232</xmax>
<ymin>149</ymax>
</bndbox>
</object>
</annotation>

```

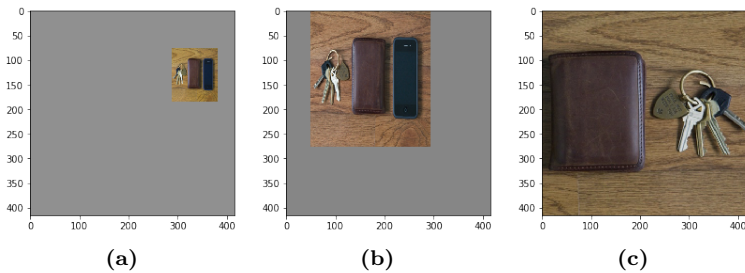
Listing 3.1: Anotasi objek



Gambar 3.3: Contoh dataset dan label

Objek deteksi memiliki *output* dalam 3 skala , karena ukur-

an benda dapat berubah ubah relatif terhadap jarak, maka pada saat training ukuran gambar akan dirubah secara *random*. Nilai label posisi dari objek juga akan menyesuaikan gambar untuk menentukan penempatan kategori skala yang benar. Dengan begitu tiap skala akan dapat mengenali objek tersebut dan dataset menjadi lebih bervariasi. Contoh gambar 3.4 dalam berbagai skala dalam ukuran kecil sedang dan besar.



Gambar 3.4: Ukuran gambar berubah secara acak untuk deteksi berbagai skala

3.3.2 Pengaturan *Training*

Proses *training* akan dilakukan sebanyak 200 *epoch*/iterasi. Untuk satu epoch dataset akan dibagi menjadi 32 *batch size* tiap *batch* hingga semua data terproses. Nilai *learning rate* sebesar 1×10^{-3} dan jika nilai loss tidak mengalami perubahan maka *learning rate* akan diperkecil mulai dari 1×10^{-4} dan seterusnya.

Penentuan nilai *batch size* dan *learning rate* juga berperan penting dalam peningkatan akurasi. *Batch size* yang besar dapat memberikan jenis data yang bervariasi sehingga saat dilakukan perhitungan *loss*, nilai perhitungan *loss* yang diperoleh dapat mewakili variasi dataset secara keseluruhan. Sebaliknya *batch size* yang kecil hanya dapat memberikan perwakilan sedikit dataset sehingga nilai perubahan *loss* untuk memperoleh nilai stabil akan semakin sulit. Karena itu *batch size* yang besar dapat diberikan *learning rate* yang besar karena arah perpindahan *loss* nya lebih stabil. Sedangkan *batch size* yang kecil harus diberikan *learning rate* yang kecil juga

agar perpindahannya tidak berpindah pindah terlalu jauh.

Sama seperti sumber aslinya ImageNet digunakan sebagai *pre-train* model YOLO [6]. ImageNet adalah dataset yang bersal dari *ImageNet Large Scale Visual Recognition Challenge* berisi data gambar untuk klasifikasi 1000 kelas [24]. Model *pretrain* digunakan untuk mendapatkan akurasi yang lebih bagus dan waktu training jadi lebih cepat. Tanpa *pretrain* proses training akan memakan waktu sangat lama. *Pretrain* digunakan agar saat training sudah ada data nilai-nilai feature untuk referensi sehingga nilai awalnya tidak berupa nilai acak. Saat *training* penyesuaian nilai *feature* dengan dataset yang sudah disiapkan akan lebih cepat dengan menggunakan nilai *feature* dari *pretrain*.

3.3.3 Penentuan nilai *Anchor*

Anchor adalah ukuran awal semua *bounding box* yang akan dijadikan referensi tiap objek yang dideteksi berupa nilai lebar (w) dan tinggi (h). Tiap objek yang terdeteksi akan diambil *anchor* dengan ukuran yang paling sesuai lalu kemudian ukuran tersebut dapat diperbesar atau diperkecil agar mendekati ukuran objek yang terdeteksi. Tiap skala memiliki 3 *anchor*. Nilai *anchor* didapatkan dari menggolongkan dataset yang digunakan menjadi beberapa kelompok menggunakan algoritma *k-means* [1].

Untuk model dengan 3 output jumlah *anchor* adalah 9, sedangkan model dengan 2 output jumlah *anchor* adalah 6. Hasil pengelompokan dataset yang digunakan dengan algoritma *k-means* adalah sebagai berikut:

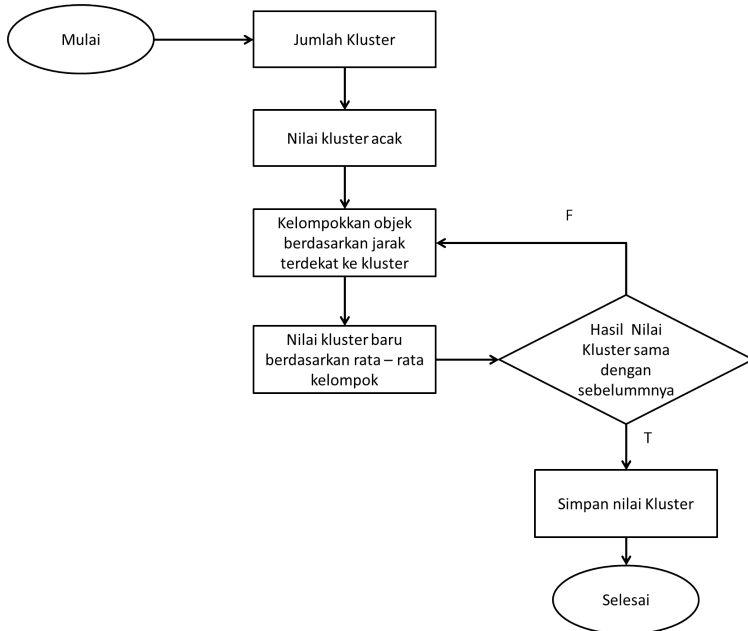
Anchor dataset VOC dengan 9 anchor terdiri dari:
(27×37), (34×83), (64×135), (64×53), (113×99),
(124×214), (221×291), (235×141), (386×29)

Anchor dataset VOC dengan 6 anchor terdiri dari:
(27×40), (40×96), (71×57),
(91×131), (171×205), (345×289)

Anchor dataset Manula dengan 9 anchor terdiri dari:
(33×46), (66×130), (95×51), (137×213), (179×93),
(185×335), (266×161), (309×305), (499×360)

Anchor dataset Manula dengan 6 anchor terdiri dari:
(37×45), (67×132), (118×61),
(150×246), (244×135), (330×322)

Cara kerja *k-means* adalah mengelompokkan data berdasarkan kemiripan nilai data menjadi beberapa kluster. Setelah itu tiap kluster akan di rata-rata. Nilai rata-rata tersebut akan digunakan sebagai nilai *anchor*. Untuk lebih jelasnya alir diagram *k-means* dilihat pada gambar 3.5 dan fungsi 3.2.



Gambar 3.5: Penentuan kelompok dengan *k-means*

```

def kmeans(self, boxes, k, dist=np.median):
    box_number = boxes.shape[0]
    distances = np.empty((box_number, k))
    last_nearest = np.zeros((box_number,))
    np.random.seed()
    clusters = boxes[np.random.choice(
        box_number, k, replace=False)] # init k clusters
    while True:

        distances = 1 - self.iou(boxes, clusters)
  
```



```

current_nearest = np.argmin(distances, axis=1)
if (last_nearest == current_nearest).all():
    break # clusters won't change
for cluster in range(k):
    clusters[cluster] = dist( # update clusters
        boxes[current_nearest == cluster], axis=0)

last_nearest = current_nearest

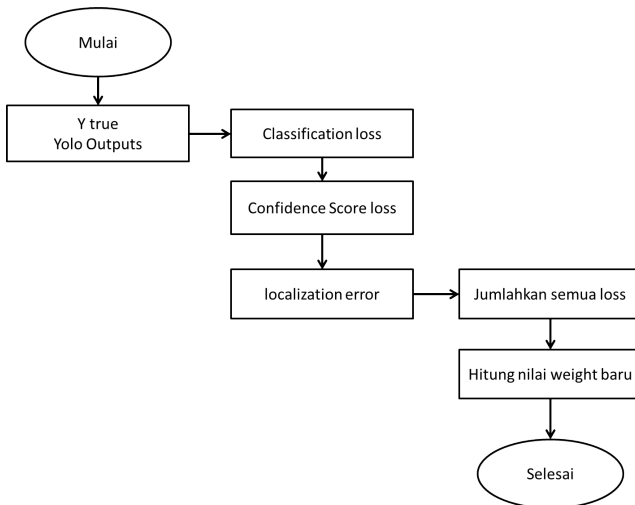
return clusters

```

Listing 3.2: Fungsi k-means

3.3.4 Perhitungan *error* (*loss*)

Saat proses training, *feed forward* akan melakukan kalkulasi seperti inference biasa. Hasil *Output* dari inference akan dibandingkan dengan label data untuk perhitungan *loss*. Nilai *Loss* adalah kombinasi dari 3 skala dan nilai dari *xywh*. Secara garis besar Fungsi *Loss* digunakan untuk mengukur 3 hal yaitu *Classification loss*, *Confidence Score loss* dan *localization error*. Alur dari perhitungan *loss* dapat dijelaskan melalui diagram 3.6 dan fungsi 3.3.



Gambar 3.6: Perhitungan *loss*

```

def yolo_loss(args, anchors, num_classes, ignore_thresh=.5,
    print_loss=False):
    num_layers = len(anchors)//3
    yolo_outputs = args[:num_layers]
    y_true = args[num_layers:]
    anchor_mask = [[6,7,8], [3,4,5], [0,1,2]]
    input_shape = K.cast(K.shape(yolo_outputs[0])[1:3] * 32, K.
        dtype(y_true[0]))
    grid_shapes = [K.cast(K.shape(yolo_outputs[l])[1:3], K.
        dtype(y_true[0])) for l in range(num_layers)]
    loss = 0
    m = K.shape(yolo_outputs[0])[0] # batch size, tensor
    mf = K.cast(m, K.dtype(yolo_outputs[0]))

    for l in range(num_layers):
        object_mask = y_true[l][..., 4:5]
        true_class_probs = y_true[l][..., 5:]
        grid, raw_pred, pred_xy, pred_wh, _, _ = yolo_head(
            yolo_outputs[l],
            anchors[anchor_mask[l]], num_classes, input_shape,
            calc_loss=True)
        pred_box = K.concatenate([pred_xy, pred_wh])

        # Darknet raw box to calculate loss.
        raw_true_xy = y_true[l][..., :2]*grid_shapes[l][::-1] -
            grid
        raw_true_wh = K.log(y_true[l][..., 2:4] / anchors[
            anchor_mask[l]] * input_shape[::-1])
        raw_true_wh = K.switch(object_mask, raw_true_wh, K.
            zeros_like(raw_true_wh)) # avoid log(0)=-inf
        box_loss_scale = 2 - y_true[l][...,2:3]*y_true[l][...
            ,3:4]

        # Find ignore mask, iterate over each of batch.
        ignore_mask = tf.TensorArray(K.dtype(y_true[0]), size=1,
            dynamic_size=True)
        object_mask_bool = K.cast(object_mask, 'bool')

    def loop_body(b, ignore_mask):
        true_box = tf.boolean_mask(y_true[l][b, ..., 0:4],
            object_mask_bool[b, ..., 0])
        iou = box_iou(pred_box[b], true_box)
        best_iou = K.max(iou, axis=-1)
        ignore_mask = ignore_mask.write(b, K.cast(best_iou<
            ignore_thresh, K.dtype(true_box)))
        return b+1, ignore_mask

```

```

_, ignore_mask = K.control_flow_ops.while_loop(lambda
b,*args: b<m, loop_body, [0, ignore_mask])
ignore_mask = ignore_mask.stack()
ignore_mask = K.expand_dims(ignore_mask, -1)

# K.binary_crossentropy is helpful to avoid exp overflow.
xy_loss = object_mask * box_loss_scale * K.
binary_crossentropy(raw_true_xy, raw_pred[...,:2],
from_logits=True)
wh_loss = object_mask * box_loss_scale * 0.5 * K.square(
raw_true_wh-raw_pred[...,:2:4])
confidence_loss = object_mask * K.binary_crossentropy(
object_mask, raw_pred[...,:4:5], from_logits=True)+ \
(1-object_mask) * K.binary_crossentropy(object_mask,
raw_pred[...,:4:5], from_logits=True) * ignore_mask
class_loss = object_mask * K.binary_crossentropy(
true_class_probs, raw_pred[...,:5:], from_logits=True)

xy_loss = K.sum(xy_loss) / mf
wh_loss = K.sum(wh_loss) / mf
confidence_loss = K.sum(confidence_loss) / mf
class_loss = K.sum(class_loss) / mf
loss += xy_loss + wh_loss + confidence_loss + class_loss
return loss

```

Listing 3.3: Fungsi YOLO *loss*

3.4 Penerapan Optimalisasi Model

Percobaan ini dilakukan untuk menerapkan metode yang dapat meningkatkan kecepatan adan akurasi dari model.

3.4.1 Desain Model

Arsitektur yang baru akan memiliki tiga output dengan parameter yang lebih kecil dan menggunakan *feature extractor* MobileNet. Output pertama untuk mendeteksi ukuran besar dengan feature map dibagi menjadi 13×13 kotak memiliki 512 parameter, kedua untuk mendeteksi ukuran sedang dengan feature map dibagi menjadi 26×26 kotak memiliki 256 parameter dan ketiga untuk mendeteksi ukuran besar dengan feature map dibagi menjadi 52×52 kotak memiliki 128 parameter. Desain ini diharapkan menjadi model dengan ukuran kecil, komputasi yang lebih cepat dan akurasi yang lebih baik dari model yang sudah ada. Desain akhir dapat

dilihat pada gambar 3.7 dan fungsi 3.4.



Gambar 3.7: Arsitektur deteksi objek YOLO modifikasi dengan *Feature Extractor* Mobilenet

```
def mobilenet_yolo_body(inputs, num_anchors, num_classes):
    mobilenet = MobileNet(input_tensor=inputs, weights='imagenet
    ')
```

```

f1 = mobilenet.get_layer('conv_pw_13_relu').output
x, y1 = make_last_layers(f1, 512, num_anchors * (
    num_classes + 5))
x = compose(DarknetConv2D_BN_Leaky(256, (1,1)),UpSampling2D
(2))(x)

f2 = mobilenet.get_layer('conv_pw_11_relu').output
x = Concatenate()([x,f2])
x, y2 = make_last_layers(x, 256, num_anchors*(num_classes
+5))
x = compose(DarknetConv2D_BN_Leaky(128, (1,1)),UpSampling2D
(2))(x)

f3 = mobilenet.get_layer('conv_pw_5_relu').output
x = Concatenate()([x, f3])
x, y3 = make_last_layers(x, 128, num_anchors*(num_classes
+5))

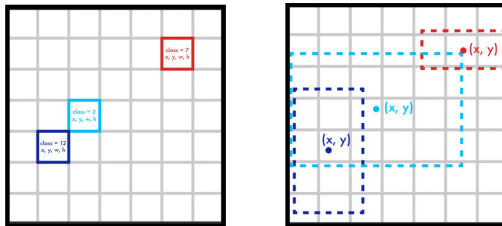
return Model(inputs = inputs, outputs=[y1,y2,y3])

```

Listing 3.4: Fungsi Mobilenet YOLO model

3.4.2 Penerapan *Knowledge Distillation* pada Object Detector

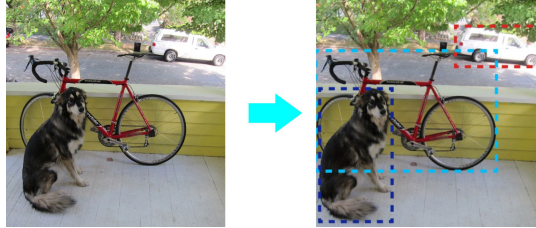
Proses Distilasi akan diterapkan dengan menggunakan YOLO sebagai *teacher* dan mobilenet YOLO sebagai *student*. Saat proses *training* biasa, tiap informasi objek label hanya akan dipelajari oleh skala dan *anchor* yang sesuai seperti pada gambar 3.8.



Gambar 3.8: Tiap *box* hanya untuk deteksi satu objek

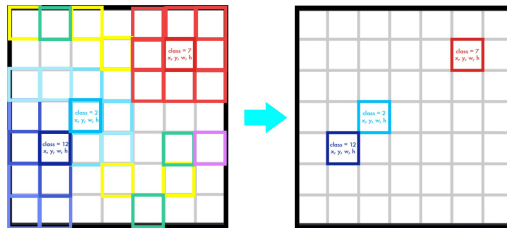
Hal ini agar hanya skala anchor yang tepat memiliki *confidence* yang paling tinggi untuk membedakannya dari prediksi anchor

dan skala lain saat *inference*. Sehingga hasil akhir sistem dapat menampilkan posisi dan bounding box yang tepat seperti pada gambar 3.9.



Gambar 3.9: Pendeteksian dengan ukuran box sesuai

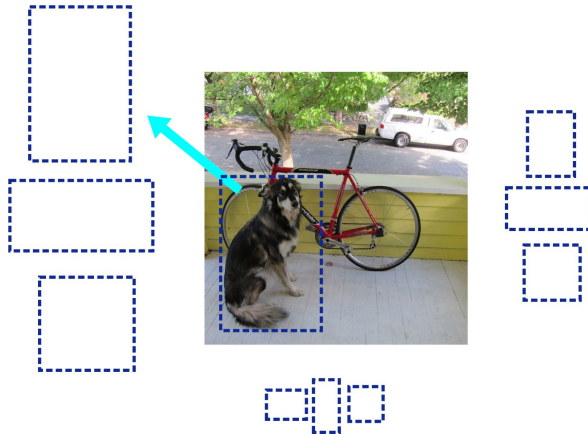
Distilasi akan dilakukan dengan mengambil *output* dari *teacher* untuk digunakan sebagai label untuk training *student*. Namun tidak seperti distilasi yang diterapkan pada klasifikasi, tidak semua output bisa digunakan untuk proses distilasi. Hal ini dikarenakan saat mendeteksi dan tidak ditemukan objek, YOLO akan mengklasifikasikannya sebagai *background*. YOLO menentukan apakah sebuah box adalah *background* atau tidak adalah dengan melihat seberapa besar nilai *confidence*-nya. Saat mendeteksi *box* dengan nilai *confidence* rendah akan dianggap sebagai *background* dan diabaikan. Saat melakukan distilasi, penggunaan data *background* dari *teacher* akan berdampak pada nilai posisi x, y, w, h dari box karena terlalu banyak variasi. Untuk menghindari distilasi menggunakan *background* dari *teacher*, maka label dari *teacher* perlu difilter terlebih dahulu berdasarkan nilai *confidence* label [25]. Proses filter *confidence* dapat diilustrasikan dengan gambar 3.10.



Gambar 3.10: Filter *Confidence* yang lebih rendah dari *threshold*

Selain *background*, *output teacher* juga dapat mengeluarkan nilai *anchor box* yang mendeteksi objek yang sama. Hal ini menyebabkan banyaknya informasi yang redundan. Hal ini dapat menyebabkan *overfitting* di objek dan dimensi yang sama [25]. Selain *overfitting* tiap *box* memiliki 3 *anchor* di tiap skala, jika satu objek yang sama dipelajari oleh anchor yang berbeda akan mengganggu hasil prediksi ukuran dan bentuk objek.

Oleh karena itu lebih baik dilakukan pemfilteran *output teacher* agar satu objek dideteksi oleh satu *anchor* sehingga label asli juga tetap bisa membantu distilasi disaat teacher mengeluarkan prediksi yang salah. Karena jika tiap anchor mempelajari objek yang sama hal ini akan berkontradiksi dengan tujuan awal pemisahan kategori berdasarkan skala dan *anchor* untuk menggolongkan objek berdasarkan bentuk agar informasi yang harus dipelajari tidak terlalu banyak. Untuk itu tiap objek hanya boleh dideteksi oleh satu *anchor* seperti pada gambar 3.11.



Gambar 3.11: Filter box atau *anchor* yang mendeteksi objek yang sama

Setelah mendapatkan label *ground truth* dan label dari teacher selanjutnya kita perlu memodifikasi rumus *loss*. Persamaan yang digunakan sama dengan rumus YOLO yang umum, perbedaannya di rumus distilasi adalah penambahan perhitungan *loss* dengan menggunakan label *teacher* sebagai pembanding dan penambahan *hyper-*

parameter α untuk meratakan *loss* teacher dan student. Selibhnya proses lainnya sama dengan proses *training* biasa. Persamaan distilasi adalah sebagai berikut (3.1) dan fungsi 3.5.

$$\begin{aligned}
 \mathcal{L}_{soft} &= f_{loc}(x_s, x_t, y_s, y_t, w_s, w_t, h_s, h_t) \\
 &\quad + f_{conf}(p_s, p_t) + f_{cls}(p_s, p_t) \\
 \mathcal{L}_{hard} &= f_{loc}(x_s, x_{true}, y_s, y_{true}, w_s, w_{true}, h_s, h_{true}) \\
 &\quad + f_{conf}(p_s, p_{true}) + f_{cls}(p_s, p_{true})
 \end{aligned} \tag{3.1}$$

$$\mathcal{L}_{YoloKD} = \alpha * \mathcal{L}_{soft} + (1 - \alpha) * \mathcal{L}_{hard}$$

Dengan,

\mathcal{L}_{soft} = nilai *loss student-teacher*

\mathcal{L}_{hard} = nilai *loss student-ground truth/asli*

x_s, y_s, w_s, h_s, p_s = nilai prediksi *student*

x_t, y_t, w_t, h_t, p_t = nilai prediksi *teacher*

$x_{true}, y_{true}, w_{true}, h_{true}, p_{true}$ = nilai label asli

$f_{loc}()$ = fungsi *localization error*

$f_{con}()$ = fungsi *confidence score loss*

$f_{cls}()$ = fungsi *classification loss*

α = hyperparameter alpha untuk distilasi

```

def yolo_distill_loss(args, anchors, num_classes,
    ignore_thresh=.5, alpha = 0, print_loss=False):

    num_layers = len(anchors)//3 # default setting
    yolo_outputs = args[:num_layers]#yolo output
    y_true = args[num_layers:num_layers*2]
    l_true = args[num_layers*2:]

    anchor_mask = [[6,7,8], [3,4,5], [0,1,2]] if num_layers==3
    else [[3,4,5], [1,2,3]]
    input_shape = K.cast(K.shape(yolo_outputs[0])[1:3] * 32, K.
        dtype(y_true[0]))
    grid_shapes = [K.cast(K.shape(yolo_outputs[l])[1:3], K.
        dtype(y_true[0])) for l in range(num_layers)]

    loss = 0
    m = K.shape(yolo_outputs[0])[0] # batch size, tensor
    mf = K.cast(m, K.dtype(yolo_outputs[0]))

```



```

for l in range(num_layers):
    #teacher
    gt_object_mask = y_true[l][..., 4:5]
    xy_loss , wh_loss , confidence_loss , class_loss ,
    ignore_mask = soft_target_yolo_loss(yolo_outputs[l],
    l_true[l], anchors[anchor_mask[l]], num_classes ,
    ignore_thresh , input_shape, grid_shapes[l], m, mf,
    gt_object_mask)

    loss += (alpha * (xy_loss + wh_loss + confidence_loss +
    class_loss))

    #student
    xy_loss , wh_loss , confidence_loss , class_loss ,
    ignore_mask = hard_target_yolo_loss(yolo_outputs[l],
    y_true[l], anchors[anchor_mask[l]], num_classes ,
    ignore_thresh , input_shape, grid_shapes[l], m, mf)

    loss += ((1-alpha) * (xy_loss + wh_loss + confidence_loss
    + class_loss))

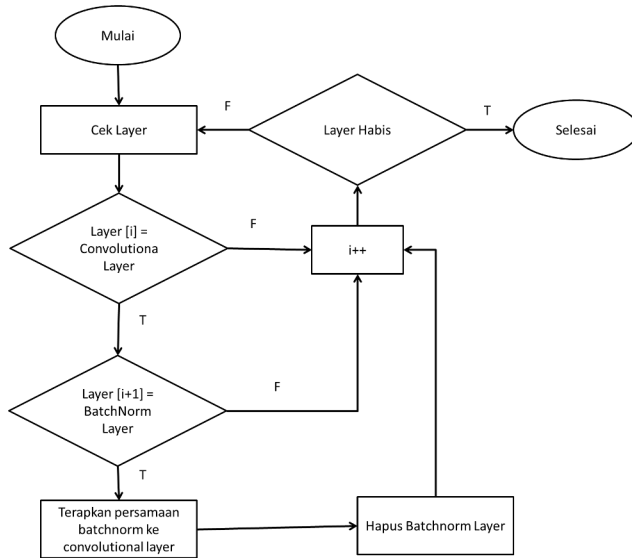
return loss

```

Listing 3.5: Fungsi *Knowledge Distillation loss* untuk YOLO

3.4.3 Penerapan *Batchnorm Fusion*

Setelah selesai *training* , Model bisa dipercepat lagi dengan menggunakan *batchnorm fusion*. Dengan *batchnorm fusion* jumlah *layer* pada model bisa dikurangi sehingga komputasi yang diperlukan untuk perpindahan *layer* bisa semakin cepat. Untuk penerapannya, pertama cari *Convolutional layer* dan *batch normalization* yang tersusun secara berurutan. Setelah ditemukan ambil parameter yang dimiliki *batch normalization* dan hapus *layer batch normalization*. Setelah itu gunakan parameter *batch normalization* untuk digabungkan ke rumus *weight* dan *bias* dari *Convolutional Layer* untuk membentuk rumus *weight* dan *bias* baru seperti pada persamaan (2.25). Setelah semua *batch normalization* telah digabung simpan konfigurasi terakhir model. Proses *batchnorm fusion* digambarkan dengan diagram alir 3.12 dan fungsi 3.6 .



Gambar 3.12: Diagram Alir *Batchnorm Fusion*

```

def optimize_conv2d_batchnorm_block(m, initial_model,
    input_layers, conv, bn, verbose=False):
    conv_layer_type = conv.__class__.__name__
    conv_config = conv.get_config()
    conv_config['use_bias'] = True
    bn_config = bn.get_config()

    # Copy Conv2D layer
    layer_copy = layers.deserialize({'class_name': conv.
        __class__.__name__, 'config': conv_config})
    layer_copy.name = bn.name

    # Create new model to initialize layer. We need to store
    other output tensors as well
    output_tensor, output_names = get_layers_without_output(m,
        verbose)
    input_layer_name = initial_model.layers[input_layers[0]].
        name
    prev_layer = m.get_layer(name=input_layer_name)
    x = layer_copy(prev_layer.output)

    output_tensor_to_use = [x]
  
```

```

for i in range(len(output_names)):
    if output_names[i] != input_layer_name:
        output_tensor_to_use.append(output_tensor[i])

if len(output_tensor_to_use) == 1:
    output_tensor_to_use = output_tensor_to_use[0]

tmp_model = Model(inputs=m.input, outputs=
    output_tensor_to_use)

if conv.get_config()['use_bias']:
    (conv_weights, conv_bias) = conv.get_weights()
else:
    (conv_weights,) = conv.get_weights()

if bn_config['scale']:
    gamma, beta, run_mean, run_std = bn.get_weights()
else:
    gamma = 1.0
    beta, run_mean, run_std = bn.get_weights()

eps = bn_config['epsilon']
A = gamma / np.sqrt(run_std + eps)

if conv.get_config()['use_bias']:
    B = beta + (gamma * (conv_bias - run_mean) / np.sqrt(
        run_std + eps))
else:
    B = beta - ((gamma * run_mean) / np.sqrt(run_std + eps))

if conv_layer_type == 'Conv2D':
    for i in range(conv_weights.shape[-1]):
        conv_weights[:, :, :, i] *= A[i]
elif conv_layer_type == 'DepthwiseConv2D':
    for i in range(conv_weights.shape[-2]):
        conv_weights[:, :, i, :] *= A[i]

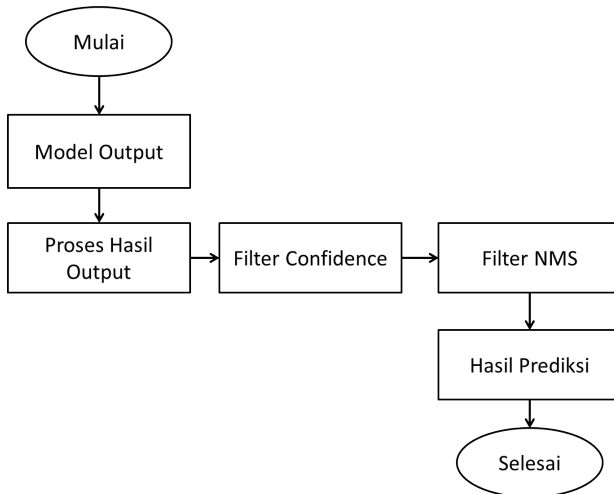
tmp_model.get_layer(layer_copy.name).set_weights((
    conv_weights, B))
return tmp_model

```

Listing 3.6: Fungsi *Batchnorm Fusion*

3.5 Sistem Deteksi Objek

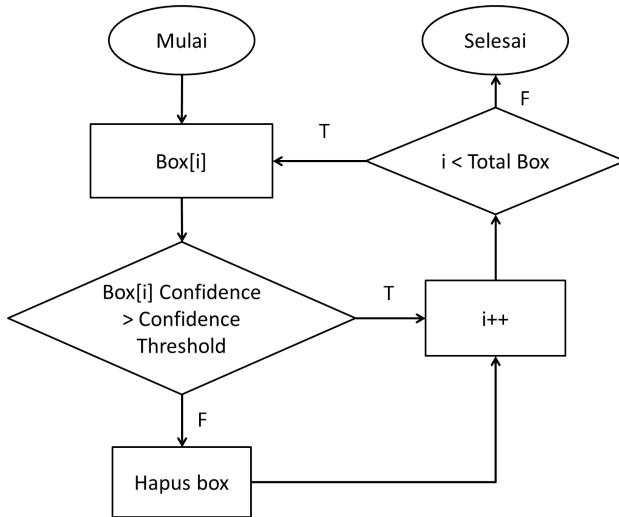
Sistem deteksi yang digunakan berasal dari model YOLO v3[1]. *Output* merupakan nilai arsitektur model. Setelah mendapatkan *output*, hasil kemudian harus di *filter* untuk mendapatkan hasil yang sesuai. Nilai *output* harus di proses terlebih dahulu untuk mendapatkan nilai yang sesuai. Lalu untuk menghilangkan prediksi yang salah , hasil harus difilter dengan *filter confidence* dan untuk menghilangkan duplikat hasil harus difilter dengan *filter nms*. Setelah itu hasil prediksi yang benar baru bisa diperoleh. Proses deteksi yang diperoleh dari arsitektur dapat dilihat pada gambar diagram 3.13.



Gambar 3.13: Diagram alir proses deteksi objek

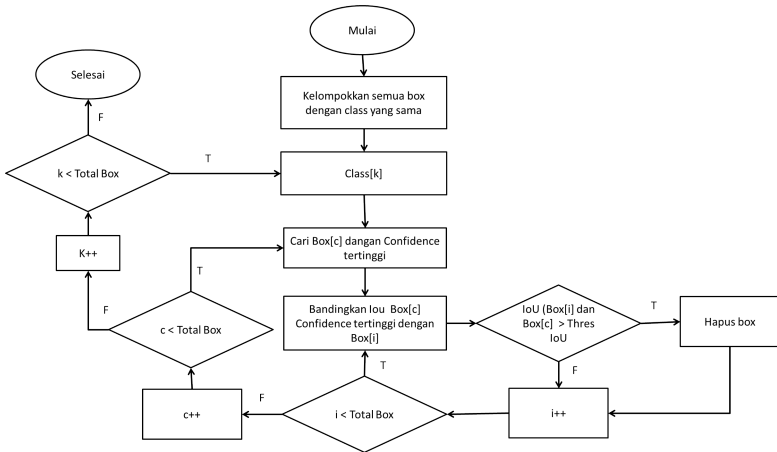
Dengan banyaknya hasil output maka besar kemungkinan tiap grid mendeteksi objek yang sama maupun , selain itu grid juga dapat salah mendeteksi. Untuk itu hasil dedeteksi perlu di filter untuk mendapatkan hasil yang lebih akurat. Untuk memfilter deteksi yang salah dilakukan dengan menghilangkan deteksi dengan nilai *confidence* yang rendah. Nilai *confidence* juga menunjukkan ada tidaknya suatu objek pada gambar. Jika tidak ada objek berarti *box* hanya berupa gambar *background*. Secara umum *threshold confidence* yang digunakan adalah 0.25 , Namun dapat diubah sesuai

dengan kebutuhan. Proses seleksi deteksi dengan nilai *confidence* dapat dijelaskan dengan gambar 3.14.



Gambar 3.14: Filter *Confidence*

Untuk menghilangkan deteksi objek yang sama, dapat menggunakan NMS *non-maximum suppression*. Metode NMS bekerja dengan cara memilih objek dengan nilai *confidence* yang tertinggi lebih dahulu, setelah itu kumpulkan semua kotak dengan nilai IoU lebih tinggi dari *threshold* yang ditentukan. Karena selisih IoU sedikit kemungkinan semua kotak tersebut mendeteksi objek sama sehingga perlu dihapus. Setelah semua kotak dengan IoU diatas *threshold* kotak yang dipilih telah terhapus, dipilih kotak tersisa yang memiliki nilai *confidence* paling tinggi berikutnya dan proses diulang kembali dari awal. Proses ini dilakukan ke semua objek yang terdeteksi. Secara umum *threshold confidence* yang digunakan adalah 0.5. Proses seleksi deteksi dengan NMS dapat dijelaskan dengan gambar 3.15.



Gambar 3.15: Filter NMS (*Non-Max Supression*)

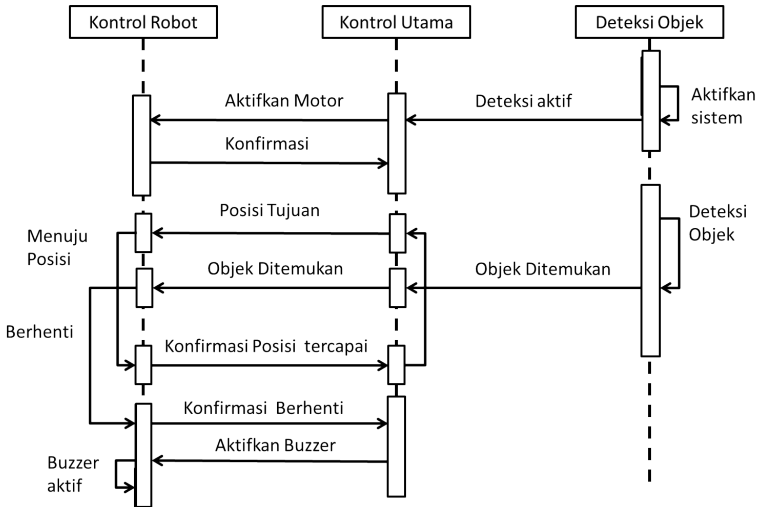
3.6 Sistem Kontrol Robot

Sistem kontrol robot akan menjelaskan proses yang dilakukan robot dalam pencarian sebuah objek dan interaksi yang diperlukan antar sistem dan antar perangkat untuk menerapkannya. Komponen utama robot menggunakan Makeblock dengan *mainboard* Auriga. *Board* Auriga memiliki desain Arduino Mega yang dimodifikasi.

3.6.1 Pencarian Objek

Untuk melakukan pencarian objek, ada beberapa langkah yang harus dilakukan. Pertama program pendeteksi objek akan diaktifkan terlebih dahulu. Setelah sistem pendeteksi objek aktif, maka sistem kontrol utama akan aktif dan mulai memberi perintah pada robot. Sistem kontrol utama akan memberikan arah yang harus dicapai oleh sistem kontrol robot. Sistem akan menjalankan robot hingga mencapai posisi yang ingin dituju. Proses ini akan terus diulang hingga semua arah telah dilaksanakan. Saat proses ini berlangsung sistem deteksi objek terus beroperasi untuk mencari objek. Saat objek ditemukan sistem akan mengirim pesan sistem kontrol utama. Setelah itu sistem kontrol utama akan memerintah sistem kontrol robot untuk menghentikan proses yang sedang dila-

kukan sebelumnya dan berhenti dan mengaktifkan *buzzer*. Sistem dijelaskan dengan diagram 3.16 dalam format *sequence diagram*.



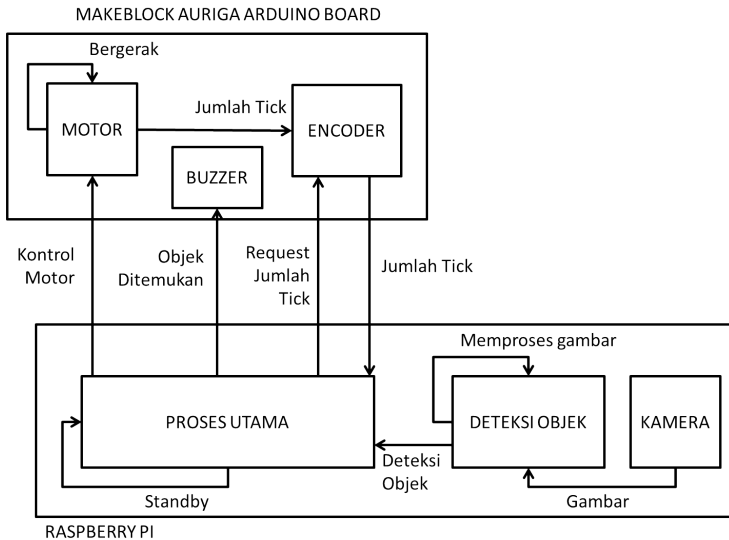
Gambar 3.16: Sekuensial Diagram Pencarian Objek

3.6.2 *Multiprocessing*

Robot harus menjalankan beberapa program secara bersamaan dan prosesnya tidak boleh diganggu. Untuk menjalankan kedua program secara bersamaan maka sistem harus dijalankan secara *multiprocessing*. Dengan *multiprocessing*, semua program dapat dijalankan secara bersamaan dan dapat berinteraksi hanya saat diperlukan dan tidak akan mengganggu kinerja satu sama lain.

Robot terdiri dari dua proses, proses yang mengontrol pergerakan robot dan proses yang mengatur pendeteksian objek. Proses pergerakan robot terjadi di Auriga, namun *Raspberry* yang memberikan perintah kepada *Auriga* dan *Auriga* menjalankan perintah seperti mengaktifkan *motor* atau memberikan nilai *sensor* sesuai dengan perintah yang dikirim melalui serial. Untuk proses pendeteksian objek akan dijalankan dengan proses terpisah dan proses tersebut akan berinteraksi dengan kamera untuk memperoleh data gambar. Kedua proses tersebut akan dikontrol dan diawasi oleh proses uta-

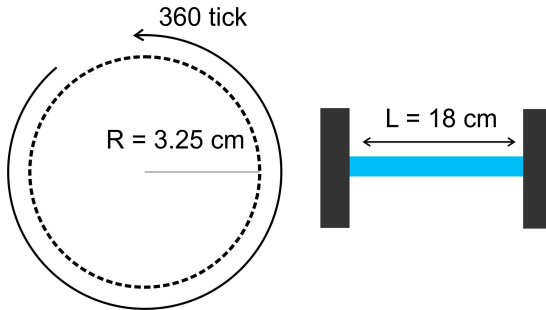
ma. Proses utama akan mengatur pergerakan robot sesuai dengan rute yang telah ditentukan. Proses tiap sistem dan interaksinya untuk lebih jelasnya bisa dilihat pada gambar 3.17.



Gambar 3.17: Blok Diagram Interaksi Antar Sistem

3.6.3 Penerapan *Odometry*

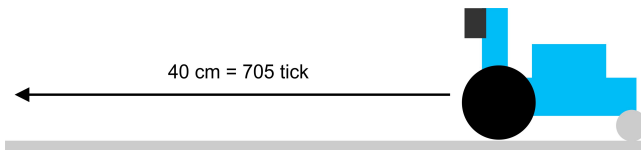
Untuk mengontrol jarak perpindahan robot akan diterapkan rumus *odometry*. Informasi yang diperlukan untuk melakukan pengukuran adalah jari=jari roda(R), jarak antar kedua roda (L) dan jumlah *tick* yang diperlukan untuk melakukan 1 rotasi. Setelah dilakukan pengukuran maka diperoleh jari-jari(R) roda sebesar 3.25 cm, jarak antar kedua roda (L) sebesar 18 cm dan jumlah tick sebesar 360 *tick*. Gambaran tentang pengukuran roda dapat dilihat pada gambar 3.18



Gambar 3.18: Ukuran Roda

Informasi pengukuran tersebut bisa digunakan untuk menentukan jumlah *tick* atau jumlah rotasi yang diperlukan robot untuk mencapai suatu posisi yang jaraknya sudah ditentukan sebelumnya. Untuk bergerak maju dan mundur jumlah *tick* antara roda kiri dan kanan akan berjumlah sama. Untuk belok kekanan maka roda sebelah kiri akan berotasi lebih banyak, sebaliknya untuk belok ke kanan maka roda kanan akan berotasi lebih banyak. Semakin jauh posisi robot dari posisi awal maka jumlah *tick* akan semakin banyak. Nilai hasil perhitungan *tick* akan selalu dibulatkan ke bilangan diskrit. Untuk contoh perhitungan *odometry* bisa dilihat pada perhitungan (3.2) dan ilustrasi pergerakan robot berdasarkan jumlah *tick* untuk mencapai posisi baru bisa dilihat pada gambar 3.19.

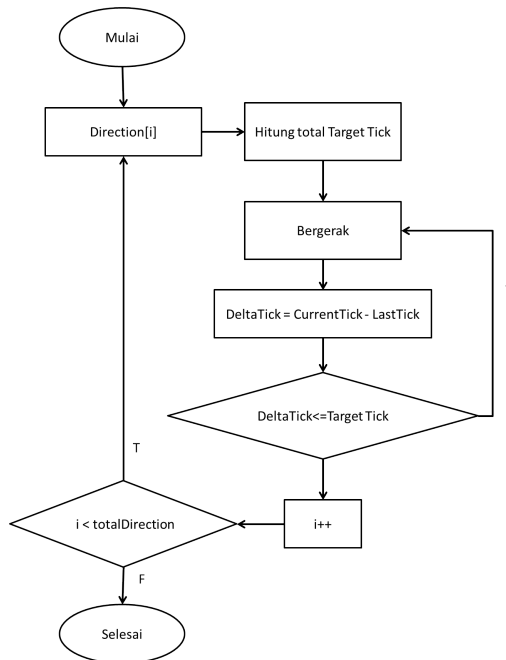
$$Tick = \frac{D * FullTick}{2\pi R} \quad Tick = \frac{40 * 360}{2 * 3.14 * 3.25} = 705 \quad (3.2)$$



Gambar 3.19: Menentukan jumlah tick berdasarkan jarak

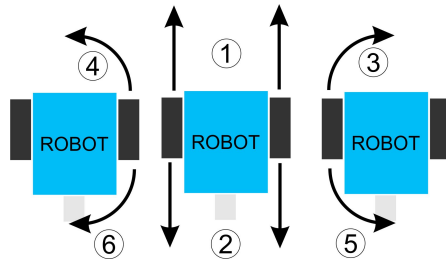
Gerakan robot diatur dengan sekumpulan *array* perintah yang berisi arah dan jarak yang harus dituju robot. Robot akan membaca

perintah dan melakukan eksekusi. Saat bergerak robot akan terus mengawasi jumlah *tick* yang terus berubah. Jika jumlah *tick* sudah sesuai dengan target maka robot akan membaca perintah selanjutnya. Setelah membaca arahan berikutnya, robot akan bergerak hingga target selanjutnya hingga semua *array* telah dilakukan. Pengecekan target sendiri dilakukan dengan mengurangi nilai *tick* saat ini dan nilai *tick* yang di simpan saat menjalankan perintah terakhir lalu dibandingkan dengan target *tick*. Secara teori robot seharusnya berhenti disaat jumlah *tick* sama dengan target. Namun karena *encoder* tidak dapat menghitung *tick* dengan akurat maka diberikan toleransi 20 *tick* lebih besar dan 20 *tick* lebih kecil dari target asli. Pengecekan 2 lebih besar dan lebih kecil ini juga dilakukan untuk mengantisipasi arah perpindahan *tick* dari negatif ke positif atau positif ke negatif. Alir Diagram mengatur pergerakan robot dapat dilihat pada diagram 3.20.



Gambar 3.20: Alir Diagram Proses Perpindahan Robot

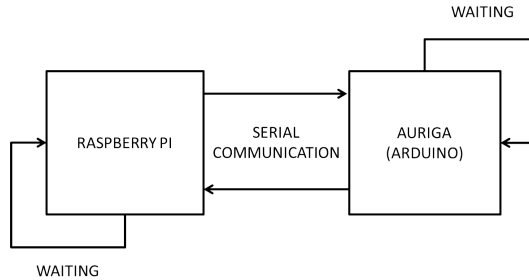
Untuk maju dan mundur kedua roda akan bergerak ke arah yang sama sesuai dengan arah yang ingin dituju. Untuk melakukan maju ke kiri dan mundur kekiri hanya roda kanan yang aktif bergerak dan roda kiri diam menjadi titik pusat. Sedangkan Untuk melakukan maju ke kanan dan mundur kekanan hanya roda kiri yang aktif bergerak dan roda kanan diam menjadi titik pusat. Pada gambar 3.21 menunjukkan arah rotasi kedua roda untuk menuju suatu arah.



Gambar 3.21: Kontrol Arah Pergerakan Robot

3.6.4 Penerapan Komunikasi Serial

Komunikasi antar prangkat dilakukan secara serial. Untuk melakukan komunikasi serial bisa dilakukan dengan dua cara, melalui kabel USB atau *Bluetooth Module HSC-05*. Cara kerja komunikasinya Auriga akan dalam keadaan *standby* dan menunggu perintah. *Raspberry* akan mengirimkan pesan serial kepada Auriga lalu menuju *standby* untuk menunggu respon. Jika Auriga menerima pesan tersebut, maka Auriga akan mengirimkan respon ke *Raspberry* dan menjalankan program sesuai isi pesan. Ada dua jenis perintah yang dapat dikirim oleh *Raspberry*, pertama adalah perintah untuk menjalankan komponen di Auriga seperti motor dan *buzzer*. Perintah ini hanya akan menerima respon konfirmasi jika perintah telah diterima. Kedua adalah perintah *Request* dengan *callback*. Perintah ini digunakan untuk memperoleh nilai sensor seperti sensor ultrasonik dan *encoder*. Respon yang diterima pesan ini akan berisi nilai sensor yang ingin dibaca. Ilustrasi proses komunikasi serial dapat dilihat pada gambar 3.22.



Gambar 3.22: Serial Komunikasi antara Raspberry dan Auriga

Pesan yang melewati komunikasi serial berupa bilangan *byte*. Untuk membedakan isi pesan yang satu dengan yang lain tiap perintah akan dikirim dalam beberapa byte lalu kemudian di parsing untuk mengetahui isi pesan.

Tabel 3.2: Format pengiriman data dari Raspberry ke Arduino

FF	55	len	idx	action	device	port	slot	data
0	1	2	3	4	5	6	7	8
Contoh:								
FF	55	06	idx	01	device id	00	slot	00
Request Encoder Motor Position								
FF	55	06	idx	02	device id	01	slot	dist speed
Encoder Motor Position								

Contoh format pengiriman pesan dari *Raspberry* ke *Arduino* dapat dilihat pada 3.2. Pengiriman pesan dari *Raspberry* ke *Arduino* memiliki format data dengan ukuran terdiri dari beberapa *byte*. Dua *byte* pertama adalah *header* untuk penanda awal dari pesan. Selanjutnya adalah *byte* dengan informasi jumlah banyaknya *byte* yang akan dikirim berikutnya. Setelah mengetahui jumlah *byte* pesan, maka arduino siap melakukan parsing pesan. secara urutan pesan *byte* terdiri dari id pesan, jenis perintah, perangkat, port, slot dan data. Id pesan adalah nomor unik dari pesan tersebut. Jenis perintah terdiri dari 4 tipe *GET* untuk meminta informasi, *RUN* untuk menjalankan perangkat, *RESET* untuk mengulang seluruh proses

sitem dan *START* untuk mengaktifkan robot. Informasi selanjutnya adalah penjelasan alamat dari perangkat yang ingin di kontrol.

Tabel 3.3: Format pengiriman data dari Arduino ke Raspberry

Respon tanpa data									
255 85 13 10									
Respon dengan data									
255 85 extID type Data 13 10									
Contoh respon dari encoder									
255 85 77 06 218 02 00 00 13 10									
Konversi									
00 00 02 218									
0x000002DA = 730									

Contoh format pengiriman pesan dari *Arduino* ke *Raspberry* dapat dilihat pada 3.2. Pengiriman pesan dari *Arduino* ke *Raspberry*, format pengiriman pesannya hampir sama dengan saat dari *Raspberry* ke *Arduino*, dua *byte* pertama berupa penanda awal pesan yang terdiri dari 255 dan 85 jika diubah ke *integer*. Jika pesan hanya berupa konfirmasi dari auriga maka *byte* berikutnya hanya bertujuan untuk menutup pesan. Penutup pesan terdiri dari dua *byte* yaitu 13 dan 10 jika diubah ke *integer*. Dengan kedua penanda tersebut program dapat menentukan titik awal dan akhir tiap pesan dalam komunikasi serial untuk mempermudah saat parsing. Jika pesan berisi sebuah nilai, maka setelah dua *byte* pembuka akan diisi *byte* berupa informasi yang ingin dikirim setelah itu disusul dengan dua *byte* penutup. Isi pesan akan diawali dengan *byte* yang berisi informasi tentang Id pesan saat raspberry mengirim ke arduino. selanjutnya disusul dengan *byte* yang berisi informasi tentang tipe variabel nilai yang ingin dikirim seperti *byte*(1), *float*(2), *short*(3), *string*(4), *double*(5), *long*(6). *byte* membutuhkan 1 *byte*, *short* membutuhkan 2 *byte*, *float*, *double*, dan *long* membutuhkan 4 *byte* dan *string* membutuhkan *byte* sebanyak jumlah *char* yang dikirim. Nilai *byte* tersebut kemudian akan di ubah menjadi nilai tipe data masing-masing.

BAB 4

PENGUJIAN DAN ANALISA

Pada penelitian ini, dilakukan beberapa pengujian serta analisa dari desain sistem dan implementasinya. Percobaan dilakukan dengan menggunakan beberapa arsitektur model. Proses *training* data dan distilasi akan dilakukan dengan dataset VOC 2012 untuk label *training* dan *validation* lalu VOC 2007 untuk *testing*. Uji coba performa akan dilakukan di CPU (*Central Processing Unit*) laptop dan Raspberry. Robot menggunakan Makeblock Auriga dan dikontrol oleh Raspberry yang dilengkapi dengan webcam sebagai kamera. Pengujian yang dilakukan akan dibagi menjadi beberapa tahap sebagai berikut:

1. Pengujian perbedaan jenis model
2. Pengujian dataset objek untuk manula
3. Pengujian di berbagai prosesor
4. Pengujian dengan berbagai ukuran *input*
5. Pengujian perbedaan jumlah skala
6. Pengujian *knowledge distillation* pada deteksi objek
7. Pengujian pengaruh *batchnorm fusion*
8. Pengujian deteksi *real-time* pada Robot

4.1 Pengujian di Berbagai Jenis Model

Pengujian berbagai jenis model bertujuan untuk membandingkan keunggulan dan kelemahan tiap model. Semua jenis model yang untuk percobaan ini akan dibandingkan dari segi ukuran, kecepatan FPS, jumlah *layer*, jumlah parameter, dan akurasi mAP. Untuk mengukur mAP menggunakan rumus (2.20) yang sudah dijelaskan di bab 3. Untuk evaluasi *IoU threshold* yang akan digunakan adalah 0.45 dan *Confidence Threshold* adalah 0.5. Nilai tersebut juga akan digunakan di semua perbandingan dengan mAP. Untuk perbandingan semua model akan menggunakan *input* 416x416 pixel. Lalu untuk mengukur FPS akan dilakukan di CPU i7.

Saat membandingkan struktur model, Model dengan ukuran terkecil adalah MobileNet YOLO(21MB) dengan 21MB dan 5.530.401 parameter sedangkan jumlah layernya 115. Namun untuk jumlah *layer* yang paling sedikit adalah Tiny YOLO dengan 44 dengan ukur-

an 33MB dan 8.858.734 parameter. Mobilenet YOLO(21MB) dapat lebih kecil dikarenakan *Depthwise Separable Convolution* yang memperkecil parameter namun akibatnya jumlah layer menjadi hampir dua kali lipat lebih banyak. Hasil pengujian bisa dilihat pada tabel 4.1.

Tabel 4.1: Hasil perbandingan struktur model

Model	Size	Layer	Params
YOLO	236 MB	252	61.678.657
Tiny YOLO	33 MB	44	8.858.734
Tiny YOLO (3 scale)	33 MB	56	8.419.793
Mobilenet YOLO	92 MB	154	24.286.881
Mobilenet YOLO(21MB)	21 MB	115	5.530.401

Saat mencoba performa model, Model dengan mAP paling tinggi adalah YOLO dengan 0.6756 mAP namun memiliki FPS paling rendah yaitu 0.576 FPS. Tiny YOLO memiliki FPS tercepat yaitu 3.85 FPS namun memiliki mAP terendah sebesar 0.2343 mAP. Jika dimodifikasi menjadi 3 skala mAP nya meningkat menjadi 0.3817 mAP dan FPS nya berkurang menjadi 3.57 FPS. Sementara untuk model dengan ukuran paling kecil Mobilenet YOLO(21MB) memiliki mAP sebesar 0.3843 mAP dan FPS sebesar 2.5 FPS. Hasil pengujian bisa dilihat pada tabel 4.2.

Tabel 4.2: Hasil perbandingan performa model

Model	mAP	FPS
YOLO	0.6756	0.576
Tiny YOLO	0.2343	3.85
Tiny YOLO (3 scale)	0.3817	3.57
Mobilenet YOLO(92MB)	0.4361	1.15
Mobilenet YOLO(21MB)	0.3843	2.5

4.2 Pengujian Dataset Objek untuk Manula

Pengujian dataset objek untuk manula bertujuan untuk melihat hasil akurasi mAP yang diperoleh dengan dataset dataset objek barang manula. Objek yang dapat dideteksi adalah kunci, sepa-

tu/sandal, kacamata, dompet dan *smartphone*. Model yang digunakan untuk uji coba adalah YOLO, Tiny YOLO, dan Mobilenet YOLO(21MB).

Pretrain yang digunakan untuk percobaan ini adalah Imagenet. *Pretrain* Imagenet hanya memberikan nilai di bagian *Feature Extractor*. Setelah dilakukan *training* YOLO memiliki 0.5268 mAP, Tiny YOLO 2 skala memiliki 0.1234 mAP, Tiny YOLO 3 skala memiliki 0.1469 mAP dan Mobilenet YOLO(21MB) memiliki 0.2516 mAP. Hasil pengujian bisa dilihat pada tabel 4.3.

Tabel 4.3: Hasil performa model dataset objek untuk manula (pretrain Imagenet)

Model	mAP
YOLO	0.5268
Tiny YOLO	0.1234
Tiny YOLO (3 scale)	0.1469
Mobilenet YOLO(2MB)	0.2516

Sedangkan *pretrain* yang digunakan untuk percobaan ini adalah VOC. Pretrain ini diperoleh dari hasil *training* yang sebelumnya. *Pretrain* VOC memberikan nilai di bagian *Feature Extractor* dan *Object Classifier*. Setelah dilakukan *training* YOLO memiliki 0.5660 mAP, Tiny YOLO 2 skala memiliki 0.1365 mAP, Tiny YOLO 3 skala memiliki 0.2445 mAP dan Mobilenet YOLO(21MB) memiliki 0.3164 mAP. Beberapa model mengalami peningkatan akurasi saat training dengan *pretrain* VOC dibandingkan *pretrain* Imagenet. Hasil pengujian bisa dilihat pada tabel 4.4.

Tabel 4.4: Hasil performa model dataset objek untuk manula (pretrain VOC)

Model	mAP
YOLO	0.5660
Tiny YOLO	0.1365
Tiny YOLO (3 scale)	0.2445
Mobilenet YOLO(21MB)	0.3164

4.3 Pengujian di Berbagai Prosesor

Pengujian di berbagai prosesor bertujuan untuk mengetahui perbedaan performa model saat di jalankan di prosesor yang berbeda-beda. Untuk prosesor yang akan digunakan adalah i7, i3 , raspberry dan GPU(*Graphics Processing Unit*). Selain itu jika untuk hasil FPS biasa tanpa proses apapun *Raspberry* hanya dapat menghasilkan 8 FPS ,sedangkan pada CPU lain FPS standarnya bisa mencapai 24 FPS. Ukuran *input* gambar yang digunakan adalah ukuran standar yaitu gambar yang di *resize* menjadi 416x416 pixel. Untuk detail prosesor dapat dilihat pada tabel 4.5

Tabel 4.5: Perbandingan Spesifikasi

Jenis	Prosesor	Frekuensi	Core	Ram
SAMSNG NP400B5B	Intel i7-2670QM	2.20 GHz	4	8GB
ASUS X452C	Intel i3 3217U	1.80 GHz	2	4GB
RASPBERRY	Cortex-A53	1.4GHz	4	1GB

Berdasarkan hasil pada tabel 4.6 , GPU memiliki memiliki performa paling tinggi. Sepertinya karena ukuran gambar yang terlalu besar nilai FPS di i7, i3 dan Raspberry tidak terlalu besar. CPU i7 dapat menjalankan program hingga 3.85 FPS, CPU i3 dapat menjalankan program hingga 1.237 FPS dan Raspberry dapat menjalankan program hingga 0.416 FPS

Tabel 4.6: Hasil pengukuran FPS di berbagai prosesor

Model	Size	FPS			
		GPU	i7	i3	Raspi
YOLO	236 MB	11.22	0.576	0.206	-
Tiny YOLO	33 MB	20.61	3.85	1.237	0.416
Mobilenet YOLO(92MB)	92 MB	18.06	1.15	0.483	0.153
Mobilenet YOLO(21MB)	21 MB	20.51	2.5	0.909	0.3125

4.4 Pengujian dengan Berbagai Ukuran *Input*

Pengujian di berbagai prosesor bertujuan untuk mengetahui pengaruh ukuran pixel *input* gambar pada performa model. Ukur-

an *input* harus merupakan kelipatan 32 dan jika dibagi dengan 32 merupakan nilai ganjil. Ketentuan itu adalah dasar dari arsitektur YOLO. Alasan kenapa harus bilangan ganjil agar saat dibuat grid akan ada titik box pusat. Uji coba akan dilakukan dengan menggunakan standar ukuran 416, 288, 224 dan 160. Model yang kan digunakan untuk uji coba adalah model berukuran kecil yang dapat dijalankan di raspberry. Untuk pengamatan yang akan dibandingkan adalah akurasi mAP dan kecepatan FPS tiap model di berbagai ukuran.

Berdasarkan hasil pada tabel 4.7, *input* dengan ukuran 416 memiliki mAP paling besar dibandingkan *input* yang lain namun memiliki FPS paling kecil. Jika semua model dibandingkan antar satu sama lain dan saat diubah ukuran, semua model memiliki kesamaan. Semakin besar ukuran *input* maka mAP akan semakin tinggi, sedangkan FPS akan semakin kecil. Sebaliknya jika ukuran *input* semakin kecil maka akurasi akan semakin rendah, namun FPS akan semakin cepat. Hal ini dikarenakan jika ukuran semakin besar informasi yang diperoleh akan semakin banyak, namun waktu yang diperlukan untuk pengolahan akan semakin lama.

Tabel 4.7: Hasil perbandingan berbagai ukuran *Input*

Input Size	Model	mAP	FPS
416	Tiny YOLO	0.2343	3.85
	Tiny YOLO (3 scale)	0.3817	3.57
	Mobilenet YOLO(21MB)	0.3850	2.5
288	Tiny YOLO	0.1714	6.67
	Tiny YOLO (3 scale)	0.3395	7.14
	Mobilenet YOLO(21MB)	0.3432	4.44
224	Tiny YOLO	0.1946	11
	Tiny YOLO (3 scale)	0.2995	11
	Mobilenet YOLO(21MB)	0.3068	8.3
160	Tiny YOLO	0.1180	16.67
	Tiny YOLO (3 scale)	0.2013	16.67
	Mobilenet YOLO(21MB)	0.2080	11.11

4.5 Pengujian Perbedaan Jumlah Skala

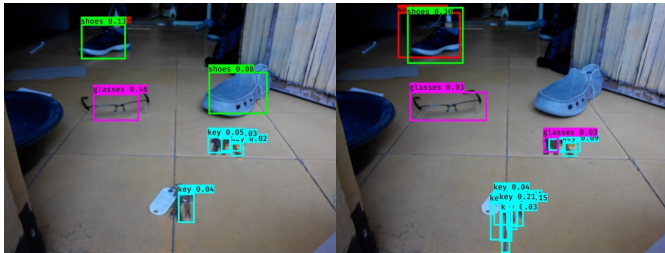
Pengujian ini dilakukan untuk mengamati pengaruh jumlah skala pada hasil deteksi objek. Pengaruh yang bisa diamati paling jelas adalah jumlah objek yang terdeteksi dan jarak objek. Penambahan skala ukuran kecil dapat meningkatkan jumlah objek yang berukuran kecil ataupun benda yang berada di posisi yang jauh. *Threshold Confidence* untuk memfilter nilai objek akan diatur sebesar 0.2 dan *threshold IoU* untuk menghilangkan duplikat objek akan diatur sebesar 0.45 untuk perbandingan. Gambar yang dijadikan uji coba adalah gambar jalan raya dengan objek yang banyak dan memiliki berbagai ukuran dan posisi objek. Gambar Objek yang terlihat adalah orang dan mobil, objek tersebut dipilih karena paling mudah dikenali, karena memiliki jumlah dataset lebih banyak dibandingkan yang lain. Selain itu fokus utamanya adalah membandingkan jumlah objek yang terdeteksi dan jarak benda yang dapat terdeteksi. Sehingga objek yang sejenis akan mempermudah dalam mengamati hasil percobaan.

Tabel 4.8: Hasil perbandingan jumlah deteksi objek dengan skala yang berbeda

Model	Scale	Objek Terdeteksi
Mobilenet YOLO(21MB)	3	6
Tiny YOLO	3	5
Tiny YOLO	2	2

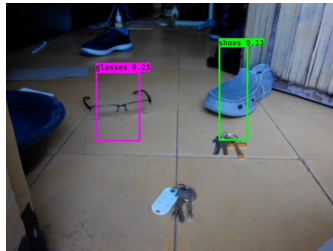
Berdasarkan tabel 4.8 , untuk model yang arsitekturnya di desain dengan 3 skala seperti Mobilenet YOLO(21MB) dapat mendeteksi 6 objek dan Tiny YOLO dapat mendeteksi 5 objek. Sedangkan Tiny YOLO dengan 2 skala hanya dapat mendeteksi 2 objek saja. Seperti yang terlihat pada gambar 4.1 dan gambar 4.2 perbedaan yang paling mencolok adalah Tiny YOLO dengan 2 skala tidak dapat mendeteksi objek yang berukuran kecil seperti kunci seperti sepatu yang berwarna hitam dibelakang dan kunci. Mobilenet YOLO(21MB) berhasil mendeteksi semua objek yang ada pada gambar. Tiny YOLO dengan 3 skala tidak dapat mendeteksi sepatu yang berwarna abu-abu namun dapat mendeteksi semua kunci di tiap set kunci, namun Tiny YOLO dengan 3 skala juga mel-

kukan kesalahan seperti menganggap kunci sebagai kacamata. Tiny YOLO dengan 2 skala hanya dapat mendeteksi objek yang besar yaitu kacamata dan sepatu abu-abu. Sepatu di bagian belakang dan kunci tidak terdeteksi oleh Tiny YOLO dengan 2 skala.



(a) Mobilenet YOLO(21MB) (3 scale) (b) Tiny YOLO (3 scale)

Gambar 4.1: Deteksi dengan model yang memiliki 3 skala



(a) Tiny YOLO (2 scale)

Gambar 4.2: Deteksi dengan model yang memiliki 2 skala

Gambar 4.3 dan Gambar 4.4 menunjukkan perbandingan jarak terjauh objek yang dapat dideteksi oleh robot dengan berbagai skala dan ukuran *input* gambar. Untuk perbandingan yang lebih jelasnya dapat dilihat pada gambar. Dengan menggunakan 3 skala dan ukuran *input* 416 pixel jarak yang terjauh adalah 180 cm. Sedangkan jika ukuran *input* diperkecil menjadi 224 maka jaraknya berkurang menjadi 160 cm. Jika menggunakan 2 skala dan ukuran *input* 416 pixel jarak yang terjauh adalah 140 cm. Sedangkan jika

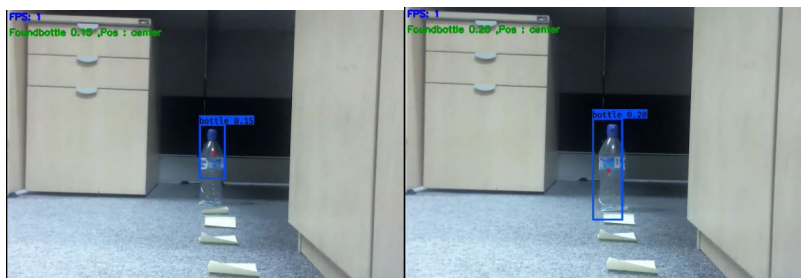
ukuran *input* diperkecil menjadi 224 maka jaraknya berkurang menjadi 120 cm. Hasil pengamatan menunjukkan jarak terjauh yang dapat dideteksi oleh sistem dapat ditingkatkan dengan penambahan skala ketiga untuk mendeteksi objek kecil.



(a) Scale 3 Input 416 180 cm

(b) Scale 3 Input 224 160 cm

Gambar 4.3: Perbandingan jarak deteksi objek dengan menggunakan 3 skala pada robot



(a) Scale 2 Input 416 140 cm

(b) Scale 2 Input 224 120 cm

Gambar 4.4: Perbandingan jarak deteksi objek dengan menggunakan 2 skala pada robot

4.6 Pengujian *Knowledge Distillation* pada Deteksi Objek

Pengujian distilasi pada deteksi objek dilakukan untuk mendapat hasil distilasi terbaik pada pendeteksi objek YOLO dengan mencoba parameter yang berbeda-beda. Semua model akan di *training* sebanyak 200 *epoch*. *Learning rate* akan diatur sebesar 1×10^{-3} yang kemudian diperkecil terus-menerus setiap beberapa *epoch*. Optimizer yang digunakan adalah adam. Model yang akan digunakan pada percobaan ini adalah YOLO sebagai *teacher* dan Mobilenet YOLO(21MB) sebagai *student*.

Pengujian pertama adalah distilasi pada objek deteksi menggunakan dataset VOC dan *pretrain* Imagenet. Setelah itu dilakukan uji coba dengan menggunakan *teacher* yang memiliki 0.6756 mAP dan *student* yang memiliki 0.3850 mAP. Hasilnya saat nilai alpha 0.1 akurasi yang diperoleh adalah 0.4126 mAP. Saat nilai alpha 0.2 akurasi yang diperoleh adalah 0.4215 mAP. Saat nilai alpha 0.3 akurasi yang diperoleh adalah 0.3963 mAP. Saat nilai alpha 0.5 akurasi yang diperoleh adalah 0.3968 mAP. Hasil pengujian bisa dilihat pada tabel 4.9 untuk model *student* dan *teacher* dan tabel 4.10 untuk pengujian distilasi.

Tabel 4.9: Teacher dan Student untuk percobaan distilasi pada objek deteksi (Dataset VOC)

Status	Model	mAP	Size
Teacher	YOLO	0.6756	236MB
Pure	Mobilenet YOLO(21MB)	0.3850	21MB

Tabel 4.10: Hasil perbandingan distilasi pada objek deteksi (Dataset VOC)

Model	Alpha	mAP	Size
Mobilenet YOLO(21MB)	0.1	0.4126	21MB
Mobilenet YOLO(21MB)	0.2	0.4215	21MB
Mobilenet YOLO(21MB)	0.3	0.3963	21MB
Mobilenet YOLO(21MB)	0.5	0.3968	21MB

Pengujian pertama adalah distilasi pada objek detek-

si menggunakan dataset Manula dan *pretrain* Imagenet. Setelah itu dilakukan uji coba dengan menggunakan *teacher* yang memiliki 0.5268 mAP dan *student* yang memiliki 0.2516 mAP. Hasilnya saat nilai alpha 0.1 akurasi yang diperoleh adalah 0.3462 mAP. Saat nilai alpha 0.2 akurasi yang diperoleh adalah 0.3416 mAP. Saat nilai alpha 0.3 akurasi yang diperoleh adalah 0.4047 mAP. Saat nilai alpha 0.5 akurasi yang diperoleh adalah 0.3248 mAP. Hasil pengujian bisa dilihat pada tabel 4.11 untuk model *student* dan *teacher* dan tabel 4.12 untuk pengujian distilasi.

Tabel 4.11: Teacher dan Student untuk percobaan distilasi pada objek deteksi (Dataset Manula) (pretrain Imagenet)

Status	Model	mAP	Size
Teacher	YOLO	0.5268	236MB
Pure	Mobilenet YOLO(21MB)	0.2516	21MB

Tabel 4.12: Hasil perbandingan distilasi pada objek deteksi (Dataset Manula) (pretrain Imagenet)

Model	Alpha	mAP	Size
Mobilenet YOLO(21MB)	0.1	0.3462	21MB
Mobilenet YOLO(21MB)	0.2	0.3416	21MB
Mobilenet YOLO(21MB)	0.3	0.4047	21MB
Mobilenet YOLO(21MB)	0.5	0.3248	21MB

Pengujian pertama adalah distilasi pada objek deteksi menggunakan dataset Manula dan *pretrain* VOC. Setelah itu dilakukan uji coba dengan menggunakan *teacher* yang memiliki 0.5660 mAP dan *student* yang memiliki 0.3164 mAP. Hasilnya saat nilai alpha 0.1 akurasi yang diperoleh adalah 0.3595 mAP. Saat nilai alpha 0.2 akurasi yang diperoleh adalah 0.3756 mAP. Saat nilai alpha 0.3 akurasi yang diperoleh adalah 0.44482 mAP. Saat nilai alpha 0.5 akurasi yang diperoleh adalah 0.3156 mAP. Hasil pengujian bisa dilihat pada tabel 4.13 untuk model *student* dan *teacher* dan tabel 4.14 untuk pengujian distilasi.

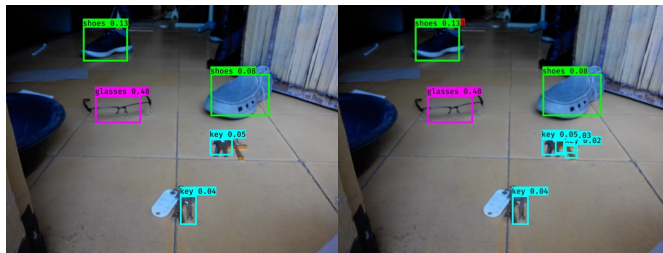
Tabel 4.13: Teacher dan Student untuk percobaan distilasi pada objek deteksi (Dataset Manula) (pretrain VOC)

Status	Model	mAP	Size
Teacher	YOLO	0.5660	236MB
Pure	Mobilenet YOLO(21MB)	0.3164	21MB

Tabel 4.14: Hasil perbandingan distilasi pada objek deteksi (Dataset Manula) (pretrain VOC)

Model	Alpha	mAP	Size
Mobilenet YOLO(21MB)	0.1	0.3595	21MB
Mobilenet YOLO(21MB)	0.2	0.3756	21MB
Mobilenet YOLO(21MB)	0.3	0.4482	21MB
Mobilenet YOLO(21MB)	0.5	0.3156	21MB

Perbandingan distilasi dapat dilihat pada gambar 4.5. Setelah Pengujian beberapa model, model dengan akurasi terbaik yaitu Model yang telah di distilasi dengan akurasi 0.4750 mAP dibandingkan dengan model tanpa distilasi yang memiliki akurasi 0.3790 mAP. Objek yang dijadikan perbandingan adalah objek yang bersal dari dataset manula. Setelah di uji coba, model dengan distilasi dapat mendeteksi lebih banyak objek pada bagian kunci.



(a) Sebelum Distilasi

(b) Sesudah Distilasi

Gambar 4.5: Perbandingan performa model Mobilenet YOLO(21MB) sesudah dan sebelum distilasi

4.7 Pengujian Pengaruh *Batchnorm Fusion*

Pengujian pengaruh *BatchNorm Fusion* dilakukan untuk mengamati kecepatan model setelah metode diterapkan. Tabel 4.15 berupa informasi tentang *teacher* dan *student*. Jika dilihat dari jumlah *textitlayer* dan parameter, Mobilenet YOLO(21MB) memiliki pengurangan *layer* paling banyak sebesar 34 *layer* dari 115 *layer* menjadi 81 *layer*. Sedangkan Tiny YOLO hanya mengalami pengurangan sebesar 11 *layer* dari 44 *layer* menjadi 33 *layer* dan Tiny YOLO dengan 3 skala hanya mengalami pengurangan sebesar 14 *layer* dari 56 *layer* menjadi 42 *layer*.

Tabel 4.15: Hasil perbandingan model sebelum dan sesudah *Batchnorm Fusion*

Model	Layer		Parameter	
	Normal	BN-Fused	Normal	BN-Fused
Tiny YOLO	44	33	8.858.734	8.849.182
MobilntYOLO(21MB)	115	81	5.530.401	5.493.153
Tiny YOLO(3 scale)	56	42	8.419.793	8.409.281

Berdasarkan tabel 4.16 , untuk uji coba di CPU i7 peningkatan FPS tertinggi terjadi pada Mobilenet YOLO(21MB) ukuran *input* 224 sebesar 8.36 dari 8.3 FPS ke 16.66 FPS. Sedangkan Tiny YOLO untuk ukuran *input* 224 mengalami peningkatan sebesar 5.66 dari 11 FPS ke 16.66 FPS dan Tiny YOLO denganv3 skala untuk ukuran *input* 224 mengalami peningkatan sebesar 5.66 dari 11 FPS ke 16.66 FPS. Saat uji coba pada Raspberry peningkatan FPS tertinggi terjadi pada Mobilenet YOLO(21MB) ukuran *input* 224 sebesar 0.2 dari 0.9 FPS ke 1.1 FPS. Sedangkan Tiny YOLO untuk ukuran *input* 224 mengalami peningkatan sebesar 0.08 dari 1.25 FPS ke 133 FPS dan Tiny YOLO dengan 3 skala untuk ukuran *input* 224 mengalami peningkatan sebesar 0.2 dari 0.9 FPS ke 1.1 FPS. pada Raspberry peningkatan yang terjadi tidak sebesar yang terjadi di CPU i7.

Tabel 4.16: Hasil perbandingan performa sebelum dan sesudah Batch-norm Fusion

Model	FPS		FPS(BN-Fused)	
	CPU i7	Raspi	CPU i7	Raspi
Tiny YOLO 416	4	0.416	5.88	0.454
Tiny YOLO 224	11	1.25	16.66	1.44
Mobilenet YOLO(21MB) 416	2.5	0.3125	5.88	0.37
Mobilenet YOLO(21MB) 224	8.3	0.9	16.66	1.1
Tiny Yolo (3 scale) 416	3.57	0.45	5.26	0.47
Tiny Yolo (3 scale) 224	11	1.25	16.66	1.31

4.8 Pengujian Deteksi *Real-Time* pada Robot

Pengujian pencarian objek dilakukan untuk memastikan apakah sistem deteksi dapat bekerja secara *real-time* jika diterapkan pada robot service. Pengujian dilakukan dengan melihat apakah sistem robot dapat berinteraksi dengan baik dan akan berhenti tepat waktu jika objek ditemukan. Objek akan diletakkan di beberapa posisi yang akan dilalui robot, robot akan terus bergerak ke rute-rute yang telah ditentukan. Jika ditengah jalan robot menemukan objek, maka robot akan berhenti dan menyalakan *buzzer* untuk menandakan bahwa objek telah ditemukan. Saat kecepatan diatur sebesar 30, robot baru bisa merespon dengan baik.

Karena keterbatasan ruang gerakan robot tidak akan terlalu banyak, namun cukup untuk melakukan pengujian sistem dengan baik. Pertama robot akan maju sepanjang 80 cm. Saat sudah mencapai posisi robot akan melakukan rotasi sebesar 90° dan maju sepanjang 40 cm. Setelah itu robot melakukan rotasi sebesar 90° lagi lalu maju sepanjang 80 cm. Proses ini akan dilakukan hingga objek yang dicari ditemukan.

Untuk Uji coba dilakukan dengan membandingkan model Mobilenet YOLO(21MB) sebelum dilakukan metode Batchnorm Fusion dan sesudah dilakukan metode *Batchnorm Fusion*. Saat diuji coba tanpa robot model bisa mencapai kecepatan yang bisa dibilang real time pada saat diuji coba di CPU i7. Di Raspberry meskipun tidak ada proses deteksi sama sekali FPS tetap turun menjadi 8 FPS dimana seharusnya 24 FPS seperti pada CPU i7 sehingga saat

ditambahkan proses deteksi , Raspberry masih belum cukup kuat untuk memproses seperti pada CPU i7. Namun dengan kecepatan robot yang telah disesuaikan Raspberry masih bisa digunakan untuk melakukan pencarian objek setelah diterapkan metode *Batchnorm Fusion*.

Tabel 4.17: Perbandingan Spesifikasi

Jenis	Prosesor	Frekuensi	Core	Ram
SAMSUNG NP400B5B	Intel i7-2670QM	2.20 GHz	4	8GB
ASUS X452C	Intel i3 3217U	1.80 GHz	2	4GB
RASPBERRY	Cortex-A53	1.4GHz	4	1GB

Tabel 4.18: Hasil perbandingan FPS untuk performa robot ukuran gambar 224 model mobilenet YOLO(21MB)

Prosesor	FPS		
	Tanpa Deteksi ^a	Deteksi ^b	Deteksi(BN-Fused) ^c
CPU i7	24	8.3	16.66
CPU i3	24	3.333	10.21
Raspberry	8	1.25	1.44

^a Kemampuan program menampilkan video tanpa tambahan proses apapun.

^b Kemampuan program memproses video dengan tambahan proses mendeteksi objek.

^c Kemampuan program memproses video dengan tambahan proses mendeteksi objek dan sudah dilakukan metode batchnorm fusion.

Setelah dilakukan uji coba deteksi pada robot deteksi yang bisa dianggap real-time oleh mata manusia adalah saat dilakukan di CPU i7 karena FPS nya bisa mencapai 16.66 FPS lebih besar daripada standar yang biasa digunakan yaitu 12 FPS. Sedangkan pada Raspberry FPS nya sekitar 1.44 FPS bisa dibidang masih kurang untuk dianggap sebagai real-time namun masih bisa digunakan untuk mendeteksi objek jika kecepatan robot diperlambat.

Setelah mengamati hasil pengujian pada tabel 4.18 dan membandingkan tiap prosesor pada tabel 4.17, dapat diamati bahwa

raspberry masih belum mampu mendapatkan FPS yang tinggi. Raspberry masih dapat digunakan untuk pendeteksian objek, namun dengan kecepatan robot yang harus disesuaikan. Hal ini disebabkan karena raspberry hanya memiliki 1 GB Ram untuk memproses komputasi dan saat tidak ada proses deteksi pun Raspberry hanya mampu memproses komputasi hingga 8 FPS saja dibandingkan dengan CPU i7 dan CPU i3 yang dapat memproses hingga 24 FPS.

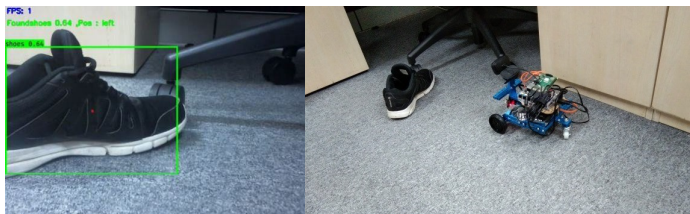
Setelah diuji coba pada *robot service* sebelum diterapkan metode *batchnorm fusion*, pada gambar 4.6 robot terlambat saat mendeteksi sehingga saat harus berhenti, robot tidak berhenti di depan objek namun agak terlewat sehingga objek yang dicari tidak tertangkap oleh kamera saat robot berhenti. Sedangkan uji coba setelah diterapkan *batchnorm fusion*, pada gambar 4.7 robot berhasil berhenti tepat di depan objek saat berhasil mendeteksi objek.



(a) Tampilan kamera robot

(b) Tampilan dari luar

Gambar 4.6: Percobaan pencarian objek sebelum *Batchnorm Fusion*



(a) Tampilan kamera robot

(b) Tampilan luar

Gambar 4.7: Percobaan pencarian objek setelah *Batchnorm Fusion*

Halaman ini sengaja dikosongkan

BAB 5

PENUTUP

5.1 Kesimpulan

Dalam penelitian telah dilakukan modifikasi model sistem pendeteksi objek YOLO dengan *feature extractor* mobilenet yang telah diterapkan metode *knowledge distillation* untuk meningkatkan akurasi dan metode *batchnorm fusion* untuk meningkatkan kecepatan komputasi. Metode-metode yang diterapkan dapat membuat model yang berukuran kecil dengan akurasi yang dapat digunakan di sistem pendeteksi objek pada *robot service*. Objek yang dapat dideteksi sistem adalah barang-barang yang sering dicari seperti dompet, handphone, kacamata, kunci, dan sepatu atau sandal. Bahasa pemrograman yang digunakan adalah python 3.6.7 dengan menggunakan framework Tensorflow 1.13.1 dan Keras 2.2.4. Hardware yang digunakan untuk uji coba adalah prosesor CPU Intel i7-2670QM di Laptop, CPU Intel i3 3217U di Laptop dan CPU Cortex-A53 di Raspberry. Setelah dilakukan beberapa percobaan dapat diambil beberapa kesimpulan yaitu:

1. Model yang dapat dijalankan di *embedded device* seperti raspberry adalah Tiny YOLO dan Mobilenet YOLO(21MB) yang memiliki ukuran kecil dan jumlah *layer* yang sedikit. Mobilenet YOLO(21MB) memiliki ukuran 21 MB dengan 0.3843 mAP, 2.5 FPS di CPU i7 dan 0.312 FPS di *Raspberry*. Sedangkan Tiny YOLO memiliki ukuran 33 MB dengan 0.243 mAP, 3.85 FPS di CPU i7 dan 0.416 FPS di *Raspberry*. Tiny YOLO memiliki akurasi mAP yang lebih rendah dari Mobilenet YOLO(21MB) namun FPS nya lebih tinggi. Alasan mengapa Mobilenet YOLO(21MB) memiliki FPS yang lebih rendah meskipun ukurannya lebih kecil adalah desain arsitekturnya yang memiliki jumlah parameter yang lebih sedikit namun memerlukan jumlah layer yang lebih banyak, sehingga kecepatan yang diperoleh karena parameter yang lebih sedikit hilang karena waktu yang diperlukan untuk perpindahan memori antar *layer*.
2. ukuran *input* gambar untuk *deep learning* model mempengaruhi

ruhi nilai mAP dan FPS yang dapat diperoleh. Berdasarkan percobaan saat ukuran *input* yang besar akan memiliki ukuran mAP yang tinggi dan FPS yang rendah dan begitu juga sebaliknya. Hal ini disebabkan gambar yang besar dapat memiliki jumlah pixel lebih banyak sehingga dapat memberikan banyak informasi namun waktu untuk memprosesnya jadi lebih lama.

3. Desain arsitektur yang memiliki 3 skala dapat mendeteksi objek yang berukuran kecil dan berada pada posisi yang jauh lebih baik dibandingkan arsitektur dengan 2 skala.
4. Berdasarkan hasil pengujian penerapan *knowledge distillation* pada dataset VOC Mobilenet YOLO(21MB) sebagai *student* dan YOLO sebagai *teacher* mengalami peningkatan mAP sebesar 9.4% dari 0.3850 mAP menjadi 0.4215 mAP. Sedangkan pada dataset Manula, Mobilenet YOLO(21MB) sebagai *student* dan YOLO sebagai *teacher* mengalami peningkatan mAP sebesar 41.6% dari 0.3164 mAP menjadi 0.4482 mAP
5. Berdasarkan hasil pengujian penerapan *batchnorm fusion* pada CPU i7, Mobilenet YOLO(21MB) terjadi peningkatan FPS hingga 100.7% dari 8.3 FPS menjadi 16.66 FPS. Sedangkan pada Raspberry terjadi peningkatan sebesar 22.22% dari 0.9 FPS menjadi 1.1 FPS. *Batchnorm fusion* mengatasi permasalahan Mobilenet yang memiliki layer lebih banyak untuk mengurangi parameter dengan menggabungkan layer *convolution* dan *batchnorm fusion*. Berdasarkan hasil pengujian proses deteksi bisa dianggap sebagai *real-time* di CPU i7 karena sudah melebihi standar yang dianjurkan 12 FPS karena sistem sudah dapat mencapai 16.66 FPS

5.2 Saran

Untuk pengembangan penelitian tugas akhir lebih lanjut berikutnya terdapat beberapa saran sebagai berikut:

1. Menerapkan metode *quantization* yaitu proses mengkonversi data dari 32 bit *float* menjadi 8 bit *integers* untuk mempercepat komputasi.
2. Menerapkan metode distilasi dengan rumus lain agar dapat memperoleh peningkatan akurasi yang lebih signifikan.
3. Penambahan algoritma yang dapat menentukan urutan lokasi rute pencarian benda berdasarkan data tempat benda yang

seharusnya dan data tempat benda sering ditemukan di pencarian sebelumnya.

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

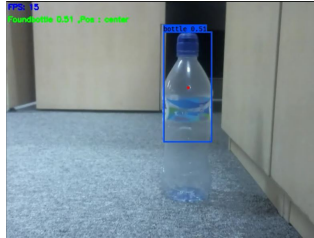
- [1] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” arXiv, 2018. (Dikutip pada halaman i, iii, 2, 11, 13, 44, 57).
- [2] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” CoRR, vol. abs/1704.04861, 2017. (Dikutip pada halaman i, iii, 1, 20, 21).
- [3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” CoRR, vol. abs/1512.03385, 2015. (Dikutip pada halaman 1, 17).
- [4] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” CoRR, vol. abs/1610.02357, 2016. (Dikutip pada halaman 1).
- [5] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, “Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size,” CoRR, vol. abs/1602.07360, 2016. (Dikutip pada halaman 1).
- [6] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” CoRR, vol. abs/1506.02640, 2015. (Dikutip pada halaman 1, 11, 12, 15, 20, 44).
- [7] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, “SSD: single shot multibox detector,” CoRR, vol. abs/1512.02325, 2015. (Dikutip pada halaman 1).
- [8] A. K. T. B. C. Georgoulas, T. Linner, “An aml environment implementation: Embedding turtlebot into a novel robotic service wall,” TU Munchen Germany. (Dikutip pada halaman 1).
- [9] B. P. Statistik, Statistik Penduduk Lanjut Usia 2017. BPS, 2018. (Dikutip pada halaman 1).

- [10] Y. R. M. Uliyah, S. Aisyah, “Hubungan usia dengan penurunan daya ingat (demensia) pada lansia,” Fak Ilmu Kesehatan UM Surabaya. (Dikutip pada halaman 1).
- [11] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, “A survey of model compression and acceleration for deep neural networks,” CoRR, vol. abs/1710.09282, 2017. (Dikutip pada halaman 2, 30).
- [12] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” in NIPS Deep Learning and Representation Learning Workshop, 2015. (Dikutip pada halaman 2, 31, 33).
- [13] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” CoRR, vol. abs/1502.03167, 2015. (Dikutip pada halaman 2).
- [14] M. A. Nielsen, “Neural networks and deep learning,” 2018. (Dikutip pada halaman 5, 6).
- [15] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016. <http://www.deeplearningbook.org>. (Dikutip pada halaman 5, 6).
- [16] Y. LeCun, K. Kavukcuoglu, and C. Farabet, “Convolutional networks and applications in vision,” in Proceedings of 2010 IEEE International Symposium on Circuits and Systems, pp. 253–256, May 2010. (Dikutip pada halaman 7, 8).
- [17] D. Hubel and T. Wiesel, “Receptive fields, binocular interaction, and functional architecture in the cat’s visual cortex,” Journal of Physiology, vol. 160, pp. 106–154, 1962. (Dikutip pada halaman 7).
- [18] S. Ren, K. He, R. B. Girshick, X. Zhang, and J. Sun, “Object detection networks on convolutional feature maps,” CoRR, vol. abs/1504.06066, 2015. (Dikutip pada halaman 15).

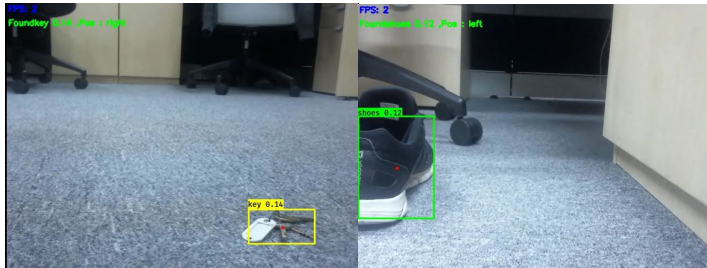
- [19] J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” arXiv preprint arXiv:1612.08242, 2016. (Dikutip pada halaman 15, 19).
- [20] D. Jung, W. Jung, B. Kim, S. Lee, W. Rhee, and J. H. Ahn, “Restructuring batch normalization to accelerate CNN training,” CoRR, vol. abs/1807.01702, 2018. (Dikutip pada halaman 34).
- [21] EdwinOlson, “A primer on odometry and motor control,” 2014. (Dikutip pada halaman 36).
- [22] Dejan, “How rotary encoder works and how to use it with arduino,” 2016. (Dikutip pada halaman 36).
- [23] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” 2010. (Dikutip pada halaman 40).
- [24] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li, “Imagenet large scale visual recognition challenge,” CoRR, vol. abs/1409.0575, 2014. (Dikutip pada halaman 44).
- [25] R. Mehta and C. Ozturk, “Object detection at 200 frames per second,” CoRR, vol. abs/1805.06361, 2018. (Dikutip pada halaman 51, 52).

Halaman ini sengaja dikosongkan

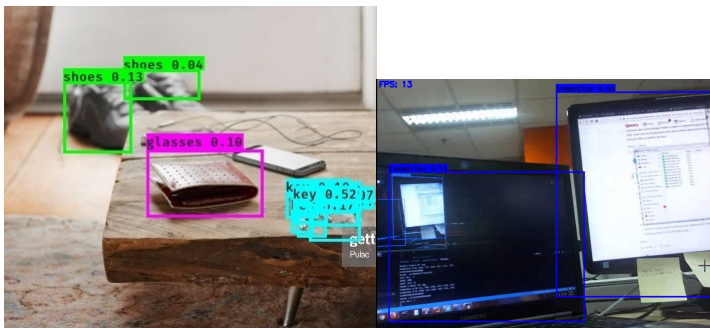
LAMPIRAN



(a) Posisi botol di sebelah tengah



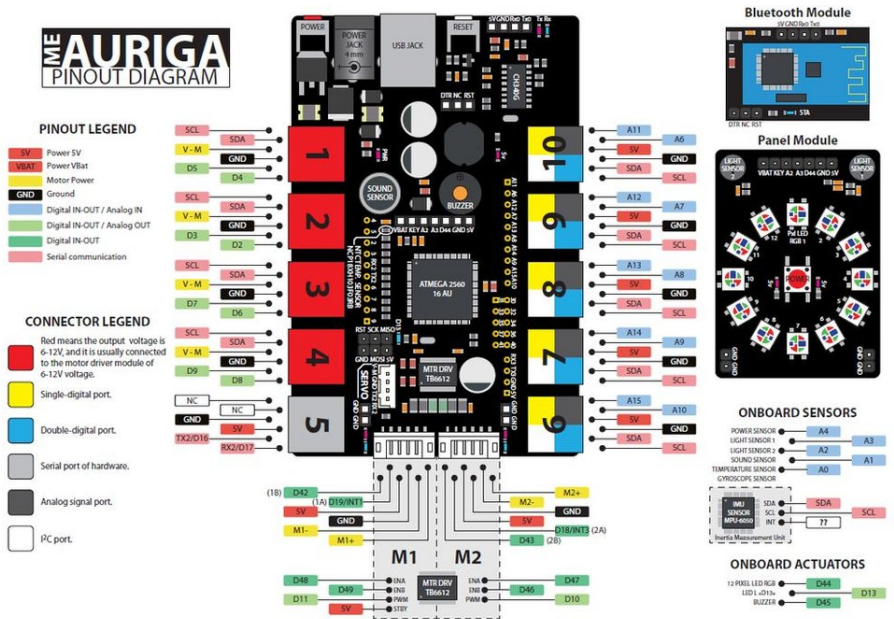
(b) Posisi kunci di sebelah kanan (c) Posisi sepatu di sebelah kiri



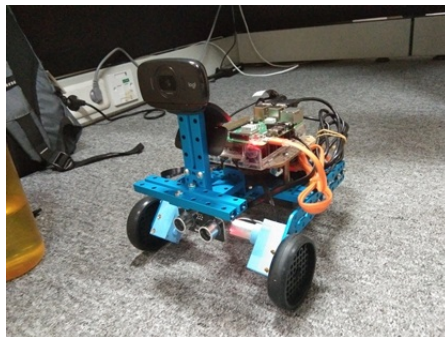
(a)

(b)

Gambar 1: Deteksi berbagai macam objek



Gambar 2: Datasheet Makeblock Auriga



Gambar 3: Robotn Pendeteksi Objek Makeblock Auriga

BIOGRAFI PENULIS



Billy, lahir pada 27 Juni 1997 di Kota Surabaya, Provinsi Jawa Timur. Penulis merupakan anak pertama dari tiga bersaudara. Saat ini penulis tinggal di jl Wiratno 37 A ,Kenjeran Surabaya. Pada tahun 2009 menyelesaikan pendidikan di SDK Pirngadi Kristen Surabaya. Tahun 2012 lulus dari SMP Negeri 18 Surabaya serta pada tahun 2015 lulus dari SMA Negeri 3 Surabaya. Penulis diterima di Program Studi S-1 Departemen Teknik Komputer Fakultas Teknologi Elektro ITS. Penulis aktif menjadi Asisten Lab B401 Komputasi Multimedia Teknik Komputer ITS. Penulis memiliki hobby menggambar dan membaca. Selama kuliah pernah mengikuti lomba di berbagai bidang seperti *Game*, Aplikasi dan *IoT*. Penulis berhasil menyelesaikan tugas akhir dengan judul ” ***Knowledge Distillation pada Sistem Pendeteksi Objek YOLO untuk Robot Service*** ”. Bagi pembaca yang memiliki kritik, saran atau pertanyaan mengenai tugas akhir ini dapat menghubungi penulis melalui email billygun27@gmail.com.

Halaman ini sengaja dikosongkan