TESIS - SM 142501

DETEKSI DAN KLASIFIKASI KERUSAKAN JALAN ASPAL MENGGUNAKAN METODE YOLO BERBASIS CITRA DIGITAL

RAVY HAYU PRAMESTYA
NRP 0611 1650 010 019

DOSEN PEMBIMBING:
Dr. Dwi Ratna S., S.Si, MT.
Dr. Imam Mukhlash, S.Si, MT.

PROGRAM MAGISTER
DEPARTEMEN MATEMATIKA
FAKULTAS MATEMATIKA, KOMPUTASI DAN SAINS DATA
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2018

THESIS - SM 142501

PAVEMENT DISTRESS DETECTION AND CLASSIFICATION USING YOLO METHOD BASED OF DIGITAL IMAGE

RAVY HAYU PRAMESTYA
NRP 0611 1650 010 019

SUPERVISOR:
Dr. Dwi Ratna S., S.Si, MT.
Dr. Imam Mukhlash, S.Si, MT.

MASTER PROGRAM
DEPARTMENT OF MATHEMATICS
FACULTY OF MATHEMATICS, COMPUTING AND DATA SCIENCE
SEPULUH NOPEMBER INSTITUTE OF TECHNOLOGY
SURABAYA
2018

Tesis ini disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Sains (M.Si.)
di
Fakultas Matematika, Komputasi dan Sains Data
Institut Teknologi Sepuluh Nopember

Oleh:
RAVY HAYU PRAMESTYA
NRP. 0611 1650 010 019

Tanggal Ujian : 30 Juli 2018
Periode Wisuda : 118

Disetujui oleh:

Dr. Dwi Ratna S., S.Si, MT.
NIP 19690405 199403 2 003 Pembimbing I

Dr. Imam Mukhlash, S.Si, MT.
NIP 19700831 199403 1 003 Pembimbing II

Dr. Darmaji, S.Si., M.T.
NIP 19691015 199412 1 001 (Penguji)

Dr. Budi Setiyono, S.Si, M.T.
NIP 19720207 199702 1 001 (Penguji)

Dekan FMKSD,

Prof. DR. Basuki Widodo, M.Sc
NIP. 19650605 198903 1 002

DETEKSI DAN KLASIFIKASI KERUSAKAN JALAN ASPAL MENGGUNAKAN METODE YOLO BERBASIS CITRA DIGITAL

Nama Mahasiswa	:	Ravy Hayu Pramestya
NRP	:	0611 1650 010 019
Pembimbing	:	1. Dr. Dwi Ratna S., S.Si, MT.
		2. Dr. Imam Mukhlash, S.Si, MT.

ABSTRAK

Jalan merupakan infrastuktur yang penting, sehingga diperlukan adanya pemeliharaan jalan secara berkala. Pengidentifikasian jenis kerusakan jalan merupakan hal yang perlu dilakukan dalam upaya penilaian kondisi jalan secara otomatis. Pengidentifikasian kerusakan jalan secara manual membutuhkan waktu yang lama dan dapat menimbulkan ambigu karna faktor subyektifitas petugas dalam penilaian kondisi jalan. Beberapa metode telah digunakan untuk mendeteksi dan mengklasifikasi kerusakan jalan secara otomatis, seperti Support Vector Machine (SVM) dan Fuzzy C-Means Clustering. Penelitian ini mendeteksi dan mengklasifikasi citra kerusakan jalan dengan menggunakan metode YOLO. Jenis kerusakan jalan yang diidentifikasi sebanyak tiga jenis, yaitu kerusakan jalan jenis lubang, retak garis dan retak non-garis. Metode YOLO menggunakan Convolutional Neural Network (CNN) dalam arsitekturnya, dan telah memberi hasil yang baik dalam identifikasi objek pada citra maupun video. YOLO telah diuji pada berbagai dataset dan terbukti memberi hasil yang cepat dan cukup akurat. Pada penelitian ini, pre-processing data yang dilakukan yaitu cropping dan resizing citra serta anotasi data. Lalu proses pelatihan dilakukan dengan fine tuning jaringan YOLO. Arsitektur YOLO yang digunakan adalah arsitektur dengan 9 layer konvolusi dan 6 layer maxpool. Terdapat empat dataset yang digunakan untuk proses pengujian, yaitu dataset dengan citra tanpa diberi gangguan, dataset citra dengan gangguan efek kabur (blur), dataset citra dengan gangguan penambahan intensitas cahaya serta dataset citra dengan pengurangan intensitas cahaya. Dari hasil pengujian pada empat dataset tersebut, didapatkan rata-rata nilai akurasi klasifikasi adalah 99.25%., rata-rata deteksi kotak pembatas adalah 74.57% sedangkan rata-rata kecepatan klasifikasi dan deteksi adalah 0.911 detik tiap citra.

Kata-kunci: Kerusakan jalan, lubang, retak, *Convolutional Neural Network*, YOLO

PAVEMENT DISTRESS DETECTION AND CLASSIFICATION USING YOLO METHOD BASED OF DIGITAL IMAGE

Name	:	Ravy Hayu Pramestya
NRP	:	0611 1650 010 019
Supervisors	:	1. Dr. Dwi Ratna S., S.Si, MT.
		2. Dr. Imam Mukhlash, S.Si, MT.

ABSTRACT

Road is an important infrastructure, so it is necessary to maintain the road regularly. Detection of road distress is a necessary step in automatically road maintenancing. Manual detection of road distress takes a long time and can cause ambiguity due to the officer's subjective factor in the assessment of road conditions. Several methods have been used to detect and classify road distress automatically. Such as Support Vector Machine (SVM) and Fuzzy C-Means Clustering. This research detects and classifies the distress pavement image by using YOLO method. The type of road distress detected are three types, they are potholes, line-cracks and non-line cracks. The YOLO method use Convolutional Neural Network (CNN) in its architecture, and has given good results in object detection both on images and videos. YOLO has been tested in various datasets and given faster and accurate results. In this research, data pre-processing steps are cropping and resizing images then annotating the data. After that, the training is done by fine tuning YOLO network process. The YOLO architecture use 9 layer convolution and 6 layer maxpool. There are four datasets used for the test process, i.e. image without disturbance dataset, blurred datasets, datasets with the addition of light intensity and dataset with light intensity reduction. The testing results for four datasets, show that the average of object classification accuracy is 99.25%, the average of bounding box detection is 74.57%, while the average of classification and detection speed is 0.911 second per image.

Key-words: Pavement Distress, Pothole, Crack, Convolutional Neural Network, YOLO

KATA PENGANTAR

Segala puji bagi Allah SWT yang telah melimpahkan rahmat, taufik, dan hidayah-Nya, sehingga penulis dapat menyelesaikan dengan lancar Tesis yang berjudul

"Deteksi dan Klasifikasi Kerusakan Jalan Aspal

Menggunakan Metode YOLO Berbasis Citra Digital".

Tesis ini dibuat untuk memenuhi salah satu syarat dalam memperoleh gelar Magister Program Strata-2 Departemen Matematika, Fakultas Matematika, Komputasi dan Sains Data, Institut Teknologi Sepuluh Nopember.

Penyusunan Tesis ini tidak terlepas dari bantuan berbagai pihak. oleh karena itu, pada kesempatan ini, penulis menyampaikan terima kasih kepada pihak-pihak tersebut diantaranya:

1. Rektor Institut Teknologi Sepuluh Nopember
2. Dekan Fakultas Matematika, Komputasi dan Sains Data, Institut Teknologi Sepuluh Nopember
3. Kepala Departemen Matematika, Fakultas Matematika, Komputasi dan Sains Data, Institut Teknologi Sepuluh Nopember
4. Kepala Program Studi Strata-2 Departemen Matematika, Fakultas Matematika, Komputasi dan Sains Data, Institut Teknologi Sepuluh Nopember
5. Ibu Dr. Dwi Ratna Sulistyaningrum, S.Si, MT. Selaku Dosen Pembimbing I dalam penyelesaian Tesis
6. Bapak Dr. Imam Mukhlash, S.Si, MT. Selaku Dosen Pembimbing II dalam penyelesaian Tesis sekaligus Dosen Wali selama menempuh program studi Strata-2
7. Bapak Dr. Budi Setiyono S.Si, MT., Bapak Subchan M.Sc, Ph.D, Bapak Dr. Darmaji S.Si, MT. selaku Dosen Penguji dalam penyelesaian Tesis ini.
8. Orang tua, kakak, dan adik yang selalu memberikan do'a serta dukungan selama menempuh program studi Strata-2
9. Teman-teman seperjuangan di Program Studi Magister Matematika. Terimakasih banyak atas semangat yang menginspirasi, dan bantuan sharing ilmu dalam diskusi, sehingga selama perkuliahan sampai Tesis ini selesai dapat berjalan dengan lancar

10. Staff Pasca Sarjana Matematika, Mbak Resty dan Mas Afif. Terimakasih banyak atas bantuan dalam menginformasikan keperluan administrasi dan bersedia menampung keluh kesah penulis selama proses penyelesaian Tesis hingga kelulusan

11. Kakak dan Adik angkatan di Program Studi Magister Matematika, serta semua pihak yang telah memberikan do'a dan dukungannya kepada penulis, yang tidak dapat penulis sebutkan satu-persatu.

Penulis menyadari bahwa dalam Tesis ini masih terdapat kelemahan dan kekurangan, oleh karena itu penulis sangat terbuka menerima saran dan ide demi kesempurnaan penulisan selanjutnya. Penulis berharap semoga Tesis ini dapat bermanfaat bagi pembaca, dan semua yang telah dikerjakan ini mendapat ridho dari Allah SWT.

Surabaya, Juli 2018

Penulis

DAFTAR ISI

HALAMAN JUDUL i

LEMBAR PENGESAHAN TESIS v

ABSTRAK vii

ABSTRACT ix

KATA PENGANTAR xi

DAFTAR ISI xiii

DAFTAR GAMBAR xv

DAFTAR TABEL xvii

BAB 1 PENDAHULUAN 1
- 1.1 Latar Belakang . . . 1
- 1.2 Rumusan Masalah . . . 3
- 1.3 Batasan Masalah . . . 3
- 1.4 Tujuan Penelitian . . . 3
- 1.5 Manfaat Penelitian . . . 3

BAB 2 KAJIAN PUSTAKA DAN DASAR TEORI 5
- 2.1 Penelitian-Penelitian Terkait . . . 5
- 2.2 Kerusakan Jalan . . . 7
  - 2.2.1 Pengertian Jalan . . . 7
  - 2.2.2 Jenis Kerusakan Jalan . . . 7
- 2.3 Perbaikan Kualitas Citra . . . 10
  - 2.3.1 *Brightness Adjustment* . . . 10
  - 2.3.2 *Motion Blur* . . . 10
- 2.4 *Deep Learning* . . . 11
- 2.5 *Convolutional Neural Network* (CNN) . . . 12
- 2.6 YOLO (*You Only Look Once*) . . . 14
  - 2.6.1 Desain Jaringan . . . 16
  - 2.6.2 Keunggulan YOLO . . . 18
- 2.7 *Intersection over Union* (IoU) . . . 19

BAB 3 METODE PENELITIAN 21
- 3.1 Tahapan Penelitian . . . 21
- 3.2 Blok Diagram Deteksi Kerusakan Jalan Menggunakan YOLO 22

BAB 4 PERANCANGAN DAN IMPLEMENTASI PROGRAM 25
- 4.1 Perancangan Data . . . 25
  - 4.1.1 Kebutuhan Perangkat Lunak . . . 25
  - 4.1.2 Data Awal . . . 26

4.1.3 *Pre-trained Weights* 27
4.1.4 Dataset Proses Pengujian 27
4.2 Perancangan Proses 28
4.2.1 Persiapan Data 29
4.2.2 Pelatihan YOLOv1-tiny dengan dataset baru 30
4.2.3 Pengujian YOLOv1-tiny 34
4.2.4 Uji Keakuratan 35
4.3 Implementasi Program 36
4.3.1 Anotasi data 37
4.3.2 Pelatihan 38
4.3.3 Pengujian 41
4.3.4 Implementasi Uji Keakuratan 41
BAB 5 UJI COBA DAN PEMBAHASAN 45
5.1 Dataset Pengujian 45
5.1.1 *Motion Blur* 45
5.1.2 Penambahan *Brightness* 46
5.1.3 Pengurangan *Brightness* 47
5.2 *Threshold* 47
5.3 *Output* Pengujian 51
5.3.1 Citra tanpa gangguan 51
5.3.2 Citra dengan *Motion Bluring* 53
5.3.3 Citra dengn Penambahan *Brightness* 53
5.3.4 Citra dengan Pengurangan *Brightness* 54
5.4 Kinerja Metode YOLO 56
5.4.1 Dataset Tanpa Gangguan 56
5.4.2 Dataset gangguan *Motion Blur* 57
5.4.3 Dataset gangguan Penambahan Kecerahan 58
5.4.4 Dataset gangguan Pengurangan Kecerahan 60
BAB 6 KESIMPULAN DAN SARAN 63
6.1 Kesimpulan 63
6.2 Saran 63

DAFTAR GAMBAR

Gambar 2.1 *Alligator Cracking* 8
Gambar 2.2 *Corrugation* 8
Gambar 2.3 *Depression* 9
Gambar 2.4 *Pothole* 9
Gambar 2.5 Contoh citra dengan pengaturan pencahayaan 11
Gambar 2.6 Contoh citra dengan *motion blur* 11
Gambar 2.7 Hubungan *Deep Learning* dengan *Machine Learning* . . . 12
Gambar 2.8 Dimensi MLP (kiri) dan CNN (kanan) 14
Gambar 2.9 Contoh Arsitektur CNN 14
Gambar 2.10 Proses Konvolusi 15
Gambar 2.11 ReLu Activation 15
Gambar 2.12 Max Pooling 15
Gambar 2.13 Model YOLO 16
Gambar 2.14 Arsitektur dasar YOLO 17
Gambar 2.15 Sistem deteksi YOLO 18
Gambar 2.16 Ilustrasi perhitungan IOU 20

Gambar 3.1 Diagram Alur Deteksi Kerusakan Jalan Menggunakan YOLO 24

Gambar 4.1 Citra bentuk awal 26
Gambar 4.2 Citra hasil *Resizing* dan *Cropping* 27
Gambar 4.3 Diagram Alur Anotasi Data 29
Gambar 4.4 Arsitektur Jaringan YOLOv1-tiny 30
Gambar 4.5 Ilustrasi representasi citra awal 31
Gambar 4.6 Hasil *padding* 32
Gambar 4.7 Contoh kernel 33
Gambar 4.8 Hasil konvolusi kolom 1 33
Gambar 4.9 Hasil konvolusi kolom 2 33
Gambar 4.10 Diagram alur proses pengujian 34
Gambar 4.11 Contoh kasus perhitungan IoU 35
Gambar 4.12 Diagram alir perhitungan IoU 36
Gambar 4.13 *Source code* menyimpan data anotasi 37
Gambar 4.14 *Source code* menggambar kotak pembatas 39
Gambar 4.15 Contoh hasil data anotasi 40
Gambar 4.16 *Source code* proses pengujian 42
Gambar 4.17 *Source code* membaca file xml 43
Gambar 4.18 *Source code* mencari IoU 44

Gambar 5.1 Citra dengan noise alami 46
Gambar 5.2 *Motion blurring* 46

Gambar 5.3 Efek Cerah 47
Gambar 5.4 Efek Gelap 48
Gambar 5.5 Kotak pembatas yang ganda 49
Gambar 5.6 Kotak pembatas yang tidak muncul 50
Gambar 5.7 Kotak pembatas yang sesuai 50
Gambar 5.8 Hasil pengujian citra tanpa gangguan 52
Gambar 5.9 Hasil penngujian citra dengan gangguan objek 52
Gambar 5.10 Hasil pengujian citra *motion blur* $l = 3$ 53
Gambar 5.11 Hasil pengujian citra *motion blur* $l = 15$ 54
Gambar 5.12 Hasil pengujian citra penambahan kecarahan $n = 30$ . . . 54
Gambar 5.13 Hasil pengujian citra penambahan kecerahan $n = 80$ . . . 55
Gambar 5.14 Hasil pengujian citra pengurangan kecerahan $n = -30$. 55
Gambar 5.15 Hasil pengujian citra pengurangan kecerahan $n = -100$ 56

DAFTAR TABEL

Tabel 4.1 Kebutuhan Perancangan 25
Tabel 4.2 Dataset Proses Pengujian 28

Tabel 5.1 Hasil Uji Coba Akurasi Dataset Tanpa Gangguan 57
Tabel 5.2 Rata-rata IoU Dataset Tanpa Gangguan 57
Tabel 5.3 Hasil Uji Coba Akurasi Dataset Gangguan *Motion Blur* .. 58
Tabel 5.4 Rata-rata IoU Dataset Gangguan *Motion Blur* 58
Tabel 5.5 Hasil Uji Coba Akurasi Dataset Gangguan Penambahan Kecerahan 59
Tabel 5.6 Rata-rata IoU Dataset Gangguan Penambahan Kecerahan 60
Tabel 5.7 Hasil Uji Coba Akurasi Dataset Gangguan Pengurangan Kecerahan 60
Tabel 5.8 Rata-rata IoU Dataset Gangguan Pengurangan Kecerahan 61

BAB 1
PENDAHULUAN

Pada bab ini diuraikan latar belakang yang mendasari usulan penelitian yang berisi identifikasi masalah, beberapa informasi penelitian terdahulu baik secara umum atau khusus yang berkaitan dengan rencana penelitian, beberapa referensi yang berkaitan, tujuan penelitian, batasan masalah, dan manfaatnya.

1.1 Latar Belakang

Kerusakan jalan dapat menimbulkan ketidak-nyamanan dalam berkendara dan bahkan dapat mengakibatkan kecelakaan. Beberapa kerusakan pada jalan raya yaitu seperti retak halus, retak kulit buaya, lubang, pelepasan butir dan sebagainya. Menurut Data Peta Kondisi Jaringan Jalan Nasional pada tahun 2017, tingkat kerusakan tipe berat dan ringan pada jalan raya di daerah Jawa Timur telah mencapai 288 Kilometer. Berdasarkan hal tersebut, diperlukan adanya peninjauan kondisi jalan dan pemeliharaan secara berkala agar kerusakan dapat diminimalisir atau dicegah. Kerusakan pada perkerasan jalan atau lapisan penutup aspal harus diprioritaskan perbaikannya, karena di daerah dengan curah hujan yang tinggi seperti Indonesia, perkerasan jalan dapat lebih cepat rusak (Departemen-Pekerjaan-Umum, 1995).

Langkah awal dari pemeliharaan jalan adalah dengan mengidentifikasi kerusakan pada suatu jalan, sehingga dapat menentukan tindakan apa yang perlu dilakukan. Metode yang digunakan untuk mengidentifikasi kondisi kerusakan jalan dapat dilakukan secara manual dan otomatis. Metode manual dilakukan dengan menyusuri jalan, mengambil gambar kerusakan jalan dengan kamera, mengukur area kerusakan, menentukan tingkat kerusakan sesuai dengan jenis kerusakan jalan, lalu menghitung dan menuliskannya dalam bentuk laporan. Metode ini sangat menyita waktu, tenaga dan biaya. Apalagi petugas harus menyusuri lalu lintas, hal tersebut akan membahayakan petugas. Metode ini juga rawan subjektivitas, sehingga dapat memberikan akurasi yang rendah tentang kerusakan jalan.

Sedangkan pengidentifikasian secara otomatis dilakukan dengan bantuan alat yang dapat mengambil citra kondisi jalan dan secara otomatis membedakan jenis kerusakan jalan, letak kerusakan jalan dalam citra serta

mengitung tingkat kerusakan jalan sesuai dengan jenis kerusakan jalan. Cara ini lebih efektif, obyektif dan aman dalam upaya pemeliharaan jalan. Pengidentifikasian ini untuk lebih jauh juga dapat digunakan sebagai acuan dalam memberikan tindakan yang tepat untuk pemeliharaan jalan.

Pengidentifikasian kerusakan jalan pernah diteliti oleh (Ouma dan Hahn, 2017) dengan judul *Pothole Detection on Asphalt Pavements From 2D-Colour Pothole Images Using Fuzzy C-Means Clustering and Morphological Reconstruction*. Penelitian ini mengidentifikasi citra jalan berlubang dan citra jalan normal. Metode yang digunakan yaitu *Wavelet Transformation* untuk proses *defect recognition* dan *Fuzzy C-Means* untuk proses *Clustering* serta beberapa metode morfologi.

Penelitian lain yang terkait yaitu *A Real-time Automatic Pavement Crack and Pothole Recognition System for Mobile Android-based Devices* (Tedeschi dan Benedetto, 2017). Penelitian ini bertujuan untuk mengklasifikasi kerusakan jalan jenis lubang dan beberapa jenis retak yang diterapkan pada *mobile aplication* menggunakan *Cascade Classifier*. Akurasi metode yang dihasilkan pada penelitian ini yaitu lebih dari 70%

Dalam penelitian ini, dilakukan deteksi dan klasifikasi jenis kerusakan jalan yaitu lubang, retak garis, dan retak *non*-garis menggunakan metode YOLO (*You Only Look Once*). YOLO menggunakan jaringan *Convolutional Neural Network* (CNN) tunggal untuk klasifikasi dan lokalisasi objek menggunakan kotak pembatas.

Penelitian tentang CNN telah banyak digunakan untuk mengidentifikasi citra, dan memberikan hasil yang meyakinkan. Penelitian terbaru tentang CNN dilakukan oleh (Fan, Wu, Lu dan Li, 2018) dengan judul *Automatic Pavement Crack Detection Based on Structured Prediction with the Convolutional Neural Network*. Penelitian ini mengidentifikasi citra jalan retak dan jalan normal. Dengan menggunakan 3 jenis dataset dan beberapa jenis pembagian data *training* dan *testing*, didapatkan presisi tertinggi sebesar 96%.

Sedangkan metode YOLO diperkenalkan pertama kali oleh (Redmon, Divvala, Girshick dan Farhadi, 2016) dalam judul *You Only Look Once : Unified, Real-Time Object Detection*. Dalam penelitiannya, selain arsitekturnya yang sederhana, dikatakan pula bahwa YOLO sangat cepat dalam mengidentifikasi objek dan akurasi rata-rata yang didapatkan mencapai 88% dalam *ImageNet 2012 Validation*.

Penerapan metode YOLO sampai saat ini belum banyak diterapkan dalam pengidentifikasian citra. Padahal, YOLO terbukti merupakan metode yang

lebih efisien dibandingkan dengan algoritma *machine learning* lainnya (Wang, Rasmussen dan Song, 2016). Maka dari itu, penulis tertarik menggunakan metode YOLO untuk mendeteksi dan mengklasifikasi citra kerusakan jalan.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dijabarkan di atas, permasalahan yang dapat diangkat adalah sebagai berikut :

1. Bagaimana mendeteksi dan mengklasifikasi kerusakan jalan menggunakan YOLO berbasis citra digital?

2. Bagaimana kinerja metode YOLO untuk mendeteksi dan mengklasifikasi kerusakan jalan berbasis citra digital?

1.3 Batasan Masalah

Terdapat beberapa batasan masalah pada penelitian ini yaitu objek yang akan dideteksi dan diklasifikasi yaitu kerusakan jalan jenis lubang, retak garis (*longitudinal* dan *transversal*) dan kerusakan retak *non*-garis. Citra yang digunakan yaitu citra dengan pengambilan gambar sedemikian rupa sehingga citra yang didapatkan fokus lurus dengan kerusakan jalan, selain itu pengambilan gambar dilakukan saat kondisi terang. Jenis jalan yang digunakan yaitu aspal.

1.4 Tujuan Penelitian

Berdasarkan rumusan masalah diatas, didapatkan tujuan dari penelitian ini adalah sebagai berikut :

1. Melakukan proses deteksi dan klasifikasi kerusakan jalan menggunakan YOLO berbasis citra digital

2. Menganalisa kinerja metode YOLO dalam mendeteksi dan mengklasifikasi kerusakan jalan berbasis citra digital

1.5 Manfaat Penelitian

Manfaat dari penelitian ini adalah sebagai langkah awal dalam menerapkan penilaian kondisi jalan berbasis citra digital untuk menentukan alternatif penanganannya. Penelitian ini dapat digunakan sebagai alat bantu dalam pemeliharaan jalan untuk pihak pemelihara kondisi jalan, baik Binamarga maupun Dinas Pekerjaan Umum (DPU). Selain itu juga sebagai refrensi penggunaan metode YOLO dalam pengidentifikasian kerusakan jalan berbasis citra.

BAB 2
KAJIAN PUSTAKA DAN DASAR TEORI

Pada bab ini akan dijelaskan pengantar untuk penelitian yang meliputi beberapa penelitian sebelumnya yang terkait dengan penelitian ini, pengertian dan jenis kerusakan jalan, dasar pengolahan citra digital, metode *Convolutional Neural Network* (CNN), metode YOLO dan *Intersection over Union* (IoU).

2.1 Penelitian-Penelitian Terkait

Beberapa penelitian dalam mendeteksi dan mengklasifikasi kerusakan jalan telah dilakukan, baik dengan metode *image processing*, *machine learning* maupun *deep learning*. Presisi dan akurasi dari proses deteksi yang dilakukan juga memberikan hasil yang baik.

Salah satu penelitian dalam pendeteksian kerusaakan jalan yang menunjukkan hasil yang baik adalah penelitian dengan judul *Image-Based Pothole Detection System for ITS Service and Road Management System*, (Ryu, Kim dan Kim, 2015). Penelitian ini membahas tentang pendeteksian lubang dengan beberapa tahap segmentasi yaitu *binarization* dengan HST, *median filter, closing, candidate extract (size, compactness, ellipticity dan linearity)*. Selanjutnya untuk penentuan lubang menggunakan OHI. Metode ini menghasilkn presisi sebesar 80%

Pendeteksian lubang juga telah dilakukan pada penelitian dengan judul *A Novel Technique for Automatic Road Distress Detection and Analysis*, (Daniel dan Preeja, 2014). Penelitian ini membahas tentang pendeteksian citra lubang dan retak. Beberapa tahap pengolahan citra dan *Extract feature* yang digunakan yaitu *grayscale, filtering, edge detection* dan *connected component labelling*. Sedangkan metode klasifikasi yang digunakan adalah *Support Machine Learning* (SVM).

Penelitian dengan menggunakan metode lain diberikan pada penelitian dengan judul *Automatic Pavement Crack and Pothole Recognition System for Mobile Android-based Devices*, (Tedeschi dan Benedetto, 2017). Tujuan dari penelitian ini yaitu mendeteksi kerusakan jalan jenis retak dan lubang dengan metode *Cascade*. Presisi yang dihasilkan sebesar 75%.

Penelitian dengan menggunakan metode *deep learning* dilakukan pada penelitian dengan judul Road *Crack Detection using Deep Convolutional Neural Network*, (Zhang, Yang, Zhang dan Zhu, 2016). Penelitian ini mendeteksi citra jalan retak dan citra jalan normal menggunakan data input citra RGB. Dibandingkan dengan metode SVM dan *Boosting*, metode CNN menghasilkan presisi yang paling tinggi, yaitu sebesar 86%.

CNN juga digunakan dalam penelitian dengan judul *Automatic Pavement Crack Detection Based on Structured Prediction with the Convolutional Neural Network*, (Fan et al., 2018). Penelitian ini mendeteksi citra jalan retak dan jalan normal. Citra yang digunakan pada proses klasifikasi menggunakan CNN berupa citra RGB. Hasil proses tersebut dibandingkan dengan hasil yang diperoleh dari metode deteksi tepi Canny, *local thresholding*, dan *CrackForest*. Diketahui bahwa jaringan CNN dapat belajar dari gambar tanpa melalui proses *preprocessing* dan *output* yang dihasilkan sangat memuaskan. Dengan menggunakan 3 jenis dataset dan beberapa jenis pembagian data *training* dan *testing*, didapatkan presisi tertinggi sebesar 96%.

Hasil presisi yang tinggi tersebut menunjukkan bahwa CNN memberikan hasil yang bagus dalam proses mendeteksi dan mengklasifikasi. Perkembangan metode CNN adalah metode YOLO. Salah satu penelitian metode YOLO yang memberikan akurasi yang tinggi dilakukan pada penelitian dengan judul *Object Recognition in Aerial Images Using Convolutional Neural Network*, (Radovic, Adarkwa dan Wang, 2017). Penelitian ini menggunakan metode YOLO untuk mendeteksi pesawat dari citra satelit. Akurasi yang dihasilkan yaitu sebesar 97.8%

Selain itu YOLO juga pernah digunakan dalam penelitian dengan judul *Real Time Object Detection for Autonomous Driving Based on Deep Learning*, (Liu, 2017). Penelitian ini menggunakan metode YOLO untuk mendeteksi objek *real-time*. Penelitian ini menyajikan berbagai eksplorasi potensi YOLO dalam memperkirakan orientasi suatu objek serta melakukan pelatihan model YOLO dengan berbagai macam parameter, ukuran petak, dataset. Penelitian ini juga membandingkan dengan metode YOLO dengan metode deteksi objek berbasis *region* yang lain. YOLO mendapatkan ketepatan orientasi sebesar 84,89% dengan kecepatan pemrosesan 0,031 detik per gambar.

Objek yang menarik dalam penggunaan metode YOLO juga dilakukan pada penelitian dengan judul *Real Time Barcode Detection and Classification Using Deep Learning*, (Hansen dan Nasrollahi, 2017). Penelitian ini mendeteksi dan mengklasifikasi *barcode* dengan menggunakan metode YOLO

dan menghasilkan *success rate* sebesar 80.7%.

2.2 Kerusakan Jalan

Pada bagian ini akan diberikan ulasan tentang Pengertian Jalan dan Jenis kerusakan jalan.

2.2.1 Pengertian Jalan

Jalan raya adalah suatu lintasan yang bermanfaat untuk melewatkan lalu lintas dari suatu tempat ke tempat lain. Karena jalan raya sebagai sarana perhubungan, sehingga kondisi lalu lintas harus lancar, memenuhi syarat teknis dan ekonomis sesuai fungsi, volume, dan sifat-sifat lalu lintas. (Suryadharma dan Susanto, 1999)

2.2.2 Jenis Kerusakan Jalan

Secara garis besar kerusakan dapat dibedakan menjadi dua bagian, yaitu kerusakan struktural dan fungsional. Kerusakan struktural yaitu kerusakan yang mencakup kegagalan perkerasan atau kerusakan dari satu atau lebih komponen perkerasan. Kerusakan jenis ini mengakibatkan perkerasan tidak dapat lagi menanggung beban lalu lintas. Sedangkan kerusakan fungsional yaitu kerusakan yang mengakibatkan keamanan dan kenyamanan pengguna jalan terganggu (Sulaksono, 2001). Di bawah ini merupakan beberapa jenis-jenis kerusakan jalan (Shahin, 1994) :

a. Retak Kulit Buaya (*Alligator Cracking*)
   Retak jenis ini berbentuk sebuah jaring-jaring yang menyerupai kulit buaya, dengan lebar jaring lebih besar atau sama dengan 3 mm. Retak jenis ini disebabkan oleh beban lalu lintas yang terlalu berat dan terus menerus. Beberapa kemungkinan penyebab dalam pembangunan adalah bahan perkerasan/kualitas material kurang baik sehingga menyebabkan jalan menjadi rapuh dan pelapukan aspal. Gambar 2.1 adalah contoh dari retak kulit buaya.

b. Keriting (*Corrugation*)
   Bentuk kerusakan ini berupa gelombang pada permukaan jalan yang arahnya melintang jalan. Kerusakan jenis ini terjadi akibat pengereman kendaraan. Salah satu penyebabnya yaitu stabilitas lapis permukaan yang rendah, terlalu banyak menggunakan agregat halus dan pondasi yang bergelombang. Gambar 2.2 adalah contoh dari kerusakan jalan jenis keriting.

Gambar 2.1: *Alligator Cracking*

Gambar 2.2: *Corrugation*

Gambar 2.3: *Depression*

Gambar 2.4: *Pothole*

c. Amblas (*Depression*)

Bentuk kerusakan jenis ini berupa turunnya permukaan lapisan pada lokasi tertentu dengan atau tanpa retak. Umumnya, kedalaman amblas lebih dari 2 cm dan akan menampung atau meresap air. Penyebabnya adalah beban/berat kendaraan yang berlebihan, sehingga struktur bagian bawah tidak mampu menahan beban, penurunan bagian perkerasan dikarenakan oleh turunnya tanah dasar, dan pelaksanaan pemadatan yang kurang baik. Gambar 2.3 adalah contoh dari jalan yang amblas.

d. Lubang (*Potholes*)

Kerusakan jenis ini berbentuk seperti mangkok dan dapat menampung dan meresap air pada bahu jalan. Kerusakan ini terkadang terjadi dekat retakan, atau di daerah yang sering tergenang oleh air. Kemungkinan

penyebabnya adalah seperti kadar aspal rendah, sehingga agregatnya mudah terlepas atau lapis permukaannya tipis, pelapukan aspal, dan suhu campuran tidak memenuhi syarat. Gambar 2.4 adalah contoh kerusakan jenis lubang.

2.3 Perbaikan Kualitas Citra

Perbaikan kualitas citra atau *image enhancement* merupakan salah satu proses awal dalam pengolahan citra. Hal ini diperlukan untuk memperbaiki kualitas citra yang buruk, misalnya terdapat derau (*noise*), citra terlalu gelap atau terang, citra kurang tajam dan sebagainya. Perbaikan kualitas citra adalah proses memperjelas dan mempertajam ciri atau fitur tertentu dari citra agar citra dapat lebih mudah dipersepsi maupun dianalisis secara lebih tetiti.

Beberapa operasi perbaikan kualitas antara lain, pengaturan kecerahan gambar (*brightness adjustment*), peregangan kontras (*contrast stretching*), pengubahan histogram citra, pelembutan citra (*image smoothing*), penapisan derau (*noise filtering*) dan sebagainya. Beberapa diantaranya akan dijelaskan pada subbab berikut :

2.3.1 *Brightness Adjustment*

Brightness adjustment merupakan operasi pixel yang paling sederhana. Penyesuaian tingkat kecerahan suatu citra dapat dinyatakan sebagai :

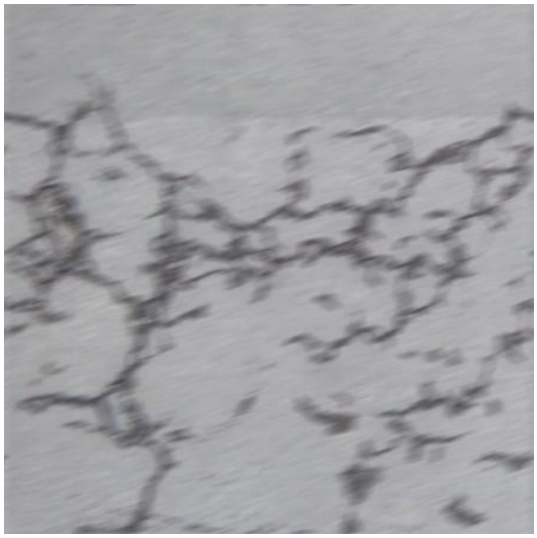
$$U' = U + n \tag{2.1}$$

dengan U' dan U berturut-turut menyatakan citra setelah dan sebelum *brightness adjustment* sedangkan n adalah suatu konstanta yang merupakan variabel penyesuaian. Proses *brightness adjustment* dilakukan dengan menambahkan atau mengurangkan nilai setiap pixel dengan suatu konstanta. Apabila nilai pixel setelah penyesuaian melebihi nilai maksimum intensitas. Nilai maksimum intensitas adalah 255, maka nilai pixel tersebut akan dijadikan 255. Demikian pula sebaliknya, bila nilai pixel hasil penyesuaian lebih kecil dari nol maka nilai pixel tersebut dijadikan nol. (Putra, 2010). Contoh dari citra hasil *Brightness Adjustment* dapat dilihat pada Gambar 2.5

2.3.2 *Motion Blur*

Derau bergerak (*motion blur*) pada citra dapat dilakukan dengan menerapkan *linear spatial filtering*. Dalam Matlab, *code* yang diinputkan adalah `k = fspecial(motion, len, theta)`. *Len* adalah parameter nilai masukan untuk menentukan panjang piksel pergerakan linear dari kamera

Gambar 2.5: Contoh citra dengan pengaturan pencahayaan

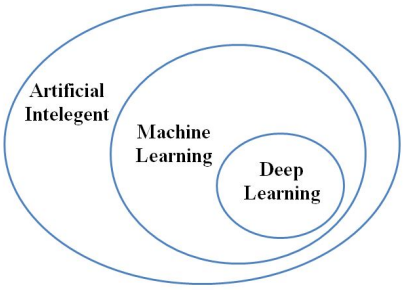


Gambar 2.6: Contoh citra dengan *motion blur*

dengan sudut *theta* searah dengan jarum jam. Pada *filter spasial linier*, nilai baru atau yang difilter dari piksel target ditentukan sebagai kombinasi linear dari nilai piksel di lingkungannya. Mekanika penyaringan spasial linier sebenarnya mengekspresikan bentuk tak tentu yang disebut proses konvolusi. Untuk alasan ini, banyak kernel filter digambarkan sebagai kernel konvolusi, maka secara implisit dipahami bahwa mereka diterapkan pada gambar dalam mode linier. Contoh dari citra hasil *Motion Blur* dapat dilihat pada Gambar 2.6. (Solomon dan Breckon, 2011)

2.4 *Deep Learning*

Gambar 2.7 memperlihatkan bahwa *deep Learning* (pembelajaran mendalam) adalah sub-bidang yang spesifik dari *Machine Learning* (pembelajaran mesin) dan merupakan cara baru untuk mempelajari representasi dari data yang menekankan pada pembelajaran *layer* (lapisan)



Gambar 2.7: Hubungan *Deep Learning* dengan *Machine Learning*

yang berturut-turut dan semakin bermakna. Kata '*deep*' pada *deep Learning* tidak mengacu pada pemahaman yang lebih dalam untuk pendekatan, sebaliknya ini merupakan gagasan dari *layer* yang berurutan. Banyak lapisan yang digunakan pada model data disebut kedalaman ('*depth*') dari model. *Deep learning* moderen sering melibatkan puluhan atau bahkan ratusan *layers* yang berurutan, dan semuanya dipelajari secara otomatis dari data pelatihan. Sementara itu, pendekatan lain untuk *machine learning* cenderung berfokus pada pembelajaran hanya satu atau dua lapis *layer*. Karena hal tersebut, mereka kadang-kadang disebut pembelajaran yang dangkal.

Meskipun *deep learning* adalah sub-bidang dari *machine learning* yang cukup lama, (mulai terkenal di awal tahun 2010-an). Dalam beberapa tahun kemudian, *deep learning* telah mencapai tidak lebih dari sebuah revolusi di lapangan, dengan hasil yang luar biasa pada masalah persepsi seperti melihat dan mendengar, masalah yang melibatkan keterampilan yang tampak alami dan intuitif bagi manusia tetapi telah lama sulit dipahami oleh mesin. Beberapa terobosan yang telah dicapai *deep learning* adalah peningkatan konversi *text-to-speech*, asisten digital seperti *Google Now* dan *Amazon Alexa* dan kemampuan untuk menjawab pertanyaan-pertanyaan bahasa alami. Beberapa metode dari *deep learning* adalah *Long Short Term Memory* (LSTM) dan *Convolutional Neural Network* (CNN) (Mooij, Bagulho dan Huisman, 2018).

2.5 *Convolutional Neural Network* (CNN)

Pada tahun 2012, Alex Krizhevsky dengan metode pengembangan CNN miliknya berhasil menjuarai kompetisi *ImageNet Large Scale Visual Recognition Challenge 2012*. Prestasi tersebut menjadi pembuktian bahwa metode CNN berhasil mengungguli metode *Machine Learning* lainnya dalam kasus klasifikasi objek pada citra.

Metode CNN merupakan pengembangan dari Metode *Multilayer Perceptron* (MLP) yang didesain untuk mengolah data dua dimensi. Seperti yang terlihat pada Gambar 2.8, cara kerja CNN memiliki kesamaan pada MLP, namun dalam CNN setiap neuron dipresentasikan dalam bentuk tiga dimensi, tidak seperti MLP yang setiap neuron hanya berukuran satu dimensi. (Putra, 2016)

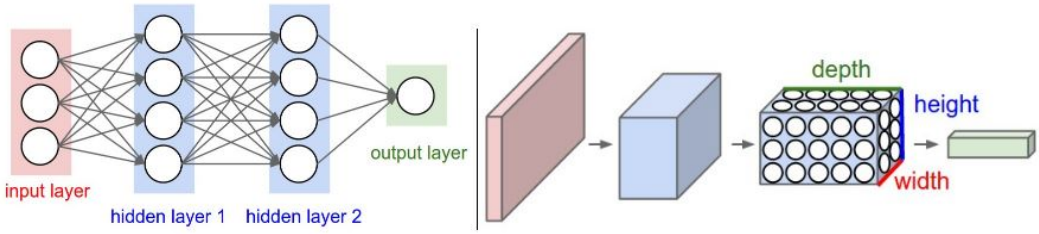
Terdapat beberapa arsitektur CNN yang umum digunakan. Arsitektur tersebut yaitu LeNet, AlexNet, ZF Net, GoogLeNet, VGGNet dan ResNet. Dapat dilihat pada contoh arsitektur CNN pada Gambar 2.9, CNN terdiri dari tiga jenis layer, yaitu *convolutional layer* (Conv), *pooling layer* (Max pooling) dan *fully-connected layer* (Full Connection). Tumpukan lapisan tersebut membentuk arsitektur dari CNN.

Fungsi dasar dari CNN di atas dapat dipecah menjadi empat bidang utama, yaitu (O'Shea dan Nash, 2015):

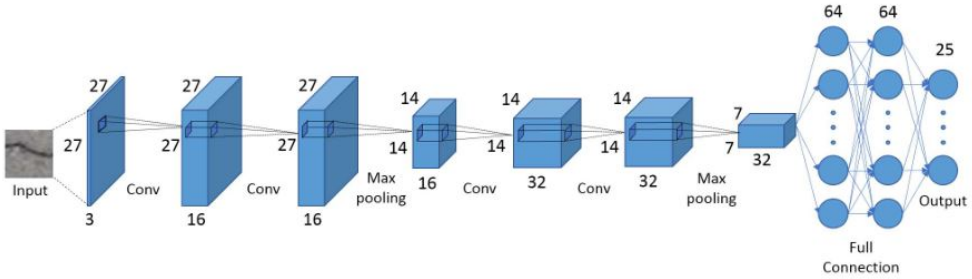
1. *Input layer*. Layer yang akan menyimpan nilai piksel dari citra masukan. Pada Gambar 2.9, ukuran dari data input yaitu 27x27x3, artinya 27x27 merupakan ukuran piksel citra dan 3 merupakan banyak chanel citra, yaitu *Red, Green, Blue*.

2. *Convolutional layer*. Layer ini akan menentukan keluaran neuron yang terhubung ke *input layer* melalui perhitungan skalar produk antara bobot dan daerah yang terhubung dengan *input*. Ilustrasi dari proses dalam *convolutional layer* dapat dilihat pada Gambar 2.10. Pada ilustrasi tersebut digunakan 1 zero *padding*, yaitu penambahan 1 baris nilai nol disepanjang garis batas input. Dalam proses konvolusi juga digunakan *Rectified Linear Unit* (biasa disingkat menjadi 'ReLu') bertujuan untuk menerapkan fungsi aktivasi '*elementwise*' seperti sigmoid ke *output* dari aktivasi yang dihasilkan oleh lapisan sebelumnya, grafik ReLu, dapat dilihat pada Gambar 2.11.

3. *Pooling layer*. Layer ini akan melakukan *downsampling* di sepanjang dimensi spasial dari *input* yang diberikan, selanjutnya mengurangi jumlah parameter dalam aktivasi tersebut. *Pooling layer* mengoperasikan peta aktivasi ke seluruh *input*, dan menggunakan fungsi "MAX". Di sebagian besar CNN, *max-pooling layer* menggunakan kernel dengan dimensi 2x2 dengan *stride* 2 di sepanjang dimensi spasial *input*, artinya berpindah sebanyak 2 langakah dalam pergerakan kernelnya. Hal



Gambar 2.8: Dimensi MLP (kiri) dan CNN (kanan)



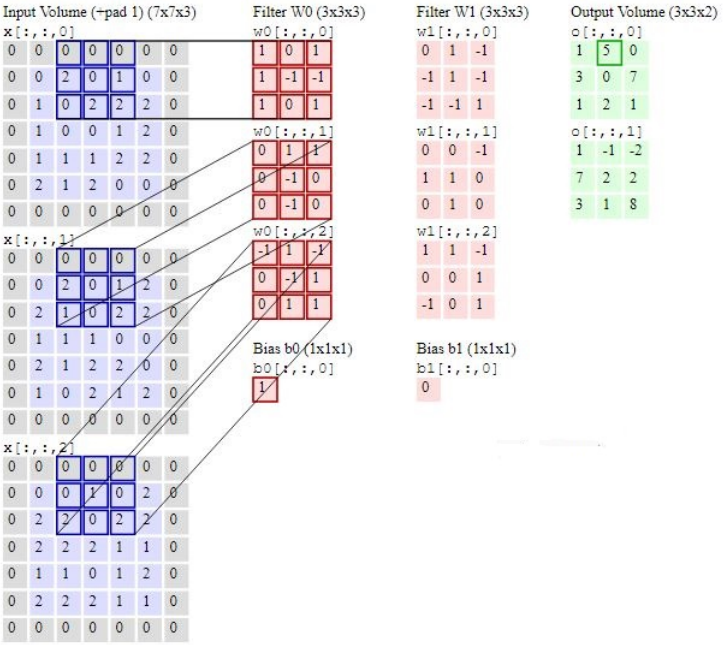
Gambar 2.9: Contoh Arsitektur CNN

ini mengakibatkan ukuran *input* turun sampai 25% dari ukuran aslinya. Ilustrasi dari proses ini disajukan pada Gambar 2.12. Pada ilutrasi tersebut digunakan kernel dengan dimensi 2x2 dengan *stride* 1.

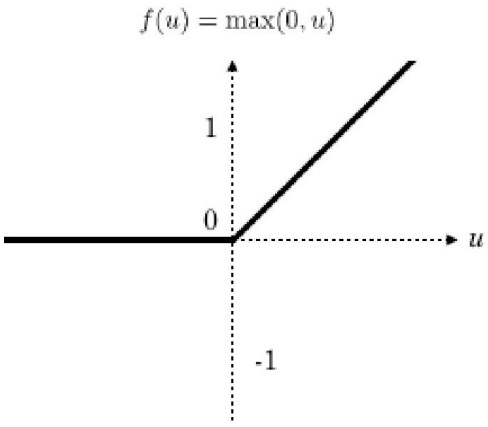
4. *Fully-connected layer*. Layer ini akan melakukan tugas yang sama dengan Jaringan Syaraf standar dan berusaha menghasilkan nilai kelas dari aktivasi, yang digunakan untuk klasifikasi. Lapisan ini sejalan dengan cara neuron yang diatur dalam JST.

2.6 YOLO (*You Only Look Once*)

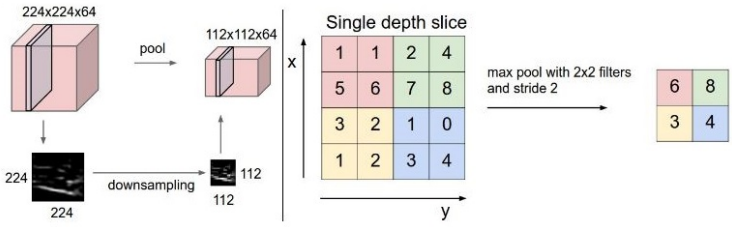
Saat ini, untuk mendeteksi sebuah objek, sistem menggunakan *clasifier* untuk objek tersebut dan mengevaluasinya diberbagai lokasi dan skala pada citra. Sitem DPM (*Deformable Parts Model*) menggunakan pendekatan jendela berjalan (*sliding window*) dimana *clasifier* dijalankan merata diseluruh citra. Sedangkan R-CNN (*Regional Convolution Neural Network*) menggunakan metode usulan daerah (*region proposal*) untuk terlebih dahulu membangkitkan kotak pembatas yang berpotensi di sebuah citra dan menjalankan *clasifier* untuk usulan kotak pembatas tersebut. Setelah mengklasifikasi, *post-processing* digunakan untuk menyaring kotak pembatas, mengeliminasi pendeteksian objek yang ganda, dan membandingkan kotak prediksi terhadap objek lain. Alur yang rumit ini membutuhkan waktu yang



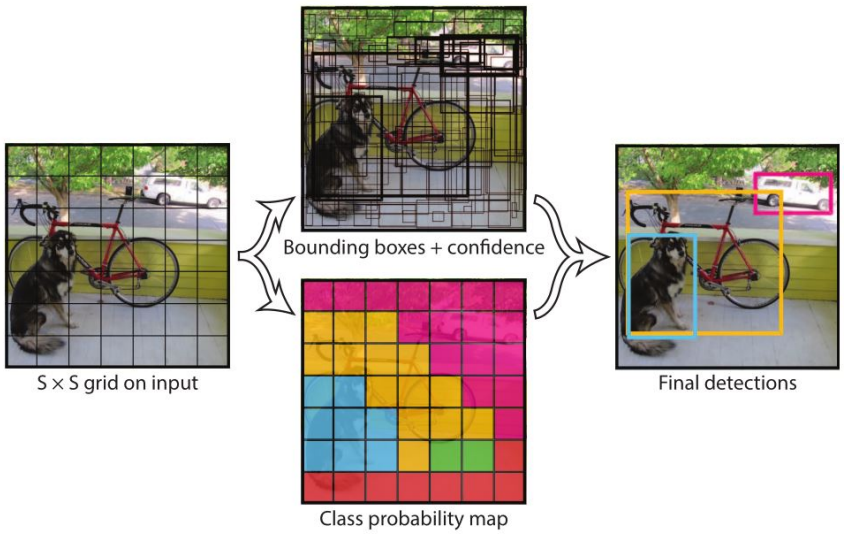
Gambar 2.10: Proses Konvolusi



Gambar 2.11: ReLu Activation



Gambar 2.12: Max Pooling



Gambar 2.13: Model YOLO
(Redmon et al., 2016)

lama dan sulit untuk dipotimalkan karena setiap komponen harus dilatih secara terpisah.

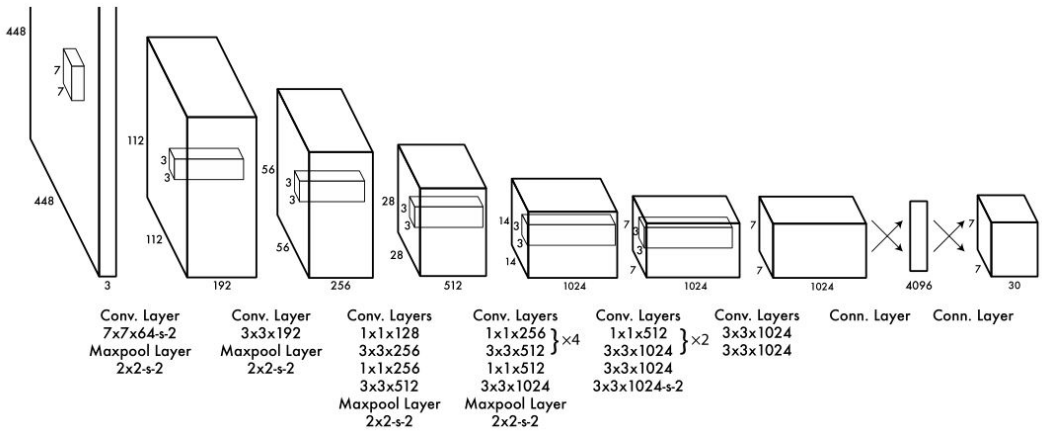
YOLO menjadikan proses pendeteksian objek sebagai masalah regresi tunggal, yang memproses langsung dari piksel gambar hingga koordinat kotak pembatas dan probabilitas kelas. Dengan menggunakan YOLO, sistem hanya melihat sekali (*You Only Look Once*) pada gambar untuk memprediksi benda apa yang ada dan dimana tempatnya. (Redmon et al., 2016)

2.6.1 Desain Jaringan

YOLO menyatukan komponen yang terpisah dari pendeteksian objek menjadi satu jaringan syaraf. YOLO memanfaatkan fitur dari keseluruhan gambar untuk memprediksi setiap kotak pembatas. YOLO memprediksi semua kotak pembatas pada semua kelas objek untuk sebuah gambar secara bersamaan. Ini berarti YOLO mempertimbangkan seluruh bagian citra secara global dan semua objek pada citra.

YOLO membagi gambar masukan menjadi $S \times S$ petak (*grid*). Jika pusat objek ada di dalam suatu petak, sel petak tersebut bertanggung jawab untuk mendeteksi objek itu.

Setiap sel petak memprediksi B kotak pembatas dan nilai keyakinan untuk kotak-kotak tersebut. Nilai keyakinan ini mencerminkan seberapa yakin kotak itu berisi objek dan juga seberapa akuratnya kotak yang diprediksi itu. Secara formal YOLO mendefinisikan kepercayaan sebagai $Pr(Object) * IoU^{truth}_{pred}$.



Gambar 2.14: Arsitektur dasar YOLO
(Redmon et al., 2016)

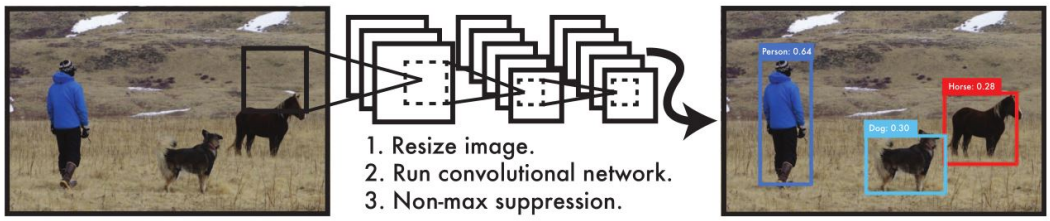
Jika tidak ada objek di sel itu, nilai keyakinan harus nol. Jika tidak, nilai keyakinan akan sama dengan *intersection of union* (IoU) antara kotak yang diprediksi dan kotak kebenaran latar belakang (*ground truth*).

Setiap kotak pembatas terdiri dari 5 prediksi: x, y, w, h, dan nilai keyakinan p. Koordinat (x, y) mewakili pusat kotak relatif terhadap batas sel petak. Lebar (w) dan tinggi (h) diprediksi relatif terhadap keseluruhan gambar. Akhirnya prediksi nilai keyakinan menyatakan IoU antara kotak yang diprediksi dan kotak *ground truth*.

Setiap sel petak juga memprediksi probabilitas kelas kondisional C, $Pr(Class_i|Object)$. Probabilitas ini dikondisikan pada sel petak yang berisi objek. YOLO memprediksi satu set probabilitas kelas per sel petak, berapapun jumlah kotak B. Pada saat uji coba kita mengalikan probabilitas kelas kondisional dan prediksi keyakinan kotak individu yang memberi nilai keyakinan khusus kelas untuk setiap kotak, dan ditunjukkan pada persamaan 2.2. Nilai ini menyandikan probabilitas kelas yang muncul di kotak dan seberapa baik kotak yang diprediksi sesuai dengan objek. Ilustrasi model dapat dilihat pada Gambar 2.13.

$$Pr(Class_i|Object) * Pr(Object) * IoU_{pred}^{truth} = Pr(Class_i) * IoU_{pred}^{truth} \quad (2.2)$$

YOLO menerapkan model ini sebagai CNN. *Convolutional layer* awal dari jaringan mengekstrak fitur dari citra sementara *fully-connected layer* memprediksi probabilitas dan koordinat keluaran.



Gambar 2.15: Sistem deteksi YOLO
(Redmon et al., 2016)

YOLO versi dasar memiliki 24 *convolutional layer* yang diikuti oleh 2 *fully-connected layer*. Arsitektur ini mengunakan lapisan reduksi 1×1 yang diikuti oleh lapisan konvolusi 3×3. Jaringan lengkap arsitektur YOLO versi dasar ditunjukkan pada Gambar 2.14. Sedangkan YOLOv1-tiny menggunakan layer konvolusi lebih sedikit, yaitu 9 layer dengan parameter yang sama seperti YOLO versi dasar. (Redmon et al., 2016)

YOLO menggunakan fungsi aktivasi linier untuk lapisan akhir dan semua lapisan lainnya menggunakan aktivasi *leaky rectified* berikut ini :

$$\Phi(x) = \begin{cases} x, & \text{jika } x > 0 \\ 0.1x, & \text{yang lain} \end{cases} \tag{2.3}$$

2.6.2 Keunggulan YOLO

Langkah pengerjaan metode YOLO versi dasar (v1) dapat dilihat pada Gambar 2.15, *Convolutional Neural Network* (CNN) secara bersamaan memprediksi beberapa kotak pembatas dan probabilitas kelas untuk kotak-kotak itu. YOLO melatih seluruh bagian gambar dan mengoptimalkan kinerja pendeteksian secara langsung. Model terpadu ini memiliki beberapa keunggulan dibandingkan metode tradisional deteksi objek.

Pertama, YOLO sangat cepat. Karena YOLO menjadikan pendeteksian objek sebagai masalah regresi sehingga tidak memerlukan alur yang kompleks. YOLO hanya menjalankan CNN pada citra untuk memprediksi pendeteksian. YOLO dapat memproses *streaming* video secara *real-time* dengan kurang dari 25 milidetik latensi dan mencapai lebih dari dua kali lipat MAP (*Mean Average Precision*) sistem *real-time* lainnya.

Kedua, YOLO mempertimbangkan secara global tentang citra saat membuat prediksi. Tidak seperti *sliding window* dan teknik berbasis *region proposal*, YOLO melihat keseluruhan gambar selama masa pelatihan dan

pengujian sehingga secara implisit mengkodekan informasi kontekstual tentang kelas sesuai objek yang ditampilkan pada citra. Fast R-CNN, metode deteksi yang paling bagus, mempunyai kesalahan mendeteksi latar belakang pada citra untuk objek karena tidak dapat melihat konteks yang lebih besar. YOLO membuat kurang dari setengah jumlah kesalahan latar belakang dibandingkan dengan Fast R-CNN.

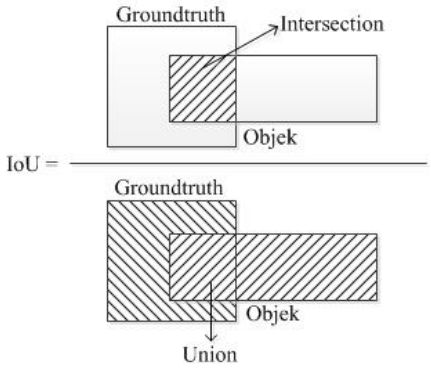
Ketiga, YOLO dapat mempelajari menggeneralisasi representasi objek. Saat dilatih tentang gambar alami dan diuji pada karya seni, YOLO mengungguli metode pendeteksian terbaik seperti DPM dan R-CNN dengan selisih lebar. Karena YOLO sangat digeneralisasikan, kemungkinan besar akan rusak saat diterapkan pada domain baru atau masukan tak terduga. (Redmon et al., 2016)

2.7 *Intersection over Union* (IoU)

Ketika mendeteksi letak objek, algoritma deteksi objek harus dipertimbangkan. Beberapa algoritma pendeteksian mungkin memerlukan pendeteksian letak objek dengan akurasi tinggi, sementara yang lain lebih toleran terhadap kesalahan dalam penempatan kotak pembatas. Keakuratan kotak biasanya diukur dengan menggunakan *Intersection over Union* (IoU). IoU menghitung area pertemuan antara kotak prediksi objek dan kotak kebenaran dasar (*ground truth*) dan membaginya dengan area persatuan mereka. Perumusan IoU ditunjukkan pada 2.4 sedangkan ilustrasi dari perumusan tersebut disajikan pada Gambar 2.16.

$$IoU = \frac{Area\ of\ overlap}{Area\ of\ union} \tag{2.4}$$

Saat mengevaluasi algoritma pendeteksian objek, ambang IoU sebesar 0.5 biasanya digunakan untuk menentukan apakah deteksi benar (Everingham, Van Gool, Williams, Winn dan Zisserman, 2010). Namun nilai IoU = 0.5 mempunyai area yang cukup longgar. Sehingga umumnya diinginkan nilai IoU yang lebih besar dari 0.5. (Zitnick dan Dollár, 2014)



Gambar 2.16: Ilustrasi perhitungan IOU

BAB 3
METODE PENELITIAN

Pada bab ini akan dijelaskan langkah-langkah yang dilakukan dalam penyelesaian permasalahan yang diangkat dalam penelitian ini.

3.1 Tahapan Penelitian

Adapun langkah-langkah dalam menyelesaikan permasalahan pada penelitian ini adalah sebagai berikut :

1. Studi literatur
   Tahap studi literatur yaitu melakukan pendalaman materi tentang tahap pengolahan citra, serta untuk pengklasifikasian citra yaitu metode CNN dan YOLO. Literatur yang digunakan yaitu berdiskusi, membaca buku, jurnal, artikel, dan website yang membahas tentang materi tersebut.

2. Pengumpulan Data
   Data yang digunakan dalam penelitian ini antara lain citra jalan beraspal yang terdapat kerusakan jalan jenis lubang, jenis retak garis (*longitudinal dan transversal*), jenis retak *non*-garis, serta citra jalan yang memuat kombinasi ketiganya. Data yang didapatkan yaitu merupakan data primer. Pengambilan data dilakukan saat kondisi langit cerah dan posisi lurus menghadap permukaan kerusakan jalan.

3. Pembagian Data
   Tahap pembagian data yaitu membagi keseluruhan data untuk proses pelatihan dan uji validasi. Data yang digunakan untuk pelatihan yaitu sebanyak 80% sedangkan untuk data uji validasi sebanyak 20% dari total seluruh data.

4. *Anotasi Data*
   Tahap anotasi data adalah tahap memberikan keterangan pada setiap data citra. Keterangan tersebut berupa koordinat, tingkat kepercayaan kotak pembatas dan jenis kerusakan dari setiap objek kerusakan jalan yang terdapat pada citra. Keterangan ini nantinya digunakan untuk proses pelatihan dan pengujian.

5. Perancangan dan Implementasi Aplikasi Deteksi Kerusakan Jalan
   Pada penelitian ini, metode yang digunakan untuk mendeteksi dan mengklasifikasi jenis kerusakan jalan adalah metode YOLOv1-tiny dengan menggunakan *pre-trained weight* dan model dari *darknet open source* YOLO. Sehingga perlu untuk menyesuaikan kebutuhan *hardware* dan *software* dalam mengimplementasikan YOLOv1-tiny dari darknet untuk dataset yang digunakan dalam penelitian ini.

6. Uji Coba dan Evaluasi
   Proses uji coba dilakukan dengan melakukan proses pengujian metode dengan menguji beberapa citra baru yang telah diberi gangguan. Proses uji coba dilakukan untuk mengetahui batasan gangguan agar objek tetap dapat dikenali. Sedangkan proses Evaluasi dilakukan untuk mengetahui tingkat keakuratan kinerja model dan untuk menganalisis hasil uji coba validasi. Validasi kinerja program dilakukan dengan menghitung nilai IoU antara prediksi kotak pembatas dengan kotak pembatas yang sebenarnya dan menghitung akurasi klasifikasi objek.

3.2 Blok Diagram Deteksi Kerusakan Jalan Menggunakan YOLO

Diagram tahapan deteksi dan klasifikasi ketusakan jalan ditunjukan pada Gambar 3.1. Proses utama dalam mendeteksi dan mengklasifikasi kerusakan jalan adalah sebagai berikut :

1. *Pre-processing Data*
   Tahap *pre-processing* data dilakukan sebelum proses pelatihan dan pengujian. Tahap *preprocessing* untuk proses pelatihan terdiri dari dua tahap, yaitu proses *resizing* dan *cropping* data citra serta anotasi data. Proses *resizing dan cropping* dilakukan untuk untuk keperluan data input YOLO, citra disamakan ukurannya menjadi 416x416 sesuai dengan ketentuan literatur acuan (Redmon et al., 2016).
   Sedangkan tahap *pre-processing* untuk proses pengujian tidak hanya melawati proses tersebut, namun juga melewati proses pemberian ganguan pada citra, yaitu *blurrring*, dan *bright adjustment*.
   Proses anotasi data yaitu proses memberi keterangan setiap data citra berupa informasi kotak pembatas dan kelas objek pada citra.

2. Merancang Model.
   Metode yang digunakan untuk proses deteksi dan klasifikasi pada

penelitian ini adalah YOLOv1-tiny. Langkah langkah metode YOLOv1-tiny dijelaskan pada bagian berikut :

(a) Menginputkan data citra RGB hasil *pre-processing*, yaitu citra dengan ukuran $416 \times 416 \times 3$

(b) Mendapatkan data *pre-trained weight* sebagai bobot awal dan model jaringan YOLO

(c) Melakukan proses pelatihan yaitu melakukan *fine tuning* bobot awal dan model YOLOv1-tiny dengan menggunakan dataset pelatihan pada penelitian ini. Arsitektur YOLOv1-tiny mempunyai arsitektur 9 *convolutional layer* dan 6 *maxpool layer*.

(d) Melakukan proses pengujian terhadap beberapa citra pengujian dengan menggunakan bobot hasil proses pelatihan

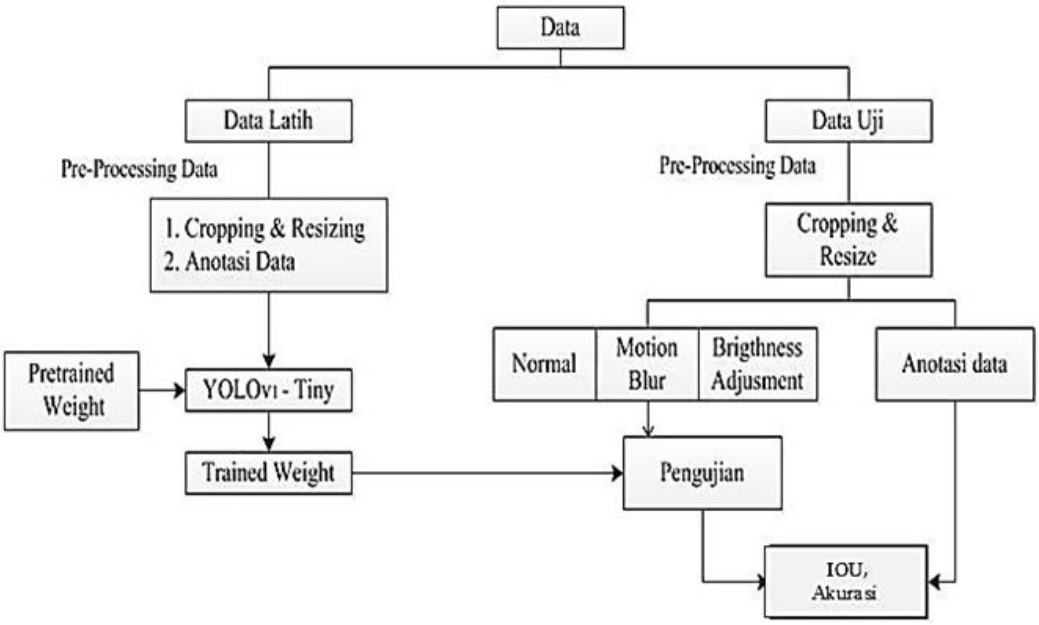
(e) Menghitung kinerja metode dengan mencari nilai IoU prediksi kotak pembatas yang dijelaskan pada bagian 2.4 dan akurasi klasifikasi objek dengan menggunakan persamaan berikut :

$$\text{Akurasi} = (\text{TP} + \text{TN})/(\text{TP} + \text{TN} + \text{FP} + \text{FN})$$

atau

$$\text{Akurasi} = (\text{TP} + \text{TN})/\text{total objek} \tag{3.1}$$

dengan TP (*true positive*) adalah kelas suatu objek yang terklasifikasi dengan dengan benar, TN (*true negative*) adalah kelas objek lain yang terklasifikasi dengan benar, FP (*false positive*) adalah kelas suatu objek yang tidak terklasifikasi dengan benar, sedangkan FN (*false negative*) adalah kelas objek lain yang tidak terklasifikasi dengan benar. (Ryu et al., 2015)



Gambar 3.1: Diagram Alur Deteksi Kerusakan Jalan Menggunakan YOLO

BAB 4
PERANCANGAN DAN IMPLEMENTASI PROGRAM

Pada bab ini akan dijelaskan rancangan pendeteksian dan pengklasifikasian kerusakan jalan aspal menggunakan metode YOLO. Selanjutnya akan dijelaskan pula implementasi dari rancangan tersebut pada sebuah perangkat lunak untuk mengetahui kinerja dari metode yang digunakan.

4.1 Perancangan Data

Proses pendeteksian dan pengklasifikasian objek berbasis citra digital memerlukan sejumlah besar data untuk dilatih, sehingga mendapatkan bobot optimal dalam pendeteksian dan pengklasifikasian objek. Selain itu, diperlukan pula sejumlah data lain untuk proses pengujian untuk menguji keakuratan dari metode. Data yang telah terkumpul dalam penelitian ini sejumlah 300 data citra yang terdiri dari citra dengan berbagai jenis kerusakan jalan, diantaranya lubang, retak garis yaitu retak *longitudinal* dan retak *transversal*, serta retak *non*-garis. Data tersebut dibagi menjadi 80% untuk proses pelatihan yaitu sebanyak 240 data dan 20% untuk proses pengujian yaitu sebanyak 60 data.

4.1.1 Kebutuhan Perangkat Lunak

Hal yang diperlukan dalam mengimplementasi metode yang digunakan dalam penelitian ini yaitu seperti perangkat keras dan perangkat lunak. Rincian dari hal tersebut akan disajikan dalam Tabel 4.1

Tabel 4.1: Kebutuhan Perancangan

Perangkat Keras	1. Intel(R) Core(TM) i3-4005U CPU 1.70GHz 1.70 GHz 2. Memory 4.00GB RAM
Perangkat Lunak	1. Sistem operasi Microsoft Windows 10 Pro 64-bit 2. Python Anaconda 3.6 3. OpenCV 3.0 4. Tensorflow CPU

Proses yang hendak dibangun adalah proses deteksi dan klasifikasi citra kerusakan jalan berbasis citra digital. Jenis kerusakan jalan yang dapat

Gambar 4.1: Citra bentuk awal

dikenali yaitu jenis lubang, retak garis, dan retak *non*-garis. Proses deteksi dan klasifikasi pada penelitian ini dapat mengenali jenis keruskan jalan dan posisinya pada citra walaupun terdapat gangguan pada citra masukan. Gangguan yang dapat ditangani meliputi gangguan objek pada jalan, kualitas citra yang kabur, atau kondisi pencahayaan yang terang dan gelap. Proses deteksi dan klasifikasi ini mampu mengenali beberapa objek kerusakan dalam suatu citra dan menunjukkannya pada sebuah kotak pembatas.

4.1.2 Data Awal

Sebelum melakukan proses pelatihan dan proses pengujian, dilakukan tahap *resizing* dan *cropping* pada seluruh data citra. Data citra awal adalah citra RGB dengan format .jpg dan berukuran 1920×2560 piksel. Contoh citra bentuk awal diperlihatkan pada Gambar 4.1. Pada tahap ini seluruh data citra dibentuk menjadi persegi dengan melakukan *cropping* citra. Selanjutnya citra akan diatur ulang ukurannya menjadi 416×416 piksel. Proses tersebut merupakan proses *resizing* citra. Tahap *cropping* dan *resizing* citra bertujuan untuk keperluan data input YOLO. Gambar 4.2 adalah contoh hasil citra pada tahap ini.

Data latih dan uji yang digunakan dalam penelitian ini terdiri dari citra dari berbagai jenis kerusakan jalan beserta keterangan jenis kerusakan. Citra

Gambar 4.2: Citra hasil *Resizing* dan *Cropping*

didapatkan secara manual menggunakan *smartphone*, sehingga keterangan objek untuk setiap citra perlu dibangun terlebih dahulu. Keterangan yang dimaksud merupakan informasi tentang letak objek dan kelas objek yang akan dideteksi dan diklasifikasi. Keterangan ini berfungsi sebagai target atau acuan untuk memperoleh bobot pada saat proses pelatihan data dan menjadi pembanding nilai *output* pada saat proses pengujian. Proses lebih disebut anotasi data dan lebih detail akan dijelaskan pada subbab 4.2.1

4.1.3 *Pre-trained Weights*

Dalam proses pelatihan menggunakan model dari *darknet open source*, diperlukan bobot awal dari sumber yang sama. Dalam pelatihan ini, bobot awal yang digunakan bukan data *random*, tetapi data *pre-trained weights* yang telah dilatih sebelumnya pada jaringan YOLO untuk suatu dataset. Sedangkan, untuk melakukan pendeteksian dan pengklasifikasian dataset baru, diperlukan proses pelatihan dataset yang akan digunakan sehingga mendapatkan bobot baru yang sesuai dengan dataset yang dimiliki. Pelatihan dataset baru dengan menggunakan bobot awal dari *pre-trained weight* membuat proses pelatihan berjalan lebih cepat. Data *pre-trained weights* untuk model YOLOv1-tiny pada penelitian ini diunduh dari `https://pjreddie.com/darknet/yolo/` dengan format file .weights.

4.1.4 Dataset Proses Pengujian

Sebanyak 60 data baru digunakan untuk proses pengujian. Data tersebut terdiri dari dataset tanpa gangguan dan dataset dengan gangguan. Terdapat 3 jenis gangguan yang diberikan pada dataset, yaitu gangguan citra kabur menggunakan proses *motion blurring*, lalu gangguan dengan penambahan intensitas cahaya dan pengurangan intensitas cahaya citra. Gangguan diberikan dalam beberapa tingkatan. Hal ini dilakukan sebagai antisipasi pendeteksian dan pengklasifikasian citra dengan kualitas pengambilan data

yang tidak bagus. Dataset tanpa gangguan atau dataset asli yaitu dataset yang berisi 60 data citra dan keseluruhan datanya tidak diberi gangguan. Hal ini dilakukan untuk mengetahui keakuratan model dalam mendeksi pola data pelatihan. Tabel 4.2 menunjukkan rincian dari dataset yang digunakan saat proses pengujian. Dapat dilihat pada tabel tersebut, total datset yang didapatkan adalah 11 dataset.

Tabel 4.2: Dataset Proses Pengujian

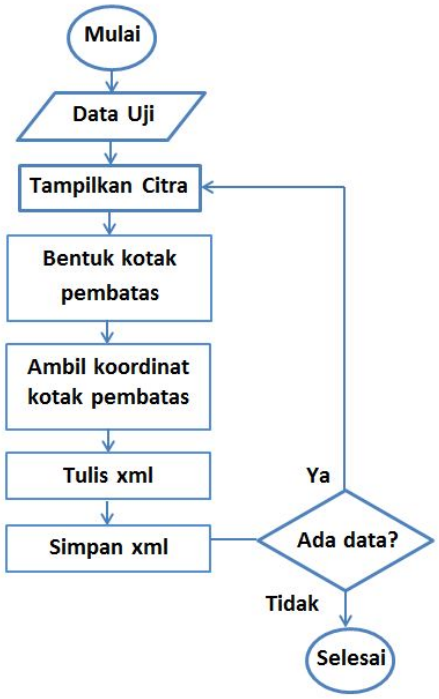
Dataset	Parameter	Total
Citra tanpa gangguan	-	60
Citra gangguan *Blur*	Panjang *Motion* (l) = 3	60
	Panjang *Motion* (l) = 7	60
	Panjang *Motion* (l) = 15	60
Citra gangguan terang	Intensitas warna (n) = 30	60
	Intensitas warna (n) = 50	60
	Intensitas warna (n) = 80	60
Citra gangguan gelap	Intensitas warna (n) = -30	60
	Intensitas warna (n) = -50	60
	Intensitas warna (n) = -80	60
	Intensitas warna (n) = -100	60

4.2 Perancangan Proses

Secara umum proses pendeteksian dan pengklasifikasian lubang jalan menggunakan YOLO dapat ditunjukkan dalam diagram alir yang telah disajikan dalam Gambar 3.1. Rincian tentang perancangan proses pendeteksian dan pengklasifikasian kerusakan jalan dengan menggunakan metode YOLO akan dijelaskan pada subbab ini. Secara garis besar, tahapan pengerjaan untuk pendeteksian dan pengklasifikasian adalah sebagai berikut :

1. Mempersiapkan hal-hal yang dibutuhkan dalam proses pelatihan model seperti yang telah disebutkan pada subbab 4.1. Diantaranya yaitu memenuhi kebutuhan penggunaan perangkat lunak, menyiapkan data yang digunakan saat pelatihan yaitu dataset citra latih, data anotasi, bobot awal atau *pre-trained weights* dan model jaringan YOLOv1-tiny.

2. Mempersiapkan hal-hal yang dibutuhkan untuk pengujian. Hal-hal ini juga telah disebutkan pada subbab 4.1. Diantaranya yaitu mempersiapkan dataset pengujian, bobot hasil pelatihan dan model jaringan.



Gambar 4.3: Diagram Alur Anotasi Data

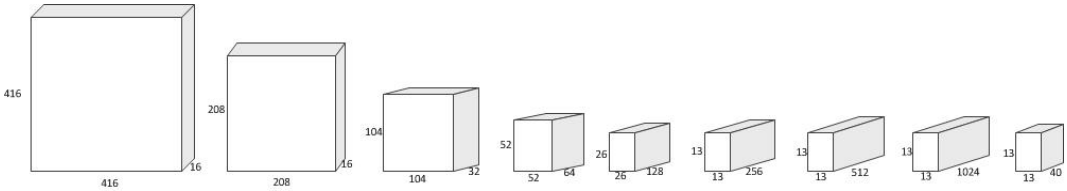
3. Melakukan proses pelatihan dan proses pengujian dengan menggunakan semua dataset proses pengujian

4. Melakukan uji keakuratan dengan menghitung nilai akurasi klasifikasi dan IoU kotak pembatas

4.2.1 Persiapan Data

Hal yang perlu dilakukan sebelum membangun proses pendeteksian dan pengklasifikasian adalah dengan mempersiapkan data yang dibutuhkan. Seluruh data baik data untuk proses pelatihan dan pengujian akan melalui proses *cropping* dan *resizing*. Proses ini dilakukan secara manual.

Selanjutnya, diperlukan data anotasi seluruh citra untuk proses pelatihan dan pengujian. Diagram Alur proses anotasi data disajikan pada Gambar 4.3. Proses anotasi data dimulai dengan menggambar kotak pembatas di setiap objek pada citra lalu menyimpan keterangan kotak pembatas tersebut dalam suatu file. Isi yang disimpan pada file adalah nilai $(x_1, y_1), (x_2, y_2), p, c$ berturut-turut adalah koordinat ujung awal kotak pembatas, koordinat ujung akhir kotak pembatas, tingkat kepercayaan kotak pembatas dan kelas objek. Penjelasan lebih detail tentang data anotasi akan diberikan pada subbab 4.3.1.



Gambar 4.4: Arsitektur Jaringan YOLOv1-tiny

Proses menggambar kotak pembatas dilakukan seakurat mungkin dan konsisten. Karena nilai-nilai yang dihasilkan dari pembentukkan kotak pembatas nantinya akan digunakan sebagai target untuk proses pelatihan. Hal tersebut akan dapat mempermudah proses pelatihan walaupun data yang digunakan tidak banyak. Proses selanjutnya adalah membuat beberapa dataset pengujian sesuai penjelasan pada subbab 4.1.4.

4.2.2 Pelatihan YOLOv1-tiny dengan dataset baru

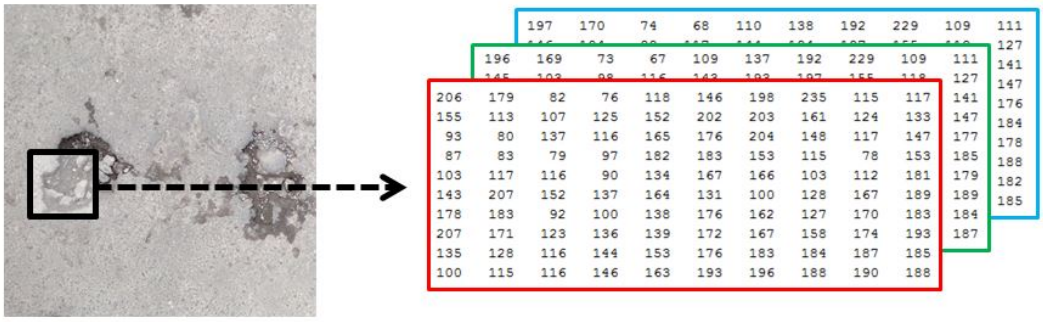
Hal-hal yang diperlukan dalam proses pelatihan adalah dataset citra latih yang sudah melewati tahap *cropping* dan *resizing* serta anotasi data. File *pre-trained weight* dengan format .weight dan model jaringan YOLOv1-tiny dengan format .cfg.

Proses pelatihan YOLOv1-tiny yaitu melakukan *fine tuning pre-trained weight* YOLO dan model jaringan YOLOv1-tiny menggunakan dataset dalam penelitian ini. Setelah melakukan proses pelatihan, didapatkan bobot baru (*trained weight*) yang sesuai dengan dataset. File tersebut juga dalam format .weights. Jaringan YOLOv1-tiny mempunyai 6 layer konvolusi yang diikuti *maxpool*, dan 3 layer konvolusi. Jaringan tersebut diilustrasikan pada Gambar 4.4 dan dijelaskan dengan detail sebagai berikut :

1. convolutional 1 : filters=16, kernel = 3×3, stride=1, padding=1
   activation : leaky
   maxpool 1 : size=2, stride=2

2. convolutional 2 : filters=32, kernel = 3×3, stride=1, padding=1
   activation : leaky
   maxpool 2 : size=2, stride=2

3. convolutional 3 : filters=64, kernel = 3×3, stride=1, padding=1
   activation : leaky
   maxpool 3 : size=2, stride=2



Gambar 4.5: Ilustrasi representasi citra awal

4. convolutional 4 : filters=128, kernel = 3×3, stride=1, padding=1
   activation : leaky
   maxpool 4 : size=2, stride=2

5. convolutional 5 : filters=256, kernel = 3×3, stride=1, padding=1
   activation : leaky
   maxpool 5 : size=2, stride=2

6. convolutional 6 : filters=512, kernel = 3×3, stride=1, padding=1
   activation : leaky
   maxpool 6 : size=2, stride=1

7. convolutional 7 : filters=1024, kernel = 3×3, stride=1, padding=1
   activation : leaky

8. convolutional 8 : filters=1024, kernel = 3×3, stride=1, padding=1
   activation : leaky

9. convolutional 9 : filters=40, kernel = 1×1, stride=1, padding=1
   activation : linear

Sebagai ilustrasi perhitungan, diberikan contoh kasus jaringan yang digunakan melalui Gambar 4.5. Dalam gambar tersebut terdapat tiga lapis matriks berukuran 10×10 yang merupakan representasi sub-bagian dari citra masukkan. Terdapat tiga lapis layer yang berarti citra tersebut mempunyai tiga *channel* warna, yaitu *red, green, blue*.

Pada poin pertama detail jaringan diatas, *padding* yang digunakan adalah 1. Artinya, matriks representasi citra tersebut diberikan *zero-padding* sebanyak satu baris dan satu kolom. Ilustrasi hasilnya dapat dilihat pada

0	0	0	0	0	0	0	0	0	0	0	0
0	206	179	82	76	118	146	198	235	115	117	0
0	155	113	107	125	152	202	203	161	124	133	0
0	93	80	137	116	165	176	204	148	117	147	0
0	87	83	79	97	182	183	153	115	78	153	0
0	103	117	116	90	134	167	166	103	112	181	0
0	143	207	152	137	164	131	100	128	167	189	0
0	178	183	92	100	138	176	162	127	170	183	0
0	207	171	123	136	139	172	167	158	174	193	0
0	135	128	116	144	153	176	183	184	187	185	0
0	100	115	116	146	163	193	196	188	190	188	0
0	0	0	0	0	0	0	0	0	0	0	0

Gambar 4.6: Hasil *padding*

Gambar 4.6. Lapis filter yang digunakan pada poin 1 adalah 16 lapis filter. Salah satu contoh filter yang digunakan untuk 3 *channel* warna pada matriks representasi citra diberikan pada Gambar 4.7. Selanjutnya setiap layer *channel* warna dikonvolusikan terhadap setiap kernel yang bersesuaian berukuran 3×3. Berikut diberikan contoh perhitungan konvolusi untuk titik (0,0) pada matriks hasil konvolusi R, dengan kernel K dan matriks representasi citra I : $R(0,0) =$

$$(K(0,0,0) * I(0,0,0)) + (K(0,1,0) * I(0,1,0)) + ... + (K(2,2,0) * I(2,2,0)) +$$

$$(K(0,0,1) * I(0,0,1)) + (K(0,1,1) * I(0,1,1)) + ... + (K(2,2,1) * I(2,2,1)) +$$

$$(K(0,0,2) * I(0,0,2)) + (K(0,1,2) * I(0,1,2)) + ... + (K(2,2,2) * I(2,2,2)) +$$

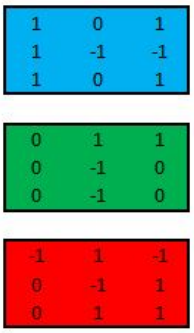
$$= (-1 * 0) + (1 * 0) + ... + (1 * 113) + (0 * 0) + (1 * 0) + ... + (1 * 103) +$$

$$(1 * 0) + (0 * 0) + ... + (1 * 104)$$

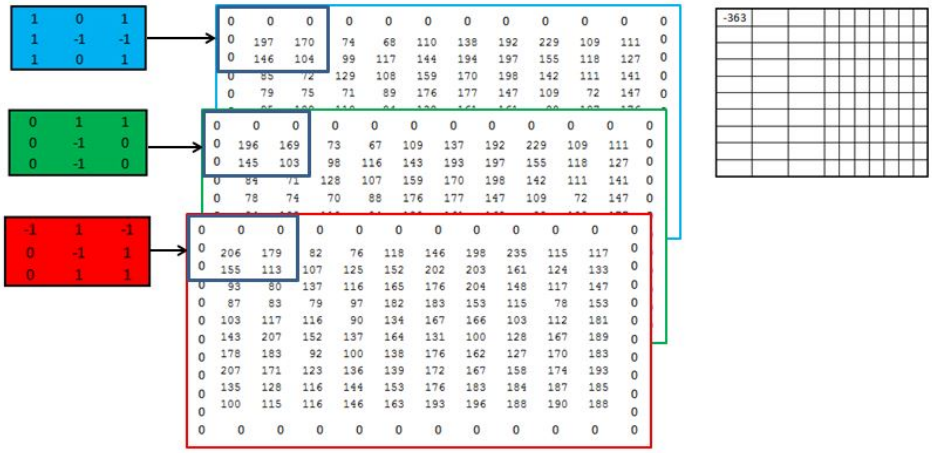
$$= (-263) + (-314) + 241 = -363$$

Ilustrasi perhitungan R(0,0) ditunjukkan pada Gambar 4.8, dan untuk R(0,1) ditunjukkan pada Gambar 4.9. Proses tersebut dilakukan terus menerus sampai seluruh nilai pada matriks representasi citra terisi dan menjadi satu layer. Proses tersebut dilakukan sebanyak 16 kali. Hasilnya yaitu ukuran sub-bagian citra menjadi berukuran $10 \times 10 \times 16$. Tahap berikutnya adalah menerapkan fungsi aktifasi *leaky relu* seperti pada persamaan 2.3 terhadap matriks hasil R. Hasil dari proses ini yaitu setiap elemen pada matriks bernilai lebih dari atau sama dengan nol dan ukuran matriks tetap sama.

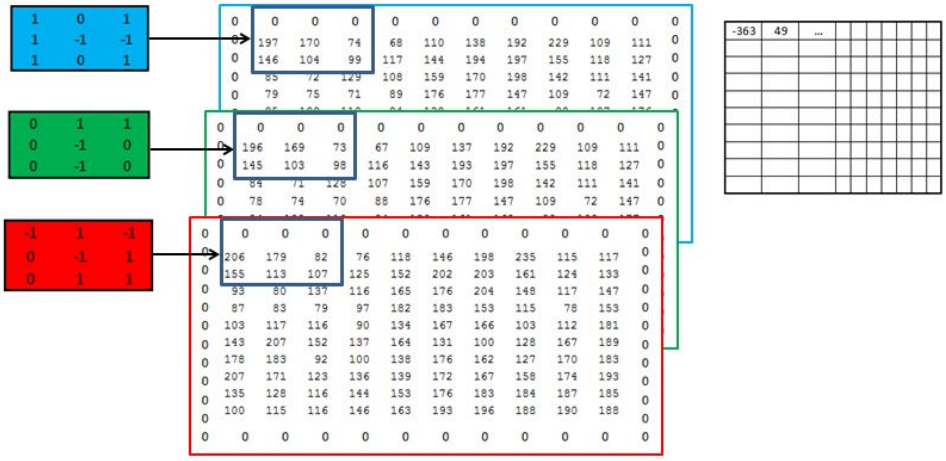
Selanjutnya melakukan *maxpooling* dengan ukuran 2×2, yaitu mengambil nilai terbesar dari setiap area berukuran 2×2 dengan perpindahan 2 langkah. Sehingga matriks hasil R menjadi berukuran $6 \times 6 \times 16$



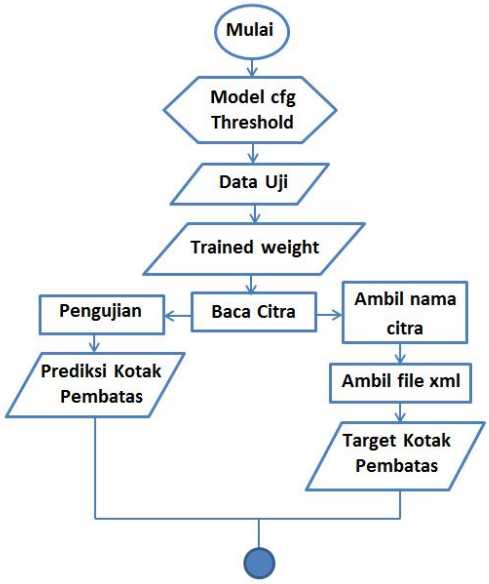
Gambar 4.7: Contoh kernel



Gambar 4.8: Hasil konvolusi kolom 1



Gambar 4.9: Hasil konvolusi kolom 2



Gambar 4.10: Diagram alur proses pengujian

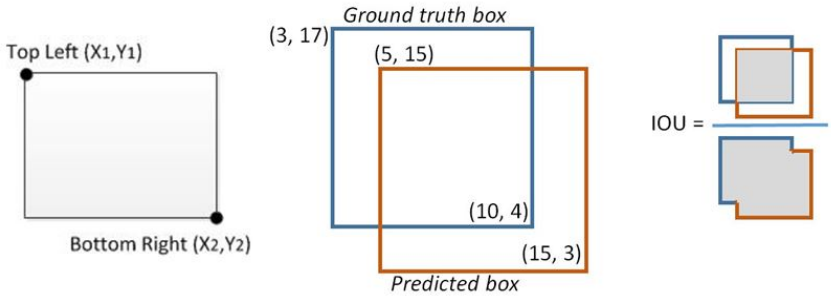
4.2.3 Pengujian YOLOv1-tiny

Hal-hal yang dibutuhkan dalam proses pengujian adalah dataset proses pengujian yang dijelaskan pada subbab 4.1, data anotasi untuk proses pengujian dan bobot baru hasil dari proses pelatihan serta model jaringan yang digunakan saat proses proses pelatihan. Rancangan pengerjaan proses pengujian diberikan pada Gambar 4.10

Dari Gambar 4.10 dapat dilihat bahwa proses pengujian dimulai degan menginisialisasi model jaringan dan *threshold*. Dalam penelitian ini, model jaringan yang digunakan adalah YOLOv1-tiny, sedangkan nilai *threshold* yang digunakan beberapa diantaranya yaitu 0.1 dan 0.05. Pemilihan nilai tersebut berdasakan pada hasil uji yang akan dilakukan pada bab selanjutnya.

Setelah itu, menginputkan data uji yaitu dataset pengujian, dan *trained weight*. Setiap citra pada dataset dibaca, lalu dilakukan pengujian citra tersebut dan menghasilkan prediksi kotak pembatas objek dan kelas objek. Disisi lain, setelah membaca citra, dilakukan pengambilan file data anotasi berformat .xml dan membaca informasi objek dari file tersebut. Kemudian menjadikannya sebagai nilai *output* yang seharusnya.

Seperti yang telah dijelaskan sebelumnya, terdapat 11 dataset dalam proses pengujian. Pada dataset dengan gangguan *motion blur*, proses pengujian dilakukan dengan menguji batas panjang *motion* agar model masih tetap dapat mendeteksi dan mengklasifikasi objek. Ukuran panjang motion yang



Gambar 4.11: Contoh kasus perhitungan IoU

digunakan untuk proses pengujian adalah $l = 3$, $l = 7$, $l = 15$

Untuk dataset dengan penambahan intensitas cahaya, proses pengujian yang dilakukan adalah dengan menambah intensitas warna $n = 30$, $n = 50$, dan $n = 80$.

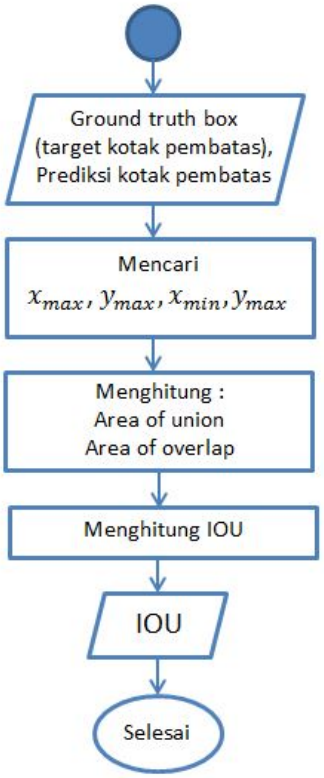
Sedangkan untuk dataset dengan pengurangan intensitas warna, nilai yang digunakan adalah -10, -30, -50, dan -100. Masing-masing dataset diuji keakuratannya dalam mengklasifikasi objek dan mendeteksi kotak pembatas objek.

4.2.4 Uji Keakuratan

Tingkat keakuratan pendeteksian kotak pembatas objek dihitung menggunakan IoU kotak pembatas dengan persamaan 2.4. IoU merupakan perhitungan untuk mendapatkan kesesuaian letak prediksi kotak pembatas dengan kotak target atau kotak *ground truth* dengan menghitung irisan dari kedua kotak tersebut. Diagram alir perhitungan kotak tersebut disajikan dalam Gambar 4.12. Sedangkan perhitungan keakuratan pengklasifikasian setiap kelas objek dilakukan dengan membagi jumlah objek yang terdeteksi tepat dengan total seluruh objek, seperti pada persamaan 3.1.

Berikut akan dijelaskan contoh perhitungan untuk mendapatkan nilai IoU prediksi kotak pembatas untuk contoh kasus yang ditunjukkan pada Gambar 4.11 :

1. Mendeteksi dan mengklasifikasi titik *topleft* dan *bottomright* setiap box:
   topleft ground truth $(x_{A1}, y_{A1}) = (3, 17)$
   bottomright ground truth $(x_{A2}, y_{A2}) = (10, 4)$
   topleft predicted box $(x_{B1}, y_{B1}) = (5, 15)$
   bottomright predicted box $(x_{B2}, y_{B2}) = (15,3)$



Gambar 4.12: Diagram alir perhitungan IoU

2. Mencari titik maksimal dan minimum :
   Topleft $(x_{max}, y_{min}) = (\max(x_{A1}, x_{B1}), \min(y_{A1}, y_{B1})) = (5, 15)$
   Bottomright $(x_{min}, y_{max}) = (\min(x_{A2}, x_{B2}), \max(y_{A2}, y_{B2})) = (10, 4)$

3. Mencari luas area of overlap :
   Luas area of overlap $= (x_{min} - x_{max}) * (y_{min} - y_{max}) = 5 * 11 = 55$

4. Mencari area of union :
   Luas ground truth $= (x_{B2} - x_{B1}) * (y_{B1} - y_{B2}) = 7 * 13 = 91$
   Luas predicted box $= (x_{B2} - x_{B1}) * (y_{B1} - y_{B2}) = 10 * 12 = 120$
   Area of union = Luas ground truth + Luas predicted box - area of overlap
   = 156

5. Menghitung IoU
   $IoU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{55}{156} = 0.352$

4.3 Implementasi Program

Rancangan data dan proses pada subbab diatas diimplementasikan pada perangkat lunak untuk mengetahui hasil kinerja metode. Pada subbab ini

```
import os
import cv2
from lxml import etree
import xml.etree.cElementTree as ET


def write_xml(folder, img, objects, tl, br, savedir):
    if not os.path.isdir(savedir):
        os.mkdir(savedir)

    image = cv2.imread(img.path)
    height, width, depth = image.shape

    annotation = ET.Element('annotation')
    ET.SubElement(annotation, 'folder').text = folder
    ET.SubElement(annotation, 'filename').text = img.name
    ET.SubElement(annotation, 'segmented').text = '0'
    size = ET.SubElement(annotation, 'size')
    ET.SubElement(size, 'width').text = str(width)
    ET.SubElement(size, 'height').text = str(height)
    ET.SubElement(size, 'depth').text = str(depth)
    for obj, topl, botr in zip(objects, tl, br):
        ob = ET.SubElement(annotation, 'object')
        ET.SubElement(ob, 'name').text = obj
        bbox = ET.SubElement(ob, 'bndbox')
        ET.SubElement(bbox, 'xmin').text = str(topl[0])
        ET.SubElement(bbox, 'ymin').text = str(topl[1])
        ET.SubElement(bbox, 'xmax').text = str(botr[0])
        ET.SubElement(bbox, 'ymax').text = str(botr[1])

    xml_str = ET.tostring(annotation)
    root = etree.fromstring(xml_str)
    xml_str = etree.tostring(root, pretty_print=True)
    save_path = os.path.join(savedir, img.name.replace('jpg', 'xml'))
    with open(save_path, 'wb') as temp_xml:
        temp_xml.write(xml_str)
```

Gambar 4.13: *Source code* menyimpan data anotasi

akan disajikan *code* dan penjelasan dari pengerjaan anotasi data, pelatihan, pengujian dan uji keakuratan.

4.3.1 Anotasi data

Output metode YOLO untuk setiap citra adalah citra dengan kotak pembatas dan kelas pada setiap objek didalam citra. Berdasarkan hal tersebut, diperlukan adanya proses anotasi data untuk menyimpan informasi letak kotak pembatas dan kelas objek. Informasi tersebut akan digunakan untuk membentuk *ground truth box* dan target kelas objek.

Terdapat dua tahap dalam proses anotasi data. Tahap pertama adalah merancang penyimpanan informasi kotak pembatas dan kelas. Informasi tersebut disimpan pada file dengan format .xml. Informasi penting yang diambil dan disimpan adalah ukuran citra, titik kotak pembatas, dan kelas objek. *Source code* tahap ini disajikan pada Gambar 4.13.

Tahap berikutnya adalah menggambar kotak pembatas setap objek pada

citra. Tahap ini digunakan untuk mendefinisikan letak dan kelas objek sebagai informasi objek. Nantinya informasi tersebut akan dikirimkan ke tahap penyimpanan informasi yang telah disiapkan pada tahap pertama. *Source code* tahap ini disajikan pada Gambar 4.14.

Proses kerja tahap anotasi data adalah memunculkan citra dalam folder, menggambar kotak pembatas pada setiap objek, lalu menggunakan fungsi `get_anotattion` untuk menyimpan informasi titik *top left* (tl) sebagai titik awal menggambar kotak pembatas dan *bottom right* (br) sebagai titik akhir menggambar kotak pembatas. Titik tersebut disimpan dalam list `tl_list` dan `br_list` lalu secara otomatis menyimpan nama objek kedalam `obj_list`. Setelah itu menggunakan fungsi `close_and_write` untuk menyimpan informasi tersebut dalam file dengan format .xml.

Hasil penyimpanan file xml dapat dilihat pada Gambar 4.15. Isi file data anotasi tersebut memberikan informasi detail tentang setiap objek pada citra. Informasi tersebut adalah ukuran citra masukan dan disimpan pada elemen `<size>`. Informasi lain yaitu kelas dan letak koordinat kotak pembatas (*bounding box*) setiap objek yang berturut-turut disimpan pada sub elemen `<name>` dan `<bndbox>` dalam setiap elemen `<object>`. Dalam contoh data anotasi tersebut terdapat dua jenis objek kerusakan jalan, yaitu *pothole* dan *distress*, sehingga muncul dua elemen `<object>`.

4.3.2 Pelatihan

Langkah-langkah yang dilakukan pada saat proses pelatihan pada penelitian ini adalah sebagai berikut :

1. Menyiapkan data-data yang dibutuhkan seperti yang dijelaskan pada subbab 4.1

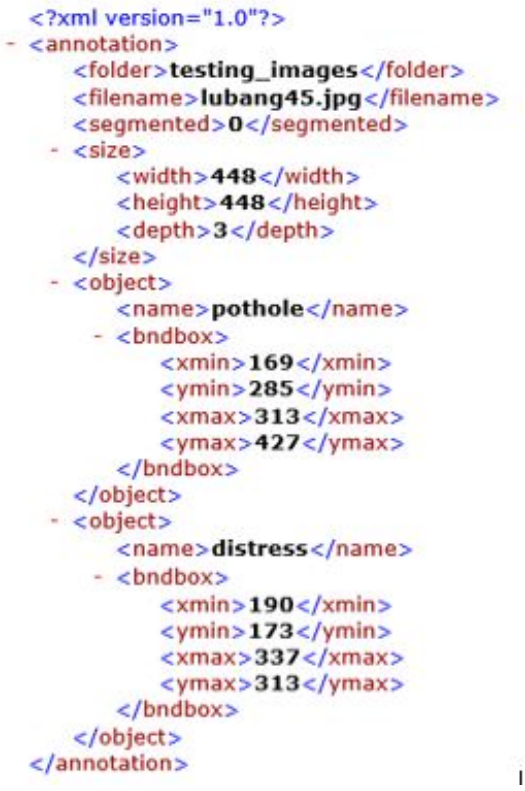
2. *Clone open source* YOLOv1-tiny untuk *custom dataset* pada `https://github.com/thtrieu/darkflow` dan *install*. Hasil dari proses ini adalah munculnya folder *darkflow* yang berisi tentang hal-hal yang dibutuhkan dalam proses pelatihan.

3. Mengubah file `labels.txt` dalam folder *darkflow* menjadi nama kelas deteksi dan klasifikasi, yaitu `pothole crack distress`. *Pothole* untuk kelas lubang, *crack* untuk kelas retak garis, dan *distress* untuk jenis keretakaan *non*-garis.

4. Mengubah file model jaringan YOLOv1-tiny dengan format .cfg sesuai yang disajikan pada subbab 4.2.2

```python
import os
import matplotlib.pyplot as plt
import cv2
from matplotlib.widgets import RectangleSelector
from coba_annotation2 import write_xml

img = None
tl_list = []
br_list = []
object_list = []
image = []
image_folder = 'foto'
savedir = 'hasil'
obj = 'distress'

def get_anotation(clk, rls):
    global tl_list
    global br_list
    global object_list
    tl_list.append((int(clk.xdata), int(clk.ydata)))
    br_list.append((int(rls.xdata), int(rls.ydata)))
    object_list.append(obj)

def close_and_write(event):
    global object_list
    global tl_list
    global br_list
    global img
    if event.key == 'x':
        write_xml(image_folder, img, object_list, tl_list, br_list, savedir)
        tl_list = []
        br_list = []
        object_list = []
        img = None
        plt.close()

def toggle_selector(event):
    toggle_selector.RS.set_active(True)

if __name__ == '__main__':
    for n, image_file in enumerate(os.scandir(image_folder)):
        img = image_file
        fig, ax = plt.subplots(1)
        image = cv2.imread(image_file.path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        ax.imshow(image)
        toggle_selector.RS = RectangleSelector(
            ax, get_anotation,
            drawtype='box', useblit=True,
            button=[1], minspanx=5, minspany=5,
            spancoords='pixels', interactive=True
        )
        bbox = plt.connect('key_press_event', toggle_selector)
        key = plt.connect('key_press_event', close_and_write)
        plt.show()
```

Gambar 4.14: *Source code* menggambar kotak pembatas



Gambar 4.15: Contoh hasil data anotasi

5. Melakukan proses pelatihan untuk mendapatkan bobot baru sesuai dengan dataset yang dimiliki dengan mengetik `python flow --model (letak file model) --load (letak file bobot) --train --annotation --(letak folder annotasi) --dataset (letak folder citra latih) --epoch (banyak epoch)` pada *command promt* dalam folder *darkflow*, sehingga untuk penelitian ini, code yang diketik adalah `python flow --model cfg/tiny-yolo-voc-3c.cfg --load bin/tiny-yolo-voc.weights --train --annotation train/annotations --dataset train/images --epoch 300`

Pada penelitian ini, proses pelatihan dengan epoch 300 dan kelas prediksi sebanyak 3 membutuhkan waktu *running* selama 4 hari dengan menggunakan CPU. Sedangkan proses *running* pelatihan dengan epoch 100 dan kelas prediksi sebanyak 3 kelas membutuhkan waktu selama 2 hari. Hasil dari proses pelatihan adalah munculnya folder baru dalam folder *darkflow* yang berisi bobot berformat .weight yang telah dilatih dengan dataset penelitian ini.

4.3.3 Pengujian

Dataset pengujian yang telah terkumpul kemudian dilakukan proses pengujian dengan menggunakan bobot yang telah dilatih (*trained weight*). Beberapa nilai *threshold* diuji pada seluruh dataset pengujian. Langkah-langkah pengerjaan pengujian secara singkat telah dijelaskan pada diagram alur Gambar 4.10. Implementasi dalam program akan disajikan dibagian ini. Gambar 4.16 merupakan *code* untuk proses pengujian.

Proses pengujian dilakukan dengan mempersiapkan model jaringan, menentukan nilai *threshold* dan memberikan bobot yang akan digunakan. Setelah itu dilakukan pembangkitan dan penyesuaian model dan bobot dengan perintah `TFNet()`. Lalu setiap data diuji dengan menggunakan perintah `tfnet.return_predict()` dan menyimpan hasilnya dalam list `obj_pred`. List tersebut berisi informasi prediksi titik kotak pembatas dan kelas kerusakan. Selanjutnya menampilkan citra bersama dengan prediksi kotak pembatas dari informasi `obj_pred`. Disisi lain juga dilakukan pemanggilan file data anotasi sesuai dengan nama data yang sedang diprediksi dengan menggunakan fungsi `read_xml` seperti pada Gambar 4.17 dan menyimpan informasinya kedalam `xml_target`. Setelah itu menghitung keakuratan antara `xml_target` dan `obj_pred`.

4.3.4 Implementasi Uji Keakuratan

Uji keakuratan klasifikasi kelas objek dilakukan dengan menggunakan persamaan 3.1. Sedangkan uji keakuratan deteksi kotak pembatas dilakukan dengan menghitung nilai IoU. Langkah-langkah pengerjaan secara manual dan diagram alur dari IoU telah dijelaskan pada subbab 4.2.4. Pada bagian ini ditunjukkan implementasi dari perhitungan IoU pada program, yaitu pada Gambar 4.18 sesuai dengan perhitungan persamaan 2.7. Hasil proses pengujian, akan menghasilkan nilai prediksi dan nilai target. Kedua nilai tersebut diproses untuk mencari nilai *intersection* kotak pembatas.

```
# model initialization
options = {"model": "cfg/tiny-yolo-voc-3c.cfg",
           "load": 5375,
           "threshold": 0.05}
tfnet = TFNet(options)

for n, image_file in enumerate(os.scandir(image_folder)):
    image = cv2.imread(image_file.path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # find and show the target
    img = image_file
    name = img.name.replace('jpg', 'xml')
    xml_target = read_xml(name)
    print('target_xml : ' , xml_target)
    # predict image
    result = tfnet.return_predict(image)
    # show the prediction in image
    obj_pred = []
    for key in range (len(result)):
        tl = (result[key]['topleft']['x'], result[key]['topleft']['y'])
        br = (result[key]['bottomright']['x'], result[key]['bottomright']['y'])
        label = result[key]['label']
        confidence = result[key]['confidence']
        obj_temp = (result[key]['label'], tl, br, confidence)
        obj_pred.append(obj_temp)
    fix_pred = min_distance(xml_target, obj_pred)
    iou, iou_val = find_iou(xml_target, fix_pred)
    sum_iouVal = sum(iou_val)
    len_iouVal = len(iou_val)
    print('prediksi akhir : ' , iou, '\n')
    if iou == [('e', 0)]:
        cv2.imshow(image_file.path, image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    else:
        for n in range (len(iou)):
            point_x = iou[n][0][1][0]
            point_y = (iou[n][0][2][1]-iou[n][0][1][1])/2
            point_y = int(point_y) + iou[n][0][1][1]
            center = (point_x, point_y)
            text1 = iou[n][0][3]
            text1 = str(text1)
            img = cv2.putText(image, iou[n][0][0]+' - conf : '+text1, center,
                              cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 0, 0), 1)
            img = cv2.rectangle(image, iou[n][0][1], iou[n][0][2],
                                (0, 255, 0), 3)
    cv2.imshow(image_file.path, image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Gambar 4.16: *Source code* proses pengujian

```
def read_xml(string):
    xmin_list = []
    ymin_list = []
    xmax_list = []
    ymax_list = []
    object_list = []
    obj_target = []

    file_name = string
    file_name = 'testing_annotation\\'+string
#    print(file_name)
    full_file = os.path.abspath(file_name)
#    print(full_file)
    dos = ET.parse(full_file)
#    print(dos)

    n_object = dos.findall('object/name')
    for c in n_object:
        object_list.append(c.text)

    n_xmin = dos.findall('object/bndbox/xmin')
    for k in n_xmin:
        a=int(k.text)
        xmin_list.append(a)

    n_ymin = dos.findall('object/bndbox/ymin')
    for k in n_ymin:
        a=int(k.text)
        ymin_list.append(a)

    n_xmax = dos.findall('object/bndbox/xmax')
    for k in n_xmax:
        a=int(k.text)
        xmax_list.append(a)

    n_xmin = dos.findall('object/bndbox/xmin')
    for k in n_xmin:
        a=int(k.text)
        xmin_list.append(a)

    n_ymin = dos.findall('object/bndbox/ymin')
    for k in n_ymin:
        a=int(k.text)
        ymin_list.append(a)

    n_xmax = dos.findall('object/bndbox/xmax')
    for k in n_xmax:
        a=int(k.text)
        xmax_list.append(a)

    n_ymax = dos.findall('object/bndbox/ymax')
    for k in n_ymax:
        a=int(k.text)
        ymax_list.append(a)
```

Gambar 4.17: *Source code* membaca file xml

```
prediksi = ['kelas_prediksi', (xp_1,yp_1), (xp_2, yp_2), conf_p]
target = ['kelas_target', (xt_1,yt_1), (xt_2, yt_2), conf_t]
##mencari titik intersection
x_max = max(prediksi[1][0], target[1][0])
y_max = max(prediksi[1][1], target[1][1])
x_min = min(prediksi[2][0], target[2][0])
y_min = min(prediksi[2][1], target[2][1])
##mencari luas intersection
interArea = max(0, x_min - x_max + 1) * max(0, y_min - y_max + 1)
##mencari luas kotak pembatas
boxArea_prediksi = (prediksi[2][0] - prediksi[1][0] + 1) *
                                (prediksi[2][1] - prediksi[1][1] + 1)
boxArea_target = (target[2][0] - target[1][0] + 1) *
                                (target[2][1] - target[1][1] + 1)
##mencari nilai IOU
iou = interArea / float(boxArea_prediksi + boxArea_target - interarea)
```

Gambar 4.18: *Source code* mencari IoU

BAB 5
UJI COBA DAN PEMBAHASAN

Pada bab ini akan diberikan beberapa pembahasan tentang pengujian metode yang digunakan, yaitu YOLOv1-tiny. Beberapa pembahasan pada bab ini antara lain dataset pengujian, pengujian nilai *threshold*, pembahasan hasil *output* pengujian dan pengukuran kinerja metode.

Pada bagian dataset pengujian akan dijelaskan hasil uji coba citra terhadap tingkat gangguan yang diberikan. Pada bagian nilai pengujian *threshold* akan dijelaskan hasil uji coba pemilihan nilai *threshold* terhadap munculnya kotak pembatas. Bagian *output* pengujian akan ditunjukkan citra hasil uji coba keluaran proses pengujian terhadap berbagai dataset. Sedangkan untuk bab kinerja metode YOLO akan menjelaskan hasil rata-rata IoU kotak pembatas dan nilai akurasi pengklasifikasian metode terhadap dataset. Pengukuran kinerja metode dipengaruhi oleh banyaknya epoch, *confidence threshold* dan kualitas citra uji.

5.1 Dataset Pengujian

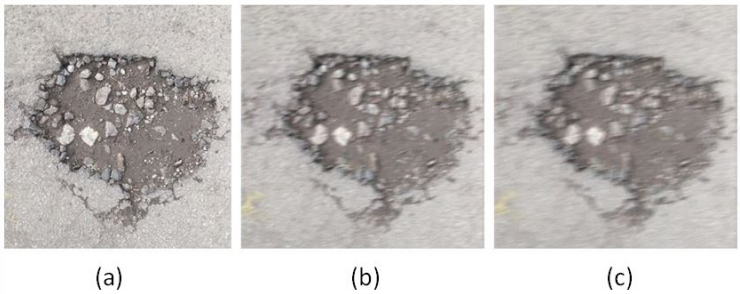
Sebelum melewati proses pengujian, dataset pengujian perlu disiapkan terlebih dahulu. Dataset yang digunakan yaitu dataset tanpa gangguan dan dataset dengan gangguan. Dataset dengan gangguan dibentuk untuk menguji sejauh mana model dapat mengenali objek walaupun dengan kondisi pengambilan citra yang kurang baik. Beberapa gangguan tersebut adalah *motion blur*, penambahan intesitas cahaya citra dan pengurangan intensitas cahaya citra.

Pada masing-masing dataset, terdapat citra dengan gangguan alami. Contohnya adalah adanya gangguan dari kendaraan, cat jalan yang mencolok, kaki manusia dan bayangan benda. Contoh citra dengan gangguan alami diperlihatkan pada Gambar 5.1. Dataset ini digunakan untuk menguji keakuratan model dalam mengenali objek sesungguhnya.

5.1.1 *Motion Blur*

Dalam kasus gangguan *motion blur*, terdapat tiga tingkat *blur* dengan panjang *motion* sebesar $l = 3$, $l = 7$ dan $l = 15$ berturut-turut disajikan pada Gambar 5.2(a), Gambar 5.2(b) dan Gambar 5.2(c). Tujuan dari proses

Gambar 5.1: Citra dengan noise alami



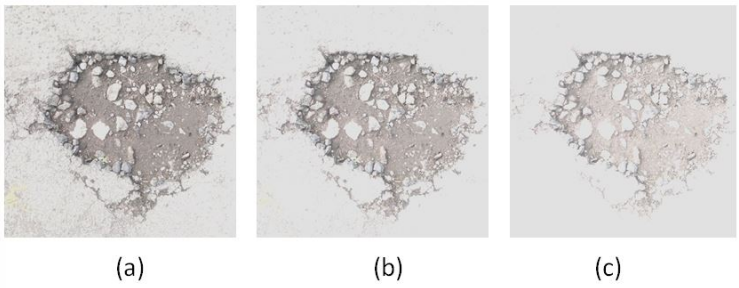
Gambar 5.2: *Motion blurring*

ini adalah untuk mengantisipasi data dengan pengambilan gambar yang tidak fokus. Hasil yang didapatkan adalah semakin besar panjang *motion* maka semakin kabur citra yang dihasikan. Hal tersebut membuat objek pada citra semakin sulit dikenali. Dalam proses pengujian untuk dataset ini, dapat diketahui pula batas kabur suatu citra agar objek tetap dapat terdeteksi dan terklasifikasi.

5.1.2 Penambahan *Brightness*

Gangguan lain yang diberikan pada citra untuk proses pengujian adalah *brightness adjustment*. Tujuan dari proses *brightness adjustment* adalah untuk mengantisipasi pengambilan gambar dengan keadaan cahaya saat terang dan gelap. Penambahan nilai intensitas warna citra sebesar $n = 30$, $n = 50$, dan $n = 80$ secara berurut disajikan dalam Gambar 5.3(a), Gambar 5.3(b), dan Gambar 5.3(c). Dataset ini digunakan untuk menguji keakuratan model dengan keadaan perncahayaan yang terlalu terang.

Dapat dilihat pada gambar tersebut, semakin terang cahaya maka semakin



Gambar 5.3: Efek Cerah

hilang objek dalam citra tersebut. Hal ini mengakibatkan model bisa saja tidak membaca adanya objek pada citra tersebut. Dalam proses pengujian untuk dataset ini, dapat diketahui bagaimana batas kondisi tingkat kecerahan dalam pengambilan citra agar model tetap dapat mengenali objek.

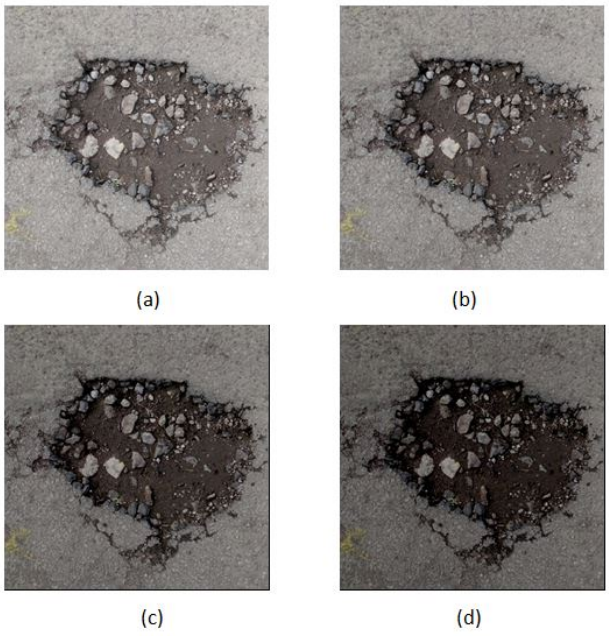
5.1.3 Pengurangan *Brightness*

Sedangkan untuk memberikan efek gelap dilakukan dengan mengurangi nilai intesitas warna citra sebesar $n = 30$, $n = 50$, $n = 80$, $n = 100$ dan secara berurut disajikan dalam Gambar 5.4(a), Gambar 5.4(b), Gambar 5.4(c), Gambar 5.4(d). Dapat dilihat pada gambar tersebut, objek pada citra masih terlihat walaupun telah diberi pengurangan intensitas warna paling besar yaitu $n = -100$. Tujuan dari pengujian dataset ini juga untuk mengetahui bagaimana batas kondisi pencahayaan dalam pengambilan citra agar objek tetap dapat dikenali untuk kondisi pencahayaan yang gelap.

5.2 *Threshold*

Beberapa dataset yang dijelaskan pada bagian sebelumnya digunakan untuk proses pengujian. Hasil dari proses pengujian adalah munculnya prediksi kotak pembatas dan kelas pada objek dalam citra. Namun dalam proses pengujian perlu juga diperhatikan nilai *threshold* sebagai batas nilai kepercayaan kotak pembatas tentang termuatnya suatu objek dalam kotak pembatas. Jika nilai *threshold* pengujian terlalu rendah, maka bisa jadi satu objek dideteksi oleh beberapa kotak pembatas. Sehingga objek yang terdeteksi lebih dari banyak objek sesungguhnya. Contoh tampilan dari kasus tersebut diberikan pada Gambar 5.5 dengan nilai *threshold* 0.1.

Hasil prediksi citra tersebut adalah : `[('pothole', (77, 0), (295, 335), 0.5435962), ('pothole', (0, 235), (117, 433), 0.2738225), ('pothole', (6, 270), (88, 400), 0.3639227), ('pothole', (44, 371), (134, 439), 0.23115823)]`. Hasil tersebut disebutkan secara urut

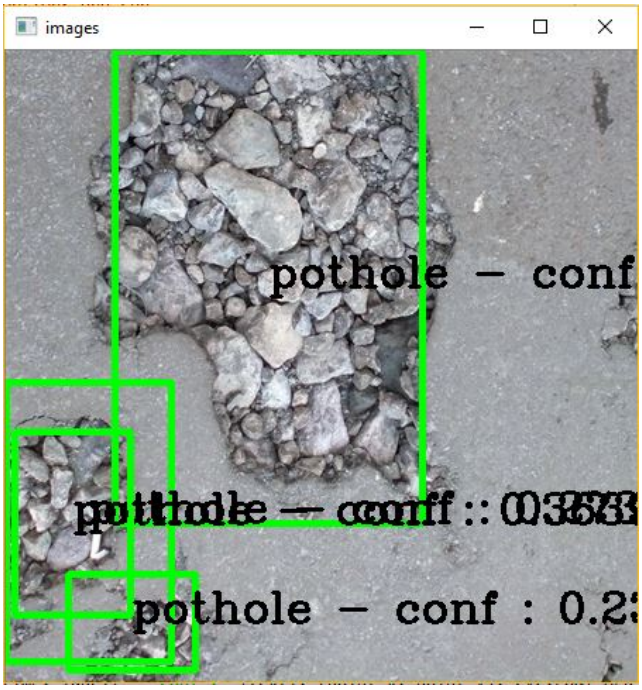


Gambar 5.4: Efek Gelap

yaitu kelas kerusakan, titik *top left* kotak pembatas, titik *bottom right* kotak pembatas, dan tingkat kepercayaan kotak pembatas. Pada gambar tersebut, hasil pengujian menunjukan terdapat 4 kotak pembatas objek lubang yang terdeteksi. Namun, nyatanya hanya terdapat 3 lubang pada citra. Hal tersebut terjadi karena beberapa kotak pembatas memiliki nilai keyakinan memuat objek yang lebih tinggi dari nilai *threshold* yang ditentukan. Padahal kotak pembatas tersebut merujuk pada objek yang sama.

Contoh lain yaitu kasus objek yang tidak terdeteksi. Contoh tampilan pada kasus ini dapat dilihat pada Gambar 5.6 dengan hasil prediksi : `[('pothole', (97, 248), (216, 408), 0.5073962)]`. Hasil menunjukkan bahwa objek yang terdapat pada citra adalah satu objek saja, yaitu lubang. Namun dapat terlihat pada gambar bahwa terdapat kerusakan jenis lain, yaitu retak *alligator*, yang termasuk dalam kelas *distress* dan tidak terdeteksi oleh kotak pembatas.

Contoh kasus ini merupakan suatu kasus dimana nilai *threshold* yang diberikan terlalu tinggi. Sehingga kotak pembatas dengan nilai kepercayaan yang lebih rendah dari nilai *threshold* tidak dapat menunjukkan objek. Hal ini menyebabkan banyaknya objek yang terdeteksi kurang dari banyaknya objek sesungguhnya. Adanya kotak pembatas yang memiliki nilai kepercayaan yang rendah karena kurangnya variasi data pelatihan pada objek tersebut atau



Gambar 5.5: Kotak pembatas yang ganda

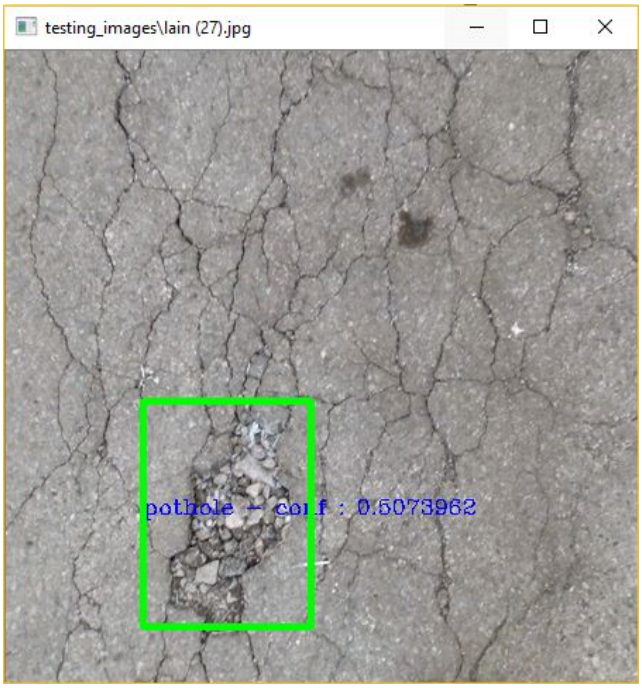
adanya gangguan pada objek. Pada contoh gambar tersebut nilai *threshold* yang digunakan adalah 0.4.

Sedangkan untuk contoh citra dengan pendeteksian yang tepat ditunjukkan pada Gambar 5.7. Hasil prediksi citra tersebut adalah : `[('distress', (40, 132), (408, 320), 0.6378045), ('distress', (117, 7), (401, 113), 0.5423641), ('distress', (160, 375), (266, 446), 0.46731982)]`.

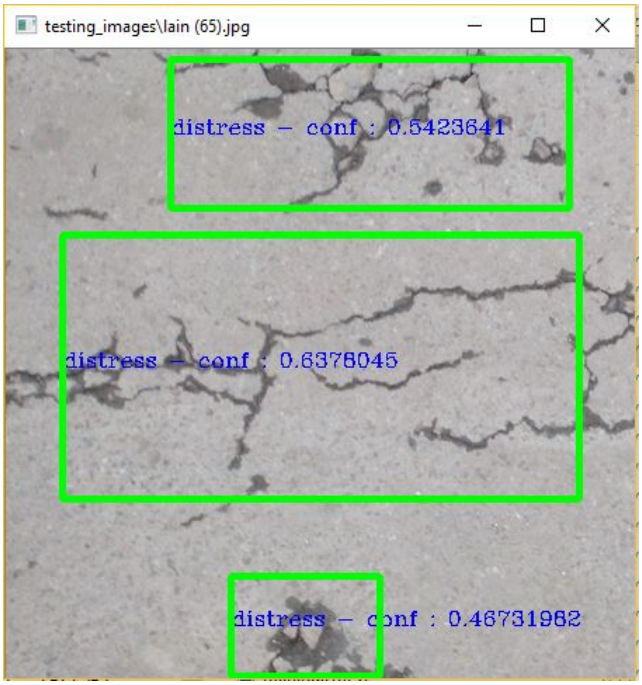
Nilai *threshold* yang optimal menghasilkan citra dengan objek yang terdeteksi sesuai dengan banyak objek sesungguhnya. Pada contoh gambar tersebut nilai *threshold* yang digunakan adalah 0.2.

Berdasarkan beberapa contoh kasus diatas, maka diperlukan penyesuaian nilai *threshold* untuk suatu dataset agar objek tetap dikenali. Contoh yang ditunjukkan pada Gambar 5.5, Gambar 5.6, dan Gambar 5.7 adalah pengujian pada satu citra dan bukan pengujian terhadap suatu dataset. Karena dalam suatu dataset, nilai kepercayaan kotak pembatas mempunyai nilai yang lebih beragam. Sehingga *threshold* untuk suatu citra bisa berbeda dengan *threshold* suatu dataset, walaupun citra tersebut termasuk dalam dataset.

Untuk mengatasi pemilihan nilai *threshold* yang optimal, penelitian ini memilih nilai *threshold* yang rendah namun hanya akan mengambil nilai kepercayaan kotak pembatas yang paling tinggi sebagai hasil deteksi.



Gambar 5.6: Kotak pembatas yang tidak muncul



Gambar 5.7: Kotak pembatas yang sesuai

Sehingga kemungkinan objek terdeteksi sempurna akan lebih besar jika menggunakan nilai *threshold* yang rendah.

5.3 *Output* Pengujian

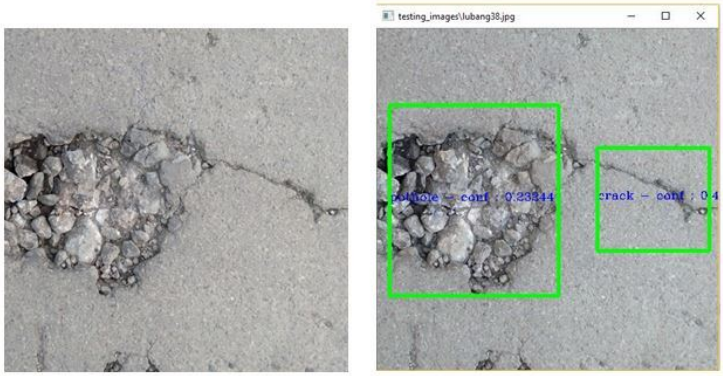
Setelah mengatur nilai *threshold*, proses pengujian dilakukan dengan menguji model menggunakan seluruh dataset pengujian. Sampel data pengujian yang dihasilkan seperti pada sub bab 5.1, digunakan untuk proses pengujian dengan nilai *threshold* sebesar 0.05.

Pada subbab ini diberikan contoh data masukan dan keluaran proses tersebut dari empat jenis dataset yang dimiliki serta nilai keakuratan dari kotak pembatas yang dihasilkan untuk data tersebut. Sedangkan nilai keakuratan setiap dataset akan dijelaskan pada subbab selanjutnya.

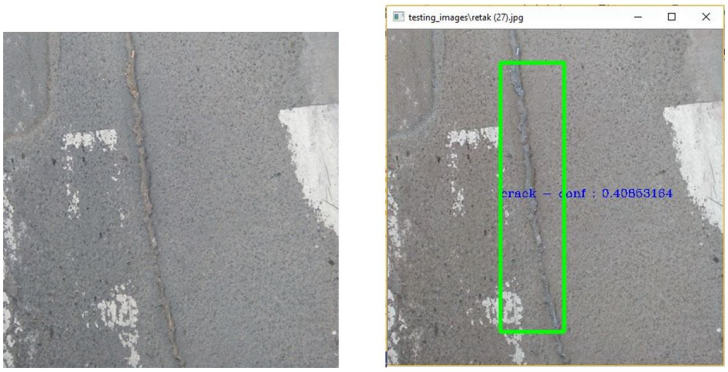
5.3.1 Citra tanpa gangguan

Gambar 5.8 merupakan pasangan *input* dan *output* data tanpa gangguan yang berhasil dikenali. Prediksi nilai kotak pembatas dan kelas yang dihasilkan adalah `hasil_prediksi : [(('pothole', (16, 100), (238, 350), 0.23244), 0.7975231649105177), (('crack', (289, 156), (436, 291), 0.41666457), 0.6404310907903331)]`. Setiap *item* pada *list* hasil tersebut secara urut merupakan prediksi jenis kelas, titik *top left*, titik *bottom right*, nilai *confidence* kotak pembatas dan nilai IoU. Dapat dilihat pada Gambar 5.8, bahwa seluruh objek berhasil dikenali dengan tepat dan memperoleh deteksi kotak pembatas yang sesuai. Hal ini ditunjukkan dengan hasil nilai IoU kotak pembatas yang cukup tinggi, yaitu sebesar 79% dan 64%.

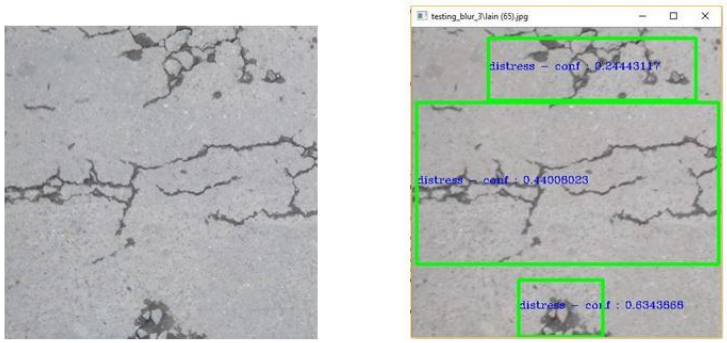
Sedangkan Gambar 5.9 merupakan pasangan *input* dan *output* data tanpa gangguan yang berhasil dikenali walaupun memiliki ganguan objek, yaitu adanya cat putih pada jalan. Prediksi nilai dan kelas yang dihasilkan adalah `hasil_prediksi : [(('crack', (150, 44), (235, 403), 0.40853164), 0.6327028273907599)]`. Gangguan objek pada citra tersebut sama sekali tidak terdeteksi dan terklasifikasi sebagai kerusakan jalan jenis manapun. Hal ini disebabkan karena dalam proses anotasi data, objek cat jalan tersebut tidak dilatih sebagai kerusakan jalan jenis manapun. Pendeteksian kotak pembatas juga memberikan letak yang cukup tepat dan memperoleh nilai IoU yang cukup tinggi yaitu 63%.



Gambar 5.8: Hasil pengujian citra tanpa gangguan



Gambar 5.9: Hasil penngujian citra dengan gangguan objek



Gambar 5.10: Hasil pengujian citra *motion blur* $l = 3$

5.3.2 Citra dengan *Motion Bluring*

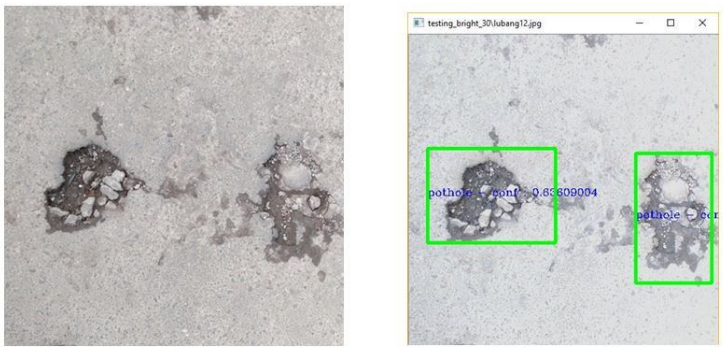
Gambar 5.10 merupakan pasangan *input* dan *output* data *motion blur* yang berhasil dikenali. Panjang *motion* yang digunakan pada gambar tersebut adalah $l = 3$. Prediksi nilai dan kelas yang dihasilkan adalah `hasil_prediksi : [(('distress', (45, 123), (405, 328), 0.5745862), 0.7343302932761088), (('distress', (130, 4), (387, 117), 0.6934038), 0.7048166786484543), (('distress', (164, 381), (258, 441), 0.23011842), 0.5432867266807125)]`. Dapat dilihat pada gambar dan hasil prediksi bahwa seluruh objek berhasil dikenali dengan tepat dan memperoleh nilai IoU yang cukup tinggi, yaitu sebesar 73%, 70% dan 54%.

Sedangkan Gambar 5.11 merupakan pasangan *input* dan *output* data *motion blur* yang tidak berhasil dikenali secara keseluruhan. Panjang *motion* yang digunakan pada gambar tersebut adalah $l = 15$. Prediksi nilai dan kelas yang dihasilkan adalah `hasil_prediksi : [('distress', (0, 17), (88, 91), 0.053324856), 0.8213579433091628]`. Dapat dilihat bahwa objek yang dapat dikenali hanya 1 dari 2 objek yang ada. Hal ini disebabkan karna dengan menggunakan panjang *motion* $l = 15$, tingkat kekaburan dari gambar tersebut terlalu tinggi. Sehingga pola garis yang merupakan ciri dari kelas retak, pada gambar tersebut menjadi melebar. Namun juga tidak menyerupai ciri pada kelas *pothole* dan *distress*. Maka dari itu, terdapat objek menjadi tidak dapat dikenali.

5.3.3 Citra dengn Penambahan *Brightness*

Sedangkan Gambar 5.12 merupakan pasangan *input* dan *output* data dengan penambahan inensitas cahaya $n = 30$. Prediksi nilai dan kelas yang dihasilkan adalah `[(('pothole', (27, 164), (212, 300),`

Gambar 5.11: Hasil pengujian citra *motion blur* $l = 15$



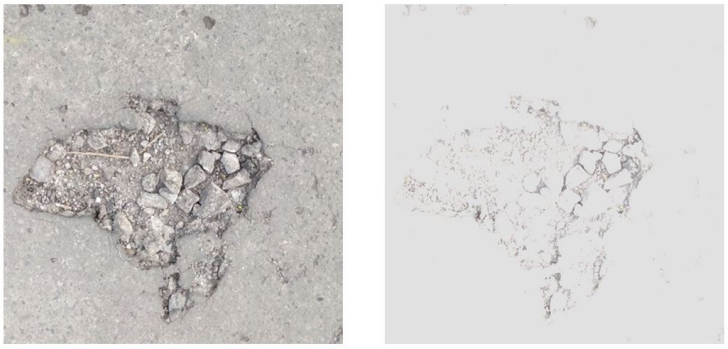
Gambar 5.12: Hasil pengujian citra penambahan kecarahan $n = 30$

`0.63609004), 0.7565340240169531), (('pothole', (328, 171), (438, 358), 0.15590923), 0.7598786828422877)]`. Dapat dilihat bahwa seluruh objek berhasil dikenali dengan tepat dan memperoleh nilai IoU sebesar 75% dan 75% walaupun diberikan penambahan intensitas cahaya sebesar $n = 30$.

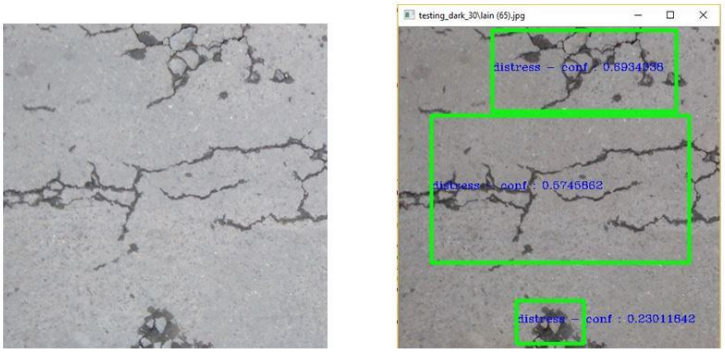
Sedangkan Gambar 5.13 merupakan pasangan *input* dan *output* data yang tidak dapat dikenali. Nilai pengambahan intensitas cahaya yang digunakan pada citra tersebut adalah $n = 80$. Dapat dilihat bahwa dengan penambahan intensitas cahaya sebesar itu, objek pada citra menjadi hilang dan tidak dapat dikenali.

5.3.4 Citra dengan Pengurangan *Brightness*

Gambar 5.14 merupakan pasangan *input* dan *output* data dengan pengurangan inensitas cahaya dan masih dapat dikenali. Nilai penambahan intensitas cahaya yang digunakan pada citra tersebut adalah $n = -30$. Prediksi nilai dan kelas yang dihasilkan adalah `[(('distress', (6, 108), (444, 342), 0.44008023), 0.7676963728080711), (('distress', (110, 15), (411, 105),`



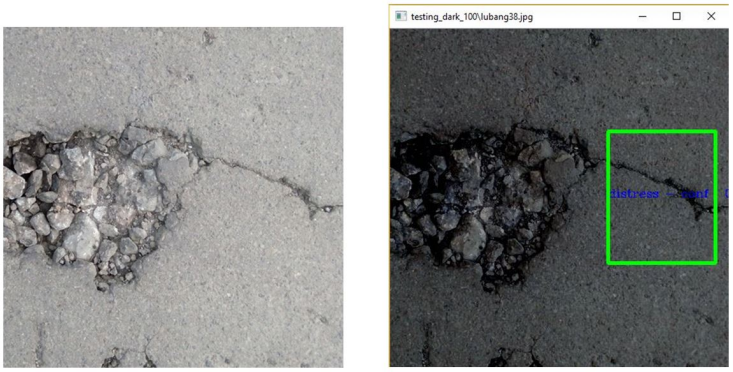
Gambar 5.13: Hasil pengujian citra penambahan kecerahan $n = 80$



Gambar 5.14: Hasil pengujian citra pengurangan kecerahan $n = -30$

`0.24443117), 0.6585669781931465), (('distress', (154, 365), (276, 447), 0.6343868), 0.7622971928920937)]`. Dapat dilihat bahwa seluruh objek masih dapat dideteksi dengan tepat dan memperoleh nilai IoU sebesar 76%, 65%, dan 76%.

Sedangkan Gambar 5.15 merupakan pasangan *input* dan *output* data dengan pengurangan inensitas cahaya namun tidak berhasil dikenali secara keseluruhan. Nilai pengambahan intensitas cahaya yang digunakan pada citra tersebut adalah $n = -100$. Prediksi nilai dan kelas yang dihasilkan adalah `hasil_prediksi : [(('distress', (288, 135), (430, 309), 0.09675159), 0)]`. Dapat dilihat bahwa terdapat dua objek pada citra, yaitu lubang (*pothole*) dan retak (*crack*). Namun hasil prediksi adalah objek *distress*. Karena klasifiakasi objek yang tidak tepat, sehingga nilai IoU yang dihasilkan adalah nol. Contoh tersebut menunjukkan bahwa efek gelap yang diberikan pada citra dapat mempengaruhi kelas objek.



Gambar 5.15: Hasil pengujian citra pengurangan kecerahan $n = -100$

5.4 Kinerja Metode YOLO

Kinerja Metode YOLO pada penelitian ini dipengaruhi oleh nilai *threshold* dan epoch. Kedua nilai tersebut diberikan kepada seluruh dataset pengujian. Kinerja metode ditentukan oleh nilai akurasi klasifikasi objek dan kesesuaian deteksi kotak pembatas. Total objek kelas lubang (*pothole*) sebanyak 40 objek, objek kelas retak garis (*crack*) sebanyak 30, sedangkan objek kelas retak *non*-garis sebanyak 30.

5.4.1 Dataset Tanpa Gangguan

Hasil pengujian nilai *threshold* dan epoch untuk dataset citra asli, yaitu citra tanpa gangguan diberikan pada Tabel 5.1 dan Tabel 5.2. Pada Tabel 5.1 menunjukkan hasil akurasi klasifikasi objek atau ketepatan objek masuk dalam kelasnya. Sedangkan Tabel 5.2 menunjukkan rata-rata nilai IoU kotak pembatas atau kesesuaian pendeteksian kotak pembatas pada objek.

Dapat dilihat pada Tabel 5.1, penggunaan nilai *threshold* yang tinggi memberikan nilai akurasi klasifikasi yang lebih rendah dibandingkan dengan penggunaan nilai *threshold* diatas 0.1. Penggunaan *threshold* sebesar 0.1 dan 0.05 lebih memberikan hasil akurasi klasifikasi objek yang tinggi, yaitu sebesar 99%. Tabel tersebut menunjukkan bahwa hasil klasifikasi objek lubang sama dengan jumlah total objek kelas lubang yang sebenarnya. Sehingga seluruh objek lubang terklasifikasi dengan tepat. Begitu juga dengan kelas *distress*. Namun berbeda dengan klasifikasi kelas retak (*crack*), tabel tersebut menunjukkan bahwa jumlah objek retak yang terklasifikasi paling banyak sebesar 29 objek. Hasil tersebut didapatkan dengan pengaturan nilai *threshold* sebesar 0.1 dan 0.05 dengan banyak epoch 300.

Seperti pada penjelasan pada subbab 5.2, bahwa penggunaan nilai *threshold* yang rendah dapat memberikan deteksi kotak pembatas yang hampir sesuai.

Tabel 5.1: Hasil Uji Coba Akurasi Dataset Tanpa Gangguan

Dataset	Epoch	Threshold	Objek terklasifikasi benar			Akurasi
			Pothole	Crack	Distress	
Citra Asli	100	0.1	40	28	30	99%
Citra Asli	100	0.05	40	28	30	99%
Citra Asli	300	0.3	40	22	29	91%
Citra Asli	300	0.2	40	28	29	97%
Citra Asli	300	0.1	40	29	30	99%
Citra Asli	300	0.05	40	29	30	99%

Dapat dilihat pada Tabel 5.2 bahwa nilai epoch 300 dan *threshold* sebesar 0.1 dan 0.05 menghasilkan nilai rata-rata deteksi kotak pembatas yang cukup tinggi, yaitu diatas 75%. Dapat dilihat pula bahwa semakin besar nilai *threshold* maka semakin kecil IoU kotak pembatas yang didapatkan, karena hanya kotak pembatas dengan kepercayaan yang tinggi yang akan keluar sebagai hasil. Disisi lain, banyaknya data yang digunakan saat proses pelatihan tergolong cukup rendah. Sehingga terdapat kotak pembatas dengan nilai keyakinan memuat objek yang rendah. Dari tabel tersebut didapatkan kecepatan deteksi dan klasifikasi paling tinggi yaitu 0.883s per citra untuk nilai *threshold* sebesar 0.1 dan 0.05.

Tabel 5.2: Rata-rata IoU Dataset Tanpa Gangguan

Dataset	Epoch	Threshold	Rata-rata IoU	Time(s)/img
Citra Asli	100	0.1	68.4%	0.850/img
Citra Asli	100	0.05	71.4%	0.883/img
Citra Asli	300	0.3	51.4%	0.882/img
Citra Asli	300	0.2	66.1%	0.882/img
Citra Asli	300	0.1	75.1%	0.883/img
Citra Asli	300	0.05	75.1%	0.883/img

5.4.2 Dataset gangguan *Motion Blur*

Tabel 5.3 dan Tabel 5.4 menunjukkan hasil uji dataset citra dengan gangguan *blur motion* dan epoch yang digunakan yaitu 300. Masing-masing dataset terdiri dari 60 citra dengan dengan panjang *motion* sebesar $l = 3$, $l = 7$, $l = 15$ dan *threshold* 0.1 dan 0.05.

Pada Tabel 5.3, dapat dilihat bahwa akurasi klasifikasi objek yang tinggi ditunjukkan pada citra dengan panjang *motion* $l = 3$ dan *threshold* rendah yaitu 0.05. Sedangkan *motion* dengan panjang lebih besar dari

itu menghasilkan nilai akurasi klasifikasi objek yang rendah. Nilai akurasi klasifikasi yang paling tinggi didapatkan sebesar 99%. Selaras dengan hasil akurasi klasifikasi citra tanpa gangguan, kelas *crack* tidak seluruhnya terklasifikasi dengan tepat. Semakin panjang *motion* yang digunakan, maka model semakin tidak dapat mengklasifikasi objek dengan tepat.

Pada Tabel 5.4 menujukkan bahwa penggunaan nilai *threshold* sebesar 0.05 dengan panjang *motion* $l = 3$ juga memberikan hasil yang baik. Rata-rata pendeteksian kotak pembatas yang dihasilkan paling tinggi sebesar 73.2% dengan kecepatan deteksi dan klasifikasi sebesar 0.912s setiap citra. Dapat dilihat pada kedua hasil tabel tersebut bahwa batas citra kabur yang dapat dikenali model adalah citra yang serupa dengan Gambar 5.2(a). Hal ini dapat disebabkan karena saat pengambilan data untuk proses pelatihan semaksimal mungkin difokuskan ke objek. Sehingga hanya citra dengan tingkat kabur yang rendah yang dapat dikenali model.

Tabel 5.3: Hasil Uji Coba Akurasi Dataset Gangguan *Motion Blur*

Dataset	Epoch	Threshold	Objek terklasifikasi benar			Akurasi
			Pothole	Crack	Distress	
Blur l = 3	300	0.05	40	29	30	99%
Blur l = 3	300	0.1	40	25	30	95%
Blur l = 7	300	0.05	9	6	45	45%
Blur l = 7	300	0.1	9	1	20	30%
Blur l = 15	300	0.05	0	0	10	10%
Blur l = 15	300	0.1	0	0	6	6%

Tabel 5.4: Rata-rata IoU Dataset Gangguan *Motion Blur*

Dataset	Epoch	Threshold	Rata-rata IoU	Time(s)/img
Blur l = 3	300	0.05	73.2%	0.912/img
Blur l = 3	300	0.1	70.1%	0.913/img
Blur l = 7	300	0.05	50.1%	1/img
Blur l = 7	300	0.1	50.1%	0.9/img
Blur l = 15	300	0.05	49.3%	0.95/img
Blur l = 15	300	0.1	48%	0.966/img

5.4.3 Dataset gangguan Penambahan Kecerahan

Tabel 5.5 dan Tabel 5.6 merupakan tabel hasil pengujian dataset citra dengan penambahan nilai intensitas cahaya citra, sehingga menghasilkan citra dengan pencahayaan yang terang. Dataset ini terdiri dari 60 citra.

Pada Tabel 5.5, dapat dilihat bahwa seluruh citra dapat dikenali dengan tepat pada dataset dengan penambahan intensitas cahaya sebesar 30 dan 50 dengan *threshold* 0.05 dan 0.1. Akurasi klasifikasi yang dihasilkan sebesar 100%. Sedangkan untuk penambahan intensitas cahaya yang lebih dari itu walaupun masih dapat menghasilkan ketepatan klasifikasi yang tinggi, namun masih terdapat kesalahan klasifikasi terutama pada kelas *crack*.

Hasil tersebut sejalan dengan hasil yang ditunjukkan pada Tabel 5.6. Diketahui bahwa dataset dengan penambahan intensitas warna sebesar 30 dan 50 dengan *threshold* 0.05 dan 0.1 menghasilkan nilai rata-rata pendeteksian kotak pembatas yang tinggi, yaitu diatas 70% dengan kecepatan deteksi dan klasifikasi kurang dari 0.95s setiap citra. Sedangkan dataset dengan nilai penambahan intensitas lebih dari itu menghasilkan nilai rata-rata pendeteksian kotak pembatas yang rendah.

Tabel 5.5: Hasil Uji Coba Akurasi Dataset Gangguan Penambahan Kecerahan

Dataset	Epoch	Threshold	Objek terklasifikasi benar			Akurasi
			Pothole	Crack	Distress	
Bright n = 30	300	0.05	40	30	30	100%
Bright n = 30	300	0.1	40	30	30	100%
Bright n = 50	300	0.05	40	30	30	100%
Bright n = 50	300	0.1	40	29	30	99%
Bright n = 80	300	0.05	40	28	29	97%
Bright n = 80	300	0.1	39	21	29	89%

Akurasi klasifikasi objek dan rata-rata pendeteksian kotak pembatas paling tinggi didapat saat nilai *threshold* yang digunakan sebesar 0.05 dengan penambahan nilai intensitas sebesar 30. Artinya citra dengan kondisi pencahayaan seperti pada Gambar 5.3(a) dan Gambar 5.3(b) masih dapat dideteksi dan diklasifikasi karena objek masih terlihat. Sedangkan deteksi kotak pembatas pada citra dengan $n = 80$ menghasilkan IoU yang rendah walaupun memiliki akurasi klasifikasi objek yang tinggi. Karena semakin besar penambahan intensitas cahaya, maka batas daerah objek juga semakin menghilang, bahkan objek juga akan semakin menghilang dan menghasilkan warna putih, baik objek kerusakan jalan maupun jalan. Hal tersebut menghasilkan nilai rata-rata pendeteksian kotak pembatas yang rendah. Maka, dapat dimengerti bahwa model tidak terlalu bagus untuk mendeteksi tingkat pencahayaan yang terlalu tinggi. Namun dataset dengan penambahan intensitas cahaya sebesar $n = 30$ menghasilkan tingkat akurasi klasifikasi objek

yang lebih tinggi dibandingkan dengan dataset tanpa gangguan. Ini berarti model dapat mengklasifikasi objek secara maksimal pada semua kelas objek jika kondisi kecerahan citra uji serupa dengan kondisi yang ditunjukkan pada Gambar 5.3(a).

Tabel 5.6: Rata-rata IoU Dataset Gangguan Penambahan Kecerahan

No	Dataset	Epoch	Threshold	Rata-rata IoU	Time(s)/img
1	Bright n = 30	300	0.05	75.3%	0.933/img
2	Bright n = 30	300	0.1	73.1%	0.883/img
3	Bright n = 50	300	0.05	70.8%	0.933/img
4	Bright n = 50	300	0.1	66.4%	0.883/img
5	Bright n = 80	300	0.05	56.2%	0.916/img
6	Bright n = 80	300	0.1	52.3%	0.9/img

5.4.4 Dataset gangguan Pengurangan Kecerahan

Pada Tabel 5.7 dan Tabel 5.8 menunjukkan hasil uji dataset citra dengan pengurangan tingkat kecerahan, sehingga citra menjadi gelap. Epoch yang digunakan sebanyak 300. Dataset ini terdiri dari 60 citra.

Pada Tabel 5.7 dapat dilihat bahwa nilai akurasi klasifikasi objek tinggi untuk seluruh tingkat pengurangan intensitas cahaya. Nilai akurasi paling tinggi yang dihasilkan adalah sebesar 99% dengan tingkat pengurangan intensitas $n = -30$, $n = -50$ dan $n = -80$ dengan *threshold* 0.1 dan 0.05.

Tabel 5.7: Hasil Uji Coba Akurasi Dataset Gangguan Pengurangan Kecerahan

Dataset	Epoch	Threshold	Objek terklasifikasi benar			
			Pothole	Crack	Distress	
Bright n = -30	300	0.05	40	29	30	99%
Bright n = -30	300	0.1	40	29	30	99%
Bright n = -50	300	0.05	40	29	30	99%
Bright n = -50	300	0.1	40	29	30	99%
Bright n = -80	300	0.05	40	29	30	99%
Bright n = -80	300	0.1	40	29	29	98%
Bright n = -100	300	0.05	36	28	30	94%
Bright n = -100	300	0.1	37	23	30	90%

Hal yang serupa juga ditunjukkan pada Tabel 5.8. Hasil rata-rata IoU yang tinggi ditunjukkan pada dataset dengan nilai penambahan intensitas sebesar -30 sampai -80 dan *threshold* 0.1 dan 0.05. Sedangkan dataset dengan nilai

penambahan intensitas lebih dari itu menghasilkan nilai rata-rata IoU yang rendah.

Tabel 5.8: Rata-rata IoU Dataset Gangguan Pengurangan Kecerahan

Dataset	Epoch	Threshold	Rata-rata IoU	Time(s)/img
Bright n = -30	300	0.05	74.7%	0.916/img
Bright n = -30	300	0.1	74.6%	0.883/img
Bright n = -50	300	0.05	74.5%	0.933/img
Bright n = -50	300	0.1	73.9%	0.883/img
Bright n = -80	300	0.05	73.8%	0.916/img
Bright n = -80	300	0.1	67%	1.05/img
Bright n = -100	300	0.05	66.2%	0.916/img
Bright n = -100	300	0.1	63.7%	1/img

Pada Tabel 5.8, terlihat bahwa model masih dapat mendeteksi objek saat nilai intensitas citra berkurang sebanyak 80. Nilai rata-rata IoU yang didapat diatas 73%. Sedangkan nilai paling tinggi yaitu 74.7%, didapatkan saat menggunakan nilai *threshold* 0.05. Walaupun intensitas cahaya citra berkurang dan akurasi klasifikasi objek serta rata-rata IoU menurun, namun nilai akurasi klasifikasi yang dihasilkan masih tergolong tinggi. Hal ini disebabkan karena walaupun citra yang dihasilkan menjadi gelap, namun objek masih terlihat dan tidak menghilang. Sehingga untuk citra dengan pencahayaan yang kurang, model masih dapat mengenali beberapa objek.

BAB 6
KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan yang dihasilkan berdasarkan penelitian yang telah dilaksanakan serta saran untuk pengembangan penelitian ini.

6.1 Kesimpulan

Pada penelitian ini, penulis telah berhasil membangun proses deteksi dan klasifikasi objek kerusakan jalan menggunakan metode YOLOv1-tiny. Penyelesaian proses ini menggunakan model dan *pre-trained weight* YOLO.

Berdasarkan pengujian yang dilakukan pada berbagai jenis dataset, baik dataset citra tanpa gangguan dan dataset citra dengan gangguan. Dataset tanpa gangguan menghasilkan nilai akurasi klasifikasi paling tinggi sebesar 99%, rata-rata deteksi kotak pembatas sebesar 75.1% dan kecepatan klasifikasi dan deteksi sebesar 0.883 detik tiap citra. Dataset dengan gangguan *motion blur* menghasilkan nilai akurasi klasifikasi paling tinggi sebesar 99%, rata-rata deteksi kotak pembatas sebesar 73.2% dan kecepatan klasifikasi dan deteksi sebesar 0.912 detik tiap citra. Dataset dengan gangguan efek cerah menghasilkan nilai akurasi klasifikasi paling tinggi sebesar 100%, rata-rata deteksi kotak pembatas sebesar 75.3% dan kecepatan klasifikasi dan deteksi sebesar 0.933 detik tiap citra. Sedangkan untuk dataset dengan gangguan efek gelap menghasilkan nilai akurasi klasifikasi paling tinggi sebesar 99%, sedangkan rata-rata deteksi kotak pembatas sebesar 74.7% dengan kecepatan klasifikasi dan deteksi sebesar 0.916 detik tiap citra.

6.2 Saran

Saran yang dapat diberikan untuk penelitian serupa adalah :

1. Memperbanyak jumlah data proses pelatihan khususnya data citra kelas retak garis dengan berbagai macam gangguan dan tingkatannya. Sehingga program dapat mendeteksi objek dalam berbagai macam kondisi pengambilan data

2. Menggunakan metode YOLO versi terbaru untuk mendapatkan akurasi yang lebih tinggi dan waktu pendeteksian yang cepat

DAFTAR PUSTAKA

Daniel, A. dan Preeja, V. (2014), 'Automatic road distress detection and analysis', *International Journal of Computer Applications* **101**(10).

Departemen-Pekerjaan-Umum (1995), *Manual Pemeliharaan Rutin untuk Jalan Nasional dan Jalan Provinsi, Jilid II:Metode Perbaikan Standar*, Departemen Pekerjaan Umum Direktorat Jendral Bina Marga Direktorat Bina Teknik.

Everingham, M., Van Gool, L., Williams, C. K., Winn, J. dan Zisserman, A. (2010), 'The pascal visual object classes (voc) challenge', *International journal of computer vision* **88**(2), 303–338.

Fan, Z., Wu, Y., Lu, J. dan Li, W. (2018), 'Automatic pavement crack detection based on structured prediction with the convolutional neural network', *arXiv preprint arXiv:1802.02208* .

Hansen, D. K. dan Nasrollahi, K. (2017), Real-time barcode detection and classification using deep learning, *in* 'International Joint Conference on Computational Intelligence'.

Liu, G. (2017), Real-time object detection for autonomous driving-based on deep learning, PhD thesis.

Mooij, G., Bagulho, I. dan Huisman, H. (2018), 'Automatic segmentation of prostate zones', *arXiv preprint arXiv:1806.07146* .

O'Shea, K. dan Nash, R. (2015), 'An introduction to convolutional neural networks', *arXiv preprint arXiv:1511.08458* .

Ouma, Y. O. dan Hahn, M. (2017), 'Pothole detection on asphalt pavements from 2d-colour pothole images using fuzzy c-means clustering and morphological reconstruction', *Automation in Construction* **83**, 196–211.

Putra, D. (2010), *Pengolahan citra digital*, Penerbit Andi.

Putra, W. S. E. (2016), 'Klasifikasi citra menggunakan convolutional neural network (cnn) pada caltech 101', *Jurnal Teknik ITS* **5**(1).

Radovic, M., Adarkwa, O. dan Wang, Q. (2017), 'Object recognition in aerial images using convolutional neural networks', *Journal of Imaging* **3**(2), 21.

Redmon, J., Divvala, S., Girshick, R. dan Farhadi, A. (2016), You only look once: Unified, real-time object detection, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 779–788.

Ryu, S.-K., Kim, T. dan Kim, Y.-R. (2015), 'Image-based pothole detection system for its service and road management system', *Mathematical Problems in Engineering* **2015**.

Shahin, M. Y. (1994), *Pavement management for airports, roads, and parking lots*.

Solomon, C. dan Breckon, T. (2011), *Fundamentals of Digital Image Processing: A practical approach with examples in Matlab*, John Wiley & Sons.

Suryadharma, H. dan Susanto, B. (1999), 'Rekayasa jalan raya', *Universitas Atma Jaya, Jakarta* .

Tedeschi, A. dan Benedetto, F. (2017), 'A real-time automatic pavement crack and pothole recognition system for mobile android-based devices', *Advanced Engineering Informatics* **32**, 11–25.

Wang, Q., Rasmussen, C. dan Song, C. (2016), Fast, deep detection and tracking of birds and nests, *in* 'International Symposium on Visual Computing', Springer, pp. 146–155.

Zhang, L., Yang, F., Zhang, Y. D. dan Zhu, Y. J. (2016), Road crack detection using deep convolutional neural network, *in* 'Image Processing (ICIP), 2016 IEEE International Conference on', IEEE, pp. 3708–3712.

Zitnick, C. L. dan Dollár, P. (2014), Edge boxes: Locating object proposals from edges, *in* 'European conference on computer vision', Springer, pp. 391–405.

LAMPIRAN

```
import os
import cv2
from lxml import etree
import xml.etree.cElementTree as ET

def write_xml(folder, img, objects, tl, br, savedir):
    if not os.path.isdir(savedir):
        os.mkdir(savedir)

    image = cv2.imread(img.path)
    height, width, depth = image.shape

    annotation = ET.Element('annotation')
    ET.SubElement(annotation, 'folder').text = folder
    ET.SubElement(annotation, 'filename').text = img.name
    ET.SubElement(annotation, 'segmented').text = '0'
    size = ET.SubElement(annotation, 'size')
    ET.SubElement(size, 'width').text = str(width)
    ET.SubElement(size, 'height').text = str(height)
    ET.SubElement(size, 'depth').text = str(depth)
    for obj, topl, botr in zip(objects, tl, br):
        ob = ET.SubElement(annotation, 'object')
        ET.SubElement(ob, 'name').text = obj
        bbox = ET.SubElement(ob, 'bndbox')
        ET.SubElement(bbox, 'xmin').text = str(topl[0])
        ET.SubElement(bbox, 'ymin').text = str(topl[1])
        ET.SubElement(bbox, 'xmax').text = str(botr[0])
        ET.SubElement(bbox, 'ymax').text = str(botr[1])

    xml_str = ET.tostring(annotation)
    root = etree.fromstring(xml_str)
    xml_str = etree.tostring(root, pretty_print=True)
    save_path = os.path.join(savedir, img.name.replace('jpg', 'xml'))
    with open(save_path, 'wb') as temp_xml:
        temp_xml.write(xml_str)
    return xml_str
```

```
import os
```

```
import matplotlib.pyplot as plt
import cv2
from matplotlib.widgets import RectangleSelector
from coba_annotation2 import write_xml

img = None
tl_list = []
br_list = []
object_list = []
image = []
image_folder = 'foto'
savedir = 'hasil'
obj = 'distress'

def get_anotation(clk, rls):
    global tl_list
    global br_list
    global object_list
    tl_list.append((int(clk.xdata), int(clk.ydata)))
    br_list.append((int(rls.xdata), int(rls.ydata)))
    object_list.append(obj)

def close_and_write(event):
    global object_list
    global tl_list
    global br_list
    global img
    if event.key == 'x':
        print(object_list)
        write_xml(image_folder, img, object_list, tl_list, br_list, savedir)
        tl_list = []
        br_list = []
        object_list = []
        img = None
        plt.close()

def toggle_selector(event):
    toggle_selector.RS.set_active(True)

if __name__ == '__main__':
    for n, image_file in enumerate(os.scandir(image_folder)):
        img = image_file
        fig, ax = plt.subplots(1)
        image = cv2.imread(image_file.path)
```

```
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    ax.imshow(image)
    toggle_selector.RS = RectangleSelector(
       ax, get_anotation,
       drawtype='box', useblit=True,
       button=[1], minspanx=5, minspany=5,
       spancoords='pixels', interactive=True
    )
    bbox = plt.connect('key_press_event', toggle_selector)
    key = plt.connect('key_press_event', close_and_write)
    plt.show()
```

```
import os
import cv2
from darkflow.net.build import TFNet
from read_xml import read_xml
from find_iou import find_iou
from eror import min_distance

# model initialization
options = {"model": "cfg/tiny-yolo-voc-3c.cfg",
        "load": 5375,
        "threshold": 0.05}
tfnet = TFNet(options)

image_folder = 'testing_images'
annotation_folder = 'testing_annotation'
img = None
image = []
eror = []
sum_eror = []
iou_sum = 0
iou_len = 0

for n, image_file in enumerate(os.scandir(image_folder)):
   image = cv2.imread(image_file.path)
   image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
   # find and show the target
   img = image_file
   name = img.name.replace('jpg', 'xml')
   xml_target = read_xml(name)
   # predict image
```

```
    result = tfnet.return_predict(image)
    obj_pred = []
    for key in range (len(result)):
        tl = (result[key]['topleft']['x'], result[key]['topleft']['y'])
        br = (result[key]['bottomright']['x'], result[key]['bottomright']['y'])
        label = result[key]['label']
        confidence = result[key]['confidence']
        obj_temp = (result[key]['label'], tl, br, confidence)
        obj_pred.append(obj_temp)
    fix_pred = min_distance(xml_target, obj_pred)
    iou, iou_val = find_iou(xml_target, fix_pred)
    sum_iouVal = sum(iou_val)
    len_iouVal = len(iou_val)
    print('prediksi akhir : ' , iou, '\n')
    if iou == [('e', 0)]:
        cv2.imshow(image_file.path, image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    else:
        for n in range (len(iou)):
            point_x = iou[n][0][1][0]
            point_y = (iou[n][0][2][1]-iou[n][0][1][1])/2
            point_y = int(point_y) + iou[n][0][1][1]
            center = (point_x, point_y)
            text = ("%.2f" % iou[n][1])
            text = str(text)
            text1 = iou[n][0][3]
            text1 = str(text1)
            img = cv2.putText(image, iou[n][0][0]+' - conf : '+text1, center,
cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 0, 0), 1)
            img = cv2.rectangle(image, iou[n][0][1], iou[n][0][2], (0, 255, 0), 3)
    cv2.imshow(image_file.path, image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    iou_sum = iou_sum + sum_iouVal
    iou_len = iou_len + len_iouVal
# average of accuracy
tingkat_akurat = iou_sum/iou_len
print ('tingkat akurasi = ', tingkat_akurat)
```

```
def find_iou (xml_target, obj_pred):
    new_list = []
```

```
    iou_val = []
    target = xml_target
    prediksi = obj_pred
    b = len(prediksi)
    if (len(prediksi)) == 0:
        new_list = 'eror'
    else:
        for i in range (len(target)):
            iou = []
            for k in range (b):
                if target[i][0] == prediksi[k][0]:
                    x_max = max(target[i][1][0], prediksi[k][1][0])
                    y_max = max(target[i][1][1], prediksi[k][1][1])
                    x_min = min(target[i][2][0], prediksi[k][2][0])
                    y_min = min(target[i][2][1], prediksi[k][2][1])
                    interArea = max(0, x_min - x_max + 1)*max(0, y_min - y_max + 1)
                    boxArea_target = (target[i][2][0] - target[i][1][0] +
                                      1) *(target[i][2][1] - target[i][1][1] + 1)
                    boxArea_prediksi = (prediksi[k][2][0] - prediksi[k][1][0] +
                                        1) * (prediksi[k][2][1] - prediksi[k][1][1] + 1)
                    iou_temp = interArea / float(boxArea_target + boxArea_prediksi - interArea)
                    iou.append(iou_temp)
                else:
                    iou_temp = 0
                    iou.append(iou_temp)
            max_iou = max(iou)
            a = iou.index(max_iou)
            list_temp = (prediksi[a], max_iou)
            if list_temp in new_list:
                continue
            iou_val.append(max_iou)
            new_list.append(list_temp)
    return new_list, iou_val
```

```
def min_distance (xml_target, obj_pred):

    import math
    new_list = []

    target = xml_target
    prediksi = obj_pred
```

```
b = len(prediksi)
if (len(prediksi)) == 0:
   new_list = 'eror'
else:
   for i in range (len(target)):
      dis = []
      for j in range (b):
         if target[i][0] == prediksi[j][0]:
            tl = math.sqrt((target[i][1][0] - prediksi[j][1][0])**2 +
                     (target[i][1][1] - prediksi[j][1][1])**2)
            br = math.sqrt((target[i][2][0] - prediksi[j][2][0])**2 +
                     (target[i][2][1] - prediksi[j][2][1])**2)
            dis_temp = (tl + br)/2
            dis.append(dis_temp)
         else:
            dis_temp = 1000
            dis.append(dis_temp)
      min_dis = min(dis)
      a = dis.index(min_dis)
      list_temp = prediksi[a]
      if list_temp in new_list:
         continue
      new_list.append(list_temp)
return(new_list)
```

BIODATA PENULIS

Penulis bernama Ravy Hayu Pramestya, lahir di Surabaya, 19 Januari 1994, merupakan anak pertama dari dua bersaudara. Penulis menempuh pendidikan formal di SDN Wedoro I, SMPN 2 Waru dan SMAN 1 Waru. Setelah lulus dari SMA penulis melanjutkan studi S1 Jurusan Matematika di Universitas Airlangga pada tahun 2011 - 2015. Penulis kemudian melanjutkan studi S2 Jurusan Matematika di Institut Teknologi Sepuluh Nopember pada tahun 2016 dengan Tesis pada bidang Ilmu Komputer.

Penulis dapat dihubungi melalui e-mail : *ravyhayyu@gmail.com.*