ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR – TM184835

PENDETEKSI *CRACK* PADA *PROPELLER* KAPAL DENGAN MENGGUNAKAN METODE *CONVOLUTIONAL NEURAL NETWORK*

RHESA BELA
NRP 02111840000083

Dosen Pembimbing
Mohammad Khoirul Effendi, S.T., M.Sc.Eng., Ph.D.
NIP. 198204142010121001

Program Studi Sarjana
Departemen Teknik Mesin
Fakultas Teknologi Industri dan Rekayasa Sistem
Institut Teknologi Sepuluh Nopember
Surabaya
2022

TUGAS AKHIR – TM184835

PENDETEKSI *CRACK* PADA *PROPELLER* KAPAL DENGAN MENGGUNAKAN METODE *CONVOLUTIONAL NEURAL NETWORK*

RHESA BELA
NRP 02111840000083

Dosen Pembimbing
Mohammad Khoirul Effendi, S.T., M.Sc.Eng., Ph.D.
NIP. 198204142010121001

Program Studi Sarjana
Departemen Teknik Mesin
Fakultas Teknologi Industri dan Rekayasa Sistem
Institut Teknologi Sepuluh Nopember
Surabaya
2022

FINAL PROJECT – TM184835

CRACK DETECTION ON SHIP'S PROPELLER USING CONVOLUTIONAL NEURAL NETWORK METHOD

RHESA BELA
NRP 02111840000083

Advisor
Mohammad Khoirul Effendi, S.T., M.Sc.Eng., Ph.D.
NIP 198204142010121001

Mechanical Engineering Undergraduate Study Program
Department of Mechanical Engineering
Faculty of Industrial Technology and Systems Engineering
Institut Teknologi Sepuluh Nopember
Surabaya
2022

HALAMAN PENGESAHAN

PENDETEKSI *CRACK* PADA *PROPELLER* KAPAL DENGAN MENGGUNAKAN METODE *CONVOLUTIONAL NEURAL NETWORK*

TUGAS AKHIR
Diajukan untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Teknik pada
Program Studi S-1 Teknik Mesin
Departemen Teknik Mesin
Fakultas Teknologi Industri dan Rekayasa Sistem
Institut Teknologi Sepuluh Nopember

Oleh: **RHESA BELA**
NRP. 02111840000083

Disetujui oleh Tim Penguji Tugas Akhir:

1. M. Khoirul Effendi, S.T., M.Sc.Eng., Ph.D. — Pembimbing
2. Ari Kurniawan Saputra, S.T., M.T. — Penguji
3. Dinny Harnany, S.T., M.Sc. — Penguji
4. Arif Wahjudi, S.T., M.T., Ph.D. — Penguji

SURABAYA
Juli, 2022

APPROVAL SHEET

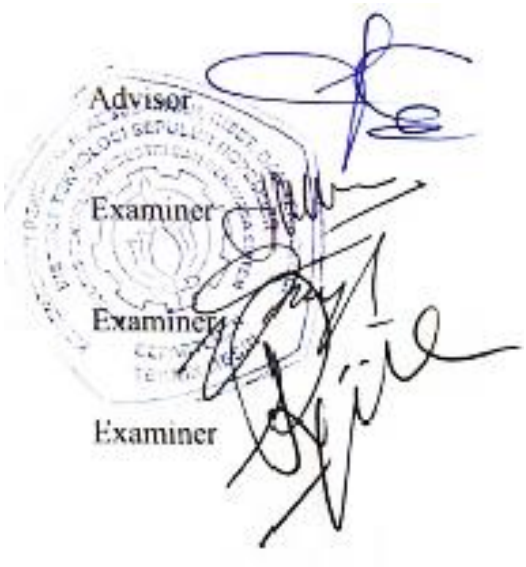
CRACK DETECTION ON SHIP'S PROPELLER USING CONVOLUTIONAL NEURAL NETWORK METHOD

FINAL PROJECT
Submitted to fulfill one of the requirements
for obtaining a bachelor of engineering degree at
Undergraduate Study Program of Mechanical Engineering
Department of Mechanical Engineering
Faculty of Industrial Technology and Systems Engineering
Institut Teknologi Sepuluh Nopember

By: **RHESA BELA**
NRP. 02111840000083

Approved by Final Project Examiner Team:

1. M. Khoirul Effendi, S.T., M.Sc.Eng., Ph.D.
2. Ari Kurniawan Saputra, S.T., M.T.
3. Dinny Harnany, S.T., M.Sc.
4. Arif Wahjudi, S.T., M.T., Ph.D.



SURABAYA
July, 2022

PERNYATAAN ORISINALITAS

Yang bertanda tangan di bawah ini:

Nama mahasiswa / NRP : Rhesa Bela / 02111840000083
Departemen : Teknik Mesin
Dosen Pembimbing / NIP : M. Khoirul Effendi, S.T., M.Sc.Eng., Ph.D.
198204142010121001

Dengan ini menyatakan bahwa Tugas Akhir dengan judul "Pendeteksi *Crack* Pada *Propeller* Kapal Dengan Menggunakan Metode *Convolutional Neural Network*" adalah hasil karya sendiri, bersifat orisinal, dan ditulis dengan mengikuti kaidah penulisan ilmiah.

Bilamana di kemudian hari ditemukan ketidaksesuaian dengan pernyataan ini, maka saya bersedia menerima sanksi sesuai dengan ketentuan yang berlaku di Institut Teknologi Sepuluh Nopember.

Surabaya, 25 Juli 2022

Mengetahui
Dosen Pembimbing

M. Khoirul Effendi, S.T., M.Sc.Eng., Ph.D.
NIP. 198204142010121001

Mahasiswa,

Rhesa Bela
NRP. 02111840000083

STATEMENT OF ORIGINALITY

The undersigned below:

Name of student / NRP : Rhesa Bela / 02111840000083
Departement : Teknik Mesin
Dosen Pembimbing / NIP : M. Khoirul Effendi, S.T., M.Sc.Eng., Ph.D.
198204142010121001

hereby declare that the Final Project with the title of "Crack Detection on Ship's Propeller Using Convolutional Neural Network Method" is the result of my own work, is original, and is written by following the rules of scientific writing.

If in the future there is a discrepancy with this statement, then I am willing to accept sanctions in accordance with the provisions that apply at Institut Teknologi Sepuluh Nopember.

Surabaya, July 25th 2022

Acknowledged
Advisor

M. Khoirul Effendi, S.T., M.Sc.Eng., Ph.D.
NIP. 198204142010121001

Student,

Rhesa Bela
NRP. 02111840000083

KATA PENGANTAR

Puji syukur penulis haturkan kehadirat Tuhan Yang Maha Esa dan Maha Pengasih atas berkat dan izin-Nya Tugas Akhir ini dapat dilaksanakan hingga selesai. Tugas Akhir ini disusun sebagai syarat kelulusan S1 Departmen Teknik Mesin Institut Teknologi Sepuluh Nopember Surabaya.

Penulis menyadari betul bahwa keberhasilan dalam proses penyelesaian Tugas Akhir ini tidak terlepas dari dukungan, bimbingan dan bantuan dari berbagai pihak. Pada kesempatan ini, secara khusus penulis ingin mengucapkan terima kasih kepada:

1. Bapak Henry Adijanta Suhartono dan Ibu Maria Relawati, selaku orang tua penulis dan seluruh keluarga yang telah memberikan segala bentuk dukungan dan doa sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Bapak Mohammad Khoirul Effendi, S.T., M.Sc.Eng. selaku dosen pembimbing Tugas Akhir penulis yang telah memberi bimbingan, arahan dan pembelajaran hingga penulis dapat menyelesaikan Tugas Akhir ini.
3. Seluruh dosen penguji Tugas Akhir yang telah meluangkan waktu dan memberikan saran untuk Tugas Akhir ini.
4. Seluruh Bapak dan Ibu Dosen Department Teknik Mesin ITS yang telah mendidik dan mengajarkan ilmu Teknik Mesin dan ilmu kehidupan selama masa perkuliahan.
5. Christoporus Risang K. dan Bara Atmaja yang selalu mengingatkan penulis untuk melakukan asistensi tugas akhir.
6. Teman – teman bimbingan Pak Khoirul yang berjuang bersama menyelesaikan Tugas Akhir terutama Benedictus H.K.E.
7. Teman – teman discord NOFIC yang telah menemani penulis selama masa suntuk dalam mengerjakan tugas akhir.
8. Teman teman KMK St. Ignatius Loyola yang telah memberikan pengalaman dan cerita selama berkuliah di ITS.
9. Keluarga M61 yang telah memberikan pengalaman dan cerita selama berkuliah di Teknik Mesin ITS.
10. Seluruh civitas akademika ITS yang telah berdinamika selama kehidupan di kampus.
11. Seluruh pihak yang tidak dapat disebutkan satu persatu oleh penulis, yang telah membantu.

Penulis menyadari masih terdapat banyak kekurangan dalam penyusunan Tugas Akhir ini, maka penulis mengharapkan saran dan masukkan dari berbagai pihak. Semoga tugas akhir ini dapat memberikan manfaat bagi pembaca dan ilmu pengetahuan.

Surabaya, Juli 2022

Penulis

PENDETEKSI *CRACK* PADA *PROPELLER* KAPAL DENGAN MENGGUNAKAN METODE *CONVOLUTIONAL NEURAL NETWORK*

Nama Mahasiswa / NRP : Rhesa Bela / 02111840000083

Departemen : Teknik Mesin FTIRS - ITS

Dosen Pembimbing : M. Khoirul Effendi, S.T., M.Sc.Eng., Ph.D.

ABSTRAK

Peningkatan jumlah pelayaran di Indonesia menyebabkan meningkatnya kerusakan kapal yang terjadi. Salah satu kerusakan yang dapat terjadi adalah *crack* pada *propeller* kapal. Untuk menjaga kondisi *propeller* kapal yang baik, *maintenance* pada bagian *propeller* kapal merupakan hal wajib yang harus dilakukan oleh pemilik kapal. *Maintenance* ini bertujuan mencegah kerusakan pada *propeller* seperti benturan, faktor alam, hingga *human error.* Dalam rangka mencegah terjadinya kerusakan dibutuhkan proses *survey* dan *marking* sebagai salah satu langkah dalam melakukan *maintenance.* Proses *survey* dapat dilakukan ketika kapal berada di air atau ketika kapal naik *dock. Survey* ketika kapal berada di air dilakukan dengan *surveyor* mengitari bagian kapal dan melakukan marking pada bagian kerusakan. Untuk hasil *survey* yang lebih menyeluruh pada bagian kapal, seorang *surveyor* harus dapat berenang dan menyelam agar kerusakan yang mungkin terjadi pada bagian bawah kapal dapat terdeteksi kerusakannya pula. Hal ini juga dilakukan pada saat *surveyor m*elakukan pengecekan pada bagian *propeller* kapal. Namun langkah ini memiliki kekurangan diantaranya membutuhkan waktu yang lama dan memiliki peluang terjadinya kecelakaan saat menyelam. Dengan alasan tersebut, dibutuhkan *Remotely Operated Vehicle (ROV)* untuk dapat meminimalisir resiko yang timbul ketika proses *survey* dan *marking. ROV in*i dirancang dengan menggunakan lampu sebagai penerang dan kamera yang bertujuan untuk merekam kondisi bagian kapal yang nantinya dapat diidentifikasi menggunakan *Convolutional Neural Network (CNN).* Data rekaman yang ditangkap oleh kamera langsung diolah dengan menggunakan deteksi *crack* dengan metode *CNN* berarsitektur *SSD mobilenet versi 1.* Dengan menggunakan *batch size 24,* data *training* sebanyak 600 data, iterasi *training* sebanyak *40,000 step*, model mampu mendeteksi crack hingga kedalaman 1,5 meter dibawah permukaan air dan dengan jarak 1 meter dari objek *crack* dengan akurasi 81,48% dan total *loss* sebesar *0,861%*.

Kata kunci: Convolutional Neural Network (CNN), Crack, ROV.

CRACK DETECTION ON SHIP'S PROPELLER USING CONVOLUTIONAL NEURAL NETWORK METHOD

Student Name / NRP : **Rhesa Bela / 02111840000083**

Department : **Mechanical Engineering FTIRS - ITS**

Advisor : **M. Khoirul Effendi, S.T., M.Sc.Eng., Ph.D.**

ABSTRACT

The increase in the number of shipping in Indonesia causes an increase in ship damage that occurs. One of the damages that can occur is on the propeller of the ship. There are so many factors cause damage to propeller including high impact, natural factors, and human error. Maintenance is a way for the ship's owner to prevent the damages that occurs on the propeller. In order to prevent damage, survey and marking process is needed as on of the step in maintenance. The survey and marking process can be done when the ship is in the water or when the ship boards the dock. When the ship is in the water, a surveyor who is able to dive while do a survey and marking process is needed. However this step takes along time to prepare the dive equipment and has a possibility of an accident to the diver. For those reason, the diver can be replace with remotely operated vehicle (ROV) to be able minimize the risk that arise during the survey and marking process. The ROV is designed using lights and the camera to record the condition of the ship which can be identified later using deep learning algorithms. The recording data can be processed directly using Convolutional Neural Network (CNN) to detecting crack that happened on the propeller. The architecture that used in the CNN is SSD Mobilenet V1 with 24 batch size, 600 taining data, and 40.000 training steps. The model was able to detect cracks to a depth of 1.5 meters below the water and with distance of 1 meter from the crack object. The model accuracy reaches 81,48% with training total loss 0,861%.

Keywords: ***Convolutional Neural Network (CNN), Crack, Remotely Operated Vehicle (ROV).***

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR PERSAMAAN

BAB I
PENDAHULUAN

1.1 Latar Belakang

Indonesia adalah negara kepulauan terbesar di dunia. Dengan luas wilayah laut sebesar 5,8juta km^2 dan panjang garis pantai sebesar 81.000 km yang merupakan panjang garis pantai terbesar kedua di dunia. Dari kondisi tersebut Indonesia dapat dikatakan sebagai negara maritime (Arafat, 2018). Kondisi ini merupakan sebuah peluang Indonesia dalam hal meningkatkan perekonomian negara. Gagasan mengenai tol laut juga menjadi salah satu langkah dalam memaksimalkan perekonomian negara. Langkah ini bertujuan untuk memperbaiki proses pengangkutan logistik yang ada di Indonesia. Dampaknya akan terasa pada pemerataan harga bahan – bahan kebutuhan masyarakat di setiap pulau yang berbeda di Indonesia. Strategi utama dalam gagasan ini yaitu dalam mempermudah proses pelayaran dengan membuat jalur pelayaran bebas hambatan yang dapat menyambungkan pelabuhan di Indonesia. Hal ini dapat menambahkan intensitas pelayaran kapal pada perairan di Indonesia.

Meningkatnya intensitas pelayaran di Indonesia, mengakibatkan penggunaan kapal juga akan meningkat. Kapal sebagai alat transportasi yang berguna untuk menyebarkan barang – barang logistik juga memiliki kapasitas tersendiri dalam penggunaannya. *Maintenance* dan *repair* merupakan sebuah langkah terbaik dalam menjaga kondisi dari sebuah kapal. Namun dalam kenyataannya terdapat banyak faktor yang berpotensi mengakibatkan kerusakan pada bagian - bagian kapal, khususnya pada bagian *propeller* kapal. Kerusakan pada *propeller* kapal sendiri dapat berasal dari benturan dan gesekan pada benda lain, faktor alam, hingga *human error*. Akibat dari faktor – faktor tersebut dapat terjadi timbulnya *crack* pada bagian *propeller* kapal.

Kerusakan pada *propeller* kapal dalam hal ini *crack*, tidak dapat diprediksi. *Crack* yang terjadi tidak dapat dianggap remeh karena dampak dari *crack* dapat menyebabkan penurunan efisiensi kerja *propeller*. Hal ini menyebabkan kerugian bagi pemilik kapal dikarenakan kerusakan ini dapat mempengaruhi performa gerak dari sebuah kapal. Efisiensi tersebut akan berdampak pada laju kecepatan sebuah kapal dan berujung pada bertambahnya konsumsi bahan bakar penggerak kapal. Untuk mencegah hal tersebut, inspeksi kapal dapat dilakukan sebagai salah satu langkah preventif kerusakan yang terjadi pada kapal. Langkah utama dalam melakukan inspeksi bagian kapal yaitu *survey* dan *marking.* Tahap ini berguna untuk menganalisis lokasi dan ukuran *crack* yang terjadi pada *propeller* kapal. *Survey* dan *marking* dilakukan pada dua kondisi, yaitu pada saat kapal naik dock/galangan kapal atau ketika kapal sedang berada di perairan. Ketika pada dock, seorang *surveyor* akan memutari seluruh bagian *propeller* kapal. Ketika *crack* ditemukan, maka akan dilakukan proses *marking* pada bagian tersebut. Sedangkan pada proses *survey* dan *marking propeller* kapal sebelum naik dock memiliki beberapa kesulitan tersendiri bagi seorang *surveyor.* Tidak hanya handal dalam menentukan titik – titik *crack* melainkan juga seorang *surveyor* harus memiliki keahlian dalam berenang. Seorang *surveyor* harus bisa melakukan *survey* dan *marking* pada seluruh bagian *propeller* kapal dalam kondisi berenang. Hal ini semakin berbahaya dengan adanya arus bawah laut yang tidak dapat diprediksi. Kondisi ini tentunya dapat membahayakan keselamatan dari *surveyor* itu sendiri.

Untuk mengurangi resiko penyelaman yang dilakukan oleh manusia, dapat digunakan *Remotely Operated Vehicle (ROV). ROV* merupakan sebuah perangkat robot yang digunakan pada bawah air dengan dikontrol menggunakan *remote control.* Penggunaan *ROV* dilengkapi dengan kamera dan lampu yang dapat merekam kondisi *propeller* kapal. Hasil rekaman tersebut dapat diolah dengan menggunakan metode *Deep Learning. Deep Learning* sendiri merupakan bagian dari berbagai macam *machine learning. Machine learning* adalah kecerdasan buatan

yang bertujuan mengoptimalkan suatu sistem dengan memanfaatkan data sampel atau data histori (Alpaydin, 2009).

Deep learning adalah sebuah penerapan jaringan syaraf tiruan (*neural network)* dengan menggunakan perangkat keras (*hardware)* modern. Didalam *deep learning* terdapat tiga jenis tipe *neural network* tiruan diantaranya yaitu, *Multilayer Perceptrons (MLP), Recurrent Neural Network (RNN),* dan *Convolutional Neural Network (CNN).* Berdasarkan data yang diproses, metode *MLP* sendiri dapat mengolah berupa data tabular, sedangkan pada metode *RNN* adalah data teks, dan pada metode *CNN* data yang diolah berupa gambar. Berdasarkan hal tersebut, penulis menggunakan *Convolutional Neural Network* sebagai metode pada *deep learning* yang digunakan menganalisis tangkapan layar pada *ROV*.

1.2 Perumusan Masalah

Dari dasar latar belakang pada bab 1.1, permasalahan yang akan dibahas pada tugas akhir ini adalah:

1. Bagaimana implementasi metode *deep learning* dengan menggunakan *CNN* dalam menentukan lokasi *crack* yang terjadi pada *propeller* kapal?
2. Berapa variasi jumlah data *training* terbaik dengan menggunakan variasi kedalaman uji, dan jarak uji dalam pengujian deteksi *crack* pada *propeller* kapal?
3. Bagaimana tingkat akurasi yang didapat dengan menggunakan *CNN?*

1.3 Batasan Masalah

Adapun batasan masalah yang digunakan dalam tugas akhir ini sebagai berikut:

1. Fokus pelaksanaan tugas akhir ini adalah dengan menggunakan *Convolutional Neural Network* yang merupakan salah satu metode dalam *deep learning.*
2. Bahasa pemrograman yang digunakan adalah *Python* versi *3.7* dengan *framework Tensorflow* 1.15 dan model arsitektur *SSD* Mobilenet V1.
3. Objek deteksi menggunakan tiga citra *Crack* pada dinding yang kemudian dimasukkan kedalam air.
4. Deteksi *crack* dilakukan dengan menggunakan variasi jumlah data *training* 400, 500, dan 600 data; variasi kedalaman 0,5 meter, 1 meter, dan 1,5 meter; variasi jarak 0,5 meter, 0,75 meter, dan 1 meter.
5. Video uji diambil dengan menggunakan kamera GoPro Hero 9.
6. Faktor pencahayaan dan getaran diabaikan.

1.4 Tujuan Penelitian

Pada penelitian ini terdapat tujuan yang akan dicapai yaitu:

1. Mengetahui implementasi metode *deep learning* dengan menggunakan *CNN* dalam menentukan lokasi *crack* yang terjadi pada *propeller* kapal.
2. Mengetahui variasi jumlah data *training* terbaik dengan menggunakan variasi kedalaman uji, dan jarak uji dalam pengujian deteksi *crack* pada *propeller* kapal.
3. Mengetahui tingkat akurasi yang didapat dengan menggunakan *CNN.*

1.5 Manfaat Penelitian

Manfaat yang didapatkan dalam penelitian ini adalah sebagai berikut:

1. Memberikan pengetahuan mengenai implementasi *deep learning* dengan menggunakan *CNN* dalam menentukan lokasi *crack* yang terjadi pada *propeller* kapal
2. Mengetahui tingkat akurasi dari implementasi *Convolutional Neural Network (CNN).*
3. Membantu *survey* dan *marking* pada langkah *repairing propeller* kapal.

BAB II
TINJAUAN PUSTAKA

2.1 Penelitian Terdahulu

Penelitian yang berjudul "Klasifikasi citra dengan menggunakan *convolutional neural network (CNN)* pada *Caltech 101*" dibuat untuk menguji keandalan *CNN* dalam mendeteksi citra dengan basis data *Caltech 101*. Penelitian ini dilakukan pada tahun 2016 oleh I Wayan Suwartika, dkk yang termasuk dalam Jurnal Teknik ITS. Data set yang digunakan berjumlah 390 citra, dengan komposisi kategori unggas 150 gambar, kategori *crocodile* 80 gambar, kategori *cougar* 80 gambar dan kategori *face* 80 gambar. Dari 150 data set unggas yang digunakan dibagi lagi menjadi *ebis, flamingo, pigeon,* dan *emu.* Hasil akurasi yang didapatkan oleh I Wayan Suwartika, dkk menunjukkan hasil akurasi pengolahan citra *convolutional neural network* cukup handal dengan persentase akurasi 20% - 50% pendeteksian total kategori unggas (Eka Putra, 2016).

Penelitian berjudul "*Underwater Object Detection using Tensorflow"* yang dilakukan Avinash Mahavarkar, dkk menunjukkan efektivitas penggunaan metode *CNN, R-CNN, Fast R-CNN, dan Faster RCNN* untuk pendeteksian benda di dalam permukaan air. Hal ini dilakukan karena semakin dalam pendeteksian sebuah objek dipengaruhi oleh perbedaan warna *BGR (blue-green-red)* pada tangkapan gambar. Dari hasil pengujian didapatkan *Faster R-CNN* merupakan metode paling baik dikarenakan menggunakan *regional proposal* yang mengakibatkan proses pengujian menjadi lebih cepat dibandingkan dengan metode lainnya (Mahavarkar et al., 2020).

Penelitian berjudul *"Vision-Based Concrete Crack Detection Using a Convolutional Neural Network"* yang dilakukan oleh Young-Jin Cha (2017) memiliki tujuan untuk menjawab tantangan pada dunia infrastruktur sipil dalam hal *image processing*. Dalam penelitian ini menggunakan metode berbasis *Convolutional Neural Network.* Penelitian ini bertujuan untuk mendeteksi kerusakan *(crack)* yang terjadi pada produk – produk infrastruktur dengan menggunakan *image processing*. Namun dalam penelitiannya, terdapat tantangan untuk dapat mendeteksi *crack* yang terjadi, dalam hal ini adalah faktor cahaya dan bayangan yang terjadi selama pengambilan data. Hasil dari penelitian ini berupa akurasi yang terjadi pada *machine learning* sebesar 98.22% dengan melakukan *training* sebanyak 277 gambar *testing* sebanyak 55 gambar (Cha et al., 2017).

Pada tahun 2018 terdapat penelitian mengenai *CNN* yang dilakukan oleh Rizky Dwi Novyantika dengan judul penelitian "Deteksi tanda nomor kendaraan bermotor pada media *streaming* dengan algoritma *convolutional neural network* menggunakan *tersoflow*". Pada penelitian ini didapatkan hasil akurasi model pendeteksi tanda nomor kendaraan bermotor dengan algoritma *convolutional neural network* berkisar 70% - 100% dengan menggunakan 25000 *epoch* dan *batch size* sebanyak 8 (Novyantika, 2018).

Syinta Nuri Mashita pada tahun 2020 melakukan penelitian mengenai "Implementasi *Deep Learning Object Detection* rambu K3 pada video menggunakan metode *convolutional neural network (CNN)* dengan *tensorflow*". Pada penelitian ini, Syinta menggunakan sistem jaringan *VGG16-NET* untuk mendeteksi model rambu jalur evakuasi dan alat pemadam api ringan (APAR) yang didapatkan dari tangkapan video. Model yang digunakan dalam penelitian adalah *SSD-mobilenet V1* dengan tingkat akurasi pendeteksian rambu sebesar 50% - 97%. Jumlah *epoch* yang dilakukan adalah sebanyak 69.284 dengan menggunakan *batch size* sebesar 2 (Mashita, 2020).

2.2 *Remotely Operated Vehicle (ROV)*

Remotely Operated Vehicle (ROV) merupakan sebuah robot menyerupai kapal yang dapat menyelam dan dikontrol dengan *remote control*. Sistem pengendalian dilakukan pada *remote control* yang disambungkan dengan menggunakan *tether* yang bertujuan sebagai penyalur sumber energi dan data yang akan ditransferkan dari kamera *(input)* yang berada pada *ROV* (Anam & Setiawan, 2015), (R. A. Septian, A. Rahmania, M. I. Nugraha, 2017).

Dalam merancang sebuah *ROV* terdapat 3 hal penting yang perlu diperhatikan dalam merancang konstruksi, *hardware* dan *software*, yaitu faktor bahan, bentuk dan komponen lain yang terpasang pada *ROV*. Komponen lain yang dimaksud adalah perangkat *hardware* seperti kamera dan juga lampu penerang. *Software* juga digunakan agar *ROV* dapat bergerak dan menangkap gambar di dalam air (Djazuli Sa'id, 2012). Gambar 2.1 merupakan salah satu contoh desain *ROV* dari perusahaan *Blue Robotics* yang terdiri dari *frame*, *thruster*, *control system / box*, *camera*, lampu, dan pelampung.

Gambar 2. 1 Desain ROV BlueROV2
(https://bluerobotics.com/store/rov/bluerov2/)

Pada *BlueROV2* terdapat enam *thruster* yang digunakan untuk sistem kendali gerak *ROV* seperti yang ditunjukkan pada gambar 2.2. Dari keenam *thruster* dibagi menjadi dua jenis yaitu *thruster* dengan putaran *clockwise* (CW) dan *counter clockwise* (CCW). Akibat perbedaan perputaran dan peletakannya, memungkinkan *ROV* dapat bergerak secara translasi hingga rotasi sekalipun.

Gambar 2. 2 Thruster pada BlueROV2
(https://bluerobotics.com/store/rov/bluerov2/)

Pada bagian depan *ROV,* terdapat lampu yang berguna untuk memberikan penerangan bagi kamera yang menangkap gambar dalam permukaan air sehingga dapat mempermudah *driver* yang menggerakan *ROV* dari permukaan air. Pemasangan jumlah lampu dapat disesuaikan dengan kebutuhan.

Kamera pada *ROV* bertujuan sebagai "mata" bagi *driver* yang mengontrol gerak *ROV* dari permukaan air. Selain itu *ROV* juga dapat dipasangkan kamera tambahan yang berguna untuk pengambilan gambar bawah air dengan spesifikasi dan komponen yang telah disesuaikan.

Kemudian terdapat pelampung pada bagian atas *ROV*. Fungsi utama dari pelampung ini untuk meningkatkan kemampuan angkat pada *ROV* dan menjaga kestabilan pada *ROV*. Pelampung ini diletakan pada bagian atas *frame ROV*.

Komponen terpenting pada *ROV* adalah sistem kontrol *ROV*. Sistem kontrol ini merupakan "otak" dari *ROV*. Seluruh sistem kendali terdapat pada *box controller* ini, sehingga dalam melakukan *transfer* data dari *ROV* menuju *driver*, dibutuhkan *tether* / kabel penghubung.

2.3 Retakan Pada *Propeller* Kapal *(Crack)*

Retakan atau *Crack* merupakan sebuah regangan yang terjadi pada sebuah material yang masih tersambung satu sama lainnya. *Crack* pada permukaan memiliki bentuk yang tidak dapat diprediksi (berbentuk simetris). Pada gambar 2.3 merupakan fenomena *crack* yang terjadi pada tembok. Fenomena ini memiliki dampak yang fatal bagi kemampuan sebuah material dalam menahan beban yang diterima. Ada beberapa faktor yang menyebabkan sebuah material mengalami *crack* diantaranya pengaruh temperature, pembebanan yang melebihi standar ijin, hingga pengaruh komposisi material yang digunakan pada saat awal pembentukan material.

Gambar 2. 3 Fenomena crack pada dinding bangunan
(Cha et al., 2017)

Crack tidak hanya dapat terjadi pada material non logam, melainkan juga dapat terjadi pada material logam seperti halnya pada *propeller* sebuah kapal seperti yang ditunjukkan pada gambar 2.4. *Crack* yang terjadi pada *propeller* kapal dapat diakibatkan oleh pemakaian material yang kurang tepat, proses pemotongan ketika *fabrication, fit-up* pada setiap sambungan, tindakan pengelasan yang kurang sempurna, dan terdapat pembebanan yang berlebihan (Djazuli Sa'id, 2012). Fenomena ini sangatlah merugikan bagi operasional kapal.

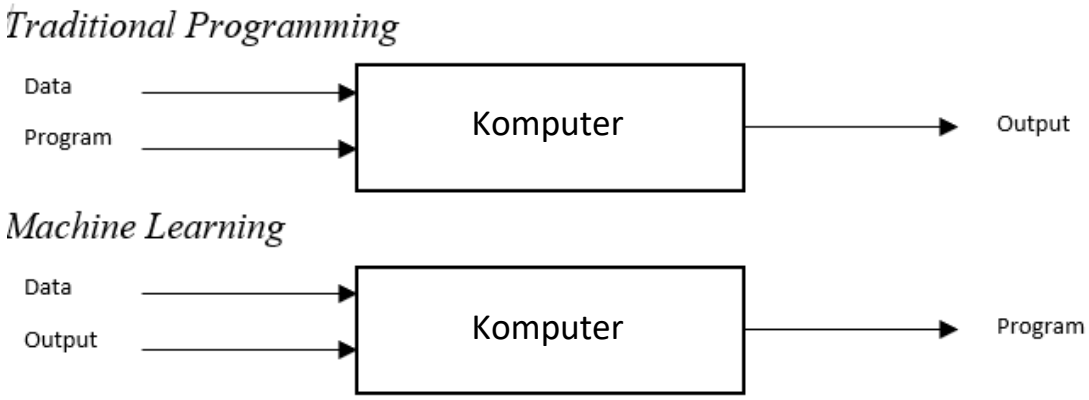
Gambar 2. 4 Crack pada propeller kapal
(Djazuli Sa'id, 2012)

Pada *propeller* kapal, *crack* merupakan sebuah fenomena yang merugikan karena dapat menyebabkan penurunan efisiensi putaran propeller dan hal ini berhubungan dengan konsumsi energi yang digunakan pada kapal. Kejadian ini jika dibiarkan akan menyebabkan konsumsi energy yang berlebih pada kapal. Untuk mencegah terjadinya dampak yang berkelanjutan, *crack* pada *propeller* kapal dapat dilakukan beberapa tindakan salah satunya ialah tahap *survey* dan *marking*. Tahap ini bertujuan untuk mengidentifikasi lokasi *crack* yang terjadi pada *propeller* kapal. Survey dan marking dapat dilakukan pada saat kapal sudah berada pada dock atau sebelum naik dock. Ketika kapal belum menaiki dock, dibutuhkan seorang surveyor untuk dapat menganalisa crack dengan cara berenang. Hal ini merupakan langkah yang memiliki banyak kendala dalam pelaksanaannya. Seorang surveyor harus memiliki kemampuan dalam berenang sekaligus dalam menentukan letak crack pada propeller kapal. Bahkan pada survey dan marking tidak menutup kemungkinan terjadinya kecelakaan pada surveyor ketika melakukan survey crack.

2.4 *Machine Learning*

Machine learning merupakan sebuah cabang ilmu dari kecerdasan buatan dimana sebuah komputer memiliki kemampuan untuk mempelajari input secara mandiri dan menghasilkan output tersendiri tanpa perlu pemrograman ulang oleh manusia, namun dalam persiapan awal dibutuhkan perintah dari manusia untuk dapat menjalankan program *machine learning.* Penyelesaian yang dilakukan memiliki pola tersendiri yang dibentuk oleh jaringan komputer.

Machine learning memiliki perbedaan dengan *traditional programming*. Pada *traditional programming*, *user* memasukan data input dan program pada komputer dengan tujuan mendapatkan *output* yang diinginkan, sedangkan pada *machine learning*, *user* memasukan data input serta *output* dengan tujuan mendapat sebuah program yang dapat digunakan sebagai *input* dari *traditional programming.* Hal ini dapat dilihat pada skema pada gambar 2.5 berikut.



Gambar 2. 5 Traditional programming vs machine learning

Pada *machine learning* terdapat beberapa algoritma yang berkembang dan populer diantaranya

- Algoritma *supervised learning* merupakan algoritma yang membutuhkan data berlabel berupa data input dan output sebenarnya. Algoritma ini memungkinkan komputer untuk mempelajari data input menjadi output yang diinginkan, lalu output aktual akan dibandingkan dengan output yang sebenarnya sehingga didapatkan *error* pada hasil *output* aktualnya. Semakin banyak data input yang digunakan, maka hasil *supervised learning* akan semakin baik.
- Algoritma *Unsupervised learning* merupakan algoritma yang tidak membutuhkan data label output. *Unsupervised learning* memungkinkan algoritma untuk belajar berdasarkan pola dan struktur data input yang diberikan, sehingga dalam algoritma ini tidak memiliki validasi kebenaran terkait hasil yang didapatkan.
- Algoritma *reinforcement learning* merupakan algoritma yang tidak memiliki dataset awal untuk dipelajari. *Reinforcement learning* mengedepankan *trial and error* pada pengujiannya. Pada algoritma ini terdapat peran *agent* yang berguna untuk pembuat keputusan dan *environment* sebagai "pengalaman" yang bertujuan untuk memperbaiki *agent* dalam mengambil setiap keputusan. Dalam hal ini, *agent* merupakan bagian yang mempelajari pengalaman (*environment*) yang ada demi hasil output yang terbaik

2.5 *Computer Vision*

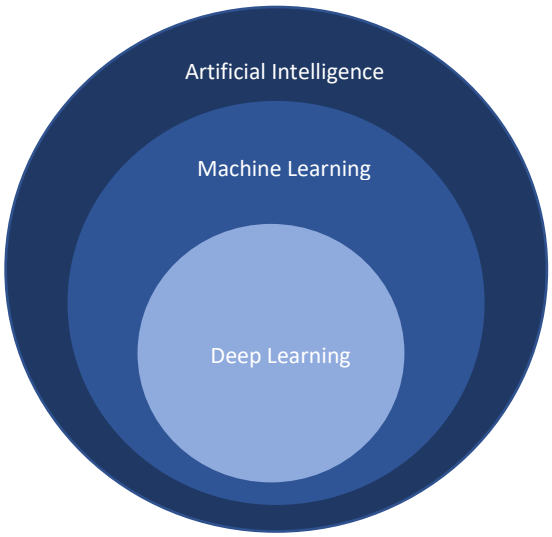
Computer Vision adalah sebuah ilmu pengetahuan dimana sebuah mesin dapat melihat gambar / citra sendiri. Tidak hanya sebagai "mata", *computer vision* juga dapat membuat sebuah mesin belajar dari informasi gambar yang diinputkan sehingga didapatkan sebuah hasil melalui proses pengolahan gambar *(image processing)* sesuai dengan *output* yang diharapkan (Alpaydin, 2009), (Paramarthalingam, 2016). Terdapat tiga tahap dalam *computer vision* yaitu *acquiring* atau mendapatkan data berupa foto bahkan video. Kemudian langkah kedua adalah *processing* yang merupakan pemrosesan data dengan bantuan *deep learning* yang sudah terlatih. Pada langkah terakhir adalah *understanding* yang merupakan langkah komputer untuk menentukan hal yang perlu dilakukan setelah mengidentifikasi sebuah data input. Gambar 2.6 berikut merupakan contoh dari penggunaan *computer vision.*

Gambar 2. 6 Computer vision untuk mendeteksi objek
(Al-Azzo et al., 2018)

Data inputan berupa gambar yang diterima oleh *computer vision* dapat diproses dengan menggunakan perintah dari kecerdasan buatan *(Artificial Intelligence)* yang kemudian gambar dapat diolah dan diidentifikasi sesuai dengan perintah yang diberikan. Menurut John Rajan (John Rajan et al., 2020) penggunaan *computer vision* dengan bantuan AI merupakan sebuah dampak yang besar bagi perkembangan dunia industri manufaktur. Pada pengaplikasian di bidang manufaktur dalam hal *quality control*, sebuah gambar objek benda hasil manufaktur yang ditangkap oleh *computer vision* dapat dipelajari dengan menggunakan metode *deep learning* dimana objek gambar tersebut dapat dianalisis berdasarkan data yang telah diuji sebelumnya. Hasil analisis tersebut dapat berupa identifikasi kerusakan pada objek sesuai dengan data uji yang digunakan (Djazuli Sa'id, 2012).

2.6 *Deep Learning*

Konsep *deep learning* pertama kali dikemukakan pada tahun 1986. *Deep learning* merupakan *machine learning* yang menggunakan *deep neural network* untuk menyelesaikan permasalahan pada domain *machine learning* (Traore et al., 2018). Pada gambar 2.7 ditunjukkan perbandingan antara *artificial intelligence, machine learning,* dan *deep learning.*



Gambar 2. 7 Hubungan antara deep learning, machine learning, dan artificial intelligence

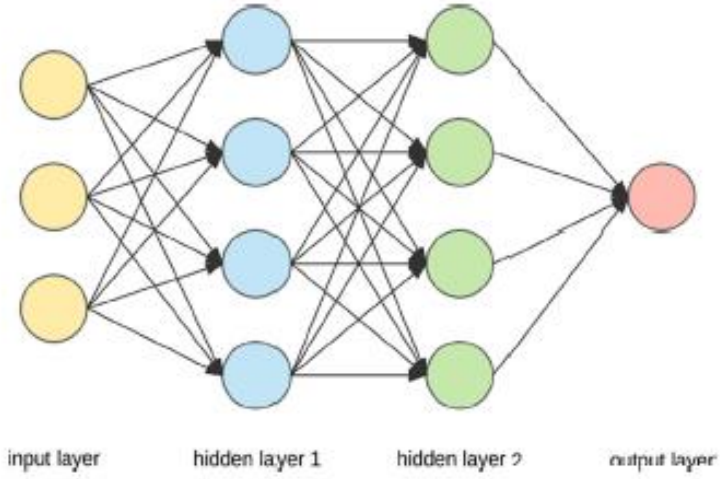
Inti dari *deep learning* adalah menirukan cara berpikir manusia atau dapat diartikan sebagai cara kerja sistem saraf otak manusia. Namun, metode *deep learning* memiliki kapasitas

yang lebih besar sehingga terdapat *neural network* dalam skala besar yang digunakan pada algoritma *deep learning*. *Deep learning* sendiri merupakan pengembangan dari *machine learning.* Tujuan utama dari *deep learning* adalah untuk menyelesaikan permasalahan manusia secara otomatis sehingga komputer dapat belajar dengan input yang diberikan.

Deep learning memiliki beberapa algoritma diantaranya *Convolutional Neural Network (CNN)* yang merupakan sebuah algoritma *deep learning* populer digunakan dalam mendeteksi gambar / visual. Lalu terdapat *long shortterm memory network (LSTM), recurrent neural network (RNN),* dan *self organizing maps (SOM).* Penggunaan keempat jenis algoritma ini menyesuaikan dengan kebutuhan dan fungsi dari algoritma.

2.7 *Neural Network*

Neural network merupakan bagian pembangun dari sistem *deep learning*. Berawal dari konsep jaringan sistem syaraf manusia yang kemudian diadopsi menjadi jaringan syaraf tiruan atau yang biasa disebut *artificial neural network (ANN).* Penggunaan *ANN* harus mengandung struktur grafik berlabel / memiliki output yang sebenarnya. Hal ini diperlukan agar *ANN* dapat melakukan perhitungan sederhana yang kemudian dapat diketahui kecocokan antara hasil perhitungan dengan output sebenarnya. *Neural network* sendiri memiliki 3 lapisan pokok yaitu *input layer, hidden layer,* dan *output layer* (Pal & Hsieh, 2021; Traore et al., 2018; Vidal Pino et al., 2021). Adapun visualisasi mengenai *neural network* seperti pada gambar 2.8.



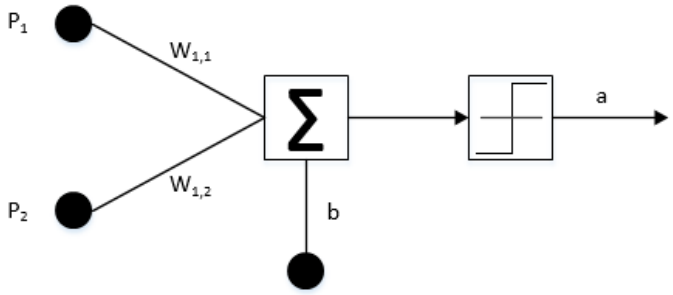
Gambar 2. 8 Jenis lapisan dalam Neural Network (Geitgey, 2016)

Pada bagian *input layer, nodes* mendapatkan inputan gambar yang memiliki tiga dimensi yaitu tinggi gambar, lebar gambar, dan dimensi warna gambar. Pada warna gambar terdapat *channel* warna hitam dan putih dan *channel* warna *RGB (red, green, blue).* Pada *hidden layer,* terdapat perhitungan didalamnya yang dilakukan oleh unit pemroses jaringan *(neuron)* dimana hasil perhitungannya tidak dapat dilihat. Hal ini yang menyebabkan layer ini bernama *hidden. Output layer* merupakan lapisan akhir pada neuron. Pada lapisan ini, hasil data komputasi dapat kita lihat (Alpaydin, 2009).

2.7.1 *Weight* (Bobot)

Pada jaringan *neural network,* terdapat bagian nodes yang bertugas untuk melakukan perhitungan sederhana dalam menentukan output. Dalam pembentukan output, terdapat label yang berupa bobot *(weight)* yang berfungsi sebagai penguat / pelemah sinyal yang melewati sambungan nodes seperti pada gambar 2.9. Sinyal yang melewati sambungan akan

mendapatkan dua macam perlakuan yaitu pelemahan sinyal atau penguatan sinyal output. Nilai *weight* besar antar sambungan menandakan *node* 1 memiliki nilai yang lebih dominan daripada node 2 atau output. Sedangkan nilai *weight* mendekati nilai 0 berarti tidak mempengaruhi nilai input terhadap output yang dihasilkan, dan untuk nilai negatif pada *weight* berarti penambahan nilai input akan mengurangi output yang dihasilkan.



Gambar 2. 9 Susunan neural network sederhana dengan input p1 dan p2

Pada gambar 2.9 terdapat 2 nilai input P yang kemudian berhubungan dengan *weight sum* dimana terdapat 2 buah *weight* tiap sambungan, terdapat *bias* dengan simbol b, simbol n mengidentifikasi penjumlahan nilai input yang dikalikan dengan *weight* dan ditambahkan dengan *bias.* Dalam fungsi umum dapat diformulasikan sebagai berikut:

$$f(w_1 \times p_1 + \ w_2 \times p_2 + \ \ldots + w_n \times p_n + b) \tag{2. 1}$$

$$f((\sum_{i=1}^{n} w_i \, p_i) + b) \tag{2. 2}$$

2.7.2 *Activation Function*

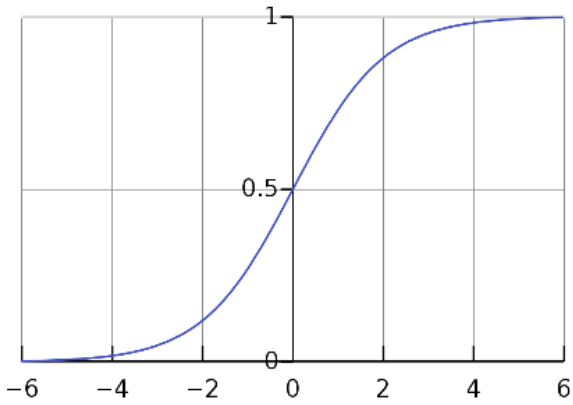
Activation Function berfungsi untuk menentukan sebuah neuron untuk dilakukan transmisi. *Step function* merupakan jenis fungsi aktivasi yang sederhana hal ini dapat dilihat dari persamaan dasarnya,

$$f(x) = \begin{cases} 1, if \ x \geq 0 \\ 0, if \ x < 0 \end{cases} \tag{2. 3}$$

Step function merupakan salah satu fungsi paling sederhana, jika nilai *weight* $\sum_{i=1}^{n} w_i \, x_i \ >$ 0 maka nilai output yang dihasilkan adalah 1, jika tidak maka nilai output yang dihasilkan adalah 0. Pada fungsi aktivasi ini tidak dapat digunakan pada kasus integrasi dikarenakan terjadi masalah ketika melakukan penurunan gradient dan training data pada network terkait. Sehingga dari hal tersebut terdapat bermacam – macam jenis fungsi aktivasi yang dapat digunakan untuk kasus tertentu yang ditampilkan pada gambar 2.10 hingga 2.14 berikut.

- *Sigmoid*

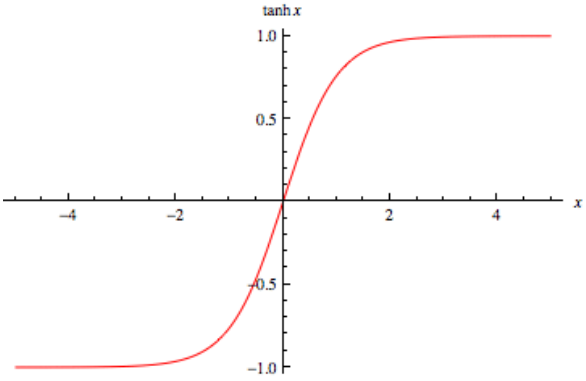
$$f(x) = \frac{1}{1 + e^{-x}} \tag{2. 4}$$



Gambar 2. 10 Hasil output sigmoid function
(https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html)

- *Hyperbolic Tangent (tanh)*

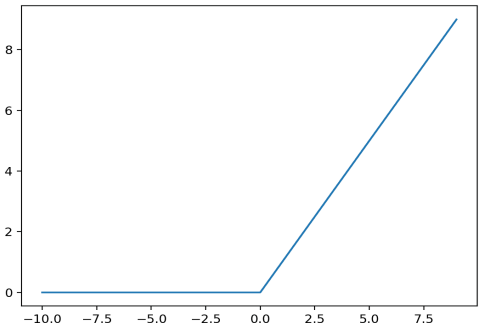
$$f(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \tag{2.5}$$



Gambar 2. 11 Hasil output Hyperbolic Tangent Function
(https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html)

- *Rectifier Linear Unit (ReLu)*

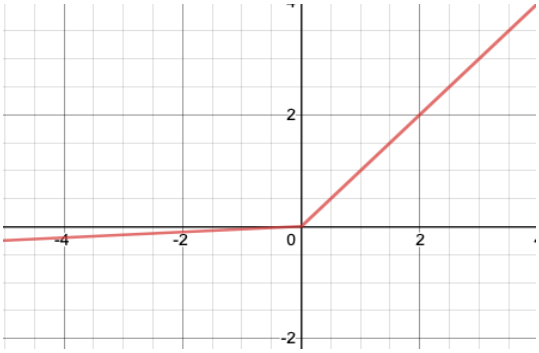
$$f(x) = (0, x) \tag{2.6}$$



Gambar 2. 12 Hasil output ReLu Function
(https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html)

- *Leaky ReLu*

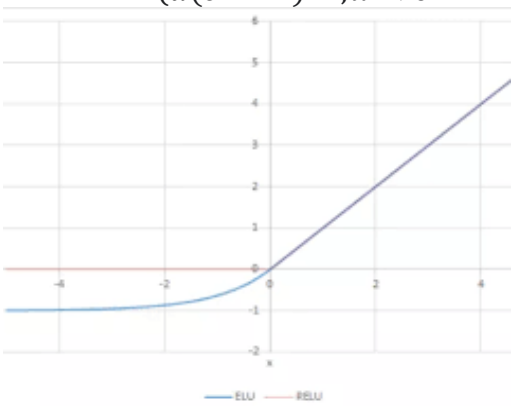
$$f(x) = (0.1x, x) \tag{2.7}$$



Gambar 2. 13 Hasil output Leaky ReLu Function (https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html)

- *Exponential Linear Units (ELUs)*

$$f(x) = \begin{cases} x & ,x \geq 0 \\ \alpha(e^x - 1) & ,x < 0 \end{cases} \tag{2.8}$$



Gambar 2. 14 Hasil output Leaky ReLu Function (https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html)

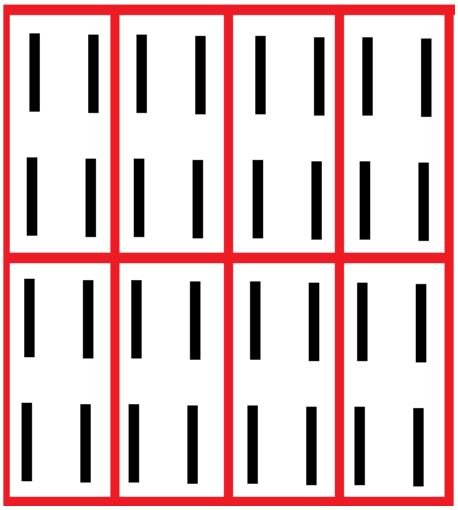
2.8 Convolutional Neural Network (*CNN*)

CNN merupakan jenis dari *deep neural network* yang digunakan pada data dengan topologi seperti grid dan memiliki efisiensi dalam mengenal gambar dan klasifikasi (Sharma et al., 2019). Inputan berupa gambar akan diolah sedemikian rupa sehingga mesin dapat "belajar" mengenali setiap gambar yang diterima. Semakin banyak data yang "dipelajari" maka semakin baik kualitas *output* yang dihasilkan (John Rajan et al., 2020). Konstruksi dalam *CNN* terinspirasi dari *visual cortex* pada manusia atau bagian pada otak manusia untuk memproses informasi dalam bentuk visual.

Langkah kerja dalam *CNN* dimulai dari pemecahan gambar menjadi gambar – gambar yang lebih kecil dan bertumpang tindih. Langkah ini bertujuan untuk mempermudah proses training data yang ingin diidentifikasi. Proses ini dapat dilihat pada gambar 2.15 berikut.

Gambar 2. 15 Pemecahan gambar
(https://medium.com/nodeflux/mengenal-convolutional-neural-network-8bd207ad4a8d)

Kemudian gambar yang berukuran kecil tersebut diproses pada *small neural network*, hal ini bertujuan agar mesin dapat mengenali setiap potongan gambar dengan detail meskipun dalam bentuk potongan. Langkah selanjutnya, potongan gambar yang telah terbaca / dikenali akan disimpan dalam bentuk *array* baru agar bentuk gambar tetap sesuai dengan gambar aslinya. Hasil *array* yang didapatkan sekarang memiliki ukuran yang besar, oleh karena itu dilakukan *downsample* pada data *array*. Data tersebut dibagi menjadi beberapa kelompok dengan matriks 2x2 tiap kelompoknya seperti pada gambar 2.16 berikut.



Gambar 2. 16 Data array dikelompokkan menjadi 2x2

Kemudian data *array* akan direduksi dengan mengambil nilai tertinggi di tiap kelompoknya sehingga didapatkan data *array* yang lebih kecil. Pada langkah akhirnya dihasilkan data *array* dengan ukuran kecil yang berasal dari data input (gambar) dengan jumlah *array* yang besar seperti pada gambar 2.17. Data ini kemudian dapat diolah pada *neural network* lainnya dan memunculkan *output* sesuai dengan data yang telah dipelajari (Afonso et al., 2019). Proses pada gambar 2.16 dapat disebut dengan *training image*. Semakin banyak gambar yang di train, semakin akurat komputer dapat mendeteksi sesuatu.

Gambar 2. 17 Beberapa gambar yang dilakukan proses training (Cha et al., 2017)

Setelah *training* gambar selesai, terdapat langkah selanjutnya yaitu pengujian arsitektur dengan menggunakan video atau tangkapan layar kamera hingga dengan melakukan pengujian dengan gambar – gambar yang berbeda dari data *training*. Ketika data uji sudah didapat, maka arsitektur yang terbentuk akan menghasilkan persentase keberhasilan arsitektur dalam mendeteksi sebuah objek. Untuk mengukur akurasi pada sebuah model dapat dilakukan dengan menggunakan metode perhitungan *confusion matrix* seperti pada tabel 2.1 berikut:

Tabel 2. 1 Confusion matrix

		Nilai sebenarnya	
		TRUE	FALSE
Nilai prediksi	TRUE	TP (True Positive) *Corect result*	FP (False Positive) *Unexpected result*
	FALSE	FN (False Negative) *Missing result*	TN (True Negative) *Corect absence of result*

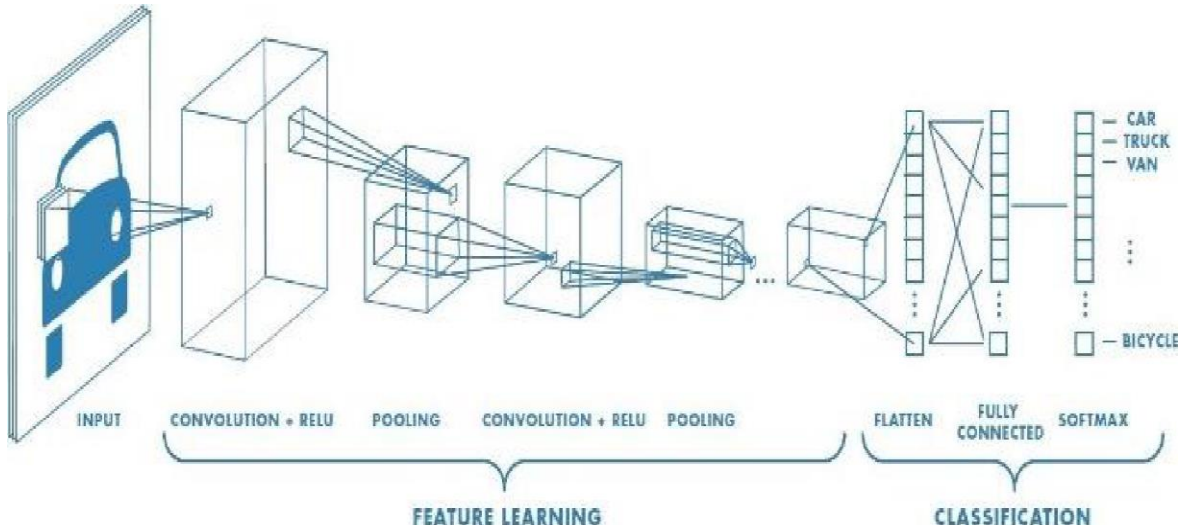
- *True positives*
  Berarti hasil model dapat dengan benar mendeteksi kelas positif. Sebagai contoh hasil model dapat mendeteksi "gambar burung" dengan testing data gambar burung
- *True negatives*
  Berarti hasil model dapat dengan benar mendeteksi kelas negative. Sebagai contoh hasil model dapat mendeteksi "bukan gambar burung" dengan testing data gambar kucing
- *False positives*
  Berarti hasil model tidak dapat mendeteksi kelas positif. Sebagai contoh hasil model dapat mendeteksi "gambar burung" dengan testing data gambar kucing
- *False negatives*
  Berarti hasil model tidak dapat mendeteksi kelas negatif. Sebagai contoh hasil model dapat mendeteksi "bukan gambar burung" dengan testing data gambar burung.

Pada tabel 2.1 yang ditunjukkan merupakan tabel *confusion matrix* yang digunakan sebagai penganalisisan seberapa bagus model yang telah diolah (Geitgey, 2016). Tabel 2.1 dapat menjadi tolak ukur keberhasilan sebuah model *CNN* yang telah terbentuk (Traore et al.,

2018). Setelah data telah diklasifikasikan pada keempat bagian (TP, FP, FN, TN) kemudian dapat dilakukan perhitungan dengan formulasi pada persamaan 2.9:

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN} \quad (2.9)$$

Secara umum, arsitektur pada *CNN* dibagi menjadi dua bagian besar yaitu *feature learning* dan *classification* (Geitgey, 2016). Adapun skema arsitektur *CNN* seperti pada gambar 2.18 berikut.



Gambar 2. 18 Arsitektur umum convolutional neural network (Afonso et al., 2019)

2.8.1 *Feature learning*

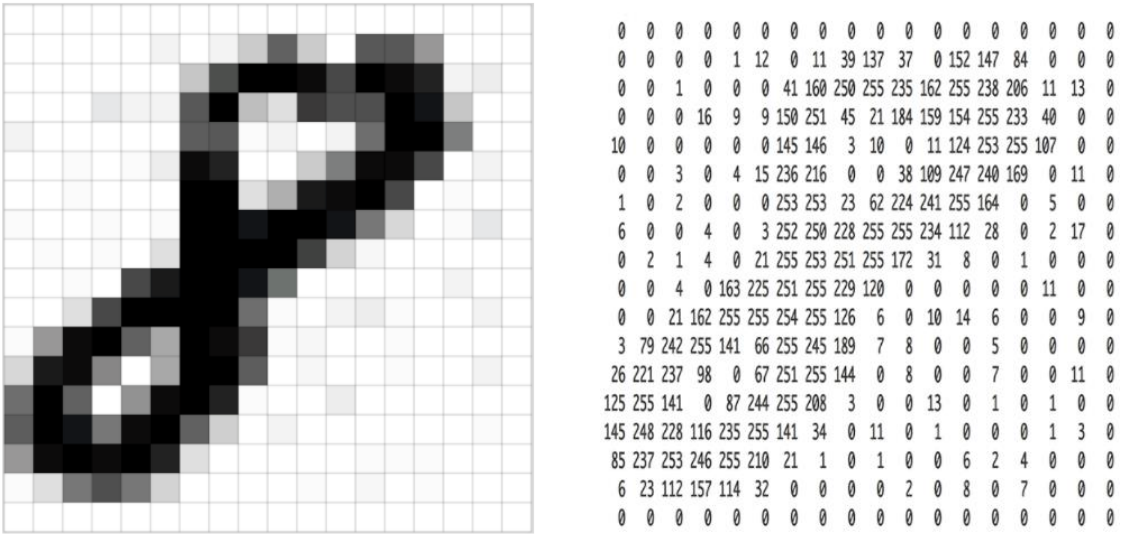
Feature learning merupakan bagian yang mengubah sebuah foto / gambar menjadi sebuah kumpulan angka (membuat sebuah kode) yang nantinya dapat diproses pada bagian *classification.* Pada bagian *feature learning* ini dibagi menjadi 2 bagian lagi yaitu *convolutional layer* dan *pooling layer*. Namun pada sebagian penelitian yang tidak menggunakan *pooling layer*.

2.8.1.1 Transformasi Gambar

Sebelum terjadi proses konvolusi, gambar input terlebih dahulu diubah menjadi nilai angka. Hal ini dilakukan karena pada komputer sebuah gambar diidentifikasikan sebagai kumpulan angka yang didapatkan dari seberapa gelap sebuah gambar. Dalam arti lain setiap *pixel* pada gambar memiliki nilai tersendiri bergantung tingkat warna yang terdapat dalam satu *pixel*. Pada gambar sendiri terdapat 2 jenis pewarnaan yang digunakan yaitu gambar dengan warna hitam dan putih dan gambar dengan warna *RGB* (*red, green, blue*). Perbedaan kedua pewarnaan tersebut terletak pada layer yang digunakan sebagai inputan pada komputer. Pada gambar hitam & putih, komputer akan mengidentifikasikan sebuah gambar menjadi 1 layer input saja yang memiliki nilai tiap pixel dengan range 0 – 255. Pada gambar 2.19 merupakan *range* nilai pixel yang ada pada gambar hitam dan putih yang akan dibaca oleh komputer. Sehingga pada sebuah gambar hitam & putih akan terbaca sebagai berikut.

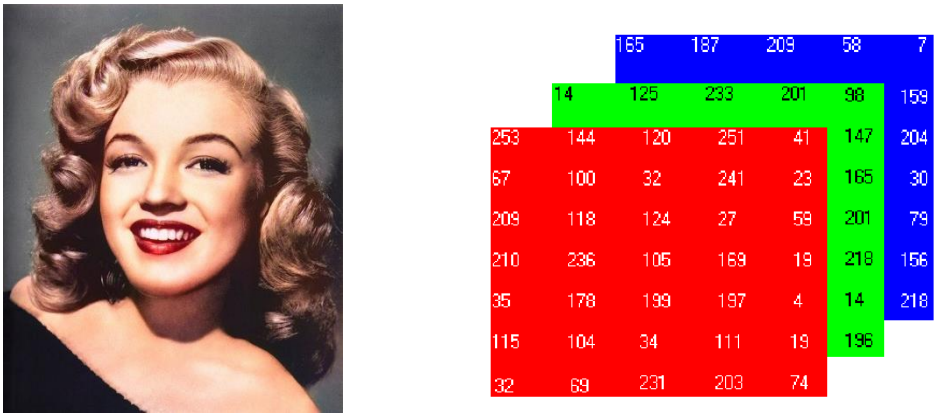
Gambar 2. 19 Range nilai pixel pada gambar hitam & putih
(Paramarthalingam, 2016)

Pada gambar 2.20 merupakan visualisasi citra yang didefinisikan oleh komputer dalam bentuk nilai pixel. Berbeda dengan gambar hitam & putih, gambar RGB memiliki 3 layer sebagai input nya dimana setiap layer merepresentasikan warna pada tiap pixel yaitu merah (*red*), hijau (*green*), dan biru (*blue*) dengan range nilai tiap pixel yang sama dengan gambar hitam & putih yaitu 0 – 255.



Gambar 2. 20 Nilai input gambar hitam & putih yang dibaca komputer
(https://www.esantus.com/blog/2019/1/31/convolutional-neural-networks-a-quick-guide-for-newbies)

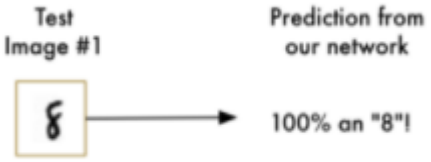
Pada gambar 2.11 merupakan ilustrasi dari gambar RGB yang dibaca oleh komputer dimana setiap pixel pada gambar terdiri atas 3 buah nilai warna yaitu merah, hijau, dan biru sesuai dengan tingkat kegelapan dari sebuah warna pixel.



Gambar 2. 21 Ilustrasi nilai input gambar RGB yang dibaca komputer
(Afonso et al., 2019)

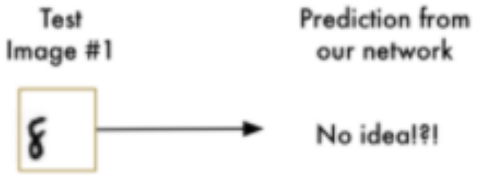
2.8.1.2 *Convolution Layer*

Pada neural network sendiri, pendeteksian sebuah gambar juga dapat dilakukan. Misalnya dalam menentukan angka 8 dalam sebuah gambar. Dengan melakukan *testing* pada data training angka 8 didapatkan keluaran seperti pada gambar 2.22 berikut.



Gambar 2. 22 Hasil testing data angka 8
(Alpaydin, 2009)

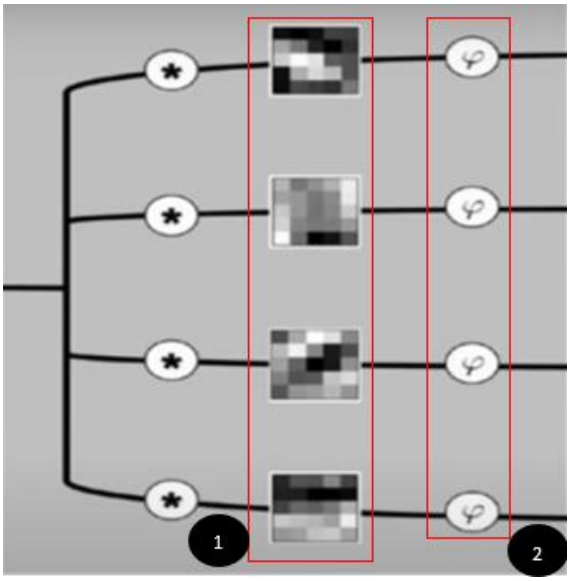
Namun jika dilakukan *testing* data dengan gambar nilai 8 yang kurang sesuai dengan data training akan didapatkan keluaran seperti pada gambar 2.23 berikut.



Gambar 2. 23 Hasil testing data angka 8 dengan posisi tidak center
(Alpaydin, 2009)

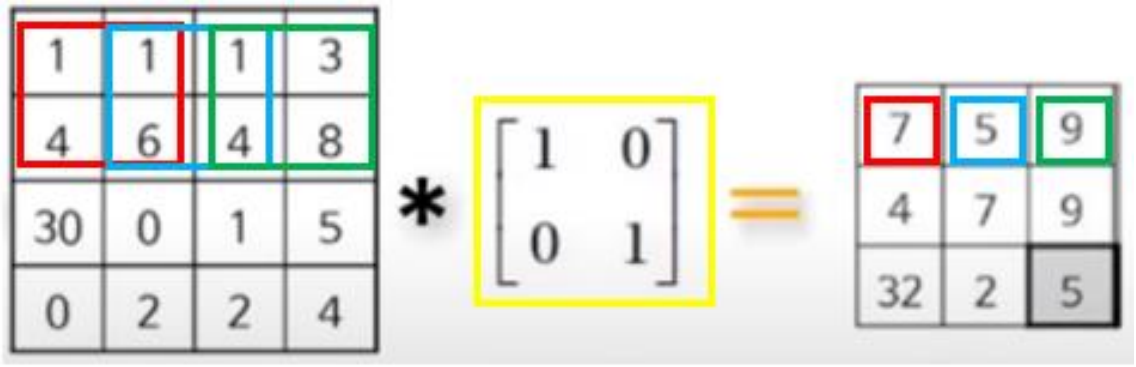
Pada gambar 2.23, komputer akan mendeteksi gambar testing data tidak sesuai dengan angka 8 akibat posisi angka tidak *center*. Tentunya hal ini merupakan sebuah kelemahan dari *machine learning.* Untuk mencegah terjadinya hal tersebut, *CNN* memiliki algoritma untuk mengatasi "*error"* yang terjadi yaitu dengan adanya proses konvolusi yang terjadi pada *convolution layer.*

Convolution layer Merupakan bagian dari *feature learning* yang berfungsi sebagai pemecah gambar menjadi bagian yang kecil dalam bentuk *array*. Pemecahan tersebut dilakukan dengan adanya *convolution filter* atau yang sering disebut sebagai *kernel*. Metode yang digunakan dalam pemecahan ini dinamakan *sliding window* atau jendela geser. Dengan metode ini, memungkinkan sebuah komputer untuk mendeteksi objek yang diinginkan dalam posisi tertentu. Selain itu di dalam *convolution layer* terdapat fungsi aktivasi yang biasa digunakan yaitu ReLu (*Rectified Linear Unit).* Fungsi aktivasi ini berguna untuk mengubah nilai negatif menjadi nilai nol. Adapun ilustrasi pada *convolution layer* seperti pada gambar 2.24.



Gambar 2. 24 (1) Convolution filter, (2) Fungsi aktivasi (Traore et al., 2018)

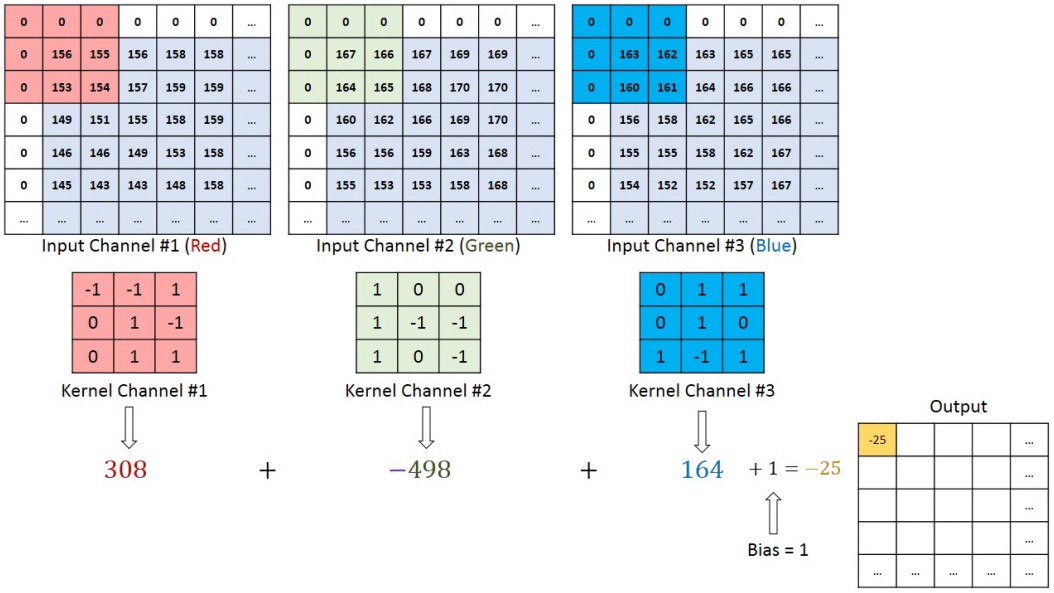
Pada dasarnya setiap *convolution filter* akan menghasilkan *feature map* yang sama dengan jumlah *convolution filter* itu sendiri seperti contoh pada gambar 2.25 berikut.



Gambar 2. 25 Convolution layer process

Gambar 2.25 menunjukkan bagaimana cara kerja dari *convolution layer* pada gambar hitam & putih. Pada matriks pertama merupakan matriks dari inputan yaitu berupa gambar. pada matriks kedua dinamakan *convolution filter* dan pada matriks ketiga / matriks hasil dinamakan *feature map*. Sistem kerja dari *convolution layer* adalah dengan menggunakan *convolution filter* sebagai pengali matriks input. Proses perhitungan yang dilakukan adalah dengan mengkali ukuran matriks input dengan *convolution layer* seperti kotak berwarna merah pada gambar. Hasil perhitungan didapatkan (1×1) + (1×0) + (4×0) + (1×6) = 7. Hasil nilai 7 tersebut lali diletakan pada matriks ke 3 sebagai bagian dari *feature map*. Perhitungan tersebut berlaku seterusnya hingga pada kolom matriks input paling bawah.

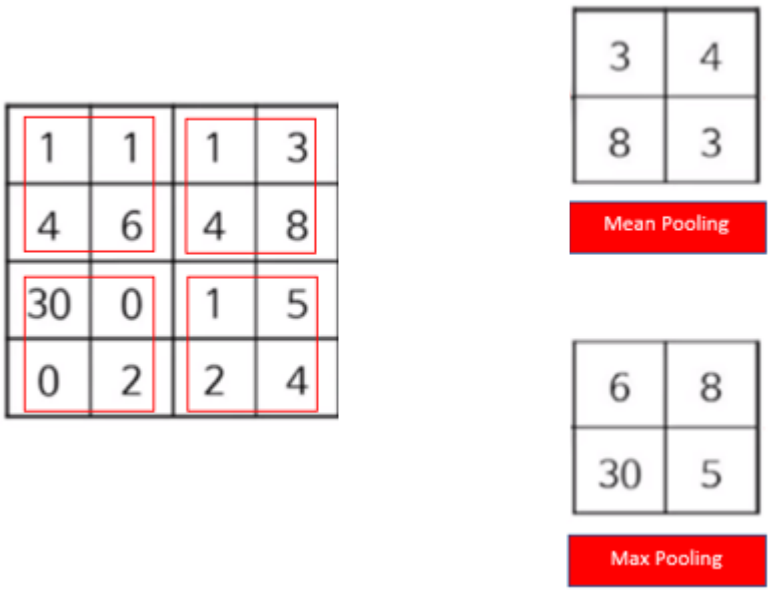
Pada gambar RGB, proses konvolusi dilakukan pada 3 layer dengan metode (*sliding window*) yang sama halnya dengan gambar hitam & putih. Perbedaannya terletak pada *feature map* yang terbentuk. Setiap pergeseran jendela yang dilakukan, nilai setiap layer akan dijumlahkan sehingga didapatkan sebuah nilai baru hasil penjumlahan tersebut. Proses ini dilakukan hingga seluruh bagian pixel dilewati oleh jendela geser seperti pada gambar 2.26 berikut.



Gambar 2. 26 Konvolusi pada gambar RGB (https://medium.com/@samuelsena/pengenalan-deep-learning-part-7-convolutional-neural-network-cnn-b003b477dc94)

2.8.1.3 *Pooling Layer*

Pooling layer merupakan bagian dari *convolution layer* yang berfungsi untuk mengambil nilai tertentu pada sebuah matriks untuk menjadi ukuran matriks yang lebih kecil. Misalkan sebuah matriks dengan ukuran 4x4 dapat diubah menjadi matriks ukuran 2x2 seperti gambar 2.27.



Gambar 2. 27 Mean pooling dan max pooling

Pada gambar 2.27 terdapat matriks hasil dari *mean pooling* dan *max pooling*. *Mean pooling* merupakan penjumlahan dari matriks ukuran tertentu pada matriks asal dan dibagi sesuai ukuran matriks tersebut. Pada kasus contoh gambar 2.27, didapatkan perhitungan awal $\frac{(1+1+4+6)}{12}$= 3. Perhitungan berlanjut hingga matriks 2×2 terpenuhi yaitu dengan nilai 4, 8, dan

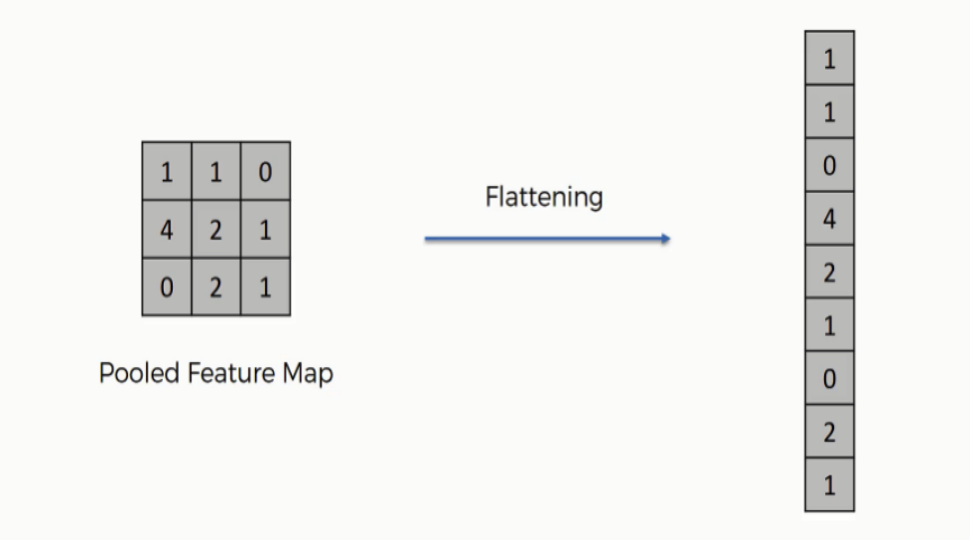
3. Adanya *max pooling* mengakibatkan *feature map* menjadi lebih kecil sehingga perhitungan dijaringan berkurang dan *overfitting* dapat dikendalikan (Suharto et al., 2020). Berbeda dengan *mean pooling, max pooling* merupakan bagian yang mengambil nilai terbesar dari ukuran matriks tertentu pada sebuah matriks induk. Seperti contoh gambar 2.27, pada matriks awal dimana terdapat nilai 1, 1, 4, 6 akan diambil nilai terbesar yaitu nilai 6, begitu pula dengan nilai matriks selanjutnya yaitu 8, 30, dan 5.

2.8.2 *Classification*

Pada bagian ini, matriks hasil *convolution layer* akan diolah dan dianalisis. Hasil analisis tersebut akan menghasilkan *output* yang diinginkan atau yang biasa disebut sebagai model. Adapun bagian dari tahap *classification* adalah *flatten, fully connected layer, dan softmax*.

2.8.2.1 *Flatten*

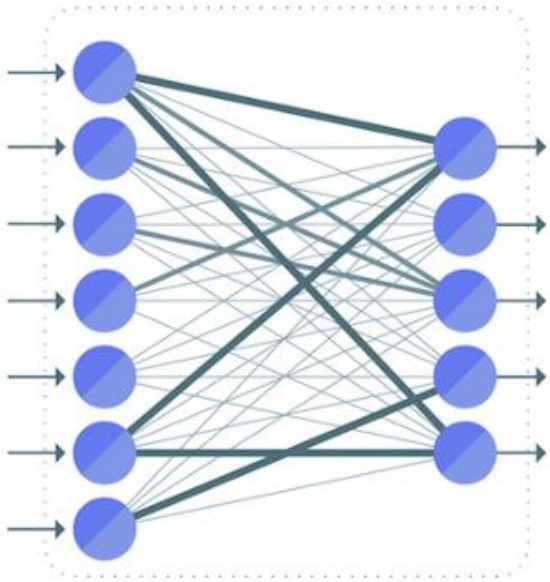
Bagian *Flatten* bertujuan untuk mengubah *feature map* menjadi vector yang dapat diolah oleh *fully connected layer* (MLP). Hal ini dikarenakan hasil *feature map* dari *convolution layer* berbentuk multidimensional array sehingga tidak dapat diolah oleh MLP. Sehingga bentuk matriks output pada proses *pooling layer* akan disubstitusi menjadi matriks dengan bentuk vektor atau matriks dengan jumlah kolom satu seperti pada gambar 2.28.



Gambar 2. 28 Proses flattening pada CNN (https://towardsdatascience.com/)

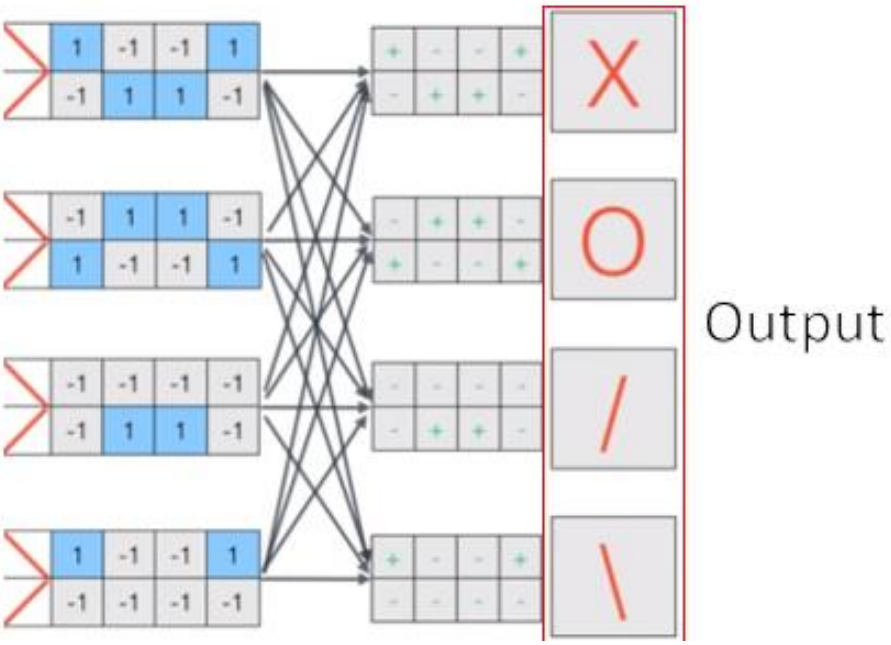
2.8.2.2 *Fully Connected Layer*

Fully Connected Layer merupakan penghubung sebuah neuron pada layer tertentu ke layer berikutnya. Pada bagian ini, hasil olahan *Flatten* akan diolah dan dianalisis seperti pada gambar 2.29. *Output* dari bagian ini merupakan hasil hitungan skor kelas pada setiap matriks hasil *Flatten*.



Gambar 2. 29 Fully connected layer
(Mahavarkar et al., 2020)

Setiap neuron hasil *Flatten* akan dihubungkan dengan neuron hasil sehingga dapat dilakukan analisis terkait kemiripan antar neuron seperti pada gambar 2.30. Hasil dari *Flatten* akan dibandingkan dengan hasil *output* sehingga akan didapatkan nilai kelas terbesar yang merupakan output yang diinginkan.



Gambar 2. 30 Proses analisis pada fully connected layer
(Mahavarkar et al., 2020)

2.8.2.3 *Softmax*

Softmax merupakan sebuah fungsi aktivasi yang berguna untuk menghitung kemungkinan / probabilitas dari setiap kelas pada hasil *fully connected layer*. Artinya dari setiap *output* yang akan dihasilkan akan menghitung probabilitas dari setiap neuron pada *fully*

connected layer yang berhubungan dengan *output* neuron. Rentang nilai probabilitas dari *softmax* sendiri antara 0 hingga 1. Setiap *feature map* hasil *flatten* yang terbentuk akan dibandingkan dengan objek yang akan dideteksi. Adapun formulasi pada *softmax* seperti persamaan 2.10 berikut:

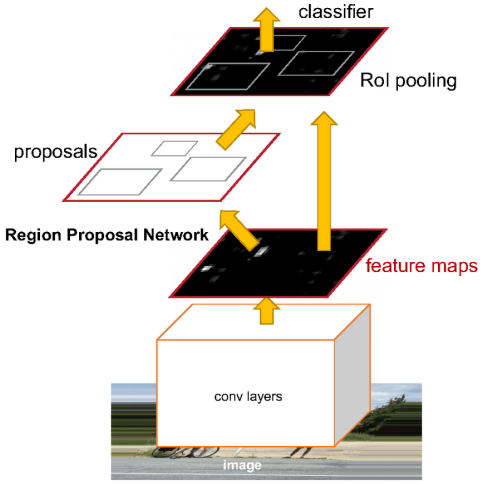
$$\sigma = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad (2.\ 10)$$

2.8.2.4 Loss Function

Loss Function adalah fungsi yang menggambarkan sebuah nilai kerugian *(loss)* pada sebuah model pelatihan *(training)*. *Loss Function* menghitung seberapa jauh nilai prediksi yang dihasilkan model terhadap data pelatihan yang ada. Semakin jauh hasil prediksi dengan data pelatihan yang ada maka semakin tinggi pula *loss* yang dihasilkan, hal ini berlaku sebaliknya jika ingin mendapatkan *loss* yang kecil. Fungsi ini terbentuk pada *fully connected layer* akibat adanya perbedaan antara hasil prediksi dengan label yang dibuat.

2.9 *Faster Region-based Convolutional Neural Network*

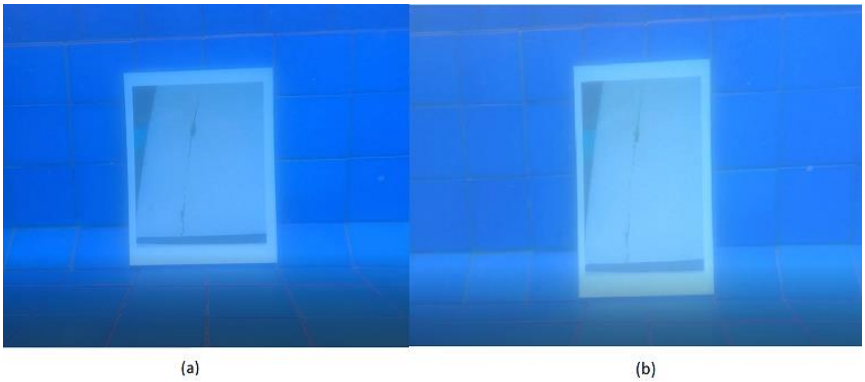
Faster Region-based convolutional neural network (*R-CNN)* merupakan sebuah algoritma *CNN* dengan menggunakan proses pendeteksian area input sesuai yang diingikan. *Faster R-CNN* pertama kali ditemukan oleh Ross Girshick pada tahun 2015 yang semula bernama *R-CNN*. Pada penemuan algoritma baru *R-CNN* ini, terdapat kekurangan dalam penggunaannya. Salah satu kelemahan dari algoritma ini adalah waktu pemrosesan model dan komputasi yang lama. Dari kekurangan tersebut, Ross menemukan algoritma baru yang diberi nama *fast R-CNN* dimana algoritma ini membutuhkan waktu yang lebih cepat dibandingkan dengan *R-CNN*. Namun dalam perjalanannya masih terdapat permasalahan mengenai waktu pemrosesan. Pada *Fast R-CNN* dengan menggunakan *selective search* pada region proposal membutuhkan waktu yang lama untuk *developed* sebuah region baru sehingga dari permasalahan tersebut, Shaoqing Ren yang dibantu oleh Ross Girshick mengembangkan *Fast R-CNN* menjadi *Faster R-CNN* dengan menggunakan region proposal network untuk *men developed* region. Pada bagian prediksi model, *faster R-CNN* terbukti hanya membutuhkan waktu 0,2 detik sedangkan *R-CNN* membutuhkan waktu 40-50 detik dan *fast R-CNN* membutuhkan waktu 2 detik (Al-Azzo et al., 2018). Adapun arsitektur *RCNN* seperti pada gambar 2.31 berikut.



Gambar 2. 31 Arsitektur R-CNN (Al-Azzo et al., 2018)

2.11 Normalisasi Gambar

Setiap gambar terdiri dari tiga perpaduan warna yaitu RGB *(red, green, blue)* dimana ketiga warna tersebut memiliki nilai tertentu untuk mendefinisikan sebuah gambar. Pada gambar 2.32 permukaan air terdapat perbedaan yang terjadi dimana nilai warna merah menjadi turun, sedangkan nilai warna hijau dan biru bertambah (Mahavarkar et al., 2020). Penambahan warna hijau dan biru akan mempengaruhi kualitas gambar yang semakin gelap. Untuk mengatasi permasalahan tersebut dapat dilakukan normalisasi gambar yaitu dengan menambahkan nilai warna merah pada sebuah gambar sehingga didapatkan nilai yang sesuai dengan penambahan nilai hijau dan biru hal ini dapat dilihat pada gambar 2.32. Langkah lain juga dapat dilakukan dengan menambahkan nilai warna merah dan menurunkan nilai warna hijau dan biru sesuai dengan kualitas yang akan dicapai.



Gambar 2. 32 Sebelum normalisasi (a), setelah normalisasi (b)

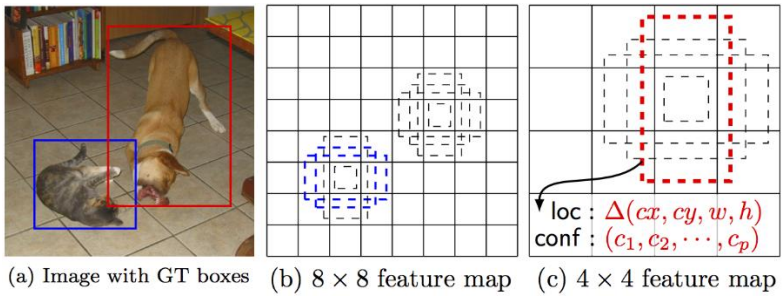
2.12 *Python*

Python merupakan bahasa pemrograman interpretative multiguna. Penggunaannya juga lebih mudah daripada bahasa pemrograman lainnya. Kemudahan dalam membaca kode merupakan hal yang sangat ditekankan pada bahasa pemrograman ini. Tak hanya kemudahan dalam membaca bahasa ini, namun juga memiliki kemudahan dalam menemukan masalah dalam pengkodingan. Bahasa pemrograman yang dirancang oleh Guido van Rossum pada tahun 1991 ini mengutamakan kemudahan penggunaan bagi *expert* maupun pemula dalam bidang pemrograman. Sampai saat ini *python* sudah dikembangkan hingga hampir semua sistem operasi dapat menggunakan bahasa ini (Belajarpython, 2010). Namun dibalik kemudahannya tersebut, bahasa *python* merupakan jenis bahasa tingkat tinggi dimana dalam penggunaannya pada mesin tertentu dibutuhkan proses "pemahaman" terhadap perintah yang terkandung agar dapat digunakan pada sebuah mesin atau yang biasa dikatakan sebagai bahasa mesin.

2.13 *Single Shot Multibox Detector (SSD)*

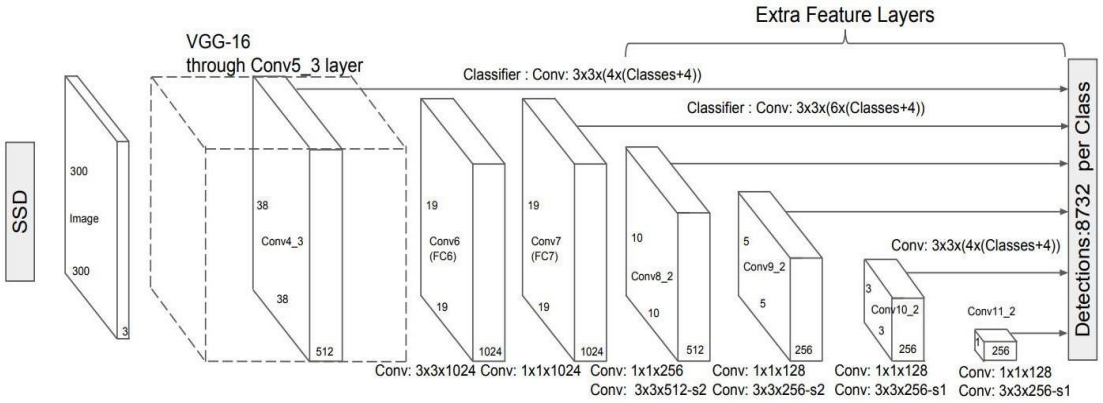
Single shot multibox detector atau lebih dikenal sebagai *SSD* merupakan metode pendeteksian objek dalam sebuah citra dengan menggunakan *single deep neural network.* Diskritisasi ruang *output* dilakukan oleh *SSD* dari kotak pembatas menjadi sebuah set kotak standar pada rasio yang beragam dan skala aspek untuk tiap lokasi *feature map.* Ketika mesin melakukan sebuah prediksi, jaringan akan menghasilkan skor di setiap kotak default untuk lokasi tiap kategori objek dan menghasilkan penyesuaian kotak yang sesuai dengan bentuk objek. Lalu untuk mengatasi perbedaan ukuran tiap objek, jaringan akan menggabungkan hasil prediksi dengan *feature map* dalam berbagai resolusi sehingga permasalahan ukuran dapat teratasi. Eksperimen yang dilakukan Liu menghasilkan *SSD* mempunyai akurasi yang jauh lebih baik dibandingkan dengan metode satu tahap lainnya dan dengan menggunakan ukuran citra yang lebih kecil (Liu,

2016). Adapun *Framework SSD* didefinisikan pada gambar 3.33 sebagai berikut:



Gambar 2. 33 (a) Input Training pada SSD, (b) Visualisasi feature map 8x8,(c) feature map 4x4 (Liu, 2016)

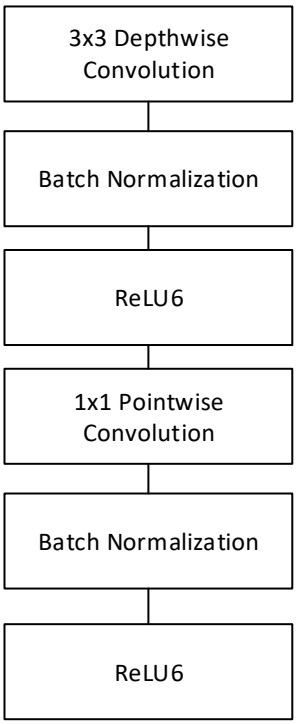
Model yang digunakan *SSD* merupakan VGG-16 sebagai jaringan dasar pembuatannya. VGG atau yang dapat disebut *Visual Geometry Group-16* merupakan arsitektur *CNN* yang memiliki *convolutional filter* pada *convolutional layer* berukuran 3x3. Dengan ukuran *convolutional filter* yang kecil, kedalaman pada *neural network* dapat ditambahkan dengan *convolutional layer* yang lebih banyak. Gambar 2.34 berikut merupakan visualisasi arsitektur *SSD.*



Gambar 2. 34 Arsitektur SSD (Liu, 2016)

2.14 *SSD* Mobilenet V1

SSD Mobilenet V1 merupakan model arsitektur yang memiliki ukuran kecil pada jumlah parameter maupun model yang dihasilkan. Arsitektur ini dikembangkan untuk mengatasi keterbatasan sumber daya perangkat keras dengan mengurangi kompleksitas model dengan menggunakan *depthwise separable convolutions.* Hal ini menyebabkan *SSD* Mobilenet V1 dapat diaplikasikan dalam bentuk *mobile.* Model ini memiliki cara kerja yang hampir sama dengan konvolusi tradisional hanya saja memiliki waktu proses yang lebih cepat dalam komputasi hingga mencapai 9 kali dari konvolusi biasa (Howard et al., 2017). Ciri – ciri arsitektur *Mobilenet V1* pada *depthwise separable convolutions* memiliki urutasn konvolusi layer 3x3 *(depthwise convolution), batch normalization, ReLU6, konvolusi 1x1 (pointwise convolution), batch normalization, ReLU6.* Adapun bentuk *depthwise separable convolution* seperti pada gambar 3.35 berikut.

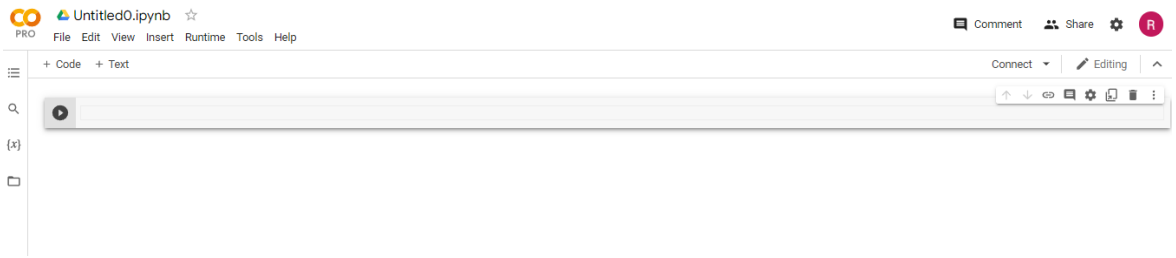


Gambar 2. 35 Depthwise separable convolution block

Gambar 2.35 merupakan model konvolusi yang terjadi pada model *mobilenet V1* yang memungkinkan terjadinya komputasi yang lebih ringan (Howard et al., 2017). Detail arsitektur pada *SSD* Mobilenet V1 dapat dilihat pada Lampiran 1.

2.15 *Google Colaboratory*

Google Colaboratory atau yang biasa disebut dengan Google Colab merupakan layanan yang dikembangkan oleh google dalam bentuk *cloud* untuk menjalankan program dengan fasilitas *processor* dengan spesifikasi yang tinggi (GPU) secara gratis. *Google colab* mempermudah pengguna dalam menjalankan program yang membutuhkan spesifikasi *hardware* yang tinggi secara online. *Software* ini merupakan replikasi dari *software* yang sudah ada yaitu Jupyter Notebook. Penggunaan *google colab* dapat dilakukan dengan menghubungkan pada akun *google drive*, sehingga pengguna dapat menjalankan program yang sudah dibuat secara *online. Google colab* memiliki tampilan *user interface* yang mudah digunakan seperti pada gambar 3.36 berikut:

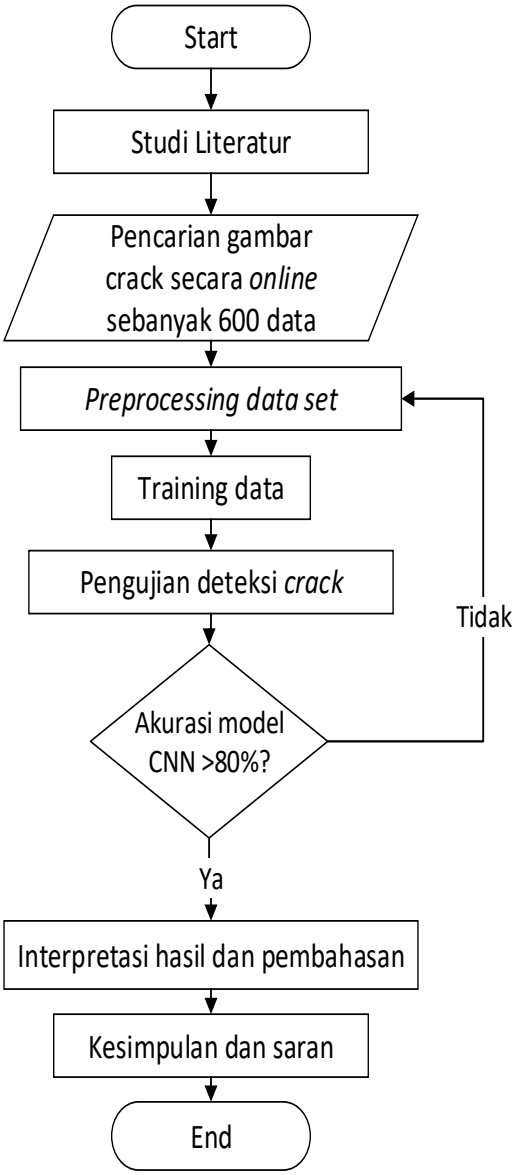


Gambar 2. 36 Tampilan awal google colab

BAB III
METODOLOGI

3.1 Diagram Alir Penelitian

Langkah awal penelitian ini adalah dengan melakukan studi literatur secara mandiri dari literatur terkait penelitian. Setelah itu dilakukan pencarian data berupa gambar *crack* dan data gambar dengan topik acak dari internet. Kemudian sebelum data diolah pada metode *CNN*, dilakukan *preprocessing* data dengan tujuan mempermudah pengolahan data pada algoritma *CNN*. *Training* data kemudian dilakukan dengan menggunakan algoritma *CNN* dan setelah itu dapat dilakukan pengujian pendeteksian *crack*. Langkah terakhir dilakukan pembahasan hasil dan pemberian kesimpulan penelitian. Adapun aktivitas yang dilakukan pada penelitian ini digambarkan pada diagram alir pada gambar 3.1.



Gambar 3. 1 Diagram Alir Penelitian

3.2 Studi Literatur

Langkah awal dalam melakukan penelitian ini adalah melakukan studi literatur secara mandiri. Pada tahap ini dilakukan pencarian data pendukung berupa teori ahli, penelitian terdahulu, dan sebagainya. Data yang telah dikumpulkan digunakan sebagai referensi penulis dalam melakukan eksperimen dan menyusun makalah.

3.3 Alat Uji

Adapun pada penelitian ini terdapat alat uji yang digunakan terdapat pada tabel 3.1 hingga 3.4 sebagai berikut:

1. *Remotely Operated Vehicle*

Tabel 3. 1 BlueROV2

No	Komponen	Spesifikasi
1	Berat	11-12 kg
2	Dimensi	457 x 338 x 254
3	Kedalaman maksimum	100 m
4	*Lighting*	1500 *lumens, angle* 135^{o}
5	*Sensor*	*gyroscope, accelerometer, magnetometer, internal barometer, current and voltage, leak detection*
6	*Chassis*	HDPE panels
7	*Thruster*	Blue Robotics T200

2. Kamera

Tabel 3. 2 GoPro Hero 9

No	Komponen	Spesifikasi
1	Resolusi	20 MP + HDR 5K/30
2	Sensor	23,6 Megapixel
3	100 Mbps Bit Rate	5K/4K/2.7K
4	*Video Stabilization*	*HyperSmooth* 3.0
5	*Battery*	1720mAh *lithium-ion rechargeable battery*
6	*Waterproof*	Hingga ke dalaman 10 meter

3. *Hardware* pengujian

Tabel 3. 3 Asus X555QG series

No	Komponen	Spesifikasi
1	*Processor*	AMD A12-9700P Radeon R7, 10 Compute Cores 4C+6G (4CPUs), ~2,5GHz
2	RAM	8GB DDR 4
3	*Operating system*	Windows 10 Pro 64-bit
4	GPU	AMD Radeon(TM) R7 Graphics

4. *Software*

Tabel 3. 4 Software yang digunakan

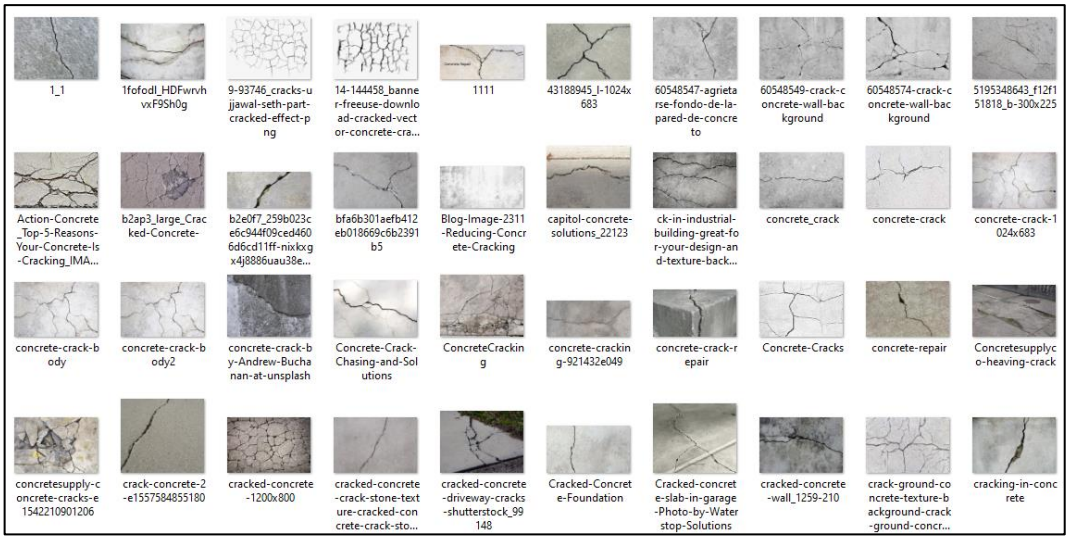
No	Komponen	Spesifikasi
1	*Python*	*Python 3.7* sebagai bahasa pemrograman
2	*Pycharm*	*Integrated development environment* dalam pemrograman

3	*Anaconda*	*Command Prompt* dengan membuat *environment* pemrograman dan tempat penginstalan *library*
4	*Tensorflow*	*Tensorflow V.1.15* sebagai *library* pemrograman
5	*OpenCv*	*OpenCv 4.2* sebagai *library* pemrograman
6	*LabelImg*	*Library* pemrograman pengolah gambar
7	*Google Colab*	*Coding environment* untuk *training data*

5. *Micro SD Card*
6. Monitor eksternal

3.4 Pencarian Data

Pencarian data merupakan tahap pengumpulan citra sebagai dasar input dari metode *CNN*. Proses ini dilakukan dengan mengambil citra dari internet yang menunjukkan sebuah fenomena *crack*. Data yang diambil sebanyak 600 data citra *crack*. Adapun gambar 3.2 merupakan sebagian data yang digunakan pada pengujian.



Gambar 3. 2 Folder yang dibuat untuk program

Pada gambar 3.2 terlihat dari bentuk citra memiliki ukuran file yang berbeda – beda, hal ini dapat menghambat proses *training* jika ukuran file terlalu besar. Maka dari itu data *training* sebelum digunakan harus diolah terlebih dahulu melalui proses *preprocessing data*.

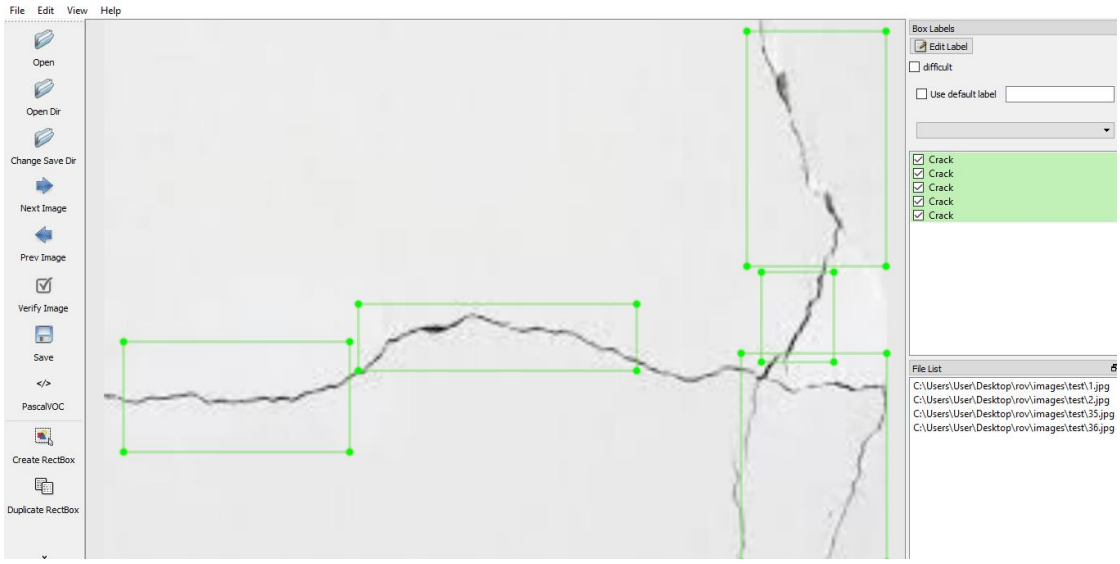
3.5 *Preprocessing Data*

Proses pengecilan *pixel* seperti pada kode berikut. Data citra yang terkumpul diolah pada tahap *preprocessing* data agar dapat mempermudah program dalam melakukan pelatihan dan pengujian. Ukuran data citra diubah menjadi 300 *pixel* x 300 *pixel* untuk setiap data seperti pada gambar 3.3. Hal ini akan mempengaruhi kecepatan *training* oleh program.

```
for img in glob.glob(inputfolder + "/*.jpeg"):
    image = cv2.imread(img)
    imgResized = cv2.resize(image, (300,300))
```

Gambar 3. 3 Kode melakukan resize citra

Kode pada gambar 3.3 digunakan untuk mengecilkan ukuran citra yang digunakan sebagai data *training* (*script* lengkap dapat dilihat pada Lampiran 2). Pada fungsi *for* yang berguna untuk mengidentifikasi data citra dengan jenis jpg file yang kemudian akan dibaca dan dilakukan resized dengan ukuran 300 pixel x 300 pixel melalui fungsi cv2.resize. Citra yang telah melalui tahap resize akan langsung tersimpan pada folder "resized" yang kemudian akan dilanjutkan dengan tahap *labelling*. Pelabelan *crack* dilakukan dengan menggunakan *open source software* yaitu *LabelImg* seperti pada gambar 3.4.



Gambar 3. 4 Pelabelan crack pada citra menggunakan LabelImg

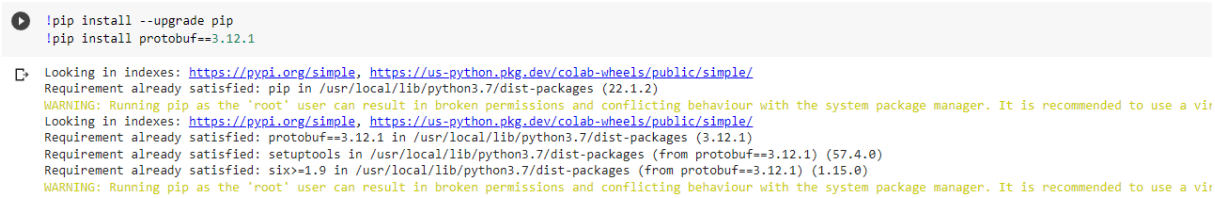
Pelabelan dilakukan dengan memberikan tanda berupa segiempat sebagai tanda *crack* pada citra. Setiap hasil citra yang dilabelkan akan tersimpan dalam file xml. File xml tersebut lalu perlu diubah (*convert*) menjadi file berekstensi csv dengan menggunakan *script* xml_to_csv (*script* lengkap dapat dilihat pada Lampiran 3). Lalu data hasil transformasi tersebut akan direkam dengan format TFRecord menggunakan *scipt* generate_tfrecord (*script* lengkap dapat dilihat pada Lampiran 4). Fungsi dari format TFRecord berguna untuk meringankan ukuran file data yang telah diklasifikasi. Lalu seluruh data citra beserta file csvnya dibagi menjadi dua kelompok yaitu data *training* dan data *testing*. Pembagian dilakukan sebanyak 90% data *training* dan 10% data *testing* dari total seluruh data citra yang digunakan. Setelah kedua tahap tersebut dilakukan, selanjutnya dilakukan konfigurasi Label Map yang berguna untuk pemetaan label yang nantinya akan digunakan untuk pemberian nama objek yang dideteksi. Konfigurasi label map yang dilakukan seperti pada gambar 3.5 dengan format file berekstensi pbtxt.

```
item {
        id: 1
        name: 'Crack'
}
```

Gambar 3. 5 Konfigurasi label map

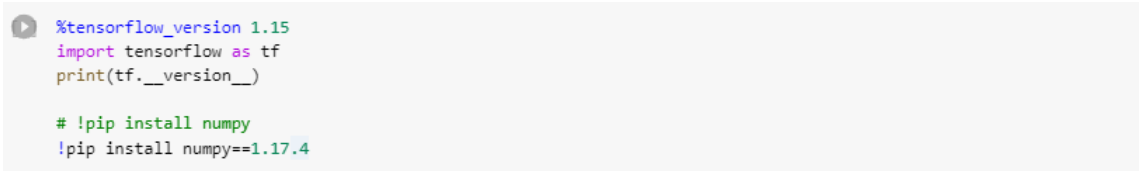
3.5 *Training* Data

Data yang telah melalui proses *preprocessing* kemudian dilakukan pengolahan data lanjut dengan menggunakan arsitektur *pretrained model Mobilenet v1 coco*. Proses *training* data merupakan proses pembelajaran mesin dengan data awal sebagai data pembelajarannya. Data awal yang digunakan sebanyak 600 data citra *crack* yang didapatkan dari internet yang telah dilakukan *preprocessing*. Training pada penelitian ini dilakukan pada *Google Colab*. Langkah awal memastikan seluruh *library* yang digunakan telah terinstalasi ataupun telah ditentukan seperti pada gambar 3.6 berikut. Pada *training* digunakan Protocol Buffers 3.12.1 sesuai dengan kebutuhan *training*.



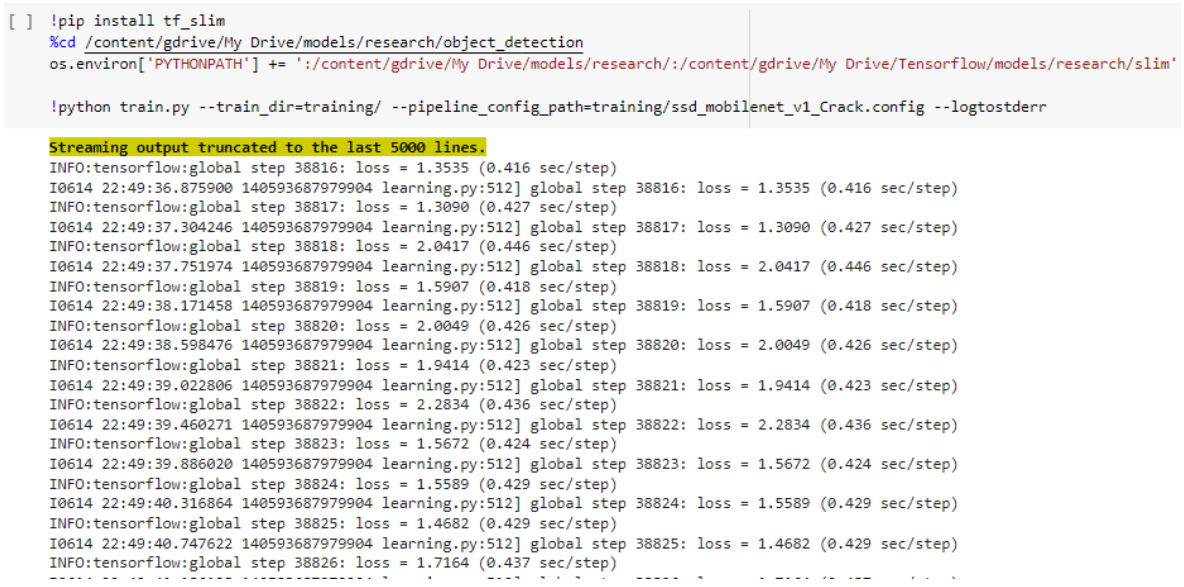
Gambar 3. 6 Instalasi pip dan protbuf

Digunakan tensorflow 1.15 yang sesuai dengan kebutuhan training dan numpy versi 1.17.4. Pemilihan tensorflow 1.15 pada gambar 3.7 dikarenakan model *pretrained* yang digunakan hanya dapat digunakan dengan tensorflow versi 1.x.



Gambar 3. 7 Instalasi tensorflow dan numpy

Setelah persiapan *library* selesai, training dapat dilakukan dengan menggunakan perintah pada gambar 3.8 berikut.



Gambar 3. 8 Training menggunakan google colab

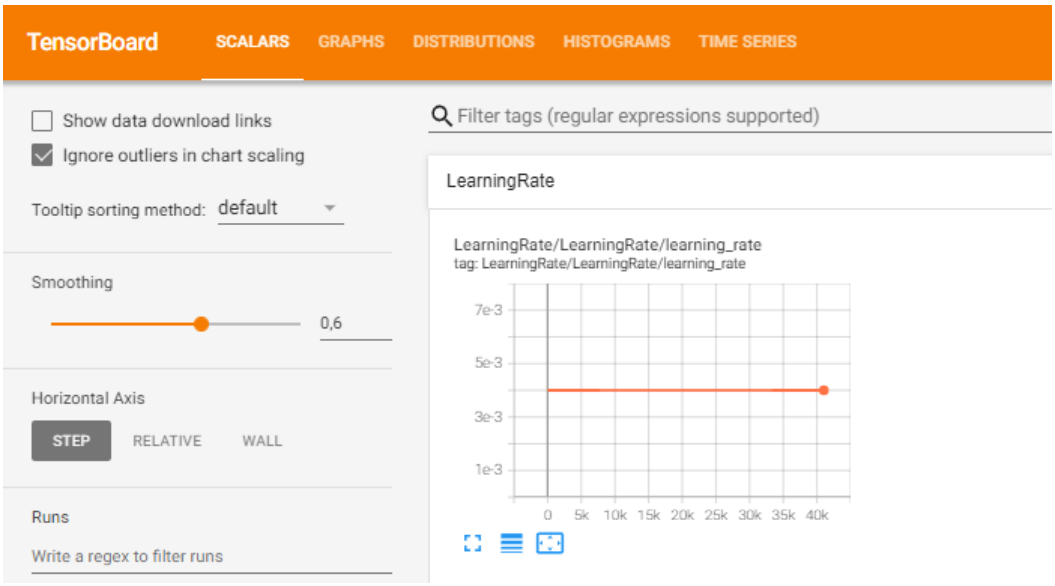
Gambar 3.8 merupakan perintah pada *google colab* untuk memulai proses *training data* dengan memanggil *script* pada *train.py* (*script* dapat dilihat pada Lampiran 5). Pada perintah *train_dir=training/* melakukan penyimpanan hasil model *training* pada folder *train.* Ketika perintah pada gambar 3.8 dijalankan, secara otomatis proses *training* akan berjalan dengan step awal yang dimulai dari 0. Proses *training* dilakukan sebanyak 40k kali dan *batch size* sebesar 24 dengan menggunakan model *checkpoint* yang berarti model terbaik yang akan tersimpan pada folder training. Pengaturan parameter *training* selengkapnya dapat dilihat pada Lampiran 6.

Ketika proses *training* telah dilakukan dan terdapat beberapa file *checkpoint* yang pada folder *training*, selanjutnya dapat dilakukan pengecekan hasil *training* dan export *model* dari model *training* terbaik. Pengecekan hasil *training* dengan menggunakan *Tensorboard* yang merupakan salah satu modul yang tersedia pada T*ensorflow.* Pada gambar 3.9, tensorboard akan membaca hasil riwayat *training* yang dilakukan dengan menghubungkan pada folder *google drive* yang telah disediakan.

```
from google.colab import drive
drive.mount('/content/gdrive')

!pip install tensorboardcolab
%cd /content/gdrive/My Drive/models/research/object_detection
%load_ext tensorboard

%tensorboard --logdir=training/
```

Gambar 3. 9 Perintah menjalankan tensorboard

Setelah perintah dijalankan maka *tensorboard* akan muncul dan menghasilkan grafik hasil *training* yang telah dilakukan. Tampilan awal pada *tensorboard* dapat dilihat pada gambar 3.10 berikut.



Gambar 3. 10 Tampilan awal Tensorboard

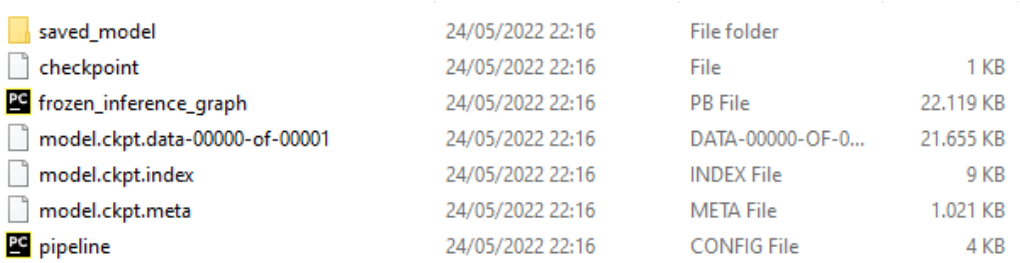
Grafik yang tertera pada *tensorboard* merupakan hasil dari *training* yang sudah dilakukan seperti contohnya grafik loss. Hasil *training* terbaik ini dapat dilihat dari hasil nilai

loss terendah pada model *checkpoint* yang ada. Adapun perintah yang dijalankan pada proses *export* ini seperti pada gambar 3.11 berikut.

```
%cd /content/gdrive/My Drive/models/research/object_detection
!python export_inference_graph.py --input_type image_tensor --pipeline_config_path training/ssd_mobilenet_v1_Crack.config
--trained_checkpoint_prefix training/model.ckpt-25990 --output_directory new_graph
```

Gambar 3. 11 Perintah export model terbaik

Pada gambar 3.11 merupakan perintah untuk menjalankan / memanggil *script* pada *export_inference_graph.py* (*script* dapat dilihat pada Lampiran 7) yang berguna mengeksekusi model *checkpoint* pada folder *training* menjadi folder file dengan ekstensi *protobuf* (.pb). setelah dilakukan *export* file, model dapat digunakan untuk pengujian deteksi *crack*. Model yang terbentuk seperti pada gambar 3.12.

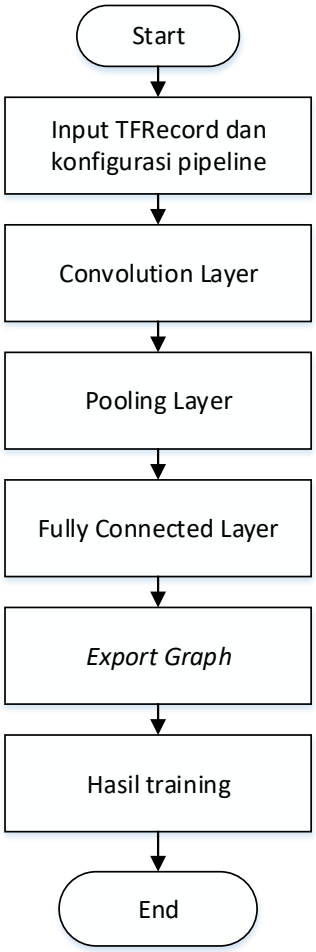


Gambar 3. 12 Model hasil training

Adapun langkah pada proses *training* sebagai berikut:

a. Data inputan berupa *TFRecord file* akan dijadikan input data pada proses *training* dengan komposisi 90% total gambar yang dimiliki.
b. Tahap selanjutnya merupakan tahap *convolution.* Pada tahap ini inputan akan melewati *convolution layer* yang akan dilakukan proses konvolusi pada data input sehingga input menjadi sebuah matriks dengan ukuran tertentu. Proses konvolusi yang terjadi menggunakan filter/kernel dengan nilai acak. Sedangkan ukuran filter menyesuaikan dengan arsitektur yang digunakan. Hasil dari proses konvolusi dinamakan *feature* map. Setelah itu terdapat fungsi aktivasi *Rectifier Linier Unit (ReLu)* yang bertujuan untuk mengubah nilai negatif menjadi nilai nol akibat proses konvolusi.
c. Kemudian akan dilanjutkan dengan proses *Pooling* yang bertujuan untuk mengurangi dimensi dari *feature map* yang dihasilkan dari proses *convolution layer.* Adapun *pooling* yang digunakan pada penelitian ini adalah *max pooling*. *Max Pooling* akan mengurangi ukuran dari *feature map* sehingga dapat mengurangi terjadinya *overfitting* pada hasil.
d. Lalu pada tahap selanjutnya dilakukan *flattening* data pada *feature map* yang terbentuk. Tahap ini akan mengubah *feature map* menjadi bentuk *vector* agar data tersebut dapat diolah pada tahap *fully connected layer.*
e. Ketika *feature map* telah diolah pada *fully connected layer* dengan bantuan fungsi aktivasi *softmax* dalam menentukan nilai perbandingan pada objek, maka akan terbentuk model dari *crack* yang telah *di* training (proses *training* selesai). Pada langkah ini dilakukan *export graph* sehingga model training tersebut dapat digunakan untuk pendeteksian *crack.*

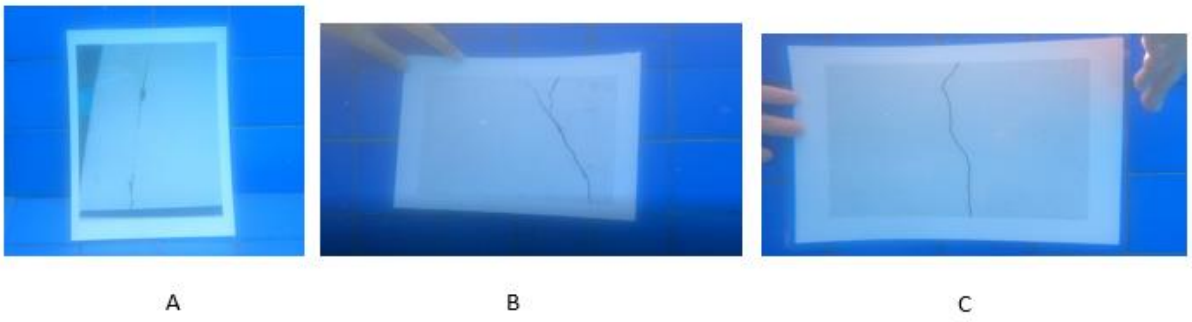
Adapun proses di atas digambarkan dengan diagram alir pada gambar 3.13 sebagai berikut:



Gambar 3. 13 Diagram alir proses training

3.6 Pengujian Deteksi *Crack*

Pengujian deteksi *crack* dilakukan setelah proses *training* selesai. Pengujian dilakukan dengan menangkap citra kondisi bawah permukaan air dengan menggunakan kamera GoPro Hero 9 yang berlokasi pada kolam renang Wisata Bukit Mas Surabaya pada pukul 09.00 – 10.00 WIB. Terdapat tiga citra yang digunakan sebagai objek deteksi seperti pada gambar 3.14.



Gambar 3. 14 Tiga objek yang dideteksi, citra 1 (A), citra 2 (B), dan citra 3 (C)

Hasil tangkapan citra berupa video kemudian diolah dengan menggunakan *script* Deteksi_Crack.py yang terhubung dengan data hasil. Tangkapan citra akan diproses oleh

program sehingga ketika terdapat *crack* pada tangkapan citra maka program akan memberi tanda kotak dan persentase pada *crack* yang terdeteksi. Warna yang ditampilkan pada video kemudian di normalisasi melalui kode pada program sehingga memiliki kualitas warna seperti pada di luar permukaan air (script dapat dilihat pada Lampiran 8). Citra yang digunakan pada penelitian ini ketika berada dalam permukaan air sebagai berikut.

3.7 Interpretasi Hasil dan Pembahasan

Setelah mendapatkan hasil pengujian, langkah selanjutnya hasil pengujian akan diolah dengan beberapa variasi uji. Variasi uji yang digunakan antara lain, jumlah data training, kedalaman, dan jarak. Parameter yang digunakan dapat dilihat pada tabel 3.5. Analisis hasil dengan menggunakan tiga variasi uji kemudian akan dilakukan pembahasan.

Tabel 3. 5 Variasi pengujian

Kombinasi ke-	Variasi pengujian		
	Data Training	Kedalaman	Jarak
1	400 Citra	0.5 m	0.5 m
2		0.5 m	0.75 m
3		0.5 m	1 m
4		1 m	0.5 m
5		1 m	0.75 m
6		1 m	1 m
7		1.5 m	0.5 m
8		1.5 m	0.75 m
9		1.5 m	1 m
10	500 Citra	0.5 m	0.5 m
11		0.5 m	0.75 m
12		0.5 m	1 m
13		1 m	0.5 m
14		1 m	0.75 m
15		1 m	1 m
16		1.5 m	0.5 m
17		1.5 m	0.75 m
18		1.5 m	1 m
19	600 Citra	0.5 m	0.5 m
20		0.5 m	0.75 m
21		0.5 m	1 m
22		1 m	0.5 m
23		1 m	0.75 m
24		1 m	1 m
25		1.5 m	0.5 m
26		1.5 m	0.75 m
27		1.5 m	1 m

Setiap gambar akan dilakukan pengujian (pengambilan video gambar) sebanyak sembilan kali variasi uji. Terdapat tiga variasi kedalaman yang digunakan dan setiap variasi kedalaman terdapat tiga variasi jarak yang akan dilakukan. Pada variasi 1–3 memiliki kedalaman yang sama sedangkan terdapat perbedaan jarak untuk tiap variasinya, hal ini juga berlaku pada variasi 4-6, dan variasi 7-9. Sedangkan pada variasi 1, 4, dan 7 memiliki variasi

jarak yang sama namun terdapat perbedaan variasi pada nilai kedalamannya, hal ini juga berlaku pada variasi 2, 5, dan 8, dan variasi 3, 6, dan 9. Adapun kesembilan variasi tersebut dapat dilihat pada tabel 3.6 berikut:

Tabel 3. 6 Variasi tiap pengujian gambar

Variasi	Variasi pengujian	
	Kedalaman	Jarak
1	0.5 m	0.5 m
2		0.75 m
3		1 m
4	1 m	0.5 m
5		0.75 m
6		1 m
7	1.5 m	0.5 m
8		0.75 m
9		1 m

Data pengujian yang didapatkan berupa akurasi dengan *range* 1% hingga 100%. Persentase nilai akurasi yang diambil merupakan nilai akurasi terbesar yang mampu didapatkan model ketika mendeteksi objek crack hal ini akan disebut sebagai nilai deteksi. Sedangkan kesalahan pengujian akan disebut dengan *error.* Nilai *error* yang didapatkan juga merupakan nilai persentase terbesar *error* yang terjadi. Setelah didapatkan hasil uji yang dilakukan berupa persentase, data kemudian akan diolah dengan metode perhitungan *confusion matrix*, dimana setiap kegagalan deteksi ataupun keberhasilan deteksi akan menentukan nilai akurasi model *CNN*. *Confusion matrix* digunakan untuk menentukan akurasi model *CNN*. Pada tabel 3.7, setiap keberhasilan dan kegagalan uji pada model dikategorikan menjadi empat bagian, yaitu *true positive (TP), true negative (TN), false negative (FN),* dan *false positive (FP)* sesuai penjelasan pada bab 2.8. Pendeteksian TP jika model mampu mendeteksi *crack* dengan akurasi hingga sebesar 80%. Jika pendeteksian kurang dari 80%, maka penilaian *confusion matrix* akan bertambah pada kategori FN. Nilai aktual merupakan perhitungan model dalam membaca data gambar *testing* sedangkan pada prediksi didapatkan dari data *training* yang dilakukan.

Tabel 3. 7 Confusion matrix

		Aktual	
		Positive	Negative
Prediksi	Positive	TP	FP
	Negative	FN	TN

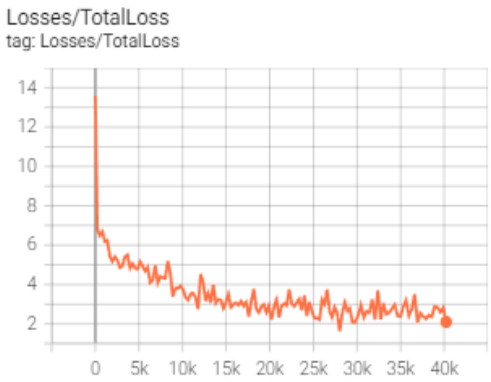
3.8 Kesimpulan dan Saran

Pada tahap terakhir dilakukan penarikan kesimpulan sesuai dengan hasil yang didapat dan pemberian saran atas hasil penelitian untuk membantu penelitian yang akan dilakukan selanjutnya.

BAB IV
HASIL DAN PEMBAHASAN

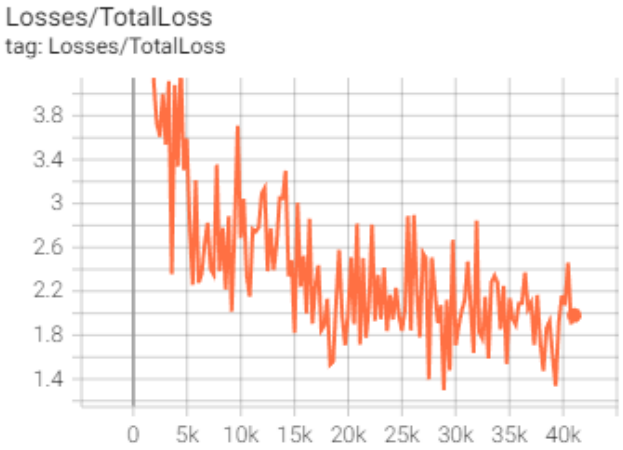
4.1 Model Hasil *Training*

Dari hasil *training* yang telah dilakukan sebanyak 40,000 step dengan *batch size* sebesar 24, dihasilkan grafik total *loss* yang terjadi pada tiga jenis variasi *training* data 400, 500, dan 600 data pembelajaran. Total *loss* merupakan riwayat *loss* atau *error* yang terjadi pada tiap *batch size* selama *training* dilakukan. Total *loss* pada *training* 400 data dapat dilihat pada gambar 4.1 berikut.



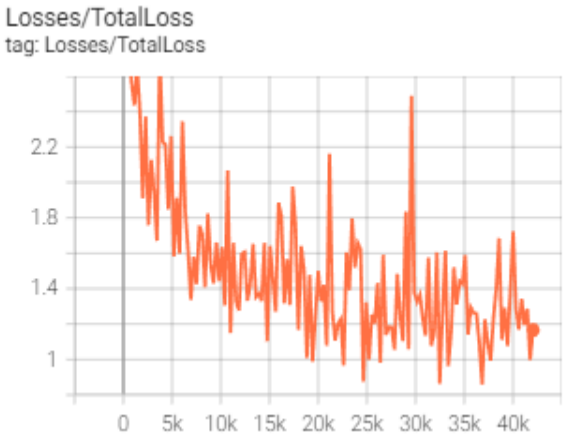
Gambar 4. 1 Grafik total loss variasi 400 data

Pada gambar 4.1 merupakan hasil visualisasi total *loss* selama *training* pada 400 data. Dapat dilihat nilai *loss* didapatkan sangat tinggi pada awal *training* hingga mencapai nilai 13%. Hal ini berlangsung pada awal *training* dan berangsur turun bersama dengan bertambahnya *global step*. Sehingga pada hasil grafik didapatkan penurunan nilai *loss* terendah pada nilai 2,145% pada *global step* ke 39144. *Training loss* memiliki perubahan *range loss* yang terjadi antara 1,5% hingga 4% dimulai pada *step* 10k hingga step 40k. Pada variasi 500 data *training* didapatkan hasil grafik yang berbeda. Variasi *training* 500 data seperti yang ditampilkan pada gambar 4.2, dapat terlihat perbedaan yang cukup signifikan jika dibandingkan dengan hasil grafik total *loss* pada variasi 400 data *training.* Pada step 5k hingga 15k terlihat *loss* mencapai *range* nilai 3,6% hingga 1,8% dan pada step 15k hingga 40k memiliki *range* nilai *loss* antara 3% hingga 1,3%. *Step* terendah pada variasi 500 didapatkan pada *step* 39310 dengan nilai *loss* sebesar 1,337%.



Gambar 4. 2 Grafik total loss variasi 500 data

Sedangkan pada grafik total *loss* yang didapat pada variasi 600 dataset didapatkan visualisasi grafik pada gambar 4.3. Grafik total *loss* pada gambar 4.3 memiliki perbedaan nilai *loss* yang lebih signifikan dibandingkan grafik total *loss* pada variasi data 400 dan 500. Terlihat pada *step* 30k, nilai *loss* yang terjadi mencapai 2,4% dari *range step* 5k hingga 40k sedangkan nilai *loss* terendah yang didapatkan sebesar 0,861% tercatat pada *step* 36931.

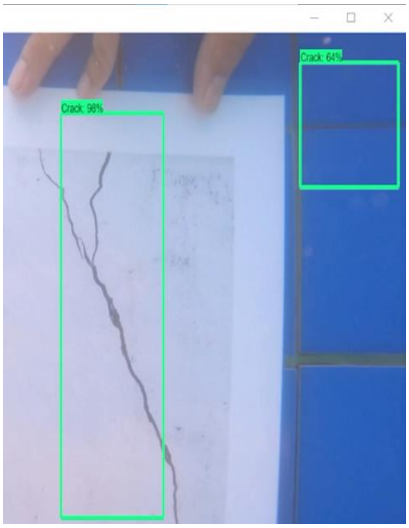


Gambar 4. 3 Grafik total loss variasi 600 data

Dari ketiga variasi yang didapat terlihat perbedaan yang signifikan dimana pada variasi 400 data memiliki fluktuasi grafik nilai *loss* yang lebih teratur dibandingkan pada grafik nilai *loss* pada variasi 500 dan 600. Sedangkan pada grafik total *loss* pada variasi 500 memiliki fluktuasi grafik nilai *loss* yang lebih teratur dibandingkan dengan grafik variasi total *loss* variasi 600 data. Hal ini dikarenakan semakin banyaknya data *training* akan memungkinkan terjadinya *error* pembelajaran yang besar akibat variasi data gambar yang berbeda – beda. Namun jika dilihat dari nilai *loss* terendah, pada variasi 600 memiliki nilai *loss* paling rendah diantara nilai *loss* terendah dari masing – masing variasi. Semakin kecil nilai *loss* yang dihasilkan maka model dapat mendeteksi objek dengan baik. Dari ketiga gambar grafik total *loss*, *trend* yang terjadi yaitu semakin bertambah jumlah *step* maka nilai *loss* akan berangsur mengecil. Hal ini akan terjadi karena setiap step yang dilakukan, terdapat *weight* dan *bias* yang akan selalu berubah. Semakin banyak data gambar serupa yang dipelajari *(training* data *crack)* akan mengakibatkan perubahan nilai *weight* dan *bias* yang semakin kecil jika dibandingkan dengan perubahan nilai pada awal *step.* Perubahan *weight* dan *bias* ini yang akan menghasilkan nilai *loss* pada hasil *training.* Sehingga seiring bertambahnya *step training,* nilai *loss* yang terjadi akan semakin mengecil.

4.2 Hasil Deteksi Objek

Model yang sudah didapatkan dari tiga macam variasi jumlah data *training* kemudian diuji pada tangkapan citra berupa video yang telah dijelaskan pada bab 3.6. Video yang diuji sebanyak tiga buah sesuai dengan jumlah objek citra *crack* yang digunakan dengan ketentuan pengambilan citra berdasarkan ketentuan variasi pada bab 3.7. Adapun hasil pengujian yang didapatkan seperti pada gambar 4.4 berikut.



Gambar 4. 4 Hasil pengujian true positive dan false positive

Terdapat 2 buah kotak berwarna hijau yang muncul pada hasil deteksi gambar 4.4. Dari kedua pengujian terebut dapat diartikan 2 hal yaitu, pada pengujian *crack* dengan persentase 96%, model mampu mendeteksi objek hal ini dapat diartikan bahwa model melakukan *true positive* pada pengujian *crack.* Sedangkan pada persentase 64%, model melakukan *false positive* yang berarti model salah mendefinisakan *crack* pada kasus gambar 4.4. Hal ini terlihat pada gambar bahwa model mendeteksi *crack* pada bagian nat keramik. Kegagalan model pada kriteria *false positive* yang selanjutnya pada hasil pengujian disebut sebagai *error*. Selanjutnya pada gambar 4.5 merupakan hasil pengujian yang menunjukkan kriteria *true negative* dan *false negative*.

Gambar 4. 5 Hasil pengujian true negative dan false negative

Pengujian pada gambar 4.5 menunjukkan bahwa terdapat 2 buah kriteria penilaian yang ada pada *confusion matrix.* Model tidak mendeteksi objek *crack* yang berarti terdapat *error* yang terjadi pada model sehingga objek *crack* tidak terdeteksi, hal ini dinamakan sebagai *false positive.* Sedangkan dari beberapa percobaan yang dilakukan, pada gambar 4.5 model tidak mengidentifikasi nat pada keramik ataupun objek selain *crack* sebagai *crack*, maka dapat dikatakan bahwa model melakukan *true negative* sesuai dengan kriteria yang ada pada *confusion matrix*. Adapun data hasil pengujian lain yang didapatkan sebagai berikut.

4.2.1 Hasil Deteksi Gambar 1

Pada variasi 400 data, dapat dilihat bahwa model dapat mendeteksi objek sebanyak lima kali dari total sembilan variasi. Namun pada model ini terdapat error deteksi yang lebih banyak jika dibandingkan dengan model variasi data *training* lainnya. Dari sembilan variasi kedalaman

dan jarak yang dilakukan, terdapat enam kali pengujian model mendeteksi *false positive* pada citra yang ditampilkan. Seperti pada tabel 4.1, terdapat model melakukan *false positive* dengan akurasi 100% yang merupakan error terbesar pada variasi kedalaman 1 meter dan jarak 0,75 meter. Sedangkan pada hasil deteksi model variasi *training* 500 ini memiliki nilai akurasi deteksi tertinggi pada kedalaman dan jarak 0,5 meter sebesar 98%.

Data deteksi pada tabel 4.1, menunjukkan nilai akurasi tertinggi yang menurun pada pengujian dengan variasi kedalaman 0,5 meter. Pada jarak 0,5 meter model memiliki nilai akurasi tertinggi deteksi sebesar 98% dan pada jarak 0,75 meter model hanya mampu mendapatkan nilai 60%. Hal ini juga terjadi pada kedalaman 1 meter, dimana jarak 0,5 meter akurasi mencapai nilai 97% dan pada jarak 0,75 mengalami penurunan nilai akurasi tertinggi menjadi 87%.

Tabel 4. 1 Data variasi 400 data

Kedalaman (m)	Jarak (m)	Deteksi	Error
0,5	0,5	98%	90%
	0,75	60%	88%
	1		
1	0,5	97%	
	0,75	87%	100%
	1		87%
1,5	0,5		68%
	0,75	80%	67%
	1		

Pada tabel 4.2, model dengan data *training* 500 hanya mampu mendeteksi objek *crack* sebanyak tiga variasi, yaitu pada kedalaman dan jarak 0,5 meter model mampu mendeteksi citra 1 dengan akurasi tertinggi sebesar 98%, kemudian pada kedalaman 0,5 meter dengan jarak 0,75 meter, model mampu mendeteksi objek *crack* dengan akurasi tertinggi sebesar 96%. Model juga dapat mendeteksi *crack* pada kedalaman 1 meter dan jarak 0,5 meter pada citra sebesar 82%. Terdapat error yang terjadi ketika model mendeteksi objek pada kedalaman 1,5 meter dan jarak 0,75 meter, dimana model mendeteksi nat keramik sebagai *crack* dengan persentase 61%. Sedangkan pada model ini masih belum bisa mendeteksi *crack* 6 variasi jarak dan kedalaman lainnya yaitu pada kedalaman 0,5 meter dan jarak 1 meter, kedalaman 1 meter dan jarak 0,75 meter dan 1 meter, dan pada kedalaman 1,5 meter.

Persentase model dalam mendeteksi objek *crack* dengan kedalaman 0,5 meter memiliki penurunan bersamaan dengan pertambahan jarak pengambilan citra yaitu pada jarak 0,5 meter dan jarak 0,75 meter. Sedangkan pada pengujian kedalaman 0,5 meter dan 1 meter juga mengalami penurunan deteksi *crack* dari persentase maksimal 98% menjadi 82%.

Tabel 4. 2 Data variasi 500 data

Kedalaman (m)	Jarak (m)	Deteksi	Error
0,5	0,5	98%	
	0,75	96%	
	1		
1	0,5	82%	
	0,75		
	1		
1,5	0,5		
	0,75		61%
	1		

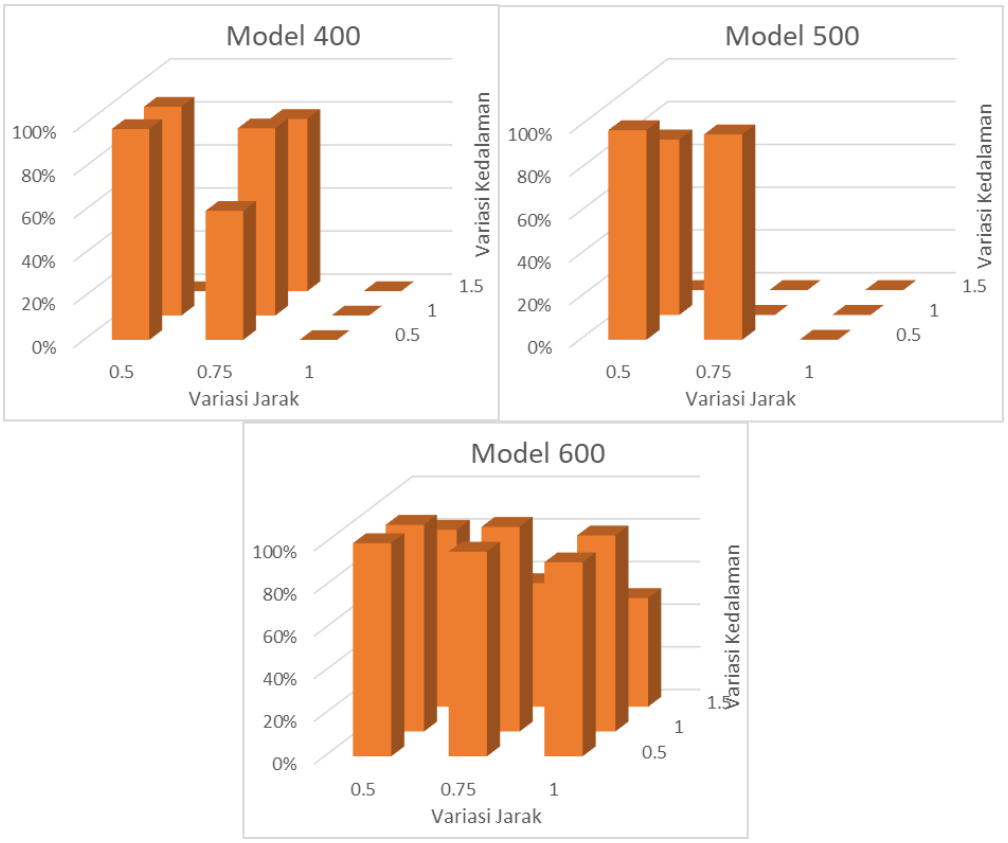
Pada tabel 4.3 menunjukkan hasil pengujian pada variasi menggunakan 600 data *training*. Dari data yang didapatkan terdapat perbedaan yang signifikan jika dibandingkan dengan pengujian dengan variasi data *training* 400 dan 500. Dengan variasi 600 data *training* didapatkan model yang lebih baik dibandingkan kedua variasi jumlah data *training* lainnya karena model dapat mendeteksi objek *crack* dengan benar pada setiap variasi yang diberikan (kedalaman dan jarak). Nilai persentase akurasi tertinggi didapat pada kedalaman 0,5 meter dengan jarak 0,5 meter dengan nilai akurasi 100% sedangkan nilai terendah didapatkan pada kedalaman 1,5 meter dan jarak 1 meter dengan nilai akurasi 51%. Pada variasi ini juga terdapat error yang terjadi yaitu pada variasi kedalaman dan jarak 0,5 meter dan pada variasi kedalaman 1 meter dan jarak 0,75 meter secara berturut – turut 57% dan 60%.

Data yang didapatkan memiliki memiliki nilai akurasi yang menurun seiring bertambahnya jarak dan kedalaman pada variasi. Model mampu mendeteksi kedalaman 0,5 dengan nilai akurasi tertinggi dari pengujian sebesar 100% dan berangsur turun hingga 91% dengan bertambahnya jarak. Hal serupa terjadi pada deteksi dengan variasi kedalaman 1 meter dan 1,5 meter. Nilai persentase semakin turun dengan bertambahnya jarak pada setiap variasi kedalaman. Jika dilihat pada variasi jarak, hasil pengujian model juga menunjukkan *trend* yang menurun seiring bertambahnya variasi kedalaman.

Tabel 4. 3 Data variasi 600 data

Kedalaman (m)	Jarak (m)	Deteksi	Error
0,5	0,5	100%	57%
	0,75	96%	
	1	91%	
1	0,5	97%	
	0,75	96%	60%
	1	92%	
1,5	0,5	83%	
	0,75	58%	
	1	51%	

Pada gambar 4.6 menunjukkan perbandingan yang terjadi hasil uji deteksi crack dengan menggunakan variasi jumlah data training, kedalaman pengambilan data citra, dan jarak pengambilan data citra pada gambar 1. Pada variasi 400 dan 500 terlihat beberapa kali grafik menyentuh pada nilai 0% yang menandakan model tidak mampu mendeteksi objek crack pada video uji. Sedangkan pada grafik variasi 600 terlihat trend berangsur turun meskipun terdapat kenaikan nilai / persentase akurasi pada titik variasi tiga dan empat dan pada variasi 6 dan 7. Berdasarkan grafik pada gambar 4.6, dapat dilihat variasi 600 data training memiliki nilai pendeteksian terbaik jika dibandingkan dengan variasi 400 dan 500 data training pada pendeteksian gambar 1.



Gambar 4. 6 Grafik perbandingan uji model pada gambar 1

4.2.2 Hasil Deteksi Gambar 2

Hasil deteksi pada variasi 400 data *training* didapatkan data seperti pada tabel 4.4 Model mampu mendeteksi objek dengan akurasi tertinggi 100% pada variasi jarak dan kedalaman 0,5 meter, kedalaman 1 meter dengan jarak 0,5 meter, dan pada kedalaman 1,5 meter dengan variasi jarak 0,5 meter dan 0,75 meter. Sedangkan nilai akurasi terendah pada variasi pengujian didapatkan nilai sebesar 96% yang terjadi pada variasi kedalaman dan jarak 1 meter. Terlihat pada jumlah eror yang dialami pada model dengan variasi data 400 sebanyak 3 kali yaitu pada variasi kedalaman dan jarak 0,5 meter dan 1 meter, lalu pada variasi kedalaman 1,5 meter dan jarak 0,75 meter. Model juga tidak mampu mendeteksi *crack* pada kedalaman 0,5 meter dengan jarak 1 meter dan pada variasi kedalaman 1,5 meter dengan jarak 1 meter.

Penurunan persentase terjadi pada perbandingan kedalaman 0,5 meter dan kedalaman 1 meter. Pada kedalaman 0,5 meter pada jarak 0,5 meter dan 0,75 meter terjadi penurunan akurasi tertinggi pendeteksian *crack* sebesar 2% dari 100% pada jarak 0,5 meter menjadi 98% pada jarak 0,75 meter. Sedangkan pada kedalaman 1 meter terdapat penurunan akurasi persentase tertinggi pada 3 variasi jaraknya secara beturut – turut 100%, 98%, dan 96%.

Tabel 4. 4 Data variasi 400 data

Kedalaman (m)	Jarak (m)	Deteksi	Error
0,5	0,5	100%	64%
	0,75	98%	
	1		
1	0,5	100%	83%
	0,75	98%	
	1	96%	
1,5	0,5	100%	
	0,75	100%	75%
	1		

Pada tabel 4.5 menunjukkan hasil pengujian dengan variasi *training* 500 data *crack*. Nilai persentase akurasi tertinggi yang didapatkan oleh model sebesar 100% pada variasi kedalaman 0,5 meter dengan jarak 0,5 meter dan 0,75 meter, kedalaman 1 meter dengan jarak 0,5 meter, dan kedalaman 1,5 meter dengan jarak 0,5 meter dan 0,75 meter. Sedangkan nilai akurasi terendah sebesar 69% yang didapatkan oleh model dengan mengacu pada variasi kedalaman dan jarak berada pada kedalaman 1 meter dengan jarak 1 meter. Eror yang terjadi pada model dengan variasi 500 data terjadi pada kedalaman dan jarak 0,5 meter dan pada kedalaman 1,5 meter dengan jarak 1 meter. Sedangkan model juga tidak dapat mendeteksi objek pada variasi kedalaman 0,5 meter dengan kedalaman 1 meter dan pada kedalaman 1,5 meter dengan jarak 1 meter. Hasil deteksi variasi ini terdapat penurunan nilai persentase pada kedalaman 1 meter, dimana pada variasi jarak 0,5 meter model dapat mendeteksi objek *crack* dengan akurasi tertinggi 100%, pada jarak 0,75 meter sebesar 70%, dan pada jarak 1 meter akurasi tertinggi yang dapat dideteksi sebesar 69%.

Tabel 4. 5 Data variasi 500 data

Kedalaman (m)	Jarak (m)	Deteksi	Error
0,5	0,5	100%	76%
	0,75	100%	
	1		
1	0,5	100%	
	0,75	70%	
	1	69%	
1,5	0,5	100%	
	0,75	100%	
	1		63%

Pada tabel 4.6, data variasi 600 data *training* memiliki nilai yang lebih baik jika dibandingkan dengan variasi data *training* 400 dan 500. Hal dibuktikan dengan jumlah eror yang lebih sedikit dibandingkan kedua variasi jumlah data *training*. Pada variasi 600 data, eror terjadi pada variasi kedalaman 1,5 meter dengan jarak 0,75 meter dan 1 meter. Dimana pada jarak 0,75 meter terjadi eror berupa *false positive* atau model mendeteksi *crack* pada bagian yang tidak terdapat gambar *crack* dan pada jarak 1 meter model tidak mampu mendeteksi *crack* pada objek yang dianalisis. Nilai persentase akurasi tertinggi pada variasi ini mencapai 100% yang terdapat pada 5 variasi kedalaman dan jarak, sedangkan nilai persentase terendah terjadi pada variasi kedalaman 0,5 meter dan 1 meter dengan variasi jarak 1 meter dengan akurasi 64%.

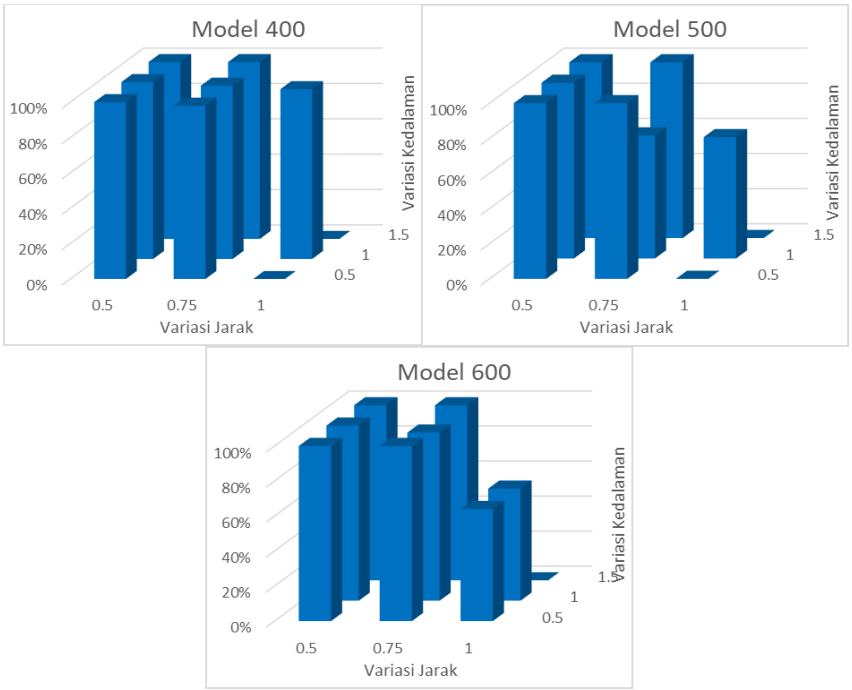
Dari hasil yang didapatkan terjadi penurunan nilai persentase yang terjadi pada variasi kedalaman 0,5 meter dan 1 meter. Pada kedalaman 0,5 meter terjadi penurunan nilai persentase

akurasi jarak 0,75 meter dan 1 meter sebesar 36%. Sedangkan pada kedalaman 0,75 meter terjadi penurunan nilai persentase pada variasi jarak secara berturut – turut 100%, 96%, 64%.

Tabel 4. 6 Data variasi 600 data

Kedalaman (m)	Jarak (m)	Deteksi	Error
0,5	0,5	100%	
	0,75	100%	
	1	64%	
1	0,5	100%	
	0,75	96%	
	1	64%	
1,5	0,5	100%	
	0,75	100%	86%
	1		

Adapun dari data pengujian yang didapatkan sesuai pada tabel 4.4 hingga tabel 4.6 kemudian dilakukan perbandingan hasil berupa grafik hasil uji. Pada gambar 4.7 menunjukkan perbandingan yang terjadi hasil uji deteksi *crack* dengan menggunakan variasi jumlah data *training*, kedalaman pengambilan data citra, dan jarak pengambilan data citra pada gambar 2. Pada variasi ke 3 terdapat dua model yang memiliki persentase 0% yaitu pada variasi 400 dan 500. Hal yang sama terjadi pada seluruh model pada variasi ke 9 dimana ketiga model tidak mampu mendeteksi *crack* sehingga grafik ketiga model berbentuk menurun dari variasi ke 8 menuju ke 9. Dari data grafik pada gambar 4.7 terlihat *trend* grafik ketiga model mengalami penurunan setiap tiga variasi. Pada pengujian model 400 data *training*, *trend* menurun didapatkan pada pengujian variasi 1-3 dan variasi 4-6. Pengujian pada model 500 data *training* terdapat *trend* menurun yang terjadi pada variasi 4-6. Sedangkan pada pengujian model 600 data *training* terdapat penurunan *trend* pada variasi 1-3, 4-6, dan 7-9. Dengan demikian pada variasi 600 data *training* yang dilakukan memiliki *trend* grafik pendeteksian *crack* terbaik jika dibandingkan dengan variasi jumlah data lainnya pada gambar 2.



Gambar 4. 7 Perbandingan uji model pada gambar 2

4.2.3 Hasil Deteksi Gambar 3

Tabel 4.7 menunjukkan hasil pengujian pada variasi 400 data *training*. Pada variasi 400 data, terjadi enam kali eror *false positive* ketika melakukan pengujian terhadap gambar 3 yaitu pada variasi kedalaman 0,5 meter dan 1 meter. Sedangkan model hanya mampu mendeteksi *crack* sebanyak lima kali dari sembilan kali variasi pengujian dengan nilai persentase akurasi terbesar yaitu 98% yang terjadi pada variasi kedalaman 1 meter dan jarak 0,5 meter dan nilai persentase terendah yang terdeteksi sebesar 83% pada variasi kedalaman 1 meter dan jarak 0,75 meter. Model variasi 400 data *training* ini juga belum mampu mendeteksi *crack* pada lima variasi uji yang lain seperti pada data tabel 4.7.

Pada data variasi 400 data *training* untuk pengujian gambar 3, terdapat penurunan nilai persentase akurasi deteksi pada variasi kedalaman 1 meter dengan kedalaman 0,5 meter dan 0,74 meter dengan nilai 98% menjadi 83%.

Tabel 4. 7 Data variasi 400 data

Kedalaman (m)	Jarak (m)	Deteksi	Error
0,5	0,5	95%	65%
	0,75		93%
	1		53%
1	0,5	98%	74%
	0,75	83%	94%
	1		71%
1,5	0,5	96%	
	0,75		
	1		

Pada gambar 3 dengan variasi 500 data *training* model mampu mendeteksi objek *crack* sebanyak lima kali dalam 9 variasi yang dilakukan seperti yang ditunjukkan pada tabel 4.8. Sedangkan terdapat eror yang terjadi ketika melakukan pendeteksian citra dimana model tidak mampu mendeteksi objek *crack* pada 4 variasi yang telah dilakukan diantaranya pada variasi kedalaman 0,5 meter dengan jarak 1 meter, variasi kedalaman dengan jarak 1 meter, dan 1,5 meter dengan jarak 0,75 meter dan 1 meter. Nilai akurasi tertinggi yang didapatkan selama pengujian model didapatkan sebesar 100% pada variasi kedalaman dan jarak 0,5 meter. Sedangkan nilai terendah dari lima keberhasilan deteksi didapatkan pada variasi kedalaman 1 meter dengan jarak 0,5 meter dan nilai persentase akurasi sebesar 56%. Berbeda dengan data yang didapat pada bab 4.2.1 dan 4.2.2 dimana pada model variasi 500 data tidak terdapat eror *false positive* untuk pengujian deteksi pada gambar 3.

Hasil deteksi yang ditunjukkan pada gambar 3 dengan variasi 500 data *training* menunjukkan hasil yang berbeda jika dibandingkan dengan pengujian pada gambar 1 dan 2, dimana tidak terdapat error *false positve* pada pengujian gambar 3. Jika dilihat pada nilai persentase akurasi tabel 4.8, terdapat penurunan nilai persentase pada variasi kedalaman 0,5 meter dengan jarak 0,5 meter dan 0,75 meter dari 100% menjadi 60%. Hal tersebut juga terjadi pada variasi kedalaman 1 meter, model mampu mendeteksi *crack* dengan persentase akurasi sebesar 97% pada jarak 0,5 meter dan 66% pada jarak 0,75 meter. Jika dilihat dari segi variasi kedalaman, akurasi yang didapatkan juga memiliki *trend* menurun yaitu dari 100%, 97%, hingga 96%.

Tabel 4. 8 Data variasi 500 data

Kedalaman (m)	Jarak (m)	Deteksi	Error
0,5	0,5	100%	
	0,75	60%	
	1		
1	0,5	97%	
	0,75	66%	
	1		
1,5	0,5	96%	
	0,75		
	1		

Data pada tabel 4.9 merupakan hasil pengujian pada model dengan variasi 600 data *training*. Dari data yang ada jika dibandingkan dengan kedua variasi data *training* 400 dan 500, variasi 600 data memiliki nilai uji yang lebih baik dibanding kedua variasi data lainnya. Hal ini terlihat dimana model mampu mendeteksi *crack* sembilan kali dari sembilan variasi yang diujikan. Namun terdapat eror *false positive* yang terjadi pada variasi kedalaman 1,5 meter dengan jarak 1 meter dimana terdapat akurasi tertinggi 70%. Dari nilai persentase akurasi yang ada terdapat nilai persentase 100% sebagai nilai tertinggi pada variasi kedalaman dan jarak 0,5 meter dan nilai persentase 70% sebagai nilai persentase terendah dari pengujian seluruh variasi kedalaman dan jarak yang terjadi pada variasi kedalaman 1,5 meter dan jarak 1 meter.

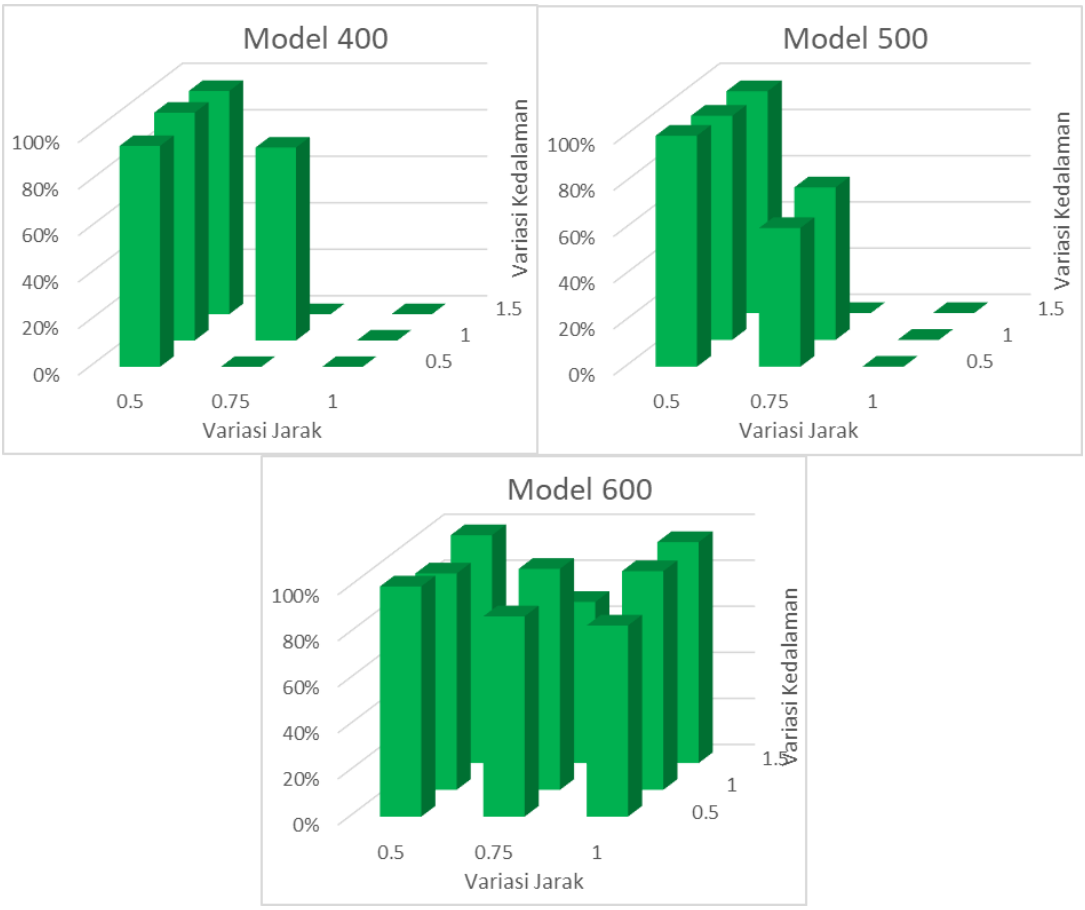
Penurunan persentase akurasi pendeteksian *crack* yang terjadi pada variasi 600 data *training* terjadi pada variasi kedalaman 0,5 meter secara berurutan 100%, 87%, dan 83%. Sedangkan pada variasi kedalaman 1 meter dan 1,5 meter memiliki perbedaan hasil deteksi pada variasi jaraknya. Dimana pada variasi kedalaman 1 meter, nilai persentase akurasi 0,5 meter memiliki nilai yang lebih rendah dibandingkan dengan variasi jarak 0,75 meter dan 1 meter. Sedangkan pada variasi kedalaman 1,5 meter, terdapat kenaikan akurasi pada pendeteksian variasi jarak 0,75 meter menuju jarak 1 meter dengan nilai persentase tertinggi sebesar 70% menuju 96%.

Tabel 4. 9 Data variasi 600 data

Kedalaman (m)	Jarak (m)	Deteksi	Error
0,5	0,5	100%	
	0,75	87%	
	1	83%	
1	0,5	94%	
	0,75	96%	
	1	95%	
1,5	0,5	99%	
	0,75	70%	
	1	96%	70%

Pada gambar 4.8 menunjukkan perbandingan yang terjadi hasil uji deteksi *crack* dengan menggunakan variasi jumlah data *training*, kedalaman pengambilan data citra, dan jarak pengambilan data citra pada gambar 3. Pada variasi data 400 dan 500, terlihat grafik menuju pada nilai 0% yang berarti model pada variasi data 400 dan 500 tidak mampu mendeteksi *crack* yang terdapat pada video uji. Sedangkan pada variasi data 600, terlihat grafik berada pada nilai persentase akurasi 60% hingga 100% dimana hal ini menunjukkan model mampu mendeteksi *crack* dari sembilan variasi yang dilakukan. Namun berbeda dengan pengujian pada gambar 1

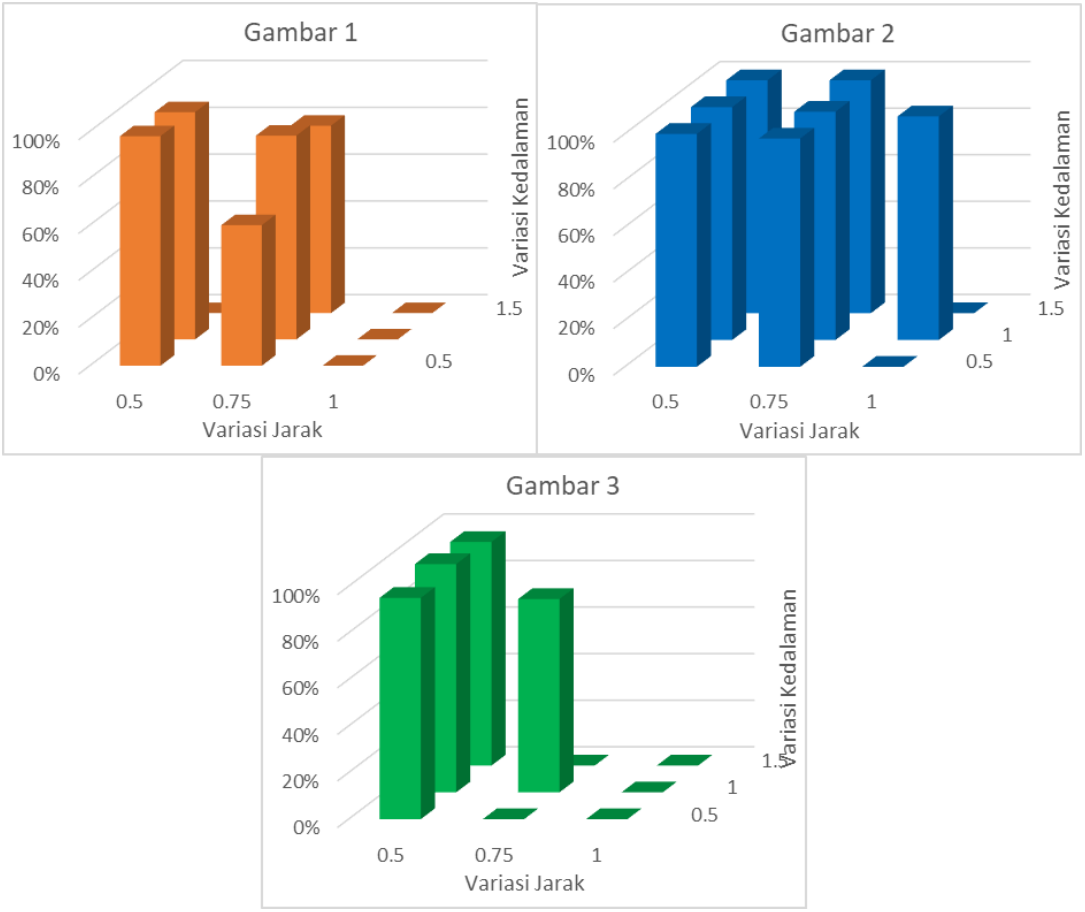
dan 2, pada gambar 3 variasi 600 data *training* memiliki *trend* menurun hanya pada tiga variasi awal yakni variasi 1 hingga 3, sedangkan pada variasi 4 hingga 9 *trend* mengalami kenaikan dan penurunan secara tidak teratur jika dibandingkan dengan hasil yang didapat pada pengujian gambar 1 dan 2. Sehingga dari grafik 4.6 model variasi jumlah data *training*, variasi 600 data *training* merupakan model terbaik dibandingkan dengan variasi model dengan data *training* 400 dan 500 pada pengujian deteksi *crack* gambar 3.



Gambar 4. 8 Grafik perbandingan uji model pada gambar 3

4.3 Pembahasan Hasil

Dari data yang telah didapatkan pada bab 4.1 dan 4.2 (hasil uji dapat dilihat pada Lampiran 9 hingga 11) menunjukkan bahwa model dengan jumlah data *training* sebanyak 600 data memiliki kemampuan deteksi *crack* dibawah permukaan air terbaik dari 3 macam model pelatihan data yang dibuat. Model ini mampu mendeteksi *crack* hingga kedalaman 1,5 meter dan jarak 1 meter dari objek. Dapat dilihat dari pertambahan jumlah data *training*, semakin banyak jumlah data yang digunakan pada pembelajaran metode *CNN*, semakin baik pula hasil yang didapatkan. Seperti yang ditunjukkan pada gambar 4.9 terlihat model mampu mendeteksi seluruh variasi kedalaman dan jarak yang diujikan sedangkan pada model variasi 400 dan 500 data *training* terdapat banyak variasi pengujian yang tidak dapat dipenuhi oleh kedua model. Namun pada model 600 data *training* terdapat satu variasi yang menyebabkan model tidak mampu mendeteksi *crack* pada pengujian gambar 2 dengan menggunakan variasi kedalaman 1,5 meter dan jarak 1 meter. Hal ini dapat disebabkan oleh berbagai macam faktor diantaranya kondisi pengambilan citra (*angle* pengambilan gambar) yang kurang baik ataupun jarak yang terlalu jauh jika menggunakan data gambar uji pada gambar 2.



Gambar 4. 9 Hasil uji model 400 pada 3 gambar uji

Grafik pada gambar 4.9 menunjukkan hasil dari pengujian pada model dengan data *training* 400. Model beberapa kali belum dapat mendeteksi *crack* pada citra pengujian atau yang biasa disebut sebagai *false positive*. Dengan melakukan perhitungan menggunakan metode *confusion matrix* didapatkan hasil seperti pada tabel 4.10 berikut:

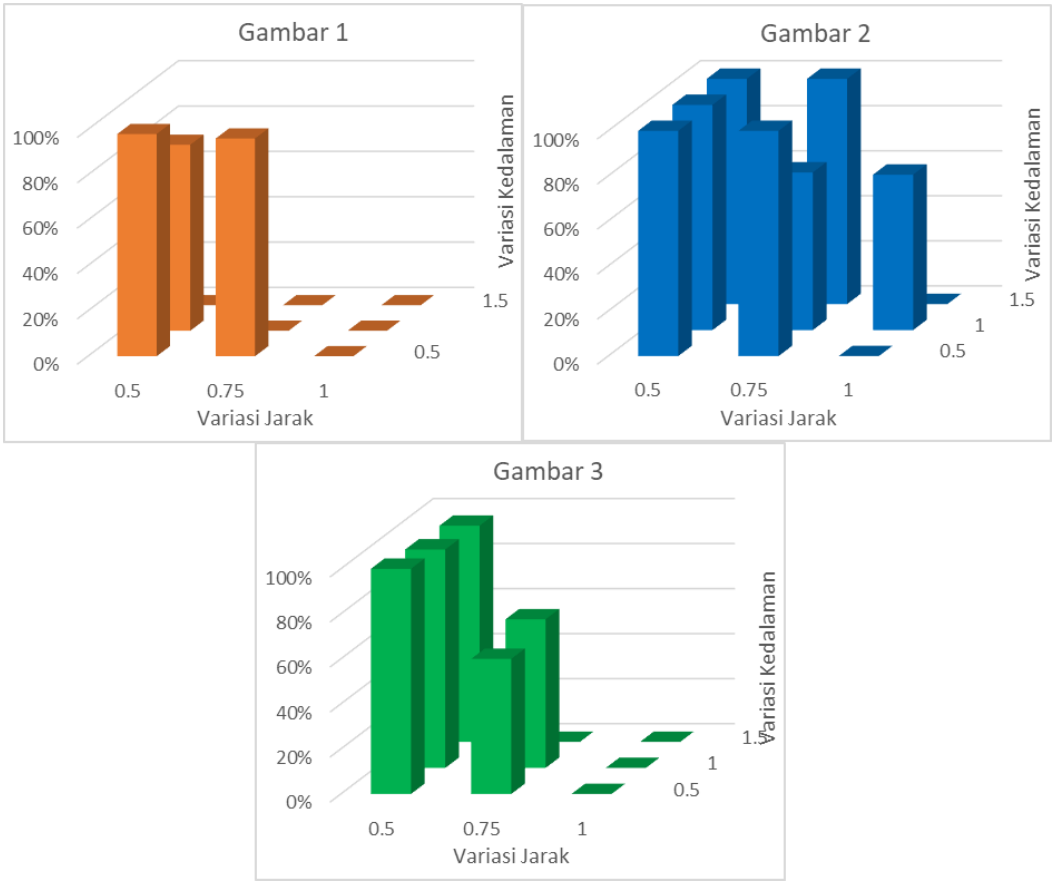
Tabel 4. 10 Confusion matrix model 400 data training

		Aktual	
		Positive	Negative
Prediksi	Positive	TP =15	FP =15
	Negative	FN =12	TN =12

Terdapat 15 data *true positive* (TP), 12 data *true negative* (TN), 15 data *false positive* (FP), dan 12 data *false negative* (FN). Pada hasil pengujian model 400 data *training*, didapatkan 15 data *true positive* dari 16 pengujian berhasil. Satu data yang didapatkan mendapatkan nilai akurasi deteksi sebesar 60% pada pengujian variasi kedalaman 0,5 meter dengan jarak 0,75 meter, hal ini menyebabkan nilai *true positive* pada tabel 4.10 menjadi 15 dikarenakan nilai akurasi pendeteksian ≤ 80%. Nilai *false positive* juga memiliki nilai 15 yang diakibatkan oleh pendeteksian *crack* pada nat keramik oleh model. Dengan menggunakan formulasi perhitungan pada formulasi 2.9, didapatkan nilai akurasi model *CNN* variasi 400 data *training* sebesar 50%.

Adapun hasil pengujian pada model dengan 500 data *training* dapat dilihat pada gambar 4.10. Pada pengujian gambar 1 didapatkan penurunan *trend* yang terjadi pada variasi 1 dan 2 dimana terdapat perubahan jarak pengujian dari 0,5 meter menjadi 0,75 meter. Penurunan *trend*

juga terjadi pada variasi 2 dan 4 dimana kedua variasi memiliki perubahan variasi kedalaman dari 0,5 meter menjadi 1 meter. Pengujian gambar 2 terjadi penurunan *trend* pada variasi 4, 5, dan 6. Hal ini dikarenakan terdapat perubahan variasi jarak pada ketiga variasi tersebut. Sedangkan pada hasil pengujian gambar 3 didapatkan penurunan *trend* pada variasi 1-3 dan variasi 4-6. Hal ini disebabkan akibat perubahan variasi jarak yang dilakukan dalam pengujian.



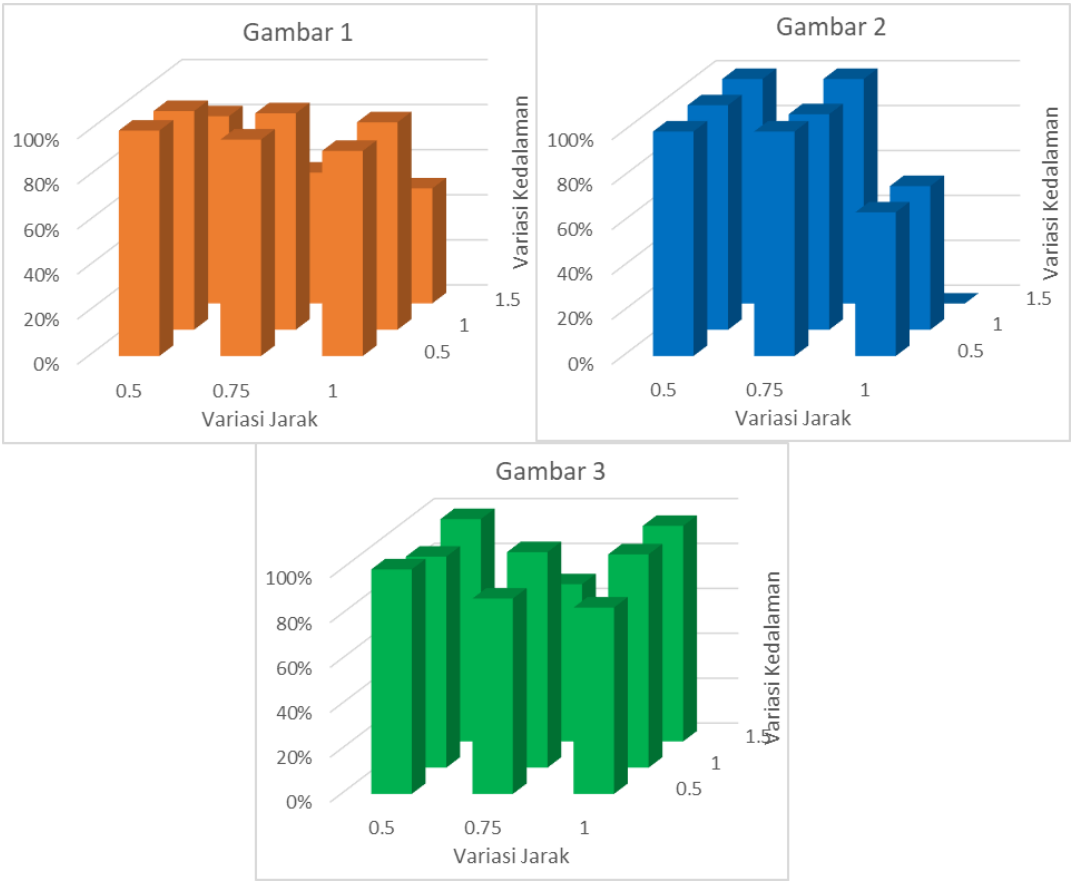
Gambar 4. 10 Hasil uji model 500 pada 3 gambar uji

Dari hasil pengujian pada gambar gambar 4.10 dapat diolah kemudian kedalam perhitungan menggunakan metode *confusion matrix*. Pada tabel 4.11 merupakan hasil pengelompokan kategori pada *confusion matrix*. Didapatkan 11 data *true positive*, 24 data *true negative*, 3 *false positive*, dan 16 *false negative*. Pada bab 4.2, model dengan 500 data *training* memiliki 15 kali keberhasilan dalam penentuan deteksi *crack*. Namun 4 diantaranya memiliki nilai akurasi yang mencapai ≤ 80%. Hal ini menyebabkan nilai pada *true positive* menjadi 11 sedangkan pada *false negative* menjadi 16. Tidak seperti model dengan menggunakan 400 data *training*, nilai dari *false positive* memiliki jumlah yang lebih sedikit dengan 3 variasi pengujian. Dengan melakukan perhitungan menggunakan formulasi pada 2.9 didapatkan nilai akurasi *CNN* yang lebih baik daripada model variasi 400 data *training* yaitu sebesar 64,81%.

Tabel 4. 11 Confusion matrix model 500 data training

		Aktual	
		Positive	Negative
Prediksi	Positive	TP = 11	FP = 3
	Negative	FN = 16	TN = 24

Pada gambar 4.11 didapatkan hasil uji yang dilakukan pada model dengan 600 data training. Hasil yang didapatkan pada gambar 1 meunjukkan penurunan *trend* setiap 3 variasi yaitu pada variasi 1-3, 4-6, dan 7-9. Penurunan nilai akurasi yang terjadi diakibatkan bertambahnya nilai variasi jarak yang digunakan pada pengujian. Jika dilakukan perbandingan pada variasi kedalaman juga didapatkan hasil yang sama dimana setiap penambahan nilai variasi kedalaman persentase akurasi *CNN* mengalami penurunan. Pada hasil uji gambar 2 dan gambar 3 didapatkan hal yang sama seperti pada gambar 1 dimana dengan penambahan kedalaman pada pengujian deteksi *crack* persentase akurasi *CNN* akan mengalami penurunan. Hal ini berlaku pula dengan penambahan jarak uji, semakin bertambah jarak objek dengan kamera uji, nilai persentase akurasi *CNN* yang dihasilkan juga berangsur menurun.



Gambar 4. 11 Hasil uji model 600 pada 3 gambar uji

Pada tabel 4.12 terdapat *confusion matrix* yang kemudian dapat dihitung menjadi nilai akurasi model. Terdapat 21 data *true positive*, 23 data *true negative*, 4 data *false positive*, dan 6 data *false negative*. Dari ketiga model variasi jumlah data training, pada variasi ini terdapat nilai akurasi ≤ 80% terbanyak jika dibandingkan dengan variasi lainnya. Terdapat lima data yang berhasil mendeteksi *crack* pada pengujian, namun nilai akurasi yang ditampilkan paling tinggi hanya sebesar 64%. Hal ini mengakibatkan nilai *true positive* yang semula 26 berubah menjadi 21 sedangkan pada kategori *false negative* bertambah dari satu poin menjadi 6 poin. Pada kategori *false positive* terdapat 4 data dimana terdapat eror pada model dengan mendeteksi *crack* pada nat keramik di belakang objek gambar, sehingga nilai *true negative* yang didapatkan pada model ini sejumlah 23. Dengan menggunakan formulasi perhitungan pada formulasi 2.9, didapatkan nilai akurasi model sebesar 81,48%.

Tabel 4. 12 Confusion matrix model 600 data training

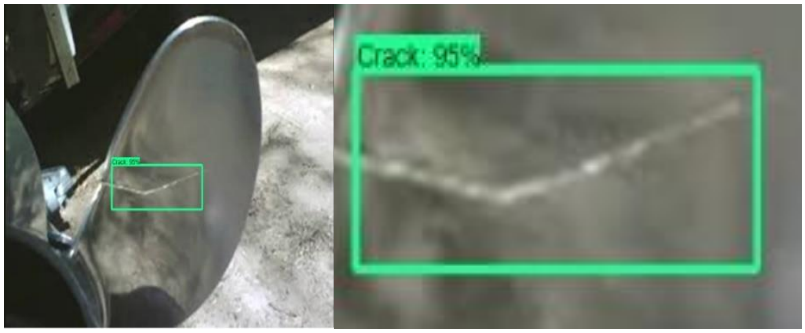
		Aktual	
		Positive	Negative
Prediksi	Positive	TP = 21	FP = 4
	Negative	FN = 6	TN = 23

Dengan melihat *trend* yang terjadi pada ketiga pengujian pada gambar 4.9, 4.10, dan 4.11 didapatkan hasil bahwa setiap penambahan kedalaman pada suatu pendeteksian objek dibawah permukaan air akan mengurangi kemampuan model dalam melakukan pendeteksian. Dalam hal ini disebabkan akibat menurunnya intensitas cahaya seiring dengan penambahan nilai kedalaman. Sedangkan dengan bertambahnya jarak pendeteksian juga akan mengurangi kemampuan model *CNN* dalam melakukan pendeteksian objek, hal ini disebabkan berkurangnya kualitas objek ketika ditangkap oleh kamera.

Pada model dengan data *training* 400 gambar didapatkan nilai akurasi *CNN* sebesar 50%, sedangkan pada model dengan 500 jumlah data *training* didapatkan nilai akurasi model sebesar 64,81%, dan pada model dengan 600 data *training* didapatkan nilai akurasi *CNN* sebesar 81,48%. Jika dilihat dari hasil akurasi *CNN* yang didapat, nilai persentase akurasi *CNN* semakin meningkat seiring pertambahan jumlah data *training.* Hal ini juga dapat dilihat dari pertambahan nilai kedalaman dan jarak dari setiap pengujian model. Semakin bertambah nilai kedalaman pengujian, hasil uji yang didapat menunjukkan bahwa persentase deteksi gambar semakin menurun. Hal ini diakibatkan akibat bertambahnya nilai kedalaman akan menurunkan intensiatas cahaya yang masuk kedalam air. Sedangkan pada penambahan jarak juga akan mengurangi kemampuan deteksi objek dari model, hal ini dapat dilihat dengan bertambahnya jarak objek maka persentase pengujian model semakin rendah. Penambahan jarak ini akan mempengaruhi kualitas gambar objek pada saat pendeteksian. Dengan demikian didapatkan model terbaik dari tiga variasi model jumlah data *training* dalam mendeteksi objek *crack* dengan variasi kedalaman hingga 1,5 meter dan jarak hingga 1 meter yaitu pada model *CNN* dengan melakukan *training* sebanyak 600 data *crack.*

4.4 Pengujian Pada *Propeller* Kapal

Setelah didapatkan model terbaik, pengujian dilakukan dengan mendeteksi *crack* yang terjadi pada *propeller* kapal. Adapun hasil yang didapatkan seperti pada gambar 4.12. Pengujian dilakukan dengan mendeteksi gambar *crack* pada *propeller* kapal yang tidak termasuk dalam data *training*. Dari pengujian menggunakan model 600 data *training* didapatkan persentase pengujian sebesar 95% pada *crack* yang terjadi.



Gambar 4. 12 Pengujian pada propeller kapal

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil penelitian yang telah dilakukan, terdapat beberapa poin yang dapat disimpulkan sebagai berikut:

1. Metode *Convolutional Neural Network (CNN)* dapat dilakukan pada pengimplementasian deteksi lokasi *crack* yang terjadi pada *propeller* kapal dengan menggunakan arsitektur *SSD mobilenet v1*. Arsitektur yang digunakan terbagi menjadi *input layer, convolution layer, pooling layer, fully connected layer,* dan *output.* Pendeteksian dilakukan dengan menangkap citra bawah laut yang kemudian diolah dengan model *CNN* yang telah dilakukan pelatihan data.
2. Variasi jumlah data *training*, kedalaman, dan jarak uji sangat mempengaruhi tingkat akurasi pengujian deteksi *crack* pada *propeller* kapal. Semakin bertambah nilai kedalaman dan jarak dari sebuah pengujian deteksi objek akan mengurangi kemampuan deteksi dari model *CNN*. Semakin banyak jumlah data *training* yang digunakan akan semakin meningkatkan kemampuan model *CNN* dalam mendeteksi objek. Variasi terbaik didapatkan pada variasi 600 data *training* dengan nilai *training loss* sebesar 0,861% yang mampu mendeteksi *crack* hingga kedalaman 1,5 meter dengan jarak 1 meter.
3. Dengan menggunakan metode perhitungan *confusion matrix*, didapatkan nilai akurasi model *CNN* pada tiga macam model dengan variasi jumlah data *traning* sebesar 50% pada model *training* 400 data, 64,81% pada model *training* 500 data, dan model terbaik dengan akurasi model sebesar 81,48% pada data *training* yang digunakan sebanyak 600 data.

5.2 Saran

Dengan dilakukannya penelitian ini, diperoleh beberapa saran guna pengembangan lanjutan dari laporan ini. Adapun saran dari penelitian ini adalah sebagai berikut:

1. Dalam pengambilan video *crack* dapat dilakukan secara kontinu untuk setiap variasi kedalaman dan jarak sehingga pergerakan yang terjadi pada tangkapan kamera dapat meningkatkan kemungkinan deteksi yang dilakukan oleh *CNN*.
2. Data uji yang digunakan dapat menggunakan objek yang sedang diteliti sehingga dapat meningkatkan akurasi aktual pendeteksian *crack* pada *propeller* kapal.

DAFTAR PUSTAKA

Afonso, M., Blok, P. M., Polder, G., Van Der Wolf, J. M., & Kamp, J. (2019). Blackleg Detection in Potato Plants using Convolutional Neural Networks. *IFAC-PapersOnLine*, *52*(30), 6–11. https://doi.org/10.1016/j.ifacol.2019.12.481

Al-Azzo, F., Taqi, A. M., & Milanova, M. (2018). Human related-health actions detection using Android Camera based on TensorFlow Object Detection API. *International Journal of Advanced Computer Science and Applications*, *9*(10), 9–23. https://doi.org/10.14569/IJACSA.2018.091002

Alpaydin, E. (2009). *Introduction to Mechine Learning*. MIT Press.

Anam, H., & Setiawan, J. (2015). Simulasi Dan Analisa Dinamika Remotely Operated Vehicle (Rov). *Jurnal Teknik Mesin*, *3*(1), 1–7.

Arafat, Y. (2018). Penyelenggaraan Pembangunan NKRI Menuju Negara Maritim Berdasarkan Prinsip Negara Kepulauan. *Akta Yudisia*, *3*(1), 1–23.

Belajarpython. (2010). *Pendahuluan Python*. Belajarpython.Com. https://belajarpython.com/tutorial/apa-itu-python

Cha, Y. J., Choi, W., & Büyüköztürk, O. (2017). Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks. *Computer-Aided Civil and Infrastructure Engineering*, *32*(5), 361–378. https://doi.org/10.1111/mice.12263

Djazuli Sa'id, S. (2012). Analisa Keretakan Pada Konstruksi Geladak Utama Km. Adri Xliv. *Gema Teknologi*, *16*(4), 164. https://doi.org/10.14710/gt.v16i4.4783

Eka Putra, W. S. (2016). Klasifikasi Citra Menggunakan Convolutional Neural Network (CNN) pada Caltech 101. *Jurnal Teknik ITS*, *5*(1). https://doi.org/10.12962/j23373539.v5i1.15696

Geitgey, A. (2016). *Machine Learning is Fun! Part 3: Deep Learning and Convolutional Neural Networks*. Medium.Com. https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. http://arxiv.org/abs/1704.04861

John Rajan, A., Jayakrishna, K., Vignesh, T., Chandradass, J., & Kannan, T. T. M. (2020). Development of computer vision for inspection of bolt using convolutional neural network. *Materials Today: Proceedings*, *45*, 6931–6935. https://doi.org/10.1016/j.matpr.2021.01.372

Liu, W. (2016). Detector, SSD: Single Shot MultiBox. *European Conference on Computer Vision*.

Mahavarkar, A., Kadwadkar, R., Maurya, S., & Raveendran, S. (2020). Underwater Object Detection using Tensorflow. *ITM Web of Conferences*, *32*, 03037. https://doi.org/10.1051/itmconf/20203203037

Mashita, S. N. (2020). Implementasi Deep Learning Object Detection Rambu K3 Pada Video Menggunakan Metode Convolutional Neural Network (CNN) Dengan Tensorflow (Studi Kasus: Rambu Kesehatan dan Keselamatan Kerja (K3) Jalur Evakuasi dan Alat Pemadam Api pada Gedung FMIPA UII). *Skripsi*, *Statistika*, *Fakultas Matematika Dan*

Ilmu Pengetahuan Alam, Universitas Islam Indonesia, Yogyakarta.

Novyantika, R. D. (2018). *DETEKSI TANDA NOMOR KENDARAAN BERMOTOR PADA MEDIA STREAMING DENGAN ALGORITMA CONVOLUTIONAL NEURAL NETWORK MENGGUNAKAN Diajukan Sebagai Salah Satu Syarat Untuk Memperoleh Gelar Sarjana Jurusan Statistika Disusun Oleh : Rizky Dwi Novyantika*. *March*.

Pal, A., & Hsieh, S. H. (2021). Deep-learning-based visual data analytics for smart construction management. *Automation in Construction*, *131*(May), 103892. https://doi.org/10.1016/j.autcon.2021.103892

Paramarthalingam, A. (2016). Machine parts recognition and defect detection in automated assembly systems using computer vision techniques. *Rev. Téc. Ing. Univ. Zulia.*, *January*. https://doi.org/10.21311/001.39.1.08

R. A. Septian, A. Rahmania, M. I. Nugraha, Y. (2017). REMOTELY OPERATED VEHICLE (ROV) UNTUK EKSPLORASI BAWAH AIR DILINGKUNGAN INDUSTRI PERKAPALAN. *Manutech*, *9*, 22.

Sharma, H. B., Panigrahi, S., Sarmah, A. K., & Dubey, B. K. (2019). Eggshell crack detection using deep 1 convolutional neural networks. *Science of the Total Environment*, 135907. https://doi.org/10.1016/j.aca.2021.338884

Suharto, E., Suhartono, Widodo, A. P., & Sarwoko, E. A. (2020). The use of mobilenet v1 for identifying various types of freshwater fish. *Journal of Physics: Conference Series*, *1524*(1), 1–6. https://doi.org/10.1088/1742-6596/1524/1/012105

Traore, B. B., Kamsu-Foguem, B., & Tangara, F. (2018). Deep convolution neural network for image recognition. *Ecological Informatics*, *48*(October), 257–268. https://doi.org/10.1016/j.ecoinf.2018.10.002

Vidal Pino, O., Nascimento, E. R., & Campos, M. F. M. (2021). Introducing the structural bases of typicality effects in deep learning. *Image and Vision Computing*, *113*, 104249. https://doi.org/10.1016/j.imavis.2021.104249

LAMPIRAN

LAMPIRAN 1 Arsitektur SSD Mobilenet V1

TYPE/STRIDE	FILTER SHAPE	INPUT SIZE
Conv/s2	3 x 3 x 3 x 32	300 x 300 x 3
Conv dw/s1	3 x 3 x 32 dw	150 x 150 x 32
Conv/s1	1 x 1 x 32 x 64	150 x 150 x 32
Conv dw/s2	3 x 3 x 64 dw	150 x 150 x 64
Conv/s1	1 x 1 x 64 x 128	75 x 75 x 64
Conv dw/s1	3 x 3 x 128 dw	75 x 75 x 128
Conv/s1	1 x 1 x 128 x 128	75 x 75 x 128
Conv dw/s2	3 x 3 x 128 dw	75 x 75 x 128
Conv/s1	1 x 1 x 128 x 256	38 x 38 x 128
Conv dw/s1	3 x 3 x 256 dw	38 x 38 x 256
Conv/s1	1 x 1 x 256 x 512	38 x 38 x 256
Conv dw/s1	3 x 3 x 512 dw	38 x 38 x 512
Conv/s1	1 x 1 x 512 x 512	38 x 38 x 512
Conv dw/s1	3 x 3 x 512 dw	38 x 38 x 512
Conv/s1	1 x 1 x 512 x 512	38 x 38 x 512
Conv dw/s1	3 x 3 x 512 dw	38 x 38 x 512
Conv/s1	1 x 1 x 512 x 512	38 x 38 x 512
Conv dw/s1	3 x 3 x 512 dw	38 x 38 x 512
Conv/s1	1 x 1 x 512 x 512	38 x 38 x 512
Conv dw/s1	3 x 3 x 512 dw	38 x 38 x 512
Conv/s1	1 x 1 x 512 x 512	38 x 38 x 512
Conv dw/s1	3 x 3 x 512 dw	38 x 38 x 512
Conv/s1	1 x 1 x 512 x 512	38 x 38 x 512
Conv/s2	3 x 3 x 512 x 1024	38 x 38 x 512
Conv/s1	1 x 1 x 1024 x 1024	19 x 19 x 1024
Conv/s1	1 x 1 x 1024 x 256	19 x 19 x 1024
Conv/s2	3 x 3 x 256 x 512	19 x 19 x 256
Conv/s1	1 x 1 x 512 x 128	10 x 10 x 512
Conv/s2	3 x 3 x 128 x 256	10 x 10 x 128
Conv/s1	1 x 1 x 256 x 128	5 x 5 x 256
Conv/s2	3 x 3 x 128 x 256	5 x 5 x 128
Conv/s1	1 x 1 x 256 x 128	3 x 3 x 256
Conv/s1	3 x 3 x 128 x 256	3 x 3 x 128
Conv/s1	1 x 1 x 256 x 128	1 x 1 x 256
Conv/s1	3 x 3 x 128 x 256	1 x 1 x 128

LAMPIRAN 2 *Script* file *resize* citra

```
import cv2
import glob
import os

inputfolder = 'foto1'
os.mkdir('resized1')

i = 0

for img in glob.glob(inputfolder + "/*.jpeg"):
    image = cv2.imread(img)
    imgResized = cv2.resize(image, (300,300))
    cv2.imwrite("resized1/1gambar%i.jpg" %i, imgResized)

    i += 1
    # cv2.imshow('image', imgResized)
    # cv2.waitkey(30)

cv2.destroyAllWindows()
```

LAMPIRAN 3 *Script* file xml_to_csv

```
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET

def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class',
'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df


def main():
    for directory in ['train','test']:
        image_path = os.path.join(os.getcwd(),
'images/{}'.format(directory))
        xml_df = xml_to_csv(image_path)
        xml_df.to_csv('data/{}_labels.csv'.format(directory),
index=None)
        print('Successfully converted xml to csv.')


main()
```

LAMPIRAN 4 *Script* file generate_tfrecord

```
from __future__ import print_function
from __future__ import absolute_import

import os
import io
import pandas as pd
import tensorflow as tf

from PIL import Image
from object_detection.utils import dataset_util
from collections import namedtuple, OrderedDict

flags = tf.app.flags
flags.DEFINE_string('csv_input', '', 'Path to the CSV input')
flags.DEFINE_string('output_path', '', 'Path to output
TFRecord')
flags.DEFINE_string('image_dir', '', 'Path to images')
FLAGS = flags.FLAGS


# replace row_label with the name you annotated your images
as
def class_text_to_int(row_label):
    if row_label == 'crack':
        return 1
    else:
        None


def split(df, group):
    data = namedtuple('data', ['filename', 'object'])
    gb = df.groupby(group)
    return [data(filename, gb.get_group(x)) for filename, x
in zip(gb.groups.keys(), gb.groups)]


def create_tf_example(group, path):
    with tf.gfile.GFile(os.path.join(path,
'{}'.format(group.filename)), 'rb') as fid:
        encoded_jpg = fid.read()
    encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size

    from __future__ import division
    filename = group.filename.encode('utf8')
    image_format = b'jpg'
    xmins = []
    xmaxs = []
```

```
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []


  for index, row in group.object.iterrows():
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
        classes_text.append(row['class'].encode('utf8'))
        classes.append(class_text_to_int(row['class']))

    tf_example =
tf.train.Example(features=tf.train.Features(feature={
        'image/height': dataset_util.int64_feature(height),
        'image/width': dataset_util.int64_feature(width),
        'image/filename':
dataset_util.bytes_feature(filename),
        'image/source_id':
dataset_util.bytes_feature(filename),
        'image/encoded':
dataset_util.bytes_feature(encoded_jpg),
        'image/format':
dataset_util.bytes_feature(image_format),
        'image/object/bbox/xmin':
dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax':
dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin':
dataset_util.float_list_feature(ymins),
        'image/object/bbox/ymax':
dataset_util.float_list_feature(ymaxs),
        'image/object/class/text':
dataset_util.bytes_list_feature(classes_text),
        'image/object/class/label':
dataset_util.int64_list_feature(classes),
    }))
    return tf_example


def main(_):
    writer = tf.python_io.TFRecordWriter(FLAGS.output_path)
    path = os.path.join(FLAGS.image_dir)
    examples = pd.read_csv(FLAGS.csv_input)
    grouped = split(examples, 'filename')
    for group in grouped:
        tf_example = create_tf_example(group, path)
        writer.write(tf_example.SerializeToString())
```

```
    writer.close()
    output_path = os.path.join(os.getcwd(),
FLAGS.output_path)
    print('Successfully created the TFRecords:
{}'.format(output_path))


if __name__ == '__main__':
    tf.app.run()
```

LAMPIRAN 5 *Script* file train.py

```
import functools
import json
import os
import tensorflow.compat.v1 as tf
from tensorflow.python.util.deprecation import deprecated


from object_detection.builders import dataset_builder
from object_detection.builders import graph_rewriter_builder
from object_detection.builders import model_builder
from object_detection.legacy import trainer
from object_detection.utils import config_util

tf.logging.set_verbosity(tf.logging.INFO)

flags = tf.app.flags
flags.DEFINE_string('master', '', 'Name of the TensorFlow
master to use.')
flags.DEFINE_integer('task', 0, 'task id')
flags.DEFINE_integer('num_clones', 1, 'Number of clones to
deploy per worker.')
flags.DEFINE_boolean('clone_on_cpu', False,
                     'Force clones to be deployed on CPU.
Note that even if '
                     'set to False (allowing ops to run on
gpu), some ops may '
                     'still be run on the CPU if they have no
GPU kernel.')
flags.DEFINE_integer('worker_replicas', 1, 'Number of
worker+trainer '
                     'replicas.')
flags.DEFINE_integer('ps_tasks', 0,
                     'Number of parameter server tasks. If
None, does not use '
                     'a parameter server.')
flags.DEFINE_string('train_dir', '',
                    'Directory to save the checkpoints and
training summaries.')

flags.DEFINE_string('pipeline_config_path', '',
                    'Path to a
pipeline_pb2.TrainEvalPipelineConfig config '
                    'file. If provided, other configs are
ignored')

flags.DEFINE_string('train_config_path', '',
                    'Path to a train_pb2.TrainConfig config
file.')
flags.DEFINE_string('input_config_path', '',
```

```
                    'Path to an input_reader_pb2.InputReader
config file.')
flags.DEFINE_string('model_config_path', '',
                    'Path to a model_pb2.DetectionModel
config file.')

FLAGS = flags.FLAGS


@deprecated(None, 'Use object_detection/model_main.py.')
def main(_):
  assert FLAGS.train_dir, '`train_dir` is missing.'
  if FLAGS.task == 0: tf.gfile.MakeDirs(FLAGS.train_dir)
  if FLAGS.pipeline_config_path:
    configs = config_util.get_configs_from_pipeline_file(
        FLAGS.pipeline_config_path)
    if FLAGS.task == 0:
      tf.gfile.Copy(FLAGS.pipeline_config_path,
                    os.path.join(FLAGS.train_dir,
'pipeline.config'),
                    overwrite=True)
  else:
    configs = config_util.get_configs_from_multiple_files(
        model_config_path=FLAGS.model_config_path,
        train_config_path=FLAGS.train_config_path,
        train_input_config_path=FLAGS.input_config_path)
    if FLAGS.task == 0:
      for name, config in [('model.config',
FLAGS.model_config_path),
                           ('train.config',
FLAGS.train_config_path),
                           ('input.config',
FLAGS.input_config_path)]:
        tf.gfile.Copy(config, os.path.join(FLAGS.train_dir,
name),
                      overwrite=True)

  model_config = configs['model']
  train_config = configs['train_config']
  input_config = configs['train_input_config']

  model_fn = functools.partial(
      model_builder.build,
      model_config=model_config,
      is_training=True)

  def get_next(config):
    return dataset_builder.make_initializable_iterator(
        dataset_builder.build(config)).get_next()
```

```
  create_input_dict_fn = functools.partial(get_next,
input_config)

  env = json.loads(os.environ.get('TF_CONFIG', '{}'))
  cluster_data = env.get('cluster', None)
  cluster = tf.train.ClusterSpec(cluster_data) if
cluster_data else None
  task_data = env.get('task', None) or {'type': 'master',
'index': 0}
  task_info = type('TaskSpec', (object,), task_data)

  # Parameters for a single worker.
  ps_tasks = 0
  worker_replicas = 1
  worker_job_name = 'lonely_worker'
  task = 0
  is_chief = True
  master = ''

  if cluster_data and 'worker' in cluster_data:
    # Number of total worker replicas include "worker"s and
the "master".
    worker_replicas = len(cluster_data['worker']) + 1
  if cluster_data and 'ps' in cluster_data:
    ps_tasks = len(cluster_data['ps'])

  if worker_replicas > 1 and ps_tasks < 1:
    raise ValueError('At least 1 ps task is needed for
distributed training.')

  if worker_replicas >= 1 and ps_tasks > 0:
    # Set up distributed training.
    server = tf.train.Server(tf.train.ClusterSpec(cluster),
protocol='grpc',
                             job_name=task_info.type,
                             task_index=task_info.index)
    if task_info.type == 'ps':
      server.join()
      return

    worker_job_name = '%s/task:%d' % (task_info.type,
task_info.index)
    task = task_info.index
    is_chief = (task_info.type == 'master')
    master = server.target

  graph_rewriter_fn = None
  if 'graph_rewriter_config' in configs:
    graph_rewriter_fn = graph_rewriter_builder.build(
        configs['graph_rewriter_config'], is_training=True)
```

```
  trainer.train(
      create_input_dict_fn,
      model_fn,
      train_config,
      master,
      task,
      FLAGS.num_clones,
      worker_replicas,
      FLAGS.clone_on_cpu,
      ps_tasks,
      worker_job_name,
      is_chief,
      FLAGS.train_dir,
      graph_hook_fn=graph_rewriter_fn)


if __name__ == '__main__':
  tf.app.run()
```

LAMPIRAN 6 *Script* file pengaturan parameter *training*

```
model {
  SSD {
    num_classes: 1
    box_coder {
      faster_rCNN_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    anchor_generator {
      SSD_anchor_generator {
        num_layers: 6
        min_scale: 0.2
        max_scale: 0.95
        aspect_ratios: 1.0
        aspect_ratios: 2.0
        aspect_ratios: 0.5
        aspect_ratios: 3.0
        aspect_ratios: 0.3333
      }
    }
    image_resizer {
      fixed_shape_resizer {
        height: 300
        width: 300
      }
    }
    box_predictor {
      convolutional_box_predictor {
        min_depth: 0
        max_depth: 0
        num_layers_before_predictor: 0
        use_dropout: false
```

```
dropout_keep_probability: 0.8
        kernel_size: 1
        box_code_size: 4
        apply_sigmoid_to_scores: false
        conv_hyperparams {
          activation: RELU_6,
          regularizer {
            l2_regularizer {
              weight: 0.00004
            }
          }
          initializer {
            truncated_normal_initializer {
              stddev: 0.03
              mean: 0.0
            }
          }
          batch_norm {
            train: true,
            scale: true,
            center: true,
            decay: 0.9997,
            epsilon: 0.001,
          }
        }
      }
    }
    feature_extractor {
      type: 'SSD_mobilenet_v1'
      min_depth: 16
      depth_multiplier: 1.0
      conv_hyperparams {
        activation: RELU_6,
        regularizer {
          l2_regularizer {
            weight: 0.00004
          }
        }
        initializer {
          truncated_normal_initializer {
            stddev: 0.03
            mean: 0.0
          }
        }
        batch_norm {
          train: true,

scale: true,
center: true,
```

```
          decay: 0.9997,
          epsilon: 0.001,
        }
      }
    }
    loss {
      classification_loss {
        weighted_sigmoid {
          anchorwise_output: true
        }
      }
      localization_loss {
        weighted_smooth_l1 {
          anchorwise_output: true
        }
      }
      hard_example_miner {
        num_hard_examples: 3000
        iou_threshold: 0.99
        loss_type: CLASSIFICATION
        max_negatives_per_positive: 3
        min_negatives_per_image: 0
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    normalize_loss_by_num_matches: true
    post_processing {
      batch_non_max_suppression {
        score_threshold: 1e-8
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 100
      }
      score_converter: SIGMOID
    }
  }
}

train_config: {
  batch_size: 24
  optimizer {
    rms_prop_optimizer: {
      learning_rate: {
        exponential_decay_learning_rate {

initial_learning_rate: 0.004
          decay_steps: 800720
          decay_factor: 0.95
```

```
        }
      }
      momentum_optimizer_value: 0.9
      decay: 0.9
      epsilon: 1.0
    }
  }
  fine_tune_checkpoint:
"SSD_mobilenet_v1_coco_2018_01_28/model.ckpt"
  from_detection_checkpoint: true
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    SSD_random_crop {
    }
  }
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "data/train.record"
  }
  label_map_path: "data/object-detection.pbtxt"
}

eval_config: {
  num_examples: 40
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "data/test.record"
  }
  label_map_path: "training/object-detection.pbtxt"
  shuffle: false
  num_readers: 1
}
```

LAMPIRAN 7 Script file export_inference_graph.py

```
import tensorflow.compat.v1 as tf
from google.protobuf import text_format
from object_detection import exporter
from object_detection.protos import pipeline_pb2

flags = tf.app.flags

flags.DEFINE_string('input_type', 'image_tensor', 'Type of input node.
Can be '
                    'one of [`image_tensor`,
`encoded_image_string_tensor`, '
                    '`tf_example`]')
flags.DEFINE_string('input_shape', None,
                    'If input_type is `image_tensor`, this can explicitly
set '
                    'the shape of this input tensor to a fixed size. The
'
                    'dimensions are to be provided as a comma-separated
list '
                    'of integers. A value of -1 can be used for unknown '
                    'dimensions. If not specified, for an `image_tensor,
the '
                    'default shape will be partially specified as '
                    '`[None, None, None, 3]`.')
flags.DEFINE_string('pipeline_config_path', None,
                    'Path to a pipeline_pb2.TrainEvalPipelineConfig
config '
                    'file.')
flags.DEFINE_string('trained_checkpoint_prefix', None,
                    'Path to trained checkpoint, typically of the form '
                    'path/to/model.ckpt')
flags.DEFINE_string('output_directory', None, 'Path to write outputs.')
flags.DEFINE_string('config_override', '',
                    'pipeline_pb2.TrainEvalPipelineConfig '
                    'text proto to override pipeline_config_path.')
flags.DEFINE_boolean('write_inference_graph', False,
                     'If true, writes inference graph to disk.')
flags.DEFINE_string('additional_output_tensor_names', None,
                    'Additional Tensors to output, to be specified as a
comma '
                    'separated list of tensor names.')
flags.DEFINE_boolean('use_side_inputs', False,
                     'If True, uses side inputs as well as image
inputs.')
flags.DEFINE_string('side_input_shapes', None,
                    'If use_side_inputs is True, this explicitly sets '
                    'the shape of the side input tensors to a fixed size.
The '
                    'dimensions are to be provided as a comma-separated
list '
                    'of integers. A value of -1 can be used for unknown '
                    'dimensions. A `/` denotes a break, starting the
shape of '
                    'the next side input tensor. This flag is required if
'
                    'using side inputs.')
flags.DEFINE_string('side_input_types', None,
                    'If use_side_inputs is True, this explicitly sets '
                    'the type of the side input tensors. The '
```

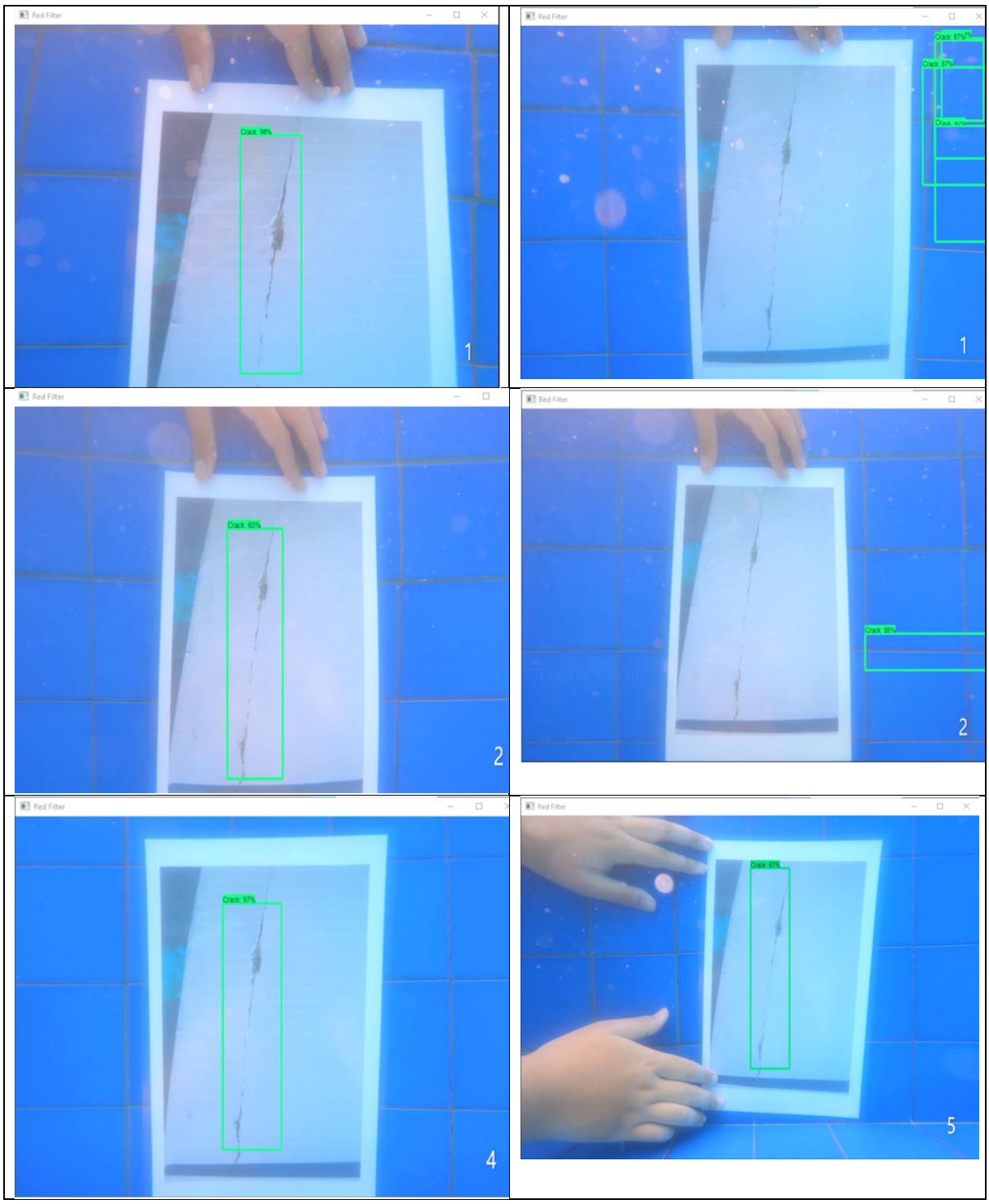
```
                    'dimensions are to be provided as a comma-separated
list '
                    'of types, each of `string`, `integer`, or `float`. '
                    'This flag is required if using side inputs.')
flags.DEFINE_string('side_input_names', None,
                    'If use_side_inputs is True, this explicitly sets '
                    'the names of the side input tensors required by the
model '
                    'assuming the names will be a comma-separated list of
'
                    'strings. This flag is required if using side
inputs.')
tf.app.flags.mark_flag_as_required('pipeline_config_path')
tf.app.flags.mark_flag_as_required('trained_checkpoint_prefix')
tf.app.flags.mark_flag_as_required('output_directory')
FLAGS = flags.FLAGS


def main(_):
  pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
  with tf.gfile.GFile(FLAGS.pipeline_config_path, 'r') as f:
    text_format.Merge(f.read(), pipeline_config)
  text_format.Merge(FLAGS.config_override, pipeline_config)
  if FLAGS.input_shape:
    input_shape = [
        int(dim) if dim != '-1' else None
        for dim in FLAGS.input_shape.split(',')
    ]
  else:
    input_shape = None
  if FLAGS.use_side_inputs:
    side_input_shapes, side_input_names, side_input_types = (
        exporter.parse_side_inputs(
            FLAGS.side_input_shapes,
            FLAGS.side_input_names,
            FLAGS.side_input_types))
  else:
    side_input_shapes = None
    side_input_names = None
    side_input_types = None
  if FLAGS.additional_output_tensor_names:
    additional_output_tensor_names = list(
        FLAGS.additional_output_tensor_names.split(','))
  else:
    additional_output_tensor_names = None
  exporter.export_inference_graph(
      FLAGS.input_type, pipeline_config, FLAGS.trained_checkpoint_prefix,
      FLAGS.output_directory, input_shape=input_shape,
      write_inference_graph=FLAGS.write_inference_graph,
      additional_output_tensor_names=additional_output_tensor_names,
      use_side_inputs=FLAGS.use_side_inputs,
      side_input_shapes=side_input_shapes,
      side_input_names=side_input_names,
      side_input_types=side_input_types)


if __name__ == '__main__':
tf.app.run()
```

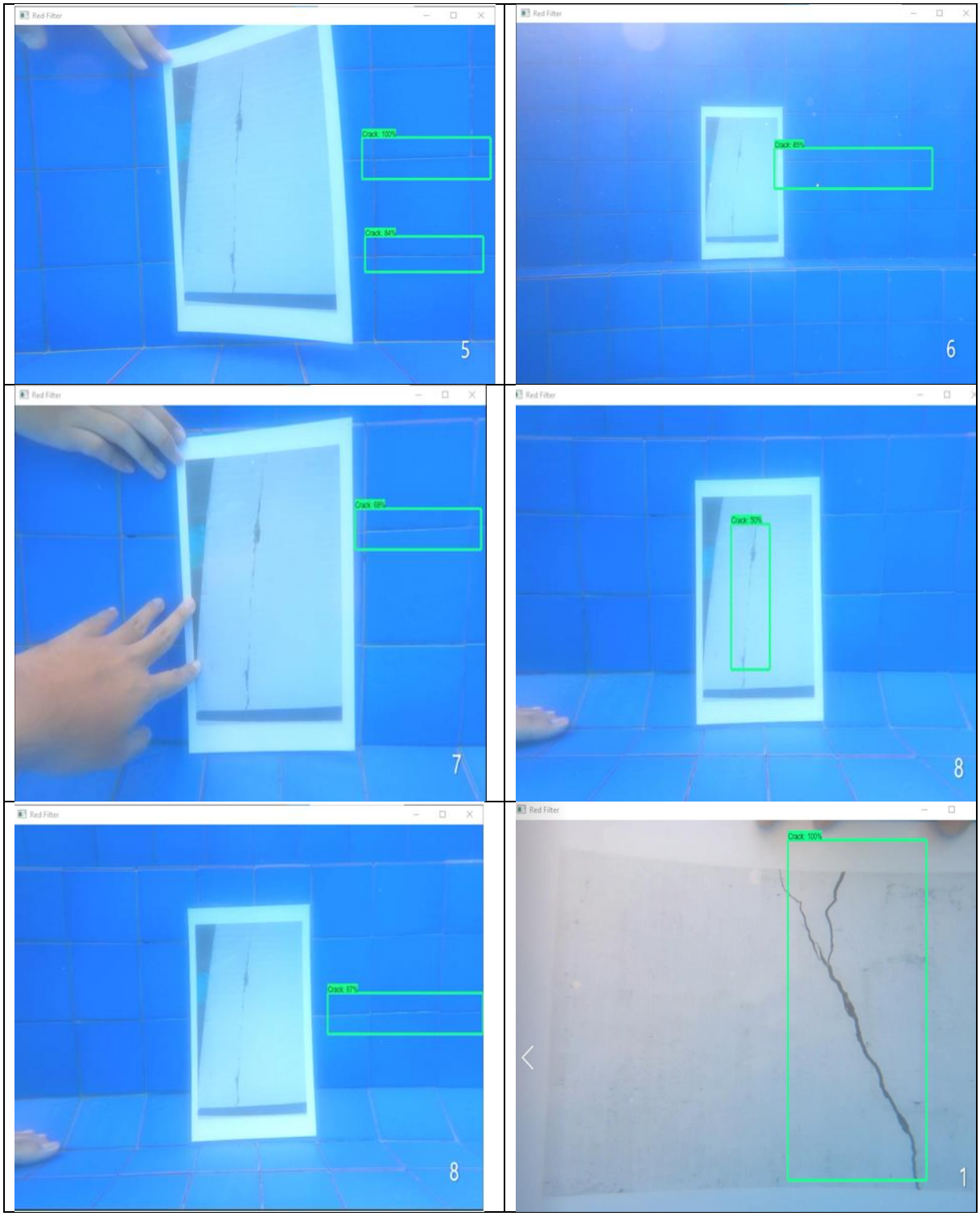
LAMPIRAN 8 *Script file* Deteksi_Crack.py

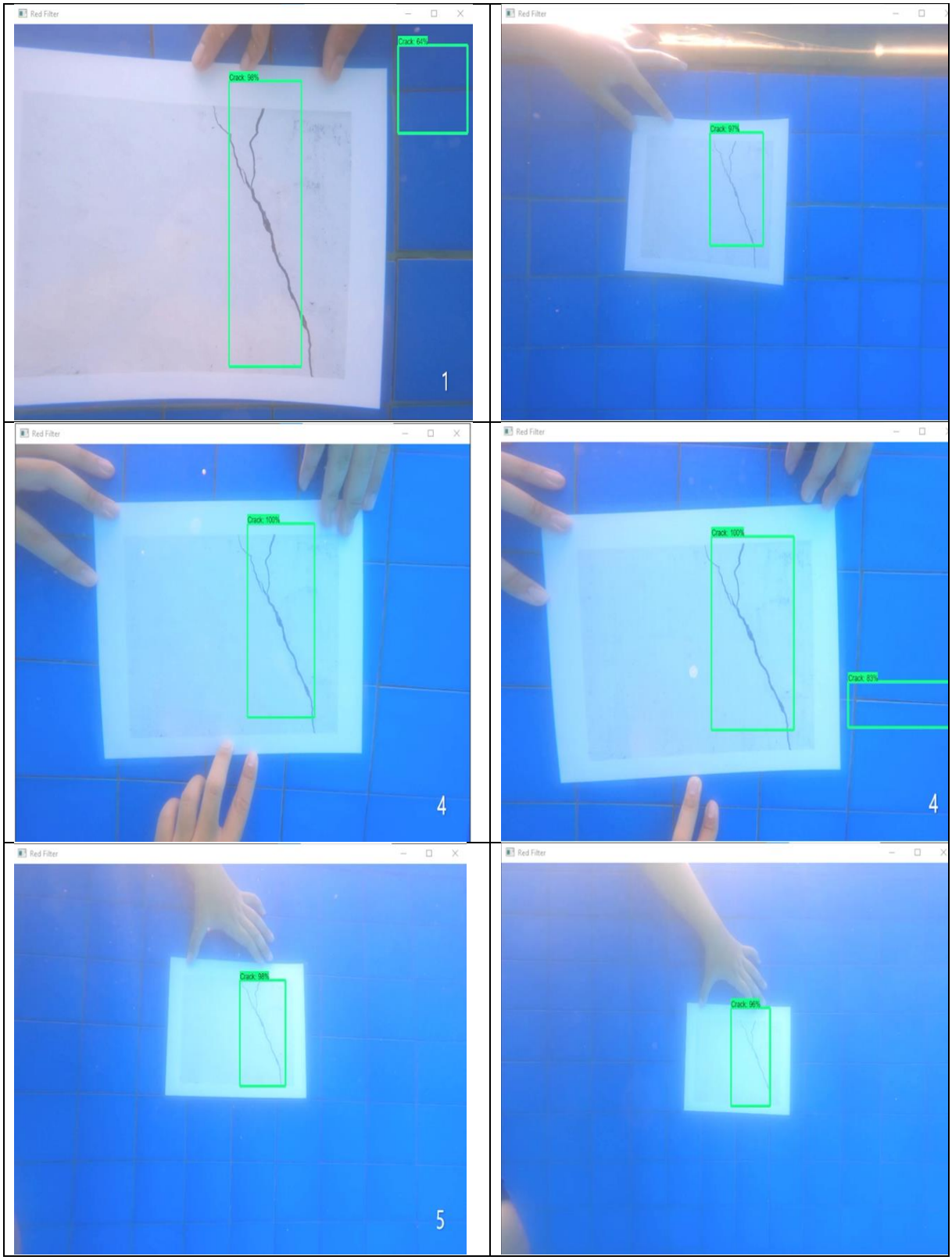
```
import numpy as np
import os
import cv2
import six.moves.urllib as urllib
import sys
import tarfile
import random
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
tf.config.experimental.list_physical_devices('GPU')
import zipfile
import glob
import math

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image

from utils import label_map_util

from utils import visualization_utils as vis_util

import cv2

cap = cv2.VideoCapture("crack3.mp4")
MODEL_NAME = 'new_graph'
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'

PATH_TO_LABELS = os.path.join('data', 'object-detection.pbtxt')

NUM_CLASSES = 1

detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

def verify_alpha_channel(frame):
    try:
        frame.shape[3]  # looking for the alpha channel
    except IndexError:
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2BGRA)
    return frame

def apply_color_overlay(frame, intensity=0.5, blue=33, green=54, red=51):
    frame = verify_alpha_channel(frame)
    frame_h, frame_w, frame_c = frame.shape
    sepia_bgra = (blue, green, red, 1)
    overlay = np.full((frame_h, frame_w, 4), sepia_bgra, dtype='uint8')
    cv2.addWeighted(overlay, intensity, frame, 1.0, 0, frame)
```

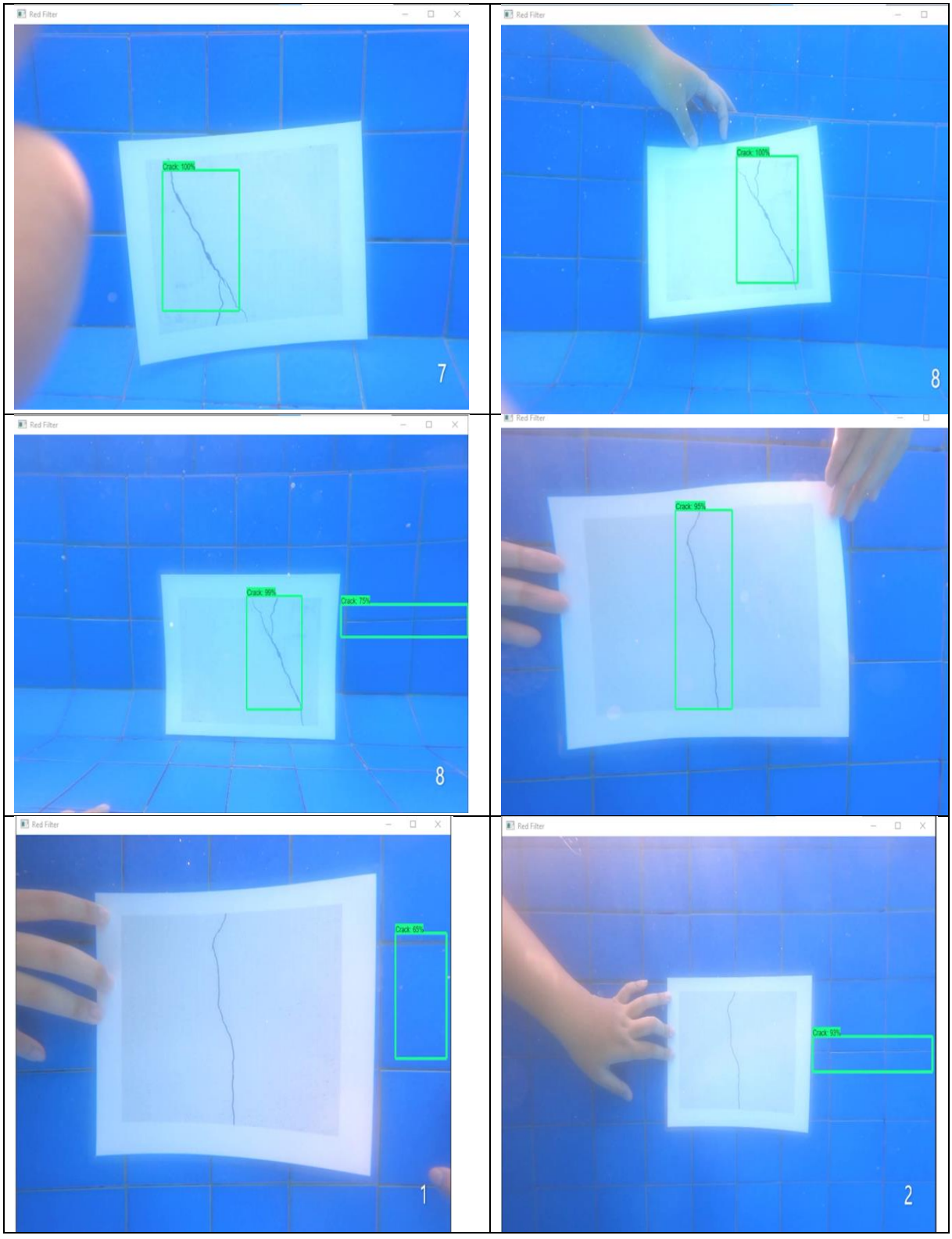
```
    return frame

def load_image_into_numpy_array(image):
    (im_width, im_height) = image.size
    return np.array(image.getdata()).reshape(
        (im_height, im_width, 3)).astype(np.uint8)



PATH_TO_TEST_IMAGES_DIR = 'test_images'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR,
'image{}.jpg'.format(i)) for i in range(1, 3) ]


IMAGE_SIZE = (12, 8)


with detection_graph.as_default():
    with tf.Session(graph=detection_graph) as sess:
        while True:
            ret, image_np = cap.read()
            image_np_expanded = np.expand_dims(image_np, axis=0)
            image_tensor =
detection_graph.get_tensor_by_name('image_tensor:0')
            boxes =
detection_graph.get_tensor_by_name('detection_boxes:0')
            scores =
detection_graph.get_tensor_by_name('detection_scores:0')
            classes =
detection_graph.get_tensor_by_name('detection_classes:0')
            num_detections =
detection_graph.get_tensor_by_name('num_detections:0')
            # Actual detection.
            (boxes, scores, classes, num_detections) = sess.run([boxes,
scores, classes, num_detections],feed_dict={image_tensor:
image_np_expanded})

            # Visualization of the results of a detection.
            vis_util.visualize_boxes_and_labels_on_image_array(
                image_np,
                np.squeeze(boxes),
                np.squeeze(classes).astype(np.int32),
                np.squeeze(scores),
                category_index,
                use_normalized_coordinates=True,
                line_thickness=8)

            merah = apply_color_overlay(image_np.copy())
            cv2.imshow('Red Filter', cv2.resize(merah, (800, 600)))



            cv2.imshow('object detection', cv2.resize(image_np,(800,
600)))
            if cv2.waitKey(25) & 0xFF == ord('q'):
                cv2.destroyAllWindows()
                break
```

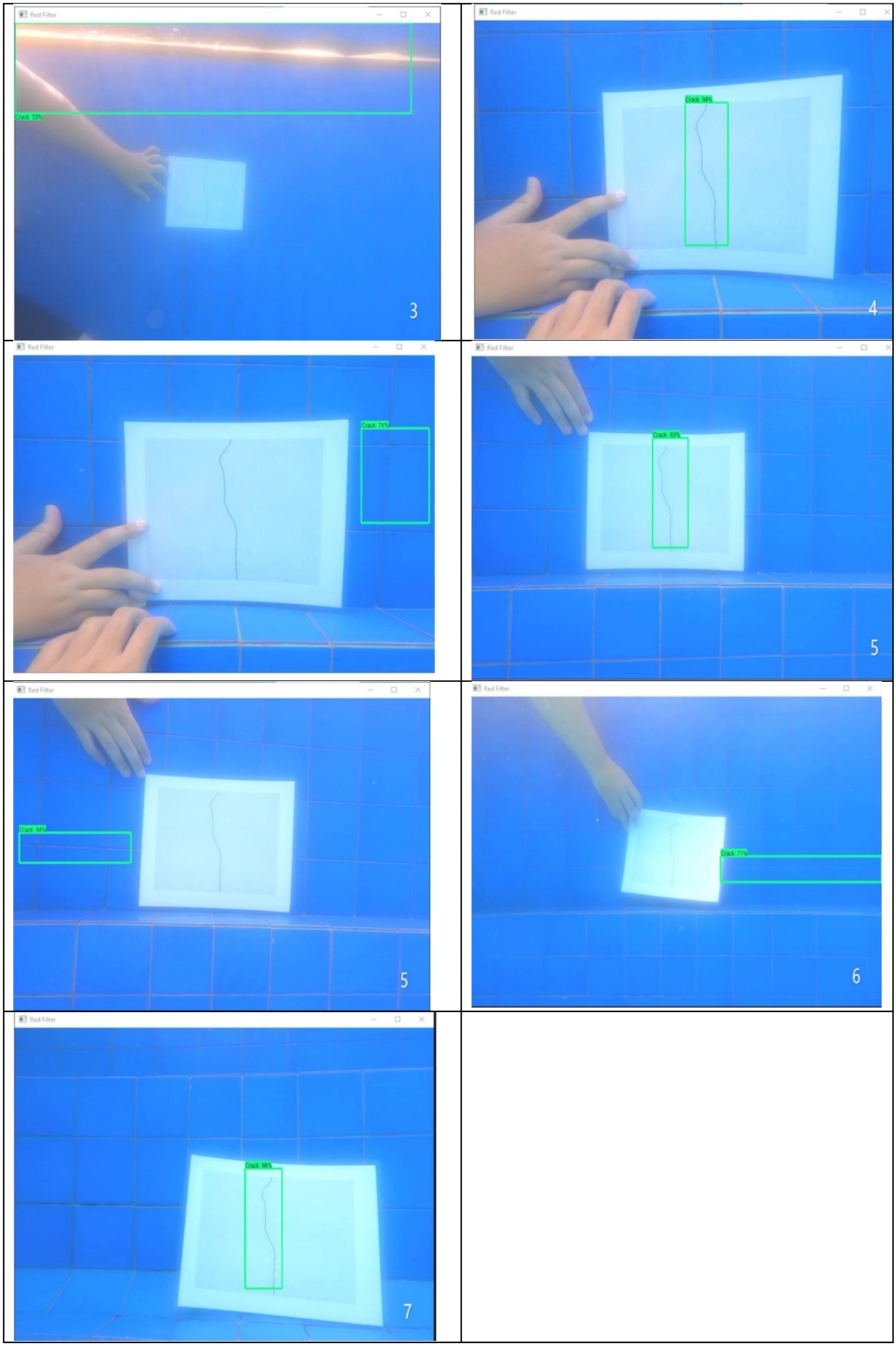
LAMPIRAN 9 *Hasil* uji model *training* 400 data



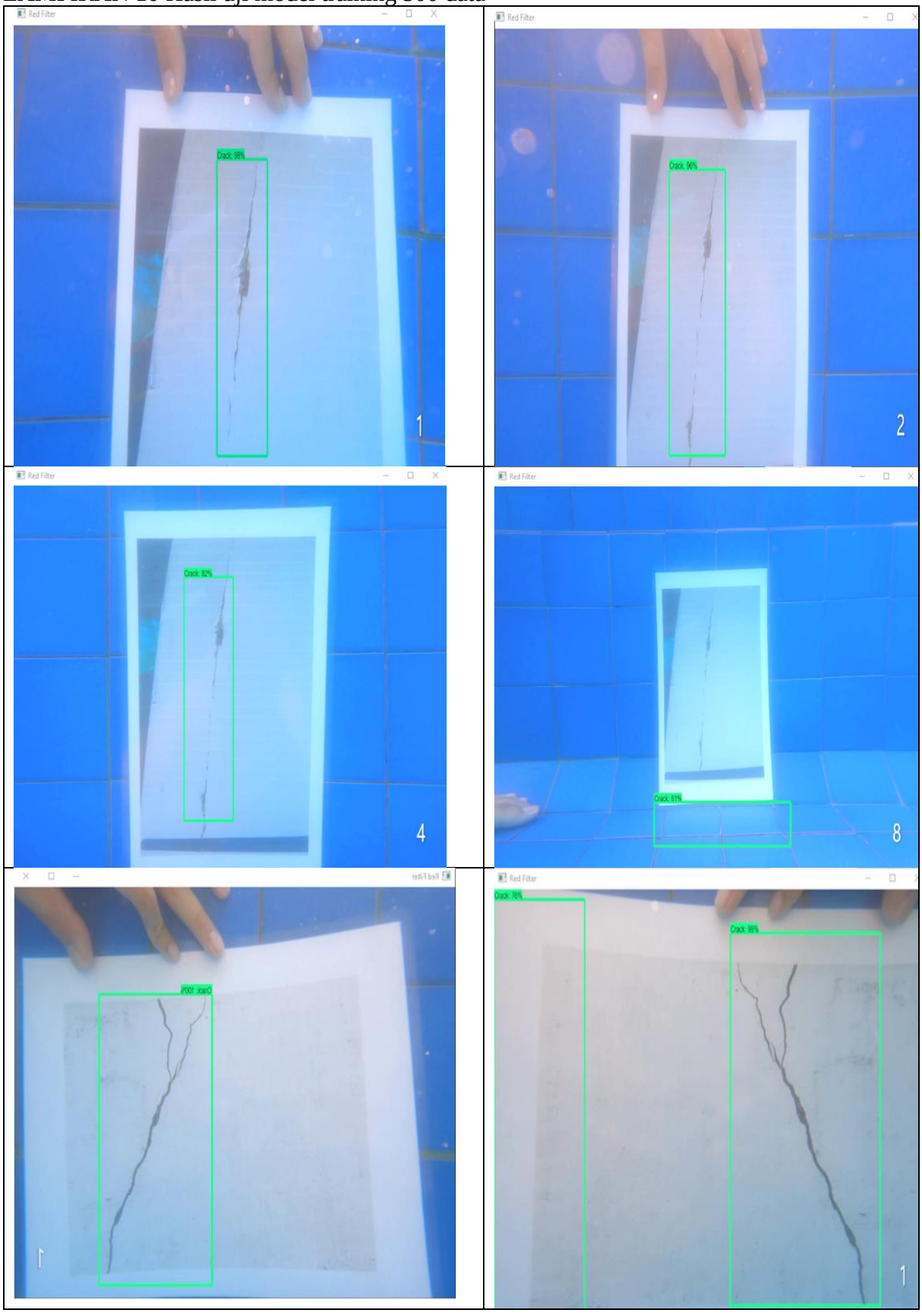

Red Filter
Crack: 100%
Crack: 84%
5
Red Filter
Crack: 85%
6
Red Filter
Crack: 68%
7
Red Filter
Crack: 50%
8
Red Filter
Crack: 87%
8
Red Filter
Crack: 100%
1

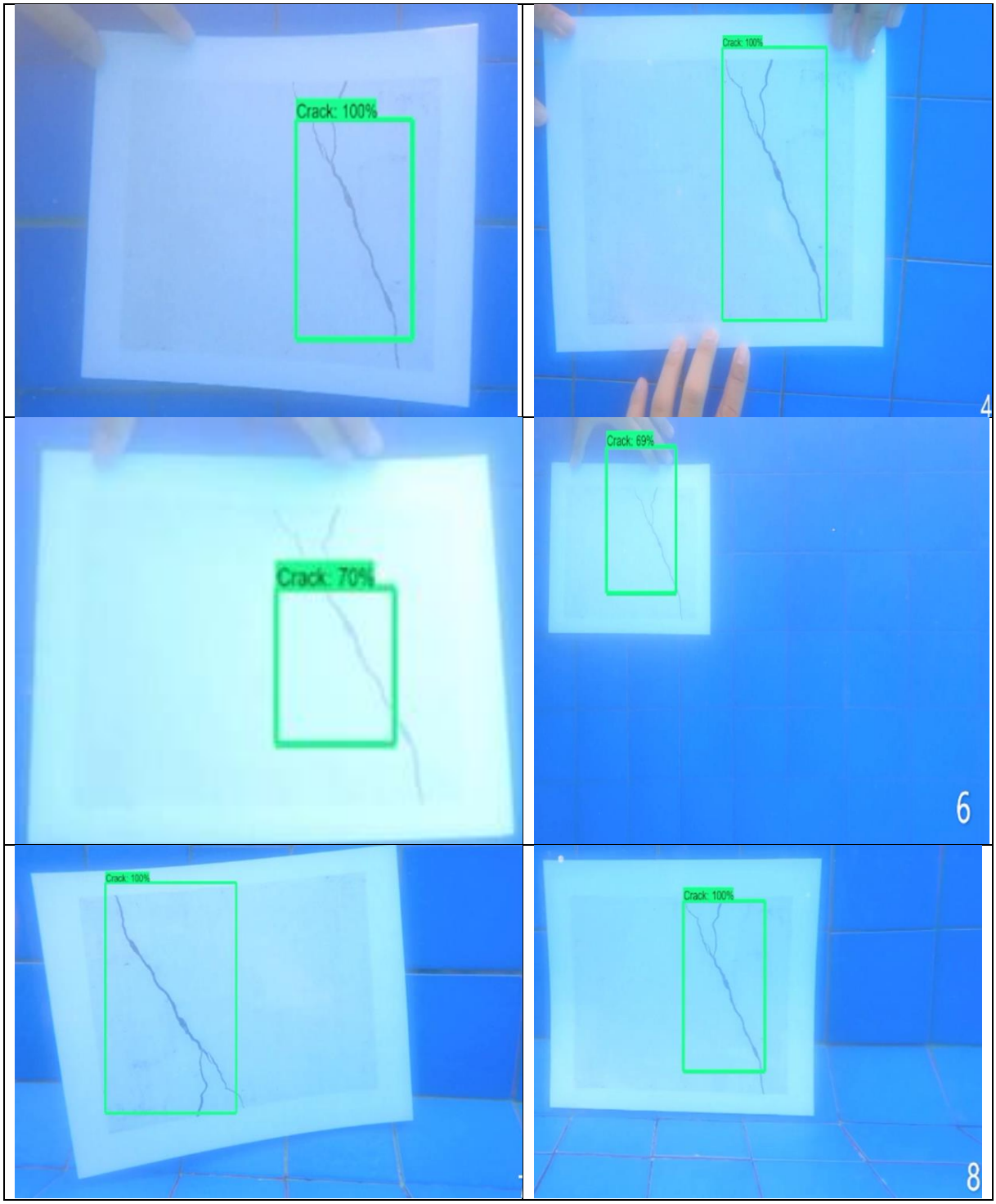

Red Filter
Crack: 64%
Crack: 98%
1
Red Filter
Crack: 97%
Red Filter
Crack: 100%
4
Red Filter
Crack: 100%
Crack: 83%
4
Red Filter
Crack: 98%
5
Red Filter
Crack: 96%

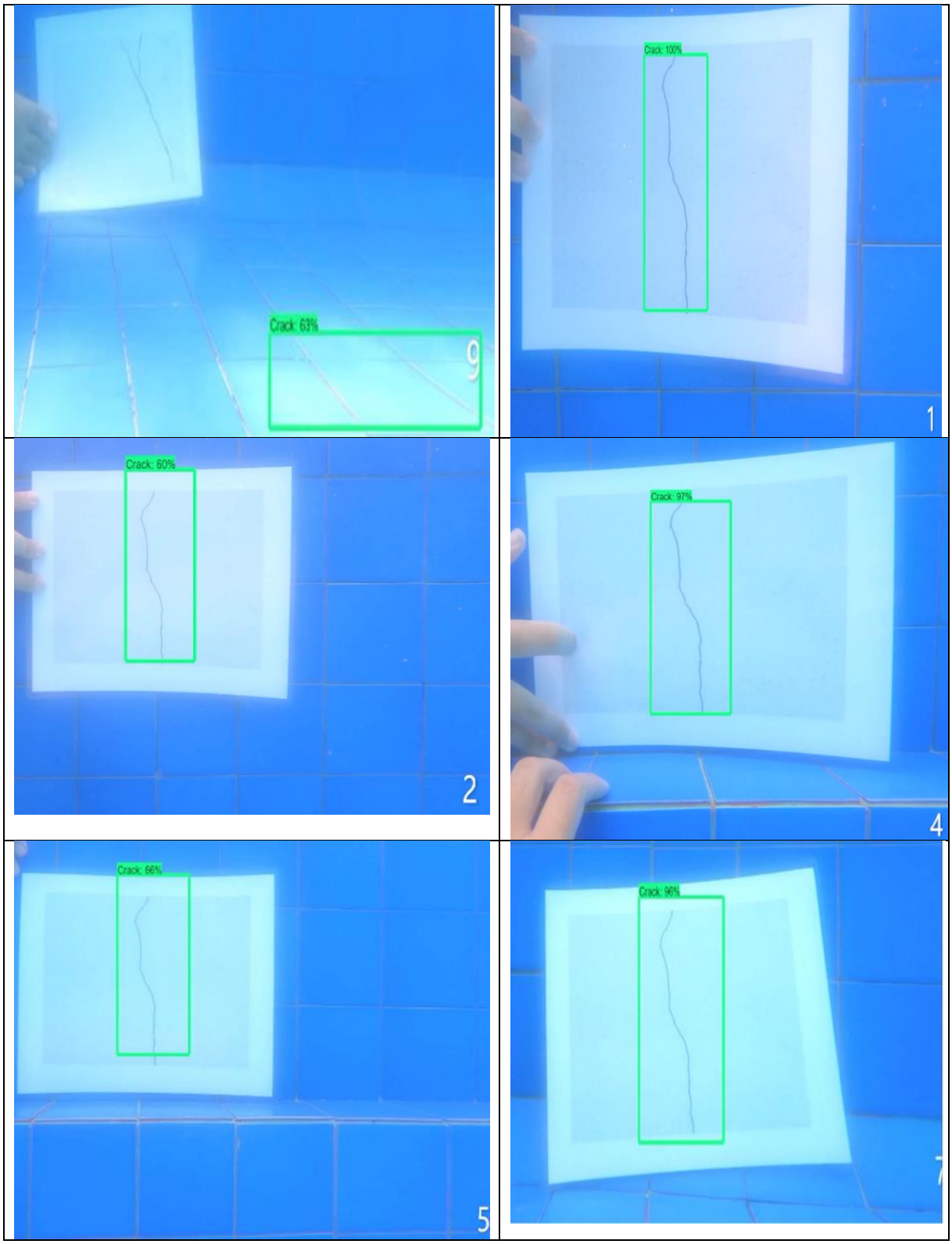

Red Filter
Crack: 100%
7
Red Filter
Crack: 100%
8
Red Filter
Crack: 99%
Crack: 75%
8
Red Filter
Crack: 95%
Red Filter
Crack: 65%
1
Red Filter
Crack: 97%
2


Red Filter
Crack: 53%
3
Red Filter
Crack: 98%
4
Red Filter
Crack: 74%
Red Filter
Crack: 83%
5
Red Filter
Crack: 94%
5
Red Filter
Crack: 71%
6
Red Filter
Crack: 98%
7

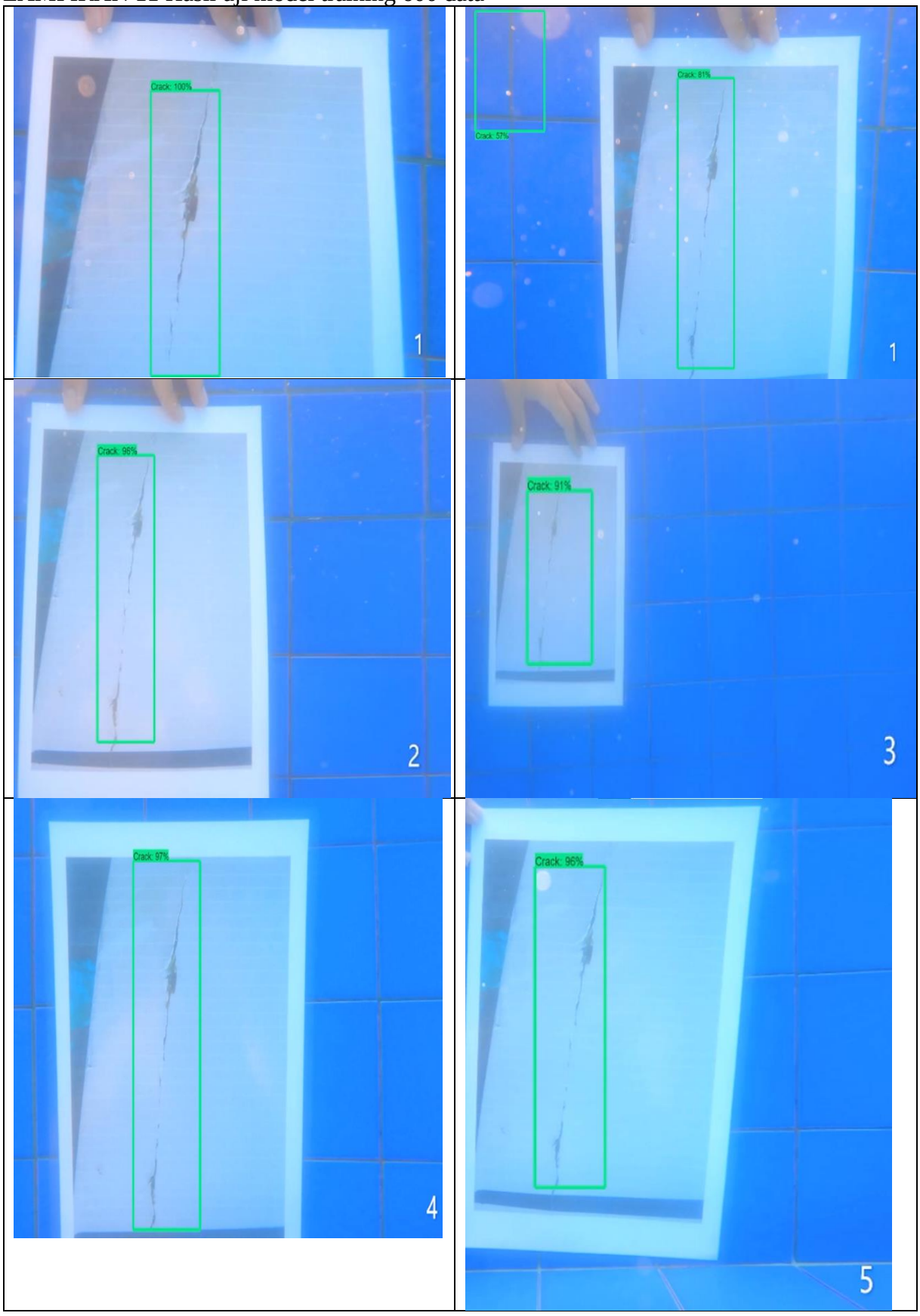
LAMPIRAN 10 Hasil uji model training 500 data

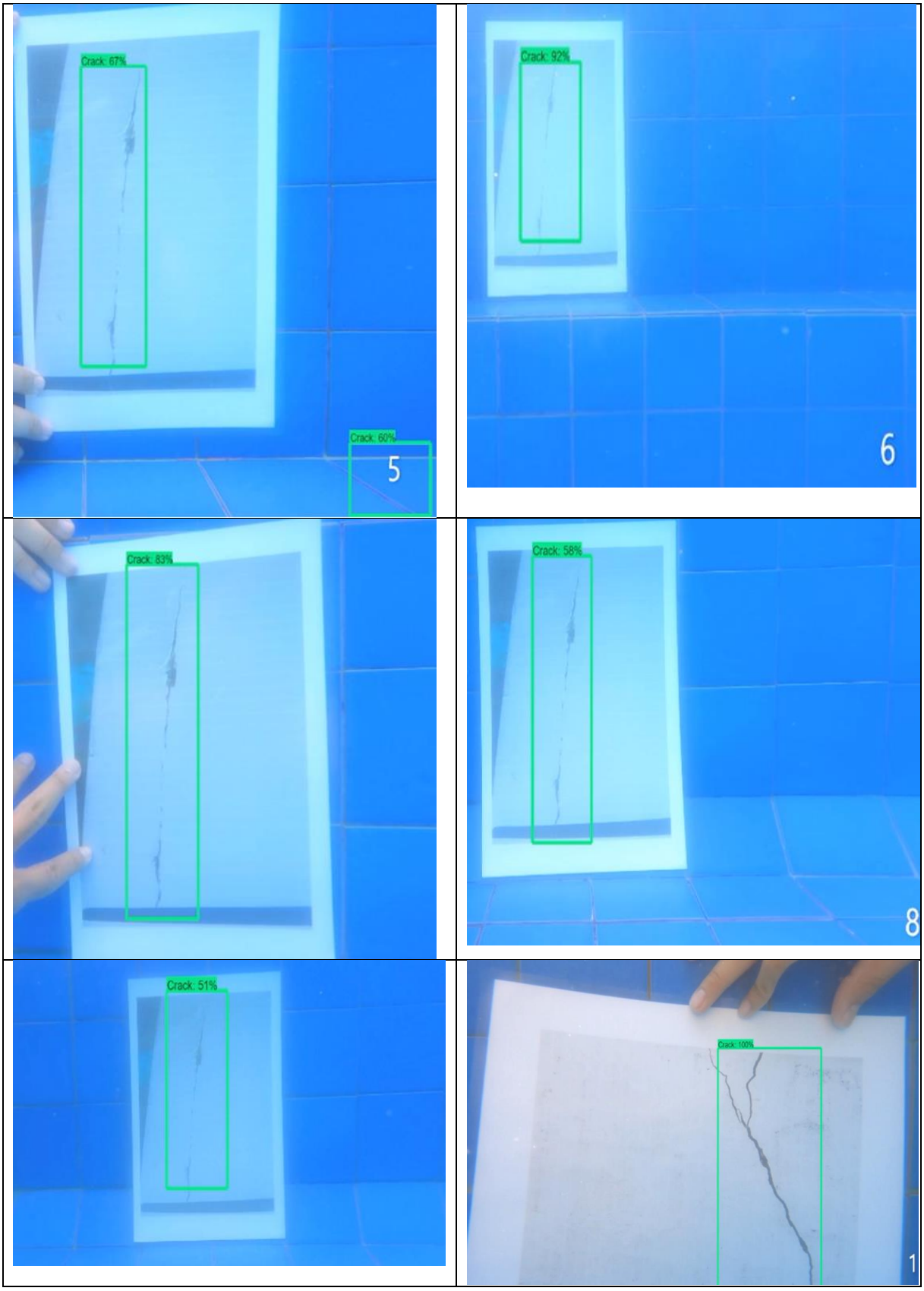


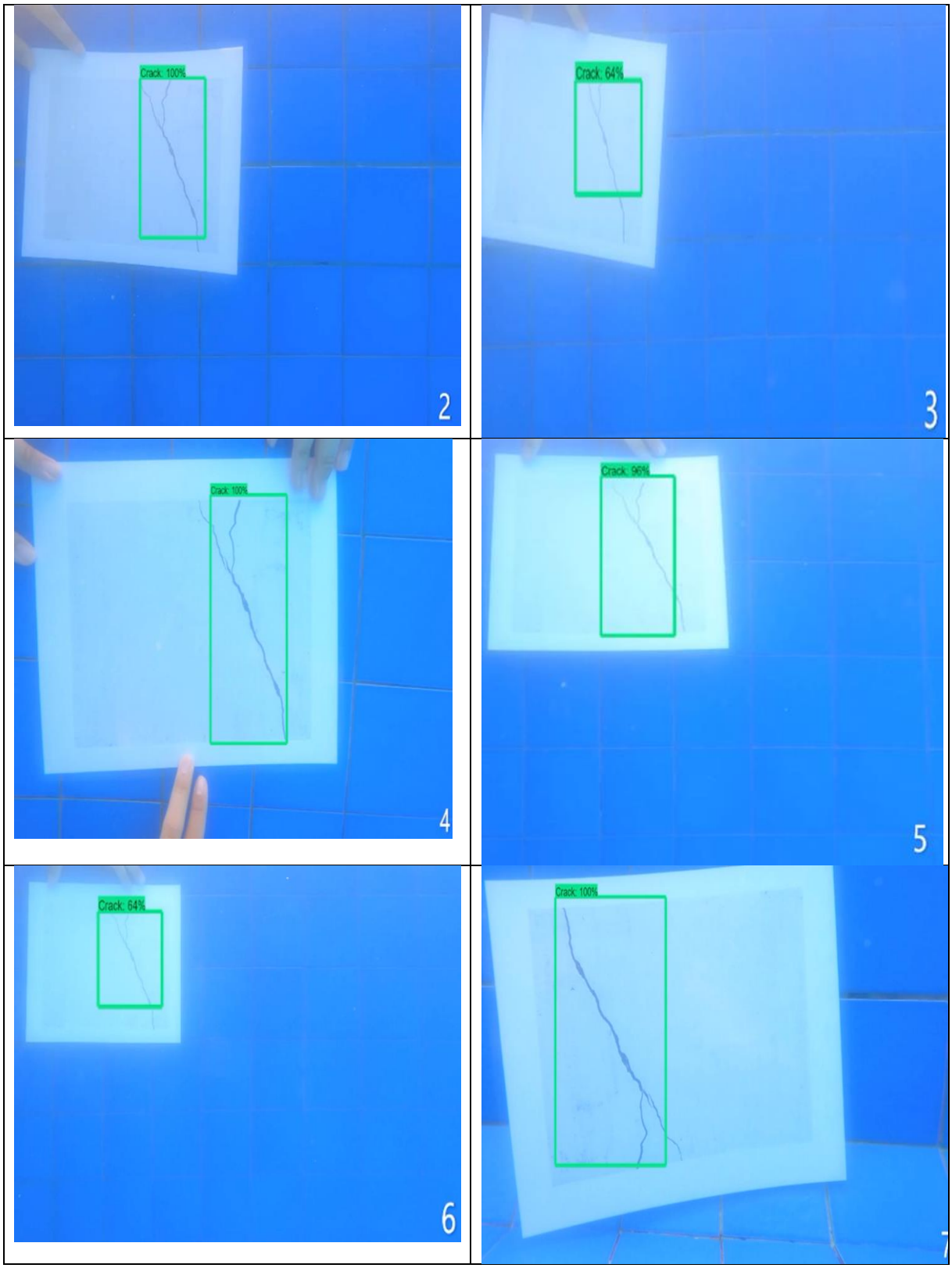

Crack: 100%
Crack: 100%
4
Crack: 70%
Crack: 69%
6
Crack: 100%
Crack: 100%
8

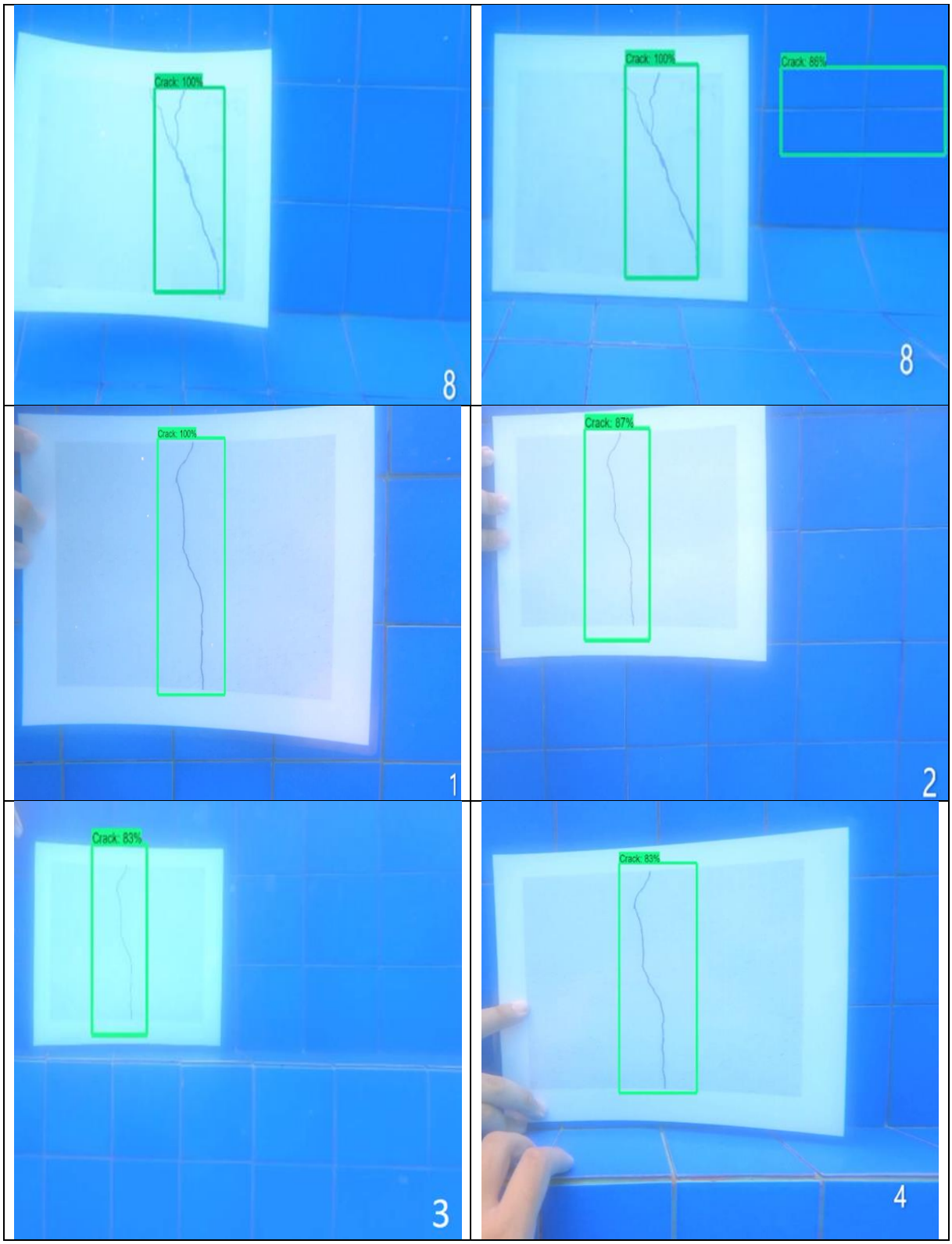

Crack: 63%
9
Crack: 100%
1
Crack: 60%
2
Crack: 97%
4
Crack: 66%
5
Crack: 96%
7

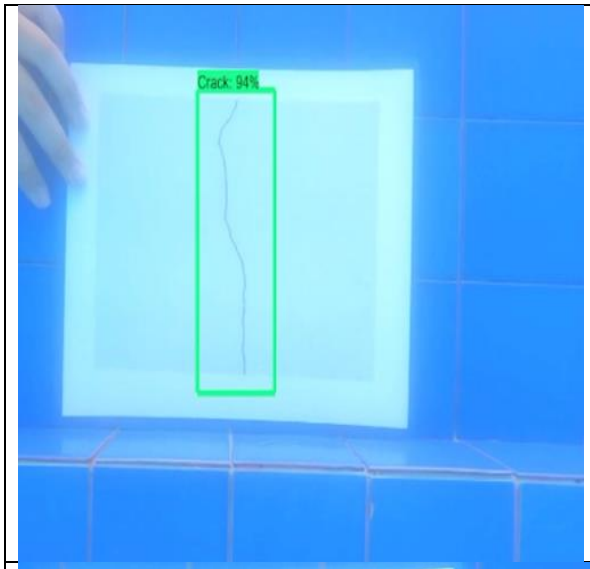
LAMPIRAN 11 Hasil uji model training 600 data

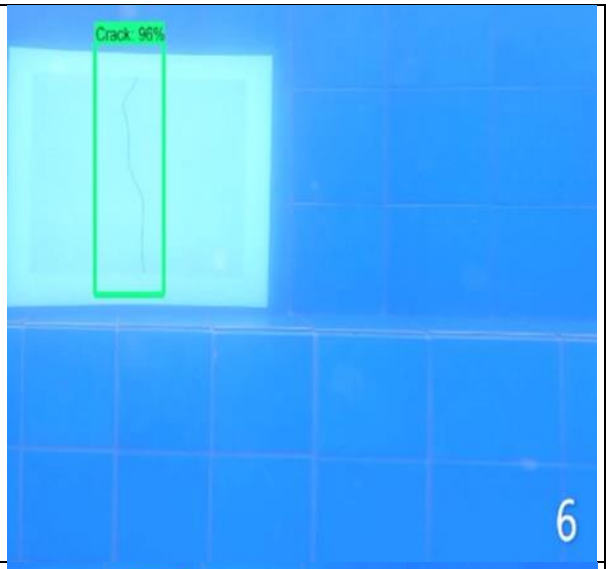


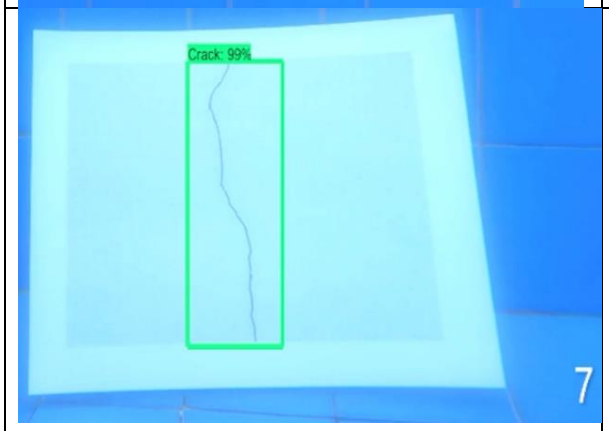

Crack: 67%
Crack: 60%
5
Crack: 92%
6
Crack: 83%
Crack: 58%
8
Crack: 51%
Crack: 100%
1

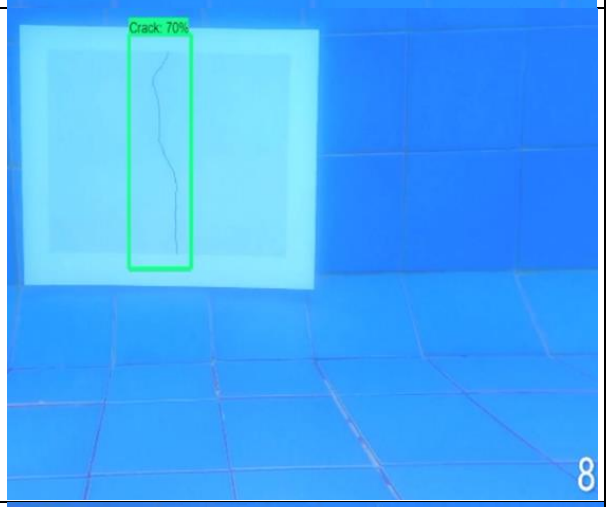

Crack: 100%
2
Crack: 64%
3
Crack: 100%
4
Crack: 96%
5
Crack: 64%
6
Crack: 100%
7

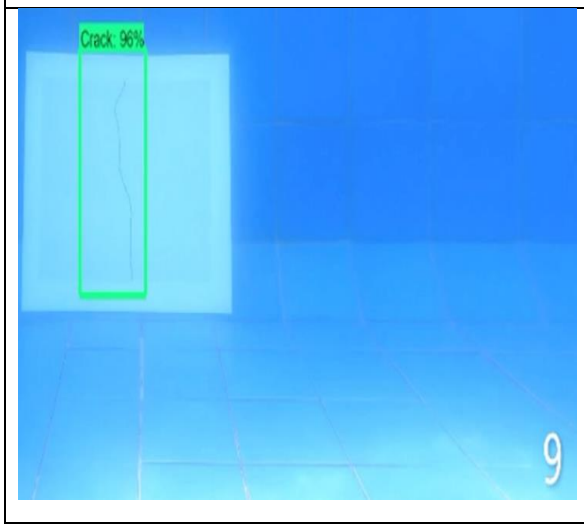

Crack: 100%
8
Crack: 100%
Crack: 86%
8
Crack: 100%
1
Crack: 87%
2
Crack: 83%
3
Crack: 83%
4

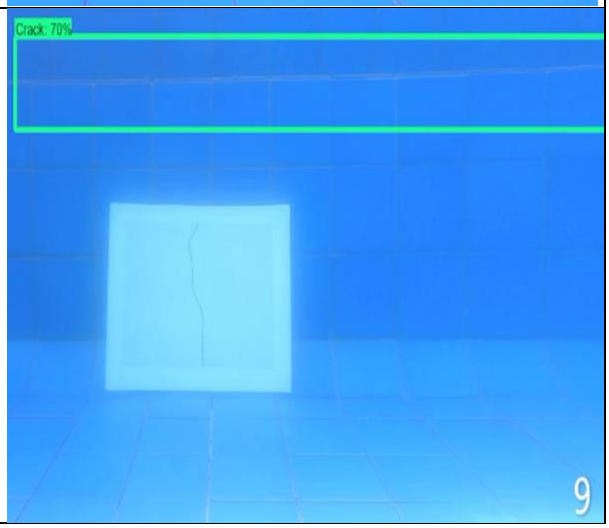

Crack: 94%


Crack: 96%
6


Crack: 99%
7


Crack: 70%
8


Crack: 96%
9


Crack: 70%
9

BIODATA PENULIS

Penulis dilahirkan di Surabaya, 19 Februari 1999, merupakan anak ketiga dari 3 bersaudara. Penulis telah menempuh pendidikan formal yaitu di TK Santa Maria Surabaya, SD Santa Maria Surabaya, SMP Santa Maria Surabaya dan SMA Santa Maria Surabaya. Setelah lulus dari SMA tahun 2017, Penulis mengikuti SBMPTN dan diterima di Departemen Teknik Mesin FTIRS - ITS pada tahun 2018 dan terdaftar dengan NRP 02112040000083.

Semasa kuliah, Penulis kerap aktif pada Lembaga Minat Bakat ITS (LMB ITS) sebagai pengelola *event* Maba *Cup* 2018, pengelola *event* POMITS Voli ITS 2019, dan wakil koordinator POMITS ITS 2019. Selain berkuliah, Penulis juga aktif pada organisasi mahasiswa Keluarga Mahasiswa Katolik ITS (KMK ITS) sebagai pengelola *event* 3C 2019, bendahara Kemah Rohani 2019, koordinator divisi JAJANI 2020, Ketua pelaksana KMK Talks 2020, pengurus departemen *Big Event* KMK ITS 2019 - 2020, wakil ketua KMK ITS 2021 dan beberapa di beberapa kegiatan kemahasiswaan lainnya. Penulis juga menjadi salah satu tim PKM-PI 2021. Penulis sempat aktif dalam beberapa kegiatan seminar yang diselenggarakan oleh Departemen Himpunan Mahasiswa Teknik Mesin (HMM) dan aktif dalam kegiatan ETU 2019.