



TUGAS AKHIR - EC184801

**DETEKSI PEJALAN KAKI PADA ZEBRACROSS
UNTUK PERINGATAN DINI PENGENDARA
MOBIL MENGGUNAKAN MASK R-CNN**

**Agung Wicaksono
NRP 0721 17 4000 0002**

**Dosen Pembimbing
Prof. Dr. Ir. Mauridhi Hery Purnomo, M.Eng.
Dr. Eko Mulyanto Yuniarno, S.T., M.T.**

**DEPARTEMEN TEKNIK KOMPUTER
Fakultas Teknologi ELEKTRO DAN INFORMATIKA CERDAS
Institut Teknologi Sepuluh Nopember
Surabaya 2021**



TUGAS AKHIR - EC184801

**DETEKSI PEJALAN KAKI PADA ZEBRACROSS
UNTUK PERINGATAN DINI PENGENDARA
MOBIL MENGGUNAKAN MASK R-CNN**

**Agung Wicaksono
NRP 0721 17 4000 0002**

**Dosen Pembimbing
Prof. Dr. Ir. Mauridhi Hery Purnomo, M.Eng.
Dr. Eko Mulyanto Yuniarno, S.T., M.T.**

**DEPARTEMEN TEKNIK KOMPUTER
Fakultas Teknologi ELEKTRO DAN INFORMATIKA CERDAS
Institut Teknologi Sepuluh Nopember
Surabaya 2021**

PERNYATAAN KEASLIAN TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir sada dengan judul "**Deteksi Pejalan Kaki pada Zebracross untuk Peringatan Dini Pengendara Mobil menggunakan Mask R-CNN**" adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan karya pihak lain yang saya akui sebagai karya sendiri.

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

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

Surabaya, 14 Juni 2021

Agung Wicaksono
0721 17 4000 0002

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

DETEKSI PEJALAN KAKI PADA ZEBRACROSS UNTUK PERINGATAN DINI PENGENDARA MOBIL MENGGUNAKAN *MASK R-CNN*

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

Oleh: Agung Wicaksono (NRP. 0721 17 4000 0002)

Tanggal Ujian : Juli 2021

Periode Wisuda : September 2021

Disetujui Oleh:

Prof. Dr. Ir. Mauridhi Hery Purnomo, M.Eng. (Pembimbing I)
NIP: 19580916 198601 1 001

Dr. Eko Mulyanto Yuniarso, S.T., M.T. (Pembimbing II)
NIP: 19680601 199512 1 009

Dosen Penguji Pertama. (Penguji I)
NIP: 00000000 000000 0 000

Dosen Penguji Kedua. (Penguji II)
NIP: 00000000 000000 0 000

Dosen Penguji Ketiga. (Penguji III)
NIP: 00000000 000000 0 000

Mengetahui,
Kepala Departemen Teknik Komputer FTEIC - ITS

Dr. Supeno Mardi Susiki Nugroho, ST., MT.
NIP. 19700313 199512 1 001

[Halaman ini sengaja dikosongkan]

ABSTRAK

Nama Mahasiswa : Agung Wicaksono
Judul Tugas Akhir : Deteksi Pejalan Kaki pada *Zebracross* untuk Peringatan Dini Pengendara Mobil menggunakan *Mask R-CNN*
Pembimbing : 1. Prof. Dr. Ir. Mauridhi Hery Purnomo, M.Eng.
 2. Dr. Eko Mulyanto Yuniarno, S.T., M.T.

Dewasa ini, fitur keselamatan pada kendaraan roda empat atau mobil sudah sangat berkembang pesat. Hal tersebut terbukti dengan banyaknya produsen mobil yang menerapkan teknologi seat belt, air bag, adaptive cruise control, electronic stability control, autonomous emergency braking, blind spot monitoring dan lain sebagainya. Namun, fitur yang sudah disebutkan diatas dinilai masih kurang ramah bagi pejalan kaki. Terbukti menurut data dari WHO, terdapat 270.000 pejalan kaki meninggal dunia setiap tahun atau sekitar 22% dari seluruh korban meninggal akibat kecelakan di jalan. Berawal dari permasalahan tersebut, penulis akan melakukan penelitian mengenai pendekripsi pejalan kaki pada zebracross untuk peringatan dini pengendara mobil sebagai topik tugas akhir. Pada tugas akhir ini, terdapat 3 objek yang akan dideteksi yaitu pejalan kaki, zebracross dan pengendara motor dengan menggunakan metode Mask R-CNN. Hasil terbaik yang didapatkan adalah pada penggunaan *ResNet-101* untuk *backbone Mask R-CNN* dengan skor *mAP* sebesar 76.605%, *mAR* sebesar 85.375% serta *F1-Score* sebesar 80.302% .

Kata Kunci: Pejalan Kaki, *Zebracross*, Mask R-CNN, Pengolahan Citra.

[Halaman ini sengaja dikosongkan]

ABSTRACT

*Name : Agung Wicaksono
Title : Pedestrian Detection on Zebracross for Car Driver Early Warning using Mask R-CNN
Advisors : 1. Prof. Dr. Ir. Mauridhi Hery Purnomo, M.Eng.
2. Dr. Eko Mulyanto Yuniaro, S.T., M.T.*

Today, the safety features on four-wheeled vehicles or cars have developed very rapidly. This is evidenced by the number of car manufacturers that apply seat belt technology, airbags, adaptive cruise control, electronic stability control, autonomous emergency braking, blind spot monitoring and so on. However, the features mentioned above are still considered less friendly for pedestrians. According to WHO data, there are 270,000 pedestrians who die every year or about 22% of all victims die due to road accidents. Starting from these problems, the author will conduct research on the detection of pedestrians at zebracross for early warning car drivers as the topic of the final project. In this final project, there are 3 objects to be detected, namely pedestrians, zebracross and motorcyclists using the Mask R-CNN method. The best results obtained are the use of ResNet-101 for backbone Mask R-CNN with a score of mAP of 76,605%, mAR of 85.375% and F1-Score of 80.302%.

Keywords: Pedestrian, Zebracross, Mask R-CNN, Image Processing

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Puji dan syukur kehadirat Tuhan Yang Maha Esa atas segala karunia-Nya, penulis dapat menyelesaikan penelitian ini dengan judul **Deteksi Pejalan Kaki pada Zebracross untuk Peringatan Dini Pengendara Mobil menggunakan Mask R-CNN**.

Penelitian ini disusun dalam rangka pemenuhan bidang riset di Departemen Teknik Komputer ITS, sera digunakan sebagai persyaratan menyelesaikan pendidikan Sarjana. Penelitian ini dapat diselesaikan tidak lepas dari bantuan berbagai pihak. Oleh karena itu, penulis mengucapkan terimakasih kepada:

1. Keluarga, Ibu, Bapak dan Saudara tercinta yang telah memberikan dorongan baik secara spiritual dan material dalam penyelesaian buku penelitian ini.
2. Bapak Dr. Supeno Mardi Susiki Nugroho, ST., MT. selaku Kepala Departemen Teknik Komputer, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember.
3. Bapak Prof. Dr. Ir. Mauridhi Hery Purnomo, M.Eng. selaku dosen pembimbing I dan Bapak Dr. Eko Mulyanto Yuniarso, S.T., M.T. selaku dosen pembimbing II yang selalu memberikan arahan selama mengerjakan penelitian tugas akhir ini.
4. Bapak-ibu dosen pengajar Departemen Teknik Komputer, atas pengajaran, bimbingan, serta perhatian yang diberikan kepada penulis selama ini.
5. Seluruh teman-teman dari angkatan e57, Teknik Komputer, Laboratorium B401, dan B201 Teknik Komputer ITS.

Kesempurnaan hanya milik Allah SWT, untuk itu penulis memohon segenap kritik dan saran yang membangun. Semoga penelitian ini dapat memberikan manfaat bagi kita semua. Amin.

Surabaya, Juni 2021

Agung Wicaksono

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

ABSTRAK	i
ABSTRACT	iii
DAFTAR ISI	vii
DAFTAR GAMBAR	xi
DAFTAR TABEL	xiii
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Permasalahan	2
1.3 Tujuan	2
1.4 Batasan Masalah	2
1.5 Sistematika Penulisan	3
2 TINJAUAN PUSTAKA	5
2.1 Dasar Teori	5
2.1.1 <i>Artificial Intelligence</i>	5
2.1.2 <i>Machine Learning</i>	5
2.1.3 <i>Deep Learning</i>	7
2.1.4 <i>Convolutional Neural Network</i>	8
2.1.5 <i>Convolutional Layer</i>	9
2.1.6 <i>Pooling Layer</i>	13
2.1.7 Fungsi Aktivasi (<i>Non-Linearity</i>)	15

2.1.8	Fully Connected Layer	17
2.1.9	<i>Loss Function</i>	17
2.1.10	<i>Regional-Based CNN</i>	19
2.1.11	<i>Fast R-CNN</i>	20
2.1.12	<i>Faster R-CNN</i>	21
2.1.13	<i>Mask R-CNN</i>	22
2.2	Penelitian Terkait	23
2.2.1	Real-Time Pedestrian Detection With Deep Network Cascades	23
2.2.2	Pedestrian Detection: The Elephant In The Room	24
2.2.3	Fast Vehicle and Pedestrian Detection Using Improved Mask R-CNN	25
3	DESAIN DAN IMPLEMENTASI	27
3.1	Deskripsi Sistem	27
3.2	Pengumpulan <i>Dataset</i> Gambar	28
3.3	Pemisahan Data	29
3.4	<i>Pre-Processing</i>	30
3.5	Membangun Model Mask R-CNN	33
3.6	<i>Training Data</i>	34
3.7	<i>Validating Data</i>	35
3.8	<i>Testing Data</i>	36
4	PENGUJIAN DAN ANALISIS	39
4.1	Pengujian Jenis <i>Backbone</i>	39
4.1.1	Resnet-50	40
4.1.2	Resnet-101	44
4.1.3	MobileNet-V1	48

4.1.4	Perbandingan Hasil Prediksi	53
4.2	Pengujian Perbedaan Waktu	56
4.2.1	Pengujian di Pagi Hari	57
4.2.2	Pengujian di Siang Hari	58
4.2.3	Pengujian di Malam Hari	61
4.2.4	Perbandingan Hasil Evaluasi pada Perbedaan Waktu	62
5	PENUTUP	67
5.1	Kesimpulan	67
5.2	Saran	67
DAFTAR PUSTAKA		69
BIOGRAFI PENULIS		73

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

2.1	Gambaran <i>Supervised Learning</i> [1]	6
2.2	Gambaran <i>Unsupervised Learning</i> [2]	7
2.3	Gambaran <i>Reinforcement Learning</i> [3]	7
2.4	Gambaran Konsep Arsitektur CNN [4]	8
2.5	Contoh Gambar RGB dan Grayscale	10
2.6	Gambaran Operasi Konvolusi pada <i>Convolution Layer</i>	11
2.7	Gambaran Operasi Konvolusi pada <i>Convolution Layer</i> menggunakan <i>Zero Padding</i>	12
2.8	Gambaran Proses <i>Max Pooling</i>	14
2.9	Grafik Fungsi Aktivasi Sigmoid [5]	15
2.10	Grafik Fungsi Aktivasi Tanh [5]	16
2.11	Arsitektur R-CNN [6]	19
2.12	Arsitektur <i>Fast R-CNN</i> [7]	21
2.13	Gambaran Deteksi Objek pada <i>Faster R-CNN</i> [8]	22
2.14	Struktur dari Arsitektur <i>Faster R-CNN</i> [9]	23
3.1	Blok Diagram Metodologi	27
3.2	Contoh Gambar dari Caltech Pedestrian Database	28
3.3	Contoh Pembuatan <i>dataset</i> dari <i>Screenshot Youtube</i>	29
3.4	Visualisasi Pembagian Data	29
3.5	Diagram Alir <i>Pre Processing</i>	31
3.6	Contoh <i>Image Resizing</i>	32
3.7	Contoh <i>Image Augmentation</i>	33
3.8	Blok Diagram Alur Mask R-CNN	34

3.9	Visualisasi <i>K Fold Cross Validation</i>	36
3.10	Diagram Alir <i>Testing Data</i>	37
4.1	Grafik Perubahan <i>Training Loss</i> pada Resnet-50 . .	42
4.2	Grafik Perubahan <i>Validation Loss</i> pada Resnet-50 .	43
4.3	Grafik Perubahan <i>Validation mAP</i> pada Resnet-50 .	44
4.4	Grafik Perubahan <i>Training Loss</i> pada Resnet-101 . .	46
4.5	Grafik Perubahan <i>Validation Loss</i> pada Resnet-101 .	47
4.6	Grafik Perubahan <i>Validation mAP</i> pada Resnet-101	48
4.7	Grafik Perubahan <i>Training Loss</i> pada Mobilenet-V1	50
4.8	Grafik Perubahan <i>Validation Loss</i> pada Mobilenet-V1	51
4.9	Grafik Perubahan <i>Validation mAP</i> pada Mobilenet-V1	52
4.10	Grafik Perbandingan Model dengan <i>backbone</i> yang berbeda	53
4.11	Hasil Uji dari <i>Backbone Resnet-50</i>	54
4.12	Hasil Uji dari <i>Backbone Resnet-101</i>	55
4.13	Hasil Uji dari <i>Backbone Mobilenet-V1</i>	56
4.14	Perbandingan Hasil pada Pagi Hari	57
4.15	Perbandingan Hasil pada Siang Hari	59
4.16	Perbandingan Hasil pada Malam Hari	61
4.17	<i>Confusion Matrix</i> ResNet-50	63
4.18	<i>Confusion Matrix</i> ResNet-101	64
4.19	<i>Confusion Matrix</i> MobileNet-v1	65

DAFTAR TABEL

4.1	Spesifikasi <i>hardware Google Colaboratory</i>	39
4.2	Spesifikasi <i>hardware Komputer yang Digunakan</i>	39
4.3	Konfigurasi Model menggunakan Resnet-50	40
4.4	Konfigurasi Model menggunakan Resnet-101	44
4.5	Konfigurasi Model menggunakan Mobilenet-V1	48
4.6	Tabel Perbandingan Model dengan <i>backbone</i> yang berbeda	52
4.7	tabel Perbandingan Hasil <i>Testing</i>	56
4.8	Perbandingan Hasil Evaluasi pada Pagi Hari	58
4.9	Waktu Prediksi pada <i>File Input</i> Video dalam <i>MM:SS</i>	59
4.10	Perbandingan Hasil Evaluasi pada Siang Hari	60
4.11	Waktu Prediksi pada <i>File Input</i> Video dalam <i>MM:SS</i>	60
4.12	Perbandingan Hasil Evaluasi pada Malam Hari	62
4.13	Waktu Prediksi pada <i>File Input</i> Video dalam <i>MM:SS</i>	62

[Halaman ini sengaja dikosongkan]

BAB I

PENDAHULUAN

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

1.1 Latar Belakang

Mobil merupakan salah satu jenis kendaraan bermotor yang banyak terdapat di Indonesia. Pada tahun 2018 Badan Pusat Statistik mencatat terdapat 16.440.987 mobil penumpang yang berada di Indonesia. Dengan bertambahnya jumlah mobil di Indonesia dari tahun ke tahun, meningkatkan juga jumlah kecelakaan mobil. Fitur keselamatan dan keamanan pada mobil sangat penting bagi para pengendara dan penumpang, sehingga para produsen mobil berusaha meningkatkan teknologi keselamatan dan keamanan pada mobil buatannya. Sebagai contoh beberapa fitur keselamatan dan keamanan yang terdapat pada mobil antara lain, adaptive cruise control, hill strat assist, blind spot monitoring, electronic stability control dan lain sebagainya.

Menurut data dari WHO, terdapat 270.000 pejalan kaki meninggal dunia setiap tahun atau sekitar 22% dari seluruh korban meninggal akibat kecelakan di jalan. Melihat kegiatan para pejalan kaki yang jarang berada di badan jalan, angka tersebut tentu cukup tinggi. Para pejalan kaki hanya menggunakan badan jalan ketika hendak menyebrang jalan lewat zebrafloor. Kelalaian dari pejalan kaki maupun pengendara mobil merupakan faktor utama mengapa angka kematian pejalan kaki cukup tinggi. Salah satu contoh kelalaian pejalan kaki adalah pada saat menyebrang jalan tidak memperhatikan kendaraan yang akan lewat dan atau melihat rambu serta lampu lalu lintas. Di sisi pengendara mobil, kelelahan, kurangnya fokus saat berkendara dan tidak memperhatikan rambu maupun marka dapat berakibat fatal baik kepada pejalan kaki dan

pengendara lain.

Teknologi artificial intelligent sudah banyak disematkan pada mobil pada masa kini, dibuktikan dengan adanya teknologi adaptive cruise control, hill start assist dan lain sebagainya. Artificial intelligent khususnya deep learning tentu dapat digunakan untuk deteksi pejalan kaki di zebracross guna mengurangi jumlah korban akibat kecelakaan. Deteksi pejalan kaki dapat digabungkan dengan buzzer dan atau LED sebagai komponen output untuk mengingatkan kepada pengendara bahwa ada pejalan kaki yang sedang menyebrangi jalan serta mengbalikkan fokus untuk berkendara.

1.2 Permasalahan

Cukup tingginya angka kematian pejalan kaki akibat kecelakaan lalu lintas dan belum adanya deteksi pejalan kaki di zebracross untuk peringatan dini kepada pengendara mobil. Oleh karena itu, diperlukan sebuah sistem yang mampu mendeteksi adanya pejalan kaki yang berada disekitar jalan raya untuk selanjutnya dapat digunakan sebagai peringatan kepada pengendara mobil.

1.3 Tujuan

Berdasarkan rumusan permasalahan di atas, tujuan dari penelitian ini adalah untuk mendeteksi dan mensegmentasikan pejalan kaki dan *zebracross* di jalan raya untuk peringatan dini pengendara mobil menggunakan metode *Mask R-CNN*.

1.4 Batasan Masalah

Batasan masalah yang timbul dari permasalahan Tugas Akhir ini adalah:

1. Menggunakan Mask R-CNN untuk pendekripsi pejalan kaki dan zebracross
2. File Input berupa video dengan format MP4 dengan 30 fps.

1.5 Sistematika Penulisan

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

1. BAB I Pendahuluan

Bab ini berisi uraian tentang latar belakang permasalahan, penegasan dan alasan pemilihan judul, sistematika laporan, tujuan dan metodologi penelitian.

2. BAB II Tinjauan Pustaka

Pada bab ini berisi tentang uraian secara sistematis teori-teori yang berhubungan dengan permasalahan yang dibahas pada penelitian ini. Teori-teori ini digunakan sebagai dasar dalam penelitian, yaitu informasi terkait pejalan kaki, *zebracross*, algoritma *Mask RCNN*, dan teori-teori penunjang lainnya.

3. BAB III Desain dan Implementasi Sistem

Bab ini berisi tentang penjelasan-penjelasan terkait eksperimen yang akan dilakukan dan langkah-langkah pengambilan data jalan raya serta proses deteksi pejalan kaki pada *zebracross*. Guna mendukung hal tersebut, digunakanlah blok diagram atau work flow agar sistem yang akan dibuat dapat terlihat dan mudah dibaca untuk implemtasi pada pelaksanaan tugas akhir.

4. BAB IV Pengujian dan Analisa

Bab ini menjelaskan tentang pengujian eksperimen yang dilakukan terhadap citra jalan raya, proses klasifikasi pejalan kaki dan *zebracross*. Serta terkait tingkat akurasi keberhasilan pengujian yang dilengkapi dengan analisanya.

5. BAB V Penutup

Bab ini merupakan penutup yang berisi kesimpulan yang diambil dari penelitian dan pengujian yang telah dilakukan. Saran dan kritik yang membangun untuk pengembangan lebih lanjut juga dituliskan pada bab ini.

[Halaman ini sengaja dikosongkan]

BAB II

TINJAUAN PUSTAKA

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

2.1 Dasar Teori

2.1.1 *Artificial Intelligence*

Artificial Intelligence mengacu pada simulasi kecerdasan manusia dalam mesin yang diprogram untuk berpikir seperti manusia dan meniru tindakan mereka.[10] Istilah ini juga dapat diterapkan pada mesin apa pun yang menunjukkan ciri-ciri yang terkait dengan pikiran manusia seperti pembelajaran dan pemecahan masalah.

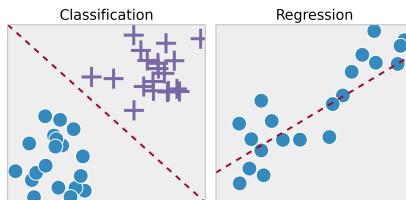
Karakteristik ideal dari *artificial intelligence* adalah kemampuannya untuk merasionalisasi dan mengambil tindakan yang memiliki peluang terbaik untuk mencapai tujuan tertentu. Bagian dari *artificial intelligence* adalah *machine learning*, yang mengacu pada konsep bahwa program komputer dapat secara otomatis belajar dari dan beradaptasi dengan data baru tanpa dibantu oleh manusia. Teknik *deep learning* memungkinkan pembelajaran otomatis ini melalui penyerapan sejumlah besar data tidak terstruktur seperti teks, gambar, atau video.

2.1.2 *Machine Learning*

Machine Learning adalah studi tentang algoritma komputer yang memberikan sistem kemampuan untuk belajar secara otomatis dan dapat meningkatkan kemampuan dari pengalaman yang sudah didapatkan [11]. Hal ini umumnya dilihat sebagai sub-bidang kecerdasan buatan. Algoritma pembelajaran mesin memungkinkan sistem membuat keputusan secara mandiri tanpa dukungan eksternal. Keputusan semacam itu dibuat dengan menemukan pola dasar

yang berharga dalam data yang kompleks. Berdasarkan pendekatan pembelajaran, jenis data *input* dan *output*, dan jenis masalah yang dipecahkan, ada beberapa kategori utama dari algoritma *machine learning supervised, unsupervised* dan *reinforcement learning*. Ada beberapa pendekatan hibrida dan metode umum lainnya yang menawarkan ekstrapolasi alami dari bentuk masalah pembelajaran mesin. Berikut merupakan penjelasan dari beberapa kategori utama dari algoritma *machine learning*:

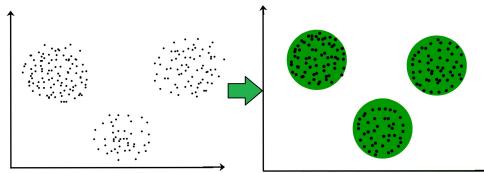
1. *Supervised Learning* diterapkan ketika data dalam bentuk variabel input dan nilai target output. Algoritma akan mempelajari fungsi pemetaan dari *input* ke *output*. Ketersediaan sampel data berlabel dengan skala besar mempunyai nilai yang tinggi dikarenakan masih terdapat kelangkaan *dataset*. Pendekatan ini secara luas dapat dibagi menjadi dua kategori utama yaitu *classification* dan *regression*. Gambar 2.1 menampilkan visualisasi dari *classification* dan *regression* pada *Supervised Learning*



Gambar 2.1: Gambaran *Supervised Learning*[1]

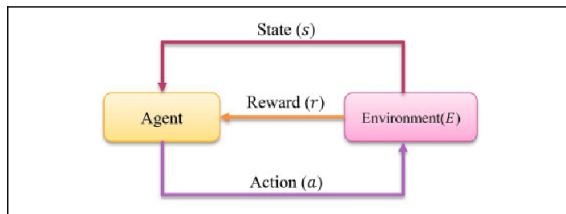
2. *Unsupervised Learning* diterapkan ketika data hanya tersedia dalam bentuk *input* dan tidak ada variabel *output* yang sesuai. Algoritma semacam itu memodelkan pola yang mendasari data untuk mempelajari lebih lanjut tentang karakteristiknya. Salah satu jenis utama dari algoritma *unsupervised* adalah pengelompokan. Dalam teknik ini, kelompok yang melekat dalam data ditemukan dan kemudian digunakan untuk memprediksi *output* untuk *input* yang tidak terlihat. Contoh dari teknik ini adalah untuk memprediksi perilaku pembelian pada

pelanggan. Gambar 2.2 merupakan visualisasi dari algoritma *unsupervised learning*.



Gambar 2.2: Gambaran *Unsupervised Learning*[2]

3. *Reinforcement learning* diterapkan ketika tugas yang ada adalah membuat urutan keputusan menuju *reward* akhir. Selama proses *learning*, *artificial agent* mendapat *reward* atau *penalties* atas tindakan yang dilakukannya. Tujuannya adalah untuk memaksimalkan total *reward* yang didapatkan. Gambar 2.3 merupakan visualisasi dari algoritma *reinforcement learning*.



Gambar 2.3: Gambaran *Reinforcement Learning*[3]

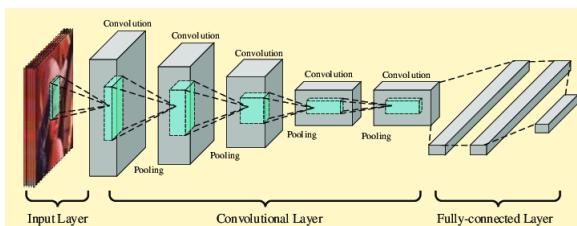
2.1.3 Deep Learning

Deep Learning adalah kelas *machine learning* yang berkinerja jauh lebih baik pada data tidak terstruktur[12]. Teknik *deep learning* mengungguli teknik *machine learning* saat ini. Ini memungkinkan model komputasi untuk mempelajari fitur secara progresif dari data di berbagai level. Popularitas *deep learning* diperkuat karena jumlah data yang tersedia meningkat serta kemajuan perangkat keras yang menyediakan komputer yang kuat.

Arsitektur *deep learning* berkinerja lebih baik daripada jaringan saraf tiruan sederhana, meskipun waktu *learning* dari struktur *deep learning* lebih tinggi dari jaringan saraf tiruan. Namun, waktu *learning* dapat dikurangi dengan menggunakan metode seperti *transfer learning* atau komputasi menggunakan GPU. Salah satu faktor yang menentukan keberhasilan jaringan saraf terletak pada desain arsitektur jaringan yang cermat.

2.1.4 Convolutional Neural Network

Convolutional Neural Network (CNN) adalah jenis khusus dari *multilayer neural network* atau arsitektur *deep learning* yang terinspirasi oleh sistem visual makhluk hidup [13]. CNN sangat cocok untuk berbagai bidang visi komputer dan *natural language processing*. *Convolutional Neural Network* (CNN), juga disebut *ConvNet*, adalah jenis *Artificial Neural Network* (ANN), yang memiliki arsitektur *feed-forward* yang dalam dan memiliki kemampuan generalisasi yang luar biasa dibandingkan dengan jaringan lain dengan lapisan FC (*Fully Connected*), ia dapat mempelajari fitur objek yang sangat abstrak terutama data spasial dan dapat mengidentifikasinya dengan lebih efisien. Model CNN yang dalam terdiri dari satu set lapisan pemrosesan yang dapat mempelajari berbagai fitur data *input* (misalnya gambar) dengan beberapa tingkat abstraksi seperti yang ditampilkan pada Gambar 2.4. Lapisan inisiatör mempelajari dan mengekstrak fitur tingkat tinggi (dengan abstraksi yang lebih rendah), dan lapisan yang lebih dalam mempelajari dan mengekstrak fitur tingkat rendah (dengan abstraksi yang lebih tinggi).



Gambar 2.4: Gambaran Konsep Arsitektur CNN [4]

Convolutional Neural Network memiliki beberapa keunggulan

dibanding dengan jaringan saraf tiruan lainnya dalam konteks visi komputer, antara lain :

1. Salah satu alasan utama untuk mempertimbangkan CNN dalam kasus tersebut adalah fitur pembagian bobot dari CNN, yang mengurangi jumlah parameter yang dapat dilatih dalam jaringan, yang membantu model untuk menghindari *overfitting* dan juga untuk meningkatkan generalisasi.
2. Pada CNN, lapisan klasifikasi dan lapisan ekstraksi fitur melakukan proses *learning* secara bersama-sama, yang membuat output model lebih terorganisir dan membuat output lebih bergantung pada fitur yang diekstraksi.
3. Implementasi pada jaringan dengan ukuran yang besar akan lebih sulit dilakukan dengan menggunakan jenis jaringan saraf lain daripada menggunakan *Convolutional Neural Network*

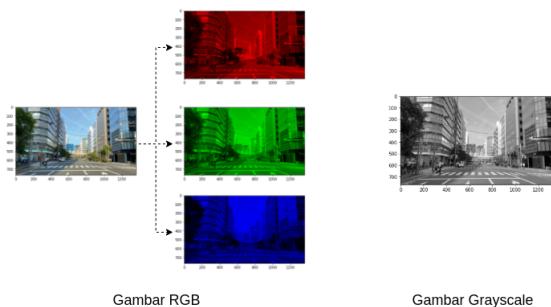
2.1.5 *Convolutional Layer*

Convolutional layer adalah komponen terpenting dari arsitektur CNN mana pun. Ini berisi satu set kernel convolutional (juga disebut filter), yang dililitkan dengan gambar input (metrik N-dimensi) untuk menghasilkan peta fitur keluaran. Kernel dapat digambarkan sebagai kisi nilai atau angka diskrit, di mana setiap nilai dikenal sebagai bobot kernel. Selama awal proses pelatihan model CNN, semua bobot kernel ditetapkan dengan angka acak (pendekatan yang berbeda juga tersedia untuk inisialisasi bobot). Kemudian, dengan setiap periode *learning*, bobot disetel dan kernel belajar mengekstrak fitur yang memberikan informasi mengenai data.

Pada *Convolutional layer* terdapat istilah kernel yang mempunyai peran penting dalam proses yang dinamakan *convolutional operation*. Kernel dapat digambarkan sebagai kisi nilai atau angka diskrit, di mana setiap nilai dikenal sebagai bobot kernel ini. Selama awal proses pelatihan model CNN, semua bobot kernel ditetapkan dengan angka acak (pendekatan yang berbeda juga tersedia di sana untuk menginisialisasi bobot). Kemudian, dengan setiap periode

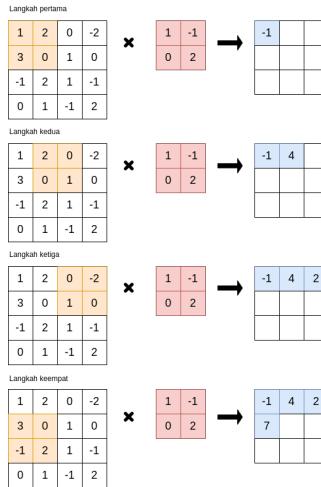
pelatihan, bobot disetel dan kernel belajar mengekstrak fitur yang berarti.

Sebelum kita mempelajari lebih jauh, mari kita pahami dulu format *input* CNN. Berbeda dengan *neural network* klasik lainnya dimana *input*-nya dalam format vektor, di CNN *input*-nya adalah gambar multi *channel*. Misalnya untuk gambar RGB seperti pada Gambar 2.5 mempunyai 3 *channel* dan untuk gambar *grayscale* hanya mempunyai satu *channel*.



Gambar 2.5: Contoh Gambar RGB dan Grayscale

Sekarang, dalam operasi konvolusi, sebagai contoh diambil kernel 2×2 lalu diimplementasikan ke semua gambar 4×4 secara horizontal maupun vertikal dan sepanjang proses berlangsung dilakukan *dot product* antara kernel dan gambar *input* dengan mengalikan nilai yang sesuai dari keduanya dan dijumlahkan semua nilai untuk menghasilkan satu nilai skalar di *feature map output*. Proses ini berlanjut hingga kernel tidak dapat lagi digeser lebih jauh. Untuk memahaminya secara lebih jelas, mari kita lakukan beberapa perhitungan awal yang dilakukan pada setiap langkah secara grafis seperti yang ditunjukkan pada Gambar 2.6, di mana setiap nilai kernel 2×2 (ditunjukkan dengan warna merah) berada dikalikan dengan wilayah berukuran sama (ditunjukkan dengan warna jingga) dalam gambar input 4×4 dan nilai yang dihasilkan dijumlahkan untuk mendapatkan entri yang sesuai (ditunjukkan dengan warna biru) di *feature map output* pada setiap langkah konvolusi.



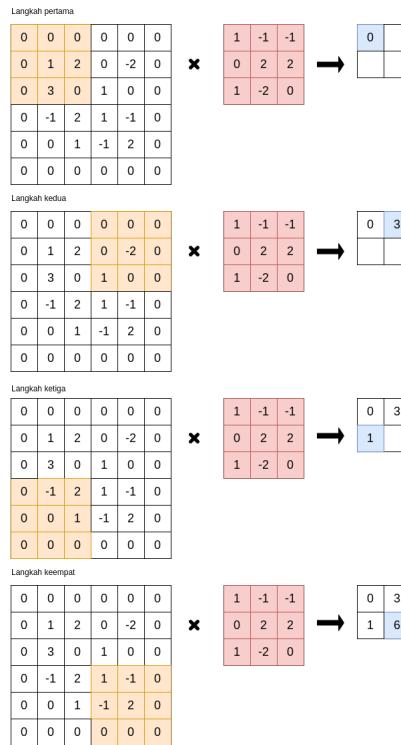
Gambar 2.6: Gambaran Operasi Konvolusi pada *Convolution Layer*

Dalam contoh di atas, diterapkan operasi konvolusi tanpa padding ke gambar masukan dan dengan *stride* (yaitu ukuran langkah yang diambil sepanjang posisi horizontal atau vertikal) sama dengan 1 ke kernel. Tapi bisa digunakan nilai *stride* lain (bukan 1) dalam operasi konvolusi. Hal yang terlihat adalah jika kita meningkatkan langkah operasi konvolusi, itu menghasilkan *feature map* berdimensi lebih rendah. *Padding* penting untuk memberikan informasi ukuran batas dari input gambar jika tidak, tanpa menggunakan *padding* apa pun, fitur sisi tepi akan terhapus terlalu cepat. *Padding* juga digunakan untuk meningkatkan ukuran gambar *input*, akibatnya ukuran *feature map output* juga meningkat. Gambar 2.7 memberikan contoh dengan menunjukkan operasi konvolusi dengan *Zero-padding* dan 3 *stride*. Rumus untuk mencari ukuran *feature map* keluaran setelah operasi konvolusi adalah sebagai berikut:

$$h' = \left[\frac{h - f + p}{s} + 1 \right] \quad (2.1)$$

$$w' = \left\lceil \frac{w - f + p}{s} + 1 \right\rceil \quad (2.2)$$

Dimana h' menunjukkan tinggi dari *feature map output*, w' menunjukkan lebar dari *feature map output*, h menunjukkan tinggi dari gambar *input*, w menunjukkan lebar dari gambar *input*, f adalah ukuran filter, p menunjukkan *padding* dari operasi konvolusi dan s menunjukkan *stride* operasi konvolusi.



Gambar 2.7: Gambaran Operasi Konvolusi pada *Convolution Layer* menggunakan *Zero Padding*

Keuntungan utama dari lapisan konvolusi adalah:

1. ***Sparse Connectivity***: Dalam jaringan saraf yang terhubung penuh, setiap *neuron* dari satu lapisan terhubung dengan setiap *neuron* dari lapisan berikutnya tetapi di CNN sejumlah kecil bobot terdapat di antara dua lapisan. Akibatnya, jumlah koneksi atau bobot yang kita butuhkan kecil, dan jumlah memori untuk menyimpan bobot itu juga kecil, sehingga hemat dalam penggunaan memori. Selain itu, operasi $\text{dot}(\cdot)$ secara komputasi lebih ringan daripada perkalian matriks.
2. ***Weight Sharing***: Di CNN, tidak ada bobot khusus yang ada di antara dua neuron dari lapisan yang berdekatan melainkan semua bobot bekerja dengan setiap piksel dari matriks *input*. Selain mempelajari bobot baru untuk setiap neuron, kita dapat mempelajari satu set bobot untuk semua input dan hal ini secara drastis dapat mengurangi waktu pelatihan.

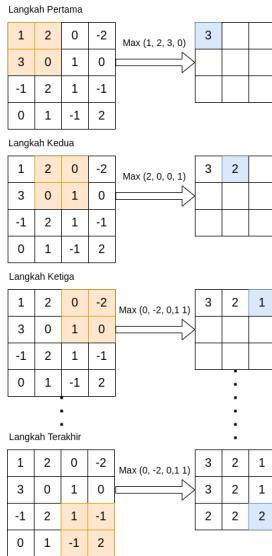
2.1.6 *Pooling Layer*

Pooling layer digunakan untuk membuat sub-sampel *feature map* (dihasilkan setelah operasi konvolusi), yaitu mengambil *feature map* berukuran lebih besar dan mengecilkannya menjadi *feature map* berukuran lebih kecil. Saat menyusutkan *feature map*, *pooling layer* tetap mempertahankan fitur (atau informasi) yang paling dominan di setiap langkah *pools*. Operasi *pooling* dilakukan dengan menentukan ukuran *pooled region* dan langkah operasi, mirip dengan operasi konvolusi. Ada berbagai jenis teknik *pooling* yang digunakan dalam berbagai *pooling layer* seperti *max pooling*, *min pooling*, *average pooling*, *gated pooling*, *tree pooling*, dan lain-lain. *Max Pooling* adalah teknik *pooling* yang paling populer dan banyak digunakan.

Kelemahan utama dari *pooling layer* adalah terkadang menurunkan kinerja CNN secara keseluruhan. Alasan di balik ini adalah bahwa lapisan penyatuhan membantu CNN untuk menemukan apakah fitur tertentu ada dalam gambar masukan yang diberikan atau tidak tanpa memperhatikan posisi yang tepat dari fitur tersebut.

Gambar 2.8 mengilustrasikan contoh yang menunjukkan beberapa langkah awal serta keluaran akhir dari operasi *max-pooling*, di mana ukuran area *pooling* adalah 2×2 (ditunjukkan dalam war-

na jingga, dalam *feature map input*) dengan *stride* sama dengan 1 dan yang sesuai dihitung nilai di *feature map output*. Rumus untuk menemukan ukuran *feature map output* setelah operasi *pooling* seperti di bawah ini:



Gambar 2.8: Gambaran Proses *Max Pooling*

$$h' = \left[\frac{h - f}{s} \right] \quad (2.3)$$

$$w' = \left[\frac{w - f}{s} \right] \quad (2.4)$$

Dimana h' menunjukkan tinggi dari *feature map output*, w' menunjukkan lebar dari *feature map output*, h menunjukkan tinggi dari gambar *input*, w menunjukkan lebar dari gambar *input*, f adalah ukuran area *pooling*, s menunjukkan *stride* operasi *pooling*.

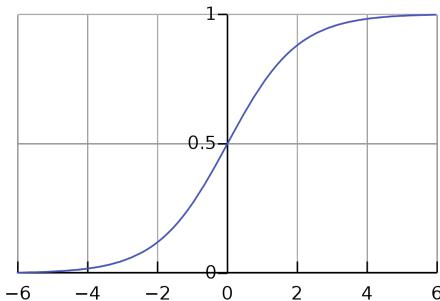
2.1.7 Fungsi Aktivasi (*Non-Linearity*)

Tugas utama dari setiap fungsi aktivasi dalam setiap model berbasis jaringan saraf adalah untuk memetakan *input output*, di mana nilai *input* diperoleh dengan menghitung jumlah terbobot *input neuron* dan selanjutnya menambahkan bias (jika ada bias). Dengan kata lain, fungsi aktivasi memutuskan apakah neuron akan aktif atau tidak untuk *input* yang diberikan dengan menghasilkan *output* yang sesuai. Terdapat beberapa fungsi aktivasi yang digunakan dalam CNN, antara lain :

1. Sigmoid

Fungsi aktivasi sigmoid mengambil bilangan real sebagai inputnya dan mengikat outputnya dalam kisaran [0,1]. Kurva fungsi sigmoid berbentuk 'S' seperti pada Gambar 2.9. Representasi tematik dari sigmoid adalah:

$$f(x)_{sigm} = \frac{1}{1 + e^{-x}} \quad (2.5)$$



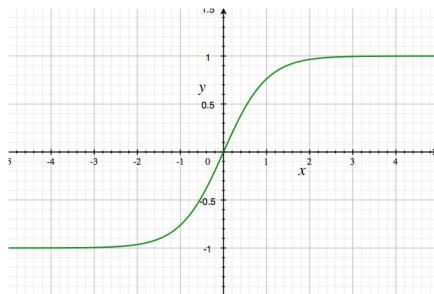
Gambar 2.9: Grafik Fungsi Aktivasi Sigmoid [5]

2. Tanh

Fungsi aktivasi Tanh digunakan untuk mengikat nilai input (bilangan real) dalam kisaran [-1,1] seperti tertampil pada

Gambar 2.10. Representasi matematis dari Tanh adalah:

$$f(x)_{tanh} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.6)$$



Gambar 2.10: Grafik Fungsi Aktivasi Tanh [5]

3. ReLU

Rectifier Linear Unit (ReLU) adalah fungsi aktivasi yang paling umum digunakan dalam Convolutional Neural Networks. Ini digunakan untuk mengubah semua nilai input menjadi bilangan positif. Keuntungan dari ReLU adalah membutuhkan beban komputasi yang sangat minimal dibandingkan dengan yang lain. Representasi matematis dari ReLU adalah:

$$f(x)_{ReLU} = \max(0, x) \quad (2.7)$$

Namun terkadang ada beberapa masalah besar dalam menggunakan fungsi aktivasi ReLU. Misalnya, jika terdapat gradient yang besar pada saat pencarian *error* dengan menggunakan algoritma *back propagation*. Gradien yang besar ini apabila dilewatkhan melalui fungsi ReLU, dapat menyebabkan bobot akan diperbarui sedemikian rupa sehingga neuron tidak pernah diaktifkan lagi. Masalah ini dikenal sebagai masalah "*Dying ReLU*". Untuk mengatasi masalah ini ada beberapa varian ReLU yang tersedia, diantaranya adalah Leaky ReLU dan Noisy ReLU. Tidak seperti ReLU, fungsi aktivasi Leaky

ReLU tidak mengabaikan input negatif sepenuhnya, melainkan menurunkan skala input negatif tersebut. Leaky ReLU digunakan untuk menyelesaikan masalah *Dying ReLU*. Representasi matematis dari Leaky ReLU adalah

$$f(x)_{\text{LeakyReLU}} = \begin{cases} x, & \text{if } x > 0 \\ mx, & \text{if } x \leq 0 \end{cases} \quad (2.8)$$

Di mana m adalah konstanta, yang disebut faktor kebocoran dan umumnya disetel ke nilai kecil (seperti 0,001). Sedangkan Noisy ReLU digunakan distribusi Gaussian untuk membuat ReLU *noisy*. Representasi matematis dari Noise ReLU adalah

$$f(x)_{\text{NoisyReLU}} = \max(x + Y), \text{ with } Y \sim N(0, \sigma(x)) \quad (2.9)$$

2.1.8 Fully Connected Layer

Biasanya bagian terakhir (atau lapisan) dari setiap arsitektur CNN (digunakan untuk klasifikasi) terdiri dari *fully-connected layers*, di mana setiap neuron di dalam lapisan terhubung dengan setiap neuron dari lapisan sebelumnya. Lapisan terakhir dari lapisan *Fully-Connected* digunakan sebagai lapisan keluaran (*classifier*) dari arsitektur CNN. Lapisan *Fully-Connected* adalah jenis jaringan saraf tiruan *feed-forward* (ANN) dan mengikuti prinsip tradisional *Multi-Layer Perceptron neural network* (MLP). Layer FC mengambil *input* dari *convolutional* atau *pooling layer* akhir, yang berupa sekumpulan metrik (*feature map*) dan matrik tersebut diratakan (*flatten*) untuk membuat vektor dan vektor ini kemudian dimasukkan ke dalam layer FC untuk menghasilkan hasil akhir *output* CNN.

2.1.9 Loss Function

Pada *output layer*, dilakukan perhitungan kesalahan prediksi yang dihasilkan oleh model CNN atas sampel *learning* menggunakan beberapa *Loss Function*. *Error* prediksi ini memberitahu jaringan bagaimana prediksinya dari *output* aktual, dan kemudian *error* ini akan dioptimalkan selama proses *learning* model CNN. *Loss function* menggunakan dua parameter untuk menghitung *error*, para-

meter pertama adalah *output* estimasi dari model CNN (juga disebut prediksi) dan yang kedua adalah *output* aktual (juga dikenal sebagai label). Berikut adalah beberapa contoh *loss function* yang banyak digunakan dalam algoritma *deep learning*:

1. *Cross-Entropy Loss Function*

Cross-entropy loss, juga disebut fungsi *log loss* banyak digunakan untuk mengukur kinerja model CNN, yang outputnya adalah probabilitas $p \in \{0, 1\}$. *Loss Function* ini banyak digunakan sebagai alternatif dari *squared error loss function* dalam masalah klasifikasi multi-kelas. *Cross-entropy loss* menggunakan aktivasi softmax di lapisan *output* untuk menghasilkan *output* dalam distribusi probabilitas, yaitu $p, y \in R^N$, di mana p adalah probabilitas untuk setiap kategori *output* dan y menunjukkan *output* yang diinginkan dan probabilitas setiap kelas *output* dapat diperoleh dengan :

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^N e^{a_k}} \quad (2.10)$$

Di mana N adalah jumlah neuron di lapisan *output* dan e^{a_i} menunjukkan setiap *output* yang tidak dinormalisasi dari lapisan sebelumnya dalam jaringan. Lalu, *cross-entropy loss* dapat didefinisikan sebagai:

$$H(p, y) = - \sum_i y_i \log(p_i) \quad (2.11)$$

dimana $i \in [1, N]$

2. *Euclidean Loss Function*

Euclidean Loss juga disebut *mean squared loss* banyak digunakan dalam masalah regresi. *Mean squared loss* antara *output* yang diprediksi $p \in R^N$ dan *output* aktual $y \in R^N$ di setiap neuron dari lapisan *output* CNN didefinisikan sebagai :

$$H(p, y) = \frac{1}{2N} \sum_{i=1}^N (p_i - y_i)^2 \quad (2.12)$$

3. *Hinge Loss Function*

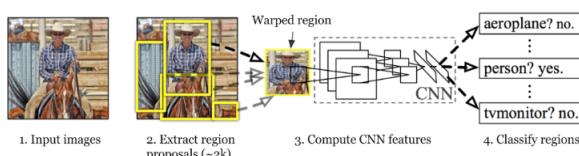
Hinge Loss banyak digunakan dalam masalah klasifikasi biner. Ini digunakan dalam masalah klasifikasi berbasis "margin maksimum", terutama untuk *support vector machine* (SVM). Di sini, *optimizer* mencoba memaksimalkan margin antara dua kelas target. Hinge Loss Function didefinisikan sebagai:

$$H(p, y) = \sum_{i=1}^N \max(0, m - (2y_i - 1)p_i) \quad (2.13)$$

Dimana m adalah margin yang biasanya bernilai sama dengan 1, dan p_i menunjukkan prediksi *output* dan y_i menunjukkan output yang diinginkan

2.1.10 *Regional-Based CNN*

Semenjak *Convolution Neural Network* (CNN) dengan *fully connected layer* tidak mampu menangani frekuensi kemunculan dan multi objek. Salah satu cara untuk mengatasinya adalah dengan menggunakan *sliding window brute force search* untuk memilih wilayah dan menerapkan model CNN pada area tersebut, tetapi masalah dari pendekatan ini adalah bahwa objek yang sama dapat di-representasikan dalam gambar dengan ukuran dan aspek rasio yang berbeda. Dengan mempertimbangkan faktor-faktor tersebut, penerapan algoritma *deep learning* (CNN) pada jumlah *region proposal* yang banyak akan menyebabkan proses komputasi yang dijalankan akan menjadi sangat berat dan rumit.



Gambar 2.11: Arsitektur R-CNN [6]

Ross Girshick dan kawan-kawan pada tahun 2013 mengusulkan arsitektur baru yang disebut R-CNN seperti pada Gambar 2.11

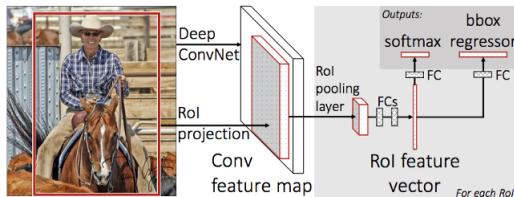
(*Regional-based CNN*) untuk menghadapi tantangan deteksi objek ini [14]. Arsitektur R-CNN menggunakan algoritma *selective search* yang menghasilkan sekitar 2000 *region proposal*. *Regional proposal* ini kemudian diterapkan ke arsitektur CNN untuk menghitung jumlah fitur CNN. Fitur-fitur ini kemudian diteruskan dalam model SVM untuk mengklasifikasikan objek yang ada di *region proposal*. Langkah selanjutnya adalah melakukan regresi pada *bounding box* untuk mengetahui lokasi objek yang ada dalam gambar dengan lebih tepat.

2.1.11 *Fast R-CNN*

Fast R-CNN ditemukan atas metode yang telah ditemukan terlebih dahulu yaitu *R-CNN* untuk mengklasifikasikan proposal objek secara efisien menggunakan *deep convolutional network* [15]. Dibandingkan dengan *R-CNN*, *Fast R-CNN* menggunakan beberapa inovasi untuk meningkatkan kecepatan *learning* dan pengujian sekaligus meningkatkan akurasi deteksi. Fast R-CNN dapat menyelesaikan proses *training* dengan jaringan VGG16 yang sangat dalam membeberikan hasil yang 9 kali lebih cepat dari R-CNN, 213 kali lebih cepat pada waktu pengujian, dan mencapai mAP yang lebih tinggi pada PASCAL VOC 2012. Dibandingkan dengan SPPnet, Fast R-CNN dapat menyelesaikan proses *training* dengan VGG16 3x lebih cepat, pengujian 10x lebih cepat, dan lebih akurat.

Jika pada R-CNN *region proposal* akan melalui proses konvolusi di CNN, sebaliknya pada *Fast R-CNN* gambar *input*-lah yang akan melalui proses konvolusi di CNN seperti yang ditampilkan pada Gambar 2.12. Hasil dari konvolusi pada *Fast R-CNN* berupa *convolutional feature map* yang akan digunakan untuk identifikasi *region proposal* dan menyatukannya ke dalam bentuk persegi. Selanjutnya dengan menggunakan layer *ROI Pooling* akan dibentuk kembali menjadi ukuran yang tetap sehingga dapat dimasukkan ke dalam *fully connected network*. Pada proses deteksi kelas dari *region proposal* dan juga nilai offset dari *bounding box* digunakan *softmax layer* dari *ROI feature vector* yang telah dihasilkan pada proses sebelumnya.

Alasan *Fast R-CNN* lebih cepat daripada R-CNN adalah karena kita tidak perlu memasukkan 2000 *region proposal* ke *convolu-*



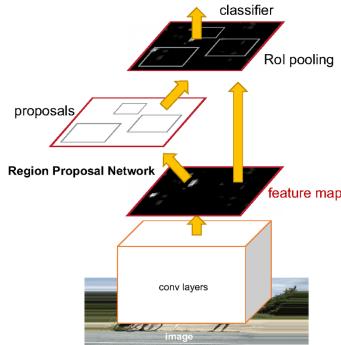
Gambar 2.12: Arsitektur *Fast R-CNN* [7]

tional neural network setiap saat. Sebagai gantinya, operasi konvolusi dilakukan hanya sekali per gambar dan *feature map* dihasilkan operasi tersebut.

2.1.12 *Faster R-CNN*

Pad kedua algoritma *R-CNN* dan *Fast R-CNN* menggunakan *selective search* untuk mengetahui *region proposal*. *Selective search* sendiri memerlukan waktu yang cukup lama dalam penyelesaian prosesnya sehingga mempengaruhi kinerja dari jaringan [16]. Oleh karena itu, dibuatlah algoritma deteksi objek baru dengan menghilangkan algoritma *selective search* dan memungkinkan jaringan mempelajari *region proposal*.

Mirip dengan *Fast R-CNN*, gambar dijadikan sebagai input ke jaringan konvolusi yang menyediakan *convolusional feature map*. Disebabkan waktu eksekusi yang lama saat menggunakan algoritma *selective search* pada *feature map* untuk mengidentifikasi *region proposal*, *Faster R-CNN* menggunakan jaringan terpisah untuk memprediksi *region proposal*. Jaringan terpisah ini dinamakan dengan *Region Proposal Network (RPN)*, dimana *RPN* berbagi *full-image convolutional features* dengan jaringan deteksi yaitu *Fast R-CNN*. *Region proposal* yang diprediksi kemudian dibentuk kembali menggunakan *ROI pooling layer* yang kemudian digunakan untuk mengklasifikasikan gambar di dalam *region proposal* dan memprediksi nilai offset untuk *bounding box*. Gambar ?? merupakan visualisasi dari proses deteksi objek menggunakan *Faster R-CNN*.



Gambar 2.13: Gambaran Deteksi Objek pada *Faster R-CNN* [8]

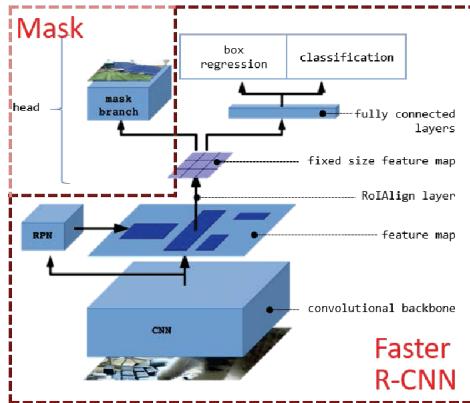
2.1.13 Mask R-CNN

Mask R-CNN adalah Convolutional Neural Network (CNN) dan segmentasi citra termutakhir untuk saat ini. Varian Deep Neural Network ini mendeteksi objek dalam gambar dan menghasilkan *mask* segmentasi berkualitas tinggi untuk setiap *instance* [17]. *Mask R-CNN* dibangun menggunakan *Faster R-CNN* dimana *Faster R-CNN* memiliki 2 *output* untuk setiap objek, label kelas dan *bounding box offset*. *Mask R-CNN* adalah penambahan cabang ketiga yang mengeluarkan *mask* objek seperti yang tertampil pada Gambar ???. *output mask* tambahan berbeda dari *output* kelas dan *bounding box*, yang membutuhkan ekstraksi tata letak spasial yang jauh lebih baik dari suatu objek.

Mask R-CNN merupakan perpanjangan dari *Faster R-CNN* dengan menambahkan cabang untuk memprediksi *tmask* objek (*Region of Interest*) secara paralel dengan cabang yang ada untuk pengenalan *bounding box*. Satu keuntungan sederhana dari *Mask R-CNN* dibandingkan *Faster R-CNN* adalah kenyataan bahwa mudah untuk menggeneralisasi tugas lain seperti estimasi pose. Elemen kunci *Mask R-CNN* adalah penyelarasan piksel-ke-piksel, yang merupakan bagian utama dari *Fast/Faster R-CNN* yang hilang. *Mask R-CNN* mengadopsi prosedur dua tahap yang sama dengan tahap pertama yang identik (yaitu *RPN*). Pada tahap kedua, secara paralel untuk memprediksi kelas dan *box offset*, *Mask R-CNN* juga

mengeluarkan *mask* biner untuk setiap *RoI*. Ini berbeda dengan sistem terbaru, di mana klasifikasi bergantung pada prediksi *mask*.

Mask R-CNN mudah diterapkan dan dilatih karena *Faster R-CNN framework*, yang memfasilitasi berbagai desain arsitektur yang fleksibel. Selain itu, cabang *mask* hanya menambahkan *overhead* komputasi kecil, memungkinkan sistem dan eksperimen yang cepat.



Gambar 2.14: Struktur dari Arsitektur *Faster R-CNN* [9]

2.2 Penelitian Terkait

2.2.1 Real-Time Pedestrian Detection With Deep Network Cascades

Penelitian ini dilakukan oleh Anelia Angelova dan kawan-kawan pada tahun 2015 yang menyajikan pendekatan real-time baru untuk deteksi objek yang mengeksplorasi efisiensi *cascade classifiers* dengan akurasi *deep neural network* [18]. *Deep network* telah terbukti unggul dalam tugas klasifikasi, dan kemampuannya untuk beroperasi pada *raw pixel input* tanpa perlu merancang fitur khusus. Namun, *deep network* terkenal lambat pada waktu inferensi. Dalam *paper* tersebut, Anelia Angelova dan kawan-kawan mengusulkan pendekatan *cascades deep network* dan *fast features*, yang sangat cepat dan sangat akurat. Mereka menerapkannya pada per-

masalah pada deteksi pejalan kaki. Algoritma mereka berjalan secara real-time pada 15 frame per detik. Pendekatan yang dihasilkan mencapai tingkat kesalahan rata-rata 26,2% pada *benchmark* deteksi Caltech Pedestrian.

2.2.2 Pedestrian Detection: The Elephant In The Room

Deteksi pejalan kaki digunakan di banyak aplikasi berbasis citra mulai dari pengawasan video hingga pengemudi otonom. Meskipun mencapai kinerja tinggi, sebagian besar masih belum diketahui seberapa baik detektor yang ada menggeneralisasi data yang tidak terlihat. Hal ini penting karena detektor yang praktis harus siap digunakan dalam berbagai skenario dalam aplikasi. Untuk tujuan tersebut, Irtiza Hasan dan kawan-kawan melakukan studi komprehensif dalam *paper* ini, menggunakan prinsip umum evaluasi *cross-dataset* langsung [19]. Melalui *paper* ini, ditemukan bahwa detektor pejalan kaki terkini yang ada, meskipun berkinerja cukup baik ketika dilatih dan diuji pada kumpulan data yang sama, namun secara umum memiliki performa yang cukup buruk dalam evaluasi *cross-dataset*.

Dalam *paper* ini ditunjukkan bahwa ada dua alasan untuk permasalahan ini. Pertama, desain yang dibuat (misalnya, *anchor settings*) mungkin bias terhadap tolok ukur dalam *training* dan *test pipeline* data tunggal, tetapi akibatnya sebagian besar membatasi kemampuan generalisasi dari keduanya. Kedua, sumber pelatihan umumnya tidak terlalu padat pada pejalan kaki dan mempunyai beragam dalam skenario. Di dalam evaluasi *cross-dataset* langsung, secara mengejutkan, ditemukan bahwa detektor objek dengan tujuan umum, tanpa adaptasi khusus untuk pejalan kaki dalam desain, digeneralisasi jauh lebih baik dibandingkan dengan detektor pejalan kaki terkini yang ada. Lebih lanjut, diilustrasikan bahwa kumpulan data yang beragam dan padat, yang dikumpulkan dengan *crawling web*, berfungsi sebagai sumber pra-pelatihan yang efisien untuk deteksi pejalan kaki. Oleh karena itu, pada *paper* mengusulkan *training pipeline* progresif dan menemukan bahwa *pipeline* tersebut berfungsi dengan baik untuk deteksi pejalan kaki yang berorientasi pada pengemudian otonom. Akibatnya, studi yang dilakukan dalam

makalah ini menunjukkan bahwa lebih banyak penekanan harus diberikan pada evaluasi *cross-dataset* untuk desain masa mendatang pada detektor pejalan kaki.

2.2.3 Fast Vehicle and Pedestrian Detection Using Improved Mask R-CNN

Penelitian ini menyajikan algoritma Mask R-CNN yang sederhana dan efektif untuk deteksi kendaraan dan pejalan kaki yang lebih cepat [20]. Metode ini memiliki nilai praktis untuk sistem peringatan anti-tabrakan dalam mengemudi cerdas. *Deep Neural Network* dengan lebih banyak lapisan memiliki kapasitas yang lebih besar, tetapi juga harus melakukan perhitungan yang lebih rumit. Untuk mengatasi kelemahan ini, penelitian ini mengadopsi jaringan Resnet-86 sebagai *backbone* yang berbeda dari struktur tulang punggung Resnet-101 dalam algoritma Mask R-CNN. Hasilnya menunjukkan bahwa jaringan Resnet-86 dapat mengurangi waktu operasi dan sangat meningkatkan akurasi. Kendaraan dan pejalan kaki yang terdeteksi juga disaring berdasarkan dataset Microsoft COCO. Dataset baru dibentuk dengan menyaring dan melengkapi COCO dataset, yang membuat pelatihan algoritma lebih efisien. Bagian terpenting dari penelitian ini adalah diusulkannya algoritma baru, *Side Fusion FPN*. Parameter dalam algoritma tidak ada perubahan, jumlah perhitungan meningkat kurang dari 0,000001, dan rata-rata presisi (mAP) meningkat 2,00 poin. Hasilnya menunjukkan bahwa, dibandingkan dengan algoritma Mask R-CNN, algoritma pada penelitian ini menurunkan ukuran memori bobot sebesar 9,43%, meningkatkan kecepatan pelatihan sebesar 26,98%, meningkatkan kecepatan pengujian sebesar 7,94%, menurunkan nilai *error* sebesar 0,26, dan meningkatkan nilai mAP sebesar 17,53 poin.

[Halaman ini sengaja dikosongkan]

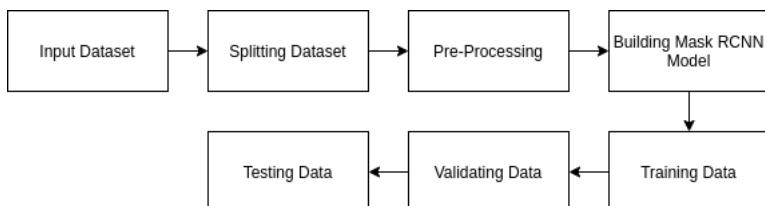
BAB III

DESAIN DAN IMPLEMENTASI

Penelitian ini dilaksanakan sesuai dengan sistem berikut dengan implementasinya. Desain sistem merupakan konsep dari pembuatan dan perancangan infrastruktur dan kemudian diwujudkan dalam bentuk blok-blok alur yang harus dikerjakan. Pada bagian implementasi merupakan pelaksanaan teknis untuk setiap blok pada desain sistem.

3.1 Deskripsi Sistem

Sistem pada tugas akhir ini merupakan implementasi dari salah satu disiplin ilmu *Deep Learning* dan pengolahan citra yang berfungsi untuk mendeteksi adanya pejalan kaki yang berada di pinggir jalan, trotoar dan jalur penyebrangan. Selain pejalan kaki, deteksi juga dilakukan pada jalur penyebrangan atau *zebracross* dengan tujuan untuk memberi informasi bahwa disekitar area tersebut terdapat banyak aktivitas pejalan kaki yang menyebrang jalan. Blok diagram metodologi sistem yang digunakan pada penelitian ini dapat dilihat pada Gambar 3.1.



Gambar 3.1: Blok Diagram Metodologi

3.2 Pengumpulan *Dataset* Gambar

Pada tugas akhir ini, *dataset* yang digunakan didapatkan dengan beberapa cara, antara lain:

1. *Caltech Pedestrian Database*, merupakan kumpulan gambar yang diambil dari sudut pandang pengendara mobil di California Amerika Serikat dengan ukuran 640 x 480 pixel. Terdapat sekitar 250.000 gambar dengan 350.000 *bounding boxes* dan sekitar 2.300 pejalan kaki dengan kriteria unik diberi tanda. Namun, pada *dataset* ini hanya pejalan kaki saja yang diberi label, sehingga perlu dilakukan proses pelabelan ulang sesuai kelas yang diinginkan. Tidak semua gambar pada *dataset* ini diambil untuk digunakan, gambar yang mempunyai objek berupa pejalan kaki dan *zebracross* saja yang akan digunakan. Gambar 3.2 merupakan contoh dari gambar yang terdapat pada *Caltech Pedestrian Database*.



Gambar 3.2: Contoh Gambar dari Caltech Pedestrian Database

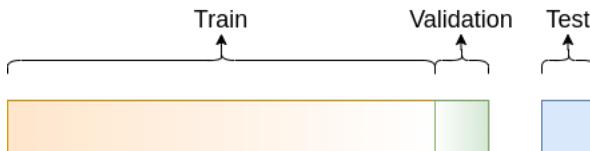
2. Tangkapan layar dari beberapa video *online Youtube*. Pada cara ini, penulis mencari video yang berada pada salah satu *website video streaming* yaitu Youtube dengan persyaratan video diambil dari sudut pandang pengendara mobil yang berkendara pada jalan raya dengan ukuran gambar 1360x768 px. Pada *frame-frame* tertentu dilakukan *screenshot* dan disimpan untuk selanjutnya dilakukan proses pemberian label pada objek-objek yang diinginkan seperti pada Gambar 3.3.
3. Pengambilan gambar secara mandiri menggunakan kamera *smartphone* yang diambil dari sudut pandang pengendara motor dengan ukuran gambar yang diambil sebesar 1280x720 px. Pengambilan gambar dilakukan di jalan-jalan Surabaya. Setelah dilakukan pengambilan gambar, proses selanjutnya adalah pemberian label pada objek-objek yang ingin dideteksi.



Gambar 3.3: Contoh Pembuatan *dataset* dari *Screenshot Youtube*

3.3 Pemisahan Data

Dalam *machine learning* pemisahan data ke beberapa *subset* merupakan suatu hal yang sangat penting. Hal ini dikarenakan setiap *subset* memiliki fungsi masing-masing. Gambar 3.4 merupakan rasio pembagian data ke masing-masing subset.



Gambar 3.4: Visualisasi Pembagian Data

1. *Training Sets*

Training Sets merupakan sampel data yang digunakan untuk melatih model yang sudah kita buat, dalam bidang *Neural Network* bisa disebut juga bobot dan bias. Model yang sudah kita buat mempelajari pola masukan dan keluaran dari data ini.

2. *Validation Sets*

Validation Sets merupakan sampel data yang digunakan untuk mengevaluasi model yang sudah dilatih menggunakan *training sets*. Selain itu, data ini digunakan untuk memperbarui

dan menyempurnakan hyperparameter dari model ke tingkat yang lebih tinggi.

3. Test Sets

Test Sets merupakan sampel data yang digunakan untuk meng-evaluasi model akhir setelah melalui proses *training dan validation*. Apabila pengujian model pada data ini sudah sesuai dengan yang diinginkan, maka proses *learning* sudah selesai. Namun apabila pengujian tidak sesuai dengan yang diharapkan maka diperlukan pengaturan ulang mulai dari proses *training*.

3.4 Pre-Processing

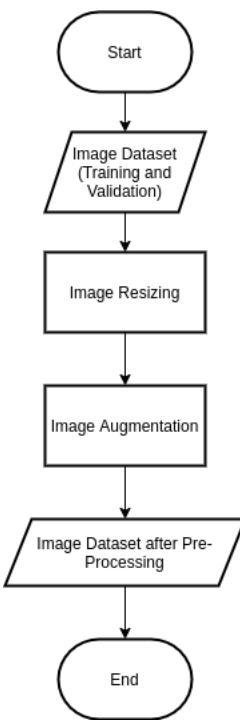
Pada tahap ini, gambar-gambar dari *dataset* akan mengalami proses penyesuaian sebelum masuk ke proses *data training*. Setiap gambar yang akan dijadikan bahan pembelajaran model harus memiliki dimensi dan kedalaman yang sama. Tujuan dari *pre processing* adalah perbaikan data gambar dengan menekan distorsi yang tidak diinginkan atau meningkatkan beberapa fitur gambar yang relevan untuk pemrosesan lebih lanjut. Gambar 3.5 merupakan tahapan dari *pre-processing* gambar *dataset* yang dilakukan.

Berikut merupakan penjelasan mengenai tahapan *pre-processing* yang dilakukan pada tugas akhir kali ini:

1. Image resizing

Langkah awal dari proses *pre-processing* adalah memastikan semua gambar dalam *dataset* kita memiliki ukuran yang sama. Selain itu, sama seperti sebagian besar model dari *neural network* lainnya, metode yang dilakukan penulis juga meng-asumsikan gambar *input* berbentuk persegi. Jadi diperlukan pemeriksaan gambar di awal, apakah gambar sudah berbentuk persegi atau belum. Berbeda dari metode *image resizing* pada model *neural network* lainnya yang menggunakan teknik *cropping* untuk membuat aspek rasio gambar input menjadi persegi, penulis menggunakan metode yang sudah terdapat pada *Mask R-CNN*.

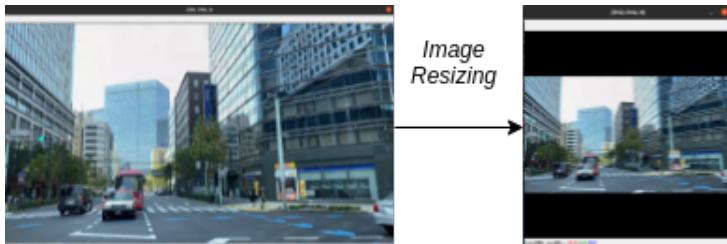
Ukuran gambar yang penulis pilih pada tugas akhir kali ini



Gambar 3.5: Diagram Alir *Pre Processing*

adalah 512x512 pixel. Pemilihan ukuran gambar ini dilakukan untuk mengurangi beban dan waktu saat *training data*. Apabila terdapat gambar pada *dataset* dengan ukuran baik panjang maupun lebar lebih dari 512 pixel. maka gambar akan di *down scaling* sampai ukuran 512 pixel. Sebaliknya, apabila ada gambar pada *dataset* dengan ukuran lebih kecil dari 512 pixel maka akan dilakukan *up scaling* sampai gambar berukuran 512 pixel. Aspek rasio gambar yang sudah melalui proses *scaling* tetap dipertahankan, namun diperlukan penambahan *zero padding* untuk membuat gambar *input* menjadi persegi seperti yang diinginkan.

Gambar 3.6 merupakan salah satu contoh *image resizing* yang



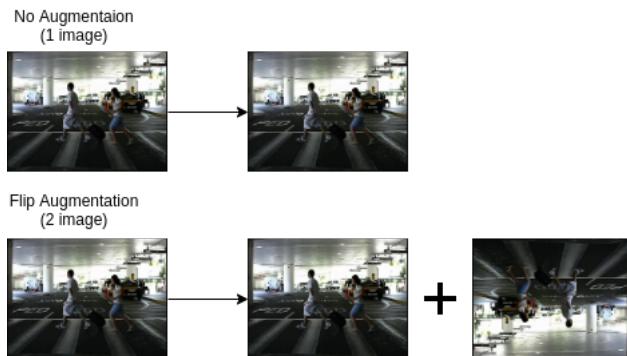
Gambar 3.6: Contoh *Image Resizing*

dilakukan. Gambar *input* (gambar sebelah kiri) mempunyai ukuran 768x1360 dengan kedalaman 3 atau mempunyai format warna RGB. Setelah mengalami *image resizing* (gambar sebelah kanan) ukuran gambar menjadi 290x512. Namun untuk membuat gambar memiliki aspek rasio 1:1 (berbentuk persegi) maka diperlukan penambahan *zero padding* pada bagian atas gambar sebesar 111 pixel dan pada bagian bawah gambar sebesar 111 pixel. Dengan penambahan *padding* seperti itu membuat gambar *input* berbentuk persegi namun tidak mengurangi informasi gambar.

2. *Image Augmentation*

Langkah selanjutnya pada *pre-processing* adalah *image augmentation*. Proses augmentasi yang dilakukan pada tugas akhir ini adalah rotasi dan transformasi. Tujuan dari penggunaan *image augmentation* adalah untuk mengekspos *neural network* ke berbagai variasi, agar dapat mengenali fitur yang akan dilakukan pada proses *training*. Hal tersebut akan sangat membantu *neural network* untuk mengenali variasi yang tidak terdapat pada dataset. Seperti yang terdapat pada Gambar 3.7, tanpa ada augmentasi maka *neural network* hanya mengenali satu kondisi saja. Jika memakai augmentasi gambar seperti transformasi, maka setidaknya *neural network* akan dapat mengenali 2 kondisi. Semakin banyak augmentasi yang digunakan semakin banyak pula kondisi yang bisa dikenali oleh *neural network*. Namun semakin banyak kondisi yang dikenali, semakin lama dan berat proses *training data*

yang dilakukan.

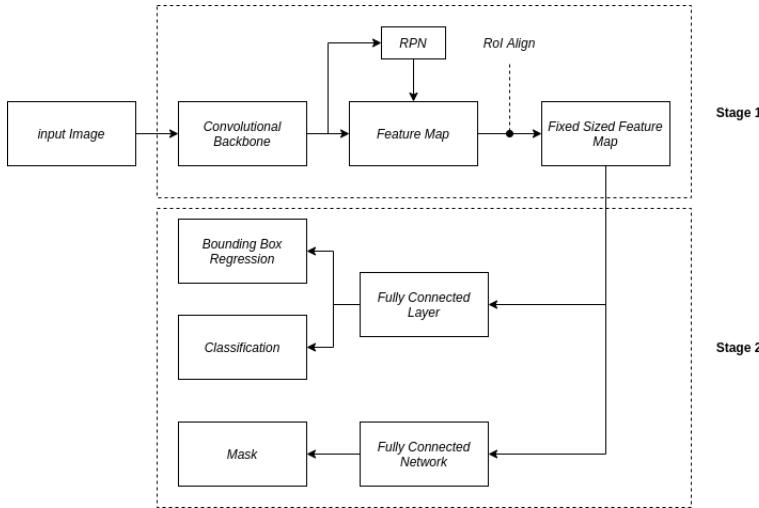


Gambar 3.7: Contoh *Image Augmentation*

3.5 Membangun Model Mask R-CNN

Mask R-CNN merupakan salah satu metode *deep learning* yang dikembangkan dari Faster R-CNN dengan menambahkan satu cabang di tahap akhir untuk menghasilkan *mask* dari objek yang didekksi. Pengembangan tersebut dilakukan untuk memecahkan masalah *instance segmentation* yang terjadi dalam *machine learning* dan pengolahan citra. Dengan kata lain, *mask r-cnn*, dapat memisahkan objek yang berbeda walaupun dalam satu kelas yang sama pada gambar atau video. Selain memberikan hasil berupa *bounding box* dan klasifikasi objek seperti kebanyakan algoritma *object detection* lainnya, *mask r-cnn* juga memberikan *mask* dimana hal ini sangat bermanfaat pada segmentasi objek.

Seperti yang ditampilkan pada Gambar 3.8, pada proses *training* menggunakan *mask r-cnn* dibagi menjadi 2 tahapan. Tahap pertama adalah tahap untuk menghasilkan proposal tentang area di mana mungkin ada objek berdasarkan gambar *input*. Lalu tahap kedua adalah tahap untuk memprediksi kelas objek, memperbaiki *bounding box* dan menghasilkan *mask* di tingkat piksel objek berdasarkan proposal tahap pertama. Kedua tahap terhubung ke struktur *backbone*.



Gambar 3.8: Blok Diagram Alur Mask R-CNN

Backbone adalah *deep neural network* yang memiliki struktur seperti FPN (*Feature Pyramid Network*). *Backbone* terdiri dari *bottom-up pathway*, *up-bottom pathway* dan *lateral connection*. *Bottom-up pathway* dapat berupa berbagai jenis *Convolutional Network*, biasanya berupa *ResNet* atau *VGG*, yang mengekstrak fitur dari *raw images*. *Up-bottom pathway* menghasilkan *Feature Map Pyramid* yang ukurannya mirip dengan *bottom-up pathway*. *Lateral connection* adalah operasi konvolusi dan penjumlahan antara dua *pathway* dengan tingkat yang sesuai. FPN mempunyai kinerja yang lebih baik dari ConvNet tunggal lainnya terutama karena FPN dapat mempertahankan fitur semantik yang sangat baik pada berbagai skala resolusi.

3.6 Training Data

Proses *training data* dilakukan setelah pembuatan model telah selesai dan *dataset* sudah melalui proses *pre-processing*. Pada saat pertama kali menjalankan proses *training*, bobot awal diam-bil dari *pre-trained weight* yang sudah tersedia pada *mask r-cnn*.

Hal ini bisa dilakukan dengan menerapkan metode *transfer learning*. *Transfer learning* sendiri adalah teknik yang sangat efisien untuk melakukan proses *training* atau *retrain* pada *neural network*. Penggunaan *transfer learning* mempunyai keuntungan diantara lain proses *training* pada data baru memakan waktu yang lebih cepat daripada memulai dari awal serta masalah dapat dipecahkan dengan menggunakan training data yang lebih sedikit daripada membangun model dari awal.

Ada beberapa hal yang perlu diperhatikan dalam melakukan pengaturan saat akan menjalankan proses *training* antara lain :

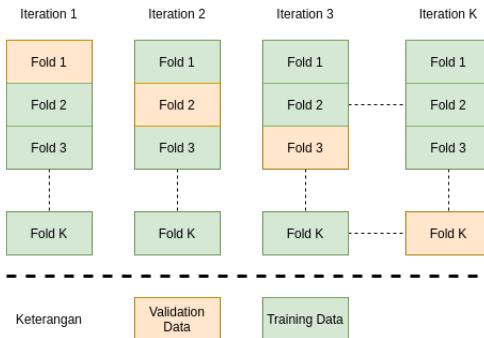
1. *Iteration* adalah banyaknya proses yang dilakukan untuk melakukan *forward* dan *backward pass*. *forward pass* adalah proses dimana *output value* dari *neural network* didapatkan setelah *input value* dari *input-neuron* telah selesai diproses. Sedangkan *backward pass* adalah proses mengkalkulasikan bobot dari *neural network* mulai dari *output neuron* ke *input neuron* untuk mendapatkan *loss* dari setiap *neuron*. Pada *mask r-cnn iteration* bisa disebut juga dengan *step_per_epoch* sesuai dengan yang tercantum pada *file* pengaturan *mask r-cnn*.
2. *Epoch*, ketika seluruh dataset sudah melalui proses *training* pada *neural network* sampai dikembalikan ke awal untuk sekali putaran. Sebagai contoh apabila kita menggunakan *iteration* sebanyak 10 kali, maka satu *epoch* sebanyak 10 *iteration* dan kelipatannya. Pada tugas akhir kali ini penulis menggunakan *epoch* sebanyak 100.

3.7 *Validating Data*

Setelah proses *training* dilakukan, perlu dilakukan apakah model yang dibuat sudah memiliki tingkat akurasi sesuai yang kita inginkan dengan menggunakan teknik validasi. Pada proses inilah, *dataset* yang telah dipisahkan pada proses sebelumnya akan berperan. Evaluasi memungkinkan pengujian model terhadap data yang belum pernah dilihat dan digunakan untuk pelatihan dan dimaksudkan untuk mewakili bagaimana model dapat menyelesaikan permasalahan tersebut. Tahapan pada validasi akan membantu untuk

menemukan parameter terbaik untuk model prediktif dan mencegah dari *overfitting*.

Pada tugas akhir kali ini, digunakan salah satu jenis teknik validasi menggunakan *Cross Validation*. Pada *Cross Validation* data akan dibagi menjadi K lipatan, dimana setiap lipatan akan diambil satu data sebagai *validation data* dan sisanya akan digunakan untuk *training data*. Pemilihan *validation data* dilakukan secara menyilang, dengan ketentuan apabila *training* terjadi pada iterasi ke k maka data yang dipilih untuk validasi adalah data k juga. Gambar 3.9 merupakan visualisasi dari K Fold *Cross Validation*.

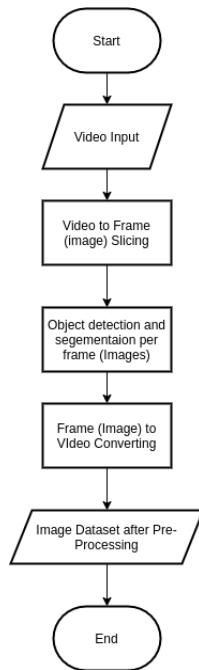


Gambar 3.9: Visualisasi K Fold *Cross Validation*

3.8 Testing Data

Testing data merupakan tahap akhir dalam algoritma *machine learning* secara umum. Pada tahap ini model akan diuji untuk didapatkan keakuratan untuk mendeteksi objek. Data uji yang digunakan pada tugas akhir kali ini adalah data yang berbentuk video. Jadi diperlukan *pre-processing* yang berbeda dengan *training data* dan *validation data*. Data video akan dipecah atau dipotong-potong menjadi format gambar dengan ketentuan 30 *frame per second*. Ketika *data test* sudah dikonversi menjadi bentuk gambar, maka proses deteksi bisa dilakukan. Gambar hasil deteksi berupa gambar asli yang sudah ditambah dengan *bounding box*, *classification*, *mask*. Lalu gambar-gambar tersebut disatukan lagi menjadi format video

dengan ketentuan sama seperti saat pemotongan menjadi format gambar, yaitu 30 *frame per second*. Gambar 3.10 merupakan diagram alir dari proses *testing* yang dilakukan pada *input file* berupa video.



Gambar 3.10: Diagram Alir *Testing Data*

[Halaman ini sengaja dikosongkan]

BAB IV

PENGUJIAN DAN ANALISIS

Pada bab ini dipaparkan hasil pengujian serta analisa dari desain sistem dan implementasi. Pengujian dilakukan guna mengetahui tingkat kesalahan dan menarik kesimpulan dari sistem yang telah dibuat.

Pada proses pengujian digunakan salah satu layanan *Google* yaitu *Google Colaboratory* dengan spesifikasi *hardware* seperti pada Tabel 4.1. Sedangkan untuk spesifikasi *hardware* komputer penulis dapat dilihat pada Tabel 4.2.

Tabel 4.1: Spesifikasi *hardware* *Google Colaboratory*

Procesor	Intel Xeon Processor @ 2.3 GHz
Graphic Card	Tesla K80 12 GB GDDR5 VRAM
RAM	16 GB

Tabel 4.2: Spesifikasi *hardware* Komputer yang Digunakan

Procesor	Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz
Graphic Card	Nvidia GeForce GTX 1650 4 GB GDDR6
RAM	8 GB

Pengujian dilakukan dengan membagi model ke beberapa jenis *backbone* yang digunakan, antara lain Resnet-50, Resnet-101, dan Mobilenet-V1.

4.1 Pengujian Jenis *Backbone*

Pengujian pada jenis *backbone* bertujuan untuk mengetahui performa dan akurasi dari setiap model yang dihasilkan dengan

backbone yang berbeda.

4.1.1 Resnet-50

Tabel 4.3 merupakan parameter-parameter yang digunakan untuk membuat model Mask R-CNN dengan menggunakan *backbone* Resnet-50.

Tabel 4.3: Konfigurasi Model menggunakan Resnet-50

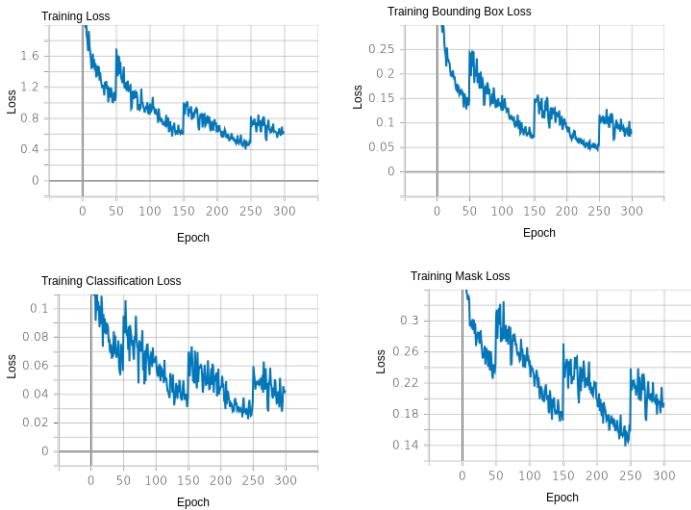
Pengaturan Model Resnet-50	
BACKBONE	resnet50
BACKBONE_STRIDES	[4, 8, 16, 32, 64]
BATCH_SIZE	1
BBOX_STD_DEV	[0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE	None
DETECTION_MAX_INSTANCES	50
DETECTION_MIN_CONFIDENCE	0.9
DETECTION_NMS_THRESHOLD	0.2
FPN_CLASSIF_FC_LAYERS_SIZE	1024
GPU_COUNT	1
GRADIENT_CLIP_NORM	5.0
IMAGES_PER_GPU	1
IMAGE_CHANNEL_COUNT	3
IMAGE_MAX_DIM	512
IMAGE_META_SIZE	16
IMAGE_MIN_DIM	400
IMAGE_MIN_SCALE	0
IMAGE_RESIZE_MODE	square
IMAGE_SHAPE	[512 512 3]
LEARNING_MOMENTUM	0.9
LEARNING_RATE	0.001
LOSS_WEIGHTS	{'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0, 'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0, 'mrcnn_mask_loss': 1.0}
<i>Dilanjutkan pada halaman berikutnya</i>	

Tabel 4.3 – Lanjutan dari halaman sebelumnya

MASK_POOL_SIZE	14
MASK_SHAPE	[28, 28]
MAX_GT_INSTANCES	50
MEAN_PIXEL	[123.7 116.8 103.9]
MINI_MASK_SHAPE	(56, 56)
NAME	object
NUM_CLASSES	4
POOL_SIZE	7
POST_NMS_ROIS_INFERENCE	1000
POST_NMS_ROIS_TRAINING	2000
PRE_NMS_LIMIT	6000
ROI_POSITIVE_RATIO	0.33
RPN_ANCHOR RATIOS	[0.5, 1, 2]
RPN_ANCHOR_SCALES	(32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE	1
RPN_BBOX_STD_DEV	[0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD	0.7
RPN_TRAIN_ANCHORS_PER_IMAGE	256
STEPS_PER_EPOCH	100
TOP_DOWN_PYRAMID_SIZE	256
TRAIN_BN	False
TRAIN_ROIS_PER_IMAGE	200
USE_MINI_MASK	True
USE_RPN_ROIS	True
VALIDATION_STEPS	30
WEIGHT_DECAY	0.0001

Setelah dilakukan serangkaian proses training yang memakan waktu sekitar 3 jam 40 menit 24 detik didapatkan *output* berupa *model file* dengan format *h5* yang mempunyai ukuran 170.9 MB. *Training loss* terendah yang berhasil dicapai dengan menggunakan *backbone* Resnet-50 (pada *epoch* ke 242) adalah 0.4061 dengan rincian *training bounding box loss* sebesar 0.04083, *training classification loss* sebesar 0.02268 serta *training mask loss* sebesar 0.139 (dimana $L = L_{bbox} + L_{cls} + L_{mask}$). Gambar 4.1 merupakan grafik

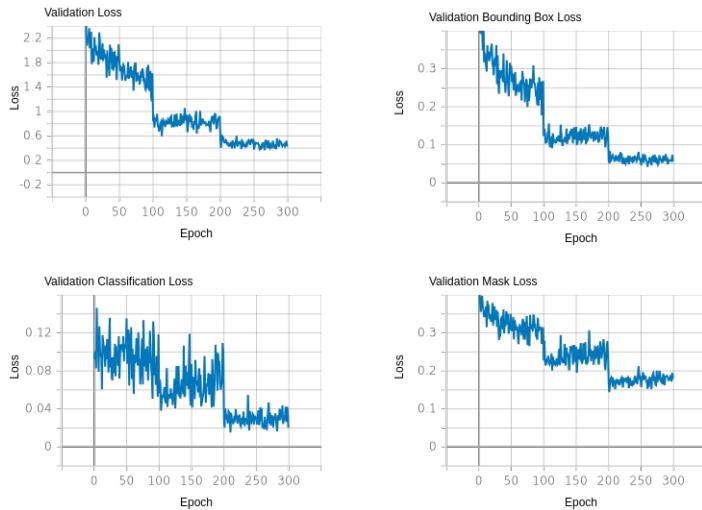
yang menunjukkan perubahan *training loss*, *training bounding box loss*, *training classification loss*, serta *training mask loss* dari epoch 1 sampai 300.



Gambar 4.1: Grafik Perubahan *Training Loss* pada Resnet-50

Sedangkan pada saat proses *validation* sendiri *Loss* terendah yang berhasil dicapai pada epoch ke 266 dengan nilai sebesar 0.3653 dengan rincian *validation bounding box loss* sebesar 0.4556, *validation classification loss* sebesar 0.01912 serta *validation mask loss* sebesar 0.1519. Namun untuk *validation bounding box loss* terendah berada pada epoch ke 260 dengan nilai sebesar 0.4203 sedangkan *validation classification loss* terendah pada epoch ke 210 dengan nilai 0.01525 serta *validation mask loss* terendah pada epoch ke 201 dengan nilai 0.1439. Gambar 4.2 merupakan grafik yang menunjukkan perubahan *validation loss*, *validation bounding box loss*, *validation classification loss*, serta *validation mask loss* dari epoch 1 sampai 300.

Selain menggunakan *Loss Function* untuk mengukur peforma hasil *training* yang sudah dilakukan, digunakan juga *mean Average Precision (mAP)*. *Precision* sendiri merupakan fungsi untuk meng-

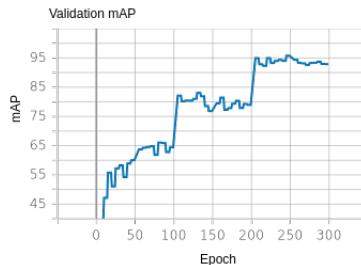


Gambar 4.2: Grafik Perubahan *Validation Loss* pada Resnet-50

gambarkan tingkat keakuratan antara data yang diminta dengan hasil prediksi yang diberikan oleh model. Maka, *precision* merupakan rasio prediksi benar positif (TP) dibandingkan dengan keseluruhan hasil yang diprediksi positif (TP dan FP). Rumus untuk mencari *Precision* adalah sebagai berikut :

$$Precision = \frac{TP}{TP + FP} \quad (4.1)$$

Perhitungan *mAP* pada penelitian ini dilakukan setiap 5 *epoch* sekali, karena jika dilakukan setiap *epoch* akan memerlukan *training time* yang lebih lama serta *resource hardware* yang diperlukan lebih besar. Nilai *mAP* tertinggi didapatkan pada *epoch* ke 220 sebesar 94.92. Gambar 4.3 merupakan grafik yang menunjukkan perubahan *validation mean Average Precision* dari *epoch* 1 sampai 300.



Gambar 4.3: Grafik Perubahan *Validation mAP* pada Resnet-50

4.1.2 Resnet-101

Tabel 4.4 merupakan parameter-parameter yang digunakan untuk membuat model Mask R-CNN dengan menggunakan *backbone* Resnet-101.

Tabel 4.4: Konfigurasi Model menggunakan Resnet-101

Pengaturan Model Resnet-101	
BACKBONE	resnet101
BACKBONE_STRIDES	[4, 8, 16, 32, 64]
BATCH_SIZE	1
BBOX_STD_DEV	[0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE	None
DETECTION_MAX_INSTANCES	50
DETECTION_MIN_CONFIDENCE	0.9
DETECTION_NMS_THRESHOLD	0.2
FPN_CLASSIF_FC_LAYERS_SIZE	1024
GPU_COUNT	1
GRADIENT_CLIP_NORM	5.0
IMAGES_PER_GPU	1
IMAGE_CHANNEL_COUNT	3
IMAGE_MAX_DIM	512
IMAGE_META_SIZE	16
IMAGE_MIN_DIM	400
<i>Dilanjutkan pada halaman berikutnya</i>	

Tabel 4.4 – Lanjutan dari halaman sebelumnya

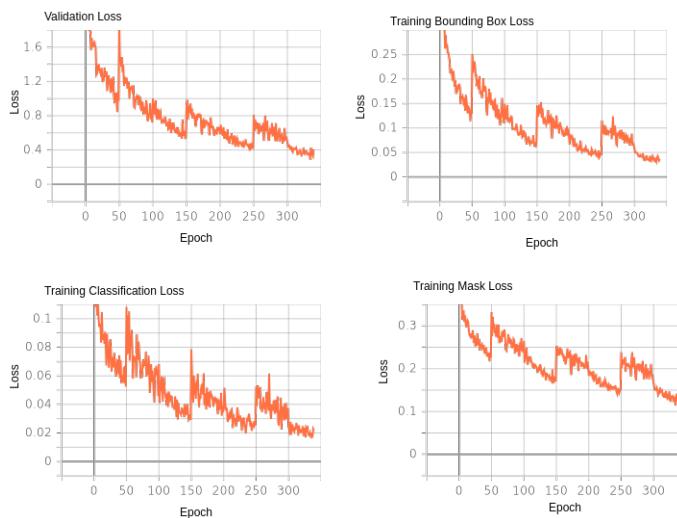
IMAGE_MIN_SCALE	0
IMAGE_RESIZE_MODE	square
IMAGE_SHAPE	[512 512 3]
LEARNING_MOMENTUM	0.9
LEARNING_RATE	0.001
LOSS_WEIGHTS	{'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0, 'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0, 'mrcnn_mask_loss': 1.0}
MASK_POOL_SIZE	14
MASK_SHAPE	[28, 28]
MAX_GT_INSTANCES	50
MEAN_PIXEL	[123.7 116.8 103.9]
MINI_MASK_SHAPE	(56, 56)
NAME	object
NUM_CLASSES	4
POOL_SIZE	7
POST_NMS_ROIS_INFERENCE	1000
POST_NMS_ROIS_TRAINING	2000
PRE_NMS_LIMIT	6000
ROI_POSITIVE_RATIO	0.33
RPN_ANCHOR RATIOS	[0.5, 1, 2]
RPN_ANCHOR_SCALES	(32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE	1
RPN_BBOX_STD_DEV	[0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD	0.7
RPN_TRAIN_ANCHORS_PER_IMAGE	256
STEPS_PER_EPOCH	100
TOP_DOWN_PYRAMID_SIZE	256
TRAIN_BN	False
TRAIN_ROIS_PER_IMAGE	200
USE_MINILMASK	True
USE_RPN_ROIS	True

Dilanjutkan pada halaman berikutnya

Tabel 4.4 – Lanjutan dari halaman sebelumnya

VALIDATION_STEPS	30
WEIGHT_DECAY	0.0001

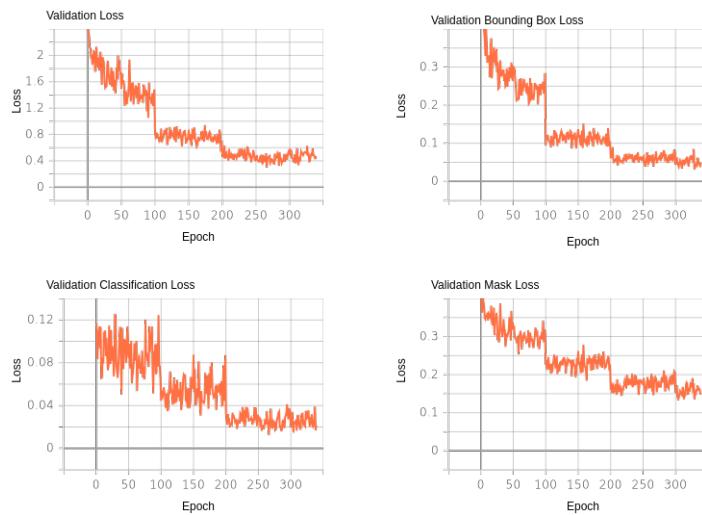
Setelah dilakukan serangkaian proses training yang memakan waktu sekitar 4 jam 9 menit 16 detik didapatkan *output* berupa *model file* dengan format *h5* yang mempunyai ukuran 244 MB. *Training loss* terendah yang berhasil dicapai dengan menggunakan *backbone* Resnet-101 (pada *epoch* ke 244) adalah 0.3933 dengan rincian *training bounding box loss* sebesar 0.04164, *training classification loss* sebesar 0.0247 serta *training mask loss* sebesar 0.1403. Namun untuk *training bounding box loss* terendah terdapat pada *epoch* ke 246 dengan nilai sebesar 0.04083, *training classification loss* terendah pada *epoch* ke 234 dengan nilai 0.01957. Gambar 4.4 merupakan grafik yang menunjukkan perubahan *training loss*, *training bounding box loss*, *training classification loss*, serta *training mask loss* dari *epoch* 1 sampai 300.



Gambar 4.4: Grafik Perubahan *Training Loss* pada Resnet-101

Sedangkan pada saat proses *validation* sendiri *Loss* terendah

yang berhasil dicapai pada *epoch* ke 266 dengan nilai sebesar 0.299 dengan rincian *validation bounding box loss* sebesar 0.03867, *validation classification loss* sebesar 0.01246 serta *validation mask loss* sebesar 0.1461. Gambar 4.5 merupakan grafik yang menunjukkan perubahan *validation loss*, *validation bounding box loss*, *validation classification loss*, serta *validation mask loss* dari *epoch* 1 sampai 300.



Gambar 4.5: Grafik Perubahan *Validation Loss* pada Resnet-101

Nilai *mAP* tertinggi didapatkan pada *epoch* ke 215 sebesar 96.21. Gambar 4.6 merupakan grafik yang menunjukkan perubahan *validation mean Average Precision* dari *epoch* 1 sampai 300.



Gambar 4.6: Grafik Perubahan *Validation mAP* pada Resnet-101

4.1.3 MobileNet-V1

Tabel 4.5 merupakan parameter-parameter yang digunakan untuk membuat model Mask R-CNN dengan menggunakan *backbone* Mobilenet-V1.

Tabel 4.5: Konfigurasi Model menggunakan Mobilenet-V1

Pengaturan Model Mobilenet-V1	
BACKBONE	mobilenetv1
BACKBONE_STRIDES	[4, 8, 16, 32, 64]
BATCH_SIZE	1
BBOX_STD_DEV	[0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE	None
DETECTION_MAX_INSTANCES	50
DETECTION_MIN_CONFIDENCE	0.9
DETECTION_NMS_THRESHOLD	0.2
FPN_CLASSIF_FC_LAYERS_SIZE	1024
GPU_COUNT	1
GRADIENT_CLIP_NORM	5.0
IMAGES_PER_GPU	1
IMAGE_CHANNEL_COUNT	3
IMAGE_MAX_DIM	512
IMAGE_META_SIZE	16
IMAGE_MIN_DIM	400
<i>Dilanjutkan pada halaman berikutnya</i>	

Tabel 4.5 – Lanjutan dari halaman sebelumnya

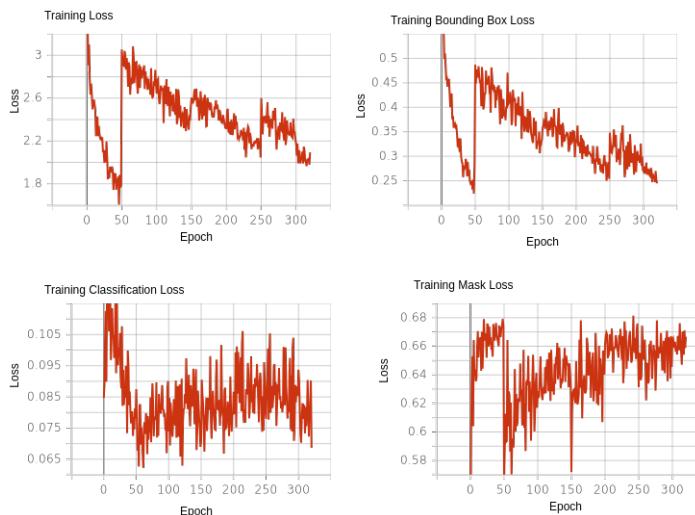
IMAGE_MIN_SCALE	0
IMAGE_RESIZE_MODE	square
IMAGE_SHAPE	[512 512 3]
LEARNING_MOMENTUM	0.9
LEARNING_RATE	0.001
LOSS_WEIGHTS	{'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0, 'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0, 'mrcnn_mask_loss': 1.0}
MASK_POOL_SIZE	14
MASK_SHAPE	[28, 28]
MAX_GT_INSTANCES	50
MEAN_PIXEL	[123.7 116.8 103.9]
MINI_MASK_SHAPE	(56, 56)
NAME	object
NUM_CLASSES	4
POOL_SIZE	7
POST_NMS_ROIS_INFERENCE	1000
POST_NMS_ROIS_TRAINING	2000
PRE_NMS_LIMIT	6000
ROI_POSITIVE_RATIO	0.33
RPN_ANCHOR RATIOS	[0.5, 1, 2]
RPN_ANCHOR_SCALES	(32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE	1
RPN_BBOX_STD_DEV	[0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD	0.7
RPN_TRAIN_ANCHORS_PER_IMAGE	256
STEPS_PER_EPOCH	100
TOP_DOWN_PYRAMID_SIZE	256
TRAIN_BN	False
TRAIN_ROIS_PER_IMAGE	200
USE_MINILMASK	True
USE_RPN_ROIS	True

Dilanjutkan pada halaman berikutnya

Tabel 4.5 – Lanjutan dari halaman sebelumnya

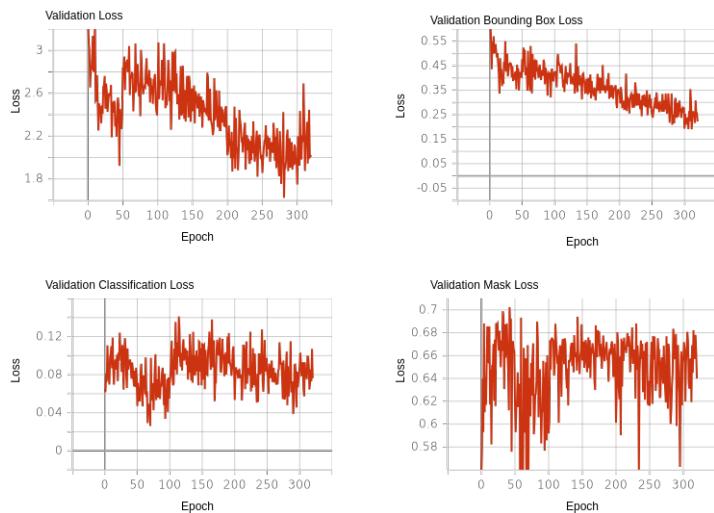
VALIDATION_STEPS	30
WEIGHT_DECAY	0.0001

Setelah dilakukan serangkaian proses training yang memakan waktu sekitar 3 jam 18 menit 44 detik didapatkan *output* berupa *model file* dengan format *h5* yang mempunyai ukuran 83.3 MB. *Training loss* terendah yang berhasil dicapai dengan menggunakan *backbone* Mobilenet-V1 (pada *epoch* ke 46) adalah 1.604 dengan rincian *training bounding box loss* sebesar 0.2322, *training classification loss* sebesar 0.07858 serta *training mask loss* sebesar 0.6729. Namun untuk *training bounding box loss* terendah terdapat pada *epoch* ke 48 dengan nilai sebesar 0.2239, *training classification loss* terendah pada *epoch* ke 61 dengan nilai 0.06219 serta *training mask loss* terendah pada *epoch* ke 61 sebesar 0.5701. Gambar ?? merupakan grafik yang menunjukkan perubahan *training loss*, *training bounding box loss*, *training classification loss*, serta *training mask loss* dari *epoch* 1 sampai 300.



Gambar 4.7: Grafik Perubahan *Training Loss* pada Mobilenet-V1

Sedangkan pada saat proses *validation* sendiri *Loss* terendah yang berhasil dicapai pada *epoch* ke 281 dengan nilai sebesar 1.624 dengan rincian *validation bounding box loss* sebesar 0.2088, *validation classification loss* sebesar 0.04799 serta *validation mask loss* sebesar 0.6001. Namun untuk *validation bounding box loss* terendah berada pada *epoch* ke 300 dengan nilai sebesar 0.1927 sedangkan *validation classification loss* terendah pada *epoch* ke 70 dengan nilai 0.02607 serta *validation mask loss* terendah pada *epoch* ke 295 dengan nilai 0.5625. Gambar 4.8 merupakan grafik yang menunjukkan perubahan *validation loss*, *validation bounding box loss*, *validation classification loss*, serta *validation mask loss* dari *epoch* 1 sampai 300.



Gambar 4.8: Grafik Perubahan *Validation Loss* pada MobileNet-V1

Nilai *mAP* tertinggi didapatkan pada *epoch* ke 210 sebesar 26.04. Gambar 4.9 merupakan grafik yang menunjukkan perubahan *validation mean Average Precision* dari *epoch* 1 sampai 300.

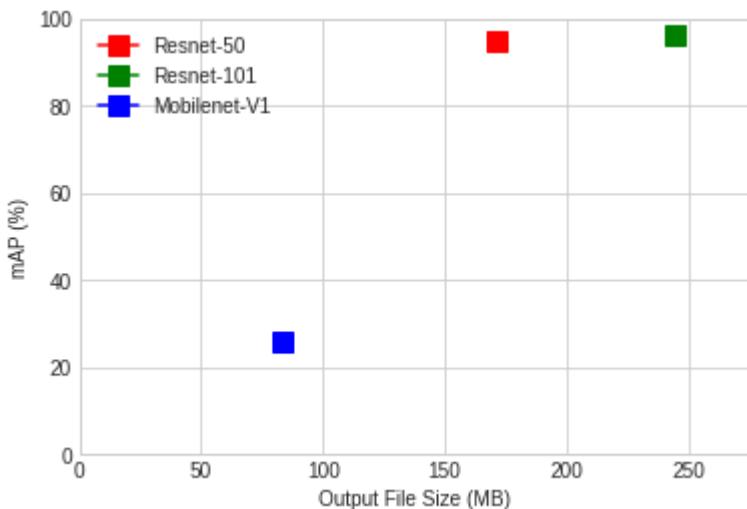


Gambar 4.9: Grafik Perubahan *Validation mAP* pada Mobilenet-V1

Tabel 4.6 adalah tabel perbandingan dari *output file size* dan *mAP* dari total 3 model dengan *backbone* berbeda yang diuji. Tahap ini masih merupakan hasil tahap *training*. Sedangkan Gambar 4.10 merupakan grafik perbandingan *mAP* dengan *file size*. Proses selanjutnya setelah mendapatkan hasil yang ditunjukkan pada setiap model adalah melakukan proses prediksi atau melakukan *testing data*.

Tabel 4.6: Tabel Perbandingan Model dengan *backbone* yang berbeda

No.	<i>Backbone</i>	<i>Output File Size</i>	<i>mAP</i>
1	Resnet-50	170.9 MB	94.92%
2	Resnet-101	244 MB	96.21%
3	Mobilenet-V1	83.3 MB	26.04%



Gambar 4.10: Grafik Perbandingan Model dengan *backbone* yang berbeda

4.1.4 Perbandingan Hasil Prediksi

Setelah mendapatkan hasil *training* pada Tabel 4.6 terdapat dua parameter yang digunakan yaitu *output file size* dan *mean Average Precision*. Parameter tersebut digunakan untuk melakukan justifikasi dalam pemilihan model yang terbaik. Pada parameter *output file size*, model yang diinginkan adalah model dengan dengan ukuran sekecil mungkin, agar tidak menghabiskan kapasitas penyimpanan terlalu banyak. Sedangkan untuk *mean Average Precision* yang diinginkan adalah model dengan *meand Average Precision* yang tinggi sehingga kemampuan model tersebut untuk melakukan proses prediksi akan semakin tepat dengan kondisi yang sesuai pada dunia nyata.

Setelah membandingkan parameter *output file size* dan *mean Average Precision*, pada tahap selanjutnya akan dilakukan proses membandingkan nilai atau hasil dari prediksi pada setiap model yang telah dibuat. Semakin tinggi *mean Average Precision* dan ke-

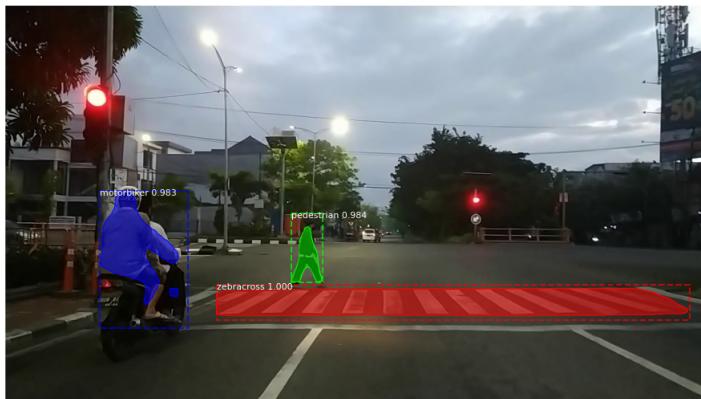
benaran dalam prediksi, maka model tersebut sangat bagus dan sesuai dengan permasalahan yang ingin diselesaikan oleh model yang telah dibuat. Tujuan dari tahapan membandingkan ini adalah untuk memilih model yang terbaik dan apakah model yang telah dibuat dapat benar-benar dapat menyelesaikan permasalahan *object detection and segmentation*. Sehingga, dari hasil percobaan ini dapat menentukan model terbaik yang dapat menyelesaikan permasalahan dan juga telah diuji dengan menggunakan *testing set*.

Hasil Pengujian dengan *Backbone Resnet-50*

Pada hasil pengujian dengan menggunakan satu gambar jalan raya pada saat pagi hari yang ditunjukkan pada gambar 4.11, didapatkan hasil skor yaitu :

1. Pejalan kaki (*score* : 0,984)
2. Zebracross (*score* : 0,983)
3. Pengendara motor (*score* : 1,00)

Dari hasil tersebut, *Backbone Resnet-50* dapat memprediksi gambar dengan benar, serta *evaluation time* dari *backbone* ini sebesar 0,76 detik



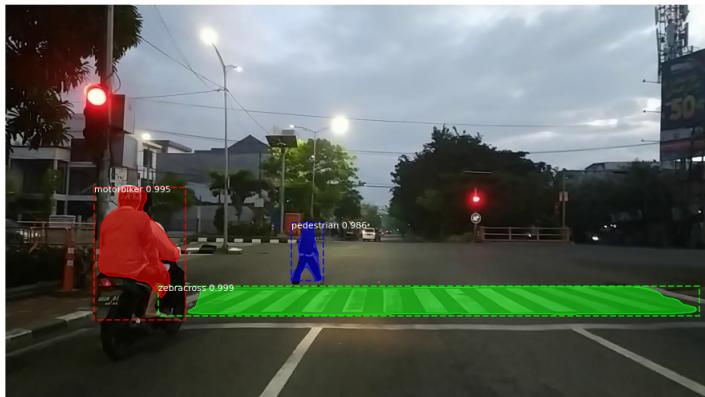
Gambar 4.11: Hasil Uji dari *Backbone Resnet-50*

Hasil Pengujian dengan *Backbone Resnet-101*

Pada hasil pengujian dengan menggunakan satu gambar jalan raya pada saat pagi hari yang ditunjukkan pada gambar 4.12, didapatkan hasil skor yaitu :

1. Pejalan kaki (*score* : 0,986)
2. Zebracross (*score* : 0,995)
3. Pengendara motor (*score* : 0,999)

Dari hasil tersebut, *Backbone Resnet-101* dapat memprediksi gambar dengan benar, serta *evaluation time* dari *backbone* ini sebesar 0.961 detik



Gambar 4.12: Hasil Uji dari *Backbone Resnet-101*

Hasil Pengujian dengan *Backbone Mobilenet-V1*

Pada hasil pengujian dengan menggunakan satu gambar jalan raya pada saat pagi hari yang ditunjukkan pada gambar 4.13, didapatkan hasil skor yaitu :

1. Pejalan kaki (tidak terdeteksi)
2. Zebracross (terdeteksi 2 dengan *score* : 0,942 dan 0.888)
3. Pengendara motor (tidak terdeteksi)

Dari hasil tersebut, *Backbone Mobilenet-V1* tidak dapat mempre-

diksi gambar dengan benar, serta *evaluation time* dari *backbone* ini sebesar 0.664 detik



Gambar 4.13: Hasil Uji dari *Backbone Mobilenet-V1*

Setelah melakukan *testing* dengan menggunakan gambar pada *testing set* pada keseluruhan model, maka tabel 4.7 menunjukkan ringkasan dari perbandingan hasil *testing* yang telah dilakukan. Parameter yang akan dibandingkan yaitu *evaluation time* dan besar *mean Average Precision*.

Tabel 4.7: tabel Perbandingan Hasil *Testing*

No.	<i>Backbone</i>	<i>Evaluation Time</i>	<i>mAP</i>
1	ResNet-50	0,763 s	100%
2	ResNet-101	0.961 s	100%
3	MobileNet-v1	0.664 s	25%

4.2 Pengujian Perbedaan Waktu

Pengujian pada perbedaan waktu bertujuan untuk mengetahui performa dan akurasi dari setiap model yang dihasilkan dengan

waktu yang berbeda-beda (pagi, siang dan malam). *File input* untuk pengujian ini terdapat 2 macam, yaitu gambar dengan format .jpg dan video dengan format .mp4.

4.2.1 Pengujian di Pagi Hari

Pada pengujian ini *testing data* diambil dengan menggunakan kamera *smartphone* yang menghasilkan video beresolusi 1280×720 px berdurasi 63.43 s. Video tersebut mempunyai banyak *frame* per detik sebanyak 30 fps serta total *frame* yang dihasilkan sejumlah 1903 *frame*. Gambar 4.14 merupakan perbandingan hasil deteksi dan segementasi dari ketiga *backbone* yang digunakan pada waktu pagi hari.



(a) ResNet-50 (b) ResNet-101 (c) MobileNet-v1

Gambar 4.14: Perbandingan Hasil pada Pagi Hari

Untuk meguji performa model yang dihasilkan pada masing-masing *backbone*, digunakan beberapa parameter nilai seperti *precision*, *recall* dan *F1-score* pada masing-masing kelas objek yang didekripsi. Perbandingan hasil evaluasi yang didapatkan masing-masing *backbone* dapat dilihat pada Tabel 4.8.

Pada Resnet-50 *Average Precision* untuk semua kelas yang berhasil dideteksi adalah sebesar 100% dan *Average Recall* 100%. Sedangkan pada Resnet-101 memberikan hasil yang sama dengan Resnet-50 dengan *AP* 100% dan *AR* 100%. Namun pada Mobilenet-v1 objek yang berhasil dideteksi hanya *zebracross* (ditambah dengan *background*), sehingga pada objek pejalan kaki dan pengendara motor mempunyai *precision* bernilai 0% serta *Recall* dan *F1-score* bernilai tidak terdefinisi atau *Nan*. Hal ini dikarenakan nilai *True Positif* pada pejalan kaki dan pengendara motor bernilai 0 (angka 0 dibagi nilai berapun memiliki hasil tidak terdefinisi). Nilai *AP* yang dihasilkan pada evaluasi Mobilenet-v1 sebesar 37.5%.

Tabel 4.8: Perbandingan Hasil Evaluasi pada Pagi Hari

(a) ResNet-50

Kelas	Precision (%)	Recall (%)	F1-Score (%)
Background	100	100	100
Pedestrian	100	100	100
Zebracross	100	100	100
Motorbiker	100	100	100

(b) ResNet-101

Kelas	Precision (%)	Recall (%)	F1-Score (%)
Background	100	100	100
Pedestrian	100	100	100
Zebracross	100	100	100
Motorbiker	100	100	100

(c) MobileNet-v1

Kelas	Precision (%)	Recall (%)	F1-Score (%)
Background	50	33.33	40
Pedestrian	0	NaN	NaN
Zebracross	100	50	66.66
Motorbiker	0	NaN	NaN

Selain menggunakan *input data* berupa gambar, pada pengujian ini juga menggunakan *input data* berupa *file* video dengan format *.mp4*. Proses *testing* dibagi menjadi dua tahapan, tahap pertama memecah *frame* video menjadi gambar dan melakukan prediksi serta tahap kedua menggabungkan kembali setiap *frame* menjadi video. Tabel ?? menunjukkan waktu yang dibutuhkan untuk melakukan semua tahap prediksi pada *file input* berbentuk video pada ketiga *backbone*.

4.2.2 Pengujian di Siang Hari

Pada pengujian ini *testing data* diambil dari video streaming *youtube*[21] yang memiliki resolusi 1920×1080 px berdurasi 29.4 s. Video tersebut mempunyai banyak *frame* per detik sebanyak 25 fps

Tabel 4.9: Waktu Prediksi pada *File Input* Video dalam *MM:SS*

<i>Backbone</i>	<i>Tahap 1</i>	<i>Tahap 2</i>	<i>Total</i>	<i>Output Size</i>
<i>ResNet-50</i>	07:24	00:38	08:03	88.8 MB
<i>ResNet-101</i>	07:54	00:49	08:44	87.7 MB
<i>MobileNet-v1</i>	06:50	00:40	07:31	89.7 MB

serta total *frame* yang dihasilkan sejumlah 735 *frame*. Gambar 4.15 merupakan perbandingan hasil deteksi dan segementasi dari ketiga *backbone* yang digunakan pada waktu siang hari.



(a) ResNet-50 (b) ResNet-101 (c) MobileNet-v1

Gambar 4.15: Perbandingan Hasil pada Siang Hari

Perbandingan hasil evaluasi yang didapatkan masing-masing *backbone* dengan membandingkan nilai *precision*, *recall* dan *F1-score* dapat dilihat pada Tabel 4.10.

Pada Resnet-50 *Average Precision* untuk semua kelas yang berhasil dideteksi adalah sebesar 62.5% dan *Average Recall* 72.915%. Sedangkan pada Resnet-101 memberikan hasil *AP* sebesar 75% dan *AR* 81.5%. Namun pada Mobilenet-v1 objek yang berhasil dide- teksi hanya *zebracross* (ditambah dengan *background*) sama saat pengujian pada pagi hari. Nilai *AP* yang dihasilkan pada evaluasi Mobilenet-v1 sebesar 25%.

Selain menggunakan *input data* berupa gambar, pada pengujian ini juga menggunakan *input data* berupa *file video* dengan format *.mp4*. Proses *testing* dibagi menjadi dua tahapan, tahap pertama memecah *frame* video menjadi gambar dan melakukan prediksi serta tahap kedua menggabungkan kembali setiap *frame* menjadi video. Tabel ?? menunjukkan waktu total yang dibutuhkan untuk melakukan semua tahap prediksi pada *file input* berbentuk video pada

Tabel 4.10: Perbandingan Hasil Evaluasi pada Siang Hari

(a) ResNet-50

Kelas	Precision (%)	Recall (%)	F1-Score (%)
Background	50	25	33.33
Pedestrian	100	100	100
Zebracross	50	100	66.66
Motorbiker	50	66.66	57.14

(b) ResNet-101

Kelas	Precision (%)	Recall (%)	F1-Score (%)
Background	100	25	40
Pedestrian	100	100	100
Zebracross	50	100	66.66
Motorbiker	50	100	66.66

(c) MobileNet-v1

Kelas	Precision (%)	Recall (%)	F1-Score (%)
Background	50	14.28	22.22
Pedestrian	0	NaN	NaN
Zebracross	50	50	50
Motorbiker	0	NaN	NaN

ketiga *backbone*.

Tabel 4.11: Waktu Prediksi pada *File Input* Video dalam *MM:SS*

Backbone	Tahap 1	Tahap 2	Total	Output Size
ResNet-50	05:33	00:30	06:03	52.8 MB
ResNet-101	07:21	00:36	07:58	50.1 MB
MobileNet-v1	04:03	00:30	04:33	42 MB

4.2.3 Pengujian di Malam Hari

Pada pengujian ini *testing data* diambil dari video *streaming youtube* [22] yang memiliki resolusi 1920×1080 px berdurasi 28.04 s. Video tersebut mempunyai banyak *frame* per detik sebanyak 25 fps serta total *frame* yang dihasilkan sejumlah 701 *frame*. Gambar 4.16 merupakan perbandingan hasil deteksi dan segementasi dari ketiga *backbone* yang digunakan pada waktu siang hari.



Gambar 4.16: Perbandingan Hasil pada Malam Hari

Perbandingan hasil evaluasi yang didapatkan masing-masing *backbone* dengan membandingkan nilai *precision*, *recall* dan *F1-score* dapat dilihat pada Tabel 4.12.

Pada gambar yang diuji untuk malam hari hanya memiliki 3 kelas saja yaitu *background*, pejalan kaki serta *zebracross* tanpa ada-nya pengendara motor. Resnet-50 mempunyai nilai *Average Precision* untuk semua kelas yang berhasil dideteksi adalah sebesar 80% dan *Average Recall* 75%. Sedangkan pada Resnet-101 memberikan hasil *AP* sebesar 86.67% dan *AR* 77.77%. Namun pada Mobilenet-v1 objek yang berhasil dideteksi hanya *zebracross* (ditambah dengan *background*) sama saat pengujian pada pagi dan siang hari. Nilai *AP* yang dihasilkan pada evaluasi Mobilenet-v1 sebesar 50%.

Selain menggunakan *input data* berupa gambar, pada pengujian ini juga menggunakan *input data* berupa *file video* dengan format *.mp4*. Proses *testing* dibagi menjadi dua tahapan, tahap pertama memecah *frame* video menjadi gambar dan melakukan prediksi serta tahap kedua menggabungkan kembali setiap *frame* menjadi video. Tabel ?? menunjukkan waktu yang dibutuhkan untuk melakukan semua tahap prediksi pada *file input* berbentuk video pada ketiga *backbone*.

Tabel 4.12: Perbandingan Hasil Evaluasi pada Malam Hari

(a) ResNet-50

Kelas	Precision (%)	Recall (%)	F1-Score (%)
Background	100	25	40
Pedestrian	40	100	57.14
Zebracross	100	100	100
Motorbiker	-	-	-

(b) ResNet-101

Kelas	Precision (%)	Recall (%)	F1-Score (%)
Background	100	33.33	50
Pedestrian	60	100	74.9
Zebracross	100	100	100
Motorbiker	-	-	-

(c) MobileNet-v1

Kelas	Precision (%)	Recall (%)	F1-Score (%)
Background	50	16.66	25
Pedestrian	0	NaN	NaN
Zebracross	100	50	66.66
Motorbiker	-	-	-

Tabel 4.13: Waktu Prediksi pada *File Input* Video dalam *MM:SS*

Backbone	Tahap 1	Tahap 2	Total	Output Size
ResNet-50	04:23	00:28	04:51	38.7 MB
ResNet-101	04:49	00:34	05:23	39.1 MB
MobileNet-v1	03:39	00:28	04:07	36.1 MB

4.2.4 Perbandingan Hasil Evaluasi pada Perbedaan Waktu

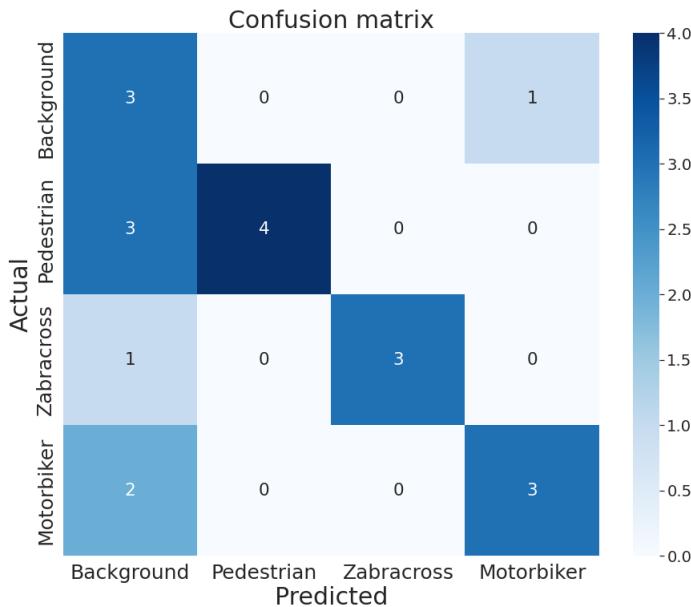
Setelah membandingkan hasil evaluasi model setiap perbedaan waktu, pada bagian ini akan dibandingkan peforma setiap model untuk keseluruhan perbedaan waktu (pagi, siang dan malam hari).

ResNet-50

Pada proses evaluasi model dengan menggunakan seluruh *testing data* (pagi, siang dan malam hari) didapatkan beberapa hasil skor yaitu :

1. *mean Average Precision(mAP)* : 66.785%
2. *mean Average Recall(mAR)* : 77%
3. *F1-score* : 71.529%

Hasil tersebut didapatkan setelah dilakukan proses pencarian nilai *Trur Positif*, *False Positif* dan *False Negatif* seperti yang ditunjukkan pada *confusion matrix* pada Gambar 4.17.



Gambar 4.17: *Confusion Matrix* ResNet-50

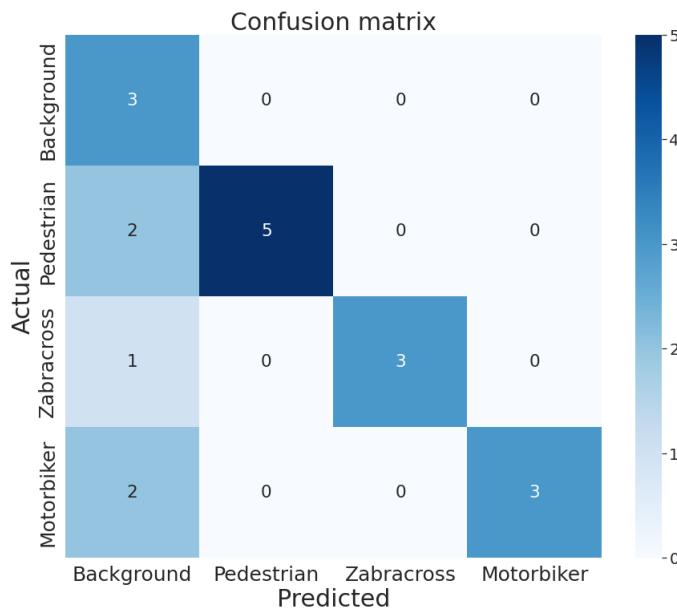
ResNet-101

Pada proses evaluasi model dengan menggunakan seluruh *testing data* (pagi, siang dan malam hari) didapatkan beberapa hasil

skor yaitu :

1. *mean Average Precision(mAP)* : 76.605%
2. *mean Average Recall(mAR)* : 85.375%
3. *F1-score* : 80.302%

Hasil tersebut didapatkan setelah dilakukan proses pencarian nilai *Trur Positif*, *False Positif* dan *False Negatif* seperti yang ditunjukkan pada *confusion matrix* pada Gambar 4.18.



Gambar 4.18: *Confusion Matrix* ResNet-101

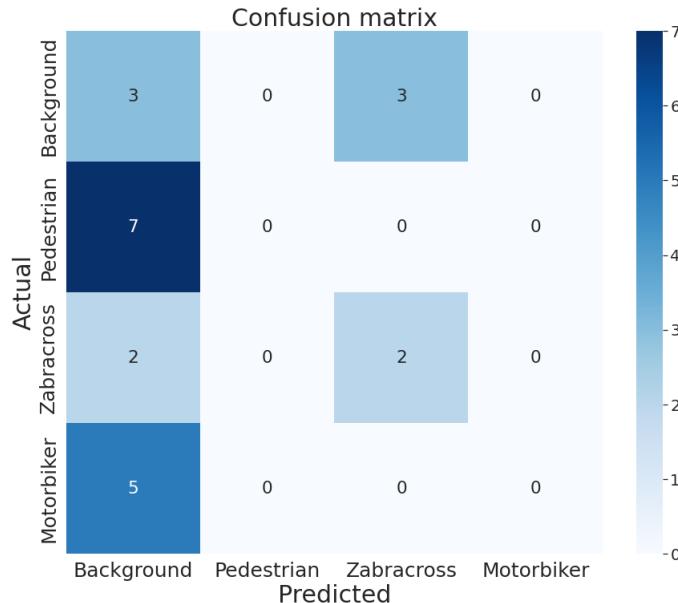
MobileNet-v1

Pada proses evaluasi model dengan menggunakan seluruh *testing data* (pagi, siang dan malam hari) didapatkan beberapa hasil skor yaitu :

1. *mean Average Precision(mAP)* : 25%
2. *mean Average Recall(mAR)* : *NaN*

3. *F1-score : NaN*

Hasil tersebut didapatkan setelah dilakukan proses pencarian nilai *Trur Positif*, *False Positif* dan *False Negatif* seperti yang ditunjukkan pada *confusion matrix* pada Gambar 4.19.



Gambar 4.19: *Confusion Matrix* MobileNet-v1

[Halaman ini sengaja dikosongkan]

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan hasil pengujian yang telah dilakukan, penulis dapat menyimpulkan beberapa hal sebagai berikut:

1. Dalam penelitian ini telah diimplementasikan dengan baik proses deteksi dan segmentasi pejalan kaki dan zebrecross dengan menggunakan Mask R-CNN, dengan *mean Average Precision* sebesar 76.62%.
2. *Backbone* ResNet-101 memiliki hasil akurasi yang lebih baik dibanding dengan *backbone* lainnya dengan performa lebih tinggi sebesar 12.82% dibanding ResNet-50 dan 67.36% dibanding MobileNet-v1.
3. Waktu yang dibutuhkan dalam proses pendektesian akan semakin lama jika objek yang berada pada gambar semakin banyak.

5.2 Saran

Untuk pengembangan lebih lanjut pada penelitian mendatang, maka penulis memiliki saran sebagai berikut:

1. Menambah jumlah sample data pada kelas pengendara motor yang dinilai masih sedikit untuk dataset yang dipilih.
2. Pembuatan dataset pejalan kaki di Indonesia karena perilaku pejalan kaki antar negara memiliki perbedaan yang cukup signifikan.

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

- [1] Supervised vs. unsupervised learning, 2018. URL <https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d>. (Dikutip pada halaman xi, 6).
- [2] Clustering in machine learning, 2020. URL <https://www.geeksforgeeks.org/clustering-in-machine-learning>. (Dikutip pada halaman xi, 7).
- [3] Hongbo Gao, Guanya Shi, Guotao Xie, and Bo Cheng. Car-following method based on inverse reinforcement learning for autonomous vehicle decision-making. *International Journal of Advanced Robotic Systems*, 15:172988141881716, 11 2018. doi: 10.1177/1729881418817162. (Dikutip pada halaman xi, 7).
- [4] N. Ferracuti, Claudia Norscini, Emanuele Frontoni, P. Gabellini, Marina Paolanti, and Valerio Placidi. A business application of rtls technology in intelligent retail environment: Defining the shopper's preferred path and its segmentation. *Journal of Retailing and Consumer Services*, 47:184–194, 03 2019. doi: 10.1016/j.jretconser.2018.11.005. (Dikutip pada halaman xi, 8).
- [5] Sigmoid function, 2021. URL https://en.wikipedia.org/wiki/Sigmoid_function. (Dikutip pada halaman xi, 15, 16).
- [6] R-cnn — region based cnns, 2020. URL <https://www.geeksforgeeks.org/r-cnn-region-based-cnns/>. (Dikutip pada halaman xi, 19).
- [7] Lan Hu. Robot indoor text contents recognition based on visual slam. *Journal of Physics: Conference Series*, 1302:032004, 08 2019. doi: 10.1088/1742-6596/1302/3/032004. (Dikutip pada halaman xi, 21).

- [8] Cuong Nguyen, Giang Son Tran, Thi Nghiem, Nhat Do-an, Damien Gratadour, Jean-Christophe Burie, and Chi Luong. Towards real-time smile detection based on faster region convolutional neural network. pages 1–6, 04 2018. doi: 10.1109/MAPR.2018.8337524. (Dikutip pada halaman xi, 22).
- [9] Lukasz Bienias, Juanjo n, Line Nielsen, and Tommy Alstrøm. Insights into the behaviour of multi-task deep neural networks for medical image segmentation. pages 1–6, 10 2019. doi: 10.1109/MLSP.2019.8918753. (Dikutip pada halaman xi, 23).
- [10] Artificial intelligence (ai), 2021. URL <https://www.investopedia.com/terms/a/artificial-intelligence-ai.asp>. (Dikutip pada halaman 5).
- [11] Shagan Sah. Machine learning: A review of learning types. 07 2020. doi: 10.20944/preprints202007.0230.v1. (Dikutip pada halaman 5).
- [12] Amitha Mathew, Amudha Arul, and S. Sivakumari. *Deep Learning Techniques: An Overview*, pages 599–608. 01 2021. ISBN 978-981-15-3382-2. doi: 10.1007/978-981-15-3383-9_54. (Dikutip pada halaman 7).
- [13] Anirudha Ghosh, A. Sufian, Farhana Sultana, Amlan Chakrabarti, and Debasish De. *Fundamental Concepts of Convolutional Neural Network*, pages 519–567. 01 2020. ISBN 978-3-030-32643-2. doi: 10.1007/978-3-030-32644-9_36. (Dikutip pada halaman 8).
- [14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(1):142–158, 2016. doi: 10.1109/TPAMI.2015.2437384. (Dikutip pada halaman 20).
- [15] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015. (Dikutip pada halaman 20).

- [16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39, 06 2015. doi: 10.1109/TPAMI.2016.2577031. (Dikutip pada halaman 21).
- [17] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. (Dikutip pada halaman 22).
- [18] Anelia Angelova, Alex Krizhevsky, Vincent Vanhoucke, Abhijit Ogale, and Dave Ferguson. Real-time pedestrian detection with deep network cascades. 2015. (Dikutip pada halaman 23).
- [19] Irtiza Hasan, Shengcai Liao, Jinpeng Li, Saad Ullah Akram, and Ling Shao. Pedestrian detection: The elephant in the room, 03 2020. (Dikutip pada halaman 24).
- [20] Chenchen Xu, Guili Wang, Songsong Yan, Jianghua Yu, Baojun Zhang, Shu Dai, Yu Li, and Lin Xu. Fast vehicle and pedestrian detection using improved mask r-cnn. *Mathematical Problems in Engineering*, 2020:1–15, 05 2020. doi: 10.1155/2020/5761414. (Dikutip pada halaman 25).
- [21] 4k60 driving around downtown jakarta from kota tua to blok m via thamrin & sudirman street view, 2020. URL https://www.youtube.com/watch?v=_37N1GrVujY&t=519s. (Dikutip pada halaman 58).
- [22] Amazing jakarta indonesia driving downtown - night drive 2021, 2021. URL <https://www.youtube.com/watch?v=JRJasFEC0eI&t=171s>. (Dikutip pada halaman 61).

[Halaman ini sengaja dikosongkan]

BIOGRAFI PENULIS



Agung Wicaksono, atau biasa dipanggil Agung, lahir di Madiun Jawa Timur pada tanggal 20 Mei 1999. Merupakan anak kedua dari tiga bersaudara. Penulis lulus dari SMP Negeri 1 Geger dan melanjutkan ke SMA Negeri 1 Geger. Penulis melanjutkan ke jenjang strata satu di Departemen Teknik Komputer Fakultas Teknologi Elektro dan Informatika Cerdas ITS. Dalam masa perkuliahan, penulis tertarik dengan pengembangan *Web Apps* dan *Machine Learning*. Penulis pernah aktif menjadi Staf Departemen Pengembangan Sumber Daya Mahasiswa Badan Eksekutif Mahasiswa Fakultas Teknologi Elektro serta Staf Ahli Kestari Mage 5. Bagi pembaca yang memiliki kritik, saran, atau pertanyaan mengenai tugas akhir ini dapat menghubungi penulis melalui email wicaksonoagun05@gmail.com.

[Halaman ini sengaja dikosongkan]