

Road Damage Detection Using Tensorflow Object Detection API and Tensorflow js

[Deteksi Kerusakan Jalan Menggunakan Tensorflow Object Detection API dan Tensorflow js]

Ahmad Irfan Masaid¹⁾, Suhendro Busono ^{*,2)}

¹⁾ Program Studi Teknik Informatika, Universitas Muhammadiyah Sidoarjo, Indonesia

²⁾ Program Studi Teknik Informatika, Universitas Muhammadiyah Sidoarjo, Indonesia

*Email Penulis Korespondensi: hendrob@umsida.ac.id

Abstract. Road damage significantly impacts mobility and the economy, with conventional detection methods being time-consuming and costly. This research develops an automated road damage detection system using the Single Shot Detector (SSD) with the MobileNetV2 architecture and TensorFlow Object Detection API, implemented in a web application. The system detects three types of road damage on asphalt and concrete surfaces using a 2019 dataset. Testing involved varying parameters such as learning rate, batch size, data split and total training steps. The best performance was achieved for model without data augmentation achieved optimal performance with the following parameter combination: a learning rate of 0.1 (total loss: 0.34), a data split ratio of 60:30:10 (total loss: 0.39), 3,000 training steps (total loss: 0.33), and a batch size of 24 (total loss: 0.41). The implementation on a web application facilitates real-time road damage detection with improved accessibility.

Keywords – deep learning; road damage; SSD Mobilenet; Tensorflow; TensorflowJS

Abstrak. Kerusakan jalan berdampak signifikan terhadap mobilitas dan perekonomian, dengan metode deteksi konvensional yang memakan waktu dan biaya tinggi. Penelitian ini mengembangkan sistem deteksi kerusakan jalan otomatis menggunakan Single Shot Detector (SSD) dengan arsitektur MobileNetV2 dan TensorFlow Object Detection API, yang diterapkan dalam aplikasi web. Sistem ini mendeteksi tiga jenis kerusakan jalan pada permukaan aspal dan beton menggunakan dataset tahun 2019. Pengujian melibatkan variasi parameter seperti laju pembelajaran, ukuran batch, dan total langkah pelatihan. Kinerja terbaik dicapai pada model tanpa augmentasi data, dengan kombinasi parameter optimal yaitu learning rate 0,1 (total loss 0,34), rasio split data 60:30:10 (total loss 0,39), total steps 3000 (total loss 0,33), dan batch size 24 (total loss 0,41). Implementasi pada aplikasi web memungkinkan deteksi kerusakan jalan secara real-time dengan peningkatan aksesibilitas.

Kata Kunci – deep learning; kerusakan jalan; SSD Mobilenet; Tensorflow; TensorflowJS

PENDAHULUAN

Kondisi infrastruktur jalan yang baik merupakan aspek penting dalam mendukung mobilitas dan pertumbuhan ekonomi suatu daerah. Namun, kerusakan jalan masih menjadi permasalahan serius di banyak wilayah, yang dapat menyebabkan ketidaknyamanan pengguna jalan, meningkatkan risiko kecelakaan, dan menimbulkan kerugian ekonomi [1]. Tujuan dilakukannya penelitian ini adalah untuk mengembangkan model machine learning dalam mendeteksi kerusakan jalan, guna mengurangi keterlambatan dalam identifikasi kerusakan infrastruktur jalan. Metode konvensional untuk mendeteksi dan memantau kerusakan jalan seringkali membutuhkan waktu yang lama, tenaga kerja yang intensif, dan biaya yang tinggi [2]. Hal ini menyebabkan keterlambatan dalam identifikasi dan perbaikan kerusakan jalan, yang pada akhirnya dapat memperburuk kondisi infrastruktur secara keseluruhan.

Berkembang pesatnya teknologi kecerdasan buatan (*Artificial Intelligence*) dan *computer vision*, membuka peluang untuk mengembangkan sistem deteksi kerusakan jalan yang lebih efisien dan akurat. Salah satu pendekatan yang menjanjikan adalah penggunaan algoritma *deep learning* untuk deteksi objek, seperti *Single Shot Detector (SSD)* dengan arsitektur *MobileNetV2*. *SSD MobileNetV2* dipilih karena kemampuannya dalam melakukan deteksi objek dengan akurasi tinggi namun tetap efisien dalam penggunaan sumber daya komputasi. Hal ini memungkinkan implementasi model pada berbagai jenis perangkat, termasuk perangkat *mobile* dan sistem *embedded*.

TensorFlow, sebagai salah satu *framework machine learning* yang populer, menyediakan *tools* dan *library* yang kuat untuk mengembangkan dan melatih model *SSD MobileNetV2*. Khususnya, *TensorFlow Object Detection API* menawarkan *framework* yang fleksibel untuk pengembangan model deteksi objek. *API* ini menyediakan berbagai

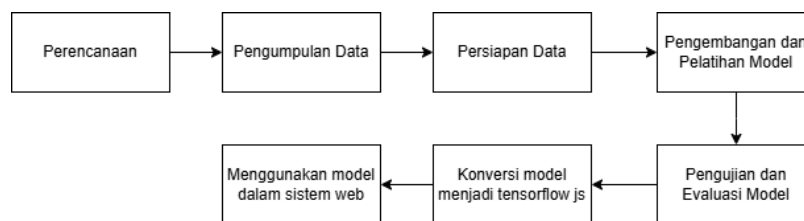
arsitektur model *pre-trained*, termasuk *SSD MobileNetV2*, yang dapat digunakan untuk *transfer learning* [3]. Hal ini memungkinkan pengembang untuk memanfaatkan model yang telah dilatih pada dataset besar untuk tugas-tugas spesifik dengan dataset yang lebih kecil.

Sementara itu, *TensorFlow.js* memungkinkan *deployment* model ke dalam lingkungan web, membuka peluang untuk pengembangan aplikasi berbasis browser yang dapat diakses secara luas. Dengan memanfaatkan kombinasi *TensorFlow Object Detection API* dan *TensorFlow.js*, penelitian ini dapat fokus pada optimisasi model untuk kasus spesifik deteksi kerusakan jalan, tanpa perlu membangun seluruh *pipeline* deteksi objek dari awal [4]. Penelitian ini akan mengembangkan model machine learning untuk mendeteksi kerusakan jalan otomatis menggunakan *SSD MobileNetV2* dengan *TensorFlow*, serta mengimplementasikannya dalam aplikasi web menggunakan *TensorFlow.js*.

Dengan menggabungkan kekuatan *machine learning*, *deep learning*, *computer vision*, dan teknologi web modern, penelitian ini berpotensi memberikan solusi inovatif untuk masalah pengelolaan infrastruktur jalan yang telah lama ada, sambil juga membuka jalan bagi pengembangan aplikasi *Artificial Intelligence* yang lebih luas di bidang manajemen infrastruktur perkotaan

II. METODE

Penelitian ini mengikuti beberapa tahapan untuk dapat melatih model yang dapat mendeteksi kerusakan jalan, Adapun langkah-langkah dalam membangun sistem ini adalah sebagai berikut:



Gambar 1. Urutan langkah penelitian

A. Perencanaan

Pada tahap ini, peneliti akan melakukan berbagai kegiatan untuk memahami topik penelitian, merumuskan tujuan dan ruang lingkup penelitian, mengidentifikasi data yang diperlukan, merancang sistem, dan membuat rencana penelitian. Kegiatan pemahaman topik penelitian bertujuan untuk memberikan pemahaman yang mendalam tentang topik penelitian, termasuk konsep-konsep dasar, teori-teori yang relevan, dan penelitian-penelitian terdahulu yang terkait.

B. Pengumpulan data

Untuk mengembangkan sistem deteksi kerusakan jalan yang efektif, langkah pertama adalah melakukan pengumpulan data. Dalam penelitian ini, digunakan dataset publik dari *Kaggle* [5] yang mencakup berbagai gambar kerusakan jalan. Dataset yang digunakan dalam penelitian ini sebanyak 1018 gambar kerusakan jalan.



Gambar 2. Sampel dataset yang digunakan

C. Persiapan data

Pada tahap ini, peneliti akan melakukan labeling, preprocessing dan augmentasi data untuk meningkatkan variasi dataset dan mempersiapkan data untuk pelatihan model, proses tersebut dilakukan di platform *Roboflow*. Tiga kategori kerusakan jalan yang akan dideteksi dalam penelitian ini: Retak, Lubang dan Tambalan.



Gambar 3. Contoh data setelah labeling

Untuk meningkatkan variasi dataset dan membantu model belajar lebih baik, dilakukan augmentasi data. Teknik augmentasi yang digunakan meliputi flipping horizontal dan vertikal, serta rotasi 90° searah jarum jam, berlawanan arah jarum jam, dan terbalik.

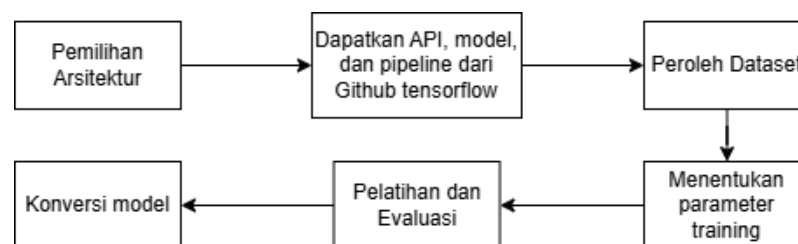


Gambar 4. Contoh data setelah proses augmentasi

Dataset yang sebelumnya berjumlah 1018 data, bertambah menjadi 1600 data setelah dilakukan proses augmentasi.

D. Pengembangan dan pelatihan model

Pada tahap ini, dilakukan pemilihan arsitektur model, mengimplementasikan model menggunakan *Tensorflow Object Detection API* dan melakukan *transfer learning* dan juga *fine tuning* pada model serta proses pelatihan model yang dilakukan pada platform *Google Colab*.



Gambar 5. Skenario pengembangan model

Arsitektur yang dipilih dalam penelitian kali ini adalah arsitektur *Single Shot Detector (SSD)* dengan *MobileNet V2* sebagai *backbone*. Pemilihan arsitektur ini didasarkan pada kebutuhan akan model yang ringan dan efisien namun tetap memiliki performa yang baik dalam deteksi objek.

Pengembangan model dilakukan menggunakan *TensorFlow Object Detection API*. Langkah ini melibatkan pengkodean dan konfigurasi model sesuai dengan arsitektur yang telah dipilih. Peneliti menyesuaikan konfigurasi model seperti total langkah pelatihan, *learning rate*, *optimizer* dan *hyperparameters* lainnya untuk mengoptimalkan performa model.

Setelah tahap konfigurasi sudah dilakukan, proses selanjutnya adalah pelatihan model. Proses pelatihan model dilakukan di *Google Colab* menggunakan *GPU* untuk mempercepat komputasi. Pelatihan berlangsung hingga model mencapai konvergensi atau performa tidak meningkat signifikan. Model terbaik dipilih berdasarkan performa tertinggi pada set validasi untuk digunakan pada tahap pengujian dan implementasi.

E. Pengujian dan evaluasi model

Pada tahap ini, model yang telah dilatih akan diuji dan dievaluasi untuk mengukur kinerjanya dalam mendeteksi kerusakan jalan. Model diuji dengan memvariasikan beberapa parameter pelatihan untuk menemukan konfigurasi optimal:

- Learning rate* mengontrol seberapa besar model menyesuaikan diri saat proses pembelajaran - nilai yang terlalu besar membuat model belajar terlalu cepat dan tidak akurat, sedangkan nilai terlalu kecil membuat model belajar terlalu lambat [6].
- Batch size* adalah jumlah data yang diproses dalam satu iterasi pembelajaran, mempengaruhi kecepatan training dan penggunaan memori komputer [7].
- Total step* adalah jumlah iterasi pembelajaran yang dilakukan model selama proses training, menentukan seberapa lama model akan belajar dari data yang diberikan.
- Split dataset* adalah teknik pembagian data menjadi beberapa bagian (training, validation, testing) untuk memastikan model dapat dilatih dan dievaluasi secara terpisah [8].

Performa model kemudian dievaluasi menggunakan metrik evaluasi yang umum digunakan yaitu *precision*, *recall*, *F1-score* dan *Total Loss*. *Precision* mengukur seberapa akurat model dalam memprediksi kelas positif. Dengan kata lain, dari semua prediksi yang diberikan oleh model sebagai positif, berapa banyak yang benar-benar positif [9].

Recall mengukur seberapa baik model dalam menangkap semua sampel positif yang ada. Dengan kata lain, dari semua sampel yang sebenarnya positif, berapa banyak yang berhasil diprediksi dengan benar oleh model [10].

F1-score digunakan untuk mengukur keseimbangan antara *precision* dan *recall* dalam model klasifikasi atau deteksi. [11]

Metrik-metrik ini digunakan untuk menilai performa model dalam mendeteksi dan mengklasifikasikan kerusakan jalan dengan akurat dan efektif.

F. Konversi model

Model yang menunjukkan hasil dan performa terbaik akan dipilih dan dikonversi menjadi model dalam format Javascript menggunakan *Tenflow JS converter*, agar dapat dijalankan dalam platform web.

III. HASIL DAN PEMBAHASAN

Deteksi kerusakan jalan dilakukan dengan memasukkan input berupa gambar melalui file gambar pada sistem, menghasilkan klasifikasi kerusakan jalan sesuai dengan jenis kerusakan yang terdeteksi. Proses *transfer learning* dan *fine tuning* dilakukan untuk mendapatkan hasil yang optimal dan stabil untuk digunakan.

A. Hasil pengujian model

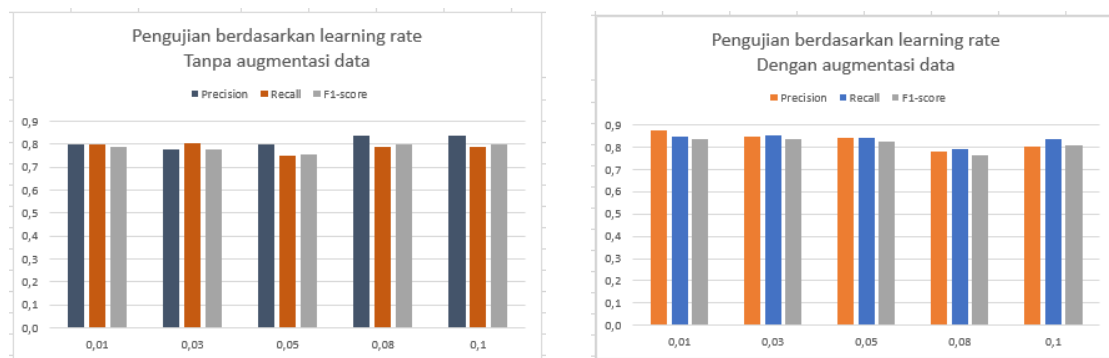
Hasil pengujian akan ditampilkan dalam bentuk tabel dan diagram untuk mempermudah proses analisis. Pengujian dilakukan dengan melakukan 5 kali percobaan untuk setiap konfigurasi parameter dan mengambil nilai rata-rata dari metrik *precision*, *recall*, *F1-score* dan *Total Loss* sebagai indikator performa model. Tabel 1 akan menunjukkan hasil pengujian berdasarkan *learning rate* yang berbeda untuk menentukan konfigurasi optimal.

Tabel 1. Tabel pengujian berdasarkan *learning rate*

Jumlah Dataset	Data Split (Train, valid, test)	Total step	Augmentasi Data	Batch Size	Learning rate	Precision	Recall	F1-score	Total loss
1018	60, 25, 15	1000	Tidak	32	0,01	0,797	0,801	0,790	0,42
1600	85, 10, 5	1000	Ya	32	0,01	0,874	0,848	0,838	0,49

1018	60, 25, 15	1000	Tidak	32	0,03	0,777	0,807	0,777	0,44
1600	85, 10, 5	1000	Ya	32	0,03	0,849	0,850	0,835	0,43
1018	60, 25, 15	1000	Tidak	32	0,05	0,798	0,753	0,757	0,35
1600	85, 10, 5	1000	Ya	32	0,05	0,843	0,840	0,826	0,45
1018	60, 25, 15	1000	Tidak	32	0,08	0,840	0,788	0,797	0,41
1600	85, 10, 5	1000	Ya	32	0,08	0,778	0,791	0,762	0,38
1018	60, 25, 15	1000	Tidak	32	0,1	0,838	0,872	0,806	0,34
1600	85, 10, 5	1000	Ya	32	0,1	0,801	0,836	0,809	0,40

Hasil pengujian berdasarkan *learning rate* menunjukkan performa terbaik pada nilai 0,1 untuk model tanpa augmentasi data dengan *total loss* terendah 0,34. Hal ini menunjukkan bahwa model dapat melakukan *update* parameter dengan langkah yang lebih besar dan mencapai konvergensi yang optimal. Sementara pada model dengan augmentasi data, *learning rate* 0,03 memberikan hasil terbaik dengan *total loss* 0,43. Ini mengindikasikan bahwa dengan dataset yang lebih besar dan beragam hasil augmentasi, model membutuhkan langkah *update* yang lebih kecil untuk mencapai keadaan yang stabil.



Gambar 6. Diagram pengujian learning rate

Dari visualisasi diagram pada gambar 6, dapat dilihat bahwa pada pengujian *learning rate*, model tanpa augmentasi data dengan *learning rate* 0,1 mencapai *precision* 83,8%, *recall* 87,2%, dan *F1-score* 80,6%. Sementara model dengan augmentasi data menunjukkan performa metrik terbaik pada *learning rate* 0,01 dengan *precision* 87,4%, *recall* 84,8%, dan *F1-score* 83,8%. Perbedaan ini menggambarkan bahwa meskipun model tanpa augmentasi mencapai *total loss* terendah, model dengan augmentasi data memiliki keseimbangan yang baik antara *precision* dan *recall* yang tercermin dalam nilai *F1-score* yang tinggi.

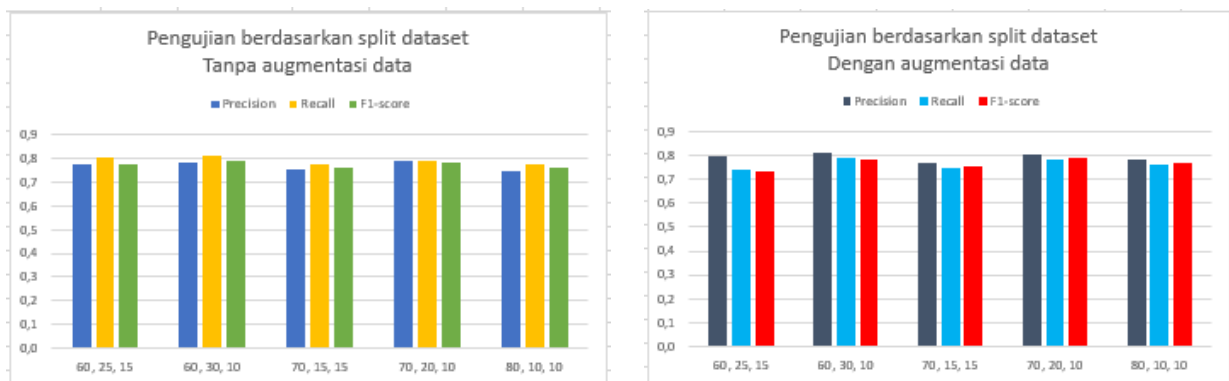
Setelah menganalisis hasil uji berdasarkan variasi *learning rate*, pembahasan berikutnya akan difokuskan pada hasil pengujian yang diperoleh dengan menggunakan variasi pembagian dataset. Data hasil pengujian tersebut disajikan dalam tabel 2 untuk dianalisis lebih lanjut.

Tabel 2. Tabel pengujian berdasarkan *split data*

Jumlah Dataset	Data Split (Train, valid, test)	Total step	Augmentasi Data	Batch Size	Learning rate	Precision	Recall	F1-score	Total loss
1018	60, 25, 15	1000	Tidak	32	0,03	0,777	0,807	0,776	0,51

1600	60, 25, 15	1000	Ya	32	0,03	0,797	0,742	0,734	0,54
1018	60, 30, 10	1000	Tidak	32	0,03	0,781	0,810	0,795	0,39
1600	60, 30, 10	1000	Ya	32	0,03	0,810	0,790	0,786	0,54
1018	70, 15, 15	1000	Tidak	32	0,03	0,752	0,780	0,765	0,53
1600	70, 15, 15	1000	Ya	32	0,03	0,768	0,748	0,755	0,65
1018	70, 20, 10	1000	Tidak	32	0,03	0,790	0,795	0,787	0,49
1600	70, 20, 10	1000	Ya	32	0,03	0,805	0,781	0,788	0,64
1018	80, 10, 10	1000	Tidak	32	0,03	0,745	0,775	0,760	0,66
1600	80, 10, 10	1000	Ya	32	0,03	0,785	0,765	0,772	0,53

Hasil pengujian dengan variasi split data menunjukkan performa terbaik pada pembagian 60:30:10 (*train:valid:test*) untuk model tanpa augmentasi data dengan *total loss* terendah 0,39. Rasio pembagian ini memberikan keseimbangan yang baik antara data pelatihan yang cukup untuk pembelajaran model dan data validasi yang memadai untuk evaluasi. Sementara untuk model dengan augmentasi data, pembagian 80:10:10 justru memberikan hasil yang lebih baik dengan *total loss* 0,53. Hal ini menunjukkan bahwa dengan dataset yang lebih besar dari hasil augmentasi, model dapat memanfaatkan porsi data training yang lebih besar tanpa mengalami *overfitting* yang signifikan.



Gambar 7. Diagram pengujian split dataset

Dari visualisasi diagram pada Gambar 7, dapat dilihat bahwa pada pengujian pembagian dataset, model tanpa augmentasi data mencapai performa terbaik pada pembagian 70:20:10 dengan *precision* 79%, *recall* 79,5%, dan *F1-score* 78,7%. Sementara itu, model dengan augmentasi data (1600 data) menunjukkan hasil optimal pada pembagian 60:30:10 dengan *precision* 81%, *recall* 79%, dan *F1-score* 78,6%.

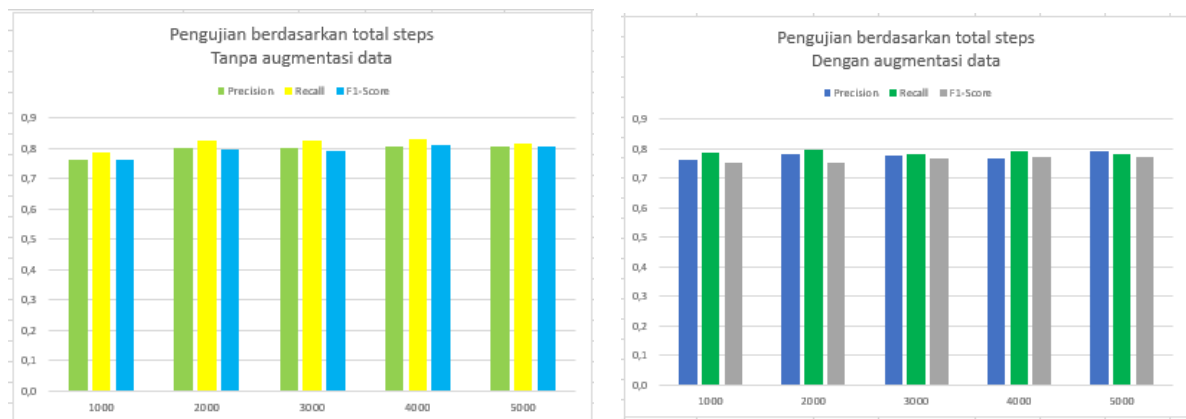
Berikutnya, pembahasan akan beralih pada analisis hasil uji yang melibatkan variasi total langkah. Data hasil pengujian ini disajikan dalam tabel 3 untuk memberikan gambaran yang lebih jelas terkait pengaruh total steps terhadap performa model.

Tabel 3. Tabel pengujian berdasarkan *total steps*

Jumlah Dataset	Data Split (Train, valid, test)	Total Steps	Augmentasi Data	Batch Size	Learning Rate	Precision	Recall	F1-Score	Total loss
----------------	------------------------------------	-------------	-----------------	------------	---------------	-----------	--------	----------	------------

1018	60, 25, 15	1000	Tidak	32	0,01	0,765	0,789	0,762	0,43
1600	60, 25, 15	1000	Ya	32	0,01	0,763	0,786	0,753	0,53
1018	60, 25, 15	2000	Tidak	32	0,01	0,801	0,826	0,797	0,34
1600	60, 25, 15	2000	Ya	32	0,01	0,781	0,799	0,754	0,49
1018	60, 25, 15	3000	Tidak	32	0,01	0,800	0,828	0,794	0,33
1600	60, 25, 15	3000	Ya	32	0,01	0,776	0,781	0,767	0,39
1018	60, 25, 15	4000	Tidak	32	0,01	0,805	0,830	0,810	0,35
1600	60, 25, 15	4000	Ya	32	0,01	0,768	0,790	0,773	0,42
1018	60, 25, 15	5000	Tidak	32	0,01	0,808	0,819	0,806	0,35
1600	60, 25, 15	5000	Ya	32	0,01	0,790	0,785	0,775	0,42

Hasil pengujian variasi total steps menunjukkan performa terbaik pada 3000 langkah untuk model tanpa augmentasi data dengan *total loss* terendah 0,33. Peningkatan jumlah langkah dari 1000 ke 3000 memungkinkan model untuk melakukan pembelajaran lebih optimal dan mencapai kestabilan yang lebih baik. Namun, penambahan langkah lebih lanjut ke 4000 dan 5000 justru menghasilkan sedikit peningkatan *total loss* menjadi 0,35, yang mengindikasikan bahwa model telah mencapai titik optimal pada 3000 steps. Pada model dengan augmentasi data, performa terbaik juga dicapai pada 3000 steps dengan *total loss* 0,39, menunjukkan bahwa meskipun dataset lebih besar, jumlah langkah yang sama sudah cukup untuk mencapai titik yang optimal.



Gambar 8. Diagram pengujian *total steps*

Gambar 8 menampilkan visualisasi model dengan augmentasi mengalami peningkatan performa pada model dengan 5.000 langkah, dengan *precision* 79%, *recall* 78% dan *F1-score* 77%. Model tanpa augmentasi mencapai performa terbaik pada 4.000 langkah, dengan *precision* 80%, *recall* 83%, dan *F1-score* 81%.

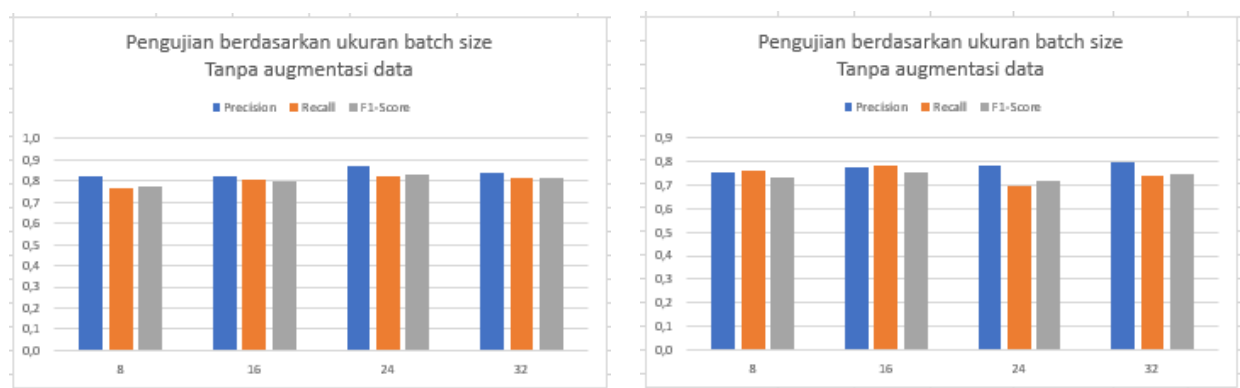
Tabel 4 akan menunjukkan hasil pengujian berdasarkan ukuran *batch size* yang digunakan untuk mengetahui perbedaan yang diberikan.

Tabel 4. Tabel pengujian berdasarkan ukuran *batch size*

Jumlah Dataset	Data Split (Train,	Total Step	Augmentasi Data	Batch Size	Learning Rate	Precision	Recall	F1-Score	Total loss
----------------	--------------------	------------	-----------------	------------	---------------	-----------	--------	----------	------------

<i>valid, test)</i>									
1018	60, 25, 15	1000	Tidak	8	0,03	0,823	0,765	0,778	0,44
1600	60, 25, 15	1000	Ya	8	0,03	0,753	0,759	0,730	0,89
1018	60, 25, 15	1000	Tidak	16	0,03	0,826	0,808	0,800	0,46
1600	60, 25, 15	1000	Ya	16	0,03	0,778	0,784	0,755	0,55
1018	60, 25, 15	1000	Tidak	24	0,03	0,868	0,824	0,831	0,41
1600	60, 25, 15	1000	Ya	24	0,03	0,787	0,697	0,722	0,45
1018	60, 25, 15	1000	Tidak	32	0,03	0,836	0,814	0,815	0,56
1600	60, 25, 15	1000	Ya	32	0,03	0,797	0,742	0,744	0,52

Hasil pengujian variasi *batch size* menunjukkan performa terbaik pada *batch size* 24 untuk model tanpa augmentasi data dengan *total loss* 0,41. Ukuran batch ini memberikan keseimbangan yang optimal antara kecepatan pembelajaran dan stabilitas model. Serupa dengan hal tersebut, model dengan augmentasi data juga menunjukkan performa terbaik pada *batch size* 24 dengan *total loss* 0,45. Menariknya, peningkatan *batch size* menjadi 32 justru mengakibatkan kenaikan total loss baik pada model tanpa augmentasi (0,56) maupun dengan augmentasi (0,52), yang mengindikasikan bahwa *batch size* yang terlalu besar dapat mengurangi kemampuan model untuk belajar dari variasi data secara efektif.



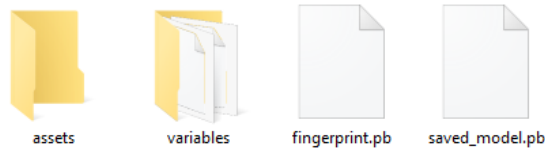
Gambar 9. Diagram garis pengujian ukuran *batch size*

Gambar 9 menampilkan model dengan augmentasi data mencapai performa optimal pada *batch size* 16 dengan *precision* 77%, *recall* 78% dan *F1-score* 75%. Sementara itu, model tanpa augmentasi menunjukkan hasil terbaik pada *batch size* 24 dengan *precision* 86%, *recall* 82% dan *F1-score* 83%. Performa model cenderung menurun pada ukuran batch yang lebih besar.

Berdasarkan keseluruhan pengujian yang dilakukan, performa terbaik dicapai oleh model tanpa augmentasi data dengan kombinasi parameter: *learning rate* 0,1 (*total loss* 0,34), rasio split data 60:30:10 (*total loss* 0,39), *total steps* 3000 (*total loss* 0,33), dan *batch size* 24 (*total loss* 0,41). Hasil ini menunjukkan bahwa model tanpa augmentasi data dapat mencapai kestabilan model yang lebih baik, kemungkinan karena pembelajaran yang lebih fokus pada variasi data asli. Model dengan augmentasi data, meskipun memiliki total loss yang sedikit lebih tinggi, menunjukkan kelebihan dalam hal generalisasi dengan dataset yang lebih besar. Secara keseluruhan, pemilihan parameter yang tepat terbukti sangat mempengaruhi kemampuan model dalam mencapai keadaan optimal untuk tugas deteksi kerusakan jalan.

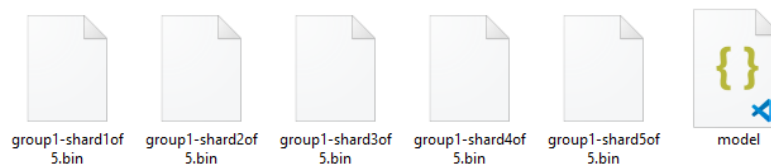
B. Konversi model

Setelah dilakukan evaluasi, model yang menunjukkan performa yang stabil antara precision, recall, dan loss pada data uji, dipilih untuk digunakan lebih lanjut. Model ini kemudian dikonversi ke format TensorFlow.js agar dapat diimplementasikan pada aplikasi berbasis web, memungkinkan deteksi kerusakan jalan dilakukan secara langsung melalui browser.



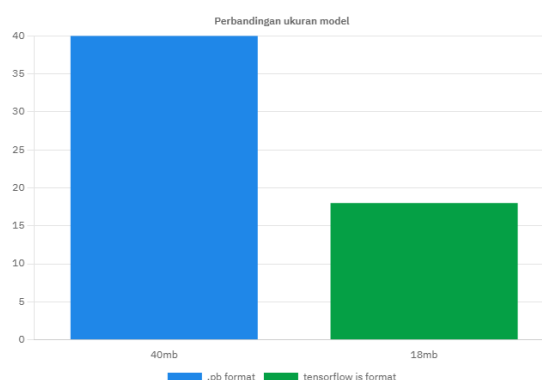
Gambar 10. Model hasil pelatihan

Gambar 10 memperlihatkan format model *TensorFlow* yang disimpan dengan ekstensi *.pb* (*Protocol Buffer*), yang merupakan format standar untuk menyimpan model *deep learning*. Pemilihan format ini sejalan dengan penelitian [12] yang juga memanfaatkan platform *TensorFlow* untuk implementasi model deteksi kerusakan jalan, dimana format ini terbukti efektif dalam menyimpan graph model dan parameter-parameternya secara efisien.



Gambar 11. Model hasil konversi

Sementara itu, Gambar 11 menunjukkan hasil konversi model ke format *TensorFlow.js* yang memungkinkan eksekusi model langsung di *browser*. Untuk mengintegrasikan model ke dalam web, diperlukan dua *file* utama hasil konversi yaitu *model.json* yang berisi arsitektur model dan *weight.bin* yang menyimpan bobot model. *File model.json* berperan sebagai manifest yang mendeskripsikan struktur model dan referensi ke file *weight.bin*. Kedua file ini kemudian dimuat menggunakan *API TensorFlow.js* yang memungkinkan model dapat dijalankan langsung di *browser* menggunakan *WebGL*. Pendekatan konversi ke *TensorFlow.js* ini merupakan pengembangan dari penelitian sebelumnya [13], dimana model masih menggunakan implementasi berbasis server. Dengan implementasi berbasis browser, sistem dapat melakukan komputasi di sisi *client* yang berpotensi meningkatkan efisiensi pemantauan kondisi jalan sekaligus mengurangi kebutuhan komputasi server dan meminimalkan *latency* dalam proses deteksi kerusakan jalan.

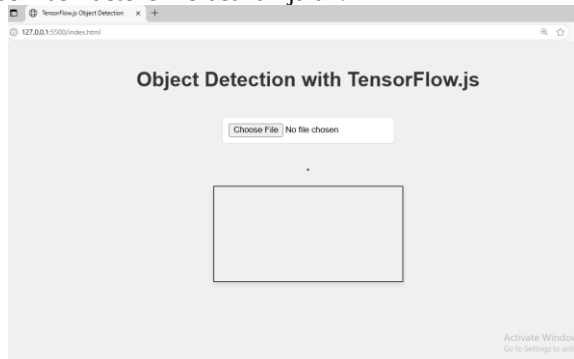


Gambar 12. Perbandingan ukuran model

Proses konversi menghasilkan perubahan ukuran model yang cukup signifikan, dimana model awal dalam format *.pb* memiliki ukuran 40 MB, setelah dikonversi menjadi format *TensorFlow.js* ukurannya menjadi 18 MB. Perubahan ukuran ini terjadi karena proses optimasi dan kuantisasi yang dilakukan oleh *TensorFlow.js converter*, namun tetap mempertahankan kemampuan model dalam melakukan deteksi.

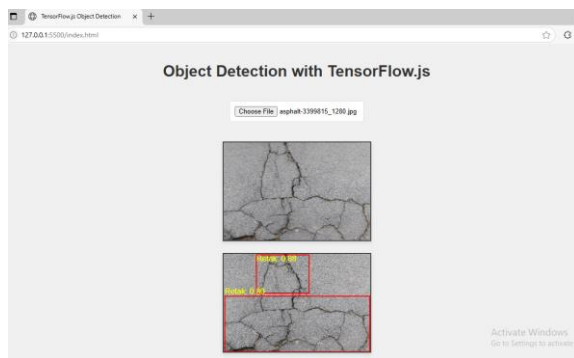
C. Tampilan web

Gambar 13 menunjukkan tampilan antarmuka web yang telah dikembangkan, dengan fitur utama berupa area unggah gambar dan visualisasi hasil deteksi kerusakan jalan.



Gambar 13. Tampilan input gambar

Gambar 14 menunjukkan hasil deteksi ketika gambar telah berhasil dimasukkan oleh pengguna. Sistem akan memuat model untuk mendeteksi jenis kerusakan jalan dan menampilkan hasilnya dalam bentuk *bounding box* pada area kerusakan beserta *confidence score* yang menunjukkan tingkat keyakinan deteksi..



Gambar 14. Tampilan hasil deteksi gambar

IV. SIMPULAN

Penelitian ini bertujuan untuk mengembangkan model deteksi kerusakan jalan menggunakan arsitektur *SSD MobileNet* yang dapat mengidentifikasi jenis kerusakan dengan presisi dan keakuratan tinggi. Hasil penelitian menunjukkan bahwa model yang diimplementasikan berhasil mencapai performa terbaik pada model tanpa augmentasi data, dengan kombinasi parameter optimal yaitu *learning rate* 0,1 (*total loss* 0,34), rasio split data 60:30:10 (*total loss* 0,39), *total steps* 3000 (*total loss* 0,33), dan *batch size* 24 (*total loss* 0,41). Model ini menunjukkan kemampuan yang lebih stabil dan optimal dalam mendeteksi kerusakan jalan.

Model ini juga berhasil dikonversi ke format *TensorFlow.js*, memungkinkan penerapan pada aplikasi berbasis web yang praktis dan mudah diakses. Dengan konfigurasi yang optimal, model mampu memberikan deteksi yang akurat pada kondisi pengambilan gambar tertentu, meskipun performa menurun pada gambar dengan visibilitas rendah atau sudut pengambilan yang tidak ideal.

Untuk pengembangan selanjutnya, penelitian ini dapat ditingkatkan dengan menambahkan variasi data pelatihan untuk menangani berbagai kondisi gambar yang lebih kompleks. Selain itu, pengembangan kemampuan model untuk mengklasifikasikan jenis dan tingkat keparahan kerusakan jalan secara lebih rinci dapat menjadi langkah lanjutan untuk meningkatkan manfaat praktis model ini.

UCAPAN TERIMA KASIH

Terimakasih kepada Universitas Muhammadiyah Sidoarjo dan seluruh pihak yang telah mendukung kelancaran dalam menulis artikel.

REFERENSI

- [1] I. Himmati, I. P. Prawesti, and A. A. Brahmantya, "ROADECT: DETEKSI PERSOALAN INFRASTRUKTUR JALAN SEBAGAI SOLUSI STRATEGI DIGITAL DALAM PENYAMPAIAN ASPIRASI MASYARAKAT BERKELANJUTAN," 2023.
- [2] A. N. Utomo and N. Lestari, "APLIKASI DETEKSI KERUSAKAN JALAN RAYA MENGGUNAKAN ALGORITMA K-NN (K-NEAREST NEIGHBOUR) ROAD DETECTION APPLICATION USING K-NN ALGORITHM (K-NEAREST NEIGHBOUR)," vol. 10, no. 1, 2021.
- [3] P. R. Aningtiyas, A. Sumin, and S. Wirawan, "Pembuatan Aplikasi Deteksi Objek Menggunakan TensorFlow Object Detection API dengan Memanfaatkan SSD MobileNet V2 Sebagai Model Pra - Terlatih," *Jurnal Ilmiah Komputasi*, vol. 19, no. 3, Mar. 2020, doi: 10.32409/jikstik.19.3.68.
- [4] P. D. Arnesia, N. A. Pratama, and F. Sjafrina, "APLIKASI ARTIFICIAL INTELLIGENCE UNTUK MENDETEKSI OBJEK BERBASIS WEB MENGGUNAKAN LIBRARY TENSORFLOW JS, REACT JS DAN COCO DATASET," 2022.
- [5] A. Basily, "Road Damage," <https://www.kaggle.com/datasets/alvarobasily/road-damage>.
- [6] K. Kualitas Buah Salak dengan Transfer Learning Arsitektur VGG and A. Luthfiarta, "VGG16 Transfer Learning Architecture for Salak Fruit Quality Classification," *Jurnal Informatika dan Teknologi Informasi*, vol. 18, no. 1, pp. 37–48, 2021, doi: 10.31515/telematika.v18i1.4025.
- [7] A. Julianto, A. Sunyoto, D. Ferry, and W. Wibowo, "OPTIMASI HYPERPARAMETER CONVOLUTIONAL NEURAL NETWORK UNTUK KLASIFIKASI PENYAKIT TANAMAN PADI (OPTIMIZATION OF CONVOLUTIONAL NEURAL NETWORK HYPERPARAMETERS FOR CLASSIFICATION OF RICE PLANT DISEASES)." [Online]. Available: <https://www.kaggle.com/tedisetiady/leaf-rice-dis->
- [8] E. Muningsih, "KOMBINASI METODE K-MEANS DAN DECISION TREE DENGAN PERBANDINGAN KRITERIA DAN SPLIT DATA," 2022.
- [9] Z. Syahputra, "Penerapan SSD-MobileNet Dalam Identifikasi Jenis Buah Apel," *Indonesian Journal of Education And Computer Science*, vol. 1, no. 1, 2023.
- [10] C. Anwar, "Deteksi Objek Berbasis Web Menggunakan Tensorflow Js dan Coco Dataset pada Framework React Js," 2022.
- [11] "Klasifikasi Penderita Diabetes menggunakan Algoritma Machine Learning dan Z-Score".
- [12] A. Pratama, "Deteksi Kerusakan Jalan Menggunakan Convolutional Neural Network Berbasis SSD EfficientNet Artikel Ilmiah," 2020.
- [13] B. Sasmito, B. H. Setiadji, and R. Isnanto, "Deteksi Kerusakan Jalan Menggunakan Pengolahan Citra Deep Learning di Kota Semarang," *TEKNIK*, vol. 44, no. 1, pp. 7–14, May 2023, doi: 10.14710/teknik.v44i1.51908.

Conflict of Interest Statement:

The author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.