Deteksi Objek Masker Menggunakan *Object Detection* API dan *TensorFlow Lite Model Maker*

Disusun Oleh:

N a m a : Ferdian Nursulistio

NIM : 18523149

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2022

HALAMAN PENGESAHAN DOSEN PEMBIMBING

Deteksi Objek Masker Menggunakan *Object Detection* API dan *TensorFlow Lite Model Maker*

TUGAS AKHIR

Disusun Oleh:

N a m a : Ferdian Nursulistio

NIM : 18523149

Yogyakarta, 2 November 2022

Pembimbing 1,

(Arrie Kurniawardhani, S.Si., M.Kom.)

Yogyakarta, 2 November 2022

Pembimbing 2,

(Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D.)

HALAMAN PENGESAHAN DOSEN PENGUJI

Deteksi Objek Masker Menggunakan *Object Detection* API dan *TensorFlow Lite Model Maker*

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 2 November 2022

Tim Penguji

Arrie Kurniawardhani, S.Si., M.Kom. _____

Anggota 1

Andhika Giri Persada, S.Kom., M.Eng. _____

Anggota 2

Elyza Gustri Wahyuni, S.T., M.Cs. _____

Mengetahui,

Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia

(Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Ferdian Nursulistio

NIM : 18523149

Tugas akhir dengan judul:

Deteksi Objek Masker Menggunakan *Object Detection* API dan *TensorFlow Lite Model Maker*

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 12 Oktober 2022

(Ferdian Nursulistio)

HALAMAN PERSEMBAHAN

Assalamualaikum Warahmatullahi Wabarakatuh

Alhamdulillahi rabbil ‘alamin, puji syukur peneliti ucapkan kepada Allah SWT berkat izin dan ridho-Nya yang telah memberikan kesehatan, kelancaran, kemudahan, dan keberkahan selama proses pengerjaan hingga penyelesaian tugas akhir ini. Dengan penyelesaian tugas akhir ini, semoga dapat bermanfaat untuk diri saya dan orang yang membaca tugas akhir ini. Halaman ini saya persembahkan kepada:

Orang tua penulis, Bapak Nano G. Warsono dan Ibu Nuriah yang selalu memberikan kesempatan untuk menempuh pendidikan tinggi di Universitas Islam Indonesia. Tak lupa ucapan terima kasih atas segala jenis dukungan yang sudah diberikan seperti dukungan moral, material dan doa yang selalu dipanjatkan pada setiap ibadah yang dijalankan. Terima kasih juga kepada keluarga penulis atas dorongan dan harapan yang diberikan kepada penulis sehingga mampu mencapai tahap ini.

Kepada Ibu Arrie Kurniawardhani, S.Si., M.Kom. dan Bapak Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D., saya ucapkan terimakasih sudah memberikan arahan, usulan, dan bimbingan selama proses pengerjaan sampai penulis dapat menyelesaikan tugas akhir ini sebagai dosen pembimbing.

Semua teman-teman penulis yang menemani, membantu dan memberi dukungan selama proses pengerjaan tugas akhir ini. Terimakasih buat teman-teman atas bantuannya selama ini.

HALAMAN MOTO

"Janganlah kamu bersikap lemah, dan janganlah (pula) kamu bersedih hati, padahal kamulah orang-orang yang paling tinggi (derajatnya), jika kamu orang-orang yang beriman."

(Q.S. Ali Imran ayat 139)

"*A person who cannot give up anything, can change nothing*"

(Armin Arlert)

KATA PENGANTAR

Alhamdulillahirabbil alamin kita panjatkan puji dan syukur kepada Allah S.W.T. atas berkat, rahmat, dan karunianya kepada kita para hamba-Nya. Atas seizin Allah, penulis mampu menyelesaikan tugas akhir dengan judul "Deteksi Objek Masker Menggunakan *Object Detection* API dan *TensorFlow Lite Model Maker*" yang merupakan salah satu syarat yang harus dipenuhi untuk memperoleh gelar Sarjana Komputer pada Program Studi Informatika, Fakultas Teknologi Industri, Universitas Islam Indonesia. Diharapkan dengan selesainya tugas akhir ini dapat membawa manfaat kepada baik kepada diri penulis sendiri, pembaca laporan ini, maupun peneliti selanjutnya yang memiliki topik serupa.

Segala kerja keras dalam penyusunan tugas akhir ini tidak lepas dari bantuan, dukungan, arahan, dan masukan dari berbagai pihak sehingga penelitian ini dapat terselesaikan dengan baik. Oleh karena itu, peneliti ingin menyampaikan ucapan terimakasih sebesar-besarnya kepada:

1. Allah S.W.T. yang telah memberikan kesehatan, rezeki, dan petunjuk sehingga peneliti dapat Menyusun dan menyelesaikan tugas akhir ini dengan baik.
2. Orang tua penulis yang selalu memberi dukungan moral, material, dan mendoakan yang terbaik dalam setiap kehidupan penulis selama masa perkuliahan.
3. Prof. Dr. Ir. Hari Purnomo., M.T., IPU., ASEAN,Eng selaku Dekan Fakultas Teknologi Industri Universitas Islam Indonesia.
4. Dr. Raden Teduh Dirgahayu, S.T., M.Sc. selaku Ketua Jurusan Fakultas Teknologi Industri Universitas Islam Indonesia.
5. Bapak Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D. selaku Ketua Program Studi Informatika, Fakultas Teknologi Industri Universitas Islam Indonesia.
6. Ibu Arrie Kurniawardhani, S.Si., M.Kom. dan Bapak Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D. selaku dosen pembimbing yang telah memberikan banyak saran dan masukan kepada penulis sehingga penulis dapat menyelesaikan tugas akhir ini dengan baik.
7. Seluruh Dosen dan Staf prodi Informatika yang telah memberikan ilmunya selama penulis menjalankan perkuliahan di Universitas Islam Indonesia.
8. Sahabat penulis yang selalu mengajak *refreshing* selama proses penulisan tugas akhir dan memberikan semangat untuk terus mengerjakan tugas akhir.
9. Teman-teman kuliah penulis yang membantu memahami materi perkuliahan selama proses perkuliahan.

Dengan selesainya tugas akhir ini, penulis mengharapkan laporan ini dapat memberikan manfaat untuk para pembaca. Meski begitu, peneliti menyadari bahwa tugas akhir ini memiliki banyak kekurangan dan jauh dari kata sempurna. Oleh karena itu penulis menerima segala masukan dan kritik yang membangun dari pembaca agar penulis dapat melakukan penelitian yang lebih baik lagi pada penelitian selanjutnya.

Yogyakarta, 12 Oktober 2022

(Ferdian Nursulistio)

SARI

Masker merupakan benda yang digunakan untuk menutupi mulut dan hidung untuk mencegah menghirup dan melepaskan cairan pernafasan. Jenis masker dibagi dua, yaitu masker medis dan non-medis yang biasa dibuat dari tekstil atau bahan lainnya yang bisa dipakai berulang kali. Baik menggunakan masker medis maupun masker kain keduanya dapat menyaring dengan baik partikel atau cairan dengan tingkat efisiensi 86.4% untuk masker kain dan 99.9% untuk masker medis 3M. Masker digunakan untuk mencegah penyakit menular seperti pada pandemi COVID-19 serta juga digunakan untuk melindungi petugas kesehatan untuk mencegah Infeksi Nosokomial di rumah sakit. Maka dibuatlah model object detection untuk mendeteksi apakah seseorang menggunakan masker medis, masker kain, masker scuba, tidak menggunakan masker dengan benar, atau tidak menggunakan masker. Metode yang digunakan adalah *TensorFlow Object Detection* API dan *TensorFlow Lite Model Maker* dengan model dasar *SSD MobileNet V2 FPNLite 320x320* dan *SSD ResNet50 V1 FPN 640x640 (RetinaNet50)* pada metode *TensorFlow Object Detection* API serta *EfficientDet-Lite0* dan *EfficientDet-Lite3* pada metode *TensorFlow Lite Model Maker*. Keempat model mendapat *Average Precicion* (AP) sebesar 72.4%, 71.11%, 70.43%, 75.84% untuk model *MobileNet V2 FPNLite, SSD ResNet50 V1 FPN, EfficientDet-Lite0* dan *EfficientDet-Lite3* berturut-turut. Keempat model ini diimplementasikan pada perangkat seluler.

Pada perangkat seluler akan dilakukan pengujian dengan mendeteksi kelima kelas dengan tiga jarak yaitu 35 cm (*close-up)*, 100 cm (sedang), 150 cm (jauh). Hasil pengujian menunjukan bahwa model dengan arsitektur *EfficientDet-Lite3* memiliki tingkat akurasi terbaik, diikuti dengan *MobileNet V2 FPNLite* dan *EfficientDet-Lite0* yang keduanya memiliki tingkat akurasi yang hampir sama, lalu *ResNet50 V1 FPN* yang tidak dilakukan pengujian.

Kata kunci: Deteksi Objek Masker, *TensorFlow Lite Model Maker, Object Detection* API

GLOSARIUM

Machine Learning	Bagian dari kecerdasan buatan untuk membuat prediksi berdasarkan data yang disiapkan.
Dataset	Sekumpulan data yang digunakan untuk melatih model.
Tensorboard	Alat untuk visualisasi pada eksperimen *Machine Learning.*
Hyperparameter	Parameter yang nilainya digunakan untuk mengontrol proses pembelajaran.
Step(s)	Satu iterasi pada pelatihan model.
Epoch	Satu iterasi penuh melewati seluruh data pada dataset dan memiliki beberapa atau banyak *steps*.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Masker wajah adalah istilah luas yang digunakan untuk benda apapun yang dikenakan di atas mulut dan hidung untuk mencegah menghirup cairan penyakit menular atau pelepasan cairan penyakit menular yang berasal dari sistem pernafasan. Cairan tersebut dihasilkan Ketika bernafas, berbicara, batuk, dan bersin. Salah satu jenis masker adalah masker medis dan masker non-medis. Masker medis adalah masker sekali pakai yang biasa digunakan pada petugas Kesehatan sedangkan masker non medis adalah berbagai macam bentuk masker yang dapat dibuat sendiri termasuk masker yang dapat digunakan berkali-kali yang dibuat dari kain atau tekstil lainnya (Ecdc, 2022).

Masker sangatlah direkomendasikan karena masa inkubasi virus yang lama dan terdapat pasien positif yang tidak memiliki gejala (Lotfi et al., 2020). Menggunakan masker dapat mencegah cairan pernafasan penderita lepas ke lingkungan sekitar dan dihirup oleh orang sehat lainnya (Ecdc, 2022). Baik menggunakan masker medis maupun masker kain keduanya dapat menyaring dengan baik partikel atau cairan dengan tingkat efisiensi 86.4% untuk masker kain dan 99.9% untuk masker medis 3M (Ho et al., 2020) sehingga dapat menurunkan tingkat penularan penyakit menular.

Sebagai contoh, penggunaan masker saat melawan *Coronavirus Disease 19* (COVID-19) yang merupakan sebuah infeksi virus dengan tingkat penularan yang sangat tinggi yang disebabkan oleh *severe acute respiratory syndrome coronavirus 2* (SARS-CoV-2) (Shereen et al., 2020). pada tahun 2019 sampai 24 Mei 2022, WHO mencatat ada lebih dari 523 juta kasus positif dengan lebih dari enam juta orang meninggal di seluruh dunia. Di antaranya, 6,053,109 kasus positif berada di Indonesia dengan 156 ribu orang meninggal. COVID-19 dapat menular dengan beberapa cara, yaitu melalui kontak langsung seperti batuk, bersin, atau saat berbicara dan secara tidak langsung melalui benda yang sudah terkena cairan pernafasan penderita COVID-19 dan melalui udara (Lotfi et al., 2020). Selain pada kasus Covid, masker juga penting digunakan sebagai alat pelindung diri pada petugas kesehatan untuk mencegah Infeksi Nosokomial di rumah sakit (Salawati Liza, 2012).

Pentingnya memakai masker membuat berbagai penelitian muncul untuk mendeteksi apakah seseorang mengenakan masker menggunakan *object detection. Object detection*

memiliki kemampuan dalam mendeteksi sebuah objek dalam suatu gambar seperti wajah, tumbuhan, kendaraan, melacak pergerakan orang, atau objek lainnya sehingga automasi sebuah pekerjaan dapat ditingkatkan (Zhao et al., 2018). Contoh penelitian terkait deteksi masker adalah penelitian yang dilakukan oleh Shashi Yadav yang melakukan penelitian dengan judul "*Deep Learning based Safe Social Distancing and Face Mask Detection in Public Areas for COVID-19 Safety Guidelines Adherence"*. Penelitian ini menggunakan *MobileNetV2* dan *SSD MultiBox* untuk membuat sistem yang dapat memantau jarak antara orang dan mendeteksi seseorang yang memakai atau tidak memakai masker (Yadav, 2020). Penelitian lain dilakukan oleh Mohammad Marufur Rahman dkk yang membuat sistem untuk mendeteksi orang yang tidak memakai masker. Penelitian ini diberi judul *"An automated system to limit COVID-19 using facial mask detection in smart city network".* Sistem yang dibuat menggunakan arsitektur *Convolutional Neural Network* (CNN). CCTV digunakan untuk mengambil gambar lalu mendeteksi apakah orang tersebut menggunakan masker atau tidak pada ruang publik. Dalam kedua penelitian tersebut, deteksi yang dilakukan hanya sebatas apakah seseorang memakai masker atau tidak mengenakan masker.

Memakai masker sangat penting, tetapi terdapat kesulitan serta memerlukan usaha lebih dalam memantau satu persatu dan mengingatkan apakah seseorang menggunakan masker sesuai standar. Maka dari itu dibuatlah model object detection untuk mendeteksi apakah seseorang mengenakan masker medis, masker kain, masker scuba, tidak memakai masker, atau tidak mengenakan masker dengan tepat dengan menggunakan model dasar *MobileNet V2 FPNLite*, *EfficientDet-Lite0* dan *EfficientDet-Lite3* yang mengedepankan efisiensi dan *SSD ResNet50 V1 FPN* yang memiliki arsitektur mendalam sehingga dapat menemukan arsitektur yang terbaik berdasarkan nilai *Average Precision* (AP) dan besarnya latensi dalam melakukan deteksi pada kelima objek tersebut pada perangkat seluler.

1.2 Rumusan Masalah

Rumusan masalah untuk penelitian ini adalah bagaimana membuat sebuah model *object detection* dengan lima kelas yang berbeda dan mencari model yang terbaik berdasarkan nilai *Average Precision* (AP) dan besarnya latensi dalam melakukan deteksi pada kelima kelas pada perangkat seluler?

1.3 Batasan Masalah

Untuk menjaga ruang lingkup penelitian tetap fokus dan tidak terlalu lebar, pada tugas akhir ini diperlukan batasan masalah. Adapun batasan masalah terkait penelitian ini adalah:

a. Kelas yang akan dideteksi adalah masker medis, masker kain, masker scuba, tidak mengenakan masker dengan benar, dan tidak mengenakan masker.
b. Tidak dilakukan optimalisasi ukuran dan latensi model pendeteksi masker saat mengkonversi model ke *tensorflow lite*.
c. Tidak membuat model dari awal.

1.4 Tujuan

Penelitian ini dilakukan dengan tujuan membuat dan menemukan model *object detection* terbaik untuk mendeteksi apakah seseorang mengenakan masker medis, masker kain, masker scuba, tidak mengenakan masker dengan tepat, atau tidak memakai masker.

1.5 Manfaat

Manfaat dari penelitian ini adalah:

a. Mendeteksi orang yang tidak menggunakan masker sesuai aturan
b. Dapat membantu mengingatkan seseorang yang tidak menggunakan masker sesuai aturan.

1.6 Metodologi Secara Umum

a. Literature review/kajian Pustaka

Tahap ini melibatkan pencarian literatur untuk mencari referensi dan membandingkan penelitian yang sudah ada dengan apa yang akan dibuat. Kekurangan dalam penelitian sebelumnya dapat dijadikan acuan untuk membuat penelitian ini lebih baik.

b. Pengumpulan data

Mencari foto yang nantinya digunakan sebagai data *train*, data *validation*, dan data *test*. Sumber dapat dapat diperoleh dari *masked face dataset*, *scrapping instagram*, dan *google images.*

c. Pelabelan data

Menandai objek yang akan dideteksi pada foto yang sudah dikumpulkan. Label yang akan dibuat masker scuba, masker kain, dan masker medis, penggunaan masker salah, tanpa masker. Pelabelan data menggunakan LabelImg.

d. Pemilihan *pre-trained* model

Menentukan model yang memiliki proporsi akurasi dan kecepatan yang terbaik. Menggunakan lebih dari satu *pre-trained* model dan membandingkannya satu sama lain sehingga dapat diketahui model yang terbaik untuk tugas akhir ini.

e. Evaluasi

Mengukur tingkat keberhasilan model yang dilihat dari besaran akurasi dan kecepatan model mendeteksi masker. Tingkat keberhasilan diukur saat model berada di *Google Colab* dan perangkat seluler.

1.7 Sistematika penulisan

Sistematika penulisan adalah urutan atau struktur untuk menyelesaikan penelitian agar hasil penelitian ini tersusun secara rapi. Adapun sistematika penulisan penelitian ini adalah:

1.7.1 BAB I PENDAHULUAN

Pendahuluan adalah langkah penelitian pertama yang menjelaskan latar belakang peneliti mengambil topik ini, rumusan masalah yang akan dibahas, batasan masalah agar topik penelitian tidak meluas, tujuan penelitian, manfaat penelitian, metodologi umum, dan sistematika penulisan.

1.7.2 BAB II LANDASAN TEORI

Pada landasan teori akan dijelaskan mengenai dasar teori yang akan digunakan pada penelitian ini dan kesimpulan yang diambil dari penelitian sejenis. Landasan teori dapat membantu pembaca memahami istilah atau hal-hal terkait yang digunakan pada penelitian ini.

1.7.3 BAB III METODOLOGI

Metodologi akan membahas gambaran umum alur penelitian. Dalam bab ini, penulis akan memaparkan tentang metode yang digunakan dalam mempersiapkan sistem penelitian, pemodelan, implementasi, dan evaluasi.

1.7.4 BAB IV HASIL DAN PEMBAHASAN

Bab ini menjelaskan tentang performa model yang sudah dilatih dan komparasi antara berbagai model yang telah digunakan. Dalam bab ini diharapkan dapat mengetahui model yang terbaik.

1.7.5 BAB V KESIMPULAN DAN SARAN

Bab ini berisi rangkuman dari seluruh tahapan yang dilalui dalam penelitian ini. Bab ini juga mencantumkan saran yang dapat dimanfaatkan dalam penelitian selanjutnya di masa mendatang.

BAB II
KAJIAN PUSTAKA

2.1 Deep Learning

Bidang studi yang dikenal sebagai ke *Artificial Intelligence* (AI) adalah sub bidang ilmu komputer yang berfokus pada pengembangan metode di mana robot (komputer) dapat melakukan tugas yang secara tradisional dilakukan oleh manusia pada tingkat efisiensi yang setara dengan manusia. Menurut John McCarthy (1956), sebagaimana dikutip dalam jurnal penelitian (Pannu, 2015), tujuan *Artificial Intelligence* (AI) adalah untuk menemukan atau memahami mekanisme yang digunakan manusia untuk berpikir, dan kemudian menciptakan komputer sehingga dapat meniru manusia.

Sistem Pakar, Pemrosesan Bahasa Alami (NLP), Pengenalan Suara (*Speech Recognition*), Visi Komputer, Instruksi Berbantuan Komputer Cerdas, dan sejumlah sub bidang lainnya merupakan bagian dari *Artificial Intelligence* (AI). AI adalah bidang studi yang luas. Upaya untuk meniru pekerjaan seorang pakar adalah sistem pakar. Tujuan dari sistem pakar adalah untuk memfasilitasi transfer pengalaman satu orang pakar ke komputer dan kemudian ke individu lain (orang yang bukan ahli). *Natural Language Processing* (NLP) memungkinkan individu untuk terhubung dengan komputer menggunakan bahasa yang mirip dengan yang digunakan dalam percakapan umum. Pengenalan suara memungkinkan orang dan komputer untuk berinteraksi satu sama lain hanya dengan menggunakan suara mereka. Ungkapan "*Computer Vision*" mengacu pada proses menganalisis gambar atau objek yang ditampilkan di layar komputer. Istilah "*Intelligent Computer-Aided Instructi*on" mengacu pada cara di mana komputer dapat berfungsi sebagai mentor yang dapat menginstruksikan atau menginstruksikan orang lain

Diperlukan algoritma yang efektif agar kita dapat menemukan solusi untuk masalah komputer. Algoritma adalah serangkaian instruksi yang harus dilakukan untuk mengubah input menjadi output untuk menyelesaikan suatu masalah. Tujuan dari algoritma adalah untuk menemukan solusi dari masalah tersebut. Misalnya seseorang dapat merancang suatu algoritma untuk proses penyortiran. Data yang dihasilkan adalah data berurutan, sedangkan data yang diinput adalah barisan bilangan bulat. Mungkin ada beberapa algoritma yang dapat melakukan pekerjaan yang sama, dan mungkin akan tertarik untuk menemukan algoritma yang paling efisien, artinya ia menggunakan jumlah instruksi paling sedikit atau jumlah memori paling

sedikit atau keduanya. Namun, ada keadaan lain di mana tidak memiliki algoritma, seperti menentukan apakah sebuah email termasuk spam atau tidak. Ini adalah salah satu aplikasi untuk *machine learning* atau dalam Bahasa Indonesia pembelajaran mesin.

Pembelajaran mesin adalah sub bidang kecerdasan buatan (AI) yang memberi komputer kemampuan untuk belajar dari data mereka sendiri, tanpa diinstruksikan secara tegas oleh pemrogram (Widodo Budiharto, 2016). Tujuan utama dari pembelajaran mesin adalah untuk membangun sistem komputer yang mampu mendidik diri mereka sendiri bagaimana beradaptasi dan berkembang dalam menanggapi masukan baru. Misalnya, sistem pembelajaran mesin dapat diajarkan untuk membedakan antara komunikasi spam dan non-spam dengan terpapar kedua jenis pesan email selama pelatihan mereka. Setelah dilatih, ia akan dapat menyortir pesan email yang baru diterima ke folder spam dan non-spam yang sesuai. Dengan mengumpulkan ribuan sampel pesan yang kita kenal diklasifikasikan sebagai spam dan dengan melakukan pembelajaran, kita mampu membuat sistem yang mampu menentukan apakah email kita termasuk spam atau tidak. Di bidang kesehatan, algoritma pembelajaran mesin sangat membantu untuk mendiagnosis penyakit. Di bidang telekomunikasi, formulir panggilan dipelajari untuk tujuan mengoptimalkan jaringan dan mencapai tingkat kualitas layanan setinggi mungkin.

Bidang robotika, pengenalan suara, dan visi komputer semuanya mendapat manfaat dari penggunaan pembelajaran mesin. Tujuan dari program komputer yang dikenal sebagai pembelajaran mesin adalah untuk meningkatkan kinerja dengan memanfaatkan data sampel atau pengalaman sebelumnya. Misalnya, dalam hal pengenalan wajah, setiap wajah memiliki bentuk atau pola unik yang terdiri dari kombinasi mata, hidung, dan mulut di tempat-tempat tertentu di wajah. Pengenalan pola adalah apa yang dikenal sebagai saat perangkat lunak memeriksa sampel gambar wajah untuk menentukan bentuk wajah seseorang yang tepat dan kemudian mengidentifikasi wajah itu dengan membandingkannya dengan templat bentuk wajah yang diketahui.

Deep Learning adalah sub bidang pembelajaran mesin yang menggunakan jaringan saraf tiruan untuk memecahkan masalah yang ditimbulkan oleh kumpulan data besar. Sub bidang pembelajaran mesin ini juga dikenal sebagai "*supervised learning*". Pendekatan Deep Learning membuat *Supervised Learning* memiliki arsitektur yang sangat efektif. Dengan menambahkan lapisan tambahan ke model, pembelajaran semacam ini dapat lebih akurat mencerminkan data yang sudah dilabeli. Di bidang *machine learning*, ada strategi yang mencakup ekstraksi fitur dari data pelatihan dan penggunaan algoritma pembelajaran khusus untuk mengkategorikan

gambar dan mendeteksi suara. Namun, masih ada batasan tertentu pada sistem ini, terutama yang berkaitan dengan kecepatan dan presisi.

Model Deep Learning dimaksudkan untuk melewati sejumlah level yang berbeda. Dalam konteks jaringan saraf tiruan, istilah "*Deep Model*" dapat digunakan untuk merujuk *pada Multi-Layer Perceptron* (MLP) yang memiliki lebih dari dua lapisan tersembunyi. *Convolution layer, dropout layer, fully connected layer, pooling layer, dan Relu layer* hanyalah beberapa layer yang sering digunakan. Setelah menerapkan banyak lapisan, data tambahan akan dikumpulkan, dan penting untuk menahan diri agar tidak memasukkan data seragam secara berlebihan. Ada beberapa metode yang dapat digunakan untuk mencegah over-fitting, dua di antaranya yang paling sering digunakan adalah regularisasi dan augmentasi data (Ponti et al., 2017)

Deep Learning adalah gagasan yang dapat diterapkan pada algoritma pembelajaran mesin saat ini, yang memungkinkan komputer belajar dengan kecepatan lebih cepat, lebih akurat, dan dalam skala yang lebih besar. *Deep Learning* menjadi lebih populer di dunia akademik dan dunia bisnis sebagai alat untuk membantu dalam penyelesaian beberapa masalah yang melibatkan sejumlah besar data. Beberapa contoh masalah ini termasuk visi komputer, pengenalan suara, dan pemrosesan bahasa alami. Salah satu aspek terpenting dari pembelajaran mendalam disebut "Rekayasa Fitur," dan tujuannya adalah untuk mengidentifikasi pola yang relevan dari data untuk mempermudah model membedakan antara berbagai jenis informasi.

Di masa depan, visi komputer akan menjadi teknologi paling signifikan untuk penciptaan robot yang berinteraksi. *Computer Vision* adalah sub bidang ilmu komputer yang berfokus pada sistem kecerdasan buatan dan penangkapan serta pemrosesan gambar. Contoh aplikasi *computer vision* dalam industri dan penelitian antara lain pengendalian proses industri, pendeteksian plat nomor kendaraan, pembuatan model 3D (fotogrametri), *Robot Vision*, *Robot Humanoid*, dan *Robot Soccer Surveillance* (pemantau penyusup, analisis lalu lintas jalan tol, dan lain-lain), Pemodelan objek atau lingkungan interaksi manusia-robot (*Human Robot Interaction*) (Widodo Budiharto, 2016).

2.2 Deteksi Objek

Pada tahun 1980 hingga 1990, perkembangan *semiconductor* mengarah pada peningkatan kekuatan komputasi komputer. Peneliti ingin memanfaatkan kekuatan komputasi ini untuk menjalankan program *machine learning* agar dapat menyelesaikan masalah dunia nyata dibanding sekedar menjalankan program pembuktian konsep (Scarpino, 2018).

Machine learning adalah sebuah aplikasi untuk menemukan pola pada data yang sangat besar dan bervariasi. Tidak seperti pemrograman pada umumnya, algoritma *machine learning* tidak dibuat berdasarkan instruksi programmer tetapi mencari sendiri berdasarkan data yang diberikan tanpa menerima instruksi secara rinci, proses mencari informasi atau pola ini berarti komputer sedang melakukan proses pembelajaran. Algoritma *machine learning* memiliki kemampuan untuk menghadapi ketidakpastian dan peluang (Scarpino, 2018).

Machine learning konvensional terbatas pada kemampuan mereka untuk memproses data pada bentuk aslinya. Sebelumnya, untuk membuat pengenalan pola pada *machine learning* membutuhkan kesulitan dan keahlian tinggi untuk bisa mendesain pengekstrak fitur yang merubah data mentah menjadi data yang cocok untuk melakukan identifikasi pola pada input. Metode *deep learning* memungkinkan komputer untuk diberi data mentah dan mampu merubah data tersebut menjadi cocok untuk melakukan deteksi atau klasifikasi (Lecun et al., 2015).

Neural network pada umumnya hanya menerima serangkaian input, mengalikan setiap input tersebut dengan bobot, dan meneruskan data tersebut melalui serangkaian *layer*. Hal ini cukup untuk menganalisis data secara umum namun tidak untuk memproses gambar dan 2D/3D data. Klasifikasi gambar memerlukan konvolusi, maka neural network yang ditujukan untuk mendeteksi gambar disebut *Convolutional Neural Network* (CNN). Setelah dari *layer* konvolusi, CNN menggunakan *fully connected layer* untuk menghasilkan *output*.

Object detection menggunakan CNN untuk mendeteksi berbagai macam object yang terdapat pada gambar digital seperti mobil, manusia, atau binatang. Tujuan dari *object detection* sendiri adalah untuk mengembangkan model dan teknik komputasi yang menyediakan informasi sederhana seperti objek apa dan di mana lokasinya. Dari sudut pandang penggunaannya, *object detection* dapat dikelompokan menjadi dua kelompok yaitu "*general object detection*" dan "*detection applications*", di mana yang satu ditujukan untuk mengeksplorasi metode untuk mendeteksi berbagai tipe objek di bawah kerangka kerja terpadu untuk mensimulasikan penglihatan manusia. Kelompok kedua ditujukan untuk pengaplikasian pada scenario khusus seperti deteksi wajah, deteksi tulisan, deteksi pejalan kaki, dan lain sebagainya. *Object detection* saat ini sudah diaplikasikan pada banyak aplikasi dunia nyata (Zou et al., 2019).

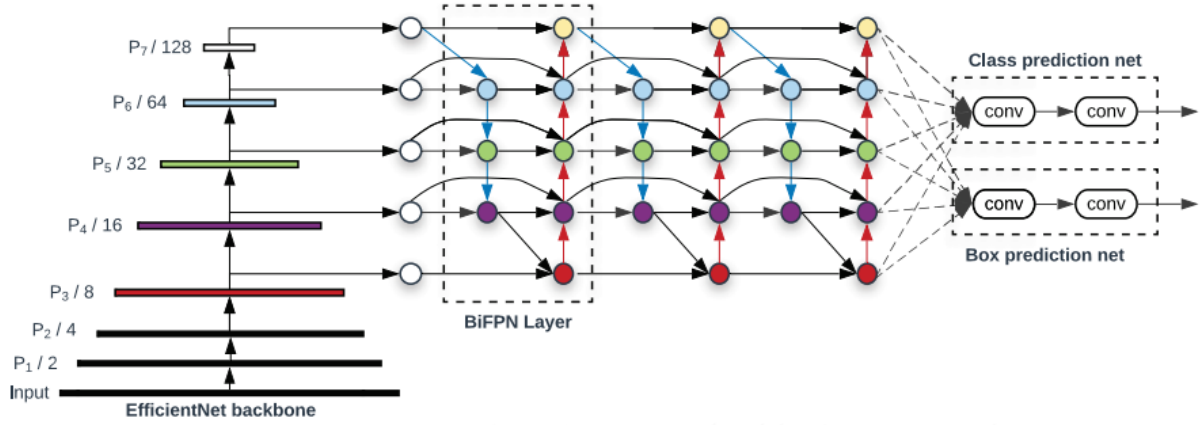
2.3 TensorFlow

Tensorflow adalah pustaka *opensource* yang dibuat oleh Google untuk melakukan pekerjaan seputar *deep learning*. Tensorflow menggunakan grafik aliran data komputasi untuk

merepresentasikan arsitektur jaringan saraf tiruan yang rumit. Perlahan-lahan Tensorflow menjadi library yang sering digunakan untuk penelitian seputar *deep learning* dan implementasi pada proyek tertentu serta digunakan untuk proyek yang membutuhkan implementasi di *cloud* (Pattanayak, 2017).

2.4 *Efficient-Det*

Efficient-Det merupakan salah satu model yang dapat digunakan untuk melatih sistem. Model ini didasarkan pada jaringan saraf tiruan yang menggunakan algoritma *Convolutional Neural Network* (CNN). Model ini dapat digunakan untuk mendeteksi objek menggunakan sekumpulan data berupa gambar. Dalam perkembangannya, model ini dibuat untuk meningkatkan efisiensi pelatihan dengan menggunakan sejumlah kecil parameter, namun tetap mencapai akurasi yang baik (Tan et al., 2020). Gambar 2.1 adalah arsitektur dari *Efficient-Det.* Model ini menggunakan metode deteksi satu tahap, yaitu melakukan deteksi pada seluruh area gambar sehingga prosesnya berjalan dengan cepat.



Gambar 2.1 Arsitektur *Efficient-Det*

Sumber: (Tan et al., 2020)

2.5 *EfficientDet-Lite*

EfficientDet-Lite sendiri merupakan sebuah versi *EfficientDet* yang sudah dioptimalkan untuk perangkat bergerak dengan menggunakan ukuran input yang lebih kecil dan fungsi aktivasi yang lebih sederhana tetapi masih memiliki arsitektur dan pemrosesan yang sama dengan Efficient-Det (Repak, 2021). Berdasarkan dokumentasi *TensorFlow Lite Model Maker* dan penelitian Tomas Repak, terdapat lima model yang dapat dipilih seperti yang terdapat pada Tabel 2.1. Semakin besar model yang dipilih maka semakin dalam jumlah lapisan model

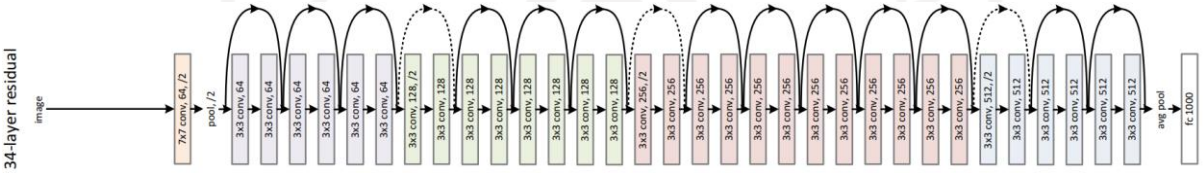
tersebut yang membuat nilai *Average Precision* semakin tinggi tetapi membuat waktu inferensi semakin lambat.

Tabel 2.1 Pilihan model *Tensorflow Lite Model Maker*

Model	**Input *size***	***BiFPN layers***	***ClassNet* dan *BoxNet layers***	***Size (MB)***	***Latency (ms)***	***Average Precision***
EfficientDet-Lite0	320	3	3	4.4	37	25.69%
EfficientDet-Lite1	384	4	3	5.8	49	30.55%
EfficientDet-Lite2	448	5	3	7.2	69	33.97%
EfficientDet-Lite3	512	6	4	11.4	116	37.70%
EfficientDet-Lite4	640	7	4	19.9	260	41.96%

2.6 *Residual Network*

Residual Network atau yang biasa disebut *ResNet* adalah sebuah model yang dibuat untuk mengatasi kesulitan dalam melatih *neural network* dengan arsitektur yang dalam. *ResNet* bahkan memiliki lapisan yang lebih mendalam daripada *neural network* yang sudah ada sebelumnya. Arsitektur mendalam dari *ResNet* dapat dilihat pada Gambar 2.2. *ResNet* menggunakan fungsi yang mereferensikan suatu *layer* ke *layer* input. *ResNet* telah terbukti yang menunjukan bahwa arsitektur ini lebih mudah dioptimalisasi serta bisa mendapat akurasi dari lapisan yang lebih dalam. *ResNet* memungkinkan untuk melewati suatu lapisan tanpa mempengaruhi performa model (He et al., 2015).



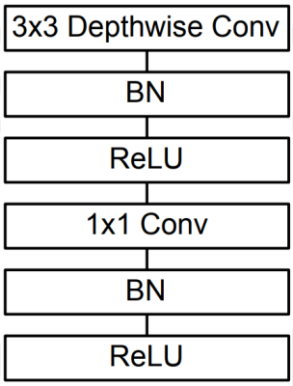
Gambar 2.2 Arsitektur *Residual Network*

Sumber: (He et al., 2015)

2.7 *MobileNet*

MobileNet adalah model yang bertujuan untuk meningkatkan efisiensi untuk perangkat bergerak maupun perangkat pengenalan citra lainnya. *MobileNet* berdasar pada arsitektur baru yang efisien bernama *depthwise separable convolutions* untuk membuat *neural network* yang ringan. Gambar 2.3 merupakan lapisan konvolusi pada *MobileNet* dengan menggunakan

depthwise separable convolutions. Lapisan konvolusi ini memisahkan tiga dimensi input lalu mengaplikasikan satu konvolusi filter pada setiap *input channel* yang sudah dipisahkan. Luaran pada setiap *channel* ini akan digabung kembali menjadi *tensor* tiga dimensi. Arsitektur ini membuat jumlah parameter berkurang drastis. Meski begitu, model ini tetap memiliki akurasi yang dapat bersaing dengan model popular lainnya (Howard et al., 2017).



Gambar 2.3 Arsitektur *MobileNet*

Sumber: (Howard et al., 2017)

MobileNet memiliki versi yang lebih baru yaitu *MobileNetV2* yang lebih mengedepankan efesiensi untuk penggunaan pada perangkat bergerak dengan cara mengurangi jumlah parameter sebanyak satu juta parameter. Setiap baris pada Gambar 2.4 merupkaan lapisan dari *MobileNetV2.* Arsitektur ini memiliki 11 lapisan yang terdiri dari lapisan konvolusi, *bottleneck* yang berfungsi untuk mengurangi parameter, dan *pooling* (Sandler et al., 2019).

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Gambar 2.4 Lapisan *MobileNetV2*

Sumber: (Sandler et al., 2019)

2.8 Matrik Evaluasi

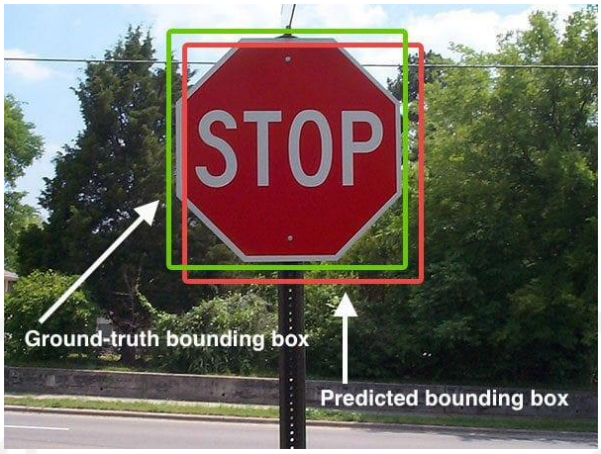
Parameter hasil pengukuran matriks evaluasi dapat digunakan untuk menilai seberapa baik kinerja model deteksi objek. Matriks penilaian yang dikenal sebagai "*average precision*" adalah salah satu yang sering digunakan dalam proses identifikasi objek. Menghitung nilai presisi dan *recall* penting dilakukan untuk dapat memperoleh nilai *Average Precision* (AP) (Tan et al., 2020). Dengan kata lain, matriks AP mengevaluasi seberapa baik model yang diperoleh dalam mengantisipasi data aktual berdasarkan data yang belum pernah dilihat sebelumnya. Rumus perhitungan presisi dapat ditemukan pada persamaan (2.1). Presisi adalah perbandingan model mendeteksi sebuah objek dengan benar dibanding keseluruhan deteksi pada objek tersebut. Perhitungan recall dapat ditemukan pada persamaan (2.2). Recall adalah perbandingan model mendeteksi sebuah objek dengan benar dibanding keseluruhan objek yang ada pada gambar tersebut. Walaupun disebut rata-rata presisi, AP tidak hanya dihitung berdasarkan hal tersebut. AP dihitung berdasarkan nilai rata-rata presisi dari keseluruhan nilai recall. Menggunakan kedua nilai tersebut rumus untuk AP dapat ditemukan dalam persamaan (2.3).

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{2.1}$$

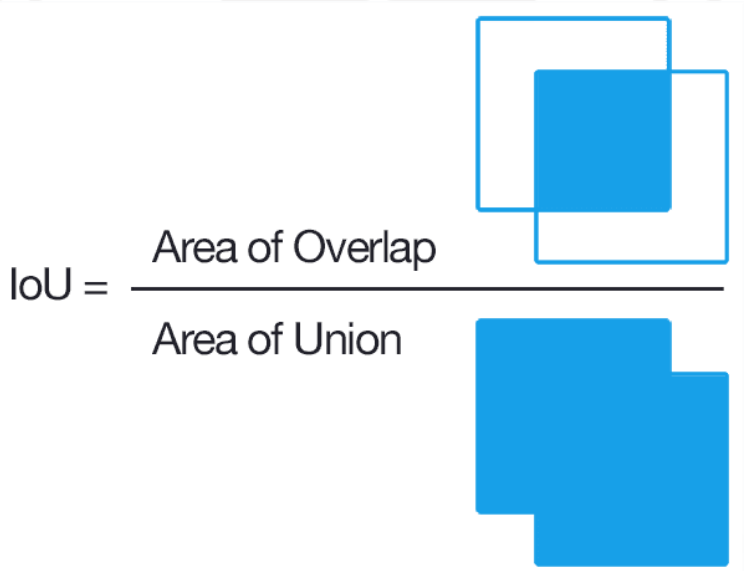
$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{2.2}$$

$$AP = \frac{1}{11}\sum_{Recall_{i-11}} Precision(Recall_i) \tag{2.3}$$

Dalam hal deteksi objek, nilai *True Positive* (TP), *False Positive* (FP), dan *False Negative* (FN) semuanya menggunakan nilai ambang batas yang ditentukan oleh *intersection of union* (IoU). Nilai IoU adalah nilai yang ditemukan pada perpotongan antara kotak prediksi dan kotak ground truth. *Ground truth* merujuk informasi yang benar tentang lokasi dari kotak pada suatu objek. Gambar 2.5 memberikan ilustrasi kotak prediksi dan *ground truth*. Salah satu dari tujuan membuat model *object detection* adalah membuat kotak prediksi (kotak merah) sedekat mungkin dengan *ground truth* (kotak hijau). sedangkan Gambar 2.6 menyajikan perhitungan IoU. Seberapa dekat kotak prediksi kepada *ground truth* dapat diukur menggunakan satuan IoU. Skor IoU diukur dengan rentang nol sampai satu.



Gambar 2.5 Ilustrasi kotak prediksi dan kotak *ground truth*
Sumber: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/



Gambar 2.6 Ilustrasi *Intersection over Union* (IoU)
Sumber: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

Nilai suatu deteksi hanya dianggap *True Positive* (TP) jika deteksi tersebut akurat dalam menentukan adanya objek dengan klasifikasi yang sesuai dan nilai IoU lebih besar dari ambang batas yang telah ditentukan. Salah satu ambang batas tersebut dapat berupa nilai IoU lebih besar dari atau sama dengan 0,5. Istilah "*False Positive*" (FP) mengacu pada situasi di mana hasil deteksi objek tidak akurat, tidak sesuai dengan klasifikasi, dan nilai IoU lebih rendah dari nilai ambang batas. Hasil deteksi dikatakan *False Negative* atau FN, jika objek yang dicari

tidak ditemukan sama sekali. Untuk mendapatkan gambaran yang lebih baik tentang nilai IoU, terdapat ilustrasi pada Gambar 2.7 yang menampilkan IoU yang kurang, baik, dan sempurna. Kotak *poor* dinyatakan sebagai *False Positive* sedangkan *good* dan *excellent* dinyatakan sebagai *True Positive* walaupun skor IoU *good* lebih rendah.

Gambar 2.7 Ilustrasi nilai IoU

Sumber: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

2.9 *Transfer Learning*

Transfer learning dalam arti luas berarti menyimpan pengetahuan yang sudah didapat ketika memecahkan masalah dan menggunakan pengetahuan tersebut untuk memecahkan masalah yang berbeda tetapi dalam satu bidang yang sama. *Transfer learning* sudah sangat sukses di bidang *deep learning* karena *deep learning* memiliki parameter yang sangat besar dan *pre-trained* model pada *transfer learning* memiliki parameter yang dapat diandalkan (Pattanayak, 2017).

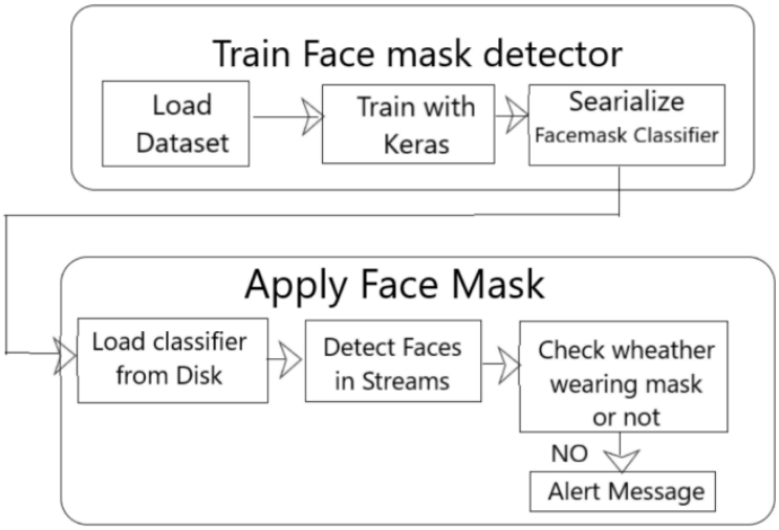
2.10 Penelitian Mengenai *Object Detection*

Sudah ada beberapa penelitian pendeteksi masker yang sudah dilakukan sebelumnya dengan menggunakan berbagai macam metode. Walaupun topik yang diteliti sama, dari kelima penelitian yang dianalisis semuanya memiliki perbedaan seperti penggunaan dataset, pemilihan metode atau *pre-trained* model, dan hasil yang didapatkan.

Salah satunya adalah penelitian yang dilakukan oleh Suressh, K dan kawan-kawan yang membuat model deteksi masker *real time* menggunakan *Convolution Neural Network* (CNN) yang merupakan kelas dari *Deep Neural Network (*DNN). Model yang dibuat dapat diimplementasikan pada kamera pengawas sekolah, universitas, pusat perbelanjaan, dan lain-

lain. Penelitian ini memonitor seseorang dan mendeteksi apakah mereka menggunakan masker, jika tidak akan dilaporkan kepada aparat terkait melalui pesan teks. Diharapkan model ini dapat menurunkan tingkat penyebaran virus.

Penelitian ini menggunakan arsitektur sistem *Optimistic Convolution Network* untuk memastikan masyarakat menggunakan masker atau tidak. Sistem ini menggunakan TensorFlow dan *Keras* bersamaan dengan model CNN dan *MobileNet*. Foto pada dataset dimuat menggunakan *Keras* lalu diubah menjadi *array*, lalu *MobileNet* memproses dataset tersebut dan menambahkannya ke list lalu dideteksi secara *real*-time menggunakan *MobileNet* dan *OpenCV*.

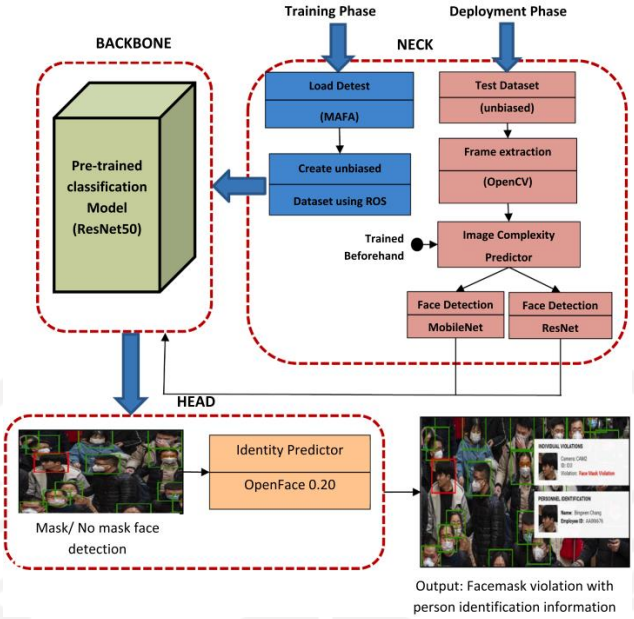


Gambar 2.8 Arsitektur sistem pada penelitian "*Face Mask Detection by using Optimistic Convolutional Neural Network*"

Sumber: (Suresh et al., 2021)

Dataset yang digunakan mengandung 3918 foto yang dibagi menjadi dua *kategori* yaitu dengan masker dan tanpa masker. Dataset yang digunakan adalah *"LFW Simulated Masked Face Dataset*" yang tersedia di Kaggle , wajah tanpa masker berisi variasi wajah dengan berbagai warna kulit, sudut pengambilan gambar, dan *occlusion*. Foto wajah yang menggunakan masker berisi wajah yang ditutupi tangan, dengan masker, dan objek lain yang menutupi wajah (Suresh et al., 2021).

Adapun penelitian lain yang menggunakan model dasar *ResNet50, AlexNet,* dan *MobileNet*. Arsitektur ini menggunakan tiga komponen utama, yaitu *Backbone*, *Neck*, dan *Head*.



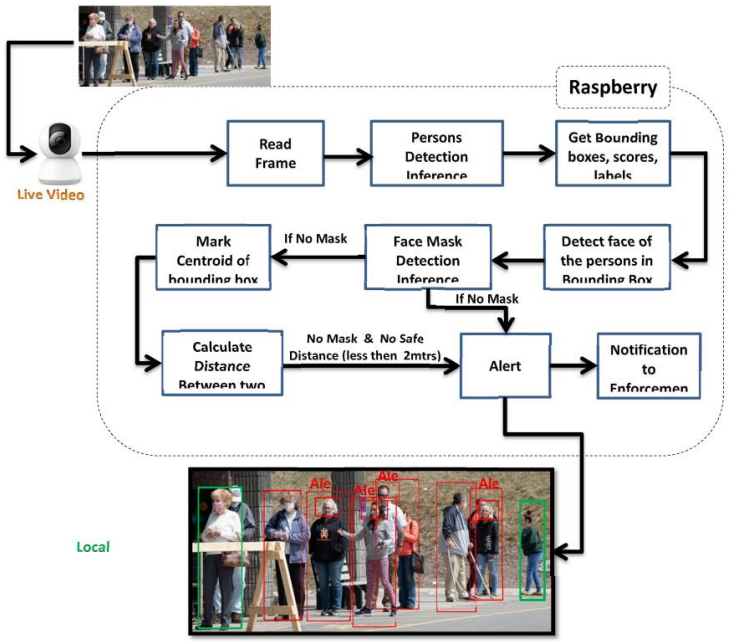
Gambar 2.9 Arsitektur penelitian "*Face mask detection using deep learning: An approach to reduce risk of Coronavirus spread*"
Sumber: (Sethi et al., 2021)

Backbone berdasar pada CNN yang mampu mengekstrak informasi dari foto dan mengubahnya menjadi peta fitur (*feature map*). *Backbone* ini menggunakan metode *transfer learning* agar dapat menggunakan atribut yang sudah dipelajari dari dataset yang besar sehingga dapat mengekstrak fitur baru untuk model ini. Diantara ketiga *pre-trained model* popular yang bernama *ResNet50, MobileNet*, dan *AlexNet*. ResNet50 dipilih karena karena model tersebut yang paling optimal untuk model ini.

Keunikan pada penelitian ini terdapat pada komponen *Neck*. Komponen ini berisi semua proses *preprocessing* yang dibutuhkan untuk klasifikasi foto. Komponen ini juga memiliki dua alur kerja, untuk *train* menggunakan dataset MAFA yang berisi 25,876 foto dan *fine-tuning* dari *ResNet50* sedangkan untuk *deployment* berisi video *real-time* yang dipecah menjadi *frame* lalu mengekstrak fitur wajah dan mengidentifikasinya. Dua dataset ini digunakan untuk menghindari bias pada salah satu kelas karena pada penelitian ini, dataset MAFA berisi 23,858 yang menggunakan masker sedangkan hanya 2018 foto tanpa masker.

Komponen terakhir adalah *head*, pendeteksi identitas menggunakan *OpenFace 2.0* yang diharapkan untuk memenuhi tujuan *deep learning neural network* ini, yaitu penegakan penggunaan masker wajah di tempat umum dengan cara mengambil identitas wajah pelanggar (Sethi et al., 2021).

Pada penelitian yang dilakukan oleh Yadav, S. Sistem yang dibuat ditujukan untuk memastikan kesehatan masyarakat yang berada di area publik dengan cara memantau jarak aman antara orang dan mendeteksi individu yang memakai atau tidak memakai masker.



Gambar 2.10 Arsitektur penelitian "*Deep Learning based Safe Social Distancing and Face Mask Detection in Public Areas for COVID-19 Safety Guidelines Adherence*"
Sumber: (Yadav, 2020)

Sistem ini menggunakan metode *transfer learning* untuk melakukan pengoptimalan kinerja dengan menggunakan *deep learning* dan *computer vision* untuk memantau individu yang berada pada ruang publik dengan kamera yang diintegrasikan pada Raspberry Pi 4 dan mendeteksi apakah individu tersebut menggunakan masker atau tidak. Sistem ini menggunakan arsitektur *MobileNetV2* dan *SSD MultiBox* sebagai model inti pendeteksi. Sistem ini mendapat 91.2% mAP (Yadav, 2020).

Pada penelitian yang dilakukan oleh Rahman, sistem yang akan dibuat untuk mendeteksi orang yang tidak memakai masker melalui kamera CCTV. Kamera tersebut digunakan untuk mengambil gambar dari ruang publik lalu mendeteksi apakah orang tersebut menggunakan masker atau tidak, jika tidak maka informasi tersebut akan diteruskan ke pihak berwajib.

Gambar yang diambil dari CCTV memerlukan *preprocessing* sebelum dapat digunakan. Gambar diubah menjadi grayscale karena gambar berwarna mengandung informasi tidak perlu

untuk melakukan deteksi masker, lalu mengubah ukuran foto menjadi 64x64 dan dinormalisasi. Untuk dataset yang digunakan berasal dari Kaggle dan Github dengan total 858 gambar untuk orang mengenakan masker dan 681 orang tidak mengenakan masker dengan proporsi 80% digunakan untuk *training* dan sisanya untuk *testing.*

Proses *training* bergantung pada arsitektur CNN yang mana akan sangat berguna untuk mendeteksi pola pada sebuah gambar (Khan et al., 2020). Arsitektur jaringan ini berisi *input layer*, beberapa *hidden layer*, dan *output layer*. *Hidden layer* berisi dari beberapa *convolution layer* yang berfungsi untuk mengekstrak fitur tertentu pada sebuah sampel. Arsitektur yang digunakan pada penelitian ini dapat dilihat pada Gambar 2.11. Proses *training* pada penelitian ini berjalan selama 100 epoch dan mendapat akurasi 98.7% pada data *train* dan 98.5% pada data test (Rahman et al., 2020).

Layer	Type	Kernel	Kernel Size	Output Size
1	Convolution2D	32	(3×3)	(62×62×32)
2	Convolution2D	32	(3×3)	(60×60×32)
3	MaxPooling2D	-	(2×2)	(30×30×32)
4	Convolution2D	32	(3×3)	(28×28×32)
5	Convolution2D	32	(3×3)	(26×26×32)
6	MaxPooling2D	-	(2×2)	(13×13×32)
7	Convolution2D	32	(3×3)	(11×11×32)
8	Convolution2D	32	(3×3)	(9×9×32)
9	MaxPooling2D	-	(2×2)	(4×4×32)
10	Flatten	-	-	512
11	Dense	-	-	100
12	Dropout	-	-	100
13	Dense	-	-	30
14	Dropout	-	-	30
15	Dense	-	-	10
16	Dropout	-	-	10
17	Dense	-	-	2

Gambar 2.11 arsitektur penelitian "*An automated system to limit COVID-19 using facial mask detection in smart city network*"
Sumber: (Rahman et al., 2020)

Penelitian yang dilakukan oleh Sanjaya & Rakhmawan membahas latar belakang masalah yang sama seperti penelitian-penelitian sebelumnya yaitu mencegah penularan COVID-19 dengan cara mendeteksi orang yang tidak menggunakan masker di tempat umum. Penelitian ini menggunakan metode CNN yang berdasar pada model *MobileNetV2* yang dikembangkan oleh Google (Sandler et al., 2018.). Dataset yang digunakan untuk penelitian ini berasal dari dua dataset yang berbeda. Dataset pertama berasal dari Kaggle dataset *Real-World Masked Face* (RMFD) yang akan digunakan untuk *training*, *validation*, dan *testing*

sedangkan dataset kedua berasal dari CCTV dari 25 kota yang berada di Indonesia. Dataset kedua digunakan untuk implementasi model. Dalam membuat model ini, sebanyak 1916 gambar orang mengenakan masker dan 1930 gambar orang tidak mengenakan masker dengan proporsi 75% digunakan untuk *training* dan 25% digunakan untuk *testing*. Proses *training* berjalan selama 20 *epoch* dengan memiliki hasil 96.85% akurasi untuk data *train* dan 91% untuk data test (Sanjaya & Rakhmawan, 2020).

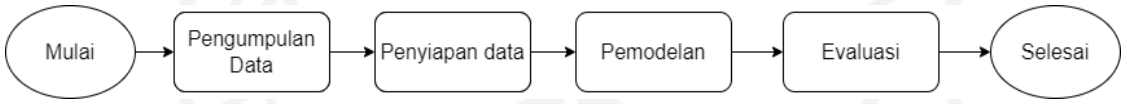
Tabel 2.2 Tabel perbedaan penelitian sejenis

Metode	**Hasil evaluasi**	**Sumber dataset**
Resnet50, Support Vector Machine (SVM), decision tree, and ensemble algorithm	*Accuracy* = ~100%	Kaggle
Transfer Learning (resnet50), open face 0.2	*Precision* = 98.86% *Recall* = 98.22%	MAFA
Transfer learning (mobilenet v2), fine tuning	*precision* = 91.7% *confidence* = 70% *recall* = 91 %	dataset pribadi berisi 3165 gambar
CNN	*Accuracy* = 98.7%	Kaggle dan Github
Transfer learning (mobilenet v2)	Masker: *Precision* = 98% *Recall* = 83% Tanpa Masker: *Precision* = 85% *Recall* = 98%	*Real-World Masked Face dataset* (RMFD), CCTV dari 25 kota di Indonesia

Tabel 2.2 menampilkan perbedaan dengan penelitian yang sebelumnya yang hanya memiliki dua label yaitu memakai masker dan tidak memakai masker. Penelitian ini menggunakan lima label yaitu tidak memakai masker, kurang tepat dalam memakai masker, dan menggunakan masker beserta jenis masker yang digunakan seperti masker medis, masker kain, dan masker scuba. Selain itu, dua arsitektur model yang digunakan pada penelitian ini belum digunakan pada penelitian terdahulu.

BAB III
METODOLOGI PENELITIAN

Pada bab tiga, peneliti akan menjelaskan alur penelitian ini seperti yang tertera pada Gambar 3.1. Metodologi penelitian ini menjelaskan tahapan-tahapan yang akan dilalui peneliti agar mencapai hasil yang diharapkan. Alur ini menjelaskan setiap tahapan yang peneliti lakukan mulai dari proses pencarian dataset hingga melakukan evaluasi.



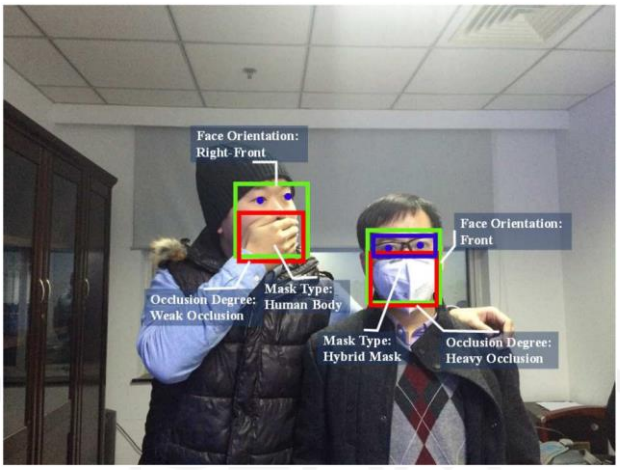
Gambar 3.1 Bagan alir metode penelitian

3.1 Pengumpulan Data

Dataset adalah sekumpulan data yang saling berkaitan satu dengan yang lainnya (Snijders et al., 2012). Dataset yang akan digunakan nantinya dapat mempengaruhi akurasi akhir model *machine learning*. Ada beberapa macam dataset yang digunakan pada penelitian ini, diantaranya adalah:

3.1.1 *Masked Face* (MAFA)

Model pendeteksi wajah sering kali mengalami penurunan akurasi ketika mendeteksi wajah yang menggunakan masker, maka *Masked Face* (MAFA) dataset dibuat untuk meningkatkan efektivitas dan efisiensi dalam pembuatan model untuk mendeteksi wajah yang menggunakan masker. Dataset ini diambil dari internet seperti Flickr, Google, dan Bing dengan total berisi 30,811 gambar di mana setiap gambarnya memiliki setidaknya satu wajah yang menggunakan masker. Dataset ini memiliki enam label yang dilabeli secara manual yaitu lokasi wajah, lokasi mata, lokasi masker, orientasi wajah, tingkat wajah yang ketutupan objek, dan jenis masker. Setelah diberi label, dataset ini memiliki 35,806 wajah yang menggunakan masker (Ge et al., 2017).



Gambar 3.2 Contoh label Dataset MAFA

Sumber: (Ge et al., 2017)

3.1.2 *Flickr-Faces-HQ*

Dataset ini merupakan koleksi yang terdiri dari 70.000 gambar wajah tanpa masker berkualitas tinggi yang diambil dari website Flicker lalu disejajarkan dan dipotong dengan wajah yang berada di tengah foto. Masing-masing foto yang ada di dataset ini memiliki resolusi 1024x1024 *pixels*. Dibandingkan dengan dataset yang sudah ada sebelumnya, dataset ini memiliki variasi yang luas meliputi umur, etnis, sudut pandang, pencahayaan, dan latar belakang foto serta memiliki foto wajah yang mengenakan aksesoris kacamata, topi, dan lainnya (Karras NVIDIA & Laine NVIDIA, 2019).

Gambar 3.3 Contoh gambar dataset *Flickr-Faces-HQ*

Sumber: (Karras NVIDIA & Laine NVIDIA, 2019)

3.1.3 *Face Mask Detection Dataset*

Dataset ini tersedia untuk umum yang tersedia di Kaggle. Dataset ini memiliki 853 foto yang terbagi menjadi tiga kelas, yaitu dengan masker, tanpa masker, dan tidak mengenakan masker dengan benar (Larxel, 2020).

3.1.4 MaskedFace-Net

MaskedFace-Net adalah sebuah dataset augmentasi yang berisi wajah manusia mengenakan masker yang bersumber dari dataset *Flickr-Faces-HQ*. Dataset ini terbagi menjadi dua kategori yaitu masker dikenakan dengan benar atau tidak mengenakan masker dengan benar (Cabani et al., 2021). Pada penelitian ini, kategori yang digunakan hanya dari tidak mengenakan masker dengan benar.

Gambar 3.4 Contoh gambar dataset MaskedFace-Net
Sumber: (Cabani et al., 2021)

3.1.5 Instagram *Scraping*

Scraping adalah sebuah metode untuk mengekstrak data dari sebuah website dan menyimpan data tersebut pada komputer pribadi (De & Sirisuriya, 2015). Peneliti melakukan *scraping* pada Instagram dengan bantuan *library Instaloader* untuk memudahkan pengunduhan gambar karena adanya otomasi yang disediakan oleh *library* tersebut.

Pada *scraping* Instagram, teknik pengumpulan data yang dilakukan adalah dengan mencari gambar yang ingin dicari dengan *hashtag* tertentu lalu postingan yang mengandung kelas yang dicari akan disimpan dengan fitur *saved* pada Instagram. *Library Instaloader* akan mengunduh semua foto yang telah disimpan dengan menggunakan perintah seperti pada Gambar 3.5. Penjelasan pada *flag* yang digunakan adalah *"login"* untuk masuk ke akun

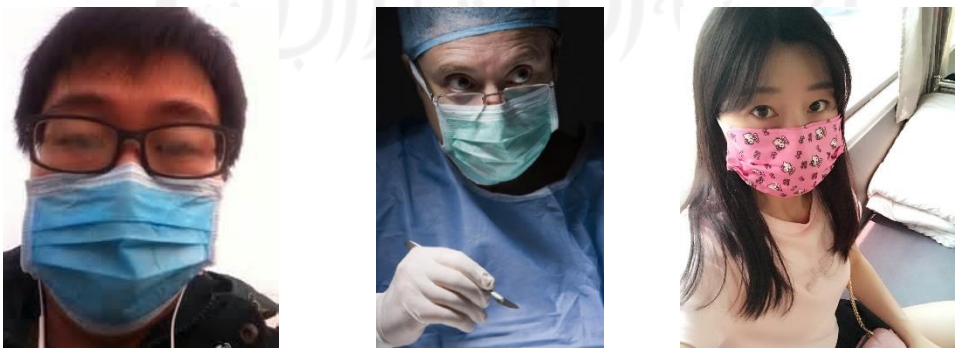
Instagram peneliti sehingga *library* ini bisa mengakses Instagram *post* yang sudah disimpan, *"no-videos"* agar tidak mengunduh video, *"no-metadata-json"* agar tidak mengunduh *metadata*, *"no-video-thumbnails"* agar tidak mengunduh keluku, *"no-captions"* agar tidak mengunduh keterangan gambar, serta *"saved"* untuk mengunduh setiap *post* yang ada pada laman *saved post*.

```
    instaloader --login=verti.240521 --no-videos --no-metadata-
json --no-video-thumbnails --no-captions :saved
```

Gambar 3.5 *Command* untuk mengunduh gambar Instagram

3.2 Penyiapan Data

Langkah pertama yang dilakukan adalah mencari, mengumpulkan, dan memilah dataset yang didapat dari empat dataset *open-source* yang tersedia di internet seperti MAFA, Kaggle, *MaskedFace-Net*, *Flickr-Faces-HQ Dataset* (FFHQ) dan mencari dataset sendiri dengan cara scraping dari Instagram. Tidak semua foto pada dataset tersebut akan digunakan karena banyaknya jumlah data dari gabungan semua dataset publik tersebut. Kumpulan foto tersebut akan dicari berdasarkan sudut, bentuk, dan warna agar bervariasi jika terdapat alternatif lain yang memungkinkan. Misalnya sudah terdapat 20 masker medis berwarna biru maka masker medis warna biru berhenti dicari lalu dilanjutkan mencari masker medis warna lain seperti warna hijau. Warna masker medis biru dan hijau dapat dilihat pada Gambar 3.6 di mana gambar kiri adalah masker warna biru dan tengah adalah masker warna hijau. Selain masker warna biru dan hijau, pada dataset ini juga terdapat masker yang lebih jarang ditemui. Salah satu contohnya adalah masker bermotif dan berwarna merah muda seperti pada bagian kanan Gambar 3.6.



Gambar 3.6 Warna masker medis

Untuk variasi bentuk, misalnya pada masker kain yang memiliki lipatan di depannya, maka dicari masker kain yang tidak memiliki lipatan seperti pada Gambar 3.7 bagian kiri di mana perempuan pada kiri foto menggunakan masker kain yang memiliki lipatan sedangkan perempuan sebelahnya menggunakan masker kain tanpa lipatan. Sedangkan untuk warna dari masker kain sendiri memiliki berbagai macam warna dan motif seperti yang ditunjukan pada Gambar 3.7 yang memiliki motif kotak-kotak, motif militer, dan motif polos. Motif polos sendiri memiliki warna yang berbeda seperti warna merah muda dan hitam.

Gambar 3.7 Bentuk dan warna masker kain

Contoh lain jika sudut pengambilan gambar selalu lurus di depan wajah, maka diprioritaskan mencari gambar dengan sudut pengambilan dari samping seperti pada Gambar 3.8 di mana gambar di tengah merupakan pengambilan lurus di depan wajah, kiri pengambilan dari samping kanan, dan kanan merupakan pengambilan dari sudut atas kepala. Untuk kelas yang sulit ditemukan variasi, jika ditemukan gambar yang tidak memiliki variasi yang sama, maka gambar tersebut diprioritaskan untuk dimasukan ke dalam dataset.

Gambar 3.8 Variasi sudut

Contoh gambar untuk setiap kelas dapat dilihat pada Tabel 3.1. Masker medis adalah masker yang biasa digunakan di bidang kesehatan seperti digunakan oleh petugas kesehatan di rumah sakit. Masker kain adalah masker yang berbahan dasar kain atau tekstil. Masker scuba adalah masker yang terbuat dari kain *scuba* atau *neoprene*. Penggunaan masker salah adalah

ketika seseorang mengenakan masker tetapi tidak menutupi hidung. Tanpa masker adalah ketika seseorang tidak mengenakan masker sama sekali.

Tabel 3.1 Nama kelas dan gambar objek

No	Nama Kelas	Gambar	Sumber
1	Masker medis		MAFA
2	Masker kain		MAFA
3	Masker scuba		MAFA
4	Penggunaan masker salah		MAFA
5	Tanpa masker		Flickr-Faces-HQ Dataset (FFHQ)

Setelah data terkumpul, langkah selanjutnya adalah pemberian label. Jumlah gambar dari gabungan keempat dataset serta *scraping* untuk data *train* sebanyak 1273 foto, sedangkan untuk data validasi berisi 152 foto dan data test berisi 173 foto. Keseluruhan foto ini terbagi menjadi lima kelas, yaitu masker medis, masker kain, masker scuba, penggunaan masker salah, dan tanpa masker. Adapun jumlah label untuk masing-masing kelas pada setiap dataset dapat dilihat pada Tabel 3.2. Jumlah label yang digunakan sebanyak 1638 pada data *train*, 166 pada data validasi, 183 pada data tes. Jumlah label pada dataset validasi dan *test* berkisar 10% dari jumlah label pada dataset *train.*

Tabel 3.2 Jumlah label per kelas pada setiap dataset

Kelas	***Train***	**Validasi**	***Test***
Masker medis	336	31	37
Masker kain	303	34	36
Masker scuba	314	31	35
Penggunaan masker salah	313	32	35
Tanpa masker	372	38	40

Pada setiap dataset tersebut juga dapat dilihat sumber data yang digunakan. Jumlah gambar untuk setiap dataset dapat dilihat pada tabel Tabel 3.3. MaskedFace-Net memiliki jumlah gambar 318 dan Flickr-Faces-HQ memiliki jumlah gambar 209, kedua dataset tersebut hanya memuat gambar untuk kelas tertentu seperti MaskedFace-Net hanya memuat gambar untuk kelas penggunaan masker salah dan Flickr-Faces-HQ hanya memuat gambar untuk kelas tanpa masker. Instagram scraping memiliki jumlah data terbanyak yaitu 678 gambar karna gambar untuk kelas masker kain dan masker scuba lebih mudah ditemukan pada sumber tersebut. Face Mask Detection Dataset memiliki sebanyak 229 gambar karena dataset tersebut memiliki karakteristik mempunyai lebih dari satu label pada gambarnya sehingga data yang digunakan semakin bervariasi. Dataset MAFA hanya memiliki 164 gambar karena kesulitan untuk mencari kelas yang sesuai pada dataset tersebut.

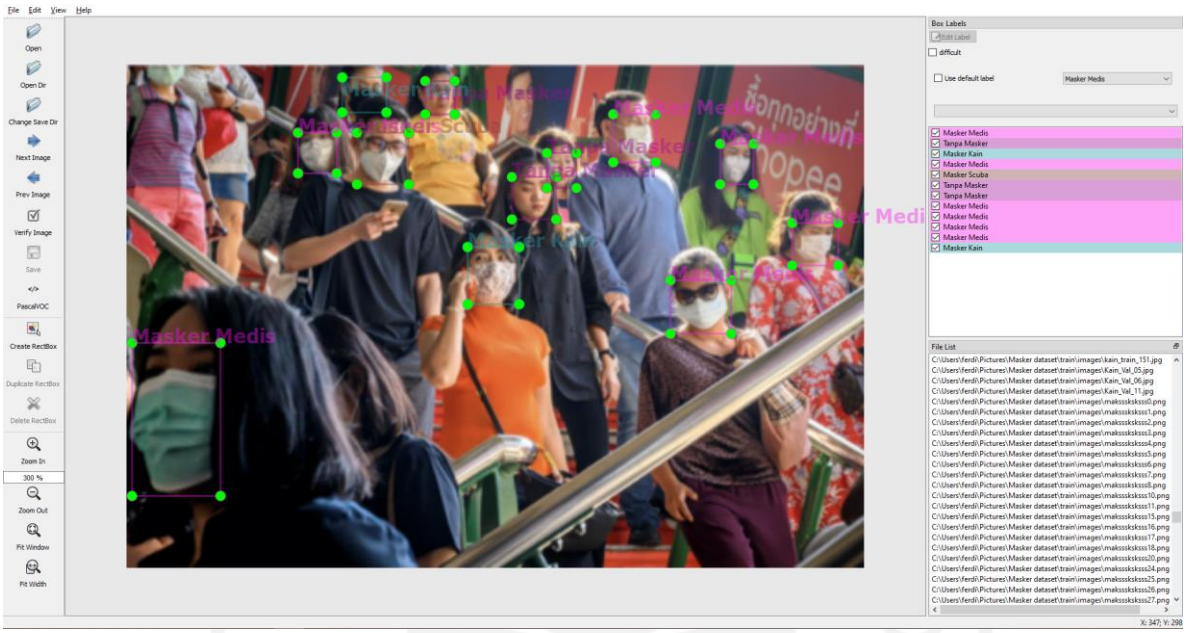
Tabel 3.3 Jumlah gambar pada setiap dataset

Jenis dataset	Sumber	Jumlah
Data Train	MAFA	91
	Flickr-Faces-HQ	160
	Face Mask Detection Dataset	203
	MaskedFace-Net	271
	Instagram *Scraping*	548
Data Validasi	MAFA	58
	Flickr-Faces-HQ	22
	Face Mask Detection Dataset	0
	MaskedFace-Net	27
	Instagram *Scraping*	45
Data Test	MAFA	15
	Flickr-Faces-HQ	27
	Face Mask Detection Dataset	26
	MaskedFace-Net	20
	Instagram *Scraping*	85

3.2.1 Pelabelan Objek

Sebelum memulai prosedur pelatihan model, perlu untuk memberi label pada kelima kelas untuk mengumpulkan informasi tentang objek yang akan dideteksi. Walaupun menggunakan dataset publik yang sudah memiliki label, pada penelitian ini tetap dilakukan pelabelan data karena terdapat perbedaan label pada dataset tersebut dan yang akan digunakan pada penelitian ini. Pada penelitian ini, label yang digunakan adalah masker medis, masker kain, masker scuba, penggunaan masker salah, dan tanpa masker sedangkan pada satu dataset publik yang digunakan, label yang disediakan hanya menggunakan masker dan tidak memakai masker. Dataset lain menyediakan menggunakan masker, penggunaan masker salah, dan tidak memakai masker. Bahkan dataset publik lainnya tidak menyediakan label yang berkaitan. Saat pelatihan berlangsung, model mengambil label yang ditetapkan untuk masing-masing gambar. Seperti terlihat pada Gambar 3.9, proses pelabelan pada penelitian ini dilakukan dengan menggunakan program yang disebut LabelImg. Proses pelabelan objek dilakukan dengan mendefinisikan kotak pembatas dan menetapkan nama kelas atau label untuk itu. Kotak pembatas atau *bounding box* adalah persegi panjang yang mengelilingi objek dalam gambar

dan berfungsi sebagai penanda untuk suatu objek dalam gambar. Koordinat setiap sudut kotak pembatas serta nama label akan disimpan ke dalam file xml oleh LabelImg.



Gambar 3.9 Pemberian label

Isi dari file xml yang dihasilkan setelah menggunakan LabelImg dapat dilihat pada Gambar 3.10. Baris 9 dan 10 menampilkan ukuran panjang dan lebar gambar, baris 11 menampilkan dimensi gambar, baris 14 sampai 25 adalah baris yang berisi informasi tentang kotak pembatas. Dalam informasi tentang kotak pembatas terdapat baris 15 yang berisi nama kelas yang digunakan dan baris 20 sampai 23 berisi informasi koordinat untuk setiap sudut kotak pembatas. Gambar 3.10 merupakan contoh file xml yang hanya memiliki satu label. Ketika dalam satu gambar memiliki lebih dari satu label maka informasi tentang kotak pembatas akan mengikuti sebanyak jumlah label tersebut.

```
<annotation>
        <folder>images</folder>
        <filename>medis_train_015.png</filename>
        <path>C:\Users\ferdi\Pictures\Masker
dataset\train\images\medis_train_015.png</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>900</width>
                <height>900</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>Masker Medis</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>91</xmin>
                        <ymin>455</ymin>
                        <xmax>472</xmax>
                        <ymax>850</ymax>
                </bndbox>
        </object>
</annotation>
```

Gambar 3.10 isi file XML

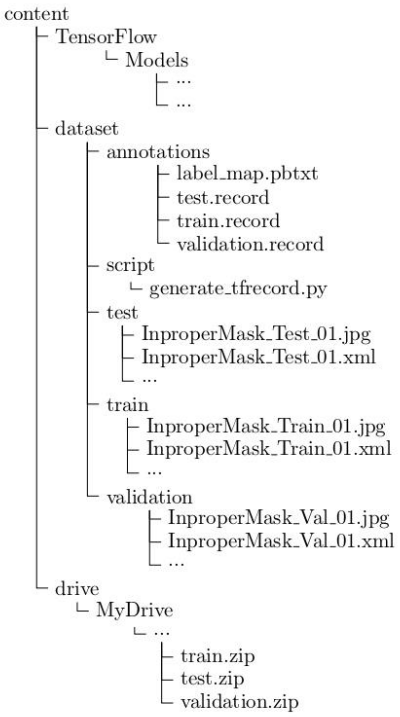
3.3 Penyiapan Sistem Penelitian

Sistem pendeteksi objek yang akan digunakan dalam investigasi ini akan menggunakan *Colaboratory*, yang terkadang dikenal dengan singkatan "*Colab*". Ini adalah produk yang dikembangkan oleh *Google Research*. *Colab* adalah platform daring yang memungkinkan siapa saja membuat dan menjalankan kode *Python arbitrer* menggunakan peramban web. Platform ini unggul dalam beberapa bidang misalnya pembelajaran mesin dan analisis data. *Colab* adalah layanan *notebook Jupyter* yang di-*hosting* yang tidak memerlukan penyiapan apa pun untuk digunakan, sekaligus memberikan akses ke sumber daya komputasi gratis termasuk *graphics processing unit* (GPU). Untuk bahasa yang lebih teknis, *Colab* adalah layanan *Jupyter Notebook* yang di-*hosting*. Agar peneliti dapat terhubung ke *Colab* dan memberikan instruksi ke server, peneliti menggunakan laptop pribadi. Terdapat dua metode pembuatan model yang peneliti gunakan yaitu menggunakan *TensorFlow Object Detection* API dan *library TensorFlow Lite Model Maker*. *Library* adalah bagian penting dari perangkat lunak atau pustaka inti yang diperlukan untuk mengaktifkan studi ini.

Kedua metode ini memerlukan pengaturan *environment* yang berbeda. *Object Detection* API menggunakan *TensorFlow* 2.7.0 dan *keras* 2.7.0 dengan *library* pendukung *tf-models-official* 2.7.0, *tensorflow_io* 0.23.1, *keras* 2.7.0, *folium* 0.2.1, *pyparsing* 2.4.7, *opencv-python-headless* 4.1.2.30 dan *librar*y pendukung lainnya sesuai dengan pengaturan bawaan. Sedangkan untuk *TensorFlow Lite Model Maker* menggunakan *TensorFlow* 2.8.0 dan *TensorFlow Lite Model Maker* 0.4.0 dengan *library* pendukung *folium* 0.2.1, *scan* 1.2.6, *sounddevice* dengan memasang *package* libportaudio2, *opencv-python-headless* 4.1.2.30, dan *librar*y pendukung lainnya sesuai dengan pengaturan bawaan. Keduanya menggunakan *python* 3.7, *Jupyter Notebook*, dan *runtime* GPU. *Python* adalah bahasa pemrograman yang digunakan, dan *JupyterNotebook* adalah editor yang digunakan. Selain itu, *library* TensorFlow yang ditawarkan oleh Google digunakan dalam penelitian ini yang berguna sebagai pendukung penelitian di bidang pembelajaran mesin, pembelajaran mendalam, dan kecerdasan buatan.

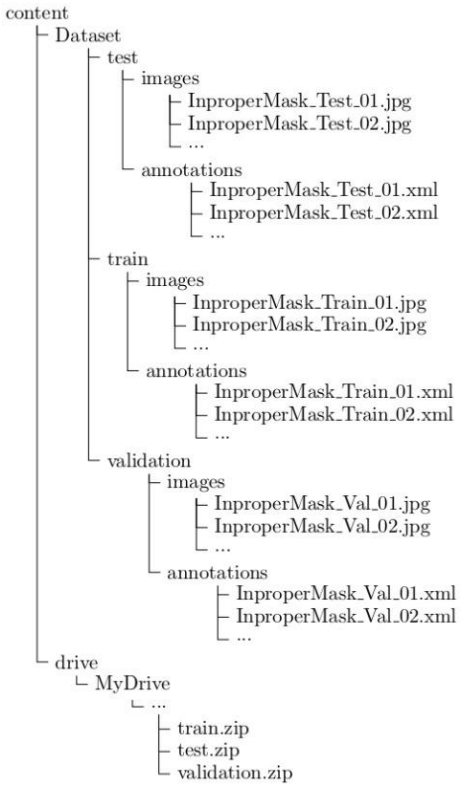
3.3.1 Penyiapan direktori pemodelan

Manajemen direktori yang buruk akan membingukan ketika harus memanggil file tertentu, karena itu menggolongkan file dan memasukkannya ke dalam folder spesifik akan membantu dan akan meningkatkan efisiensi.



Gambar 3.11 *Directory tree* pada *Object Detection API*

Gambar 3.11 merupakan *directory tree* pada *Object Detection API* . Metode *TensorFlow Object Detection* API tidak hanya menggunakan bahasa *python*, tetapi juga menggunakan perintah sistem operasi. Baik bahasa *python* atau perintah sistem operasi, keduanya akan memanggil atau menaruh suatu file pada folder tertentu. File yang terorganisir akan meningkatkan efisiensi dan memudahkan dalam menjalankan tugas tersebut. Adapun folder *TensorFlow* digunakan untuk menaruh sumber kode yang dapat diambil dari Github yang nanti dapat digunakan pada proses *training* seperti pada Gambar 3.28. Folder dataset berisi semua keperluan terkait dataset yang akan digunakan dan berisi lima *sub-folder*. *Annotations* berisi file label dan gambar yang sudah dirubah menjadi *tfrecords* serta *file label map*. *Scripts* berisi *script python* untuk merubah gambar dan label dalam bentuk xml menjadi satu file tfrecords. Folder *test, train,* dan *validation* berisi gambar dan label yang sudah diekstrak dari file *zip* yang terdapat pada *Google Drive*. Dalam folder *drive* yang merupakan folder yang terhubung langsung ke *Google Drive* adalah tiga file dataset untuk *training*, validasi, dan tes. Folder *drive* tidak akan terhapus walaupun *runtime* pada *Google Colab* berhenti sehingga cocok untuk menyimpan dataset yang ukurannya besar dan nantinya akan menyimpan model yang sudah dilatih. Dalam folder *drive* terdapat banyak folder yang tak berkaitan dengan penelitian ini, untuk menjaga simplisitas, *directory tree* ini hanya menampilkan bagian yang akan digunakan pada penelitian ini.



Gambar 3.12 *Directory tree* pada *TFLite Model Maker*

Gambar 3.12 merupakan *directory tree* pada *TFLite Model Maker*. Hampir serupa dengan TF *Object Detection* API, sebagian besar metode *TFLite Model Maker* hanya menggunakan bahasa *python* karena sintaks yang sebelumnya harus menggunakan *script* tersendiri sudah termasuk dalam pustaka tersebut. Pada metode ini juga membagi dataset menjadi tiga bagian yaitu *train*, validasi, dan tes tetapi anotasi dan gambarnya dipisah dan dimasukan pada folder tersendiri. Folder *drive* juga digunakan untuk menyimpan dataset serta menyimpan hasil pelatihan model sama seperti TF *Object Detection* API.

3.4 Pemodelan

Pada proses pemodelan, peneliti akan menjalankan model empat kali, setiap kali menggunakan jenis arsitektur yang berbeda tetapi menggunakan dataset yang sama. Model yang dipilih adalah *MobileNet V2 FPNLite* dan *SSD ResNet50 V1 FPN* dari metode *Object Detection* API serta *EfficientDet-Lite0* dan *EfficientDet-Lite3* dari *TensorFlow Lite Model Maker.*

MobileNet V2 FPNLite dipilih karena model tersebut berfokus pada efisiensi dan didesain untuk perangkat seluler. *SSD ResNet50 V1 FPN* dipilih karena model tersebut

memiliki arsitektur yang mendalam, bahkan lebih dalam dibanding *neural network* sebelumnya.

Berdasarkan dari Tabel 2.1, model yang dipilih menggunakan metode *TFLite Model Maker* adalah *EfficientDet-Lite0 dan EfficientDet-Lite3.* Kedua model tersebut dipilih berdasarkan *latency* dan *Average Precision*. *EfficientDet-Lite0* dipilih karena model tesebut memiliki *latency* terkecil sedangkan *EfficientDet-Lite3* dipilih karena model tersebut memiliki *Average Precision* yang tinggi tetapi *latency* yang tidak terlalu besar. Selanjutnya masing-masing arsitektur tersebut akan dilanjutkan ke tahap pelatihan hingga merubah model menjadi dapat beroperasi pada perangkat seluler.

Tahap pemodelan dimulai dari mempersiapkan sistem penelitian dengan menyiapkan *package* dan *library* sesuai dengan yang tertulis bab 3.3 diikuti dengan bertahap membuat direktori pada *runtime* menjadi seperti bab 3.3.1. Setiap tahapan pada bagian ini menjelaskan masing-masing tahapan untuk metode *Object Detection* API dan *TensorFlow Lite Model Maker.*

3.4.1 Penyiapan *package* dan *library*

Mengunduh dan menginstal setiap *package dan library* yang dibutuhkan untuk menjalankan proses pelatihan. Tahapan pada *Object Detection* API dimulai dari mengunduh *package* dari Github, menyesuaikan versi *library,* lalu menginstal *package*. Sedangkan pada *TensorFlow Lite Model Maker* tahapannya hanya menginstal *library* melalui perintah sistem operasi.

Object Detection API

Pada metode ini, hal pertama yang dilakukan adalah mengunduh *package TensorFlow Object Detection* API yang dapat diunduh dari Github dengan cara seperti pada Gambar 3.13. Baris pertama pada Gambar 3.13 berfungsi untuk membuat direktori khusus untuk menyimpan *package* ini, baris kedua berfungsi untuk mengubah *current working directory* sehingga pada saat menjalankan baris ketiga, *package* tersebut akan berada di dalam folder "*TensorFlow*". Baris ketiga mengununduh *package TensorFlow Object Detection* dari Github.

```
1 !mkdir TensorFlow
2 %cd /content/TensorFlow
3 !git clone https://github.com/tensorflow/models.git
```

Gambar 3.13 Sintaks mengunduh *package TensorFlow Object Detection* API

Setelah *package* selesai diunduh, tahap selanjutnya adalah merubah isi dari variabel *" REQUIRED_PACKAGES*" pada sebuah file yang dapat ditemukan pada "*TensorFlow/models/research/object_detection/packages/tf2/setup.py*" menjadi seperti Gambar 3.14. Menggunakan versi *library* terbaru dapat menyebabkan konflik dengan *library* penting yang lain sehingga beberapa *library* penting tidak dapat di-*install*. Menurunkan versi *library* tertentu dapat mengatasi masalah tersebut. Versi *library* yang perlu diturunkan terdapat pada baris 16 hingga 22.

```
 1 REQUIRED_PACKAGES = [
 2     # Required for apache-beam with PY3
 3     'avro-python3',
 4     'apache-beam',
 5     'pillow',
 6     'lxml',
 7     'matplotlib',
 8     'Cython',
 9     'contextlib2',
10     'tf-slim',
11     'six',
12     'pycocotools',
13     'lvis',
14     'scipy',
15     'pandas',
16     'tensorflow==2.7.0',
17     'tf-models-official==2.7.0',
18     'tensorflow_io==0.23.1',
19     'keras==2.7.0',
20     'folium==0.2.1',
21     'opencv-python-headless==4.1.2.30',
22     'pyparsing==2.4.7'  # TODO(b/204103388)
23 ]
```

Gambar 3.14 *Library* yang diperlukan pada *TensorFlow Object Detection* API

Setelah tahapan persiapan selesai, barulah dapat menjalankan sintaks instalasi *package TensorFlow Object Detection* API seperti pada Gambar 3.15. Baris pertama mengganti *current working directory* sehingga dapat menjalankan baris kedua. *Object Detection* API memerlukan *protobufs* untuk mengkonfigurasi model dan parameter pelatihan. Sebelum *Object Detection* API dapat digunakan, library *Protobuf* harus diunduh dan di-*compile*. Baris kedua berfungsi untuk mengunduh dan meng-*compile Protobuf*. Baris keempat dan kelima berfungsi untuk menginstal *TensorFlow Object Detection* API.

```
%cd /content/TensorFlow/models/research
!protoc object_detection/protos/*.proto --python_out=.

!cp object_detection/packages/tf2/setup.py .
!python -m pip -q install .
```

Gambar 3.15 Sintaks menginstal *package TensorFlow Object Detection* API

TensorFlow Lite Model Maker

Persiapan pada metode ini tidak serumit *Object Detection* API, karena dasar dari metode ini adalah *library TensorFlow Lite Model Maker*, meski begitu tetap ada *library* pendukung yang versinya harus diturunkan agar cocok dengan *library* yang lain. Gambar 3.16 merupakan seluruh sintaks yang diperlukan dalam persiapan sistem penelitian pada metode *TensorFlow Lite Model Maker*. Baris tiga dan empat merupakan *library* inti dari metode ini, yaitu library *TensorFlow Lite Model Maker.* Baris lainnya berfungsi untuk menurunkan versi *library* tertentu dengan cara menghapus lalu menginstal kembali *library* yang sama namun dengan versi yang sudah diturunkan.

```
!pip uninstall -y folium
!pip install -q folium==0.2.1
!pip install -q tflite-model-maker
!pip install -q tflite-support
!pip uninstall -y scann
!pip install -q scann==1.2.6
!pip uninstall -y opencv_python_headless
!pip install -q opencv-python-headless==4.1.2.30
```

Gambar 3.16 Sintaks persiapan sistem

3.4.2 *Preprocessing*

Sebelum pelatihan dimulai, dataset yang sudah dikumpulkan akan disiapkan terlebih dahulu agar bisa dimasukkan ke model. Proses *preprocessing* berbeda untuk setiap metode yang digunakan karena kebutuhan input setiap metode berbeda. Secara garis besar, tahapan pada *Object Detection* API adalah membuat file label dan menyatukan dataset. Langkah yang dilakukan pada *TensorFlow Lite Model Maker* adalah mengkonversi gambar dengan format selain *jpeg* menjadi *jpeg* lalu menampung lokasi dataset pada sebuah variabel.

***Object Detection* API**

Tahap pertama adalah membuat direktori berdasar pada bab 3.3.1 dan memindahkan dataset dari *Google Drive* ke *Colab runtime* seperti pada Gambar 3.17. Baris pertama berfungsi

untuk mengganti *current working directory* sehingga saat membuat direktori menggunakan sintaks pada baris dua sampai empat, direktori tersebut berada di dalam folder *content*. Baris kelima hingga ketujuh berfungsi untuk meng-ekstrak dataset yang berada pada *Google Drive* langsung ke *Colab Runtime*. File yang diekstrak dimasukan ke dalam folder sesuai nama file zip tersebut.

```
1 %cd /content
2 !mkdir dataset
3 !mkdir dataset/annotations
4 !mkdir dataset/script
5 !unzip -q /content/drive/MyDrive/.../train.zip -d /content/dataset
6 !unzip -q /content/drive/MyDrive/.../validation.zip -d /content/dataset
7 !unzip -q /content/drive/MyDrive/.../test.zip -d /content/dataset
```

Gambar 3.17 Sintaks membuat direktori dan memindahkan dataset

Memiliki direktori yang bercabang dapat menyulitkan jika harus mengetik *path* setiap merujuk file yang terdapat dalam folder tersebut. Maka dapat disederhanakan dengan masukan *path* kedalam variabel sehingga dapat memudahkan dalam menulis dan membaca sintaks. Cara pendeklarasian variabel dapat dilihat pada Gambar 3.18. Baris kesatu hingga ketiga merupakan pendeklarasian variabel sederhana berupa teks berisi *path* menuju folder tertentu. Langkah selanjutnya adalah membuat file label.

```
1 SCRIPTS_PATH = "/content/dataset/script"
2 IMAGE_PATH = "/content/dataset"
3 ANNOTATION_PATH = "/content/dataset/annotations"
```

Gambar 3.18 Sintaks mendeklarasi *path*

Object Detection API memerlukan input bertipe *TensorFlow record*. *TensorFlow record* adalah sebuah format sederhana untuk menyimpan serangkaian data. Untuk membuat file tersebut diperlukan file pbtxt yang berisi daftar kelas atau label yang akan dideteksi dalam bentuk *dictionary* lalu *dictionary* tersebut di-*export* menjadi file pbtxt. File pbtxt adalah file yang berisi label yang akan digunakan saat proses pelatihan model. Gambar 3.19 berisi sintaks untuk membuat file pbtxt. Pada baris dua sampai enam adalah *dictionary* yang berisi nama kelas dan id kelasnya. Urutan id pada label berdasarkan urutan *predefined class* pada LabelImg. Pada baris Sembilan sampai empat belas berfungsi untuk menulis file pbtxt. File tersebut akan ditaruh pada folder *annotations*.

```
labels = [
    {'name':'Masker Medis', 'id':1},
    {'name':'Masker Kain', 'id':2},
    {'name':'Masker Scuba', 'id':3},
    {'name':'Penggunaan Masker Salah', 'id':4},
    {'name':'Tanpa Masker', 'id':5},
]

with open(ANNOTATION_PATH + '/label_map.pbtxt', 'w') as f:
    for label in labels:
        f.write('item { \n')
        f.write('\tname:\'{}\'\n'.format(label['name']))
        f.write('\tid:{}\n'.format(label['id']))
        f.write('}\n')
```

Gambar 3.19 Baris kode membuat pbtxt file

Isi file pbtxt yang telah dibuat menggunakan sintaks pada Gambar 3.19 dapat dilihat pada Gambar 3.20. Isi file ini cukup sederhana, hanya berisikan nama kelas dan id kelas tersebut untuk setiap kelas yang didefinisikan.

```
item {
        name:'Masker Medis'
        id:1
}
item {
        name:'Masker Kain'
        id:2
}
item {
        name:'Masker Scuba'
        id:3
}
item {
        name:'Penggunaan Masker Salah'
        id:4
}
item {
        name:'Tanpa Masker'
        id:5
}
```

Gambar 3.20 Isi file pbtxt

Setelah membuat file pbtxt, proses tahapan selanjutnya adalah membuat *tensorflow record* sehingga gambar pada dataset dapat diproses saat pelatihan. Dalam membuat *tensorflow record*, langkah pertama yang dilakukan adalah mengunduh *script tensorflow record generator* dengan menggunakan sintaks yang dapat dilihat pada Gambar 3.21. Baris ketiga berisi variabel

yang menampung link *tfrecord generator*, lalu file tersebut dapat diunduh menggunakan sintaks pada baris keempat.

```
1 import urllib.request
2
3 url = 'https://tensorflow-object-detection-api-
  tutorial.readthedocs.io/en/latest/_downloads/da4babe668a8afb093cc7776d7
  e630f3/generate_tfrecord.py'
  urllib.request.urlretrieve(url=url,
4 filename=SCRIPTS_PATH+'/generate_tfrecord.py')
```

Gambar 3.21 Sintaks untuk mengunduh *tfrecord generator*.

Setelah diunduh, *script python* tersebut dijalankan menggunakan perintah seperti pada Gambar 3.22 beserta *flag* untuk mendefinisikan lokasi folder yang berisi foto dan anotasi, lokasi file *label map*, dan *path output tensorflow record*. Baris pertama, kedua, ketiga berfungsi untuk membuat file *tensorflow record* untuk data *train*, data validasi, dan data test secara berurutan.

```
1 !python {SCRIPTS_PATH + '/generate_tfrecord.py'} -x {IMAGE_PATH +
  '/train'} -l {ANNOTATION_PATH + '/label_map.pbtxt'} -o {ANNOTATION_PATH
  + '/train.record'}
2 !python {SCRIPTS_PATH + '/generate_tfrecord.py'} -x {IMAGE_PATH +
  '/validation'} -l {ANNOTATION_PATH + '/label_map.pbtxt'} -o
  {ANNOTATION_PATH + '/validation.record'}
3 !python {SCRIPTS_PATH + '/generate_tfrecord.py'} -x {IMAGE_PATH +
  '/test'} -l {ANNOTATION_PATH + '/label_map.pbtxt'} -o {ANNOTATION_PATH
  + '/test.record'}
```

Gambar 3.22 Sintaks membuat *tensorflow record*

TensorFlow Lite Model Maker

Sama seperti metode *Object Detection* API, Langkah pertama adalah membuat direktori mengikuti bab 3.3.1 dan memindahkan dataset dari *Google Drive* ke *Colab Runtime.* Proses ini dapat dijalankan menggunakan sintaks seperti pada Gambar 3.23. Baris pertama berfungsi untuk membuat direktori dataset. Sedangkan baris ketiga sampai kelima berfungsi untuk mengekstrak file dari *Google Drive* ke dalam direktori dataset. File yang telah diekstrak berada dalam folder sesuai dengan nama masing-masing file zip. Setelah dataset berada di *Colab Runtime*, tahap *preprocessing* dapat dijalankan.

```
1 !mkdir dataset
2
3 !unzip -q /content/drive/MyDrive/.../train.zip -d /content/dataset
4 !unzip -q /content/drive/MyDrive/.../test.zip -d /content/dataset
5 !unzip -q /content/drive/MyDrive/.../validation.zip -d /content/dataset
```

Gambar 3.23 Sintaks membuat direktori dan memindahkan dataset

Ukuran input pada model ini beragam tergantung *pre-trained* model yang dipilih. Untuk *EfficientDet-Lite3* ukuran inputnya adalah 512x512 *pixel*s sedangkan *EfficientDet-Lite0* ukuran inputnya adalah 320x320 *pixel*s. Gambar pada dataset memiliki ukuran yang beragam, namun pada *library tensorflow lite model make*r sudah melakukan *resize* untuk menyesuaikan ukuran gambar input menjadi sesuai dengan input model.

Meski begitu *library TensorFlow Lite Model Maker* memiliki kekurangan yaitu tidak dapat menerima gambar dengan ekstensi selain ‘jpeg’ sehingga *preprocessing* tetap perlu dilakukan untuk mengubah gambar yang memiliki ekstensi ‘png’ dan ‘jpg’ menjadi ‘jpeg’ dengan melakukan *re-encoding* dengan menggunakan *library cv2*. Selanjutnya adalah menyesuaikan nama foto pada *tag ‘filename’* pada file label (.xml) dengan mengganti tulisan ekstensi ‘png’ atau ‘jpg’ menjadi ‘jpeg’ menggunakan *library lxml*. Sintaks untuk merubah ekstensi ‘png’ atau ‘jpg’ menjadi ‘jpeg’ dapat dilihat pada Gambar 3.24. Baris ke-27 berfungsi untuk membaca nama semua file pada folder lalu membuat iterasi sekaligus menampilkan *progress bar*. Baris ke-28 sampai 33 adalah proses *re-encoding* dengan cara membaca gambar dengan format ‘png’ atau ‘jpg’ dengan bantuan *library cv2* lalu menyimpan kembali foto tersebut dengan format ‘jpeg’, sesuai dengan yang *TensorFlow Lite Model Maker* butuhkan. Langkah selanjutnya adalah memanggil fungsi untuk mengubah *tag* “*filename*” pada file xml. Langkah tersebut berada di baris 10 sampai 22. Pertama, fungsi tersebut mencari file xml yang merujuk kepada gambar yang ekstensinya sedang diganti, lalu mencari *tag* “*filename*” di dalam file xml tersebut dan mengambil nilainya seperti pada baris 17. Selanjutnya adalah mendapatkan *path* ke *tag* “*filename”* seperti baris ke-19. Setelah mendapat nilai dan *path* dari “*filename*”, barulah dapat merubah nilai dari “*filename*” tersebut dan menulisnya kembali seperti pada baris 21 dan 22.

```
from os import listdir
from os.path import splitext
from tqdm.notebook import tqdm_notebook
import cv2
import shutil

from lxml import etree
import os

def xml_editor(path, filename):
  path = path+"/annotations"
  xml_filename = path + '/' + splitext(filename)[0]+'.xml'

  tree = etree.parse(xml_filename)
  root = tree.getroot()

  filename = tree.find('filename')

  code = root.xpath('//annotation/filename')
  if code:
      code[0].text = os.path.splitext(filename.text)[0]+'.jpeg'
      etree.ElementTree(root).write(xml_filename, pretty_print=True)

def change_format(path):
  raw_path = path
  path = path+"/images"
  for filename in tqdm_notebook(listdir(path)):
    if filename.endswith(('.jpg', '.png')):
      try:
        read = cv2.imread(f'{path}/{filename}')
        cv2.imwrite(f'{path}/{splitext(filename)[0]}.jpeg', read)
        xml_editor(raw_path, filename)
        os.remove(f'{path}/{filename}')
      except Exception as e:
        print(f'Error on: {filename}, with error {e}')

change_format('dataset/train-biased')
change_format('dataset/test')
change_format('dataset/validation')
```

Gambar 3.24 Sintaks png re-encoding

Setelah membuat tipe gambar kompatibel dengan model, langkah selanjutnya adalah mendefinisikan lokasi dataset tersebut ke dalam objek *"ObjectDetectorDataLoader"* untuk dataset *train,* validasi, dan tes. Proses tersebut dapat dijalankan dengan sintaks yang terdapat pada Gambar 3.25. Baris pertama sampai kelima adalah lokasi penyimpanan dataset *train,* baris ketujuh sampai sebelas adalah lokasi penyimpanan dataset validasi, dan baris tiga belas sampai tujuh belas adalah lokasi penyimpanan dataset tes. Terdapat beberapa parameter dalam kode "*object_detector.DataLoader.from_pascal_voc",* yaitu lokasi gambar, lokasi anotasi, dan daftar label yang ingin dideteksi dengan penulisan berurutan.

```
train_data = object_detector.DataLoader.from_pascal_voc(
    'dataset/train/images',
    'dataset/train/annotations',
    ['Masker Medis', 'Masker Kain', 'Masker Scuba', 'Penggunaan Masker Salah', 'Tanpa Masker']
)

val_data = object_detector.DataLoader.from_pascal_voc(
    'dataset/validation/images',
    'dataset/validation/annotations',
    ['Masker Medis', 'Masker Kain', 'Masker Scuba', 'Penggunaan Masker Salah', 'Tanpa Masker']
)

test_data = object_detector.DataLoader.from_pascal_voc(
    'dataset/test/images',
    'dataset/test/annotations',
    ['Masker Medis', 'Masker Kain', 'Masker Scuba', 'Penggunaan Masker Salah', 'Tanpa Masker']
)
```

Gambar 3.25 Sintaks mendefinisikan lokasi dataset

3.4.3 *Training*

Proses *training* pada kedua metode menggunakan sintaks dan langkah-langkah yang berbeda. Pada *Object Detection* API Langkah-langkah yang diperlukan adalah mengunduh *pre-trained* model dari *TensorFlow Zoo,* menyiapkan *tensorboard*, mengatur konfigurasi *pipeline,* barulah bisa memulai proses *training*. Sedangkan pada *TensorFlow Lite Model Maker* hanya memilih model dan sudah dapat memulai *training*.

Object Detection API

Langkah pertama adalah mengunduh *pre-trained* model. Model yang ditawarkan bervariasi dengan berbagai macam arsitektur, akurasi, dan waktu inferensi. Pilihan model dapat dilihat pada laman GitHub *TensorFlow Zoo*. Pada metode *Object Detection* API, model yang digunakan adalah *SSD MobileNet V2 FPNLite 320x320* dan *SSD ResNet50 V1 FPN 640x640 (RetinaNet50)*.

Gambar 3.26 berisi sintaks yang digunakan untuk mengunduh model. Baris pertama pada kode adalah perintah untuk mengunduh file dari tautan yang diberikan. Dari contoh sintaks ini, model yang diunduh adalah *ResNet50 V1 FPN 640x640 (RetinaNet50)* yang didapat dari laman Github *TensorFlow zoo*. File yang diunduh berbentuk *archive* sehingga perlu diekstrak untuk mendapatkan kontennya. Baris kedua merupakan perintah untuk mengekstrak file *archive* tersebut.

```
1 !wget
  http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd
  _resnet50_v1_fpn_640x640_coco17_tpu-8.tar.gz
2
3 !tar -xzvf ssd_resnet50_v1_fpn_640x640_coco17_tpu-8.tar.gz
```

Gambar 3.26 Sintaks unduh *pre-trained* model

Sintaks pada Gambar 3.27 dapat digunakan untuk menampilkan statistik ketika melatih model. Adapun variabel yang dapat dilihat adalah *loss* dan *learning rate*. Statistik ini dapat membantu untuk menentukan kapan sebuah model mengalami memiliki nilai *loss* terendah dan dapat menentukan apakah model mengalami *overfitting*. *Overfitting* merupakan keadaan ketika model terlalu fokus pada data *train* sehingga model gagal menggeneralisasi kelas yang ingin diprediksi. Baris pertama berfungsi untuk memuat *tensorboard* dan baris kedua berfungsi untuk mendefinisikan lokasi menyimpan *tensorboard*.

```
1 %load_ext tensorboard
2 %tensorboard --logdir="/content/drive/MyDrive/TugasAkhir/trial-
  5/ssd_resnet50_v1_fpn_640x640/output_training/"
```

Gambar 3.27 Sintaks untuk menampilkan tensorboard

Dalam folder model yang telah diunduh, terdapat file "*pipeline.config".* Dalam file tersebut berisi banyak variabel dan *hyperparameter*, namun tidak semuanya perlu dirubah. Hanya bagian penting seperti jumlah kelas, jumlah *steps*, ukuran *batch*, tipe *checkpoint*, dan *path* menuju label serta *tensorflow records* seperti yang ditampilkan pada Tabel 3.4. Variabel *num_classes* digunakan untuk menulis jumlah kelas yang akan dideteksi, *num_steps* digunakan untuk menulis seberapa banyak iterasi ketika melatih model, *fine_tune_checkpoint* berisi *path* menuju file *checkpoint* untuk model awal atau melanjuti model yang sudah dilatih sebagian, *batch_size* digunakan untuk melatih model dengan seberapa banyak gambar pada setiap *step*, *fine_tune_checkpoint_type* digunakan untuk memilih jenis deteksi seperti klasifikasi atau deteksi, *label_map_path* berisi *path* untuk menuju file yang berisi label yang akan digunakan untuk melatih model, dan *input_path* berisi *path* untuk menuju file *tensorflow records*. *Input_path* memiliki dua bagian yang harus diisi yaitu pada bagian *train* dan *eval*, bagian *train* digunakan untuk data yang melatih model sedangkan *eval* digunakan pada data yang mengevaluasi model atau memeriksa performa model.

Tabel 3.4 *Hyperparameter* pada *Object Detection* API

Hyperparameter	***Value***
num_classes	5
num_steps	30000 (MobileNet) dan 20000 (ResNet)
fine_tune_checkpoint	*"/content/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpoint/ckpt-0"* (MobileNet) dan *"/content/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/checkpoint/ckpt-0"* (ResNet)
batch_size	8
fine_tune_checkpoint_type	"*detection*"
label_map_path	*"/content/dataset/annotations/label_map.pbtxt"*
input_path (train)	*"/content/dataset/annotations/train.record"*
input_path (eval)	*"/content/dataset/annotations/validation.record"* (untuk validasi) dan *"/content/dataset/annotations/test.record"* (untuk *test*)

Setelah mengatur file konfigurasi, tahapan selanjutnya adalah menjalankan sintaks seperti pada Gambar 3.28. Baris kesatu berfungsi untuk mengganti direktori kerja menjadi folder *Object Detection* API. Baris ketiga sampai kelima berfungsi menjalankan *script python* yang melaksanakan proses pelatihan dengan mendefinisikan *path* konfigurasi model dan direktori luaran hasil pelatihan. Konfigurasi file berisi *hyperparameter* dan variabel yang menampung lokasi label dan *tensorflow record*. Adapun isi keseluruhan file konfigurasi tersebut dapat dilihat pada lampiran, namun *hyperparameter* yang perlu diubah dapat dilihat pada Tabel 3.4. Jumlah *steps* dan *batch* didapatkan dari *trial and error* yang dilihat dari hasil pelatihan dan validasi, meski begitu selalu terdapat kemungkinan untuk menemukan nilai *hyperparameter* yang lebih baik lagi. Misalnya pada *MobileNet* yang menurunkan iterasi dari 35000 *steps* menjadi 30000 *steps* lalu pada *ResNet* menggunakan 20000 *steps* dari yang sebelumnya 30000 *steps*, menurunkan jumlah iterasi membuat selisih nilai *loss* dari data *train* dan data validasi berukurang yang artinya menurunkan kemungkinan terjadinya *overfitting*. Ukuran *batch* didapat dari kelipatan dua terbesar yang dapat dijalankan berdasarkan sistem penelitian saat ini. Luaran hasil pelatihan berupa folder yang berisi file *checkpoint* yang dibuat setiap seribu *steps*. Proses pelatihan bisa berjalan sangat lama sehingga file *checkpoint* dapat membantu dengan melanjutkan proses pelatihan yang terhenti. Selain itu, nantinya folder

checkpoint ini akan digunakan dalam membuat *saved model*. *Saved model* merupakan format penyimpanan *Object Detection* API.

```
1 %cd "/content/TensorFlow/models/research/object_detection"
2
3 !python model_main_tf2.py \
4 --pipeline_config_path="/content/ssd_resnet50_v1_fpn_640x640_coco17_tpu
  -8/pipeline.config" \
5 --model_dir="/content/drive/MyDrive/TugasAkhir/trial-
  5/ssd_resnet50_v1_fpn_640x640/output_training/" --alsologtostderr
```

Gambar 3.28 Sintaks untuk memulai pelatihan model

TensorFlow Lite Model Maker

Proses *training* pada *TensorFlow Lite Model Maker* lebih sederhana dibanding *object detection* API. Proses yang dibutuhkan hanya memilih model dan menjalankan pelatihan. Pemilihan model dapat dilakukan menggunakan sintaks seperti pada baris pertama di Gambar 3.29. Setelah memilih model, tahap selanjutnya adalah melakukan *training* dengan sintaks pada baris ketiga sampai kedelapan. Beberapa parameter diperlukan dalam fase pelatihan yang dapat ditemukan di baris kode *"object_detector.create()."* Baris kesatu yang nantinya akan dipanggil pada baris keempat, menampilkan model dasar yang akan digunakan untuk melatih model. Dalam sintaks ini, model *efficientdet-lite0* yang akan digunakan oleh peneliti. Baris kelima menunjukkan jumlah ukuran *batch* yang akan digunakan, peneliti menggunakan ukuran *batch* 8. Ukuran *batch* adalah jumlah sampel yang tersebar ke dalam arsitektur *neural network*. Kemudian, pada baris ketujuh, menentukan jumlah *epochs* yang akan digunakan, yaitu 25 *epochs* pada *efficientdet-lite0* dan 35 *epoch* pada *efficientdet-lite3*. *Epochs* adalah jumlah iterasi yang dilakukan melewati seluruh data pada dataset. Terakhir, pada baris keenam terdapat parameter *"train whole model"* yang digunakan peneliti untuk memanfaatkan fitur yang ditawarkan *tensorflow lite model maker* agar pelatihan lebih mendalam dan tidak hanya pada *layer output*. Jumlah *Epochs* ditentukan berdasarkan *trial and error* yang dilihat dari nilai *loss* pelatihan dan validasi. Misalnya pada *EfficientDet-Lite3* yang memiliki *epoch* 35 setelah mencoba jumlah *epoch* 80 dan 40 yang nilai *loss* validasinya tidak menurun sedangkan nilai *loss train* terus menurun. Nilai *loss* validasi *EfficientDet-Lite3* pada *epoch* terakhir berkisar 0.5-0.4 sedangkan pada *loss train* berkisar 0.2-0.1. *Epoch* yang rendah seperti 25 *epoch* pada *EfficientDet-Lite3* juga membuat nilai *loss* validasinya tidak mendekati *loss* terendah pada *epoch* 80. Pada *EfficientDet-Lite0* menggunakan *epoch* 25 setelah mencoba *epoch* 35 seperti *EfficientDet-Lite3*, namun nilai *loss* valdiasinya tidak menurun tetapi nilai *loss train* terus

menurun. Pada *epoch* terakhir, nilai *loss EfficientDet-Lite0* dengan 35 *epoch* berkisar 0.3-0.25 pada *loss train* dan 0.6-0.45 pada *loss* validasi. Sedangkan *batch size* ditentukan berdasarkan kelipatan dua terbesar yang dapat dijalankan berdasarkan sistem penelitian saat ini. Jumlah *steps* ditentukan oleh *library* berdasarkan jumlah data pada dataset dan *batch size* dengan cara membagi jumlah data *train* yakni 1273 foto dibagi dengan jumlah *batch*, yakin 8 sehingga didapatkan jumlah *steps* sejumlah 159 per *epochs.*

```
1 spec = model_spec.get('efficientdet_lite0')
2
3 model = object_detector.create(train_data,
4                                model_spec=spec,
5                                batch_size=8,
6                                train_whole_model=True,
7                                epochs=25,
8                                validation_data=val_data)
```

Gambar 3.29 Sintaks pemilihan model dan melakukan pelatihan

3.4.4 Ekspor

Setelah melakukan pelatihan model, hasilnya dapat diekspor ke dalam format tflite dan *saved model*. Format tflite adalah tipe file penyimpanan model yang kompatibel dengan perangkat seluler. Proses deteksi juga dapat dilakukan dengan menggunakan komputer atau perangkat seluler. Tahapan ekspor model untuk setiap metode berbeda, adapun tahapan pada *Object Detection* API secara garis besar adalah menyimpan model dalam format *saved model*, lalu model dengan latensi terendah akan diubah menjadi tflite serta menambahkan *metadata*. Model *TensorFlow Lite Model Maker* langsung tersimpan dalam format tflite.

Object Detection API

Pada *Object Detection* API model yang selesai dilatih akan diekspor dengan format *saved model* yang mengambil bobot dan arsitektur yang terdapat pada folder *checkpoint*. Format *saved model* tidak dapat diimplementasikan langsung pada perangkat seluler sehingga diharuskan untuk mengubah *saved model* menjadi format yang kompatibel dengan perangkat seluler, yaitu format tflite. Setelah menjadi tflite, perlu ditambahkan *metadata* sehingga memudahkan proses integrasi dan menjalankan inferensi model. Proses ekspor model pada metode ini sangat kompleks, tahap pertama yang dilakukan adalah membuat *saved model* menggunakan sintaks pada Gambar 3.30. Baris pertama mengganti *current working directory*, baris kedua menjalankan *script python* dengan 3 *flag* seperti pada baris 4 sampai 6. Baris 4

berisi *path* menuju direktori *checkpoint* model, baris 5 berisi *path* tempat menyimpan file *config*, dan baris 6 berisi *path* lokasi dari hasil ekspor ini.

```
1 %cd "/content/TensorFlow/models/research/object_detection"
2 !python exporter_main_v2.py \
3 --trained_checkpoint_dir="/content/drive/MyDrive/.../output_training/"
  \
4 --pipeline_config_path="/content/.../pipeline.config" \
5 --output_directory "/content/drive/MyDrive/.../inference"
```

Gambar 3.30 Sintaks ekspor model dalam format *saved model*

Untuk merubah *saved model* menjadi kompatibel dengan perangkat bergerak, perlu dijalankan serangkaian proses mulai dari menyiapkan sistem konversi, mendapatkan *graph* inferensi, merubah ke format tflite, dan menambah *metadata*. Berdasarkan forum GitHub "*https://github.com/tensorflow/models/issues/9287"*, Model *ResNet* memiliki masalah di mana model tersebut tidak dapat bekerja ketika dikonversi menjadi format tflite. Maka, hanya model *MobileNet* yang akan dikonversi menjadi tflite.

Dalam menyiapkan sistem konversi, langkah-langkah yang diperlukan sama seperti tahapan penyiapan *package* dan *library* pada metode *Object Detection* API. Langkah kedua adalah mendapatkan *graph* inferensi lalu membuat *saved model* yang dapat dikonversi menjadi tflite. Langkah tersebut dapat dilakukan dengan memindahkan *saved model* ke *Colab runtime*, lalu menjalankan sintaks pada Gambar 3.31. *Flag* pada Gambar 3.31 serupa dengan Gambar 3.30, namun format penyimpanan yang dihasilkan berbeda dengan ekspor *saved model* sebelumnya karena file yang dipanggil berbeda.

```
1 !python models/research/object_detection/export_tflite_graph_tf2.py \
2     --trained_checkpoint_dir {'/content/inference/checkpoint'} \
3     --output_directory {'/content/drive/MyDrive/.../tflite'} \
4     --pipeline_config_path {'/content/inference/pipeline.config'}
```

Gambar 3.31 Sintaks mendapatkan *inference graph*

Langkah ketiga adalah merubah *saved model* menjadi tflite dengan menggunakan *TensorFlow Lite Converter*. Sintaks untuk melakukan melakukan tahap ini dapat dilihat pada Gambar 3.32. Baris pertama berfungsi untuk mendeklarasikan lokasi penyimpanan tflite, baris ketiga memanggil *saved model* khusus tflite lalu dikonversi dengan baris empat dan lima. Baris kelima hanya menyimpan model dalam *local* variabel sehingga harus ditulis ke dalam file dengan menggunakan baris ketujuh dan kedelapan.

```
1 _TFLITE_MODEL_PATH = ".../model.tflite"
2
3 converter =
  tf.lite.TFLiteConverter.from_saved_model('.../tflite/saved_model')
4 converter.optimizations = [tf.lite.Optimize.DEFAULT]
5 tflite_model = converter.convert()
6
7 with open(_TFLITE_MODEL_PATH, 'wb') as f:
8   f.write(tflite_model)
```

Gambar 3.32 Sintaks konversi *saved model* menjadi tflite

Langkah terakhir adalah menambah *metadata* sehingga memudahkan proses integrasi ke perangkat seluler dan memudahkan proses inferensi dengan melakukan *pre & post processing* yang diperlukan model. Sintaks untuk menambah *metadata* terdapat pada Gambar 3.33. Dalam sintaks ini, terdapat file "*label_map.pbtxt*" yang diperoleh dengan cara yang sama seperti Gambar 3.19. Baris ketujuh sampai enam belas berfungsi untuk merubah label map *Object Detection* API menjadi file "*label.txt*" yang sesuai dengan apa yang dibutuhkan oleh *"object_detector.MetadataWriter".* File ini berisi satu nama kelas pada setiap baris sampai semua kelas tertulis. Tahap selanjutnya adalah menulis *metadata* tersebut sekaligus melakukan normalisasi seperti pada baris 20 hingga 23 lalu menyimpan model dengan *metadata* pada tempat yang sudah dideklarasikan pada baris 18.

```
from tflite_support.metadata_writers import object_detector
from tflite_support.metadata_writers import writer_utils

_ODT_LABEL_MAP_PATH = '/content/label_map.pbtxt'
_TFLITE_LABEL_PATH = "/content/tflite_label_map.txt"

category_index = label_map_util.create_category_index_from_labelmap(
    _ODT_LABEL_MAP_PATH)
f = open(_TFLITE_LABEL_PATH, 'w')
for class_id in range(1, 6):
  if class_id not in category_index:
    f.write('???\n')
    continue
  name = category_index[class_id]['name']
  f.write(name+'\n')
f.close()

_TFLITE_MODEL_WITH_METADATA_PATH =
"/content/drive/MyDrive/.../model.tflite"

writer = object_detector.MetadataWriter.create_for_inference(
    writer_utils.load_file(_TFLITE_MODEL_PATH),
input_norm_mean=[127.5],
    input_norm_std=[127.5], label_file_paths=[_TFLITE_LABEL_PATH])
writer_utils.save_file(writer.populate(),
_TFLITE_MODEL_WITH_METADATA_PATH)
```

Gambar 3.33 Sintaks menambah *metadata*

TensorFlow Lite Model Maker

Proses ekspor model menggunakan pustaka ini lebih sederhana karena sudah ada *method* yang langsung merubah model menjadi file tflite yang kompatibel dengan perangkat seluler. Tahapan yang perlu dilakukan hanya satu, yaitu menjalankan baris kode yang terdapat pada gambar Gambar 3.34 dengan mendefinisikan parameter lokasi penyimpanan model dan nama filenya.

```
model.export(export_dir='/content',
             tflite_filename='efficientdet_lite0.tflite')
```

Gambar 3.34 Baris kode untuk mengekspor model

Besar ukuran model setelah proses ekspor untuk model *EfficientDet-Lite0, EfficientDet-Lite3, MobileNet,* dan *ResNet* dapat dilihat pada Tabel 3.5. *EfficientDet-Lite0* dan *EfficientDet-Lite3* tidak memiliki *saved model* karena *library TensorFlow Lite Model Maker* mengekspor model tersebut langsung dalam format tflite. Sedangkan pada *ResNet* tidak memiliki tflite karna model tersebut tidak dikonversi. Ukuran tflite lebih kecil dibanding *saved model* karena format

tflite dioptimalisasi untuk perangkat bergerak yang mengutamakan efisiensi. *ResNet* memiliki terbesar, yaitu 281 MB karena model tersebut memiliki *layers* yang sangat dalam, yaitu 50 *layers*. *MobileNet* mengalami penurunan ukuran sebesar 48.7 MB setelah dikonversi menjadi tflite yang sebelumnya memiliki ukuran 51.9 MB dalam format *saved model* menjadi 3.2 MB dalam format tflite. Hal ini terjadi karena terdapat optimalisasi ukuran dan latensi model ketika dikonversi. *EfficientDet-Lite3* memiliki 10 *layers* dan *EfficientDet-Lite0* memiliki 6 *layers*, berbanding lurus dengan ukuran model tersebut yaitu 11.1 MB untuk *EfficientDet-Lite3* dan 4.2 MB untuk *EfficientDet-Lite0*. *MobileNet* memiliki 11 *layers* tetapi memiliki ukuran yang lebih kecil dibanding *EfficientDet-Lite0*, hal ini mungkin terjadi karena fitur *depthwise separable convolutions* yang dimiliki oleh *MobileNet.*

Tabel 3.5 Ukuran model

Nama Model	Ukuran
EfficientDet-Lite0	4.2 MB (tflite)
EfficientDet-Lite3	11.1 MB (tflite)
MobileNet	51.9 MB (*saved model*) dan 3.2 MB (tflite)
ResNet	281 MB (*saved model*)

3.5 Implementasi dan Inferensi

Selama tahap ini, proses yang dilakukan adalah membangun aplikasi menggunakan IDE (*Integrated Development Environment*) yang biasa digunakan untuk pengembangan aplikasi. Dalam penelitian ini, Android Studio Chipmunk 2021.2.1 dan bahasa pemrograman Kotlin 212-1.7.0-release-281-AS5457.46 akan digunakan untuk mengembangkan aplikasi berbasis Android menggunakan IDE resmi Android Studio. Nantinya, inferensi model dapat dilakukan menggunakan perangkat seluler setelah aplikasi selesai dibuat.

Proses pemrograman perangkat seluler menggunakan bantuan *library* TensorFlow dan kode sumber dari *tensorflow examples*. Contoh kode tersebut dapat diakses pada laman GitHub *"https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/android"*. Langkah untuk mempersiapkan aplikasi ini dapat dilihat pada bab 4.3. Dalam aplikasi ini, proses deteksi dilakukan dengan mengambil data kamera lalu menjalankan deteksi dan langsung menampilkannya pada layar perangkat seluler. Proses deteksi ini berlangsung tanpa henti selama aplikasinya sedang digunakan.

Aplikasi ini kemudian memodifikasi format dan dimensi gambar input sehingga model deteksi objek dapat memprosesnya. Model kemudian akan menjalankan prosedur deteksi dan menghasilkan koordinat *bounding-box*, nama kelas, dan tingkat *confidence.* Program akan menampilkan tayangan langsung kamera dan menggambar *bounding-box*, nama kelas, dan tingkat *confidence* di atas tampilan kamera. Selain itu program ini dapat menampilkan waktu inferensi dan *frame rate*. Dalam penelitian ini, *frame rate* yang dimaksud adalah jumlah deteksi per detik.

3.6 Evaluasi

Evaluasi dilakukan untuk menilai kinerja model secara keseluruhan. Matriks penilaian yang digunakan adalah matriks *Average Precision*. Persamaan (2.1), (2.2), dan (2.3) menunjukkan rumus untuk AP, presisi, dan *recall*. Kemudian, dalam desain model ini, penilaian AP menggunakan standar COCO, sehingga nilai ambang batas *Intersection of Union* secara otomatis diatur ke 0,5. Gambar 3.35 menggambarkan baris kode yang akan dieksekusi untuk memulai evaluasi model pada metode *TensorFlow Lite Model Maker*. Ketika kode pada baris pertama dijalankan, sistem akan menentukan angka *Average Precision* secara otomatis.

```
1 model.evaluate(test_data)
```

Gambar 3.35 Sintaks evaluasi TF Lite Model Maker

Pada metode *Object Detection API,* tahapan yang perlu dilakukan adalah menjalankan *script* "*model_main_tf2.py"* beserta mengisi *flag* tempat direktori model, file konfigurasi, dan *checkpoint* seperti pada sintaks yang terdapat pada Gambar 3.36.

```
1 APIMODEL_PATH =
  "/TensorFlow/models/research/object_detection/model_main_tf2.py"
2 MODEL_PATH= "/content/drive/MyDrive/.../inference/saved_model"
3 CONFIG_PATH = "/content/drive/MyDrive/.../inference/pipeline.config"
4 CHECKPOINT_PATH= '/content/drive/MyDrive/.../output_training/'
5 !python {APIMODEL_PATH} --model_dir={MODEL_PATH} --
  pipeline_config_path={CONFIG_PATH} --checkpoint_dir={CHECKPOINT_PATH}
```

Gambar 3.36 Sintaks evaluasi *Object Detection API*

Selama proses penghitungan, sistem akan melakukan pengujian dengan memanfaatkan data *test*. Data *test* ini terdiri dari foto kelima kelas yang tidak termasuk dalam proses pelatihan.

Selain menggunakan evaluasi pada data test, proses evaluasi juga dilakukan dengan melakukan pengujian dengan mendeteksi setiap kelas dari ketiga jarak, yaitu *close-up* (35cm), sedang (100 cm), dan jauh (150 cm).

BAB IV
HASIL DAN PEMBAHASAN

4.1 Hasil Nilai *loss*

Ada sebanyak empat model telah dilatih sebagai hasil dari proses pemodelan yang telah diselesaikan menggunakan empat arsitektur yang berbeda. Selama proses pemodelan, keempat model dilatih menggunakan dataset yang sama namun dengan *hyperparameter* yang berbeda sehingga setiap model memiliki hasil yang lebih optimal. *Hyperparameter* yang berbeda saat pelatihan adalah *MobileNet* menggunakan 30000 *steps*, *ResNet* menggunakan 20000 *steps*, dan merubah *path checkpoint* sesuai pada Tabel 3.4. *TensorFlow Lite Model Maker* yang berbeda hanya *epochs* yaitu 25 *epochs* pada *EfficientDet-Lite0* dan 35 *epochs* pada *EfficientDet-Lite3.* Tabel 4.1 menyajikan representasi dari hasil perbandingan nilai *loss training* untuk masing-masing model. Nilai *loss* dapat menjadi tolak ukur pertama apakah sebuah model mengalami *overfitting* atau tidak.

Terlihat pada Tabel 4.1 bahwa selisih dari *loss training* dan *loss validasi* terbesar dimiliki oleh model yang dibuat menggunakan metode *Object Detection* API, metode ini menggunakan banyak step yaitu sebesar 20000 *steps* pada *ResNet* dan 30000 *steps* pada *MobileNet* sedangkan metode *TFLite Model Maker* hanya 25 *epoch* pada *EfficientDet-Lite0* dan 35 *epoch* pada *EfficientDet-Lite3* dengan masing-masing *epoch* menjalankan 159 *steps* yang berarti keseluruhan proses *training* pada metode *TFLite Model Maker* hanya menjalankan 3975 *steps* pada *EfficientDet-Lite0* dan 5565 *steps* pada *EfficientDet-Lite3*. Perbedaan lama dari proses *training* ini membuat model dari *Object Detection* API memiliki nilai *loss training* yang lebih kecil karena model terus mencari nilai *loss* minimal dari semua gambar pada dataset. Meski begitu, ketika menggunakan data validasi, keempat model tidak memiliki selisih yang jauh, misalnya pada *EfficientDet-Lite*3 yang memiliki nilai *loss* validasi terendah yaitu 0.4271 dan *MobileNet* yang memiliki nilai *loss* validasi tertinggi yaitu 0.5005. Kedua model tersebut hanya memiliki selisih nilai *loss* 0.0734. Hal ini bisa jadi menunjukan bahwa model dari *object detection* API mengalami *overfitting*. Model dengan metode TFLite Model Maker memiliki selisih nilai *loss training* dan *loss* validasi yang kecil sehingga dapat dikatakan bahwa kedua model ini lebih kecil kemungkinan mengalami *overfitting*. Diantara kedua model dari *TFLite Model Maker*, *EfficientDet-Lite3* merupakan model dengan nilai *loss* terbaik karena model

tersebut memiliki nilai *loss training* dan *loss* validasi yang lebih rendah dibanding *EfficientDet-Lite0*.

Tabel 4.1 Perbandingan nilai *loss*

	EfficientDet-Lite0	*EfficientDet-Lite3*	*SSD MobileNet V2 FPNLite 320x320*	*SSD ResNet50 V1 FPN 640x640 (RetinaNet50)*
Loss training	0.3329	0.2793	0.1329	0.1384
Loss validasi	0.4774	0.4271	0.5005	0.4406

4.2 Hasil Skor *Average Precision* (AP) dan recall

Empat model yang selesai dilatih akan diekspor ke sebuah model yang dioptimalkan untuk perangkat seluler menggunakan *framework tensorflow lite* dan disimpan dengan ekstensi file tflite, baik menggunakan *Object Detection* API maupun menggunakan *TF Lite Model Maker*. Model tersebut kemudian diuji menggunakan matriks evaluasi AP COCO pada data tes. Nilai presisi yang dihasilkan dari penilaian ini dapat digunakan sebagai dasar untuk menilai kinerja setiap model.

Tabel 4.2 Perbandingan nilai AP

	EfficientDet-Lite0	*EfficientDet-Lite3*	*SSD MobileNet V2 FPNLite 320x320*	*SSD ResNet50 V1 FPN 640x640 (RetinaNet50)*
Average Precision	70.43%	75.84%	72.4%	71.11%
Recall	80.9%	82.25%	77.9%	78.4%

AP pada Tabel 4.2 didapatkan sebelum model dirubah menjadi format tflite, setelah dirubah, AP pada model dapat menurun karena proses *quantization* yang membuat kebutuhan komputasi model menjadi ringan tetapi dengan menurunkan nilai akurasi. Nilai AP tertinggi terdapat pada model dengan arsitektur *EfficientDet-Lite3*, hal ini dikarenakan arsitektur tersebut memiliki ukuran masukan yang besar yaitu 512x512 dan tingkat kompleksitas model yang lebih tinggi dibanding *EfficientDet-Lite0*. Model dengan arsitektur *SSD ResNet50* memiliki ukuran masukan yang lebih tinggi dan arsitektur yang lebih mendalam dibanding dengan *EfficientDet-Lite3* dan *SSD MobileNet V2* tetapi memiliki nilai AP yang lebih rendah,

hal ini dapat menjadi indikasi bahwa model tersebut mengalami *overfitting*. *EfficientDet-Lite0* memiliki nilai AP terendah diantara kesemua model, hal tersebut dapat dipahami karena model tersebut memiliki ukuran masukan yang terkecil dan memiliki jumlah lapisan yang tidak begitu dalam dibanding *SSD ResNet50* dan *MobileNet*. Sesuai nama modelnya, *SSD MobileNet V2 FPNLite 320x320* memiliki ukuran masukan 320x320 piksel dengan kedalaman 11 *layers* seperti pada Gambar 2.4. Ukuran input dan kedalaman lapisan pada *SSD ResNet50 V1 FPN 640x640* piksel terdapat pada nama model itu sendiri yaitu 50 *layers* dengan ukuran input 640x640. Sedangkan ukuran input dan jumlah *layers EfficientDet-Lite0* adalah 320x320 piksel dengan kedalaman 6 *layers* dan *EfficientDet-Lite3* adalah 512x512 piksel dengan kedalaman 10 *layers* sesuai dengan Tabel 2.1.

Jika dilihat pada nilai *recall, EfficientDet-Lite3* memiliki nilai tertinggi diikuti dengan *EfficientDet-Lite0, ResNet50*, dan *MobileNet*. Memiliki nilai *recall* yang tinggi berarti model berhasil mendeteksi objek pada suatu kelas dengan benar dari keseluruhan objek yang ada pada kelas tersebut. Performa *EfficientDet-Lite3* berbanding lurus dengan hasil AP yang memiliki hasil terbaik diantara ketiga model lainnya. Sedangkan *EfficientDet-Lite0* yang memiliki arsitektur sederhana bahkan mendapat hasil yang lebih baik dibanding model dari metode *Object Detection API*. Hal ini mungkin dapat disebabkan oleh selisih nilai *loss* yang besar pada metode *Object Detection* API.

Selain mengukur AP secara keseluruhan, pada metode *Tensorflow Lite Model Maker* pengukuran AP juga disertai pengukuran AP untuk setiap kelasnya sehingga model dengan arsitektur *EfficientDet-Lite0* dan *EfficientDet-Lite3* dapat diketahui nilai AP per kelas. Tabel 4.3 Menunjukan nilai AP pada setiap kelas dengan menggunakan kedua model dari metode *Tensorflow Lite Model Maker*. Kelas masker medis memiliki nilai AP terbaik diikuti dengan kelas tanpa masker. Pada *EfficientDet-Lite3* urutan nilai AP tersebut terbalik dengan *EfficientDet-Lite0,* hal tersebut dikarenakan secara umum kedua kelas tersebut memiliki variasi yang lebih memungkinkan untuk digeneralisasi dibanding kelas masker scuba, terlebih masker kain yang tidak memiliki desain umum. Perbedaan ukuran masukan juga dapat membuat model dapat melihat fitur objek yang lebih kecil, seperti pada kelas penggunaan masker salah yang diharapkan model dapat mendeteksi fitur hidung tetapi karena ukuran input *EfficientDet-Lite0* lebih kecil model kesulitan mendeteksi fitur hidung tersebut dibanding dengan *EfficientDet-Lite3*. Perbedaan kedalaman layer dapat mempengaruhi hasil deteksi dengan objek yang mirip seperti pada kelas masker kain yang beberapa variasinya menyerupai masker medis dan masker scuba karna semakin banyak lapisan maka jumlah kalkulasi untuk menentukan perbedaan

setiap kelas lebih banyak. Selisih AP pada kelas masker kain antara kedua model juga cukup jauh.

Tabel 4.3 Perbandingan nilai AP pada setiap kelas

No	Nama kelas	*EfficientDet-Lite0*	*EfficientDet-Lite3*
1	Masker medis	74.61%	82.76%
2	Masker kain	59.35%	68.72%
3	Masker scuba	73.23%	76.71%
4	Penggunaan masker salah	65.01%	72.59%
5	Tanpa masker	79.95%	78.44%

4.3 Implementasi Perangkat Seluler

Peneliti menggunakan model dengan ekstensi file tflite yang diekspor untuk perangkat seluler, file yang kompatibel dengan aplikasi seluler. Dalam hal ini, peneliti membuat aplikasi berbasis Android. Oleh karena itu, pembuatan kode dilakukan menggunakan *integrated development environment* (IDE) Android Studio dengan versi Chipmunk 2021.2.1 yang menggunakan bahasa pemrograman Kotlin dengan versi 212-1.7.0-release-281-AS5457.46. Menggunakan bantuan *library* tensorflow dan kode sumber dari *tensorflow examples*. Berikut ini adalah langkah-langkah yang diperlukan untuk membangun aplikasi dan melakukan prosedur deteksi:

1. Langkah pertama adalah mengunduh kode sumber aplikasi android yang telah disediakan oleh *tensorflow examples* yang terdapat pada folder GitHub dengan link *"https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/android"*.
2. Tahapan selanjutnya adalah menaruh model pada folder *assets* serta mengganti beberapa variabel untuk memanggil model tersebut dengan menggunakan sintaks pada gambar Gambar 4.1 yang nantinya keempat model tersebut dapat dipilih ketika program sedang dijalankan. Sintaks tersebut terdapat pada file "*ObjectDetectorHelper.kt".* Mengganti nama variabel dilakukan dengan menggunakan *refactor*.

```
val modelName =
            when (currentModel) {
                MODEL_EFFICIENTDETV0 ->
"efficientdet_lite0.tflite"
                MODEL_EFFICIENTDETV3 ->
"efficientdet_lite3.tflite"
                MODEL_MOBILENETV2 -> "MobileNetV2.tflite"
                else -> "mobilenetv1.tflite"
            }
```

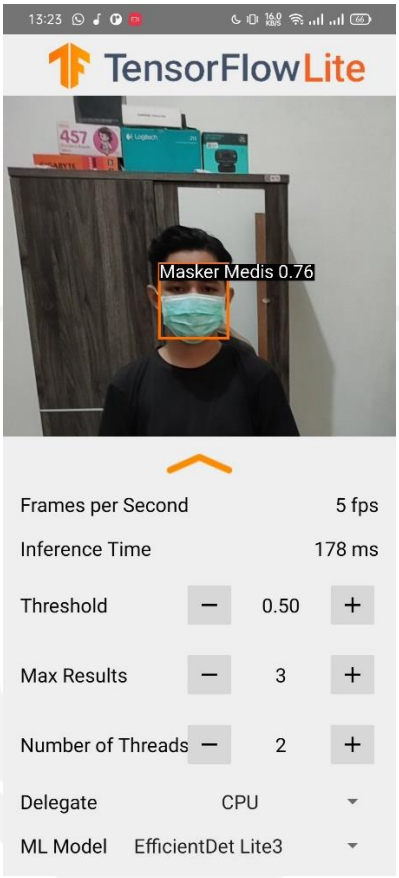
Gambar 4.1 Sintaks memanggil model

3. Menyesuaikan tampilan nama model pada *dropdown menu/spinner* pilihan model sehingga memudahkan mengganti antara model dengan memilih model tersebut pada *user interface*. Perubahan nama tampilan ini terdapat pada file "*strings.xml"* yang berisi semua tulisan pada aplikasi ini. Gambar 4.2 adalah sintaks untuk menulis nama model pada *user interface*, urutannya menyesuaikan dengan pengambilan model seperti pada Gambar 4.1.

```
<string-array name="models_spinner_titles">
        <item>EfficientDet Lite0</item>
        <item>EfficientDet Lite3</item>
        <item>MobileNet V2</item>
    </string-array>
```

Gambar 4.2 Sintaks tampilan nama model

Sintaks untuk melakukan deteksi, menggambar *bounding box,* pemberian label, waktu inferensi, dan tampilan *user interface* sudah terdapat pada kode sumber dan tidak perlu dilakukan penyuntingan. Pada Gambar 4.3 terdapat variabel seperti *fps, inference time, threshold, max results, number thread, delegate,* dan *ML model. Inference time* adalah waktu yang dibutuhkan untuk melakukan satu deteksi. Waktu yang dibutuhkan bervariasi tergantung dari kemampuan komputasi perangkat seluler dan tingkat kompleksitas model. *Fps* adalah jumlah gambar yang ditampilkan dalam satu detik yang didapat dari membagi 1000 ms dengan waktu inferensi. *Threshold* adalah ambang batas ketika sebuah hasil deteksi akan dianggap terdeteksi jika nilai *confidence* objek tersebut di atas nilai *threshold*. *Max result* adalah jumlah maksimal deteksi dalam satu waktu. *Number thread* adalah jumlah proses bersamaan yang dapat dilakukan dalam satu waktu sehingga semakin besar nilainya maka semakin kecil waktu inferensinya, begitu pula sebaliknya. Menu *delegate* memungkinkan untuk memilih tempat pemrosesan terjadi seperti CPU, GPU, atau NNAPI. Terakhir adalah ML *Model* yang merupakan pilihan model untuk menjalankan deteksi.



Gambar 4.3 Tampilan aplikasi deteksi objek

4.4 Hasil Pengujian Perangkat Seluler

Pengukuran menggunakan perangkat seluler Realme X2 PRO yang memiliki *chipset SnapDragon 855+*. Pengujian dilakukan di dalam ruangan dengan pencahayaan cukup. Proses pengujian dilakukan dengan menjalankan aplikasi dan melihat apakah model berhasil mendeteksi objek dengan jarak 35 cm (*close-up)*, 100 cm (sedang), 150 cm (jauh). Berdasarkan forum diskusi GitHub *"https://github.com/tensorflow/models/issues/9287"*, model *SSD ResNet* memiliki masalah ketika dirubah ke tflite, maka model *ResNet* tidak diimplementasikan pada perangkat seluler. Selain itu, pada Tabel 4.7, model *ResNet* memiliki waktu inferensi lebih lambat dibanding *MobileNet* serta *MobileNet* memiliki nilai AP yang lebih tinggi dibanding *ResNet.* Karena alasan ini, hanya *MobileNet* yang dikonversi menjadi tflite agar dapat dilakukan pengujian. Selain itu, *EfficientDet-Lite* sudah diekspor dalam format tflite sehingga kedua model dapat diimplementasikan pada perangkat seluler.

Tabel 4.4 Perbandingan hasil pengujian

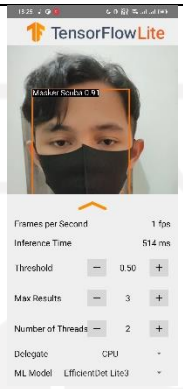
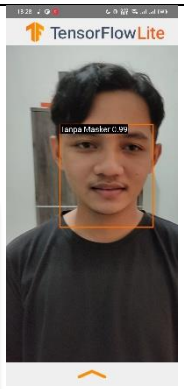
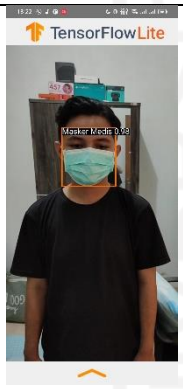
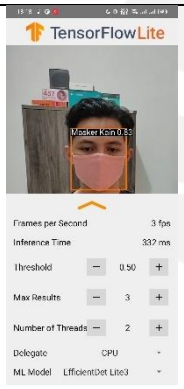
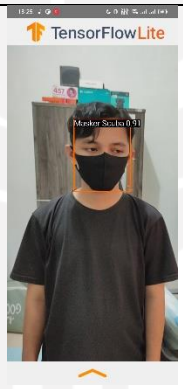
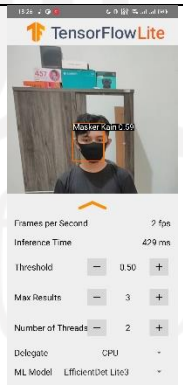
Kelas	Jarak	*EfficientDet-Lite0*	*EfficientDet-Lite3*	*SSD MobileNet V2 FPNLite 320x320*
Masker medis	*close-up*	Benar	Benar	Benar
	sedang	Benar	Benar	Benar
	jauh	Benar	Benar	Benar
Masker kain	*close-up*	Benar	Benar	Benar
	sedang	Salah deteksi	Benar	Salah deteksi
	jauh	Salah deteksi	Benar	Salah deteksi
Masker scuba	*close-up*	Benar	Benar	Benar
	sedang	Benar	Benar	Benar
	jauh	Benar	Salah deteksi	Salah deteksi
Penggunaan masker salah	*close-up*	Salah deteksi	Benar	Benar
	sedang	Benar	Salah deteksi	Salah deteksi
	jauh	Salah deteksi	Salah deteksi	Salah deteksi
Tidak mengenakan masker	*close-up*	Benar	Benar	Benar
	sedang	Benar	Benar	Benar
	jauh	Benar	Benar	Benar

Hasil pengujian pada Tabel 4.4 memiliki berbagai status hasil pengujian, benar berarti model berhasil mendeteksi objek (*True Positive*) dan salah deteksi berarti model mendeteksi objek ke kelas yang tidak seharusnya (*False Positive)*. Selain *True Positive* dan *False Positive*, ada kemungkinan model tidak dapat mendeteksi objek sama sekali (*False Negative)*. Pada pengujian ini, nilai *False Negative* adalah 0 karena pada pengujian ini semua objek berhasil dideteksi walau dilabeli dengan kelas yang salah. Pengujian ini bersifat *real time,* sehingga nilai hasil pengujian ini diambil berdasarkan hasil deteksi terbanyak/modus untuk setiap skenarionya. Model dengan arsitektur *EfficientDet-Lite3* mendapat hasil terbaik dibanding *MobileNet* dan *EfficientDet-Lite0* karena *EfficientDet-Lite3* memiliki 12 dari 15 hasil benar dengan status pengujian 12 *True Positive* dan 3 *False Positive*. Hal ini dikarenakan tingkat kompleksitas dan ukuran masukan gambar yang relatif lebih besar dibanding *MobileNet* dan *EfficientDet-Lite0*. Sedangkan *EfficientDet-Lite0* memiliki 11 hasil benar dengan status pengujian 11 *True Positive* dan 4 *False Positive*, sedikit lebih baik dibanding *MobileNet* yang

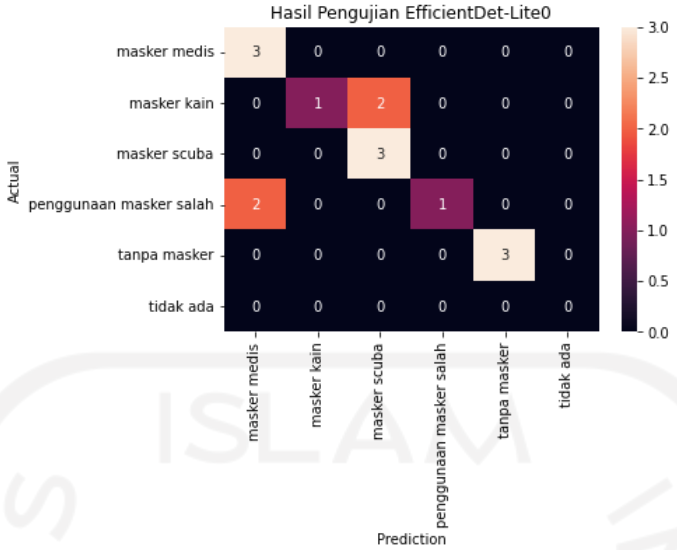
memiliki 10 hasil benar dengan status pengujian 10 *True Positive* dan 5 *False Positive*. Kedua model ini memiliki ukuran masukan serupa namun memiliki arsitektur berbeda. Urutan nilai *loss* validasi pada *EfficientDet-Lite0* dan *MobileNet* juga berbanding lurus dengan hasil pengujian ini tetapi nilai AP pada *efficientDet-Lite0* lebih rendah dibanding *MobileNet*. *MobileNet* memberikan kinerja yang lebih bagus dibandingkan *EffieceinDet-Lite0* saat diuji menggunakan data test. *MobileNet* dapat menebak benar masker scuba dengan warna selain warna hitam lebih banyak dibandingkan *EfficientDet-Lite0*. Hal ini dikarenakan pada data test, model yang digunakan bertipe *saved model* yang memiliki ukuran 51.9 MB sedangkan tipe tflite yang digunakan pada pengujian ini hanya berukuran 3.2 MB. Sedangkan *EfficientDet-Lite0* lebih bagus dibandingkan *MobileNet* saat diuji secara *real time* mengunakan perangkat seluler. *EfficientDet-Lite0* dapat menebak dengan benar masker scuba berwarna hitam lebih banyak daripada *MobileNet* dengan jumlah data yang sama. Hal ini dapat disebabkan karena *EfficientDet-Lite0* memiliki ukuran 4.2 MB, ukuran yang lebih besar dibanding *MobileNet* dalam bentuk tflite.

Tabel 4.5 merupakan contoh hasil deteksi kelima kelas dengan tiga jarak, hasil deteksi tersebut dihasilkan oleh *efficientDet-Lite3* yang merupakan model terbaik pada pengujian ini. Masker medis dan masker scuba menggunakan masker dengan desain yang umum dijumpai, seperti masker medis warna hijau yang biasa digunakan oleh dokter di rumah sakit dan masker scuba warna hitam yang banyak ditemui di pasaran. Penguunan masker salah dalam pengujian ini adalah masker medis yang menutupi mulut tetapi tidak menutupi hidung. Masker kain yang digunakan masker dengan bahan dasar *cotton,* warna *cream.* dan tanpa lipatan, sedangkan tanpa masker adalah wajah manusia.

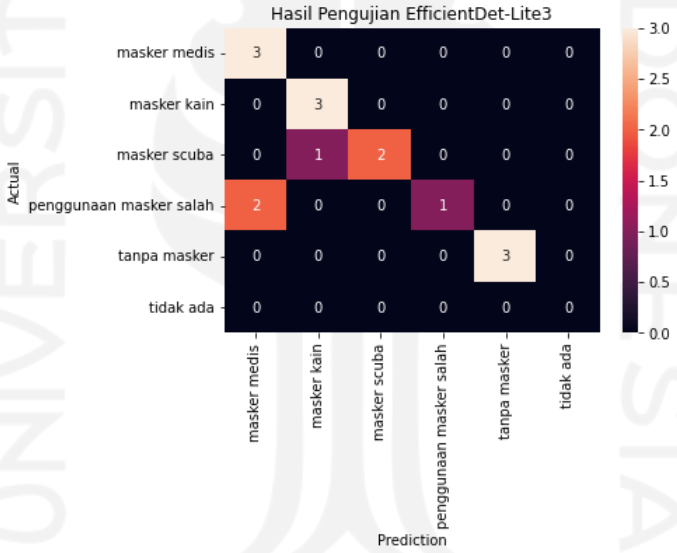
Tabel 4.5 Hasil deteksi kelima kelas dengan tiga jarak

	Masker medis	**Masker kain**	**Masker scuba**	**Penggunaan masker salah**	**Tanpa masker**
35 cm (*close-up*)					
100 cm (sedang)					
150 cm (jauh)					

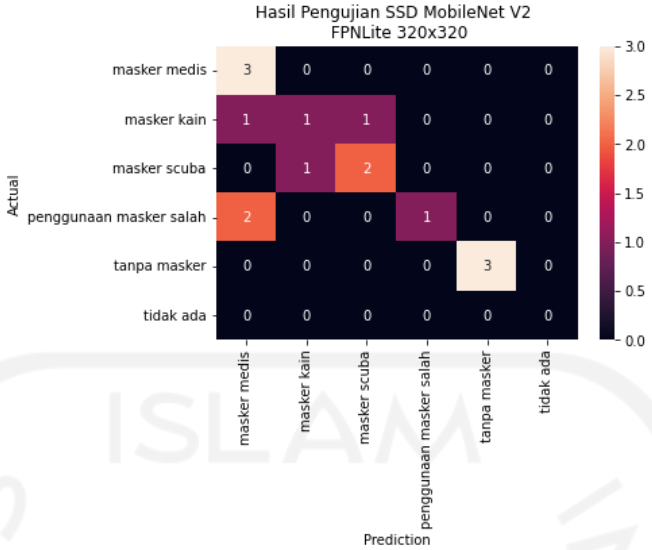
Hasil pengujian pada Tabel 4.4 dapat dibuatkan *confusion matrix* pada model *efficientDet-Lite0*, *efficientDet-Lite3*, dan *MobileNet*. Nantinya, *confusion matrix* dapat menjadi tolak ukur untuk melihat di mana kesalahan dalam pengujian ini terjadi



Gambar 4.4 *Confusion matrix* pengujian *EfficientDet-Lite0*



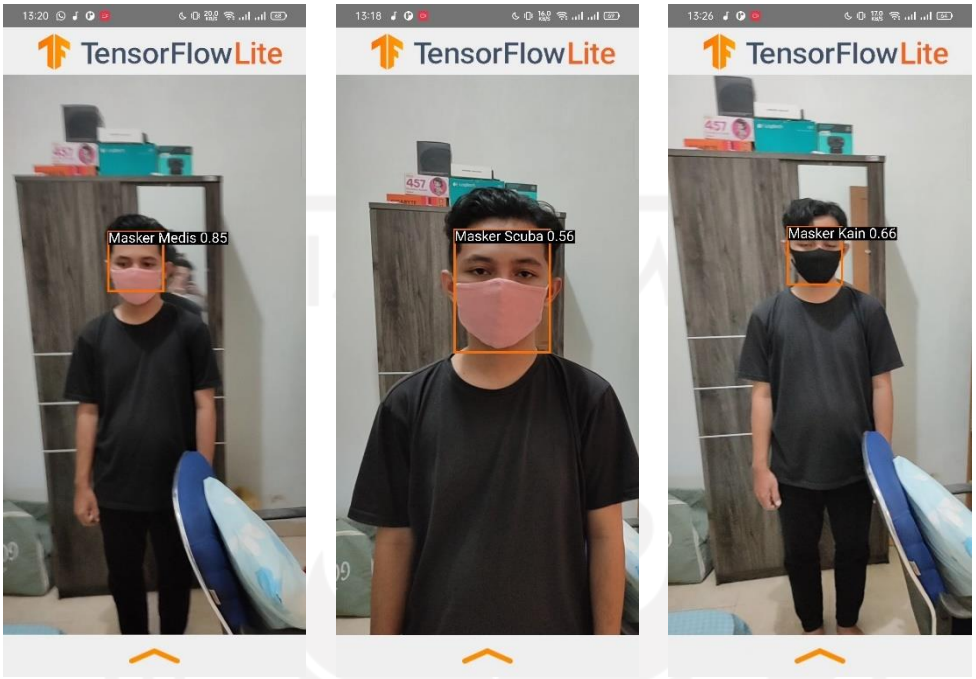
Gambar 4.5 *Confusion matrix* pengujian *EfficientDet-Lite3*



Gambar 4.6 *Confusion matrix* pengujian *SSD MobileNet V2 FPNLite 320x320*

Pada Gambar 4.4, Gambar 4.5, dan Gambar 4.6 kesalahan terbesar dimiliki oleh kelas penggunaan masker salah yang terdeteksi sebagai masker medis. Kesalahan ini dapat terjadi karena fitur hidung yang sulit terdeteksi. Dalam jarak dekat dan sedang terdapat model yang berhasil mendeteksi kelas tersebut di mana fitur hidung dapat terlihat lebih jelas, namun ketika jarak dari kamera menjauh menyebabkan fitur hidung manusia lebih sulit dideteksi dibanding fitur masker medis yang ukurannya lebih besar, seperti yang terdapat pada Tabel 4.5 pada kolom penggunaan masker salah jarak sedang dan jauh. Kesalahan lainnya adalah masker kain yang terdeteksi sebagai masker scuba dan masker medis, serta masker scuba yang terdeteksi sebagai masker kain seperti pada Gambar 4.7 di mana kiri gambar adalah masker kain yang terdeteksi sebagai masker medis, tengah gambar adalah masker kain terdeteksi sebagai masker scuba, dan kanan gambar adalah masker scuba terdeteksi sebagai masker kain. Hal ini dapat terjadi karena banyaknya variasi serta kemiripan fitur antara kelas masker medis, masker kain, dan masker scuba pada data *train* yang membuat tingkat *confidence* model menurun dan hasil deteksi tidak stabil. Kelas masker medis dan tanpa masker merupakan kelas dengan tingkat keberhasilan tertinggi karena dapat mendeteksi semua objek untuk kelas tersebut disetiap skenario pengujian dengan ketiga model, contoh deteksi kedua kelas tersebut dapat dilihat pada Tabel 4.5 kolom masker medis dan tanpa masker. Beberapa jenis masker medis cukup banyak digunakan oleh masyarakat sehingga peluang memiliki jenis masker medis yang sama pada data *train* dengan yang digunakan saat pengujian lebih tinggi. Sedangkan tanpa masker dapat

dibilang mendeteksi wajah manusia tanpa halangan di mana fitur wajah berbeda-beda tapi cenderung serupa dan jauh berbeda dengan keempat kelas lainnya.



Gambar 4.7 Contoh salah deteksi

Tabel 4.6 Waktu inferensi model pada perangkat bergerak

Model	**Waktu Inferensi**
EfficientDet-Lite0	70-80 ms
EfficientDet-Lite3	290-533 ms
SSD MobileNet V2 FPNLite 320x320	110-120 ms

Waktu inferensi pada Tabel 4.6 berbanding lurus dengan kompleksitas arsitektur dan ukuran masukan model, yaitu semakin kompleks dan besar ukuran model maka semakin lama waktu yang dibutuhkan untuk menjalankan inferensi. *EfficientDet-Lite0* memilki ukuran input 320x320 sama seperti *MobileNet* tetapi hanya memilki 6 *layers* tidak seperti *MobileNet* yang memilki 11 *layers* sehingga *EfficientDet-Lite0* memiliki latensi terendah. Walaupun *EfficientDet-Lite3* memiliki hanya memiliki 10 *layers*, model tersebut memiliki waktu inferensi yang lebih lambat karena ukuran input yang lebih besar, yaitu 512x512.

Tabel 4.7 Waktu memuat model dan inferensi pada *Google Colab*

Model	**Waktu memuat model dan inferensi**
EfficientDet-Lite0	4.257 detik (tflite)
EfficientDet-Lite3	21.193 detik (tflite)
SSD MobileNet V2 FPNLite 320x320	18.123 detik (model awal) dan 2.834 detik (*tflite*)
SSD ResNet50 V1 FPN 640x640 (RetinaNet50)	21.521 detik (model awal)

Waktu inferensi pada Tabel 4.7 menunjukan bahwa dengan menggunakan komputer, masih membutuhkan waktu yang lebih lama dibanding dengan menggunakan perangkat seluler. Hal ini karena *runtime* memerlukan waktu untuk memuat model. Model awal adalah model yang masih tersimpan dalam bentuk *saved model* dan inferensi menggunakan metode *Object Detection API*, sedangkan model tflite adalah model yang sudah dirubah menjadi model yang kompatibel dengan perangkat seluler. *MobileNet* mengalami penurunan waktu yang signifikan setelah dikonversi, yaitu sebesar 15.289 detik dari yang sebelumnya 18.123 detik menjadi 2,834 detik, hal ini menunjukan bahwa *MobileNet* sangat cocok untuk perangkat seluler. *MobileNet* (*tflite*) juga memiliki waktu memuat model dan inferensi selama 2.834 detik, sedangkan *EfficientDet-Lite0* membutuhkan waktu 4.257 detik, yang berarti waktu memuat model dan inferensi *MobileNet* (*tflite*) 1.423 detik lebih singkat dibanding *EfficientDet-Lite0*, berbanding terbalik dengan waktu inferensi perangkat seluler di mana *EfficientDet-Lite0* membutuhkan waktu 70-80 ms sedangkan *MobileNet* (*tflite*) membutuhkan waktu 110-120 ms. *EfficientDet-Lite0* mendeteksi *40* ms lebih cepat dibanding *MobileNet (tflite).* Hal ini mungkin terjadi karena *layer MobileNet* lebih banyak, yaitu *11 layer,* daripada *EfficientDet-Lite0* yang hanya memiliki 6 *layer*. Namun, saat melakukan inferensi di *Google Colab*, waktu memuat model dan inferensi *EfficientDet-Lite0* lebih lama dibandingkan *MobileNet*. Hal ini mungkin disebabkan oleh waktu untuk memuat model *EfficientDet-Lite0* lebih lama dibandingkan *MobileNet. EfficientDet-Lite3* membutuhkan 21.193 detik untuk memuat dan inferensi, waktu yang cukup lama dibanding *EfficientDet-Lite0* dan *MobileNet,* sebanding dengan waktu inferensi ketika menjalankan pengujian pada perangkat seluler yang membutuhkan 290-533 ms per deteksi.

BAB V
PENUTUP

5.1 Kesimpulan

Berdasarkan hasil pengujian deteksi objek masker menggunakan berbagai dataset dan arsitektur, dapat ditarik kesimpulan berupa:

a. Penelitian ini dibuat untuk melawan penyebaran penyakit menular dengan menggunakan model *EfficientDet-Lite0, EfficientDet-Lite3, SSD MobileNet V2 FPNLite 320x320, SSD ResNet50 V1 FPN 640x640 (RetinaNet50)* dengan hasil AP sebesar 70.43%, 75.84%, 72.4%, dan 71.11% berturut-turut. Pada pengujian perangkat seluler, *EfficientDet-Lite0, EfficientDet-Lite3, SSD MobileNet V2 FPNLite 320x320* berhasil mendapat sebelas jawaban benar, dua belas jawaban benar, dan sepuluh jawaban benar dengan latensi 70-80 ms, 290-310 ms, 110-120 ms berturut. *ResNet* tidak memiliki pengujian dan latensi pada perangkat seluler karena tidak diimplementasikan.
b. Model dengan arsitektur *Efficient-DetLite 3* memiliki hasil terbaik baik dalam AP maupun hasil pengujian dikarenakan model tersebut memiliki ukuran masukan yang lebih besar dan arsitektur yang relatif lebih dalam tetapi tetap mampu mendeteksi beberapa kali dalam satu detik.
c. Kesemua kelas memiliki variasi tak terbatas, tetapi kelas masker medis dan tanpa masker memiliki beberapa variasi yang umum sehingga kemungkinan model dapat mendeteksi dengan benar pada kedua kelas tersebut lebih tinggi. Kelas masker scuba dan masker kain tidak ada variasi yang umum sehingga model sulit menggeneralisasi kelas tersebut sehingga kelas tersebut memiliki akurasi yang lebih rendah pada pengujian atau nilai AP per kelas. Kelas penggunaan masker kebanyakan mengandalkan dataset *MaskedFace-Net* sehingga variasi kelas tersebut kurang banyak dan cukup berpengaruh pada hasil pengujian perangkat seluler.

5.2 Saran

Untuk meningkatkan performa deteksi objek, peneliti berharap pada pengembangan penelitian berikutnya untuk:

a. Menambah variasi pada kelas penggunaan masker salah karena kelas tersebut memiliki *false positive* terbesar. Akan lebih baik lagi jika menambah variasi dan jumlah dataset untuk ke semua kelas untuk membuat tingkat *confidence* model meningkat.

b. Mengatur *hyperparameter* lebih terperinci karena pengaturan *hyperparameter* untuk penelitian ini hanya *epoch/steps, batch,* dan *path* menuju dataset.
c. Menggunakan jumlah *batch* yang lebih besar agar lebih banyak sampel yang digeneralisasi pada satu iterasi.

DAFTAR PUSTAKA

Cabani, A., Hammoudi, K., Benhabiles, H., & Melkemi, M. (2021). MaskedFace-Net – A dataset of correctly/incorrectly masked face images in the context of COVID-19. *Smart Health*, *19*. https://doi.org/10.1016/j.smhl.2020.100144

De, S., & Sirisuriya, S. (2015). *A Comparative Study on Web Scraping*.

Ecdc. (2022). *Considerations for the use of face masks in the community in the context of the SARS-CoV-2 Omicron variant of concern*.

Ge, S., Li, J., Ye, Q., & Luo, Z. (2017). Detecting masked faces in the wild with LLE-CNNs. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, *2017-January*. https://doi.org/10.1109/CVPR.2017.53

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition*. http://arxiv.org/abs/1512.03385

Ho, K. F., Lin, L. Y., Weng, S. P., & Chuang, K. J. (2020). Medical mask versus cotton mask for preventing respiratory droplet transmission in micro environments. *Science of the Total Environment*, *735*. https://doi.org/10.1016/j.scitotenv.2020.139510

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

Karras NVIDIA, T., & Laine NVIDIA, S. (2019). #StyleGAN - A Style-Based Generator Architecture for Generative Adversarial Networks Timo Aila NVIDIA. *Cvpr 2019*.

Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, *53*(8), 5455–5516. https://doi.org/10.1007/s10462-020-09825-6

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. In *Nature* (Vol. 521, Issue 7553, pp. 436–444). Nature Publishing Group. https://doi.org/10.1038/nature14539

Lotfi, M., Hamblin, M. R., & Rezaei, N. (2020). COVID-19: Transmission, prevention, and potential therapeutic opportunities. In *Clinica Chimica Acta* (Vol. 508, pp. 254–266). Elsevier B.V. https://doi.org/10.1016/j.cca.2020.05.044

Pannu, A. (2015). Artificial Intelligence and its Application in Different Areas. In *Certified International Journal of Engineering and Innovative Technology (IJEIT)* (Vol. 9001, Issue 10).

Pattanayak, S. (2017). Pro Deep Learning with TensorFlow. In *Pro Deep Learning with TensorFlow*. https://doi.org/10.1007/978-1-4842-3096-1

Ponti, M. A., Ribeiro, L. S. F., Nazare, T. S., Bui, T., & Collomosse, J. (2017). Everything You Wanted to Know about Deep Learning for Computer Vision but Were Afraid to Ask. *Proceedings - 2017 30th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials SIBGRAPI-T 2017, 2018-January*, 17–41. https://doi.org/10.1109/SIBGRAPI-T.2017.12

Rahman, M. M., Manik, M. M. H., Islam, M. M., Mahmud, S., & Kim, J. H. (2020, September 1). An automated system to limit COVID-19 using facial mask detection in smart city network. *IEMTRONICS 2020 - International IOT, Electronics and Mechatronics Conference, Proceedings*. https://doi.org/10.1109/IEMTRONICS51293.2020.9216386

Repak, T. (2021). *Fast object detection on mobile platforms using neural networks BACHELOR'S THESIS.*

Salawati Liza. (2012). [4] PENGENDALIAN INFEKSI NOSOKOMIAL DI RUANG ICU RUMAH SAKIT. *JURNALKEDOKTERAN SYIAH KUALA*, *12*.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (n.d.-b). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2019a). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*.

Sanjaya, S. A., & Rakhmawan, S. A. (2020, October 26). Face Mask Detection Using MobileNetV2 in the Era of COVID-19 Pandemic. *2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy, ICDABI 2020*. https://doi.org/10.1109/ICDABI51230.2020.9325631

Scarpino, M. (2018). TensorFlow For Dummies. In *Climate Change 2013 - The Physical Science Basis*.

Sethi, S., Kathuria, M., & Kaushik, T. (2021). Face mask detection using deep learning: An approach to reduce risk of Coronavirus spread. *Journal of Biomedical Informatics*, *120*. https://doi.org/10.1016/j.jbi.2021.103848

Shereen, M. A., Khan, S., Kazmi, A., Bashir, N., & Siddique, R. (2020). COVID-19 infection: Origin, transmission, and characteristics of human coronaviruses. In *Journal of Advanced Research* (Vol. 24, pp. 91–98). Elsevier B.V. https://doi.org/10.1016/j.jare.2020.03.005

Suresh, K., Palangappa, M. B., & Bhuvan, S. (2021). Face Mask Detection by using Optimistic Convolutional Neural Network. *Proceedings of the 6th International Conference on*

Inventive Computation Technologies, ICICT 2021, 1084–1089. https://doi.org/10.1109/ICICT50816.2021.9358653

Tan, M., Pang, R., & Le, Q. v. (2020). EfficientDet: Scalable and efficient object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 10778–10787. https://doi.org/10.1109/CVPR42600.2020.01079

Widodo Budiharto. (2016). *Machine learning dan computational intelligence* (Th. A. Prabawati, Ed.). Andi.

Yadav, S. (2020). Deep Learning based Safe Social Distancing and Face Mask Detection in Public Areas for COVID-19 Safety Guidelines Adherence. *International Journal for Research in Applied Science and Engineering Technology*, *8*(7), 1368–1375. https://doi.org/10.22214/ijraset.2020.30560

Zhao, Z.-Q., Zheng, P., Xu, S., & Wu, X. (2018). *Object Detection with Deep Learning: A Review*. http://arxiv.org/abs/1807.05511

Zou, Z., Shi, Z., Guo, Y., & Ye, J. (2019). *Object Detection in 20 Years: A Survey*. http://arxiv.org/abs/1905.05055

LAMPIRAN

```
model {
  ssd {
    num_classes: 5
    image_resizer {
      fixed_shape_resizer {
        height: 640
        width: 640
      }
    }
    feature_extractor {
      type: "ssd_resnet50_v1_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 0.0004
          }
        }
        initializer {
          truncated_normal_initializer {
            mean: 0.0
            stddev: 0.03
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.997
          scale: true
          epsilon: 0.001
        }
      }
      override_base_feature_extractor_hyperparams: true
      fpn {
        min_level: 3
        max_level: 7
      }
    }
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
```

```
53          force_match_for_each_row: true
54          use_matmul_gather: true
55        }
56      }
57      similarity_calculator {
58        iou_similarity {
59        }
60      }
61      box_predictor {
62        weight_shared_convolutional_box_predictor {
63          conv_hyperparams {
64            regularizer {
65              l2_regularizer {
66                weight: 0.0004
67              }
68            }
69            initializer {
70              random_normal_initializer {
71                mean: 0.0
72                stddev: 0.01
73              }
74            }
75            activation: RELU_6
76            batch_norm {
77              decay: 0.997
78              scale: true
79              epsilon: 0.001
80            }
81          }
82          depth: 256
83          num_layers_before_predictor: 4
84          kernel_size: 3
85          class_prediction_bias_init: -4.6
86        }
87      }
88      anchor_generator {
89        multiscale_anchor_generator {
90          min_level: 3
91          max_level: 7
92          anchor_scale: 4.0
93          aspect_ratios: 1.0
94          aspect_ratios: 2.0
95          aspect_ratios: 0.5
96          scales_per_octave: 2
97        }
98      }
99      post_processing {
100       batch_non_max_suppression {
101         score_threshold: 1e-08
102         iou_threshold: 0.6
103         max_detections_per_class: 100
104         max_total_detections: 100
105         use_static_shapes: false
106       }
107       score_converter: SIGMOID
```

```
    }
    normalize_loss_by_num_matches: true
    loss {
      localization_loss {
        weighted_smooth_l1 {
        }
      }
      classification_loss {
        weighted_sigmoid_focal {
          gamma: 2.0
          alpha: 0.25
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
train_config {
  batch_size: 8
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.04
          total_steps: 25000
          warmup_learning_rate: 0.013333
          warmup_steps: 2000
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  fine_tune_checkpoint: "/content/last_saved/output_training/ckpt-18"
  num_steps: 20000
```

```
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  use_bfloat16: true
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  label_map_path: "/content/dataset/annotations/label_map.pbtxt"
  tf_record_input_reader {
    input_path: "/content/dataset/annotations/train.record"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "/content/dataset/annotations/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "/content/dataset/annotations/validation.record"
  }
}
```

File *Config Object Detection* API