**DAFTAR ISI BAB 4**

[**BAB VI HASIL PENELITIAN DAN PEMBAHASAN** 2](#_Toc165664113)

[4.1. Pengumpulan Data 2](#_Toc165664118)

[4.2. *Pre-processing* Data 2](#_Toc165664119)

[4.2.1. *Annotate Image* 2](#_Toc165664120)

[4.2.2. *Rescale / Resize Image* 4](#_Toc165664121)

[4.2.3. *Augmentasi Image* 4](#_Toc165664122)

[4.2.4. *Data Split* 7](#_Toc165664123)

[4.3. Modeling 8](#_Toc165664124)

[4.3.1. Menginstall YOLO 8](#_Toc165664125)

[4.3.2. Mengimpor Dataset 8](#_Toc165664126)

[4.3.3. Pelatihan Model 10](#_Toc165664127)

[4.4. Analisis Hasil Pengujian 13](#_Toc165664128)

[4.4.1. Hasil Pelatihan Model 13](#_Toc165664137)

[*4.4.2.* Uji Coba *Hyperparameter* 15](#_Toc165664138)

[4.4.3. Analisis Uji Hyperparameter 17](#_Toc165664139)

[4.5. Implementasi Aplikasi Deteksi Objek 17](#_Toc165664140)

[4.6. Pengujian Kegunaan Aplikasi 17](#_Toc165664141)

[4.8. Daftar Pustaka 17](#_Toc165664145)

# **BAB VI**

# **HASIL PENELITIAN DAN PEMBAHASAN**



## Pengumpulan Data

Data yang dikumpulkan adalah data-data *random* yang diambil dari internet. Data yang dikumpulkan ini kemudian dipilah sesuai kelas-kelas yang nantinya akan digunakan pada penelitian seperti pada gambar 3.3. Setelah terkumpul, data akan dimasukkan ke dalam projek yang sebelumnya sudah dibuat di Roboflow seperti yang dilihat pada gambar 4.1. Dengan demikian, maka data akan siap masuk pada tahap *pre-processing.*



Gambar 4. 1: Proses upload data ke Roboflow

## *Pre-processing* Data

Tahap *pre-processing* pada data akan melalui beberapa tahap seperti yang telah disebutkan pada bab 3.

### *Annotate Image*

Pada tahap ini, gambar yang sebelumnya sudah diunggah ke *workshop* Roboflow akan dilakukan *annotate* atau melabelkan objek yang terdapat di dalam gambar. Pelabelan pada sebuah gambar tidak terpaku hanya pada satu objek. Jika di dalam gambar terdapat lebih dari satu objek yang jelas, maka objek tersebut akan dilabeli. Hal ini dapat terlihat seperti pada gambar 4.2.



Gambar 4. 2: Proses annotate di Roboflow

Seperti yang dijelaskan pada bagian 3.3.1 mengenai *labeling*, label atau *bounding box* yang dibuat ini akan menyimpan informasi-informasi yang dibutuhkan algoritma untuk proses pemodelan. Adapun data yang tersimpan seperti yang terlihat pada gambar 4.3. Data tersebut memuat mengenai *class* dari objek yang dilabel dan koordinat dari objek tersebut.

Gambar 4. 3: Data hasil labeling atau bounding box

### *Rescale / Resize Image*

Setelah dilabel, gambar akan dijadikan dataset. Untuk dapat melakukan *resize* ditahap ini, terlebih dahulu mesti membuat sebuah versi dari dataset. Kemudian gambar akan diubah ukurannya sesuai dengan skenario pengujian pada tabel 3.1.



Gambar 4. 4: Proses resize pada dataset

*Resize* ini membuat semua gambar pada versi yang dipilih akan memiliki ukuran yang sama. Hal ini dilakukan agar dapat membantu meningkatkan efisiensi dalam pengolahan data dan penggunaan sumber daya komputasi pada saat pemodelan nantinya (Iryanto & Zaini, 2014; Zhao & Li, 2020).

### *Augmentasi Image*

Pada tahap ini akan dilakukan proses augmentasi seperti yang sudah dijelaskan pada bab 3.

1. 

Gambar 4. 5: Proses augmentasi dataset

* + - 1. *Grayscale*

Pada augmentasi untuk *grayscale,* akan dipilih tingkat *grayscale* sebesar 15%. Tingkatan ini dapat membuat variasi dataset yang signifikan jika dibandingkan dengan tingkatan di bawahnya juga tidak membuat gambar terlalu gelap sehingga dapat menghilangkan detail dari gambar.

* + - 1. *Exposure*



Gambar 4. 6: Proses exposure dataset

Seperti yang dijelaskan pada bab 3 di bagian 3.3.3, untuk *exposure* akan menggunakan tingkatan sebesar 15%.

* + - 1. *Rotation*



Gambar 4. 7: Proses rotation dataset

Pada tahap ini dataset akan dilakukan proses *rotation* dengan tingkatan sebesar 10%. Hal ini dilakukan agar terdapat beberapa variasi dataset yang memiliki titik yang berbeda dari gambar aslinya. *Rotation* ini akan membuat gambar memutar dengan sejumlah derajat tertentu (Perez dkk., 2018).

* + - 1. *Flip*

Proses ini nantinya akan membalikan gambar secara horizontal. Proses ini digunakan agar dataset memiliki variasi terhadap data dari beberapa sudut pandang.



Gambar 4. 8: Proses flip dataset

### *Data Split*

Pada tahap *data split, s*etelah tahapan *pre-processing* dan augmentasi dilakukan, maka secara otomatis Roboflow akan menghitung total gambar kemudian membaginya ke dalam bagiannya masing-masing. Untuk pembagiannya akan menerapkan perbandingan 70:20:10 dengan rincian 70% untuk *train,* 20% untuk *valid,* dan 10% untuk *testing.*



Gambar 4. 9: Data split oleh Roboflow

Ditahap ini dataset dengan versinya telah siap digunakan ke dalam projek yang akan dibuat.

## Modeling

Proses pembuatan model dari YOLO ini nantinya akan dibuat dengan bantuan Kaggle Notebook. Hal ini dikarenakan platform ini menyediakan akses GPU dengan *limit* 30 jam setiap minggunya. Meskipun ada batasan, fitur ini sangat menguntungkan karena mampu menjalankan program yang membutuhkan GPU di dalamnya (Wang dkk., 2021).

### Menginstall YOLO

Proses pembuatan model yang pertama kali dilakukan adalah dengan mendapatkan *depedency* YOLO terbaru yang sudah disiapkan oleh Ultralytics. Untuk mendapatkannya dapat dengan menginstall *library* yang sudah disiapkan seperti pada gambar 4.10. Setelah mendapatkan *depedency* dari Ultralytics, selanjutnya akan mengimpor YOLO ke dalam projek yang dibuat dengan menjalankan *code* seperti pada gambar 4.11. Dengan menjalankan kedua *code* tersebut, YOLO telah siap digunakan ke dalam projek untuk pembuatan model.



Gambar 4. 10: Mengimpor YOLO ke dalam projek



Gambar 4. 11: Code untuk menginstall library

### Mengimpor Dataset

Dataset yang sebelumnya sudah dibentuk di Roboflow akan diimpor ke dalam projek untuk membantu pembuatan model dengan *custom* dataset. Untuk mengimpor dataset, Roboflow sendiri telah memberikan kemudahan dengan *code* yang sudah disiapkan seperti yang dapat dilihat dari gambar 4.12. *Code* yang diberikan ini nantinya akan mengunduh dalam bentuk zip dan mengekstrak dataset versi pertama tersebut agar bisa dipakai. Jika dataset yang diimpor berhasil diunduh dan diekstrak dari bentuk zip-nya maka pada *output* yang dihasilkan akan terlihat seperti pada gambar 4.13.



Gambar 4. 12: Code untuk mengimpor dataset dari Roboflow



Gambar 4. 13: Keterangan jika berhasil mengunduh dan mengekstrak dataset

Setelah berhasil diekstrak, dapat dilihat pada gambar 4.14 bahwa Roboflow telah membagikan dataset ke dalam foldernya masing-masing sesuai pembagian yang ada di bagian 4.2.4 sebelumnya mengenai *data split.* Pembagian dataset ini juga telah dilengkapi dengan label yang memiliki *extention* .txt. File inilah yang berisi data-data objek dari gambar sesuai pelabelan yang telah dilakukan. Selain itu, Roboflow juga telah membuat *file* data.yaml. *File* ini digunakan untuk mengonfigurasi dataset. Sehingga *file* data.yaml ini akan berisi informasi mengenai *path* dataset, kelas-kelas yang digunakan dalam model nantinya, dan beberapa informasi yang relevan. Secara lebih jelas terkait isi dari *file* data.yaml, dapat dilihat dari gambar 4.15.



Gambar 4. 14: Hasil ekstrasi dataset



Gambar 4. 15: Isi dari file data.yaml

### Pelatihan Model

Terdapat dua cara yang dapat digunakan untuk melakukan *training,* yaitu dengan melalui *python environment* dan melalui *Command Line Interface* atau CLI (Jocher & Sergiuwaxmann, 2023). Pada tahap pembuatan model ini akan menggunakan cara CLI sehingga untuk *code* yang digunakan dapat dilihat pada gambar 4.16. Pada gambar tersebut, di awal akan menginisialisasi variabel data\_yaml dengan *path* yang mengarah pada *file* data.yaml. Setelah itu, program akan menjalankan perintah untuk *training* pada *task detect.* Selanjutnya terdapat beberapa parameter tambahan, seperti pada *code* ini akan menggunakan model yolov8m.pt. Penggunaan parameter model yang sudah dilatih sebelumnya ini bertujuan untuk membantu pembuatan model baru sehingga pada saat pelatihan atau pembuatan model tidak memakan waktu dan daya komputasi yang berlebihan (Jocher & Sergiuwaxmann, 2023). Jenis model yang digunakan ini adalah model yolov8 versi medium (yolov8m.pt). Hal ini dilakukan dengan mengingat bahwa ukuran gambar yang digunakan adalah ukuran medium sehingga dapat membantu meringankan beban komputasi (Zhao & Li, 2020).



Gambar 4. 16: Code untuk pembuatan model dengan custom dataset

Parameter selanjutnya yang diatur adalah *epoch.* Nilai *default* dari *epoch* adalah 100 *epoch.* Berdasarkan skenario pengujian pada tabel 3.1, maka nilai *epoch* akan diatur dengan nilai 200 *epoch*. Selanjutnya untuk parameter imgz (*image size*) akan diatur sesuai dengan ukuran dari dataset yang sebelumnya sudah dilakukan *resize*. Nilai imgsz ini akan mempengaruhi bagaimana model YOLO akan memproses gambar dan mendeteksi objek di dalamnya (Jocher & Sergiuwaxmann, 2023). Pada parameter terakhir yang diatur adalah terkait *plots*. Parameter ini akan menyimpan grafik-grafik hasil penelitian sehingga dapat membantu analisis di akhir nantinya. Terlepas dari parameter yang diatur, sebenarnya ada banyak parameter yang dijalankan atau diimplementasi di dalam pelatihan. Namun *value* dari parameter yang lain akan diatur sesuai dengan nilai *dafault.* Untuk lebih rinci dapat dilihat pada gambar 4.17.



Gambar 4. 17: Detail yang digunakan pada pelatihan

Pada tiap *epoch* nantinya akan ditampilkan nilai-nilai yang dapat digunakan untuk menganalisis performa dari model yang dibuat. Seperti yang terlihat pada gambar 4.18, terdapat nilai-nilai seperti nilai box\_loss, cls\_loss, df1\_loss, mAP50, dan nilai-nilai yang lain seperti yang ditampilkan pada gambar 4.18. Nilai-nilai ini juga akan disimpan ke dalam sebuah *file* csv yang nanti akan digunakan untuk menganalisis performa model.



Gambar 4. 18: Output dari rincian tiap epoch

Pelatihan yang dilakukan sebanyak 200 *epoch* ini akan dilakukan dengan *patience* dengan nilai 50 *epoch*. Sehingga ketika dalam 50 *epoch* terakhir tidak mengalami peningkatan pada performa maka pelatihan akan dihentikan. Hal ini dilakukan agar membantu untuk mencegah *overfitting*.

## Analisis Hasil Pengujian



### Hasil Pelatihan Model

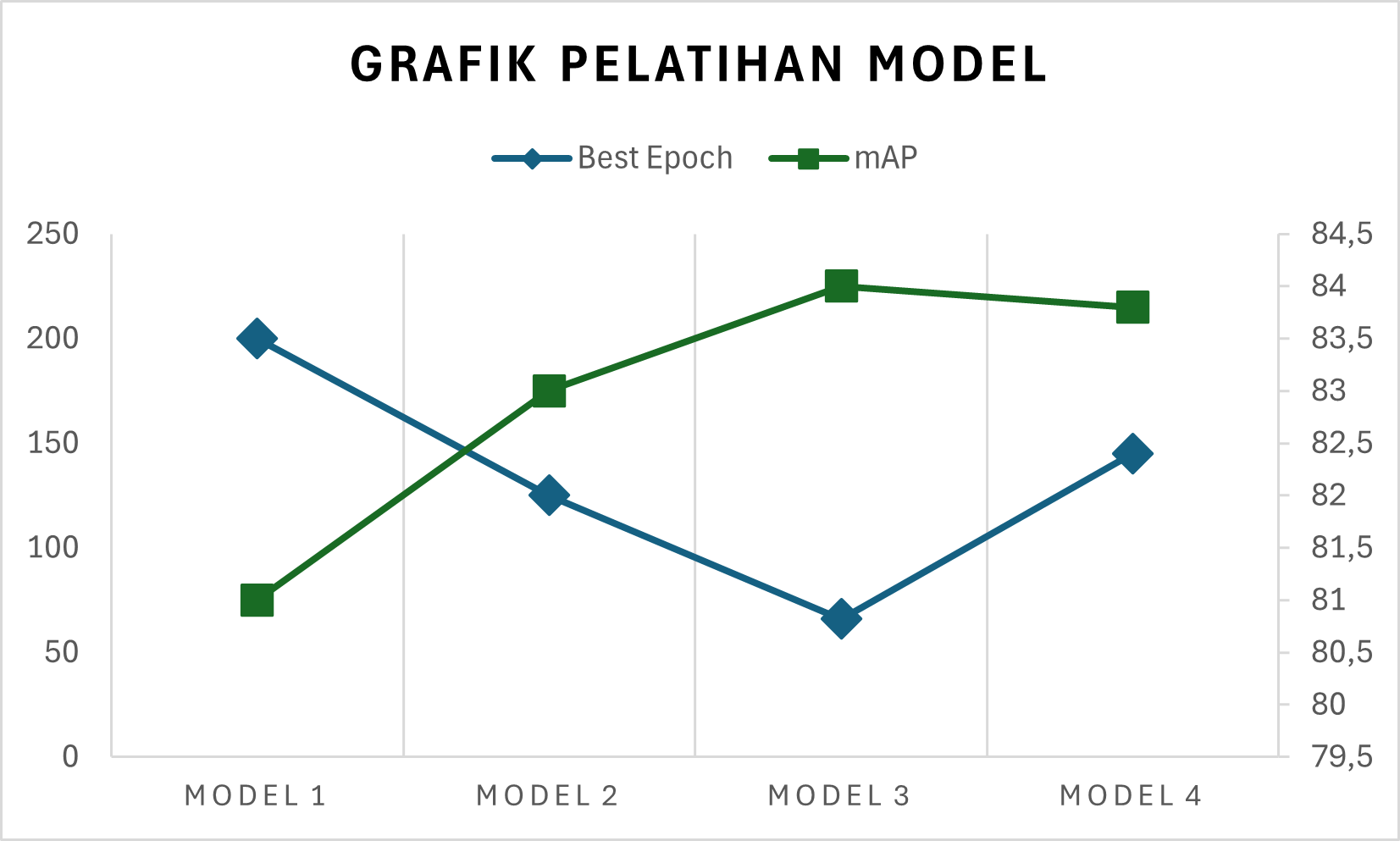
Berdasarkan skenario pengujian pada tabel 3.1, maka dilakukan pelatihan model sebanyak empat kali. Hasil dari pelatihan ini akan ditampilkan pada tabel 4.1. Pada tabel tersebut dapat dilihat

Tabel 4. 1: Hasil penelitian model berdasarkan skenario pengujian

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Summary Result** | | | | | |
| **Model** | **Epoch** | **mAP** | **Epoch Best** | **mAP** | **Jenis Dataset** |
| Model 1 | 200 | 0,816 | - | - | 608 & augmentasi |
| Model 2 | 175 | 0,821 | 125 | 0,83 | 416 & augmentasi |
| Model 3 | 116 | 0,812 | 66 | 0,84 | 416 !augmentasi |
| Model 4 | 195 | 0,833 | 145 | 0,838 | 608 !augmentasi |

bahwa pada pelatihan model pertama dengan dataset berukuran 608x608 dan memiliki augmentasi melalui pelatihan dengan 200 *epoch* dan mendapatkan mAP sebesar 81%, selanjutnya pelatihan pada model kedua berhenti pada *epoch* ke-175 dengan mAP 82% dan mAP terbaiknya didapatkan pada *epoch* ke-125 dengan mAP sebesar 82%. Berhentinya pelatihan ini karena selama 50 *epoch* terakhir tidak ada peningkatan mAP. Lebih lanjut lagi pada pemodelan ketiga mengalami kenaikan mAP pada *epoch* ke-66 dengan mAP sebesar 84% dan pada pemodelan keempat pada *epoch* ke-145 didapatkan mAP terbaiknya dengan nilai 83%. Secara lebih ringkas dapat dilihat pada gambar grafik 4.19.

Secara garis besar hasil yang didapatkan sudah cukup bagus dengan mAP terbesar didapatkan pada model ketiga. Namun secara detail dan dilihat dari grafik tersebut, perbedaan mAP tiap model tidak memiliki perbedaan yang signifikan sehingga belum dapat menentukan apakah model tersebut merupakan model dengan hasil yang memuaskan (Karna dkk., 2023; Sholahuddin dkk., 2023). Perubahan yang kurang signifikan ini juga dapat dilihat pada perbandingan *precision* dan *recall* tiap model pada tabel 4.2.



Gambar 4. 19: Grafik pelatihan model

Tabel 4. 2: Tabel perbandingan precision, recall, dan validasi tiap model

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Precision** | **Recall** | **Validasi** |
| model 1 | 0,84 | 0,75 | 0,821 |
| model 2 | 0,85 | 0,77 | 0,839 |
| model 3 | 0,83 | 0,78 | 0,839 |
| model 4 | 0,86 | 0,77 | 0,846 |

Pada proses validasi data pun tidak menemukan perbandingan yang signifikan di antara keempat model.

Dari hasil analisis, dapat disimpulkan sementara bahwa keempat model yang dievaluasi, yaitu model 1, model 2, model 3, dan model 4, tidak menunjukkan perbedaan yang signifikan dalam kinerja mereka. Hal ini dapat dilihat dari nilai mAP, *precision*, *recall*, dan validasi yang relatif serupa antara keempat model tersebut. Oleh karena itu, akan dilakukan pengembangan model dengan melibatkan penyesuaian *hyperparameter* untuk memperbaiki dan mengoptimalkan model.

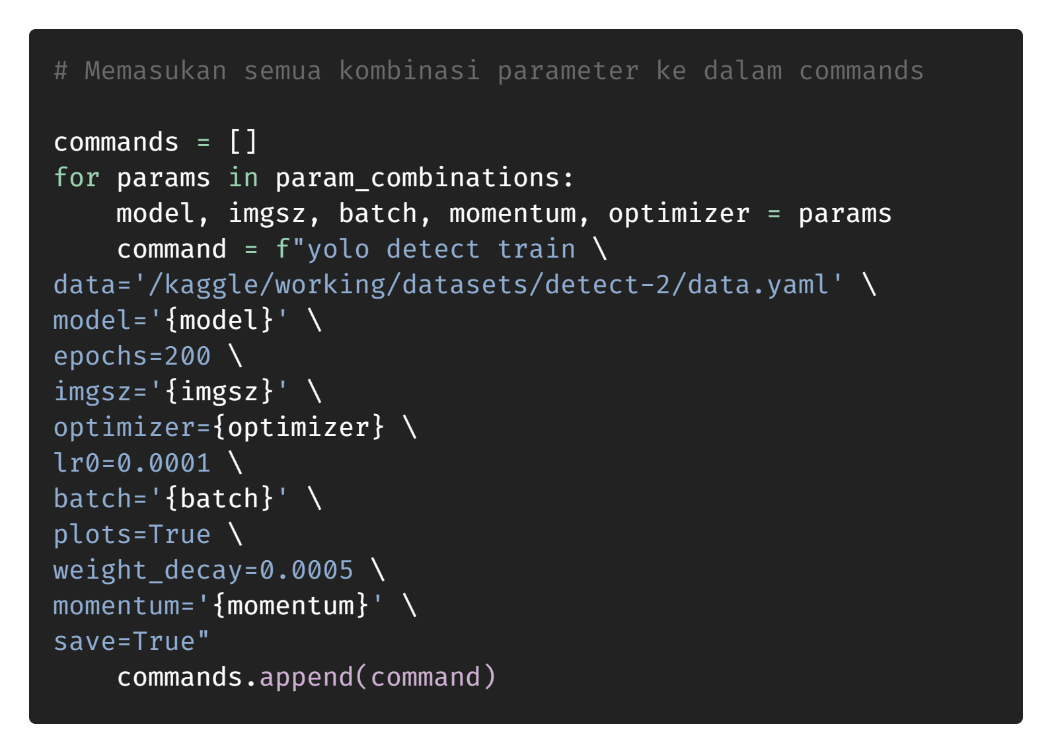
### Uji Coba *Hyperparameter*

Pengujian ini hanya akan melibatkan dataset yang memiliki augmentasi dan akan dilakukan variasi pada ukuran datasetnya. Hal ini dilakukan untuk tetap menjaga variasi yang beragam dalam dataset. Pada pengujian menggunakan *hyperparemeter* ini akan menggunakan beberapa parameter yang akan membantu untuk pengoptimalan model seperti yang dapat dilihat pada tabel 4.13. (Karna dkk., 2023; Sandhya & Kashyap, 2024; Sholahuddin dkk., 2023; Tamang dkk., 2023).

Tabel 4. 3: Rincian hyperparameter

|  |  |  |
| --- | --- | --- |
| **No.** | **Hyperparameter** | **Variasi** |
| 1. | Model YOLO | yolov8n.pt |
| yolov8s.pt |
| yolov8m.pt |
| 2. | Ukuran Gambar | 240 |
| 256 |
| 416 |
| 3. | Batch | 16 |
| 32 |
| 64 |
| 4. | Momentum | 0,937 |
| 0,8 |
| 5. | Weight Decay | 0,0005 |
| 6. | Learning Rate | 0,0001 |
| 7. | Optimizer | adam |
| adamW |

Untuk melakukan pengujian di tiap parameter, maka sebelumnya akan mengombinasikan tiap parameter sehingga nantinya akan menjalankan *code* secara otomatis. Untuk mengombinasikan tiap parameter dapat dilihat pada gambar 4.20. hasil dari tiap kombinasi ini nantinya akan dimasukkan ke dalam *code* CLI yang masih akan ditampung seperti pada gambar 4.21. pada tahap ini variabel array commads akan menyimpan tiap *code* yang nantinya hanya akan tinggal dipanggil untuk proses pembuatan modelnya. Lebih lanjut dapat dilihat pada gambar 4.22 di mana dengan bantuan *library* subprocess, tiap *code* CLI yang ada di variabel commands akan dieksekusi secara otomatis. Secara garis besar, dari hasil kombinasi ini menghasilkan 108 kombinasi parameter.



Gambar 4. 20: Code untuk memasukkan hasil kombinasi ke dalam code CLI YOLO



Gambar 4. 21: Code untuk mengombinasikan tiap parameter

Gambar 4. 22: Code untuk mengeksekusi tiap perintah yang ada di commands

### Analisis Uji *Hyperparameter*

Uji coba dengan menggunakan *hyperparameter* ini membuat model mengalami kenaikan yang cukup signifikan jika dibandingkan dengan sebelumnya dengan melihat perbandingan mAP yang naik sekisar 3% – 4%. Perkembangan ini dapat dilihat pada tabel 4.4 yang berisi parameter dengan mAP yang didapatkan. Pada tabel tersebut dapat dilihat hasil terbaik yang didapatkan dengan mAP sebesar 0.88971 dengan kombinasi parameter seperti pada gambar 4.23.



Gambar 4. 23: Kombinasi parameter dengan hasil mAP tertinggi

Tabel 4. 5: Tabel hasil pelatihan dengan hyperparameter

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Parameter** | | | | | | | | **Hasil** | | | |
| **model** | **imgsz** | **optimizer** | **lr0** | **batch** | **weight decay** | **momentum** | **epoch** | **Max epoch** | **mAP50** | **Best Epoch** | **Best mAP50** |
| m | 240 | Adam | 0,0001 | 64 | 0,0005 | 0,937 | 200 | 200 | 0,871 | 65 | 0,88971 |
| l | 240 | AdamW | 32 | 0,937 | 200 | 0,878 | 168 | 0,88942 |
| s | 240 | AdamW | 16 | 0.8 | 199 | 0,868 | 38 | 0,88603 |
| x | 240 | Adam | 32 | 0.8 | 66 | 0,862 | 29 | 0,88482 |
| m | 256 | Adam | 32 | 0.8 | 130 | 0,86786 | 78 | 0,88275 |
| s | 256 | AdamW | 16 | 0,937 | 83 | 0,85998 | 16 | 0,88161 |
| x | 256 | Adam | 32 | 0,937 | 103 | 0,87453 | 85 | 0,88135 |
| n | 240 | AdamW | 16 | 0,937 | 116 | 0,858 | 24 | 0,8799 |
| l | 256 | Adam | 16 | 0.8 | 199 | 0,87475 | 150 | 0,87703 |
| n | 256 | Adam | 64 | 0,937 | 114 | 0,85891 | 59 | 0,87651 |

Model yang dioptimalkan menggunakan *hyperparameter* ini ketika dilakukan validasi pun mendapatkan hasil yang bagus seperti yang terlihat pada gambar 4.24. Pada hasil validasi ini dapat dilihat bahwa selain objek sepatu, model telah dapat memprediksi tiap objek dengan baik yang ditunjukkan dengan hasil di atas 85%. Hal ini pun diperkuat dengan *confusion matrix* yang dapat dilihat dari gambar 4.25. Dari *confusion matrix* tersebut dapat dipastikan bahwa model memiliki *precision* dan *recall* yang tinggi untuk sebagian besar kelas objek, seperti handphone, jam, mobil, orang, dan tas. Namun dibalik itu, model ini memiliki *precision* dan *recall* yang rendah untuk kelas sepatu yang sering dideteksi sebagai *background*.

Gambar 4. 24: Confusion matrix



Gambar 4. 25: Hasil validasi terhadap model

Secara lebih lanjut, dapat dilihat pada gambar 4.26 yang menampilkan garis besar dari beberapa grafik yang menentukan perkembangan model. Pada gambar tersebut dapat dilihat bahwa model menunjukkan performa yang baik dengan ditunjukkan pada grafik *loss* yang semakin menurun dalam proses pelatihan, grafik *precision, recall,* dan mAP yang tinggi. Kombinasi dari penurunan nilai *loss*, nilai *precision*, *recall* dan nilai mAP yang tinggi menunjukkan bahwa model mampu mendeteksi objek dengan akurat dan efisien.

Gambar 4. 26: Kumpulan grafik evaluasi

## Implementasi Aplikasi Deteksi Objek

### *Interface* Aplikasi

Implementasi model ke dalam *interface* akan dibantu dengan *framework* Streamlit. Alasan mengapa menggunakan *framework* ini adalah karena pihak Streamlit telah menyediakan *cloud* khusus bagi para anggota komunitas untuk *deploy* hasil kerjanya (Sholahuddin dkk., 2023). Pada *website* ini akan terdapat beberapa fitur utama. Fitur-fitur tersebut antara lain mendeteksi gambar dengan mengunggah gambar atau dengan mengambilnya secara langsung, mendeteksi video dengan mengunggah video atau dengan beberapa video yang sudah disiapkan sebelumnya, mendeteksi dengan memasukan *link* YouTube, dan mendeteksi secara *realtime.* Pengimplementasian ini akan melibatkan beberapa *function* untuk mempermudah proses deteksi dengan model yang sudah dibuat sebelumnya.



Gambar 4. 27: Tampilan halaman pertama ketika website diakses

Berikut adalah *interface* yang dibuat di Streamlit. Ketika pertama kali diakses maka akan menampilkan halaman seperti pada gambar 4.27. Pada halaman ini akan menampilkan *dropdown* dari menu-menu yang disediakan juga menampilkan video tutorial penggunaan *website.* Berikutnya tampilan pada gambar 4.28 adalah tampilan jika memilih menu gambar dan telah melakukan deteksi dengan mengunggah foto atau gambar. Dapat dilihat bahwa di bawah *dropdown* menu terdapat *tab* untuk memilih ingin menggunakan fitur yang sama. Hal ini pun berlaku pada tampilan jika memilih menu video seperti yang terlihat pada gambar 4.29. pada menu ini telah disediakan lima video yang dapat langsung digunakan untuk deteksi sehingga user tidak perlu mengunggah videonya sendiri. Selanjutnya pada gambat 4.30 dapat dilihat tampilan jika memilih menu youtube. Pada menu ini, user hanya perlu memasukan link YouTube yang sudah disalin dan menekan tombol ‘deteksi’. Terakhir adalah tampilan jika user memilih menu Real-Time. Seperti yang terlihat pada gambar 4.31, menu ini akan mendeteksi secara realtime menggunakan kamera baik ponsel maupun komputer.



Gambar 4. 28: Tampilan halaman jika memilih menu gambar



Gambar 4. 29: Tampilan jika memilih menu video



Gambar 4. 30: Tampilan jika memilih menu youtube



Gambar 4. 31: Tampilan ketika memilih menu Real-Time

### Implementasi *Code* untuk Deteksi

Terdapat beberapa *function* yang diimplementasi untuk menunjang proses deteksi di tiap menu. Pada gambar 4.32 adalah *code* yang digunakan untuk mendeteksi gambar yang diunggah oleh user. Pada gambar tersebut, jika terdapat *value* di variabel source\_img maka akan muncul tombol ‘deteksi’ yang akan memproses prediksi jika ditekan dan menampilkan hasil prediksi tersebut. Berikut pada gambar 4.33 adalah *function* yang digunakan untuk melakukan deteksi dan menampilkan hasil deteksi ke *frame* yang sudah disiapkan sebelumnya. Perlu diperhatikan bahwa nilai *confidence* yang digunakan pada tiap deteksi akan menggunakan *confidence* sebesar 0.5. Pada gambar 4.34 merupakan *function* untuk mengisi *link* YouTube yang disalin. Ketika tombol deteksi maka program akan *capture* tiap *frame* video dengan bantuan *library* open-cv dan mengirimnya ke *function* showDetectFrame. Selanjutnya terdapat *function* untuk memproses video yang diunggah user. Pada *function* yang dapat dilihat dari gambar 4.35 ini, setelah video diunggah, video akan disimpan pada penyimpanan sementara. Dari hasil penyimpanan ini, video akan diambil gambar tiap *frame* video kemudian mengirimkan hasil *capture* kepada *function* showDetectFrame. Setelah selesai dideteksi tiap *frame*, video yang disimpan di penyimpanan sementara akan dihapus. Algoritma ini juga digunakan pada *function* untuk mendeteksi video-video yang sudah disiapkan. *Function* selanjutnya yang digunakan adalah *function* take\_picture yang dapat dilihat pada gambar 4.36. *Function* ini akan mengambil foto atau gambar secara langsung menggunakan bantuan st.camera\_input(). Sama seperti video yang diupload, gambar ini pun akan disimpan di penyimpanan sementara. Kemudian gambar tersebut akan dikirim ke *function* showDetectShow untuk melakukan deteksi dan menampilkan hasil. *Function* terakhir yang digunakan adalah *function* untuk membantu proses *realtime.* Proses ini akan menggunakan fitur webrtc\_streamer dari *framework* Streamlit. Ada beberapa konfigurasi yang diperlukan untuk dapat menjalankan fitur ini dengan baik. Hal utama yang diperlukan adalah konfigurasi rtc. Pada perubahan infrastruktur terbaru terkait webrtc\_streamer, ditetapkan harus menggunakan bantuan Twilio agar dapat menjalankan *live* tersebut. Selain itu, terdapat juga class VideoTransformer seperti yang terlihat pada gambar 4.38 yang digunakan untuk mendeteksi secara *realtime* tiap *frame* dari hasil *live* tersebut.



Gambar 4. 32: Code untuk mendeteksi gambar yang diupload

Gambar 4. 33: Function untuk menampilkan gambar hasil deteksi



Gambar 4. 34: Function untuk mencari video dari link youtube



Gambar 4. 35: Function untuk memproses video yang diupload



Gambar 4. 36: Function untuk mengambil foto atau gambar secara langsung



Gambar 4. 37: Function untuk realtime

Gambar 4. 38: Code untuk mendeteksi realtime

Penerapan-penerapan *code* ini mampu menjalankan fitur-fitur yang sudah dibuat dengan Streamlit. *Code* ini akan disimpan di sebuah *repository* GitHub dan dihubungkan pada Cloud Community Share agar dapat diakses secara *online* baik dari ponsel maupun komputer.

## Pengujian Kegunaan Aplikasi



## Daftar Pustaka

Iryanto, S. Y., & Zaini, T. M. (2014). *Pengolahan Citra Digital*. Anggota IKAPI.

Jocher, G., & Sergiuwaxmann. (2023, Januari 9). *Ultralytics YOLOv8 Docs*. Ultralytics. https://docs.ultralytics.com

Karna, N. B. A., Putra, M. A. P., Rachmawati, S. M., Abisado, M., & Sampedro, G. A. (2023). Toward Accurate Fused Deposition Modeling 3D Printer Fault Detection Using Improved YOLOv8 With Hyperparameter Optimization. *IEEE Access*, *11*, 74251–74262. https://doi.org/10.1109/ACCESS.2023.3293056

Perez, F., Vasconcelos, C., Avila, S., & Valle, E. (2018). Data augmentation for skin lesion analysis. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *11041 LNCS*, 303–311. https://doi.org/10.1007/978-3-030-01201-4\_33

Sandhya, & Kashyap, A. (2024). A novel method for real-time object-based copy-move tampering localization in videos using fine-tuned YOLO V8. *Forensic Science International: Digital Investigation*, *48*. https://doi.org/10.1016/j.fsidi.2023.301663

Sholahuddin, M. R., Harika, M., Awaludin, I., Dewi, Y. C., Dhia Fauzan, F., Sudimulya, B. P., & Widarta, V. P. (2023). Optimizing YOLOv8 for Real-Time CCTV Surveillance: A Trade-off Between Speed and Accuracy. *Jurnal Online Informatika*, *8*(2), 261–270. https://doi.org/10.15575/join.v8i2.1196

Tamang, S., Sen, B., Pradhan, A., Sharma, K., & Singh, V. K. (2023). Enhancing COVID-19 Safety: Exploring YOLOv8 Object Detection for Accurate Face Mask Classification. *International Journal of Intelligent Systems and Applications in Engineering IJISAE*, *11*(2), 892–897. www.ijisae.org

Wang, A. Y., Wang, D., Drozdal, J., Liu, X., Park, S., Oney, S., & Brooks, C. (2021). What Makes aWell-Documented Notebook? A Case Study of Data Scientists’ Documentation Practices in Kaggle. *Conference on Human Factors in Computing Systems - Proceedings*. https://doi.org/10.1145/3411763.3451617

Zhao, L., & Li, S. (2020). Object Detection Algorithm Based on Improved YOLOv3. *Electronics (Switzerland)*, *9*(3). https://doi.org/10.3390/electronics9030537