JURNAL FASILKOM
Volume 14 No. 3 | Desember 2024: 646-653

P-ISSN : 2089-3353
E-ISSN : 2808-9162

Pengembangan Deteksi Objek Dalam Rumah Bagi Tunanetra Berbasis Optimasi *YOLOv8* Menggunakan Metode *Ghost Module* dan *Attention Mechanism*

Muhammad Insan Kamil[1], Syamsul Mujahidin[2], Riska Kurniyanto Abdullah[3]
[123]Program Studi Informatika, Jurusan Matematika dan Teknologi Informasi, Institut Teknologi Kalimantan
[1]11211058@student.itk.ac.id*, [2]syamsul@lecturer.itk.ac.id, [3]riska.abdullah@lecturer.itk.ac.id

Abstract

Visually impaired individuals often face difficulties in daily mobility due to due to the limitations of real-time object detection tools. Although specialized canes can aid in walking, but not yet effective enough. Advances in image-based object detection, especially the use of machine learning, offer promising solutions. Machine learning-based object recognition technologies, such as YOLOv8, could provide potential solutions. However, when implemented on small devices such as the Raspberry Pi 4, computing power is limited.This research aims to develop a lighter computational burden version of YOLOv8 nano. The proposed method involves utilizing ghost modules, downsampling, and attention mechanisms. The findings reveal that the use of ghost modules and downsampling effectively reduces the GFLOPs of the YOLOv8n model from 8.09 GFLOPs to 1.77 GFLOPs, decreasing the model's inference time by 57.6%, from 401.56 ms to 170.33 ms on Raspberry Pi 4 hardware, without compromising detection performance. Moreover, integrating attention mechanisms through attention max pooling enhances the model's accuracy, increasing the mAP by 1.3% compared to standard max pooling. This model successfully delivers more accurate and efficient detection, making it a potential solution for building embedded systems to assist visually impaired individuals in real-time object detection.

Keywords: ghost module, downsampling, attention mechanism, visually impaired, YOLOv8, Raspberry Pi 4

Abstrak

Penyandang tunanetra sering menghadapi kesulitan dalam mobilitas sehari-hari karena keterbatasan alat bantu deteksi objek secara *real-time*. Meskipun tongkat khusus dapat membantu dalam berjalan, namun belum cukup efektif. Kemajuan pengenalan objek berbasis citra, terutama penggunaan *machine learning*, menawarkan solusi yang menjanjikan. Teknologi pengenalan objek berbasis *machine learning*, seperti *YOLOv8*, bisa memberikan solusi potensial. Namun saat diimplementasikan di perangkat kecil seperti Raspberry Pi 4 terkendala kemampuan komputasi. Penelitian ini bertujuan untuk mengembangkan model *YOLOv8* versi nano dengan beban komputasi yang lebih ringan. Metode yang diusulkan melibatkan penggunaan *ghost module*, *downsampling*, dan *attention mechanism*. Hasil penelitian menunjukkan bahwa penggunaan *ghost module* dan *downsampling* efektif mengurangi *GFLOPs* model *YOLOv8n* dari 8.09 *GFLOPs* menjadi 1.77 *GFLOPs*, menurunkan waktu *inference* model hingga 57,6%, dari 401,56 ms menjadi 170,33 ms pada perangkat keras Raspberry Pi 4, tanpa mengorbankan performa deteksi. Selain itu, integrasi *attention mechanism* melalui *attention max pooling* meningkatkan akurasi model dengan peningkatan *mAP* sebesar 1,3% dibandingkan *max pooling* standar. Model ini berhasil memberikan deteksi yang lebih akurat dan efisien, menjadikannya solusi yang potensial dalam membangun sistem benam untuk membantu penyandang tunanetra dalam mendeteksi objek secara *real-time*.

Kata kunci: *ghost module*, *downsampling*, *attention mechanism*, tunanetra, *YOLOv8*, Raspberry Pi 4

1. Pendahuluan

Tunanetra adalah istilah umum yang digunakan untuk kondisi seseorang yang mengalami gangguan atau hambatan dalam indra penglihatannya. Untuk saat ini dalam melakukan mobilitas kegiatan sehari-hari, para penyandang tunanetra dibantu dengan menggunakan tongkat khusus [1]. Namun demikian, kendala yang dihadapi tunanetra ketika menggunakan tongkat khusus tersebut yakni belum mampu mendeteksi sebuah objek secara *real-time* dan mengeluarkan output berupa suara dari nama objek tersebut. Tentunya hal tersebut sangat merepotkan bagi para penyandang tunanetra [2].

Dengan dukungan metode *machine learning*, sistem dapat mengenali dan mengidentifikasi objek secara akurat, sehingga memberikan kemudahan dan kenyamanan lebih bagi tunanetra. Metode ini memungkinkan sistem bekerja secara konsisten seperti manusia, meskipun masih terbatas pada kriteria yang telah ditentukan. Untuk mewujudkan sistem pendeteksi objek yang efektif, diperlukan perangkat kecil seperti Raspberry Pi 4 yang ringan dan kompatibel untuk kebutuhan mobilitas tinggi sehingga memberikan kenyamanan bagi penyandang tunanetra ketika melakukan aktivitas sehari-hari. Namun, keterbatasan kemampuan komputasi Raspberry Pi 4 menjadi tantangan, mengingat deteksi objek [3] membutuhkan daya komputasi besar. Oleh karena itu, diperlukan optimasi deteksi objek agar memiliki waktu *inference* yang rendah sehingga dapat berjalan efisien pada perangkat dengan kemampuan komputasi yang terbatas.

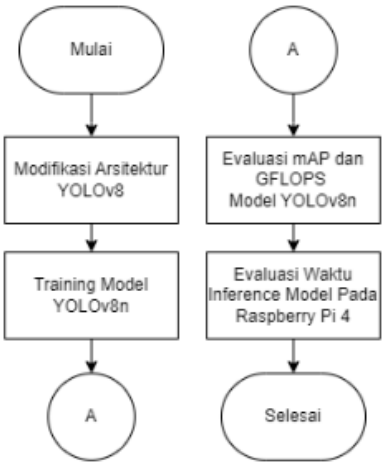
P-ISSN : 2089-3353
E-ISSN : 2808-9162

Penelitian yang berkaitan dengan optimasi deteksi objek telah banyak dilakukan sebelumnya. Akan tetapi, meskipun telah dilakukan optimasi, beban komputasi model masih cukup besar untuk diimplementasikan pada *small device* yang memiliki kemampuan komputasi yang terbatas. Pemilihan penggunaan *small device* dengan komputasi rendah dimaksudkan untuk mengurangi biaya yang dikeluarkan ketika implementasi sistem deteksi objek untuk tunanetra. Fan dkk [4] telah mengusulkan untuk menggunakan metode *channel* prunning pada *YOLOv8m* yang berhasil mengurangi beban komputasi model sebesar 40% dari 78.9 *GFLOPs* menjadi 43.7 *GFLOPs* dengan pengurangan akurasi 0,1 % saja. Wu dkk [5] telah mengusulkan *backbone* baru pada *YOLOv8n* khusus segmentasi berupa *LCANet* yang berhasil mengurangi beban komputasi model sebesar 51% dari 12.4 *GFLOPs* menjadi 6.1 *GFLOPs* dengan pengurangan akurasi sebesar 3%. Namun, kedua pendekatan tersebut masih memiliki kelemahan yakni dari sisi beban komputasi yang masih relatif tinggi untuk *small device*. Maka berdasarkan hal tersebut, pada penelitian ini akan dibangun sebuah model *YOLOv8* versi nano yang memiliki beban komputasi yang ringan dan performa yang baik dengan menggunakan metode modifikasi *module convolution* menjadi, penambahan proses *downsampling*, *ghost module* dan integrasi *attention mechanism*. Tujuan dari penelitian ini adalah menganalisis efektivitas penggunaan metode *ghost module* dan *downsampling* dalam mengurangi *GFLOPs* dari model *YOLOv8n*, sehingga menghasilkan waktu *inference* yang lebih sesuai dengan keterbatasan perangkat keras Raspberry Pi 4. Selain itu, penelitian ini juga bertujuan untuk menganalisis efektivitas penggunaan metode *attention mechanism* dalam meningkatkan *mAP* dari model *YOLOv8n*, sehingga mampu memberikan hasil deteksi objek yang lebih akurat.

2. Metode Penelitian

2.1. Alur Penelitian

Alur dalam penelitian ini terdiri dari beberapa langkah seperti yang tercantum pada Gambar 1.



Gambar 1. Alur Penelitian

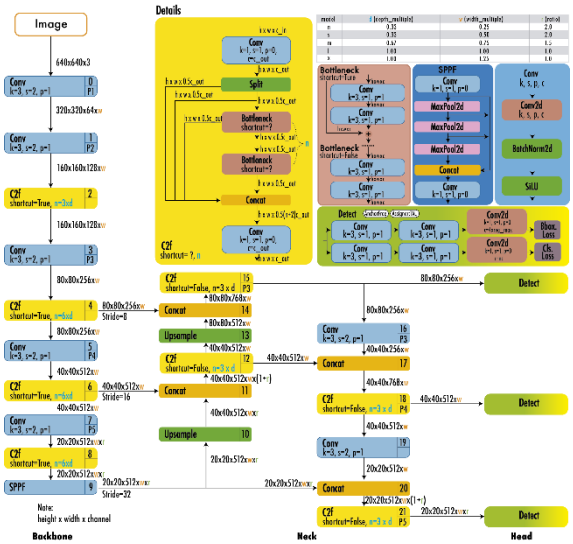
Langkah pertama dimulai dengan memodifikasi arsitektur *YOLOv8* melalui penerapan berbagai metode optimasi, meliputi penambahan proses *downsampling*, integrasi *attention mechanism*, dan penggantian modul *convolution* dengan *ghost module*. Modifikasi ini bertujuan untuk mengurangi beban komputasi model sekaligus meningkatkan akurasi deteksi objek. Proses ini mencakup analisis awal terhadap pengaruh masing-masing metode terhadap struktur model dan implementasinya secara bertahap untuk menghasilkan kombinasi terbaik. Langkah kedua adalah melakukan *training YOLOv8 baseline* dan *YOLOv8* yang telah dimodifikasi menggunakan dataset yang di dapatkan dari *roboflow* [6]. Dataset ini terdiri dari delapan kelas objek, yaitu manusia, televisi, tempat tidur, kursi, pintu, kipas angin, sofa, dan meja. Dataset mencakup 5669 gambar, dengan resolusi gambar pada dataset adalah 640x640 piksel, yang sesuai dengan kebutuhan arsitektur *YOLOv8*. Kemudian untuk *epoch* yanng digunakan pada saat *training* sebesar 100 dan *batch* sebesar 16 [7]. Langkah ketiga, melakukan evaluasi hasil dari *training* dari segi performa berdasarkan nilai *mAP* model dan beban komputasi berdasarkan nilai *GFLOPs* model. *mAP* (*mean AveragePrecision*) adalah metrik yang digunakan untuk mengevaluasi performa model deteksi objek. Metrik ini menggabungkan *precision* dan *recall* pada berbagai nilai *threshold* untuk memberikan gambaran seberapa baik model dapat mendeteksi dan mengklasifikasikan objek. Sementara *GFLOPs* (*Giga Floating-Point Operations*) merupakan ukuran yang mengindikasikan jumlah operasi *floating-point* yang dapat dieksekusi oleh sebuah model atau perangkat keras dalam satu detik. Setelah proses *training* model, langkah selanjutnya adalah melakukan evaluasi terhadap performa model-model tersebut. Evaluasi ini bertujuan untuk membandingkan nilai *GFLOPs* dan *mAP* antara model baseline *YOLOv8n* dengan model *YOLOv8n* yang telah dimodifikasi. Perbandingan ini dilakukan untuk melihat apakah implementasi *downsampling*, *attention mechanism*, dan *ghost module* memberikan pengaruh terhadap performa dan kecepatan deteksi model *YOLOv8n*. Langkah keempat adalah melakukan pengujian waktu *inference* pada perangkat keras dengan sumber daya terbatas, yaitu Raspberry Pi 4. Model yang diujikan meliputi *YOLOv8n* baseline dan model hasil optimasi yang memiliki keseimbangan terbaik antara akurasi dan beban komputasi. Kedua pengujian tersebut dilakukan menggunakan data *testing* berupa video yang sama untuk memastikan bahwa perbandingan yang dilakukan valid. Waktu inferensi diukur untuk menilai kelayakan model ketika dijalankan dalam keadaan *real-time*.

2.2. *YOLOv8*

YOLOv8 atau "*You Only Look Once*" versi 8, merupakan Algoritma untuk mendeteksi objek pada citra, baik berupa gambar maupun video [8]. Selain itu, *YOLOv8* juga bisa digunakan untuk melakukan tugas segmentasi dan klasifikasi. Gambar 2, menampilkan

P-ISSN : 2089-3353
E-ISSN : 2808-9162

arsitektur *YOLOv8* yang memiliki tiga bagian utama yaitu *backbone*, *neck* dan *head* [9].



Gambar 2. Arsitektur *YOLOv8*

Berdasarkan Gambar 3, *YOLOv8* memiliki berbagai macam varian yaitu *YOLOv8n*, *YOLOv8s*, *YOLOv8m*, *YOLOv8l* dan *YOLOv8x* [10].

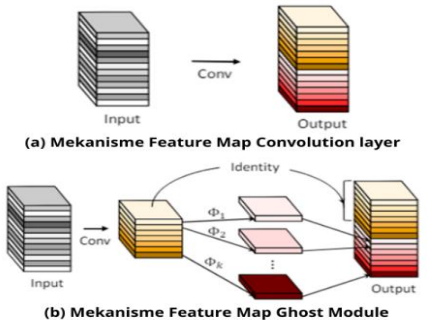
Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Gambar 3. Perbandingan Performa dan Beban Komputasi Varian *YOLOv8*

Pada penelitian ini, peneliti akan melakukan optimasi pada *YOLOv8n* karena memiliki beban komputasi yang paling kecil yaitu 8.7 *GFLOPs* ketika dievaluasi menggunakan COCO dataset.

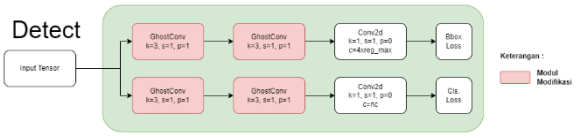
2.3. *Ghost Module*

Ghost module adalah inovasi dalam arsitektur jaringan neural yang dirancang untuk mengurangi beban komputasi dan parameter dalam jaringan, dengan tetap mempertahankan kemampuan ekstraksi fitur yang kuat [11]. Gambar 4, menampilkan perbedaan mekanisme dalam menghasilkan *feature map* antara *convolution layer* dan *ghost module*.



Gambar 4. Perbandingan Mekanisme *Convolution Layer* Dan *Ghost Module*

Untuk modifikasi modul *convolution* menjadi *ghost module* akan dilakukan pada modul *detect* di bagian *head* arsitektur *YOLOv8* seperti yang terlihat pada Gambar 5.



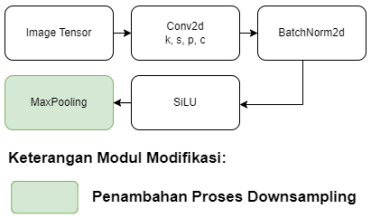
Gambar 5. Arsitektur Modifikasi Modul *Detect*

Penggunaan *ghost module* hanya diterapkan pada bagian *head* arsitektur *YOLOv8* karena peran masing-masing komponen dalam proses deteksi objek memiliki urgensi yang berbeda. *Backbone*, yang bertugas mengekstraksi fitur dasar dari citra masukan, merupakan bagian krusial dalam menghasilkan fitur kompleks. Modifikasi konvolusi pada bagian ini, seperti dengan *ghost module*, berisiko mengurangi kemampuan model dalam menangkap detail penting, terutama pada objek kecil atau dengan variasi bentuk yang kompleks. Sementara itu, *neck*, yang berfungsi untuk menggabungkan informasi fitur dari berbagai tingkat resolusi, juga berperan penting dalam memastikan kualitas informasi fitur yang didapatkan dari *backbone* untuk mendukung prediksi akurat. Modifikasi pada *neck* dapat mengurangi efektivitas pengolahan informasi tersebut, sehingga menurunkan performa model secara keseluruhan. Sebaliknya, *head* bertugas memproses fitur yang telah disediakan oleh backbone dan *neck* untuk menghasilkan prediksi akhir berupa *bounding box* dan klasifikasi objek. Penggunaan *ghost module* pada bagian ini lebih aman karena hanya memengaruhi tahap akhir prediksi tanpa mengorbankan kualitas fitur awal yang telah diekstraksi. Selain itu, penerapan *ghost module* pada *head* membantu mengurangi beban komputasi model dengan menghasilkan fitur *redundant* secara efisien, sehingga *GFLOPs* dapat ditekan tanpa berdampak signifikan terhadap akurasi. Hal ini sangat penting untuk mengoptimalkan performa model pada perangkat dengan kapasitas komputasi terbatas, seperti Raspberry PI 4. Dengan membatasi modifikasi hanya pada bagian *head*, keseimbangan antara efisiensi komputasi dan akurasi model tetap terjaga.

2.4. *Downsampling*

Downsampling adalah proses mengurangi dimensi data input pada *feature map* menggunakan *pooling layer*, yang secara efektif memperkecil ukuran data dan biaya komputasi [12]. Penambahan proses *downsampling* akan dilakukan pada modul *convolution* pertama arsitektur *YOLOv8*. Hal ini bertujuan untuk menekan beban komputasi model semaksimal mungkin dengan mengurangi dimensi dari *feature map* dari awal proses ekstraksi *feature*. Gambar 6, menujukkan arsitektur dari modul *convolution* yang telah ditambahkan *pooling layer* berupa *max pooling*.

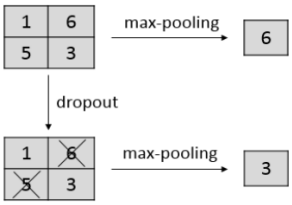
P-ISSN : 2089-3353
E-ISSN : 2808-9162



Gambar 6. Arsitektur Modifikasi Modul *Convolution* Dengan *Max Pooling*

2.5. *Max Pooling Dropout*

Dalam upaya mengurangi beban komputasi dan mencegah *overfitting*, akan dicoba penggunaan *max pooling dropout* [13]. Gambar 7, menampilkan mekanisme dari *max pooling dropout*.



Gambar 7. Mekanisme *Max Pooling Dropout*

Implementasi *max pooling dropout* mirip dengan *max pooling* seperti yang ditunjukkan pada Gambar 6, yaitu ditambahkan setelah lapisan fungsi aktivasi *Silu*. Hanya saja ada sedikit perbedaan ketika *training* dan proses *inference*. Untuk *dropout* hanya akan digunakan ketika proses *training*. Hal ini bertujuan untuk mencegah model terlalu bergantung pada pola tertentu dalam data *training*, sehingga model menjadi lebih general dan risiko *overfitting* dapat diminimalkan. Sementara itu, pada proses *inference*, *dropout layer* tidak diterapkan agar informasi pada data baru tidak hilang dan hasil prediksi tetap optimal. Algoritma implementasi *max pooling dropout* dapat dilihat sebagai berikut:

Algoritma Implementasi *Max Pooling Dropout*

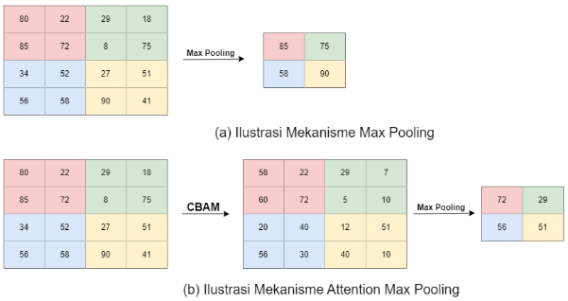
```
Input: feature_map
Output: feature_map
Initialization conv, b_norm, act, dropout, max pooling

feature_map=act(b_norm(conv(feature_map)))
If training_mode is True:
    feature_map = dropout(feature_map)
feature_map = max_pool(feature_map)
```

2.6. *Attention Mechanism*

Attention mechanism adalah teknik yang digunakan dalam pembelajaran mesin untuk meningkatkan kinerja model dengan memusatkan perhatian pada informasi yang relevan. Secara umum terdapat 4 modul yang biasa digunakan pada teknik *attention mechanism* yakni SENET (*Squeeze-and-Excitation Network*), *CBAM* (*Convolutional Block Attention Module*), BAM (*Bottleneck Attention Module*), ECA (*Efficient Channel Attention*). Modul *attention mechanism* yang digunakan pada penelitian ini adalah *CBAM* karena lebih cocok digunakan untuk deteksi objek.

CBAM) adalah modul yang terdiri dari dua *layer* yaitu *channel attention* dan *spatial attention* seperti yang terlihat pada Gambar 2.6 [14]. Berbeda dengan metode lain seperti *SENet* (*Squeeze-and-Excitation Network*) [15], yang hanya berfokus pada *channel attention* , *CBAM* secara berurutan menggabungkan *channel* dan *spatial attention* . Pendekatan ini memungkinkan *CBAM* untuk mengidentifikasi fitur penting berupa *channel* sekaligus menentukan lokasi atau *spatial* objek, sehingga menghasilkan penyempurnaan fitur yang lebih akurat. Selain itu, *CBAM* memanfaatkan kombinasi *average pooling* dan *max pooling* dalam proses *channel attention* , yang memberikan representasi fitur lebih kaya dibandingkan *SENet* yang hanya menggunakan *average pooling*. Dengan tambahan arsitektur *CBAM*, berbagai model CNN menunjukkan peningkatan performa signifikan, misalnya pada eksperimen klasifikasi di *ImageNet-1K*. Model *ResNet50* yang dilengkapi *CBAM* mencatatkan error top-1 sebesar 22,66%, lebih rendah dibandingkan *SENet* yang hanya mencapai 23,14%. Hal ini menunjukkan bahwa *CBAM* lebih unggul dibandingkan *SENet* dalam meningkatkan akurasi model.

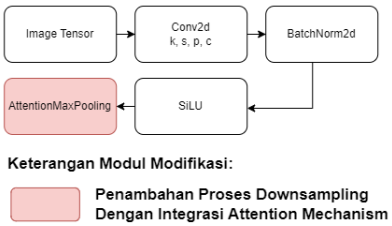
Pada penelitian ini, diajukan metode *downsampling* baru yaitu *attention max pooling*. Pada *attention max pooling* dilakukan penambahan modul *CBAM* sebelum proses *downsampling* menggunakan *layer max pooling*. Hal ini bertujuan untuk mengatasi potensi hilangnya informasi penting akibat penambahan proses *downsampling* pada modul *convolution*. Mekanisme ini diterapkan untuk memastikan bahwa fitur-fitur yang relevan dan signifikan diprioritaskan, sehingga kualitas representasi fitur tetap optimal meskipun dilakukan pengurangan resolusi *feature map*. Gambar 8, menampilkan perbandingan ilustrasi mekanisme *max pooling* dan *attention max pooling*.



Gambar 8. Perbandingan *Attention Max Pooling* dan *Max Pooling*

Attention max pooling dimulai dengan memproses hasil *feature map* dari proses konvolusi menggunakan modul *CBAM*. Module tersebut akan mempertahankan *feature* yang relevan dan mengurangi nilai dari *feature* yang tidak relevan. Kemudian, hasil *feature map* dari modul *CBAM* akan diproses menggunakan *max pooling*, sehingga *feature* yang relevan tetap bisa dipertahankan setelah proses *downsampling*. Gambar 9, menujukkan arsitektur dari modul *convolution* yang telah ditambahkan *pooling layer* berupa *attention max pooling*.

P-ISSN : 2089-3353
E-ISSN : 2808-9162



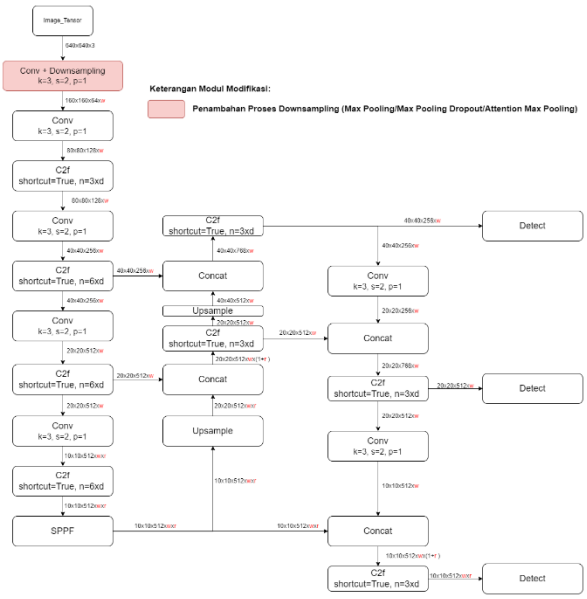
Gambar 9. Arsitektur Modifikasi Modul *Convolution* Dengan *Attention Max Pooling*

3. Hasil dan Pembahasan

Penelitian ini mengevaluasi efektivitas berbagai metode *downsampling* dan kombinasi dengan *ghost module* pada *YOLOv8* untuk meningkatkan efisiensi komputasi sekaligus mempertahankan akurasi deteksi objek melalui integrasi *attention mechanism*.

3.1. Hasil Eksperimen *Downsampling* Dan Integrasi *Attention Mechanism*

Pada bagian ini, Arsitektur *YOLOv8n* dimodifikasi dengan metode *downsampling* pada *layer* pertama, yaitu *max pooling*, *max pooling dropout*, dan *attention max pooling* seperti yang terlihat pada Gambar 10.



Gambar 10. Arsitektur Modifikasi *YOLOv8* Dengan *Downsampling*

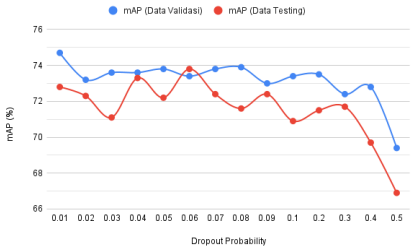
Tabel 1, menyajikan hasil evaluasi performa dan beban komputasi berbagai variasi model yang menggunakan metode *downsampling*, baik dalam bentuk standar maupun yang dikombinasikan dengan teknik lainnya seperti *dropout* dan *attention mechanism*. Data pada tabel mencakup nilai *mAP* untuk data validasi dan *testing*, serta jumlah *GFLOPs* yang dibutuhkan oleh setiap model. *mAP* digunakan sebagai metrik utama untuk mengukur akurasi deteksi objek, dengan nilai yang lebih tinggi menunjukkan kemampuan model yang lebih baik dalam mengenali objek secara akurat. Sementara itu, *GFLOPs* menunjukkan beban komputasi model, yang berkaitan langsung dengan efisiensi pengolahan data pada perangkat keras dengan sumber daya terbatas.

Tabel 1. Perbandingan Performa Dan Beban Komputasi Model Menggunakan Metode *Downsampling* dan *Attention Mechanism*

Model	*Dropout Probability*	*mAP* (Data Validasi)	*mAP* (Data *Testing*)	*GFLOPs*
YOLOv8n Baseline	-	73.1	71.1	8.09
YOLOv8n + Max Pooling	-	74.3	71.9	2.10
YOLOv8n + Max Pooling Dropout	0.01	74.7	72.8	2.10
YOLOv8n + Max Pooling Dropout	0.02	73.2	72.3	2.10
YOLOv8n + Max Pooling Dropout	0.03	73.6	71.1	2.10
YOLOv8n + Max Pooling Dropout	0.04	73.6	73.3	2.10
YOLOv8n + Max Pooling Dropout	0.05	73.8	72.2	2.10
YOLOv8n + Max Pooling Dropout	0.06	73.4	73.8	2.10
YOLOv8n + Max Pooling Dropout	0.07	73.8	72.4	2.10
YOLOv8n + Max Pooling Dropout	0.08	73.9	71.6	2.10
YOLOv8n + Max Pooling Dropout	0.09	73	72.4	2.10
YOLOv8n + Max Pooling Dropout	0.1	73.4	70.9	2.10
YOLOv8n + Max Pooling Dropout	0.2	73.5	71.5	2.10
YOLOv8n + Max Pooling Dropout	0.3	72.4	71.7	2.10
YOLOv8n + Max Pooling Dropout	0.4	72.8	69.7	2.10
YOLOv8n + Max Pooling Dropout	0.5	69.4	66.9	2.10
YOLOv8n + Attention Max Pooling	-	74.4	72.1	2.13

P-ISSN : 2089-3353
E-ISSN : 2808-9162

Pada teknik *max pooling dropout* dengan berbagai probabilitas *dropout*, terlihat bahwa penggunaan *dropout* dengan nilai 0.01 hingga 0.09 umumnya mampu mempertahankan atau meningkatkan performa model.



Gambar 11. Grafik Perbandingan Nilai *mAP* Model Berdasarkan Probabilitas *Dropout* Pada Metode *Max Pooling Dropout*

Gambar 11, menggambarkan perubahan nilai *mAP* pada data validasi dan data *testing* terhadap variasi probabilitas *dropout*. Nilai *mAP* tertinggi pada data *testing* dicapai pada probabilitas *dropout* 0.06, yaitu sebesar 73,8%. Namun, saat *dropout* meningkat lebih dari 0.1, *mAP* menurun secara bertahap. Penurunan paling signifikan terjadi pada *dropout* dengan nilai 0.5, mendapatkan *mAP* hanya 69,4% pada data validasi dan 66,9% pada data *testing*, yang menunjukkan bahwa *dropout* dengan nilai yang cukup tinggi menyebabkan hilangnya informasi penting dalam data, sehingga mengurangi performa model secara drastis.

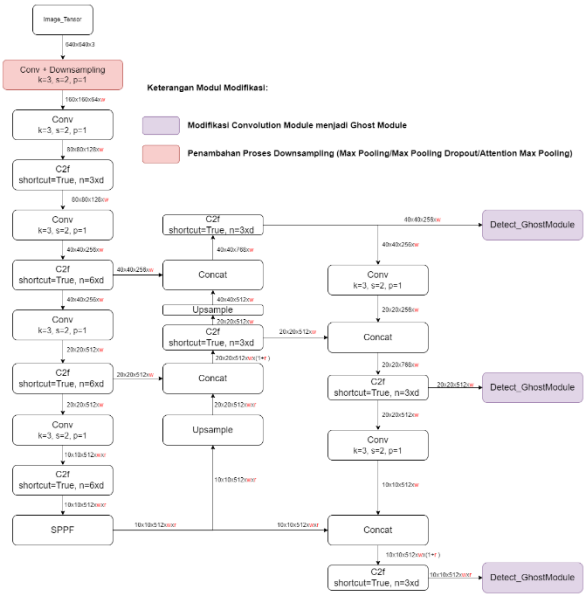
Pada metode *attention max pooling*, model mencapai nilai *mAP* 74,4% pada data validasi dan 72,1% pada data *testing*, dengan sedikit peningkatan beban komputasi menjadi 2.13 *GFLOPs*. Hal ini menunjukkan bahwa *attention mechanism* membantu mempertahankan informasi penting selama proses *downsampling*, memberikan performa yang stabil dengan beban komputasi yang relatif rendah.

Peningkatan performa pada *attention max pooling* dibandingkan dengan *max pooling dropout* dapat dijelaskan oleh perbedaan mendasar dalam cara kedua metode ini memproses informasi. Dalam *attention max pooling*, mekanisme *attention* secara adaptif memberikan bobot lebih besar pada fitur yang dianggap relevan, sementara fitur yang kurang signifikan ditekan. Pendekatan ini memungkinkan model untuk secara efektif mempertahankan informasi yang penting untuk deteksi objek, sehingga menghasilkan prediksi yang lebih akurat.

Sebaliknya, pada *max pooling dropout*, penghapusan acak node selama pelatihan mengurangi risiko overfitting dengan meningkatkan kemampuan generalisasi. Namun, metode ini tidak memiliki selektivitas adaptif seperti pada *attention* . Akibatnya, pada probabilitas *dropout* tertentu, informasi yang relevan dapat ikut dihapus, sehingga mengurangi akurasi model. Hal ini terutama terlihat pada nilai *dropout* yang lebih tinggi, di mana kehilangan informasi menjadi lebih signifikan. Dengan demikian, *attention max pooling* menawarkan pendekatan yang lebih terarah dan efisien dalam mempertahankan informasi penting, menghasilkan performa yang lebih baik tanpa mengorbankan stabilitas atau efisiensi komputasi secara berlebihan.

3.2. Hasil Kombinasi *Ghost Module*, *Downsampling* Dan Integrasi *Attention Mechanism*

Pada bagian ini, Arsitektur *YOLOv8* dimodifikasi dengan metode *downsampling*, integrasi *attention mechanism* dan *ghost module* seperti yang terlihat pada Gambar 12.



Gambar 12. Arsitektur Modifikasi *YOLOv8* Dengan *Downsampling*, *Attention Mechanism* Dan *Ghost Module*

Berdasarkan Tabel 2, *ghost module* berhasil mengurangi *GFLOPs* model secara signifikan dibandingkan *baseline YOLOv8n*. Penambahan *ghost module* saja menghasilkan *GFLOPs* sebesar 6.67, jauh lebih rendah dari *baseline*, dengan nilai *mAP* sebesar 73% pada data validasi dan 73.1% pada data *testing*. Ini menunjukkan bahwa *ghost module* mampu mengurangi beban komputasi model dengan tetap mempertahankan akurasi deteksi dan meningkat generalisai karena rentang *mAP* data validasi dan data *testing* tidak jauh.

Ghost module secara signifikan mengurangi *GFLOPs* karena prinsip kerjanya yang menggantikan sebagian besar operasi konvolusi dengan operasi yang lebih sederhana berupa *depthwise convolution* [11]. *Ghost module* menghasilkan fitur tambahan melalui transformasi linier dari fitur utama, sehingga mengurangi jumlah operasi matriks intensif tanpa kehilangan informasi penting. Dengan desain ini, *ghost module* mempertahankan representasi fitur yang cukup untuk mendeteksi objek secara akurat, meskipun menggunakan komputasi yang lebih efisien. Akurasi model tetap tinggi karena *ghost module* secara strategis mempertahankan informasi penting melalui kombinasi fitur utama dan fitur sintetis yang dihasilkan dari depthwise *convolution*, memastikan kinerja deteksi tetap stabil meskipun beban komputasi menurun secara signifikan.

P-ISSN : 2089-3353
E-ISSN : 2808-9162

Tabel 2. Perbandingan Performa dan Beban Komputasi Model Menggunakan Metode *Ghost Module*, *Downsampling* Dan *Attention Mechanism*

Model	*Dropout Probability*	*mAP* (Data Validasi)	*mAP* (Data *Testing*)	*GFLOPs*
YOLOv8n + Ghost Module	-	73	73.1	6.67
YOLOv8n + Ghost Module + Max Pooling Dropout	0.06	72.8	71.4	1.75
YOLOv8n + Ghost Module + Max Pooling	-	73.6	71.9	1.75
YOLOv8n + Ghost Module + Attention Max Pooling	-	73.3	73.2	1.77

Ketika *ghost module* dikombinasikan dengan berbagai metode *downsampling*, variasi performa model dapat diamati. Penggunaan *max pooling* sebagai metode *downsampling* menghasilkan *mAP* sebesar 73.6% pada data validasi dan 71.9% pada data *testing*, dengan beban komputasi yang rendah, yaitu 1.75 *GFLOPs*. Namun, selisih *mAP* yang cukup besar antara data validasi dan *testing* menunjukkan bahwa pola deteksi objek yang dihasilkan kurang general dan tidak stabil terhadap data baru.

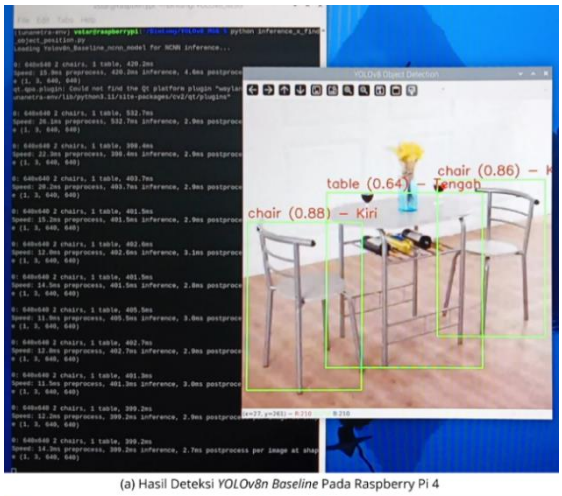
Ketika *max pooling dropout* dikombinasikan dengan *ghost module* menggunakan probabilitas sebesar 0.06, *mAP* yang diperoleh turun menjadi 72.8% pada data validasi dan 71.4% pada data *testing*. Meskipun *dropout* bertujuan untuk mencegah *overfitting*, kombinasi ini justru tidak memberikan keunggulan signifikan dibandingkan *max pooling* tanpa *dropout*. Selain itu, selisih antara *mAP* validasi dan *testing* tetap cukup besar, menunjukkan bahwa stabilitas generalisasi model belum optimal.

Penambahan *attention mechanism* melalui metode *attention max pooling* memberikan hasil yang lebih baik dan stabil. Kombinasi *ghost module* dengan *attention max pooling* menghasilkan *mAP* sebesar 73.3% pada data validasi dan 73.2% pada data *testing*, dengan beban komputasi sebesar 1.77 *GFLOPs*. Perbedaan *mAP* yang sangat kecil antara data validasi dan *testing* menunjukkan bahwa model mampu menghasilkan pola deteksi yang lebih general dan konsisten di berbagai data. Hal ini mengindikasikan bahwa *attention mechanism* membantu mempertahankan informasi penting selama proses *downsampling*, sehingga meningkatkan stabilitas dan akurasi model meskipun terjadi reduksi dimensi.

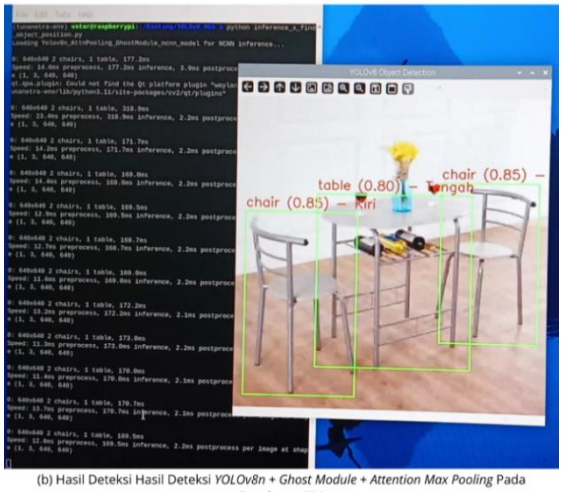
Dari analisis ini, dapat disimpulkan bahwa kombinasi *ghost module* dan *attention max pooling* adalah yang paling optimal. Dengan stabilitas performa yang baik pada data validasi dan *testing* serta beban komputasi yang tetap rendah, pendekatan ini menunjukkan potensi besar untuk aplikasi deteksi objek dalam kondisi sumber daya komputasi yang terbatas.

3.2. Pengujian Waktu Inference Model

Pengujian waktu *inference* dalam penelitian ini dilakukan untuk mengevaluasi efisiensi model dalam mendeteksi objek pada perangkat keras dengan keterbatasan daya komputasi, seperti Raspberry Pi 4. Pengujian dilakukan menggunakan input berupa video yang sama untuk memastikan hasil perbandingan antara *YOLOv8n* baseline dengan model hasil optimasi tetap valid. Untuk model hasil optimasi yang digunakan berupa *YOLOv8n* yang telah ditingkatkan menggunakan *ghost module* dan *attention max pooling*. Skenario pengujian melibatkan pemrosesan video secara *real-time* pada Raspberry Pi 4. Hal ini mencerminkan kondisi penggunaan model yang sesungguhnya, di mana sistem diharapkan mampu memberikan deteksi objek secara langsung. Faktor eksternal seperti pencahayaan dan jarak objek tidak divariasikan dalam pengujian ini untuk menjaga konsistensi hasil dan fokus pada evaluasi waktu *inference*.



(a) Hasil Deteksi *YOLOv8n Baseline* Pada Raspberry Pi 4



(b) Hasil Deteksi Hasil Deteksi *YOLOv8n + Ghost Module + Attention Max Pooling* Pada Raspberry Pi 4

Gambar 13. Perbandingan Waktu Inference *YOLOv8n Baseline* dan *YOLOv8n* Setelah Optimasi Pada Raspberry Pi 4

Gambar 13, menunjukkan hasil deteksi pada Raspberry Pi 4, dimana *YOLOv8n baseline* membutuhkan rata-rata waktu *inference* sebesar 401,56 ms, sedangkan model hasil optimasi hanya memerlukan 170,33 ms. Hal ini menunjukkan bahwa waktu *inference* berhasil dikurangi hingga 57,6%. Penurunan waktu *inference* ini mencerminkan keberhasilan integrasi *ghost module* dan *attention max pooling*, yang terbukti mampu menyederhanakan komputasi internal model tanpa mengurangi kemampuan deteksi. Efisiensi yang dicapai melalui optimasi ini menjadikan model lebih sesuai untuk aplikasi *real-time*, khususnya pada perangkat keras dengan keterbatasan daya komputasi seperti Raspberry Pi 4.

Pengurangan waktu *inference* yang signifikan ini memiliki dampak yang sangat penting, terutama dalam mendukung sistem bantuan visual bagi tunanetra yang memerlukan respons cepat dan akurat untuk mengidentifikasi objek di lingkungan sekitar. Kecepatan pemrosesan yang lebih tinggi memastikan bahwa informasi yang diterima oleh pengguna tetap relevan dalam situasi *real-time*, sehingga meningkatkan efektivitas sistem secara keseluruhan.

4. Kesimpulan

Penelitian ini menunjukkan bahwa penggunaan metode *ghost module* dan *downsampling* pada model *YOLOv8n* secara efektif mengurangi *GFLOPs*, sehingga waktu inferensi menjadi lebih sesuai dengan keterbatasan perangkat keras seperti Raspberry Pi 4. Optimasi yang dilakukan mampu menurunkan waktu inferensi hingga 57,6%, dari 401,56 ms pada model *baseline* menjadi 170,33 ms pada pengujian data *testing* berupa video, yang relevan untuk aplikasi dunia nyata seperti pendeteksian objek secara *real-time*. Pengurangan waktu inferensi ini berdampak signifikan pada pengalaman pengguna, karena memungkinkan respons sistem yang lebih cepat dan akurat dalam memberikan informasi tentang objek di sekitar mereka.Selain itu, penerapan *attention mechanism* melalui *attention max pooling* terbukti meningkatkan akurasi deteksi model. Kombinasi *ghost module* dan *attention max pooling* menghasilkan peningkatan *mAP* yang lebih tinggi dan stabil, dengan akurasi pada data pengujian meningkat hingga 1,3% dibandingkan metode *max pooling* standar. Penelitian ini membuka peluang untuk implementasi model yang lebih efisien pada perangkat keras dengan keterbatasan komputasi.

Daftar Rujukan

[1] Rahmawati, R.Y., Sunandar, A.: Peningkatan Keterampilan Orientasi dan Mobilitas melalui Penggunaan Tongkat bagi Penyandang Tunanetra. J. ORTOPEDAGOGIA. 4, 100–103 (2018). https://doi.org/10.17977/um031v4i12018p100
[2] A. Fauroq, R. Alfita, D.R.: Rancang Bangun Tongkat Cerdas Untuk Penyandang Tunanetra Berbasis Mikrokontroler Menggunakan Fuzzy Logic metode Sugeno. J. Tek. Elektro dan Komput. TRIAC. 5, (2018)
[3] Tikasni, E., Utami, E., Ariatmanto, D.: Analisis Akurasi Object Detection Menggunakan Tensorflow Untuk Pengenalan Bahasa Isyarat Tangan Menggunakan Metode SSD. J. Fasilkom. 14, 385–393 (2024)
[4] Fan, Y., Mao, S., Li, M., Wu, Z., Kang, J.: CM-YOLOv8: Lightweight YOLO for Coal Mine Fully Mechanized Mining Face. Sensors. 24, (2024). https://doi.org/10.3390/s24061866
[5] Wu, Y., Han, Q., Jin, Q., Li, J., Zhang, Y.: LCA-YOLOv8-Seg: An Improved Lightweight YOLOv8-Seg for Real-Time Pixel-Level Crack Detection of Dams and Bridges. Appl. Sci. 13, (2023). https://doi.org/10.3390/app131910583
[6] Roboflow, 2024. *Indoor Object Dataset*. [Online] (Updated 7 November 2024) Available at: https://universe.*roboflow*.com/csgitk/indoor_object_ta/dataset/10 [Accessed 20 November 2024].
[7] Chien, C.-T., Ju, R.-Y., Chou, K.-Y., Chiang, J.-S.: YOLOv8-AM: YOLOv8 with Attention Mechanisms for Pediatric Wrist Fracture Detection. 1–25 (2024)
[8] Wijanarko, R.G., Pradana, A.I., Hartanti, D.: IMPLEMENTASI DETEKSI DRONE MENGGUNAKAN YOLO (You Only Look Once). J. Fasilkom. 14, 437–442 (2024)
[9] Terven, J., Córdova-Esparza, D.M., Romero-González, J.A.: A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. Mach. Learn. Knowl. Extr. 5, 1680–1716 (2023). https://doi.org/10.3390/make5040083
[10] Ultralytics, 2024. *Ultralytics YOLOv8*. [Online] (Updated 27 September 2024) Available at: https://docs.ultralytics.com/models/yolov8/#performance-metrics [Accessed 20 November 2024].
[11] Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., Xu, C.: GhostNet: More features from cheap operations. Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. 1577–1586 (2020). https://doi.org/10.1109/CVPR42600.2020.00165
[12] Emmert-Streib, F., Yang, Z., Feng, H., Tripathi, S., Dehmer, M.: An Introductory Review of Deep Learning for Prediction Models With Big Data. Front. Artif. Intell. 3, 1–23 (2020). https://doi.org/10.3389/frai.2020.00004
[13] Wu, H., Gu, X.: Max-pooling dropout for regularization of convolutional neural networks. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). 9489, 46–54 (2015). https://doi.org/10.1007/978-3-319-26532-2_6
[14] Woo, S., Park, J., Lee, J.Y., Kweon, I.S.: CBAM: Convolutional block attention module. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). 11211 LNCS, 3–19 (2018). https://doi.org/10.1007/978-3-030-01234-2_1
[15] Hu, J., Shen, L., Albanie, S., Sun, G., Wu, E.: Squeeze-and-Excitation Networks. IEEE Trans. Pattern Anal. Mach. Intell. 42, 2011–2023 (2020). https://doi.org/10.1109/TPAMI.2019.2913372