JITET (Jurnal Informatika dan Teknik Elektro Terapan)
Vol. 14 No. 3, pISSN: 2303-0577 eISSN: 2830-7062 http://dx.doi.org/10.23960/jitet.v14i3.11023

ANALISIS OPTIMASI MODEL YOLOv8 UNTUK DETEKSI JENIS AWAN PADA JETSON NANO

Arfany Dhimas Muftareza[1*], Agustina Rachmawardani[1], Hapsoro Agung Nugroho[2], Marzuki Sinambela[3]

[1,2]Program Studi Instrumentasi-MKG, Sekolah Tinggi Meteorologi Klimatologi dan Geofisika, Indonesia; Jalan Meteorologi No 5, Tanah Tinggi, Tangerang, Kota Tangerang, Banten.

Keywords:
Optimasi;
YOLOv8;
Jetson Nano;
Jenis Awan.

Corespondent Email:
arfanydhimuf@gmail.com

Abstrak. Penerapan deteksi jenis awan berbasis *deep learning* pada perangkat *edge* menghadapi keterbatasan sumber daya komputasi yang dapat menurunkan efisiensi inferensi. Penelitian ini menganalisis optimasi model YOLOv8 untuk deteksi 10 jenis awan pada NVIDIA Jetson Nano melalui konversi model ke format TorchScript, ONNX, NCNN, dan MNN dengan presisi FP32 dan FP16. Dataset yang digunakan terdiri atas 4.916 citra, kemudian melalui augmentasi menghasilkan 11.791 citra pelatihan. Model YOLOv8s dilatih selama 100 *epoch* menggunakan *batch size* 16, *learning rate* 0,001, optimizer AdamW, dan *pre-trained weight* YOLOv8s. Model menghasilkan *precision* 0,86557, *recall* 0,90721, F1-score 0,887, mAP50 0,92518, dan mAP50-95 0,73288, dengan bobot terbaik diperoleh pada *epoch* ke-42. Pengujian pada Jetson Nano menunjukkan NCNN FP16 memberikan performa komputasi terbaik dengan *processing time* 761,99 ms dan FPS 1,31, meningkat sekitar 6,9 kali dibandingkan baseline PyTorch sebesar 5.238,57 ms dan 0,19 FPS. Nilai mAP50 tetap 0,9927, sedangkan mAP50-95 sebesar 0,8683. Hasil tersebut menunjukkan bahwa NCNN FP16 merupakan format yang paling optimal untuk implementasi YOLOv8 pada Jetson Nano karena mampu meningkatkan efisiensi inferensi dengan penurunan akurasi yang relatif kecil.

Abstract. *The implementation of deep learning-based cloud type detection on edge devices faces computational resource limitations that can reduce inference efficiency. This study analyzes YOLOv8 model optimization for detecting 10 cloud types on the NVIDIA Jetson Nano through model conversion into TorchScript, ONNX, NCNN, and MNN formats using FP32 and FP16 precision. The dataset consisted of 4,916 images, which were augmented to produce 11,791 training images. The YOLOv8s model was trained for 100 epochs using a batch size of 16, learning rate of 0.001, AdamW optimizer, and YOLOv8s pre-trained weights. The trained model achieved a precision of 0.86557, recall of 0.90721, F1-score of 0.887, mAP50 of 0.92518, and mAP50-95 of 0.73288, with the best weights obtained at epoch 42. Testing on the Jetson Nano showed that NCNN FP16 achieved the best computational performance, with a processing time of 761.99 ms and 1.31 FPS, representing approximately a 6.9-fold improvement over the PyTorch baseline of 5,238.57 ms and 0.19 FPS. The mAP50 remained at 0.9927, while mAP50-95 reached 0.8683. These results indicate that NCNN FP16 provides the most optimal format for YOLOv8 deployment on the Jetson Nano by substantially improving inference efficiency with only a minor reduction in detection accuracy.*

1. PENDAHULUAN

Perkembangan kecerdasan buatan mendorong berbagai upaya untuk mengurangi subjektivitas pengamatan jenis awan. Tugas *image classification* dengan pendekatan berbasis *Convolutional Neural Network* (CNN) telah banyak digunakan pada tugas klasifikasi citra awan [1][2][3][4]. Namun, tugas *image classification* hanya mampu menentukan kategori objek tanpa mengetahui lokasi objek dalam citra, sehingga tugas *object detection* menjadi pendekatan yang lebih relevan karena mampu menghasilkan lokalisasi sekaligus klasifikasi objek secara bersamaan [5].

Algoritma *You Only Look Once* (YOLO) menjadi salah satu arsitektur *object detection* satu tahap yang populer karena mampu menghasilkan prediksi *bounding box* dan probabilitas kelas secara langsung dari satu kali *forward pass*, sehingga menghasilkan waktu inferensi yang cepat dengan akurasi yang kompetitif [6][7]. YOLOv8, yang dirilis oleh Ultralytics pada awal 2023, merupakan salah satu generasi YOLO yang menunjukkan peningkatan signifikan pada metrik *precision*, *recall*, dan *mean Average Precision* (mAP) dibandingkan versi sebelumnya, sekaligus mempertahankan efisiensi komputasi [8], dan telah diadopsi pada berbagai domain aplikasi seperti deteksi kerusakan jalan, bidang medis, lalu lintas, hingga pertanian [9][10][11][12].

Penerapan model *deep learning* pada perangkat lapangan menuntut solusi komputasi yang ringkas dan hemat daya. *Edge computing* menjadi paradigma yang relevan karena memindahkan proses komputasi lebih dekat ke sumber data, sehingga mengurangi latensi dan ketergantungan terhadap konektivitas jaringan terpusat [13][14]. NVIDIA Jetson Nano merupakan salah satu *single-board computer* yang umum digunakan untuk inferensi model *deep learning* pada *edge device* karena mendukung akselerasi GPU dengan konsumsi daya rendah [15]. Meski demikian, keterbatasan sumber daya komputasi pada Jetson Nano menjadikan model YOLOv8 sulit mencapai kecepatan inferensi yang memadai untuk kebutuhan pemantauan mendekati waktu nyata.

Untuk mengatasi keterbatasan tersebut, optimasi model menjadi tahap penting sebelum model diterapkan pada perangkat *edge*. Beberapa pendekatan optimasi yang umum digunakan meliputi konversi model ke format inferensi yang lebih ringkas. Studi terkait optimasi YOLO pada perangkat Jetson menunjukkan bahwa penggunaan *runtime* inferensi yang dioptimalkan dapat meningkatkan kecepatan inferensi secara signifikan sebesar 16,11% dibandingkan model yang tidak dioptimalkan, meskipun *trade-off* antara akurasi dan kecepatan tetap perlu dievaluasi secara empiris pada tiap kasus penggunaan [16]. Berdasarkan uraian tersebut, paper ini berfokus pada analisis optimasi model YOLOv8 untuk deteksi jenis awan pada perangkat Jetson Nano. Model YOLOv8 yang telah dilatih menggunakan dataset citra jenis awan dievaluasi dan dioptimasi melalui beberapa format ekspor (TorchScript, ONNX, NCNN, dan MNN) pada dua tingkat presisi (FP32 dan FP16), untuk memperoleh format yang memberikan keseimbangan terbaik antara akurasi deteksi dan efisiensi komputasi ketika dijalankan pada perangkat *edge* dengan sumber daya terbatas.

2. TINJAUAN PUSTAKA

2.1. Jenis Awan

Berdasarkan ketinggiannya dari permukaan laut hingga lapisan atas troposfer, awan diklasifikasikan menjadi tiga kelompok utama, yaitu awan rendah, awan menengah, dan awan tinggi, yang masing-masing ditentukan oleh rentang ketinggian di mana jenis awan tertentu paling sering terjadi [17]. Berdasarkan definisi jenis awan menurut WMO, karakteristik jenis awan dapat diringkas seperti yang disajikan pada tabel 1.

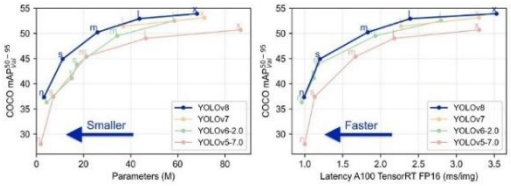
Tabel 1. Klasifikasi Awan Berdasarkan Ketinggian

Kategori	Tinggi (Tropis)	Jenis Awan	Karakteristik
Tinggi	6 – 18 km (20000 – 60000 ft)	Cirrus	putih, filamen halus, berserat, tipis, terpisah
		Cirrocumulus	riak kecil, bulat, teratur, tanpa bayangan, tipis, putih
		Cirrostratus	halo, tipis, jernih, menutupi sebagian langit, halus
Menengah	2 – 8 km (6500 – 25000	Altocumulus	bulat, bergulung, berbayang, putih keabu-abuan
		Altostratus	tanpa halo, matahari samar, abukebiruan, seragam lapisan lebar

Kategori	Tinggi (Tropis)	Jenis Awan	Karakteristik
	ft)	Nimbostratus	langit gelap pekat, tertutup sepenuhnya, hujan
Rendah	0 – 2 km (0 – 6500 ft)	Stratus	abu-abu, dasar datar, seragam, curah hujan ringan, matahari tidak terlihat.
		Stratocumulus	abu-abu keputihan, membulat, bergulung, tidak berserat, elemen besar, menyatu atau terpisah
		Cumulus	terpisah, padat, berbatas tegas, kembang kol
		Cumulonimbus	menara raksasa, landasan, dasar yang sangat gelap, vertikal ekstrem, virga

2.2. Model YOLOv8

YOLO merumuskan deteksi objek sebagai masalah regresi tunggal melalui satu *forward pass* jaringan saraf dengan memetakan citra masukan secara langsung ke koordinat *bounding box* dan probabilitas kelas [7]. Dibandingkan dengan pendahulunya [18], arsitektur YOLOv8 dirancang untuk mencapai keseimbangan yang lebih baik antara akurasi dan efisiensi komputasi, sebagaimana ditunjukkan dalam studi perbandingan pada dataset COCO yang dapat dilihat pada Gambar 2 [8]. Konfigurasi YOLOv8 ditentukan oleh parameter-parameter kunci seperti *depth_multiple, width_multiple*, dan *max_channels*, yang secara bersama-sama mengatur kedalaman jaringan, lebar saluran (*channel width*), dan kompleksitas model. Konfigurasi YOLOv8 terdiri dari tiga komponen utama: *backbone, neck*, dan *head* [19]. Gambar 2 memperlihatkan keseimbangan antara akurasi dan efisiensi komputasi tersebut.



Gambar 2. Trade-off Akurasi dan Efisiensi Komputasi

Bagian *backbone* bertugas mengekstraksi fitur dari citra masukan pada berbagai skala. Dimulai dari masukan berukuran 640×640×3, *backbone* menerapkan serangkaian lapisan konvolusi untuk melakukan *downsampling* bertahap pada peta fitur, yang diselingi dengan modul C2f yang dikonfigurasi menggunakan koneksi *shortcut* untuk menjaga aliran gradien. Bagian *neck* mengadopsi struktur *Path Aggregation Network* (PAN) yang menggabungkan fusi fitur jalur *top-down* menggunakan operasi *upsample*, sementara jalur *bottom-up* mengagregasi ulang fitur-fitur hasil fusi tersebut melalui blok konvolusi dan C2f tambahan. Bagian *head* merupakan *decoupled detection head* tanpa *anchor* yang memproses setiap skala keluaran dari *neck* secara independen melalui cabang Conv2d khusus untuk regresi *bounding box* dan klasifikasi.

2.3. Jetson Nano

Jetson Nano adalah sebuah *single-board computer* yang dikembangkan oleh NVIDIA yang dirancang untuk pengembangan proyek atau sistem kecerdasan buatan. Ilustrasi bentuk perangkat Jetson Nano dapat dilihat pada gambar 3 yang diambil langsung dari website NVIDIA *Corporation*. Jetson Nano dapat mendukung berbagai *framework* populer dan menyediakan antarmuka untuk menghubungkan perangkat lain, sehingga ideal digunakan dalam pengembangan sistem *embedded* [20].

Gambar 3. NVIDIA Jetson Nano

Perangkat ini dilengkapi dengan GPU NVIDIA Maxwell berisi 128 CUDA cores serta CPU quad-core ARM Cortex-A57 MPCore. Perangkat ini memiliki memori 4 GB LPDDR4 64-bit dengan bandwidth 25,6 GB/s dan penyimpanan internal 16 GB eMMC 5.1. Dari sisi multimedia, Jetson Nano mendukung video encode hingga 250 MP/sec dengan kemampuan menangani 4K @30 fps HEVC, beberapa konfigurasi 1080p, dan 720p pada berbagai frame rate. Untuk video decode, performanya

mencapai 500 MP/sec, mampu memproses 4K @60 fps HEVC hingga sembilan aliran 720p @60 fps. Jetson Nano juga mendukung kamera dengan 12 lanes MIPI CSI-2 D-PHY 1.1 berkecepatan 1,5 Gb/s per pair, serta konektivitas Gigabit Ethernet dan slot M.2 Key E. Output tampilan tersedia melalui HDMI 2.0 dan eDP 1.4, sementara antarmuka eksternal mencakup 4 port USB 3.0 dan USB 2.0 Micro-B. Selain itu, Jetson Nano menyediakan berbagai antarmuka I/O seperti GPIO, I2C, I2S, SPI, dan UART, dengan dimensi fisik 69,6 mm × 45 mm dan menggunakan 260-pin edge connector sebagai penghubung utama.

2.4. Optimasi Model

Optimasi model merupakan proses penyesuaian pada model dengan teknik tertentu untuk meningkatkan performa model secara optimal. Salah satu teknik optimasi model pada YOLO adalah *Half Precision Conversion*, yaitu teknik yang mengubah representasi parameter model dari format *floating point* 32-bit (FP32) menjadi 16-bit (FP16) untuk efisiensi memori dan komputasi, tanpa menurunkan akurasi model secara signifikan. Teknik *loss scaling* pada FP16 mampu menggeser nilai gradien kecil yang terpotong menjadi nol agar tetap berada dalam rentang representasi FP16 dengan menskalakan gradien, sehingga mencegah gradien menjadi nol dan menjaga akurasi model setara dengan pelatihan FP32 [21].

Berdasarkan standar IEEE 754, *Half Precision Conversion* mengubah representasi bilangan dari 32 bit menjadi 16 bit. Kedua format menggunakan struktur yang sama, yaitu 1 bit tanda (sign), eksponen bias, dan field mantissa, namun dengan parameter yang berbeda. FP32 memiliki eksponen 8 bit (bias 127) dan mantissa 23 bit, sedangkan FP16 memiliki eksponen yang dipersempit menjadi 5 bit (bias 15) dan mantissa 10 bit. Proses konversi ke format yang lebih sempit mengharuskan hasil dibulatkan sesuai atribut arah pembulatan yang berlaku, karena presisi dan rentang dinamis FP16 lebih terbatas dibanding FP32 [22].

2.5. Format Ekspor Model YOLOv8

2.5.1. PyTorch

PyTorch adalah *library deep learning* sumber terbuka yang menyediakan gaya pemrograman imperatif dan *Pythonic* dengan dukungan grafik komputasi dinamis, sehingga memudahkan proses *debugging* dan iterasi model, serta mendukung akselerator perangkat keras seperti GPU [23].

2.5.2. Torchscript

TorchScript adalah representasi antara (*intermediate representation*) dari model PyTorch yang memungkinkan model diubah menjadi format yang dapat diserialisasi dan dioptimasi untuk dieksekusi secara independen tanpa memerlukan *runtime* Python [23].

2.5.3. *Open Neural Network Exchange* (ONNX)

ONNX adalah format sumber terbuka standar yang dirancang untuk merepresentasikan model *machine learning* secara agnostik terhadap *framework*, sehingga memungkinkan interoperabilitas antar berbagai *framework* dan penerapan model di berbagai platform perangkat keras [24].

2.5.4. *Neural Computing and Neural Network* (NCNN)

NCNN adalah *framework* komputasi inferensi jaringan saraf tiruan yang dikembangkan oleh Tencent dan dioptimalkan secara khusus, tanpa dependensi pihak ketiga dan mampu berjalan lebih cepat dibandingkan *framework* sumber terbuka lain pada CPU perangkat *mobile* [25].

2.5.5. *Mobile Neural Network (MNN)*

MNN (*Mobile Neural Network*) adalah *inference engine deep learning* yang ringan (*lightweight*) dan efisien yang dikembangkan oleh Alibaba, dirancang khusus untuk mendukung inferensi dan pelatihan model *deep learning* pada perangkat *mobile* dan *edge* [26].

2.6. Evaluasi Optimasi Model YOLOv8

2.6.1. Precision

Precision mengukur akurasi dari seluruh prediksi positif yang dihasilkan oleh suatu model klasifikasi. Nilai *precision* berkisar antara 0 hingga 1, dengan nilai yang mendekati 1 menunjukkan bahwa sebagian besar prediksi positif model benar adanya. Secara matematis, precision dapat dinyatakan pada persamaan 1[27].

$$\text{Precision} = \frac{TP}{TP + FP} \qquad (1)$$

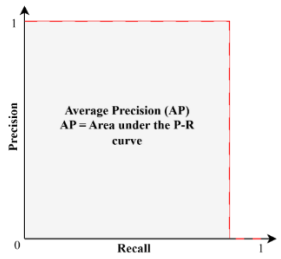
2.6.2. Recall

Recall mengukur proporsi data positif yang berhasil diidentifikasi dengan benar oleh model dari seluruh data positif yang tersedia. Nilai recall berkisar antara 0 hingga 1, dengan nilai mendekati 1 menunjukkan bahwa model mampu mengenali hampir seluruh data positif secara akurat. Secara matematis, *recall* dapat dinyatakan pada persamaan 2 [27].

$$\text{Recall} = \frac{TP}{TP + FN} \qquad (2)$$

2.6.3. mAP50

Mean Average Precision pada *threshold* 0,5 merupakan metrik evaluasi yang digunakan dalam tugas deteksi objek untuk mengukur akurasi model dalam mengidentifikasi objek dengan tingkat *Intersection over Union* minimal 0,5 antara prediksi dan *ground truth*. Secara matematis, *Average Precision* (AP) dihitung sebagai area di bawah kurva *precision-recall*, kemudian *mean Average Precision* (mAP) diperoleh dengan mengambil rata-rata AP dari seluruh kelas objek yang ada [27]. Ilustrasi nilai *average precision* dapat dilihat pada gambar 4.



Gambar 4. *Average Precision* (AP)

Rumus dasar perhitungan *Average Precision* (AP) dapat dinyatakan pada persamaann 3 [27]:

$$AP = \int_0^1 P(r)\, dr \qquad (3)$$

Selanjutnya, nilai mAP dihitung sebagai rata-rata dari nilai AP pada seluruh kelas. Nilai mAP50 berkisar antara 0 hingga 1, dengan nilai yang lebih tinggi menunjukkan performa model yang lebih baik dalam mendeteksi objek dengan akurasi tumpang tindih minimal 50%. Persamaan mAP dituliskan pada persmaan 4 [27]:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} A\,P_i \qquad (4)$$

2.6.4. mAP50-95

Mean Average Precision pada threshold 0,5 hingga 0,95 merupakan metrik evaluasi untuk mengukur akurasi model dalam mengidentifikasi objek dengan tingkat *Intersection over Union* minimal 0,5 hingga 0,95 antara prediksi dan *ground truth* dengan kenaikan sebesar 0,05. Secara matematis, mAP50-95 diperoleh dengan mengambil rata-rata dari nilai AP di seluruh ambang IoU yang dihitung, ditunjukan pada persamaan 5 [27].

$$mAP_{50-95} = \frac{1}{10}\sum_{i=0}^{9} A\,P_{0.5+i\times 0.05} \qquad (5)$$

2.6.5. *Processing Time*

Mengukur durasi yang dibutuhkan model untuk menyelesaikan prediksi pada satu citra input. Nilai ini penting karena menunjukkan efisiensi komputasi dan memengaruhi kecepatan sistem dalam aplikasi real-time.

2.6.6. *Frame Per Second* (FPS)

Mengukur jumlah frame yang dapat diproses model setiap detik. Semakin tinggi FPS, semakin lancar performa sistem dalam menangani aliran video atau data citra secara berkelanjutan.

2.6.7. *CPU Usage*

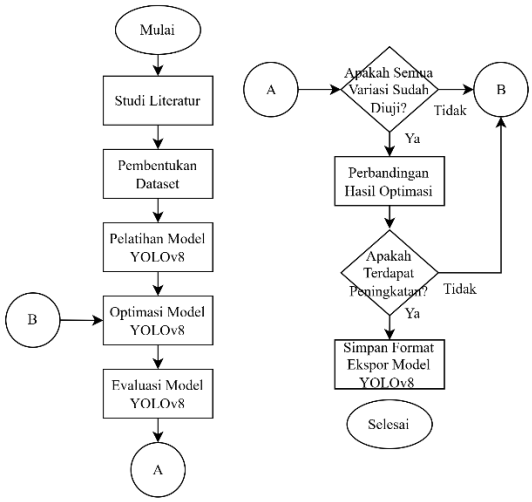
Mengukur persentase kapasitas prosesor yang sedang digunakan untuk menjalankan proses prediksi dan operasi pendukung. Nilai ini mencerminkan beban kerja CPU dan dapat digunakan untuk menilai efisiensi algoritma terhadap sumber daya komputasi.

2.6.8. *RAM Usage*

Mengukur jumlah memori utama (RAM) yang dipakai sistem untuk menyimpan data sementara selama proses prediksi berlangsung. Tingkat penggunaan RAM memengaruhi kemampuan sistem dalam menangani input berukuran besar dan stabilitas eksekusi model.

3. METODE PENELITIAN

Metode penelitian pada penelitian ini difokuskan pada tahap optimasi model YOLOv8, mengacu pada diagram alir penelitian yang ditunjukkan pada Gambar 5.



Gambar 5. Diagram Alir Penelitian

Tahap 1: Studi Literatur. Penelitian diawali dengan studi literatur mengenai klasifikasi jenis awan, algoritma YOLOv8, teknik optimasi model, serta karakteristik perangkat *edge computing* Jetson Nano sebagai dasar teoritis penelitian.

Tahap 2: Pembentukan Dataset. Dataset citra jenis awan dibentuk melalui pengambilan citra langsung menggunakan kamera dan *web scraping* dari sumber daring, kemudian melalui proses *splitting* (80% data latih dan 20% data validasi), augmentasi (rotasi ±5° dan *horizontal flip*), anotasi *bounding box*, serta pra-pemrosesan berupa *resize* ke resolusi 640×640 piksel dan koreksi orientasi otomatis.

Tahap 3: Pelatihan Model YOLOv8. Model dilatih menggunakan kombinasi *hyperparameter* (*pre-trained weitght* YOLOv8s, *epochs* 100, *batch size* 16, *dropout* 0.0, *optimizer* AdamW, dan *learning rate* 0,001) melalui *framework* Ultralytics, kemudian dievaluasi dengan metrik *precision*, *recall*, F1-score, mAP50, dan mAP50-95 untuk memperoleh bobot model terbaik sebagai baseline optimasi.

Tahap 4: Optimasi Model YOLOv8. Model dengan bobot terbaik dikonversi ke beberapa format ekspor, yaitu TorchScript, ONNX, NCNN, dan MNN, masing-masing pada dua tingkat presisi (FP32 dan FP16), sehingga menghasilkan delapan varian format optimasi yang dibandingkan terhadap baseline PyTorch. Setiap varian diuji secara bergiliran pada Jetson Nano menggunakan citra uji dan *ground truth* yang identik, kemudian dievaluasi berdasarkan delapan parameter: *Precision* (P), *Recall* (R), mAP50, mAP50-95, *Processing Time*, *Frame Per Second* (FPS), *CPU Usage*, dan *RAM Usage*.

Tahap 5: Perbandingan dan Evaluasi Peningkatan. Setelah seluruh variasi format selesai diuji, hasil optimasi dibandingkan secara menyeluruh antarformat pada kedelapan metrik evaluasi. Jika perbandingan menunjukkan belum ada peningkatan performa komputasi yang berarti tanpa penurunan akurasi, proses dikembalikan ke tahap pengujian optimasi untuk eksplorasi/verifikasi ulang.

Tahap 6: Penyimpanan Format Ekspor Terbaik. Format model yang memberikan keseimbangan terbaik antara akurasi deteksi dan efisiensi komputasi (waktu proses, FPS, penggunaan CPU, dan penggunaan RAM) disimpan sebagai model akhir yang akan diterapkan pada sistem deteksi jenis awan berbasis Jetson Nano, dan proses penelitian pada tahap optimasi ini dinyatakan selesai.

4. HASIL DAN PEMBAHASAN

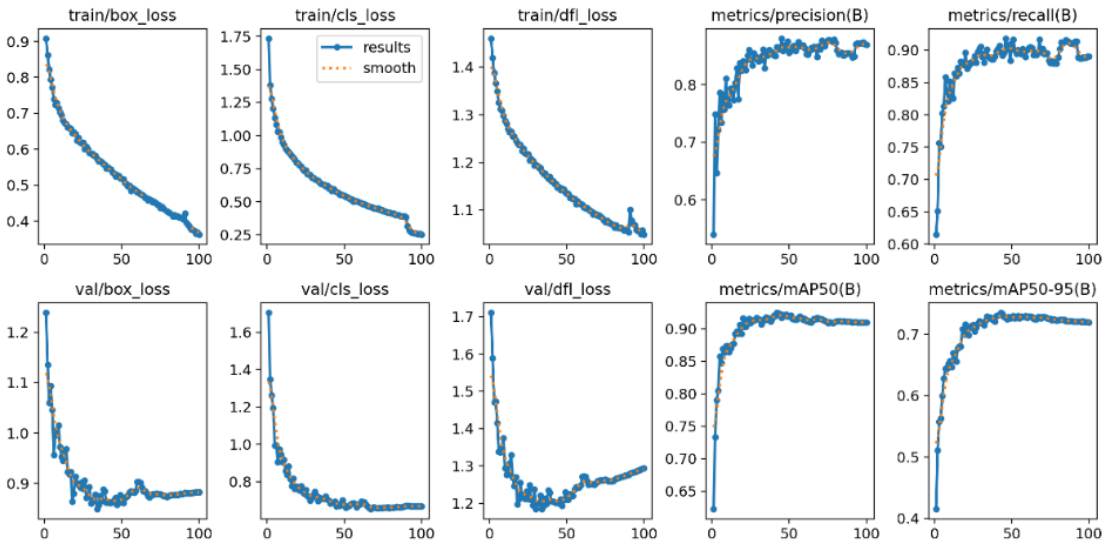
Pembentuka dataset dimulai dari roses pengumpulan data yang menghasilkan 4.916 gambar awan yang selanjutnya dikelompokkan ke dalam folder berdasarkan jenis awan yang berbeda. Dari total tersebut, 3.926 gambar dialokasikan untuk pelatihan dan 990 gambar dialokasikan untuk keperluan validasi. Dengan menerapkan teknik augmentasi, jumlah data pelatihan meningkat menjadi 11.791 gambar, sedangkan data validasi tetap. Setelah itu, seluruh data menjalani proses anotasi dan pra-pemrosesan untuk memastikan bahwa format dan kualitasnya konsisten. Rangkaian proses pembentukan data, mulai dari pengumpulan hingga proses anotasi, dirangkum secara teratur dalam Tabel 2.

Tabel 2. Pembentukan Dataset

Jenis Awan	Pengumpulan Data	Pembagian		Augmentasi		Anotasi	
		Train	Val	Train	Val	Train	Val
Altocumulus	472	375	97	1125	97	1125	97

Jenis Awan	Pengumpulan Data	Pembagian		Augmentasi		Anotasi	
		Train	Val	Train	Val	Train	Val
Altostratus	436	340	96	1022	96	1022	96
Cirrocumulus	437	347	90	1043	90	1106	98
Cirrostratus	472	381	91	1144	91	1144	91
Cirrus	487	400	87	1200	87	1212	88
Cumulonimbus	468	377	91	1131	91	1164	93
Cumulus	725	581	144	1745	144	4261	353
Nimbostratus	466	372	94	1118	94	1118	94
Stratocumulus	476	374	102	1124	102	1124	102
Stratus	477	379	98	1139	98	1142	98
Total Data	**4916**	**3926**	**990**	**11791**	**990**	**14418**	**1210**

Gambar 6 merupakan hasil pelatihan model YOLOv8 pada dataset yang telah dibentuk sebelumnya dengan kombinasi *hyperparameter* yang telah ditetapkan menggunakan *framework* Ultralytics secara *end to end*.



Gambar 6. Hasil Pelatihan Model YOLOv8

Hasil pelatihan model YOLOv8 pada dataset yang terdiri atas 10 kelas jenis awan menghasilkan nilai *precision* 0.86557, *recall* 0.90721, F1-*score* 0.887, mAP50 0.92518, dan mAP50-95 0.73288 dengan total waktu pelatihan 6 jam 41 menit 17 detik. Grafik pelatihan menunjukkan proses pembelajaran yang berlangsung secara stabil hingga akhir pelatihan. Pada data latih, seluruh komponen *loss* memperlihatkan tren penurunan secara konsisten, yang menunjukkan bahwa model mampu mengoptimalkan pembelajaran dengan mengurangi kesalahan dalam proses ekstraksi fitur, klasifikasi objek, dan lokalisasi *bounding box*. Sementara itu, *loss* pada data validasi mengalami penurunan yang cukup signifikan pada fase awal pelatihan, kemudian cenderung berfluktuasi dengan peningkatan *DFL loss* yang relatif kecil setelah *epoch* ke-42.

Peningkatan metrik evaluasi berlangsung paling cepat hingga sekitar *epoch* ke-30, yang menunjukkan terjadinya fase pembelajaran awal (*fast learning phase*) sebagai dampak dari penerapan *transfer learning* menggunakan *pre-trained weight* YOLOv8s. Pada tahap tersebut, model telah memanfaatkan representasi fitur yang diperoleh dari proses *pre-training* sehingga proses adaptasi terhadap karakteristik dataset awan berlangsung lebih cepat. Berdasarkan hasil evaluasi selama 100 *epoch*,

bobot terbaik diperoleh pada epoch ke-42, yang menghasilkan nilai metrik evaluasi tertinggi dibandingkan *epoch* lainnya. Kondisi tersebut menunjukkan bahwa model telah mencapai performa optimal pada *epoch* ke-42, sedangkan penambahan *epoch* hingga batas 100 tidak memberikan peningkatan performa yang berarti.

Tabel 3 menyajikan informasi hasil optimasi model YOLOv8 pada perangkat jetson nano secara detail.

Tabel 3. Hasil Optimasi Model YOLOv8 Pada Jetson Nano

Format	P	R	mAP50	mAP50-95	Processing Time	FPS	CPU Usage	RAM Usage
PyTorch	0.8815	0.9935	0.9950	0.8829	5238.57	0.19	5.70	391.53
Torchscript (FP32)	0.8501	0.9890	0.9927	0.8683	4821.88	0.21	2.00	378.60
Torchscript (FP16)	0.8499	0.9891	0.9927	0.8691	4810.42	0.21	3.50	395.21
ONNX (FP32)	0.8501	0.9890	0.9927	0.8683	1429.92	0.70	2.50	209.19
ONNX (FP16)	0.8502	0.9889	0.9927	0.8658	1316.04	0.71	3.50	233.11
NCNN (FP32)	0.8501	0.9890	0.9927	0.8683	766.34	1.30	2.30	283.23
NCNN (FP16)	0.8501	0.9886	0.9927	0.8683	761.99	1.31	3.00	284.43
MNN (FP32)	0.8500	0.9887	0.9927	0.8757	1472.03	0.68	2.00	248.25
MNN (FP16)	0.8500	0.9887	0.9927	0.8683	1468.52	0.68	3.30	262.52

Baseline PyTorch. Format PyTorch menghasilkan akurasi tertinggi di antara seluruh varian, dengan mAP50 sebesar 0,9950 dan mAP50-95 sebesar 0,8829, sekaligus nilai *precision* dan *recall* tertinggi. Namun performa komputasinya paling lemah, dengan *processing time* mencapai 5238,57 ms (FPS 0,19) serta penggunaan CPU tertinggi (5,70%). Hal ini menegaskan bahwa format representasi Python native, meski akurat, tidak efisien untuk inferensi pada perangkat *edge* berdaya rendah seperti Jetson Nano.

TorchScript (FP32 & FP16). Konversi ke TorchScript menurunkan sedikit akurasi (mAP50 turun menjadi 0,9927) namun memberikan penurunan *processing time* yang relatif kecil (menjadi sekitar 4,8 detik) dan sedikit peningkatan FPS (0,21). Perbedaan antara varian FP32 dan FP16 pada format ini tidak signifikan, mengindikasikan bahwa TorchScript belum banyak memanfaatkan optimasi *low-level* kernel komputasi pada arsitektur ARM/GPU Jetson Nano, sehingga percepatannya terbatas.

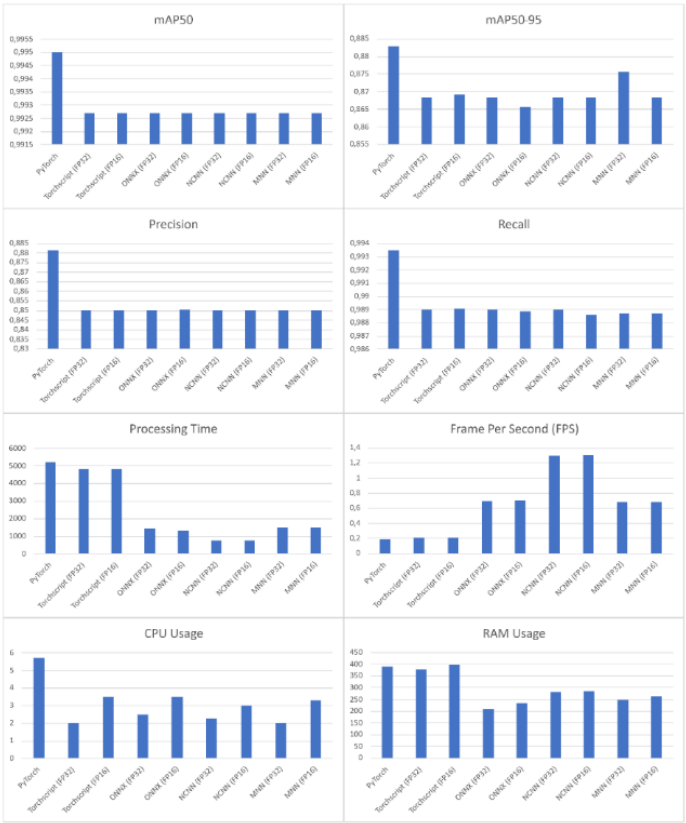
ONNX (FP32 & FP16). Format ONNX menghasilkan lompatan efisiensi yang jauh lebih besar dibandingkan TorchScript, dengan *processing time* turun menjadi 1429,92 ms (FP32) dan 1316,04 ms (FP16), serta FPS meningkat menjadi sekitar 0,70–0,71, lebih dari tiga kali lipat baseline PyTorch, sementara RAM usage juga turun signifikan menjadi kisaran 209–233 MB. Hal ini konsisten dengan karakteristik ONNX sebagai format antar-*framework* yang dioptimalkan untuk inferensi lintas *runtime*. Varian FP16 sedikit lebih cepat dibanding FP32, namun terjadi penurunan mAP50-95 yang sedikit lebih besar (0,8658 vs 0,8683), menunjukkan trade-off presisi numerik yang mulai terlihat pada metrik yang lebih ketat.

NCNN (FP32 & FP16). Format NCNN memberikan kinerja komputasi terbaik di antara seluruh varian yang diuji. Varian NCNN (FP16) mencatat *processing time* terendah sebesar

761,99 ms dan FPS tertinggi sebesar 1,31, meningkat sekitar 6,9 kali dibandingkan baseline PyTorch, dengan *precision* 0,8501, *recall* 0,9886, dan RAM usage 284,43 MB. NCNN (FP32) menghasilkan hasil yang hampir setara (762,34 ms, FPS 1,30), menunjukkan bahwa keunggulan performa NCNN terutama berasal dari efisiensi arsitektur *runtime*-nya yang dirancang tanpa dependensi pihak ketiga untuk CPU perangkat *edge*, bukan semata dari reduksi presisi FP16. Penurunan akurasi pada mAP50 (0,9927) dan mAP50-95 (0,8683) tergolong kecil dibanding baseline, sehingga NCNN (FP16) menjadi format dengan keseimbangan terbaik antara akurasi dan kecepatan inferensi.

MNN (FP32 & FP16). Format MNN menunjukkan efisiensi yang lebih baik dibanding PyTorch dan TorchScript, namun masih di bawah ONNX dan NCNN, dengan *processing time* pada kisaran 1468–1472 ms dan FPS sekitar 0,68. Menariknya, MNN (FP32) mencatat mAP50-95 tertinggi kedua (0,8757) di antara format hasil optimasi, mengindikasikan bahwa MNN relatif menjaga presisi lokalisasi pada IoU ketat lebih baik dibanding format lain, meski dengan konsekuensi kecepatan inferensi yang lebih rendah dibanding NCNN.

. Gambar 7 memperlihatkan hasil optimasi model YOLOv8 pada perangkat Jetson Nano, ditampilkan melalui perbandingan metrik evaluasi pada setiap formatnya.



Gambar 7. Optimasi Model YOLOv8 Pada Jetson Nano

Secara keseluruhan, seluruh format hasil optimasi menghasilkan nilai mAP50, *precision*, dan *recall* yang relatif seragam (mAP50 ≈ 0,9927 di semua format optimasi) dibandingkan baseline PyTorch (mAP50 = 0,9950), yang menunjukkan bahwa proses konversi format maupun reduksi presisi ke FP16 tidak menurunkan akurasi deteksi secara signifikan, sejalan dengan prinsip *half precision conversion* yang menjaga akurasi model mendekati pelatihan FP32 melalui mekanisme *loss scaling*. Sebaliknya, perbedaan performa komputasi antarformat sangat besar, rentang *processing time* membentang dari 5238,57 ms

(PyTorch) hingga 761,99 ms (NCNN FP16), atau peningkatan kecepatan inferensi mendekati 7 kali lipat.

Temuan ini menegaskan bahwa pemilihan *runtime*/format inferensi memiliki dampak jauh lebih besar terhadap efisiensi komputasi pada perangkat *edge* seperti Jetson Nano dibandingkan reduksi presisi numerik semata, selaras dengan berbagai studi optimasi YOLO pada perangkat Jetson yang melaporkan percepatan signifikan melalui konversi ke *runtime* inferensi teroptimasi. Berdasarkan keseluruhan hasil pengujian, NCNN (FP16) dipilih sebagai format optimal untuk diterapkan pada sistem deteksi jenis awan berbasis Jetson Nano, karena memberikan kombinasi *processing time* tercepat, FPS tertinggi, serta penurunan akurasi yang dapat diabaikan dibandingkan model baseline.

5. KESIMPULAN

Penelitian ini berhasil menganalisis optimasi model YOLOv8 untuk deteksi 10 jenis awan pada perangkat NVIDIA Jetson Nano melalui perbandingan format TorchScript, ONNX, NCNN, dan MNN pada presisi FP32 dan FP16. Model YOLOv8s yang dilatih menggunakan 4.916 citra awal dan augmentasi hingga 11.791 citra pelatihan menghasilkan precision sebesar 0,86557, recall 0,90721, F1-score 0,887, mAP50 0,92518, dan mAP50-95 0,73288, dengan bobot terbaik dicapai pada epoch ke-42. Hasil pengujian pada Jetson Nano menunjukkan bahwa optimasi melalui konversi format memberikan pengaruh yang jauh lebih besar terhadap efisiensi inferensi dibandingkan reduksi presisi semata. NCNN FP16 menjadi format terbaik dengan processing time 761,99 ms dan FPS 1,31, atau sekitar 6,9 kali lebih cepat dibandingkan baseline PyTorch yang membutuhkan 5.238,57 ms dengan FPS 0,19. Meskipun terjadi sedikit penurunan metrik akurasi, NCNN FP16 tetap mempertahankan mAP50 sebesar 0,9927 dan mAP50-95 sebesar 0,8683. Secara kualitatif, hasil tersebut menunjukkan bahwa efisiensi runtime merupakan faktor dominan dalam optimasi model pada perangkat edge, sedangkan penggunaan FP16 bukan satu-satunya faktor utama peningkatan kecepatan. Dengan demikian, NCNN FP16 dipilih sebagai format model paling sesuai untuk implementasi deteksi jenis awan berbasis YOLOv8 pada Jetson Nano karena memberikan keseimbangan terbaik antara akurasi deteksi dan efisiensi komputasi.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih khususnya kepada Dosen Pembimbing dan Dosen Penguji, serta pihak-pihak terkait yang telah memberi dukungan terhadap penelitian ini.

DAFTAR PUSTAKA

[1] M. N. Fikriansyah, H. A. Nugroho, and M. Sinambela, "Low Cloud Type Classification System Using Convolutional Neural Network Algorithm," *2022 7th Int. Conf. Informatics Comput. ICIC 2022*, pp. 1–6, 2022, doi: 10.1109/ICIC56845.2022.10006907.

[2] S. Kopeć, G. Duniec, B. Bochenek, and M. Figurski, "Artificial neural networks in automatic image classifications of cloud from ground-based observations using deep learning models," *Q. J. R. Meteorol. Soc.*, vol. 150, no. 765, pp. 5206–5224, 2024.

[3] M. Wang, S. Zhou, Z. Yang, and Z. Liu, "Clouda: A ground-based cloud classification method with a convolutional neural network," *J. Atmos. Ocean. Technol.*, vol. 37, no. 9, pp. 1661–1668, 2020.

[4] J. Zhang, P. Liu, F. Zhang, and Q. Song, "CloudNet: Ground-based cloud classification with deep convolutional neural network," *Geophys. Res. Lett.*, vol. 45, no. 16, pp. 8665–8672, 2018.

[5] Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu, "Object Detection with Deep Learning: A Review," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 30, no. 11, pp. 3212–3232, 2019, doi: 10.1109/TNNLS.2018.2876865.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788. doi: 10.1109/CVPR.2016.91.

[7] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *Mach. Learn. Knowl. Extr.*, vol. 5, no. 4, pp. 1680–1716, 2023, doi: 10.3390/make5040083.

[8] Ultralytics, "Ultralytics YOLOv8 Documentation," 2025. [Online]. Available: https://docs.ultralytics.com/models/yolov8/

[9] M. Bakirci, "Utilizing YOLOv8 for enhanced traffic monitoring in intelligent transportation systems (ITS) applications," *Digit. Signal Process.*, vol. 152, p. 104594, 2024, doi:

10.1016/j.dsp.2024.104594.
[10] U. A. Hasib, M. A. Raihan, J. Yang, U. A. Bhatti, C. S. Ku, and L. Y. Por, "YOLOv8 framework for COVID-19 and pneumonia detection using synthetic image augmentation," *Digit. Heal.*, vol. 11, 2025, doi: 10.1177/20552076251341092.
[11] D. Kumar and N. Muhammad, "Object detection in adverse weather for autonomous driving through data merging and YOLOv8," *Sensors*, vol. 23, no. 20, p. 8471, 2023.
[12] X. Li, Y. Wang, H. Zhang, J. Liu, M. Chen, and L. Zhao, "Crop pest identification and real-time monitoring system design based on improved YOLOv8s," *Sensors*, vol. 26, no. 2, p. 404, 2026, doi: 10.3390/s26020404.
[13] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surv. Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.
[14] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," no. c, pp. 1–10, 2016.
[15] P. S. Mittapalli, M. R. N. Tagore, P. A. Reddy, G. B. Kande, and Y. M. Reddy, "Deep learning based real-time object detection on jetson nano embedded gpu BT - Microelectronics, Circuits and Systems: Select Proceedings of Micro2021," Singapore: Springer Nature Singapore, 2023, pp. 511–521.
[16] T. P. Swaminathan, C. Silver, T. Akilan, and J. Kumar, "Benchmarking deep learning models on nvidia jetson nano for real-time systems: An empirical investigation," *Procedia Comput. Sci.*, vol. 260, pp. 906–913, 2025.
[17] WMO, "Definition of a cloud," 2017, *World Meteorological Organization*. [Online]. Available: https://cloudatlas.wmo.int/en/definition-of-a-cloud.html
[18] T. Ahmad, Y. Ma, M. Yahya, B. Ahmad, S. Nazir, and A. Haq, "Object Detection through Modified YOLO Neural Network," vol. 2020, 2020, doi: 10.1155/2020/8403262.
[19] P. Hidayatullah, N. Syakrani, M. R. Sholahuddin, T. Gelar, R. Tubagus, and P. N. Bandung, "YOLOv8 to YOLO11 : A Comprehensive Architecture In-depth Comparative Review," 2025.
[20] "Jetson Nano," 2026, *NVIDIA Corporation*. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano
[21] P. Micikevicius, S. Narang, J. Alben, and et al., "Mixed Precision Training," in *International Conference on Learning Representations (ICLR)*, 2018. [Online]. Available: https://arxiv.org/abs/1710.03740
[22] "IEEE Standard for Floating-Point Arithmetic," 2019. doi: 10.1109/IEEESTD.2019.8766229.
[23] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," *Adv. Neural Inf. Process. Syst.*, vol. 32, pp. 8024–8035, 2019, [Online]. Available: https://arxiv.org/abs/1912.01703
[24] A. D. Hirve, "Introduction to ONNX (Open Neural Network Exchange) framework," 2023. [Online]. Available: https://www.researchgate.net/publication/400950696
[25] Tencent, "NCNN: A high-performance neural network inference computing framework optimized for mobile platforms," 2017, *GitHub*. [Online]. Available: https://github.com/Tencent/ncnn
[26] X. Jiang *et al.*, *MNN: A Universal and Efficient Inference Engine*. 2020. doi: 10.48550/arXiv.2002.12418.
[27] L. He, Y. Zhou, L. Liu, W. Cao, and J. Ma, "Research on object detection and recognition in remote sensing images based on YOLOv11," pp. 1–25, 2025.