# "YOLO Algorithm: Revolusi Pendeteksian Objek dengan Kecepatan dan Akurasi"

Desi Arti

November 4, 2023

**Abstract**

Pendeteksian objek adalah salah satu tugas penting dalam pengolahan citra dan visi komputer. Pendeteksian objek bertujuan untuk mengidentifikasi dan memisahkan objek. YOLO ( You Only Look Once) merupakan salah satu algoritma pendeteksian objek yang baru-baru ini dikembangkan dan telah menunjukkan hasil yang sangat baik dalam hal kecepatan dan akurasi

## 1 Pendahuluan

Pendeteksian objek adalah tugas kunci dalam pengolahan citra dab pengenalan computer yang memiliki banyak aplikasi praktis. Terdapat peningkatan pesat dalam beberapa tahun terakhir, dalam pengembangan algoritma deteksi objek yang semakin akurat dan efisien. Salah satu terobiosan utama dalam bisang ini adalah Algoritma YOLO( You Only Look Once), yang telah merevolusi cara kita mendekati masalah pendeteksian objek. Sebelumnya, algoritma pendeteksian objek sering menghadapi kendala antara akurasi dan kecepatan. Metode konvensional yang membagi tugas menjadi dua tahap terpisah, yaitu deteksi dan klasifikasi,seringkali memiliki kinerja yang kurang memadai untuk aplikasi real-time. Algoritma ini juga memerlukan pemrosesan berulang untuk setiap daerah proposisi, yang membuat komputasi mahal. YOLO (You Only Look Once) hadir sebagai solusi untuk tantangan tersebut. Algoritma ini mengubah pendekatan deteksi objek dengan menyatukan deteksi dan klasifikasi menjadi satu tahap dan mengimplementasikan pemrosesan sekuensial yang efisien. Algoritma YOLO memungkinkan penggunaanya dalam aplikasi real-time seperti kendaraan otonom,pengawasan video dan lainnya. Dengan memahami keunggulan YOLO dalam mencapai kecepatan dan akurasi secara bersamaan, sangat penting untuk menyelidiki prinsip=prinsip dasar di balik algoritma ini dan bagaimana ia mengatasi beberapa kendala yang dihadapi oleh metode sebelumnya.Oleh karena itu dalam makalah ini akan dibahas tentang algoritma YOLO, implementasi dan dampaknya dalam revolusi pendeteksian objek. Penulis akan mengesplorasi perkerkembangan Algoritma YOLO dan bagaimana hal ini telah menginspirasi berbagai penelitian dibidang computer vision dan kecerdasan buatan. Dengan latar belakang ini, makalah ini bertujuan memberikan pemahaman yang kuat tentang algoritma YOLO, mengapa ia dianggap sebagai revolusi dalam pendeteksian object dan bagaimana ia dapat diterapkan berbagai konteks aplikasi.

## 2 Pendeteksian Objek

Sebelum munculnya algoritma YOLO, sebagian besar pendekatan pendeteksian objek menggunakan dua tahap terpisah: tahap deteksi dan tahap klasifikasi. Metode ini seringkali memerlukan waktu dan sumber daya komputasi yang signifikan. Beberapa dari metode ini termasuk R-CNN (Region-based Convolutional Neural Networks) dan Fast R-CNN.

### 2.1 YOLO (You Only Look Once)

YOLO, yang merupakan singkatan dari "You Only Look Once," adalah algoritma pendeteksian objek yang mengubah pendekatan konvensional dengan mengintegrasikan deteksi dan klasifikasi menjadi satu tahap tunggal. Prinsip dasar YOLO adalah bahwa ia "melihat" gambar hanya sekali dan secara simultan menghasilkan kotak pembatas (bounding boxes) yang membatasi objek-objek yang terdeteksi

serta mengklasifikasikannya. YOLO (You Only Look Once) merupakan algoritma populer yang telah merevolusi bidang visi komputer. Algoritma ini cepat dan efisien, menjadikannya pilihan yang sangat baik untuk tugas pendeteksian objek secara real-time. YOLO memandang deteksi target sebagai masalah regresi, yang berbeda dari algoritme CNN lainnya. YOLO dianggap sebagai pendekatan baru untuk deteksi objek, dan telah mencapai kinerja canggih pada berbagai tolok ukur.

Algoritma YOLO menggunakan jaringan saraf tiruan konvolusional (CNN) untuk melakukan deteksi objek. Model ini dibagi menjadi beberapa sel-sel yang bertanggung jawab atas berbagai aspek gambar. Hasil deteksi dan klasifikasi dikombinasikan dalam satu proses, yang membuatnya sangat efisien dan cepat.

### 2.2 Implementasi YOLO dalam Ultralytics

Ultralytics adalah kerangka kerja yang menyediakan alat dan sumber daya untuk bekerja dengan model YOLO, termasuk pelatihan model, evaluasi, dan penggunaan model yang sudah ada. Ultralytics memungkinkan pengguna untuk mengoptimalkan model YOLO sesuai dengan kebutuhan mereka dan menerapkannya dalam berbagai aplikasi, termasuk pengawasan video, kendaraan otonom, dan pengenalan objek dalam waktu nyata.

### 2.3 Dampak dan Aplikasi

Algoritma YOLO dan kerangka kerja Ultralytics telah membawa revolusi dalam pendeteksian objek dengan menggabungkan kecepatan dan akurasi. Mereka digunakan dalam berbagai aplikasi, termasuk pengawasan keamanan, kendaraan otonom, pemantauan lalu lintas, pengenalan wajah, dan banyak lagi. Kecepatan dan efisiensi dalam pendeteksian objek telah membuka berbagai peluang baru dalam penerapan teknologi pengenalan objek

## 3 Simulasi

### 3.1 Simulasi Model dengan Ultralytics

Berikut Langkah-langkah yang dilakukan dalam membuat simulasi model dengan menggunakan library Ultralytics dengan Bahasa pemograman phyton

a) Instalasi: Baris pertama kode menginstal perpustakaan Ultralytics menggunakan pip. !pip install ultralytics

b) Mengimpor Ultralytics YOLO: Baris kedua kode mengimpor kelas YOLO dari perpustakaan Ultralytics.

c) Membuat atau Memuat Model YOLO Dua bagian berikutnya dari kode menunjukkan bagaimana membuat model YOLO dari awal dan cara memuat model YOLO yang sudah dilatih. Untuk membuat model YOLO dari awal, tentukan konfigurasi model dengan menyediakan file YAML, seperti 'yolov8n.yaml'. Untuk memuat model YOLO yang sudah dilatih, sediakan jalur berkas bobot model, seperti 'yolov8n.pt'.

d) Membuat model YOLO baru dari awal model = YOLO('yolov8n.yaml')

e) Memuat model YOLO yang sudah dilatih (disarankan untuk pelatihan) model = YOLO('yolov8n.pt')

f) Melatih Model Kode tersebut melatih model YOLO menggunakan dataset yang didefinisikan dalam berkas konfigurasi 'coco128.yaml' selama 3 epoch.

g) Hasil pelatihan disimpan dalam variabel 'results'. results = model.train(data='coco128.yaml', epochs=3)

h) Evaluasi Model Kode tersebut mengevaluasi kinerja model pada set validasi. Hasil evaluasi disimpan dalam variabel 'results'. results = model.val()

i) Deteksi Objek pada Gambar Kode tersebut melakukan deteksi objek pada gambar yang ditentukan melalui URL ' https://ultralytics.com/images/bus.jpg' menggunakan model YOLO yang sudah dilatih. Hasil deteksi objek disimpan dalam variabel 'results'. results = model('https://ultralytics.com/images/bus.jpg')

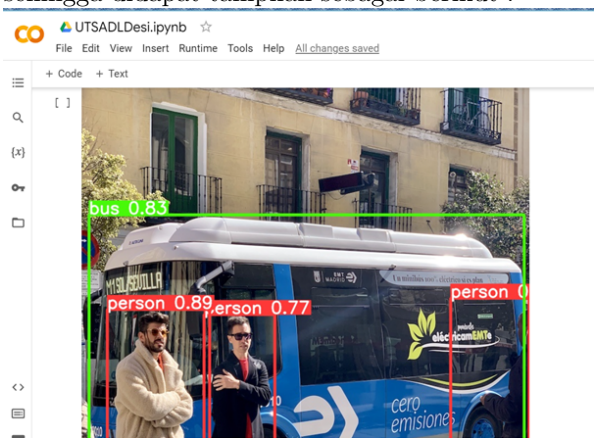
j) Ekspor Model ke Format ONNX - Bagian terakhir dari kode menghasilkan ekspor model YOLO yang sudah dilatih ke format ONNX, yang merupakan standar untuk merepresentasikan model pembelajaran mesin. success = model.export(format='onnx') Secara ringkas, kode ini memperlihatkan

bagaimana bekerja dengan model YOLO menggunakan kerangka kerja Ultralytics, mulai dari pembuatan model dan pelatihan hingga evaluasi dan ekspor model. Model YOLO yang digunakan di sini adalah 'yolov8n'.

### 3.2 Simulasi Model 1 dan 2

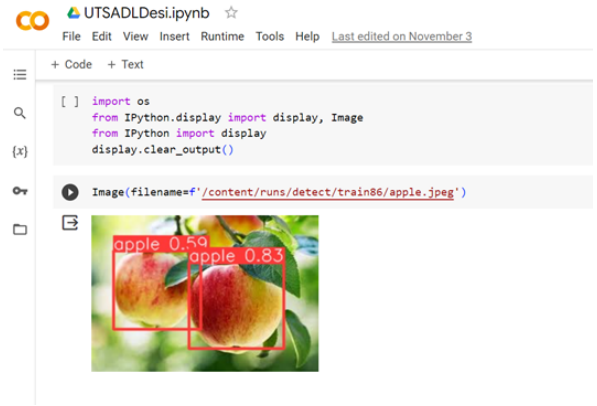
Berikut akan ditampilkan simulasi gambarnya

Dengan code model.predict('bus.jpg', save=True, imgsz=320, conf=0.5) yang diberikan bertujuan untuk menjalankan inferensi (deteksi objek) pada gambar 'bus.jpg' dengan beberapa argumen khusus sehingga didapat tampilan sebagai berikut :




Deteksi pada bus.

Dilakukan hal yang sama pada model ke 2 dilakukan deteksi model yang kedua model.predict('/content/apple.jpeg', save=True, imgsz=320, conf=0.5)




Deteksi pada apple.

## 4 Pembahasan

### 4.1 Model 1

Pada model 1 dapat dilihat bahwa waktu yang dibutuhkan untuk melakukan pra-pemrosesan (preprocessing) pada gambar atau data sebelum diberikan ke model adalah 4.2 milidetik.. Preprocessing dapat mencakup penyesuaian ukuran, normalisasi, dan transformasi lainnya agar data siap digunakan oleh model, waktu yang diperlukan untuk menjalankan model pada data (gambar atau input lainnya) dan menghasilkan prediksi atau output. Ini adalah tahap di mana model melakukan prediksi atau inferensi berdasarkan data yang diberikan kepadanya. Dalam konteks ini, waktu yang diperlukan untuk inferensi adalah 16.6 milidetik. Waktu yang diperlukan untuk melakukan pemrosesan pasca-inferensi pada hasil yang dihasilkan oleh model. Ini bisa termasuk pemrosesan hasil, pengubahan format, penyimpanan, atau langkah-langkah lainnya yang perlu dilakukan setelah model mengeluarkan prediksi. Dalam konteks ini, waktu yang diperlukan untuk pemrosesan pasca-inferensi adalah 2.8 milidetik per gambar atau data yang diolah. Semua waktu ini menggambarkan berapa lama waktu yang diperlukan

untuk menjalankan berbagai tahapan dalam proses yang Anda lakukan. Waktu yang dibutuhkan untuk masing-masing tahapan ini dapat bervariasi tergantung pada kompleksitas model, ukuran data input, dan perangkat keras yang digunakan. Dalam contoh ini, total waktu yang diperlukan untuk satu gambar atau data dapat dihitung dengan menggabungkan waktu pra-pemrosesan, inferensi, dan pemrosesan pasca-inferensi, yaitu 4.2 ms + 16.6 ms + 2.8 ms = 23.6 ms per gambar.

### 4.2 Model 2

Pada gambar ke 2 (apple),waktu yang dibutuhkan untuk melakukan pra-pemrosesan (preprocessing) pada gambar atau data sebelum diberikan ke model adalah 0.3 milidetik. Preprocessing dapat mencakup penyesuaian ukuran, normalisasi, dan transformasi lainnya agar data siap digunakan oleh model. Waktu yang diperlukan untuk menjalankan model pada data (gambar atau input lainnya) dan menghasilkan prediksi atau output. Ini adalah tahap di mana model melakukan prediksi atau inferensi berdasarkan data yang diberikan kepadanya. Dalam konteks ini, waktu yang diperlukan untuk inferensi adalah 7.6 milidetik. Waktu yang diperlukan untuk melakukan pemrosesan pasca-inferensi pada hasil yang dihasilkan oleh model. Ini bisa termasuk pemrosesan hasil, pengubahan format, penyimpanan, atau langkah-langkah lainnya yang perlu dilakukan setelah model mengeluarkan prediksi. Dalam konteks ini, waktu yang diperlukan untuk pemrosesan pasca-inferensi adalah 2.9 milidetik per gambar atau data yang diolah. Nilai kerugian atau loss function yang dihasilkan oleh model pada data tertentu. Nilai 0.0 menunjukkan bahwa pada data ini, model tidak menghasilkan kerugian atau loss. Ini adalah hasil yang sangat baik jika tujuan Anda adalah untuk mengurangi kerugian seminimal mungkin. Waktu yang dibutuhkan untuk masing-masing tahapan ini dapat bervariasi tergantung pada kompleksitas model, ukuran data input, dan perangkat keras yang digunakan.

## 5 Kesimpulan

YOLO memiliki sejumlah keunggulan yang membuatnya menjadi algoritma yang sangat menonjol dalam pendeteksian objek:

- Kecepatan: YOLO terkenal karena kecepatannya. Dengan satu kali pengolahan gambar, ia dapat mendeteksi dan mengklasifikasikan objek-objek dalam waktu nyata, menjadikannya sangat cocok untuk aplikasi waktu-nyata seperti kendaraan otonom.
- Akurasi yang Baik: Meskipun YOLO sangat cepat, ia memberikan akurasi yang memadai dalam banyak kasus. Kecepatan dan akurasi yang seimbang menjadikan YOLO sangat berguna dalam berbagai aplikasi.
- Implementasi yang Mudah: Kerangka kerja Ultralytics memudahkan implementasi YOLO dengan menyediakan alat dan sumber daya yang diperlukan.

  Algoritma YOLO dan kerangka kerja Ultralytics telah mengubah cara kita mendekati pendeteksian objek dengan menggabungkan kecepatan dan akurasi. Meskipun ada beberapa kelemahan, YOLO tetap menjadi salah satu algoritma paling penting dalam pengenalan komputer dan aplikasi pendeteksian objek. Dengan terus melakukan penelitian dan pengembangan, kita dapat memanfaatkan potensi maksimal dari teknologi ini dalam berbagai aplikasi di masa depan.