

Subscribe to DeepL Pro to translate larger documents. Visit [www.DeepL.com/pro](https://www.deepl.com/pro?cta=edit-document&pdf=1) for more information.

*Deteksi Objek Waktu Nyata Bab* 4

Jika kita memperhitungkan operasi konvolusi, nilai 16 x 16 x 64 masing-masing dikalikan dengan 16, dan sekarang kita memiliki dimensi ketiga, yang biasanya tidak kita tampilkan karena selalu sama dengan dimensi ketiga masukan. Dengan demikian, output Anda akan menjadi 16 x 16 x 64. Semua nilai yang melewati filter akan dikalikan dengan 16 x 16 x

64. Kita kemudian menjumlahkan semua matriks ini dengan hanya mendapatkan satu keluaran, dan keluarannya adalah nilai persegi panjang ini. Jika kita menerapkan 256 struktur 16 x 16 x 64 ini, kita akan mendapatkan 256 nilai persegi panjang. Sekarang, perhatikan bagaimana setiap input berkontribusi pada output, jadi pada dasarnya sifat pertama dari lapisan yang terhubung sepenuhnya telah tercapai.

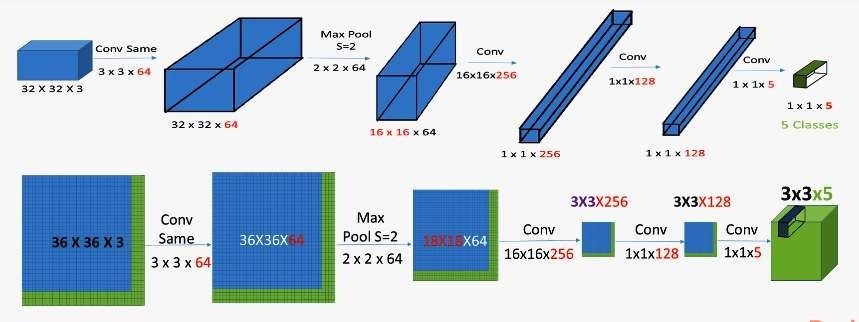
Selanjutnya, kita menerapkan konvolusi 1 x 1 x 128, yang memberikan kita 1 x 1 x 128, jadi 128 neuron cocok dengan vektor ini. Ketika kita menggunakan konvolusi 1 x 1, selain mengalikannya dengan semua nilai dan menjumlahkannya, kita juga menerapkan fungsi aktivasi, dan kita menggunakannya sebagai output dari konvolusi, yang memenuhi syarat kedua dari nonlinieritas.

Kami menerapkan rutinitas konvolusi, kami hanya menggunakan fungsi aktivasi ke output, dan kami menggunakan hasil fungsi aktivasi sebagai output akhir. Dan pada dasarnya itu membuat fungsi kita menjadi nonlinier.

Terakhir, kami menerapkan konvolusi 1 x 1 x 5, dan softmax akan memungkinkan kami untuk memprediksi 5 kelas.

Pertanyaan yang masih tersisa adalah, jika lapisan yang terhubung sepenuhnya dan konvolusi memiliki efek matematis yang sama, mengapa kita merasa perlu untuk mengubahnya sejak awal?

Untuk memahaminya, kami menggunakan gambar yang lebih besar; alih-alih 32 x 32, kami menggunakan gambar 36 x 36:

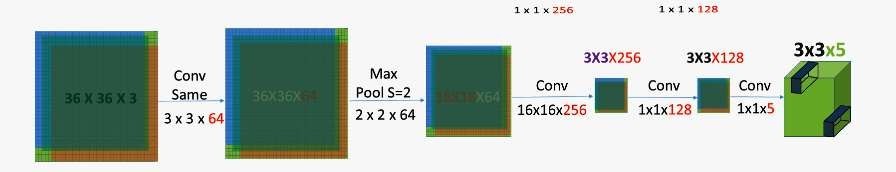


Agar gambar 32 x 32 dapat mencakup 36 x 36, kita memerlukan sembilan gerakan. Persegi panjang yang digambarkan dalam persegi panjang gambar adalah 32 x 32, dan jika kita menggunakan nilai langkah 2, kita memerlukan sembilan gerakan untuk mencakup semua piksel dalam gambar.

Sekarang, mari kita terapkan lapisan convolutional dan max pooling. Lapisan convolutional pertama meningkatkan jumlah saluran. Max pooling membagi dan di sini kita membutuhkan sembilan gerakan, atau 16 x 16, untuk mencakup *18 x 18,* kemudian dengan menerapkan lapisan convolutional yang sama kita memiliki 3 x 3 x 256. Hal ini memperjelas bahwa kita membutuhkan sembilan gerakan atau 1 x 1 untuk mencakup matriks 3 x 3.

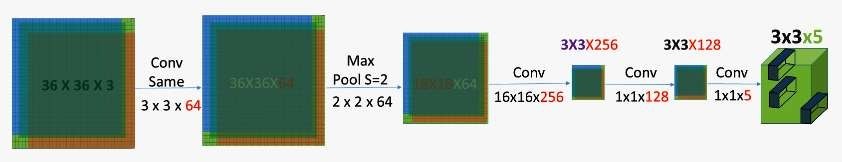
Dengan cara yang sama, kita menerapkan lapisan konvolusi yang sama lagi dan mendapatkan 3 x 3 x 128 dengan sembilan gerakan lagi. Terakhir, kami menerapkan 1 x 1 x 5. Output yang diperoleh adalah 3 x 3 x 5. Perhatikan bahwa output yang kami peroleh mirip dengan sembilan gerakan 1 x 1 x 5 karena output yang diperoleh adalah 3 x 3 x 9 dan kami memiliki sembilan gerakan untuk mencakup seluruh struktur 36 x 36 menggunakan 32 x 32.

Dengan kata lain, kami memperoleh sembilan prediksi dengan hanya satu eksekusi jaringan untuk masing-masing jendela ini. Pada akhirnya, jendela Anda akan berpindah melintasi gambar, seperti ini:



Perhatikan, bahwa tidak ada piksel yang dieksekusi lagi; sebaliknya, piksel tersebut digunakan kembali.

Apabila kita bandingkan dengan metode jendela geser, kita perlu menjalankannya secara terpisah. Asumsikan posisi lain, seperti yang ditunjukkan pada diagram berikut:



Pada dasarnya, struktur ini memiliki prediksi untuk piksel yang dipilih.

Untuk memahami hal ini dengan lebih baik, mari kita bandingkan jendela geser dengan jendela geser konvolusi. Gambar untuk jendela geser terlihat seperti ini:



Beberapa ukuran jendela berpindah ke gambar lama untuk menutupinya, dan setiap kali jendela berpindah, bagian yang dipilih diberikan ke jaringan saraf dan diminta untuk membuat prediksi. Ini berarti jaringan saraf menjalankan semua bobot setiap kali. Ketika kita menggunakan jendela geser konvolusi, hanya dengan satu kali eksekusi, kita menambahkan semua prediksi, seperti yang ditunjukkan pada gambar berikut:



Semua ini karena kemampuan untuk menggunakan kembali piksel yang dibagikan. Ini benar-benar peningkatan yang sangat besar, yang memungkinkan kami melakukan pendeteksian video objek secara real-time. Namun, meskipun kinerjanya telah meningkat, kita masih memiliki satu masalah terakhir: terkadang algoritma geser akan memberikan output berupa kotak pembatas yang tidak akurat. Untuk mengatasi hal ini, kita akan menggunakan algoritma YOLO.

## }162l

Mendeteksi objek dengan algoritme YOLO

Pada bagian ini, kita akan melihat bagaimana cara kerja algoritma YOLO. YOLO adalah singkatan dari you only look once. Nama ini berasal dari fakta bahwa Anda hanya perlu satu kali eksekusi jaringan syaraf untuk mendapatkan semua prediksi, yang dimungkinkan karena penggunaan jendela geser konvolusi.

YOLO memecahkan masalah akurasi kotak pembatas. Jadi, seperti yang kita lihat di bagian sebelumnya, kita memiliki gambar ini:

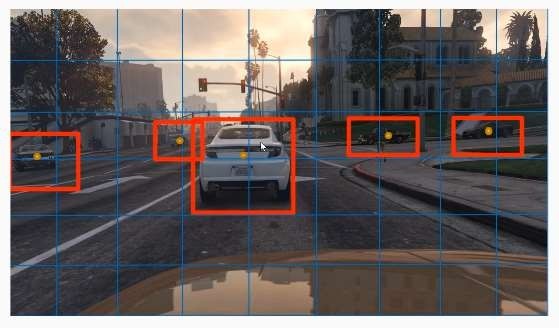


Dengan bantuan jendela geser konvolusi, kami dapat mendeteksi semua prediksi jendela dengan satu kali eksekusi. Jadi, untuk setiap jendela ini, kami dapat mendeteksi apakah piksel yang dipilih merepresentasikan sebuah mobil.

Masalahnya, kalaupun kita bisa melakukan itu, jendela ini agak stabil, sehingga tidak mampu merepresentasikan kotak pembatas yang baik. Amati gambar secara saksama, dan perhatikan, bahwa tidak ada satu pun mobil yang berada di dalam kotak pembatas yang baik.

Melihat hal ini, hampir 90% jendela berisi informasi non-mobil, dan ini tidak bagus.

Di sinilah YOLO berperan. Ini memungkinkan Anda menentukan kotak pembatas secara bebas, di mana masing-masing dapat menyeberang melalui beberapa jendela:



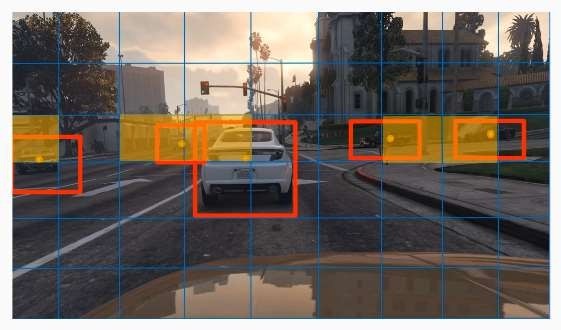
### Mari kita asumsikan bahwa dimensi kotak pembatas ditentukan secara relatif terhadap lebar dan tinggi gambar. Oleh karena itu, untuk gambar sebelumnya, kita bisa mengasumsikan tingginya sebagai

0,2 atau 20% dari tinggi gambar dan lebarnya bisa 0, 1 atau 10% dari lebar gambar. Kita bisa menentukan bagian tengah kotak pembatas dan kemudian menentukan batas-batasnya. Kiri atas adalah (0, 0), kanan bawah adalah (1,1), dan koordinat pusat kotak pembatas akan bervariasi sesuai dengan lokasi kotak.

### Setelah jaringan saraf dilengkapi dengan data berlabel, jaringan saraf akan mengembalikan struktur yang sama. Jaringan ini akan memprediksi struktur kotak pembatas.

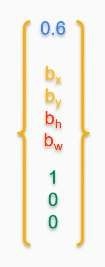
Ada satu masalah: kita perlu menghubungkan kotak pembatas ke jendela, karena apa yang akan diberikan oleh jendela geser berbelit-belit pada akhirnya hanyalah jendela. YOLO memecahkan masalah ini dengan menetapkan tinggi, lebar, dan pusat kotak pembatas ke jendela yang memegang pusat kotak pembatas.

Contohnya, dalam kasus seperti itu, hanya jendela yang digambarkan dalam tangkapan layar berikut ini yang akan bertanggung jawab untuk mendeteksi objek dan memiliki spesifikasi kotak pembatas:



### Jendela mana pun yang berisi bagian tengah kotak pembatas akan dipilih.

Struktur algoritma YOLO mirip dengan struktur algoritma convolutional window:



### Nilai pertama adalah fq yang mungkin 60% untuk kotak pembatas yang terletak di ujung kiri layar, kemudian kita memiliki properti kotak pembatas, yang relatif terhadap gambar, dan kemudian nomor kelas. Dengan cara yang sama, kita dapat menggambarkan struktur untuk kotak-kotak lainnya.

Ada beberapa jendela tertentu yang memiliki peran penting, meskipun faktanya P, nilainya

1. Ini karena algoritma YOLO percaya bahwa jendela yang berisi pusat adalah yang paling penting.

Algoritma YOLO menggunakan metode ini untuk pelatihan dan hanya jendela yang berisi bagian tengah yang akan dioptimalkan untuk kotak pembatas yang memiliki objek di dalamnya.

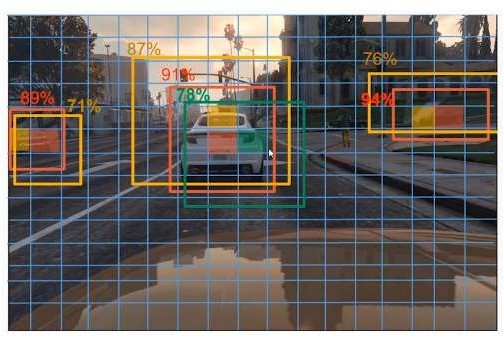
Penekanan maksimum dan kotak jangkar

Sekarang kita akan memecahkan masalah memilih kotak pembatas terbaik dari banyak kotak yang dihasilkan selama waktu pengujian, dan tantangan ketika sebuah jendela memiliki lebih dari satu pusat kotak pembatas.

Penekanan maksimum

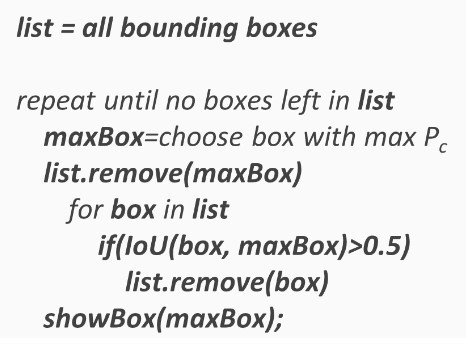
Meskipun selama pelatihan kami hanya menetapkan satu kotak pembatas ke sebuah jendela, yaitu jendela yang memiliki bagian tengah, namun selama pengujian bisa saja banyak jendela yang mengira bahwa mereka memiliki bagian tengah kotak pembatas yang terbaik.

Sebagai contoh, kita mungkin memiliki tiga kotak pembatas, dan oleh karena itu ada tiga jendela di tengah-tengah kotak pembatas, dan masing-masing jendela ini berpikir bahwa mereka memiliki kotak pembatas terbaik:



Tetapi yang kita perlukan hanya satu kotak pembatas, dan sebaiknya yang terbaik. Suspensi maksimum akan menyelesaikan masalah ini.

Algoritma penekanan maksimal terlihat seperti ini jika ditulis dalam kode yang dapat diamati:



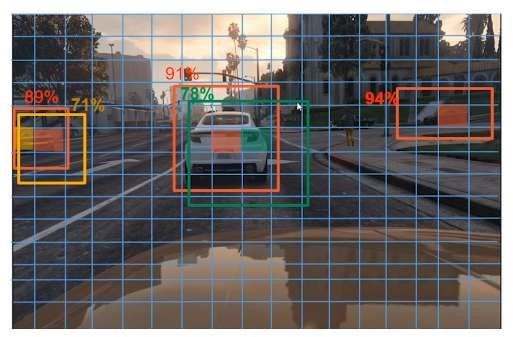
Mari ikuti langkah-langkah berikut untuk memahami bagaimana penggabungan maksimal menyelesaikan kotak pembatas

masalah:

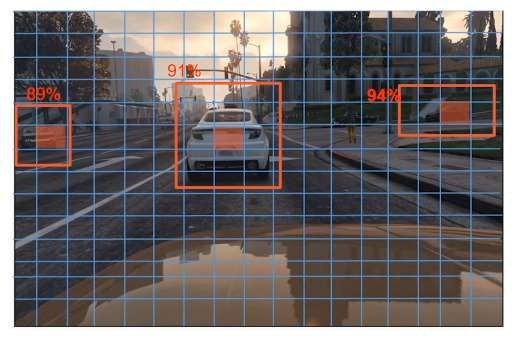
1. Menetapkan semua kotak pembatas ke daftar.
2. Lanjutkan mengulangi kode sebelumnya, selama daftar tidak kosong.
3. Pilih kotak pembatas dengan nilai maksimum P" atau probabilitas piksel yang dipilih memiliki kotak pembatas. Dalam kasus ini, jika kita melihat probabilitas pada gambar mobil, yaitu 94% jendela, maka kotak pembatas tersebut akan menjadi kotak maksimum pada gambar mobil.
4. Hapus kotak maks dari daftar.
5. Untuk kotak pembatas yang tersisa, seperti kotak kuning di bagian tengah gambar mobil dan sekumpulan kotak di sekelilingnya, lakukan hal berikut ini:
   1. Jika salah satu kotak pembatas yang tersisa berbagi lebih dari 50% dengan kotak maks, hapuslah. Dan seperti yang bisa kita lihat, kotak kuning di sebelah kiri sekelompok kotak, dengan probabilitas 74%, berbagi lebih dari 50% dengan kotak max, jadi kita hapus.
   2. Intersection over union (IOU) membagi area persimpangan dengan union, misalnya, pada gambar mobil di dalam kotak probabilitas 94%, dan jika lebih dari 50%, itu berarti kedua kotak - 76% dan 94% - berbagi lebih dari 50%. Ini akan menghapus kotak tersebut dari daftar, sehingga kotak kuning 76% akan dihapus.
   3. Pilih kotak maksimal dari kotak yang tersisa, dan jika probabilitas

kotak maksimal berada di 91%, kami segera menghapus kotak 90% dari daftar.

* 1. Periksa semua kotak yang tersisa dan periksa apakah ada yang memiliki lebih dari 50O ñ. Hapus keduanya. Kita akan mendapatkan kotak-kotak pembatas:



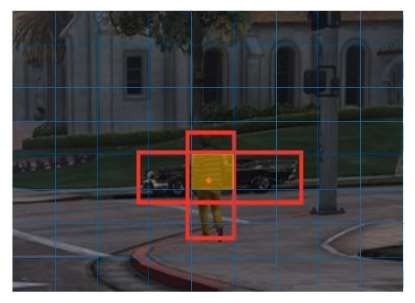
1. Sekarang daftar tersebut hanya memiliki dua kotak pembatas. Kita pilih maks lagi, yaitu 89%, hapus kotak itu, lalu untuk kotak yang tersisa dalam daftar, yang hanya kotak kuning, kita periksa apakah kotak itu memiliki lebih dari 50% dengan kotak maks, yang ternyata memang demikian, dan oleh karena itu kita hapus kotak kuning dengan probabilitas 71%:



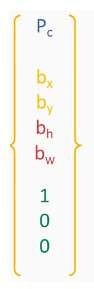
Sekarang kita periksa daftarnya. Daftarnya kosong, dan kita hanya memiliki tiga kotak pembatas, dan memang itu adalah kotak pembatas terbaik dengan probabilitas maksimum.

# Kotak jangkar

Masalah ini terkait dengan situasi ketika jendela memiliki beberapa pusat kotak pembatas. Seperti yang ditunjukkan tangkapan layar ini di sini:



Kotak pembatas mewakili mobil, dan bagian tengah kotak pembatas mewakili seseorang. Melihat struktur yang telah kita lihat sejauh ini, kita tidak dapat menyelesaikan masalah ini, karena setiap jendela ini memiliki struktur sebagai berikut:



Struktur ini hanya bisa mewakili satu kotak pembatas, sehingga jendela pada tangkapan layar bisa memiliki dimensi kotak pembatas mobil atau orang, tetapi tidak keduanya.

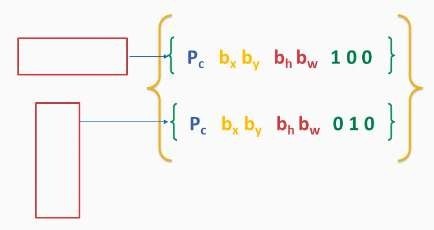
Dan jika kita melihat kisi-kisi untuk masing-masing jendela, kita memiliki struktur ini:



Kita akan memecahkan masalah ini dengan menggunakan berbagai bentuk kotak pembatas, yang disebut kotak jangkar.

Dalam hal ini, kami memilih dua kotak: satu kotak yang sesuai dengan bentuk mobil dan satu kotak yang sedikit lebih tinggi daripada lebarnya, yang sesuai dengan seseorang.

Untuk masing-masing kotak jangkar ini, kami menggunakan struktur ini:



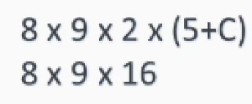
[ 170

)

P, adalah properti kotak pembatas dan kelas:



Sekarang, perhatikan, bagaimana dimensinya berubah:



Ini akan menjadi jumlah kotak jangkar. Kami telah menggandakan jumlah informasi yang dibutuhkan.

Pada gambar mobil dan orang, misalnya, P, untuk kotak jangkar yang mengelilingi mobil akan menjadi 1, karena kita memiliki objek dengan bentuk persegi panjang. Dan kelasnya adalah 1 0 0 karena ini adalah mobil.

Kemudian kita coba bentuk kotak pembatas yang mengelilingi orang tersebut: P, akan menjadi 1, properti kotak pembatas akan m e r e p r e s e n t a s i k a n bentuk daftar, dan sekarang kita akan memiliki 0 1 0, yang merepresentasikan orang tersebut.

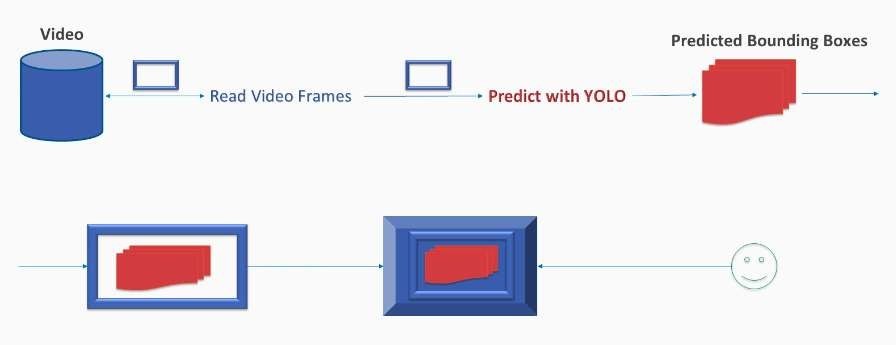
Dengan cara ini, dengan memilih bentuk yang berbeda, kita dapat menetapkan lebih banyak informasi ke jendela, dan pada dasarnya jendela dapat memiliki banyak pusat kotak pembatas.

# Membangun aplikasi deteksi video, mobil, dan pejalan kaki secara real-time

Kita akan menggunakan algoritma YOLO untuk membangun aplikasi pendeteksi objek waktu nyata di Java. Kita akan menggunakan pembelajaran transfer untuk memuat model YOLO yang telah dilatih pada ImageNet dan dataset COCO. Di antara objek-objek lainnya, aplikasi ini mencoba mendeteksi mobil, pejalan kaki, dan lampu lalu lintas dengan akurasi yang cukup tinggi.

Arsitektur aplikasi

### Sebelum masuk ke dalam kode, mari kita lihat seperti apa arsitektur dari aplikasi ini:



Pertama, kami membaca frame video dengan kecepatan tertentu, mungkin 30 frame per detik. Kemudian, kami memberikan setiap frame ke model YOLO, yang memberikan kami prediksi kotak pembatas untuk setiap objek. Setelah kita memiliki dimensi bounding box, yang relatif terhadap jendela yang menjadi pusatnya, dan jumlah grid bersama dengan ukuran frame, kita dapat menarik bounding box secara tepat ke dalam frame, seperti yang ditunjukkan pada diagram sebelumnya, untuk setiap objek. Kemudian kita tinggal menunjukkan bingkai yang sudah dimodifikasi dengan kotak pembatas kepada pengguna.

### Sekarang, jika kita memiliki kekuatan pemrosesan yang cukup, arsitektur ini akan bekerja dengan baik, tetapi ketika digunakan dengan CPU, arsitektur i n i tidak berskala dengan baik. Alasannya adalah meskipun YOLO dioptimalkan dan sangat cepat, namun tidak akan bekerja dengan baik dengan CPU berdaya rendah. Langkah-langkah ini membutuhkan waktu, dari 300 milidetik hingga 1,5 detik, tergantung pada resolusi bingkai dan ukuran kisi yang kita pilih. Pengguna akan d a p a t melihat satu frame per 1,5 detik, yang tidak terlihat bagus-kita akan mendapatkan video gerak lambat. Ini hanya akan terjadi ketika digunakan dengan CPU; di sisi lain, ketika kita menggunakan GPU, YOLO melakukan pekerjaan dengan baik dan memberikan prediksi waktu nyata, jadi arsitektur ini akan bekerja dengan baik.