Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) DOI: 10.25126/jtiik.201961275
Vol. 6, No. 4, Agustus 2019, hlm. 413-419 p-ISSN: 2355-7699
Akreditasi KEMENRISTEKDIKTI, No. 30/E/KPT/2018 e-ISSN: 2528-6579

PENGEMBANGAN DETEKSI CITRA MOBIL UNTUK MENGETAHUI JUMLAH TEMPAT PARKIR MENGGUNAKAN CUDA DAN MODIFIED YOLO

Sisco Jupiyandi[1], Fadhil Rizqullah Saniputra[2], Yoga Pratama[3], Muhammad Robby Dharmawan[4], Imam Cholissodin*[5]

[1,2,3,4,5]Fakultas Ilmu Komputer Universitas Brawijaya
Email: [1]sjupiyandi@gmail.com, [2]fadhil97@gmail.com, [3]pratamaayogaa@gmail.com, [4]robby.danu@gmail.com, [5*]imamcs@ub.ac.id

(Naskah masuk: 22 November 2018, diterima untuk diterbitkan: 16 Januari 2019)

Abstrak

Besarnya lahan pada parkir dan jumlah kendaraan roda empat dalam hal ini adalah mobil, dapat menjadi kendala bagi pengendara lain dalam mengetahui posisi parkir mana yang masih dapat digunakan. Sistem pengembangan perparkiran yang ada masih kurang maksimal dalam memanfaatkan lahan dan efisiensi waktunya. Berdasarkan banyaknya kendaraan mobil yang semakin bertambah, maka kebutuhan akan lahan parkir juga semakin dibutuhkan. Banyak sekali sistem yang belum dapat menangani berbagai permasalahan yang ada. Sistem ini dapat mengetahui jumlah slot pada lahan parkir dengan akurat sehingga memudahkan pengelola. Selain itu sistem ini juga dikembangkan agar waktu pencarian lahan parkir oleh pengguna parkir bisa sangat cepat. Sistem ini menggunakan penerapan pemrograman GPU yang dikombinasi dengan *Modified* Yolo (M-Yolo). GPU pada M-Yolo dibutuhkan untuk mengolah citra sekaligus mengolah data untuk mendeteksi citra mobil dan jumlah mobil secara paralel. Hasil uji coba menunjukkan bahwa dengan menggunakan GPU dibandingkan dengan CPU dapat mempercepat waktu komputasi rata-rata sebesar 0,179 detik dengan rata-rata akurasi sebesar 100%.

Kata kunci: *cuda, modified yolo, parkir, deteksi citra*

DEVELOPMENT OF CAR IMAGE DETECTION TO FIND OUT THE NUMBER OF PARKING SPACE USING CUDA AND MODIFIED *YOLO*

Abstract

The width of parking lot and the number of cars in the parking lot can be an obstacle for motorists to know the parking area in which part is still empty. Parking systems that exist at this time are still not maximal in the utilization of parking lots and time efficiency. Based on the number of vehicles that are growing, then the need for parking space is also more needed. Many of the existing parking systems have not been able to handle the various problems. This system can know the number of slots on the parking lot, making it easier for operators to know the empty parking lot. In addition, this system will also be designed so that parking time search by parking users doesn't take a long time. This system uses implementation of GPU programming mixed with Modified Yolo (M-Yolo). GPU on M-Yolo is needed to process images while processing data to detect car and the number of cars using parallel computing. The test results show that using the GPU compared to the CPU can speed up the average computing time by 0.179 seconds and it obtained an average accuracy of 100%.

Keywords: *cuda, modified yolo, parking, image detection*

1. PENDAHULUAN

Tempat parkir merupakan salah satu sarana penting pada suatu pusat perbelanjaan, perkantoran, dan lain-lain. Besarnya lahan parkir dan jumlah mobil di tempat parkir dapat menjadi suatu kendala bagi pengendara mobil lain untuk mengetahui lahan parkir di bagian manakah yang masih kosong, sehingga mengharuskan para pengendara mengelilingi lahan parkir untuk mencari tempat yang kosong. Perkembangan pemanfaatan pengolahan citra saat ini berkembang dengan sangat pesat. Pengolahan citra adalah suatu teknik yang digunakan untuk memproses dan memanipulasi sebuah citra digital untuk mendapatkan informasi tertentu dari citra yang diproses. Salah satu pemanfaatannya yaitu untuk pendeteksian ketersediaan slot parkir dengan cara mengambil informasi berupa deteksi citra mobil pada lahan parkir tersebut (Yusnita, 2012). Namun dari beberapa penelitian kurang mengembangkan dalam hal kecepatan waktu komputasi untuk pengolahan

data citranya, dan masih hanya menggunakan variasi banyaknya data yang digunakan (Zhang, 2018). Padahal bagaimanapun tata letak parkirnya, kunci utamanya adalah pada optimalnya ketika mendeteksi mobilnya, karena selalu identik dan standar.

Manajemen parkir pada berbagai tempat sangatlah dibutuhkan. Karena sering kali para pengelola dan pengguna parkir sangat kesulitan dalam mengetahui berapa jumlah slot parkir yang kosong dan di mana letak kosongnya hampir setiap hari dan setiap kali masuk lahan parkir. Para pengelola atau petugas parkir selama ini banyak mengandalkan cara-cara konvensional dalam pengelolaan parkir tersebut, misalnya memerika setiap sudut parkir yang mungkin masih ada slot yang kosong maupun sampai terkadang harus melakukan bongkar pasang posisi parkir dari beberapa mobil yang sudah diparkir sebelumnya.

Berdasarkan beberapa permasalahan yang telah dijelaskan diatas, maka kebutuhan akan informasi lokasi slot parkir tersebut mendorong untuk mengusulkan penelitian ini guna mengetahui jumlah slot tempat parkir secara otomatis menggunakan Pemrograman CUDA dan *Modified* Yolo (M-Yolo). Pada saat ini tempat parkir telah banyak dilengkapi dengan kamera CCTV yang berguna hanya untuk memantau keadaan parkir tersebut. Dari citra yang dihasilkan oleh CCTV tersebut dapat diproses lebih lanjut menjadi informasi melalui sistem yang telah dibuat sehingga dapat mendeteksi serta menginformasikan tempat parkir yang kosong kepada para pengendara dengan teknik pengolahan citra.

2. DASAR TEORI

2.1. Deteksi Citra Parkir Mobil

Konsep yang digunakan untuk mendeteksi citra mobil adalah menggunakan CUDA dan *Modified* Yolo (M-Yolo). Hal tersebut dilakukan untuk mengetahui berapa jumlah slot tempat parkir yang kosong pada sebuah tempat parkir. Untuk mengetahui hal tersebut, dilakukan dengan mendeteksi semua mobil yang terdapat pada tempat parker tersebut dan mengetahui jumlah keseluruhan slot dari tempat parkir tersebut. Sehingga Persamaan 1 digunakan untuk menghitung jumlah slot parkir yang kosong yaitu:

$$JSK = JSS - JSM \quad (1)$$

Keterangan:

JSK = Jumlah Slot Kosong
JSS = Jumlah Seluruh Slot
JSM = Jumlah Seluruh Mobil

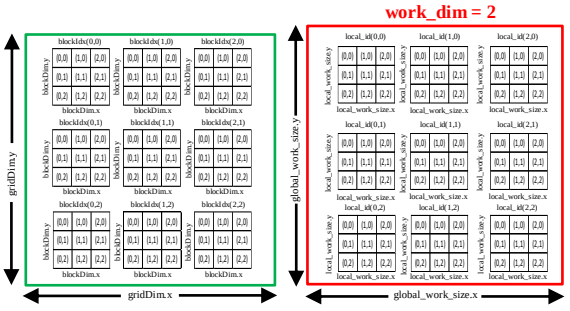
2.2. Konsep Pemograman GPU

Paradigma paralel pada GPU sangat penting dalam penyelesaian operasi grafis secara cepat. Secara umum GPU melakukan komputasi data dalam bentuk angka menjadi bentuk citra (komputer grafik) dan juga bisa dimanfaatkan sebaliknya (visi komputer) (Prahara, 2015). Hal tersebut termasuk dalam *General Purpose* GPU (Fung, 2008). GPU sendiri memiliki arsitektur yang unik, yaitu adanya prosesor multithread yang mampu menjalankan jutaan proses secara bersamaan (Mukhlis, 2007). Secara simultan, proses paralel menggunakan beberapa komponen komputasi sehingga mampu memecahkan permasalahan besar dan komplek dalam komputasi (Barney, 2009). Teknologi GPU dapat dikombinasikan dengan beberapa teknologi lain pada sistem operasi tertentu seperti pada Gambar 1.

Gambar 1. Yolo-Darknet, Cuda, OpenCV, Win10

2.3. CUDA

CUDA di mata pengembang sangat mudah untuk digunakan. Salah satunya karena arsitekturnya yang handal, juga kodenya sangat mirip dengan bahasa C yang sudah sangat populer, sehingga mudah dipahami (Kurniawan, 2015). CUDA dalam pemrogramannya membagi device ke dalam grid, block, dan thread. Jika dikomparasi dengan OpenCL, maka akan sangat terlihat identik seperti pada Gambar 2 (Irawan, 2017).



Gambar 2. CUDA Grid Vs OpenCL NDRange

Berdasarkan pada Gambar 2, maka dapat dituliskan perbedaan antara bagian koding sederhana dengan menggunakan CUDA Grid dan OpenCL NDRange seperti dalam Tabel 1.

Tabel 1. CUDA Grid Vs OpenCL NDRange

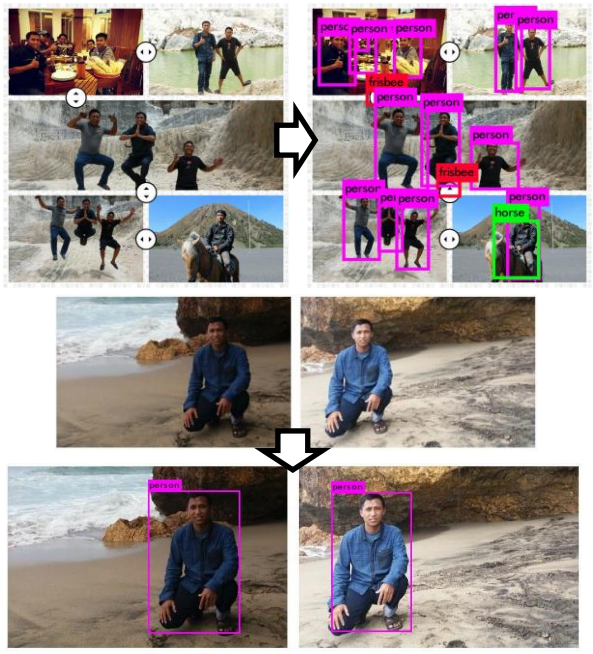
CUDA Grid	OpenCL NDRange
"dim3 grid(3, 3, 1);" dan "dim3 block(3, 3, 1);"	"global_work_size[3]={ 3, 3, 0 };" dan "local_work_size[3]={ 3, 3, 0 };" kemudian, "clEnqueueNDRangeKernel(command_queue, kernel, work_dim=2, NULL, global_work_size, local_work_size, 0, NULL, NULL);"

CUDA umumya dipakai untuk pengolahan grafis seperti pada *image* dan *video processing* secara digital seperti *classification with localization*, *image segmentation*, perubahan citra, pengenalan pola dan dapat diaplikasikan pada berbagai multi-disiplin keilmuan (Basuki, 2005).

2.4. Algoritma Yolo

You Only Look Once (Yolo) adalah sebuah algoritma yang dikembangkan untuk mendeteksi sebuah objek secara *real-time*. Sistem pendeteksian yang dilakukan adalah dengan menggunakan *repurpose classifier* atau *localizer* untuk melakukan deteksi. Sebuah model diterapkan pada sebuah citra di beberapa lokasi dan skala. Daerah dengan citra yang diberi *score* paling tinggi akan dianggap sebagai sebuah pendeteksian (Unsky, 2017).

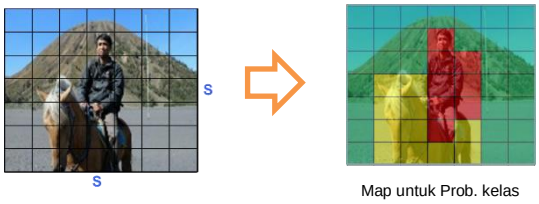
Yolo menggunakan pendekatan jaringan saraf tiruan (JST) untuk mendeteksi objek pada sebuah citra. Jaringan ini membagi citra menjadi beberapa wilayah dan memprediksi setiap kotak pembatas dan probabilitas untuk setiap wilayah. Kotak-kotak pembatas ini kemudian dibandingkan dengan setiap probabilitas yang diprediksi. Yolo memiliki beberapa kelebihan dibandingkan dengan sistem yang berorientasi pada *classifier*, terlihat dari seluruh citra pada saat dilakukan *test* dengan prediksi yang diinformasikan secara global pada citra (Redmon, 2016). Hal tersebut juga membuat prediksi dengan sintesis jaringan saraf ini tidak seperti sistem *Region-Convolutional Neural Network* (R-CNN) yang membutuhkan ribuan untuk sebuah citra sehingga membuat Yolo lebih cepat hingga beberapa kali daripada R-CNN. Pada Gambar 3 merupakan contoh hasil deteksi beberapa objek dengan menggunakan Yolo-Darknet.



Gambar 3. Contoh deteksi objek

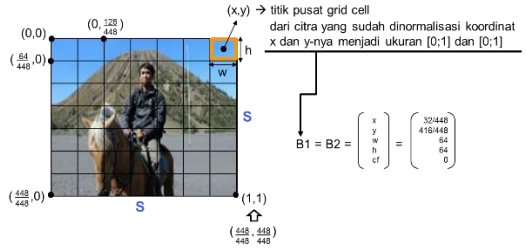
Klasifikasi secara umum adalah proses untuk mengidentifikasi label dari data yang diuji, sedangkan pada Yolo, klasifikasinya dengan *localization*, yaitu terdapat tambahan pemberian lokasi objek dalam bentuk *bounding box* (b_x,b_y,b_h,b_w), seperti pada Gambar 3. Tahapan algoritma Yolo dapat dilihat pada delapan langkah berikut.

a. Baca data citra dengan ukuran sembarang.
b. Ubah ukuran citra menjadi 448 x 448, lalu buat *grid* pada citra dengan ukuran *S* x *S grids*.

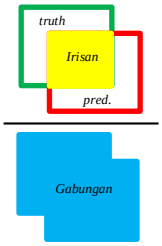


Jika *S*=7, maka tiap *grid cell* ukurannya 64 x 64. Sehingga terdapat sebanyak 49 *grid cell*. Dan misalkan pada citra ada 2 kelas (*nC*=2), yaitu "Manusia" (*c1*), "Kuda" (*c2*), dan "Background".

c. Tiap *grid cell*, misal *nB*=2, terdapat *B* yang berisi 5 nilai, yaitu lokasi koordinat *x* diasumsikan berdasar baris (*x_br*), koordinat *y* berdasar kolom (*y_kol*), ukuran dan nilai *confidence* (*x*,*y*,*w*,*h*,*cf*) terhadap *nB* bbox yang ada, yaitu B1 dan B2.



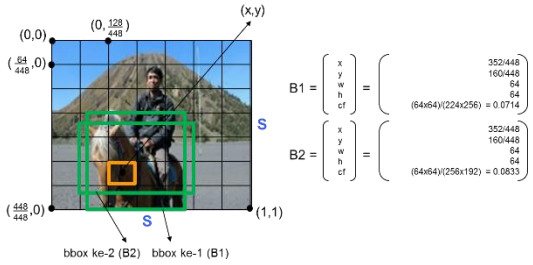
cf = 0 (selalu di-*set* = 0, jika dalam *grid cell* adalah *background*). *Confidence* (*cf*) = P(object)*IoU, yang mana, *nB* menyatakan banyaknya *bbox*, ***B1*** untuk *bbox1*, *B2* untuk *bbox2*, dan *bbox* menyatakan *bounding box*. Persamaan 2 merupakan *Intersection Over Union* (IoU^{truth}_{pred}).



$$IoU = \frac{Irisan}{Gabungan} \qquad (2)$$

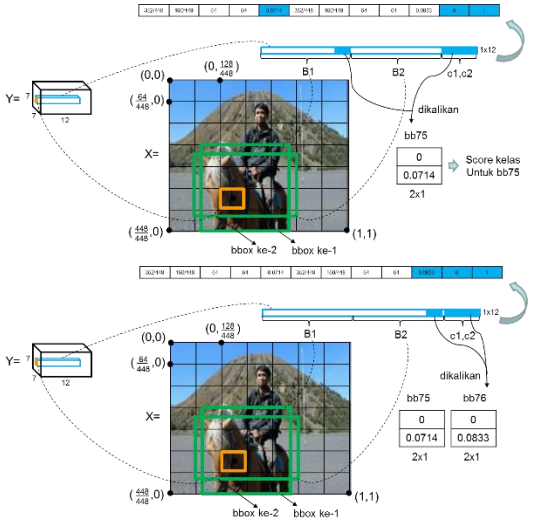
Pada perhitungan *confidence* (*cf*) = P(*object*)*IoU, P(*object*) dapat dilewati terlebih dahulu, sehingga cukup dengan *cf* = IoU. Karena $P(class_i \mid object)$* P(*object*)*IoU= $P(class_i)$*IoU.

d. Jika 2 bbox mengacu pada kelas yang sama, maka hasil ukuran tensornya adalah (*S* x *S* x (*nB* x 5 + *nC*)) = (7 x 7 x (2 x 5 + 2)) = (7 x 7 x 12) tensor.

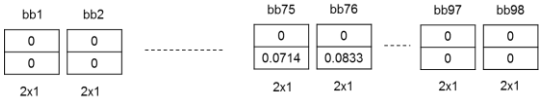


Di mana, bbox ke-1 dan bbox ke-2, dibuat bebas dan sebaiknya berbeda.

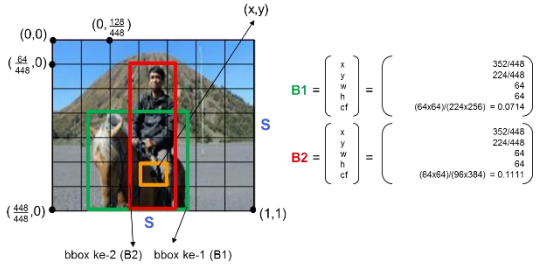
Hitung matriks *bb* dengan ukuran (*nC* x (*S* x *S* x *nB*)) = 2 x (7 x 7 x 2) = (2 x 98), mulai dari *grid cell* ke-1 sampai ke-49 pada 2 *bbox* yang mengacu pada kelas yang sama, misal untuk bb75 dan bb76.



Hasil dari matriks bb.

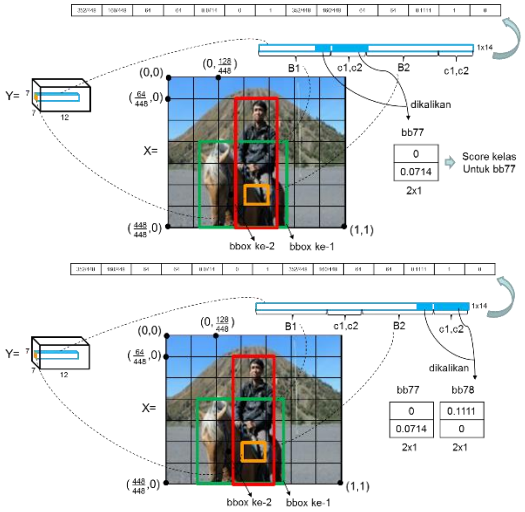


e. Jika 2 bbox mengacu pada kelas yang berbeda atau karena terdapat *overlapping object*, maka hasil ukuran tensornya adalah (*S* x *S* x (*nB* x 5 + 2 x *nC*)) = (7 x 7 x (2 x 5 + 2 x 2)) = (7 x 7 x 14) tensor.

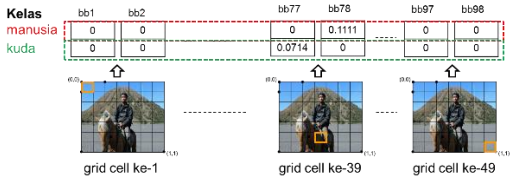


Di mana, bbox ke-1 dan bbox ke-2, mengacu pada objek yang berbeda kelas.

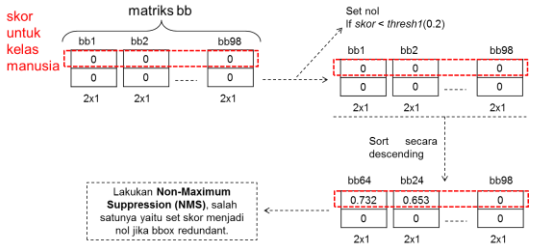
Hitung matriks *bb* dengan ukuran (*nC* x (*S* x *S* x *nB*)) = 2 x (7 x 7 x 2) = (2 x 98), mulai dari *grid cell* ke-1 sampai ke-49 pada 2 *bbox* yang mengacu pada kelas yang berbeda, misal untuk bb77 dan bb78.



Hasil dari matriks bb.



f. Tiap kelas pada matriks *bb*, lakukan set skor = 0, jika skor < *thresh1*(0.02), kemudian urutkan secara *descending*.

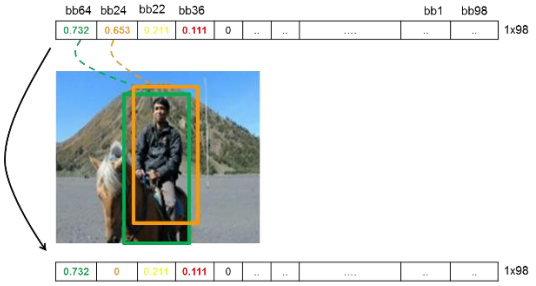


g. Lakukan *Non-Maximum Suppression* (NMS).

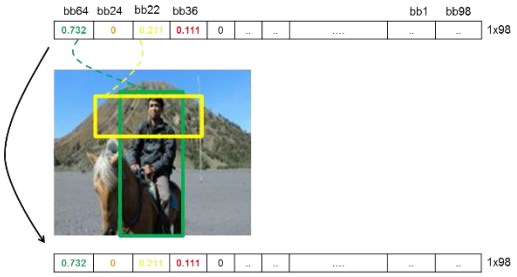
- Daftar semua bbox.

- *Set* bbox dengan skor maks, sebagai "bbox_max". Misal bb64 diketahui sebagai bbox_max.
- Bandingkan "bbox_max" dengan bbox lainnya sebagai "bbox_cur" yang memiliki skor dibawahnya dan tidak nol. Jika IoU(bbox_max,bbox_cur) > 0.5, maka set skor nol untuk bbox_cur. Dari hasil perbandingan, jika bbox_max = bb64, bbox_cur = bb24, maka

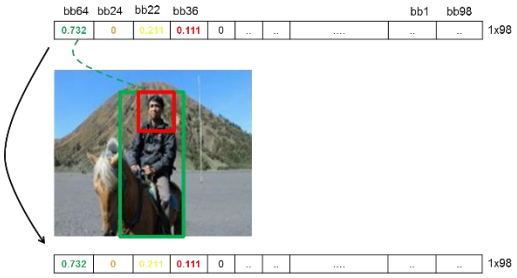
IoU(bbox_max,bbox_cur) > 0.5 (true), maka set skor bb24 = 0.



- Lanjutkan ke bb22, dari hasil perbandingan, jika bbox_max = bb64, bbox_cur = bb22, maka IoU(bbox_max,bbox_cur) > 0.5 (false), maka skor bb22 tetap.

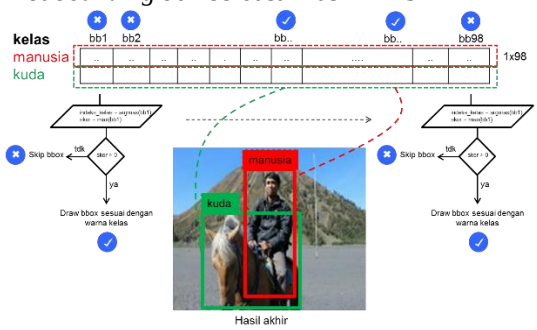


- Lanjutkan ke bb36, dari hasil perbandingan, jika bbox_max = bb64, bbox_cur = bb36, maka IoU(bbox_max,bbox_cur) > 0.5 (false), maka skor bb36 tetap.



- Lanjutkan melakukan pembandingan sampai bbox_max = bb64, bbox_cur = bb98. Lalu lanjutkan lagi, set bbox_max = bb22, dan bbox_cur = bb36, dst.
- Kemudian lanjutkan ke kelas berikutnya, lakukan hal yang sama seperti yang telah dilakukan pada kelas manusia.

h. Plot *bounding box* berdasar hasil NMS

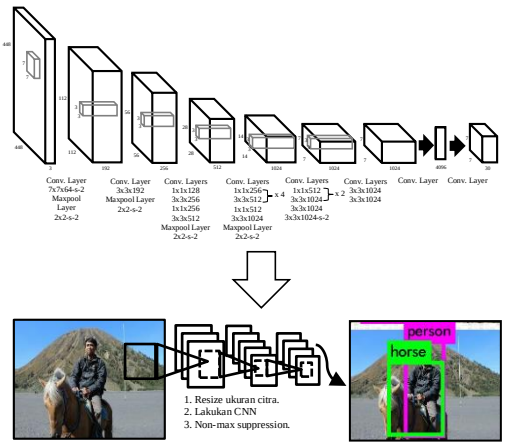


3. PERANCANGAN DAN IMPLEMENTASI

Tahap perancangan dibuat berdasarkan data-data yang telah dikumpulkan yang kemudian dirancang menggunakan CUDA dan *Modified* Yolo (M-Yolo) pada Visual Studio 2015. Kemudian untuk skenario pengujian dalam penelitian ini menggunakan metode *whitebox*. Pada skenario pengujian tersebut menggunakan unit *testing* yang dilakukan dengan membandingkan hasil aktual dengan yang diolah oleh komputer.

```
1  void draw_objdetect(image imx, int
2  numq, float threshold, box *boxeses,
3  float **myprobs, char **names, image
4  **alphabets, int uclasses){
5   int ii, x = 0;
6   for(ii = 0; ii < numq; ++ii){
7      int iclass = max_index(myprobs[ii],
8      uclasses);
9      float myprob = myprobs[ii][iclass];
10     if(myprob > threshold){
11       x++;
12       int wid = imx.h * .012;
13       if(0){
14          wid = pow(myprob,1./2.)*10+1;
15          alphabets = 0;
16       }
17       printf("%s: %.0f%%\n",
18       names[iclass], myprob*100);
19       int offset = iclass*123457 %
20       uclasses;
21       float cred =
22       get_color(2,offset,uclasses);
23       float cgreen =
24       get_color(1,offset,uclasses);
25       float cblue =
26       get_color(0,offset,uclasses);
27       float crgb[3];
28       //wid = myprob*20+2;
29       crgb[0] = cred;
30       crgb[1] = cgreen;
31       crgb[2] = cblue;
32       box b = boxeses[ii];
33       int left_  = (b.x-b.w/2.)*imx.w;
34       int right_ = (b.x+b.w/2.)*imx.w;
35       int top_   = (b.y-b.h/2.)*imx.h;
36       int bot_   = (b.y+b.h/2.)*imx.h;
37       if(left_ < 0) left_ = 0;
38       if(right_ > imx.w-1)
39          right_ = imx.w-1;
40       if(top_ < 0) top_ = 0;
41       if(bot_ > imx.h-1)bot_ = imx.h-1;
42       draw_box_width(imx, left_, top_,
43       right_, bot_, wid, cred, cgreen,
44       cblue);
45       if (alphabets) {
46          image mylabel =
47          get_label(alphabets,
48          names[iclass],(imx.h*.03)/10);
49          draw_label(imx, top_ + wid,
50          left_, mylabel, crgb);
51       }
52    }
53  }
54  printf("Jumlah slot yang tersisa : %d
55  slot \n", (12-x));
56 }
```

Kode Program 1. Hitung slot parkir kosong

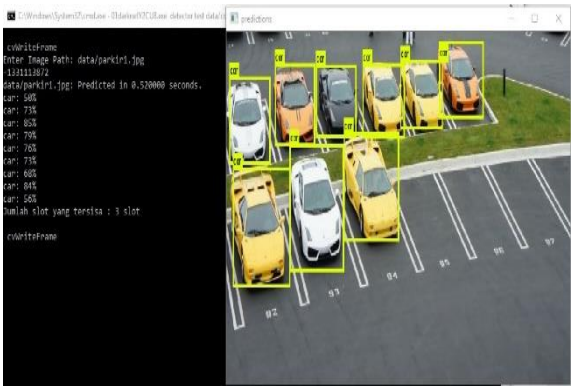


Gambar 4. Ilustrasi CNN pada *Modified* Yolo (M-Yolo)

Pada Gambar 4, CNN yang digunakan meliputi *convolutional layers* beserta proses *max pooling* yang ada pada arsitektur jaringan, dan *fully connected* layers, kemudian yang paling akhir *terdapat final output* dengan ukuran 7 x 7 x 30 tensor sebagai bahan kelas prediksi atau targetnya. Hasil implementasi pada potongan Kode Program 1 merupakan fungsi untuk melakukan deteksi jumlah slot parkir yang kosong. Penjelasan dari Kode Program 1:

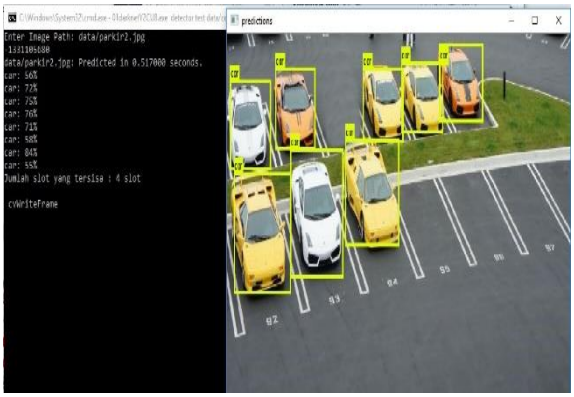
- Baris ke-1 hingga 4 merupakan deklarasi method bernama draw_detections.
- Baris ke-5 hingga 9 merupakan proses perulangan yang dilakukan untuk mengecek probabilitas setiap bagian parkir pada citra terhadap class objek yang akan dideteksi.
- Baris ke-10 hingga 12 merupakan proses menghitung banyaknya objek mobil yang terdeteksi dari citra berdasarkan nilai threshold yang telah ditentukan.
- Baris ke-13 hingga 16 merupakan proses seleksi jika probabilitas dari sebuah lahan parkir melebihi threshold maka di identifikasi sebagai adanya objek mobil yang terdeteksi.
- Baris ke-17 hingga 18 merupakan proses untuk menampilkan setiap kelas dari objek-objek yang dideteksi pada lahan parkir dan nilai probabilitasnya.
- Baris ke-19 hingga 53 merupakan proses untuk melakukan deteksi objek mobil pada lahan parkir dengan memberikan *bounding boxes* sehingga ada perbedaan warna yang terjadi pada daerah lahan parker untuk setiap mobil yang berhasil di identifikasi disertai label “car”.
- Baris ke-54 hingga 56 merupakan proses untuk menampilkan informasi jumlah lahan parkir yang tersedia menggunakan rumus 12-*x*, yang mana 12 merupakan JSS dan *x* merupakan JSM sesuai dengan Persamaan 1.

Kemudian tahap selanjutnya di jalankan kode program diatas yang hasilnya dapat dilihat pada Gambar 5. Maka didapatkan hasil deteksi slot yang kosong sebanyak 3 sesuai dengan kondisi aktual.



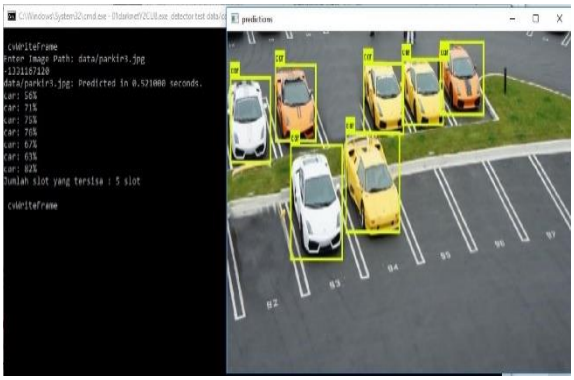
Gambar 5. Misal sebagai awal kondisi

Jika kondisi pada Gambar 5 dilajutkan, misal dengan adanya 1 mobil keluar dari parkir, maka hasil deteksi slot kosongnya akan bertambah 1 secara otomatis seperti pada Gambar 6.



Gambar 6. Ketika terdapat mobil yang sudah tidak parkir

Kemudian terdapat lagi mobil yang sudah tidak parkir lagi maka system akan mendeteksi dan akan menampilkan seperti pada Gambar 7.



Gambar 7. Ketika terdapat mobil yang sudah tidak parkir lagi

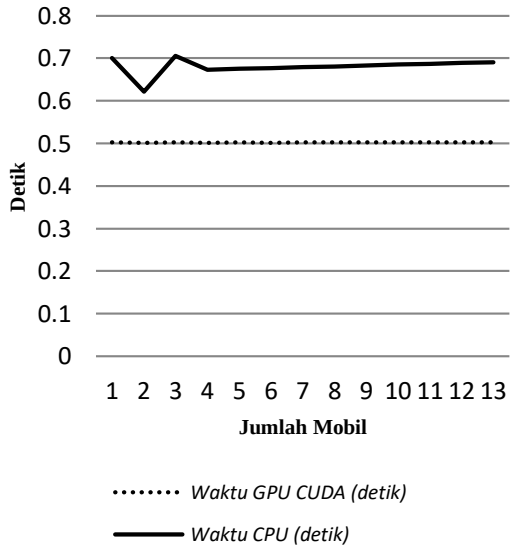
4. PENGUJIAN DAN ANALISIS

Operasi deteksi citra mobil ini diuji pada CUDA dan CPU dengan menggunakan citra slot lahan parkir yang terisi dengan jumlah mobil yang berbeda. Hasil perbandingan waktu compile CUDA dan CPU dengan menguji 13 data dapat dilihat pada Tabel 2.

Tabel 2. Waktu Komputasi di CUDA dan CPU, serta Akurasi

Jumlah mobil	Waktu (detik)		Akurasi (%)
	CUDA	CPU	
0	0,502	0,701	100
1	0,501	0,621	100
2	0,502	0,705	100
3	0,501	0,673	100
4	0,502	0,675	100
5	0,501	0,677	100
6	0,502	0,679	100
7	0,503	0,681	100
8	0,502	0,683	100
9	0,502	0,685	100
10	0,503	0,687	100
11	0,502	0,689	100
12	0,502	0,691	100

Berdasarkan perbandingan waktu *compile* GPU CUDA dan CPU pada Tabel 2 dapat dilihat pada Gambar 8 yang menampilkan grafik untuk perbandingan dari hasil pengujian menunjukkan bahwa dengan tingkat akurasi yang sama untuk mendeteksi slot parkir yang kosong, dengan GPU selalu lebih cepat dibandingkan dengan menggunakan CPU.



Gambar 8. Perbandingan Komputasi di GPU CUDA dan CPU

5. KESIMPULAN

Berdasarkan pengujian yang dilakukan dengan menjalankan program pada 13 buah data citra yang menunjukan bahwa algoritma *Modified* Yolo (M-Yolo) dapat mendeteksi jumlah mobil dengan tepat terlihat dari hasil akurasi 100% ketika *running* program yang menampilkan nilai estimasi dari tiap objek yang dideteksi sebagai mobil yang ada pada slot parkir. Dari hasil pengujian yang dilakukan dengan menjalankan pada GPU dan CPU didapatkan hasil bahwa waktu *compile* pada GPU lebih cepat dari CPU dengan perbedaan rata-rata 0,179 detik. Artinya jika semakin banyak slot parkir, maka perbedaan waktu tersebut akan semakin besar dan berpengaruh pada waktu *compile* pada GPU ataupun CPU.

Dalam penelitian selanjutnya perlu dilakukan pelatihan dan pengujian dengan menggunakan lebih banyak variasi jenis kendaraan maupun bentuk slot parkir yang lebih luas pada beberapa tempat yang berbeda. Dan algoritma M-Yolo dengan CUDA ini dapat dikembangkan misal dengan OpenCL sebagai alternatifnya CUDA atau memperbaiki pada beberapa bagian langkah pembentukan struktur arsitektur jaringan syaraf tiruannya, misal dengan menggunakan teknik optimasi, sehingga didapatkan hasil map arsitektur yang handal, cepat, dan akurat serta dapat diterapkan pada kasus lain yang komplek.

DAFTAR PUSTAKA

BASUKI, A. 2005. Pengolahan Citra Digital Menggunakan Visual Basic 6, Yogyakarta.

BARNEY, B. 2009. Parallel Image Processing Based on Cuda. China: Polytechnical University.

FUNG, J. dan MANN, S. 2008. Using Graphics Devices in Reverse: GPU-Based Image Processing and Computer Vision, Nvidia Corporation, USA.

IRAWAN, F. T., MA'RUFI, M. R., CHOLISSODIN, I. 2017. Optimasi Rendering Game 2D Asteroids Menggunakan Pemrograman CUDA. Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) Vol. 4 No. 4 2017.

KURNIAWAN, B., NOOR A. S., TEGUH B. A. 2015. Analisis Perbandingan Komputasi GPU dengan CUDA dan Komputasi CPU untuk *Image* dan *Video Processing*. *Seminar Nasional Aplikasi Teknologi Informasi (SNATi) 2015*. 6 Juni, Yogyakarta.

MUKHLIS, Y., HARMANTO, L. 2007. Metode Sorting Bitonic pada GPU. 2 Februari.

PRAHARA, A. 2015. Deteksi Kebakaran pada Video Berbasis Pengolahan Citra dengan Dukungan GPU. . *Seminar Nasional Aplikasi Teknologi Informasi (SNATi)*. 6 Juni, Yogyakarta.

REDMON, J., DIVVALA, S., GIRSHICK, R., FARHADI, A. 2016. You Only Look Once: Unified, Real-Time Object Detection.

UNSKY. 2017. yolo-for-windows-v2. *GitHub*. https://github.com/unsky/yolo-for-windows-v2. Web. diakses 30 April 2018.

YUSNITA R., NORBAYA F., dan BASHARUDDIN N. 2012. Intelligent Parking Space Detection System Based on Image Processing. International Journal of Innovation, Management and Technology, Vol. 3, No. 3, June 2012.

ZHANG L., LI X., HUANG J., SHEN Y., dan WANG D. 2018. Vision-Based Parking-Slot Detection: A Benchmark and A Learning-Based Approach. Symmetry 2018, 10, 64; doi:10.3390/sym10030064.