TUGAS AKHIR - EE 184801

DETEKSI HALANGAN MENGGUNAKAN METODE STEREO R-CNN PADA MOBIL OTONOM

Adrian Aryaputra Firmansyah
NRP 07111640000088

Dosen Pembimbing
Ir. Rusdhianto Effendi A.K., M.T.
Dr. Ir. Ari Santoso, DEA.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro Dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

Halaman ini sengaja dikosongkan

TUGAS AKHIR - EE 184801

DETEKSI HALANGAN MENGGUNAKAN METODE STEREO R-CNN PADA MOBIL OTONOM

Adrian Aryaputra Firmansyah
NRP 07111640000088

Dosen Pembimbing
Ir. Rusdhianto Effendi A.K., M.T.
Dr. Ir. Ari Santoso, DEA.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro Dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

Halaman ini sengaja dikosongkan

FINAL PROJECT – EE 184801

OBSTACLE DETECTION USING STEREO R-CNN IN AUTONOMOUS CAR

Adrian Aryaputra Firmansyah
NRP 07111640000088

Supervisor
Ir. Rusdhianto Effendi A.K., M.T.
Dr. Ir. Ari Santoso, DEA.

DEPARTMENT OF ELECTRICAL ENGINEERING
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2020

Halaman ini sengaja dikosongkan

PERNYATAAN KEASLIAN
TUGAS AKHIR

Dengan ini menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "**Deteksi Halangan Menggunakan Metode Stereo R-CNN pada Mobil Otonom**" adalah hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diizinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka. Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, Juli 2020

Adrian Aryaputra Firmansyah
NRP. 07111640000088

Halaman ini sengaja dikosongkan

DETEKSI HALANGAN MENGGUNAKAN METODE STEREO R-CNN PADA MOBIL OTONOM

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan Untuk
Memperoleh Gelar Sarjana Teknik Elektro
Pada
Bidang Studi Teknik Sistem Pengaturan
Departemen Teknik Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui:

Dosen Pembimbing I

Ir. Rusdhianto Effendi A.K., M.T.
NIP. 195704241985021001

Surabaya,
Juli 2020

Halaman ini sengaja dikosongkan

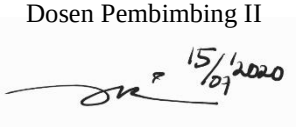
DETEKSI HALANGAN MENGGUNAKAN METODE STEREO R-CNN PADA MOBIL OTONOM

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan Untuk
Memperoleh Gelar Sarjana Teknik Elektro
Pada
Bidang Studi Teknik Sistem Pengaturan
Departemen Teknik Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui:

Dosen Pembimbing II



Dr. Ir. Ari Santoso, DEA.
NIP. 196602181991021001

Surabaya,
Juli 2020

Halaman ini sengaja dikosongkan

DETEKSI HALANGAN MENGGUNAKAN METODE STEREO R-CNN PADA MOBIL OTONOM

Adrian Aryaputra Firmansyah
07111640000088

Dosen Pembimbing : 1. Ir. Rusdhianto Effendi A.K., M.T.
2. Dr. Ir. Ari Santoso, DEA.

ABSTRAK

Mobil otonom merupakan inovasi yang muncul akibat perkembangan teknologi komputasi yang semakin mumpuni. Mobil otonom beroperasi dengan meminimalkan campur tangan manusia sehingga dapat mengurangi kecelakaan lalu lintas yang disebabkan oleh kesalahan pengemudi. Untuk menunjang kinerja mobil otonom, dibutuhkan sistem deteksi halangan yang mampu mendeteksi halangan dengan akurat. Pada penelitian ini, akan digunakan algoritma *Stereo Regional Convolutional Neural Network* (*Stereo* R-CNN) untuk merancang sistem deteksi halangan. Algoritma ini akan diterapkan pada kamera stereo untuk membuat *bounding box* tiga dimensi dengan memproses perbedaan sudut pandang pada gambar di kedua kamera. Metode ini dapat mendeteksi bounding box 3D dari objek pada gambar. Prediksi *bounding box* memiliki tingkat presisi hingga 70% dengan recall hingga 95% dan memiliki error lokasi deteksi yang berkorelasi dengan jarak dan ukuran objek di mana pada deteksi objek berjarak 5 meter, diperoleh RMSE deteksi sebesar 0.073 meter (1.46%), sementara pada deteksi objek berjarak 75 meter, diperoleh RMSE deteksi sebesar 1.278 meter (1.70%). Saat ini metode ini masih kurang *feasible* untuk dapat diterapkan sebagai pengganti LIDAR sebagai sensor utama karena kecepatan deteksinya yang masih lambat di kisaran 0.81 detik (1,2 frame per detik).

Kata Kunci : *Stereo Camera*, R-CNN, Kendaraan Otonom, Deteksi Halangan, *3D Bounding Box*.

Halaman ini sengaja dikosongkan

OBSTACLE DETECTION USING STEREO R-CNN IN AUTONOMOUS CAR

Adrian Aryaputra Firmansyah
07111640000088

Supervisor : 1. Ir. Rusdhianto Effendi A.K., M.T.
2. Dr. Ir. Ari Santoso, DEA.

ABSTRACT

Autonomous cars are innovations that arise due to the development of computing technology. Autonomous vehicles operate by minimizing human intervention to reduce traffic accidents caused by driver errors. Obstacle detection systems that can detect obstacles accurately are needed to support the performance of autonomous cars. In this study, we use the Stereo Regional Convolutional Neural Network (Stereo R-CNN) algorithm to detect obstacles around the autonomous car. This algorithm will be applied to stereo cameras to create three-dimensional bounding boxes by processing differences in the angle of view of the images in the two cameras. This method can detect 3D bounding boxes of objects in the image. Bounding box prediction has a precision level of up to 70% with a recall of up to 95%. It has a detection location error that correlates with the distance and size of the object where the detection of objects within 5 meters, obtained detection RMSE of 0.073 meters (1.46%), while the detection of distant objects 75 meters, obtained detection RMSE of 1,278 meters (1.70%). Currently, this method is not feasible to be applied directly as a substitute for LIDAR as the primary sensor because the detection speed is slow in the range of 0.81 seconds (1.2 frames per second).

Keywords: *Stereo Camera, R-CNN, Autonomous Vehicle, Obstacle Detection, 3D Bounding Box.*

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan YME yang telah memberikan rahmat dan karunianya, sehingga penulis dapat membuat dan menyelesaikan tugas akhir ini dengan semestinya serta tepat waktu.

Kegiatan tugas akhir ini termasuk salah satu mata kuliah yang wajib ditempuh di Departemen Teknik Elektro Institut Teknologi Sepuluh Nopember ini. Penelitian yang penulis lakukan mengambil topik Deteksi Halangan Menggunakan Metode Stereo R-CNN pada Mobil Otonom. Laporan tugas akhir ini disusun untuk melengkapi hasil capaian dari penelitian yang telah dilaksanakan.

Laporan tugas akhir ini bisa diselesaikan tidak terlepas dari ulur tangan pihak ketiga yang meliputi dosen pembimbing dan pembina laboratorium teknik pengaturan serta rekan-rekan sekalian yang telah senantiasa mendukung selama penelitian yang penulis kerjakan, sehingga penulis dengan tulus mengucapkan kasih yang sebesar-besarnya.

Penulis menyadari bahwa banyak kekurangan dari laporan tugas akhir ini, baik dari segi materi maupun teknis penyajiannya, mengingat masih kurangnya pengetahuan dan pengalaman penulis. Namun pengalaman dan wawasan baru yang penulis banyak dapatkan juga patut untuk penulis syukuri dan diterapkan menjadi ilmu yang manfaat.

Surabaya, 16 Juli 2020

Adrian Aryaputra Firmansyah

Halaman ini sengaja dikosongkan

DAFTAR ISI

PERNYATAAN KEASLIAN TUGAS AKHIR VII
ABSTRAK XIII
ABSTRACT XV
KATA PENGANTAR XVII
DAFTAR ISI XIX
DAFTAR GAMBAR XXIII
DAFTAR TABEL XXV
BAB 1 PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Perumusan Masalah 1
1.3 Batasan Masalah 2
1.4 Tujuan Penelitian 2
1.5 Metode Penelitian 2
1.5.1 Studi Literatur 2
1.5.2 Perancangan Sistem 2
1.5.3 Penyusunan Buku Tugas Akhir 2
1.6 Sistematika Penulisan 2
BAB 2 TEORI PENUNJANG ALGORITMA STEREO R-CNN 5
2.1 Intepretasi Gambar 5
2.2 Penginderaan Komputer 5
2.3 Pemrosesan Gambar 7
2.3.1 Interpolasi 7
2.3.2 Transformasi Spasial Geometris 8
2.3.3 *Kernel* 9
2.3.4 *Filter* 9
2.3.5 Konvolusi 11
2.4 Penginderaan *Stereo* 11
2.5 *Machine Learning* 14
2.5.1 *Supervised Learning* 15

2.5.2 *Unsupervised Learning* 16
2.5.3 *Reinforcement Learning* 17
2.5.4 *Deep Learning* 17

2.6 Jaringan Syaraf Tiruan (*Artificial Neural Network)* 17

2.6.1 *Single Layer Feedforward Neural Network / Perceptron* 19
2.6.2 *Multi Layer Feedforward Neural Network* 20
2.6.3 *Recurrent Neural Network* 21
2.6.4 *Symmetrically Connected Networks* 22

2.7 *Convolutional Neural Network* (CNN) 23

2.7.1 *Convolution Layer* 24
2.7.2 *Pooling Layer* 24
2.7.3 *Fully Connected Layer* 25
2.7.4 Zero Padding 25
2.7.5 *Stride* 26
2.7.6 Fungsi Softmax 26
2.7.7 *Stochastic Gradient Descent* 27
2.7.8 *Adaptive Moment Estimation* (Adam) 27
2.7.9 RMSProp 28
2.7.10 *Cross Entropy* 28
2.7.11 *Dropout* 29
2.7.12 *Batch Normalization* 29

2.8 Region with CNN features (R-CNN) 30

2.8.1 *Region Proposal Network* 30
2.8.2 *Intersection Over Union* (IoU) 31
2.8.3 *Region of Interest Pooling* (RoI *Pooling*) 31
2.8.4 RPN *Loss Function* 32
2.8.5 Confusion Matrix 32
2.8.6 Augmentasi 32

2.9 *Precision* dan *Recall* 33

BAB 3 PERANCANGAN ARSITEKTUR STEREO R-CNN 35

3.1 Data Masukan (*Training Data*) 35
3.2 Arsitektur Stereo R-CNN 36

3.2.1 ResNet-101 FPN 37
3.2.2 Stereo RPN 39

3.2.3 Penyelarasan RoI / *RoI Alignment* 41
3.2.4 Regresi Stereo 43
3.2.5 Prediksi *Keypoint* 45
3.2.6 Estimasi *bounding box* tiga dimensi 47

BAB 4 PENGAMBILAN DATA DAN PENGUJIAN SISTEM 51

4.1 Training Neural Network 51
4.2 Testing Neural Network 52
4.3 Data Demonstrasi Algoritma Stereo R-CNN 60
4.4 Kecepatan Deteksi Algoritma Stereo R-CNN 62
4.5 Pengaruh Jarak dan Ukuran Objek Terhadap Akurasi 63

BAB 5 PENUTUP 65

5.1 Kesimpulan 65
5.2 Saran 65

DAFTAR PUSTAKA 67

LAMPIRAN 69

BIODATA PENULIS 96

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

Gambar 2.1. Intepretasi gambar oleh komputer 5
Gambar 2.2. Pemrosesan gambar oleh komputer..... 6
Gambar 2.3 Hasil interpolasi biliner and interpolasi bikubik..... 8
Gambar 2.4. Perbandingan hasil interpolasi Mean, Gaussian dan Median 10
Gambar 2.5. Ilustrasi poin 3D pada bidang gambar..... 11
Gambar 2.6. Permodelan posisi kedalaman dan sumbu optik kamera .. 12
Gambar 2.7. Metode korelasi silang dengan jendela..... 13
Gambar 2.8. Metode korelasi silang dengan pencocokan fitur 13
Gambar 2.9. Bidang Epipolar..... 14
Gambar 2.10. Garis Epipolar 15
Gambar 2.11. Diagram sistem syaraf utama manusia 18
Gambar 2.12. Diagram jaringan syaraf tiruan sederhana 19
Gambar 2.13. Visualisasi single layer feedforward neural network..... 20
Gambar 2.14. Visualisasi multi layer feedforward neural network..... 21
Gambar 2.15. Visualisasi recurrent neural network 22
Gambar 2.16. Arsitektur Convolutional Neural Network sederhana untuk mengenali tulisan tangan..... 23
Gambar 2.17. Ilustrasi max pooling 25
Gambar 2.18. Visualisasi metode zero padding 26
Gambar 2.19. Visualisasi pergerakan jendela kernel dengan stride 1 ... 26
Gambar 2.20. Ilustrasi proses dropout 29
Gambar 2.21. Arsitektur sederhana Faster R-CNN..... 30
Gambar 2.22. Illustrasi Intersection over Unit 31
Gambar 2.23. Ilustrasi hasil skor dan variasi IoU 31
Gambar 2.24. Precision dan Recall 33
Gambar 3.1. Sampel data training (kamera kiri) 35
Gambar 3.2. Sampel data training (kamera kanan) 35
Gambar 3.3. Visualisasi data training yang dihasilkan LIDAR 35
Gambar 3.4. Pengaturan tata letak yang digunakan untuk pengambilan dataset KITTI Stereo 2015 yang digunakan pada penelitian ini. 36
Gambar 3.5. Visualisasi arsitektur stereo R-CNN yang diusulkan 37
Gambar 3.6. Perbedaan antara RNN dengan NN konvensional..... 38
Gambar 3.7. Perbedaan penugasan target untuk regresi dan klasifikasi 40

Gambar 3.8. Visualisasi ROI Align yang memangkas patch yang tepinya tidak sesuai dengan batasan cell 42
Gambar 3.9. Titik hitam adalah sel di peta fitur, sementara titik merah adalah sel hasil RPN 42
Gambar 3.10. Relasi antara orientasi θ, azimuth β, dan viewpoint ($\theta + \beta$). 44
Gambar 3.11. Ilustrasi keypoint semantik 3D, keypoint perspektif 2D, dan keypoint batas. 45
Gambar 3.12. Estimasi sparse bounding box 3D 47
Gambar 4.1. Hasil tangkapan layar proses training..... 52
Gambar 4.2. Plot precision-recall deteksi bounding box 2D..... 55
Gambar 4.3. Plot precision-recall deteksi bounding box 3D..... 58
Gambar 4.4. Plot precision-recall deteksi orientasi mobil 58
Gambar 4.5. Demonstrasi Stereo R-CNN. (atas: bounding box 3D; bawah : klasifikasi & bounding box 2D; kanan: bird eye's view) 61
Gambar 4.6. Demonstrasi Stereo R-CNN dengan gambar mobil yang terpotong sebagian. 61
Gambar 4.7. Demonstrasi Stereo R-CNN dengan beberapa kendaraan tidak terdeteksi. 62
Gambar 4.8. Relasi antara disparitas dengan error jarak terhadap jarak objek yang dideteksi..... 63

DAFTAR TABEL

Tabel 4.1. Hardware yang digunakan pada proses training..... 51
Tabel 4.2. Kriteria tingkat kesulitan (mudah) 52
Tabel 4.3. Kriteria tingkat kesulitan (normal) 53
Tabel 4.4. Kriteria tingkat kesulitan (sulit) 53
Tabel 4.5. Data precision-recall deteksi bounding box 2D 53
Tabel 4.6. Data precision-recall deteksi bounding box 3D 56
Tabel 4.7. Tabel precision-recall deteksi orientasi mobil..... 59
Tabel 4.8. Hardware yang digunakan pada proses deteksi..... 62
Tabel 4.9. Hubungan antara jumlah objek pada gambar dengan lama waktu pemrosesan 63

Halaman ini sengaja dikosongkan

BAB 1
PENDAHULUAN

Dalam bab ini akan dipaparkan tentang latar belakang penelitian, permasalahan yang akan diselesaikan, tujuan dari penelitian, metodologi yang digunakan, sistematika penulisan serta relevansi penelitian ini terhadap penelitian sejenis atau lebih rumit di masa mendatang.

1.1 Latar Belakang

Mobil otonom merupakan salah satu inovasi yang belakangan ini muncul akibat perkembangan teknologi komputasi yang semakin mumpuni. Mobil otonom beroperasi dengan meminimalkan campur tangan manusia sehingga dapat mengurangi kecelakaan lalu lintas yang berdasarkan data investigasi kecelakaan KNKT mayoritasnya disebabkan oleh kesalahan pengemudi. Untuk menunjang kinerja mobil otonom, dibutuhkan sistem deteksi halangan yang mampu mendeteksi halangan dengan akurat. Berbagai macam sensor diperlukan oleh mobil otonom agar dapat bekerja dan melakukan deteksi halangan. Sensor yang umumnya digunakan pada mobil otonom adalah Lidar, namun sensor tersebut memiliki keterbatasan pada rentang deteksi serta adanya komponen bergerak yang membutuhkan perawatan lebih, rentan terhadap perubahan cuaca, serta harganya yang relatif mahal.

1.2 Perumusan Masalah

Dibutuhkan sistem deteksi halangan yang mampu menunjang kinerja mobil otonom dengan baik, namun dengan komponen yang lebih terjangkau. Karenanya, dilakukan deteksi menggunakan kamera. Namun, kamera tidak memiliki kemampuan untuk melakukan deteksi terhadap objek yang ada di sekitar mobil otonom jika tidak dilengkapi sebuah sistem penunjang yang mampu melakukan proses deteksi halangan. Oleh karena itu, diperlukan suatu algoritma untuk mendeteksi objek tersebut di ruang tiga dimensi.

1.3 Batasan Masalah

Perlu diperhatikan batasan dalam penelitian ini antara lain:

a. Objek yang dideteksi pada penelitian ini dibatasi pada mobil.
b. Training Neural Network dilakukan menggunakan dataset training yang tersedia pada dataset KITTI [1].
c. implementasi klasifikasi gambar menggunakan pemrograman berbasis Python.

1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah merancang sistem yang mampu untuk melakukan deteksi obstacle dan menghasilkan *bounding box* 3D dan menggunakan gambar dari kamera stereo.

1.5 Metode Penelitian

Penelitian ini akan dilaksanakan dengan melalui beberapa tahap proses yang telah dirancang sebagai berikut :

1.5.1 Studi Literatur

Dalam tahap Studi Literatur, akan dipelajari mengenai konsep pra-pemrosesan gambar agar dapat digunakan pada jaringan saraf tiruan, konsep pembuatan peta kedalaman berdasarkan *photometric alignment*, konsep jaringan saraf tiruan R-CNN yang akan digunakan, dan proses pembuatan *key point* untuk *bounding box* tiga dimensi.

1.5.2 Perancangan Sistem

Dalam tahap Perancangan Sistem, program jaringan saraf tiruan akan dibentuk, serta kamera stereo akan dirangkai dan diintegrasikan dengan perangkat lunak yang telah dibuat.

1.5.3 Penyusunan Buku Tugas Akhir

Dalam tahap Penyusunan Buku Tugas Akhir, laporan terkait hasil proses yang telah dilakukan pada penelitian ini akan disusun dan disajikan dalam bentuk buku.

1.6 Sistematika Penulisan

BAB 1 PENDAHULUAN

Dalam pendahuluan akan dijelaskan mengenai latar belakang, permasalahan, batasan masalah, tujuan, metodologi, sistematika penulisan, serta relevansi dari penelitian.

BAB 2 TEORI PENUNJANG ALGORITMA STEREO R-CNN

Dalam teori penunjang dijelaskan mengenai teori dasar dari penelitian yang dilakukan. Teori dasar yang mendasari konsep dari Stereo R-CNN serta beberapa library pendukung yang digunakan untuk proses training data pada model yang dibangun

BAB 3 PERANCANGAN ARSITEKTUR STEREO R-CNN

Perancangan sistem untuk deteksi halangan menggunakan Stereo R-CNN beserta rancangan *network* yang akan digunakan. Selanjutnya dilakukan proses uji data dengan data sebenarnya.

BAB 4 PENGAMBILAN DATA DAN PENGUJIAN SISTEM

Dalam pengujian dan analisis data akan dilakukan proses implementasi untuk pengujian model pada CNN dan pengujian data serta membandingkan keluaran yang dihasilkan antara data hasil pelatihan dan data yang sebenarnya.

BAB 5 PENUTUP

Dalam Bab ini akan dijelaskan suatu kesimpulan terkait hasil yang telah dicapai pada penelitian yang telah dilakukan.

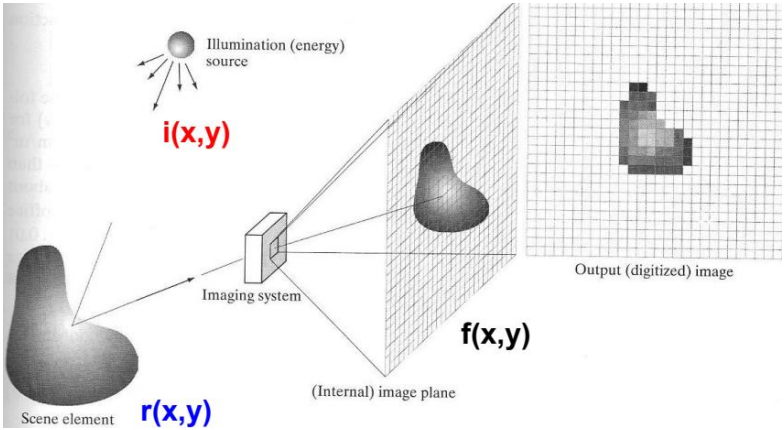
Halaman ini sengaja dikosongkan

BAB 2
TEORI PENUNJANG ALGORITMA STEREO R-CNN

Dalam bab ini akan dijelaskan mengenai teori dasar untuk membangun algoritma *Stereo R-CNN*. Arsitektur dan beberapa teori pendukung lainnya dijelaskan dengan tujuan untuk memberikan gambaran umum mengenai metode yag digunakan untuk melakukan deteksi objek pada gambar.

2.1 Intepretasi Gambar

Gambar merupakan salah satu objek 2 dimensi yang dapat diproses oleh komputer. Komputer melihat gambar sebagai fungsi $f(x, y)$ yang menyatakan intensitas pada posisi (x, y) seperti yang divisualisasikan pada Gambar 2.1. Gambar berwarna memiliki setiap komponen penyusun warna (umumnya merah, hijau, dan biru) yang dinyatakan sekaligus pada satu posisi dalam fungsi $f(x, y) = [red(x, y)\ green(x, y)\ blue(x, y)]$'.



Gambar 2.1. Intepretasi gambar oleh komputer

2.2 Penginderaan Komputer

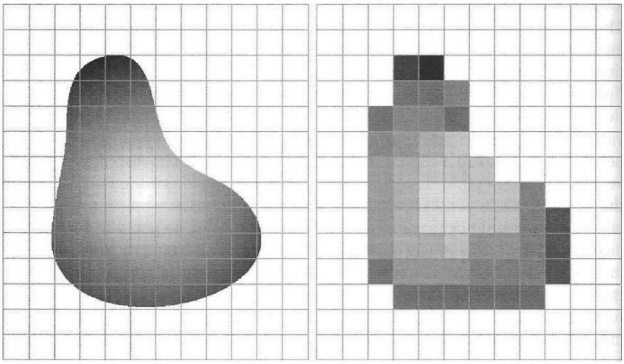
Penginderaan komputer merupakan salah satu bidang keilmuan yang mempelajari proses mengenai cara agar komputer dapat mengenali objek yang diamati seperti gambar atau video [2]. Komputer akan

mengekstraksi gambar sebagai data informasi yang akan digunakan untuk klasifikasi gambar dan deteksi objek pada gambar.

Pada dasarnya, pengindraan komputer mencoba meniru cara kerja sistem indra penglihatan manusia (*Human Vision*). Pada komputer, hal ini dilakukan dengan melakukan penangkapan citra atau video melalui kamera, lalu dilakukan proses analisis terhadap gambar tersebut. Hasil analisis digunakan untuk melakukan keputusan-keputusan yang dibuat berdasarkan kondisi citra atau video yang ditangkap oleh kamera.

Pengindraan komputer dibuat agar dapat membantu manusia melakukan proses pengamatan dan pengambilan keputusan yang sulit jika dilakukan dalam kondisi yang spesifik. Pengindraan komputer saat ini sering digunakan untuk mendeteksi wajah pada gambar (*face detection*), mengenali ekspresi wajah (*facial expression recognition*) dan pengenalan pola lainnya. Dalam implementasinya sering digunakan bersamaan dengan jaringan syaraf tiruan (*artificial neural network*) [2].

Terdapat tiga tahapan dalam pemrosesan gambar komputer yang pada umumnya dilakukan operasi pada gambar digital (diksrit). Pertama, dilakukan proses pengambilan sampel ruang dua dimensi pada grid regular. Setelah itu, sampel-sampel tersebut dibulatkan ke bilangan bulat terdekat di mana setiap sampel adalah piksel dengan nilai yang bervariasi dari 0 hingga 255. Hasil dari pemrosesan gambar komputer ini dapat divisualisasikan seperti pada Gambar 2.2.



Gambar 2.2. Pemrosesan gambar oleh komputer

2.3 Pemrosesan Gambar

Operasi pemrosesan gambar bertujuan untuk mengubah gambar yang ada f ke gambar baru g, di mana parameter domain atau rentang f dapat diubah. Terdapat bermacam-macam metode pemrosesan gambar, diantaranya adalah :

2.3.1 Interpolasi

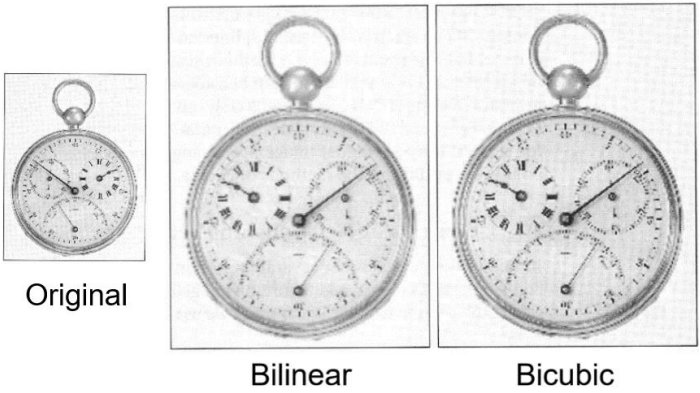
Interpolasi adalah metode untuk menentukan suatu nilai baru dalam jangkauan nilai-nilai yang diketahui sebelumnya. Nilai baru yang dihasilkan dari interpolasi tidaklah selalu tepat, sangat mungkin terdapat galat dengan nilai yang sebenarnya [3]. Pengolahan ukuran citra digital meliputi perkecilan dan perbesaran ukuran gambar. Yang dimaksud perkecilan atau perbesaran yaitu menampilkan gambar dengan jumlah pixel yang lebih sedikit atau lebih banyak. Proses memperkecil gambar dapat dilakukan dengan mengambil sampel pixel tertentu dari gambar awal. Pada proses membesarkan ukuran gambar terdapat sebuah masalah, yaitu kita tidak mengetahui informasi untuk menentukan warna dari pixel yang akan ditambahkan. Terdapat beberapa macam jenis interpolasi, misalnya:

a. Interpolasi Bilinier
   Interpolasi Bilinier merupakan metode perluasan dari interpolasi linier. Interpolasi bilinier menghitung nilai piksel $v(x, y)$ sebagai Persamaan (2.1) dimana a, b, c, dan d ditentukan dari *nearest neighbour* (x, y). Hasil dari interpolasi bilinier divisualisasikan pada Gambar 2.3.

$$v(x, y) = ax + by + cxy + d \tag{2.1}$$

b. Interpolasi Bikubik
   Interpolasi bikubik digunakan di sebagian besar program pengeditan gambar komersial. Interpolasi bilinier menghitung nilai piksel $v(x, y)$ sebagai Persamaan (2.2) dimana a_{ij} ditentukan dari 16 *nearest neighbour* (x, y). Hasil dari interpolasi bilinier divisualisasikan pada Gambar 2.3.

$$v(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j \tag{2.2}$$



Gambar 2.3 Hasil interpolasi biliner and interpolasi bikubik

2.3.2 Transformasi Spasial Geometris

Transformasi spasial geometris adalah suatu fungsi yang domain dan jangkauannya merupakan himpunan titik. Terdapat bermacam-macam jenis transformasi spasial geometrik, misalnya :

a. Translasi

Translasi adalah istilah yang digunakan dalam geometri untuk menggambarkan fungsi yang memindahkan objek dari jarak tertentu. Pada translasi, setiap titik objek harus dipindahkan ke arah yang sama dan untuk jarak yang sama. Fungsi translasi dapat dinyatakan seperti pada Persamaan (2.3) dimana x dan y menyatakan koordinat akhir, v dan w menyatakan koordinat asal, sementara t_x dan t_y menyatakan ukuran dan arah translasi.

$$[x \quad y \quad 1] = [v \quad w \quad 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \tag{2.3}$$

b. Skala

Skala mengacu pada rasio perbesaran atau pengecilan gambar dibandingkan dengan ukuran objek sebelumnya. Fungsi skala dapat dinyatakan seperti pada Persamaan (2.4) dimana x dan y menyatakan koordinat akhir, v dan w menyatakan koordinat asal, sementara c_x dan c_y menyatakan ukuran skala.

$$[x \quad y \quad 1] = [v \quad w \quad 1] \begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

c. Rotasi
Rotasi adalah pergerakan geometris memutar terhadap titik tertentu. Jumlah rotasi dijelaskan dalam derajat. Jika derajatnya positif, rotasi dilakukan berlawanan arah jarum jam; jika negatif, rotasi searah jarum jam. Fungsi rotasi dapat dinyatakan seperti pada Persamaan (2.5) dimana x dan y menyatakan koordinat akhir, v dan w menyatakan koordinat asal, sementara θ menyatakan sudut rotasi.

$$[x \quad y \quad 1] = [v \quad w \quad 1] \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

2.3.3 *Kernel*

Dalam pemrosesan gambar, *kernel* adalah matriks kecil. *Kernel* juga dikenal sebagai *mask*. Konvolusi *kernel* dan *image* menghasilkan berbagai jenis efek seperti *blur*, *sharpen*, *emboss* dan sebagainya. Itu sebabnya, *kernel* juga dikenal sebagai matriks konvolusi. Jadi, kita dapat mengatakan bahwa *kernel* adalah matriks yang digunakan untuk menghasilkan berbagai jenis efek pada gambar melalui operasi konvolusi. Secara matematis, kernel dapat dinyatakan seperti pada Persamaan (2.6) dan Persamaan (2.7) dimana $g(x,y)$ adalah gambar yang difilter, $f(x,y)$ adalah gambar asli dan H adalah kernel, yang merupakan matriks kecil. Ukuran dari kernel dapat beragam, namun umumnya ukuran kernel yang digunakan adalah ukuran 3×3.

$$g(x,y) = H * f(x,y) \tag{2.6}$$

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} H(s,t) * f(x-s, y-t) \tag{2.7}$$

2.3.4 *Filter*

Filter adalah suatu proses membersihkan gambar sehingga memungkinkan untuk menyoroti informasi tertentu secara selektif. Terdapat beberapa macam teknik *filter*, misalnya :

a. Filter *Mean*
   Filter *mean* merupakan salah satu jenis filter yang menghitung nilai rata-rata dari elemen yang ada pada kernel. Filter *mean* memiliki *kernel* $H(x, y)$ berukuran 3×3 dengan elemen seperti pada Persamaan (2.8).

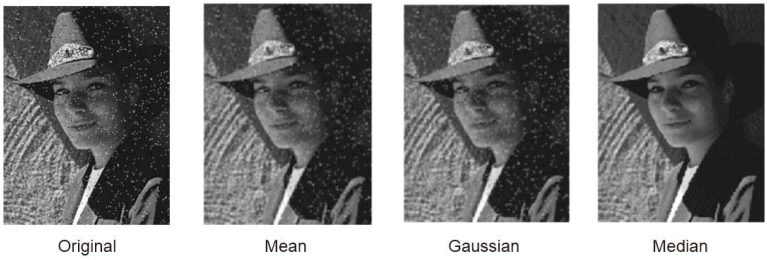
$$H(x,y) = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.8}$$

b. Filter *Gaussian*
   Filter *gaussian* merupakan salah satu jenis filter yang memberikan bobot yang lebih kecil ke piksel lebih jauh dari pusat *kernel*. Filter *gaussian* memiliki *kernel* $H(x, y)$ berukuran 3×3 dengan elemen seperti pada Persamaan (2.9).

$$H(x,y) = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.9}$$

c. Filter *Median*
   Filter *median* merupakan salah satu jenis filter yang menggantikan nilai piksel dengan *median* nilai intensitas tetangga. Filter *median* tidak menggunakan kernel, melainkan mencari nilai median dari intensitas tetangga di sekitarnya.

Hasil dari filter mean, gaussian dan median telah divisualisasikan dan dapat dilihat pada Gambar 2.4



Gambar 2.4. Perbandingan hasil interpolasi Mean, Gaussian dan Median

2.3.5 Konvolusi

Konvolusi adalah korelasi silang di mana filter diputar secara horizontal dan vertikal sebelum diterapkan pada gambar. Konvolusi dapat dinyatakan secara matematis dengan Persamaan (2.10), atau umumnya ditulis sebagai Persamaan (2.11).

$$G(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v] \tag{2.10}$$

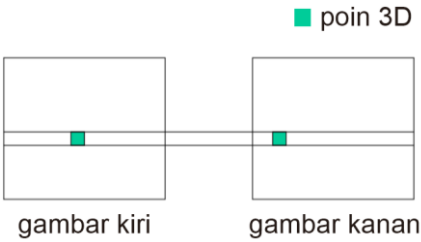
$$G = H * f \tag{2.11}$$

Konvolusi bersifat asosiatif, artinya $f * (G * I) = (F * G) * I$. Konvolusi juga memungkinkan efek filter dianalisis menggunakan analisis Fourier.

2.4 Penginderaan *Stereo*

Penginderaan stereo adalah proses ekstraksi informasi tiga dimensi dari gambar digital dengan membandingian informasi tentang adegan dari dua titik sudut pandang yang berbeda. Informasi tiga dimensi dapat diekstraksi dengan memeriksa posisi relatif objek dalam dua panel.

Persepsi kedalaman muncul dari perbedaan lokasi gambar dari titik tiga dimensi yang sama ketika diproyeksikan di bawah perspektif ke dua kamera berbeda (*disparity*). Gambar 2.5 menampilkan ilustrasi hasil tangkapan dua kamera terhadap sebuah poin tiga dimensi dimana nilai *disparity* poin tersebut dapat diukur menggunakan Persamaan (2.12) dimana d melambangkan nilai *disparity*, sementara x_{kiri} dan x_{kanan} masing-masing melambangkan posisi poin tiga dimensi pada sumbu x gambar.

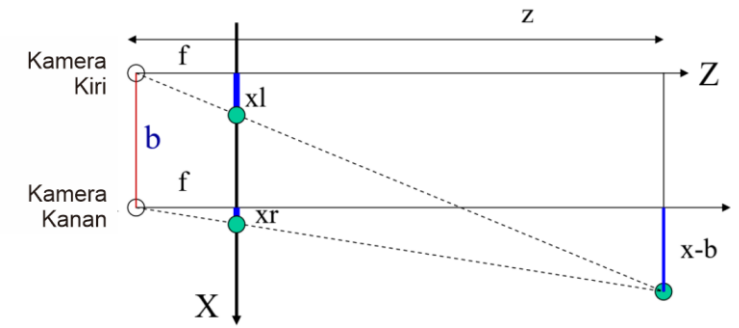


Gambar 2.5. Ilustrasi poin 3D pada bidang gambar

$$d(x) = x_{kiri} - x_{kanan} \tag{2.12}$$

Dalam proses penginderaan stereo tradisional digunakan dua kamera di mana kedua kamera tersebut dipindahkan secara horizontal dan berguna untuk memperoleh dua sudut pandang yang berbeda dari suatu pemandangan (mirip dengan sudut pandang manusia).

Sumbu optik dari 2 kamera yang paralel dapat dimodelkan secara sederhana seperti pada Gambar 2.6 dimana X menyatakan bidang gambar, Z menyatakan kedalaman (*depth*), f menyatakan jarak titik fokus kamera, b menyatakan jarak antara kedua kamera (*baseline*), serta xl dan xr masing-masing menyatakan koordinat poin tiga dimensi pada bidang gambar.



Gambar 2.6. Permodelan posisi kedalaman dan sumbu optik kamera

Apabila kamera paralel, maka kedalaman dari suatu objek tiga dimensi pada gambar dapat dihitung menggunakan Persamaan (2.13), (2.14), dan (2.15) dimana d menyatakan nilai *disparity*. Metode penentuan kedalaman dari perbedaan nilai *disparity* ini disebut sebagai triangulasi.

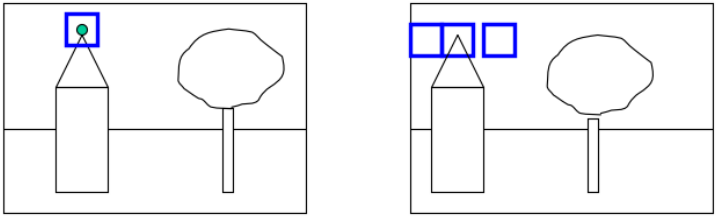
$$z = f\frac{b}{d} \tag{2.13}$$

$$x = xl\frac{z}{f} \tag{2.14}$$

$$y = yl\frac{z}{f} \tag{2.15}$$

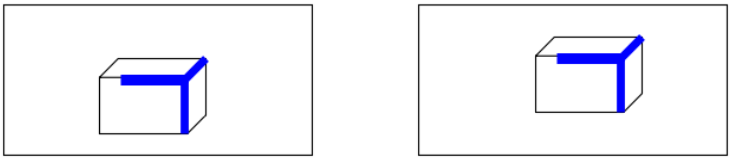
Permasalahan dari metode triangulasi ini adalah kita perlu mengetahui jarak titik fokus lensa (f) dan jarak antara kedua kamera (b) yang dapat diselesaikan menggunakan pengetahuan mengenai struktur kamera serta kalibrasi kamera. Permasalahan lain yang harus diselesaikan sebelum dapat menggunakan metode triangulasi ini adalah perlunya menemukan titik yang sesuai di (xr, yr) untuk setiap (xl, yl), atau umumnya disebut sebagai permasalahan korespondensi.

Terdapat bermacam-macam metode untuk menyelesaikan permasalahan korespondensi ini, misalnya melakukan korelasi silang menggunakan kumpulan jendela kecil seperti pada Gambar 2.7, atau pencocokan fitur simbolik, biasanya menggunakan segmen / sudut seperti pada Gambar 2.8.



Gambar 2.7. Metode korelasi silang dengan jendela

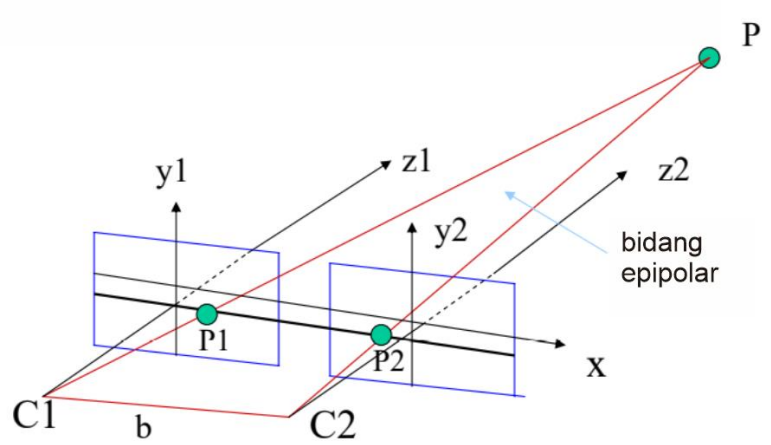
Permasalahakan korespondensi memerlukan waktu komputasi yang panjang apabila setiap titik di gambar kiri harus dibandingkan dengan semua titik di gambar kanan. Untuk mempersingkat waktu komputasi, dapat digunakan batasan geometri *epipolar* untuk menyelesaikan permasalahan korespondensi ini.



Gambar 2.8. Metode korelasi silang dengan pencocokan fitur

Ketika dua kamera melihat objek tiga dimensi dari dua posisi yang berbeda, ada sejumlah hubungan geometris antara titik tiga dimensi dan proyeksi mereka ke gambar dua dimensi yang mengarah ke batasan antara

titik gambar. Batasan inilah yang disebut dengan geometri *epipolar*. Geometri epipolar divisualisasikan pada Gambar 2.9 dimana C1 dan C2 menyatakan kamera 1 dan 2, sementara P1 dan P2 menyatakan posisi objek tiga dimensi P di gambar 1 dan 2.



Gambar 2.9. Bidang Epipolar

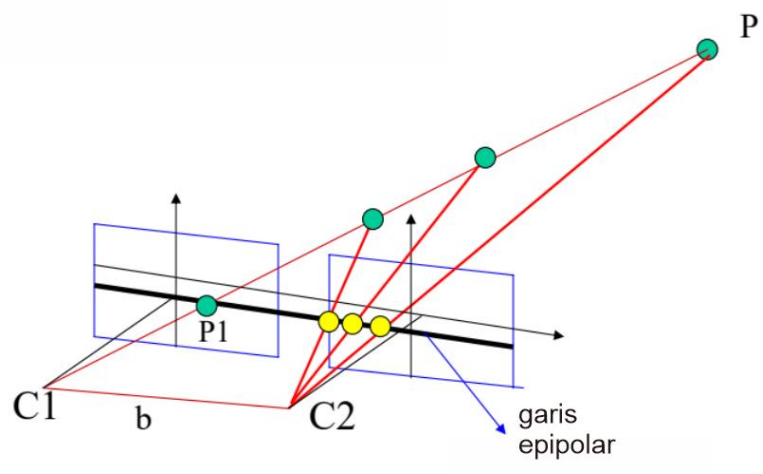
Bidang epipolar memotong bidang gambar dan membentuk garis epipolar di setiap bidang seperti pada Gambar 2.10. Karenanya, pasangan korespondensi untuk P1 (atau P2) di gambar lain harus terletak pada baris epipolar, jadi korespondensinya hanya perlu dicari di sepanjang garis ini. Penggunaan garis *epipolar* ini dapat menghemat proses komputasi secara signifikan.

2.5 Machine Learning

Machine learning merupakan salah satu cabang dalam sistem kecerdasan buatan yang memungkinkan komputer dapat belajar dari data dan pengalamannya. Pada [4] dikatakan bahwa: "*Machine learning* merupakan cabang ilmu yang mempelajari cara membuat komputer dapat belajar tanpa memprogramnya secara eksplisit".

Machine learning diperlukan karena permasalahan yang ingin diselesaikan terlalu kompleks untuk diprogram secara analitis. Sebagai ganti dari kita menyelesaikannya secara analitis, kita menyediakan data dalam jumlah besar ke algoritma *machine learning* dan membiarkan

algoritma *machine learning* tersebut mencari model seperti yang diinginkan oleh pembuat program.



Gambar 2.10. Garis Epipolar

Setiap algoritma tentunya memiliki keunggulan dan kelemahannya. Beberapa contoh tugas yang paling baik diselesaikan dengan pembelajaran mesin meliputi: (1) Pengenalan Pola, misalnya mengenali objek pada gambar, identitas wajah, ekspersi, dan ucapan; (2) Pengenalan Anomali, misalnya transaksi kartu kredit serta pola pembacaan sensor yang tidak biasa; dan (3) Prediksi, misalnya prediksi pergerakan harga saham, atau prediksi barang yang akan dibeli oleh seseorang berdasarkan histori belanjanya. Sementara itu, hal yang bukan merupakan pengenalan pola, anomali maupun prediksi sebaiknya diselesaikan menggunakan metode analitis karena proses komputasinya akan lebih cepat. Dalam *machine learning* terdapat beberapa metode dalam proses pembelajaran yaitu :

2.5.1 *Supervised Learning*

Supervised learning merupakan metode pembelajaran dimana data yang akan digunakan sebagai pembelajaran diberi pelabelan untuk dapat diklasifikasikan per kelas dan dijadikan sebagai model prediksi dari data

tersebut . Metode ini disebut sebagai s*upervised learning* karena proses pembelajaran algoritma dari dataset pembelajaran dapat dianggap sebagai guru yang mengawasi proses pembelajaran. Jawaban yang benar diketahui, kemudian algoritma iteratif membuat prediksi pada data pelatihan dan dikoreksi. Belajar berhenti ketika algoritma mencapai tingkat kinerja yang dapat diterima. *Supervised learning* dapat dibedakan menjadi permasalahan regresi dan permasalahan klasifikasi.

a. Permasalahan Regresi
   Permasalahan regresi merupakan permasalahan dimana nilai variabel keluarannya merupakan bilangan terukur, misalnya: berat, tinggi, dan ukuran.

b. Permasalahan Klasifikasi
   Permasalahan klasifikasi merupakan permasalahan dimana nilai variabel keluarannya merupakan kategori, misalnya: penyakit, atau bukan penyakit.

Ada pula beberapa jenis masalah yang dibangun di atas klasifikasi dan regresi misalnya rekomendasi dan prediksi deret waktu.

2.5.2 *Unsupervised Learning*

Unsupervised learning merupakan metode pembelajaran dimana data pembelajaran yang akan digunakan tidak diberikan pelabelan. Tujuan *unsupervised learning* adalah untuk memodelkan struktur atau distribusi yang mendasari data untuk mempelajari lebih lanjut tentang data tersebut.

Metode pembelajaran ini disebut *unsupervised learning* karena tidak seperti *supervised learning*, tidak disediakan jawaban yang benar dan tidak ada guru. Algoritma dibiarkan sendiri untuk menemukan dan menyajikan struktur yang menarik dalam data. Permasalahan yang dapat diselesaikan menggunakan metode *unsupervised learning* dapat dikelompokkan menjadi permasalahan *clustering* dan asosiasi.

a. Permasalahan *Clustering*
   Permasalahan *clustering* adalah menemukan *cluster* yang melekat dalam data, misalnya mengelompokkan pelanggan berdasarkan perilaku belanja mereka.

b. Permasalahan Asosiasi
   Permasalahan asosiasi adalah permasalahan di mana Anda ingin menemukan aturan yang menggambarkan sebagian besar data Anda, seperti orang yang membeli X juga cenderung membeli Y.

Beberapa algoritma *unsupervised learning* yang cukup populer misalnya K-Means untuk permasalahan *clustering* dan Apriori untuk permasalahan asosiasi.

2.5.3 *Reinforcement Learning*

Pada *Reinforcement learning* menerapkan metode diberlakukan hadiah sebagai hasil dari pembelajaran yang positif dan hukuman pada hasil pembelajaran yang negatif. Komputer dilatih dengan cara belajar dari pengalaman melalui *trial and error* saat melakukan tugas tertentu. Dan memaksimalkan *reward* untuk belajar secara berulang-ulang.

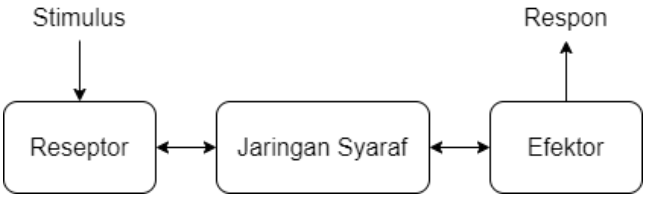
2.5.4 *Deep Learning*

Deep learning adalah jenis pembelajaran mesin yang melatih komputer untuk melakukan tugas yang mirip manusia, seperti mengenali ucapan, mengidentifikasi gambar, atau membuat prediksi. Alih-alih mengatur data untuk dijalankan melalui persamaan yang telah ditentukan sebelumnya, *deep learning* menetapkan parameter dasar tentang data dan melatih komputer untuk belajar sendiri dengan mengenali pola menggunakan banyak lapisan pemrosesan.

Deep Learning adalah area baru dalam penelitian *Machine Learning*, yang diperkenalkan dengan tujuan untuk membuat *Machine Learning* lebih dekat ke salah satu tujuan awalnya: Kecerdasan Buatan.

2.6 Jaringan Syaraf Tiruan (*Artificial Neural Network)*

Sistem jaringan syaraf tiruan merupakan salah satu pengembangan algoritma yang diterapkan pada mesin atau komputer untuk mengerjakan suatu tugas dengan cara kerja selayaknya otak manusia [5]. Dalam sistem otak manusia mampu untuk membangun, mengambil kesimpulan, dan belajar dari pengalaman yang telah diterima. Sistem jaringan syaraf utama pada manusia terbagi atas tiga tahap seperti pada Gambar 2.11.



Gambar 2.11. Diagram sistem syaraf utama manusia

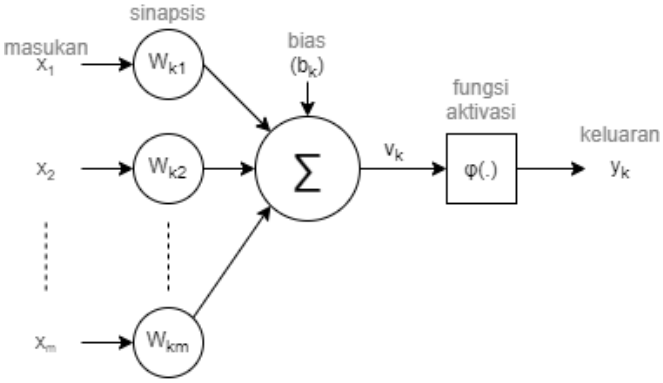
Reseptor berfungsi untuk mengubah stimulus dari rangsangan yang membawa informasi untuk diproses menuju ke jaringan syaraf. Sedangkan *effector* berfungsi untuk merubah informasi dari jaringan syaraf untuk dijadikan sebagai keluaran.

Unit pemrosesan dalam jaringan syaraf (*neural network)* disebut sebagai *neuron* atau *node*. Dengan kata lain suatu sistem proses komputasional yang terinspirasi oleh bagaimana jaringan syaraf pada manusia bekerja. Dalam implementasinya *neuron* dapat direpresentasikan sebagai *node* yang menghubungkan antar jaringan satu dengan lainnya.

Jaringan syaraf tiruan pada manusia apabila dikelompokkan berdasarkan fungsinya, dapat dibagi menjadi tiga elemen dasar yakni Sinapsis, *Summing Junction*, dan Fungsi Aktivasi. Jaringan syaraf tiruan sederhana divisualisasikan pada Gambar 2.12.

a. Sinapsis
   Sinapsis merupakan elemen yang terdapat nilai pembobotan dan sinyal masukan. Nilai pembobotan merupakan nilai acak dengan rentang nilai negatif hingga nilai positif.

b. Summing Junction
   summing junction berfungsi sebagai penjumlahan untuk masing-masing nilai pembobotan sebelumnya. Nilai keluaran pada *neuron* diperoleh dari fungsi aktivasi. Elemen ini digunakan untuk membatasi rentang sinyal keluaran pada *neuron*.



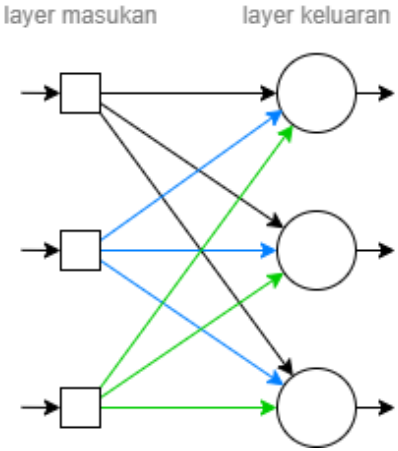
Gambar 2.12. Diagram jaringan syaraf tiruan sederhana

Pada dasarnya *neuron* pada *neural network* saling terhubung satu dengan lainnya agar dapat digunakan untuk melakukan proses pembelajaran. Secara umum arsitektur *neural network* dibagi menjadi beberapa bagian yakni :

2.6.1 *Single Layer Feedforward Neural Network / Perceptron*

Dianggap sebagai generasi pertama dari jaringan saraf, *perceptron* hanyalah model komputasi dari satu neuron. Mereka dipopulerkan oleh Frank Rosenblatt pada awal 1960 [5]. Mereka tampaknya memiliki algoritma pembelajaran yang sangat kuat dan banyak klaim besar dibuat untuk apa yang bisa mereka pelajari.

Pada tahun 1969, [6] menganalisis apa yang dapat dilakukan oleh *perceptron* dan menunjukkan keterbatasannya. Banyak orang mengira keterbatasan ini berlaku untuk semua model jaringan saraf. Namun, prosedur pembelajaran *perceptron* hingga saat ini masih banyak digunakan untuk tugas-tugas dengan ukuran vektor fitur yang sangat besar (berisi jutaan fitur).

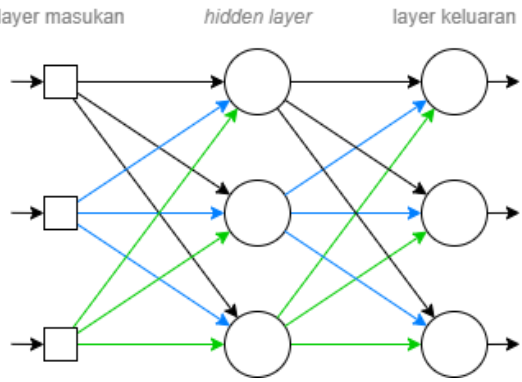


Gambar 2.13. Visualisasi single layer feedforward neural network

Arsitektur jaringan *perceptron* terbentuk dari : (1) *input layer*, dan (2) *output layer* yang terdiri atas sekumpulan *neuron*. Pada Gambar 2.13 diilustrasikan *node* masukan dalam *input layer* yang terhubung dengan *output layer*. Skema gambar tersebut merupakan *perceptron* dengan perancangan *layer* tunggal yang didasarkan perhitungan *node* pada *layer* keluaran, sedangkan untuk *layer* masukan tidak terdapat *node* perhitungan.

2.6.2 *Multi Layer Feedforward Neural Network*

Jaringan *multi layer feedforward* memiliki arsitektur yang mirip dengan jaringan *perceptron* namun dengan adanya satu atau lebih *hidden layer* yang mana *node* perhitunganya disebut sebagai *hidden neuron*, atau pada [5] [7] [8] disebut sebagai *hidden unit*. *Hidden neuron* berfungsi untuk melakukan proses perhitungan dengan orde tinggi. Gambar 2.14 menunjukan jaringan *fully connected multi layer feedforward* dimana tiap *node* di setiap *layer* terhubung ke tiap-tiap *node* yang lain pada *layer* berikutnya. Namun jika terdapat salah satu *node* yang tidak terhubung disebut sebagai *partially connected* [4].

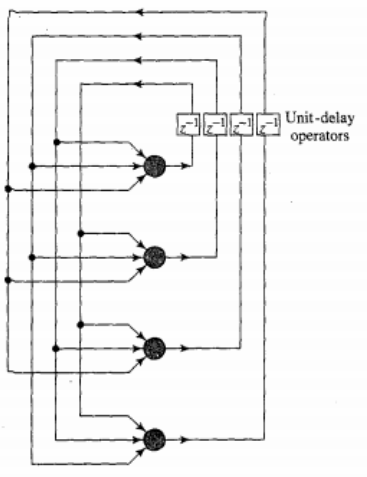


Gambar 2.14. Visualisasi multi layer feedforward neural network

2.6.3 *Recurrent Neural Network*

Recurrent neural network merupakan tipe jaringan yang berbeda dengan lainnya. *Recurrent network* mempunyai satu atau lebih *feedback loop*. Gambar 2.15 menunjukan ilustrasi *recurrent network* yang mana keluaran tiap *neuron* dijadikan sebagai masukan dari *neuron* lainnya. Tujuan dari *feedback loop* jaringan yaitu untuk meningkatkan kemampuan dalam proses pembelajaran.

RNN memiliki kemampuan yang handal karena tersusun atas *distributed hidden state* yang mampu menyimpan banyak informasi seiring berjalannya waktu secara efisien dan *non-linear dynamics* yang membuat sistem mampu melakukan proses *update* pada *hidden state* kompleks. Dengan jumlah *neuron* dan waktu yang mencukupi, RNN mampu melakukan proses komputasi apa saja.



Gambar 2.15. Visualisasi recurrent neural network

Walaupun memiliki kemampuan komputasi yang luar biasa, proses *training* pada RNN tergolong kompleks dibandingkan jenis jaringan syaraf tiruan yang lain. Proses *training* RNN lebih rumit karena dilakukan proses *backpropagation* pada banyak *layer* sedangkan *gradient* pada layer tersebut membesar dan mengecil secara eksponensial dan tidak bisa diprediksi. Proses training RNN dilakukan dalam waktu yang panjang sehingga ukuran *gradient* dapat meledak dikarenakan membesar berlebihan secara eksponensial dan dapat hilang karena mengecil berlebihan secara eksponensial. Walaupun sudah diawali dengan *initial weight* yang ideal, sulit untuk mendeteksi target *output* terkini dari input beberapa saat yang lalu.

2.6.4 *Symmetrically Connected Networks*

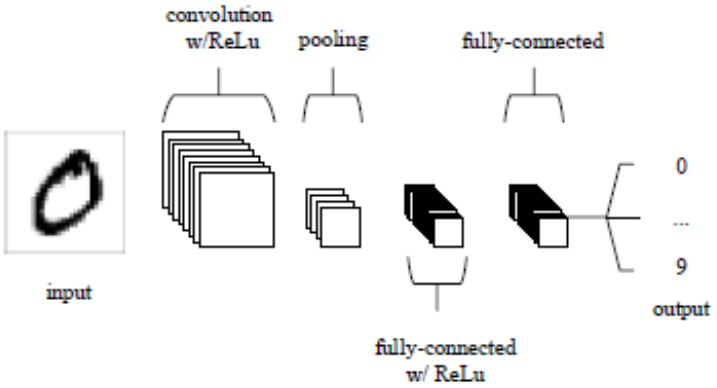
Symmetrically Connected Networks memiliki arsitektur seperti *Recurrent Neural Network*, tetapi koneksi antar unit simetris (memiliki bobot yang sama di kedua arah). *Symmetrically Connected Networks* jauh lebih mudah untuk dianalisis daripada *Recurrent Neural Network*. Jaringan ini juga lebih dibatasi dalam apa yang dapat mereka lakukan karena mereka mematuhi fungsi energi. Jaringan yang terhubung secara simetris tanpa unit tersembunyi disebut *Hopfield Nets*, sementara itu

jaringan yang terhubung secara simetris dengan unit *hidden layer* disebut *Boltzmann Machine*.

2.7 Convolutional Neural Network (CNN)

Pada tahun 1998, Yann LeCun mengembangkan sistem pengenalan tulisan tangan dengan nama LeNet [9]. Ia menggunakan metode *back propagation* dalam jaringan *feedforward* dengan menggunakan banyak *hidden layer* , beberapa memetakan replikasi unit di setiap layer dan menyatukan output dari unit replikasi terdekat. Sistem pengenalan yang memiliki jaringan luas yang mampu mengenali banyak karakter sekaligus bahkan ketika terjadi *overlap* ini dapat disempurnakan menjadi sebuah sistem *training* yang sekarang biasa dikenal dengan sebutan *Convolutional Neural Network* (CNN).

Convolutional neural network (CNN) merupakan salah satu metode dalam *deep learning* yang banyak digunakan untuk mengenali sebuah pola, seperti pengolahan gambar dan pengenalan suara. Pada Gambar 2.16 menunjukan ilustrasi dari arsitektur sederhana CNN dalam mengenali tulisan tangan.



Gambar 2.16. Arsitektur Convolutional Neural Network sederhana untuk mengenali tulisan tangan

Pendekatan fitur merupakan pendekatan yang sering digunakan oleh neural network untuk menyelesaikan masalah deteksi objek. Ia menggunakan banyak salinan berbeda dari sistem deteksi yang sama dengan posisi yang berbeda. CNN mampu melakukan replikasi melewati

batas skala dan orientasi yang telah ditentukan. Replikasi mengurangi jumlah parameter yang harus dipelajari dalam proses training. CNN menggunakan beberapa tipe fitur di mana setiap fitur memetakan deteksi replikasi miliknya dan membuat tiap bagian dari gambar direpresentasikan dalam beberapa bentuk yang berbeda.

2.7.1 *Convolution Layer*

Convolutional layer merupakan fitur ekstraksi dengan dimensi ukuran volume $W_1 * H_1 * D_1$ yang mana W_1 adalah lebar, H_1 adalah tinggi dan D_1 adalah kedalaman. Nilai keluaran dari *neuron* dalam *convolution layer* merupakan kalkulasi dari hasil yang diperoleh antara *weight* dan *local region* yang terhubung dengan volume masukan. Sedangkan nilai keluaran yang diperoleh dari volume $W_2 * H_2 * D_2$ disebut dengan *convolution maps* yang mana W_2 merupakan lebar, H_2 merupakan tinggi, dan D_2 merupakan kedalaman jika kita menggunakan D_2 filter atau *convolution kernel*. *Convolution maps* menghasilkan volume yang dihasilkan berdasarkan Persamaan (2.16), (2.17), dan (2.18). Metode untuk melakukan konvolusi telah dibahas pada subbab 2.3.5.

$$W_1 = \frac{W_1 - F + 2P}{S} + 1 \tag{2.16}$$

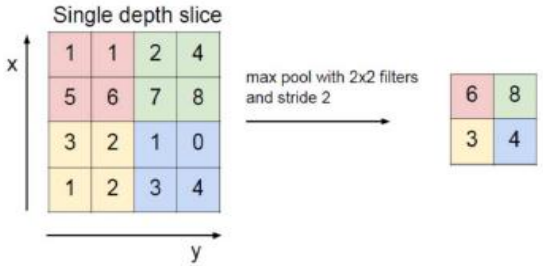
$$H_2 = \frac{H_1 - F + 2P}{S} + 1 \tag{2.17}$$

$$D_2 = K \tag{2.18}$$

2.7.2 *Pooling Layer*

Pooling layer memiliki sebuah fungsi utama berupa *down-sampling* untuk mengurangi kompleksitas dari beberapa layer. Dalam hal pengolahan gambar istilah *down-sampling* dapat diartikan sebagai mereduksi tingkat resolusi pada gambar. *Pooling layer* tidak mempengaruhi nilai dari filter. *Pooling layer* dapat dibagi menjadi 2 tipe, yaitu *max pooling* dan *average pooling*. Dalam CNN pada umumnya digunakan metode *max pooling* yaitu membagi keluaran dari *convolution layer* dan menjadi matriks-matriks kecil dan memiliki hasil akhir berupa pengambilan nilai terbesar dari matriks yang telah mengalami proses reduksi sebelumnya. Sedangkan pada *average pooling layer* bekerja

dengan cara mengambil nilai rata-rata pada tiap matriks yang telah mengalami proses reduksi sebelumnya. Pada Gambar 2.17 ditunjukkan prinsip kerja dari *max pooling* yang menggunakan filter 2x2 dengan nilai stride 2, yang artinya tiap matrik 2x2 akan diambil nilai tertinggi dan digeser sebanyak 2 kotak.



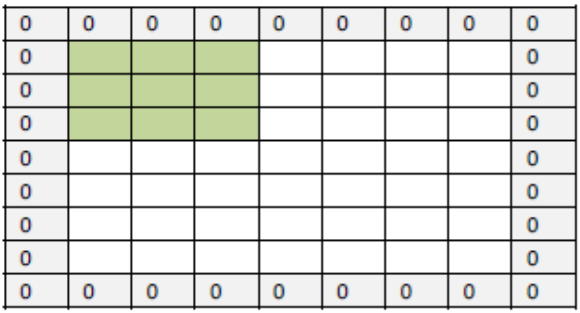
Gambar 2.17. Ilustrasi max pooling

2.7.3 *Fully Connected Layer*

Pada *fully connected layer* tiap komponen lapisan dalam jaringan akan saling terhubung dengan lapisan lainnya. Penerapan dari *fully connected layer* sama dengan *multi layer perceptron* (MLP) yang memiliki tujuan untuk mentransformasikan dimensi data agar proses klasifikasi data dapat dilakukan secara linear. Data masukan dari *convolution layer* mengalami proses transformasi terlebih dahulu menjadi data yang hanya memiliki satu dimensi atau disebut sebagai *feature vector* sebelum berubah menjadi masukan dari *fully connected layer* agar tidak kehilangan informasi spasialnya

2.7.4 Zero Padding

Zero padding adalah teknik yang digunakan untuk memanipulasi dimensi pada masukan citra untuk penambahan nilai 0 disetiap tepi citra. Metode konvolusi mempunyai kelemahan, yaitu hilangnya beberapa informasi pada citra. Metode *padding* / *zero padding* bertujuan untuk mengatasi permasalahan citra tersebut. Dengan metode *zero padding* dimensi keluaran akan sama atau dipastikan tidak berubah secara signifikan seperti dimensi masukan. Contoh aplikasi penggunaan *zero padding* dapat dilihat pada Gambar 2.18. dapat diasumsikan N = 7, F = 3, dan *Stride* = 1 maka keluaran akan menjadi 5x5 namun karna adanya penambahan satu *zero padding* maka keluaran akan tetap berukuran 7x7, yang sama seperti aktualnya.

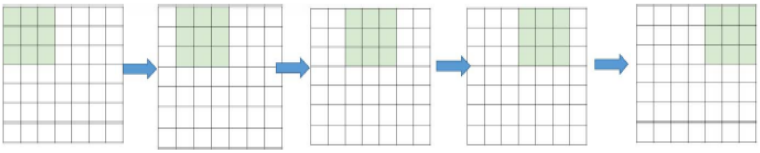


Gambar 2.18. Visualisasi metode zero padding

2.7.5 *Stride*

Terdapat beberapa pilihan pada CNN untuk menurunkan parameter-parameter dan mengurangi efek pada saat yang bersamaan. Salah satunya yaitu *stride*. Diasumsikan sebuah ilustrasi pada Gambar 2.19 gambar memiliki dimensi ukuran 7x7 dengan menggunakan *stride* 1. Akan didapatkan keluaran dengan ukuran dimensi 5x5 dan ketika digunakan *stride* 2 maka keluaran yang didapatkan memiliki ukuran dimensi 3x3. Didapatkan persamaan seperti dalam Persamaan (2.19) dimana N merupakan ukuran masukan, F merupakan ukuran filter, S merupakan ukuran stride, dan O merupakan ukuran keluaran.

$$O = 1 + \frac{N - F}{S} \tag{2.19}$$



Gambar 2.19. Visualisasi pergerakan jendela kernel dengan stride 1

2.7.6 Fungsi Softmax

Fungsi *softmax* biasa digunakan untuk melakukan klasifikasi banyak kelas. *Softmax* memberikan nilai probabilitas untuk setiap label

kelas, dimana jumlah seluruh probabilitas adalah 1. *Softmax* pada dasarnya adalah probabilitas eksponensial yang dinormalisasi dari nilai masukan sejumlah kelas pada model klasifikasi seperti pada Persamaan (2.20). Dimana y adalah nilai masukan. Operasi akan menghasilkan nilai probabilitas. Label dari data masukan akan ditentukan berdasarkan kelas dengan nilai probabilitas tertinggi.

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \tag{2.20}$$

2.7.7 *Stochastic Gradient Descent*

Dalam proses melatih sebuah model, dibutuhkan sebuah *loss function* yang dapat mengukur kualitas dari setiap bobot atau parameter tertentu. *Stochastic Gradient Descent* (SGD) merupakan algoritma pengotimalan. Tujuan pengoptimalan adalah untuk menemukan parameter yang dapat meminimalisir nilai *error* dari *loss function.* SGD adalah algoritma yang digunakan untuk memperbarui nilai bobot dan bias pada *neuron* di *neural network*. Operasi yang dilakukan hanya mengurangi bobot awal dengan sebagian nilai dari nilai gradien yang sudah kita dapat. Nilai sebagian disini diwakili oleh parameter bernama *learning rate*, seperti yang terlihat pada Persamaan (2.21) dan (2.22).

$$W_{j+1} = w_j - a\frac{\partial}{\partial_{w_j}} J(W, b) \tag{2.21}$$

$$b_{j+1} = w_j - a\frac{\partial}{\partial_{b_j}} J(W, b) \tag{2.22}$$

2.7.8 *Adaptive Moment Estimation* (Adam)

Adaptive Moment Estimation (Adam) merupakan salah satu algoritma pengoptimalan yang dapat digunakan. Terdapat beberapa kesamaan dengan *Adaptive Gradient* (Adagrad). Adam memiliki sebuah *learning rate* untuk setiap parameter dan mampu beradaptasi secara terpisah saat proses pelatihan. Adam memperbarui nilai setiap parameter seperti RMSProp. Perbedaannya, Adam menggunakan gradien yang telah diperhalus dan nilainya semakin mengecil seperti yang dapat dilihat pada Persamaan (2.23). Lalu gradien tersebut akan digunakan untuk

memperbarui parameter, seperti yang dapat dilihat pada Persamaan (2.24) dan (2.25).

$$m_j = \beta_1 . m_{j-1} + (1 - \beta_1) g_j \tag{2.23}$$

$$s_j = \beta_2 . s_{j-1} + (1 - \beta_2) \left(g_j\right)^2 \tag{2.24}$$

$$\theta_{j+1} = \theta_j - \frac{a}{\sqrt{s_j + \varepsilon}} m_j \tag{2.25}$$

2.7.9 RMSProp

RMSProp (*Root Mean Square*) adalah metode pengoptimalan berbasis *adaptive learning rate* yang diusulkan oleh Geoffrey Hinton. RMSProp memodifikasi Adagrad dengan mengganti akumulasi gradien menjadi rata-rata bergerak gradien yang diberi bobot secara kuadratik, seperti yang dapat dilihat pada Persamaan (2.26) dan (2.27).

$$s_j = \beta . s_{j-1} + (1 - \beta) \left(g_j\right)^2 \tag{2.26}$$

$$\theta_{j+1} = \theta_j - \frac{a}{\sqrt{s_j + \varepsilon}} g_j \tag{2.27}$$

2.7.10 *Cross Entropy*

Loss function adalah fungsi yang menyatakan kerugian yang dihasilkan oleh model. *Loss function* dapat dikatakan baik, ketika menghasilkan *error* yang diharapkan memiliki nilai paling rendah. Pada permasalahan klasifikasi banyak kelas, *cross entropy* adalah *loss function* yang biasa digunakan. *Cross entropy* menghitung *error* antara nilai prediksi *S* dengan nilai sebenarnya *T*, seperti pada Persamaan (2.28). Selanjutnya, nilai *error* akhir diambil dari rata-rata hasil *cross entropy*, seperti pada Persamaan (2.29).

$$D(S_i, T_i) = -\sum_j T_{ij} \log\left(S_{ij}\right) \tag{2.28}$$

$$J(W,b) = \frac{1}{n}\sum_{i} D(S_i, T_i) \tag{2.29}$$

2.7.11 *Dropout*

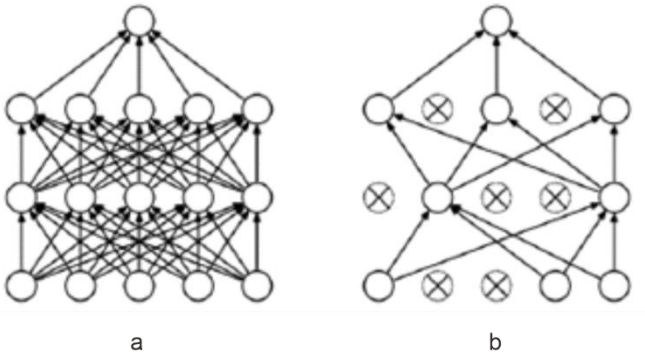
Dropout merupakan proses mencegah terjadinya *overfitting* dan juga mempercepat proses *learning*. *Dropout* mengacu kepada menghilangkan *neuron* yang berupa *hidden layer* mapun *visible layer* di dalam jaringan. Dengan menghilangkan suatu *neuron*, berarti menghilangkannya sementara dari jaringan yang ada. *Neuron* yang akan dihilangkan akan dipilih secara acak.

Pada Gambar 2.20(a) *neuron* tetap utuh pada *neural network* yang belum memakai *Dropout*, dan (b) *neural network* dengan sebagian dari *neuron* yang tidak digunakan setelah aplikasi *Dropout*.

2.7.12 *Batch Normalization*

Batch Normalization adalah teknik melakukan normalisasi terhadap *batch* atau kumpulan data masukan, seperti yang terlihat pada Persamaan (2.30). Dimana x adalah nilai masukan, μ_b adalah *mean* dari *batch*, σ_b adalah standar deviasi dari *batch*. Normalisasi dilakukan agar data memiliki *mean* mendekati nol dan standar deviasi mendekati satu.

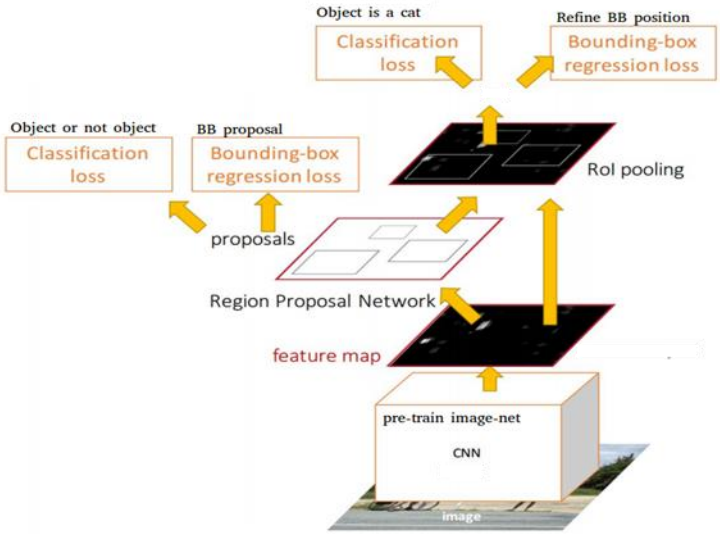
$$x_{baru} = \frac{x - \mu_b}{\sigma_b} \tag{2.30}$$



Gambar 2.20. Ilustrasi proses dropout

2.8 Region with CNN features (R-CNN)

Dalam pendeteksian objek pada gambar terdapat beberapa algoritma yang cukup populer yaitu (1) R-CNN, (2) Fast R-CNN, (3) *Faster* R-CNN, dan (3) *Mask* R-CNN. *Faster Region based Convolutional Neural Network* atau bisa disebut *Faster* R-CNN merupakan algoritma yang menggunakan *Fast* R-CNN dan *Regional Proposal Network* (RPN) dalam arsitekturnya. Secara garis besar arsitektur *Faster* R-CNN dapat dilihat pada Gambar 2.21.



Gambar 2.21. Arsitektur sederhana Faster R-CNN

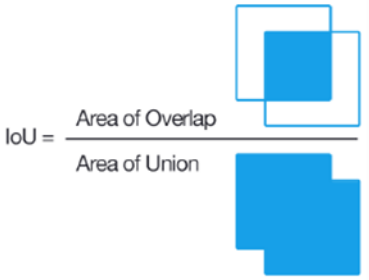
2.8.1 *Region Proposal Network*

Regional Proposal Netwok biasa disebut sebagai RPN merupakan metode yang digunakan untuk menjalankan *sliding window* supaya menghasilkan proposal *anchor* yang telah ditentukan. Proposal yang telah dihasilkan akan dibandingkan dengan *Ground Truth Box* menggunakan fungsi dari *Intersection Over Union* (IoU). Setiap proposal akan dilakukan proses konvolusi yang menghasilkan objektivitas objek pada gambar dengan ratio antara 0–1 yang menandakan adanya objek. Jika hasil dari IoU lebih besar dari 0,7 diasumsikan adanya objek namun

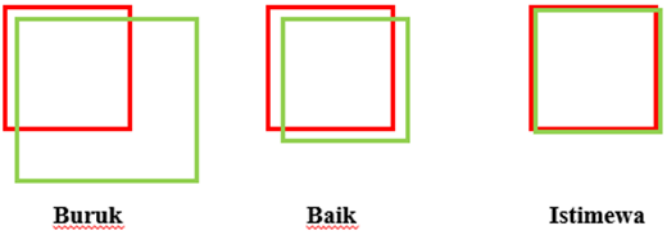
sebaliknya jika hasil yang diperoleh lebih kecil dari 0,3 diasumsikan tidak adanya objek.

2.8.2 *Intersection Over Union* (IoU)

Dalam proses penentuan letak objek terhadap suatu dataset dapat digunakan metode *Intersection Over Union* (IoU) untuk melakukan evaluasi terhadap gambar. IoU mempunyai 2 area yang digunakan dalam menentukan akurasi terhadap objek antara lain *area ground-truth bounding box* dan area yang dideteksi dalam model yang dibangun. Dalam proses untuk menentukan nilai IoU dilakukan 2 proses operasi yakni *intersection* dan *union.* Gambar 2.22 dan Gambar 2.23 mengilustrasikan variasi dari hasil skor IoU yang terdiri dari 3 penilaian yakni buruk, baik, dan istimewa.



Gambar 2.22. Illustrasi Intersection over Unit



Gambar 2.23. Ilustrasi hasil skor dan variasi IoU

2.8.3 *Region of Interest Pooling* (RoI *Pooling*)

Dalam klasifikasi gambar untuk sistem deteksi objek pada masukan *fully connected layer* ukuran gambar harus tetap dan tidak

berubah. Namun, adanya kendala yang mana keluaran dari *proposal region network* (RPN) sangat bervariasi. *Region of Interest Pooling* (RoI) dapat dijadikan untuk merubah ukuran gambar pada RPN agar sesuai dengan masukan terhadap *fully connected layer*. RoI memanfaatkan *feature maps* untuk mengeneralisasi ukuran dari tiap *region proposal network* sebelum dijadikan masukan sebagai *fully connected layer*.

2.8.4 RPN *Loss Function*

RPN *Loss Function* merupakan fungsi yang didapatkan dari hasil *regional proposal network.* Dalam RPN akan menghasilkan *loss regression* dan *loss classification.* Dapat dinyatakan melalui dalam bentuk persamaan matematis seperti pada Persamaan (2.31).

$$L\left(p_i, t_i\right) = \frac{1}{N_{cls}} \sum_i L_{cls}\left(p_i, {p^*}_i\right) \lambda \frac{1}{N_{reg}} \sum_i {p^*}_i L_{reg}\left(t_i, {t^*}_i\right) \tag{2.31}$$

2.8.5 Confusion Matrix

Confusion Matrix adalah tabel yang digunakan untuk mengetahui kualitas model yang dibangun. Dari tabel tersebut dapat dihitung tingkat akurasi, presisi, dan *recall* terhadap sistem yang dibangun. Data akurasi merupakan pengukuran seberapa sistem dapat mengklasifikasikan data. Persamaan akurasi dapat dihitung melalui Persamaan (2.32). Presisi yaitu membandingkan jumlah data kategori positif yang dapat diklasifikasikan benar dengan data keseluruhan dengan kategori positif. Persamaan presisi dapat dilihat di Persamaan (2.33). Sedangkan *recall* yaitu pengukuran data dengan klasifikasi positif dan benar oleh sistem. Persamaan *recall* dapat dilihat di Persamaan (2.34).

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.32}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.33}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.34}$$

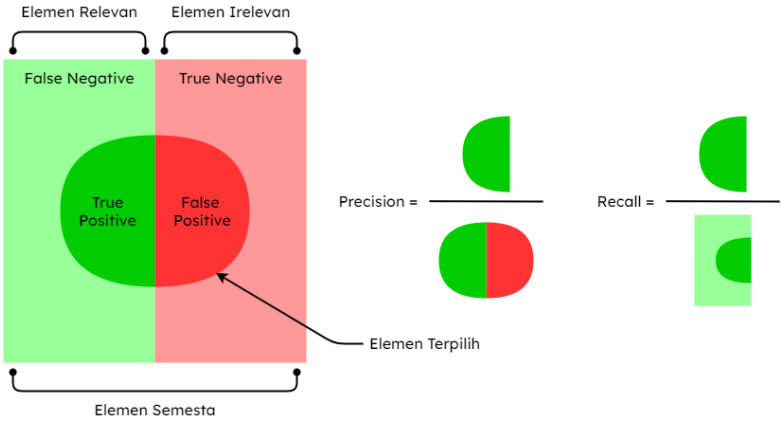
2.8.6 Augmentasi

Untuk mendapatkan performa yang optimal, umumnya *deep learning* seperti *Convolutional Neural Network* membutuhkan jumlah

data yang lebih besar dari jenis algoritma yang lain, karena itu diperlukan proses augmentasi data. Augmentasi data adalah sebuah teknik menambah data dengan cara memanipulasi data yang telah ada dengan pengaturan keragaman tertentu. Untuk data dalam bentuk citra, dapat dilakukan tahapan seperti translasi, refleksi, rotasi, perbesaran ukuran, dan lain-lain.

2.9 Precision dan *Recall*

Dalam pengenalan pola, pengambilan informasi dan klasifikasi (*machine learning*), *precision* adalah fraksi dari *instance* yang relevan di antara *instance* yang diambil, sedangkan *recall* adalah fraksi dari total jumlah *instance* yang relevan yang sebenarnya diambil. Baik *precise* dan *recall* didasarkan pada relevansi. *Precision* dan *Recall* divisualisasikan pada Gambar 2.24 dan dapat dinyatakan seperti pada persamaan (2.33) dan (2.34).



Gambar 2.24. Precision dan Recall

Halaman ini sengaja dikosongkan

BAB 3
PERANCANGAN ARSITEKTUR STEREO R-CNN

Bab ini akan menjelaskan mengenai sistem yang akan dibangun seperti perancangan data *training*, serta arsitektur yang digunakan dalam stereo region based convolutional neural network.

3.1 Data Masukan (*Training Data*)

Data masukan yang digunakan sebagai data *training* algoritma Stereo R-CNN adalah data gambar dari kamera stereo, serta data yang dihasilkan oleh LIDAR. Ketiga data tersebut divisualisasikan pada Gambar 3.1, Gambar 3.2, dan Gambar 3.3.

Gambar 3.1. Sampel data training (kamera kiri)

Gambar 3.2. Sampel data training (kamera kanan)

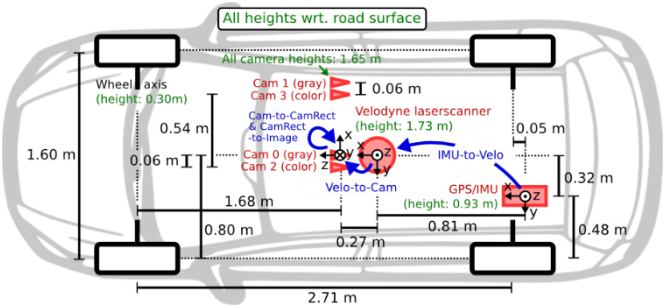
Gambar 3.3. Visualisasi data training yang dihasilkan LIDAR

Data masukan yang digunakan berasal dari dataset KITTI 3D Object Detection Evaluation 2017. *Dataset* ini terdiri dari 7481 data

training dan 7518 adegan tes beserta dengan 14936 data point cloud yang dihasilkan oleh LIDAR, dengan total 80,256 objek berlabel.

Dataset ini diambil menggunakan kendaraan Volkswagen Passat B6 yang telah dimodifikasi seperti pada Gambar 3.4 sehingga dapat menampung sensor yang dibutuhkan. Data direkam menggunakan komputer delapan inti Intel Core-i7 yang dilengkapi dengan sistem RAID, menjalankan Ubuntu Linux, dan database realtime [1]. Sensor yang digunakan untuk pengambilan data ini adalah sebagai berikut:

- **1 buah *Inertial Navigation System* (GPS/IMU):** OXTS RT 3003
- **1 buah Laserscanner (LIDAR):** Velodyne HDL-64E
- **2 buah kamera *grayscale*, 1.4 MP:** FLIR FL2-14S3M-C
- **2 buah kamera berwarna, 1.4 MP:** FLIR FL2-14S3C-C
- **4 buah lensa *Varifocal*, 4-8 mm:** Edmund Optics NT59-917



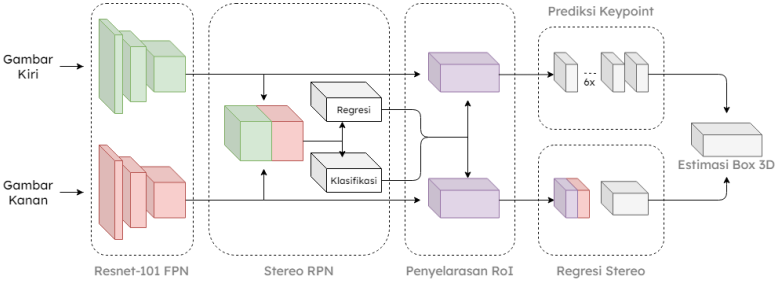
Gambar 3.4. Pengaturan tata letak yang digunakan untuk pengambilan dataset KITTI Stereo 2015 yang digunakan pada penelitian ini.

Pemindai laser berputar pada 10 frame per detik, menangkap sekitar 100rb poin per siklus. Resolusi vertikal LIDAR adalah 64. Kamera dipasang kira-kira sejajar dengan bidang tanah. Gambar kamera dipangkas ke ukuran 1382 x 512 piksel menggunakan mode format 7 libdc. Setelah perbaikan, gambar menjadi sedikit lebih kecil. Kamera dipicu pada 10 *frame* per detik oleh pemindai laser (saat menghadap ke depan) dengan waktu rana yang disesuaikan secara dinamis (waktu rana maksimum: 2 ms).

3.2 Arsitektur Stereo R-CNN

Secara garis besar, arsitektur jaringan Stereo R-CNN yang diusulkan terbagi menjadi beberapa tahap, dimulai dengan tahap deteksi

feature menggunakan *Residual Neural Network Feature Pyramid Network* (ResNet-101 FPN), kemudian dilanjutkan dengan pencarian wilayah yang memiliki fitur-fitur khusus di kedua gambar menggunakan *Stereo Region Proposal Network* (Stereo RPN), lalu dilakukan regresi stereo pada hasil Stereo RPN sehingga dihasilkan peta kedalaman (*depth map*) yang selanjutnya akan diestimasi menjadi *bounding box* tiga dimensi. Visualisasi arsitektur Stereo R-CNN yang diusulkan dapat dilihat pada Gambar 3.5.



Gambar 3.5. Visualisasi arsitektur stereo R-CNN yang diusulkan

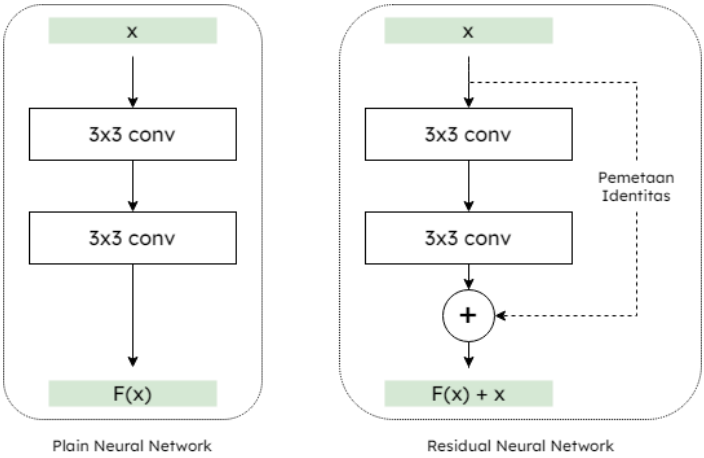
Secara umum, algoritma Stereo R-CNN ini berusaha mempelajari fitur *dense* dan *sparse* dari suatu objek tiga dimensi. Algoritma ini akan mendeteksi dan mengasosiasikan objek tiga dimensi secara simultan.

3.2.1 ResNet-101 FPN

Feature Pyramid Network merupakan suatu metode pembelajaran dimana algoritma mempelajari tiap fitur mulai dari fitur yang sederhana, dan terus meningkat hingga ke fitur yang lebih rumit. Pada deteksi gambar, metode ini dapat diumpamakan mempelajari sudut, tepi, dan bentuk dari suatu objek di gambar. Secara teori, semakin banyak *layer* dari *neural network* yang digunakan, maka semakin banyak fitur yang bisa didapatkan. Maka, bukankah ini berarti kita hanya perlu menambahkan *layer* konvolusi secara terus menerus agar fitur yang didapatkan bisa maksimal? Ternyata, He et al. [8] menemukan bahwa hasil akhir *training* neural network pada suatu titik akan menjadi lebih buruk, bukan lebih baik dari sebelumnya. Masalah ini dapat diselesaikan dengan menambahkan opsi "*Skip Connection*" atau pemetaan identitas pada *neural network*. Perbedaan antara *residual neural network* dengan *neural network* konvensional dapat dilihat pada Gambar 3.6. Koneksi pintas

antar *layer* ini menghasilkan kemampuan untuk melatih jaringan yang lebih dalam dari apa yang sebelumnya mungkin

Pemetaan identitas pada RNN tidak memiliki parameter apa pun dan hanya ada untuk menambahkan output dari lapisan sebelumnya ke lapisan di depan. Namun, terkadang x dan F (x) tidak akan memiliki dimensi yang sama. Perlu diingat bahwa operasi konvolusi biasanya mengecilkan resolusi spasial suatu gambar, misalnya konvolusi 3x3 pada gambar 32 x 32 piksel menghasilkan gambar 30 x 30 piksel.



Gambar 3.6. Perbedaan antara RNN dengan NN konvensional

Karena itu, pemetaan identitas dikalikan dengan nilai proyeksi linear W agar dimensi dari pemetaan identitas sesuai dengan residual dan memungkinkan input x dan F(x) untuk digabungkan sebagai input ke lapisan berikutnya. Secara matematis, fungsi dari residual neural network dapat dituliskan sebagai Persamaan (3.1).

$$y = F(x, \{W_i\}) + W_s x \tag{3.1}$$

Modul *layer Residual Neural Network* secara umum dapat dituliskan sebagai pseudocode berikut :

```
Def Unit(x,filters,pool=false):
    residual = x

```

```
4       # pooling
5       if pool:
6            x = MaxPooling2D(pool_size=(2, 2))(x)
7            res = Conv2D(filters=filters, kernel_size=[1,1], \
8                         strides=(2,2), padding="same")(res)
9       # first layer
10      out = BatchNormalization
11      out = Activation("relu")(out)
12      out = Conv2D( filters=filters, kernel_size=[3, 3], \
13                    strides=[1, 1], padding="same")(out)
14      # second layer
15      out = BatchNormalization()(out)
16      out = Activation("relu")(out)
17      out = Conv2D(filters=filters, kernel_size=[3, 3], \
18                    strides=[1, 1], padding="same")(out)
19      # adding output with residual
20      out = keras.layers.add([res,out])
21      return out
```

Dari *pseudocode* Unit diatas dapat dilihat bahwa nilai input x disimpan pada variabel residual dan ditambahkan ke hasil dari layer *batchnorm-relu-conv* di baris 20 *pseudocode*. Selanjutnya ResNet dapat dibuat dengan menumpuk *layer* Unit tersebut sesuai dengan kebutuhan. Pada penelitian ini digunakan 101 *layer* Unit untuk membentuk *ResNet-101 FPN*.

Pada penelitian ini, ResNet-101 FPN akan digunakan untuk mengekstraksi fitur-fitur *sparse* yang ada pada gambar kiri dan gambar kanan. Setelah melalui ekstraksi fitur di ResNet-101 FPN, gambar akan dikurangi dimensinya menggunakan *layer* konvolusi 3×3.

3.2.2 Stereo RPN

Menurut [10], Region Proposal Network (RPN) adalah detektor objek berbasis *sliding window*. Output dari RPN adalah sekelompok kotak / *proposal* yang selanjutnya akan diperiksa oleh *classifier* dan *regressor* akan adanya objek di dalamnya. Pada penelitian ini, RPN yang digunakan diubah agar dapat menerima output dari FPN dengan mengevaluasi *anchor* pada beberapa skala di *feature map*. Kunci utama dari kemampuan algoritma ini mendeteksi dan mengasosiasikan objek secara simultan adalah penugasan kotak *ground-truth* (GT) yang berbeda untuk pengklasifikasi objek dan regresi kotak stereo.

Pada Gambar 3.7 dapat dilihat bahwa gabungan kotak GT kiri dan kanan (selanjutnya disebut sebagai kotak union GT) ditetapkan sebagai target untuk klasifikasi objek. Sebuah *anchor* diberi label positif jika rasio

Intersection-over-Union (IoU) dengan salah satu kotak union GT di atas 0.7, dan label negatif jika IoU dengan salah satu kotak union di bawah 0.3. Manfaat dari desain ini adalah, *anchor* positif mengandung wilayah objek kiri dan kanan. Selanjutnya, offset *anchor* positif sehubungan dengan kotak GT kiri dan kanan yang terkandung dalam kotak GT union gabungan dihitung, kemudian offset ke regresi kiri dan kanan dapat ditetapkan.

Gambar 3.7. Perbedaan penugasan target untuk regresi dan klasifikasi

Digunakan enam ketentuan yang digunakan pada *stereo regressor* yakni $\Delta u, \Delta u', \Delta v, \Delta v', \Delta w,$ dan Δh dimana u dan v menyatakan koordinat horizontal dan vertikal dari titik tengah kotak 2D di bidang gambar, w dan h menyatakan panjang dan lebar dari kotak 2D, serta tanda petik ($'$) menyatakan gambar sebelah kanan. Perlu diperhatikan bahwa pada penelitian ini digunakan *offset* w dan h yang sama yakni Δw dan Δh karena gambar stereo yang digunakan telah difilter dan diperbaiki. Oleh karena itu pada penelitian ini digunakan enam saluran keluaran untuk regresor RPN stereo, bukan empat di implementasi RPN pada umumnya. Karena proposal kiri dan kanan hasil RPN dihasilkan dari *anchor* yang sama dan berbagi skor objektivitas, mereka dapat dikaitkan satu sama lainnya. Pseudocode dari algoritma Stereo RPN adalah:

```
# define convrelu, etc.
RPN_Conv = nn.Conv2d(depthInput, 512, 3, 1, 1, bias=True)
RPN_score = nn.Conv2d(512*2, self. score_out, 1, 1, 0)
```

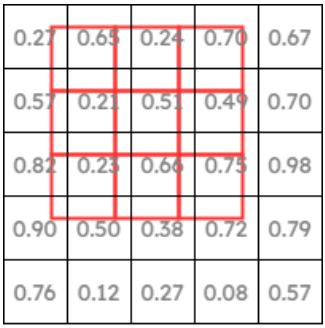
```
RPN_bbox_pred = nn.Conv2d(512*2, self. bbox_out, 1, 1, 0)
self.RPN_proposal = ProposalLayer(self.feat_stride, \
                                  self.anchor_ratios)

# get all feature map
for i in range(feature_maps):
    batch_size = rpn_feature_maps_left[i].size(0)

    # get rpn classification score
    rpn_conv1 = torch.cat((\
        F.relu(self.RPN_Conv(rpn_feature_maps_left[i]), \
            inplace=True), \
        F.relu(self.RPN_Conv(rpn_feature_maps_right[i]), \
            inplace=True)),1)
    rpn_score_reshape = reshape(rpn _score, 2)
    rpn_prob_reshape = softmax(rpn _score_reshape, 1)
    rpn_prob = reshape(rpn_prob_reshape, self.score_out)

    # get rpn offsets to the anchor boxes
    rpn_bbox_pred = RPN_bbox_pred(rpn_conv1)

rpn_score_alls = torch.cat(rpn_scores, 1)
rpn_prob_alls = torch.cat(rpn_probs, 1)
rpn_bbox_pred = torch.cat(rpn_bbox_pred, 1)
```

3.2.3 Penyelarasan RoI / *RoI Alignment*

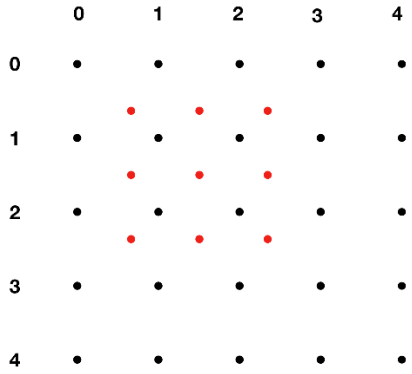
Penyelarasan RoI atau *RoI alignment* merupakan suatu teknik yang didasarkan pada interpolasi bilinear untuk memangkas *patch* dengan mulus dari peta fitur gambar penuh berdasarkan hasil RPN, dan kemudian mengubah ukuran *patch* yang dipangkas menjadi ukuran spasial yang diinginkan. *RoI alignment* pertama kali diperkenalkan pada [11], dan telah terbukti mengungguli alternatifnya, yaitu *RoI Pooling*.

RoI alignment dapat menggunakan peta fitur (misalnya *array* 2D) dan *bounding box* hasil dari RPN sebagai masukannya. Pada penelitian ini, penyelasaran RoI akan menggunakan *bounding box* hasil dari RPN sebagai masukannya. Cara kerja dari penyelarasan RoI ini sendiri dapat divisualisasikan seperti pada Gambar 3.8. Dengan peta fitur 5x5 dan kotak pembatas merah, *RoI* Align menghasilkan peta fitur 3x3 yang dihitung dari nilai yang dibatasi dalam kotak merah. Lalu, bagaimana kita harus berurusan dengan nilai-nilai yang sebagian dibagikan oleh sel-sel di kotak merah, karena tepi hitam tidak selalu selaras dengan tepi merah? Akan lebih mudah apabila sel-sel dalam kotak hitam dan merah diumpamakan sebagai titik dalam *array* 2-D, seperti yang ditunjukkan di

Gambar 3.9: Nilai interpolasi dari titik merah dihitung berdasarkan pada nilai dari empat titik hitam terdekat, yang merupakan masalah interpolasi bilinear.



Gambar 3.8. Visualisasi ROI Align yang memangkas patch yang tepinya tidak sesuai dengan batasan cell



Gambar 3.9. Titik hitam adalah sel di peta fitur, sementara titik merah adalah sel hasil RPN

Seperti yang dapat kita lihat pada Gambar 3.9, sel-sel dalam peta fitur output direpresentasikan sebagai titik-titik yang berjarak sama dalam ruang 2-D (titik merah), jadi pertama-tama kita perlu menghitung koordinatnya. Dan karena koordinat *bounding box* relatif, kita perlu

mengubahnya menjadi koordinat absolut dimana *img_width* dan *img_height* merupakan lebar dan tinggi dari peta fitur, dan *height* dan *width* merupakan ukuran bounding box RPN yang diinginkan.

```
y_coord = linspace(y_min, y_max, height) * (img_height - 1)
x_coord = linspace(x_min, x_max, width) * (img_width - 1)
```

Karena koordinat dari titik merah berupa pecahan, kita dapat mencari koordinat dari keempat titik hitam yang bertetangga dengan titik merah menggunakan *ceil* & *floor*.

```
y_low, y_high = floor(y), ceil(y)
x_low, x_high = floor(x), ceil(x)
```

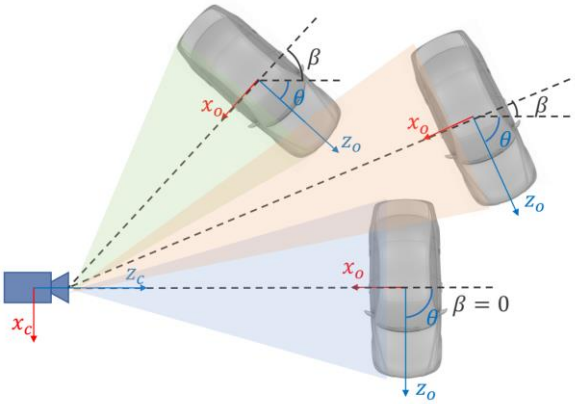
Selanjutnya, kita dapat mencari nilai dari sel RPN menggunakan interpolasi bilinier. Secara keseluruhan, algoritma untuk penyelarasan ROI dapat dituliskan dalam bentuk *pseudocode* sebagai berikut:

```
for y in linspace(y_min, y_max, height)*(img_height - 1):
    for x in linspace(x_min, x_max, width)*(img_width - 1):

      y_l, y_h = floor(y), ceil(y)
      x_l, x_h = floor(x), ceil(x)

      a = image[y_l, x_l]
      b = image[y_l, x_h]
      c = image[y_h, x_l]
      d = image[y_h, x_h]

      y_weight = y - y_l
      x_weight = x - x_l

      val = a * (1 - x_weight) * (1 - y_weight) + \
            b * x_weight * (1 - y_weight) + \
            c * y_weight * (1 - x_weight) + \
            d * x_weight * y_weight

      feature_map.append(val)

feature_map.reshape(height, width)
```

3.2.4 Regresi Stereo

Fitur RoI kiri dan kanan digabungkan dan dimasukkan ke dalam dua lapisan yang terhubung sepenuhnya secara berurutan (masing-masing diikuti oleh lapisan ReLU) untuk mengekstraksi informasi semantik. Pada

penelitian ini, digunakan empat sub-cabang untuk memprediksi kelas objek, kotak pembatas stereo, dimensi, dan sudut pandang masing-masing. Perlu diperhatikan bahwa sudut pandang tidak sama dengan orientasi objek yang tidak dapat diobservasi dari gambar yang dipangkas RoI. Pada Gambar 3.10 diilustrasikan di mana kita menggunakan θ untuk menunjukkan orientasi kendaraan terhadap frame kamera, dan β untuk menunjukkan objek azimuth sehubungan dengan pusat kamera. Tiga kendaraan memiliki orientasi berbeda, namun proyeksi mereka persis sama pada gambar RoI yang dipangkas. Karena itu dilakukan regresi sudut pandang α yang didefinisikan sebagai: $\alpha = \theta + \beta$. Untuk menghindari diskontinuitas, target *training* yang ditentukan adalah pasangan $[sin\,\alpha, cos\,\alpha]$ alih-alih nilai sudut deteksi mentah. Dengan *bounding box* stereo dan dimensi objek, informasi kedalaman dan orientasi kendaraan dapat diselesaikan dengan memisahkan hubungan antara sudut pandang dengan posisi 3D dari setiap objek.



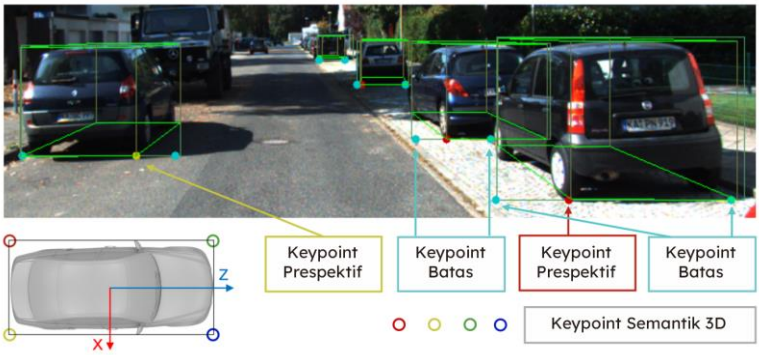
Gambar 3.10. Relasi antara orientasi θ, azimuth β, dan viewpoint $(\theta + \beta)$.

Saat mengambil sampel RoI, pasangan RoI kiri-kanan dianggap sebagai latar depan jika IoU maksimum antara RoI kiri dengan kotak GT kiri, serta IoU antara RoI kanan dengan kotak GT yang sesuai lebih tinggi dari 0,5. Pasangan RoI kiri-kanan dianggap sebagai latar belakang jika IoU maksimum untuk RoI kiri atau RoI kanan terletak pada interval 0.1

hingga 0.5. Untuk pasangan RoI latar depan, target regresi ditetapkan dengan menghitung *offset* antara RoI kiri dengan kotak GT kiri, dan *offset* antara RoI kanan dengan kotak GT kanan yang sesuai. Dimana nilai Δv dan Δh yang digunakan untuk RoI kiri dan kanan bernilai sama.

3.2.5 Prediksi *Keypoint*

Menurut [7], selain *bounding box* stereo dan sudut pandang, sudut-sudut kotak 3D yang diproyeksikan di tengah kotak dapat memberikan batasan yang lebih ketat untuk estimasi kotak 3D. Seperti yang ditunjukkan Gambar 3.11, empat *keypoint* semantik 3D didefinisikan untuk menunjukkan empat sudut di bagian bawah *bounding box* 3D. Hanya ada satu *keypoint* semantik 3D yang dapat diproyeksikan ke kotak tengah (alih-alih tepi kiri atau kanan). Proyeksi *keypoint* semantik ini didefinisikan sebagai *keypoint* perspektif.



Gambar 3.11. Ilustrasi keypoint semantik 3D, keypoint perspektif 2D, dan keypoint batas.

Keypoint pada algoritma ini diprediksi seperti pada metode Mask R-CNN [12]. Perlu diperhatikan bahwa pada algoritma ini, hanya peta fitur kiri yang digunakan untuk prediksi keypoint. Selanjutnya, keluaran dari peta fitur 14×14 RoI dijadikan sebgai masukan ke enam lapisan konvolusi 3×3 secara berurutan seperti yang diilustrasikan pada Gambar 3.5, masing-masing diikuti oleh lapisan fungsi aktivasi *Rectified Linear Unit* (ReLU). Langkah terakhir, lapisan dekonvolusi 2×2 digunakan untuk mengubah dimensi keluaran menjadi 28×28. Karena informasi tambahan hanya diberikan oleh koordinat u dari *keypoint*,

Untuk menghemat komputasi, *channel* ketinggian dalam output $6 \times 28 \times 28$ dijumlahkan untuk menghasilkan prediksi 6×28. Akibatnya, setiap kolom dalam fitur RoI akan dikumpulkan dan berkontribusi pada prediksi *keypoint*, algoritma ini dapat dituliskan dalam bentuk *pseudocode* sebagai berikut:

```
# 6 * cfg.KPTS_GRID * cfg.KPTS_GRID
kpts_pred = kpts_class(roi_feat_dense)
# 6 * cfg.KPTS_GRID
kpts_pred_all = kpts_pred.sum(height)
```

Dari enam *channel* yang dihasilkan pada prediksi keypoint, empat *channel* pertama mewakili probabilitas dari empat titik kunci semantik yang diproyeksikan ke lokasi u yang sesuai sementara dua *channel* lainnya mewakili probabilitas masing-masing u terletak di batas kiri dan kanan masing-masing. Perhatikan bahwa hanya satu dari empat keypoint 3D yang terletak di tengah kotak 2D, sehingga fungsi aktivasi *softmax* diterapkan pada output 4×28 untuk mendorong *keypoint* semantik eksklusif diproyeksikan ke satu lokasi. Strategi ini menghindari kemungkinan kebingungan jenis perspektif keypoint (sesuai dengan setiap *keypoint* semantik). Untuk titik kunci batas kiri dan kanan, fungsi aktivasi *softmax* juga diterapkan pada output 1×28 di masing-masing *keypoint*.

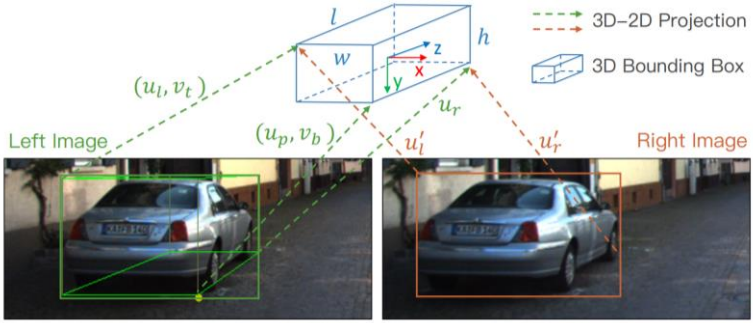
```
kpts_pred = kpts_pred_all[:3].view(-1, 4*cfg.KPTS_GRID)
kpts_prob = softmax(kpts_pred,1)
# kpts_prob.size = 4 * cfg.KPTS_GRID

left_border_pred = kpts_pred_all[4].view(-1, cfg.KPTS_GRID)
left_border_prob = softmax(left_border_pred,1)
# left_border_prob.size = 1 * cfg.KPTS_GRID

right_border_pred = kpts_pred_all[5].view(-1, cfg.KPTS_GRID)
right_border_prob = softmax(left_border_pred,1)
# left_border_prob.size = 1 * cfg.KPTS_GRID
```

Selama proses training, *cross-entropy loss* pada hasil keluaran *softmax* 4×28 untuk prediksi keypoint perspektif diminimalkan. Hanya satu lokasi dalam keluaran 4×28 yang dilabeli sebagai target *keypoint* perspektif. Pada metode ini, kasus di mana tidak ada titik kunci semantik 3D yang diproyeksikan secara nyata di tengah kotak (mis., Kasus proyeksi pemotongan dan ortografis) diabaikan. Pada dua keypoint batas, *cross-entropy loss* diminimalkan secara independen. Setiap RoI

foreground akan dipasangkan dengan keypoint batas kiri dan kanan sesuai dengan hubungan oklusi antara kotak GT.

3.2.6 Estimasi *bounding box* tiga dimensi

Pada segmen estimasi *bounding box* 3D ini, informasi *sparse keypoint* dari segmen prediksi *keypoint*, serta informasi *bounding box* 2D yang telah dicari sebelumnya digunakan untuk mengestimasi secara kasar *bounding box* 3D. *State* dari *bounding box* 3D dapat direpresentasikan dengan persamaan $\bar{x} = \{x, y, z, \theta\}$, yang menunjukkan posisi pusat dan orientasi horizontal masing-masing *bounding box* 3D. Dengan *bounding box* 2D kiri dan kanan, sudut pandang perspektif, dan dimensi yang diregresikan, kotak 3D dapat dicari dengan meminimalkan *error* proyeksi ulang kotak 2D dan *keypoint*.



Gambar 3.12. Estimasi sparse bounding box 3D

Seperti yang divisualisasikan pada Gambar 3.12, tujuh pengukuran dari *bounding box* stereo dan *keypoint* prespektif diekstrak. Ketujuh hasil pengukuran tersebut adalah u_l, v_t, u_r dan v_b yang menyatakan batas kiri, atas, kanan dan bawah dari *bounding box* 2D sebelah kiri, u_l' dan u_r' yang menyatakan batas kiri dan kanan dari *bounding box* 2D sebelah kanan, serta u_p yang menyatakan *keypoint* prespektif. Setiap pengukuran dinormalisasi oleh kamera intrinsik untuk menyederhanakan representasi. Korespondensi antara sudut *bounding box* 2D dan 3D dapat ditemukan menggunakan *keypoint* prespektif. Mengikuti apa yang diterapkan pada penelitian [13], hubungan 3D-2D dengan transformasi proyeksi dapat dirumuskan seperti pada Persamaan (3.2) hingga (3.8) dimana b menyatakan *baseline* dari kamera stereo dan

$w, h,$ dan l menyatakan dimensi regresi. Terdapat tujuh persamaan yang sesuai dengan tujuh pengukuran dimana signum dari $\frac{w}{2}$ dan $\frac{l}{2}$ harus diubah agar sesuai berdasarkan sudut kotak 3D yang sesuai.

$$v_t = \frac{y - \frac{h}{2}}{z - \frac{w}{2}sin\theta - \frac{l}{2}cos\theta} \tag{3.2}$$

$$v_b = \frac{y + \frac{h}{2}}{z + \frac{w}{2}sin\theta + \frac{l}{2}cos\theta} \tag{3.3}$$

$$u_l = \frac{x - \frac{w}{2}cos\theta - \frac{l}{2}sin\theta}{z + \frac{w}{2}sin\theta - \frac{l}{2}cos\theta} \tag{3.4}$$

$$u_r = \frac{x + \frac{w}{2}cos\theta + \frac{l}{2}sin\theta}{z - \frac{w}{2}sin\theta + \frac{l}{2}cos\theta} \tag{3.5}$$

$$u_p = \frac{x + \frac{w}{2}cos\theta - \frac{l}{2}sin\theta}{z - \frac{w}{2}sin\theta - \frac{l}{2}cos\theta} \tag{3.6}$$

$$u_l' = \frac{x + b - \frac{w}{2}cos\theta - \frac{l}{2}sin\theta}{z + \frac{w}{2}sin\theta - \frac{l}{2}cos\theta} \tag{3.7}$$

$$u_r' = \frac{x - b + \frac{w}{2}cos\theta + \frac{l}{2}sin\theta}{z - \frac{w}{2}sin\theta + \frac{l}{2}cos\theta} \tag{3.8}$$

Tidak seperti penelitian [13] yang menggunakan satu buah *bounding box* 2D untuk mencari orientasi dan posisi *bounding box* 3D, pada penelitian ini *bounding box* 3D dicari berdasarkan informasi

kedalaman dari langkah Regresi Stereo sehingga hasilnya lebih akurat. Dalam beberapa kasus di mana kurang dari dua permukaan samping dapat sepenuhnya diamati dan tidak ada *keypoint* prespektif u_p, orientasi dan dimensi tidak dapat diamati dari batasan geometri murni sehingga digunakan sudut pandang α untuk mengkompensasi kondisi yang tidak dapat diobservasi.

Halaman ini sengaja dikosongkan

BAB 4
PENGAMBILAN DATA
DAN PENGUJIAN SISTEM

Pada bab ini akan membahas tentang pengujian algoritma Stereo R-CNN yang telah dirancang pada bab sebelumnya, serta pada bab ini juga akan dilakukan analisis data dari setiap pengujian yang telah dilakukan. Semua data yang digunakan untuk pengujian diambil secara langsung dari dataset KITTI [1].

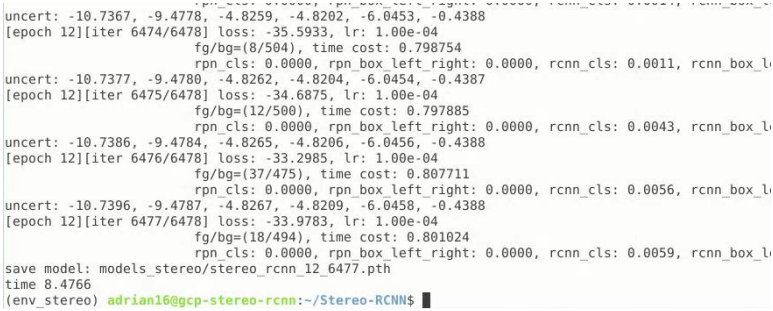
4.1 Training Neural Network

Training Neural Network dilakukan menggunakan Google Cloud Computing Platform dengan hardware komputasi seperti pada Tabel 4.1. Training neural network dilakukan dengan input dataset sejumlah 7481 untuk masing-masing gambar kamera kiri, gambar kamera kanan, file kalibrasi, label, dan data lidar.

Dari 7481 data dari dataset, dilakukan filter pada dataset yang tidak memiliki objek deteksi sehingga hanya 6477 data yang digunakan untuk melakukan training. Training dilakukan sebanyak 12 epoch dengan kecepatan komputasi training sebesar 0.8 detik untuk setiap datanya. Lama waktu training dari algoritma ini menggunakan hardware seperti pada Tabel 4.1 adalah 84766 detik atau 23 jam 32 menit 46 detik.

Tabel 4.1. Hardware yang digunakan pada proses training

*v*CPU	8 *core* 16 *thread allocation from* Intel Xeon
GPU	Nvidia Tesla V100 – 16GB VRAM
RAM	30 GB
Disk	300 GB SSD Network Storage
OS	Ubuntu 18.04 LTS untuk Google Cloud Platform



Gambar 4.1. Hasil tangkapan layar proses training

Hasil dari proses training adalah bobot dari neural network yang disimpan ke dalam file di folder ./models_stereo dengan nama stereo_rcnn_{jumlahEpoch}_{jumlahData}.pth yang dalam kasus ini bernama stereo_rcnn_12_6477.pth seperti pada Gambar 4.1.

4.2 Testing Neural Network

Hasil dari training neural network di tes menggunakan data testing yang terdapat pada dataset KITTI. Selanjutnya dilakukan plot nilai *Precision* dan *Recall* dari hasil tes tersebut. Kinerja deteksi objek 3D dievaluasi menggunakan kriteria PASCAL. Tingkat kesulitan deteksi dibagi menjadi tiga yakni:

a. *Easy* / Mudah
Tingkat kesulitan mudah didefinisikan seperti pada Tabel 4.2.

Tabel 4.2. Kriteria tingkat kesulitan (mudah)

tinggi minimal bounding box	40 piksel
Tingkat oklusi	Terlihat sepenuhnya
Maksimal terpotong	15%

b. *Moderate* / normal
Tingkat kesulitan normal didefinisikan seperti pada Tabel 4.3.

Tabel 4.3. Kriteria tingkat kesulitan (normal)

tinggi minimal bounding box	25 piksel
Tingkat oklusi	Tertutup Sebagian
Maksimal terpotong	30%

c. *Hard* / Sulit
Tingkat kesulitan sulit didefinisikan seperti pada Tabel 4.4

Tabel 4.4. Kriteria tingkat kesulitan (sulit)

tinggi minimal bounding box	25 piksel
Tingkat oklusi	Sulit Terlihat
Maksimal terpotong	50%

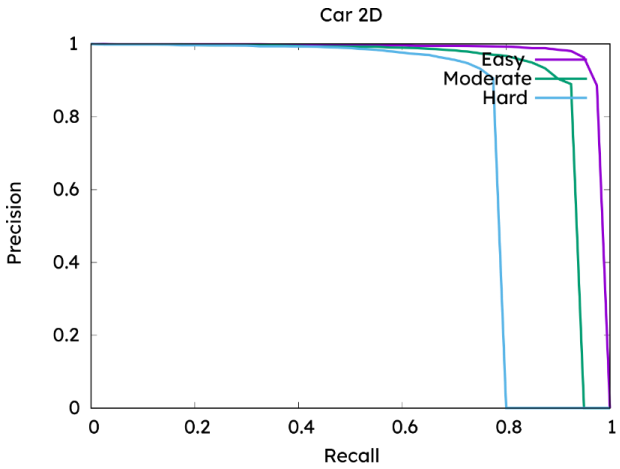
Data *precision-recall* dari hasil *testing* deteksi bounding box 2D mobil menunjukkan bahwa pada objek dengan tipe "Easy" memiliki hasil deteksi dengan tingkat recall 97,5% dengan presisi 88% dan tipe "Hard" memiliki tingkat recall 77,5% dan presisi 90%, dapat dilihat pada Tabel 4.5 dengan hasil plotnya divisualisasikan pada Gambar 4.2.

Tabel 4.5. Data precision-recall deteksi bounding box 2D

Recall	***Precision Easy***	***Precision Moderate***	***Precision Hard***
0	1	1	1
0,025	1	0,99855	0,998547
0,05	0,998948	0,99855	0,998547
0,075	0,998948	0,99855	0,998547

Recall	*Precision Easy*	*Precision Moderate*	*Precision Hard*
0,1	0,998948	0,99855	0,998547
0,125	0,998948	0,99855	0,998547
0,15	0,998948	0,99855	0,9982
0,175	0,998948	0,99855	0,996883
0,2	0,998948	0,997468	0,996778
0,225	0,998948	0,996652	0,996778
0,25	0,998948	0,996339	0,996378
0,275	0,998948	0,996339	0,995721
0,3	0,998948	0,996123	0,995471
0,325	0,998948	0,996123	0,99372
0,35	0,998064	0,996033	0,99372
0,375	0,9973	0,994961	0,99372
0,4	0,99714	0,99432	0,993211
0,425	0,99714	0,993092	0,992959
0,45	0,99714	0,993092	0,99135
0,475	0,99714	0,993092	0,990483
0,5	0,99714	0,992555	0,988634
0,525	0,99714	0,992555	0,985947
0,55	0,99714	0,992442	0,98399
0,575	0,99714	0,990589	0,980389
0,6	0,99714	0,989939	0,976088
0,625	0,995071	0,988557	0,972712
0,65	0,994629	0,987076	0,969887
0,675	0,994629	0,984806	0,962876
0,7	0,994629	0,982347	0,956457
0,725	0,994348	0,979231	0,947488
0,75	0,994069	0,974088	0,93213

Recall	*Precision Easy*	*Precision Moderate*	*Precision Hard*
0,775	0,993383	0,970555	0,905606
0,8	0,992741	0,967271	0
0,825	0,991319	0,958087	0
0,85	0,988391	0,949064	0
0,875	0,98834	0,933053	0
0,9	0,984193	0,90447	0
0,925	0,980657	0,890022	0
0,95	0,962679	0	0
0,975	0,885723	0	0
1	0	0	0



Gambar 4.2. Plot precision-recall deteksi bounding box 2D

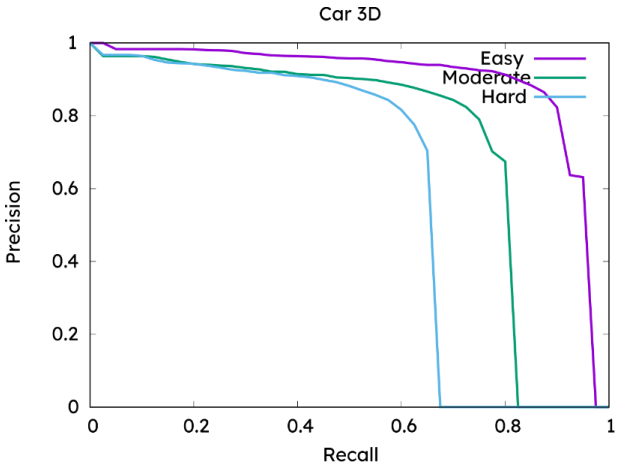
Data *precision-recall* dari hasil *testing* deteksi bounding box 3D mobil menunjukkan bahwa pada objek dengan tipe "Easy" memiliki hasil deteksi dengan tingkat recall 95% dengan presisi 63% dan tipe "Hard" memiliki tingkat recall 65% dan presisi 70%, dapat dilihat pada Tabel 4.6 dengan hasil plotnya divisualisasikan pada Gambar 4.3.

Tabel 4.6. Data precision-recall deteksi bounding box 3D

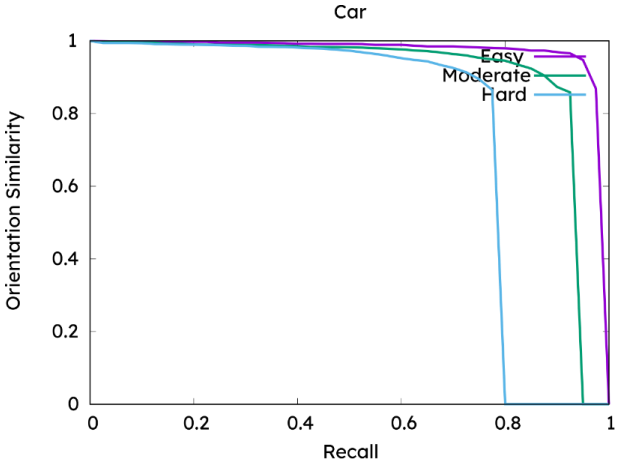
Recall	***Precision Easy***	***Precision Moderate***	***Precision Hard***
0	1	1	1
0,025	1	0,964169	0,96729
0,05	0,983051	0,964169	0,96729
0,075	0,983051	0,964169	0,96729
0,1	0,983051	0,963855	0,964564
0,125	0,983051	0,961575	0,953393
0,15	0,983051	0,954473	0,946347
0,175	0,983051	0,947945	0,944282
0,2	0,982085	0,942433	0,942539
0,225	0,980769	0,940615	0,937761
0,25	0,98	0,938154	0,932607
0,275	0,977995	0,936097	0,926245
0,3	0,972222	0,931265	0,923379
0,325	0,969789	0,927693	0,918519
0,35	0,965812	0,921307	0,918519
0,375	0,964664	0,920823	0,911471
0,4	0,964315	0,914344	0,909091
0,425	0,962646	0,912582	0,905503
0,45	0,961905	0,912439	0,898796
0,475	0,958564	0,905615	0,892337
0,5	0,957994	0,903366	0,881823
0,525	0,957994	0,901003	0,86948
0,55	0,9546	0,898133	0,857651
0,575	0,949943	0,891471	0,843198
0,6	0,947368	0,88543	0,816375

Recall	*Precision Easy*	*Precision Moderate*	*Precision Hard*
0,625	0,942946	0,876849	0,775524
0,65	0,939831	0,866509	0,703789
0,675	0,939655	0,855532	0
0,7	0,933915	0,843362	0
0,725	0,929894	0,823827	0
0,75	0,924873	0,789882	0
0,775	0,922541	0,702211	0
0,8	0,912873	0,674396	0
0,825	0,898426	0	0
0,85	0,883679	0	0
0,875	0,864626	0	0
0,9	0,823104	0	0
0,925	0,637032	0	0
0,95	0,631579	0	0
0,975	0	0	0
1	0	0	0

Data *precision-recall* dari hasil *testing* deteksi orientasi mobil menunjukkan bahwa pada objek dengan tipe “Easy” memiliki hasil deteksi dengan tingkat recall 97,5% dengan presisi 86% dan tipe “Hard” memiliki tingkat recall 77,5% dan presisi 86%, dapat dilihat pada Tabel 4.7 dengan hasil plotnya divisualisasikan pada Gambar 4.4.



Gambar 4.3. Plot precision-recall deteksi bounding box 3D



Gambar 4.4. Plot precision-recall deteksi orientasi mobil

Tabel 4.7. Tabel precision-recall deteksi orientasi mobil

Recall	***Precision Easy***	***Precision Moderate***	***Precision Hard***
0	0,999959	0,999615	0,999999
0,025	0,998964	0,995658	0,994262
0,05	0,99765	0,995658	0,994262
0,075	0,99765	0,995658	0,994262
0,1	0,99765	0,995287	0,993737
0,125	0,99765	0,995058	0,991424
0,15	0,99765	0,992119	0,990958
0,175	0,99765	0,992119	0,989926
0,2	0,99765	0,990467	0,98977
0,225	0,99765	0,989751	0,988677
0,25	0,995004	0,989152	0,988252
0,275	0,995004	0,989046	0,987014
0,3	0,995004	0,988599	0,986629
0,325	0,995004	0,988599	0,98396
0,35	0,994309	0,987603	0,983744
0,375	0,993731	0,986251	0,983076
0,4	0,991922	0,985384	0,981531
0,425	0,991922	0,983279	0,980125
0,45	0,991561	0,983238	0,978388
0,475	0,991561	0,983007	0,975675
0,5	0,991561	0,982038	0,97318
0,525	0,991193	0,981619	0,968311
0,55	0,989221	0,980061	0,964198
0,575	0,989166	0,97806	0,958766
0,6	0,989166	0,976645	0,952357

Recall	*Precision Easy*	*Precision Moderate*	*Precision Hard*
0,625	0,986861	0,973887	0,947487
0,65	0,984723	0,971942	0,943654
0,675	0,984723	0,968047	0,934009
0,7	0,984723	0,964058	0,925503
0,725	0,983345	0,959949	0,913768
0,75	0,98192	0,953375	0,89455
0,775	0,98072	0,949163	0,864972
0,8	0,97959	0,945398	0
0,825	0,977161	0,934491	0
0,85	0,973758	0,923895	0
0,875	0,973706	0,904861	0
0,9	0,968956	0,873514	0
0,925	0,965832	0,858471	0
0,95	0,947389	0	0
0,975	0,86831	0	0
1	0	0	0

4.3 Data Demonstrasi Algoritma Stereo R-CNN

Pada demonstrasi algoritma Stereo R-CNN ini akan ditampilkan beberapa data kualitatif dari 437 data potongan video yang dihasilkan. Pada Gambar 4.5 dapat dilihat bahwa algoritma dapat mendeteksi posisi dan orientasi mobil dengan cukup akurat. Algoritma ini juga mampu mendeteksi mobil kedua yang terletak di luar rentang deteksi lidar. Pada Gambar 4.5 juga terlihat batasan dari algoritma ini, yakni algoritma ini hanya dapat mendeteksi objek yang sudah di *training* sebelumnya sehingga pengendara sepeda pada Gambar 4.5 tidak terdeteksi.

Gambar 4.5. Demonstrasi Stereo R-CNN. (atas: bounding box 3D; bawah : klasifikasi & bounding box 2D; kanan: bird eye's view)

Pada Gambar 4.6 ditampilkan hasil yang kurang baik. Hal ini disebabkan karena prediksi orientasi objek yang tidak sesuai akibat mobil yang ada pada gambar terpotong sebagian. Sementara itu, pada Gambar 4.7 ditampilkan trem yang tidak terdeteksi oleh algoritma Stereo R-CNN ini karena objek trem tidak termasuk dalam data training.

Gambar 4.6. Demonstrasi Stereo R-CNN dengan gambar mobil yang terpotong sebagian.

Gambar 4.7. Demonstrasi Stereo R-CNN dengan beberapa kendaraan tidak terdeteksi.

4.4 Kecepatan Deteksi Algoritma Stereo R-CNN

Proses deteksi dilakukan dengan hardware yang berbeda dengan proses training dan testing data. Pada proses deteksi, digunakan komputer konvensional dengan spesifikasi hardware seperti pada Tabel 4.8 .

Tabel 4.8. Hardware yang digunakan pada proses deteksi

CPU	6 core 12 thread Ryzen 5 3600
GPU	Nvidia GTX 1660Ti – 6GB VRAM
RAM	16 GB
Disk	128 GB SSD NVMe
OS	Elementary OS 5.1.2 Hera

Dilakukan uji coba deteksi objek dengan jumlah objek dalam satu frame yang bervariasi antara nol hingga empat objek. Uji coba ini dilakukan menggunakan 437 data gambar yang berasal dari data demonstrasi algoritma stereo r-cnn.

Dari data pada Tabel 4.9 terlihat bahwa lama dari waktu pemrosesan tidak berkorelasi dengan jumlah objek yang dideteksi untuk objek berjumlah nol hingga empat pada satu gambar. Namun, tidak menutup kemungkinan bahwa waktu deteksi terpengaruh jumlah objek

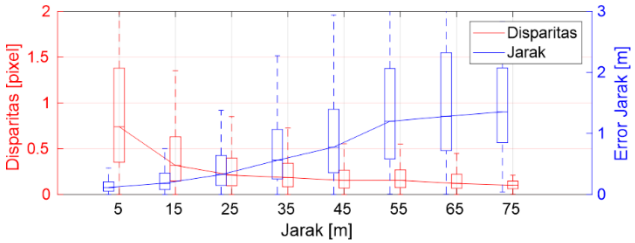
yang terdeteksi karena ujicoba ini hanya dilakukan pada 437 data dengan jumlah objek maksimal pada satu gambar berjumlah empat.

Tabel 4.9. Hubungan antara jumlah objek pada gambar dengan lama waktu pemrosesan

Jumlah objek pada gambar	waktu pemrosesan (detik)		
	Rata-Rata	Terlama	Tercepat
0	0,808243	0,883845	0.723242
1	0,810192	0,893267	0,732364
2	0,809334	0,882736	0,722135
3	0,810075	0,882453	0,731351
4	0,816123	0,903240	0,723710
5	0,814956	0,893234	0,730354

4.5 Pengaruh Jarak dan Ukuran Objek Terhadap Akurasi

Jarak dan ukuran objek (piksel) sangat berpengaruh terhadap akurasi dari algoritma deteksi stereo r-cnn ini. Akurasi deteksi 3D cenderung menurun dengan meningkatnya jarak objek terhadap kamera. Fenomena ini divisualisasikan pada Gambar 4.8 . Error jarak menjadi lebih besar ketika jarak objek meningkat karena hubungan proporsional terbalik antara disparitas dan kedalaman gambar di mana pada deteksi bounding box dengan objek berjarak 5 meter, diperoleh RMSE deteksi sebesar 0.073 meter (1.46%), sementara pada jarak 75 meter, diperoleh RMSE deteksi sebesar 1.278 meter (1.70%).



Gambar 4.8. Relasi antara disparitas dengan error jarak terhadap jarak objek yang dideteksi.

Halaman ini sengaja dikosongkan

BAB 5
PENUTUP

Dari perancangan dan analisa yang telah dilakukan, dapat dirangkum dalam sebuah kesimpulan. Untuk kekurangan dan kendala yang dihadapi, dituliskan dalam bagian saran, untuk membantu penelitian selanjutnya.

5.1 Kesimpulan

Setelah melakukan serangkaian ujicoba algoritma Stereo R-CNN, dapat disimpulkan bahwa algoritma dapat melakukan deteksi *bounding box* 3D dan prediksi jarak objek (mobil) pada gambar stereo yakni :

a. Hasil deteksi *bounding box* masih memiliki error yang berkorelasi dengan jarak dan ukuran objek. Hal ini ditunjukkan dengan error yang semakin meningkat dimana pada deteksi objek berjarak 5 meter, diperoleh RMSE deteksi sebesar 0.073 meter (1.46%), sementara pada deteksi objek berjarak 75 meter, diperoleh RMSE deteksi sebesar 1.278 meter (1.70%).
b. Pada frame tertentu terdapat kesalahan deteksi *false positive*, terutama pada objek dengan ukuran lebih kecil dari 25 piksel dan objek yang tertutupi objek lain atau terpotong lebih dari 50%.
c. Kecepatan deteksi algoritma ini berada pada kisaran 0.81 detik dengan deviasi sebesar 0.1 detik. Kecepatan deteksi ini bergantung pada spesifikasi hardware / komputer yang digunakan.

5.2 Saran

Untuk pengembangan selanjutnya pada topik penelitian deteksi halangan menggunakan kamera stereo dengan algoritma Stereo R-CNN, terdapat beberapa saran yang diberikan, antara lain :

a. Kecepatan deteksi
   Saat ini, dengan kecepatan deteksi sebesar 0.81 detik (1,2 frame per detik) algoritma Stereo R-CNN yang digunakan masih belum *feasible* untuk diterapkan sebagai detektor obstacle utama pada kendaraan otonom.

b. Penambahan jenis objek yang dideteksi
   Saat ini, algoritma hanya dilatih untuk mendeteksi mobil, sehingga masih belum dapat mendeteksi objek lain seperti pedestrian, pesepeda, mobil *minibus*, truk, trailer, dan sebagainya.

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

[1] Karlsruhe Institute of Technology, Toyota Technological Institute at Chicago, "The KITTI Vision Benchmark Suite," Karlsruhe Institute of Technology, Toyota Technological Institute at Chicago, [Online]. Available: http://www.cvlibs.net/datasets/kitti/setup.php. [Diakses 02 06 2020].

[2] B. Cyganek dan J. P. Siebert, An Introduction to 3D Computer Vision Techniques and Algorithms 1st Edition, Wiley, 2009.

[3] V. M. Jaya, "Aplikasi Interpolasi Bilinier pada Pengolahan Citra Digital," dalam *Makalah IF2123 Aljabar Geometri ITB*, Bandung, 2016.

[4] L. Fausett, Fundementals of Neural Network Architectures, Algorithm and Applications, Prentice Hall, 1994.

[5] S. Haykin, Neural Network : A Comprehensive Foundation 2nd edition, Prentice Hall, 1931.

[6] M. L. Minsky dan S. A. Papert, Perceptrons, The MIT Press, 1988.

[7] P. Li, X. Chen dan S. Shen, "Stereo R-CNN Based 3D Object Detection for Autonomous Driving," *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7636-7644, 2019.

[8] K. He, X. Zhang, S. Ren dan J. Sun, "Deep Residual Learning for Image Recognition," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770-778, 2016.

[9] Y. LeCun, L. Bottou, Y. Bengio dan P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.

[10] S. Ren, K. He, R. Girshick dan J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137 - 1149, 2017.

[11] X. Chen, K. Kundu, H. M. Z. Zhang, S. Fidler dan R. Urtasun, "Monocular 3d object detection for autonomous driving," *European Conference on Computer Vision*, pp. 2147-2156, 2016.

[12] K. He, G. Gkioxari, P. Dollar dan R. Girshick, “Mask r-cnn,” *IEEE International Conference on Computer Vision (ICCV),* p. 2980–2988, 2017.

[13] P. Li, T. Qin dan S. Shen, “Stereo vision-based semantic 3d object and ego-motion tracking for autonomous driving,” *European Conference on Computer Vision,* p. 664–679, 2018.

LAMPIRAN

Python Setup : requirements.txt

```
cython
cffi
opencv-python
scipy
easydict
pillow
scipy
```

Program : test.sh

```
CUDA_VISIBLE_DEVICES=0 python test_net.py \
--checkepoch 12 \
--checkpoint 6477
```

Program : test_net.py

```
# --------------------------------------------------------
# Tensorflow Faster CNN
# Licensed under The MIT License [see LICENSE for details]
# Written by Jiasen Lu, Jianwei Yang, based on code from Ross
Girshick
# Modified by Adrian A for Stereo R-CNN
# --------------------------------------------------------
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import _init_paths
import os
import sys
import numpy as np
import argparse
import shutil
import time
import cv2
import torch
from torch.autograd import Variable
import torch.nn as nn
import torch.optim as optim
import math as m
from roi_data_layer.roidb import combined_roidb
```

```
from roi_data_layer.roibatchLoader import roibatchLoader
from model.utils.config import cfg
from model.rpn.bbox_transform import clip_boxes
from model.roi_layers import nms
from model.rpn.bbox_transform import bbox_transform_inv, \
  kpts_transform_inv, border_transform_inv
from model.utils.net_utils import \
  save_net, load_net, vis_detections
from model.stereo_rcnn.resnet import resnet
from model.utils import kitti_utils
from model.utils import vis_3d_utils as vis_utils
from model.utils import box_estimator as box_estimator
from model.dense_align import dense_align

try:
  xrange          # Python 2
except NameError:
  xrange = range  # Python 3

def parse_args():
  """
  Parse input arguments
  """
  parser   =   argparse.ArgumentParser(description='Test   the
Stereo R-CNN network')

  parser.add_argument('--load_dir', dest='load_dir',
    help='directory to load models', default="models_stereo",
    type=str)
  parser.add_argument('--checkepoch', dest='checkepoch',
    help='checkepoch to load network',
    default=12, type=int)
  parser.add_argument('--checkpoint', dest='checkpoint',
    help='checkpoint to load network',
    default=6477, type=int)

  args = parser.parse_args()
  return args

if __name__ == '__main__':

  args = parse_args()

  np.random.seed(cfg.RNG_SEED)

  cfg.TRAIN.USE_FLIPPED = False
  imdb, roidb, ratio_list, ratio_index = combined_roidb(\
```

```
    'kitti_val', False)
  print('{:d} roidb entries'.format(len(roidb)))

  input_dir = args.load_dir + "/"
  if not os.path.exists(input_dir):
    raise Exception(\
      'There is no input directory for loading network from
'\
      + input_dir)
  load_name = os.path.join(input_dir,
    'stereo_rcnn_{}_{}.pth'.format(args.checkepoch,\
      args.checkpoint))

  result_dir = args.load_dir + '/result/'
  if os.path.exists(result_dir):
    shutil.rmtree(result_dir)

  # initilize the network here.
  stereoRCNN = resnet(imdb.classes, 101, pretrained=False)
  stereoRCNN.create_architecture()

  print("load checkpoint %s" % (load_name))
  checkpoint = torch.load(load_name)
  stereoRCNN.load_state_dict(checkpoint['model'])
  print('load model successfully!')

  with torch.no_grad():
    # initilize the tensor holder here.
    im_left_data = Variable(torch.FloatTensor(1).cuda())
    im_right_data = Variable(torch.FloatTensor(1).cuda())
    im_info = Variable(torch.FloatTensor(1).cuda())
    num_boxes = Variable(torch.LongTensor(1).cuda())
    gt_boxes_left = Variable(torch.FloatTensor(1).cuda())
    gt_boxes_right = Variable(torch.FloatTensor(1).cuda())
    gt_boxes_merge = Variable(torch.FloatTensor(1).cuda())
    gt_dim_orien = Variable(torch.FloatTensor(1).cuda())
    gt_kpts = Variable(torch.FloatTensor(1).cuda())

    stereoRCNN.cuda()

    eval_thresh = 0.05
    vis_thresh = 0.7

    num_images = len(imdb.image_index)

    dataset = roibatchLoader(roidb, ratio_list, \
      ratio_index, 1, imdb.num_classes, \
```

```
    training=False, normalize = False)
  dataloader = torch.utils.data.DataLoader(\
    dataset, batch_size=1, shuffle=False, \
    num_workers=0,  pin_memory=True)

  data_iter = iter(dataloader)

  stereoRCNN.eval()
  for i in range(num_images):
    data = next(data_iter)
    im_left_data.data.resize_(data[0].size())\
    .copy_(data[0])
    im_right_data.data.resize_(data[1].size())\
    .copy_(data[1])
    im_info.data.resize_(data[2].size())\
    .copy_(data[2])
    gt_boxes_left.data.resize_(data[3].size())\
    .copy_(data[3])
    gt_boxes_right.data.resize_(data[4].size())\
    .copy_(data[4])
    gt_boxes_merge.data.resize_(data[5].size())\
    .copy_(data[5])
    gt_dim_orien.data.resize_(data[6].size())\
    .copy_(data[6])
    gt_kpts.data.resize_(data[7].size())\
    .copy_(data[7])
    num_boxes.data.resize_(data[8].size())\
    .copy_(data[8])

    det_tic = time.time()
    rois_left, rois_right, cls_prob, bbox_pred,\
    bbox_pred_dim, kpts_prob, left_prob, right_prob,\
    rpn_loss_cls, rpn_loss_box_left_right,\
    RCNN_loss_cls, RCNN_loss_bbox, RCNN_loss_dim_orien,\
    RCNN_loss_kpts, rois_label = stereoRCNN(\
      im_left_data, im_right_data, im_info,\
      gt_boxes_left, gt_boxes_right,\
      gt_boxes_merge, gt_dim_orien, gt_kpts, num_boxes)

    scores = cls_prob.data
    boxes_left = rois_left.data[:, :, 1:5]
    boxes_right = rois_right.data[:, :, 1:5]

    bbox_pred = bbox_pred.data

    box_delta_left = bbox_pred.new(\
      bbox_pred.size()[1], 4*len(imdb._classes)).zero_()
```

```
box_delta_right = bbox_pred.new(\
  bbox_pred.size()[1], 4*len(imdb._classes)).zero_()

for keep_inx in range(box_delta_left.size()[0]):
  box_delta_left[keep_inx, 0::4] = \
    bbox_pred[0,keep_inx,0::6]
  box_delta_left[keep_inx, 1::4] = \
    bbox_pred[0,keep_inx,1::6]
  box_delta_left[keep_inx, 2::4] = \
    bbox_pred[0,keep_inx,2::6]
  box_delta_left[keep_inx, 3::4] = \
    bbox_pred[0,keep_inx,3::6]

  box_delta_right[keep_inx, 0::4] = \
    bbox_pred[0,keep_inx,4::6]
  box_delta_right[keep_inx, 1::4] = \
    bbox_pred[0,keep_inx,1::6]
  box_delta_right[keep_inx, 2::4] = \
    bbox_pred[0,keep_inx,5::6]
  box_delta_right[keep_inx, 3::4] = \
    bbox_pred[0,keep_inx,3::6]

box_delta_left = box_delta_left.view(-1,4)
box_delta_right = box_delta_right.view(-1,4)

dim_orien = bbox_pred_dim.data
dim_orien = dim_orien.view(-1,5)

kpts_prob = kpts_prob.data
kpts_prob = kpts_prob.view(-1,4*cfg.KPTS_GRID)
max_prob, kpts_delta = torch.max(kpts_prob,1)

left_prob = left_prob.data
left_prob = left_prob.view(-1,cfg.KPTS_GRID)
_, left_delta = torch.max(left_prob,1)

right_prob = right_prob.data
right_prob = right_prob.view(-1,cfg.KPTS_GRID)
_, right_delta = torch.max(right_prob,1)

box_delta_left = box_delta_left \
* torch.FloatTensor(cfg.TRAIN.BBOX_NORMALIZE_STDS)\
  .cuda() \
+ torch.FloatTensor(cfg.TRAIN.BBOX_NORMALIZE_MEANS)\
  .cuda()

box_delta_right = box_delta_right \
```

```
* torch.FloatTensor(cfg.TRAIN.BBOX_NORMALIZE_STDS)\
  .cuda() \
+ torch.FloatTensor(cfg.TRAIN.BBOX_NORMALIZE_MEANS)\
  .cuda()

dim_orien = dim_orien \
* torch.FloatTensor(cfg.TRAIN.DIM_NORMALIZE_STDS)\
  .cuda() \
+ torch.FloatTensor(cfg.TRAIN.DIM_NORMALIZE_MEANS)\
  .cuda()

box_delta_left = box_delta_left.view(\
  1,-1,4*len(imdb._classes))
box_delta_right = box_delta_right.view(\
  1, -1,4*len(imdb._classes))
dim_orien = dim_orien.view(1, -1, 5*len(imdb._classes))
kpts_delta = kpts_delta.view(1, -1, 1)
left_delta = left_delta.view(1, -1, 1)
right_delta = right_delta.view(1, -1, 1)
max_prob = max_prob.view(1, -1, 1)

pred_boxes_left = bbox_transform_inv(\
  boxes_left, box_delta_left, 1)
pred_boxes_right = bbox_transform_inv(\
  boxes_right, box_delta_right, 1)
pred_kpts, kpts_type = kpts_transform_inv(\
  boxes_left, kpts_delta,cfg.KPTS_GRID)
pred_left = border_transform_inv(\
  boxes_left, left_delta,cfg.KPTS_GRID)
pred_right = border_transform_inv(\
  boxes_left, right_delta,cfg.KPTS_GRID)

pred_boxes_left = clip_boxes(\
  pred_boxes_left, im_info.data, 1)
pred_boxes_right = clip_boxes(\
  pred_boxes_right, im_info.data, 1)

pred_boxes_left /= im_info[0,2].data
pred_boxes_right /= im_info[0,2].data
pred_kpts /= im_info[0,2].data
pred_left /= im_info[0,2].data
pred_right /= im_info[0,2].data

scores = scores.squeeze()
pred_boxes_left = pred_boxes_left.squeeze()
pred_boxes_right = pred_boxes_right.squeeze()
```

```
        pred_kpts = torch.cat((pred_kpts, \
          kpts_type, max_prob, pred_left, pred_right),2)
        pred_kpts = pred_kpts.squeeze()
        dim_orien = dim_orien.squeeze()

        det_toc = time.time()
        detect_time = det_toc - det_tic

        img_path = imdb.img_left_path_at(i)
        split_path = img_path.split('/')
        image_number = split_path[len(split_path)\
          -1].split('.')[0]
        calib_path = img_path.replace("image_2", "calib")
        calib_path = calib_path.replace("png", "txt")
        calib = kitti_utils.read_obj_calibration(calib_path)
        label_path = calib_path.replace("calib", "label_2")
        lidar_path = calib_path.replace("calib", "velodyne")
        lidar_path = lidar_path.replace("txt", "bin")

        im2show_left = np.copy(cv2.imread(\
          imdb.img_left_path_at(i)))
        im2show_right = np.copy(cv2.imread(\
          imdb.img_right_path_at(i)))

        pointcloud = kitti_utils.get_point_cloud(\
          lidar_path, calib)
        im_box = vis_utils.vis_lidar_in_bev(\
          pointcloud, width=im2show_left.shape[0]*2)

        for j in xrange(1, imdb.num_classes):
          inds = torch.nonzero(scores[:,j] > eval_thresh)\
          .view(-1)
          # if there is det
          if inds.numel() > 0:
            cls_scores = scores[:,j][inds]
            _, order = torch.sort(cls_scores, 0, True)

            cls_boxes_left = pred_boxes_left[inds]\
              [:, j * 4:(j + 1) * 4]
            cls_boxes_right = pred_boxes_right[inds]\
              [:, j * 4:(j + 1) * 4]
            cls_dim_orien = dim_orien[inds]\
              [:, j * 5:(j + 1) * 5]

            cls_kpts = pred_kpts[inds]

            cls_dets_left = torch.cat(\
```

```
  (cls_boxes_left, cls_scores.unsqueeze(1)), 1)
cls_dets_right = torch.cat(\
  (cls_boxes_right, cls_scores.unsqueeze(1)), 1)

cls_dets_left = cls_dets_left[order]
cls_dets_right = cls_dets_right[order]
cls_dim_orien = cls_dim_orien[order]
cls_kpts = cls_kpts[order]

keep = nms(cls_boxes_left[order, :],\
  cls_scores[order], cfg.TEST.NMS)
keep = keep.view(-1).long()
cls_dets_left = cls_dets_left[keep]
cls_dets_right = cls_dets_right[keep]
cls_dim_orien = cls_dim_orien[keep]
cls_kpts = cls_kpts[keep]

# optional operation, can check the regressed
# borderline keypoint using 2D box inference
infered_kpts = kitti_utils.infer_boundary(\
  im2show_left.shape, cls_dets_left.cpu().numpy())
infered_kpts = torch.from_numpy(\
  infered_kpts).type_as(cls_dets_left)
for detect_idx in range(cls_dets_left.size()[0]):
  if cls_kpts[detect_idx,4] \
  - cls_kpts[detect_idx,3] \
  < 0.5*(infered_kpts[detect_idx,1]\
  -infered_kpts[detect_idx,0]):
    cls_kpts[detect_idx,3:5] \
    = infered_kpts[detect_idx]

im2show_left = vis_detections(\
  im2show_left, imdb._classes[j], \
  cls_dets_left.cpu().numpy(),\
  vis_thresh, cls_kpts.cpu().numpy())
im2show_right = vis_detections(\
  im2show_right, imdb._classes[j], \
  cls_dets_right.cpu().numpy(), vis_thresh)

# read intrinsic
f = calib.p2[0,0]
cx, cy = calib.p2[0,2], calib.p2[1,2]
bl = (calib.p2[0,3] - calib.p3[0,3])/f

boxes_all = cls_dets_left.new(0,5)
kpts_all = cls_dets_left.new(0,5)
poses_all = cls_dets_left.new(0,8)
```

```
solve_tic = time.time()
for detect_idx in range(cls_dets_left.size()[0]):
  if cls_dets_left[detect_idx, -1] \
  > eval_thresh:
    box_left = cls_dets_left[detect_idx,0:4]\
    .cpu().numpy()
    # based on origin image
    box_right = cls_dets_right[detect_idx,0:4]\
    .cpu().numpy()
    kpts_u = cls_kpts[detect_idx,0]
    dim = cls_dim_orien[detect_idx,0:3]\
    .cpu().numpy()
    sin_alpha = cls_dim_orien[detect_idx,3]
    cos_alpha = cls_dim_orien[detect_idx,4]
    alpha = m.atan2(sin_alpha, cos_alpha)
    status, state = box_estimator\
    .solve_x_y_z_theta_from_kpt(\
      im2show_left.shape, calib, alpha, \
      dim, box_left, box_right,\
      cls_kpts[detect_idx].cpu().numpy())
    if status > 0: # not faild
      poses = im_left_data.data.new(8).zero_()
      xyz = np.array(\
        [state[0], state[1], state[2]])
      theta = state[3]
      poses[0], poses[1], poses[2], poses[3],\
      poses[4], poses[5], poses[6], poses[7] = \
        xyz[0], xyz[1], xyz[2], float(dim[0]),\
        float(dim[1]), float(dim[2]), theta, alpha

      boxes_all = torch.cat((\
        boxes_all,cls_dets_left[detect_idx,0:5]\
        .unsqueeze(0)),0)
      kpts_all = torch.cat((\
        kpts_all,cls_kpts[detect_idx]\
        .unsqueeze(0)),0)
      poses_all = torch.cat((\
        poses_all,poses.unsqueeze(0)),0)

if boxes_all.dim() > 0:
  # solve disparity by dense alignment
  succ, dis_final = dense_align.align_parallel(\
    calib, im_info.data[0,2],\
    im_left_data.data, im_right_data.data, \
    boxes_all[:,0:4], kpts_all, poses_all[:,0:7])
```

```
            # do 3D rectify using the aligned disparity
            for solved_idx in range(succ.size(0)):
              if succ[solved_idx] > 0: # succ
                box_left = boxes_all[solved_idx,0:4]\
                .cpu().numpy()
                score = boxes_all[solved_idx,4].cpu().numpy()
                dim = poses_all[solved_idx,3:6].cpu().numpy()
                state_rect, z = box_estimator\
                .solve_x_y_theta_from_kpt(\
                  im2show_left.shape, calib, \
                  poses_all[solved_idx,7].cpu().numpy(),\
                  dim, box_left, \
                  dis_final[solved_idx].cpu().numpy(),\
                  kpts_all[solved_idx].cpu().numpy())
                xyz = np.array(\
                  [state_rect[0], state_rect[1], z])
                theta = state_rect[2]

                if score > vis_thresh:
                  im_box = vis_utils.vis_box_in_bev(\
                    im_box, xyz, dim, theta,\
                    width=im2show_left.shape[0]*2)
                  im2show_left = vis_utils\
                  .vis_single_box_in_img(\
                    im2show_left, calib, xyz, dim, theta)

                # write result into txt file
                kitti_utils.write_detection_results(\
                  result_dir, image_number, calib, box_left,\
                  xyz, dim, theta, score)
          solve_time = time.time() - solve_tic

      sys.stdout.write('test:{:d}/{:d} det_time {:.2f}s,\
        solve time {:.2f}s (Press Esc to exit!) \r'\
        .format(i + 1, num_images, detect_time, solve_time))
      sys.stdout.flush()

      im2show = np.concatenate(\
        (im2show_left, im2show_right), axis=0)
      im2show = np.concatenate((im2show, im_box), axis=1)
      cv2.imshow('result', im2show)

      k = cv2.waitKey(1)
      if k == 27:    # Esc key to stop
          print('exit!')
          sys.exit()
  print('test finish, result is saved in %s' %result_dir)
```

Program : train.sh

```
CUDA_VISIBLE_DEVICES=0 python trainval_net.py --bs 1
```

Program : trainval_net.py

```
# --------------------------------------------------------
# Pytorch multi-GPU Faster CNN
# Licensed under The MIT License [see LICENSE for details]
# Written by Jiasen Lu, Jianwei Yang, based on code from Ross
Girshick

# Modified by Adrian A for Stereo R-CNN
# --------------------------------------------------------
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import _init_paths
import os
import sys
import numpy as np
import argparse
import time

import torch
from torch.autograd import Variable
import torch.nn as nn
import torch.optim as optim

import torchvision.transforms as transforms
from torch.utils.data.sampler import Sampler

from roi_data_layer.roidb import combined_roidb
from roi_data_layer.roibatchLoader import roibatchLoader
from model.utils.config import cfg
from model.utils.net_utils import weights_normal_init,\
  save_net, load_net, adjust_learning_rate, \
  save_checkpoint, clip_gradient

from model.stereo_rcnn.resnet import resnet

def parse_args():
  '''
  Parse input arguments
  '''
  parser = argparse.ArgumentParser(\
```

```
    description='Train the Stereo R-CNN network')

  parser.add_argument('--start_epoch', dest='start_epoch',\
    help='starting epoch', default=1, type=int)
  parser.add_argument('--epochs', dest='max_epochs',\
    help='number of epochs to train', default=12, type=int)

  parser.add_argument('--save_dir', dest='save_dir',\
    help='directory to save models', \
    default="models_stereo", type=str)
  parser.add_argument('--nw', dest='num_workers',
    help='number of worker to load data',\
    default=8, type=int)
  parser.add_argument('--bs', dest='batch_size',\
    help='batch_size', default=1, type=int)

  # config optimization
  parser.add_argument('--lr_decay_step', \
    dest='lr_decay_step',\
    help='step to do learning rate decay, unit is epoch',\
    default=10, type=int)
  parser.add_argument('--lr_decay_gamma', \
    dest='lr_decay_gamma',\
    help='learning rate decay ratio',\
    default=0.1, type=float)

  # resume trained model
  parser.add_argument('--r', dest='resume',\
    help='resume checkpoint or not',\
    default=False, type=bool)
  parser.add_argument('--checkepoch', dest='checkepoch',\
    help='checkepoch to load model',\
    default=1, type=int)
  parser.add_argument('--checkpoint', dest='checkpoint',\
    help='checkpoint to load model',\
    default=6477, type=int)

  args = parser.parse_args()
  return args


class sampler(Sampler):
  def __init__(self, train_size, batch_size):
    self.num_data = train_size
    self.num_per_batch = int(train_size / batch_size)
    self.batch_size = batch_size
    self.range = torch.arange(0,batch_size)\
```

```
    .view(1, batch_size).long()
    self.leftover_flag = False
    if train_size % batch_size:
      self.leftover = torch.arange(\
        self.num_per_batch*batch_size, train_size).long()
      self.leftover_flag = True

  def __iter__(self):
    rand_num = torch.randperm(self.num_per_batch)\
    .view(-1,1) * self.batch_size
    self.rand_num = rand_num.expand(\
      self.num_per_batch, self.batch_size) + self.range

    self.rand_num_view = self.rand_num.view(-1)

    if self.leftover_flag:
      self.rand_num_view = torch.cat(\
        (self.rand_num_view, self.leftover),0)

    return iter(self.rand_num_view)

  def __len__(self):
    return self.num_data

if __name__ == '__main__':

  args = parse_args()

  print('Using config:')
  np.random.seed(cfg.RNG_SEED)

  imdb, roidb, ratio_list, ratio_index = combined_roidb(\
    'kitti_train')
  train_size = len(roidb)

  print('{:d} roidb entries'.format(len(roidb)))

  output_dir = args.save_dir + '/'
  if not os.path.exists(output_dir):
    print('save dir', output_dir)
    os.makedirs(output_dir)
  log_info = open((output_dir + 'trainlog.txt'), 'w')

  def log_string(out_str):
    log_info.write(out_str+'\n')
    log_info.flush()
    print(out_str)
```

```
sampler_batch = sampler(train_size, args.batch_size)

dataset = roibatchLoader(\
  roidb, ratio_list, ratio_index, args.batch_size, \
  imdb.num_classes, training=True)

dataloader = torch.utils.data.DataLoader(dataset,\
  batch_size=args.batch_size,
  sampler=sampler_batch, num_workers=args.num_workers)

# initilize the tensor holder here.
im_left_data = Variable(torch.FloatTensor(1).cuda())
im_right_data = Variable(torch.FloatTensor(1).cuda())
im_info = Variable(torch.FloatTensor(1).cuda())
num_boxes = Variable(torch.LongTensor(1).cuda())
gt_boxes_left = Variable(torch.FloatTensor(1).cuda())
gt_boxes_right = Variable(torch.FloatTensor(1).cuda())
gt_boxes_merge = Variable(torch.FloatTensor(1).cuda())
gt_dim_orien = Variable(torch.FloatTensor(1).cuda())
gt_kpts = Variable(torch.FloatTensor(1).cuda())

# initilize the network here.
stereoRCNN = resnet(imdb.classes, 101, pretrained=True)

stereoRCNN.create_architecture()

lr = cfg.TRAIN.LEARNING_RATE

uncert = Variable(torch.rand(6).cuda(), requires_grad=True)
torch.nn.init.constant(uncert, -1.0)

params = []
for key, value in dict(stereoRCNN.named_parameters())\
.items():
  if value.requires_grad:
    if 'bias' in key:
      params += [{'params':[value],\
        'lr':lr*(cfg.TRAIN.DOUBLE_BIAS + 1), \
        'weight_decay': cfg.TRAIN.BIAS_DECAY \
          and cfg.TRAIN.WEIGHT_DECAY or 0}]
    else:
      params += [{'params':[value],'lr':lr, \
        'weight_decay': cfg.TRAIN.WEIGHT_DECAY}]
params += [{'params':[uncert], 'lr':lr}]

optimizer = torch.optim.SGD(params,\
```

```
    momentum=cfg.TRAIN.MOMENTUM)

  if args.resume:
    load_name = os.path.join(output_dir,
      'stereo_rcnn_{}_{}.pth'.format(args.checkepoch,
args.checkpoint))
    log_string('loading checkpoint %s' % (load_name))
    checkpoint = torch.load(load_name)
    args.start_epoch = checkpoint['epoch']
    stereoRCNN.load_state_dict(checkpoint['model'])
    lr = optimizer.param_groups[0]['lr']
    uncert.data = checkpoint['uncert']
    log_string('loaded checkpoint %s' % (load_name))

  stereoRCNN.cuda()

  iters_per_epoch = int(train_size / args.batch_size)
  for epoch in range(args.start_epoch, args.max_epochs + 1):

    stereoRCNN.train()
    start = time.time()

    if epoch % (args.lr_decay_step + 1) == 0:
        adjust_learning_rate(optimizer, args.lr_decay_gamma)
        lr *= args.lr_decay_gamma

    data_iter = iter(dataloader)
    for step in range(iters_per_epoch):
      data = next(data_iter)
      im_left_data.data.resize_(data[0].size())\
      .copy_(data[0])
      im_right_data.data.resize_(data[1].size())\
      .copy_(data[1])
      im_info.data.resize_(data[2].size())\
      .copy_(data[2])
      gt_boxes_left.data.resize_(data[3].size())\
      .copy_(data[3])
      gt_boxes_right.data.resize_(data[4].size())\
      .copy_(data[4])
      gt_boxes_merge.data.resize_(data[5].size())\
      .copy_(data[5])
      gt_dim_orien.data.resize_(data[6].size())\
      .copy_(data[6])
      gt_kpts.data.resize_(data[7].size())\
      .copy_(data[7])
      num_boxes.data.resize_(data[8].size())\
      .copy_(data[8])
```

```
start = time.time()
stereoRCNN.zero_grad()
rois_left, rois_right, cls_prob, bbox_pred,\
dim_orien_pred, kpts_prob, \
left_border_prob, right_border_prob, \
rpn_loss_cls, rpn_loss_box_left_right,\
RCNN_loss_cls, RCNN_loss_bbox, \
RCNN_loss_dim_orien, RCNN_loss_kpts, rois_label =\
stereoRCNN(im_left_data, im_right_data,\
  im_info, gt_boxes_left, gt_boxes_right, \
  gt_boxes_merge, gt_dim_orien, gt_kpts, num_boxes)

loss = rpn_loss_cls.mean() * torch.exp(-uncert[0]) \
  + uncert[0] + rpn_loss_box_left_right.mean() \
  * torch.exp(-uncert[1]) + uncert[1] \
  + RCNN_loss_cls.mean() * torch.exp(-uncert[2]) \
  + uncert[2] + RCNN_loss_bbox.mean() \
  * torch.exp(-uncert[3]) + uncert[3] \
  + RCNN_loss_dim_orien.mean() * torch.exp(-uncert[4])\
  + uncert[4] + RCNN_loss_kpts.mean() \
  * torch.exp(-uncert[5]) + uncert[5]
uncert_data = uncert.data
log_string('uncert: %.4f, %.4f, %.4f, %.4f, %.4f, %.4f'\
  % (uncert_data[0], uncert_data[1], \
    uncert_data[2], uncert_data[3], \
    uncert_data[4], uncert_data[5]))

optimizer.zero_grad()
loss.backward()
clip_gradient(stereoRCNN, 10.)
optimizer.step()

end = time.time()

loss_rpn_cls = rpn_loss_cls.item()
loss_rpn_box_left_right =  rpn_loss_box_left_right\
.item()
loss_rcnn_cls = RCNN_loss_cls.item()
loss_rcnn_box = RCNN_loss_bbox.item()
loss_rcnn_dim_orien = RCNN_loss_dim_orien.item()
loss_rcnn_kpts = RCNN_loss_kpts
fg_cnt = torch.sum(rois_label.data.ne(0))
bg_cnt = rois_label.data.numel() - fg_cnt

log_string('[epoch %2d][iter %4d/%4d] \
  loss: %.4f, lr: %.2e' \
```

```
        %(epoch, step, iters_per_epoch, loss.item(), lr))
      log_string('\t\t\tfg/bg=(%d/%d), \
        time cost: %f' %(fg_cnt, bg_cnt, end-start))
      log_string('\t\t\trpn_cls: %.4f, \
        rpn_box_left_right: %.4f, \
        rcnn_cls: %.4f,\
        rcnn_box_left_right %.4f,dim_orien %.4f, kpts %.4f' \
        %(loss_rpn_cls, loss_rpn_box_left_right,\
          loss_rcnn_cls, loss_rcnn_box, \
          loss_rcnn_dim_orien, loss_rcnn_kpts))

      del loss, rpn_loss_cls, rpn_loss_box_left_right,\
      RCNN_loss_cls, RCNN_loss_bbox, RCNN_loss_dim_orien,\
      RCNN_loss_kpts

    save_name = os.path.join(output_dir,\
      'stereo_rcnn_{}_{}.pth'.format(epoch, step))
    save_checkpoint({
      'epoch': epoch + 1,
      'model': stereoRCNN.state_dict(),
      'optimizer': optimizer.state_dict(),
      'uncert':uncert.data,
    }, save_name)

    log_string('save model: {}'.format(save_name))
    end = time.time()
    log_string('time %.4f' %(end - start))
```

Program : demo.py

```
# --------------------------------------------------------
# Tensorflow Faster CNN
# Licensed under The MIT License [see LICENSE for details]
# Written by Jiasen Lu, Jianwei Yang, based on code from Ross
Girshick

# Modified by Peiliang Li for Stereo CNN
# Modified by Adrian A for Stereo R-CNN
# --------------------------------------------------------
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import _init_paths
import os
```

```
import sys
import numpy as np
import argparse
import shutil
import time
import cv2
import torch
from torch.autograd import Variable
import torch.nn as nn
import torch.optim as optim
import math as m
from roi_data_layer.roidb import combined_roidb
from roi_data_layer.roibatchLoader import roibatchLoader
from model.utils.config import cfg
from model.rpn.bbox_transform import clip_boxes
from model.roi_layers import nms
from model.rpn.bbox_transform import bbox_transform_inv, \
  kpts_transform_inv, border_transform_inv
from model.utils.net_utils import save_net, \
  load_net, vis_detections
from model.stereo_rcnn.resnet import resnet
from model.utils import kitti_utils
from model.utils import vis_3d_utils as vis_utils
from model.utils import box_estimator as box_estimator
from model.dense_align import dense_align

try:
  xrange          # Python 2
except NameError:
  xrange = range  # Python 3

def parse_args():
  """
  Parse input arguments
  """
  parser = argparse.ArgumentParser(description= \
    'Test the Stereo R-CNN network')

  parser.add_argument('--load_dir', dest='load_dir',
                      help='directory to load models',
                      default="models_stereo", type=str)
  parser.add_argument('--checkepoch', dest='checkepoch',
                      help='checkepoch to load network',
                      default=12, type=int)
  parser.add_argument('--checkpoint', dest='checkpoint',
                      help='checkpoint to load network',
                      default=6477, type=int)
```

```
  args = parser.parse_args()
  return args

if __name__ == '__main__':

  # timer for Neural Network load time
  start_time = time.time()

  args = parse_args()

  np.random.seed(cfg.RNG_SEED)

  input_dir = args.load_dir + "/"
  if not os.path.exists(input_dir):
    raise Exception( \
      'There is no input directory for loading network from'\
      + input_dir)
  load_name = os.path.join(input_dir,
    'stereo_rcnn_{}_{}.pth'.format(args.checkepoch, \
    args.checkpoint))
  kitti_classes = np.asarray(['__background__', 'Car'])

  # initilize the network here.
  stereoRCNN = resnet(kitti_classes, 101, pretrained=False)
  stereoRCNN.create_architecture()

  print("load checkpoint %s" % (load_name))
  checkpoint = torch.load(load_name)
  stereoRCNN.load_state_dict(checkpoint['model'])
  print('load model successfully!')

  with torch.no_grad():
    # initilize the tensor holder here.
    im_left_data = Variable(torch.FloatTensor(1).cuda())
    im_right_data = Variable(torch.FloatTensor(1).cuda())
    im_info = Variable(torch.FloatTensor(1).cuda())
    num_boxes = Variable(torch.LongTensor(1).cuda())
    gt_boxes = Variable(torch.FloatTensor(1).cuda())

    stereoRCNN.cuda()

    eval_thresh = 0.05
    vis_thresh = 0.7

    stereoRCNN.eval()
```

```
# path list
data_selected = 'data_437'
img_l_path = f'demo/left/{data_selected}/'
img_r_path = f'demo/right/{data_selected}/'
lidar_path = f'demo/lidar/{data_selected}/'
result_path = 'demo/result/'

# get all available data name
file_name_list = os.listdir(img_l_path)
file_name_list.sort()

print(file_name_list)

# timer / fps counter
print(f'model load time : {time.time() - start_time} ms')
start_time = time.time()

#initialize video flag
is_init_video = True

# for each files in demo, use Stereo RCNN.
for file_name in file_name_list:

  print(f"calculating {file_name} ...")

  # set path to file
  file_name_no_ext,_ = os.path.splitext(file_name)
  current_img_l_path = img_l_path + file_name
  current_img_r_path = img_r_path + file_name
  current_lidar_path = lidar_path + file_name_no_ext \
                       + '.bin'
  current_result_path = result_path + 'data/' + file_name

  print(current_img_l_path)
  print(current_img_r_path)
  print(current_lidar_path)

  img_left = cv2.imread(current_img_l_path)
  img_right = cv2.imread(current_img_r_path)

  # rgb -> bgr
  img_left = img_left.astype(np.float32, copy=False)
  img_right = img_right.astype(np.float32, copy=False)

  img_left -= cfg.PIXEL_MEANS
  img_right -= cfg.PIXEL_MEANS
```

```
im_shape = img_left.shape
im_size_min = np.min(im_shape[0:2])
im_scale = float(cfg.TRAIN.SCALES[0]) \
         / float(im_size_min)

img_left = cv2.resize(img_left, None, None, \
                fx=im_scale, fy=im_scale, \
                interpolation=cv2.INTER_LINEAR)
img_right = cv2.resize(img_right, None, None, \
                fx=im_scale, fy=im_scale,
                interpolation=cv2.INTER_LINEAR)

info = np.array([[img_left.shape[0], \
                  img_left.shape[1], \
                  im_scale]], dtype=np.float32)

img_left = torch.from_numpy(img_left)
img_left = img_left.permute(2, 0, 1) \
           .unsqueeze(0).contiguous()

img_right = torch.from_numpy(img_right)
img_right = img_right.permute(2, 0, 1) \
            .unsqueeze(0).contiguous()

info = torch.from_numpy(info)

im_left_data.data.resize_(img_left.size()) \
.copy_(img_left)
im_right_data.data.resize_(img_right.size()) \
.copy_(img_right)
im_info.data.resize_(info.size()).copy_(info)

det_tic = time.time()
rois_left, rois_right, cls_prob, bbox_pred, \
bbox_pred_dim, kpts_prob, left_prob, right_prob, \
rpn_loss_cls, rpn_loss_box_left_right,\
RCNN_loss_cls, RCNN_loss_bbox, RCNN_loss_dim_orien, \
RCNN_loss_kpts, rois_label = stereoRCNN( \
  im_left_data, im_right_data, \
  im_info, gt_boxes, gt_boxes,\
  gt_boxes, gt_boxes, gt_boxes, num_boxes)

scores = cls_prob.data
boxes_left = rois_left.data[:, :, 1:5]
boxes_right = rois_right.data[:, :, 1:5]

bbox_pred = bbox_pred.data
```

```
box_delta_left = bbox_pred.new(bbox_pred.size()[1], \
  4*len(kitti_classes)).zero_()
box_delta_right = bbox_pred.new(bbox_pred.size()[1], \
  4*len(kitti_classes)).zero_()

for keep_inx in range(box_delta_left.size()[0]):
  box_delta_left[keep_inx, 0::4] = \
    bbox_pred[0,keep_inx,0::6]
  box_delta_left[keep_inx, 1::4] = \
    bbox_pred[0,keep_inx,1::6]
  box_delta_left[keep_inx, 2::4] = \
    bbox_pred[0,keep_inx,2::6]
  box_delta_left[keep_inx, 3::4] = \
    bbox_pred[0,keep_inx,3::6]

  box_delta_right[keep_inx, 0::4] = \
    bbox_pred[0,keep_inx,4::6]
  box_delta_right[keep_inx, 1::4] = \
    bbox_pred[0,keep_inx,1::6]
  box_delta_right[keep_inx, 2::4] = \
    bbox_pred[0,keep_inx,5::6]
  box_delta_right[keep_inx, 3::4] = \
    bbox_pred[0,keep_inx,3::6]

box_delta_left = box_delta_left.view(-1,4)
box_delta_right = box_delta_right.view(-1,4)

dim_orien = bbox_pred_dim.data
dim_orien = dim_orien.view(-1,5)

kpts_prob = kpts_prob.data
kpts_prob = kpts_prob.view(-1,4*cfg.KPTS_GRID)
max_prob, kpts_delta = torch.max(kpts_prob,1)

left_prob = left_prob.data
left_prob = left_prob.view(-1,cfg.KPTS_GRID)
_, left_delta = torch.max(left_prob,1)

right_prob = right_prob.data
right_prob = right_prob.view(-1,cfg.KPTS_GRID)
_, right_delta = torch.max(right_prob,1)

box_delta_left = box_delta_left \
  * torch.FloatTensor( \
    cfg.TRAIN.BBOX_NORMALIZE_STDS).cuda() \
  + torch.FloatTensor( \
    cfg.TRAIN.BBOX_NORMALIZE_MEANS).cuda()
```

```
box_delta_right = box_delta_right  \
  * torch.FloatTensor( \
    cfg.TRAIN.BBOX_NORMALIZE_STDS).cuda() \
  + torch.FloatTensor( \
    cfg.TRAIN.BBOX_NORMALIZE_MEANS).cuda()
dim_orien = dim_orien \
  * torch.FloatTensor( \
    cfg.TRAIN.DIM_NORMALIZE_STDS).cuda() \
  + torch.FloatTensor( \
    cfg.TRAIN.DIM_NORMALIZE_MEANS).cuda()

box_delta_left = box_delta_left.view( \
                 1,-1,4*len(kitti_classes))
box_delta_right = box_delta_right.view( \
                 1, -1,4*len(kitti_classes))
dim_orien = dim_orien.view(1, -1, 5*len(kitti_classes))
kpts_delta = kpts_delta.view(1, -1, 1)
left_delta = left_delta.view(1, -1, 1)
right_delta = right_delta.view(1, -1, 1)
max_prob = max_prob.view(1, -1, 1)

pred_boxes_left = bbox_transform_inv( \
                  boxes_left, box_delta_left, 1)
pred_boxes_right = bbox_transform_inv( \
                   boxes_right, box_delta_right, 1)
pred_kpts, kpts_type = kpts_transform_inv(boxes_left,\
                       kpts_delta,cfg.KPTS_GRID)
pred_left = border_transform_inv(boxes_left, \
            left_delta,cfg.KPTS_GRID)
pred_right = border_transform_inv(boxes_left, \
            right_delta,cfg.KPTS_GRID)

pred_boxes_left = clip_boxes(pred_boxes_left, \
                  im_info.data, 1)
pred_boxes_right = clip_boxes(pred_boxes_right, \
                   im_info.data, 1)

pred_boxes_left /= im_info[0,2].data
pred_boxes_right /= im_info[0,2].data
pred_kpts /= im_info[0,2].data
pred_left /= im_info[0,2].data
pred_right /= im_info[0,2].data

scores = scores.squeeze()
pred_boxes_left = pred_boxes_left.squeeze()
pred_boxes_right = pred_boxes_right.squeeze()
```

```
        pred_kpts = torch.cat((pred_kpts, kpts_type, \
                    max_prob, pred_left, pred_right),2)
        pred_kpts = pred_kpts.squeeze()
        dim_orien = dim_orien.squeeze()

        det_toc = time.time()
        detect_time = det_toc - det_tic

        calib = kitti_utils \
                .read_obj_calibration('demo/calib.txt')

        im2show_left = np.copy(cv2.imread(current_img_l_path))
        im2show_right = np.copy(cv2.imread(current_img_r_path))

        pointcloud = kitti_utils \
          .get_point_cloud(current_lidar_path, calib)
        im_box = vis_utils.vis_lidar_in_bev( \
          pointcloud, width=im2show_left.shape[0]*2)

        for j in xrange(1, len(kitti_classes)):
          inds = torch.nonzero(scores[:,j] > \
            eval_thresh).view(-1)
          # if there is det
          if inds.numel() > 0:
            cls_scores = scores[:,j][inds]
            _, order = torch.sort(cls_scores, 0, True)

            cls_boxes_left = pred_boxes_left[inds]\
              [:, j * 4:(j + 1) * 4]
            cls_boxes_right = pred_boxes_right[inds]\
              [:, j * 4:(j + 1) * 4]
            cls_dim_orien = dim_orien[inds]\
              [:, j * 5:(j + 1) * 5]

            cls_kpts = pred_kpts[inds]

            cls_dets_left = torch.cat((cls_boxes_left, \
              cls_scores.unsqueeze(1)), 1)
            cls_dets_right = torch.cat((cls_boxes_right, \
              cls_scores.unsqueeze(1)), 1)

            cls_dets_left = cls_dets_left[order]
            cls_dets_right = cls_dets_right[order]
            cls_dim_orien = cls_dim_orien[order]
            cls_kpts = cls_kpts[order]

            keep = nms(cls_boxes_left[order, :], \
```

```
  cls_scores[order], cfg.TEST.NMS)
keep = keep.view(-1).long()
cls_dets_left = cls_dets_left[keep]
cls_dets_right = cls_dets_right[keep]
cls_dim_orien = cls_dim_orien[keep]
cls_kpts = cls_kpts[keep]

# optional operation, can check the regressed
# borderline keypoint using 2D box inference
infered_kpts = kitti_utils \
.infer_boundary(im2show_left.shape, \
  cls_dets_left.cpu().numpy())
infered_kpts = torch.from_numpy(infered_kpts)\
.type_as(cls_dets_left)
for detect_idx in range(cls_dets_left.size()[0]):
  if cls_kpts[detect_idx,4] \
     - cls_kpts[detect_idx,3] \
     < 0.5*(infered_kpts[detect_idx,1] \
     - infered_kpts[detect_idx,0]):
    cls_kpts[detect_idx,3:5] = \
      infered_kpts[detect_idx]

im2show_left = vis_detections(im2show_left, \
  kitti_classes[j], \
  cls_dets_left.cpu().numpy(),\
  vis_thresh, cls_kpts.cpu().numpy())

im2show_right = vis_detections(im2show_right, \
  kitti_classes[j], \
  cls_dets_right.cpu().numpy(), vis_thresh)

# read intrinsic
f = calib.p2[0,0]
cx, cy = calib.p2[0,2], calib.p2[1,2]
bl = (calib.p2[0,3] - calib.p3[0,3])/f

boxes_all = cls_dets_left.new(0,5)
kpts_all = cls_dets_left.new(0,5)
poses_all = cls_dets_left.new(0,8)

solve_tic = time.time()
for detect_idx in range(cls_dets_left.size()[0]):
  if cls_dets_left[detect_idx, -1] > eval_thresh:
    box_left = cls_dets_left[detect_idx,0:4]\
    .cpu().numpy()  # based on origin image
    box_right = cls_dets_right[detect_idx,0:4]\
    .cpu().numpy()
```

```
            kpts_u = cls_kpts[detect_idx,0]
            dim = cls_dim_orien[detect_idx,0:3]\
            .cpu().numpy()
            sin_alpha = cls_dim_orien[detect_idx,3]
            cos_alpha = cls_dim_orien[detect_idx,4]
            alpha = m.atan2(sin_alpha, cos_alpha)
            status, state = box_estimator\
            .solve_x_y_z_theta_from_kpt( \
              im2show_left.shape, calib, alpha, \
              dim, box_left, box_right, \
              cls_kpts[detect_idx].cpu().numpy())
            if status > 0: # not faild
              poses = im_left_data.data.new(8).zero_()
              xyz = np.array(\
                [state[0], state[1], state[2]])
              theta = state[3]
              poses[0], poses[1], poses[2], poses[3], \
              poses[4], poses[5], poses[6], poses[7] = \
                xyz[0], xyz[1], xyz[2], float(dim[0]) ,\
                float(dim[1]), float(dim[2]), theta, alpha

              boxes_all = torch.cat((\
                boxes_all,cls_dets_left[detect_idx,0:5]\
                .unsqueeze(0)),0)
              kpts_all = torch.cat((\
                kpts_all,cls_kpts[detect_idx]\
                .unsqueeze(0)),0)
              poses_all = torch.cat((\
                poses_all,poses.unsqueeze(0)),0)

        if boxes_all.dim() > 0:
          # solve disparity by dense alignment
          succ, dis_final =  dense_align\
          .align_parallel(calib, im_info.data[0,2], \
             im_left_data.data, im_right_data.data, \
             boxes_all[:,0:4], kpts_all, poses_all[:,0:7])

          # do 3D rectify using the aligned disparity
          for solved_idx in range(succ.size(0)):
            if succ[solved_idx] > 0: # succ
              box_left = boxes_all[solved_idx,0:4]\
              .cpu().numpy()
              score = boxes_all[solved_idx,4].cpu().numpy()
              dim = poses_all[solved_idx,3:6].cpu().numpy()
              state_rect, z = box_estimator\
              .solve_x_y_theta_from_kpt(\
                im2show_left.shape, calib, \
```

```
                poses_all[solved_idx,7].cpu().numpy(),\
                dim, box_left, \
                dis_final[solved_idx].cpu().numpy(), \
                kpts_all[solved_idx].cpu().numpy())
              xyz = np.array(\
                [state_rect[0], state_rect[1], z])
              theta = state_rect[2]

              if score > vis_thresh:
                im_box = vis_utils.vis_box_in_bev(\
                  im_box, xyz, dim, theta, \
                  width=im2show_left.shape[0]*2)
                im2show_left = vis_utils\
                .vis_single_box_in_img(\
                  im2show_left, calib, xyz, dim, theta)

      im2show = np.concatenate(\
        (im2show_left, im2show_right), axis=0)
      im2show = np.concatenate((im2show, im_box), axis=1)

      if(is_init_video):
        height, width, channels = im2show.shape
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        video_out = cv2.VideoWriter(result_path \
          + data_selected \
          + '.mp4', fourcc, 10.0, (width, height))
        is_init_video = False

      video_out.write(im2show)

      # cv2.imshow('result', im2show)
      # cv2.imwrite(current_result_path, im2show)
      print(f"Saving to {current_result_path} ...")

      # timer / fps counter
      print(f'FPS : {1/(time.time() - start_time)}')
      start_time = time.time()

    cv2.imread()
    sys.exit()
```

Lain-lain :

Program, data percobaan, gambar, dan video terkait dapat dilihat secara lebih lengkap dan jelas di tautan berikut : https://s.id/mwtYy

Halaman ini sengaja dikosongkan

BIODATA PENULIS

Adrian Aryaputra Firmansyah, lahir di Yogyakarta pada tanggal 23 Mei 1998. Penulis merupakan putra pertama dari tiga bersaudara. Penulis menyelesaikan pendidikan di SD Sutomo 2 Medan pada tahun 2010. Kemudian melanjutkan pendidikan di SMP Sutomo 2 Medan pada tahun 2010. Dan pada tahun 2013 penulis melanjutkan pendidikan di SMAN 10 Yogyakarta. Pada tahun 2016 penuis memulai jenjang S1 di Departemen Teknik Elektro Institut Teknologi Sepuluh Nopember Surabaya, dengan mengambil program studi Teknik Sistem Pengaturan. Selama kuliah, penulis aktif mengikuti kegiatan dari Laboraturium Dasar Sistem Pengaturan, dan Laboratorium Sistem dan Sibernetika. Untuk menghubungi penulis, dapat melalui surat elektronik pada alamat adrianaryaputra@icloud.com