BAB 3 METODOLOGI PENELITIAN

Bab ini menjelaskan analisis masalah yang diteliti serta garis besar perancangan perangkat lunak yang dikembangkan. Analisis masalah yang dilakukan meliputi alur kerja perangkat lunak dan hasil akhir yang ingin dicapai.

3.1 Analisis Masalah

Masalah segmentasi citra kian meningkat dan semakin dibutuhkan di berbagai bidang, terutama dalam bidang pengembangan sistem transportasi pintar. Transportasi pintar perlu mampu melakukan pengenalan citra kota secara presisi untuk mengambil keputusan dengan baik. Masalah yang kerap dihadapi dalam proses segmentasi semantik adalah perlunya mendeteksi objek dengan beragam ukuran, posisi, dan pengaruh oklusi serta pencahayaan. Penelitian ini melakukan penerapan arsitektur CNN DeepLabV3+ [10] pada dataset *Bandung Cityscapes*.

Masukan sistem segmentasi semantik adalah citra perkotaan dengan *color mode* RGB beserta label *ground truth* untuk setiap *pixel* pada citra. Data citra dilakukan *preprocessing* dan kemudian dilewatkan melalui arsitektur *convolutional neural network*. Keluaran sistem adalah berupa *segmentation map* dengan masing masing *pixel* diprediksi ke dalam salah satu *class object*.

3.2 Kerangka Pemikiran

Berikut ini adalah kerangka pemikiran dari metode yang diusulkan untuk melakukan segmentasi semantik citra perkotaan.

1. *Indicators* adalah variabel-variabel yang memengaruhi hasil keluaran dari metode yang diusulkan. Dalam penelitian ini diuji indikator sebagai berikut.

   (a) *Learning rate* adalah seberapa besar langkah koreksi yang diambil oleh CNN pada setiap iterasi. Semakin besar *learning rate*, maka CNN semakin cepat belajar, namun dengan risiko bahwa langkah yang diambil terlalu besar sehingga tidak tercapainya konvergensi.

   (b) Jumlah *filter* konvolusi menentukan banyaknya fitur yang dipelajari oleh *network*. Semakin banyak fitur yang dipelajari, maka akurasi semakin tinggi, namun berujung pada masalah *overfitting*.

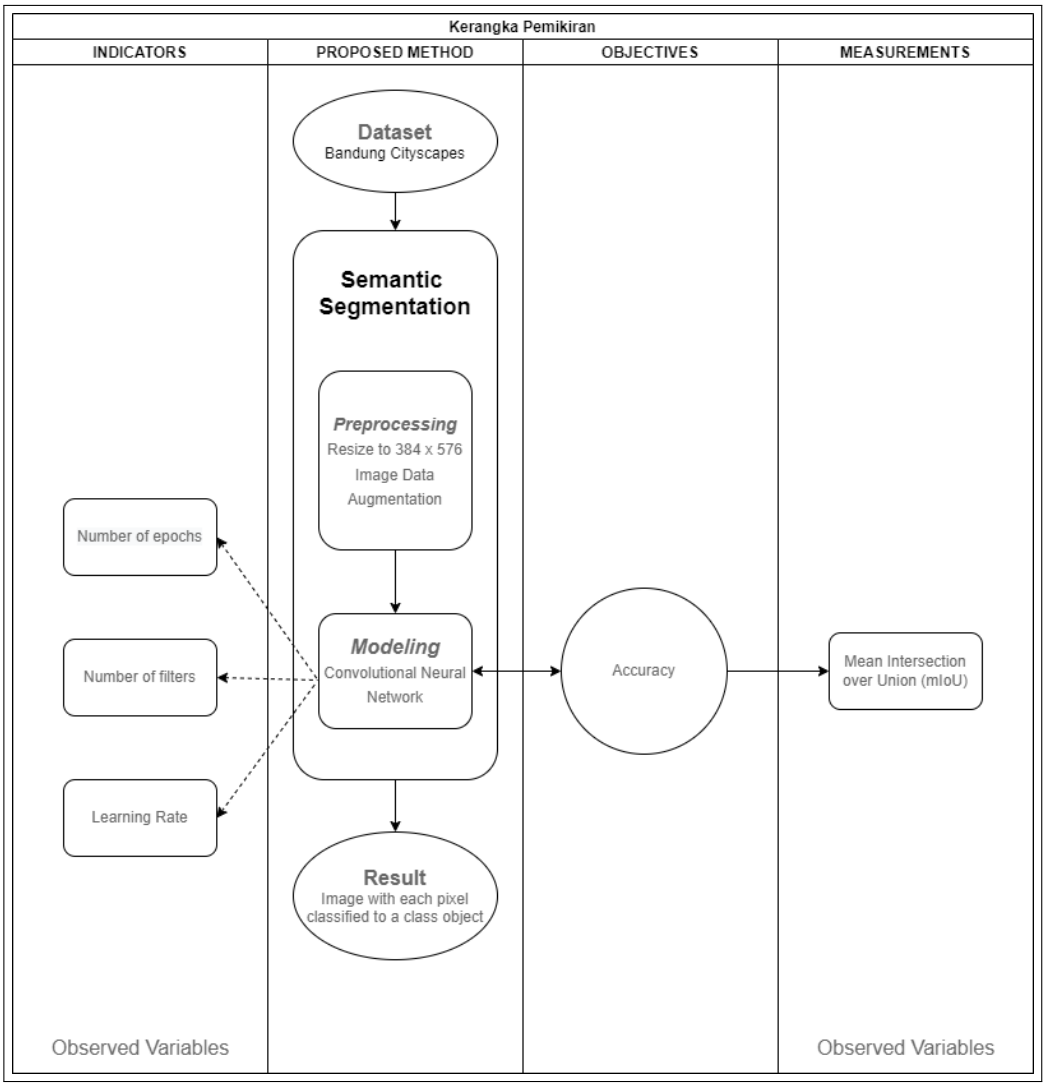
   (c) Jumlah epoch adalah *hyperparameter* yang menentukan berapa kali model melakukan pembaruan terhadap data pelatihan. Jumlah epoch

yang optimal adalah ketika model memiliki *training error* dan selisih antara *training error* dan *generalization error* paling minimal.

2. *Proposed method* adalah metode yang diusulkan untuk melakukan segmentasi semantik citra perkotaan. Pertama, citra dimuat dengan masing-masing label *ground truth* yang sesuai. Selanjutnya, dilakukan *preprocessing* dengan teknik *image data augmentation*, dan konversi citra label ke dalam bentuk *one-hot encoded*. Citra dan label kemudian dilalui ke dalam *convolutional neural network* untuk dilakukan fase *training*. *Convolutional neural network* menghasilkan keluaran berupa hasil probabilitas setiap *class* pada setiap *pixelnya*. *Class* dengan probabilitas tertinggi dianggap sebagai hasil prediksi.

3. *Objectives* adalah acuan pengukuran evaluasi model. Dalam hal ini, fokus pengukuran adalah akurasi prediksi.

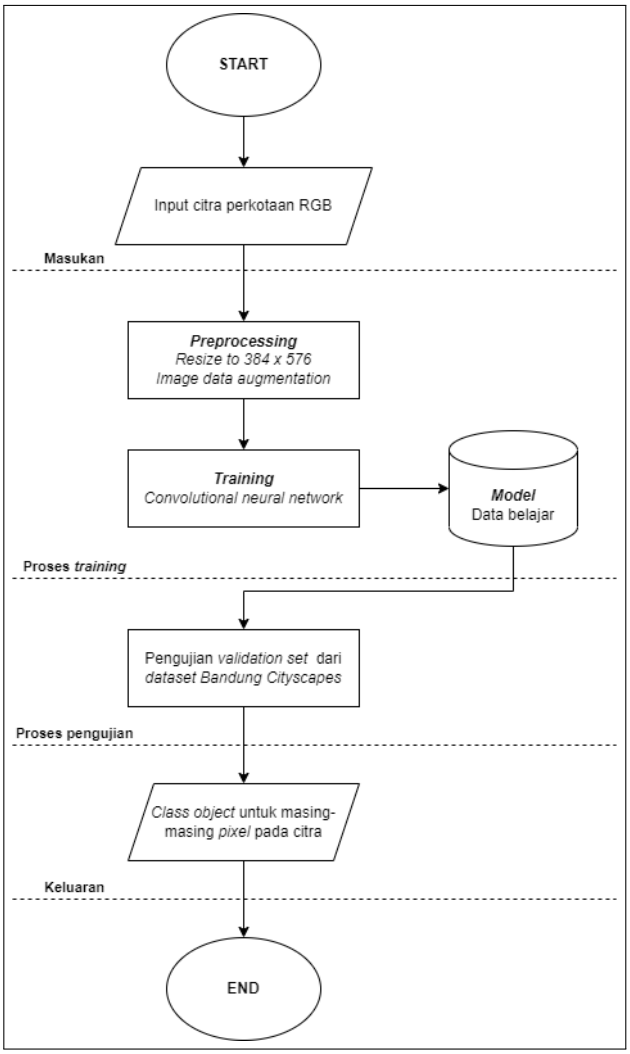
4. *Measurements* adalah metrik pengukuran yang digunakan untuk menguji model yang dibangun. Dalam penelitian ini, metrik yang digunakan adalah *mean intersection over union* (mIoU).



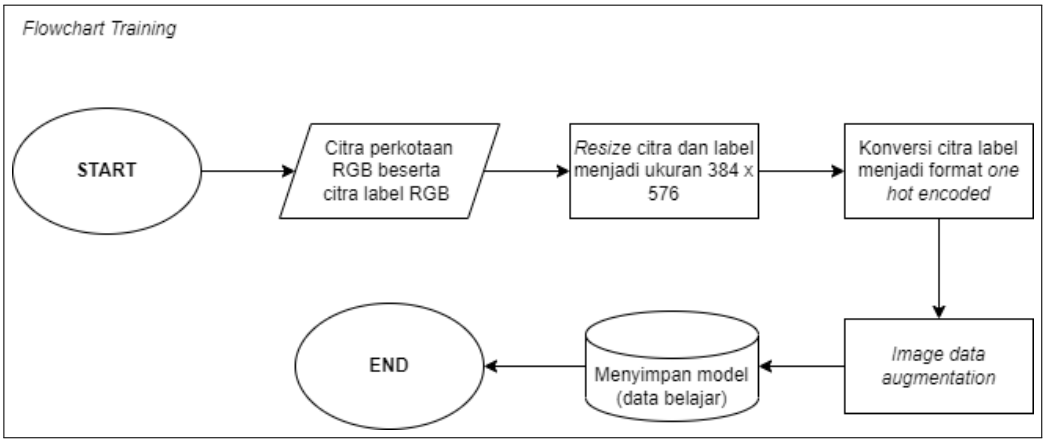
Gambar 3.1 Kerangka Pemikiran

3.3 Analisis Urutan Proses Global

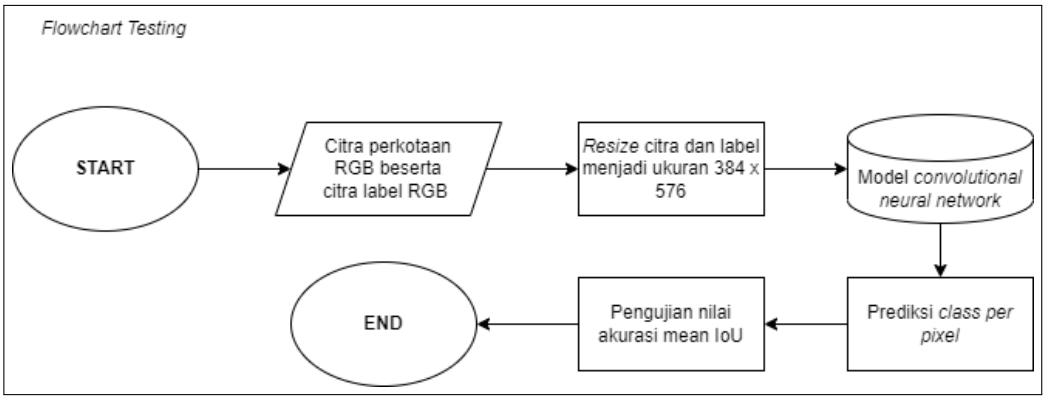
Sistem segmentasi semantik citra perkotaan yang dibangun melibatkan dua fase yaitu pelatihan (*training*) dan pengujian (*testing*). Pada proses *training*, *model convolutional neural network* memperbarui nilai-nilai pada *kernel* dan parameter. *Model* tersebut kemudian diuji dengan *validation set* dan *testing set* dan dianalisis kemiripannya dengan *ground truth*. Kedua proses pelatihan dan pengujian digambarkan dalam Gambar 3.4.



Gambar 3.2 *Flowchart* urutan Proses Global



Gambar 3.3 *Flowchart* sistem *training*



Gambar 3.4 *Flowchart* sistem *testing*

Fase *forward propagation* dan *backward propagation* dilakukan berulang kali sejumlah *epoch*, dengan sejumlah gambar yang diproses pada setiap *epoch* oleh *batch size*. Fase *forward propagation* terbagi ke dalam dua proses besar, yaitu *downsampling*, di mana input citra dikurangi resolusinya melalui operasi, citra RGB dilewatkan melalui CNN untuk memperoleh *feature map* hasil prediksi.

Algoritme 3.1 *Convolutional Neural Network: Backward Propagation* menggunakan *Adam optimizer*

1: Inisialisasi $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, dan $\varepsilon=10^{-8}$.
2: Inisialisasi $m_0 = 0$, $v_0 = 0$, dan $t = 0$.
3: Lakukan inkremen *timestep t* dengan 1.
4: Hitung gradien lokal dari $f(w)$ dengan *timestep t* menggunakan Persamaan 2.11.
5: Lakukan perhitungan momen pertama (m_t) dan kedua (v_t) dengan Persamaan 2.12.
6: Hitung koreksi bobot dan *bias* estimasi momen pertama ($\hat{m}_t$) dan momen kedua ($\hat{v}_t$) dengan Persamaan 2.13.
7: Perbarui nilai bobot dan *bias* dengan hasil perhitungan koreksi estimasi momen pertama dan momen kedua dengan Persamaan 2.14.
8: Ulangi pembaruan bobot dari langkah ketiga sampai terjadi konvergensi dan mengeluarkan parameter yang berisi bobot dan *bias* yang baru.

3.4 Analisis Manual

Pada bagian ini, dilakukan analisis tahapan proses yang dilakukan oleh sistem dengan melakukan analisis dan perhitungan secara manual.

3.4.1 *Dataset*

Dataset yang digunakan pada penelitian ini adalah dataset *Bandung Cityscapes* yang dikumpulkan dan dianotasikan oleh penulis seperti dijelaskan pada bagian 2.4.2

Berikut adalah salah satu contoh dari citra RGB.

Gambar 3.5 Citra RGB

Tabel 3.1 Contoh matriks citra

(48, 48, 44)	(29, 29, 25)	(25, 25, 21)	(29, 29, 25)	(30, 30, 26)	(36, 36, 32)	...
(47, 47, 43)	(29, 29, 25)	(27, 27, 23)	(31, 31, 27)	(33, 33, 29)	(37, 37, 33)	...
(43, 43, 39)	(26, 26, 22)	(27, 27, 23)	(33, 33, 29)	(37, 37, 33)	(37, 37, 33)	...
...	...	...	...	...	...	...

Gambar 3.6 Citra Ground Truth

Tabel 3.2 Contoh matriks *ground truth*

(128, 128, 0)	(128, 128, 0)	(128, 128, 0)	...
(128, 128, 0)	(128, 128, 0)	(128, 128, 0)	...
(128, 128, 0)	(128, 128, 0)	(128, 128, 0)	...
...	...	...	...

3.4.2 Konversi nilai *ground truth* RGB menjadi *integer*

File classdict.csv memuat sekumpulan nilai *red*, *green*, dan *blue* untuk setiap *class object* pada *ground truth label*.

Algoritme 3.2 Konversi nilai *ground truth* pada *class_dict.csv* menjadi *dictionary*

1: Inisialisasi *dictionary labels* yang diisi dengan pasangan nama *class* beserta nilai RGBnya.
2: Inisialisasi array *subset_classes* untuk memuat 9 *class object* yang diprediksi.
3: Load berkas *class_dict.csv* dan lakukan iterasi terhadap semua baris pada berkas.
4: Untuk setiap baris, masukkan nilai pasangan nama *class* dan nilai RGB ke dalam *dictionary labels*.
5: *Dictionary* yang sudah terisi ditunjukkan pada tabel 3.3.

Tabel 3.3 *Dictionary labels*

Key	Value
'Building'	(128, 0, 0)
'Car'	(64, 0, 128)
'MotorcycleScooter'	(192, 0, 192)
'Pedestrian'	(64, 64, 0)
'Road'	(128, 64, 128)
'Sidewalk'	(0, 0, 192)
'SignSymbol'	(192, 128, 128)
'Sky'	(128, 128, 128)
'Tree'	(128, 128, 0)

Algoritme 3.3 Konversi Nilai RGB Menjadi Nilai *integer*

1: Lakukan iterasi terhadap setiap citra *ground truth label*. Pada setiap iterasi, lakukan:
   - Inisialisasi *array labels* berukuran (*height* $\times$ *width*) dan isi setiap elemennya dengan panjang dari *dictionary labels*. Dalam kasus ini, setiap *pixel* diisi dengan nilai awal 9 yang dianggap sebagai *background*.
   - Perbarui nilai setiap *pixel* pada citra *ground truth* menjadi index yang sesuai pada *dictionary*. *Pixel* yang tidak termasuk ke dalam salah satu nilai pada *dictionary* tetap bernilai 9 (*background*).
   - Lakukan *one-hot encoding* untuk setiap *pixel integer*.

Tabel 3.4 Contoh matriks *ground truth* setelah dikonversi ke *integer*

0	0	0	...
1	1	1	...
2	2	2	...
...	...	...	...

Tabel 3.5 Contoh matriks *ground truth* setelah *one-hot encoding*

(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,0)	(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,0)	(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,0)	...
(0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,0)	(0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,0)	(0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,0)	...
(0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 ,0)	(0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 ,0)	(0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 ,0)	...
...	...	...	...

3.4.3 *Image Data Augmentation*

Sebelum dilewatkan melalui *convolutional neural network* citra RGB dan label dilakukan *data augmentation*, yaitu teknik memperbanyak data secara artifisial dengan melakukan operasi pada citra. Berikut ini dilakukan analisis untuk masing-masing metode, dengan mengambil contoh *channel red* pada citra RGB. Pengolahan untuk kedua *channel* lainnya (*red* dan *green*) dilakukan secara serupa.

Algoritme 3.4 *Image Data Augmentation*

1: Inisialisasi sebuah matriks citra bernilai 0 dengan ukuran sesuai citra asal.

2: Lakukan perhitungan masing-masing sel pada citra asal untuk dipetakan ke citra baru.
3: Lakukan teknik *interpolation* untuk mengisi nilai-nilai yang kosong.

Tabel 3.6 Contoh inisialisasi matriks citra

0	0	0
0	0	0
0	0	0

3.4.3.1 *Horizontal flipping*

Horizontal flipping dilakukan dengan mengalikan matriks citra dengan matriks transformasi *scale* yang ditunjukkan pada Persamaan 2.15, dengan $c_x = -1$. Perhitungan untuk sel (1,1) ditunjukkan sebagai berikut.

$$x' = -1 * 1 = -1$$
$$y' = 1 * 1 = 1$$

Tabel 3.7 Matriks citra sebelum *horizontal flipping*

1	2	3
4	5	6
7	8	9

Tabel 3.8 Matriks citra setelah *horizontal flipping*

3	2	1
6	5	4
9	8	7

3.4.3.2 *Gaussian Blur*

Gaussian blur dilakukan dengan melakukan konvolusi antara citra dengan *kernel* yang diperoleh dari Persamaan 2.16. Berikut adalah contoh *kernel Gaussian* dengan dan $\sigma^2 = 1$.

Tabel 3.9 Contoh *kernel Gaussian*

$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$
$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{8}$
$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$

Perhitungan untuk sel (1,1) dilakukan dengan Persamaan 2.1 dengan *bias* = 0 sebagai berikut.

$$
\begin{aligned}
v_k &= (w_{k_{0,0}} x_{0,0}) + (w_{k_{0,1}} x_{0,1}) + (w_{k_{0,2}} x_{0,2}) \\
&\quad + (w_{k_{1,0}} x_{1,0}) + (w_{k_{1,1}} x_{1,1}) + (w_{k_{1,2}} x_{1,2}) \\
&\quad + (w_{k_{2,0}} x_{2,0}) + (w_{k_{2,1}} x_{2,1}) + (w_{k_{2,2}} x_{2,2}) \\
&= (\frac{1}{16} * 1) + (\frac{1}{8} * 2) + (\frac{1}{16} * 3) \\
&\quad + (\frac{1}{8} * 4) + (\frac{1}{4} * 5) + (\frac{1}{8} * 6) \\
&\quad + (\frac{1}{16} * 7) + (\frac{1}{8} * 8) + (\frac{1}{16} * 9) \\
&= 5
\end{aligned}
$$

3.4.4 *Convolutional Neural Network*: *Forward Propagation*

Fase *forward propagation* memproses citra untuk memperoleh hasil prediksi. Berikut ini dianalisis fase *forward propagation* pada CNN.

3.4.4.1 *Convolutional Layer*

Citra RGB dilewatkan melalui *convolutional neural network* dalam bentuk matriks yang berisi nilai numerik seperti pada Gambar 3.5. Operasi konvolusi dilakukan antara citra dengan sebuah kernel berukuran *height* $\times$ *width* dan berjumlah *n_filters*, ditentukan tergantung arsitektur. Kernel bergerak ke samping dan ke bawah dalam ruang dua dimensi, dengan lompatan yang ditentukan oleh parameter *stride* dan jarak antar nilai pada *kernel*.

Tabel 3.10 Contoh matriks citra RGB dengan *zero-padding*

0	0	0	0	0	0	0	0
0	29	25	48	29	25	48	0
0	29	25	48	29	25	48	0
0	29	25	48	29	25	48	0
0	29	25	48	29	25	48	0
0	29	25	48	29	25	48	0

0	29	25	48	29	25	48	0
0	0	0	0	0	0	0	0

Tabel 3.11 Contoh *kernel* berukuran 3 $\times$ 3

0.02020051	0.03247714	0.04682105
0.00455129	0.00515560	-0.00222645
0.00603185	-0.44903945	-0.00425856

Berikut adalah perhitungan konvolusi untuk sel (0, 0)

$$
\begin{aligned}
v_k &= (w_{k_{0,0}} x_{0,0} + b_{0,0}) + (w_{k_{0,1}} x_{0,1} + b_{0,1}) + (w_{k_{0,2}} x_{0,2} + b_{0,2}) \\
&\quad + (w_{k_{1,0}} x_{1,0} + b_{1,0}) + (w_{k_{1,1}} x_{1,1} + b_{1,1}) + (w_{k_{1,2}} x_{1,2} + b_{1,2}) \\
&\quad + (w_{k_{2,0}} x_{2,0} + b_{2,0}) + (w_{k_{2,1}} x_{2,1} + b_{2,1}) + (w_{k_{2,2}} x_{2,2} + b_{2,2}) \\
&= (0 * 0.02020051 + 0) + (0 * 0.03247714 + 0) + (0 * 0.04682105 + 0) \\
&\quad + (0 * 0.00455129 + 0) + (29 * 0.00515560 + 0) + (25 * -0.00222645 + 0) \\
&\quad + (0 * 0.00603185 + 0) + (29 * -0.44903945 + 0) + (25 * -0.00425856 + 0) \\
&= -13.0347569
\end{aligned}
$$

Dengan *parameter stride* yang bernilai 1 $\times$ 1, maka kernel digeser sejauh 1 *pixel* untuk dilakukan komputasi selanjutnya. Setelah operasi baris pertama selesai, kernel digeser sejauh 1 *pixel* ke bawah dan dilakukan komputasi dimulai dari daerah paling kiri. Operasi ini dilakukan hingga *kernel* mencapai baris y dan kolom x, yang menandakan bahwa keseluruhan baris dan kolom pada matriks citra berhasil dilalui.

Tabel 3.12 Contoh *feature map* hasil konvolusi

-13.03476	-11.10147	-21.22991	-12.52677	-11.10147	-21.04185
-10.92239	-7.45631	-17.80819	-9.44478	-7.45631	-18.97793
-10.92239	-7.45631	-17.80819	-9.44478	-7.45631	-18.97793
-10.92239	-7.45631	-17.80819	-9.44478	-7.45631	-18.97793
-10.92239	-7.45631	-17.80819	-9.44478	-7.45631	-18.97793

2.20621	3.79916	3.71840	3.39430	3.79916	2.42517

3.4.4.2 *Batch Normalization*

Feature map hasil konvolusi kemudian dinormalisasi menggunakan *batch normalization*. Komputasi *batch normalization* dimulai dengan menghitung *mean* μ_B dan *variance* σ_B^2 sebagai berikut.

$$\begin{aligned}\mu_B &= \frac{(-13.035)+(-11.101)+(-21.230)+...+2.425}{16}\\ &= -9.05680306972\\ \sigma_B^2 &= \frac{(-13.035-(-9.057))^2+...+(2.425-(-9.057))^2}{36-1}\\ &= 45.38168177\end{aligned}$$

Kemudian, dilakukan penghitungan *batch normalization* untuk masing-masing sel pada *feature map*. Berikut adalah perhitungan *batch normalization* pada sel (0, 0) dengan inisialisasi nilai γ=1, nilai β=0, dan nilai ε=0.001.

$$\begin{aligned}\breve{x}_i &= \frac{x_i-\mu_B}{\sqrt{\sigma_B^2+\varepsilon}}\\ &= \frac{-13.035-(-9.05680306972)}{\sqrt{45.46010334+0.001}}\\ &= \frac{-3.97819693028}{6.736666963}\\ &= -0.59049287312\\ y_i &= (\breve{x}_i * \gamma)+\beta\\ &= -0.59049287312*1+0\\ &= -0.59049287312\end{aligned}$$

Tabel 3.13 Contoh *feature map* hasil *batch normalization*

-0.59049	-0.30351	-1.80699	-0.51509	-0.30351	-1.77908
-0.27693	0.23758	-1.29907	-0.05759	0.23758	-0.25131

-0.27693	0.23758	-1.29907	-0.05759	0.23758	-0.25131
-0.27693	0.23758	-1.29907	-0.05759	0.23758	-0.25131
-0.27693	0.23758	-1.29907	-0.05759	0.23758	-0.25131
1.67190	1.90836	1.89637	1.84826	1.90836	1.70440

3.4.4.3 *Activation Layer*

Nilai pada *feature map* hasil normalisasi kemudian dimasukkan ke fungsi aktivasi *Rectified Linear Unit* (ReLU) yang ditunjukkan pada persamaan berikut. Fungsi aktivasi menentukan apakah sebuah *neuron* (dalam hal ini setiap bobot pada *feature map* diaktifkan. *Feature map* hasil fungsi aktivasi ReLU ditunjukkan pada Tabel 3.14. Berikut adalah contoh perhitungan ReLU untuk sel (0, 0).

$$\begin{aligned} f(x) &= max(0,x) \\ &= \max(0,-0.59049) \\ &= 0 \end{aligned}$$

Tabel 3.14 Contoh *feature map* hasil fungsi aktivasi ReLU

0	0	0	0	0	0
0	0.23758	0	0	0.23758	0
0	0.23758	0	0	0.23758	0
0	0.23758	0	0	0.23758	0
0	0.23758	0	0	0.23758	0
1.67190	1.90836	1.89637	1.84826	1.90836	1.70440

3.4.4.4 *Skip Connection*

Operasi *convolutional layer*, *batch normalization layer*, dan *activation layer* dilakukan berulang kali tergantung pada arsitektur yang dipakai. Pada arsitektur ResNet [18], terdapat koneksi antar layer bernama *skip connection* yang diimplementasikan dengan menjumlahkan hasil *feature map* dengan *feature map* hasil konvolusi beberapa tahap sebelumnya yang diilustrasikan pada gambar 2.10. Sebagai contoh, apabila kita memiliki dua *feature map* yang digambarkan pada Tabel 3.15 dan Tabel 3.16, hasil *skip connection* ditunjukkan pada Tabel 3.17 berdasarkan persamaan 2.7.

$$x''_{1,1} = x_{1,1} + x'_{1,1}$$
$$x''_{1,1} = 0.23758 + 0.23758$$
$$x''_{1,1} = 0.47516$$

Tabel 3.15 Contoh *feature map* hasil operasi *convolution layer* pertama

0	0	0	0	0	0
0	0.23758	0	0	0.23758	0
0	0.23758	0	0	0.23758	0
0	0.23758	0	0	0.23758	0
0	0.23758	0	0	0.23758	0
1.67190	1.90836	1.89637	1.84826	1.90836	1.70440

Tabel 3.16 Contoh *feature map* hasil operasi *convolution layer* ketiga

0	0	0	0	0	0
0	0.23758	0	0	0.23758	0
0	0.23758	0	0	0.23758	0
0	0.23758	0	0	0.23758	0
0	0.23758	0	0	0.23758	0
1.67190	1.90836	1.89637	1.84826	1.90836	1.70440

Tabel 3.17 Contoh *feature map* hasil *skip connection*

0	0	0	0	0	0
0	0.47516	0	0	0.47516	0
0	0.47516	0	0	0.47516	0
0	0.47516	0	0	0.47516	0
0	0.47516	0	0	0.47516	0
3.34380	3.81671	3.79274	3.69652	3.81671	3.40880

3.4.4.5 *Pooling Layer*

Hasil *feature map* dari fungsi aktivasi kemudian menjadi input pada *pooling layer*. *Pyramid pooling module* melakukan *pooling feature map* menjadi 4 *level* yang berbeda, yaitu *level* 1, 2, 3, dan 6, di mana masing-masing level memiliki rasio ukuran 1×1, 2×2, 3×3, dan 6×6. Analisis *pooling layer* dilakukan dengan contoh *feature map* pada Tabel 3.17

$$\begin{aligned}\bar{x} &= \frac{1}{n}\sum_{i}^{n-1} x_i \\ &= \frac{1}{36} * (0+0+...+3.40880) \\ &= \frac{1}{36} * 25.67654 = 0.71324\end{aligned}$$

Tabel 3.18 Contoh *feature map* hasil *average pooling layer* level 1

0.71324

Tabel 3.19 Contoh *feature map* hasil *average pooling layer* level 2

0.10559	0.10559
1.32262	1.31915

Tabel 3.20 Contoh *feature map* hasil *average pooling layer* level 3

0.11879	0	0.11879
0.23758	0	0.23758
1.90892	1.87231	1.92517

Tabel 3.21 Contoh *feature map* hasil *average pooling layer* level 6

0	0	0	0	0	0
0	0.47516	0	0	0.47516	0
0	0.47516	0	0	0.47516	0
0	0.47516	0	0	0.47516	0

0	0.47516	0	0	0.47516	0
3.34380	3.81671	3.79274	3.69652	3.81671	3.40880

3.4.4.6 *Upsampling Layer*

Seluruh *feature map* hasil keempat level *pooling layer* kemudian dikonvolusi dengan satu buah kernel berukuran 1×1 untuk menghasilkan satu buah *feature map* per level. Masing-masing *feature map* lalu dilakukan *upsampling* menjadi ukuran saat sebelum *pooling* menggunakan teknik *bilinear interpolation*. Tabel 3.22 menunjukkan hasil *upsampling layer* pada *feature map* level 1 dengan *bilinear interpolation*. Seluruh hasil *upsampling* kemudian dilakukan *concatenation* yang diilustrasikan pada gambar 2.9.

Tabel 3.22 Contoh *feature map* level 1 hasil *upsampling*

0.71324	0.71324	0.71324	0.71324	0.71324	0.71324
0.71324	0.71324	0.71324	0.71324	0.71324	0.71324
0.71324	0.71324	0.71324	0.71324	0.71324	0.71324
0.71324	0.71324	0.71324	0.71324	0.71324	0.71324
0.71324	0.71324	0.71324	0.71324	0.71324	0.71324
0.71324	0.71324	0.71324	0.71324	0.71324	0.71324

3.4.4.7 Softmax Layer

Feature map hasil operasi sebelumnya dilakukan konvolusi akhir dengan jumlah *filter* sebanyak 12 buah dan upsampling ke dimensi semula. Pada tahap ini, terdapat 12 buah *feature map* berukuran *height* $\times$ *width* yang masing-masing berkorespondensi terhadap masing-masing *class object* pada Tabel 3.3. Setiap *pixel* pada *feature map* kemudian dimasukkan sebagai input *softmax* layer untuk dihitung probabilitas masing-masing class object. Analisis berikut dilakukan dengan asumsi bahwa ke-12 hasil keluaran *upsampling layer* sudah dihitung. Berikut adalah contoh *feature map* untuk salah satu *pixel*.

-2.9291515, -0.3962857, -4.978477, -1.015957, -4.731256, -3.1681068, -5.3845425, -4.0225053, -4.8746705, 2.4508085, 0.15910412, -0.70510334

$$\begin{aligned}\phi_i &= \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \\ &= \frac{e^{0.1055907}}{e^{0.1055907} + e^{0.34899624} + \ldots + e^{0.59240178}} \\ &= 0.00370087547\end{aligned}$$

Berikut adalah hasil keluaran dari *softmax layer*. Setiap nilai hasil probabilitas berkorespondensi terhadap *class* pada Tabel 3.3

0.00370087547, 0.04659229609, 0.0004767533, 0.02507230642, 0.00061046452, 0.00291425469, 0.00031764477, 0.00124012924, 0.00052890333, 0.8031402912, 0.08119267449, 0.0342134064

Pada *feature map* hasil prediksi, didapatkan bahwa probabilitas tertinggi adalah 0.8031402912 yang terletak pada indeks ke-9. Tabel 3.3 menunjukkan bahwa *class object* pada indeks ke-9 yang menjadi hasil prediksi adalah *Sky*. Nilai-nilai di atas belum optimal dan masih memiliki tingkat kesalahan yang relatif tinggi, sehingga perlu dibandingkan dengan nilai *ground truth* untuk dilakukan perhitungan *loss function* pada tahap *backward propagation*.

3.4.5 *Convolutional Neural Network*: *Backward Propagation*

Operasi *backward propagation* merupakan cara untuk memperbarui nilai-nilai bobot pada *convolutional neural network*. Pada penelitian ini, digunakan metode optimisasi *adaptive momentum estimate* (adam) dengan analisis perhitungan sebagai berikut.

3.4.5.1 Inisialisasi nilai *hyperparameter*

Lakukan inisialisasi nilai parameter α = 0.001, β_1 = 0.9, β_2 = 0.999, $\varepsilon = 10^8$. Nilai-nilai tersebut merupakan nilai yang disarankan [15] dan bekerja baik di banyak algoritme.

3.4.5.2 Perhitungan *categorical cross-entropy loss* dari iterasi sebelumnya

Perhitungan *loss function* dilakukan untuk menentukan *error* atau kesalahan antara hasil prediksi dengan *ground truth*.

$$
\begin{aligned}
FL(p_t) &= -(1-p_t)^{\gamma}\log p_t \\
FL(p_t) &= -(1-0.00370087547)^2 * \log 0.00370087547 \\
&= 2.42269612581
\end{aligned}
$$

Pada analisis ini digunakan asumsi bahwa nilai gradien g_t bernilai 2.42269612581.

3.4.5.3 Perhitungan Nilai Momen Pertama

Nilai momen pertama dilakukan dengan *timestep* (t) = 1, yang menandakan bahwa komputasi adalah untuk iterasi ke-1.

$$
\begin{aligned}
m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \\
&= 0.9 * 0 + (1-0.9) * 2.43169552815 \\
&= 0 + 0.1 * 2.43169552815 \\
&= 0.242269612581
\end{aligned}
$$

3.4.5.4 Perhitungan Nilai Momen Kedua

$$
\begin{aligned}
\upsilon_t &= \beta_2 \cdot \upsilon_{t-1} + (1-\beta_2) \cdot g_t^2 \\
&= 0.999 * 0 + (1-0.999) * (2.42269612581)^2 \\
&= 0 + 0.001 * (2.42269612581)^2 \\
&= 0.00586945651801
\end{aligned}
$$

3.4.5.5 Perhitungan Nilai Koreksi Bias Momen Pertama Dan Kedua

Nilai koreksi bias dihitung dengan menggunakan persamaan 2.13.

$$\begin{aligned}\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ &= \frac{0.242269612581}{1-0.9} \\ &= 2.42269612581 \\ \hat{\upsilon}_t &= \frac{\upsilon_t}{1-\beta_2^t} \\ &= \frac{0.00586945651801}{1-0.999} \\ &= 5.86945651801\end{aligned}$$

Bobot awal yang diasumsikan bernilai 0.7493028 kemudian diperbarui dengan menggunakan Persamaan 2.14. *Learning rate* α pada perhitungan ini diinisialisasi dengan nilai 0.001 atau 10^{-3}.

3.4.5.6 Pembaruan Bobot

$$\begin{aligned}w_t &= w_{t-1} - \alpha\frac{\hat{m}_t}{\sqrt{\hat{\upsilon}_t}+\varepsilon} \\ &= 0.7493028 - 10^{-3} * \frac{2.42269612581}{\sqrt{5.86945651801}+10^{-8}} \\ &= 0.7493028 - 10^{-3} * \frac{2.42269612581}{2.42269612581+10^{-8}} \\ &= 0.7493028 - 10^{-3} * \frac{2.42269612581}{2.42269613581} \\ &= 0.7493028 - 10^{-3} * 0.99999999587 \\ &= 0.7493028 - 0.00099999999 \\ &= 0.74830280001\end{aligned}$$

Pembaruan tersebut dilakukan untuk setiap *neuron* dan hasilnya disimpan untuk fase pengujian.

3.4.6 Pengujian Hasil Prediksi

Hasil prediksi dari model yang telah dilatih kemudian dibandingkan dengan label *ground truth*. Contoh skenario pengujian adalah sebagai berikut. Citra RGB berukuran 6×6 sebagai berikut diprediksi oleh CNN dan menghasilkan keluaran **A** sebagai berikut.

Tabel 3.23 Contoh matriks citra RGB hasil prediksi

0	1	1	1	2	1
0	1	1	1	2	1
0	1	1	1	2	1
0	1	1	1	2	1
0	1	1	1	2	1
0	1	1	1	2	1

Tabel 3.24 Contoh matriks citra *ground truth*

0	1	2	2	1	0
0	1	2	2	1	0
0	1	2	2	1	0
0	1	2	2	1	0
0	1	2	2	1	0
0	1	2	2	1	0

Maka, perhitungan IoU untuk masing-masing *class* dan rata-rata dihitung berdasarkan Persamaan 2.17 sebagai berikut.

$$IoU_i = \frac{n_{ii}}{\sum_{j=1}^{k} n_{ij} + \sum_{j=1}^{k} n_{ji} - n_{ii}}$$

$$IoU_0 = \frac{6}{6+6-6} = \frac{1}{2} = 0.5$$

$$IoU_1 = \frac{6}{6+18-6} = \frac{1}{3} = 0.333$$

$$IoU_2 = \frac{0}{12+6-6} = \frac{0}{12} = 0$$

$$mIoU = \frac{1}{k}\sum_{i=1}^{k} IoU_i$$

$$mIoU = \frac{1}{3}(0.5+0.333+0) = \frac{1}{3}(0.833) = 0.27766666666$$