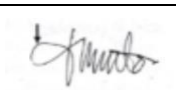
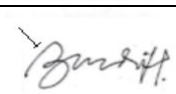
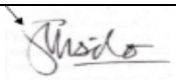
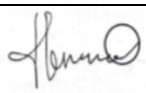
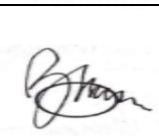
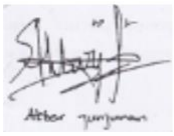
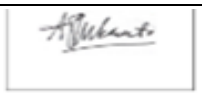
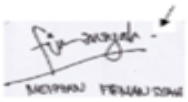
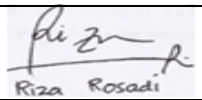
BAB 2
LANDASAN TEORI

2.1 Tulisan Tangan Berbentuk Tanda Tangan dalam Grafologi

Dalam grafologi tulisan tangan berbentuk tanda tangan dapat melakukan beberapa prediksi kepribadian, pada penelitian ini menggunakan 7 fitur yaitu awal kurval engkung mundur, awal kurval engkung tajam, awal kurval engkung lembut, coretan akhir menaik, coretan akhir menurun, adanya coretan ditengah, adanya garis bawah, berikut adalah tabel dari prediksi kepribadian berdasarkan fitur - fitur tanda tangan [4].

Tabel 2.1 Prediksi Kepribadian Berdasarkan Fitur - Fitur Tanda Tangan

No	Gambar	Fitur	Kepribadian
1		Cangkang Lengkung tertutup	Ketakutan berlebihan, introvert, tidak mempedulikan sekitar, tidak suka bergaul dan bekerja sama.
2		Awal Kurva Lengkung Mundur	Nyaman akan masa lalu.
		Awal Kurva Lengkung Tajam	Mampu memformulasi pikiran secara tajam.
		Awal Kurva Lengkung Lembut	Hati-hati,ramah, diplomatis.
3		Coretan Akhir Menaik	Terbuka, pandangan ke depan, ingin maju dan percaya diri.
		Coretan Akhir Menurun	Kurang semangat, berfikir realistis, kurang percaya diri, mudah putus asa.
4		Adanya Coretan Tengah	Mudah percaya diri dan mudah depresi.
5		Adanya garis bawah	Membutuhkan dukungan membuat keputusan, serta

			memiliki keandalan dalam memimpin.
6		Margin Ekstrim Cenderung ke kanan	Ceroboh, kurang perhatian.
		Margin Ekstrim Cenderung ke kiri	Takut gagal, takut pada orang lain, kurang percaya diri, pesimis.
		Margin Ekstrim Cenderung ke atas	Respek pada diri sendiri, mencerminkan pribadi bahagia.
		Margin Ekstrim Cenderung ke bawah	Depresi, pemalu, merasa asing.
7		Adanya Struktur titik	Pendirian stabil, memiliki rasa curiga, selalu menjaga jarak tidak mudah percaya.
8		Adanya tanda tangan terpisah	Memiliki pengalaman tidak mengenangkan di masa lalu.
9		Adanya garis terpisah	Membatasi keinginannya, tidak berani mengambil resikosering patah semangat dan ragu

2.2 Pengolahan Citra Digital

Menurut Efford, Secara umum, pengolahan citra digital dapat diartikan sebagai teknik yang keberadaannya untuk memanipulasi dan memodifikasi citra dengan berbagai cara. Sedangkan menurut Jain, pengolahan citra digital dapat dinyatakan sebagai pemrosesan gambar berdimensi-dua melalui komputer digital. Adapula prinsip-prinsip dasar pada pengolahan citra, seperti peningkatan kecerahan dan kontras, penghilangan derau pada citra, dan pencarian bentuk objek [11].

Citra dibagi lagi menjadi tiga jenis citra, umumnya ada tiga jenis citra yang sering digunakan pada pengolahan citra digital. Ketiga jenis citra tersebut adalah citra berwarna, citra berskala keabuan, dan citra biner. Ketiga jenis citra tersebut akan dijelaskan lebih detail sebagai berikut.

1. Citra berwarna

   Citra berwarna atau biasa disebut juga sebagai citra RGB, citra ini memiliki tiga komponen warna. Warna-warna tersebut adalah Red (merah), Green (hijau), dan Blue (biru). Dimana masing-masing warnanya menggunakan nilai delapan bit yang berarti nilainya berkisar antara 0 sampai 255. Sehingga nilai yang dapat dihasilkan oleh citra berwarna adalah 255x255x255 atau sebanyak 16.581.375 warna [11].
2. Citra berskala keabuan

   Citra berskala keabuan, memiiki nilai yang serupa dengan citra berwarna, hanya saja pada citra berskala keabuan nilai yang ada menyatakan nilai keabuannya, dimana nilai 0 yang berarti hitam, dan nilai 255 yang berarti putih [11].
3. Citra biner

   Citra biner, citra jenis ini banyak diguankan dalam pemrosesan citra, misalnya untuk memperolah tepi bentuk suatu objek. Citra ini hanya memiliki dua jenis nilai, yaitu nilai 0 yang berarti hitam dan nilai 1 yang berarti putih [11].

Contoh dari metode-metode yang ada pada pengolahan citra digital akan dibahas pada sub-bab ini.

2.2.2 Grayscale

Citra grayscale atau juga citra berskalah keabuan. Gunanya adalah merubah citra berwarna menjadi citra berskala keabuan. Secara umum perubahan citra berwarna menjadi citra keabuan dengan menggunakan rumus berikut.

$$GS_{x,y} = a * R + b * G + c * B$$

Dimana (R) adalah nilai pada warna merah, (G) adalah nilai pada warna hijau, dan (B) adalah ilai pada warna biru. Sementara untuk (a,b,c) adalah nilai tetap, nilai tetap yang biasa dipakai untuk (a,b,c) adalah.

$$a = 0.2989$$

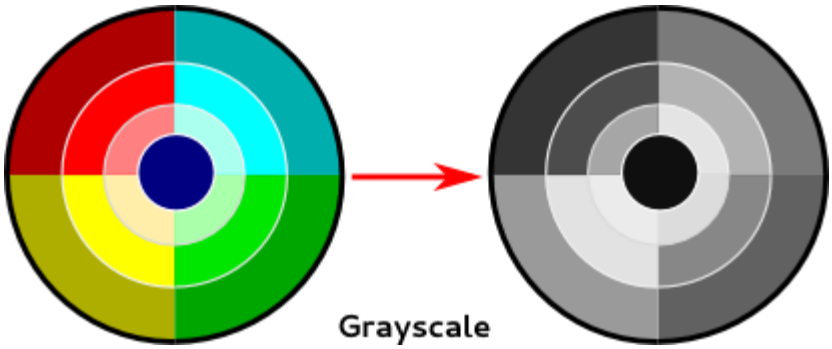
$$b = 0.5870$$

$$c = 0.1141$$

Sehingga didapatkan persamaan (2.1) untuk merubah citra berwarna menjadi citra berskala keabuan adalah sebagai berikut [11].

$$GS_{x,y} = 0.2989 * R + 0.5870 * G + 0.1141 * B \quad (2.1)$$

Contoh hasil dari teknik grayscale sendiri telah tergambarkan pada Gambar 2.1.



Gambar 2.1 Contoh Hasil *Grayscale*

2.2.3 Deteksi Tepi Canny

Deteksi tepi berfungsi untuk memperoleh tepi objek. Operator Canny dikemukakan oleh John F. Canny pada tahun 1986, terkenal sebagai operator deteksi tepi yang paling optimal. Algoritma ini memberikan tingkat kesalahan yang rendah, melokalisasi titik-titik tepi (jarak piksel-piksel tepi yang ditemukan deteksi dan tepi yang sesungguhnya sangat pendek), dan hanya memberikan satu tanggapan untuk satu tepi [12]. Terdapat langkah-langkah yang digunakan untuk melakukan deteksi tepi canny. Langkah-langkah tersebut sebagai berikut.

1. Image Smoothing

   Merupakan sebuah proses untuk mengaburkan gambar untuk menghilangkan derau. Pada proses ini dapat dilakukan dengan menggunakan filter gausian dapat dilihat pada persamaan (2.2) sebagai berikut [12].

$$\frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 5 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \quad (2.2)$$

2. Finding Gradient

   Merupakan sebuah proses untuk mendapatkan kekuatan tepi. Tepian harus ditandai pada gambar memiliki gradien yang besar. Digunakan operrator sobel dan dilakukan pencarian secara horizontal dan vertikal dapat dilihat pada persamaan (2.3) sebagai berikut.

$MH_{x,y}$ = | -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

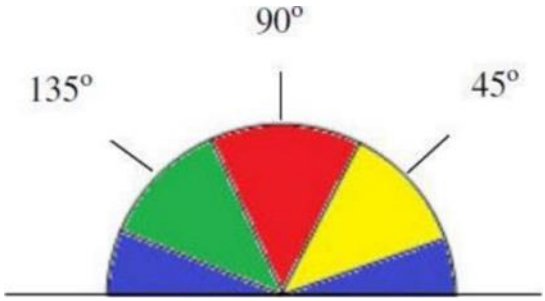
$MV_{x,y}$ = | 1 | 2 | 1 |
| 0 | 0 | 0 |
| - 1 | -2 | -1 | (2.3)

Hasil dari Gausian filter tahap pertama deteksi tepi canny dikonvolusi kembali dengan filter $MH_{x,y}$ sehingga menghasilkan nilai $GH_{x,y}$, sedangkan untuk mengetahui nilai $GV_{x,y}$ adalah hasil konvolusi gausifan filter dikonvolusi kembali dengan filter $MV_{x,y}$. Magnitudo gradien (juga dikenal sebagai kekuatan tepi) dapat ditentukan sebagai jarak Euclidean yang diukur mengukur dengan menerapkan hukum pythagoras seperti yang ditunjukkan dalam Persamaan (2.4).

$$G_{x,y} = \sqrt{GH_{x,y}{}^2 + GV_{x,y}{}^2} \quad (2.4)$$

Selanjutnya menentukan tepian yang sebenarnya ini, arah tepian harus ditentukan dan disimpan seperti ditunjukkan dalam Persamaan (2.5) [12].

$$\theta = arctan\left(\frac{GH_{x,y}}{GV_{x,y}}\right) \quad (2.5)$$



Gambar 2.2 Area Untuk Konversi Arah Tepi

Aturan konversi yang berlaku sebagai berikut.

a. Semua arah tepi yang berkisar antara 0 dan 22,5 serta 157,5 dan 180 derajat (warna biru) diubah menjadi 0 derajat.
b. Semua arah tepi yang berkisar antara 22,5 dan 67,5 derajat (warna kuning) diubah menjadi 45 derajat.
c. Semua arah tepi yang berkisar antara 67,5 dan 112,5 derajat (warna merah) diubah menjadi 90 derajat.

d. Semua arah tepi yang berkisar antara 112,5 dan 157,5 derajat (warna hijau) diubah menjadi 135 derajat. [12].

3. *Non-maximum Suppresion*

   Merupakan proses yang digunakan untuk melakukan penghilangan non-maximum. Penghilangan non-maximum dilakukan di sepanjang tepi pada arah tepi dan menghilangkan piksel-piksel (piksel diatur menjadi 0) yang tidak dianggap sebagai tepi. [12].

4. *Edge Tracking by Hysteresis*

   Merupakan proses tepian final ditentukan dengan menekan semua sisi yang tidak terhubung dengan tepian yang sangat kuat. Pengambangan hysteresis dilakukan dengan melibatkan dua ambang T_1 dan T_2. Nilai yang kurang dari T_1 akan diubah menjadi warna hitam (nilai 0) dan nilai yang lebih besar dari T_2 akan diubah menjadi putih (nilai 255). Sementara nilai yang lebih atau sama dengan T_1 tetapi kurang dari T_2 akan diberi nilai 128 dan yang menyatakan nilai abu-abu atau belum jelas, akan dijadikan 0 atau 255, apabila kondisi seperti itu terpenuhi, angka 128 diubah menjadi 255 [12].

5. *Thresholding*

   Mengubah citra menjadi citra biner dengan menggunakan rumus persamaan berikut [12].

$$T_{x,y} = \begin{Bmatrix} 1\ if\ G_{x,y} \geq 128 \\ 0\ if\ G_{x,y} \leq 128 \end{Bmatrix} \tag{2.6}$$

2.2.4 Segmentasi Objek

Segmentasi adalah sebuah teknik untuk membagi suatu citra menjadi beberapa daerah [13] atau bisa juga digunakan untuk memisahkan antara citra objek yang akan digunakan dari background. Pada penelitian ini akan digunakan segmentasi citra dengan menggunakan metode *Connected Component Labeling*. Metode ini adalah metode yang paling umum digunakan pada pengolahan citra digital [14]. Metode ini memiliki sifat ketetanggaan, dimana ketetanggaan yang dimaksud adalah 1 jarak unit dari piksel yang ditandai. Ada 2 jenis ketetanggan pada citra 2 dimensi [15], yaitu :

1. *4-Connectivity Neighbors*

   Ketetanggaan di sini adalah nilai-nilai piksel yang berada pada garis horizontal dan garis vertical. Contoh dari ketetanggaan di sini telah digambarkan pada Gambar 2.3.

	P(x, y-1)	
P (x-1, y)	P (x,y)	P(x+1, y)
	P(x, y+1)	

Gambar 2.3 *4-Connectivity Neighbors*

2. *8-Connectivity Neighbors*

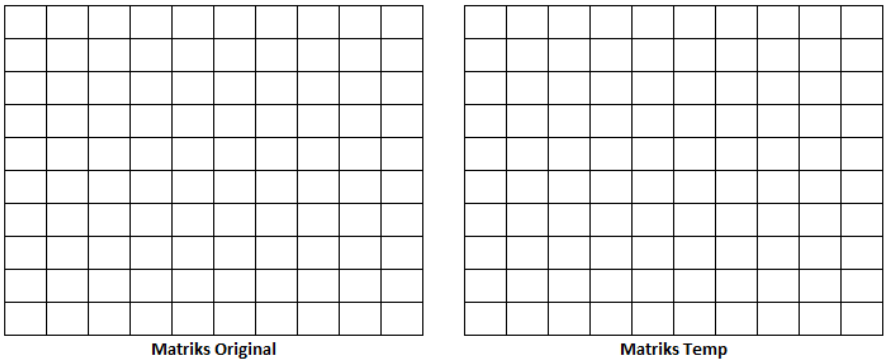
   Ketetanggaan di sini adalah nilai-nilai piksel yang berada pada garis horizontal, garis vertical, dan garis diagonal. Contoh dari ketetanggaan di sini telah tergambarkan pada Gambar 2.4.

P (x-1, y-1)	P(x, y-1)	P (x+1, y-1)
P (x-1, y)	P (x,y)	P(x+1, y)
P (x-1, y+1)	P(x, y+1)	P (x+1, y+1)

Gambar 2.4 *8-Connectivity Neighbors*

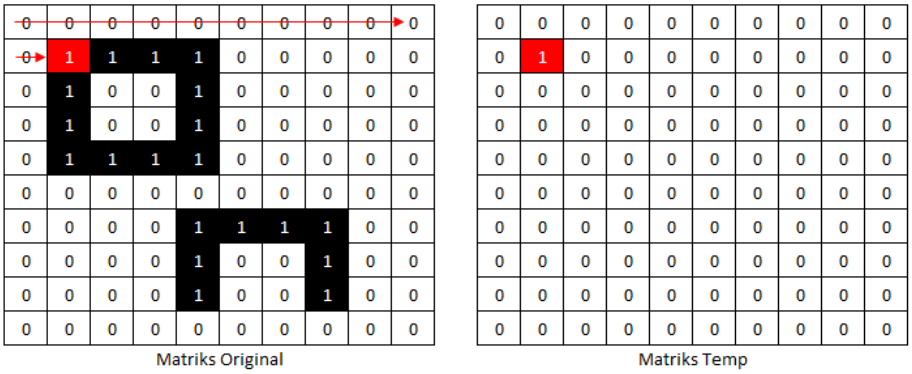
Berikut adalah langkah-langkah dari metode *Connected Component Labeling.*

a. Buat dua buah matriks, matriks pertama (*original*) akan menyimpan citra yang akan diolah, matriks kedua (temp) akan menyimpan nilai-nilai dari objek pada citra. Contoh matriks telah digambarkan pada Gambar 2.5.



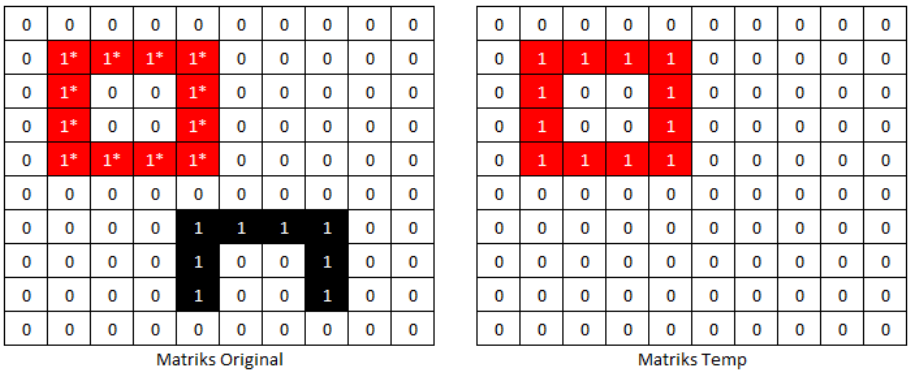
Gambar 2.5 Matriks *Original* dan Matrix Temp

b. Lakukan *scanning* pada setiap piksel dari citra, mulai dari atas kiri berjalan ke kanan lalu mulai dari baris baru pada posisi kiri kembali. Jika saat proses scanning ditemukan piksel dengan nilai 1 (atau 255) maka lakukan pelabelan terhadap piksel tersebut, salin posisi piksel tersebut ke dalam matriks temp dan beri label 1 sebagai tanda bahwa posisi tersebut adalah objek ke-1. Contoh tahapan ini telah tergambarkan pada Gambar 2.6.



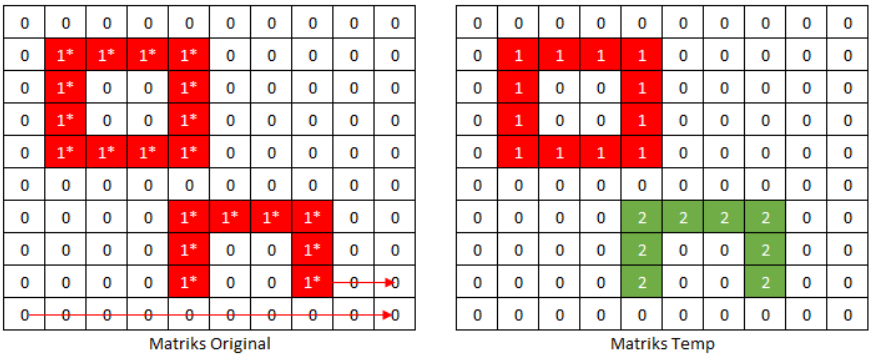
Gambar 2.6 Pergeseran dari *Scanning*

c. Kemudian tandai posisi piksel tersebut agar tidak terbaca kembali. Kemudian lakukan *8-Connectivity Neighbors* terus menerus hingga 8 tetangga tersebut tidak lagi bernilai 1 (atau 255) yang belum bertanda telah dibaca. Sehingga dapat dilihat hasil dari satu objek pertama pada Gambar 2.7.



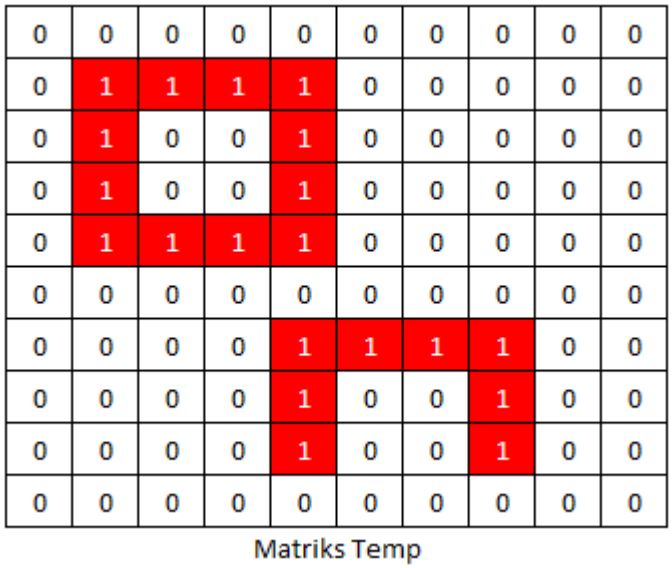
Gambar 2.7 Objek pertama pada Segmentasi

d. Lakukan kembali tahap 2 sampai dengan 3 terus menerus hingga posisi piksel terakhir telah terbaca. Contoh tahapan terakhir ini tergambarkan pada Gambar 2.8.



Gambar 2.8 Seluruh objek yang tersegmentasi

e. Tahapan ini adalah tahapan tambahan sesuai dengan kebutuhan, dimana seluruh objek akan digabungkan menjadi 1. Sehingga seluruh objek pada matriks temp dengan label penanda objek selain dari 1 akan dirubah menjadi label objek ke-1. Contoh pada tahapan ini telah digambarkan pada Gambar 2.9.



Gambar 2.9 Penggabungan seluruh objek pada segmentasi

Setelah proses *Connected Component Labeling* akan dilakukan pencarian batas, dimana batasan-batasan tersebut digunakan untuk menentukan dimanakan posisi piksel paling atas, paling kanan, paling kiri, dan paling bawah. Untuk melakukan pencarian batas tersebut, digunakanlah tahapan-tahapan sebagai berikut.

1) Pertama akan dilakukan inisialisasi awal terhadap batasan yang akan digunakan sebagai berikut.

$$batas[0] = temp[0][x] \; //batas\ atas$$
$$batas[1] = temp[0][y] \; //batas\ kanan$$
$$batas[2] = temp[0][x] \; //batas\ bawah$$
$$batas[3] = temp[0][y] \; //batas\ kiri$$

2) Kemudian akan dilakukan pencarian batasan terhadap piksel dengan persamaan 2.7 sebagai berikut.

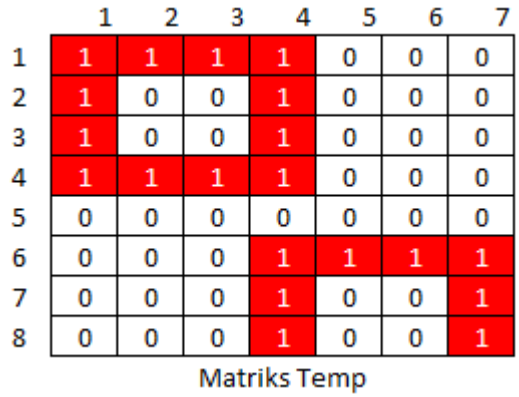
$$batas[0] = \begin{cases} temp[i][x] & jika\ temp[i][x] < batas[0] \\ batas[0] & jika\ tidak \end{cases}$$
$$batas[1] = \begin{cases} temp[i][y] & jika\ temp[i][y] > batas[1] \\ batas[1] & jika\ tidak \end{cases} \quad (2.7)$$
$$batas[2] = \begin{cases} temp[i][x] & jika\ temp[i][x] > batas[2] \\ batas[2] & jika\ tidak \end{cases}$$

$$batas[3] = \begin{cases} temp[i][y] & jika\ temp[i][y] < batas[3] \\ batas[3] & jika\ tidak \end{cases}$$

Pencarian batasan dengan persamaan 2.6 akan diulang terus menerus hingga seluruh nilai pada matriks temp telah dibaca. Apabila digunakan matriks temp pada contoh sebelumnya, maka akan didapatkan batasan-batasan sebagai berikut: batas atas = 1 (pada posisi piksel x); batas kanan = 8 (pada posisi piksel y); batas bawah = 9 (pada posisi piksel x); dan batas kiri = 2 (pada posisi piksel y); batasan dari contoh tersebut telah digambarkan pada Gambar 2.10.

Gambar 2.10 Hasil pencarian batas pada matriks temp

3) Selanjutnya, akan dilakukan pemotongan matriks, dimana pemotongan disini menggunakan batasan-batasan yang telah didapatkan sebelumnya. Sehingga nilai koordinat pun akan berubah akibat perpotongan. Dengan digunakannya contoh pada matriks temp sebelumnya, akan didapatkan hasil seperti pada Gambar 2.11.



Gambar 2.11 Hasil pemotongan matriks temp

2.2.5 Resize

Resize atau penskalaan adalah sebuah operasi geometri yang digunakan untuk memperbesar atau memperkecil ukuran dari sebuah citra sesuai dengan ukuran yang dibutuhkan [13]. Untuk melakukan penskalaan pada penelitian ini tidak digunakan metode khusus. Untuk mendapatkan ukuran yang diinginkan maka akan dilakukan perbandingan ukuran antara citra segmentasi dengan target ukuran citra dengan menggunakan persamaan (2.8) untuk mendapatkan nilai dari titik x dan persamaan (2.9) untuk mendapatkan nilai dari titik y. Dimana kedua persamaan tersebut adalah sebagai berikut.

$$xbaru = \frac{pb * pp}{pa} \tag{2.8}$$

Keterangan:

$xbaru$ = Posisi x baru
pb = Ukuran panjang dari matriks baru.
pp = Posisi piksel x lama.
pa = Ukuran panjang dari matriks lama.

$$ybaru = \frac{lb * pp}{la} \tag{2.9}$$

Keterangan:

$ybaru$ = Posisi y baru
lb = Ukuran lebar dari matriks baru.
pp = Posisi piksel y lama.
la = Ukuran lebar dari matriks lama.

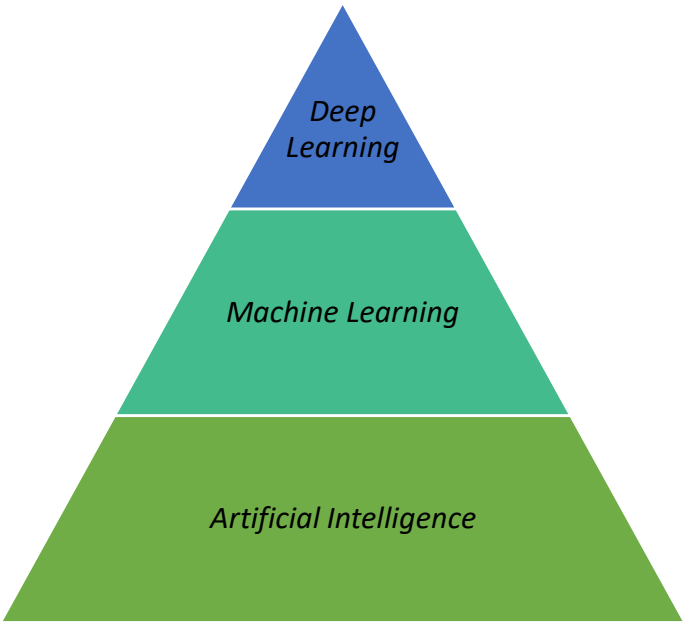
2.2.6 Segmentasi Vertikal dan Horizontal

Segmentasi vertikal dan horizontal adalah sebuah teknik untuk membagi suatu citra menjadi dua daerah yaitu vertikal dan horizontal. Daerah tersebut digunakan untuk mengenali sebuah objek pada citra dan membedakan antar fitur citra. Segmentasi yang telah dilakukan akan digunakan sebagai nilai masukan untuk membedakan antara objek satu dengan lainnya pada tahapan *processing*. Pada tahapan ini citra hasil dari *resize* akan dilanjutkan ketahapan segmentasi vertikal dan horizontal. Daerah vertikal diproses dengan membagi tiga bagian wilayah citra secara vertikal dan bagian yang akan di ambil dari bagian itu hanya dua yaitu kiri, dan kanan digunakan untuk membedakan wilayah fitur awal kurva dan coretan akhir. Sedangkan daerah horizontal diposes dengan membagi tiga bagian wilayah secara horizontal dan bagian yang akan di ambil dari bagian itu hanya dua yaitu tengah dan bawah digunakan untuk membedakan wilayah fitur garis tengah dan garis bawah.

2.3 *Deep Learning*

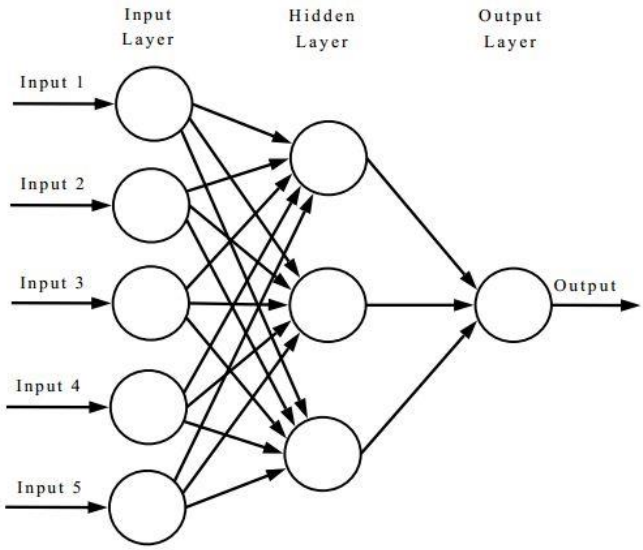
Deep Learning adalah bagian dari *Arificial Neural Network. Google trends* mengutarakan bahwa *deep learning* akan mulai berkembang pada tahun "2014". Penelitian yang menggunakan *deep learning* sendiri sudah banyak digunakan, diantaranya adalah pada *Medical Imaging*, *Bioinformatics*, *Speech Recognition*, *Text Mining* dan *Natural Language Processing* (NLP) [16].

Menurut Goeffrey *deep learning* diartikan sebagai *"Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algoriithms. Learning can be supervised, semi-seupervised or unsupervised"*. *Deep learning* adalah salah satu cabang *machine learning* yang menggunakan *Deep Neural Network* untuk menyelesaikan permasalahan pada domain *machine learning*. Berikut ilustrasi dari *deep learning* yang tergambarkan pada Gambar 2.12.



Gambar 2.12 *Deep Learning Pyramid*

Pada *network* model dari *deep learning* terdapat 3 layer yang berhubungan, ya itu *input layer*, *hidden layer*, dan *output layer*. Yang mana model tersebut sudah digambarkan pada Gamabr 2.13. Pada diagram model dapat dilihat jumlah hidden layer hanya satu buah, padahal kenyataannya bisa sangat banyak hidden layer yang digunakan [6].

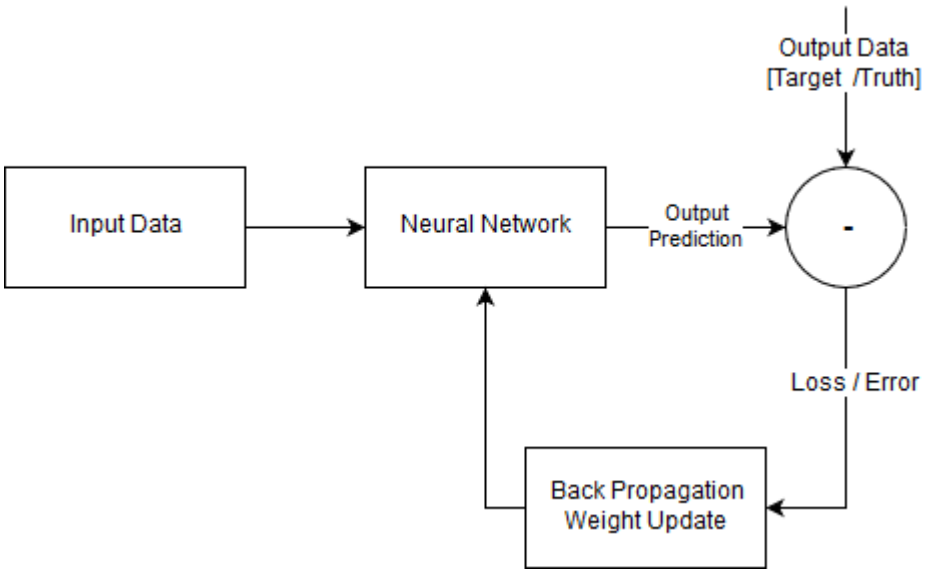


Gamabr 2.13 Diagram *network deep learning*

Deep learning sudah banyak dikembangkan ke berbagai model yang bermacam-macam. Sehingga algortma yang ada pada *deep* learning pun sangat bervariasi. Berikut adalah beberapa model yang sudah dikembangkan pada deep learning adalah:

1. *Recurrent Neural Networks* (RNN)
2. LSTM/GRU *Networks*
3. *Convolutional Neural Networks* (CNN)
4. *Deep Belief Networks* (DBN)
5. *Deep Stacking Networks* (DSN)

Proses *learning* pada *deep learning* pada dasarnya adalah data yang dimasukkan kemudian memasuki tahap *neural network* kemudian prediksi keluarannya akan diambil, lalu prediksi keluarannya akan dibandingkan dengan target, apabila pada prediksi keluaran masih terdapat error, maka akan masuk ke tahap *back propagation*, lalu kemudian dikembalikan lagi ke tahap *neural network*. Agar lebih memudahkan proses pemahaman *learning* pada *deep learning*, prosesnya telah tergambarkan pada Gambar 2.14.

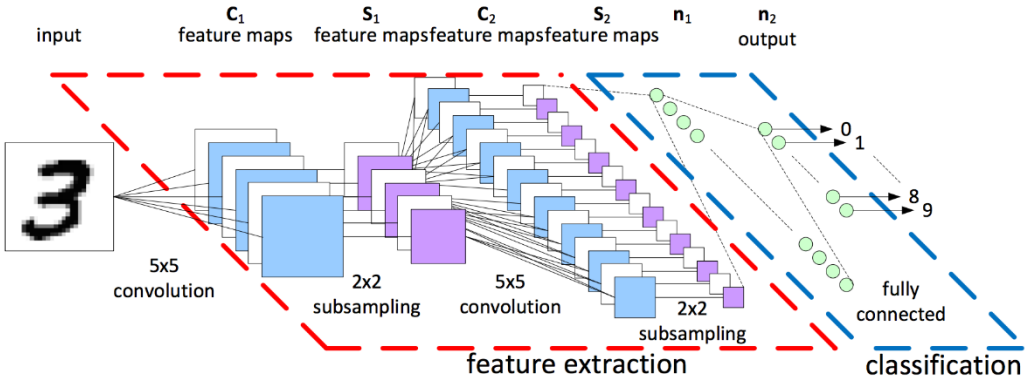


Gambar 2.14 Proses *learning* pada *neural network*

2.3.1 Metode *Convolutional Neural Network*

Convolutional neural Network (CNN) merupakan salah satu dari jenis model yang ada pada *deep learning*. Model jenis ini banyak digunakan untuk keperluan analisis citra. Secara prinsip, CNN meniru visual cortex pada mamalia. CNN memiliki neuron-neuron yang disusun secara tiga dimensi, yaitu panjang, lebar, dan tinggi. CNN sendiri terdiri dari neuron-neuron yang memiliki *weight*, bias, dan *activation function*. Sehingga membuat CNN sangat efektif dan efisien untuk menganalisis gambar atau citra digital [6].

Algoritma CNN adalah sebuah MLP (*Multi Layer Perceptron*) yang didisain secara khusus untuk mengidentifikasi citra digital. Cara kerja dari CNN mengikuti cara kerja otak manusia saat mengenali objek yang dilihat. Pada dasarnya CNN sendiri memiliki beberapa jenis layer yang dapat digunakan. Salah satu contoh penggambaran penggunaan layer pada CNN sudah digambarkan pada Gambar 2.15.



Gambar 2.15 Arsitektur CNN

Berikut adalah penjelasan masing-masing layer yang ada menurut Zhifei Zhang [17].

2.3.1.1 Convolutional Layer

Pada layer ini akan dilakukan perhitungan "dot" antara dua nilai, yaitu nilai input dan nilai filter yang digunakan, dimana filter yang digunakan sendiri berukuran $k * k$. Untuk melakukan perhitungan "dot" akan digunakan persamaan (2.10).

$$C_{j,x,y} = f\left(I \otimes K_{u,v} + B_i\right), \tag{2.10}$$

$$dimana \otimes = \sum_{u=1}^{3}\sum_{v=1}^{3} I_{x+u,y+v} * K_{u,v}$$

Keterangan:

$C_{j,x,y}$ = Layer konvolusi ke-j dari matriks x,y.
$I_{x,y}$ = Nilai pixel dari input ke-x,y.
$K_{u,v}$ = Nilai pixel dari filter ke-u,v.
B_i = Nilai bias ke-i.
f = Fungsi aktifasi *non-linearity* dengan ReLU.

Hasil dari layer ini akan menghasilkan *activation map* yang kemudian akan diaktifasi menggunakan fungsi aktifasi *Rectified Linear Unit* (ReLU). Kegunaan dari fungsi aktifasi ini adalah dimana pixel yang memiliki nilai < 0 akan dirubah menjadi bernilai 0, dimana fungsi aktifasi ReLU dijelaskan sebagai berikut.

$$f(x) = ReLU(x) = \begin{cases} x\ (x \geq 0) \\ 0\ (x < 0) \end{cases}$$

Keterangan:

$ReLU(x)$ = Fungsi aktifasi ReLU.
x = Hasil konvolusi pada layer konvolusi.

Namun, untuk melakukan *convolutional layer* di layer selanjutnya, akan digunakan rumus yang sedikit berbeda, dimana inputan sebanyak jumlah filter masing-masing akan dikonvolusi kemudian dijumlahkan. Rumus tersebut dituliskan pada persamaan (2.11) sebagai berikut:

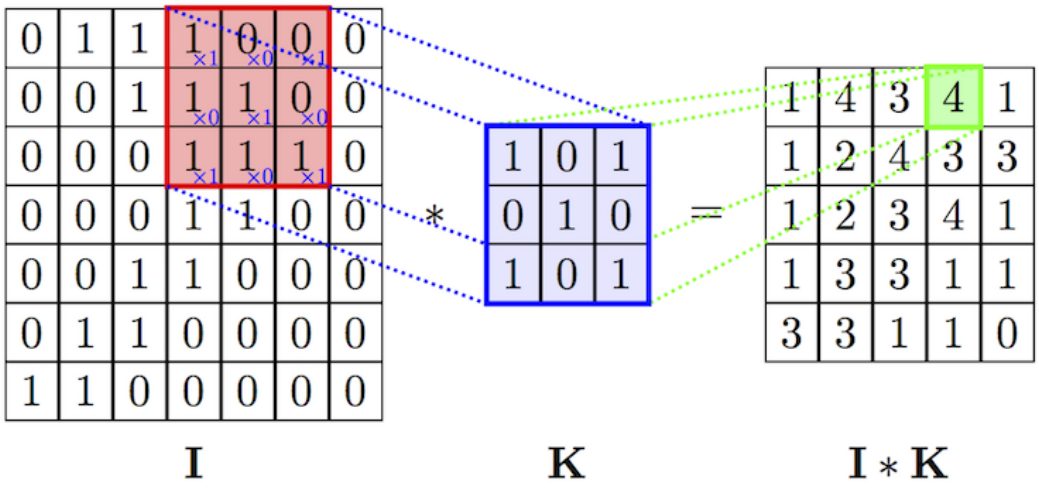
$$C_{j,x,y} = f\left(\sum_{i=1}^{filter} S_i \otimes K_{u,v} + B_i\right) \quad (2.11)$$

Keterangan:

$C_{j,x,y}$ = Layer konvolusi ke-j.
$S_{i,x,y}$ = Layer *pooling* ke-i.
$K_{u,v}$ = Nilai pixel dari filter ke-u,v.
B_i = Bias ke-i.
f = Fungsi aktifasi *non-linearity* dengan ReLU.

$\otimes$ = Operasi konvolusi [6].

Proses dari konvolusi sendiri telah tergambarkan pada Gambar 2.16.



Gambar 2.16 Proses *Convolutional Layer*

2.3.1.2 Pooling Layer

Pada layer ini akan disediakan matriks *mask* berukuran 2x2 dan akan dilakukan perhitungan *sample* terhadap hasil dari *convolutional layer* untuk menghasilkan output. Perhitungan *sample* yang digunakan sendiri adalah mengambil nilai maksimum dari area *mask*. Berikut adalah fungsi yang digunakan untuk mengambil nilai maximum dari area *mask* dengan persamaan (2.12).

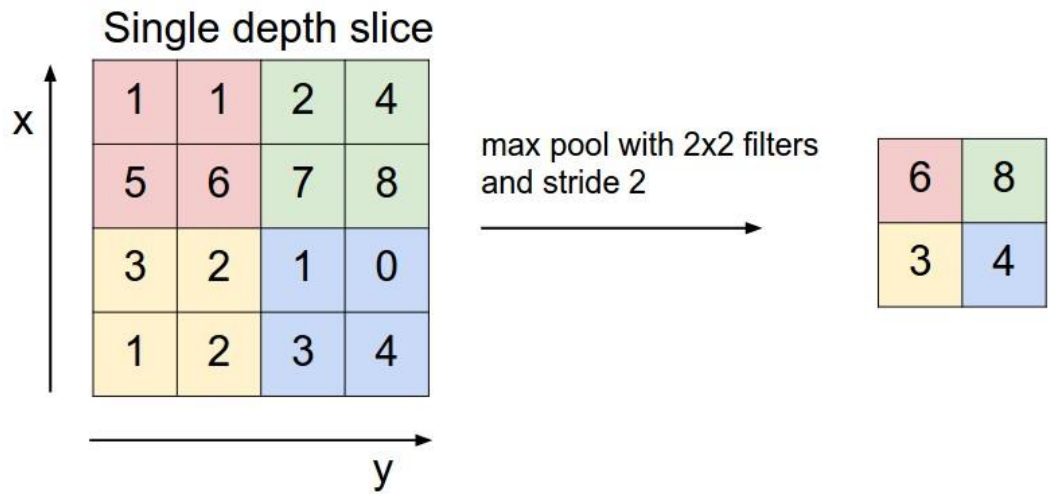
$$S_{x,y} = Max\left(C_{x,y}, C_{x+1,y}, C_{x,y+1}, C_{x+1,y+1}\right) \quad (2.12)$$

Keterangan:

$S_{x,y}$ = Hasil dari *pooling layer*.

$C_{j,x,y}$ = Nilai pixel dari hasil *convolutional layer*.

Dimana *mask* akan bergerak dengan jarak 2 pixel ($stride = 2$) sehingga nilai pixel yang sudah dicari nilai maksimumnya tidak akan dihitung kembali. Proses dari tahapan ini tergambarkan pada Gambar 2.17.



Gambar 2.17 Proses *Pooling Layer*

2.3.1.3 Fully-Connected Layer

Pada layer ini semua neuron yang ada di layer sebelumnya akan saling dikoneksikan satu sama lain. Tujuan dari layer ini adalah untuk klasifikasi citra. Namun sebelum dilakukan *Fully-Connected Layer* akan ada tahap *flatten* terlebih dahulu. Pada tahapan *flatten* matriks hasil dari *pooling layer* terakhir akan dibuat menjadi vektor, kemudian barulah vektor tersebut dihitung dengan persamaan (2.13).

$$Fully_i = \sum_{j}^{Length(flatten)} W_{i,j} * flatten_j + b_i \qquad (2.13)$$

Keterangan:

$Fully_i$ = Hasil dari perhitungan pada *fully-connected layer*.
$W_{i,j}$ = Nilai bobot yang digunakan dari hasil *convolutional layer*.
$flatten_j$ = Nilai dari vektor ke-j.
i = Kelas ke-i ($i = 1,2,3$).

Kemudian hasil dari $Fully$ akan diaktifasi dengan menggunakan fungsi *softmax*, digunakan fungsi *softmax* karena pada penelitian ini menghasilkan *output* atau jumlah kelas lebih dari dua [17], atau biasa disebut dengan *multi-class*. Tujuan dari fungsi *softmax* sendiri adalah agar dapat diketahui prediksi yang dihasilkan dari arsitektur yang dibangun, dimana rumus untuk fungsi *softmax* ditulis pada persamaan (2.14).

$$\widehat{y_i} = \frac{e^{Fully_i}}{(\sum_{i=1}^{kelas} e^{Fully_i})} \quad (2.14)$$

Keterangan:
$\widehat{y_i}$ = Hasil dari aktifasi fungsi *softmax*.
$Fully_i$ = Hasil dari perhitungan pada *fully-connected layer* ke-i.
i = Kelas ke-i ($i = 1,2,3$).

2.3.1.4 Cross-Entropy Loss Function

Selanjutnya akan dicari nilai *error* yang didapat dari hasil prediksi pada *fully-connected layer* sebelumnya, dimana nilai *error* ini digunakan untuk menentukan apakah hasil prediksi dari *CNN* sudah mencapai target atau belum, maka dari itu akan dilakukan perbandingan antara *error* dengan *target error* dimana, apabila *error* masih dibawah dari target, maka akan dilakukan *loss function*. Dari beberapa jenis *loss function* yang ada, salah satu-nya adalah *Cross-entropy Loss Function*, dimana rumus dari *cross-entropy* tersebut dituliskan pada persamaan (2.15).

$$Loss = -\sum_{i}^{kelas} t_i * Log(\widehat{y_i}) \quad (2.15)$$

Keterangan:
$\widehat{y_i}$ = Hasil dari aktifasi fungsi *softmax*.
t_i = Nilai dari target, bernilai 1 apabila target adalah kelas yang dituju, dan bernilai 0 apabila nilai target bukanlah kelas yang dituju.
i = Kelas ke-i ($i = 1,2,3$).

2.4 *Backpropagation*

Backpropagation adalah sebuah metode pengoptimalan umum untuk melakukan diferensiasi otomatis dari fungsi nested yang kompleks [8]. Intinya *Bacpropagation* adalah sebuah metode untuk mendapatkan turunan fungsi dari fungsi-fungsi yang digunakan. Seiring berjalannya waktu, penggunaan dari *Backpropagation* sendiri semakin banyak digunakan, beberapa contohnya adalah pada pengaplikasian untuk *pattern recognition, dynamic modelling, sensitivity analysis,* dan lainnya. *Backpropagation* sendiri bisa diaplikasikan pada *Neural Network* [19].

Neural Network dapat mempelajari dan menentukan bobot dan bias menggunakan algoritma *gradient descent*. *Gradient descent* merupakan algoritma optimisasi orde pertama untuk menemukan nilai minimum lokal dari sebuah fungsi [6]. Pada *backpropagation* ini digunakanlah rumus *chain rule* terhadap *gradient descent* untuk kemudian memperbaiki bobot dan bias yang digunakan. Turunan gradien dari setiap fungsi akan dibahas pada sub-bab selanjut-nya [17] [18].

2.4.1 Turunan gradien error terhadap *softmax*

Tahapan turunan gradien *error* terhadap *softmax* berdasarkan rumus yang ditulis oleh Peter Sadowski [18].

$$\Delta\hat{y}_i = \frac{\partial Loss}{\partial \hat{y}_i} = \hat{y}_i - t_i \tag{2.16}$$

Keterangan:

$\Delta\hat{y}_i$ = Nilai turunan dari fungsi *softmax*.

t_i = Nilai dari target, bernilai 1 apabila target adalah kelas yang dituju, dan bernilai 0 apabila nilai target bukanlah kelas yang dituju.

i = Kelas ke-i ($i = 1,2,3$).

2.4.2 Turunan gradien error terhadap bobot

Tahapan turunan gradien *error* terhadap bobot berdasarkan rumus yang ditulis oleh Zhifei Zhang [17].

$$\Delta W_{i,j} = \frac{\partial Loss}{\partial W_{i,j}} = \Delta\hat{y}_i * flatten_j \tag{2.17}$$

Keterangan:

$\Delta W_{i,j}$ = Nilai turunan dari bobot.

$\Delta\hat{y}_i$ = Nilai turunan dari *softmax*.

$flatten_j$ = Nilai vektor pada *flatten*.

i = Kelas ke-i ($i = 1,2,3$).

2.4.3 Turunan gradien error terhadap bias

Tahapan turunan gradien *error* terhadap bobot berdasarkan rumus yang ditulis oleh Zhifei Zhang [16].

$$\Delta b_i = \frac{\partial Loss}{\partial b_i} = \Delta\hat{y}_i \tag{2.18}$$

Keterangan:

Δb_i = Nilai turunan dari bias.

$\Delta \hat{y}_i$ = Nilai turunan dari *softmax*.

i = Kelas ke-i ($i = 1,2,3$).

2.5 *Stochastic Gradient Descent*

Stochastic Gradient Descent adalah sebuah algoritma untuk menemukan nilai minimum lokal dari sebuah fungsi [20]. Algoritma dari *Stochastic Gradient Descent* dapat dilihat pada Gambar 2.18.

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ
    # where: L is the loss function
    # f is a function parameterized by θ
    # x is the set of training inputs $x^{(1)}, x^{(2)}, ..., x^{(n)}$
    # y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, ..., y^{(n)}$

$\theta \leftarrow 0$
repeat T times
  For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)
  Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}$?
  Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$ from the true output $y^{(i)}$?
  $g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss ?
  $\theta \leftarrow \theta - \eta\, g$ # go the other way instead
return θ

Gambar 2.18 Algoritma *Stochastic Gradient Descent*

Dimana dari algoritma tersebut, bisa juga dirumuskan nilai perbaikan bobot dan bias baru seperti pada persamaan berikut.

1. Perbaikan nilai pada bobot

$$\theta W_i = W_i - \alpha(\Delta W_i) \tag{2.19}$$

Keterangan:

ΔW_i = Nilai turunan dari bobot.

W_i = Nilai bobot.

α = Nilai *learning rate*.

θW_i = Nilai bobot baru.

i = Kelas ke-i ($i = 1,2,3$).

2. Perbaikan nilai pada bias

$$\theta b_i = b_i - \alpha(\Delta b_i) \quad (2.20)$$

Keterangan:

Δb_i = Nilai turunan dari bias.
b_i = Nilai bias.
α = Nilai *learning rate.*
θb_i = Nilai bias baru.
i = Kelas ke-i ($i = 1,2,3$).

2.6 *Unified Modeling Language* (UML)

UML adalah sebuah teknik atau bahasa pembuatan model untuk pengembangan atau pembangunan perangkat lunak dan sistem. Secara umum *modeling language* dapat juga dibuat dengan menggunakan *pseudo-code, actual code*, gambar, maupun diagram. UML sendiri memiliki 6 kelebihan yaitu [21]:

1. *Formal Language*

   Setiap elemen yang digunakan sudah sangat jelas penjelasannya sehingga tidak akan ada terjadinya kesalah pahaman terhadap elemen yang dimaksud.

2. *Concise*

   Concise atau ringkas, seluruh elemen yang ada pada UML dibuat sangat sederhana.

3. *Comprehensive*

   Comprehensive atau luas, UML sudah menjelaskan semua aspek-aspek penting pada system.

4. *Scalable*

   Scalable atau mudah untuk dikembangkan, untuk melakukan pengembangan pada sistem akan sangat mudah apabila menggunakan UML.

5. *Built on lessons learned*

   UML berdiri setelah adanya pelatihan-pelatihan pada komunitas *object-oriented* selama 15 tahun.

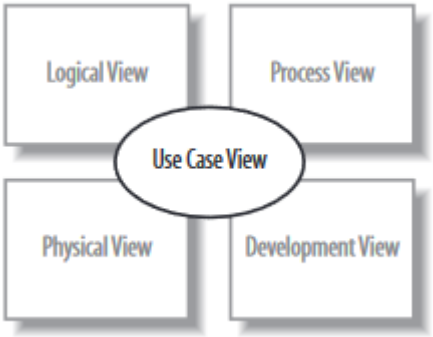
6. *Standard*

   UML banyak digunakan dimanapun sehingga dapat diibaratkan sebagai bahasa yang sudah umum atau standar.

Pada sub-bab selanjutnya akan dibahas beberapa model yang digunakan dalam UML.

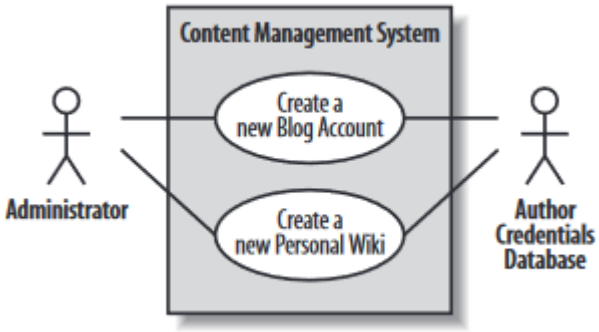
2.6.1 *Use Case* Diagram

Use case adalah sebuah situasi dimana sistem digunakan untuk memenuhi satu atau seluruh kebutuhan *user*. *Use case* sendiri adalah bagian terpenting atau bagian utama dari model pembangunan. Berdasarkan UML, *use case* sendiri dapat diumpamakan bagian-bagian dari *logical*, *process*, *physical*, dan *development* seperti pada Gambar 2.19.



Gambar 2.19 Posisi *Use Case* berdasarkan UML

Adapun contoh untuk penggambaran *use case* pada permisalan kasus sebagai berikut, seorang admin dan *author* dapat melakukan pembuatan akun blog dan juga dapat melakukan pembuatan wiki. *Use case* dari contoh kasus ini bisa dilihat pada Gambar 2.20.



Gambar 2.20 Contoh sederhana dari *Use Case*

Setelah pembangunan *use case* akan dibangun pula sebuah skenario dari masing-masing *use case* yang ada. Kegunaan dari skenario *use case* sendiri adalah untuk memperjelas proses yang ada di dalam masing-masing *use case* tersebut

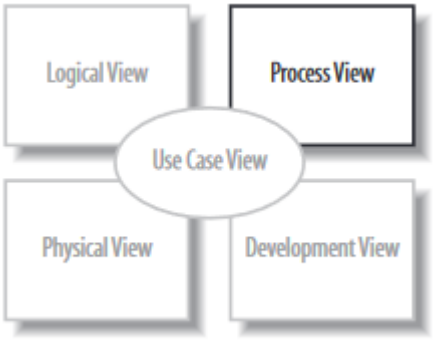
secara tekstual. Contoh dari skenario *use case* dari "*Create a new Personal Wiki*" dari *use case* sebelumnya dan telah tergambarkan pada Gambar 2.21 [21].

Use case name	**Create a new Personal Wiki**	
Related Requirements	Requirement A.2.	
Goal In Context	A new or existing author requests a new personal Wiki from the Administrator.	
Preconditions	The author has appropriate proof of identity.	
Successful End Condition	A new personal Wiki is created for the author.	
Failed End Condition	The application for a new personal Wiki is rejected.	
Primary Actors	Administrator.	
Secondary Actors	Author Credentials Database.	
Trigger	The Administrator asks the CMS to create a new personal Wiki.	
Main Flow	**Step**	**Action**
	1	The Administrator asks the system to create a new personal Wiki.
	2	The Administrator enters the author's details.
	3	The author's details are verified using the Author Credentials Database.
	4	The new personal Wiki is created.
	5	A summary of the new personal Wiki's details are emailed to the author.
Extensions	**Step**	**Branching Action**
	3.1	The Author Credentials Database does not verify the author's details.
	3.2	The author's new personal Wiki application is rejected.

Gambar 2.21 Contoh skenario *use case*

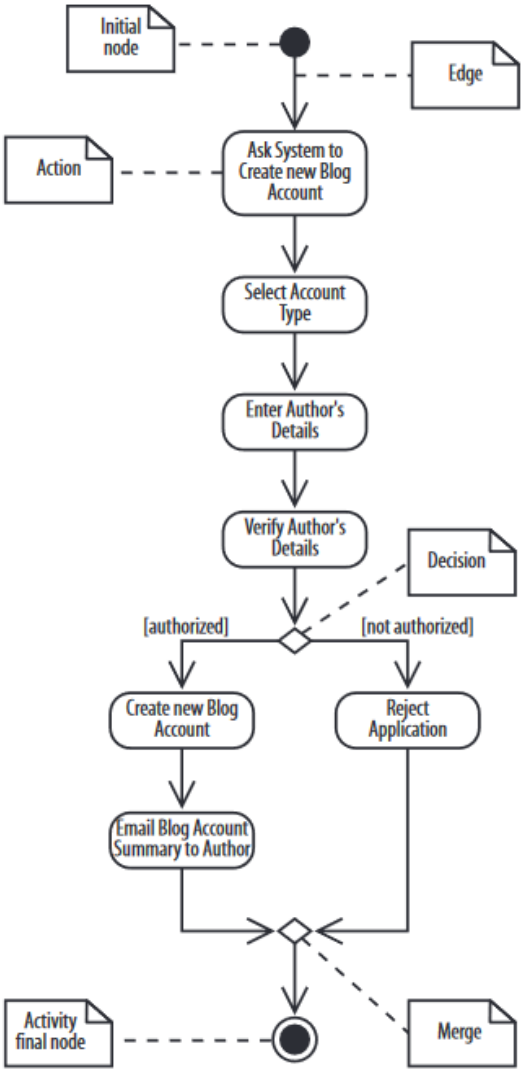
2.6.2 *Activity* Diagram

Activity diagram digunakan untuk menjelaskan alur atau cara dari sebuah sistem mencapai tujuannya dengan diagram. *Activity* diagram sendiri menjelaskan alurnya dengan cara *actions chained* atau aksi-aksi yang saling berhubungan pada sistem tersebut. Jika kita melihat Gambar 2.19 kembali, maka berdasarkan UML, *activity* diagram adalah bagian dari *process view* seperti pada Gambar 2.22.



Gambar 2.22 Posisi *Activity* Diagram berdasarkan UML

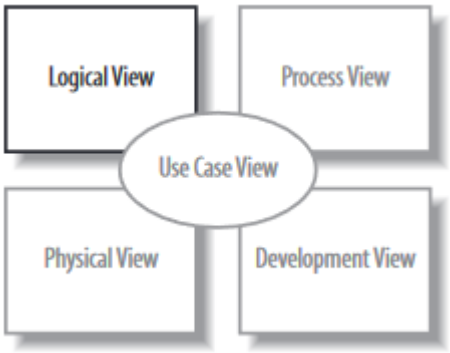
Dengan contoh kasus yang sama pada *use case* sebelumnya, maka *activity diagram* dari kasus tersebut telah digambarkan pada Gambar 2.23.



Gambar 2.23 Contoh dari *Activity* Diagram

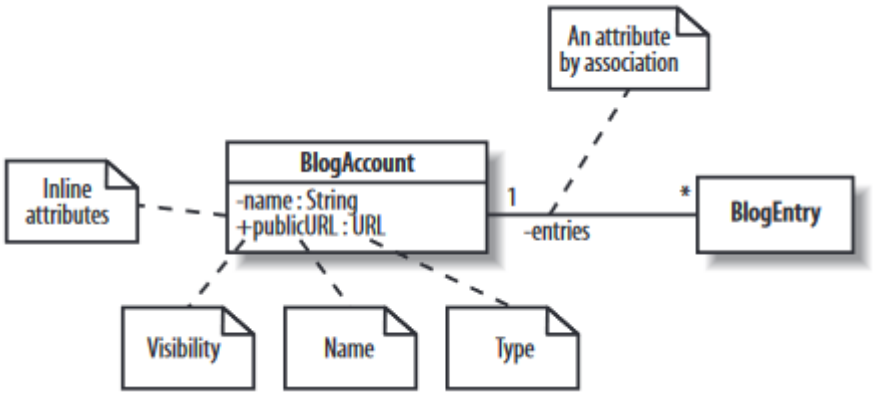
2.6.3 *Class* Diagram

Class diagram digunakan untuk menjelaskan bagian objek-objek yang berbeda yang dibutuhkan pada sistem. *Class* diagram adalah bagian terpenting dari *object-oriented system* maka dari itu *class* diagram adalah yang paling populer dari UML. Bagian dari *class* diagram pada UML sendiri telah digambarkan pada Gambar 2.24.



Gambar 2.24 Posisi *Class* Diagram berdasarkan UML

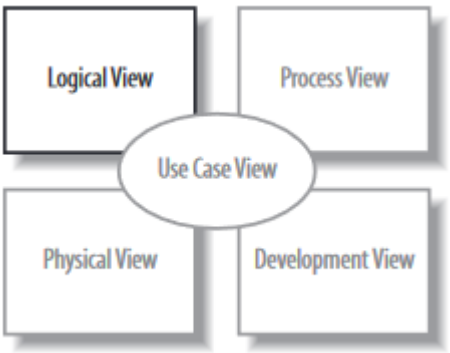
Contoh pada *class diagram* disini digunakan *class BlogAccount* dari kasus sebelumnya, maka didapatkan hasil dari *class diagram* seperti pada Gambar 2.25.



Gambar 2.25 Contoh dari *Class* Diagram

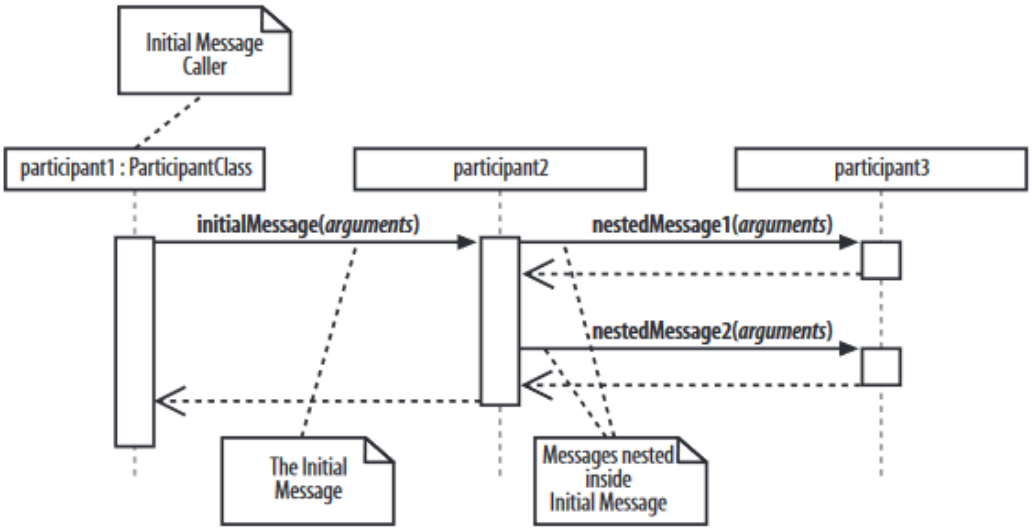
2.6.4 *Sequence* Diagram

Sequence diagram digunakan untuk menjelaskan urutan hubungan atau interaksi terhadap bagian-bagian dari sistem. Dengan *sequence* diagram setiap interaksi dapat dijelaskan akan berjalan ketika sesuatu menjalankannya atau melakukan *trigger* terhadap interaksi tersebut. Bagian dari *sequence* diagram pada UML sendiri serupa dengan *class* diagram dan telah tergambarkan pada Gambar 2.26 contoh untuk *sequence* diagram sendiri telah digambarkan pada Gambar 2.26.



Gambar 2.26 Posisi *Sequence* Diagram berdasarkan UML

Contoh pada *sequence diagram* disini adalah contoh saat melakukan pengiriman pesan pada *class* yang ada. *Sequence diagram* dari contoh sendiri telah tergambarkan pada Gambar 2.27.



Gambar 2.27 Contoh dari *Sequence* Diagram

2.7 *Python*

Python adalah bahasa pemrograman interpretatif multiguna dengan filosofi perancangan yang berfokus pada tingkat keterbacaan kode. *Python* merupakan salah satu bahasa pemrograman yang populer di dunia kerja Indonesia. *Python* dikembangkan oleh Guido van Rossum pada tahun 1990 di CWI, Amsterdam sebagai kelanjutan dari bahasa pemrograman ABC.

Python diklaim sebagai bahasa yang menggabungkan kapabilitas, kemampuan dengan sitaks kode yang sangat jelas, dan dilengkapi dengan fungsionalitas pustaka standar yang besar serta komprehensif. Pada umumnya *Python* mendukung multi paradigma pemrograman, seperti pemrograman berorientasi objek, pemrograman imperatif dan pemrograman fungsional [22].

Pada *Python* sendiri tersedia sangat banyak *library* yang dapat dimanfaatkan sesuai dengan kebutuhan, beberapa dari *library* yang digunakan akan dibahas pada sub-bab ini.

2.7.1 PyQt

PyQt adalah sebuah *library* pada *python* untuk membangun *Graphic User Interface* (GUI) dengan lebih mudah. Di dalam PyQt sudah terdapat paket bernama QtDesigner untuk mempermudah pembangunan GUI hanya dengan *drag & drop* desain saja. PyQt sendiri sudah memiliki lebih dari 35 ekstensi modul yang siap digunakan dan mampu membuat *Python* sebagai bahasa alternatif dari C++ (https://wiki.python.org/moin/PyQt diakses pada tanggal 21 September 2019).

2.7.2 OpenCV

OpenCV adalah sebuah *library* yang mengkhususkan dirinya untuk pembangunan *computer vision* dan *machine learning*. OpenCV sendiri sudah memiliki sangat banyak algoritma yang teroptimasi dengan jumlah lebih dari 2500. *Library* OpenCV sendiri tidak hanya ada untuk bahasa pemrograman *Python* saja, melainkan ada pula untuk bahasa lain seperti C++, Java, dan MATLAB Interfaces dan sudah sangat *support* penggunaannya pada operasi sistem Windows, Linux, Mac OS dan Android (https://opencv.org/about/ diakses pada tanggal 21 September 2019).

2.7.3 Numpy

Numpy adalah sebuah *library* pada *Python* yang berguna untuk melakukan perhitungan *scientific*. Di dalamnya terdapat paket-paket untuk melakukan operasi terhadap objek array yang berupa matriks dengan N-dimensi, aljabar linear, dan banyak lagi (https://www.numpy.org/ diakses pada tanggal 21 September 2019).

2.8 Pengujian Akurasi

Pengujian akurasi dilakukan untuk mengetahui seberapa besar akurasi atau nilai kebenarannya yang didapatkan terhadap metode yang digunakan, adapun rumus dari pengujian akurasi yang digunakan adalah sebagai berikut.

$$Akurasi = \frac{Jumlah\ Kepribadian\ Yang\ Tepat}{Jumlah\ Data\ Yang\ Diuji} * 100\% \qquad (2.21)$$