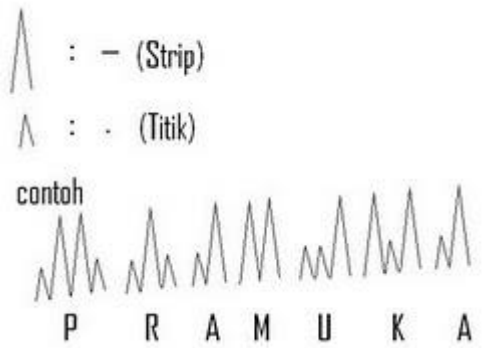
BAB 2
TINJAUAN PUSTAKA

2.1 Sandi Rumput

Sandi rumput pramuka merupakan sistem representasi huruf, angka, dan tanda baca yang dikembangkan oleh penggiat kepramukaan Indonesia [1]. Sandi rumput mempunyai bentuk yang khas yaitu menyerupai rumput dan cara pengerjaan yang baku. Sandi ini mempunyai kemiripan dengan sandi dengan sandi air, hanya saja sandi air merupakan sandi rumput yang dibalik ke bawah. Pada dasarnya, sandi rumput bukanlah sandi asli, melainkan turunan dari sandi morse, sehingga representasi masing-masing huruf, angka, dan tanda baca yang ada di sandi rumput sama dengan sandi morse. Perbedaan diantara keduanya terletak pada cara penulisannya, dimana titik dan garis yang ada di sandi morse diganti dengan rumput kecil dan rumput besar. Titik yang ada di sandi morse diganti dengan rumput kecil sedangkan garis diganti dengan rumput besar. Contohnya sandi rumput bisa dilihat pada Gambar 2.1.



Gambar 2.1 Contoh Sandi Rumput

2.2 Citra

Citra bisa diartikan sebagai fungsi 2 dimensi f(x,y), dimana x dan y adalah koordinat spasial, dan amplitude pada koordinat tersebut disebut sebagai intensitas atau *gray level* pada titik itu [14]. Ketika nilai x,y, dan nilai intensitas dari f merupakan nilai yang dapat ditentukan, maka dapat disebut sebagai citra digital. Citra digital direpresentasikan oleh *array* dua dimensi atau sekumpulan *array* dua dimensi dimana setiap array merepresentasikan satu kanal warna [15].

2.3 Pengolahan Citra

Menurut Arymurthy dalam [16], pengolahan citra merupakan bidang studi yang mempelajari proses pengolahan gambar dimana baik masukan maupun keluarannya berbentuk berkas citra digital. Menurut Efford dalam [17], pengolahan citra adalah istilah umum untuk berbagai teknik yang keberadaanya untuk memanipulasi citra dengan berbagai cara. Foto adalah contoh gambar yang dapat diolah secara mudah. Setiap foto dalam bentuk citra digital dapat diolah melalui perangkat lunak tertentu.

2.4 Citra RGB

Citra RGB atau citra berwarna adalah citra yang mempunyai 3 buah *channel* warna [15]. Pada umunya jenis citra ini terbentuk dari komponen merah/*red* (R), hijau/*green* (G), dan biru/*blue* (B) yang dimodelkan kedalam ruang warna RGB. RGB adalah standar yang digunakan untuk menampilkan citra berwarna pada layar televisi maupun layar komputer. Citra berwarna sering disebut juga sebagai 24-bit color image karena untuk setiap nilai pikselnya memerlukan penyimpanan sebesar 24-bit. Dengan kombinasi warna yang ada, citra berwarna memiliki kemungkinan variasi warna sebesar 16.777.216 warna.

2.5 Citra Keabuan

Citra keabuan atau citra *grayscale* merupakan citra yang hanya memiliki satu buah *channel* sehingga yang ditampilkan hanyalah nilai intensitas atau derajat keabuan [15]. Dalam hal ini, intensitas berkisar antara 0 sampai 255. Nilai 0 menyatakan hitam dan nilai 255 menyatakan putih [17]. Ada beberapa cara untuk mengkonversikan citra RGB ke citra *grayscale*. Cara paling mudah adalah dengan merata-ratakan nilai RGB dengan menggunakan persamaan (2.1) sebagai berikut.

$$y = \frac{R+G+B}{3} \tag{2.1}$$

Keterangan : R = Nilai komponen merah (*Red*) untuk setiap pixel

G = Nilai komponen hijau (*Green*) untuk setiap pixel

B = Nilai komponen biru (*Blue*) untuk setiap pixel

Cara kedua untuk mengkonversikan citra RGB ke citra grayscale dengan menggunakan persamaan (2.2) sebagai berikut.

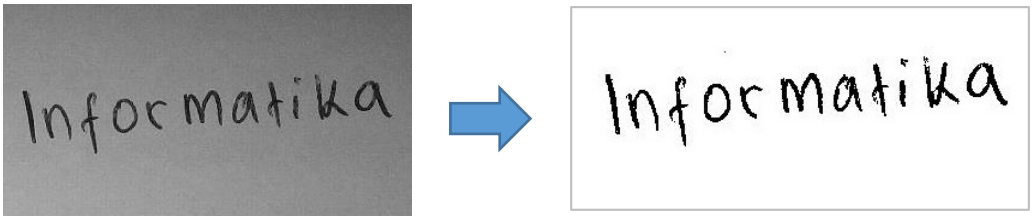
$$y = 0{,}2989*R + 0{,}5870*G + 0{,}1141*B \tag{2.2}$$

Dari kedua persamaan diatas, persamaan (2.2) menghasilkan konversi yang lebih baik daripada persamaan (2.1) [15]. Oleh karena itu, pada penelitian ini persamaan (2.2) dipilih sebagai persamaan untuk mengkonversi citra RGB ke citra *grayscale*. Contoh citra grayscale ditujukan pada Gambar 2.2

Gambar 2.2 Contoh Citra *Grayscale*

2.6 Thresholding

Thresholding merupakan suatu proses dimana hasilnya berupa citra biner dari citra *grayscale* atau citra berwarna dengan mengatur nilai piksel ke nilai 0 atau 1 tergantung dari nilai ambang batasnya apakah nilai piksel tersebut berada dibawah atau diatas ambang batas [18]. *Thresholding* terbagi menjadi dua yaitu *global tresholding* dan *local thresholding*. *Global thresholding* merupakan *thresholding* yang apabila nilai ambang t bergantung hanya pada satu nilai aras keabuan f(x,y). Sedangkan *local thresholding* adalah *thresholding* yang dimana nilai ambang t bergantung pada f(x,y) dan g(y,x) dengan g(y,x) menyatakan properti citra local pada titik (y,x) [17]. Pada penelitian ini, metode *thresholding* yang digunakan yaitu *Sauvola Threshold* [19]. Metode ini termasuk ke dalam *local thresholding* yang dimana cara kerjanya yaitu menghitung ambang batas menggunakan rentang nilai dinamis dari nilai standar deviasi citra *grayscale* [20]. Dalam hal ini, nilai ambang batas ditentukan oleh nilai pixel tetangga. Contoh hasilnya bisa dilihat pada Gambar 2.3



Gambar 2.3 Contoh *Sauvola Thresholding*

Adapun persamaan dari metode *Savoula Threshold* adalah sebagai berikut.

$$T(x,y)= m(x,y)* \left[1+k* \left(\frac{s(x,y)}{R}-1\right)\right] \quad (2.3)$$

$$m\,(x,y) = \frac{\sum_{i=min}^{i=max}\sum_{j=min}^{j=max} img(i,j)}{i*j} \quad (2.4)$$

$$s = \sqrt{\frac{\sum_{i=min}^{i=max}\sum_{j=min}^{j=max}(img(i,j)-m(x,y))^2}{(i*j)-1}} \quad (2.5)$$

dimana:

T : Fungsi *Threshold*

m : Nilai rata-rata dari sejumlah piksel citra

k : Nilai antara 0.2 – 0.5

s : Standar deviasi dari sejumlah piksel citra

R : Nilai maksimum dari standar deviasi

x : Lebar citra

y : Tinggi citra

i : Nilai pixel tetangga

j : Nilai pixel tetangga

Setelah nilai T(x,y) sudah didapatkan, maka masukkan ke dalam persamaan berikut.

$$f(x,y)= \begin{cases} 0, & img(x,y) < T(x,y) \\ 255, & img(x,y) \geq T(x,y) \end{cases} \quad (2.6)$$

Keterangan : f(x,y) : Fungsi yang menghasilkan nilai antara 0 atau 255

img : Nilai grayscale pada citra

Setelah didapatkan nilai threshold, maka selanjutnya melakukan proses binerisasi yaitu mengubah nilai *threshold* ke dalam nilai biner dengan mengubah nilai 255 menjadi 0 dan nilai 0 menjadi 1.

2.7 Segmentasi

Segmentasi merupakan suatu proses generik dimana gambar dibagi menjadi beberapa wilayah/daerah (*region*) [18]. Pada pengolahan citra, peran segmentasi sangatlah penting sebelum memasuki proses-proses seperti klasifikasi, ekstraksi fitur, dan lain-lain. Tujuan dari segmentasi adalah untuk membagi citra menjadi beberapa bagian atau wilayah yang memiliki atribut yang sama. Objek yang tersegmentasi disebut sebagai *foreground* sementara sisanya disebut *background*. Contoh segmentasi citra bisa dilihar pada Gambar 2.4

Gambar 2.4 Segmentasi Citra

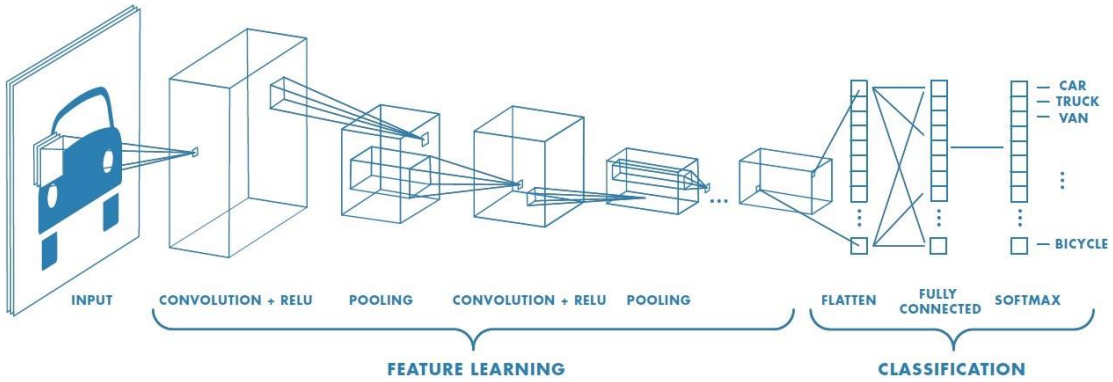
2.8 Resize

Resize merupakan sebuah proses mengubah ukuran piksel citra. Proses perubahan piksel citra bisa menjadi lebih besar atau lebih kecil dari ukuran aslinya. *Resize* dapat dilihat pada Gambar 2.5

Gambar 2.5 *Resize* Citra

2.9 Convolutional Neural Network

Convolutional Neural Network (CNN) adalah pengembangan dari *Multi Layer Perceptron* (MLP) yang didesain untuk mengolah data dua dimensi [21]. CNN termasuk dalam jenis *Deep Neural Network* atau *Deep Learning* karena kedalaman jaringan yang tinggi dan banyak diaplikasikan pada data citra. CNN terinspirasi oleh proses-proses biologi dimana pola konektivitas antar neuron menyerupai organisasi visual cortex pada binatang [5]. Gagasan awal dari pengembangan *Deep Learning* yaitu mempelajari apa yang sebenarnya terjadi pada lapisan tersembunyi (*hidden layer*) di dalam sebuah jaringan syaraf tiruan [37].

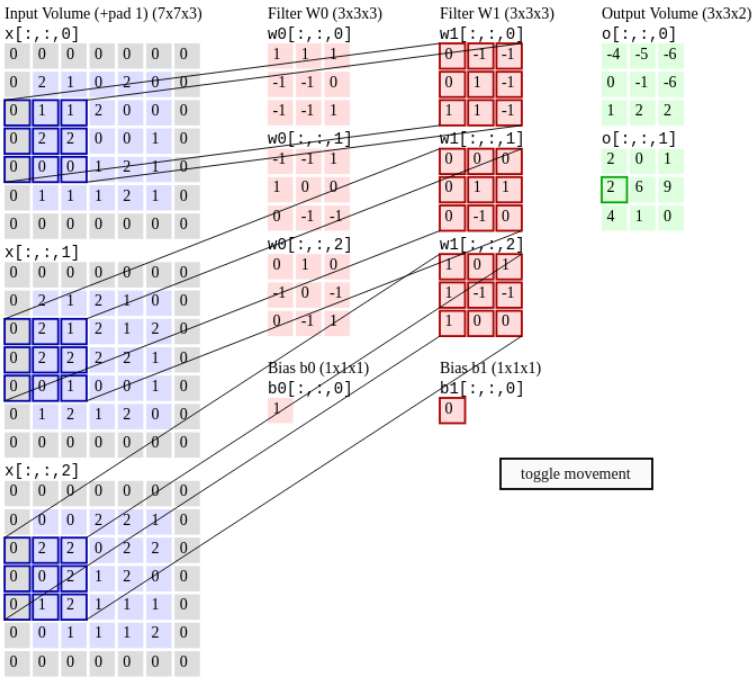


Gambar 2.6 Arsitektur CNN

CNN terdiri atas satu lapis data masukan (*input layer*), sejumlah lapisan tersembunyi (*hidden layer*), dan satu lapisan keluaran (*output layer*) [5]. Di dalam CNN terdapat empat lapisan utama antara lain lapisan konvolusi (*Convolution Layer*), lapisan aktivasi (*Activation Layer*), lapisan penggabungan (*Pooling Layer*), dan lapisan terhubung penuh (*Fully-Connected Layer*).

2.9.1 Convolution Layer

Convolution Layer merupakan lapisan yang didalamnya terdapat operasi konvolusi. Sebagian besar komputasi berada pada lapisan ini [5]. Lapisan ini juga yang menjadi proses utama yang mendasari sebuah CNN [21]. Konvolusi merupakan sebuah operasi yang mengalikan sebuah citra dengan sebuah mask atau kernel [15]. Operasi konvolusi dilakukan dengan cara menggeser mask atau kernel dengan langkah/stride yang sudah ditentukan. Tujuan dilakukannya konvolusi adalah untuk mengekstraksi fitur dari citra masukan. Hasil dari proses ini disimpan di dalam matriks yang baru. Proses konvolusi dapat ditujukan pada Gambar 2.7



Gambar 2.7 Proses Konvolusi

Convolutional layer dalam CNN umumnya menggunakan lebih dari satu mask (dalam hal ini disebut sebagai filter). Jika menggunakan empat *filters* dan diterapkan pada citra dengan ukuran sebesar 28x28, maka *convolution layers* akan berisi sejumlah neurons yang tersusun dalam grid berukuran 28x28x4. Dengan demikian, akan ada empat neurons yang melihat area yang sama pada citra masukan tersebut.

Pada *convolution layer* juga terdapat hyperparamter yang harus ditentukan, antara lain yaitu *Depth*, *Stride*, dan *Padding* [22]. *Depth* merupakan jumlah kernel (*filter*) yang akan digunakan pada saat proses konvolusi. *Stride* merupakan jumlah langkah pergeseran filter. Semakin kecil *stride* maka semakin detail informasi yang didapat, namun membutuhkan komputasi lebih berat dibanding jumlah *stride* yang lebih besar. Sedangkan *Padding* merupakan jumlah piksel yang akan ditambahkan pada setiap sisi dari citra masukan. Biasanya *padding* berisi piksel yang nilainya 0. Tujuan dari *padding* adalah untuk mengurangi informasi yang terbuang sehingga ukuran citra *input* dan *output* tetap sama.

2.9.2 Activation Layer

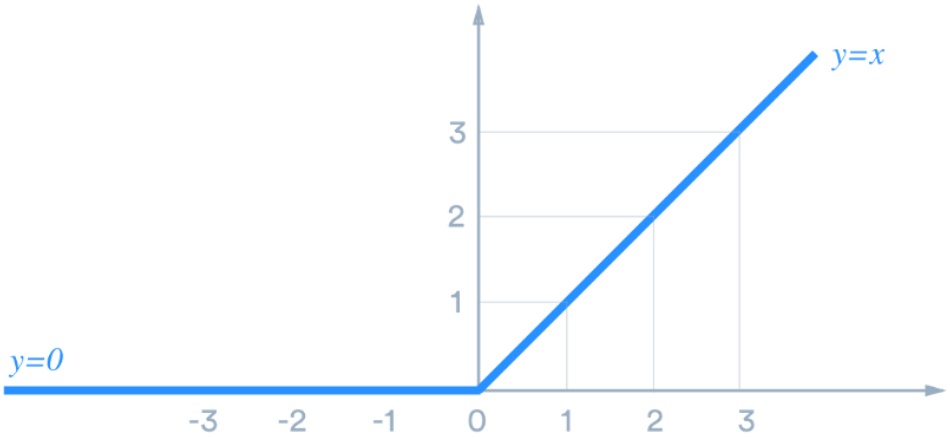
Fungsi aktivasi atau *Activation Layer* adalah fungsi non linear yang memungkinkan sebuah JST untuk dapat mentransformasikan data input menjadi dimensi yang lebih tinggi sehingga dapat dilakukan pemotongan *hyperlane* sederhana yang memungkinkan dilakukan klasifikasi [23]. Pada penelitian ini, fungsi aktivasi yang digunakan yaitu fungsi *Rectified Linear Unit* (ReLU). Fungsi ini meningkatkan sifat nonlinearitas fungsi keputusan dan jaringan secara keseluruhan tanpa mempengaruhi bidang-bidang repesitif pada *convolution layer*. Fungsi ReLU mempunyai persamaan sebagai berikut.

$$f(x) = \max(0,x) \tag{2.7}$$

dimana: f(x) : fungsi ReLU

max(0,x) : fungsi maximum

x : nilai input



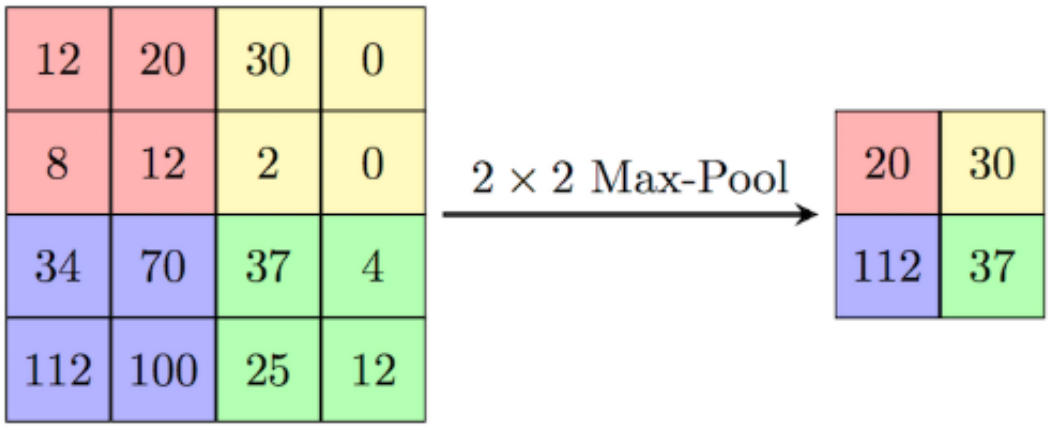
Gambar 2.8 Grafik Fungsi ReLU

Sederhananya, fungsi ReLU ini cara kerjanya sama seperti proses *thresholding*, yaitu menentukan nilai ketika nilai tersebut berada di ambang batas nilai yang sudah ditentukan [21]. Nilai ambang batas pada fungsi ini adalah 0. Ketika nilai x kurang dari atau sama dengan 0 maka nilai x = 0 sedangkan apabila nilai x lebih dari 0 maka nilai x = x. Bila digambarkan dengan persamaan maka akan persamaanya adalah sebagai berikut.

$$\max(0,x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{2.8}$$

2.9.3 Pooling Layer

Lapisan penggabungan atau *Pooling Layer* (kadang disebut *Subsampling Layer*) adalah proses mereduksi sebuah data citra [22]. Lapisan *pooling* dapat secara progresif mengurangi ukuran volume *output Feature Map*, sehingga mengurangi jumlah parameter dan perhitungan di jaringan, dan untuk mengendalikan *Overfitting* [22]. Proses pooling yang umum digunakan dalam CNN yaitu *Max Pooling* [5], yaitu memilih nilai maksimum dalam suatu area tertentu. Proses pooling umumnya menggunakan filter berukuran 2x2 dengan ukuran langkah sebanyak 2 langkah. Operasi *Max Pooling* ditujukan pada Gambar 2.9



Gambar 2.9 *Max Pooling*

Dari Gambar 2.8, filter *Max Pooling* yang digunakan yaitu 2x2 dengan *stride* (langkah) sebanyak 2. Mula-mula filter *pooling* ditempatkan di pojok kanan atas dari matriks lalu filter tersebut akan mencari nilai maksimum pada area filter 2x2 tersebut, setelah nilai maksimum pada area tersebut terpilih maka filter akan bergeser sebanyak 2 langkah, kemudian mencari lagi nilai maksimum dari area tersebut, begitu seterusnya sampai semua pada area pada matriks dilalui oleh filter tersebut. Proses ini akan menyebabkan dimensi pada matriks menjadi berkurang sehingga dapat mempercepat proses komputasi.

2.9.4 Fully-Connected Layer

Fully-Connected Layer adalah *layer* yang biasanya digunakan dalam penerapan MLP dan bertujuan untuk melakukan transformasi pada dimensi data agar data

dapat diklasifikasikan secara linear [22]. Setiap *neurons* memiliki koneksi penuh ke semua aktivasi dalam lapisan sebelumnya [5].

Perbedaan antara *Fully-Connected Layer* dengan *Convolution Layer* adalah neuron di *Convolution Layer* terhubung hanya ke daerah tertentu pada input, sementara *Fully-Connected Layer* memiliki neuron yang secara keseluruhan terhubung [22]. *Fully-Connected Layer* berperan untuk mengklasifikasi data masukan.

Sebelum masuk ke *Fully-Connected Layer*, ouput dari layer sebelumnya terlebih dahulu ditransformasikan menjadi bentuk vektor satu dimensi. Istilah ini disebut sebagai proses *flatten*. Di dalam *Fully-Connected Layer* terdapat *hidden layer* dan *output layer*. Bentuk persamaan dari *hidden layer* dan *output layer* adalah sebagai berikut.

a. Hidden Layer

$$z\_in_i = \sum_{i=1}^{n} X_j * V_{j,i} + V_{o,i} \quad (2.9)$$

dimana: z_in_i : masukkan untuk node *hidden layer* Z ke-i dengan jumlah node n

X_j : node X ke-j

$V_{j,i}$: bobot V untuk X_j dan node Z_i

$V_{o,i}$: bias V untuk z_in_i

b. Output Layer

$$y\_in_i = \sum_{i=1}^{m} Z_j * W_{j,i} + W_{o,i} \quad (2.10)$$

dimana: y_in_i : masukkan untuk node *hidden layer* Z ke-i dengan jumlah node m

Z_j : node Z ke-j

$W_{j,i}$: bobot W untuk Z_j dan node Y_i

$W_{o,i}$: bias W untuk y_in_i

2.10 Softmax Classifier

Softmax Classifier merupakan standar fungsi yang digunakan ketika proses klasifikasi melibatkan lebih dari dua kelas [24]. Bentuk persamaan *Softmax Classifier* adalah sebagai berikut.

$$Y_i = \frac{e^{y\_in_i}}{\sum_{i=1}^{m} e^{M}} \quad (2.11)$$

Keterangan: Y_i : keluaran untuk *output layer* ke-i

y_in_i : masukan untuk *node layer* ke-i

M : semua masukan untuk *output layer* sejumlah *m* buah

e : nilai 2.7182…

2.11 Loss Function

Loss Function merupakan fungsi untuk menghitung kerugian yang terkait dengan semua kemungkinan yang dihasilkan oleh suatu model. *Loss function* bekerja ketika model pembelajaran memberikan kesalahan yang harus diperhatikan. *Loss Function* yang baik adalah fungsi yang menghasilkan error seminimal mungkin. Pada penelitian ini *loss function* yang digunakan yaitu *Cross Entrophy Function*. Adapun persamaannya adalah sebagai berikut.

$$L = - \sum_{i}^{m} t_i \log(Y_i) \quad (2.12)$$

Keterangan: L : nilai *loss function*

t_i : nilai vektor yang diharapkan

m : jumlah kelas keluaran

Y_i : nilai keluaran ke-i sejumlah kelas *m* buah

2.12 Backpropagation

Backpropagation adalah salah satu algoritma yang digunakan pada Jaringan Syaraf Tiruan (JST). *Backpropagation* dilakukan dalam dua tahap, yaitu:

1. *Feedforward*

   Pola yang dilatih diset ke setiap unit di *input layer*, lalu *output* yang dihasilkan di transmisikan ke layer selanjutnya sampai ke *output layer* [22].

2. *Backpropagation*

Setiap bobot disesuaikan agar menghasilkan *error* seminimal mungkin mulai dari bobot yang terhubung ke *output layer* sampai ke *input layer* [22]. Tahapan *backpropagation* adalah sebagai berikut:

1. Hitung *loss function* dengan persamaan

$$L = -\sum_{i}^{m} t_i \log(Y_i) \tag{2.13}$$

2. Hitung gradien kesalahan terhadap bobot W_{ji} menggunakan *chain rule*

$$\frac{\partial L}{\partial W_{ji}} = \frac{\partial L}{\partial Y_i}\frac{\partial Y_i}{\partial y\_in_i}\frac{\partial y\_in_i}{\partial W_{ji}} \tag{2.14}$$

3. Hitung gradien kesalahan terhadap bobot V_{kj} menggunakan *chain rule*

$$\frac{\partial L}{\partial V_{kj}} = \frac{\partial L}{\partial Y_i}\frac{\partial Y_i}{\partial y\_in_i}\frac{\partial y\_in_i}{\partial Z_j}\frac{\partial Z_j}{\partial z\_in_i}\frac{\partial z\_in_i}{\partial V_{kj}} \tag{2.15}$$

4. *Update* nilai paramater menggunakan metode *Adam Optimizer* [25]. Adapun langkah-langkahnya sebagai berikut.

a. Hitung nilai m_t dengan persamaan berikut

$$m_t = \beta_1 * m_{t-1} + (1-\beta_1) * g_t \tag{2.16}$$

Keterangan: m_t = Nilai momentum pertama

β_1 = Konstanta Laju Peluruhan (0.9)

m_{t-1} = Inisialisasi vektor momentum pertama

g_t = Nilai gradien bobot

b. Hitung nilai v_t dengan persamaan berikut

$$v_t = \beta_2 * v_{t-1} + (1-\beta_2) * g_t^2 \tag{2.17}$$

Keterangan: v_t = Nilai momentum kedua

β_2 = Konstanta Laju Peluruhan (0.999)

v_{t-1} = Inisialisasi vektor momentum kedua

g_t = Nilai gradien bobot

c. Hitung nilai $\breve{m}_t$ dengan persamaan berikut

$$\breve{m}_t = \frac{m_t}{\sqrt{1 - \beta_1^t}} \tag{2.18}$$

Keterangan: $\breve{m}_t$ = Nilai momentum pertama dengan pembaruan bias

β_1 = Konstanta Laju Peluruhan (0.9)

m_t = Nilai momentum pertama

t = *Epoch*

d. Hitung nilai $\breve{v}_t$ dengan persamaan berikut

$$\breve{v}_t = \frac{v_t}{\sqrt{1 - \beta_2^t}} \tag{2.19}$$

Keterangan: $\breve{v}_t$ = Nilai momentum kedua dengan pembaruan bias

β_2 = Konstanta Laju Peluruhan (0.999)

v_t = Nilai momentum Kedua

t = *Epoch*

e. *Update* parameter dengan persamaan berikut

$$\theta_t = \theta_{t-1} - \alpha\left(\frac{\breve{m}_t}{\sqrt{\breve{v}_t + \varepsilon}}\right) \tag{2.20}$$

Keterangan: θ_t = Nilai bobot yang telah diperbaharui

θ_{t-1} = Nilai bobot lama

$\breve{m}_t$ = Nilai momentum pertama dengan pembaruan bias

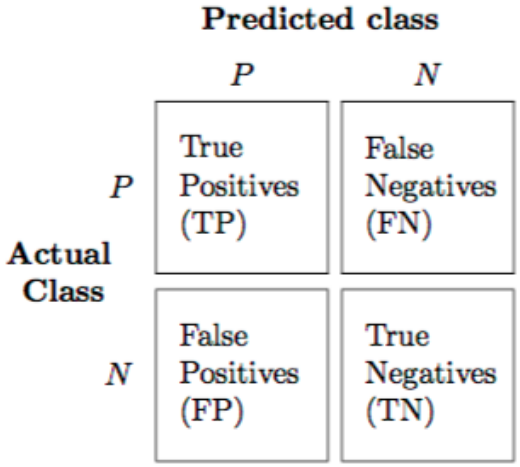
$\breve{v}_t$ = Nilai momentum kedua dengan pembaruan bias

α = Laju Pembelajaran (*Learning Rate*)

ε = Konstanta Epsilon (10^{-8})

2.13 Confussion Matrix

Confussion Matrix adalah sebuah tabel yang menyatakan jumlah data uji yang benar diklasifikasikan dan jumlah data uji yang salah diklasifikasikan [26]. Confussion Matrix merupakan matriks yang terdiri dari jumlah *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), *False Negative* (FN). TP dan TN menggambarkan informasi yang benar sedangkan FP dan FN menggambarkan informasi yang salah. Adapun gambaran *confussion matrix* ditujukan pada Gambar 2.10



Gambar 2.10 *Confussion Matrix*

Adapun untuk perhitungan tingkat akurasi menggunakan persamaan (2.19) sebagai berikut.

$$Akurasi = \frac{TP + TN}{TP + FN + FP + TN} \text{ x } 100\% \qquad (2.19)$$

Keterangan: TP : *True Positive*

TN : *True Negative*

FP : *False Positive*

FN : *False Negative*

2.14 UML (Unified Modelling Language)

Unified Modelling Language (UML) adalah bahasa pemodelan untuk sistem atau perangkat lunak yang berparadigma berorientasi objek. Abstraksi konsep dasar UML terdiri dari *structural classification, dynamic behavior, dan model*

management dapat kita pahami *main concepts* sebagai term yang akan muncul pada saat membuat diagram dan *view* adalah kategori dari diagram tersebut [27].

UML mendefinisikan beberapa diagram antara lain: *Class Diagram, Package Diagram, Use Case Diagram, Sequence Diagram, Communication Diagram, Statechart Diagram, Activity Diagram, Component Diagram dan Deployment Diagram* [28].

Dalam pengembangan perangkat lunak pada penelitian ini, hanya menggunakan 3 diagram UML yaitu *Use Case Diagram, Activity Diagram, dan Sequence Diagram*.

2.14.1 Use Case Diagram

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah use case merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, membuat sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. Adapun komponen komponen dalam *use case diagram* diantaranya:

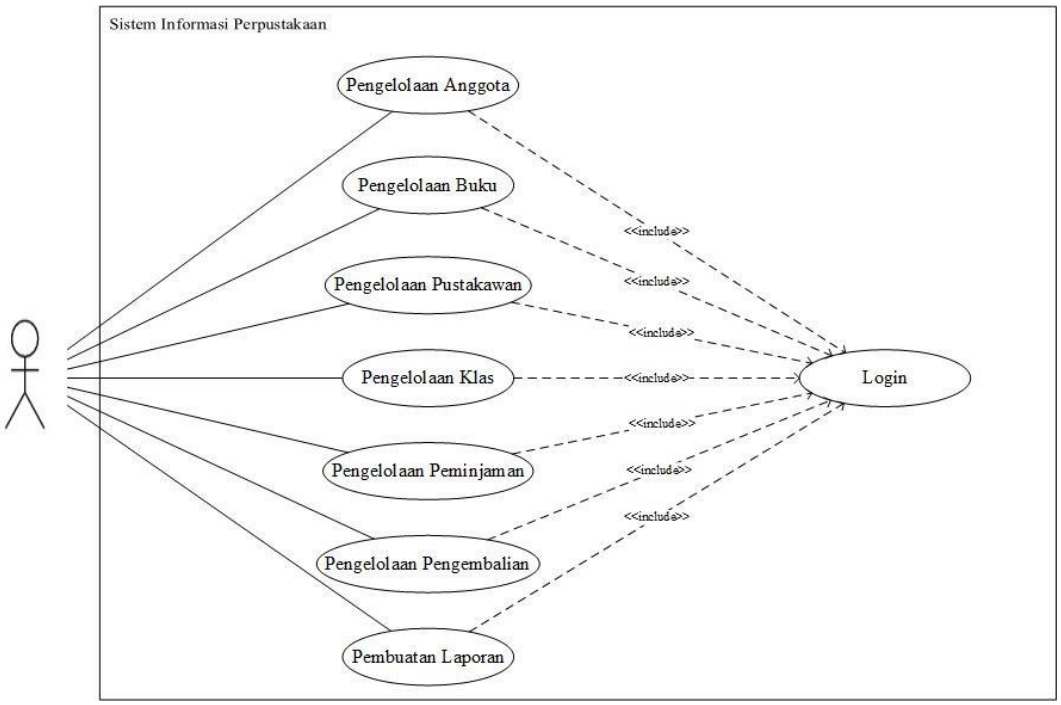
1. Aktor

Aktor merupakan suatu entitas yang berkaitan dengan sistem tapi bukan dari bagian dalam sistem itu sendiri. Aktor berbeda diluar sistem namun berkaitan erat dengan fungsionalitas didalamnya. Aktor dapat memiliki hubungan secara langsung terhadap fungsi utama baik terhadap salah satu atau semua fungsionalitas utama. Aktor juga dapat dibagi terhadap berbagai jenis atau tingkatan dengan cara digeneralisasi atau dispesifikasi tergantung kebutuhan sistemnya. Aktor biasanya dapat berupa pengguna atau database yang secara pandang berada dalam suatu ruang lingkup sistem tersebut.

2. *Use Case*

Use case merupakan gambaran umum dari fungsi proses utama yang menggambarkan tentang salah satu perilaku sistem. Perilaku sistem ini terdefinisi dari proses bisnis sistem yang akan dimodelkan. Tidak semua proses bisnis

digambarkan secara fungsional pada *use case*, tetapi yang digambarkan hanya fungsionalitas utama yang berkaitan dengan sistem. *Use case* menitikberatkan bagaimana suatu sistem dapat berinteraksi baik antar sistem maupun di luar sistem.

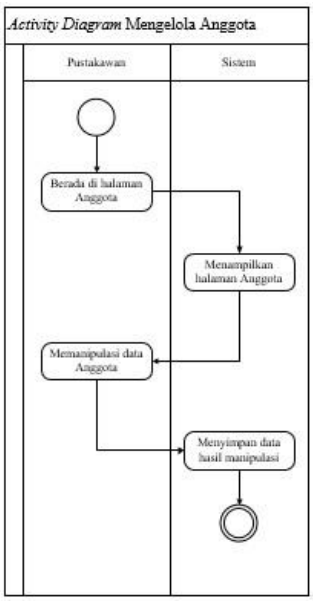


Gambar 2.11 ***Use Case Diagram***

2.14.2 Activity Diagram

Activity diagram menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, keputusan yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

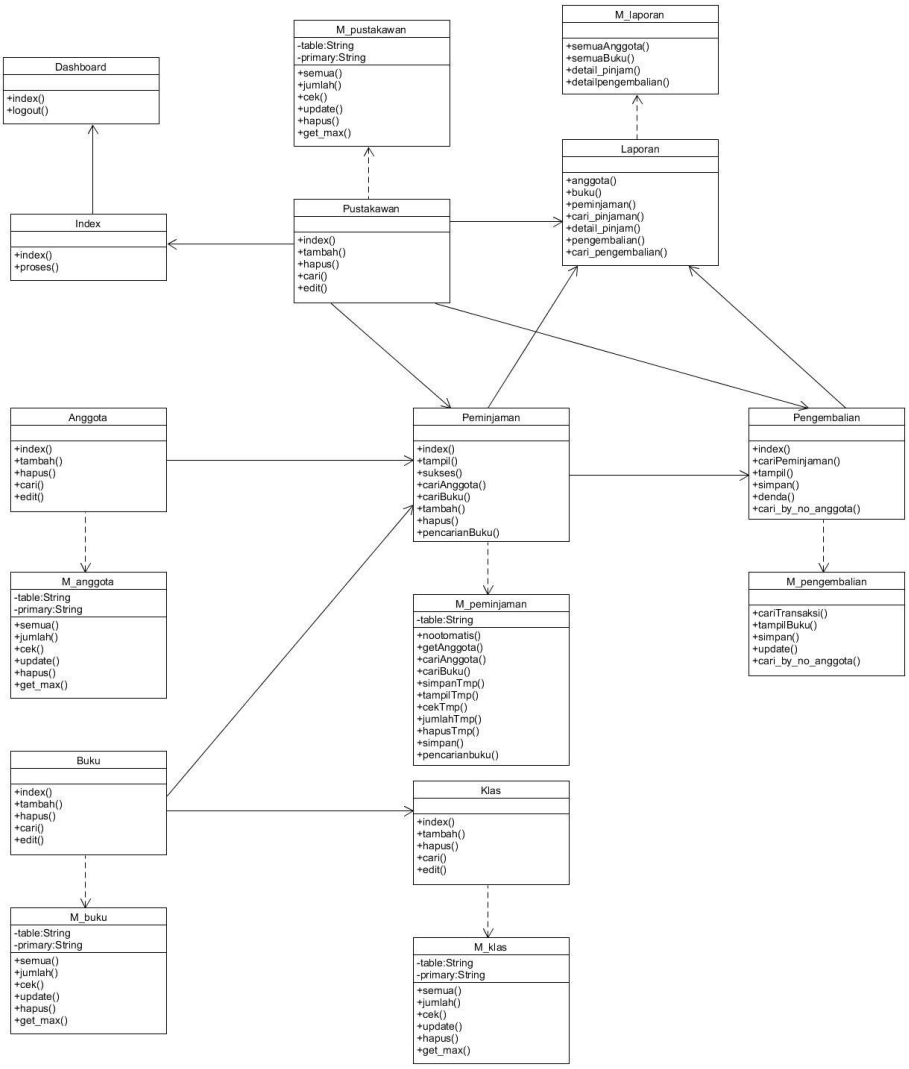
Activity diagram merupakan *state diagram* khusus, di mana sebagian besar state adalah aksi dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan *behaviour internal* sebuah sistem (dan interaksi antar sub sistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.



Gambar 2.12 ***Activity Diagram***

2.14.3 Class Diagram

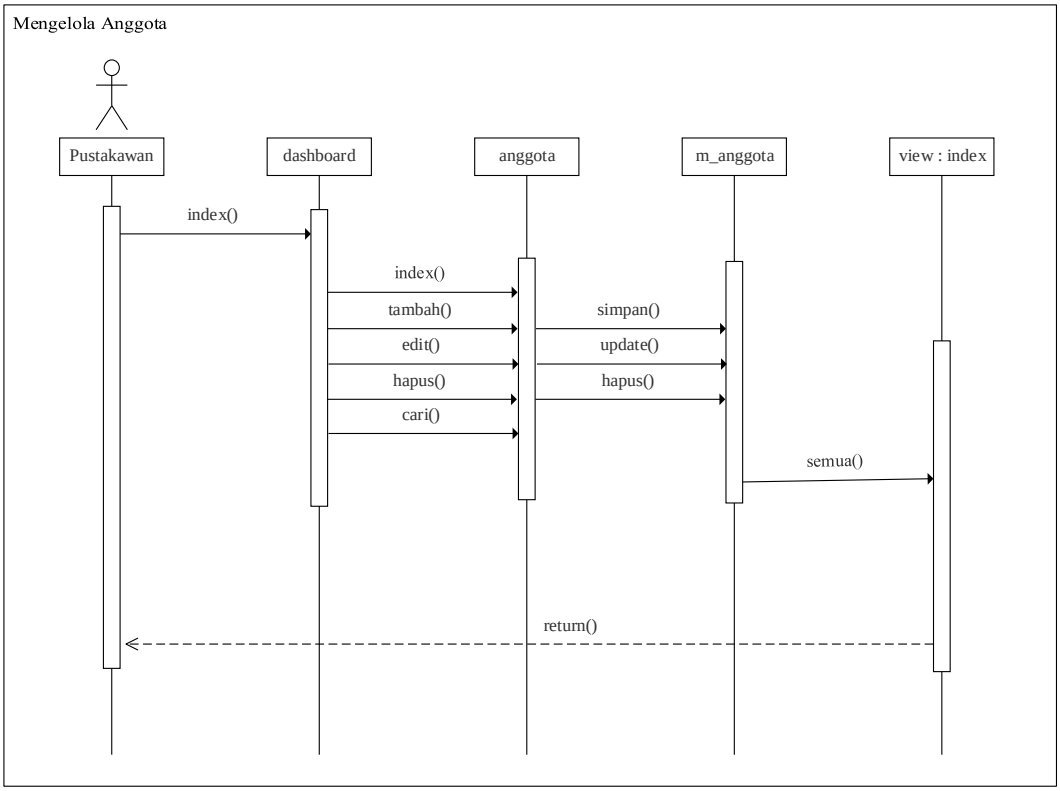
Class Diagram adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. Class menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metode/fungsi).



Gambar 2.13 *Class Diagram*

2.14.4 Sequence Diagram

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, tampilan, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atas dimensi vertikal (waktu) dan dimensi horisontal (objek-objek yang terkait). *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respon dari sebuah event untuk menghasilkan output tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan output apa yang dihasilkan.



Gambar 2.14 *Sequence Diagram*

2.15 Python

Python adalah bahasa pemrograman interpretatif multiguna. Tidak seperti bahasa lain yang susah untuk dibaca dan dipahami, *python* lebih menekankan pada keterbacaan kode agar lebih mudah untuk memahami sintaks. Hal ini membuat *Python* sangat mudah dipelajari baik untuk pemula maupun untuk yang sudah menguasai bahasa pemrograman lain [29].

Bahasa ini muncul pertama kali pada tahun 1991, dirancang oleh seorang bernama Guido van Rossum. Sampai saat ini *Python* masih dikembangkan oleh *Python Software Foundation*. Bahasa *Python* mendukung hampir semua sistem operasi, bahkan untuk sistem operasi Linux, hampir semua distronya sudah menyertakan *Python* di dalamnya.

Dengan kode yang simpel dan mudah diimplementasikan, seorang programmer dapat lebih mengutamakan pengembangan aplikasi yang dibuat, bukan malah sibuk mencari *syntax error*.

2.16 OpenCV

OpenCV merupakan salah satu *library open-source* yang digunakan untuk berbagai macam operasi pemrosesan gambar dan video. Beberapa fitur seperti pengenalan wajah atau pelacakan objek dapat dibuat dengan *OpenCV* [30]. *Library* ini tersedia untuk beberapa bahasa pemograman yaitu bahasa C, C++, dan Python serta kompatibel untuk sistem operasi Windows, Linux, dan MacOS[31].

Adapun contoh *syntax* dari OpenCV yang digunakan pada penelitian ini adalah sebagai berikut.

Tabel 2.1 Contoh *Syntax* Pada OpenCV

Contoh Code	Fungsi	Keterangan
`cv2.Resize(img, dsize, interpolation)`	Mengubah ukuran citra	`img` = citra yang akan dirubah ukurannya `dsize` = ukuran citra `interpolation` = metode interpolasi
`cv2.cvtColor(img, cv.COLOR_BGR2GRAY)`	Mengubah citra RGB ke *Grayscale*	`img` = citra yang akan dirubah jenis warnanya `cv.COLOR_BGR2GRAY` = fungsi untuk mengubah warna citra dari RGB ke Grayscale
`cv.imread(img, flags)`	Membaca citra	`img` = citra yang akan dibaca `flags` = mode warna pada citra yang dibaca

2.17 NumPy

NumPy adalah sebuah *library* untuk komputasi ilmiah yang menggunakan bahasa pemrograman *Python*. *NumPy* menyediakan berbagai macam operasi seperti matriks, *sorting*, aljabar linear dasar, statistik dasar dan lain-lain.[32]

Adapun contoh *syntax* dari *NumPy* yang digunakan pada penelitian ini adalah sebagai berikut.

Tabel 2.2 Contoh *Syntax* Pada NumPy

Contoh Code	Fungsi	Keterangan
`numpy.random.seed(seed)`	Membangkitkan nilai acak	`seed` = diisi dengan bilangan bulat
`numpy.array()`	Membuat array	-
`numpy.zeros(shape, dtype)`	Membuat array bernilai 0	`shape` = ukuran array `dtype` = tipe data untuk angka di dalam array

2.18 Keras

Keras merupakan sebuah API untuk neural network yang ditulis dengan bahasa pemograman Python [33]. Keras menggunakan *TensorFlow*, *Theano* atau *CNTK* sebagai *backend* untuk menjalan kode dari model *neural network* yang dibuat seperti CNN maupun RNN. *Keras* dapat berjalan dengan CPU maupun dengan GPU.

Adapun contoh *syntax* dari *Keras* yang digunakan pada penelitian ini adalah sebagai berikut.

Tabel 2.3 Contoh *Syntax* Pada *Keras*

Contoh Code	Fungsi	Keterangan
`Sequential()`	*Class* model arsitektur *deep learning*	-
`Conv2D(kernel, filter, padding=, activation=, kernel_initializer=)`	Membuat *convolution layer*	`kernel` = jumlah filter konvolusi `filter` = ukuran filter konvolusi `padding` = menambahkan *padding* pada matriks input `activation` = fungsi aktivasi

		`kernel_initializer` = metode inisialisasi nilai pada filter
`MaxPool2D(pool_size=, strides=, padding=)`	Membuat *pooling layer* (*Max Pooling*)	`pool_size` = ukuran filter *pooling* `strides` = ukuran perpindahan `padding` = menambahkan *padding* pada matriks input


UNIVERSITAS KOMPUTER INDONESIA
UNIKOM