BAB 2
LANDASAN TEORI

2.1 Pucuk Daun Teh

Teh adalah hasil pengolahan pucuk (daun teh) dari tanaman teh (*Camellia sinensis*) yang dipakai sebagai bahan minuman. Teh yang baik dihasilkan dari bagian pucuk peko (pucuk paling ujung atau tunas yang sedang aktif) ditambah 2-3 helai daun muda, karena pada daun muda tersebut kaya akan senyawa polifenol. Istilah teh juga digunakan untuk minuman yang dibuat dari buah, rempah-rempah ataw tanaman obat lain yang di seduh, misal teh *rosehip*, *camomile*, *krisan* dan *jiaogulan*. Teh yang tidak mengandung daun teh disebut teh herbal.

Teh merupakan sumber alami kafein, teofilin dan antioksidan dengan kadar lemak, karbohidrat atau protein mendekati nol persen. Teh bila diminum terasa sedikit pahit yang merupakan kenikmatan tersendiri dari teh. Teh bunga dengan campuran kuncup bunga melati yang disebut teh melati atau teh wangi melati merupakan jenis teh yang paling popular di Indonesia. Konsumsi teh di Indonesia sebesar 0,8 kilogram per kapita per tahun masih jauh di bawah negara-negara lain di dunia, walaupun Indonesia merupakan negara penghasil teh terbesar nomor lima di dunia.

Penyakit atau hama pada daun teh yang dialami pada saat ini ada dua jenis adalah hama *Helopeltis* dan Hama *Blister blight* (cacar daun).

2.1.1 Hama *Helopeltis*

Hama ini menyerang daun muda, pucuk dan ranting-ranting muda dengan menusukkan stiletnya untuk menghisap isi sel daun serta mengeluarkan air liur yang beracun menyebabka kerusakan di sekitar jaringan tanaman yang ditusuknya. Komposisi kimia air liur hama ini penting untuk memanfaatkan cairan tanaman inang dan detoksifikasi senyawa kimia yang dikeluarkan tanaman. Menurut Sarker dan Mukhopadhyay (2006), *Helopeltis* memiliki enzim hidrolitik dan oksidoreduktase di dalam kelenjar ludah dan pada perut bagian tengah (*midgut*). Kedua tipe enzim tersebut berkaitan dengan extra-oral digestion dan pertahanan. Hal ini yang menyebabkan

terjadinya nekrosis jaringan dan fitotoksik pada daun teh. Sarker & Mukhopadhyay (2006) menambahkan enzim oksidoreduktase (katalase, peroksidase, dan polifenol-oksidase) bersifat sebagai pertahanan yang dapat melakukan detoksifikasi metabolit sekunder tanaman dan juga dapat menyebabkan fitotoksaemia. Enzim kaltalase dapat mencegah formasi quinone, peroksidase dapat mendegradasi korofil, sedangkan polifenol oksidase dan peroksidase mampu mengoksidasi senyawa fenol yang dihasilkan oleh tanaman [14].

Stadia pradewasa dan dewasa *Helopeltis*, mampu merusak daun dan pucuk teh. Kepik merobek jaringan daun dengan menusukkan stiletnya dan menghisap cairan tanaman. Bekas tusukan stiletnya akan menunjukan gejala berupa bercak-bercak yang tidak teratur (Gambar 2.1) [14].

Gambar 2. 1 Gejala Serangan Helopeltis

Pada titik tempat tusukan stilet akan terbentuk lingkaran transparan kemudian berubah warna menjadi coklat terang, akhirnya mengembang menjadi coklat kehitaman, bercak-bercak dan mongering dalam waktu 24 jam, terjadi penebalan dinding sel (Gambar 2.2) [14].

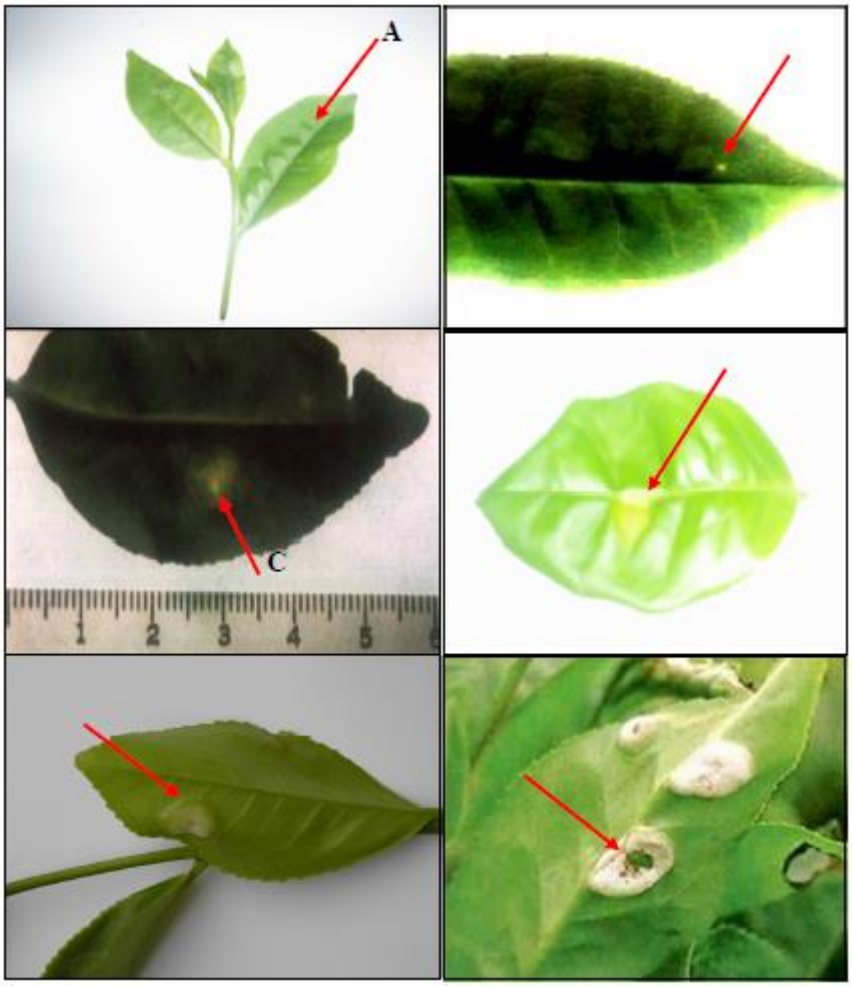
Gambar 2. 2 Bercak Sirkular Coklat Terjadi Penebalan Dinding Sel

2.1.2 Hama *Blister blight* (cacar daun)

Cacar daun dapat menyerang daun, tunas dan ranting-ranting yang masih muda. Pada tanaman yang terserang tampak adanya bitnik-bintik yang mula-mula berukuran kecil tetapi kemudian membesar mencapai ukuran 10-15 mm. Pada bagian bawah daun yang terserang tampak pada permukannya lapisan selaput yang berwarna putih, terdiri dari spora-spora (basidiospora) yang berjuta-juta jumlahnya. Dalam keadaan telah masuk (tua), spora-spora akan terlepas dan kemudian hinggap dan melekat pada daun atau ranting lain [15].

Dalam waktu kurang lebih lima hari seteah menghasilkan spora, jamur mati. Bagian daun atau ranting yang terserang mongering kemudian mati. Setelah beberapa hari, bekas-bekas serangan lapuk dan menimbulkan lubang-lubang pada daun. Serangan yang hebat menggugurkan daun perdu teh dan menurunkan kuantitas maupun kualitas produksi. Serangan ini akan lebih hebat bila keadaan kebun tidak mendukung dengan cuacua sangat lembab. Penyakit ini sangat berbahaya pada musim hujan.

Penyebab dari penyakit cacr daun teh ini adlah jamur *Exobasidium Vexans Massee*. Penularannya mudah karena spora tergolong halus dan mudah tersebar dibawa angina. Tanaman yang mempunyai kondisi fisik lemah bias hancur diserang. Bibit yang masih dipersemaian tidak lepas dari ancaman penyakit cacar daun. Contoh cacar daun dapat dilihat pada gambar 2.3 [15].



Gambar 2. 3 Perkembangan Gejala Cacar Daun Teh

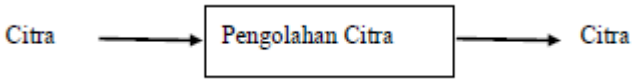
2.2 Citra Digital

Sebuah harifah, citra (*image*) adalah gambar pada bidang dwimatra (dua dimensi) [5]. Ditinjau dari sudut pandang matematis, citra merupakan fungsi menerus (*continue*) dari intensitas cahaya pada bidang dwimatra. Sumber cahaya menerangi objek kemudian objek memantulkan kembali sebagai dari berkas cahaya tersebut. Pantulan cahaya ini ditangkap oleh alat-alat optic, misalnya mata pada manusia, kamera, pemindai (*scanner*), dan sebagainya. Sehingga bayangan objek yang disebut citra tersebut terekam.

2.3 Pengolahan Citra

Pengolahan citra (*image processing*) adalah pemrosesan citra, khususnya dengan menggunakan computer, menjadi citra yang kualitasnya lebih baik [5]. Pengolahan citra ini sangan diperlukan karena walaupun citra sangat kaya dengan

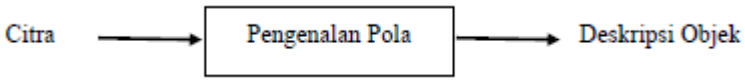
informasi, namun seringkali citra mengalami penurunan mutu (*degradasi*), misalnya mengandung cacat atau derau (*noise*), warnanya terlalu kontras, kurang tajam, kabur (*blurring*), dan sebagainya. Tentu saja citra semacam ini menjadi lebih sulit diinterpretasi karena informasi yang disampaikan oleh citra terebut menjadi berkurang.



Gambar 2. 4 Pengolahan Citra

2.4 Pengenalan Pola

Pengenalan pola merupakan kegiatan mengelompokan data numerik dan simbolik (termasuk citra) secara otomatis oleh mesin (dalam hal ini komputer) [5]. Tujuan pengelompokan adalah untuk mengenali suatu objek di dalam citra. Manusia bisa mengenali objek yang dilihatnya karena otak manusia telah belajar mengklasifikasi objek-objek di sekitarnya sehingga mampu membedakan suatu objek dengan objek lainnya. Kemampuan sistem visual manusia inilah yang dicoba ditiru oleh mesin. Komputer menerima masukan berupa citra objek yang akan diidentifikasi, memproses citra tersebut, dan memberikan keluaran berupa deskripsi objek di dalam citra.



Gambar 2. 5 Pengenalan Pola

2.5 Model Warna Pada Citra

Warna adalah persepsi yang dirasakan oleh system visual manusia terhadap Panjang gelombang cahaya yang dipantulkan oleh objek [5]. Setiap warna mempunyai panjang gelombang (1) yang berbeda. Warna merah mempunyai panjang gelombang paling tinggi, sedangkan warna ungu (violet) mempunyao panjang gelombang paling rendah.

Warna-warna yang diterima oleh mata (system visual manusia) merupakan hasil kombinasi cahaya dengan panjang gelombang berbeda. Penelitian memperlihatkan bahwa kombinasi warna yang memberikan rentang warna yang paling lebar adalah *red* (R), *green* (G) dan *blue* (B).

2.5.2 Citra RGB

Citra RGB, yang disebut juga citra "true color", disimpan dalam citra berukuran (m x n) x 3 yang mendefinisikan warna merah (*red*), hijau (*green*), dan warna biru (*blue*) untuk setiap pikselnya. Warna pada tiap piksel ditentukan berdasarkan kombinasi dan warna *red*, *green*, dan *blue* (RGB). RGB merupakan citra 24bit dengan komponen merah, hijau, biru yang masing-masing umumnya bernilai 8bit sehingga intensitas kecerahan warna sampai 256 level dan kombinasi warnanya kurang lebih sekitar 16 juta warna sehingga disebut "true color".

2.5.2 Citra Keabuan

Citra dengan derajat keabuan berbeda dengan citra RGB, citra ini didefinisikan oleh satu nilai derajat warna. Umumnya bernilai 8bit sehingga intensitas kecerahan warna sampai 256 level dan kombinasi warnanya 256 varian. Tingkat kecerahan paling rendah yaitu 0 untuk warna hitam dan putih bernilai 255.

Untuk mengkonfersikan citra yang memiliki warna RGB ke derajat keabuan bias menggunakan:

$$Gray = \frac{R+G+B}{3} \qquad (2.1)$$

atau

$$Gray = 0.299.R + 0.587.G + 0.114.B \qquad (2.2)$$

2.6 *Edge Detection* (*Sobel)*

Oprator sobel adalah salah satu operator yang menghindari adanya perhitungan gradien dititik interpolasi. Operator ini menggunakan kernel ukuran 3x3 piksel untuk perhitungan gradien sehingga perkiraan gradien berada tepat ditengah jendela.

Misal dibawah ini suatu pengaturan piksel di sekitar piksel (x, y)[6].

$$x = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} \qquad y = \begin{bmatrix} y_1 & y_2 & y_3 \\ y_4 & y_5 & y_6 \\ y_7 & y_8 & y_9 \end{bmatrix}$$

Operator *Sobel* adalah magnitude dari *Gradien* yang dihitung dengan:

$$G = \sqrt{x^2 + y^2} \qquad (2.3)$$

Dimana G adalah besaran gradien dititik tengah kernel dan turunan parsial dihitung menggunakan persamaan berikut:

$$x = (x_7 + 2x_8 + x_9) - (x_1 + 2x_2 + x_3) \quad (2.4)$$

$$y = (y_3 + 2y_6 + y_9) - (y_1 + 2y_4 + y_7) \quad (2.5)$$

x dan y dapat dinyatakan sebagai:

$$x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (2.6)$$

2.7 Standarisasi

Standarisasi merupakan salah satu proses dalam preprocessing citra yang dilakukan sebelum masuk ke proses pendeteksian. Proses untuk merubah atau menembalikan nilai yang dinormalkan dari tingkat intensitas cahaya pada citra tersebut, citra dipetakan pada *pixel* dengan ukuran tertentu sehingga memberikan representasi dimensi yang tetap. Bertujuan untuk menyesuaikan data citra pada basis data, proses normalisasi disesuaikan dengan kebutuhan pada proses pendeteksian yang digunakan [16]. Proses yang digunakan pada tahap *standarization* ini meliputi, menghitung *mean* (rata-rata) untuk citra pembelajaran dan menghitung nilai *standar deviasi* (menormalkan intensitas cahaya pada citra).

$$s_j = \frac{x_j - \bar{x}}{\sigma}$$

Dimana:

x_j= data ke-j

$\bar{x}$ = rata – rata

σ = standar deviasi

Rumus untuk menghitung *mean* sebagai berikut:

$$\bar{x} = \frac{(1,1)+(2,1)+(3,1)+\cdots+(n,n)}{n} \quad (2.7)$$

Sedangkan rumus untuk menghitung *standar deviasi* sebagai berikut:

$$\sigma = \sqrt{\frac{\sum_{j=1}^{n}(x_j-\bar{x})^2}{n-1}} \quad (2.8)$$

Keterangan:

σ : Standar Deviasi

x_j : nilai x ke-j

$\bar{x}$: Rata-rata

n : ukuran sampel

2.8 *Convolution*

Convolutional layer terdiri dari neuron yang tersusun sedemikian rupa sehingga membentuk sekumpulan filter dengan panjang dan tinggi (*pixels*). Masing-masing filter ini akan digeser keseluruh bagian dari gambar satu persatu. Setiap pergeseran akan dilakukan operasi "dot" antara input dan nilai dari filter tersebut sehingga menghasilkan sebuah output atau biasa disebut sebagai *activation map* atau *feature map*.

Ada 3 *hyperparameter* yang harus ditentukan dalam sebuah *convolutional layer*, yaitu *depth, stride* dan *zero-padding*.

1. Depth

   Depth adalah jumlah filter yang digunakan dalam suatu convolutional layer. Setiap filter akan mencari suatu ciri yang berbeda dalam masukan.

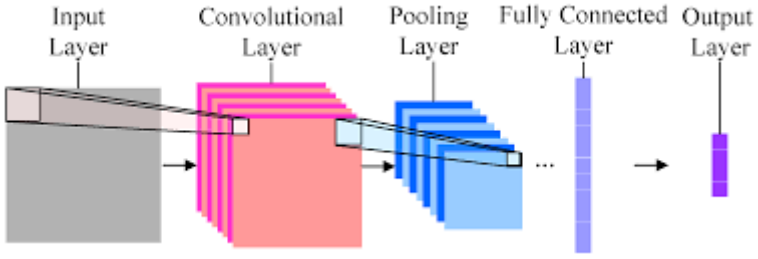
2. Stride

   Stride adalah parameter yang menentukan jumlah pergeseran filter. Jika nilai stride adalah 2, maka filter akan bergeser sebanyak 2 pixel secara horizontal lalu vertikal. Semakin kecil stride maka akan semakin detail informasi yang kita dapatkan dari sebuah input, namun membutuhkan komputasi yang lebih jika dibandingkan dengan stride yang besar.

3. Zero-Padding

   Zero-Padding adalah parameter yang menentukan jumlah pixel berisi nilai 0 yang akan ditambahkan di setiap sisi dari masukan. Hal ini digunakan dengan tujuan untuk memanipulasi dimensi output dari feature map. Dengan ditambahkannya zero-padding, maka ukuran citra output dan input akan tetap sama sehingga mengurangi informasi yang terbuang.

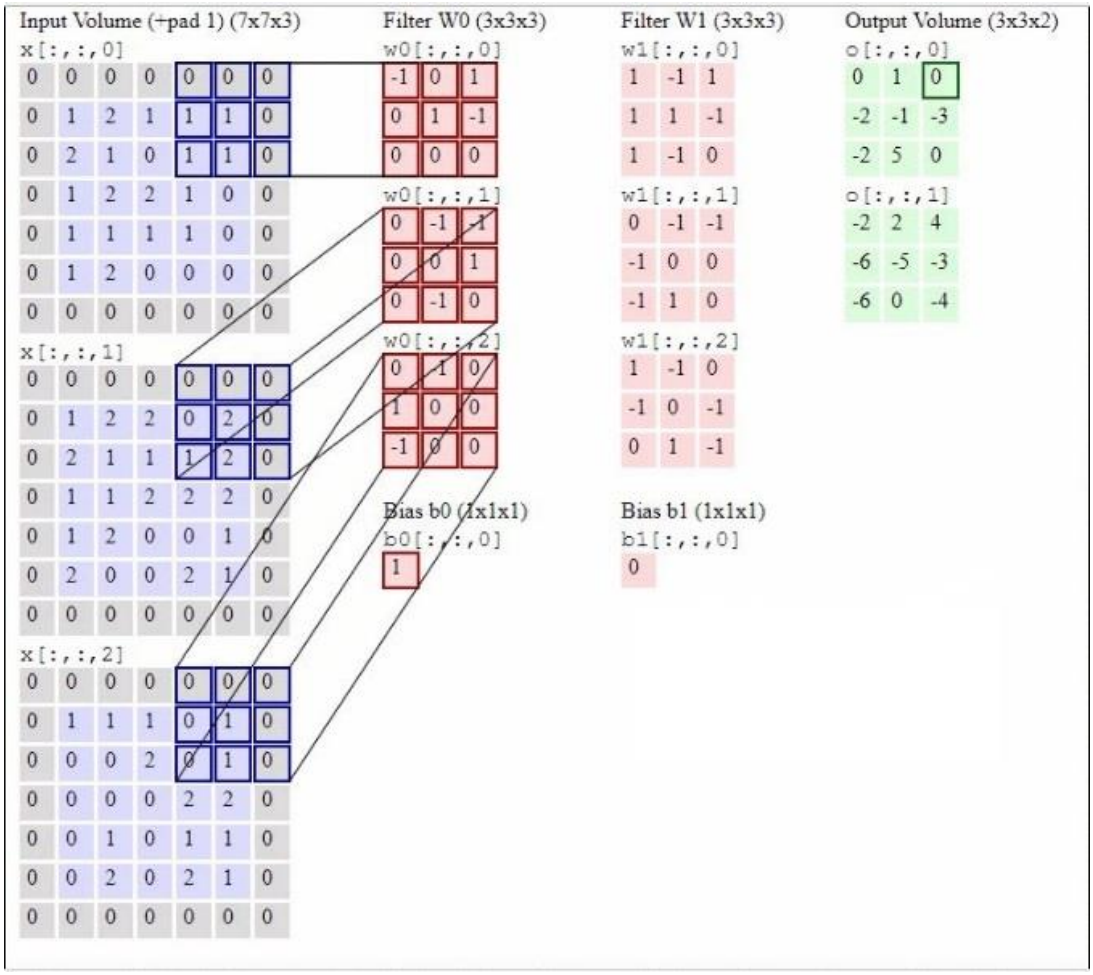
Gambar 2.6 menunjukkan contoh jaringan convolution.



Gambar 2. 6 Contoh Jaringan *Convolution*

Dapat dilihat pada gambar diatas dimana setiap lapisannya input yang dimasukkan memiliki volume yang berbeda dan diwakiki dengan kedalaman, tinggi dan lebar. Setiap besaran yang didapat tergantung dari hasil filterasi dari lapisan sebelumnya dan juga banyaknya filter yang digunakan. Model jaringan seperti ini sudah terbukti sangat ampuh dalam menangani permasalahan klasifikasi citra [7].

Dalam citra yang memiliki fungsi dua dimensi, operasi *convolution* didefinisikan pada gambar 2.7.



Gambar 2. 7 Operasi *Convolution Layer*

Konvolusi *image* dengan filter berbeda dapat melakukan operasi seperti deteksi tepi, buram dan pertajam citra dengan menerapkan filter.

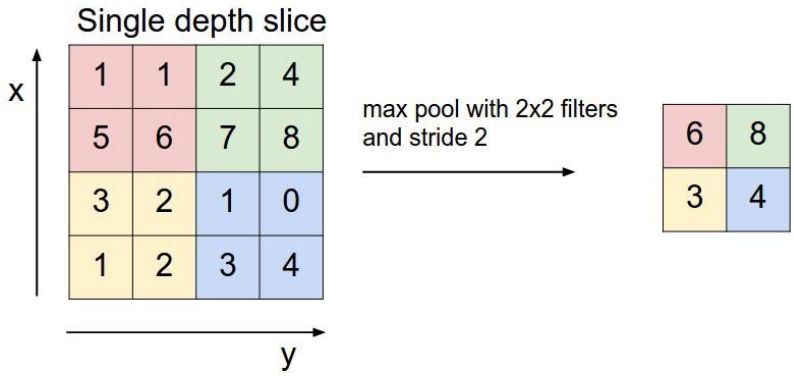
Sehingga *convolution* untuk $f(x, y)$ dapat didefinisikan sebagai berikut:

$$f(xy) = k1.(1,1) + k2.(2,1) + k3.(3,1) + k4.(1,2) + k5.(2,2) + k6.(3,2) + k7.(1,3) + k8.(2,3) + k9.(3,3) \quad (2.9)$$

Operasi *convolution* dilakukan dengan menggeser *kernel convolution pixel* per *pixel*/satu stride. Hasil *convolution* disimpan di dalam matriks yang baru.

2.9 *Pooling Layer*

Pooling Layer adalah lapisan yang menggunakan fungis *Feature Map* sebagai masukan dan mengolahnya dengan berbagai macam operasi stastistik berdasarkan nilai piksel terdekat. Pada model CNN, lapisan *Pooling* biasanya disisipkan secara teratur setelah beberapa lapisan konvolusi. Lapisan *Pooling* yang dimasukkan diantara lapisan konvolusi secara berturut-turut dalam arsitektur model CNN dapat secara progresif mengurangi ukuran volume output *Feature Map*, sehingga mengurangi jumlah parameter dan perhitungan di jaringan, dan untuk mengendalikan *Overfitting*. Hal terpenting dalam pembuatan CNN adalah dengan memilih banyak jenis lapisan *Pooling* dan hal ini bisa menguntungkan kinerja model. Lapisan *Pooling* bekerja di setiap tumpukan *Feature Map* dan mengurangi ukurannya. Bentuk lapisan *Pooling* yang paling umum adalah dengan menggunakan filter berukuran 2x2 yang diaplikasikan dengan langkah sebanyak 2 dan kemudian beroperasi pada setiap irisan dan input. Bentuk seperti ini akan mengurangi *Feature Map* hingga 75% dari ukuran aslinya. Berikut operasi Max Pooling dapat dilihat pada gambar 2.8.



Gambar 2. 8 Operasi Max Pooling

Fungsi *Max Pooling:*

$$P_j = \max((1,1), \dots, (n.n)) \qquad (2.10)$$

Lapisan *Pooling* akan beroperasi pada setiap irisan kedalaman volume input secara bergantian. Pada gambar di atas, lapisan *pooling* menggunakan salah satu operasi maksimal yang merupakan operasi yang paling umum. Contoh Jaringan

CNN menunjukkan operasu dengan langkah 2 dan ukuran filter 2x2. Dari ukuran input 4x4, pada masing-masing 4 angka pada input operasi mengambil nilai maksimalnya dan membuat ukuran output baru menjadi 2x2[12].

2.10 *Convolutional Neural Network*

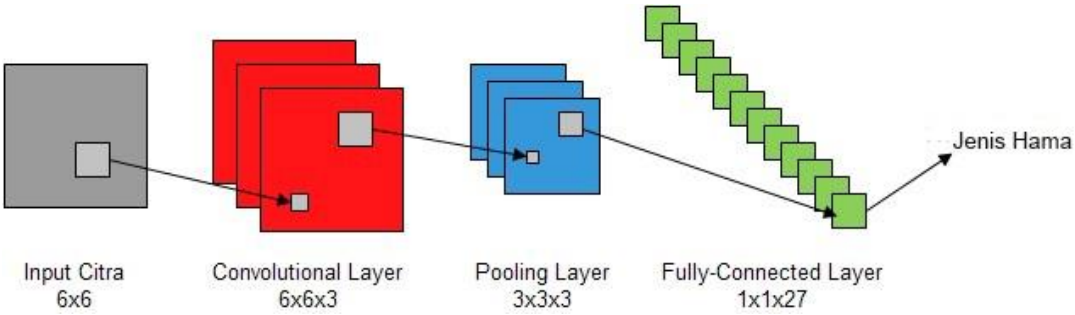
CNN adalah variasi dari *Multilayer Perceptron* yang terinspirasi dari jaringan syaraf manusia. Penelitian awal yang mendasari penemuan ini pertama kali dilakukan oleh Hubel dan Wiesel [19]. Yang melakukan penelitian *visual cortex* pada indera penglihatan kucing. *Visual cortex* pada hewan sangan *powerful* dalam system pemrosesan visual yang pernah ada. Hingga banyak penelitian yang terinspirasi dari cara kerjanya dan menghasilkan model-model baru diantaranya seperti Neocognitron [20], HMAX [21] dan LeNet-5 [22].

Convolutinal Neural Network merupakan suatu *layer* yang memiliki susunan *neuron* 3D (lebar, tinggi, kedalaman). Lebar dan tinggi merupakan ukuran *layer* sedangkan kedalaman mengacu pada jumlah *layer*. Secara umum jenis layer pada CNN dibedakan menjadi dua yaitu:

a. *Layer* ekstraksi fitur gambar, letaknya berbeda pada awal arsitektu tersusun atas beberapa *layer*, dan setiap *layer* tersusun atas *neuron* yang terkoneksi pada daerah local (*local region*) *layer* sebelumnya. *Layer* jenis pertama adalah *layer* konvolusi dan *layer* kedua adalah *layer pooling*. Setiap *layer* diberlakukan fungsi aktivasi. Posisinya berseling-seling antara jenis pertama dengan jenis kedua. *Layer* ini menerima input gambar secara langsung dan memprosesnya hingga keluaran berupa vektor untuk diolah pada *layer* berikutnya.
b. *Layer* klasifikasi, tersusun atas beberapa *layer* dan setiap *layer* tersusun atas *neuron* yang terkoneksi secara penuh (*fully connected*) dengan *layer* lainnya. *Layer* ini menerima input dari hasil keluaran *layer* ekstraksi fitur gambar berupa *hidden layer*. Hasil keluaran berupa skoring kelas untuk klasifikasi.

Dengan demikian CNN merupakan metode untuk mentransformasikan gambar original *layer* per *layer* dari nilai piksel gambar kedalam nilai skoring kelas untuk klasifikasi. Dan setiap *layer* ada yang memiliki *hyperparameter* dan ada yang

tidak memiliki parameter (bobot dan bias pada *neuron*). Analisis yang digunakan dalam implementasi convolution neural network dapat dilihat pada gambar 2.9.



Gambar 2. 9 Analisis *Convolutional Neural Network*

2.11 *Backpropagation*

Backpropagation adalah salah satu algoritma *supervised learning* yang digunakan dalam *artificial neural networks*. *Backpropagation* mencari kombinasi bobot untuk meminimalkan kesalahan output untuk dianggap menjadi solusi yang benar [28] . Secara sederhana *Backpropagation* dilakukan dalam dua tahap, yaitu:

1. *Feedfoward*

Pola yang akan dilatih diset ke setiap unit di *input layer*, lalu *output* yang dihasilkan ditransmisikan ke *layer* selanjutnya terus sampai *output layer*.

2. *Backpropagation*

Berdasarkan *output* yang diharapkan, setiap bobot disesuaikan agar menghasilkan *error* yang minimal mulai dari bobot yang terhubung ke *output neuron*, lalu terus mundur sampai ke *input layer*. Berikut tahap-tahap *backpropagation*.

1. Perhitungan *loss function* (*Cross Entropy Loss*)

$$\mathrm{L} = -\sum_i^m t_i \log(Y_i) \qquad (2.11)$$

2. Perhitungan gradien kesalahan terhadap parameter bobot W_{ji} menggunakan rumus *chain rule.*

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

$$\frac{\partial L}{\partial Y_i} = \frac{Y_i - t_i}{Y_i(1 - Y_i)}$$

$$\frac{\partial Y_i}{\partial y\_in_i} = Y_i(1 - Y_i)$$

$$\frac{\partial y\_in_i}{\partial W_{ji}} = Z_j$$

$$\frac{\partial L}{\partial W_{ji}} = (Y_i - t_i)Z_j \quad (2.12)$$

3. Perhitungan gradien kesalahan terhadap parameter bobot V_{kj} menggunakan rumus *chain rule*.

$$\frac{\partial L}{\partial V_{kj}} = \sum_i^m \frac{\partial L}{\partial Y_i}\frac{\partial Y_i}{\partial y\_in_i}\frac{\partial y\_in_i}{\partial Z_j}\frac{\partial Z_j}{\partial z\_in_j}\frac{\partial z\_in_j}{\partial V_{kj}}$$

$$\frac{\partial L}{\partial Y_i} = \frac{Y_i - t_i}{Y_i(1 - Y_i)}$$

$$\frac{\partial Y_i}{\partial y\_in_i} = Y_i(1 - Y_i)$$

$$\frac{\partial y\_in_i}{\partial Z_j} = W_{ji}$$

$$\frac{\partial Z_j}{\partial z\_in_j} = Z_j\left(1 - Z_j\right)$$

$$\frac{\partial z\_in_j}{\partial V_{kj}} = X_k$$

$$\frac{\partial L}{\partial V_{kj}} = \sum_i^m (Y_i - t_i)(W_{ji})(Z_j\left(1 - Z_j\right))(X_k) \quad (2.13)$$

4. *Update* nilai parameter filter W

$$W_{1,1}' = W_{1,1} - \alpha(\frac{\partial L}{\partial W_{1,1}}) \quad (2.14)$$

Parameter filter F

$$\mathrm{F}[1]_{1,1}' = \mathrm{F}[1]_{1,1} - \alpha(\frac{\partial \mathrm{P}[1]}{\partial \mathrm{F}[1]_{1,1}}) \quad (2.15)$$

Parameter filter V

$$V_{1,1}' = V_{1,1} - \alpha(\frac{\partial L}{\partial V_{1,1}}) \quad (2.16)$$

2.12 *Fully Connected Layer*

Lapisan *Fully-Connected* adalah lapisan di mana semua neuron aktivasi dari lapisan sebelumnya terhubung semua dengan neuron di lapisan selanjutnya seperti halnya jaringan syaraf tiruan biasa. Setiap aktivasi dari lapisan sebelumnya perlu diubah menjadi data satu dimensi sebelum dapat dihubungkan ke semua neuron di lapisan *Fully-Connected*. Lapisan *Fully-Connected* biasanya digunakan pada metode Multi Lapisan Perceptron dan bertujuan untuk mengolah data sehingga bisa diklasifikasikan.

Perbedaan antara lapisan *Fully-Connected* dan lapisan konvolusi biasa adalah neuron di lapisan konvolusi terhubung hanya ke daerah tertentu pada input, sementara lapisan *Fully-Connected* memiliki neuron yang secara keseluruhan terhubung. Namun, kedua lapisan tersebut masih mengoperasikan produk dot, sehingga fungsinya tidak begitu berbeda.

Fully Connected Layer pada intinya adalah sebuah neural netrowk multilayer perceptron (MLP), yang memiliki beberapa *hidden layer, activation function, output layer* dan *loss function*. *Fully connected layer*-lah yang akan berperan untuk mengklasifikasi data masukan. Bentuk persamaan *hidden layer* dan *output layer* sebagai berikut.

Hidden layer:

$$z\_in_i = \sum_{i=1}^{n} X_j * V_{j,i} + V_{0,i} \qquad (2.17)$$

z_in_i = masukan untuk *node hidden layer Z* ke-*i* dengan jumlah node *n*

X_j = *node X* ke-*j*

$V_{j,i}$ = *weight V* untuk *node Xj* dan *node Zi*

$V_{o,i}$ = bias *V* untuk *node* z_in_i

Output layer:

$$y\_in_i = \sum_{i=1}^{m} Z_j * W_{j,i} + W_{0,i} \qquad (2.18)$$

y_in_i = masukan untuk *node hidden layer Z* ke-*i* dengan jumlah node *m*

Z_j = *node Z* ke-*j*

$W_{j,i}$ = *weight W* untuk *node Zj* dan *node Yi*

$W_{o,i}$ = bias W untuk *node* y_in_i

Output yang dihasilkan dari *pooling layer* masih berbentuk *multidimensional array*, sehingga akan di-*flatten* terlebih dahulu untuk mengubah data menjadi vektor sebelum dijadikan input untuk *fully connected layer*

2.13 Aktivasi *ReLU*

Aktivasi *ReLU (Rectified inear Unit)* merupakan lapisan aktivasi pada model CNN yang mengaplikasikan fungsi f(x)=max(0,x) yang berarti fungsi ini melakukan *Thresholding* dengan nilai nol terhadap nilai piksel pada input citra. Aktivasi ini membuat seluruh nilai piksel yang bernilai kurang dari nol pada suatu citra akan dijadikan 0.

$$f(x) = max(0, x) \qquad (2.19)$$

2.14 *Softmax Classifier*

Softmax Classifier merupakan bentuk lain dari algoritma *Logistic Regression* yang dapat kita gunakan untuk mengklasifikasi lebih dari dua kelas. Standar klasifikasi yang biasa dilakukan oleh algoritma *Logistic Regression* adalah tugas untuk klasifikasi kelas biner. Pada *Softmax* bentuk persamaan yang muncul adalah sebagai berikut.

$$Y_i = \frac{e^{y\_in_i}}{\sum_{i=1}^{m} e^M} \qquad (2.20)$$

Y_i = keluaran untuk *node output layer* ke-*i*

y_in_i = masukan untuk *node output layer* ke-*i*

M = semua masukan untuk *node output layer*, berjumlah *m* buah

Softmax juga memberikan hasil yang lebih intuitif dan juga memiliki interpretasi probabilistic yang lebih baik disbanding algoritma klasifikasi lainnya. *Softmax* memungkinkan kita untuk menghittung probabilitas untuk semua label. Dari label yang ada akan diambil sebuah vector nilai bernilai riil dan merubahnya menjadi vector dengan nilai antara nol dan satu yang bila semua dijumlah akan bernilai satu.

2.15 *Crossentropy Loss Function*

Loss Function atau *Cost Function* merupakan fungsi yang menggambarkan kerugian yang terkait dengan semua kemungkinan yang dihasilkan oleh model. *Loss Function* bekerja ketika model pembelajaran memberikan kesalahan yang harus diperhatikan. *Loss Function* yang baik adalah fungsi yang menghasilkan error yang diharapkan paling rendah.

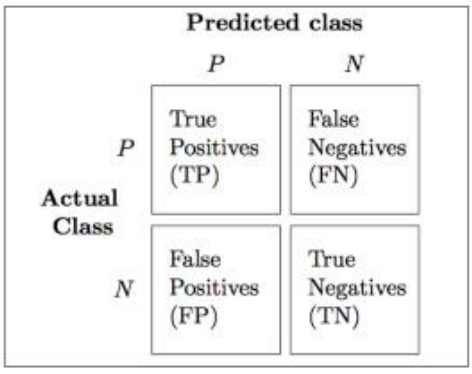
Ketika suatu model memiliki kelas yang cukup banyak, perlu adanya cara untuk mengukur perbedaan antara probabilitas hasil hipotesis dam probabilitas kebenaran yang asli, dan selama pelatihan banyak algoritma yang dapat menyesuaikan parameter sehingga perbedaan ini diminimalkan. *Crossentropy* adalah pilihan yang masuk akal.

Gambaran umum algoritma ini adalah meminimalkan kemungkinan log negatif dari dataset, yang merupakan ukuran langsung dari performa prediksi model.

$$\mathrm{L} = -\sum_i^m t_i \log(Y_i) \qquad (2.21)$$

2.16 *Confusion Matrix*

Confusion Matrix adalah matriks yang menjabarkan kinerja algoritma pembelajaran, *confusion matrix* sebuah matriks persegi yang melaporkan jumlah dari *True Positve, False Negatif, False Positive* dan *True Negative*. TP dan TN memberikan informasi ketika klasifikasi benar, sedangkan FP dan FN memberitahu ketika klasifikasi salah [24]. Seperti yang ditunjukan pada gambar 2.10.



Gambar 2. 10 *Confusion Matrix*

Perhitungan *confusion matrix* yang muncul adalah sebagai berikut:

$$Accuracy = \frac{coreectly\ predict\ data}{total\ testing\ data} \times 100\% \quad (2.22)$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \times 100\% \quad (2.23)$$

Keterangan:

TP = *True Positive*

TN = *True Negative*

FP = *False Positive*

FN = *False Negative*

2.17 Pemrograman Berorientasi Objek

Pemrograman berorientasi objek (Inggris: object-oriented programming disingkat OOP) merupakan paradigma pemrograman yang berorientasikan kepada objek. Ini adalah jenis pemrograman di mana programmer mendefinisikan tidak hanya tipe data dari sebuah struktur data, tetapi juga jenis operasi (fungsi) yang dapat diterapkan pada struktur data. Dengan cara ini, struktur data menjadi objek yang meliputi data dan fungsi. Selain itu, pemrogram dapat membuat hubungan antara satu benda dan lainnya. Sebagai contoh, objek dapat mewarisi karakteristik dari objek lain.

Salah satu keuntungan utama dari teknik pemrograman berorientasi obyek atas teknik pemrograman prosedural adalah bahwa memungkinkan programmer untuk membuat modul yang tidak perlu diubah ketika sebuah jenis baru objek ditambahkan. Seorang pemrogram hanya dapat membuat objek baru yang mewarisi banyak fitur dari objek yang sudah ada. Hal ini membuat program object-oriented lebih mudah untuk memodifikasi. Konsep dasar berorientasi objek diantaranya:

1. Kelas (*Class*) adalah kumpulan objek-objek dengan karakteristik yang sama. Kelas merupakan definisi statik dan himpunan objek yang sama yang mungkin lahir atau diciptakan dari kelas tersebut. Sebuah kelas akan mempunyai sifat (atribut), kelakuan (operasi/metode), hubungan (*relationship*) dan arti. Suatu kelas dapat diturunkan dan kelas semula dapat diwariskan ke kelas yang baru.
2. Objek (*Object*) adalah abstraksi dan sesuatu yang mewakili dunia nyata seperti benda, manusia, satuan organisasi, tempat, kejadian, strutur, status, atau hal-

hal lain yang bersifat abstrak. Objek merupakan suatu entitas yang mampu menyimpan informasi (status) dan mempunyai operasi (kelakuan) yang dapat diterapkan atau dapat berpengaruh pada status objeknya. Objek mempunyai siklus hidup yaitu diciptakan, dimanipulasi, dan dihancurkan.

3. Metode (*Method*) adalah operasi atau metode pada sebuah kelas hampir sama dengan fungsi atau prosedur pada terstruktur. Sebuah kelas boleh memiliki lebih dari satu metode atau operasi. Metode atau operasi yang berfungsi untuk memanipulasi objek itu sendiri.
4. Atribut (*Attribute*) dari sebuah kelas adalah variabel global yang dimiliki sebuah kelas. Atribut dapat berupa nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek. Atribut secara individual oleh sebuah objek, misalnya berat, jenis, nama, dan sebagainya.
5. Abstraksi (*Abstraction*) merupakan prinsip untuk merepresentasikan dunia nyata yang kompleks menjadi satu bentuk model yang sederhana dengan mengabaikan aspek-aspek lain yang tidak sesuai dengan permasalahan.
6. Enkapulasi (*Encapsulation*) adalah pembungkusan atribut data dan layanan (operasi-operasi) yang dipunyai objek untuk menyembunyikan implementasi dan objek sehingga objek lain tidak mengetahui cara kerja.
7. Pewarisan (*Inheritance*) adalah mekanisme yang memungkinkan satu objek mewarisi sebagian atau seluruh definisi dan objek lain sebagai bagian dari dirinya.
8. Antarmuka (*Interface*) sangat mirip dengan kelas, tetapi tanpa atribut kelas dan tanpa memiliki metode yang dideklarasikan. Antarmuka biasanya digunakan agar kelas lain tidak langsung mengakses ke suatu kelas.
9. Reusability merupakan pemanfaatan kembali objek yang sudah didefinisikan untuk suatu permasalahan pada permasalahan lainnya yang melibatkan objek tersebut.
10. Generalisasi dan Spesialisasi menunjukkan hubungan antara kelas dan objek yang umum dengan kelas dan objek yang khusus. Misalnya kelas yang lebih umum (generalisasi) adalah kendaraan darat dan kelas khususnya (spesialisasi) adalah mobil dan motor.

11. Komunikasi antar objek dilakukan lewat pesan (*message*) yang dikirim dan satu objek ke objek lainnya.
12. Polimorfisme (*Polymorphism*) adalah kemampuan suatu objek untuk digunakan dibanyak tujuan yang berbeda dengan nama yang sama sehingga menghemat baris program.
13. *Package* adalah sebuah kontainer atau kemasan yang daoat digunakan untuk mengelompokkan kelas-kelas sehingga memungkinkan beberapa kelas yang bernama sama disimpan dalam *package* yang berbeda [11].

2.18 *Unifed Modeling Language*

Unified Modeling Language (UML) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak. Unified Modeling Language (UML) adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya. UML adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat tool untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh Object Management Group, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP.UML merupakan dasar bagi perangkat (tool) desain berorientasi objek dari IBM.UML adalah suatu bahasa yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan suatu sistem informasi. UML dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson. Namun demikian UML dapat digunakan untuk memahami dan mendokumentasikan setiap sistem informasi. Penggunaan UML dalam industri terus meningkat. Ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industri peranti lunak dan pengembangan sistem [9].

2.18.1 Sejarah UML

Sampai era tahun 1990 puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: metodologi booch, metodologi coad, metodologi OOSE, metodologi OMT, metodologi shlaer-mellor, metodologi

wirfs-brock, dsb. Masa itu terkenal dengan masa perang metodologi (method war) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan kelompok/perusahaan lain yang menggunakan metodologi yang berlainan. Dimulai pada bulan Oktober 1994 Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease draft pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh Object Management Group[9].

2.18.2 Diagram UML

UML menyediakan 10 macam diagram untuk memodelkan aplikasi berorientasi objek[10], yaitu:

1. *Use Case Diagram* untuk memodelkan proses bisnis.
2. *Conceptual Diagram* untuk memodelkan konsep-konsep yang ada di dalam aplikasi.
3. *Sequence Diagram* untuk memodelkan pengiriman pesan (*message*) antar objek.
4. *Collaboration Diagram* untuk memodelkan interaksi antar objek.
5. *State Diagram* untuk memodelkan perilaku objek di dalam sistem.
6. *Activity Diagram* untuk memodelkan perilaku *Use Cases* dan objek di dalam sistem.
7. *Class Diagram* untuk memodelkan struktur kelas.
8. *Object Diagram* untuk memodelkan struktur objek.
9. *Component Diagram* untuk memodelkan komponen objek.
10. *Deployment Diagram* untuk memodelkan distribusi aplikasi.

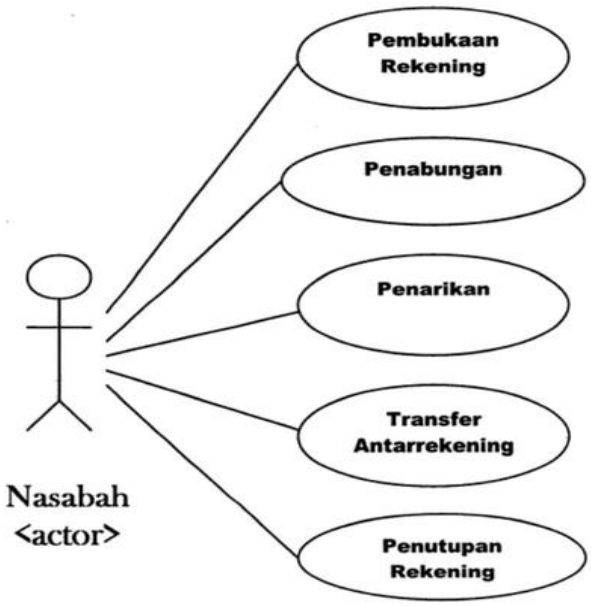
2.18.3 *Use Case* Diagram

Dalam konteks UML, tahap konseptualisasi dilakukan dengan pembuatan *use cas diagram* yang sesungguhnya merupakan deskripsi peringkat tinggi bagaimana perangkat lunak (aplikasi) akan digunakan oleh penggunanya. Selanjutnya, *use case diagram* tidak hanya sangat penting pada analisis, tetapi juga sangat penting untuk

perancangan (*design*), untuk mencari (mencoba menemukan) kelas-kelas yang terlibat dalam aplikasi, dan untuk melakukan pengujian (*testing*).

Membuat *use case diagram* yang komprehensif merupakan hal yang sangat penting dilakukan pada tahap analisis. Dengan menggunakan *use case diagram*, kita akan mendapatkan banyak informasi yang sangat penting yang berkaitan dengan aturan –aturan bisnis yang coba kita tangkap. Dalam hal ini, setiap objek yang berinteraksi dengan sistem/perangka lunak (misalnya, orang, suatu perangkat keras, sistem lain. Dan sebagainya) merupakan *actor* untuk sistem/perangkat lunak kita, sementara *use case* merupakan deskripsi lengkap tentang bagaimana sistem/perangkat lunak berprilaku untuk para *actor*-nya. Dengan demikian, *use case program* merupakan deskripsi lengkap tentang interaksi yang terjadi antara para *actor* dengan sistem/perangkat lunak yang sedang kita kembangkan.

Saat kita akan mengembangkan *use case diagram,* hal yang pertama kali kita lakukan adalah mengenali *actor* untuk sistem/aplikasi yang sedang kita kembangkan. Dalam hal ini, ada beberapa karakteristik untuk para *actor,* yaitu (1) *actor* ada diluar sistem yang sedang kita kembangkan dan (2) *actor* verinteraksi dengan sistem yang sedang kita kembangkan [10]. Contoh penggambaran use case diagram dapat dilihat pada gambar 2.11 Usecase Diagram.

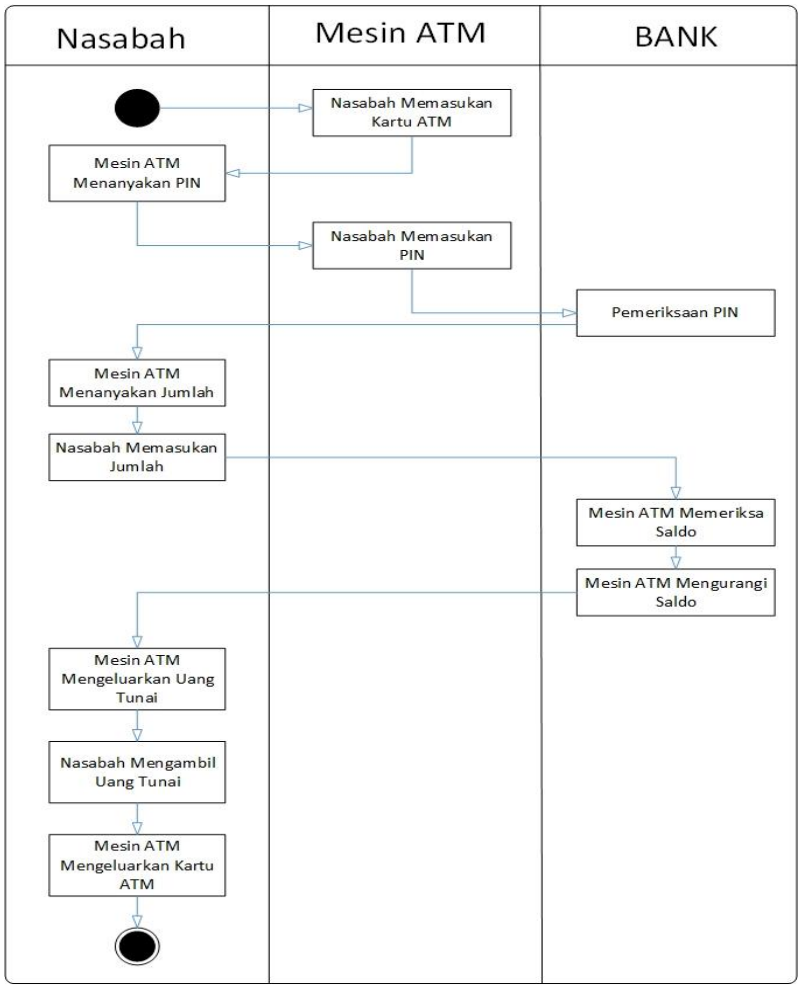


Gambar 2. 11 *Usecase* Diagram

2.18.4 *Activity* Diagram

Activity diagrams menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, decision 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 action 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 subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum. Sebuah aktivitas dapat direalisasikan oleh satu use case atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara use case menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

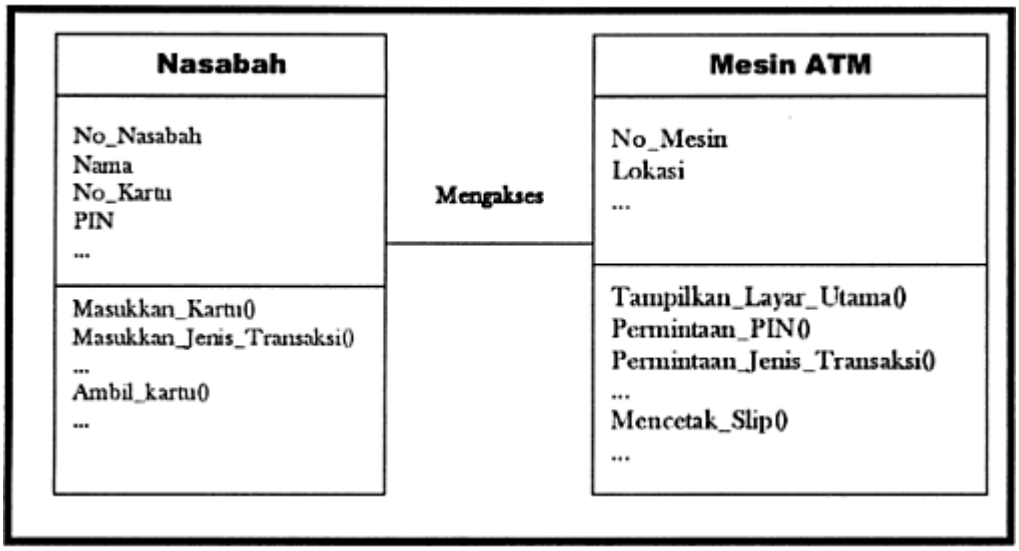
Apa langkah yang harus kita lakukan selanjutnya setelah kita membuat *use case diagram? Use case diagram* merupakan gambaran menyeluruh dan pada umumnya sangatlah tidak terperinci. Oleh karena itu, kita harus memperinci lagi perilaku sistem untuk masing-masing *use case* yang ada. Apa perkakas (*tool)* yang bisa menggunakan skenario seperti yang tercantum berikut, sementara jika kasusnya cukup kompleks, kita mungkin bisa menggunakan *activity diagram* agar bisa mendapatkan gambaran yang lebih menyeluruh. Contoh penggambaran class diagram dapat di lihat dalam gambar 2.12 *activity diagram* dibawah ini.



Gambar 2. 12 Activity Diagram

2.18.5 *Class* Diagram

Class diagram merupakan diagram yang selalu ada di permodelan sistem berorientasi objek. Class diagram menunjukkan hubungan antar class dalam sistem yang sedang dibangun dan bagaimana mereka saling berkolaborasi untuk mencapai suatu tujuan. Contoh penggambaran class diagram dapat di lihat dalam gambar 2.13 *class diagram* dibawah ini [9].



Gambar 2. 13 *Class* Diagram

Classs diagram menggambarkan interaksi dan relasi antar kelas yang ada di dalam suatu sistem. Kelas memiliki atribut dan metode. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas. Metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas. Atribut dan metode dapat memiliki salah satu sifat sebagai berikut:

1. *Private,* tidak dapat dipanggil dari luar kelas yang bersangkutan.
2. *Protected,* hanya dapat dipanggil oleh kelas yang bersangkutan dan anak-anak yang mewarisinya.
3. *Public,* dapat dipanggil oleh siapa saja.

Class diagram menggambarkan relasi atau hubungan antar kelas dari sebuah sistem. Berikut ini beberapa gambaran relasi yang ada dalam *class diagram*:

4. *Association*

   Hubungan antar class yang statis. *Class* yang mempunyai relasi asosiasi menggunakan *class* lain sebagai atribut pada dirinya.

5. *Aggregation*

   Relasi yang membuat *class* yang saling terikat satu sama lain namun tidak terlalu berkegantungan.

6. *Composition*

   Relasi agregasi dengan mengikat satu sama lain dengan ikatan yang sangat kuat dan saling berkegantungan.

7. *Dependency*

   Hubungan antar *class* dimana *class* yang memiliki relasi *dependency* menggunakan *class* lain sebagai atribut pada *method*.

8. *Realization*

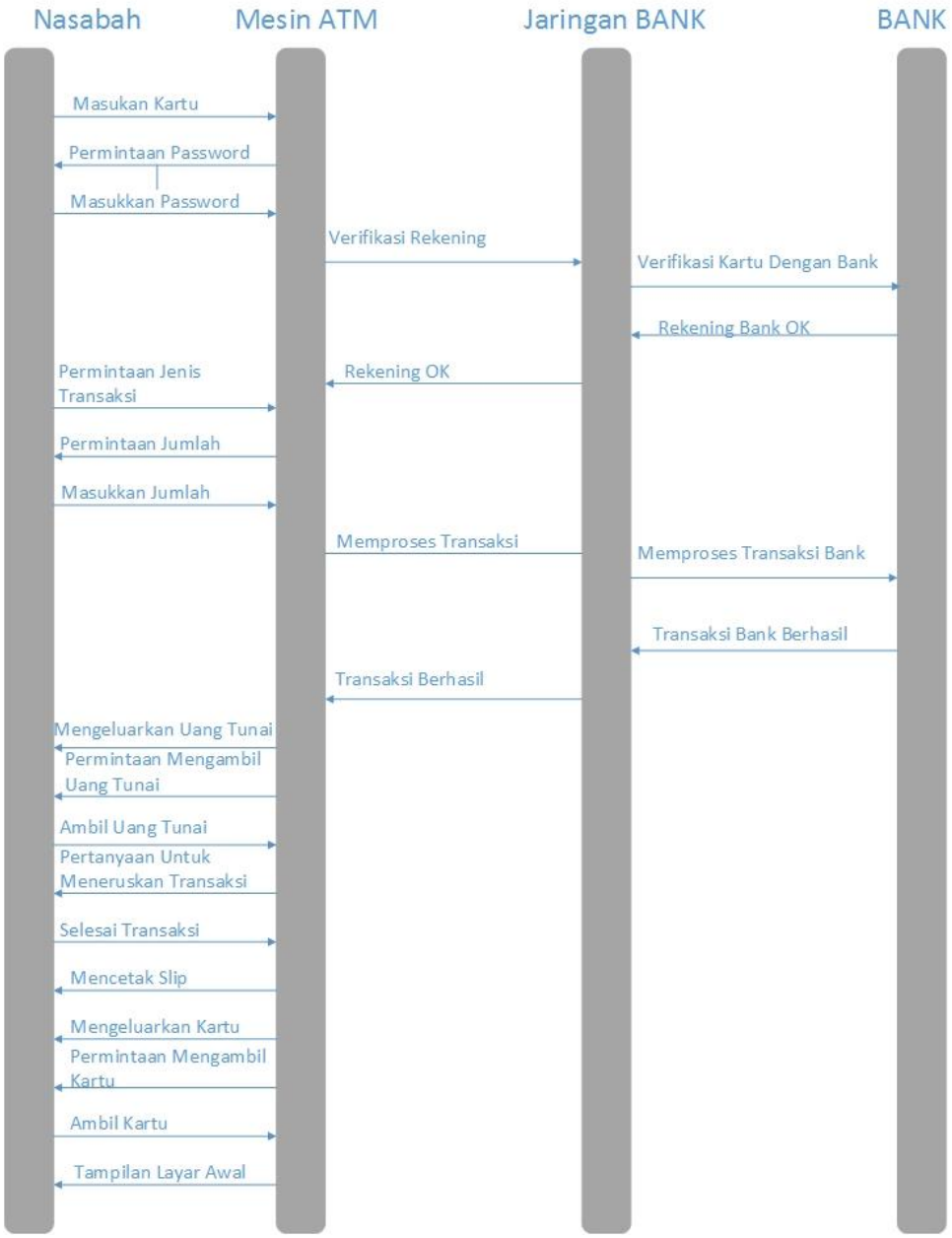
   Hubungan antar *class* dimana sebuah *class* memiliki keharusan untuk mengikuti aturan yang ditetapkan *class* lainnya.

2.18.6 *Sequence* Diagram

Sequence diagram menjelaskan secara detil urutan proses yang dilakukan dalam sistem untuk mencapai tujuan dari *use case*: interaksi yang terjadi antar *class*, operasi apa saja yang terlibat, urutan antar operasi, dan informasi yang diperlukan oleh masing-masing operasi [10]. *Sequence diagram* menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan *sequence diagram* maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

Banyaknya *sequence diagram* yang harus digambar adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada *sequence diagram* sehingga semakin banyak *use case* yang didefinisikan maka *sequence diagram* yang harus dibuat juga semakin banyak. Penomoran pesan berdasarkan urutan iteraksi pesan. Penggambaran letak pesan harus berurutan, pesan yang lebih atas dari lainnya adalah pesan yang berjalan terlebih dahulu.

Contoh *sequence diagram* dapat di lihat pada gambar 2.14 contoh *sequence diagram*



Gambar 2. 14 Contoh *Sequence* Diagram

2.19 *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.

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.

2.20 Python Flask

Flask adalah *microframework* untuk Bahasa pemrograman Python yang berbasis Werkzeug dan Jinja 2, sekaligus memiliki lisensi BSD yang berarti siapapun bias menggunakan flask untuk berbagai keperuan baik pribadi maupun komersial tanpa dipungut bayaran (*free software*) asalkan hak cipta flask tetap disertakan [13].

Keunggulan utama dari *framework* flask memang bias digunakan untuk menerapkan konsep MVC, namun tidak memaksa pengguna (*strict*) untuk menerapkannya, maka *developer* tetap bias menggunakan paradigma pemrograman terstruktur.

Masalah utama dalam menggunakan flask sebagai framework adalah karena terlalu sederhana maka *developer* perlu menerapkan desain yang *scalable* dan *modular* (penulisan *script* berdasarkan modul-modul) ketika membuat program dengan flask, salah satu contoh terbaik adalah penerapan konsep MVC.