# BAB II
# DASAR TEORI

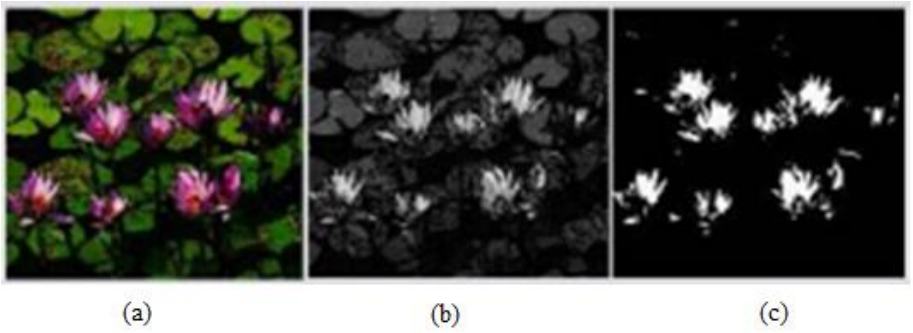
## 2.1 Citra Digital

Citra Digital adalah sebuah representasi nilai fungsi intensitas cahaya diskrit dalam sebuah bidang dua dimensi (Kholik, 2021). Citra digital tersusun dari satuan-satuan terkecil yang disebut *pixel (picture element)* dengan koordinat (x,y) dan intensitas f(x,y) (Wahyudi *et al.*, 2015). Persamaan 2.1 merupakan gambaran matriks yang terdapat pada sebuah citra (Marbun, 2017).

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,M) \\ f(1,0) & f(1,1) & \dots & f(1,M \\ \vdots & \vdots & & \vdots \\ f(N-1,0) & f(N-1,1) & & f(N-1,M-1) \end{bmatrix} \quad (2.1)$$

Persamaan 2.1 menunjukkan bahwa sebuah citra digital yang dianggap sebagai matriks memiliki indeks baris dan kolomnya dinyatakan dengan x dan y untuk menginterpretasikan koordinat setiap titik citra tersebut. Banyaknya jumlah baris (N) dan jumlah kolom (M) yang jika dikalikan akan menghasilkan piksel sebanyak N×M.

Citra digital menyimpan informasi warna yang berasal dari intensitas cahaya hasil dari *grid* sensor cahaya pada kamera. Pada dasarnya kombinasi gambar pada citra dibagi menjadi 3 jenis, yaitu citra RGB, citra *grayscale*, dan citra biner (Gazali *et al.*, 2012).




**Gambar 2. 1** (a) Citra RGB, (b) Citra *Grayscale*, (c) Citra Biner (Santi, 2011)

Gambar 2.1 menunjukkan bahwa citra RGB atau citra berwarna merupakan citra yang memiliki lebih banyak variasi warna jika dibandingkan citra *grayscale* dan citra biner. Umumnya citra RGB tersusun dari komponen merah (R), hijau (G), dan biru (B) yang diinterpretasikan ke dalam sebuah matriks sehingga dikenal dengan citra RGB (Atqiya *et al.*, 2019). Citra *grayscale* sendiri adalah citra yang memiliki skala keabuan

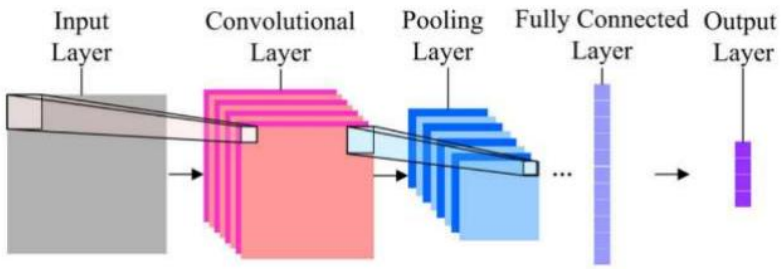
sebesar $2^8$ atau setara dengan 256 tingkat dengan rentang 0 sampai 255. Intensitas terbesar yaitu 255 berwarna putih dan intensitas terkecil yaitu 0 yang berwarna hitam. Citra biner merupakan citra yang hanya memiliki dua skala keabuan yaitu 0 yang menginterpretasikan warna hitam dan 1 yang menginterpetasikan warna putih (Wahyudi *et al.*, 2015).

Citra CT-*scan* termasuk dalam golongan citra grayscale. Citra CT-*scan* merupakan sebuah citra yang berasal dari hasil proses pemeriksaan CT-*scan* yang berguna untuk menampilkan penampang tubuh manusia tanpa harus dilakukan pembedahan. Citra CT-*scan* merupakan citra digital yang di dalamnya tersimpan data-data terkait keadaan tubuh pasien yang digunakan dalam interpretasi penyakit.

### *2.2 Convolutional Neural Network (CNN)*

*Convolutional Neural Network* (CNN) merupakan pengembangan *Artificial Neural Network* (ANN) yang memiliki kinerja yang meniru sistem jaringan saraf (neuron) pada otak manusia. Pada ANN, terdapat sekurang-kurangnya dua lapisan utama yaitu *input layer* (lapisan masukan) dan *output layer* (lapisan keluaran). Sedangkan pada *Multilayer Neural Network*, diantara lapisan masukan dan lapisan keluaran terdapat *hidden layer* (lapisan tersembunyi). Lapisan tersembunyi tersebut memiliki nilai node keluaran yang digunakan sebagai nilai node masukan pada lapisan lain setelahnya (Sakinah *et al.*, 2020).

CNN termasuk salah satu algoritma *deep learning* yang dapat digunakan untuk mengolah data citra dua dimensi dan dapat menghasilkan *output* dengan variasi kelas tertentu (Setiawan, 2019). CNN menjadi metode *supervised learning* karena dapat dimanfaatkan untuk mengklasifikasi data yang sudah memiliki label. CNN sering digunakan pada *computer vision* karena kemampuannya yang mampu melampaui level kinerja manusia dalam hal mengklasifikasi citra (Mawaddah *et al.*, 2021).

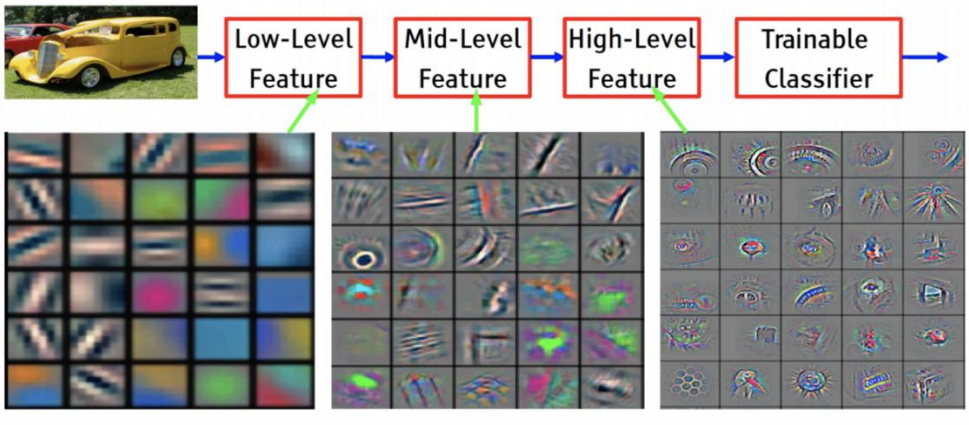



**Gambar 2. 2** Arsitektur CNN (Alwanda *et al*., 2020).

Gambar 2.2 merupakan gambaran arsitektur CNN secara umum yang digunakan untuk mengolah data 2 dimensi dalam bentuk citra. Arsitektur CNN umumnya berisi *input layer*, *hidden layer*, dan *output layer*. Pada gambar di atas, *hidden layer* diisi oleh *convolutional layer*, *pooling layer* dan *fully connected layer* (Alwanda *et al.*, 2020). Pada *input layer* ini gambar masukkan yang digunakan akan diteruskan untuk dilakukan ekstraksi fitur pada *layer* berikutnya (Sajja *et al.*, 2019). Pada *hidden layer*, *convolutional layer* memiliki tugas utama dalam mengekstraksi fitur yang telah diterima dari *input layer* dengan melakukan operasi konvolusi (Anugerah, 2018). Kemudian dilanjutkan dengan aktivitas *pooling* atau dikenal dengan pengurangan dimensi spasial sebuah *feature map* (Alwanda *et al.*, 2020). Sedangkan *fully connected layer* akan dilakukan transformasi multi dimensi menjadi dimensi satu yang mengubah bentuk spasial matriks tanpa menghilangkan neuron yang telah dihasilkan atau biasanya disebut dengan istilah *flattening*. Setelah dilakukan *flattening*, neuron yang dihasilkan dari proses pada *hidden layer* akan diklasifikasi (Mubarok, 2019).

### 2.2.1 Convolutional *Layer*

*Convolutional layer* adalah tahap pertama pada *feature learning* sekaligus langkah utama pada sebuah arsitektur CNN. *Convolutional layer* memiliki tugas komputasi paling kompleks diantara lapisan lainnya (Mushtaq *et al.*, 2021). Fungsi utama dari *convolutional layer* adalah mengekstraksi fitur dari *layer* sebelumnya.




**Gambar 2. 3** Contoh *Convolutional layer* (Anugerah, 2018).

Ekstraksi fitur yang dilakukan pada *convolutional layer* terlihat seperti pada Gambar 2.3. *Convolution layer* pada tahap-tahap awal akan mengesktraksi fitur tingkat rendah (*low-level feature)* dari citra, kemudian pada *convolution layer* berikutnya akan

mengekstraksi fitur pada level yang lebih tinggi seperti fitur level menengah dan tinggi. *Convolution* atau konvolusi merupakan istilah matematis yang bermakna mengaplikasikan sebuah fungsi pada hasil keluaran fungsi lain secara berulang (Marbun, 2017).

$$h(x) = f(x) * g(x) = \int_{-\infty}^{\infty} f(a).g(x-a)da \tag{2.2}$$

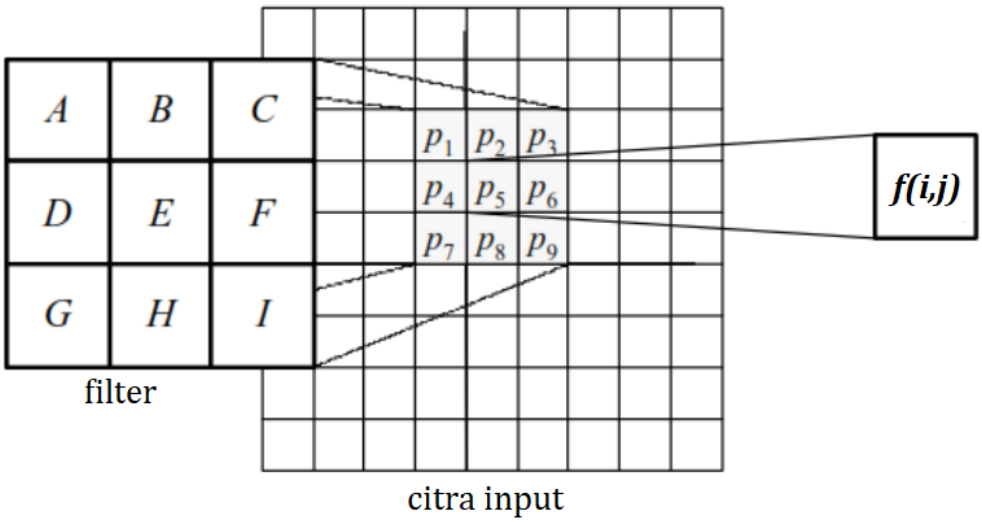
Diketahui nilai f(x) adalah input, g(x) adalah *filter*, sedangkan tanda * adalah operator konvolusi, dan variabel a adalah peubah bantu. Pada pengolahan citra, karena nilai pada koordinat-koordinat piksel merupakan bilangan diskrit, maka operasi konvolusi yang dilakukan juga operasi diskrit (Gazali *et al.*, 2012).

$$h(x) = f(x) * g(x) = \sum_{a=-\infty}^{\infty} f(a).g(x-a) \tag{2.3}$$

Persamaan 2.3 memperlihatkan bahwa operasi $\sum$ (sigma) menggantikan operasi $\int$ (integral) pada persamaan 2.2, hal ini menggambarkan perubahan operasi dari bentuk kontinu menjadi bentuk terbatas atau diskrit. Namun, persamaan yang disebutkan sebelumnya baru menggambarkan operasi konvolusi untuk satu dimensi yakni hanya pada sumbu-x, sehingga pada persamaan 2.4 dituliskan operasi konvolusi bentuk diskrit pada pengolahan citra dua dimensi (Gazali *et al.*, 2012).

$$h(x,y) = f(x,y) * g(x,y) = \sum_{a=-\infty}^{\infty} \sum_{a=-\infty}^{\infty} f(a,b).g(x-a,y-b) \tag{2.4}$$

Persamaan 2.4 menunjukkan operasi konvolusi diskrit untuk data dua dimensi dengan menambahkan koordinat y dan variabel b sebagai peubah untuk koordinat sumbu-y. Karena data yang diolah merupakan data diskrit, maka operasi konvolusi biasanya dalam bentuk matriks. Matriks input f(x) biasanya berukuran lebih besar daripada matriks *filter* g(x).

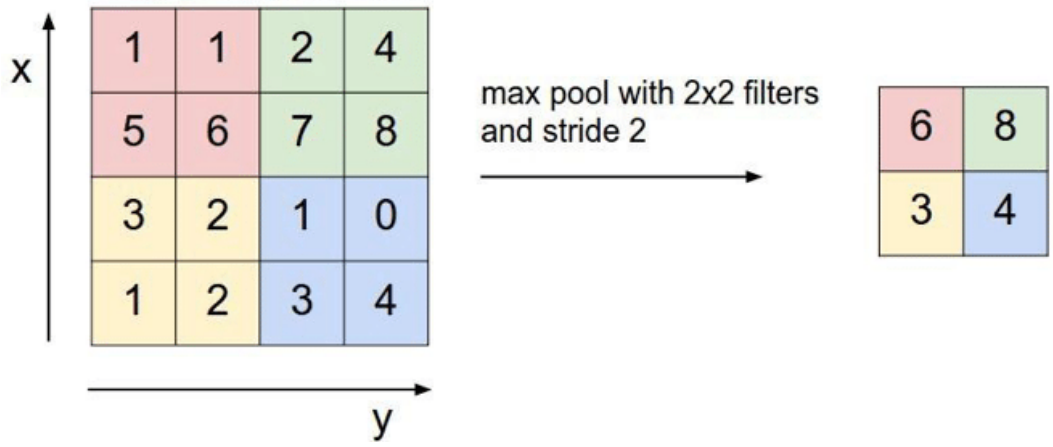



**Gambar 2. 4** Ilustrasi operasi konvolusi (Gazali *et al.*, 2012)

Gambar 2.4 menampilkan ilustrasi dari operasi konvolusi pada matriks dua dimensi, operasi konvolusi dilakukan dengan menggeser *filter* pada tiap-tiap piksel citra input. Konvolusi dilakukan dari kiri kemudian digeser ke kanan dan dari atas lalu diturunkan ke bawah. Hasil dari sebuah proses konvolusi disebut *feature map*. *Feature map* tersebut akan menjadi input untuk *layer* berikutnya.

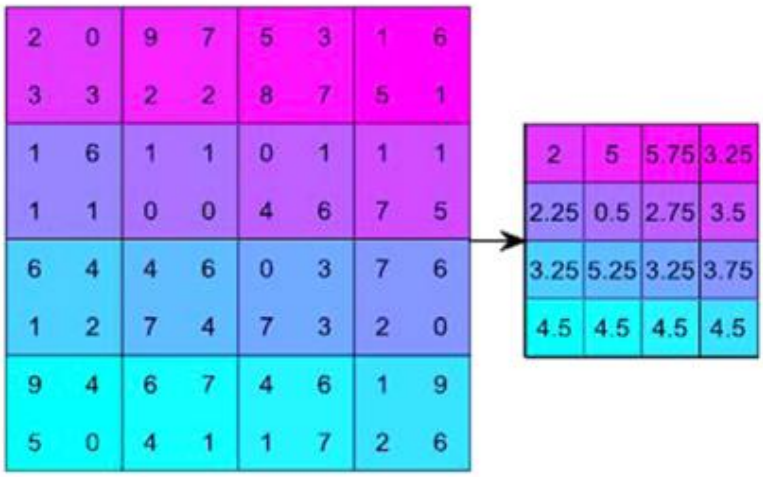
### 2.2.2 *Pooling Layer*

*Pooling layer* adalah proses dilakukan setelah tahap *convolution layer*. Proses ini bertujuan untuk mengurangi ukuran dimensi dari *feature map* (Anugerah, 2018). Selain itu, *pooling layer* dapat mengurangi jumlah parameter yang di-*update* sehingga mampu mempercepat proses komputasi, dan mengurangi *overfitting*. Terdapat dua metode pada *pooling* yang digunakan yaitu *max pooling* dan *average pooling* (Fattah, 2021). *Max pooling* membagi matriks output dari *convolutional layer* menjadi beberapa *grid* kecil dan kemudian mengambil nilai maksimal dari setiap *grid* kecil itu untuk disusun kembali menjadi matriks baru yang berisi kumpulan nilai maksimum. Sedangkan *average pooling* adalah merata-ratakan dari setiap *grid-grid* kecil agar satu nilai hasil rataan tiap *grid* disusun menjadi sebuah matriks yang ukurannya lebih kecil daripada ukuran matriks inputnya. Berikut adalah operasi dari *max pooling* (Marbun, 2017).




**Gambar 2. 5** Contoh *max pooling* (Marbun, 2017).

Gambar 2.5 menunjukkan contoh dari *max pooling*. Matriks *input* berukuran 4x4 kemudian dibagi menjadi beberapa *grid* kecil sesuai dengan ukuran *pooling layer* yang digunakan. Pada gambar diatas, matriks input diaplikasikan *pooling layer* berukuran 2×2 dan 2 stride sehingga matriks input terbagi menjadi *grid* kecil yang berwarna merah, hijau, kuning dan biru. Masing-masing *grid* kecil tersebut akan diambil nilai terbesar untuk disusun pada matriks output yang baru (Marbun, 2017). *Average pooling* adalah merata-ratakan dari setiap *grid-grid* kecil agar satu nilai hasil rataan tiap *grid* disusun menjadi sebuah matriks yang ukurannya lebih kecil daripada ukuran matriks inputnya.

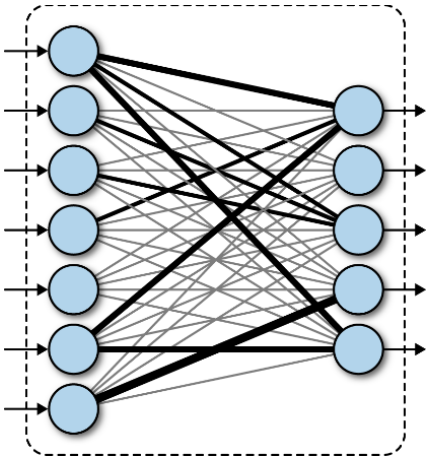



**Gambar 2. 6** Contoh *average pooling* (Li *et al.*, 2019).

Gambar 2.6 menunjukkan contoh dari *average pooling*. Matriks input berukuran 8x8 kemudian dibagi menjadi beberapa *grid* kecil yang masing-masing berisi 4 elemen berdekatan. Pada gambar diatas, warna-warna yang beragam menginformasikan *grid* kecil yang akan dilakukan pooling. Matriks input yang ada diaplikasikan *pooling layer* berukuran 2×2 dan 2 *stride*. Sesuai dengan namanya *average*, operasi yang dilakukan adalah merata-ratakan nilai elemen-elemen yang ada pada tiap *grid*, kemudian nilai yang diperoleh menjadi elemen baru pada matriks output yang dihasilkan. Matriks mula-mula berukuran 8x8 sekarang menjadi ukuran 4x4.

### 2.2.3 *Fully Connected Layer*

*Fully connected layer* adalah lapisan yang menghubungkan seluruh neuron yang sudah dihasilkan pada tahap sebelumnya dengan *layer* setelahnya (Anugerah, 2018). Namun, sebelum itu dilakukan maka *feature map* yang dihasilkan harus melakukan *flatten*. Hal tersebut dilakukan dengan tujuan untuk mentransformasi data dari bentuk multidimensional menjadi sebuah data satu dimensi agar dapat diklasifikasi secara linear (Suartika *et al.*, 2016). *Fully connected layer* umumnya terletak di akhir sebuah arsitektur CNN (Anugerah, 2018).



**Gambar 2. 7** Contoh *Fully Connected Layer* (Kost *et al.*, 2019).

Pada Gambar 2.7 menunjukkan contoh *fully connected layer*. Garis-garis yang menghubungkan neuron-neuron yang ada di sebelah kiri ke neuron yang ada di sebelah kanan merupakan gambaran mengenai *fully connected layer* yang tidak melewatkan atau menghilangkan satupun informasi yang ada sebelumnya. Garis yang memiliki warna

lebih hitam menandakan node atau neuron tersebut lebih aktif daripada neuron lain yang lebih pudar warnanya.

### *2.3 Hyperparameter*

*Hyperparameter* dalam sebuah program dianggap sebagai suatu variabel yang mengatur konfigurasi arsitektur untuk menjadi optimal. Variabel tersebut bebas dan tidak terikat dengan arsitektur itu sendiri (Aszemi and Dominic, 2019). Semua parameter yang ada pada sebuah arsitektur yang diatur sebelum dilakukannya proses *training* adalah *hyperparameter* (Ningsih, 2022). *Hyperparameter* dibagi menjadi dua jenis yaitu hyperparameter yang menentukan struktur jaringan dan *hyperperameter* yang menentukan jaringan pelatihan (Aszemi and Dominic, 2019).

Berdasarkan struktur jaringan yag ditentukan *hyperparameter* terdiri dari ukuran kernel, jenis kernel, *stride, padding, hidden layer*, dan *activation function*. Sedangkan *hyperparameter* yang diatur untuk pelatihan jaringan diantaranya adalah *batch size*, *epoch*, *learning rate*, dan *optimizer*. *Hyperparameter* diatur untuk mengoptimalkan performa arsitektur agar menjadi maksimal.

#### *2.3.1 Activation Function*

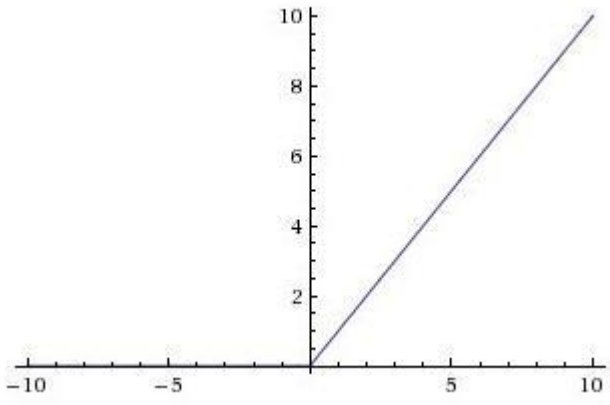
*Activation function* atau fungsi aktivasi adalah sebuah fungsi non-linear yang menjelaskan hubungan pada tiap-tiap tingkat aktivasi atau *layer* di dalam arsitektur (Suartika *et al.*, 2016). Fungsi aktivasi digunakan untuk menentukan aktif atau tidaknya suatu neuron (Mubarok, 2019). Beberapa fungsi aktivasi yang biasa digunakan pada arsitektur CNN adalah *Sigmoid, Softmax, Logsoftmax* dan lainnya.

a. Fungsi *Rectified Linear Unit* (ReLU)

*Rectified Linear Unit* (ReLU) merupakan fungsi aktivasi linear yang popular digunakan untuk aplikasi *deep learning* (Anugerah, 2018). Fungsi aktivasi ReLU sederhananya adalah membuat batasan pada nilai output yang kurang dari 0, secara matematis persamaan fungsi ReLU dituliskan sebagai berikut:

$$f(x) = \max(0, x) = \begin{cases} x_i \; jika \; x_i \geq 0 \\ 0 \; jika \; x_i < 0 \end{cases} \tag{2.5}$$

Persamaan 2.5 menunjukkan nilai *output* minimal yang dihasilkan fungsi ReLU adalah 0 dan maksimalnya adalah x. Fungsi ReLU mengubah *input* yang bernilai negatif menjadi 0 dan input yang bernilai positif maka output tersebut akan tetap.




**Gambar 2. 8** Kurva Fungsi ReLU (Anugerah, 2018).

Gambar 2.8 menunjukkan bahwa tidak ada output yang dihasilkan dari fungsi ReLU yang bernilai negatif, semua hasil bernilai ≥ 0. Fungsi aktivasi ReLU mampu mengatasi masalah *gradien exploding* yang muncul pada aktivasi lain seperti *sigmoid* dan *softmax.* Selain itu fungsi ReLU sangat mungkin untuk melakukan proses *training* lebih cepat dibanding dengan fungsi *sigmoid* (Anugerah, 2018).

b. Fungsi *Softmax*

Fungsi *Softmax* adalah merupakan fungsi yang digunakan pada *fully connected layer* untuk melakukan klasifikasi *multiclass*. Hal tersebut karena fungsi *Softmax* mampu mengubah nilai output menjadi probabilitas distribusi (Anugerah, 2018). Berikut adalah persamaan fungsi *softmax* (Mahmud *et al.*, 2019).

$$f(x) = \frac{e^{xi}}{\sum_{j=1}^{n} e^{xj}} \tag{2.6}$$

Persamaan 2.6 membandingkan eksponensial nilai input yang diperoleh dengan jumlah eksponensial keseluruhan nilai pada input. Fungsi aktivasi *softmax* menghasilkan rentang output pada angka 0 sampai 1. Jumlah probabilitas seluruhnya dalam satu objek bernilai 1 (Anugerah, 2018). Keunggulan dari *softmax* adalah dapat digunakan untuk *multiple classification logistic regression* model.

*2.3.2. Batch size*

*Batch size* adalah istilah yang digunakan untuk menggambarkan banyaknya data dalam satu bagian yang akan disebar pada proses *training* dalam arsitektur. *Batch size*

mampu memudahkan komputasi program yang berjalan karena dapat melakukan *update weight* pada keseluruhan data seukuran *batch size* saat iterasi sebelumnya. Iterasi adalah banyaknya jumlah pengambilan data untuk dilakukan proses *training*. Iterasi dapat dihitung dengan membagi seluruh jumlah data input dengan besarnya *batch size*.

*2.3.3 Epoch*

*Epoch* adalah banyaknya pengulangan proses memasukkan keseluruhan data pada sebuah model yang dibuat. *Epoch* juga dapat diartikan sebagai satuan lamanya sebuah model untuk belajar. Jadi proses pembelajaran dapat dilakukan secara berulang-ulang tergantung dari banyak sedikitnya *epoch* yang diatur. *Epoch* berbanding lurus dengan waktu, semakin banyak *epoch* yang digunakan maka semakin lama waktu yang dibutuhkan untuk belajar. Sebaliknya, jika *epoch* yang digunakan sedikit, maka waktu pembelajaran menjadi lebih singkat.

*2.3.4 Learning rate*

*Learning rate* adalah besarnya langkah yang diambil pada sebuah program untuk melakukan *update* pada *weight*. *Learning rate* juga diartikan sebagai kecepatan dalam kegiatan pembaharuan bobot pada suatu pelatihan (Rismiyati and Luthfiarta, 2021). *Learning rate* yang terlalu besar dapat mengakibatkan langkah untuk menuruni grafik *loss* terlalu besar sehingga memungkinkan tidak tercapainya global minima karena terlewat. Namun, jika *learning rate* terlalu kecil maka kemungkinan tercapainya global minima ada namun butuh waktu yang sangat lama untuk dapat mencapai titik tersebut.

*2.3.5 Loss Function*

Grafik *loss* dapat terbentuk karena adanya langkah-langkah yang dibuat oleh *learning rate* untuk menuruni grafik tersebut. *Loss function* menggambarkan tingkat ketidakpastian informasi yang diperoleh selama proses *training*. *Loss function* erat kaitannya dengan probabilitas yang dimiliki suatu *weight* yang dihasilkan. *Cross Entropy Loss* adalah *loss function* yang digunakan untuk mengidentifikasi dataset yang digunakan untuk klasifikasi. *Loss* yang lain digunkan untuk regresi adalah L2 atau L1 *loss*.

*2.3.6 Optimizer*

*Optimizer* adalah algoritma yang digunakan oleh model untuk memperbarui bobot setiap lapisan setelah setiap iterasi untuk meminimalkan nilai *loss* (Khedkar *et al.*, 2020). Jenis *optimizer* beragam seperti SGD, Adam, Adelta, RMSProp, AdaGrad, AdamW,

Nadam dan sebagainya. Namun yang akan digunakan adalah 4 jenis jaitu SGD, AdaGrad, RMSProp, dan Adam.

a. *Stochastic Gradient Descent* (SGD)

SGD adalah salah satu algoritma yang digunakan pada *deep learning*. SGD pertama kali diperkenalkan oleh Herbert Robbins dan Sutton Manro. Stokastik pada algoritma ini diartikan sebagai random arau acak. Jadi, pada pengambilan data dilakukan secara acak untuk kemudian dilakukan *training* pada model dan proses seterusnya (Vani and Rao, 2019). Berikut ini adalah persamaan yang dimiliki oleh SGD dalam melakukan *update* pada *weight* :

$$W_{i,t} = W_{i,t-1} - \eta \frac{\partial L}{\partial W_{i,t}} \tag{2.7}$$

Persamaan 2.7 menampilkan nilai $W_{i,t}$ adalah *weight* yang baru, $W_{i,t-1}$ adalah *weight* sebelumnya, $\eta$ adalah *learning rate*, dan L adalah *Loss*. Nilai *weight* baru diperoleh dari nilai *weight* sebelumnya dikurangi dengan besar *learning rate* dikali dengan gradien *loss* terhadap *weight*.

b. *Adaptive Gradient* (AdaGrad)

AdaGrad adalah singkatan dari *Adaptive Gradient* merupakan kolaborasi dari SGD dan momentum. AdaGrad memiliki strategi untuk menyesuaikan *learning rate* agar setiap *weight* yang ada memiliki besar *learning rate* yang beragam tergantung karakterisktik dari *weight* tersebut. Persamaan untuk melakukan *update weight* pada algoritma AdaGrad mirip dengan SGD, namun *learning rate* pada AdaGrad telah disesuaikan dengan *weight* itu sendiri sehingga secara matematis dituliskan sebagai berikut (Vani and Rao, 2019):

$$W_{i,t} = W_{i,t-1} - \frac{\eta}{\sqrt{G_i + \varepsilon}} \frac{\partial L}{\partial W_{i,t}} \tag{2.8}$$

$$G_{i,t} = \sum_t \left( \frac{\partial L}{\partial W_{i,t}} \right)^2 \tag{2.9}$$

Persamaan 2.8 menampilkan besar *learning rate* $\eta$ pada persamaan 2.9 telah disesuaikan dengan *weight* sehingga persamaan *learning rate* saat ini menjadi $\frac{\eta}{\sqrt{G_{i,t}+\varepsilon}}$. Hal ini akan mengakibatkan *weight* yang mendapatkan *update* lebih sering akan memiliki *learning rate* kecil, sedangkan *weight* yang jarang mendapatkan *update* akan memiliki

*learning rate* yang besar. Diketahui $G_{i,t}$ adalah jumlah kuadrat gradien *loss* terhadap *weight* ke-i. dan $\varepsilon$ adalah konstanta yang bernilai $10^{-8}$.

c. *Root Mean Square Propagation* (RMSProp)

RMSProp merupakan singkatan dari *Root Mean Square Propagation.* RMSProp akan mengatasi permasalahan yang dimiliki oleh AdaGrad yaitu jika $G_{i,t}$ terlalu besar maka *learning rate* akan semakin kecil dan mendekati nol. Hal tersebut tidak boleh terjadi karena dapat mengakibatkan *weight* berhenti untuk *update* (Vani and Rao, 2019). RMSProp memanfaatkan teknik yang mirip dengan AdaGrad dengan melakukan *weighted average* pada $G_{i,t}$. Pengoptimal ini dapat dianggap sebagai algoritma yang sangat efektif dan praktis untuk *deep learning*. Berikut ini adalah persamaan untuk *update weight* pada RMSProp (Poojary and Pai, 2019):

$$W_{i,t} = W_{i,t-1} - \frac{\eta}{\sqrt{E[G_i]_t + \varepsilon}} \frac{\partial L}{\partial W_{i,t}} \tag{2.10}$$

$$E[G_i]_t = \gamma E[G_i]_{t-1} + (1-\gamma)\left(\frac{\partial L}{\partial W_{i,t}}\right)^2 \tag{2.11}$$

Persamaan 2.11 memiliki nilai $\gamma$ yang merupakan *hyperparameter*, $G_{i,t-1}$ adalah jumlah kuadrat gradien *loss* terhadap *weight* sebelumnya dan adanya *weighted average* (E) yang menjadi solusi untuk mengatasi *learning rate* yang akan berhenti.

d. *Adaptive Moment Estimation* (Adam)

*Adaptive Moment Estimation* atau dikenal dengan pengoptimasi Adam diperkenalkan oleh Jimmy Lei Ba dan Diederik P. Kingma. Algoritma Adam menggabungkan keunggulan dari algoritma AdaGrad dan RMSProp. Adam menggabungkan momentum secara langsung sebagai perkiraan momen orde pertama dari gradien. Kemudian memasukkan koreksi bias pada estimasi momen orde pertama dan momen orde kedua untuk melakukan *update* pada *weight*. Adam memiliki komputasi yang efisien, selain itu kebutuhan untuk memorinya yang sangat kecil. Momen orde pertama dan orde kedua dari gradien dapat dihitung sebagai berikut (Poojary and Pai, 2019):

$$m_{i,t} = \beta_1 m_{t-1} + (1-\beta_1)\left(\frac{\partial L}{\partial W_{i,t}}\right)^2 \quad (2.12)$$

$$v_{i,t} = \beta_2 v_{t-1} + (1-\beta_2)\left(\frac{\partial L}{\partial W_{i,t}}\right)^2 \quad (2.13)$$

Persamaan 2.13 adalah momen orde pertama dan persamaan 2.14 adalah momen orde kedua. Nilai $\beta_1$ dan $\beta_2$ adalah *hyperparameter*. Adam memiliki *bias correction* dari kedua orde di atas, secara matematis dituliskan sebagai berikut:

$$\widehat{m_{i,t}} = \frac{m_{i,t}}{1-\beta_1^t} \quad (2.15)$$

$$\widehat{v_{i,t}} = \frac{v_{i,t}}{1-\beta_2^t} \quad (2.16)$$

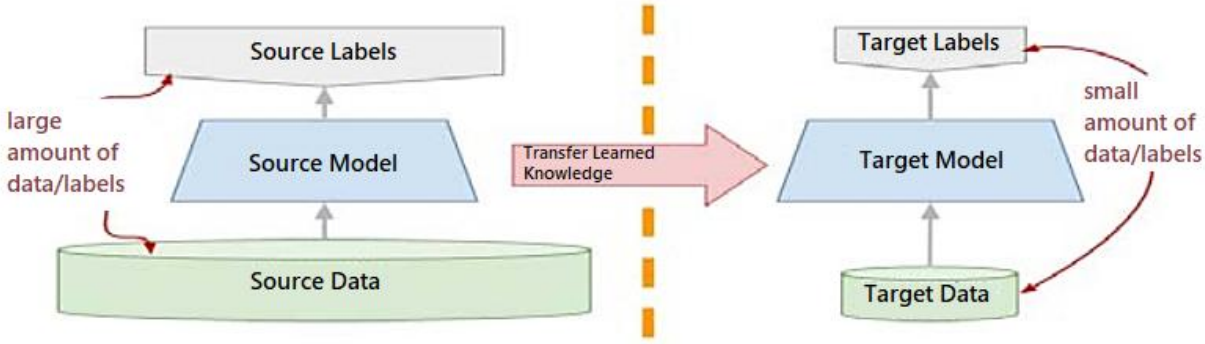
*Bias correction* persamaan 2.15 dan 2.16 kemudian disubtitusikan ke dalam persamaan *update weight* algoritma Adam yakni sebagai berikut:

$$W_{i,t} = W_{i,t-1} - \frac{\eta}{\sqrt{\widehat{v_{i,t}} + \varepsilon}}\,\widehat{m_{i,t}} \quad (2.17)$$

Persamaan diatas merupakan rumus untuk melakukan *update* pada *weight* pada algoritma Adam. Diketahui $\widehat{m_{i,t}}$ merupakan *bias correction* orde pertama yang mirip dengan momentum, dan $\widehat{v_{i,t}}$ adalah *bias correction* orde kedua yang mirip dengan RMSProp.

### *2.4 Transfer learning*

Ide untuk menggunakan *transfer learning* dalam penyelesaian masalah pada *machine learning* dimulai dari tahun 1995 pada NISP *workshop* yang membahas tentang metode untuk melatih ulang dan menggunakan kembali pengetahuan dari pembelajaran sebelumnya. Kini, metode tersebut sudah dimanfaatkan secara luas untuk mengklasifikasi citra (Al-Sharhan *et al.*, 2019). *Transfer learning* ialah metode yang memanfaatkan jaringan yang dilatih menggunakan dataset berukuran sangat besar sebelumnya, kemudian digunakan kembali untuk mempelajari tugas lain. *Convolutional layer* dan *pooling layer* pada *transfer learning* jumlahnya lebih banyak daripada arsitektur CNN yang sederhana.



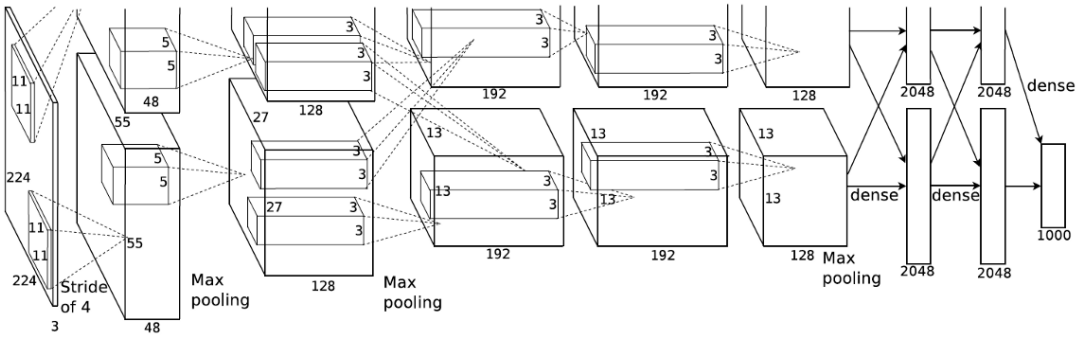

**Gambar 2.9** Cara kerja *transfer learning* ( Al-Sharhan *et al.*, 2019).

Gambar 2.9 menampilkan cara kerja dari *transfer learning* dalam mentransfer pengetahuan yang telah dipelajari dengan jumlah data yang besar kemudian diaplikasikan ke data yang berukuran lebih kecil. Kesamaan antara data sumber dan data target menentukan tingkat pengetahuan yang dapat ditransfer. Jika data sumber dan data target sangat mirip, maka pengetahuan yang ditransfer dapat lebih banyak yang berasal dari sejumlah besar lapisan sumber. Jika berbeda, maka lebih sedikit pengetahuan dari lapisan-lapisan tersebut yang dapat ditransfer (Al-Sharhan *et al.*, 2019).

Pada *transfer learning* yaitu: Pertama, *fine tuning* dengan *weight* yang berasal dari model CNN yang terlatih dan *freeze* atau pembekuan pada beberapa *layer*. Kedua, ekstraksi fitur yang merupakan ide umum untuk mengakses fitur pada tiap *layer* dan melatih dataset agar klasifikasi hasil sesuai dengan keinginan. ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*) merupakan sebuah kompetisi yang diadakan oleh ImageNet untuk para peneliti untuk dapat membandingkan kemajuan dalam pendeteksian objek dan klasifikasi gambar pada dataset yang raksasa.

### 2.4.1 AlexNet

AlexNet adalah arsitektur yang didesain oleh Alex Krizhevsky *et al.* dan memenangkan kompetisi yang diadakan oleh ImageNet tahun 2012. Jaringan ini mampu mencapai top-5 *error* di angka 15,3% atau 10,8 poin lebih kecil daripada hasil yang dicapai oleh juara kedua (Wahlang *et al.*, 2022). Jaringan saraf ini, memiliki 60 juta parameter dan 650.000 neuron, jaringan ini terdiri dari lima *convolutional layer*, beberapa di antaranya diikuti oleh *max pooling layer*, dan tiga *fully connected layer* yang terhubung dengan 1000 kelas pada *activation function softmax* (Krizhevsky *et al.*, 2012)

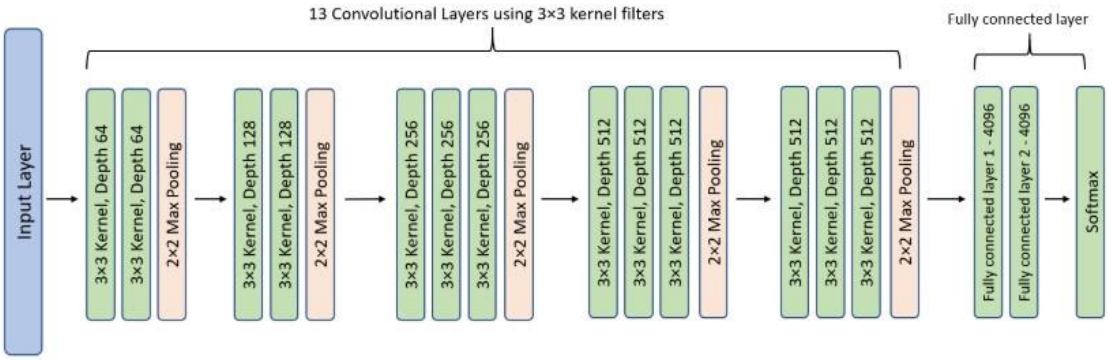



**Gambar 2. 10** Arsitektur AlexNet (Krizhevsky *et al.*, 2012)

Gambar 2.10 menunjukkan arsitektur AlexNet yang menjadi titik awal pengembangan tentang klasifikasi citra menjadi menarik. Langkah dari arsitektur AlexNet adalah *convolution layer* pertama memfilter 224×224×3 piksel citra *input* dengan *filter* ukuran 11×11×3 dan *stride* 4 piksel dengan *output* sebanyak 96 kernel. *Output* dari *layer* ini kemudian dinormalisasi dan dilakukan *max pooling*. *Output* dari lapisan pertama adalah 256 kernel dengan ukuran *filter* 5×5×48. *Layer* ketiga, keempat dan kelima terhubung tanpa normalisasi atau *pooling layer*. *Convolution layer* ketiga memiliki ukuran 384 kernel dengan *filter* 3×3×256 terhubung ke *output* dari *layer* kedua. *Convolution layer* keempat memiliki 384 kernel dan *filter* ukuran 3×3×192, dan yang *layer* kelima memiliki 256 kernel dengan *filter* ukuran 3×3×192. *Fully connected layer* masing-masing memiliki total 4096 neuron (Krizhevsky *et al.*, 2012).

2.4.2 VGG16

VGG16 dikembangkan oleh Simonyan dan Zisserman pada tahun 2014. VGG16 mampu bersaing pada lokalisasi dan klasifikasi suatu objek sehingga berhasil menempati posisi *runner-up* pada kompetisi ImageNet setelah GoogLeNet dengan top-5 *error rate* sebesar 7,3%. Arsitektur VGG terdiri atas *layer* konvolusi, ReLU, *max pooling layer* dan *fully connected layer* dengan 1.000 kelas, *activation function* yang digunakan adalah *softmax* (Sandhopi *et al.*, 2020).

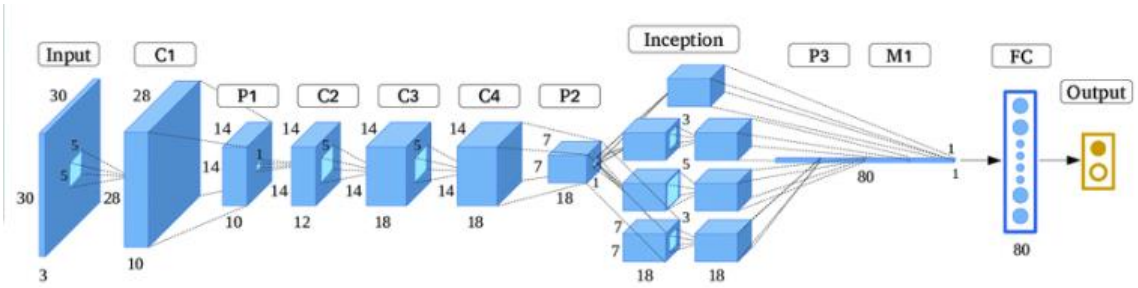



**Gambar 2. 11** Arsitektur VGG16 (Rismiyati and Luthfiarta, 2021).

Gambar 2.11 menunjukkan susunan layer pada arsitektur VGG16. Arsitektur ini mempunyai 16 *layer* dengan pembagian 13 *convolutional layer* dan 3 *fully connected layer* dengan variasi ReLU dan *max pooling* pada beberapa *layer*. *Layer input* memiliki ukuran 224×224 piksel dan 3 *channel* RGB (Suberi *et al.*, 2019). Perbedaan pada 13 *convolutional layer* adalah ukuran *filter* yang berbeda-beda. Rinciannya adalah: 2 *convolutional layer* memiliki 64 kernel. *Layer* ke-3 dan 4 mempunyai 128 kernel. *Layer* berikutnya memiliki 256 kernel pada *layer* 5, 6, dan 7. *Layer* ke-7 sampai 13 memiliki 512 kernel. *Max-pooling* dengan *filter* 2×2 dilakukan setelah *convolution layer* ke-2, 4, 7, 10 dan 13. Hasil *max pooling* terakhir dihubungkan ke *fully connected layer* dan fungsi *softmax* untuk mengklasifikasi kelas citra (Rismiyati and Luthfiarta, 2021).

### 2.4.3 GoogLeNet

GoogleNet yang juga dikenal dengan Inception ialah arsitektur CNN yang diperkenalkan pada tahun 2014 oleh Google. Pada ajang ILSVRC tahun 2014 GoogleNet mampu menjuarai lomba tersebut dengan *top-5 error rate* di angka 6,67%. GoogLeNet terinspirasi dari LeNet namun implementasi dari idenya GoogLeNet memiliki elemen baru yang disebut dengan *inception module* (Fattah, 2021). Arsitektur googleNet memiliki jumlah lapisan yang berjumlah 144 *layer.*




**Gambar 2.12** Arsitektur GoogLeNet (Fattah, 2021)

Gambar 2.12 menunjukkan arsitektur GoogleNet menggunakan konvolusi pendek atau yang dikenal sebagai *inception module* yang memiliki beberapa keunggulan yaitu dapat mengurangi dimensi dan menghilangkan kemacetan komputasi. Jaringan ini menguraikan *filter* konvolusi bidang resiptif yang luas menjadi kelompok konvolusi kecil paralel dengan ukuran 1×1, 3×3, dan 5×5 serta lapisan *pooling*. *Output* dari blok paralel ini kemudian digabungkan (Tobias *et al.*, 2016).

## *2.5 Confusion matrix*

*Confusion matrix* merupakan salah satu metode yang menghitung banyaknya piksel yang berhasil diklasifikasikan ke dalam suatu kelas. *Confusion matrix* digunakan pada suatu model klasifikasi agar dapat diukur performanya yang melingkupi nilai *precision, recall, accuracy, dan F1-score* (Suberi *et al.*, 2019).

Tabel 2.1 Contoh *confusion matrix* (Sumber: Vani and Rao, 2019).

| *Actual Class* | *Predicted Class* | |
|---|---|---|
| | *Positive* | *Negative* |
| *Positive* | *True Positive (TP)* | *False Negative (FN)* |
| *Negative* | *False Positive (FP)* | *True Negative (TN)* |

*Confusion matrix* pada Tabel 2.1 menunjukkan indikator tingkat keakuratan yang dikenal memiliki beberapa elemen yaitu: TP (*true positive*) artinya data positif yang bernilai benar, TN (*true negative*) artinya data negatif yang bernilai benar, FP (*false positive*) artinya data positif yang bernilai salah, dan FN (*false negative*) artinya data negatif yang bernilai salah .

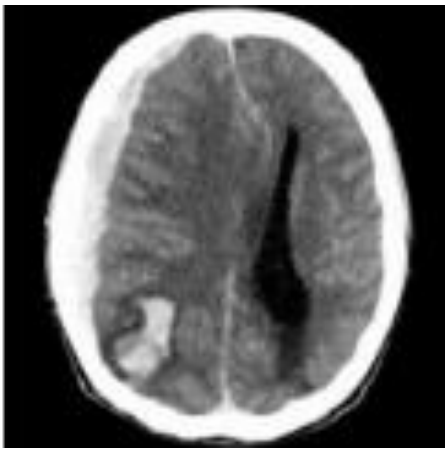
*Accuracy* atau akurasi dihitung untuk menemukan rasio jumlah prediksi yang tepat (label positif diprediksi positif dan label negatif diprediksi negatif) dengan jumlah total prediksi. (Sajja *et al.*, 2019).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.18}$$

Persamaan 2.18 menampilkan perhitungan akurasi dengan membandingkan nilai benar untuk kelas positif (TP) dan negatif (TN) kemudian dibandingkan dengan serluruh data yang ada pada semua kelas (TP, TN, FP, FN).

## 2.6 Stroke

Stroke merupakan kondisi pendarahan otak yang terjadi pada saat pasokan darah ke otak terputus akibat adanya penyumbatan akibat pembekuan darah atau diakibatkan oleh pecahnya pembuluh darah yang ada di otak, sehingga terjadi kematian pada sebagian area di otak (Mushtaq *et al.*, 2021).



**Gambar 2. 13** Citra stroke pada CT-*scan* (Sakinah *et al.*, 2020)

Gambar 2.13 memperlihatkan kondisi kepala yang memiliki pendarahan pada otak, tampak di bagian kiri bawah kepala terdapat bercak berwarna abu-abu yang lebih terang daripada daerah kepala yang lainnya. Bercak tersebut adalah darah yang berasal dari pecahnya pembuluh darah yang ada di otak. Penyebab dari penyakit stroke begitu beragam diantaranya adalah penyumbatan akibat timbunan lemak yang terkandung di dalam pembuluh darah otak. Lemak tersebut dapat berupa kolesterol seperti atheroma dan emboli, selain itu stroke juga dapat disebabkan penggunaan obat-obatan, infeksi dan hipotensi, penyumbatan pada pembuluh darah besar (arteri karotis) atau pembuluh darah

sedang (arteri serebri) dapat juga pada pembuluh darah kecil. Penyebab lainnya yang dapat menyebabkan munculnya penyakit stroke adalah pembekuan darah secara tiba-tiba pada arteri yang menyuplai darah ke otak, dapat juga akibat trauma, tekanan darah tinggi, *aneurisma*, kelainan pada pembuluh darah, *angiopati amiloid*, gangguan perdarahan dan tumor otak (Mushtaq *et al.*, 2021).

Berbagai aktivitas yang menjadi kebiasaan buruk masyarakat menjadi salah satu faktor yang mengakibatkan meningkatnya jumlah penderita stroke diantaranya adalah kecenderungan untuk mengonsumsi makanan cepat saji, kurangnya berolahraga, merokok, stres, kerja berlebih, dan penggunaan obat-obatan yang tidak tepat serta faktor lainnya.