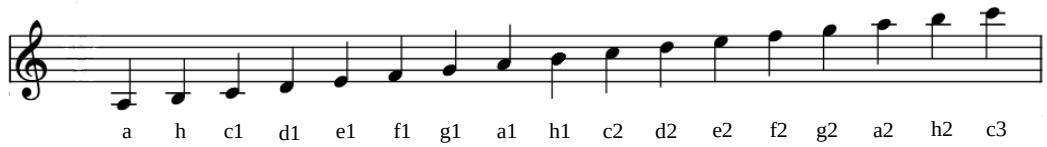
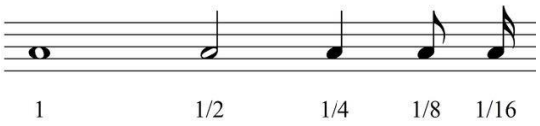
BAB 2
LANDASAN TEORI

2.1. Dasar-dasar Notasi Musik

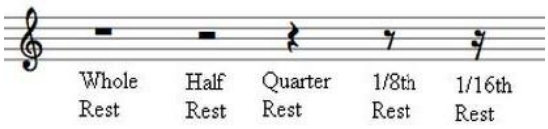
Pada notasi musik memiliki beberapa komponen penting di dalamnya antara lain yaitu: garis paranada atau *staff/stave*, kunci atau *cleff*, not merupakan objek nada pada notasi musik dan tanda istirahat [1]. Berikut dasar-dasar notasi musik yang digunakan pada penelitian ini dapat dilihat pada **Gambar 2.1** sampai **Gambar 2.3**.



Gambar 2.1. Notasi musik



Gambar 2.2. Ketukan notasi musik



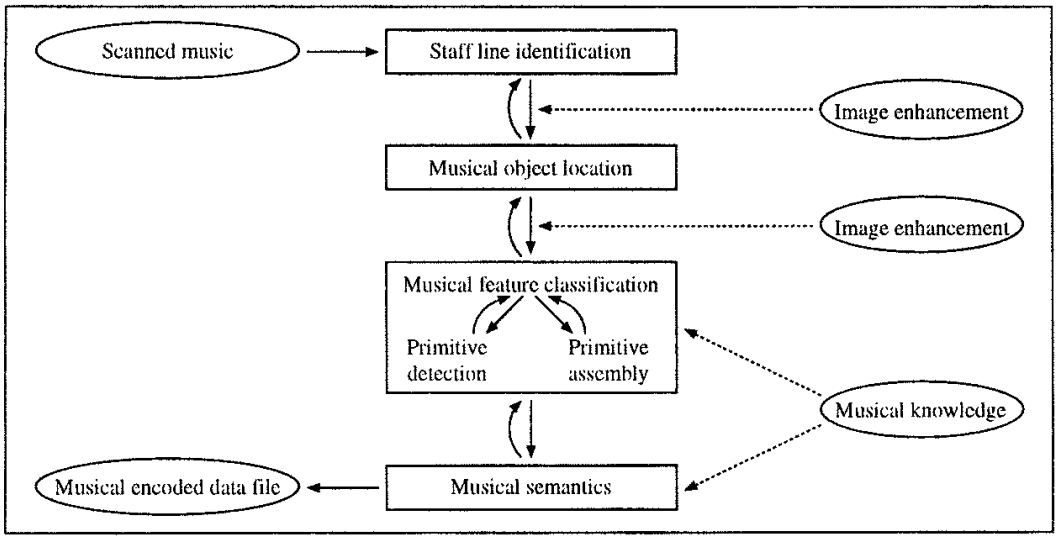
Gambar 2.3. Ketukan tanda istirahat

Pada **Gambar 2.1**, **Gambar 2.2** dan **Gambar 2.3** memiliki lima garis yang dinamakan garis paranada atau *staff/stave*, dimana berfungsi untuk mengenali jenis nada pada notasi musik dimana pada penelitian ini dibatasi dari nada A oktaf bawah sampai nada oktaf kedua. Pada **Gambar 2.1** dan **Gambar 2.3** memiliki sebuah simbol pada bagian sebelah kiri yang dinamakan kunci atau *cleff*. Pada penelitian ini kunci yang digunakan hanya kunci G atau kunci yang berada pada **Gambar 2.1** dan **Gambar 2.3**. **Gambar 2.2** merupakan jenis-jenis ketukan dari nada yang akan dimainkan, pada penelitian ini ketukan dibatasi sampai 1/16. **Gambar 2.3**

merupakan jenis-jenis ketukan tanda istirahat, pada penelitian ini yang akan digunakan hanya sampai ketukan tanda istirahat 1/16.

2.2. *Optical Music Recognition*

Pengerjaan OMR (*Optical Music Recognition*) pertama kali dikenalkan pada pertengahan 1960-an oleh Pruslin di *Massachusetts Institute of Technology*. OMR memiliki empat tugas penting: *staff line identification, musical object location, musical feature classification,* dan *musical semantics* [6].



Gambar 2.4. Kerangka umum untuk OMR

Pada **Gambar 2.4** menunjukkan proses kerja untuk OMR, tahap pertama yang dilakukan yaitu melakukan memindai halaman musik yang selanjutnya akan dilakukan proses identifikasi garis paranada yang bertujuan untuk memisahkan garis paranada dengan objek yang akan dideteksi. Proses selanjutnya yaitu mendeteksi lokasi objek yang akan dilakukan klasifikasi pada objek tersebut. Proses selanjutnya yaitu proses klasifikasi pada objek, dimana sistem telah melalui proses *training*. Proses terakhir yaitu semantik musik yaitu dengan mengubah objek yang dideteksi menjadi hasil dari prediksi klasifikasi.

2.2.1. Identifikasi Garis Paranada

Sebelum OMR dapat mengenali bentuk-bentuk yang ada pada sebuah gambar, diperlukan menetapkan jumlah notasi musik yang dikenali. *Staff line*

identification adalah fitur yang mengenali garis-garis paranada. Semua yang masuk ke dalam OMR akan mendeteksi garis-garis paranada sebagai *initial stage*.

Mendeteksi garis paranada dibutuhkan algoritma mendeteksi garis paranada untuk membuat beberapa kemungkinan tentang gambar yang dideteksi. Dua kemungkinan umum adalah garis paranada menutupi sebagian besar gambar dan garis paranada tidak termasuk *noise* akan menjadi salah satu yang berhasil tidak terdeteksi [6].

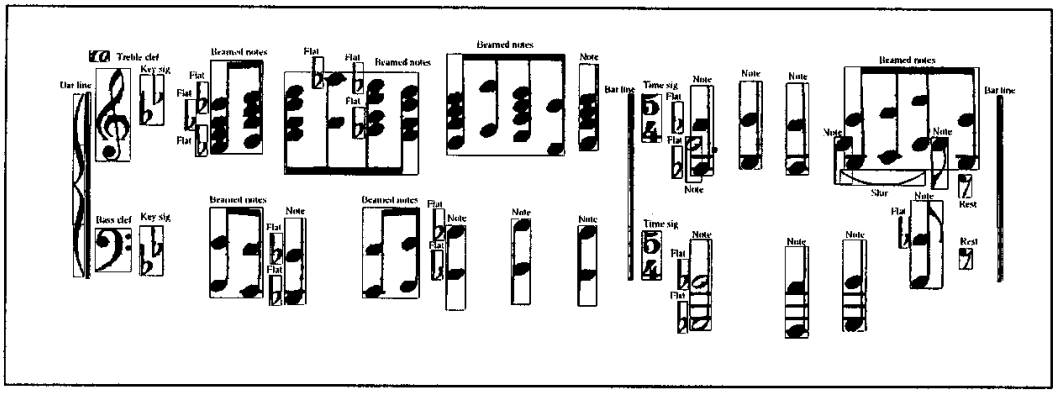
2.2.2. Lokasi Objek Musik

Lokasi objek musik dapat dilakukan dengan mengabaikan atau menghapus garis paranada. Setelah garis-garis paranada diketahui, komputer dapat mencari di antara garis-garis paranada untuk mengetahui objek-objek musik. Tugas utama menggunakan pendekatan ini adalah memutuskan kapan satu objek selesai diketahui dan dimulainya mengenal objek berikutnya [6].

Mengabaikan garis paranada memiliki hubungannya dengan klasifikasi fitur musik yaitu menyederhanakan masalah tugas utama. Pencarian objek antara garis-garis paranada dimulai dan ketika suatu objek ditemui, tahap klasifikasi fitur musik digunakan. Hasil *match* menentukan tingkat simbol yang ditemukan, dan mengarahkan pencarian untuk pola lebih lanjut. Kekurangan dari pendekatan ini adalah karena rangkaian notasi musik yang diproses menjadi lebih kompleks, demikian juga algoritma yang digunakan [6].

2.2.3. Klasifikasi Fitur Musik

Klasifikasi fitur musik adalah pengenalan terhadap objek, dimana objek telah diketahui lokasinya melalui tahapan lokasi objek musik. Proses dari klasifikasi fitur musik dapat diilustrasikan pada **Gambar 2.5**, dimana setiap kotak yang telah diketahui lokasinya akan diberi label dengan nama sesuai bentuk dari objek tersebut. Tentunya pemberian label tersebut tidak seluruhnya benar, karena bagaimana model pengenalan dan *dataset* yang dipakai.



Gambar 2.5. Proses klasifikasi fitur musik

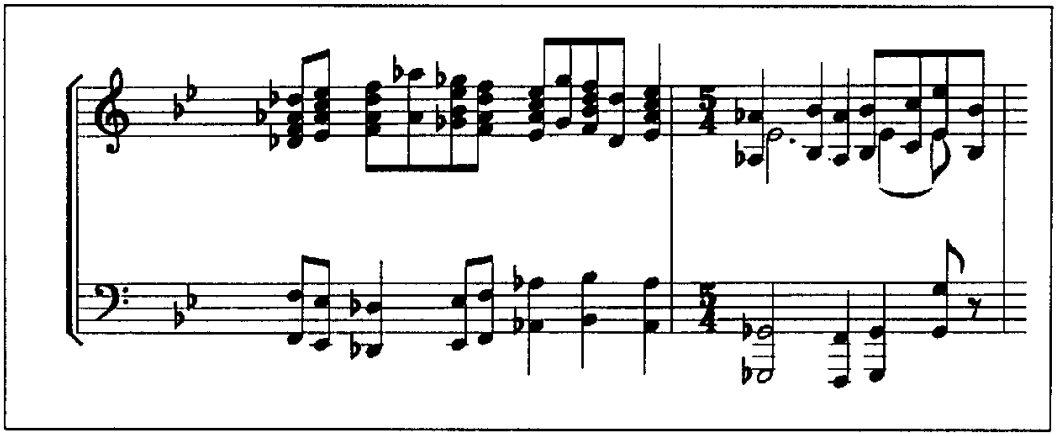
2.2.4. Semantik Musik

Tahapan akhir dari OMR adalah mengekstrak semantik musik dari bentuk yang dikenal secara grafis dan menyimpannya dalam struktur data musik [6]. Pada dasarnya, ini melibatkan pengenalan pada deteksi yang ditemukan pada sebuah gambar. Misalnya, untuk menentukan suatu objek di antara dua notasi adalah slur atau tie, nada dari dua not perlu dipertimbangkan dan titik di atas kepala not mengurangi ketukan nada sedangkan titik di sebelah kanan kepala not meningkatkan ketukan nada.

```
Staff System 1:
Staff 1:
{4Db 4F 4Ab 5Db}(0.5),{4Eb 4Ab 5C 5Eb}(0.5),
{4F 4Ab 5Db 5F}(0.5),{4Ab 5Ab}(0.5),
{4Gb 4Bb 5Eb 5Gb}(0.5),{4F 4Ab 5Db 5F}(0.5),
{4Eb 4Ab 5C 5Eb}(0.5), .....

Staff 2:
{2F 3F}(0.5),{2Eb 3Eb}(0.5),{3Db 2Db}(1.0),
{2Eb 3Eb}(0.5),{2F 3F}(0.5),
{2Ab 3Ab}(1.0),{2Bb 3Bb}(1.0),{2Ab 3Ab}(1.0) | ....
```

Gambar 2.6. Contoh semantik musik

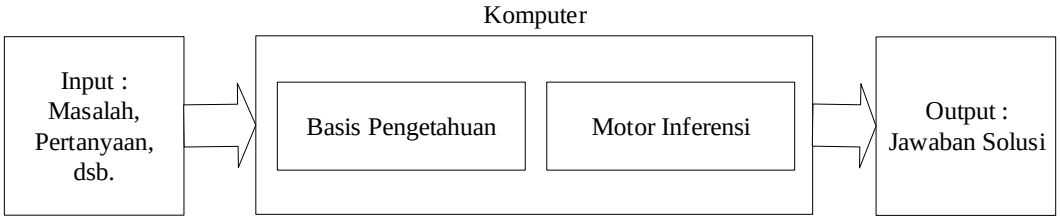


Gambar 2.7. Contoh merekonstruksi dari editor musik

Pada **Gambar 2.6** menunjukkan contoh dari semantik musik dimana sistem mengubah nilai setiap notasi nada dan tanda istirahat lengkap dengan ketukan nada pada **Gambar 2.7**.

2.3. *Artificial Intelligence* (AI)

Pada subbab sebelumnya OMR yaitu suatu sistem yang dapat melakukan pengenalan objek musik, maka dibutuhkan sistem yang memiliki daya pikir seperti manusia dan *Artificial Intelligence* (AI) merupakan bagian dari ilmu komputer yang mempelajari tentang bagaimana menjadikan mesin (komputer) dapat melakukan pekerjaan seperti dan sebaik yang dilakukan manusia. *Artificial Intelligence* (AI) menurut John McCarthy (1956) mengatakan bahwa AI bertujuan untuk mengetahui atau memodelkan proses berpikir manusia dan mendesain mesin sehingga bisa menirukan perilaku manusia [7]. Mesin dapat bertindak seperti manusia dengan dibekali pengetahuan serta kemampuan menalar yang baik. Bagian utama yang dibutuhkan dalam AI adalah basis pengetahuan (*knowlegde base*) dan motor inferensi (*inference engine*). Basis pengetahuan berisi fakta, pemikiran, serta teori-teori. Sedangkan motor inferensi merupakan kemampuan dalam penarikan kesimpulan yang didasarkan oleh pengalaman. Berikut gambaran bagian utama dalam *Artificial Intelligence* dapat dilihat pada **Gambar 2.8**.



Gambar 2.8. Bagian utama dalam *Artificial Intelligence*

Pada **Gambar 2.8** menunjukkan bahwa *input* dari AI merupakan masalah, pertanyaan atau dsb. dari *input* tersebut akan dilakukan suatu proses oleh komputer dengan bermodalkan pengetahuan komputer dapat melakukan penarikan suatu kesimpulan yang merupakan solusi dari yang data *input* lalu akan dilakukan *output* dari solusi tersebut.

Artificial Intelligence (AI) merupakan salah satu disiplin ilmu yang luas, beberapa lingkup utama AI antara lain adalah Sistem Pakar (*Expert System*), Pengolahan Bahasa Alami (*Natural Languange Processing*/NLP), Pengenalan Ucapan (*Speech Recognition*), *Computer Vision*, *Intelligent Computer-Aided Instruction*, dan lainnya. Sistem pakar adalah usaha untuk menirukan seorang pakar. Tujuan dari sistem pakar yaitu untuk mentransfer kepakaran dari seorang pakar ke komputer, kemudian ke orang lain (orang yang bukan pakar). Pengolahan Bahasa Alami yaitu dimana pengguna bisa melakukan komunikasi dengan komputer menggunakan bahasa sehari-hari. Pengenalan ucapan yaitu dimana manusia dapat melakukan komunikasi dengan komputer menggunakan suara. *Computer vision* yaitu dalam hal menginterpretasikan objek atau gambar yang tampak melalui komputer. *Intelligent Computer-Aided Instruction* yaitu bagaimana komputer dapat berperan sebagai tutor yang dapat mengajar atau melatih.

Artificial Intelligence (AI) dibuat berdasarkan sistem yang memiliki keahlian seperti manusia pada domain tertentu yaitu disebut dengan *soft computing*. *Soft computing* merupakan inovasi baru dalam membangun sistem cerdas yang mampu beradaptasi dan bekerja lebih baik jika terjadi perubahan lingkungan. *Soft computing* juga mengeksploitasi adanya toleransi terhadap ketidakpastian, ketidaktepatan, dan kebenaran parsial sehingga dapat diselesaikan dan dikendalikan dengan mudah agar sesuai dengan kenyataan. Metodologi yang sering digunakan

dalam *soft computing* salah satunya adalah Jaringan Syaraf (menggunakan pembelajaran), yaitu Jaringan Syaraf Tiruan (*Artificial Neural Network*/ANN). Metologi lain yang juga digunakan adalah Sistem *Fuzzy* (mengakomodasi ketepatan), *Probabilistic Reasoning* (Mengakomodasi Ketidakpastian), *Evolutionary Computing* (Optimasi).

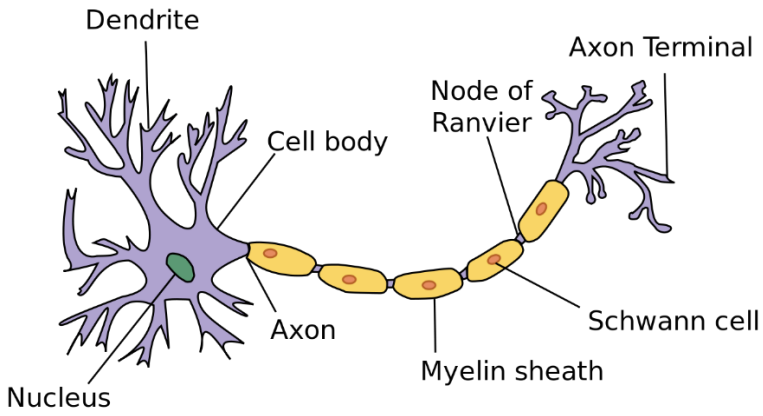
2.4. *Machine Learning*

Pada subbab sebelumnya bagian AI proses komputer dengan basis pengetahuan dan penarikan kesimpulan salah satu metode yang dapat digunakan yaitu *Machine Learning* (ML). *Machine Learning* pertama kali didefinisikan oleh Arthur Samuel (1959). Menurut Arthur Samuel, *machine learning* adalah salah satu bidang ilmu komputer yang memberikan kemampuan pembelajaran kepada komputer untuk mengetahui sesuatu tanpa pemrogram yang jelas. Menurut Mohri dkk *machine learning* juga dapat didefinisikan sebagai metode komputasi berdasarkan pengalaman untuk meningkatkan performa atau membuat prediksi yang akurat [8]. Definisi pengalaman di sini ialah informasi sebelumnya yang telah tersedia dan bisa dijadikan data pembelajar.

Dalam pembelajaran *machine learning*, terdapat beberapa konsep umpan balik untuk belajar. Seperti: *Supervised Learning*, *Unsupervised Learning*, *Semi-Supervised Learning* dan *Reinforcement Learning* [9]. *Supervised Learning* adalah pembelajaran dari pasangan *input-output* (contekan), contekan di sini yaitu masukkan data yang telah diberi label. Setelah itu membuat prediksi dari data yang telah diberi label. *Unsupervised Learning* adalah pembelajaran tanpa pasangan *input-output* atau kebalikan dari *supervised learning* yaitu masukkan data yang tidak diberi label. *Semi-Supervised Learning* adalah pembelajaran dari gabungan 2 konsep umpan balik yaitu *supervised learning* dan *unsupervised learning*. *Semi-Supervised Learning* biasanya karena banyak *noise* atau kurangnya data. *Reinforcement Learning* adalah pembelajaran dari efek suatu tindakan, yang berupa hadiah (*reward*) dan hukuman (*punishment*).

2.5. *Artificial Neural Network*

Pada ML banyak model yang dapat digunakan salah satu model yang menirukan cara kerja otak manusia yaitu *Artificial Neural Network* (ANN). *Artificial Neural Network* adalah suatu jaringan yang memodelkan sistem syaraf otak manusia untuk melakukan pengenalan pola. Pemodelan ini didasari oleh otak manusia dalam melakukan mengorganisir *neuron* sehingga mampu mengenali pola secara efektif. Dalam otak manusia terdiri dari milyaran *neuron* yang saling berhubungan. Hubungan antar neuron disebut dengan *Synapses*. Komponen *neuron* terdiri dari satu inti sel yang melakukan pemrosesan informasi, satu akson (*axon*) dan minimal satu *dendrit*. Informasi yang masuk akan diterima oleh *dendrit*. Selain itu, *dendrit* juga menyertasi akson sebagai keluaran dari suatu pemrosesan informasi [10].



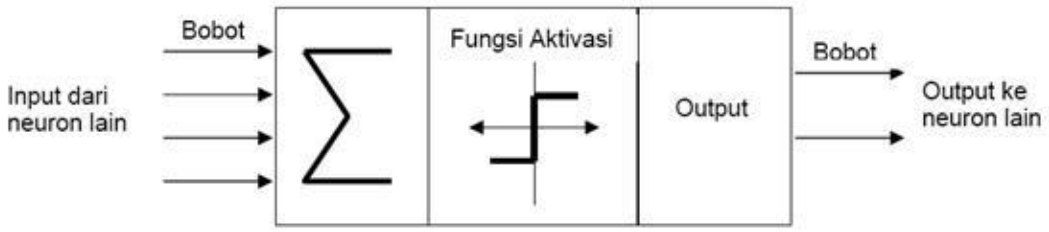
Gambar 2.9. Jaringan syaraf manusia

Cara kerja dari **Gambar 2.9** adalah sebuah neuron menerima impuls dari neuron lain melalui *dendrit* dan mengirimkan sinyal yang dihasilkan oleh badan sel melalui akson. *Akson* dari sel syaraf ini bercabang-cabang dan berhubungan dengan *dendrit* dari sel syaraf lain dengan cara mengirimkan impuls melalui *sinapsis*. *Sinapsis* adalah unit fungsional antara dua buah sel syaraf, misal A dan B, dimana yang satu adalah serabut akson dari neuron A dan satunya lagi adalah *dendrit* dari neuron B. Kekuatan *sinapsis* bisa menurun/meningkat tergantung seberapa besar tingkat *propagasi* (penyiaran) sinyal yang diterimanya. Impuls-impuls sinyal (informasi) akan diterima oleh neuron lain jika memenuhi batasan tertentu, yang sering disebut dengan nilai ambang (*threshold*) [10].

Konsep di atas adalah awal adanya konsep ANN, dimana konsep tersebut adalah bentuk representasi buatan dari otak manusia yang selalu mencoba untuk menstimulasikan proses pembelajaran pada otak manusia. Konsep ANN ini diimplementasikan ke dalam program komputer mampu menyelesaikan proses perhitungan selama proses pembelajaran.

2.5.1. Komponen *Artificial Neural Network*

ANN memiliki beberapa tipe yang berbeda-beda, akan tetapi hampir semua komponen yang dimiliki sama. Sama halnya jaringan syaraf pada otak manusia, ANN juga memiliki beberapa neuron unit yang saling berhubungan. *Neuron-neuron* tersebut melakukan transformasi informasi yang diterima melalui sambungan keduanya menuju neuron lain. Hubungan ini disebut dengan sebutan bobot (*Weight*). Informasi tersebut disimpan pada suatu nilai tertentu pada bobot tertentu. Berikut adalah struktur *Neuron* pada ANN :



Gambar 2.10. Struktur neuron ANN

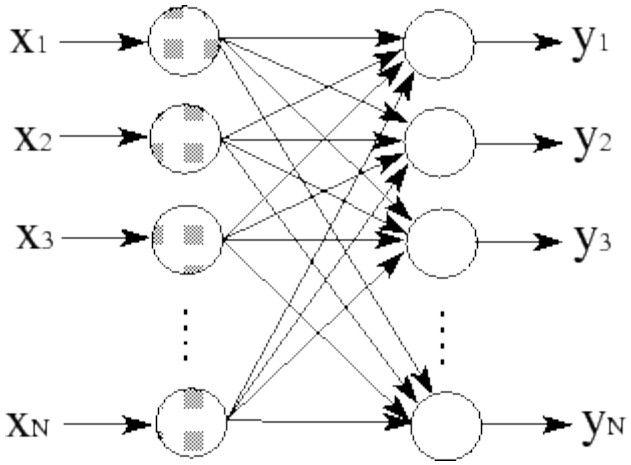
Cara kerja dari struktur neuron ANN sama dengan struktur jaringan syaraf pada manusia. Informasi berperan sebagai *input* akan dikirimkan dengan bobot kedatangan tertentu. *Input* tersebut kemudian diproses oleh suatu fungsi perambatan yang akan menjumlahkan nilai-nilai semua bobot yang datang. Hasil penjumlahan ini kemudian akan dibandingkan dengan suatu nilai ambang (*treshold*) tertentu melalui fungsi aktivasi setiap *neuron*. Jika *input* tersebut melewati suatu nilai ambang tertentu, maka *neuron* tersebut akan diaktifkan. Jika tidak, neuron tersebut tidak akan diaktifkan. Apabila neuron diaktifkan, selanjutnya neuron tersebut akan mengirimkan *output* melalui bobot-bobot *output*-nya ke semua *neuron* yang berhubungan dengannya, begitu seterusnya [10].

Penempatan *neuron* akan dikumpulkan ke dalam lapisan-lapisan *neuron* (*neuron layer*). Neuron-neuron pada satu lapisan dihubungkan dengan lapisan sebelumnya dan setelahnya, kecuali pada lapisan *input* dan *output*. *Input*-an yang masuk pada ANN akan dirambatkan ke lapisan *output.* Lapisan ini disebut dengan lapisan tersembunyi (*hidden layer*) [10]. Biasanya setiap *neuron* terletak pada lapisan yang sama sehingga setiap lapisan sama dan setiap neuron memiliki fungsi aktivasi. Lapisan dengan neuron harus selalu berhubungan. Faktor terpenting dalam menentukan kelakuan suatu neuron adalah terletak pada pola bobot dan fungsi aktivasinya.

2.5.2. Arsitektur *Artificial Neural Network*

Sebagaimana otak manusia yang memiliki banyak lapisan, ANN adalah jaringan *neuron* yang dikelompokkan ke dalam lapisan-lapisan. Beberapa arsitektur ANN diantaranya :

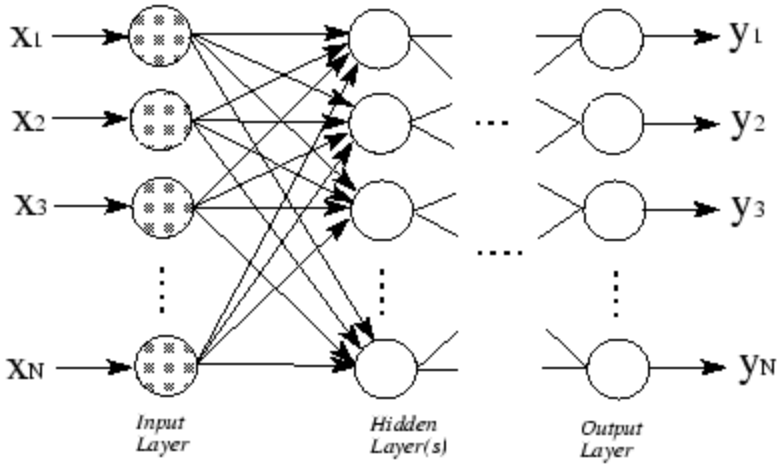
1. *Single Layer Perceptron* : Arsitektur lapisan tunggal terdiri dari 1 lapisan *input* dan 1 lapisan *output*. Setiap *neuron* pada lapisan *input* selalu terhubung dengan setiap *neuron* pada lapisan *output*. Arsitektur ini tidak memerlukan lapisan tersembunyi, karena *input* secara langsung akan diolah menjadi *output*.



Gambar 2.11. Arsitektur *Single Layer*

2. *Multiple Layers Perceptron* : Arsitektur lapisan jamak memiliki 3 lapisan utama yaitu lapisan *input*, lapisan *output* dan lapisan tersembunyi.

Arsitektur seperti ini dapat menyelesaikan permasalahan yang lebuh kompleks. Namun dalam proses pelatihan akan membutuhkan waktu yang lama.



Gambar 2.12. Arsitektur *Multiple Layers*

Pada bagian arsitektur MLP ANN merupakan lahirnya metode baru yang dapat disebut dengan *Deep Learning* atau *Deep Neural Network*. *Deep Learning* uang berbasis pada ekstraksi fitur data secara rinci dengan menirukan cara kerja otak manusia dalam hal melakukan pengiriman informasi. Sehingga *deep learning* dapat menghasilkan representasi pengetahuan yang lebih terperinci.

2.6. *Deep Learning*

Deep Learning merupakan salah satu bidang dari *Machine Learning* yang memanfaatkan *Artificial Neural Network* untuk implementasi permasalahan dengan dataset yang besar. Teknik *Deep Learning* memberikan arsitektur yang sangat kuat untuk *Supervised Learning*. Penambahan lebih banyak lapisan menjadikan model pembelajaran yang dapat mewakili data citra berlabel dengan lebih baik.

Aplikasi model ANN dengan banyak lapisan dapat ditangguhkan pada algoritma *Machine Learning* yang sudah ada sehingga komputer sekarang bisa belajar dengan kecepatan, akurasi, dan skala yang besar. Prinsip ini terus berkembang hingga *Deep Learning* semakin sering digunakan pada komunitas riset dan industri untuk membantu memecahkan banyak masalah data besar seperti *Computer vision*, *Speech recognition*, dan *Natural Language Processing*.

Apabila dihapakan pada kasus klasifikasi, salah satu fitur *Deep Learning yang* sering kali digunakan yaitu *Feature Engineering. Feature Engineering* adalah salah satu fitur utama dari *Deep Learning* berfungsi untuk mengekstrak pola yang berguna dari suatu data yang dapat memudahkan model untuk membedakan kelas. *Feature Engineering* juga merupakan teknik yang paling penting untuk mencapai hasil yang baik pada tugas prediksi. Namun, fitur ini lebih sulit untuk dipelajari dan dikuasai karena kumpulan data dan jenis data yang berbeda memerlukan pendekatan teknik yang berbeda juga.

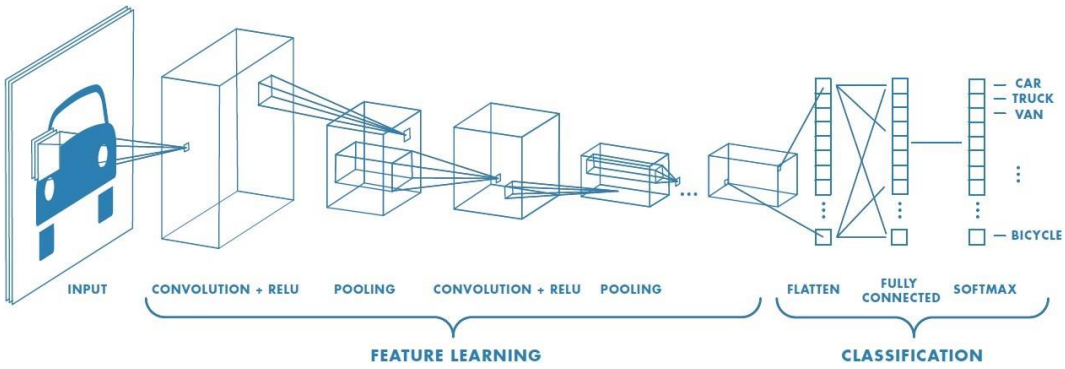
Algoritma yang digunakan pada *Feature Engineering* dapat menemukan pola umum yang penting untuk membedakan antara kelas dalam *Deep Learning*. Model *Convolutional Neural Network* sangat bagus untuk menemukan fitur yang baik pada citra ke lapisan berikutnya untuk membentuk hipotesis non linier yang dapat meningkatkan kekompleksitasan sebuah model. Model yang kompleks tentunya akan membutuhkan waktu pelatihan yang lama sehingga di dunia *Deep Learning* penggunaan GPU sudah sangatlah umum [11].

2.7. *Convolutional Neural Network*

Convolutional Neural Network (CNN) adalah salah satu metode dari *deep learning* yang merupakan pengembangan dari *multilayer perceptron* (MLP) yang dirancang khusus untuk melakukan mengolah data citra dua dimensi. CNN termasuk ke dalam kategori *Deep Neural Network*, karena kedalaman jaringan yang tinggi serta banyak diimplementasikan pada klasifikasi data citra. Klasifikasi citra pada dasarnya dapat digunakan dengan MLP. Namun dengan menggunakan metode MLP kurang sesuai, karena tidak dapat menyimpan informasi spesial dari data citra dan menganggap setiap *pixel* adalah fitur yang independen sehingga mendapatkan hasil yang kurang baik. Penelitian awal yang mendasari penemuan CNN pertama kali dilakukan oleh Hubel dan Wiesel mengenai *visual cortex* pada indra penglihatan kucing [12].

CNN adalah sebuah arsitektur yang dapat dilatih dan terdiri dari beberapa tahap. *Input* dan *output* dari setiap tahap terdiri dari beberapa *array* yang disebut dengan *feature map*. Setiap tahapan terbagi menjadi tiga *layer* yaitu *convolution layer*, *pooling layer* dan *full-connected layer*. Pada sebagian ahli peneliti CNN

membagi menjadi empal layer yaitu *convolution layer*, *subsampling layer*, *activation layer* dan *full-connected layer* [13]. Pada penelitian ini menggunakan arsitektur dengan jumlah tiga layer. Berikut adalah arsitektur dari *Convolutional Neural Network* dapat dilihat pada **Gambar 2.13**.



Gambar 2.13. Arsitektur *Convolutional Neural Network*

Berdasarkan pada **Gambar 2.13**, tahap pertama pada arsitektur CNN setelah *input* layer adalah tahapan *convolution layer*. Tahapan ini dilakukan dengan menggunakan sebuah kernel dengan ukuran tertentu. Perhitungan jumlah dari kernel tergantung dari jumlah fitur yang dihasilkan. Setelah itu dilanjutkan menuju proses fungsi aktivasi ReLU (*Rectifier Linear Unit*). Selanjutnya setelah keluar dari proses fungsi aktivasi kemudian melalui proses *pooling layer*. Proses dari *convolution layer* sampai proses fungsi aktivasi diulang beberapa kali sampai mendapatkan peta fitur yang cukup untuk melanjutkan ke proses *fully-connected neural network* dan selanjutnya adalah *output class.*

2.7.1. *Convolution Layer*

Convolution layer adalah bagian tahap awal setelah *input layer* pada arsitektur CNN. Tahap ini melakukan proses konvolusi pada *output* dari layer sebelumnya dan proses utama yang mendasari dari arsitektur CNN. Konvolusi adalah pengaplikasian sebuah fungsi pada *output* fungsi lain secara berulang. Operasi konvolusi merupakan operasi dua fungsi argumen bernilai riil dan menerapkan fungsi *output* sebagai *feature map* dari *input* citra. *Input* dan *output* dapat dilihat sebagai dua argumen bernilai riil. Operasi konvolusi dapat dituliskan sebagai berikut :

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \quad \textbf{(2.1)}$$

Keterangan :

s(t) = fungsi hasil operasi konvolusi

x = *input*

w = bobot (kernel)

Fungsi s(t) menghasilkan *output* tunggal berupa *feature map*. Argumen pertama adalah *input* yang merupakan x dan argumen kedua w sebagai kernel. Jika input sebagai citra dua dimensi, maka t sebagai *pixel* dan menggantinya dengan i dan j. Operasi konvolusi ke *input* dengan lebih dari satu dimensi dapat ditulis sebagai berikut :

$$s(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(i+m, j+n)K(m,n) \quad \textbf{(2.2)}$$

Berdasarkan persamaan **(2.2)** merupakan perhitungan dasar dalam operasi konvolusi dengan i dan j sebagai sebuah *pixel* dari citra. Perhitungan tersebut bersifat kumulatif dan muncul saat K sebagai kernel, kemudian I sebagai *input* dan kernel dapat dibalik relatif terhadap *input*. Alternatif operasi konvolusi dapat dilihat sebagai perkalian matriks antara citra masukan dan kernel dimana keluarannya dihitung dengan *dot product*. Selain itu, penentuan volume *output* juga dapat ditentukan dari masing-masing lapisan dengan *hyperparameters*. *Hyperparameter* yang digunakan pada persamaan di bawah ini digunakan untuk menghitung banyaknya neuron aktivasi dalam sekali *output*.

$$(W - F + 2P)/S + 1 \quad \textbf{(2.3)}$$

Keterangan :

W = Ukuran volume gambar

F = Ukuran filter

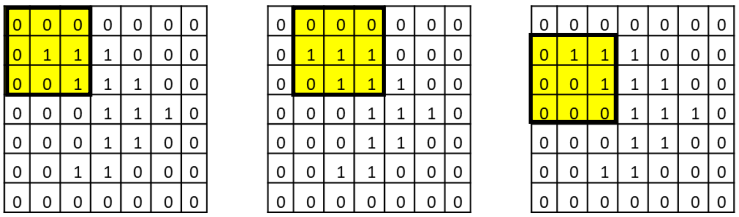
P = Nilai *padding* yang digunakan

S = Ukuran pergeseran (*Stride*)

Berdasarkan persamaan **(2.3)** dapat dihitung ukuran spasial dari volume *output* dimana *hyperparameter* yang dipakai adalah ukuran volume (W), filter (F), *Stride* yang diterapkan (S) dan jumlah *padding* nol yang digunakan (P).

2.7.2. *Stride*

Stride adalah parameter yang dapat menentukan berapa jumlah pergeseran filter. Jika nilai *stride* adalah satu, maka filter akan bergeser sebanyak satu *pixel* secara horizontal lalu vertikal [2]. Semakin kecil *stride* yang digunakan, maka semakin detail informasi yang didapatkan dari sebuah *input*, akan tetapi membutuhkan komputasi lebih jika dibandingkan dengan *stride* yang besar.



Gambar 2.14. Contoh proses *stride*

Pada **Gambar 2.14** menunjukkan pergeseran kernel berwarna kuning dengan ukuran kernel 3x3 bergeser sebanyak 1 baik vertikal maupun horizontal. Kernel tersebut akan terus bergerak sampai telah mensyuri semua nilai dari citra.

2.7.3. *Zero Padding*

Zero padding adalah parameter yang menentukan jumlah *pixel* (berisi nilai nol) yang akan ditambahkan di setiap sisi dari *input*. Hal ini digunakan untuk memanipulasi dimensi *output* dari *convolutional layer* (*faeture map*) [2].

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

Gambar 2.15. Contoh *zero padding*

Pada **Gambar 2.15** adalah contoh pemberian *zero padding*, dengan matriks semula 5x5 menjadi 7x7 dengan memberi *zero padding* sebesar 1.

2.7.4. *Rectified Linear Units* (ReLU)

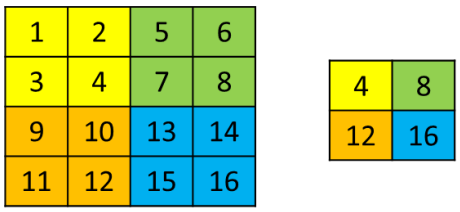
ReLU adalah salah satu fungsi aktivasi yang dapat digunakan dalam *deep learning*, terutama dalam metode CNN yang akan digunakan untuk *convolution layer dan hidden layers*. Berikut persamaan fungsi ReLU :

$$f(x) = \max(x, 0) \quad \textbf{(2.4)}$$

Cara kerja pada persamaan **(2.4)** yaitu apabila *input*-an fungsi di bawah 0 maka *output* fungsi tersebut ialah 0, sebaliknya apabila *input* fungsi di atas angka 0 maka *output* dari fungsi adalah angka tersebut [14].

2.7.5. *Pooling Layer*

Pooling layer adalah suatu cara untuk melakukan pengurangan ukuran *matrix*. *Pooling layer* terdiri dari sebuah filter dengan ukuran dan *stride* tertentu yang akan secara bergantian bergeser pada seluruh area *feature map*. Terdapat dua macam *pooling* yang dapat digunakan yaitu *average pooling* dan *max-pooling*. Nilai yang diambil pada *average pooling* adalah nilai rata-rata dari pooling, sedangkan untuk *max-pooling* adalah nilai tertinggi dari *pooling*. *Pooling layer* yang berada di antara lapisan konvolusi secara berulang dalam arsitektur CNN dapat secara progresif mengurangi ukuran volume *output* pada *feature map*, sehingga mengurangi jumlah parameter dan perhitungan jaringan bertujuan untuk mengendalikan *overfitting* [2].



Gambar 2.16. Contoh *max-pooling*

Pada **Gambar 2.16** menunjukkan contoh penggunaan *max-pooling* dengan *matrix input* sebesar 4x4 dan parameter pada kernel berukuran 2x2 serta *stride* sebanyak 2, menghasilkan *feature map* sebesar 2x2 dengan nilai tertinggi pada setiap kernel.

2.7.6. *Fully-Connected Layer*

Fully-connected layer adalah sebuah lapisan dimana semua *neuron* aktivasi dari lapisan sebelumnya terhubung dengan semua *neuron* di lapisan selanjutnya. *Input* layer ini adalah hasil dari proses *convolution layer*. Semua *neuron* aktivasi dari lapisan sebelumnya akan diubah menjadi data satu dimensi sebelum dihubungkan dengan lapisan selanjutnya, dengan tujuan agar diklasifikasikan secara linear. Berikut ini persamaan untuk menghitung nilai pada layer selanjutnya.

$$fc_k = b_k + \sum_{i=0}^{n} x_i w_{i,k} \ , k = 0,1,2,\cdots,t \qquad \textbf{(2.5)}$$

Keterangan:

y = keluaran

b = bias

n = banyaknya data pada *flatten*

x = nilai *flatten*

w = bobot

t = banyaknya target pada layer FC

2.7.7. Inisialisasi Bobot *Glorot Uniform*

Inisialisasi bobot *glorot uniform* merupakan salah satu metode untuk mendapatkan bobot awal. Metode bertujuan untuk mendapatkan rentang *random* angka untuk bobot awal [15]. Adapun persamaan untuk mendapatkan rentang *ramdom* angka sebagai berikut.

$$w = [-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}] \qquad \textbf{(2.6)}$$

Keterangan:

w = bobot

n_j = jumlah neuron pada layer tersebut

n_{j+1} = jumlah neuron pada layer berikutnya

2.7.8. *One Hot Encoding*

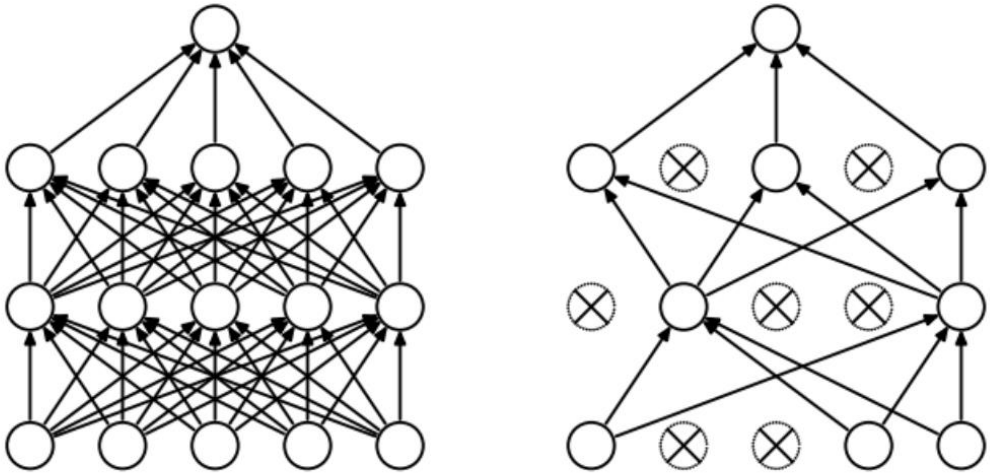
One hot encoding merupakan cara untuk mempresentasikan variabel kategori berupa vektor biner [16]. *One hot encoding* diperlukan untuk *machine*

learning karena pada algoritma *machine learning* tidak bisa bekerja secara langsung untuk mengategorikan. Maka data kategori tersebut akan diubah menjadi *number* yang dapat dikenali pada *machine learning*.

Sebagai contoh ada tiga kategori yaitu note-c1, note-d1 dan note-e1, data yang akan dikategorikan yaitu note-c1, note-c1, note-d1 dan note-e1. Tiga kategori tersebut akan diubah menjadi indeks *array* dimana note-c1 = 0, note-d1 = 1 dan note-e1 = 2. Hasil mengategorikan data tersebut akan berupa seperti note-c1 = [1,0,0], note-c1 = [1,0,0], note-d1 = [0,1,0] dan note-e1 = [0,0,1].

2.7.9. *Dropout Regularization*

Dropout Regularization adalah suatu teknik regulasi dengan tujuan memilih beberapa *neuron* secara acak dan tidak dipakai selama proses pelatihan. Hal ini berarti bahwa *neuron* yang tidak terpakai akan di berhentikan sementara dari jaringan dan bobot baru saat melakukan *backpropagation*. Berikut adalah gambar dari proses *dropout regularization* :



Gambar 2.17. Proses *dropout regularization*

Berdasarkan **Gambar 2.17** menunjukkan pada bagian (a) merupakan *neural network* yang biasa dengan memiliki dua *hidden layer*. Sedangkan (b) *neural network* dengan menggunakan *dropout regularization.* Penggunaan teknik ini akan berdampak pada performa model dalam melatih serta mengurangi *overfitting*.

2.7.10. *Softmax Classifier*

Softmax Classifier adalah algoritma *Logistic Regression* yang dapat digunakan untuk pengklasifikasian. Klasifikasi yang dapat dilakukan oleh algoritma *Softmax Classifier* adalah tugas klasifikasi kelas biner. Berikut persamaan *Softmax Classifier* dapat dilihat pada persamaan **(2.7)**.

$$f(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} = \frac{e^{(w^T x)_j}}{\sum_{k=1}^{K} e^{(w^T x)_k}} \quad \textbf{(2.7)}$$

Keterangan:

f(z) = hasil fungsi

j = pengulangan sebanyak kelas

w^T = bobot yang telah dilakukan *transpose*

x = nilai pada *hidden layer*

K = banyaknya kelas

2.7.11. *Cross Entropy Loss Function*

Loss function adalah fungsi yang mengukur seberapa bagus performa yang dihasilkan oleh model dalam melakukan prediksi terhadap target [17]. *Loss function* bekerja ketika model pembelajaran memberikan kesalahan yang perlu diperhatikan. *Loss function* yang baik adalah fungsi yang menghasilkan *error* yang diharapkan paling rendah.

Apabila suatu model memiliki kelas yang cukup banyak, perlu adanya cara untuk mengukur perbedaan antara probabilitas hasil hipotesis dan probabilitas kebenaran yang asli. Selama proses pelatihan banyak algoritma yang dapat menyesuaikan parameter sehingga perbedaan ini diminimalkan. Algoritma *Cross Entropy* adalah fungsi untuk mengetahui seberapa tinggi *error* atau *loss* yang dilakukan *machine learning* dengan nilai *output* berupa probabilitas [18]. Berikut persamaan yang digunakan untuk menghitung *Cross Entropy*:

$$L = -\sum_{i=0}^{N} y_i \log(p_i) \quad \textbf{(2.8)}$$

Keterangan:

N = banyaknya kelas

y = nilai *output* (1 atau 0)

p = nilai prediksi

2.7.12. *Back-propagation*

Back-propagation adalah suatu proses untuk memperbaiki bobot dan bias berdasarkan *error* yang didapat pada proses *pass-forward* dengan cara menghitung gradien pada setiap parameternya [17]. Metode *back-propagation* yang digunakan pada penelitian ini yaitu metode *chain rule* yaitu turunan dari suatu rumus [19]. Berikut ini persamaan yang digunakan untuk melakukan *back-propagation* serta tahapan-tahapan yang dilakukan:

1. Menghitung *gradient* pada bobot *fully-connected layer*.

$$\frac{\partial L}{\partial w_{i,j}} = \frac{\partial L}{\partial fcO_i} * \frac{\partial fcO_i}{\partial fcI_i} * \frac{\partial fcI_i}{\partial w_{i,j}} \quad \textbf{(2.9)}$$

2. Menghitung turunan dari fungsi *cross entropy*.

$$\frac{\partial L}{\partial fcO_i} = \left(1\{y = y_i\} - \frac{e^{fcI_i}}{\sum_{j=0}^{t} e^{fcI_j}}\right) fcO_i, i = 0,1,2, \dots, t \quad \textbf{(2.10)}$$

3. Menghitung turunan *output* dari *fully-connected layer* setelah melalui fungsi aktivasi *softmax*.

$$\frac{\partial fcO_i}{\partial fcI_i} = \frac{e^{fcI_i} * (\sum_{n \neq i}^{t} e^{fcI_n})}{\sum_{n=0}^{t} {e^{fcI_n}}^2}, i = 0,1,2, \dots, t \quad \textbf{(2.11)}$$

4. Menghitung turunan bobot dari *fully-connected layer*.

$$\frac{\partial fcI_i}{\partial w_{i,j}} = OutputLayerSebelumnya_j, i = 0,1,2, \dots, t \quad \textbf{(2.12)}$$

Keterangan:

fcI_i = Nilai masukan *fully-connected layer* pada indeks ke i

fcO_i = Nilai keluaran *fully-connected layer* pada indeks ke i

L = fungsi *loss* yang digunakan (*cross entropy*)

w = bobot

t = jumlah neuron

j = 0,1,2, ... jumlah neuron pada layer sebelumnya

b = bias

2.7.13. *Adam Optimizer*

Adam optimizer merupakan suatu cara untuk mengoptimasi suatu parameter, optimasi tersebut dapat membuat parameter menjadi maksimum atau minimum. *Adam Optimizer* adalah salah satu optimasi yang menggabungkan metode AdaGrad dan RMSProp [20]. Ada beberapa parameter yang perlu diinisialisasi dari awal yaitu:

$m_0 = 0$ $\beta_1 = 0.9$

$v_0 = 0$ $\beta_2 = 0.999$

$t = 0$ $\epsilon = 10^{-8}$

$\alpha = 0.001$

Adapun tahapan *adam optimizer* dalam melakukan optimasi yaitu [20]:

1. Menambah t pada setiap iterasi

$$t = t + 1 \quad \textbf{(2.13)}$$

2. Menghitung gradien

$$g_t = \nabla_\theta f_t(\theta_{t-1}) \quad \textbf{(2.14)}$$

3. Memperbaharui bias momen pertama

$$m_t = \beta_1 . m_{t-1} + (1 - \beta_1) . g_t \quad \textbf{(2.15)}$$

4. Memperbaharui bias momen kedua

$$v_t = \beta_2 . v_{t-1} + (1 - \beta_2) . g_t^2 \quad \textbf{(2.16)}$$

5. Menghitung koreksi bias momen pertama

$$\widehat{m_t} = \frac{m_t}{(1 - \beta_1^t)} \quad \textbf{(2.17)}$$

6. Menghitung koreksi bias momen kedua

$$\widehat{v_t} = \frac{v_t}{(1 - \beta_2^t)} \quad \textbf{(2.18)}$$

7. Memperbaharui parameter

$$\theta_t = \theta_{t-1} - \alpha . \widehat{m_t} / (\sqrt{\widehat{v_t}} + \epsilon) \quad \textbf{(2.19)}$$

Keterangan:

g = Gradien

m = Momen pertama

v = Momen kedua

β_1, β_2 = *Exponential decay rates*

α = *Stepsize*

θ = Parameter yang diperbaiki

2.8. UML (*Unified Modeling Language*)

UML merupakan salah satu standar bahasa pemodelan untuk mendefinisikan kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun dan dokumentasi dari suatu perangkat lunak [21]. Pemodelan UML dirancang untuk memodelkan perangkat lunak berstandar pemrograman berorientasi objek. UML berperan sebagai *blueprint* dari sebuah perangkat lunak yang berfungsi apabila developer perangkat lunak tersebut berganti developer maka developer baru akan dapat mudah memahami model perangkat lunak tersebut dengan melihat *blueprint* UML.

Pada pemodelan UML memiliki beberapa diagram yang dapat digunakan pada penelitian ini yaitu:

1. *Use case* diagram

   Use case diagram merupakan pemodelan untuk mengetahui fungsi apa saja yang ada di dalam perangkat lunak yang akan dibangun dan siapa saja yang berhak untuk menggunakan fungsi-fungsi tersebut [21].

2. *Use case* skenario

   Use case skenario merupakan detail penjelasan fungsi dari sebuah *use case* agar dapat mudah dipahami oleh pembaca.

3. *Activity* diagram

   Activity diagram merupakan gambaran aliran kerja dari sebuah sistem pada perangkat lunak. Dalam *activity* diagram yang perlu diperhatikan adalah menggambarkan aktivitas dari sistem bukan apa yang dilakukan oleh aktor [21].

4. *Sequence* diagram

Sequence diagram merupakan gambaran kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* akan dikirimkan dan diterima antar objek [21].

5. *Class* diagram

Class diagram menggambarkan struktur dari sistem yang akan dibangun dari segi pendefinisian kelas-kelas yang akan dibuat [21].

2.9. Python

Python yang didirikan pada tahun 1991 oleh seorang developer Guido Van Rossum merupakan salah satu bahasa pemrograman tingkat tinggi yang bersifat interpreter, *general-purpose* dan pemrograman berorientasi objek. Dalam melakukan *coding*, python membantu programmer untuk melakukan *coding* yang lebih sedikit dan simpel dari pada Java atau C++ [22].

Beberapa kelebihan python yang menjadi pilihan pada penelitian ini yaitu banyaknya *library* yang tersedia dan sampai sekarang masih terus dikembangkan. Kelebihan lain yang dimiliki python dapat diintegrasikan dengan bahasa pemrograman lain seperti C, C++ atau Java [23].

Pada penelitian ini menggunakan beberapa *library* antara lain keras, numpy, pillow, opencv dan matplotlib. Keras merupakan suatu *library* dengan tujuan untuk membangun suatu sistem *machine learning*, *library* keras banyak digunakan pada pembangunan *machine learning* karena mudah diimplementasikan dan menghemat *script code* yang ditulis. Numpy merupakan suatu *library* yang berfungsi untuk mempermudah perhitungan *array* yang lebih dari satu dimensi, karena pada penelitian ini data masukan berupa citra yang memiliki nilai *pixel* yang banyak dan akan dimasukkan ke dalam *array* yang lebih dari satu dimensi. Pillow dan opencv merupakan suatu *library* yang berfungsi untuk melakukan pengolahan citra. Matplotlib merupakan *library* yang berfungsi untuk menampilkan data analisis berbentuk *chart* dan dapat pula untuk menampilkan gambar secara rapih.