BAB II

TINJAUAN PUSTAKA

2.1 Kajian Pustaka

Berikut adalah beberapa penelitian relevan yang menjadi kajian pustaka dalam penelitian ini, yaitu penelitian dari Kotecha dan Young (2018) [28], Irene, Borrelli, Zanoni, Buccoli, dan Sarti (2019) [29], Adiyansjah, Gunawan, dan Suhartono (2019) [30], Dimas Fiqh Puskoaji (2019) [31], dan Gunawan, Iman, dan Suhartono (2020) [32].

Penelitian yang dilakukan oleh Kotecha dan Young (2018) dengan judul "Generating Music using an LSTM Network" [28] mempunyai tujuan untuk membuat model musik dengan arsitektur jaringan yang dapat memprediksi struktur musik dan menghasilkan musik polifoni yang harmonis. Metode penelitian yang digunakan adalah metode model probabilitas RNN-LSTM Bi-aksial yang dilatih dengan kernel seperti kernel konvolusi. Hasil penelitian menunjukkan bahwa model LSTM dua *layer* dapat mengingat peluang harmonik dan ritme melodi dari masukan data MIDI polifoni *Bach*. Hasil keluarannya dianalisis melalui pendekatan kuantitatif dan kualitatif dan hasil analisisnya menunjukkan model bekerja dengan baik dalam merancang musik polifoni.

Persamaan penelitian Kotecha dan Young (2018) dengan penelitian ini yaitu sama-sama menghasilkan musik yang harmonis dengan teknik penyelesaian menggunakan jaringan saraf RNN. Sedangkan perbedaan penelitian Kotecha dan Young (2018) dengan penelitian ini yaitu terletak pada *dataset* yang diaplikasikan dan jenis sel RNN yang digunakan. Pada penelitian milik Kotecha dan Young (2018) menggunakan *dataset* berupa MIDI polifoni piano dengan luaran berupa musik instrumen piano dalam bentuk MIDI, sedangkan pada penelitian ini *dataset* yang digunakan berupa MP3 dari suara vokal manusia dan suara instrumen musik gitar dengan luaran berupa instrumen musik gitar itu sendiri. Selain itu, penelitian milik Kotecha dan Young mengunakan sel LSTM sebagai model prediksinya, sedangkan pada penelitian ini menggunakan sel GRU [28].

Penelitian yang dilakukan oleh Irene, Borrelli, Zanoni, Buccoli, dan Sarti (2019) dengan judul "Automatic Playlist Generation using Convolutional Neural Networks and Recurrent Neural Networks" [29] mempunyai tujuan untuk membuat daftar putar otomatis yang menganalisis daftar putar buatan tangan, memahami struktur dan evolusi serta menghasilkan daftar putar baru yang sesuai. Metode penelitian yang digunakan adalah metode RNN dan CNN. Metode RNN sebagai pemodelan urutan pada struktur daftar putar sebagai evolusi dari waktu ke waktu terhadap representasi fiturnya, sedangkan metode CNN sebagai model untuk representasi fitur untuk setiap lagu. Hasil penelitian menunjukkan bahwa ketika model dievaluasi menggunakan kumpulan data pada daftar putar *Art of the Mix-2011*, model mampu menghasilkan performa yang memuaskan, dari segi pengurangan *input* maupun peningkatan ruang lingkup prediksi untuk dua langkah ke depan dari lagu saat ini.

Persamaan penelitian Irene, Borrelli, Zanoni, Buccoli, dan Sarti (2019) dengan penelitian ini yaitu sama-sama menggunakan metode RNN untuk memprediksi data sebagai *output* dari hasil ekstraksi fitur melalui *latent* yang dihasilkan dari *autoencoder* menggunakan model CNN. Sedangkan perbedaan penelitian Irene, Borrelli, Zanoni, Buccoli, dan Sarti (2019) dengan penelitian ini yaitu pada pengaplikasian model RNN terhadap objek. Pada penelitian milik Irene, Borrelli, Zanoni, Buccoli, dan Sarti (2019) model RNN diaplikasikan pada objek berupa daftar/*playlist* lagu di suatu *platform streaming* musik, sedangkan pada penelitian ini objek yang diaplikasikan pada model RNN berupa suara vokal manusia dan instrumen musik gitar yang bernada [29].

Penelitian yang dilakukan oleh Adiyansjah, Gunawan, dan Suhartono (2019) dengan judul "Music Recommender System Based on Genre using Convolutional Recurrent Neural Networks" [30] mempunyai tujuan untuk membuat sistem rekomendasi musik secara otomatis dengan memberikan rekomendasi lagu yang cocok untuk pengguna. Metode penelitian yang digunakan adalah CRNN, yaitu kombinasi dari model CNN dan RNN yang diimplementasikan untuk ekstraksi fitur dan klasifikasi *genre* musik pada sistem rekomendasi musik. Hasil penelitian menunjukkan bahwa CRNN memiliki

kinerja yang baik secara keseluruhan, baik dalam mempertimbangkan fitur frekuensi maupun pola urutan waktunya. Sehingga, dapat dikatakan bahwa pengguna lebih tertarik pada sistem rekomendasi yang mempertimbangkan *genre* musik daripada sistem rekomendasi yang hanya berdasarkan pada kesamaan jenis musik itu sendiri.

Persamaan penelitian Adiyansjah, Gunawan, dan Suhartono (2019) dengan penelitian ini adalah sama-sama mengkombinasikan model *deep learning* CNN dan RNN pada studi kasus di bidang musik dan *dataset* yang digunakan sama-sama dalam bentuk MP3. Sedangkan perbedaan penelitian Adiyansjah, Gunawan, dan Suhartono (2019) dengan penelitian ini yaitu pada pengimplementasian metode CNN-RNN terhadap teknik penyelesaian dalam ekstraksi fitur dan objek yang diaplikasikan. Pada penelitian milik Adiyansjah, Gunawan, dan Suhartono (2019) mengimplementasikan metode CNN-RNN pada ekstraksi fitur untuk mengetahui jarak kemiripan antar fitur ketika mengklasifikasikan *genre* dan jenis musik, sedangkan pada penelitian ini metode CNN diimplementasikan sebagai model untuk ekstraksi fitur-fitur penting dari sinyal suara secara sekuensial yang kemudian akan diprediksi menggunakan model RNN untuk menghasilkan *output* berupa suara instrumen dari data masukan vokal manusia [30].

Penelitian yang telah dilakukan oleh Dimas Fiqh Puskoaji (2019) dengan judul "Pembuatan Musik Menggunakan Musik Generator Dengan Metode Recurrent Neural Network" [31] mempunyai tujuan untuk membuat teknologi generator musik otomatis yang ditujukan untuk mereka yang ingin berekspresi dalam menciptakan musik meskipun mereka tidak memiliki kemampuan dalam bermusik. Metode penelitian yang digunakan adalah RNN, yaitu sebagai model yang digunakan untuk proses *training file* MIDI yang telah terekstraksi dari MIDI menjadi teks dan kemudian akan dilakukan prediksi terhadap notasi musik. Setelah diprediksi, maka notasi hasil prediksi akan di-*generate* dalam bentuk MIDI yang nantinya akan diproses kembali menggunakan bantuan generator musik. Hasil penelitian menunjukkan bahwa metode RNN dapat diaplikasikan dalam pembuatan musik menggunakan *file* MIDI dan menghasilkan evaluasi *loss*

yang rendah. Selain itu, hasil evaluasi melalui ahli musik menunjukkan bahwa sistem mampu menghasilkan musik yang unik dan bervariatif.

Persamaan penelitian Dimas Fiqh Puskoaji (2019) dengan penelitian ini adalah sama-sama menggunakan RNN sebagai model untuk memprediksi data dari hasil ekstraksi fitur. Sedangkan perbedaan penelitian Dimas Fiqh Puskoaji (2019) dengan penelitian ini adalah pada *dataset* yang digunakan serta hasil keluarannya. Pada penelitian Dimas Fiqh Puskoaji (2019) menggunakan satu jenis *dataset* berupa MIDI dengan keluaran dalam bentuk MIDI yang di-*generate* ke dalam editor *platform Digital Audio Workstation*, sedangkan pada penelitian ini *dataset* yang digunakan ada dua jenis, yaitu *dataset* suara vokal manusia dan instrumen musik gitar dengan format MP3 yang di-*generate* menggunakan model RNN itu sendiri [31].

Penelitian yang telah dilakukan oleh Gunawan, Iman, dan Suhartono (2020) dengan judul "Automatic Music Generator Using Recurrent Neural Network" [32] mempunyai tujuan untuk mengembangkan sebuah generator musik otomatis menggunakan dua sub-model RNN yaitu LSTM dan GRU yang kemudian hasil evaluasinya dibandingkan satu sama lain. Metode penelitian yang digunakan adalah sub-model dari RNN yaitu LSTM dan GRU yang dirancang sebagai model generator yang nantinya akan dievaluasi untuk analisa performa dari setiap model generator yang mengklasifikasikan tiap *input* berdasarkan era musiknya. Hasil penelitian menunjukkan bahwa model GRU dua *layer* menghasilkan nilai *recall* dengan skor 70% menyerupai pola komposer dalam musik dan hasil evaluasi secara subjektif menunjukkan bahwa musik yang dihasilkan dengan skor 6,85 dari 10.

Persamaan penelitian Gunawan, Iman, dan Suhartono (2020) dengan penelitian ini adalah sama-sama mengaplikasikan sub-model GRU pada teknik penyelesaian studi kasus di bidang musik. Sedangkan perbedaan penelitian Gunawan, Iman, dan Suhartono (2020) dengan penelitian ini adalah pada proses ekstraksi fitur dan *dataset* yang digunakan. Pada penelitian Gunawan, ekstraksi fitur yang dilakukan adalah mengkonversi *file* MIDI menjadi matriks melalui proses *encoding*, dan *dataset* yang digunakan hanya satu jenis berupa file MIDI.

Sedangkan pada penelitian ini proses ekstraksi fitur yang dilakukan adalah mengkonversi masukan suara vokal manusia dan instrumen musik gitar menjadi representasi *latent* menggunakan CNN agar lebih mudah diprediksi menggunakan model RNN. Selain itu, *dataset* yang digunakan ada dua yaitu suara vokal manusia dan suara instrumen musik gitar [32].

Tabel 2.1 Tabel Penelitian Sebelumnya

No	**Penulis (Tahun)**	**Masalah**	**Metode/Teknik Penyelesaian**	**Perbedaan**
1.	Kotecha dan Young (2018) [28]	Model musik untuk mengingat detail masa lalu dengan pemahaman yang jelas dan koheren tentang struktur musik agar dapat menghasilkan musik polifoni yang harmonis.	Metode model probabilitas RNN-LSTM Bi-aksial yang dilatih dengan kernel seperti kernel konvolusi.	*Dataset* yang diaplikasikan berupa file MIDI Polifoni dan MIDI piano serta luaran musik yang dihasilkan yaitu berupa instrumen piano.
2.	Irene, Borrelli, Zanoni, Buccoli, dan Sarti (2019) [29]	Sebagian besar layanan di *platform streaming* musik didasarkan pada daftar putar yang dihasilkan secara manual oleh kurator atau pengguna. Beberapa kasus, sistem tidak bekerja dengan baik untuk menangani sejumlah besar daftar musik.	Metode RNN untuk pemodelan urutan lagu pada struktur daftar putar sebagai evolusi dari waktu ke waktu terhadap representasi fiturnya dan CNN untuk mempelajari representasi fitur untuk setiap lagu.	Aplikasi model RNN terhadap objek di bidang musik sebagai model pengurutan. Objek yang diaplikasikan yaitu daftar/*playlist* lagu di suatu *platform streaming* musik.
3.	Adiyansjah, Gunawan, dan Suhartono (2019) [30]	Ketersediaan layanan streaming musik komersial di berbagai platform membuat proses sortir musik digital memakan waktu yang lama dan	Menggunakan model arsitektur *Convolutional Recurrent Neural Network* untuk ekstraksi fitur dan mengetahui jarak	Implementasi ekstraksi fitur, yaitu pada klasifikasi *genre* dan jenis musik sebagai dasar untuk sistem

No	Penulis (Tahun)	Masalah	Metode/Teknik Penyelesaian	Perbedaan
		menyebabkan kelelahan informasi.	kemiripan antar fitur ketika mengklasifikasikan genre dan jenis musik pada sistem rekomendasi musik otomatis.	rekomendasi.
4.	Dimas Fiqh Puskoaji (2019) [31]	Tidak semua orang mempunyai kemampuan dalam bermusik dan menciptakan musik karena bukan bidangnya dan ketidakpahaman tentang musik. Bagi mereka yang ingin berekspresi dalam menciptakan musik, maka dibutuhkan generator musik otomatis agar mereka dapat menciptakan musik yang sesuai dengan keinginan mereka.	Metode RNN sebagai model yang digunakan untuk proses training file MIDI yang telah terekstraksi dari MIDI ke teks.	*Dataset* yang digunakan hanya satu dan dalam bentuk MIDI serta luaran yang dihasilkan yaitu dalam bentuk MIDI yang di-*generate* ke dalam editor *platform DAW*.
5.	Gunawan, Iman, dan Suhartono (2020) [32]	Perbandingan performa sub-model RNN yaitu LSTM dan GRU dalam mengaplikasikan generator musik otomatis yang menggunakan file MIDI sebagai input. Hasilnya akan dievaluasi dengan responden relawan di bidang musik.	Metode LSTM dan GRU yang dirancang sebagai model generator yang nantinya akan dievaluasi untuk analisa performa dari setiap model generator yang mengklasifikasikan tiap *input* berdasarkan era musiknya.	*Dataset* yang diaplikasikan hanya satu jenis, dalam bentuk MIDI. Luaran yang dihasilkan yaitu musik dalam bentuk MIDI. Ekstraksi fitur dilakukan dengan mengkonversi *file* MIDI menjadi matriks menggunakan *encoding* MIDI.

2.2 Dasar Teori

Pada penelitian ini, akan digunakan beberapa dasar teori untuk mendukung pemahaman dan konteks mengenai topik yang akan diteliti, serta sebagai pedoman untuk pengembangan dan strategi yang tepat untuk mendapatkan hasil penelitian yang diharapkan. Berikut adalah penjelasan beberapa dasar teori yang relevan dengan penelitian ini.

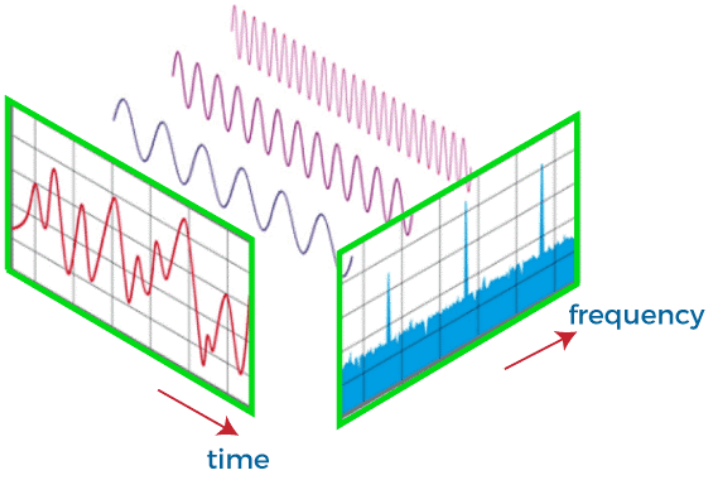
2.2.1 *Pitch* Harmoni

Pitch atau nada merupakan istilah yang digunakan pada musik, yaitu rendah-tingginya suara. *Pitch* mempunyai peran penting dalam pembuatan sebuah musik, karena untuk menghasilkan sebuah melodi maka perlu mempertimbangkan kombinasi urutan suatu *pitch*. Secara ilmiah, *pitch* dihasilkan melalui frekuensi getaran gelombang pada suara atau bunyi. Dapat dikatakan bahwa, frekuensi menentukan rendah-tingginya nada. Semakin besar atau cepat suatu frekuensi maka akan menghasilkan nada yang tinggi, sebaliknya semakin kecil atau lambat suatu frekuensi maka akan menghasilkan nada yang rendah. Ketika mendengarkan musik, urutan suara yang terdengar dari instrumen sebenarnya berasal dari frekuensi nada dasar yang diikuti dengan serangkaian frekuensi nada gabungan tunggal. Gabungan dari frekuensi nada dasar dan frekuensi nada gabungan tunggal inilah yang disebut sebagai harmoni. Jika seorang musisi memainkan sebuah musik dan membunyikan kombinasi nada tanpa membunyikan frekuensi nada dasarnya, maka musisi tersebut telah membuat suatu harmoni [33][34][35].

2.2.2 *Discrete Cosine Transform*

Discrete cosine transform (DCT) merupakan metode matematika untuk mengubah fungsi waktu diskrit menjadi fungsi frekuensi diskrit, atau melakukan transformasi secara matematis dari *domain* waktu ke *domain* frekuensi. Implementasinya dapat diterapkan pada berbagai kasus, seperti kompresi data,

analisis gambar, pemrosesan sinyal pada radio, dan pemrosesan suara. Pada pemrosesan suara, DCT seringkali digunakan sebagai ekstrasi fitur untuk kompresi data pada sinyal suara. Dengan mentransformasi sinyal suara pada domain waktu menjadi sinyal suara pada domain frekuensi, kita dapat melakukan analisis distribusi frekuensi sinyal yang berbeda-beda, sehingga untuk mengetahui di mana letak *pitch* berada di setiap blok sinyal suara akan lebih mudah untuk diidentifikasi ke dalam domain frekuensi.



Gambar 2.1 Ilustrasi Perubahan Domain Waktu ke Domain Frekuensi pada DCT

Rumus umum DCT adalah sebagai berikut:

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right), \quad k = 0, 1, \ldots, N-1 \qquad (1)$$

Berdasarkan rumus di atas, X_k adalah koefisien frekuensi ke-*k*, x_n adalah sampel sinyal dalam domain waktu ke-*n*, dan *N* adalah panjang sinyal.

Pada studi kasus pemrosesan suara, DCT merupakan metode komputasi yang efisien dengan jumlah perhitungan yang lebih sedikit dibanding *Fast Fourier Transform* (FFT), karena DCT cenderung memiliki sifat simetris dalam mengidentifikasi setiap blok sinyal suara ke dalam domain frekuensi serta

mengahasilkan data frekuensi yang lebih nyata dalam keadaan sebenarnya, dibandingkan dengan FFT yang menghasilkan data lebih kompleks [36][37][38].

2.2.3 Instrumen Musik

Instrumen musik adalah sebuah salah komponen dari suatu musik yang dihasilkan melalui gabungan dari harmoni beberapa alat musik. Dalam musik, instrumentasi mengacu pada kombinasi instrumen yang digunakan untuk menampilkan lagu atau karya, serta suara masing-masing instrumen. Musisi harus dapat membedakan instrumen mana yang dimainkan pada waktu tertentu. Penulis lagu dan komposer dapat sangat meningkatkan pilihan kreatif mereka dengan memiliki kosakata mental yang besar dari suara instrumen yang mereka miliki. Latar belakang musik menjadi salah satu komponen terpenting dalam menciptakan sebuah karya lagu. Pada saat ini, terdapat banyak contoh latar belakang otomatis yang dibuat berdasarkan alat musiknya, seperti gitar dan piano [39].

2.2.4 Akapela

Akapela adalah salah satu jenis musik yang mengacu pada aransemen musik dengan suara nyanyian tunggal maupun grup tanpa iringan alat musik atau instrumental. Pada umumnya akapela dapat dilakukan untuk nyanyian solo atau vokal grup tergantung nyanyian seperti apa yang ingin ditampilkan. Ketika seseorang bernyanyi tanpa bantuan alat musik atau backing track, maka dapat dikatakan dia sedang melakukan akapela. Semua suara dalam musik akapela berasal dari suara asli manusia, bahkan jika beberapa suara vokal terdengar seperti suara alat musik sungguhan. Di industri musik saat ini, suara nyanyian menjadi komponen penting dalam menghasilkan sebuah karya musik yang kompleks [9][10][40].

2.2.5 *Dataset*

Dataset adalah istilah yang sering digunakan pada *machine learning* yang secara sederhananya adalah kumpulan data yang siap diproses oleh suatu program

komputer sebagai suatu unit untuk dianalisis dan diprediksi. Komputer tidak melihat data dengan cara yang sama dengan manusia, maka dari itu data yang dikumpulkan harus seragam, agar dapat dipahami oleh komputer agar lebih mudah untuk diproses. Dataset yang baik adalah dataset yang sesuai dengan standar kuantitas dan kualitas tertentu, agar proses *training* program dapat berjalan dengan cepat dan lancar [41]. Terdapat beberapa dataset yang sering digunakan pada *deep learning*, yaitu tabular, teks, suara, gambar.

2.2.6 *Spleeter*

Spleeter adalah sebuah pustaka *Python* yang dibuat oleh tim R&D Deezer (suatu *platform streaming* musik) yang dapat melakukan pemisahan sumber audio menggunakan model terlatih dengan *Tensorflow*. Pada dasarnya, model pemisahan sumber audio harus membedakan antara batang audio yang berbeda dalam trek musik, seperti vokal, instrumen tertentu, atau suara instrumen grup. *spleeter* memiliki model terlatih untuk pemisahan sumber audio yang berbeda, seperti pemisahan dua batang (vokal dan iringan), pemisahan empat batang (vokal, bass, drum, dan lain-lain), dan pemisahan lima batang (vokal, bass, drum, piano, dan lain-lain), yang merupakan alat pertama yang menawarkan pemisahan lima batang. Selain itu, *spleeter* memungkinkan pengguna untuk melatih model pemisahan sumber audio mereka sendiri atau menyesuaikan model yang telah dilatih sebelumnya untuk kasus penggunaan tertentu. *Spleeter* menggunakan model terlatih U-Nets, yaitu jaringan saraf *convolutional encoder*/*decoder* (CNN). U-Nets terdiri dari 12 lapisan, 6 lapisan untuk pembuat *encoder* dan 6 untuk *dekoder* yang dilatih pada dataset internal dari Deezer [42].

2.2.7 *Deep Learning*

Deep learning merupakan bagian dari *machine learning* sebagai pendekatan analisis data dengan ukuran relatif besar, yang sering diaplikasikan pada kasus regresi, pemodelan grafis, optimasi, pengenalan pola, pemrosesan sinyal [43]. Dalam ilmu data, *deep learning* menjadi komponen penting yang mencakup pemodelan prediktif dan statistika. *Deep learning* dirancang untuk mengenali pola

dan keterkaitan antar data, berdasarkan jaringan syaraf pada otak manusia, sehingga secara alami dapat memahami korelasi antar data secara otomatis. Maka tak heran, *deep learning* menjadi jenis khusus *machine learning* yang didasarkan pada jaringan syaraf tiruan. Jaringan syaraf tiruan yang dibentuk dapat berlapis-lapis, sehingga dapat mensimulasikan syaraf otak manusia untuk mempelajari sejumlah data yang besar. Tentu saja, model komputasi dengan arsitektur jaringan syaraf berlapis-lapis ini juga dilatih layaknya *machine learning* pada umumnya dengan menggunakan sekumpulan pasangan data *x* dan data *y* (berlabel) dengan jumlah yang besar [43][44][45]. Beberapa jaringan syaraf tiruan ini mempunyai perannya masing-masing, seperti membuat perkiraan dan meningkatkan akurasi.

Contoh implementasi *deep learning* adalah klasifikasi teks pada tinjauan komprehensif [46], pendeteksi suara patologis [47], segmentasi gambar [48], deteksi objek biner pada video CCTV [49], bahkan sampai penyintesisan data tabular otomatis [50]. Selain itu, teknologi ini juga dapat diterapkan pada kasus *self-driving car* yang memungkinkan sebuah mobil mengenali tanda berhenti dan membedakan pejalan kaki dari lampu lalu lintas [51].

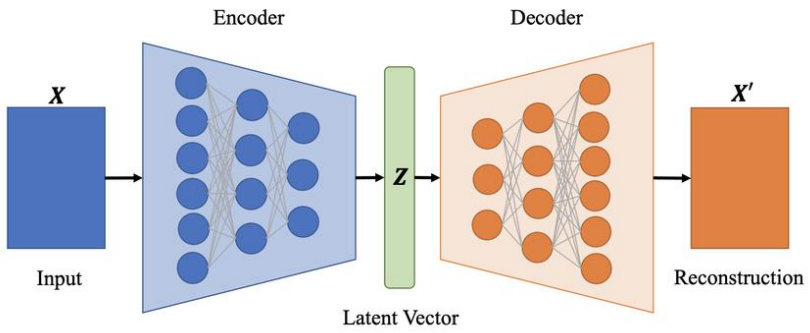
Struktur inti *deep learning* sama dengan *machine learning*, yaitu *training*, *validation*, dan *testing*. *Training* adalah proses melatih model terhadap sekumpulan data acak dari bagian *dataset* yang dikhususkan untuk *training*. Model akan dilatih pada setiap *epoch* secara berulang kali pada *dataset* yang sama untuk mempelajari fitur dari data tersebut. Ketika model menghasilkan prediksi yang akurat, maka model dapat diterapkan pada *dataset* baru untuk diuji coba kembali.

Validation adalah proses melatih model menggunakan dua bagian *dataset*, bagian pertama untuk *training* dan bagian sisanya untuk validasi model selama *training*. Selama proses *training*, model akan mengklasifikasikan setiap *input* dari *dataset* validasi. Klasifikasi dilakukan berdasarkan data yang dipelajari saat proses *training*. Dengan adanya *validation*, akan lebih mudah mengetahui bahwa model tersebut *overfitting* atau *underfitting*. Saat proses *training* dan validasi, data yang digunakan harus diberi label agar lebih mudah mengetahui metrik dan nilai akurasi dari setiap *epoch*nya [52][53].

Testing adalah tahap dimana menguji suatu model dari proses *training* menggunakan sekumpulan data yang terpisah dari data *training* dan data validasi. Setelah model di-*train* dan divalidasi, kemudian model akan di-*test* untuk memprediksi keluaran dari data yang tidak berlabel. Perlu diketahui bahwa, data yang melalui proses *testing* bukanlah sama dengan data *training*, karena kita bermaksud menguji pemahaman suatu model jika di-*test* dengan data yang sama sekali belum pernah ia pelajari sebelumnya. Ini perbedaan utama antara tahapan *testing* dengan tahapan lainnya.

2.2.8 *Autoencoder*

Autoencoder adalah suatu jenis model jaringan saraf yang memiliki *input* dan *output* yang sama. Tujuannya adalah untuk mempelajari data *input* dan merekonstruksinya kembali. *Autoencoder* seringkali digunakan untuk mereduksi dimensi fitur. Dalam proses pelatihannya, *autoencoder* akan menerima data *input* yang tidak memiliki label dan melakukan pengkodean pada data tersebut. Setelah dilakukan pengkodean, *autoencoder* akan berusaha merekonstruksi data seakurat mungkin menggunakan model *decoder*. Oleh karena itu, *autoencoder* terdiri dari dua model utama, yaitu *encoder* dan *decoder*, seperti yang diilustrasikan pada Gambar 2.1 [54].



Gambar 2.2 Ilustrasi *Autoencoder*

Fungsi *encoder* adalah untuk menentukan pemetaan data *input X* ke dalam data *latent Z*. Data *latent* biasanya jauh lebih kecil daripada data *input* asli,

sehingga hanya fitur-fitur utama dari variasi data yang diambil dalam data *latent* tersebut. Sementara itu, *decoder* akan memetakan kembali vektor *latent* Z ke dalam ruang dimensi *input* asli, sehingga menghasilkan data asli yang diwakili oleh *X'*. Sebagai contoh, sebuah arsitektur *autoencoder* menggunakan dua *layers* sebagai bentuk paling sederhana. *Encoder* dengan satu *hidden layer* akan melakukan pemetaan data *input* $X \in \mathbb{R}^{N \times P}$, yang terdiri dari *N* jumlah sampel *input*, ke dalam representasi data *latent* $Z \in R^{N \times P}$. Ruang *latent* tersebut didefinisikan dalam Persamaan 1, di mana *K* adalah dimensi fitur asli dan *P* adalah dimensi fitur terkompresi.

$$Z = \sigma(WX + b) \quad (2)$$

W adalah matriks bobot, vektor *b* adalah bias, dan σ adalah fungsi aktivasi seperti *ReLU*. Tugas *decoder* adalah memetakan representasi *latent Z* kembali ke ruang *input* asli yang didefinisikan seperti berikut:

$$X' = \sigma(W'Z + b') \quad (3)$$

Dalam konteks ini, $X' \in \mathbb{R}^{N \times P}$ merujuk pada data hasil rekonstruksi yang dihasilkan oleh *decoder*. Tujuan utama dari *aut oencoder* adalah untuk untuk meminimalkan jarak antara data asli *X* dan data hasil rekonstruksi *X'* yang didefinisikan pada Persamaan 3:

$$L = ||X - X'||^2 \quad (4)$$

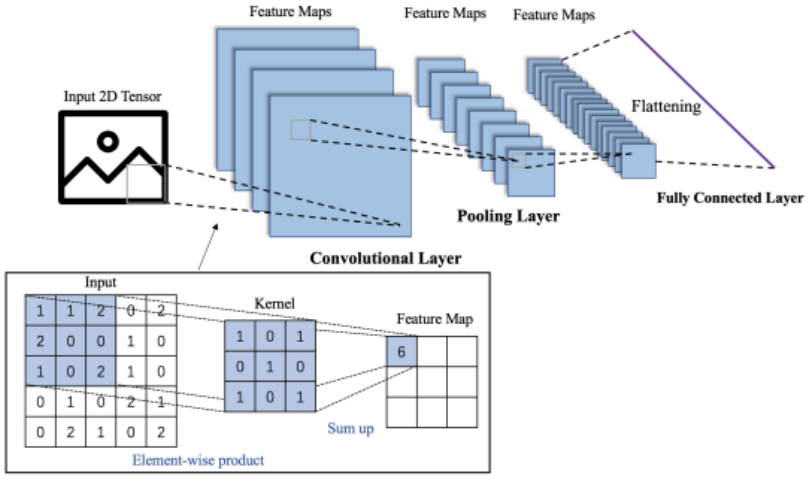
Dalam bentuk paling sederhana ini $\theta = \{W, b, W', b'\}$, parameter dioptimalkan selama *training*.

Bagian *encoder* bertanggung jawab untuk melakukan kompresi data *input* sehingga menghasilkan data *latent* yang dianggap sebagai sidik jari dari data *input*. Data *latent* dapat digunakan sebagai representasi terkompresi dari data asli dan dapat membantu mengurangi dimensi data. Bagian *decoder* bertugas untuk mengembalikan data *latent* ke bentuk aslinya seperti data *input*. Dalam hal ini, *encoder* dianggap sebagai alat kompresi sedangkan *decoder* dianggap sebagai alat ekstraksi [55].

2.2.9 *Convolutional Neural Network*

Convolutional Neural Network (CNN) merupakan salah satu arsitektur *deep learning* yang yang mempunyai *convolutional layer* sebagai filter masukan untuk mendapatkan informasi yang berguna [56]. CNN menjadi salah satu jenis *deep learning* yang populer dan sering diaplikasikan pada banyak bidang teknologi, seperti pemrosesan suara [57], *computer vision* [58], pengenalan wajah [59], dan yang lainya. Struktur jaringan syaraf CNN terinspirasi dari mata kucing, di mana rangkaian sel pada mata kucing sangat berlapis-lapis filternya, sehingga urutan korteks visual ini disimulasikan pada CNN [60]. Kelebihan utama dari CNN adalah dapat mendeteksi fitur-fitur penting secara otomatis tanpa pengawasan dari manusia. Arsitektur CNN dapat terdiri dari beberapa jenis *layer*, yaitu *convolutional layer*, *pooling layer*, *activation*, *fully connected layer*, *loss function* [45][61].

CNN memanfaatkan struktur tiga dimensi dalam arsitekturnya, yaitu lebar (*width*), tinggi (*height*), dan kedalaman (*depth*). Dalam setiap lapisan CNN, *volume input* tiga dimensi (3D *input volume*) akan diubah menjadi *volume ouput* tiga dimensi yang berisi aktivasi-aktivasi sel saraf (3D *output volume of neuron activations*). Gambar 2.3 menunjukkan bahwa dalam operasi konvolusi, kernel 3×3 digunakan untuk mengekstraksi fitur dari saluran gambar yang berukuran 5×5. Hasil dari operasi konvolusi tersebut adalah peta fitur dengan ukuran 3×3 [55].



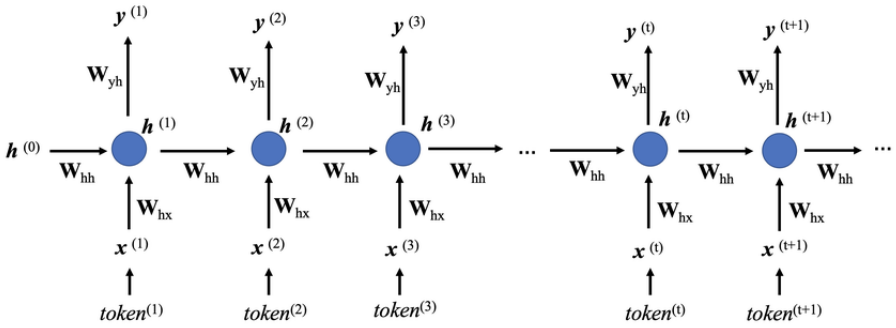
Gambar 2.3 Ilustrasi *Convolutional Neural Network* (CNN)

Convolutional layer adalah komponen penting dalam membentuk arsitektur CNN yang terdiri dari kumpulan filter yang disebut kernel. Kernel menjadi bagian penting dari layer ini yang selama proses *training* dimanfaatkan sebagai salah satu komponen ekstraksi fitur. Pada kasus pemrosesan gambar, gambar dinyatakan dalam metrik N-dimensi agar dapat menyatu dengan kernel untuk menghasilkan keluaran *feature map* [45][62]. *Pooling layer* adalah proses mengurangi *input* pada *feature map* hasil konvolusi menjadi satu nilai (subsampling) agar *feature map* berukuran lebih kecil dari sebelumnya. *Pooling layer* mampu meningkatkan kemampuan deteksi suatu objek dari jaringan konvolusi. Beberapa jenis *pooling* yang sering digunakan adalah *MaxPooling*, *MinPooling*, *AveragePooling*, dan *Global Average Pooling* (GAP) [45][63]. *Activation* merupakan fungsi pemetaan *input* ke *output* menjadi non-linear yang mampu mempelajari hal-hal yang sangat rumit. *Activation* yang sering digunakan pada CNN adalah *Sigmoid*, *ReLu*, dan *Tanh* [45]. *Fully connected layer* merupakan komponen terakhir dari arsitektur CNN. Pada *layer* ini, setiap syaraf terhubung ke semua syaraf dari *layer* sebelumnya, yang disebut dengan *fully connected*. *Fully connected* digunakan sebagai pengklasifikasi pada CNN yang mengikuti dasar metode dari *multi-layer* konvensional dari jaringan syaraf tiruan dengan masukan berupa vektor [45][56]. *Loss function* adalah fungsi yang digunakan untuk memprediksi kesalahan dari

model arsitektur CNN pada proses *training*. *Loss function* menunjukkan perbedaan keluaran yang aktual dan yang diprediksi [45].

2.2.10 *Recurrent Neural Network*

Recurrent Neural Network (RNN) adalah salah satu jenis arsitektur jaringan syaraf tiruan *deep learning* yang menggunakan data sekuensial atau *time-series* dengan konsep *supervised learning* [64][65]. RNN biasanya diaplikasikan pada masalah ordinal atau temporal, seperti terjemahan bahasa, pemrosesan bahasa alami, pengenalan ucapan, pasar saham dan yang lain. Teknologi yang menerapkan RNN adalah aplikasi populer seperti *Siri* dan *Google Translate*. Sama halnya CNN, RNN membutuhkan data pelatihan khusus untuk mempelajari datanya. RNN dibedakan oleh "memori" karena mengambil informasi dari data *input* sebelumnya untuk mempengaruhi *input* dan *output* setelahnya [66][45]. Sebagai *deep learning*, masalah yang sering ditemukan pada RNN adalah *vanishing gradient problem*. Dengan adanya permasalahan ini, maka dikembangkanlah dua jenis sel RNN, yaitu *Long Short-Term Memory* (LSTM) dan *Gated Recurrent Unit* (GRU).



Gambar 2.4 *Ilustrasi Recurrent Neural Network*

Pada Gambar 2.3, terlihat sebuah urutan token dengan panjang *n* sebagai *input* dari model. Setiap token diubah menjadi vektor terlebih dahulu, dan urutan *input* dapat direpresentasikan sebagai $(x^{(1)}, x^{(2)}, \ldots, x^{(t)}, \ldots, x^{(n)})$. Model RNN menerima urutan data dan menerapkan serangkaian jaringan saraf pada

setiap titik data satu per satu. Pada ilustrasi tersebut, sekelompok jaringan saraf pada RNN terdiri dari satu *layer* jaringan saraf tunggal. *Input* dari sekumpulan jaringan saraf tersebut adalah *ouput* dari posisi sebelumnya sebagai $h^{(i-1)}$ dan elemen saat ini dalam urutan sebagai $x^{(i)}$. *Ouput* dari jaringan saraf tersebut adalah lapisan *output* sebagai $y^{(i)}$ dan $y^{(i)}$ yang akan diteruskan ke posisi berikutnya. Formulasi untuk mengatasi elemen ke-*i* dalam urutan dapat diekspresikan sebagai berikut:

$$h^{(i)} = \sigma_h(W_{hh} \cdot h^{(i-1)} + W_{hx}x^{(i)} + b_h) \quad (5)$$

$$y^{(i)} = \sigma_y(W_{yh}h^{(i)} + b_y) \quad (6)$$

Berdasarkan formulasi di atas, menyatakan bahwa W_{hh}, W_{hx}, dan W_{yh} belajar untuk melakukan transformasi linear pada *output* posisi sebelumnya, elemen saat ini dalam urutan, dan *output* pemrosesan elemen saat ini. Selain itu, b_h dan b_y merupakan istilah untuk *bias*, sedangkan σ_h dan σ_y sebagai fungsi aktivasi. $h^{(0)}$ diinisialisasi sebagai vektor 0. Saat memproses elemen saat ini, model tidak hanya mempertimbangkan elemen saat ini, tetapi juga menerima informasi dari posisi sebelumnya, terlihat pada persamaan Persamaan 4. Setiap lapisan jaringan saraf dalam struktur berantai identik untuk setiap pemrosesan elemen dalam urutan [55][67].

2.2.11 *Gated Recurrent Unit*

Gated Recurrent Unit (GRU) adalah variasi pengembangan LSTM yang lebih sederhana karena strukturnya dalam beberapa kasus menghasilkan hasil yang sama bagusnya. Arsitektur GRU memiliki dua gerbang unit, yaitu *Update Gate* dan *Reset Gate*. Kedua gerbang ini menentukan jenis informasi apa yang akan dilewati sebagai hasilnya. *Update gate* menentukan berapa banyak informasi atau unit yang sebelumnya disimpan (dari langkah waktu sebelumnya) untuk diteruskan ke gerbang selanjutnya. Sedangkan *reset gate* memutuskan seberapa banyak informasi sebelumnya akan dihapus atau dilupakan. *Update gate* memiliki formulasi yang ditunjukkan pada Persamaan 6:

$$z_t = \sigma(W_z \cdot x^{(t)} + U_z \cdot h^{(t-1)} + b_z) \quad (7)$$

σ digunakan untuk mengecilkan *output* agar nilainya berada di antara 0 dan 1, sehingga dapat menentukan informasi mana yang akan diabaikan dari $h^{(t-1)}$ dan informasi mana yang akan ditambahkan dari $x^{(t)}$. Sedangkan formulasi *reset gate* adalah sebagai berikut:

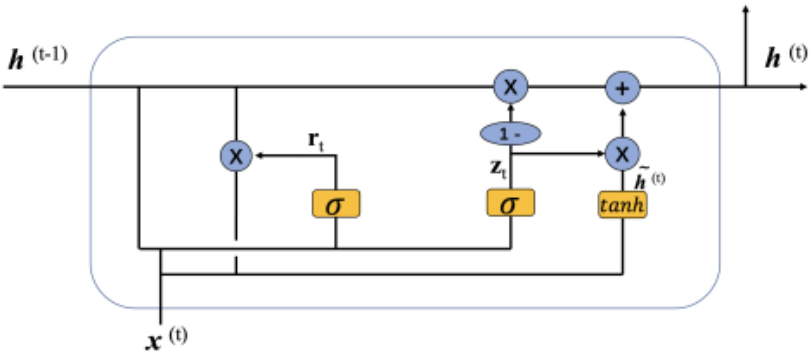
$$r_t = \sigma(W_r \cdot x^{(t)} + U_r \cdot h^{(t-1)} + b_r) \quad (8)$$

Kemudian, *gate reset* akan digunakan untuk menyimpan informasi yang relevan dari masa lalu ke dalam konten memori yang baru yang ditunjukkan pada Persamaan 8:

$$\tilde{h}^{(t)} = tanh(W \cdot x^{(t)} + U \cdot (r_t \odot h^{(t-1)} + b) \quad (9)$$

Pada tahap akhir, jaringan perlu menghitung *output* akhir berupa vektor $h^{(t)}$ yang berisi informasi tentang langkah waktu saat ini dan mengirimkannya ke jaringan. Untuk mencapai hal ini, maka dilakukan pembaruan *gate*. *Gate* ini berfungsi untuk memilih informasi apa yang harus diambil dari langkah waktu saat ini, yaitu $\tilde{h}^{(t)}$, dan informasi mana yang harus dikumpulkan dari langkah-langkah sebelumnya, yaitu $h^{(t-1)}$.

$$h^{(t)} = (1 - z_t) \odot \tilde{h}^{(t)} + z_t \odot h^{(t-1)}) \quad (10)$$



Gambar 2.5 Ilustrasi *Gated Recurrent Unit* (GRU)

Formula dari *reset gate* ini sama dengan formula *update gate*. Perbedaannya terletak pada bobot yang digunakan dan penggunaan gerbang [55][67][68][69].

2.2.12 *Bidirectional*

Bidirectional adalah jenis lapisan yang menghubungkan dua lapisan tersembunyi dari arah yang berlawanan menuju *output* yang sama. Hal ini memungkinkan informasi dari masa lalu dan masa depan diakses secara bersamaan, membuat lapisan keluaran lebih dalam dan lebih banyak informasi. Lapisan berulang dua arah atau BRNN tidak perlu mengandalkan *input* data untuk diperbaiki, sehingga informasi di masa depan dapat dijangkau dari kondisi saat ini [70]. Teknik BRNN ini lebih umum digunakan dalam pendekatan *supervised learning*, daripada *unsupervised* atau *semi-supervised*, karena perhitungan model probabilistik andal yang sangat sulit. Berbeda dengan RNN standar, BRNN dilatih untuk memprediksi arah waktu positif dan negatif secara bersamaan, sehingga *input* data dari masa lalu dan masa depan dapat digunakan untuk menghitung *output* yang sama. Sedangkan pada RNN standar membutuhkan lapisan tambahan untuk memasukkan informasi di masa mendatang [71].

2.2.13 *Time Distributed*

TimeDistributed adalah lapisan pada arsitektur jaringan saraf yang dapat digunakan untuk menerapkan suatu lapisan pada setiap langkah waktu input secara terpisah. Dalam model RNN, *TimeDistributed* dapat digunakan sebagai lapisan tersembunyi pada setiap langkah waktu dari *input* yang disusun secara berurutan. Ini memungkinkan model untuk memproses *input* secara sekuensial dan menjaga koneksi antara langkah waktu. Sebagai contoh, jika *input* memiliki urutan 3 langkah waktu dan lapisan konvolusi diterapkan pada *input*, TimeDistributed dapat digunakan untuk menerapkan lapisan konvolusi pada setiap 3 langkah waktu yang sama dari *input*. Dengan menggunakan *TimeDistributed* pada model RNN, model dapat memproses *input* secara sekuensial dan mempertahankan koneksi antara langkah waktu, yang memungkinkan model untuk mempertahankan informasi dari langkah waktu sebelumnya [72].

2.2.14 *Loss Function*

Loss function adalah suatu metrik yang membandingkan nilai target dan nilai prediksi yang dihasilkan oleh jaringan saraf, serta mengevaluasi kinerja model dalam melakukan *training* data. *Loss* merupakan suatu nilai yang menunjukkan seberapa besar selisih antara prediksi model dengan target yang sebenarnya. Jika model berhasil memprediksi dengan sempurna, maka nilai *loss* akan bernilai nol. Namun, jika prediksi model kurang akurat, maka nilai *loss* akan semakin besar. Saat proses pelatihan model, tujuan utama peneliti adalah untuk meminimalkan nilai *loss* antara *output* yang dihasilkan dengan target. *Loss function* ini digunakan ketika melatih *perceptron* dan jaringan saraf dengan cara memperbarui bobot yang ada. Semakin besar nilai *loss*, maka semakin besar pula perubahan yang terjadi pada bobot. Dengan cara meminimalkan nilai *loss*, akurasi model dapat meningkat. Namun, pengambilan keputusan mengenai besarnya perubahan bobot yang sesuai dengan minimal nilai *loss* harus dievaluasi dengan cermat dalam penerapan *machine learning* maupun *deep learning* [73][74].

2.2.15 *Mean Absolute Error* (MAE)

Mean absolute error (MAE) adalah ukuran yang menghitung rata-rata kesalahan prediksi dalam sebuah kumpulan data, tanpa memperhitungkan arah dari kesalahan tersebut. MAE dihitung dengan mencari selisih absolut antara nilai prediksi dan nilai aktual, kemudian diambil rata-rata dari selisih tersebut.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \qquad (10)$$

MAE sering digunakan dalam evaluasi model regresi dan merupakan skor linier, yang berarti setiap perbedaan individu memberikan kontribusi yang sama terhadap nilai rata-rata. Dengan demikian, MAE memberikan perkiraan tentang besarnya kesalahan dalam model, tetapi tidak memberikan informasi tentang arah kesalahan (misalnya, apakah kesalahan di atas atau di bawah prediksi). MAE memberikan informasi penting dalam membandingkan model dan memilih model yang terbaik, serta membantu dalam perbaikan model dengan menentukan rata-rata persentase kesalahan absolut yang diprediksi [75].

2.2.16 *Bias*

Bias merujuk pada perbedaan antara nilai aktual dan prediksi yang dihasilkan oleh suatu model. Ini adalah suatu asumsi yang dibuat oleh model tentang data untuk dapat memprediksi data baru. Jika bias tinggi, maka asumsi yang dibuat oleh model terlalu sederhana dan tidak dapat menangkap fitur penting dari data tersebut. Akibatnya, model tidak dapat mengidentifikasi pola dalam data pelatihan, sehingga tidak dapat bekerja dengan baik pada data pengujian. Hal ini menyiratkan bahwa model tersebut tidak dapat digunakan untuk memproses data baru dan tidak dapat diimplementasikan secara efektif [76].

2.2.17 Standar Deviasi

Standard deviation adalah sebuah nilai yang digunakan untuk mengukur penyebaran data pada suatu sampel dan seberapa dekat data-data tersebut dengan nilai rata-rata. Jika nilai dari *standard deviation* semakin besar, maka semakin bervariasi nilai-nilai pada setiap data atau semakin tidak sesuai dengan nilai rata-rata, sedangkan jika nilai *standard deviation* semakin kecil, maka semakin mirip nilai-nilai pada setiap data atau semakin dekat dengan nilai rata-rata [77].

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{N}} \tag{11}$$

Implementasi pada RNN dapat digunakan untuk mengukur seberapa jauh atau dekat hasil prediksi yang dihasilkan model RNN terhadap hasil sebenarnya dalam setiap periode waktu [78].

2.2.18 *Cumulative Moving Average*

Cumulative Moving Average (CMA) adalah teknik perhitungan rata-rata bergerak yang digunakan untuk menghitung nilai rata-rata dari suatu data secara bertahap dengan memberikan bobot lebih besar pada data yang lebih baru. Teknik ini sering digunakan untuk meratakan data yang fluktuatif atau bervariasi.

$$CMA = \frac{\sum_{i=1}^{N} x_i}{N} \tag{12}$$

$$CMA_{t+N} = \sum_{t+1}^{t=N} CMA_t + \frac{x_{t+1} - CMA_t}{N+1} \qquad (13)$$

Dalam implementasinya pada evaluasi RNN, CMA dapat membantu menerjemahkan model yang membentuk fluktuasi dan mengetahui apakah model tersebut mengalami *overfitting* atau *underfitting*. Caranya adalah dengan melakukan perhitungan CMA pada *loss function* dari model pada setiap langkah waktu selama proses *training*. Melalui cara ini, maka dapat dikethaui kinerja sebuah model secara keseluruhan dan berdasarkan perhitungan CMA pada pengujian data [79].

2.2.19 *Decision Tree*

Decision tree merupakan suatu algoritma *supervised learning* yang tidak parametrik, yang digunakan untuk tugas klasifikasi dan regresi. Algoritma ini memiliki struktur pohon dengan hierarki yang terdiri dari simpul akar, cabang, simpul internal, dan simpul daun [80]. Pembelajaran *decision tree* menggunakan strategi membagi dan menaklukkan dengan mencari titik perpecahan optimal dalam pohon secara *greedy approach*. Proses pemisahan ini kemudian diulangi secara *top-down* dan rekursif hingga semua atau sebagian besar rekaman telah diklasifikasikan ke dalam label kelas tertentu. Apakah semua poin data diklasifikasikan sebagai kumpulan homogen atau tidak, sangat tergantung pada kompleksitas pohon keputusan. Pohon yang lebih kecil lebih mudah mencapai simpul daun murni, yaitu titik data dalam satu kelas. Namun, semakin besar pohon, semakin sulit untuk mempertahankan kemurnian ini dan biasanya menghasilkan terlalu sedikit data yang termasuk dalam subpohon tertentu [81].

2.2.20 Musik Gitar

Gitar merupakan salah satu alat musik populer yang dapat digunakan dalam berbagai *genre* musik di seluruh dunia. Gitar diakui sebagai instrumen utama dalam *genre* seperti pop, *country*, *rock*, *jazz,* dan yang lainnya. Orang yang memainkan gitar disebut gitaris. Secara umum, peran gitar dalam instrumental musik adalah sebagai penghasil melodi. Instrumen yang dihasilkan gitar tergolong

jenis instrumen polifonik, karena dapat menghasilkan lebih dari satu nada dalam satu waktu [82][83].

2.2.21 Generator Musik

Generator musik menerapkan jaringan pembelajaran mendalam, sejenis kecerdasan buatan yang mengandalkan analisis data dalam jumlah besar. Sebagai contoh ketika sebuah program diberikan data dengan jumlah yang besar dari berbagai sumber yang mencakup berbagai jenis musik, kemudian program tersebut akan menganalisis data untuk menemukan pola, seperti akord, tempo, panjang, dan bagaimana nada berhubungan satu sama lain. Dengan belajar dari semua masukan, maka program dapat menulis melodi sendiri. Salah satu generator musik seperti *Soundraw.io* memungkinkan pengguna dalam menyesuaikan lagu menggunakan frasa yang dibuat oleh kecerdasan buatan. Kombinasi kecerdasan buatan dan alat penghasil musik membantu pengguna menghasilkan musik dengan mudah sambil menyesuaikan dengan hasil musik yang inovatif [84][85].

2.2.22 *Streamlit*

Streamlit merupakan sebuah *framework python* yang bersifat *open-source* untuk membangun aplikasi *web* di bidang *machine learning* dan ilmu data. Dengan *streamlit*, pengguna dapat dengan mudah mengembangkan dan menerapkan aplikasi *web* tanpa memerlukan pengetahuan mendalam tentang pemrograman web. *Streamlit* memungkinkan pengguna menulis kode aplikasi web dengan bahasa yang sama seperti menulis kode *python*. Streamlit membuatnya mudah bagi pengguna untuk bekerja pada *loop* interaktif pengkodean dan melihat hasil aplikasi *web* secara *real-time* [86].