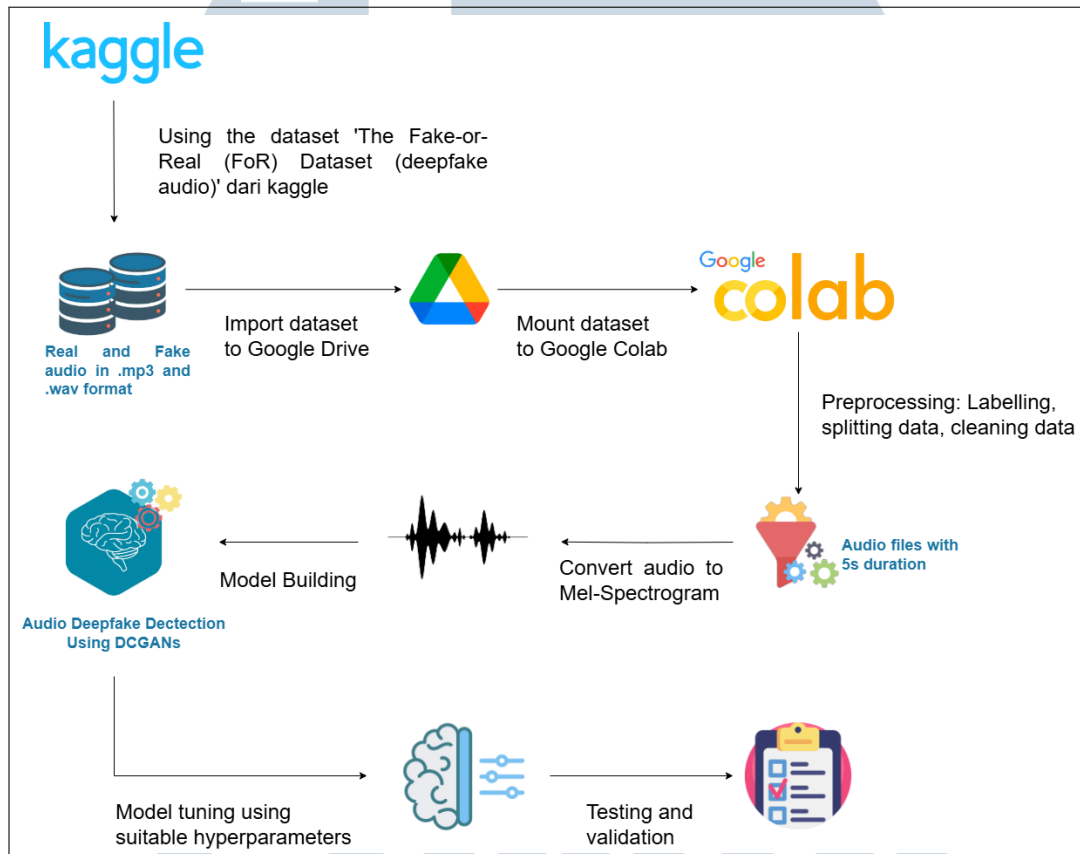


BAB 3

METODOLOGI PENELITIAN

3.1 Proses Pembangunan Model DCGANs untuk Deteksi Audio Deepfoax



Gambar 3.1. Diagram Methodology Penelitian DCGANs untuk Deteksi Audio Deepfoax

Gambar 3.1 merupakan diagram metodologi penelitian yang menjelaskan langkah-langkah yang dilakukan dalam pembangunan model agar proses pengembangan berjalan dengan lancar serta menghasilkan model yang memuaskan. Langkah-langkah penelitian ditulis secara komprehensif yang dimulai dari pengumpulan data, pra-proses data, pembangunan dan implementasi model DCGANs, *testing* beserta evaluasi dari hasil akhir pembangunan model.

3.1.1 Pengumpulan Data

Sebelum dilakukannya langkah pertama, ditentukan *environment* untuk membangun model. Dengan besarnya jumlah dataset dan untuk optimisasi *runtime*, digunakannya *virtual environment* google colab dengan tujuan untuk meminimalisir waktu yang dibutuhkan dalam *preprocessing* dan *training* model.

A Dataset

Langkah pertama yang dilakukan ialah pencarian dataset yang cocok dengan skenario deteksi audio asli dan palsu. Sehingga digunakannya dataset 'The Fake-or-Real (FoR) Dataset (deepfake audio)' yang terdiri dari audio autentik (asli) dan sintesis (palsu).

Dataset FoR merupakan dataset yang tidak berpasangan (*unpaired*). Artinya, sampel audio palsu yang ada di dalam dataset bukanlah versi *deepfake* dari sampel audio asli yang juga ada di dalam dataset yang sama. Sebaliknya, dataset ini merupakan kumpulan dari dua koleksi yang terpisah: audio asli yang didapatkan dari dataset 'Arctic Dataset', 'LJSpeech Dataset', 'VoxForge Dataset', dan juga rekaman yang dilakukan oleh pembuat dataset FoR, dan satu koleksi lain berisi beragam audio palsu yang didapatkan dari TTS (*Text-to-speech*) Deep Voice 3 dan Google Wavenet TTS. Struktur ini melatih model untuk mengenali karakteristik umum dari audio sintetis, bukan untuk membandingkan pasangan audio secara langsung.

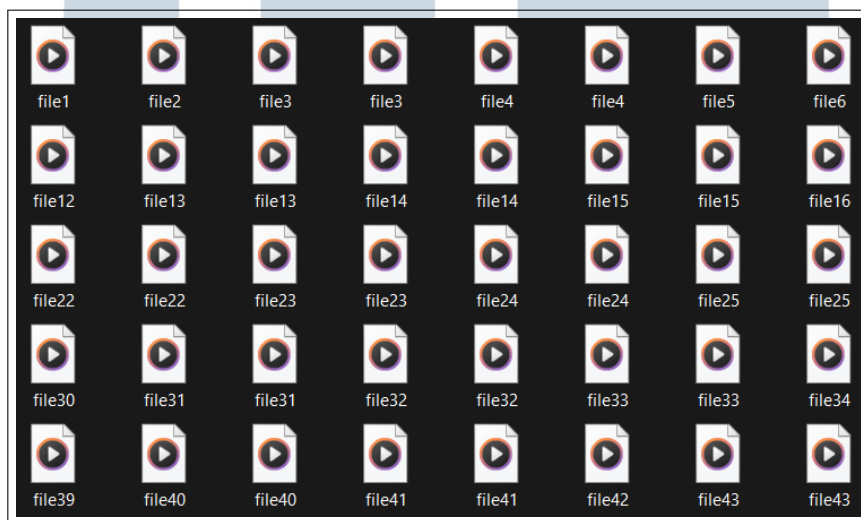
Tabel 3.1. Distribusi Dataset FoR Setelah Segmentasi

The Fake or Real (FoR) Dataset	for-original		
	Total Entri	Training	Testing
Audio Asli	11916	9533	2383
Audio Palsu	21529	17223	4306
Sum Total	33445		

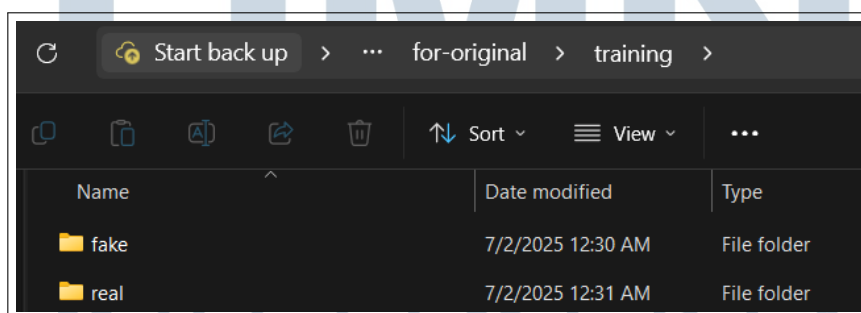
Berdasarkan tabel 3.1, jumlah total file audio yang didapatkan dari dataset setelah dilakukannya audio *segmenting* dan juga konversi Mel-Spektrogram ialah 33445, yang dimana 20% dari total jumlah file dijadikan sebagai *test-data*.

Setelah proses segmentasi, dataset akhir menunjukkan adanya ketidakseimbangan kelas (*class imbalance*), di mana jumlah sampel audio palsu lebih banyak daripada audio asli. Dalam penelitian ini, diambil keputusan

untuk tidak melakukan penyeimbangan dataset secara artifisial, seperti melalui *oversampling* atau *undersampling*. Alasan di balik keputusan ini adalah untuk melatih dan menguji model pada distribusi data yang lebih merepresentasikan skenario dunia nyata, di mana kemungkinan untuk menghadapi salah satu kelas (asli atau palsu) bisa jadi tidak selalu seimbang sempurna. Dengan tidak menyeimbangkan dataset, evaluasi kinerja model dianggap dapat memberikan gambaran yang lebih otentik mengenai kemampuannya untuk melakukan generalisasi tanpa bias dari intervensi penyeimbangan data [53]. Gambar 3.2 merupakan contoh isi dari dataset tersebut.



Gambar 3.2. Audio files dari dataset FoR



Gambar 3.3. Isi mentah dataset FoR-original

Dataset berisi audio file dalam format .wav dan .mp3 dengan durasi audio yang bervariasi yang berada dalam folder *fake* dan *real*. Pemilihan dataset 'The Fake-or-Real (FoR) Dataset (deepfake audio)' [47] didasarkan pada beberapa pertimbangan krusial yang selaras dengan tujuan penelitian:

1. Dataset ini secara spesifik dirancang untuk tugas deteksi audio deepfake, menyediakan koleksi data yang terbagi jelas antara audio asli (*bonafide*) dan audio palsu (*spoof*).
2. Audio palsu dalam dataset FoR dihasilkan menggunakan beragam teknik sintesis dan konversi suara yang canggih, yang merepresentasikan tantangan deteksi di dunia nyata dan memastikan model dilatih untuk dapat mengenali berbagai jenis artefak sintetis, bukan hanya dari satu metode pembuatan.
3. Ketersediaannya sebagai dataset publik yang telah digunakan dalam riset lain juga memungkinkan perbandingan dan validasi hasil penelitian secara lebih objektif.

B Mount Dataset dari Google Drive

Langkah selanjutnya setelah dataset ditemukan ialah download dan upload dataset ke Google Drive. Setelah dataset berhasil di upload, maka akan dilakukannya proses *mounting*.

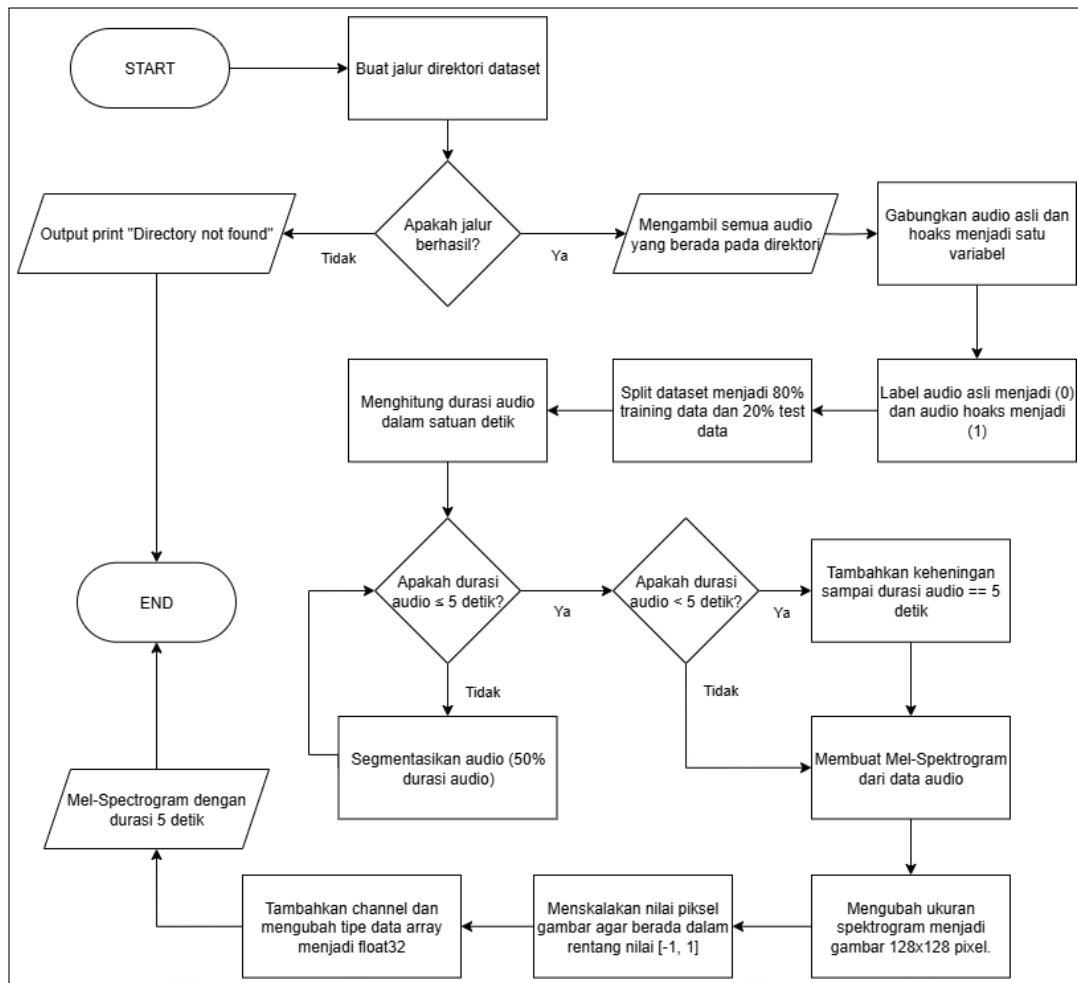
```
1 # Import the drive module from google.colab
2 from google.colab import drive
3
4 # Mount the Google Drive to the '/content/drive' directory
5 drive.mount('/content/drive')
```

Kode 3.1: Mount Dataset dari Google Drive

Mounting merupakan proses yang berfungsi untuk menghubungkan dataset yang ada pada Google Drive ke Google Colab, sehingga penelitian bisa dilakukan dengan menggunakan dataset tersebut.

3.1.2 Preprocessing

Dilakukannya beberapa tahap pra-proses sehingga model DCGANs bisa mengolah data tanpa adanya masalah. Gambar 3.4 menunjukkan alur kerja dari tahap ini.



Gambar 3.4. Flowchart preprocessing

Alur kerja pra-pemrosesan data dimulai dengan membangun jalur direktori menuju dataset, dengan validasi untuk memastikan direktori tersebut ada. Selanjutnya, semua file audio dari folder 'real' dan 'fake' dimuat dan digabungkan menjadi satu list, yang kemudian diberi label numerik (0 untuk asli, 1 untuk palsu) untuk keperluan pelatihan terawasi (*supervised learning*). Data yang telah berlabel ini kemudian dibagi menjadi set pelatihan dan pengujian. Langkah krusial berikutnya adalah standarisasi durasi audio. Setiap file audio diperiksa durasinya; jika kurang dari 5 detik, audio akan diperpanjang dengan keheningan (*padding*) hingga mencapai 5 detik. Jika lebih panjang, audio akan dipotong menjadi beberapa segmen 5 detik yang saling tumpang tindih (50% overlap). Pemilihan durasi 5 detik ini merupakan praktik umum dalam penelitian deteksi audio *deepfake* untuk menyeimbangkan antara menangkap konteks akustik yang cukup dan menjaga ukuran data agar tetap dapat dikelola secara komputasi [54],

durasi ini mampu menangkap informasi fonetik dan prosodik yang esensial dalam ucapan, seperti intonasi, ritme, dan transisi antar fonem, yang seringkali menjadi lokasi kemunculan artefak sintesis yang dapat dideteksi oleh model. Segmen yang lebih pendek dari ini berisiko kehilangan konteks akustik yang penting, sementara segmen yang lebih panjang akan secara signifikan meningkatkan beban komputasi tanpa memberikan tambahan informasi fitur yang sepadan untuk tugas klasifikasi ini. Dengan demikian, durasi 5 detik menjadi kompromi yang optimal antara kelengkapan fitur dan efisiensi komputasi.

Setelah semua audio memiliki durasi yang seragam, setiap segmen dikonversi menjadi representasi visual berupa Mel-Spektrogram melalui tiga tahapan utama:

1. *Short-Time Fourier Transform* (STFT) diterapkan pada segmen audio untuk menghasilkan spektrogram daya (*power spectrogram*) yang merepresentasikan energi sinyal pada setiap frekuensi linear.
2. Sumbu frekuensi linear ini diubah ke skala Mel dengan menerapkan sekumpulan *filterbank* Mel, yang secara efektif menekankan frekuensi rendah yang lebih relevan secara perseptual.
3. Nilai amplitudo dari spektrogram tersebut dikonversi ke skala desibel (dB) yang bersifat logaritmik, sehingga lebih sesuai dengan cara manusia mempersepsikan kenyaringan suara dan menghasilkan rentang nilai yang lebih baik untuk diproses oleh model.

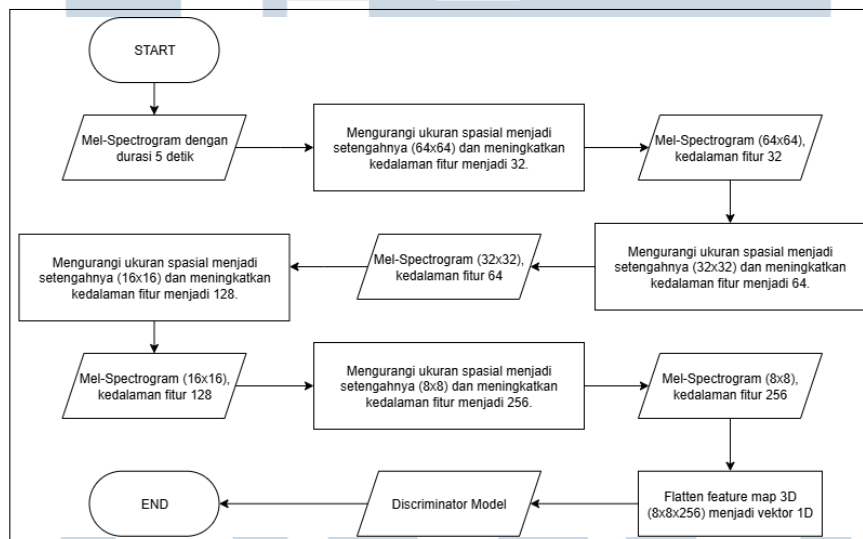
Hasil akhir dari proses ini kemudian diubah menjadi gambar skala abu-abu (*grayscale*) berukuran 128x128 piksel. Pemilihan skala abu-abu dilakukan karena informasi esensial dari spektrogram terletak pada intensitas energi di setiap frekuensi, bukan pada warna. Penggunaan satu kanal intensitas lebih efisien daripada tiga kanal warna (RGB). Ukuran 128x128 dipilih sebagai kompromi yang baik antara menjaga detail fitur akustik yang penting dengan efisiensi komputasi, karena ukuran ini merupakan standar umum untuk banyak arsitektur CNN. Akhirnya, nilai piksel pada spektrogram ini dinormalisasi ke rentang $[-1, 1]$ dan dimensi *channel* ditambahkan, mengubahnya menjadi format tensor (128, 128, 1) yang siap diolah oleh model CNN.

3.1.3 Konfigurasi Model DCGANs

Tahap ini mencakup perancangan arsitektur, kompilasi, dan proses pelatihan model.

A Pembangunan Arsitektur Model

Arsitektur yang digunakan adalah model Diskriminator dari DCGANs yang dimodifikasi untuk tugas klasifikasi. Gambar 3.5 mengilustrasikan alur data melalui lapisan-lapisan model.



Gambar 3.5. Flowchart arsitektur model

Alur kerja arsitektur model dimulai dengan input berupa Mel-Spektrogram berukuran 128x128 piksel yang telah diproses sebelumnya. Input ini pertama kali melewati blok konvolusi yang mengurangi ukuran spasialnya menjadi setengah (64x64) sambil meningkatkan kedalaman fitur menjadi 32. Proses ini, yang dikenal sebagai *downsampling*, diulangi secara bertahap melalui tiga blok konvolusi berikutnya. Pada setiap blok, dimensi spasial terus diperkecil (32x32, lalu 16x16, dan terakhir 8x8), sementara kedalaman fitur (jumlah filter) secara progresif ditingkatkan (64, 128, dan 256). Transformasi ini adalah inti dari cara kerja CNN; dengan mengurangi ukuran spasial dan menambah kedalaman fitur, model dipaksa untuk belajar fitur-fitur yang lebih kompleks dan abstrak di setiap lapisan. Hasil akhir dari proses ini adalah sebuah *feature map* berukuran 8x8x256, yang merupakan representasi data yang kaya akan fitur dan siap untuk diklasifikasikan.

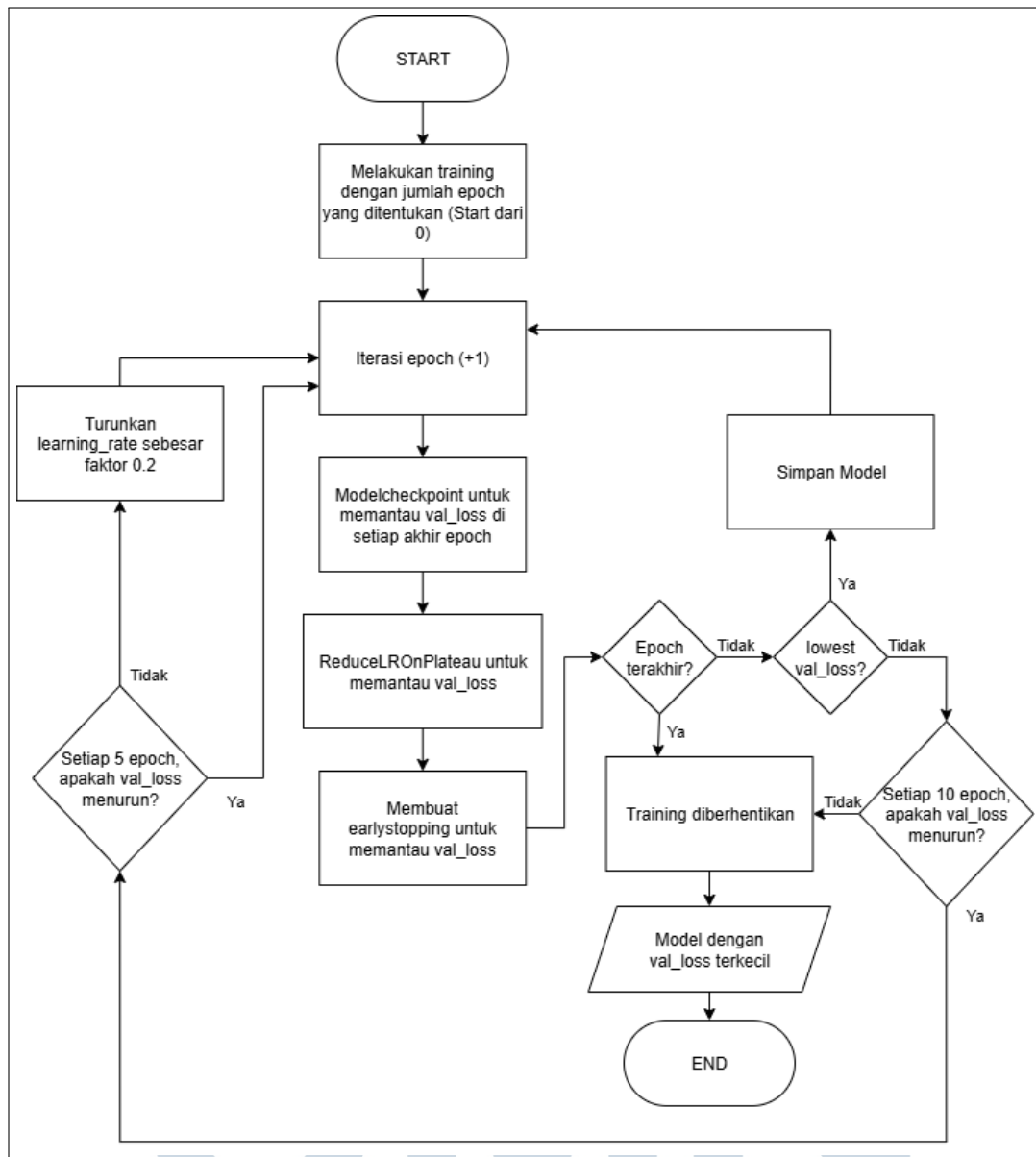
Setelah blok konvolusi terakhir, *feature map* 3D ini akan diratakan (*flattened*) menjadi sebuah vektor 1D. Vektor ini kemudian dimasukkan ke dalam lapisan *Dense* akhir yang berfungsi sebagai *classifier*, menghasilkan output tunggal yang merepresentasikan probabilitas apakah audio tersebut asli atau palsu.

Pemilihan arsitektur Diskriminator untuk tugas klasifikasi ini didasarkan pada hubungan arsitektural yang simetris namun berlawanan dengan komponen Generator dalam sebuah DCGAN. Secara konseptual, arsitektur Diskriminator dapat dianggap sebagai "kebalikan" dari arsitektur Generator. Sebuah Generator melakukan proses *upsampling*, di mana ia dimulai dari sebuah vektor noise yang kecil dan abstrak, lalu secara bertahap membangunnya menjadi sebuah representasi data yang besar dan konkret seperti gambar spektrogram melalui lapisan *Transposed Convolution* [55].

Sebaliknya, arsitektur Diskriminator yang digunakan dalam penelitian ini melakukan proses sebaliknya, yaitu *downsampling*. Ia mengambil input berupa data yang besar dan konkret (spektrogram 128x128) dan secara progresif "membongkar" atau menganalisisnya menjadi representasi fitur yang semakin kecil secara spasial namun semakin dalam, hingga akhirnya menghasilkan satu kesimpulan probabilitas. Dengan demikian, proses pengembangan arsitektur Diskriminator ini secara efektif memanfaatkan logika yang merupakan kebalikan dari proses generatif untuk mencapai tujuan ekstraksi fitur dan klasifikasi.

B Model Training

Proses pelatihan model dirancang untuk menjadi efisien dan otomatis dengan bantuan *callbacks*. Gambar 3.6 merinci logika dari proses pelatihan ini.



Gambar 3.6. Flowchart proses training

Proses pelatihan model yang diilustrasikan pada Gambar 3.6 merupakan sebuah siklus iteratif yang dikelola secara cerdas oleh tiga mekanisme *callback*. Inti dari proses ini adalah pelatihan model selama jumlah *epoch* yang telah ditentukan. Selama pelatihan, kinerja model pada data validasi (*val_loss*) terus dipantau. Pertama, *callback* ModelCheckpoint akan memeriksa *val_loss* di setiap akhir *epoch*. Jika nilai *loss* tersebut adalah yang terendah yang pernah tercatat, model beserta bobotnya akan disimpan. Kedua, *callback* ReduceLRonPlateau juga memantau *val_loss*; jika tidak ada penurunan selama 5 *epoch*, maka laju

pembelajaran (*learning rate*) akan diturunkan secara otomatis untuk membantu model melakukan penyesuaian yang lebih halus. Ketiga, *callback* `EarlyStopping` bertindak sebagai pengaman untuk mencegah *overfitting*. Jika *val_loss* tidak menunjukkan perbaikan dalam kurun waktu 10 *epoch*, proses pelatihan akan dihentikan sepenuhnya. Setelah pelatihan berakhir, baik karena mencapai batas *epoch* atau dihentikan oleh *EarlyStopping*, model terbaik yang disimpan oleh `ModelCheckpoint` (dengan *val_loss* terendah) akan menjadi output akhir dari proses ini.

3.1.4 Testing dan Evaluasi

Setelah model selesai dilatih, performanya diuji secara menyeluruh pada set data uji.

A Pengujian Model

Model dengan bobot terbaik (disimpan oleh `ModelCheckpoint` dan dipulihkan oleh `EarlyStopping`) dievaluasi pada data uji.

1. Metode `evaluate` digunakan untuk menghitung nilai *loss*, akurasi, dan AUC pada data uji.
2. Metode `predict` digunakan untuk mendapatkan probabilitas prediksi dari model untuk setiap sampel data uji.

B Analisis Hasil

Hasil prediksi kemudian dianalisis lebih dalam untuk mendapatkan metrik evaluasi yang komprehensif.

1. Probabilitas prediksi diubah menjadi label kelas (0 atau 1) menggunakan 0.5 *threshold*.
2. Classification Report: Dihasilkan untuk melihat nilai *Precision*, *Recall*, dan *F1-Score* untuk setiap kelas.
3. Equal Error Rate (EER): Dihitung dengan mencari titik di mana *False Positive Rate* (FPR) dan *False Negative Rate* (FNR) memiliki nilai yang sama, yang menunjukkan keseimbangan performa model.

3.1.5 Library yang Digunakan

Berikut merupakan daftar dari semua *library* yang digunakan dalam pembangunan model.

```
1 # Imports
2 import tensorflow as tf
3 import numpy as np
4 import os
5 import datetime
6 import librosa
7 import librosa.display
8 import matplotlib.pyplot as plt
9 import cv2
10 import random
11 from tensorflow.keras import layers, Input, regularizers
12 from tensorflow.keras.callbacks import ModelCheckpoint,
    EarlyStopping, TensorBoard
13 from IPython.display import Audio, display
14 from sklearn.model_selection import train_test_split
15 from sklearn.metrics import classification_report, roc_curve
```

Kode 3.2: Imports dan Constraints

1. TensorFlow (tf): Ini adalah fondasi dari seluruh model. TensorFlow digunakan tidak hanya untuk mendefinisikan arsitektur jaringan saraf melalui API Keras, tetapi juga untuk menjalankan seluruh proses pelatihan. Ini mencakup komputasi gradien untuk *backpropagation*, pembaruan bobot model melalui optimizer, dan eksekusi operasi pada GPU untuk akselerasi.
2. NumPy (np): Pustaka ini adalah tulang punggung dari semua manipulasi data numerik. Spektrogram yang dihasilkan oleh Librosa pada dasarnya adalah larik NumPy 2D. Operasi seperti normalisasi (menghitung rata-rata dan standar deviasi) dan penskalaan nilai piksel sebelum dimasukkan ke model sangat bergantung pada kecepatan dan efisiensi NumPy.
3. os: Pustaka ini sangat penting untuk otomatisasi alur kerja data. Fungsinya digunakan untuk membuat path file yang valid secara dinamis (menggunakan `os.path.join`) yang bekerja di berbagai sistem operasi, serta untuk memindai direktori dan mengumpulkan daftar semua file audio yang akan diproses.

4. `datetime`: Meskipun tidak esensial untuk fungsi inti model, pustaka ini sering digunakan dalam praktik *machine learning* yang baik untuk membuat nama file yang unik untuk menyimpan log pelatihan, bobot model, atau hasil visualisasi, misalnya `log-2025-07-03.txt`.
5. `Librosa`: Ini adalah pustaka utama untuk pemrosesan audio. Perannya sangat krusial, mencakup: (a) memuat file audio dari disk ke dalam larik NumPy (`librosa.load`), (b) mengubah sinyal audio 1D menjadi representasi frekuensi-waktu 2D melalui *Short-Time Fourier Transform* (STFT), dan (c) memetakan sumbu frekuensi STFT ke skala Mel untuk menghasilkan Mel-spektrogram (`librosa.feature.melspectrogram`).
6. `Matplotlib`: Pustaka ini digunakan untuk visualisasi hasil setelah proses pelatihan selesai. Secara spesifik, `matplotlib.pyplot` digunakan untuk memplot grafik akurasi dan *loss* dari data histori pelatihan, yang memungkinkan analisis visual terhadap fenomena seperti *overfitting*.
7. `OpenCV` (`cv2`): Berfokus pada pemrosesan gambar, peran utama OpenCV di sini adalah untuk memanipulasi spektrogram seolah-olah itu adalah gambar. Fungsi `cv2.resize` digunakan untuk memastikan bahwa setiap spektrogram, terlepas dari variasi kecil dalam dimensinya, diubah ukurannya secara seragam menjadi 128x128 piksel agar sesuai dengan ukuran input model.
8. `random`: Pustaka ini digunakan untuk mengontrol keacakan dalam proses. Dengan menetapkan nilai *seed* yang tetap di awal skrip, kita memastikan bahwa proses seperti pembagian data latih/uji dan inisialisasi bobot model dapat direproduksi, sehingga siapa pun dapat menjalankan ulang kode dan mendapatkan hasil yang sama persis.
9. `Keras` (`tensorflow.keras`): Keras menyediakan antarmuka yang lebih sederhana untuk TensorFlow. Dalam penelitian ini, `layers` digunakan untuk membangun blok-blok model seperti `Conv2D`, `Dense`, dan `Dropout`. `callbacks` seperti `ModelCheckpoint` dan `EarlyStopping` digunakan untuk mengelola proses pelatihan secara cerdas, yaitu menyimpan model terbaik secara otomatis dan menghentikan pelatihan jika tidak ada kemajuan.
10. `IPython.display`: Pustaka ini berguna selama fase pengembangan dan analisis eksplorasi dalam lingkungan notebook. Fungsi seperti `display(Audio(filepath))` memungkinkan peneliti untuk mendengarkan

sampel audio secara langsung di dalam output sel kode, yang membantu dalam verifikasi data.

11. Scikit-learn (sklearn): Pustaka ini menyediakan alat-alat penting untuk alur kerja *machine learning* standar. `train_test_split` digunakan untuk membagi dataset secara terstratifikasi (menjaga rasio kelas). Setelah pelatihan, `classification_report` digunakan untuk menghasilkan metrik evaluasi yang komprehensif (presisi, recall, f1-score), dan `roc_curve` digunakan untuk menghitung Tingkat Kesalahan Setara (EER).

